І.М. Пістунов,
О.П. Антонюк,

ЕЛЕКТРОННА ЕКОНОМІКА

Том 1

Криптовалюта. Big Data.

ДВНЗ «НГУ»
2017

Міністерство освіти і науки України
Державний вищий навчальний заклад
технічний університет
«Дніпровська політехніка»

,

І.М. Пістунов, Антонюк О.П.

ЕЛЕКТРОННА ЕКОНОМІКА
Том. 1
Криптовалюта. Big Data

Навчальний посібник

Дніпро
ДВНЗ «НГУ»
2017

УДК 004.738.5:338.46(075)
П34
Затверджено вченою радою університету як навчальний посібник по дисципліні "Електронна економіка" для студентів очної та заочної форм навчання зі спеціальності 051, Економіка, 071 Облік і оподаткування, 072 Фінанси, банківська справа та страхування (Протокол № 20 від 26.12.2017 р).

Рецензенти:
К.Ф. Ковальчук, док. екон. наук, проф., завідувач кафедри фінансів Дніпропетровської національної металургійної академії України
О.А. Паршина, докт. екон. наук, проф., завідувач кафедри кафедри економіки та моделювання бізнес-процесів Дніпропетровського університету ім. Альфреда Нобеля

Пістунов І.М.
П34 Електронна економіка. Том 1. Криптовалюта. Big Data [Електронний ресурс]: Навч. посібник/ І.М. Пістунов, О.П. Антонюк – / М-во освіти і науки України; Нац. Гірн. ун-т. – Д.: НГУ, 2017. – 291 с. Режим доступу: http://pistunovi.inf.ua/EE_KC_BD.pdf (дата звернення: 17.12.2017). – Назва з екрана.

У посібнику подано інформацію про сучасний етап розвитку економіки: криптовалюти та операції з даними, що містяться в Інтернеті.

Описано близько восьми найбільш популярних криптовалют, методів торгівлі, видобування (майнінг), конвертації. Подано основи поняття Big Data, принципи організації даних цього типу, методи їх комп'ютерної обробки.

У книзі подано завдання для самостійного виконання, тому він може слугувати і як посібник для практичних чи лабораторних занять із застосуванням комп'ютерної техніки.

Призначений для студентів вищих учбових закладів і може бути корисним для фінансистів, економістів, плановиків.

Посібник базується на літературних джерелах вітчизняних, зарубіжних авторів, ресурсах Інтернету та на досвіді викладання дисципліни «Електронна економіка» в Державному ВНЗ НТУ «ДП».

Навчальне видання

Пістунов Ігор Миколайович
Антонюк Оксана Петрівна

ЕЛЕКТРОННА ЕКОНОМІКА
Том. 1
Криптовалюта. Big Data

Навчальний посібник

Електронне видання

Видано в авторській редакції

Підписано до друку .2017. Формат 30 х 42/4.
Папір офсетний. Ризографія. Умовн. друк. арк. 13,72.
Обліково-видавн. арк. 13,73. Тираж 150 прим. Зам. № 96/12

Підготовлено до друку та видруковано
у Державному вищому навчальному закладі
«Національний гірничий університет».
Свідоцтво про внесення до державного реєстру ДК №1842 від 11.06.2004 р.
49005, м. Дніпро, просп. Д. Яворницького, 19.

ЗМІСТ

ВСТУП

Сучасна економіка є повністю електронною не тільки через те, що сучасний обіг економічної інформації включно з банківськими транзакціями та економічними стосунками здійснюється через електронні засоби комунікації. Тотальне переведення всіх економічних розрахунків у віртуальну площину викликала появу нових явищ, таких, як криптовалюта та bid data, що у свою чергу поступово змінює саме поняття валюти та забезпечує можливість абсолютно точно відслідковувати всі тенденції у розвитку суспільства як на рівні глобальних явищ світової економіки так і на мікрорівні: області, району, навіть окремому будинку.

Криптовалюта – цьому новому платіжному засобу, уперше створеному у 2009 році, приділяється все більше уваги. Ці платіжні засоби не мають ніякого покриття товарами чи послугами, але протягом всього періоду існування їх загальна сума і кількість видів постійно зростає.

Сама назва походить від слова «Криптографія» (від грецького *kryptós* – прихований і *gráphein* – писати) – наука про математичні методи забезпечення конфіденційності, цілісності і автентичності інформації. Розвинулась з практичної потреби передавати важливі відомості найнадійнішим чином.

Переведення даних у цифрову форму даних в багатьох галузях створило безпрецедентний обсяг масивів, які визначаються великим обсягом, швидкістю обробки, різноманітністю та достовірністю. Найновіші технології, такі як хмарні обчислення та розподілені системи в поєднанні з останніми розробками програмного забезпечення та підходами аналізу дозволили використовувати дані всіх типів для повного та змістовного аналізу для отримання додаткової вартості.

Саме тому ця дисципліна, яка навчить студентів працювати з криптовалютами та з великими обсягами даних, які містить Інтернет, і називається «Електронна економіка».

Після кожного розділу подані індивідуальні завдання, які студенти мають виконати протягом часу на засвоєння предмету «Електронна економіка».

Індивідуальні завдання оформляються як документ, який подається в електронному вигляді, вміщеним на будь-який носій інформації. Формат електронного документу має відповідати електронному процесору Excel Microsoft Office або MathLab. Окрім того, активні вікна мають бути скопійовані в документ Word з коментарями і описом роботи на кожному етапі.

Титульний лист оформлюється згідно прикладу, поданому нижче.

Міністерство освіти і науки України
Державний вищий навчальний заклад
технічний університет «Дніпровська політехніка»
Кафедра електронної економіки та економічної кібернетики

ІНДИВІДУАЛЬНА РОБОТА З ДИСЦИПЛІНИ
«ЕЛЕКТРОННА ЕКОНОМІКА»
Криптовалюти (або Big Data)

Розроби(ла)в ст. гр. ЕК-09-1 Косач-Квітка Л.П.
Варіант № 5

Прийняв проф., д.т.н. Пістунов І.М.
Доц., к.е.н., Антонюк О.П.

Дніпро
2017

Кожне виконане завдання повинно містити опис задачі, початкові значення змінних, які обираються за номером по списку студентської групи, вирішення та висновки щодо отриманих результатів.

Розділ 1. КРИПТОВАЛЮТИ

В розділі студенти ознайомляться з поняттям, криптовалют, узнають про алгоритми шифрування, визначать зміст блокчейну, навчаться торгувати криптовалютами.

Криптовалюта – цьому новому платіжному засобу, уперше створеному у 2009 році, приділяється все більше уваги. Ці платіжні засоби не мають ніякого покриття товарами чи послугами, але протягом всього періоду існування їх загальна сума і кількість видів постійно зростає.

Сама назва походить від слова «Криптографія» (від грецького *kryptós* – прихований і *gráphein* – писати) – наука про математичні методи забезпечення конфіденційності, цілісності і автентичності інформації. Розвинулась з практичної потреби передавати важливі відомості найнадійнішим чином.

Криптовалюти використовують досягнення криптографії для унеможливлення підробок записів про транзакції. Самі записи зберігаються за допомогою технології, що називається «Блокчейн» (Blockchain).

У табл. 1.1 наведено список найпопулярніших криптовалют. Загальна їх кількість наприкінці листопада 2013 перевищувала 80.

Така їх величезна кількість пояснюється новими можливостями, що відкрилися для інвесторів, які заробляють на курсових різницях різних валют.

Але головна відміна криптовалют у швидкості проходження транзакцій (не більше 10 хвилин), яка для сучасних фондових бірж, обладнаних можливостями торгівлі через Інтернет складає не менше, аніж півгодини. Ця відміна дозволяє швидше реагувати на курсові зміни і несе в собі (теоретично) можливості більшого заробітку.

Таблиця 1.1

Список найбільш популярних криптовалют

#		Крипто-валюта	Символ	Цена, BTC	Объём 24, BTC	Всего монет	Алгоритм
77		Bitcoin	BTC	1	1414440.5766	21000000	SHA-256
402		Litecoin	LTC	0,0104	28675.4544	84000000	Scrypt
570		Ripple	XRP	0,00002641	14147.9738	100000000000	ECDSA
241		Ethereum	ETH	0,0395	13036.3181	~90000000	Dagger-Hashimoto
242		EthereumClassic	ETC	0,0027	3292.6461	~90000000	Dagger-Hashimoto
253		Factom	FCT	0,0054	1889.012		Транзакционный сбо
440		Monero	XMR	0,0167	1720.858	18446744	CPU mining, CryptoNi
460		NEM	XEM	0,00002587	1689.9682	8999999999	blockchain
187		Dash	DASH	0,0585	1676.8648	22000000	X11
195		DigiByte	DGB	0,000001	1524.574	21000000000	SHA256,Scrypt,Qubit,
190		Decred	DCR	0,0118	901.9615	21000000	Blake256
725		Zcash	ZEC	0,0551	774.768	21000000	Equihash
413		MaidSafeCoin	MAID	0,0002	692.4312	4300000000	Транзакционный сбо
95		BitShares	BTS	0,00001023	626.5175	2000000000	Транзакционный сбо
632		Stratis	STRAT	0,0005	330.165	100000000	
41		Augur	REP	0,0092	311.0466		Smart contract
204		Dogecoin	DOGE	0,00000041	289.8612	100000000000	Scrypt
524		PinkCoin	PINK	0,0000056	260.1718	380000000	X11

Всі криптовалюти діють із застосуванням проксі-серверів.

Проксі-сервер (від англ. *proxy* — «представник, уповноважений») – сервер (комп'ютерна система або програма) в комп'ютерних мережах, що дозволяє клієнтам виконувати непрямі (через посередництво проксі-сервера) запити до мережевих сервісів. Спочатку клієнт з'єднується з проксі-сервером і запитує який-небудь ресурс (наприклад, e-mail), розташований на іншому сервері. Потім проксі-сервер або підключається до вказаного сервера і отримує ресурс у нього, або повертає ресурс з власного кешу (у випадках, якщо проксі має свій кеш). У деяких випадках запит клієнта або відповідь сервера може бути змінена проксі-сервером з певною метою. Також проксі-сервер дозволяє захищати клієнтський комп'ютер від деяких мережевих атак і допомагає зберігати анонімність клієнта.

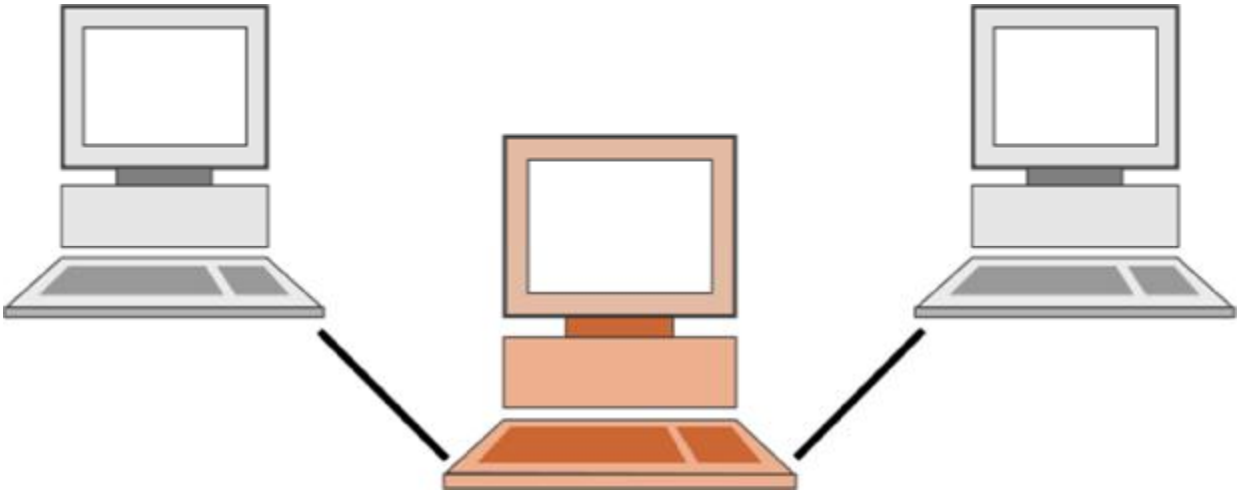


Рис. 1.1. Схематичне зображення проксі-сервера, де комп'ютер в середині діє як проксі-сервер між двома іншими.

1.1. Історія походження

У 2009 р була запущена принципово нова електронна валюта «Біткоїн», що стала першою серед анархічних пірінгових криптовалют – принципово нового феномена в грошовій сфері суспільства.

Bitcoin, біткоїни (англ. Bit - одиниця інформації «біт», англ. Coin - «монета ») – електронна пірінгова валютна система, створена в 2009 р Сатосі Накамото, реальна особа якого (або групи осіб) залишається невідомою. Принципова відмінність біткоїни від інших систем електронних грошей в тому, що біткоїн не є грошовим замінником. Він не забезпечений нічим. Його емітентом є не конкретна особа, а учасники системи. Однак ніхто в момент емісії біткоїну не перебирає обов'язки обмінювати їх на що-небудь інше за фіксованим курсом.

Швидкість емісії біткоїнів має жорстке алгоритмічне обмеження в внаслідок якого загальна кількість біткоїнів в кожен конкретний момент обмежена.

Алгоритмом передбачено уповільнення швидкості емісії. після емісії 10,5 млн. біткоїнів, швидкість емісії знизиться в два рази. Після 15,75 млн. біткоїнів швидкість емісії знизиться ще в два рази і так далі. У підсумку, загальний об'єм біткоїнів буде не більше 21 млн. Станом на листопад 2012 р розвиток системи біткоїнів може бути охарактеризований наступним чином:

• котирування одного біткоїну на біржі становить близько 10 дол. США;

• ринкова оцінка всіх біткоїнів становить близько 110 млн. Дол. США;

• кількість угод (транзакцій) за день складає близько 30 тис.;

• кількість унікальних адрес біткоїнів близько 40 тис.

Станом на січень 2018 року було згенеровано 80% всіх біткоїнів.

Спочатку біткоїни цікавили тільки групу ентузіастів, які генерували собі монети в надії, що дана діяльність буде мати якийсь сенс й майбутньому. Ці люди сподівалися, що біткоїни дійсно стануть грошима. Однак навряд чи ними рухало раціональне бажання отримати прибуток. Біткоїни є втіленням анархічних ідей. Анархізм є ідея про те, що суспільство може і повинно бути організовано без державного примусу або втручання. Оскільки одним з головних джерел насильства держави по відношенню до суспільства є грошова сфера, то створення незалежної від держави і пов'язаних з ним банківсько-олігархічних груп, було одним із завдань анархічного руху. Сама ідея криптовалюти з'явилася в середовищі криптоанархістів (або шифропанків) – групи, яка закликає до використання сильної

кріптографії на основі публічних ключів для захисту приватності і індивідуальної свободи. Таким чином, спочатку цінність біткоїнів носила виключно суб'єктивний характер, відбиваючи прагнення певних людей до моральних ідеалів і надію на їх досягнення. Біткоїни були цінними для них, оскільки дозволяв сподіватися на правильність їх ідей, що повинно було підтвердитися успішним розвитком системи.

Вважається, що перша офіційна продаж біткоїнів відбулася 25 квітня 2010 р (1000 монет було продано за 0,3 цента кожна). При цьому завважимо, що перша покупка реального товару за біткоїни відбулася пізніше. У травні 2010 року, один з учасників офіційного форуму, присвяченого біткоїну (https://bitcointalk.org/) купив піцу за 10 000 біткоїнів. Таким чином, на момент покупки сума угоди склала близько 30 дол. США. Стверджується, що піца коштувала трохи дешевше цього і покупець міг заощадити обмінявши біткоїни на долари і вже за них купив піцу, але він хотів підтримати розвиток криптовалюти здійснивши платіж виключно в біткоїнах.

В мережі Інтернет збереглася хронологія подій, яка показує, що цінність біткоїнів зовсім не виникла зі сфери обміну грошей на товар.

Навпаки, перш ніж відбулася угода купівлі реального товару, біткоїн отримав оцінку в сфері обміну валют. Інакше і не могло бути, адже біткоїни нічим не забезпечені. Але цей факт вже суперечить еволюційній теорії грошей. Якщо припустити, що хронологія вірна, то біткоїни – це валюта, що отримала початкову цінність не в процесі її обміну на реальні товари, а в процесі її обміну на інші валюти. І лише потім ця цінність була підтверджена в сфері обміну на товари.

«10 лютого 2011 року на сайті http://slashdot.org/ з'явилася новина про досягнення паритету між BTC і USD ». Менш ніж за рік ціна біткоїнів виросла більш ніж в 300 разів. Подальший розвиток біткоїнів призвів до творення необхідної інфраструктури, перш за все бірж з обміну біткоїнів на інші валюти. Перша з них була створена в липні 2010 р – MtGox (Https://mtgox.com/). Створення

біржі збільшило ліквідність біткоїнів і дозволило привернути до нього увагу спекулянтів. Це в свою чергу сказалось на купівельній спроможності валюти. В даний час дана біржа є найбільшою з купівлі-продажу біткоїнів. Її щоденний оборот сягає декількох десятків тисяч біткоїнів. 1 червня 2011 на сайті http://gawker.com/ була опублікована стаття про популярність Bitcoin серед торговців наркотиками, після чого курс BTC різко підскочив. Стверджувалося, що біткоїни використовується в якості грошей частиною наркоторговців, для здійснення реалізації забороненого товару через сегмент Інтернету, в якому користувач може зберігати повну анонімність.

Криптовалюта ідеально підходить для подібних операцій, оскільки дозволяє:

- зберігати повну анонімність транзакцій;

- фактично є системою готівкового електронного грошового обороту.

Стаття викликала ажіотаж на біржі. В результаті, 9 червня 2011 року – курс біткоїну досяг свого історичного максимуму $ 29,57, після чого почав знижуватися і в листопаді 2011 р курс становив $ 2,14. Після цього курс біткоїну знову почав рости. Таким чином, навіть після суттєвого зниження курсу, знайшлося певне «дно» і курс біткоїнів не опустився до нуля.

Біткоїни не має немонетарною цінності, він також не є законним засобом платежу, тобто його ходіння не підтримується державною владою. Однак всупереч еволюційної теорії грошей, біткоїни продовжують залишатися засобом платежу, тобто грошима. Існувала думка, що біткоїн не зможе проіснувати тривалий час. Зростання його курсу це лише спекуляція, на зразок «тюльпанової лихоманки» в Голландії в першій половині XVII в. Мовляв, настане момент, коли курс почне падати. В цій ситуації він опуститися до нуля, оскільки біткоїни не має ніякої підтримки в вигляді матеріального забезпечення. Люди втратять віру в цю валюту і вона перестане існувати. Однак цього не сталося.

Думка про те, що припинення спекуляції біткоїнами призведе до краху платіжної системи засноване на тому, що в умовах відсутності забезпечення курс може впасти до нуля, після чого не буде ніякого механізму повернення курсу на

більш високий рівень. Оскільки біткоїн не має немонетарного попиту, то не буде і механізму, який потім штовхне курс вгору. Однак ці міркування поки не підтвердилися фактами. Більш того, подібні міркування ігнорують той факт, що початково біткоїн був нічим і зміг отримати грошовий статус на основі монетарного попиту. Єдине, що є у біткоїну – це обмеженість обсягу грошової маси в кожний момент часу і розуміння учасниками ринку меж її збільшення в кожний конкретний момент. Можливо, це і є те, що не дозволяє біткоїну впасти до нуля.

Обмеженість кількості біткоїнів істотно відрізняє його від сучасних паперових грошей. Відносний успіх біткоїну привів до появі інших криптовалют. Велика частина з них має шахрайський характер або не отримала значного поширення.

1.2. Блокчейн

Blockchain – це розподілена децентралізована база даних транзакцій, в основі якої лежать складні математичні обчислення і криптографія. Простіше кажучи - це книга обліку всіх цифрових записів в інтернеті.

Першою особливістю технології є те, що вона не має централізованого сервера або процесинг-центру. Всі користувачі Blockchain утворюють собою якусь мережу, в якій кожен учасник зберігає копію даних. Це дозволяє проводити будь-які операції, без будь-яких посередників.

Blockchain практично неможливо зламати, тому що інформація знаходиться не на одному сервері, а розподілена між всіма учасниками мережі. Грубо кажучи, для злому потрібно атакувати всі комп'ютери. Чим більше користувачів в мережі, тим захищеності і ширше стає мережу. При цьому кожен учасник мережі - рівноправний.

Всі дані, що зберігаються в Blockchain – відкриті. Система дозволяє відстежити повний ланцюжок зміни даних, що дозволяє з легкістю перевірити будь-яку інформацію.

Кожна транзакція в Blockchain шифрується криптографічними ключами, що в свою чергу забезпечує захищеність даних від фальсифікації і злому.

1.2.1. Приклад формування відкритого щоденника, записи якого неможливо підробити

Коля вирішив вести щоденник. Для цього він завів зошит і почав писати там рядки на кшталт таких:

1. Купив хліба

2. Подзвонив Геннадію

...

132. Дав Васі в борг 100 рублів

133. Поцілував Люду

134. Зробив фіззарядку

Він дуже старався вести щоденник чесно, і якщо у нього з кимось виникав спір про щось, що сталося раніше, він діставав його і тикав усім носом в свої записи. Одного разу Коля сильно посперечався з Васею на тему того, чи давав він Васі в борг 100 рублів чи ні. У момент спору у Колі не було з собою щоденника, але він обіцяв завтра ж принести і все показати Васі.

Вася вирішив не спокушати долю, пробрався до Колі в будинок, знайшов щоденник, дістався рядка 132 і замінив його на «Поцілував Олю». На наступний день Коля дістав щоденник, довго шукав у ньому запис про борг Васі, не знайшов і прийшов вибачатися.

Пройшов рік, Васю замучила совість, і він зізнався у всьому Колі. Коля пробачив друга, але вирішив на майбутнє використовувати якусь більш надійну систему запису, яку не можна було б так просто підробляти.

Придумав він наступне. Він знайшов програму md5sum, яка брала будь-який текст і перетворювала його в хеш – 32 незрозумілі цифри. Як саме вона це робила, Коля не розумів, але в цілому здавалося, що вона видавала повну нісені-

тницю. Наприклад, якщо в програму ввести слово «привіт», вона у відповідь видає «1FA72D8916096572FEA50DDAA7E37B7F». А якщо ввести, здавалося б, майже те ж саме, але зі знаком пробілу, то вже «F0701C6067598E01B2F936E9A73B78D3».

Почухавши потилицю, Коля придумав спосіб ускладнити майбутнім Васям заміну записів наступним чином: після кожного запису він вставляв хеш, який виходив, якщо згодувати програмі тему запису і минулий хеш. Новий щоденник виходив таким:

0000 (початковий хеш, обмежимося для простоти чотирма знаками)

1. Купив хліба

4178 (хеш від 0000 і «Купив хліба»)

2. Подзвонив Геннадію

4234 (хеш від 4178 і «Подзвонив Геннадію»)

...

4492

132. Дав Васі в борг 100 рублів

1010

133. Поцілував Люду

8204 (хеш від 1010 і «Поцілував Люду»)

Якщо тепер якийсь Вася захоче змінити рядок 132, зміниться і хеш цього рядка (він буде не 1010, а якимось іншим). Це, в свою чергу, вплине на хеш рядка «133. Поцілував Люду » (він буде не 8204, а іншим), і так далі до кінця щоденника. Тепер заради однієї записи Васі доведеться підмінити весь щоденник після неї, що складно.

Минув час, Коля відкрив банк. Він все так же писав у щоденник записи «дав в борг» і «взяв в кредит», забезпечуючи їх хешами. Банк розрісся, і одного разу він дав в борг Васі мільйон. Наступної ночі десять найнятих Васею за півмільйона хакерів пробралися в кімнату Колі, замінили запис «143313. Дав в борг Нового Васі 1000000» на «143 313. Дав в борг 10» і швидко перерахували всі хеші аж до кінця щоденника.

Дивом Коля виявив підміну і, раз така справа, вирішив ускладнити спосіб підробки щоденника: «Тепер, – вирішив Коля, – я буду в кінці кожного запису в дужках додавати яке-небудь число ("НОНС"), а підбирати його буду так, щоб кожен хеш закінчувався на два нуля». Єдиний спосіб це зробити – перебирати числа, поки не вийде потрібний хеш:

0000 (початковий хеш, обмежимося для простоти чотирма знаками)

1. Купив хліба (22)

4100 (хеш від 0000 і «Купив хліба (22)», 22 було підібрано, щоб хеш закінчувався на 00)

2. Подзвонив Геннадію (14)

3100 (хеш від 4100 і «Подзвонив Геннадію (14)»)

...

1300

132. Дав Васі в борг 100 рублів (67)

9900

133. Поцілував Люду (81)

8200 (хеш від 9900 і «Поцілував Люду (81)»)

Для створення кожного запису Колі тепер в середньому потрібно буде перебрати близько 50 чисел, що потребує багато часу. Відповідно, якщо запис хтось підмінить, підробка її та всіх наступних буде теж в 50 разів складніше, а це значить, що тепер Васі навіть хакерами не впоратися.

Через якийсь час Коля взяв собі партнера і вони стали обидва вести аналогічний щоденник. Для кожного нового запису обидва одночасно починали підбирати НОНС і той, кому першому вдавалося знайти потрібний, вносив запис. Так як удвох підбирати НОНС швидше, Коля ускладнив завдання і вимагав, щоб всі хеші кінчалися вже на три нуля, а не на два.

Цей остаточний Колін щоденник по суті і є справжній блокчейн, тільки Колю з його партнером треба замінити на купу з'єднаних по мережі комп'ютерів, а обчислення хешів ускладнити, щоб навіть комп'ютерам було тяжко.

Маючи такий щоденник, можна будувати різні цікаві системи. Наприклад, біткоїни. Біткоїни – це щоденник, де кожен запис має вигляд «Передати стільки-то грошей з гаманця X на гаманець У». Так як щоденник не можна підробити і в ньому зберігається вся історія переказів, в будь-який момент з нього можна обчислити кількість грошей на кожному гаманці. Ну а щоб в системі взагалі були якісь гроші, біткоїни зроблений так, що кожен запис у щоденнику закінчується словами «Провести Z монет і перевести мені», де «мені» – це той користувач, хто першим «вгадає» НОНС, який забезпечить хеш з потрібною кількістю нулів в кінці.

Поверх щоденника з деякою кількістю криптографії якої формуються численні додаткові цікаві системи. Наприклад, можна робити записи в дусі «Хто вирішить рівняння $f(x) = 14$, той отримує 10 монеток». Відповідно, перший запис у щоденнику, де буде надано рішення, буде автоматично вважатися одержувачем монеток. Навколо цієї та подібних ідей будуються так звані «контракти».

1.1.2. Опис системи Blockchain

Ланцюжок блоків транзакцій (англ. *Blockchain, Block chain* від *block* – блок, *chain* – ланцюг) – розподілена база даних, яка підтримує постійно зростаючий перелік записів, званих блоками, від підробки та переробки. Кожен блок містить часову мітку та посилання на попередній блок хеш дерева.

Така розподілена база даних закладена в основу криптовалюти «Біткоїн», – вона була описана у 2008 і реалізована в 2009 році, – де слугує бухгалтерською книгою для всіх операцій. Таку базу називають *Блокчейн*.

У квітні 2017 року в ЗМІ з'явилася інформація, що Україна планує перевести державні дані у блокчейн. І вже у кінці цього ж року оголошено, що земельний кадастр має сам е таку структуру.

Блок транзакцій – спеціальна структура для запису групи транзакцій в системі Біткоїн та аналогічних їй.

Щоб транзакція вважалася достовірною («підтвердженою»), її формат і підписи повинні перевірити і потім групу транзакцій записати в спеціальну структуру – *блок*. Інформацію у блоках можна швидко перевірити. Кожен блок завжди містить інформацію про попередній блок. Усі блоки можна вибудувати в один ланцюжок, яка містить інформацію про всі вчинені коли-небудь операції з біткоїнами. Перший блок в ланцюжку – *первинний блок* (англ. *genesis block*) — розглядається як окремий випадок, так як у нього відсутній батьківський блок[7].

Блок складається із заголовку та списку транзакцій. Заголовок блоку включає в себе свій хеш, хеш попереднього блоку, хеші транзакцій та додаткову службову інформацію. Першою транзакцією в блоці завжди вказується отримання комісії, яка стане нагородою користувачеві за створений блок.

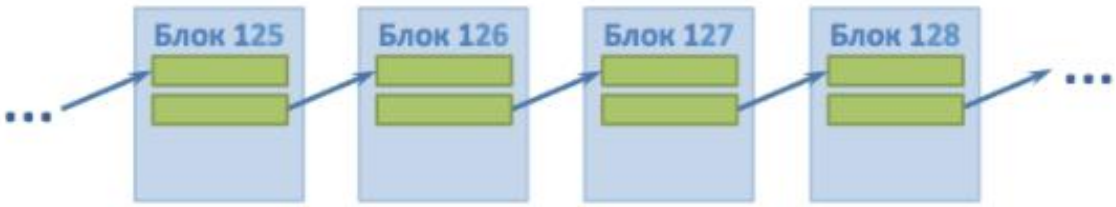
Далі йдуть всі або деякі з останніх транзакцій, які ще не були записані в попередні блоки. Для транзакцій в блоці використовується деревоподібне хешування, аналогічне формування хеш-суми файлу в протоколі BitTorrent. Транзакції, крім нарахування комісії за створення блоку, містять всередині атрибута *input* посилання на транзакцію, за якою на цей рахунок були отримані біткоїни. Комісійні операції можуть містити в атрибуті будь-яку інформацію (для них це поле носить назву англ. *Coinbase parameter*), оскільки у них немає батьківських транзакцій.

Створений блок буде прийнятий іншими користувачами, якщо числове значення хешу заголовка дорівнює або нижче певного числа, величина якого періодично коригується. Так як результат хешування (функції SHA-256) непередбачуваний, немає алгоритму отримання бажаного результату, крім випадкового перебору. Якщо хеш не задовольняє умові, то довільно змінюється блок службової інформації в заголовку і хеш перераховується. Зазвичай потрібна велика кількість перерахунків. Коли варіант знайдено, вузол розсилає отриманий блок іншим підключеним вузлів, які перевіряють блок. Якщо помилок немає, то блок вважається доданим в ланцюжок і наступний блок повинен включити в себе його хеш.

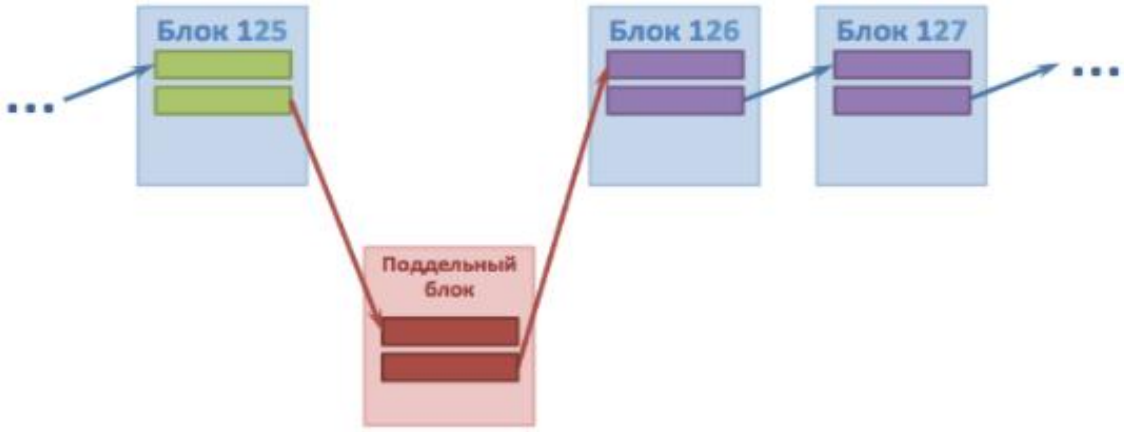
Величина цільового числа, з яким порівнюється хеш, коригується через кожні 2016 блоків. Заплановано, що вся мережа буде витрачати на генерацію одного блоку приблизно 10 хвилин, на 2016 блоків – близько двох тижнів. Якщо 2016 блоків сформовані швидше, то мета трохи зменшується і досягти її стає важче, в іншому випадку мета збільшується. Зміна складності обчислень не впливає на надійність мережі Біткоїн і потрібно лише для того, щоб система генерувала блоки майже з постійною швидкістю, що не залежить від потужності мережі.

Підробити ланцюжок або змінити записи в блоці не вийде, тому що змінюються ключі - і це буде відразу видно. Що ж станеться, якщо зловмисник все-таки спробує підробити blockchain? Розглянемо на конкретних прикладах.

Нехай в ланцюзі вже створена послідовність блоків.



Нехай, керуючись своїми підлими цілями, зловмисник вирішив вставити свій підроблений блок з потрібними йому записами між двома вже існуючими блоками і тим самим підробити історію.



У заголовок свого блоку він вставляє ключ попереднього блоку і розсилає інформацію про новий блоці всім іншим учасниками. Однак вони відразу ж виявлять підробку, тому що в ключі блоку 126 (і всіх наступних теж) не врахована ключ підробленого блоку.

Єдина можливість обману - лиходій повинен зробити повторний Майнінг всіх наступних блоків. Але це неможливо, тому що швидкість генерації нових блоків сумлінними учасниками точно така ж, як швидкість генерації підроблених блоків зловмисниками (бо саме так задуманий алгоритм Майнінг). «Лиходії» просто ніколи не доженуть "хороших".

Варто визнати, що строго теоретично шанс так званої «атаки 51%» (коли «лиходіїв» більше, ніж «хороших») все ж залишається. Але на практиці її втілити майже неможливо.

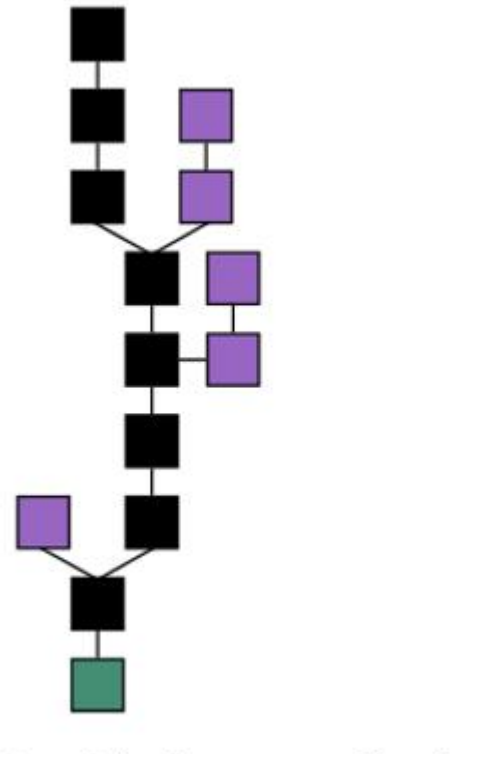


Рис. 1.2. Ланцюжок блоків

Основна послідовність блоків (чорні) є найдовшою від початкового (зелений) до поточного. Побічні гілки (фіолетові) відсікаються.

Блоки одночасно формуються безліччю «майнерів». Блоки, які задовольняють критерії, відправляються в мережу, включаючись в розподілену базу блоків. Регулярно виникають ситуації, коли кілька нових блоків в різних частинах розподіленої мережі називають попереднім один і той же блок, тобто ланцюжок блоків може розгалужуватися. Спеціально або випадково можна обмежити ретрансляцію інформації про нових блоках (наприклад, одна з ланцюжків може розвиватися в рамках локальної мережі). У цьому випадку можливе паралельне нарощування різних гілок.

У кожному з нових блоків можуть зустрічатися як однакові транзакції, так і різні, увійшли тільки один з них. Коли ретрансляція блоків поновлюється, майнери починають вважати головною ланцюжок з урахуванням рівня складності хешу і довжини ланцюжка. При рівності складності і довжини перевага віддається тій ланцюжку, кінцевий блок якої з'явився раніше. Транзакції, що увійшли тільки в відхилену гілку (в тому числі по виплаті винагороди), втрачають статус підтверджених. Це викликає втрату вже згенерованих біткоїнів.

Якщо це операції з передачі біткоїнів, то вона буде поставлена в чергу і потім включена в черговий блок. Транзакції отримання винагороди за створення відтяті блоків не дублюються в іншій гілці, є «зайві» біткоїни, виплачені за формування відтяті блоків, не отримують подальших підтверджень і «втрачаються».

Розподілена база даних Blockchain формується як безперервно зростаюча ланцюжок блоків з записами про всіх транзакціях. Копія бази даних або її частини одночасно зберігається на безлічі комп'ютерів та синхронізуються відповідно до формальних правил побудови ланцюжка блоків. Інформація в блоках не шифрована і доступна у відкритому вигляді, але захищена від змін криптографічно через хеш-ланцюжок.

Найчастіше умисне зміна інформації в будь-якій з копій бази або навіть в досить великій кількості копій не буде визнане істинним, так як не відповідатиме правилам. Деякі зміни можуть бути прийняті, якщо будуть внесені в усі копії бази (наприклад, видалення кількох останніх блоків через помилку в їх формуванні).

До версії 0.8.0 для зберігання ланцюжка блоків основний клієнт використовував Berkeley DB, починаючи з версії 0.8.0 розробники перейшли на LevelDB.

Підтвердження транзакцій

Поки транзакція не включена в блок, система вважає, що кількість біткоїнів на якійсь адресі залишається незмінною. У цей час є технічна можливість оформити кілька різних транзакцій по передачі з однієї адреси одних і тих же біткоїнів різним одержувачам. Але як тільки одна з подібних транзакцій буде

включена в блок, інші транзакції з цими ж біткоїнами система буде вже ігнорувати.

Наприклад, якщо в блок буде включена більш пізня транзакція, то більш рання буде вважатися помилковою. Є невелика ймовірність, що при розгалуженні дві подібні транзакції потраплять в блоки різних гілок. Кожна з них буде вважатися правильною, лише при відмиранні гілки одна з транзакцій стане вважатися помилковою. При цьому не буде мати значення час здійснення операції.

Таким чином, попадання транзакції в блок є підтвердженням її достовірності незалежно від наявності інших транзакцій з тими ж біткоїнами. Кожен новий блок вважається додатковим підтвердженням транзакцій з попередніх блоків. Якщо в ланцюжку три блоки, то транзакції з останнього блоку будуть підтверджені один раз, а вміщені в перший блок будуть мати три підтвердження. Досить дочекатися декількох підтверджень, щоб звести ймовірність скасування транзакції до мінімуму.

Для зменшення впливу таких ситуацій на мережу існують обмеження на розпорядження щойно отриманими біткоїнами. Згідно сервісу blockchain.info до травня 2015 року максимальна довжина відкинутих ланцюжків була 5 блоків. Необхідне число підтверджень для розблокування отриманого залежить від програми-клієнта, або від вказівок приймаючої сторони. Клієнт «Bitcoin-qt» для відправлення не вимагає наявності підтверджень, але у більшості одержувачів за замовчуванням виставлено вимогу 6 підтверджень, тобто реально скористатися отриманим зазвичай можна через годину. Різні онлайн-сервіси часто встановлюють свій поріг підтверджень.

Біткоїни, отримані за створення блоку, протокол дозволяє використовувати після 100 підтверджень, але стандартна програма-клієнт показує комісію через 120 підтверджень, тобто зазвичай скористатися комісією можна приблизно через 20 годин після її нарахування.

«Подвійне витрачання»

Якщо контролювати більше 50 % сумарної обчислювальної потужності мережі, то існує теоретична можливість при будь-якому порозі підтверджень одні і ті ж біткоїни передати два рази різним одержувачам – одна з транзакцій буде публічною і буде підтверджуватися в загальному порядку, а друга не буде афішуватися, її підтвердження буде відбуватися блоками прихованої паралельної гілки. Лише через деякий час мережа отримає відомості про другу транзакцію, вона стане підтвердженою, а перша втратить підтвердження і буде ігноруватися. В результаті не відбудеться подвоєння біткоїнів, але зміниться їх поточний власник, при цьому перший отримувач втратить біткоїни без будь-яких компенсацій.

Відкритість ланцюжка блоків дозволяє внести в довільний блок зміни. Але тоді потрібно перерахунок хешу не тільки зміненого блоку, але і всіх наступних. Фактично, для такої операції буде потрібно потужність не менше тієї, яка була використана для створення зміненого і наступних блоків (тобто всієї поточної потужності), що робить таку можливість вкрай малоймовірною.

Подвійне витрачання біткоїнів на практиці ніколи не було зафіксовано. На травень 2015 року паралельні ланцюжки ніколи не перевищували 5 блоків.

Складність

За вимогу до хешам блоків відповідає спеціальний параметр, званий «складність». Так як обчислювальні потужності мережі непостійні, цей параметр перераховується клієнтами мережі через кожні 2016 блоків таким чином, щоб підтримувати середню швидкість формування розподіленої БД на рівні 2016 блоків в два тижні. Таким чином 1 блок повинен створюватися приблизно раз на десять хвилин. На практиці, коли обчислювальна потужність мережі зростає – відповідні часові проміжки коротше, а коли знижується – довший. Перерахунок складності з прив'язкою до часу можливий завдяки наявності в заголовках блоків

часу їх створення. Воно записано в Unix-форматі і взято за системним годинником автора блоку (якщо блок створений у пулі, то за системним годинником сервера цього пулу).

1.1.3. Українські держреєстри переведуть на блокчейн

Уряд України і блокчейн-компанія Bitfury попередньо домовилися про переведення держреєстрів України на систему блокчейн.

Bitfury Group – блокчейн-компанія з офісами в США і в інших країнах. Її директор – Валерій Вавилов, відомий в Україні як співзасновник файлообмінника EX.ua.

Він зазначив, що це найбільша угода з урядом щодо технології блокчейн. Вона включає в себе передачу всіх електронних урядових даних України на платформу блокчейн, – повідомляє Reuters.

“Безпечна державна система, побудована на блокчейн, може забезпечити мільярди доларів активів і мати значний соціальний та економічний вплив у всьому світі”, – заявив Вавилов.

Це другий блок-проект Bitfury для урядових структур. У квітні 2016 р компанія підписала угоду з Грузією про проведення експерименту – запуску першого блокчейн-реєстру по реєстрації прав на землю.

Також блокчейн з метою державного управління почали використовувати Швеція та Естонія, однак український проект набагато перевершує їх за масштабами.

На сьогодні BitFury є однією з найбільших інфраструктурних та процесингових компаній на цьому специфічному ринку.

У компанії є офіси в Сан-Франциско, Вашингтоні, Амстердамі, Лондоні, Гонконзі, а сервери розміщені в Грузії й Ісландії.

Нагадаємо, в 2014 році BitFury залучив 20 млн доларів інвестицій від ряду інвесторів, а в 2015 році інвестором BitFury став iTech Capital. За даними Crunchbase, за весь час BitFury привернула 90 млн доларів.

1.3. Алгоритми формування хешу

Хеш-сумою (хешем, хеш-образом, хеш-кодом) називається значення хеш-функція на якихось вхідних даних.

В криптографії хеш-суму іноді також називають дайджестом повідомлення.

Значення хеш-суми може використовуватися для перевірки цілісності даних, їх ідентифікації та пошуку (наприклад в p2p мережах), а також заміняти собою дані, які небезпечно зберігати в явному вигляді (наприклад, паролі, відповіді на питання тестів і т.д.).

Явне значення хеш-суми, зазвичай, записується в шістнадцятковому вигляді. Так, утиліта md5sum, яка обчислює значення хеш-функції MD5 від заданого файлу, видає результат у вигляді рядка з 32-х шістнадцяткових цифр, наприклад, 026f8e459c8f89ef75fa7a78265a0025.

Хеш-функцією називається будь-яка функція h: $X \rightarrow Y$, легко обчислювана і така, що для будь-якого повідомлення M значення h (M) = H (згортка) має фіксовану бітову довжину. X – множина всіх повідомлень, Y – множина двійкових векторів фіксованої довжини.

Як правило хеш-функції будують на основі так званих однокрокових функцій, що стискають текст $y = f(x_1, x_2)$ двох змінних, де x_1, x_2 і y – виконавчі вектори довжини m, n і n відповідно, причому n – довжина згортки, а m – довжина блоку повідомлення.

Для отримання значення $h(M)$ повідомлення спочатку розбивається на блоки довжини m (при цьому, якщо довжина повідомлення не кратна m то остан-

ній блок якимось спеціальним чином доповнюється до повного), а потім до отриманих блокам M_1, M_2, .., M_N застосовують наступну послідовну процедуру обчислення згортки:

$$H_0 = v,$$
$$H_i = f(M_i, H_{i-1}),\ i = 1, .., N,$$
$$h(M) = H_N$$

Тут v - деяка константа, часто її називають вектором ініціалізації, вона вибирається з різних міркувань і може являти собою секретну константу або набір випадкових даних (вибірку дати і часу, наприклад).

При такому підході властивості хеш-функції повністю визначаються властивостями однокрокової стискаючої функції.

Виділяють два основних види криптографічних хеш-функцій – ключові і без ключа. Ключові хеш-функції називають кодами аутентифікації повідомлень. Вони дають можливість без додаткових засобів гарантувати як правильність джерела даних, так і цілісність даних в системах з довіряють один одному користувачами.

Безключові хеш-функції називаються кодами виявлення помилок. Вони дають можливість за допомогою додаткових коштів (шифрування, наприклад) гарантувати цілісність даних. Ці хеш-функції можуть застосовуватися в системах з користувачами, що довіряють або й не довіряють один одному.

1.3.1. Алгоритм MD5

MD5 *(Message Digest 5)* – 128-бітний алгоритм хешування, розроблений професором Рональдом Л. Рівестом в 1991 році. Призначений для створення «відбитків» або «дайджестів» повідомлень довільної довжини. Прийшов на зміну MD4, що був недосконалим.

У 2004 році китайські дослідники Сяоюнь Ван (Xiaoyun Wang), Денгуо Фен (Dengguo Feng), Сюецзя Лай (Xuejia Lai) і Хонбо Ю (Hongbo Yu) повідомили про знаходження ними вразливості в алгоритмі, що дозволяє за невеликий час (1 годину на кластері IBM p690) знаходити колізії хеш-функцій. На жаль, автори так і не відкрили свій секрет широкій публіці.

У 2006 році чеський дослідник Властимил Клима опублікував алгоритм, що дозволяє знаходити колізії на звичайному комп'ютері з довільним початковим вектором (A,B,C,D), за допомогою методу, що був названий: "тунелювання".

Початковий етап підготовки

- Вхідні дані вирівнюються так, щоб їхній розмір можна було порівняти з 448 по модулю з 512. Спочатку дописують одиничний біт (навіть якщо довжина порівняна з 448), далі необхідна кількість нульових бітів .
- Дописування 64-бітного представлення довжини даних по вирівнюванню. Якщо довжина перевищує , то дописують молодші біти.

Допоміжні таблиці та функції

- Ініціалізуть 4 змінних розміром по 32 біта:
  - A = 01 23 45 67;
  - B = 89 AB CD EF;
  - C = FE DC BA 98;
  - D = 76 54 32 10.

Вирівняні дані розбиваються на блоки по 32 біта, і кожен проходить 4 раунди з 16 операторів. Всі оператори однотипні і мають вигляд: [abcd k s i], визначений як , X – блок даних, а $T[1..64]$ – 64х елементна таблиця побудована наступним чином: , s – циклічний зсув вліво на s біт отриманого 32-бітного аргументу.

- В першому раунді Fun F(X, Y, Z) = XY v (not X)Z
- В другому раунді Fun G(X, Y, Z) = XZ v (not Z)Y.
- В третьому раунді Fun H(X, Y, Z) = X xor Y xor Z.
- В четвертому раунді Fun I(X, Y, Z) = Y xor (X v (not Z)).

Циклічна процедура обчислення

Саме обчислення проходить наступним чином:

- Зберігаються значення A, B, C і D, що залишились після операцій з попередніми блоками(або їх початкові значення якщо блок перший)

AA = A

BB = B

CC = C

DD = D

Раунд 1

/*[abcd k s i] a = b + ((a + F(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 0 7 1][DABC 1 12 2][CDAB 2 17 3][BCDA 3 22 4]

[ABCD 4 7 5][DABC 5 12 6][CDAB 6 17 7][BCDA 7 22 8]

[ABCD 8 7 9][DABC 9 12 10][CDAB 10 17 11][BCDA 11 22 12]

[ABCD 12 7 13][DABC 13 12 14][CDAB 14 17 15][BCDA 15 22 16]

Раунд 2

/*[abcd k s i] a = b + ((a + G(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 1 5 17][DABC 6 9 18][CDAB 11 14 19][BCDA 0 20 20]

[ABCD 5 5 21][DABC 10 9 22][CDAB 15 14 23][BCDA 4 20 24]

[ABCD 9 5 25][DABC 14 9 26][CDAB 3 14 27][BCDA 8 20 28]

[ABCD 13 5 29][DABC 2 9 30][CDAB 7 14 31][BCDA 12 20 32]

Раунд 3

/*[abcd k s i] a = b + ((a + H(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 5 4 33][DABC 8 11 34][CDAB 11 16 35][BCDA 14 23 36]

[ABCD 1 4 37][DABC 4 11 38][CDAB 7 16 39][BCDA 10 23 40]

[ABCD 13 4 41][DABC 0 11 42][CDAB 3 16 43][BCDA 6 23 44]

[ABCD 9 4 45][DABC 12 11 46][CDAB 15 16 47][BCDA 2 23 48]

Раунд 4

/*[abcd k s i] a = b + ((a + I(b,c,d) + X[k] + T[i]) <<< s). */

[ABCD 0 6 49][DABC 7 10 50][CDAB 14 15 51][BCDA 5 21 52]

[ABCD 12 6 53][DABC 3 10 54][CDAB 10 15 55][BCDA 1 21 56]

[ABCD 8 6 57][DABC 15 10 58][CDAB 6 15 59][BCDA 13 21 60]

[ABCD 4 6 61][DABC 11 10 62][CDAB 2 15 63][BCDA 9 21 64]

Проміжний результат

Виконати наступні операції

A = AA + A

B = BB + B

C = CC + C

D = DD + D

Після цього перевірити, чи є ще блоки, якщо є, то повторюють циклічну процедуру обчислення для наступного 32-х бітового блоку.

Результат

Після обчислення для всіх блоків даних, отримуємо кінцевий хеш у регістрах A B C D. Якщо вивести слова у зворотному порядку DCBA, то отримаємо MD5 хеш.

MD5-хеші

Хеш містить 128 біт (16 байт) і зазвичай представляється як послідовність з 32 шістнадцяткових цифр.

Кілька прикладів хешу:

MD5 ("md5") = 1bc29b36f623ba82aaf6724fd3b16718

Навіть невелика зміна вхідного повідомлення (у нашому випадку на один біт: ASCII символ «5» з кодом $0x35_{16} = 00011010\ '1'_2$ замінюється на символ «4» з кодом $0x34_{16} = 00011010\ '0'_2$) призводить до повної зміни хешу. Така властивість алгоритму називається лавинним ефектом.

MD5 ("md4") = c93d3bf7a7c4afe94b64e30c2ce39f4f

Приклад MD5-хеша для «нульового» рядка:

MD5 ("") = d41d8cd98f00b204e9800998ecf8427e

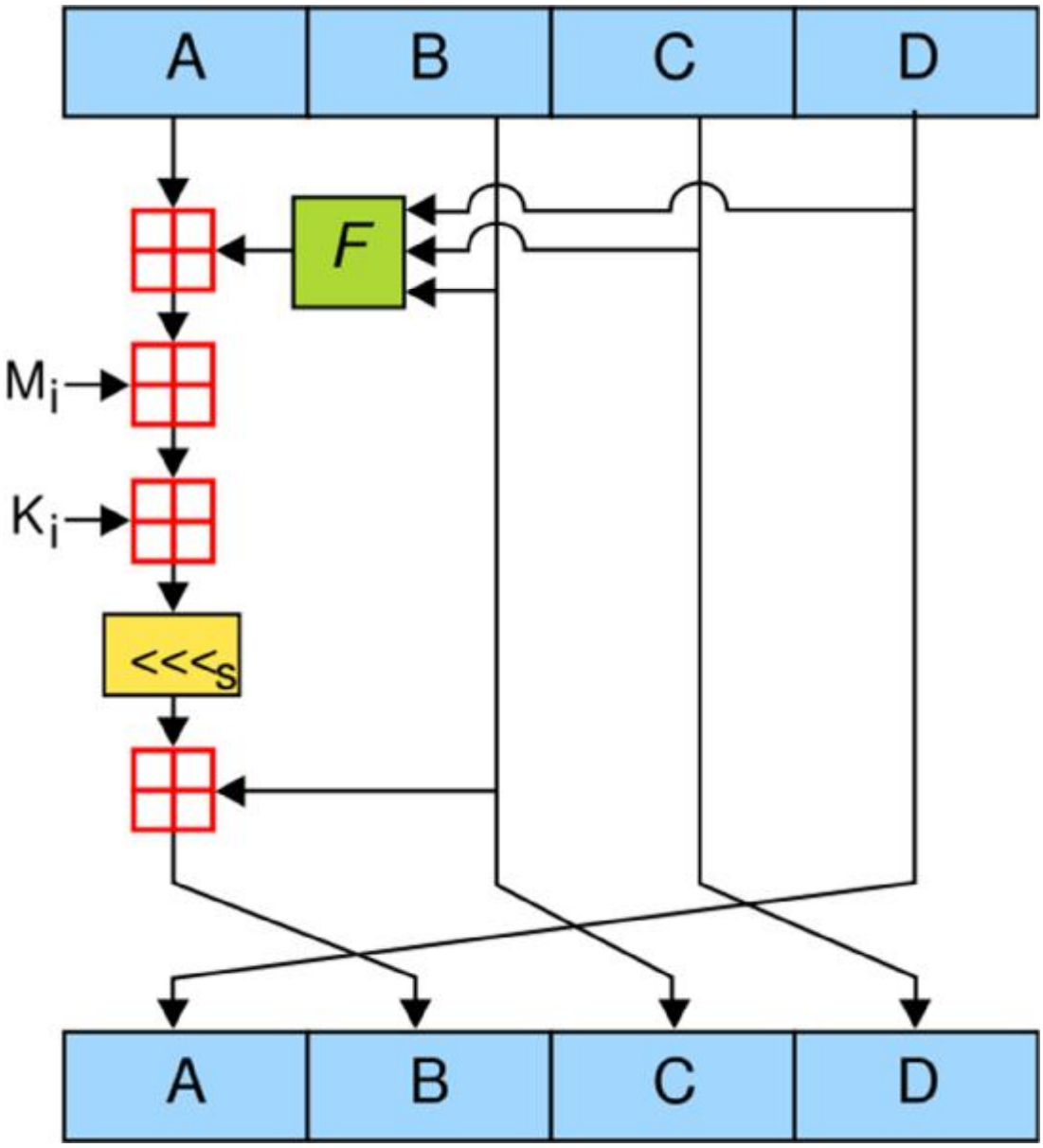


Рис. 1.3. Блок схема роботи алгоритму MD5

1.3.2. Алгоритм SHA-2

SHA-2 (англ. *Secure Hash Algorithm Version 2* – безпечний алгоритм хешування, версія 2) – збірна назва односторонніх хеш-функцій SHA-224, SHA-256, SHA-384 і SHA-512. Хеш-функції призначені для створення «відбитків» або «дайджестів» повідомлень довільної бітової довжини. Застосовуються в різних додатках або компонентах, пов'язаних із захистом інформації.

Хеш-функції *SHA-2* розроблені Агентством національної безпеки США і опубліковані Національним інститутом стандартів і технології США у федеральному стандарті обробки інформації *FIPS PUB 180-2* в серпні 2002 року У цей стандарт також увійшла хеш-функція *SHA-1*, розроблена в 1995 році. У лютому 2004 року до *FIPS PUB 180-2* була додана *SHA-224*.

У жовтні 2008 року вийшла нова редакція стандарту — *FIPS PUB 180-3*.

В липні 2006 року з'явився стандарт RFC 4634 «Безпечні хеш-алгоритми США (*SHA* і *HMAC-SHA*)», що описує *SHA-1* і сімейство *SHA −2*.

Агентство національної безпеки від імені держави випустило патент на *SHA-2* під ліцензією *Royalty Free*.

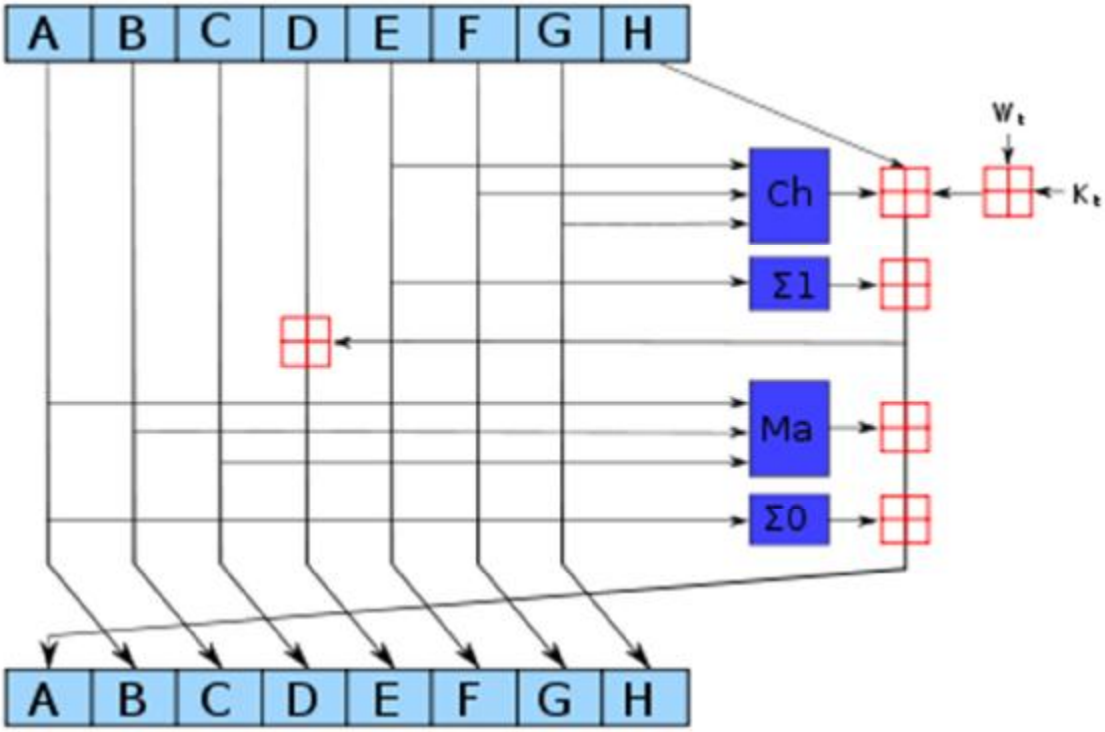


Рис. 1.4. Схема однієї ітерації алгоритмів SHA-2

Хеш-функції сімейства *SHA-2* побудовані на основі структури Меркла-Демґарда.

Початкове повідомлення після доповнення розбивається на блоки, кожен блок – на 16 слів. Алгоритм пропускає кожен блок повідомлення через цикл з 64-ма чи 80-ма ітераціями (раундами). На кожній ітерації 2 слова перетворюються,

функцію перетворення задають інші слова. Результати обробки кожного блоку складаються, сума є значенням хеш-функції.

Алгоритм використовує такі бітові операції:

- ‖ – Конкатенація,
- + – Додавання,
- *And* – Побітове «І»,
- *Or* – Побітове «АБО»,
- *Xor* – Виключне «АБО»,
- *Shr* (Shift Right) – Логічний зсув вправо,
- *Rotr* (Rotate Right) – Циклічний зсув вправо.

У наступній таблиці показані деякі технічні характеристики різних варіантів SHA-2. «Внутрішній стан» означає проміжну хеш-суму після обробки чергового блоку даних:

Хеш-функція	**Довжина дайджесту повідомлення (біт)**	**Довжина внутрішнього стану (біт)**	**Довжина блоку (біт)**	**Максимальна довжина повідомлення (біт)**	**Довжина слова (біт)**	**Кількість ітерацій в циклі**
SHA-256/224	256/224	256	512	$2^{64} - 1$	32	64
SHA-512/384	512/384	512	1024	$2^{128} - 1$	64	80

Нижче наведені приклади хешів *SHA-2*. Для всіх повідомлень мається на увазі використання кодування ASCII.

SHA-224 ("The quick brown fox jumps over the lazy dog")
= 730E109B D7A8A32B 1CB9D9A0 9AA2325D 2430587D DBC0C38B AD911525
SHA-256 ("The quick brown fox jumps over the lazy dog")

= D7A8FBB3 07D78094 69CA9ABC B0082E4F 8D5651E4 6D3CDB76 2D02D0BF 37C9E592

SHA-384 ("The quick brown fox jumps over the lazy dog")

= CA737F10 14A48F4C 0B6DD43C B177B0AF D9E51693 67544C49 4011E331 7DBF9A50 9CB1E5DC 1E85A941 BBEE3D7F 2AFBC9B1

SHA-512 ("The quick brown fox jumps over the lazy dog")

= 07E547D9 586F6A73 F73FBAC0 435ED769 51218FB7 D0C8D788 A309D785 436BBB64 2E93A252 A954F239 12547D1E 8A3B5ED6 E1BFD709 7821233F A0538F3D B854FEE6

Найменша зміна повідомлення в переважній більшості випадків призводить до зовсім іншого хешу внаслідок лавинного ефекту. Наприклад, при зміні dog на cog вийде:

SHA-256 ("The quick brown fox jumps over the lazy cog")

= E4C4D8F3 BF76B692 DE791A17 3E053211 50F7A345 B46484FE 427F6ACC 7ECC81BE

1.3.3. Опис сайтів, що здійснюють хеш-кодування MD5

http://www.md5.cz/

Набравши цю адресу, отримаємо сторінку, показану на рис. 1.5. цифрові виноски на рисунку мають пояснення в тексті нижче.

У вікно 1 було введено текст першого абзацу п.2.4. Тобто, абзац було відмічено, далі потрібно клацнути лівою кнопкою мишки у вікно, позначене як md5 і вставити текст. Потім натисну ти кнопку 2. В результаті, у полі 3 отримаємо код MD5, наприклад: e1918db55774b3c8939ed2049cded229.

Операції перенесення тексту на сайт та коду в текст здійснюються наступним чином: відмітити потрібний рядок символів мишкою, натиснути «Ctrl + C», перейти місця, куди його необхідно вставити й натиснути «Ctrl+ V».

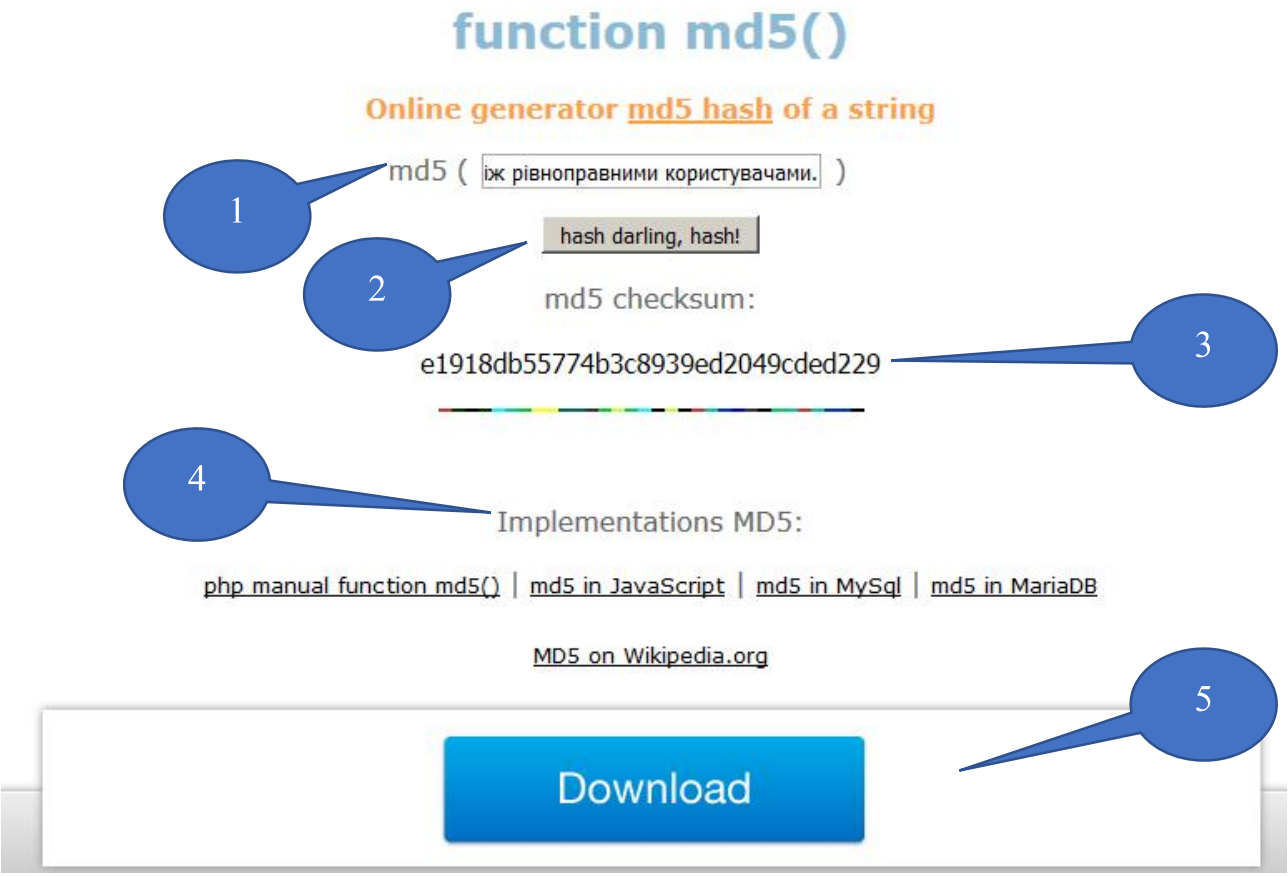


Рис. 1.5. Інтерфейс он-лайн генератора хеш-кодів

Пункт 4 пропонує вийти на інші сторінки цього сайту, щоб скачати коди генерації хешу MD5 для таких мов програмування як: PHP, JAVA, SQL, MariaDB, а також ознайомитися з алгоритмом генерації MD5 на сайті Вікіпедії.

Кнопка 5 пропонує скачати додаток до браузера FireFox, що дозволяє безплатно дивитися кінофільми та серіали англійською мовою (див. рис. 1.6), тому не потрібна для вивчення цього курсу. Варто зауважити, що сам браузер блокує спробу скачування. Можливо цей додаток містить загрози для вашого комп'ютера.

Рис. 1.6. Інтерфейс для скачування додатку, що дозволяє безплатно дивитися кінофільми англійською мовою

http://passwordsgenerator.net/md5-hash-generator/

Наступний сайт працює іще простіше (рис. 1.7). Досить ввести йому текст, як негайно генерується хеш-код у форматі MD5. Не потрібно навіть натискати кнопку «Generate».

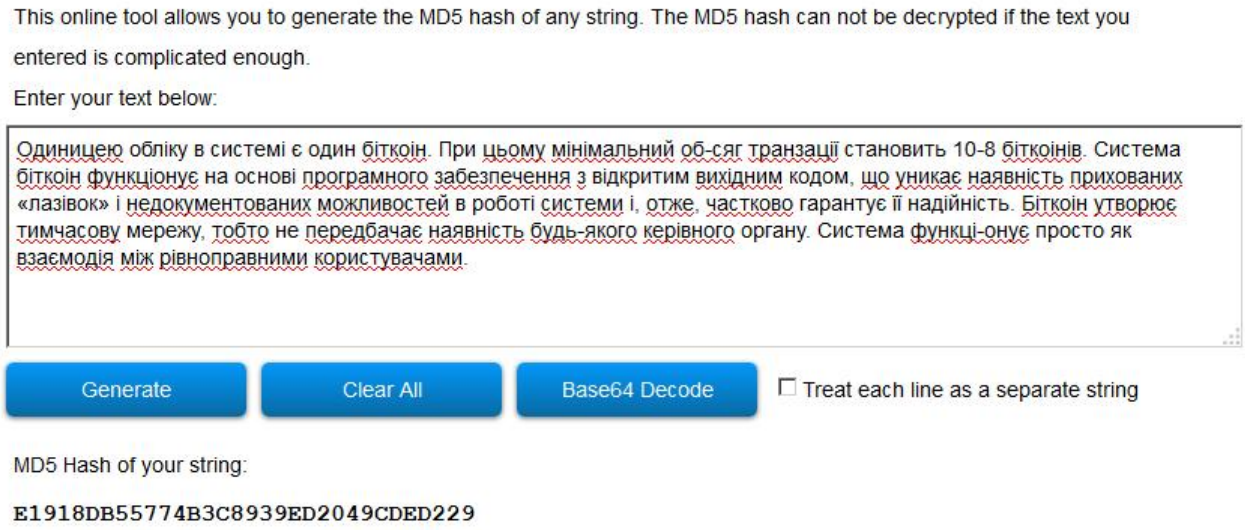


Рис. 1.7. Приклад роботи генератора хеш-кодів MD5

Кнопка «Base 64 Decode» викликає появу іще одного вікна (рис. 1.8), що-дозволяє провести кодування та декодування тексту, закритого хеш-кодом Base64.

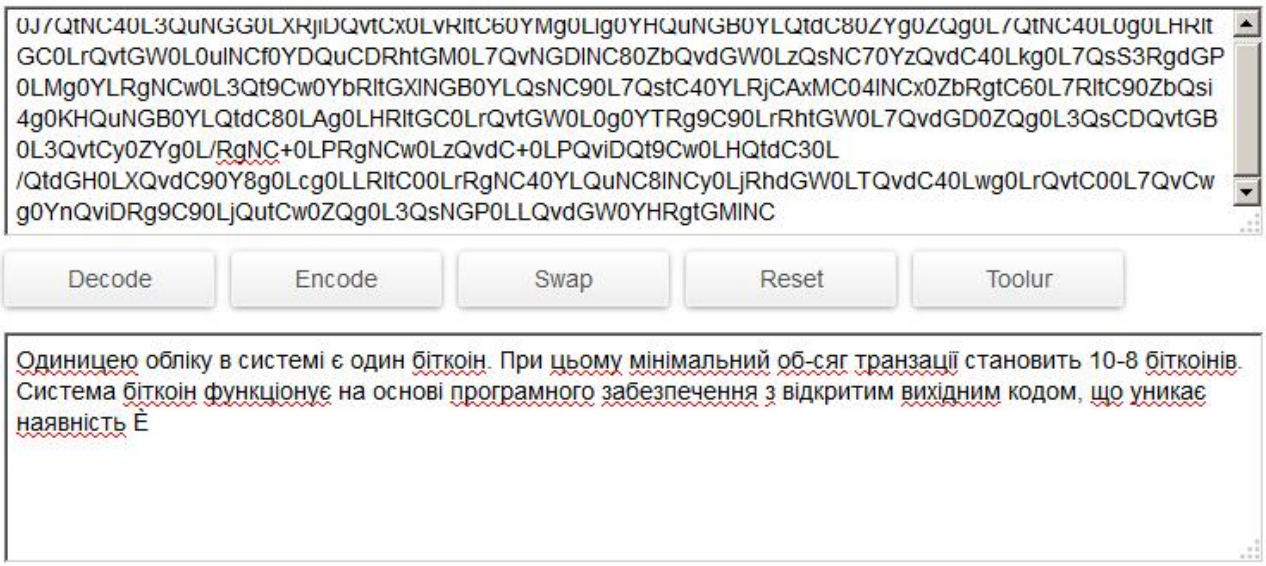


Рис. 1.8. Приклад роботи кодера-декодера хеш-кодом Base64

Цей код, на відміну від MD5, дозволяє провести декодування зашифрованого тексту. І отже, дає можливість тримати якісь тексти, адреси сайтів та поштові адреси у зашифрованому вигляді. У разі потреби цим текстам легко повернути попередній вигляд. Але при цьому розмір рядка тексту є обмеженою, що видно при порівнянні рис. 2.7 та рис. 2.8.

Для кодування тексту, його потрібно вставити у верхнє вікно і натиснути кнопку “Encode”. У нижньому вікні з’явиться закодований текст. Його можна зберегти у файлі будь-якої текстової форми. Для відновлення зашифрованого тексту, вставляємо у верхнє вікно шифр, натискаємо кнопку “Decode” – і отримуємо оригінал. Зверніть увагу, що декодування відбулося тільки двох перших рядків оригіналу, що говорить про обмеженість можливостей декодування.

Лівіше від вікон розташований список інших можливостей конвертації різних типів файлів у інші, а саме: Web Proxy, MP3 Cutter, Alarm Clock, Photo Resizer, Password Generator, GIF Maker, Images to GIF, Photo Compressor, Base64

Decode, Convert Case , Word Counter, ICO Convert, EXIF Viewer, PDF to JPG, MP3 Converter.

http://www.miraclesalad.com/webtools/md5.php

Цей генератор працює аналогічно попереднім (рис. 1.9). Достатньо увести рядок тексту, як видається хеш-код MD5.

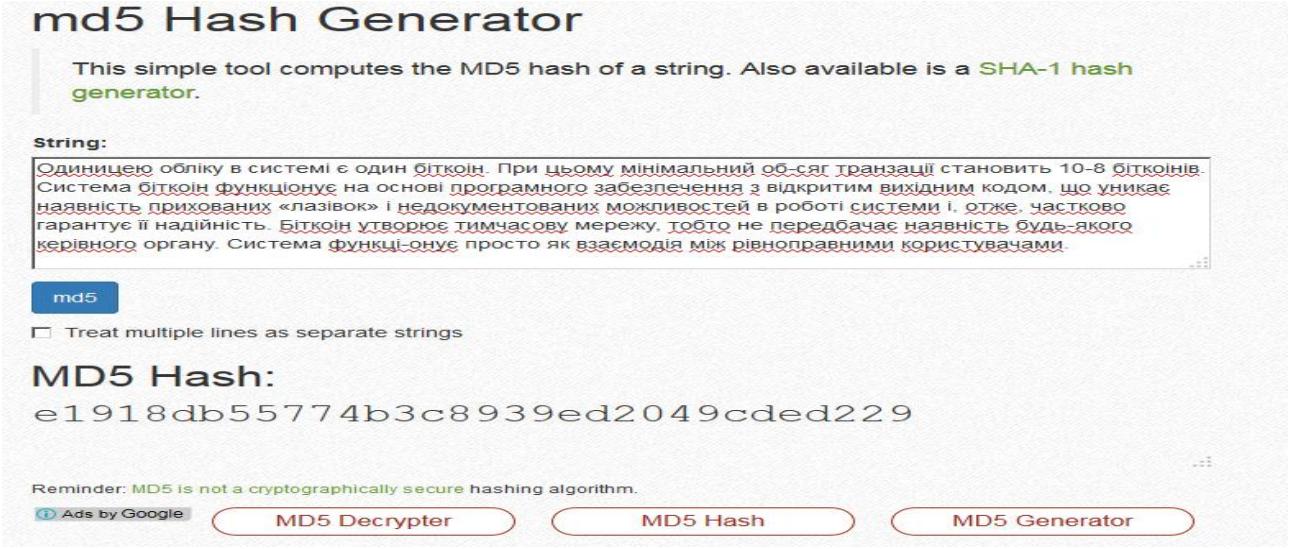


Рис. 1.9. Приклад створення хеш-коду MD5

1.3.4. Опис сайтів, що здійснюють хеш-кодування SHA-256

Як було зазначено вище, хеш-код MD5 не забезпечує повної гарантії щодо можливостей його розкриття. Більш надійним вважається сімейство кодів SHA-2. Саме цими кодами користуються зараз біржі криптовалют. Розглянемо деякі сайти, що забезпечують саме таке кодування.

http://www.xorbin.com/tools/sha256-hash-calculator

Я видно з рис 1.10, цей код значно довший, а отже, тільки за цією ознакою вже можна вважати, що він надійніший за MD5. Однак, спеціалісти стверджують, що у ньому значно менше можливостей до його розкриття чи підробки.

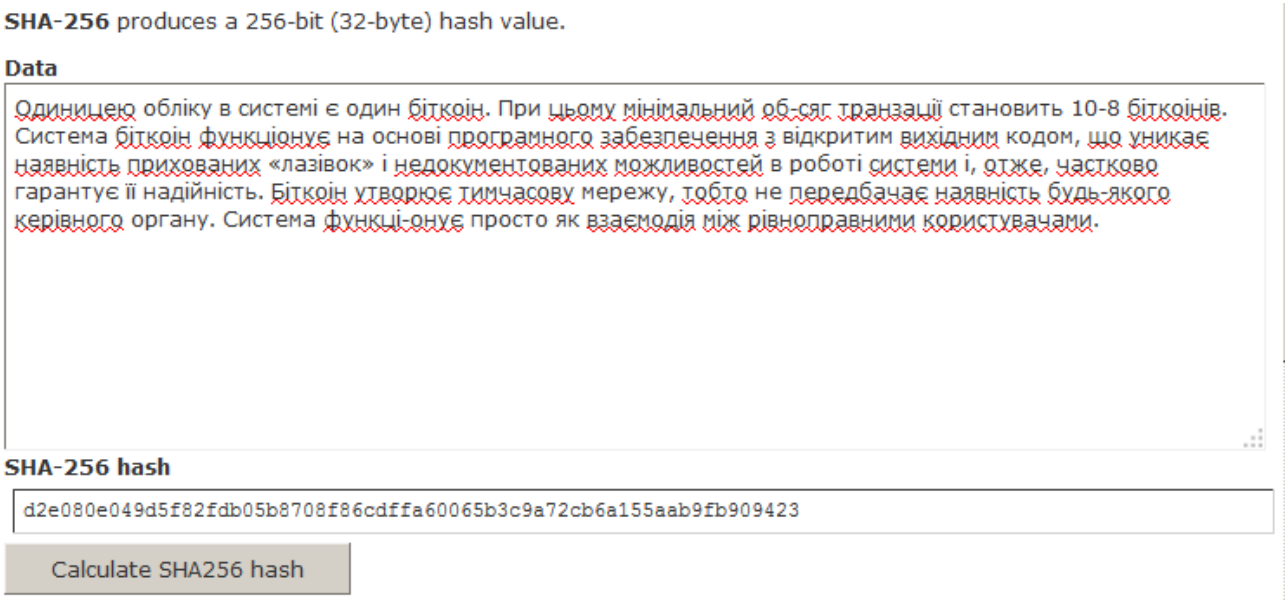


Рис. 1.10. Приклад створення коду SHA-256

Зверніть увагу і на те, що для генерації коду обов'язково потрібно натиснути кнопку «Calculate SHA256 hash».

http://passwordsgenerator.net/sha256-hash-generator/

Приклад роботи он-лайн-генератора наведено на рис. 1.11.

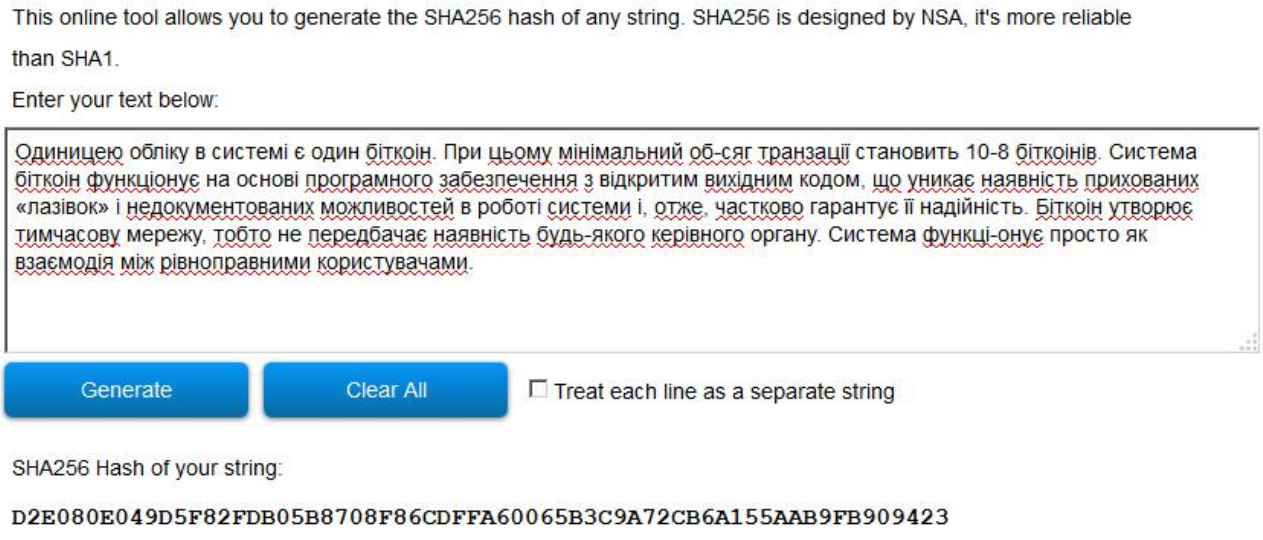


Рис. 1.11. Приклад генерації коду SHA-256

Для цього сайту навіть не потрібно натискати кнопку «Generate». Створення коду відбувається негайно після введення тексту у вікно під написом «Enter your text below».

http://hash.online-convert.com/ru/sha256-generator

Цей сайт є русифікованим і надає більше можливостей для створення хеш-кодів. Його інтерфейс подано на рис. 1.12.

Так, для введення тексту, який необхідно закодувати, є можливість не тільки його вставити з оперативної пам'яті (поле 1). Текст можна ввести з файлу, розташованому на вашому комп'ютері (поле 2) або зі хмарного сховища (поле 4). Причому, в останньому випадку система дозволяє вибирати як зі сховища Гугл так і зі сховища Дропбокс. Поле 3 показує можливість вставлення створеного хеш-коду у вказану користувачем сторінку Інтернету. В полі 5 існує можливість до вашого тексту додати іще секретний код, який унеможливить розшифровку коду base64.

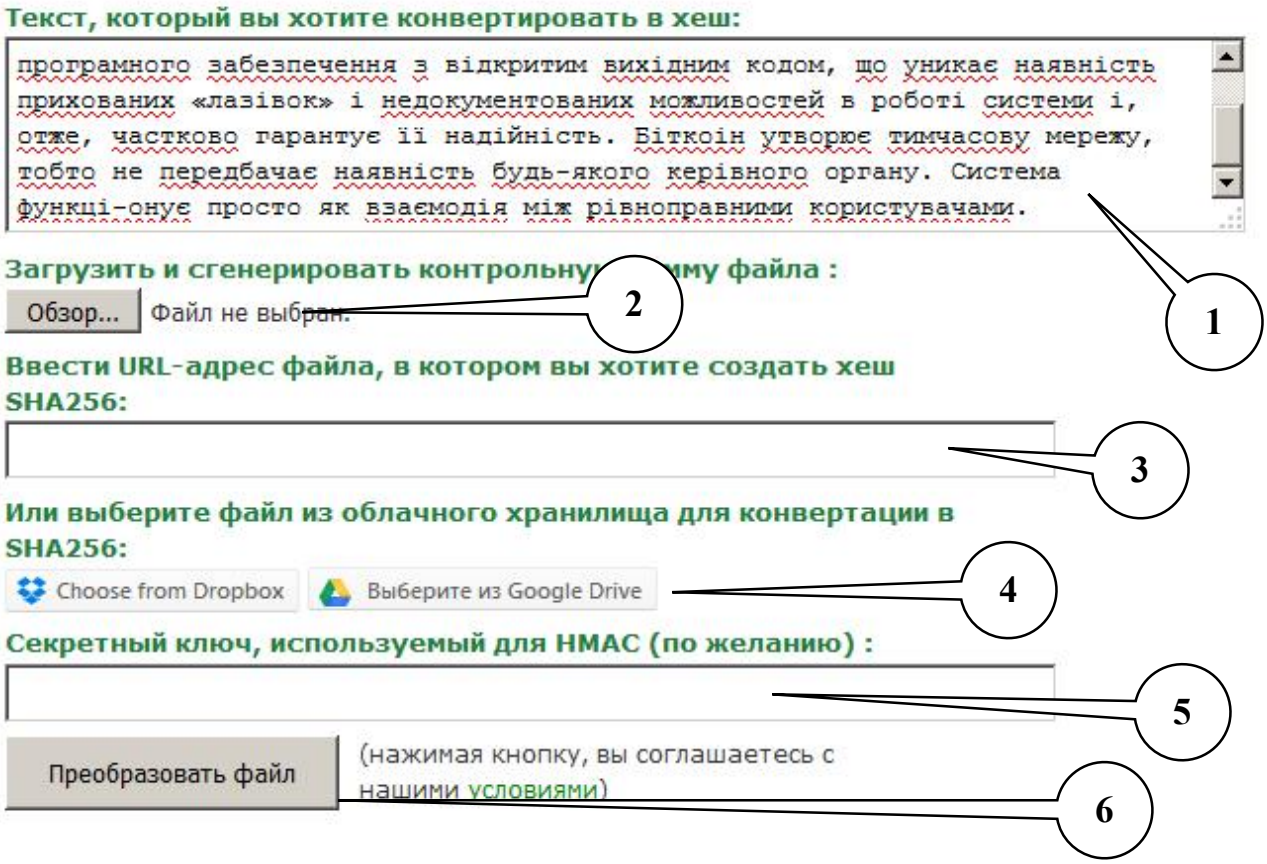


Рис. 1.12. Інтерфейс сайту для генерації коду SHA-256

Коли після вводу тексту ви натиснете кнопку 6, відкриється нове вікно (рис. 1.13), де окрім коду SHA-256, розписаного як малими так і великими літерами латинського алфавіту, є кодування в base64.

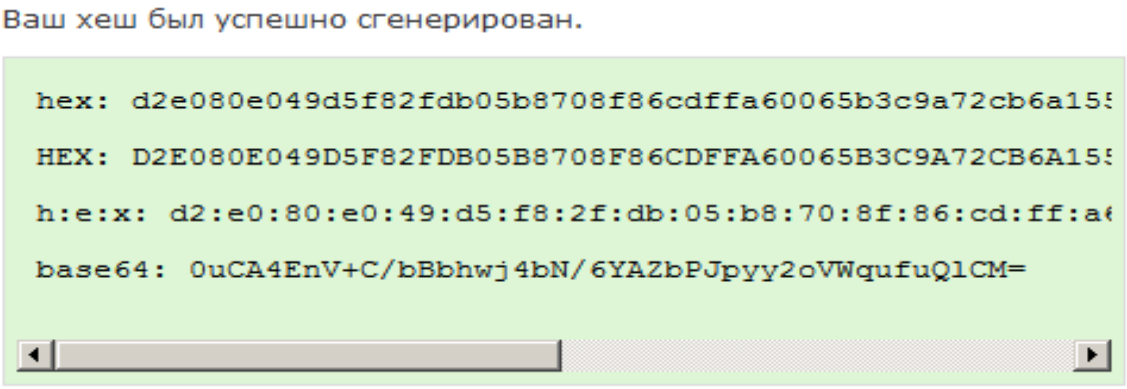


Рис. 1.13. Приклад створеного хеш-коду SHA-256

Для перевірки того, як секретний ключ змінює хеш-код, для того ж тексту, було введено ключ 123456789. Нижче показано новий хеш-код SHA-256. Перевірка навіть перших символів показує, що код змінився кардинально.

af2b5fc86482f103b55a909882be0ef87de812c73f33654ec9778a59be5b4172

1.4. Криптовалюта Біткоїн

Одиницею обліку в системі є один біткоїн. При цьому мінімальний обсяг транзакції становить 10-8 біткоїнів. Система біткоїн функціонує на основі програмного забезпечення з відкритим вихідним кодом, що уникає наявність прихованих «лазівок» і недокументованих можливостей в роботі системи і, отже, частково гарантує її надійність. Біткоїн утворює тимчасову мережу, тобто не передбачає наявність будь-якого керівного органу. Система функціонує просто як взаємодія між рівноправними користувачами.

Біткоїни не залежать від будь-якого центру, що займається емісією валюти. Емітувати валюту може будь-який користувач системи. дані про переміщенні грошових коштів зберігаються в розподіленій базі даних, та на комп'ютерах користувачів. Синхронізація бази даних між користувачами проводиться автоматично з використанням технології, побудованої на пірінговому мережевому

протоколі. Надійність системи гарантується використанням криптографічних засобів захисту. Біткоїни можуть бути відправлені будь-якому користувачеві мережі з використанням адреси Bitcoin. Однак ця адреса гарантує повну анонімність, оскільки представлять собою просто комбінацію букв і цифр, наприклад таку: «1D5wZqCjxNuPqfUN3RMFsxxxtqRBwiAeTZ». Існуюча база даних біткоїнів зберігає інформацію про всі транзакції, тобто переміщенні грошових одиниць з однієї адреси на інший і, відповідно, кінцеве сальдо адрес.

Користувач системи має у себе файл, який є ключем з адресами до біткоїнів, що належить йому. З'єднання цього ключа з базою даних за допомогою спеціальної програми-гаманця через мережу інтернет, дозволяє дописувати записи про транзакції в базу даних, тобто проводити платежі. Платіж проводиться миттєво, однак для повної впевненості отримання біткоїнів необхідно дочекатися так званих «Підтверджень». В результаті платіж може займати від 10 хвилин до 1 години, що істотно швидше банківського переказу, проте поступається швидкості проведення операцій в централізованих системах електронних грошей (Webmoney, QIWI і т.п.), де транзакції займають лічені секунди. система біткоїнів не передбачає обов'язкових комісійних зборів.

З точки зору користувача біткоїн – це система електронної готівки. Під цим розуміється той факт, що втрата файлу-ключа тягне за собою втрату біткоїнів, оскільки доступ до відповідних адрес буде неможливий. Зберігання файлу аналогічно зберіганню грошей у гаманці. Крадіжка файлу-ключа може також призвести до втрати біткоїнів.

Говорячи про біткоїни, як про систему електронної готівки, потрібно пам’ятати, що це лише метафора. Отримання біткоїнів на свою адресу не вимагає підключення програми-гаманця з ключем до інтернету. Усе це потрібно тільки для того, хто платить, але не для того, хто отримує.

Саме платник виробляє додаткові записи в базу даних про транзакції. Принцип тимчасової мережі та відсутність єдиного центру унеможливлює державне втручання і маніпуляції курсом через зміну грошової маси. В даний час система знаходиться в стадії інфляційного розширення. Це означає, що кількість

біткоїнів збільшується. Емісію біткоїнів може здійснювати будь-який користувач мережі. Однак з цього не випливає, що будь-який користувач здатен згенерувати нескінченну кількість біткоїнів.

Таблиця 1.1

Максимальна кількість біткоїнів в обороті

Дата	Кількість біткоїнів в обороті	Темп приросту за рік,%
Січень 2010	2,63 млн.	-
Січень 2011	5,25 млн.	100,00%
Січень 2012	7,88 млн.	50,00%
Січень 2013	10,50 млн.	33,30%
Січень 2014	11,81 млн.	12,50%
Січень 2015	13,13 млн.	11,10%
Січень 2016	14,44 млн.	10,00%
Січень 2017	15,75 млн.	9,10%
Січень 2018	16,41 млн.	4,20%
Січень 2019	17,06 млн.	4,00%

Як працюють біткоїни? Кожен учасник системи встановлює собі на комп'ютер програму-клієнт (https://bitcoin.org/en/choose-your-wallet). Інтерфейс сайту показано на рис. 1.14.

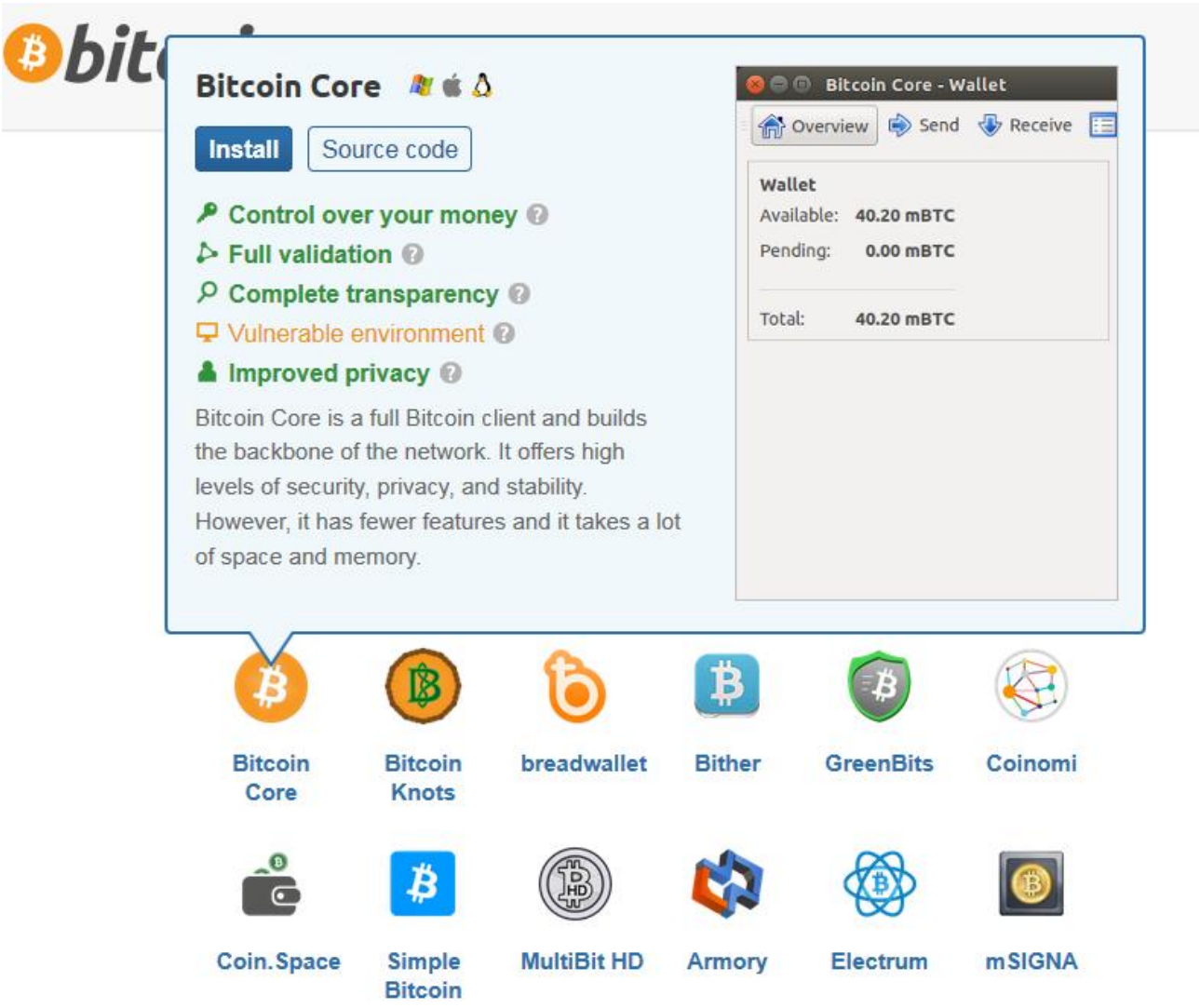
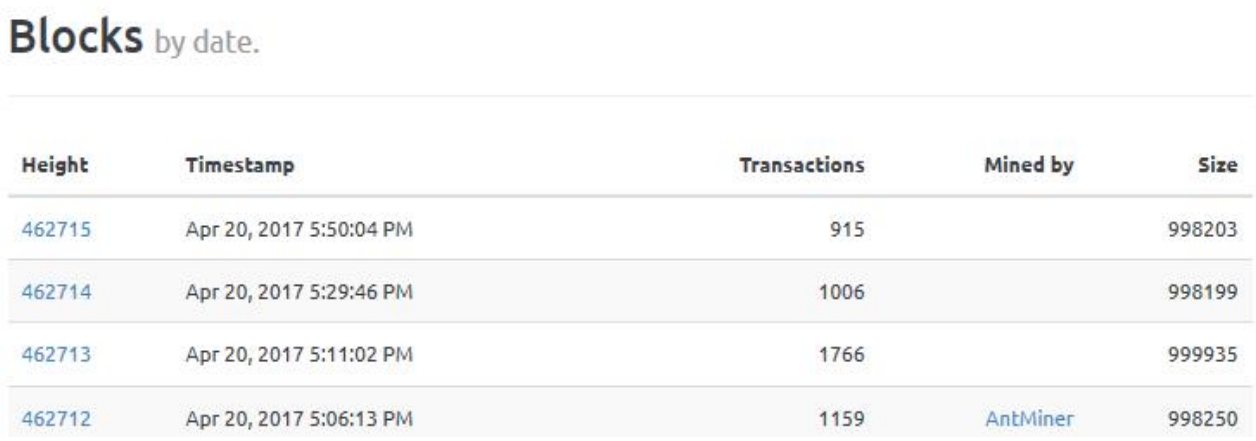


Рис. 1.14. Інтерфейс сайту, що пропонує різні програми, доступні для встановлення на вашому комп'ютері. Наразі була вибрано програма Bitcoin Core

Обравши якусь програму, ви бачите її інтерфейс та читаєте опис. Якщо вам підходить, тиснете на кнопку «Install». Інша кнопка «Source Code», дає можливість ознайомитися з текстом цього «електронного гаманця». Саме так перекладається назва програм, що працюють з криптовалютами.

Усі транзакції зберігаються в ланцюжку блоків (https://blockexplorer.com/blocks), кожен блок містить список транзакцій (рис. 2.15). Ланцюжок блоків містить інформацію про усі транзакції за увесь час роботи Біткоїн. Кожна програма-клієнт завантажує увесь ланцюжок блоків – це і робить систему абсолютно децентралізованою. Майте на увазі, цей процес може

тривати досить довго (2 дні наприклад) і добряче завантажувати ваш комп'ютер. Ці дані ніяк не шифруються і кожен може побачити усі транзакції. Кількість блоків зростає на 1 приблизно кожні 10 хвилин.



Blocks by date.

Height	Timestamp	Transactions	Mined by	Size
462715	Apr 20, 2017 5:50:04 PM	915		998203
462714	Apr 20, 2017 5:29:46 PM	1006		998199
462713	Apr 20, 2017 5:11:02 PM	1766		999935
462712	Apr 20, 2017 5:06:13 PM	1159	AntMiner	998250

Рис. 1.15. Список блоків, що діють в системі біткоїн

Якщо клацнути мишкою на будь-який блок, ви побачите (рис. 1.16) поточний стан цього блоку.

Коли хтось створює нову транзакцію, його клієнт розсилає її іншим клієнтам – які зайняті генерацією поточного блоку. Вони додають цю транзакцію до блоку і продовжують генерацію – доки блок повністю не згенерується. Такий блок запечатується (до нього більше не можна додавати транзакції) і розсилається по мережі. Клієнти мережі перевіряють блок і транзакції, які до нього записані, на валідність. Якщо проблем немає – транзакція вважається ухваленою. Цей блок отримує кожен клієнт і додає до ланцюжка блоків. Після чого клієнти починають генерувати новий блок і процес іде по колу.

Транзакції містяться в блоках у вигляді списку, вони теж вибудовуються в ланцюжки. Кожна транзакція має вказувати звідки вона бере гроші (з якої існуючої транзакції) і куди направляє. Для вказання адресата використовується пуб-

лічний ключ. Щоб використати отримані кошти, потрібно створити нову транзакцію, яка братиме гроші з попередньої і перенаправлятиме їх до наступного адресу. Щоб підтвердити право використання цих коштів, вам потрібно свою транзакцію скріпити цифровим підписом. Так в будь-який момент часу можна пересвідчитись, що усі транзакції в системі є валідними, тобто, вірними.

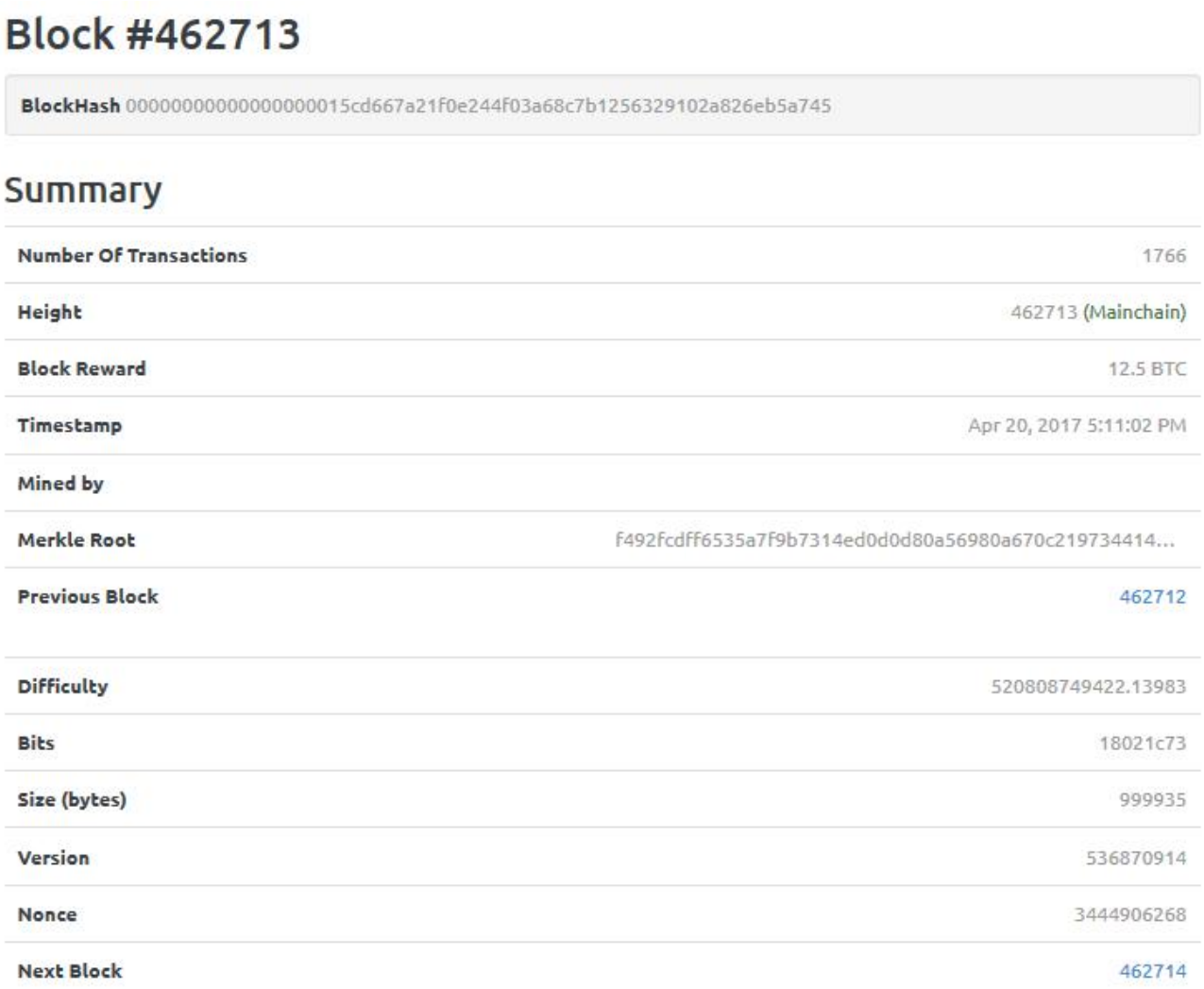


Рис. 1.16. Поточний стан блоку №462713 в системі Біткоїн

На сайті http://search.soft112.com/ можна знайти також вільні для використання програми електронного гаманця, де база розміщається на вашому комп’ютері.

1.4.1. Генерація нових грошей в системі Біткоїн

У кожному блоці перша транзакція у списку – особлива. У неї є лише один вхід, вона перенаправляє 25 монеток тому, хто згенерував блок, у якому розташована ця транзакція. Відповідно, приблизно кожні 10 хвилин система поповнюється 25 монетками. Це така собі винагорода за витрачений час і ресурси на генерацію блоку. Адже створюючи новий блок у ланцюжку клієнт робить внесок у роботу системи Bitcoin. Кожні 4 роки ця винагорода зменшуватиметься удвічі: власне, один раз вона вже зменшилася – раніше за генерування блоку винагорода становила 50 монеток.

Якщо хтось створить шахрайський блок за допомогою модифікованої версії клієнта і припише собі 25 монеток – інші (справжні, чесні) клієнти відторгнуть такий блок і не додадуть його до ланцюжка блоків. Тобто доки в системі більшість клієнтів офіційні (не шахрайські) – системі нічого не загрожує.

Біткоїни можна придбати за гроші – а можна «видобувати», по аналогії із золотом (до речі, багато хто порівнює біткоїни із золотом через те, що і те і те важко видобувати і кількість їх однаково обмежена). Генерування блоків зі списком транзакцій – навмисне складний процес, який вимагає значних затрат розрахункових можливостей. І складність його регулюється із тим, щоб швидкість генерування блоків на одиницю часу не змінювалася. Щоб легше зрозуміти, що це таке – згадайте, що таке прості числа. Число, яке дітися без остачі лише саме не себе і на одиницю. Це цифри 1, 3, 5, 7… 907, 911, 919 – чим далі, тим рідше вони зустрічаються. Наприклад, за останні 2 роки було знайдене лише одне нове просте число. Наступне орієнтовно можуть знайти теж за кілька років.

Така ж ситуація з біткоїнами – це фактично унікальний набір цифр, і знаходження нового унікального набору з кожним днем стає все важче. Складність генерування блоків наразі настільки ускладнилася, що індивідуально «видобувати» монетки абсолютно невигідно. Тому для «видобування» монеток-біткоїнів люди почали об’єднуватися в «пули». Англійською це називається «poole min-

ing». Наприклад, ви можете долучитися до одного з найпопулярніших наразі пулів із «видобування» монеток – mining.bitcoin.cz. Або долучитися до www.bitcoinplus.com.

На рис. 1.17 подана статистика активності «видобувачів» для пулу mining.bitcoin.cz. Цю статистику можна вивчити, натиснувши кнопку «Изучить Live Demo».

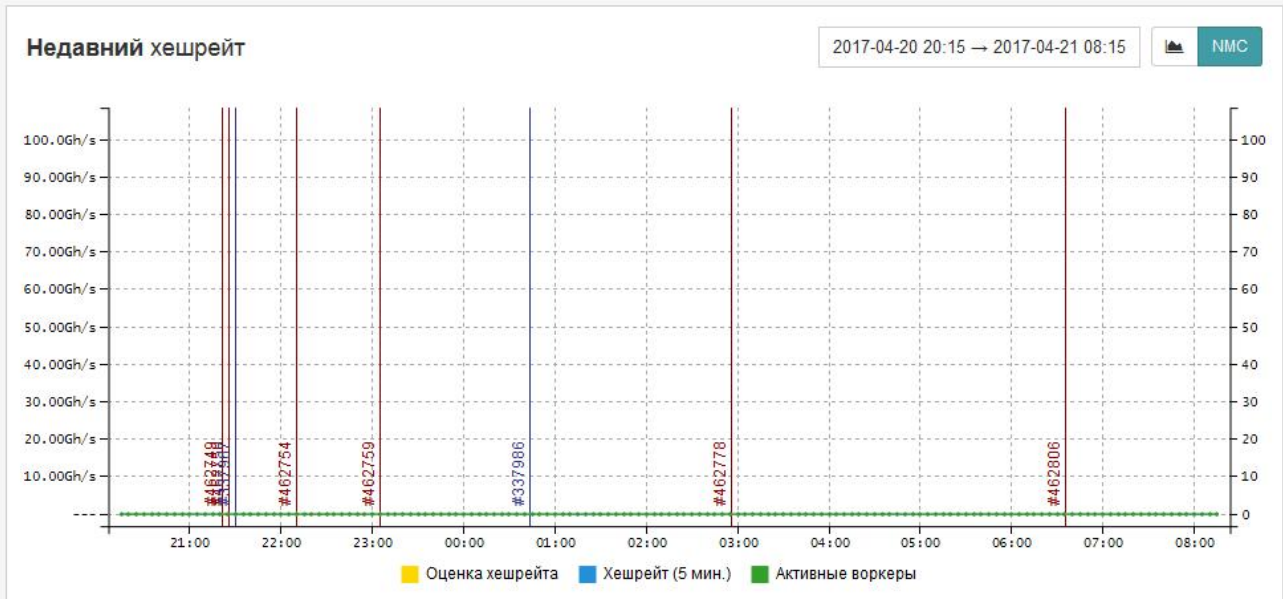


Рис. 1.17. Статистика з сайту mining.bitcoin.cz

Вертикальні червоні лінії – це моменти утворення нового блоку, на якому вказується його номер. Як видно з графіку, протягом доби було утворено 7 нових блоків.

На рис. 1.18 показано графік виплат за участь у пулі. Доступ до нього здійснюється через кнопку «Выплаты».

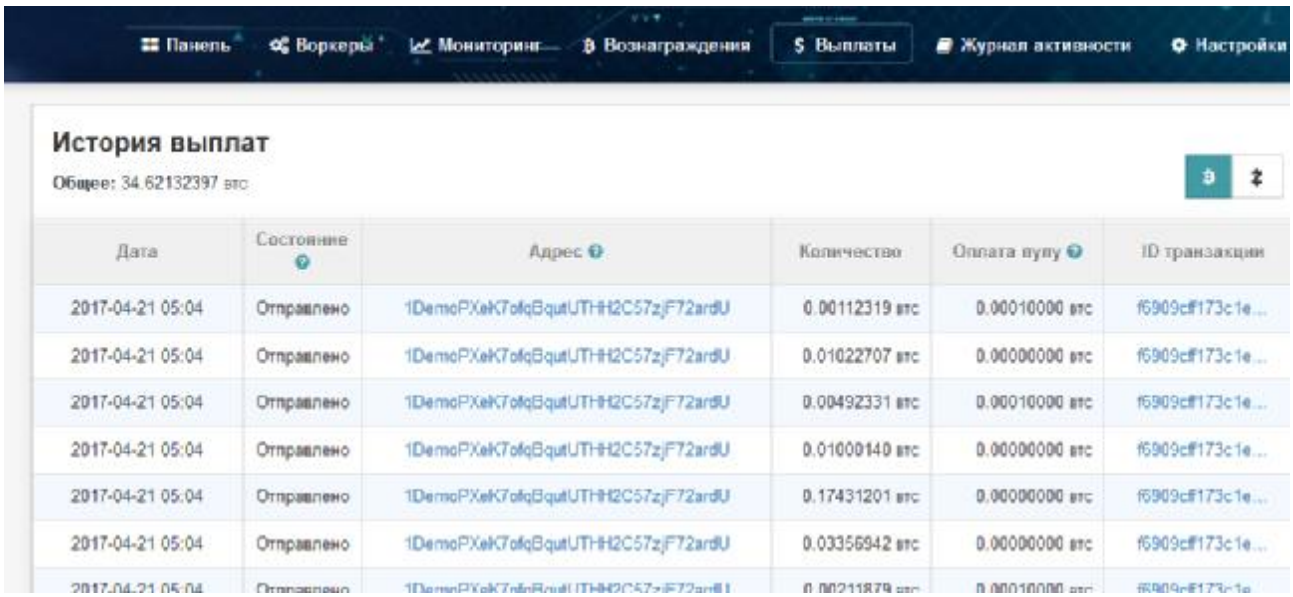


Рис. 1.18. Графік виплат «видобувачам» пулу mining.bitcoin.cz

Окрім біткоїнів цей пул обслуговує і криптовалюту Zcash.

1.4.2. Основні поняття щодо «пулів видобутку» біткоїнів

Пул видобутку – підхід, при якому кілька генеруючих клієнтів сприяють генерацій блоку, а потім розділяють нагороду блоку згідно внесеної обчислювальної потужності.

Зі збільшенням кількості згенерованих блоків зростають труднощі «видобування» нових на комп'ютерах малої продуктивності. Процес «видобутку» може зайняти дуже багато часу, перш ніж буде згенеровено нове покоління блоків. Наприклад, при швидкості «видобутку» 1000 КГ/сек, при складності 14484 (яка була дійсною в кінці грудня 2010 року), середній час генерації блоку складе майже 2 роки.

Для того, щоб більше стимулювати «видобувачів», що мають низьку продуктивність, кілька «пулів видобувачів» використовують різні підходи. У процесі «видобування» багато різних людей, сприяють генерації блоку, і нагорода потім ділиться між ними відповідно до їх вкладу. Таким чином, замість того, щоб

чекати протягом багатьох років, щоб зробити 50 BTC за генерацію блоку, «видобувач» може отримати частку в Bitcoin на більш регулярній основі.

Узагальнені підходи «видобування».

Проблема з узагальнені «видобутку» є те, що повинні бути вжиті заходи для запобігання обману клієнтами і сервером. В даний час існує декілька різних підходів:

– Підхід Slash або Bitcoin Pooled Mining (BPM), іноді званий «Slash пул» базується на основі методів оцінки внеску членів пулу. Старі акції (від початку раунду генерації блоку), мають меншу вагу, ніж більш пізні акції, що знижує мотивацію обдурити шляхом перемикання між «пулами» в межах раунду.

– Підхід Pay-per-Share. (ПФС) пропонує миттєві негайні виплати за кожну акцію, за домовленістю між учасниками. Виплати пропонуються від існуючого балансу пулу і, отже, можуть бути зняті негайно, не чекаючи появи нового блоку або попереднього уже підтвердженого блоку. Можливість обману «видобувачів» від оператора пулу і синхронізації атак, таким чином, повністю виключається.

Цей метод призводить до мінімально можливій дисперсії для майнерів при передачі всіх ризиків оператору пулу. В результаті чого можливість втрати для сервера компенсується шляхом установки виплати нижче повного очікуваного значення.

– Підхід Luke-Jr's є запозиченням методів від попередніх двох. Як підхід BPM, «видобувачі» надають докази обсягу своєї роботи, щоб заробити акції. Як підхід ППФС, в пулі виплачують відразу через покоління блоків. При розподілі блоків нагороди, вона ділиться порівну між усіма акціями з моменту появи дійсного (підтвердженого) блоку.

– Комплексний підхід Pay-per-Share (ППФС) створений командою BTC.com, спрямована на користь «видобувачів» через високу вартість угоди. Він розрахує стандартну плату за транзакцію протягом певного періоду – 12,5 BTC

за кожен блок в даний час – а потім розподілити між усіма «видобувачами» відповідно до угод.

Цей метод зберігає переваги ПФС, але виплати збільшені, бо «видобувачі» розділяють плату за транзакцію. Це збільшить доходи «видобувачів» на 3% –11%.

На відміну від будь-якого раніше існуючого підходу, це означає, що акції тих, хто сприяв появі старих блоків переробляються в акції наступного блоку. Для того, щоб заощадити витрати від зборів угоди, в якій беруть участь «видобувачі», винагорода виплачується тільки якщо «видобувач» заробив принаймні 0.67108864 BTC (400 TBC). Якщо сума належних виплат менше, вона буде додана до прибутку більш пізнього блоку (яка потім може стати більшою за порогову суму). Якщо «видобувач» не представляє частку протягом тижня, пул надсилає залишок коштів, незалежно від його розміру.

– Підхід P2Pool (P2P) «видобувачі» працюють по ланцюжку акцій, подібних до організації системи blockchain криптовалюти Bitcoin. Коли блок знайдений, нагорода ділиться між останніми акціями цією часткою участі. Як і в підходах ПТФС, Luke-Jr's та P2P виплата здійснюється через покоління.

Порівнюючи різні підходи в організації пулів, можна сказати наступне.

Кооперативний «видобуток» для підходів BPM і Luke-Jr's використовує набагато менше ресурсів на сервері пулу, оскільки замість того, щоб постійно перевіряти хеш-коди, що генеруються у процесі пошуку нових блоків, все, що має бути підтверджено – це достовірність поданих акцій. Кількість акцій, які були надіслані можна регулювати шляхом регулювання штучного рівня складності.

Крім того, підхід кооперативного «видобутку» дозволяє клієнтам використовувати існуючих майнерів без будь-яких змін, в той час як підхід P2P вимагає призначення для користувача конкретного пулу «видобувачів», які уклали договір з цим пулом – це не так ефективно у плані «видобутку».

У підході P2P «видобувачі» отримують монети безпосередньо, що виключає затримку в отриманні прибутку від пулових серверів. Проте, через деякі послуги для електронного гаманця, згенеровані монети можуть бути втрачені.

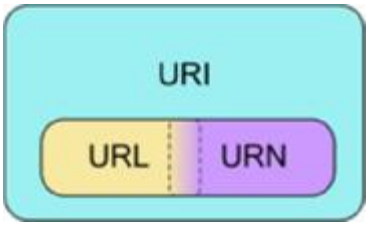
1.4.3. Опис типового електронного гаманця біткоїнів

Для початку ознайомимося з поняттям, яке буде необхідне для користування електронним гаманцем.

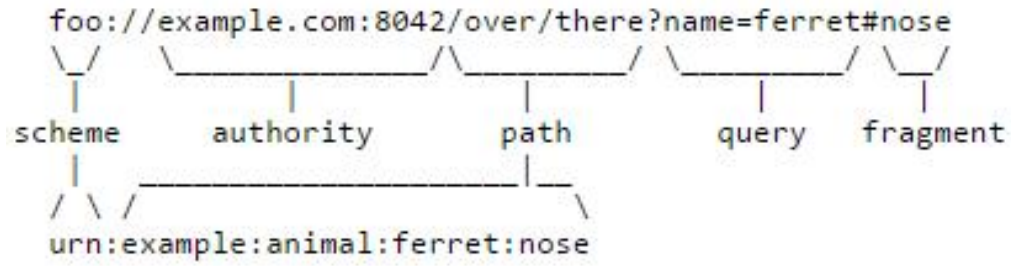
Уніфіко́ваний ідентифіка́тор ресу́рсів (англ. *Uniform Resource Identifier*, URI) – компактний рядок літер, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс. Основне призначення таких ідентифікаторів — зробити можливою взаємодію з представленнями ресурсів через мережу, переважно Всесвітнє павутиння, використовуючи спеціальні протоколи. URI визначається схемами, які визначають синтаксис та відповідні протоколи. Найбільш поширеною формою URI є уніфікований локатор ресурсів (URL), який неофіційно називають *веб-адресою*. Рідше використовується уніфіковане ім'я ресурсів (URN), яке було розроблене, щоб доповнити URL забезпеченням механізму для ідентифікації ресурсів в просторі імен.

Стаття 1 Закону України «Про телекомунікації» визначає термін «Адреса мережі Інтернет» як визначений чинними в Інтернеті міжнародними стандартами цифровий та/або символьний ідентифікатор доменних імен в ієрархічній системі доменних назв.

Раніше URI називався *Universal Resource Identifier — універсальний ідентифікатор ресурсів* (див. RFC 1630 та http://www.w3.org/Addressing/URL/uri-spec.html). Діаграма Венна на якій зображено підмножини схем URI: URL та URN демонструє рівень вкладеності звичайної адреси в Інтернеті та додатку, що складає ім’я конкретної людини.



За наведеною нижче схемою видно, як адреса URL та ім’я, перетворюються на URI. Фактично, через двокрапку потрібно записати ці елементи.



Причому елемент «over/there?name=» випускається повністю. Інші елементи розділяються двокрапкою.

Для прикладу, з сайту https://bitcoin.org/en/choose-your-wallet була взята програма Bitcoin Core. Вона постачається у zip-файлі і не потребує інсталяції. Одразу після розпаковки готова до роботи. Стартовою є програма …/bin/bitcoin-qt.exe.

На рис. 2.19 показано головне меню програми. Одразу після запуску програма починає процес синхронізації з мережею Bitcoin. Час на таку синхронізацію –10-14 годин. По її закінченні клієнт отримує власний URI та має можливість здійснювати транзакції наявних сум біткоїнів (вони позначаються як BTC). Процес синхронізації показується у пункті “Progress”, а час, який залишився до кінця синхронізації – у пункті “Estimated time left until synced”.

Програма автоматично визначає основну мову на вашому комп’ютері і подає інтерфейс саме в ній.

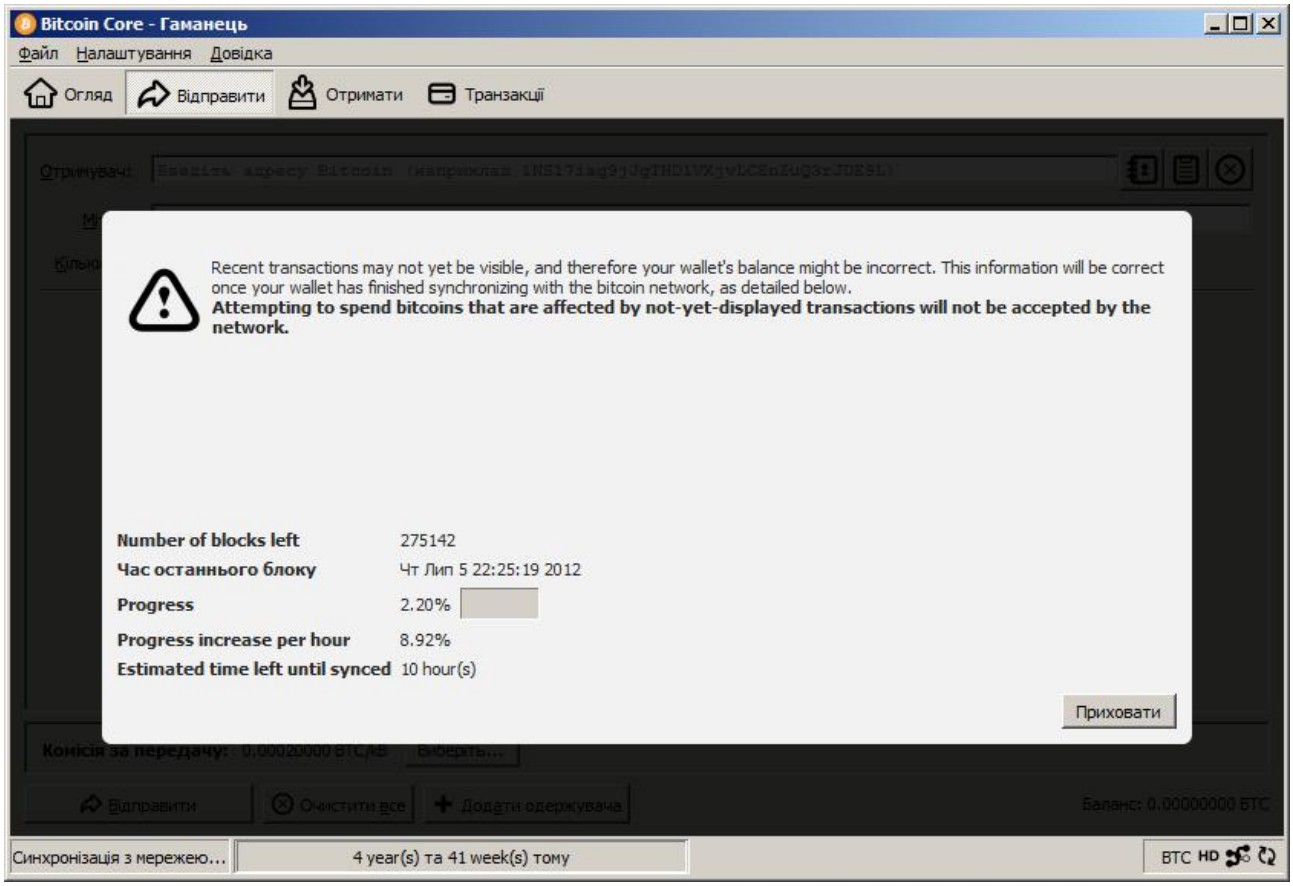
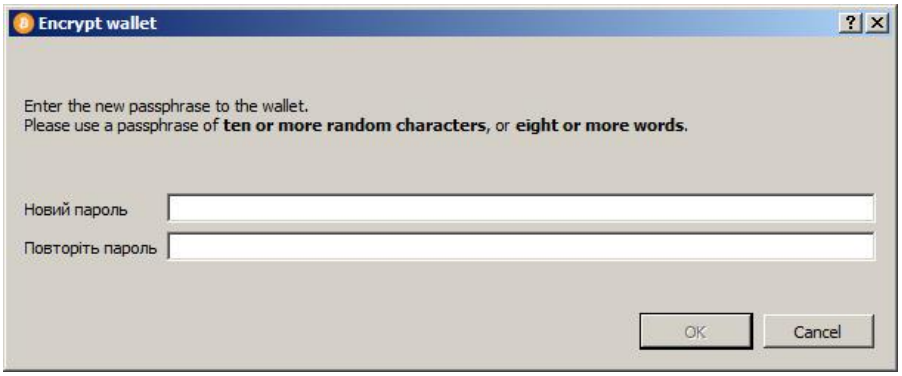


Рис. 1.19. Інтерфейс програми Bitcoin Core в процесі синхронізації

В пункті меню «Налаштування» потрібно вибрати пункт «Параметри» для визначення ваших можливостей, хоча стандартні налаштування там вже є. Так, зокрема, стандартно система вимагає 300 МБ на жорсткому диску для встановлення туди бази даних про транзакції біткоїнів. Причому, автоматично визначає, що ця база повинна міститися на диску C:, що не завжди зручно. Також, система автоматично визначає вашу IP-адресу та порт, через які будуть здійснюватися операції з біткоїнами.

У тому ж пункті потрібно налаштувати свій пароль, який має складатися не менше, аніж з 10 символів, обраних випадковим чином, або не менше восьми слів, якщо це якась фраза.



1.4.4. Детальний опис операцій з майнінгу

З плином часу Proof-of-Work примушує майнерів до кооперації, що погано для індивідуалів, але добре для всієї мережі в цілому. Примус до об'єднання обчислювальних потужностей майнерских ферм по всьому світу і їх організація в пули дозволяє всій мережі придбати велику впорядкованість, при цьому не втрачаючи децентралізації: нехай пули виступають на програмному рівні в якості «єдиної обчислювальної потужності», фізично ферми можуть мати абсолютно різну географію. Також це спрощує процес підтримки і розвитку блокчейн-проекту: для прийняття тих чи інших змін у технологію необхідна згода більшості учасників. Пули можуть приймати рішення колективно, всередині власної структури, що спрощує життя розробникам.

Крім цього, PoW забезпечує безпеку всієї мережі на достатньому рівні. Всі обчислені майнерами блоки збираються в один ланцюг - блокчейн. Кожен новий блок містить в собі запис про попередні блоки. У теорії, хтось може створити іншу «гілка» ланцюга, але для цього в системі, яка дотримується PoW, доведеться повторити з самого початку все зроблені розрахунки. З урахуванням нинішнього рівня розвитку системи основних криптографічних валют, це малоймовірно.

Найбільший його мінус PoW в тому, що з ростом складності обчислень виключає з екосистеми такі елементи, як приватні майнери з персональними комп'ютерами. На прикладі біткоїни, коли просто ПК стало не вистачати, ентузіасти почали збирати «домашні ферми», а в якості обчислювальних блоків використовувати відразу кілька ігрових відеокарт, які більш пристосовані до маніпуляцій подібного роду, ніж центральні процесори. Цього рішення вистачило ненадовго, і в підсумку на ринку з'явилися спеціалізовані апаратні платформи з власною логікою, адаптовані спеціально під майнинг тих же біткоїни. Саме це спеціалізоване ASIC-обладнання, яке не придатне більше ні для яких інших цілей, окрім розшифровки криптографічного ключа блоку, зараз і використовується безліччю промислових ферм по всьому світу. Однак це не означає, що мільйони ферм на

базі відеокарт вирушили на звалище: зараз вони використовуються в блокчейнах з меншою складністю, де їх використання ще вигідніше дорогих інтегральних схем ASIC. І якщо ферми з відеокарт можна було розмістити у себе під столом або в шафі, то промислові ферми на базі ASIC-рішень можна порівняти за розмірами з дата-центрами і споживають відповідну кількість електроенергії. До 2020 року на Майнінг біткоїни в рік буде витрачається стільки ж електроенергії, скільки споживає вся Данія.

Все обладнання, що використовується в майнінгу сильно гріється, а надлишки тепла викидаються в атмосферу. Зараз майнінг криптографічних токенів за принципом PoW схожий на примітивний видобуток золота: величезні суми йдуть на закупівлю спеціального обладнання і електроенергію. У перспективі рентабельність майнінгових ферм можна підвищити за рахунок їх перенесення в Ісландію і на полюса, або ж в пустелі, наприклад, в Сахару. У першому випадку, це дозволить знизити витрати на охолодження обладнання (а для Ісландії можливий видобуток геотермальної енергії), а в другому - на електриці за рахунок сонячних батарей.

Зараз за статистикою на головній сторінці порталу Bits.media, сукупно все майнери біткоїнів виробляють близько 5 x 10^{15} операцій додавання і перевірки. При цьому по всьому світу загальна швидкість розшифровки впала вже до 6,29 блоку / рік.

Саме тому блокчейн-спільнота, яке підхопило ідею розвитку криптовалюти, створило інші моделі винагороди для майнінгу: Proof-of-Capacity (підтвердження об'ємом) і Proof-of-Stake (підтвердження володінням) з його підмножинами. PoC не завоював достатньої популярності, хоча в його основі лежить більш гуманний процес, ніж обчислення: для Майнінг криптографічних токенов, що добуваються за принципом PoC, учаснику мережі необхідно надавати не обчислювальні потужності, а простір для зберігання даних на жорсткому або будь-якому іншому диску. А ось PoS або «підтвердження володінням» розглядається як одна з життєздатних альтернатив застосовуваного в BTC PoW.

Вперше про PoS заговорили на профільному форумі BitcoinTalk ще в 2011 році. PoS, або «підтвердження володінням», розглядається як інструмент, який може використовуватися як разом з PoW, так і повністю незалежно від дорогого процесу класичного майнінгу. Парадигма PoS побудована навколо обсягу криптовалюта на конкретному гаманці або пулі гаманців. Саме це, а не обсяг проведених обчислень, як у випадку з PoW, є визначальним фактором для здійснення запису в наступний блок ланцюга. У PoS є очевидні переваги перед PoW – не потрібні титанічні зусилля і витрати на майнінг криптографічних токенів. PoS – це більш цивілізована версія видобутку, яка має більше спільного з сучасними фінансовими інструментами та інститутами, які займаються емісією грошових знаків, ніж PoW, який швидше схожий на видобуток корисних копалин. Звичайно, в майнінгу криптовалют є елемент випадковості, і це правило працює і для PoS – розмір збережених на гаманці монет лише збільшує шанси на отримання нових, але не гарантує, що винагорода за наступний блок отримає саме найбільш «багатий» учасник.

З іншого боку сама парадигма PoS загрожує основі блокчейна, закладеної в маніфесті 2008 року, – децентралізації. Ця модель теоретично може підштовхнути когось до концентрування 51% частки монет в своїх руках для встановлення «диктату» в рамках блокчейну – подібна частка дозволить одноосібно керувати проектом і приймати рішення в обхід інших учасників, наприклад проводити атаки. Але це лише теорія, так як без децентралізованого спільноти криптовалюта не має сенсу і втрачає свою стабільність. Таким чином, атакуючий в першу чергу зашкодить сам собі, так як буде власником найбільшої кількості монет.

PoS очікувано піддався критиці, в тому числі і об'єктивною. Якщо в разі PoW криптографічні токени забезпечуються мінімум собівартістю на свою здобич (вартість обладнання та електроенергії), то маркери, здобуті за принципом PoS, позбавлені і цього і забезпечуються попитом на самих себе. Також в PoS є проблеми з безпекою всьому ланцюгу блокчейну. Зловмисник для отримання солідної частки може почати вибудовувати нову ланцюжок блоків, і після досяг-

нення рівня вище, ніж в основної гілки, вся мережа переключиться на «альтернативну». Причому механіка PoS дозволяє Майні відразу обидві гілки блокчейну – «основну» і «альтернативну», збільшуючи потенційний прибуток, адже сам процес Майнінг нічого не варто. У PoW можна Майні тільки одну гілку, що є стримуючим фактором.

При всіх недоліках PoS побачив світло. Першим проектом з використанням цієї механіки Майнінг став PPCoin (зараз називається PeerCoin). Надалі у цього підходу з'явилися свої спадкоємці, наприклад DPoS (делеговане підтвердження володіння) і LPoS (лізингове підтвердження володіння). У першому випадку майнери, що вносять записи в блокчейн, вибираються випадковим чином і постійно змінюються, у другому – використовується система лізингу, коли учасник системи надає баланс свого гаманця іншому учаснику в обмін на невеликі дивіденди.

Найвідомішим проектом, що використовують механіку LPoS, є блокчейн-платформа Waves. Кількість Майнер в мережі Waves строго обмежена, але інші учасники можуть за моделлю пулів в PoW-проектах надавати їм свої монети для збільшення шансу розшифровки блоку. В обмін учасники пулу отримують близько 1% в місяць від обсягу наданих ними коштів. Подібний механізм дозволяє більш рівномірно розподіляти здобуті монети в порівнянні зі звичайними PoS-мережами, де левова частка прибутку дістається учасникам з найбільшим балансом.

Майнінг Ethereum Classic

Що потрібно, щоб почати майнінг

1. Відеокарта (GPU) як мінімум одна, так само більш менш сучасна.
2. Власне комп'ютер (системний блок) або ферма з встановленою ОС (Windows x64). Саме 64 розрядної версії
3. Визначитися з валютою яку будемо добувати. Залежить від відео карти, в нашому прикладі і опис буде Майнінг etc. На Nvidia краще добувати в даний момент ZCASH.

4. Так як Майнінг у нас онлайн, то потрібен інтернет. Швидкість швидка не потрібна, але бажаний хороший пінг.

5. Вибрати пул (POOL) де ми будемо добувати ethereum. Далі вибираємо програму майнер і налаштовуємо.

6. Вибрати біржу або гаманець куди будуть капати і накопичується наші намайнені монети ефіріум, і так само сервіси де можна перевести наші зароблені монети на гривню і вивести на карту.

Пули для майнінгу Ethereum Classic:

http://epool.io (support thread)

etc.ethermine.org

http://etc.gpuminer.ru (support thread)

https://minergate.com (support thread)

https://etc.suprnova.cc (support thread)

http://ethteam.com/

https://ethc.coin-miners.info

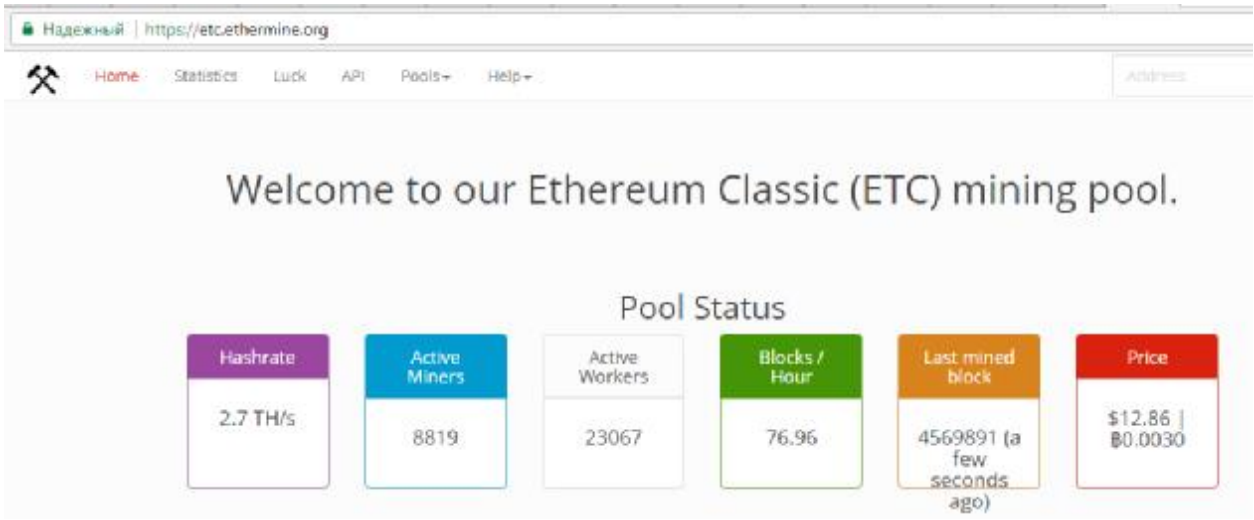
http://etc.digger.ws/

http://etc.clona.ru (solo pool, support thread)

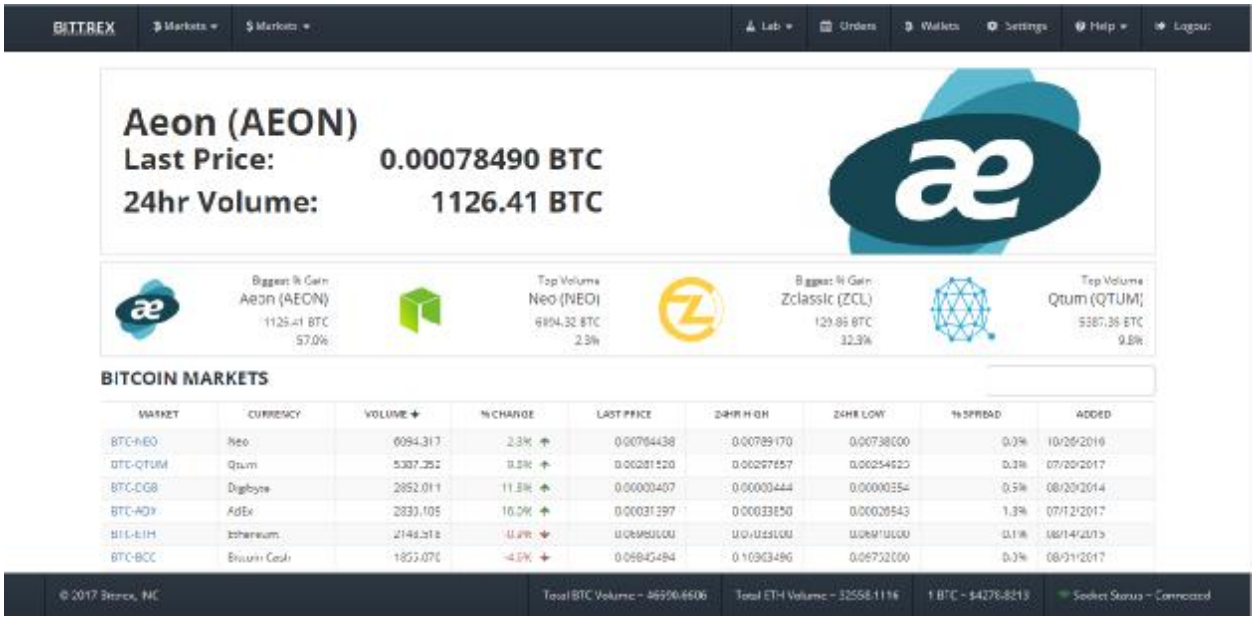
http://etc-poolcrypto.org/ (supp rus, supp eng)

https://etcmine.pro/

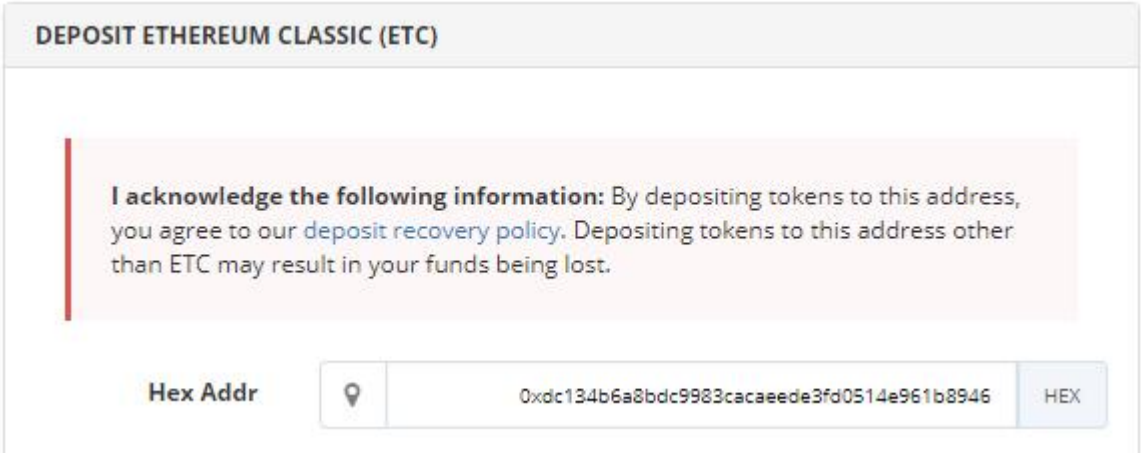
Якщо обрати пул etc.ethermine.org, то побачимо наступний інтерфейс



Серед безлічі бірж можна обрати, наприклад, bittrex.com. Звичайно потрібно зареєструватися,



та створити гаманець ETC.



Серед програм для майнінгу наприклад, можна взяти Claymore's Dual Ethereum AMD+NVIDIA GPU Miner v10.0 (Windows/Linux)

Для того щоб процес майнінгу зараховував кошти на ваш гаманець потрібно налаштувати бат-файл програми, що здійснює майнінг.

Наприклад:

TIMEOUT 90

setx GPU_FORCE_64BIT_PTR 0

setx GPU_MAX_HEAP_SIZE 100

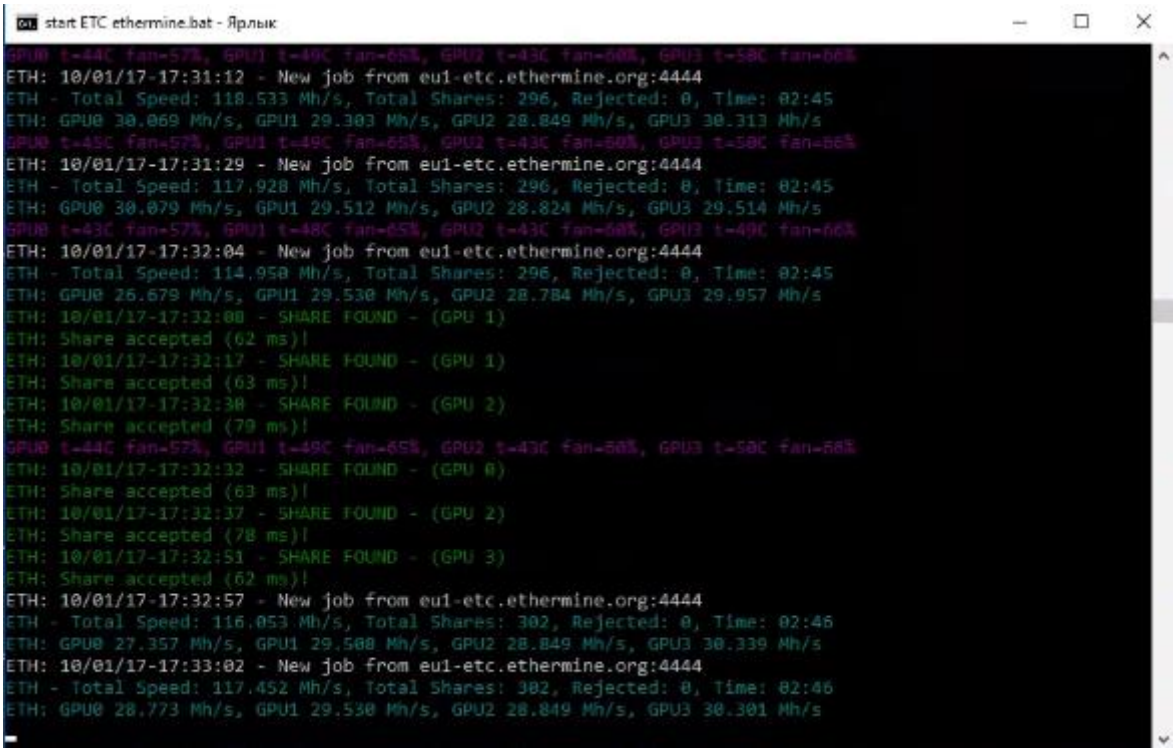
setx GPU_USE_SYNC_OBJECTS 1

setx GPU_MAX_ALLOC_PERCENT 100

setx GPU_SINGLE_ALLOC_PERCENT 100

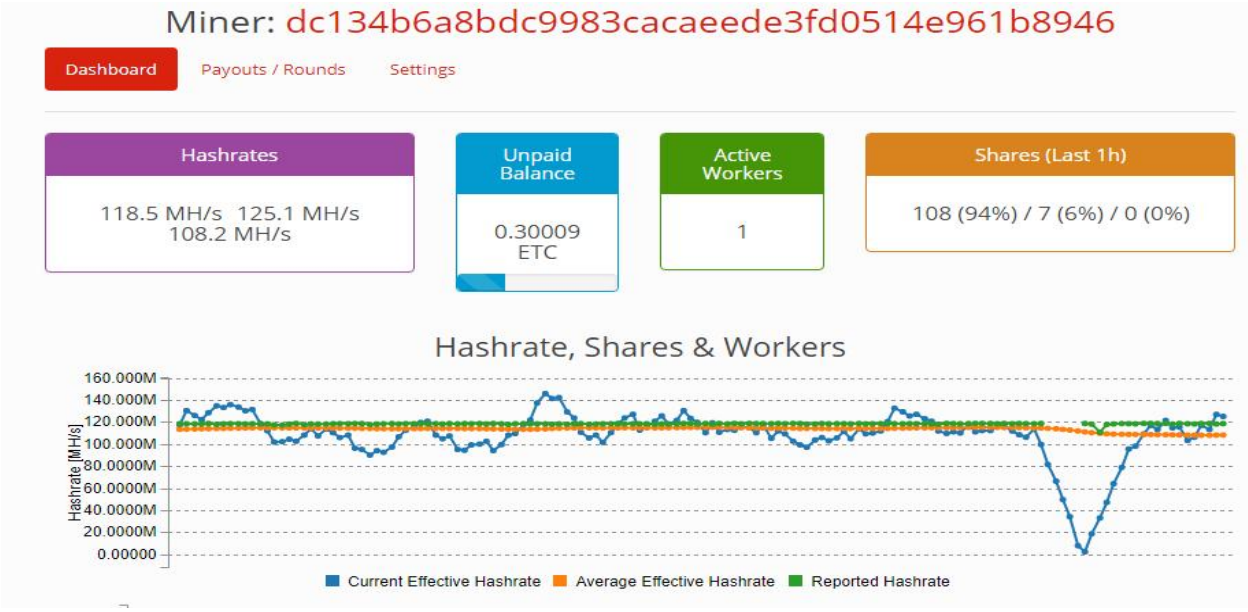
EthDcrMiner64.exe -epool eu1-etc.ethermine.org:4444 -ewal 0xdc134b6a8bdc9983cacaeede3fd0514e961b8946.ETCrig -epsw x -dcri 9 -allpools 1 -allcoins etc -mode 1

Далі, потрібно запустити бат-файл і почнеться процес майнінгу,



який буде відображатися тільки у ДОС-вкладці вашого комп’ютера.

Через декілька хвилин на пулі ви зможете побачити статистику



1.5. Криптовалюта Litecoin

Litecoin (від англ. *Lite* – «легкий», англ. *Coin* – «монета») – форк Bitcoin, пірінгова електронна платіжна система, що використовує однойменну криптовалюту.

Створення і надсилання Litecoin ґрунтується на протоколі без централізованого адміністрування, заснованому на технології Bitcoin. Програма має відкритий вихідний код.

Litecoin замислювався розробниками, як еволюція Bitcoin і має ряд відмінностей від нього. Станом на 18 жовтня 2015 1 LTC коштував приблизно 3 USD на біржі BTC-E і був другою за величиною капіталізації криптовалютою у світі.

Litecoin можуть використовуватися для обміну на bitcoin або звичайні гроші в обмінниках, а також для електронної оплати товарів і послуг у продавців, готових їх приймати.

Для забезпечення функціонування та захисту системи використовуються криптографічні методи.

1.5.1. Історія виникнення Litecoin

Проект Litecoin був задуманий і створений Чарльзом Лі як альтернатива Bitcoin, програмний код якого був взятий за основу. Проект був запущений 13 жовтня 2011 року. Платіжна система Litecoin підтримується одночасною роботою великої кількості копій програми-клієнта, відкритий вихідний код якої був опублікований на сервісі GitHub 7 жовтня 2011 року. Станом на лютий 2014 поточною версією клієнта є 0.8.6.2. Нова версія включає в себе поліпшення безпеки та продуктивності програми-клієнта і мережі в цілому. Також у версії 0.8.6.1 операційна комісія була зменшена в 20 разів. Також можуть виходити інші клієнти.

У квітні 2013 року в новинах Litecoin позначалася як альтернатива / резерв / заміна Bitcoin. У листопада 2013 року капіталізація Litecoin в доларах США значно зросла, збільшившись на 100 % протягом 24 годин.

1.5.2. Порівняння Bitcoin і Litecoin

У мережах Bitcoin і Litecoin транзакції здійснюються за адресами. Звичайні адреси Bitcoin складаються з 27-34 символів і починаються з 1 або 3. Адреси Litecoin складаються з 33 символів і завжди починаються з букви L.

Для підтримки працездатності мережі, а також для забезпечення необхідного рівня захищеності (зокрема, для запобігання можливості атаки «Double Spending») використовується механізм циклічного хешування. У випадку, якщо числове значення хешу заголовка блоку дорівнює або нижче згенерованого системою параметра, умова вважається виконаною і створюється новий блок. В іншому випадку, змінюється блок випадкової інформації в заголовку і хеш перераховується. Коли варіант знайдений, вузол розсилає отриманий блок іншим підключеним вузлам. Інші вузли перевіряють блок. Якщо помилок немає, то блок вважається доданим в ланцюжок і наступний блок повинен включити в себе його хеш. Результат хешування практично непередбачуваний. Таким чином, імовірність створити новий блок для кожного окремо взятого користувача дорівнює відношенню кількості хешів в секунду (виражається звичайно в KH / s), обчислюваного на його обладнанні, до кількості обчислюваних хешів в секунду у всій мережі. Той, хто створив новий блок, отримує винагороду з деякої кількості нових монет. Процес пошуку відповідного хешу для формування нового блоку називається «Mining» або «видобування». За знаходження нового блоку в мережі встановлена нагорода, спочатку рівна 50 LTC і зменшена вдвічі за кожні 840 000 блоків. Для доказу виконання роботи Bitcoin використовує хеш-функцію SHA256, що робить Майнінг Bitcoin надзвичайно завданням з розпаралеленими обчисленнями.

Litecoin використовує scrypt як доказ виконання роботи. Хеш-функція scrypt використовує SHA256 як підпрограму, покладаючись на велику кількість арифметичних обчислень, але також вимагаючи наявності швидкого доступу до великих обсягів пам'яті. Це робить запуск декількох екземплярів scrypt на АЛУ сучасної відеокарти кілька більш складним завданням. Це також означає, що вартість виробництва спеціалізованого обладнання для видобутку лайткоїнів на інтегральних схемах спеціального призначення (ASIC) або на ПКВМ буде значно вище, ніж вартість виробництва подібних пристроїв для SHA256. Оскільки сучасні GPU володіють великими обсягами пам'яті, вони більшою мірою придатні для Майнінг Litecoin, проте їх перевага в порівнянні з CPU є менш значною, ніж чим у випадку з Bitcoin (перевага в 10 разів проти 20 для Bitcoin). Параметри функції scrypt використовувані Litecoin (N = 1024, p = 1, r = 1) дозволяють користувачам, що не майнять Litecoin запускати клієнт в багатозадачному режимі, не зачіпаючи продуктивність системи. Ці параметри, за твердженням Коліна Персиваля, творця scrypt, також зменшують ефективність використання ASIC приблизно в 10 разів. Оскільки імовірність створення нового блоку та отримання нагороди залежить від обчислювальної потужності обладнання користувача, то із зростанням кількості майнерів та їх сумарної продуктивності для звичайного користувача шанс вельми невисокий. Щоб підвищити ймовірність отримання нагороди, майнери об'єднують свої обчислювальні потужності в пули. У разі успіху нагорода розподіляється між учасниками.

1.5.3. Швидкість проведення транзакцій

Складність обчислення Litecoin підбирається таким чином, щоб, в середньому, один блок генерувався 2,5 хвилини, що в чотири рази швидше, ніж Bitcoin, що дозволяє швидше отримувати підтвердження транзакцій. Транзакція, як правило, вважається завершеною після 6 блоків, або 15 хвилин.

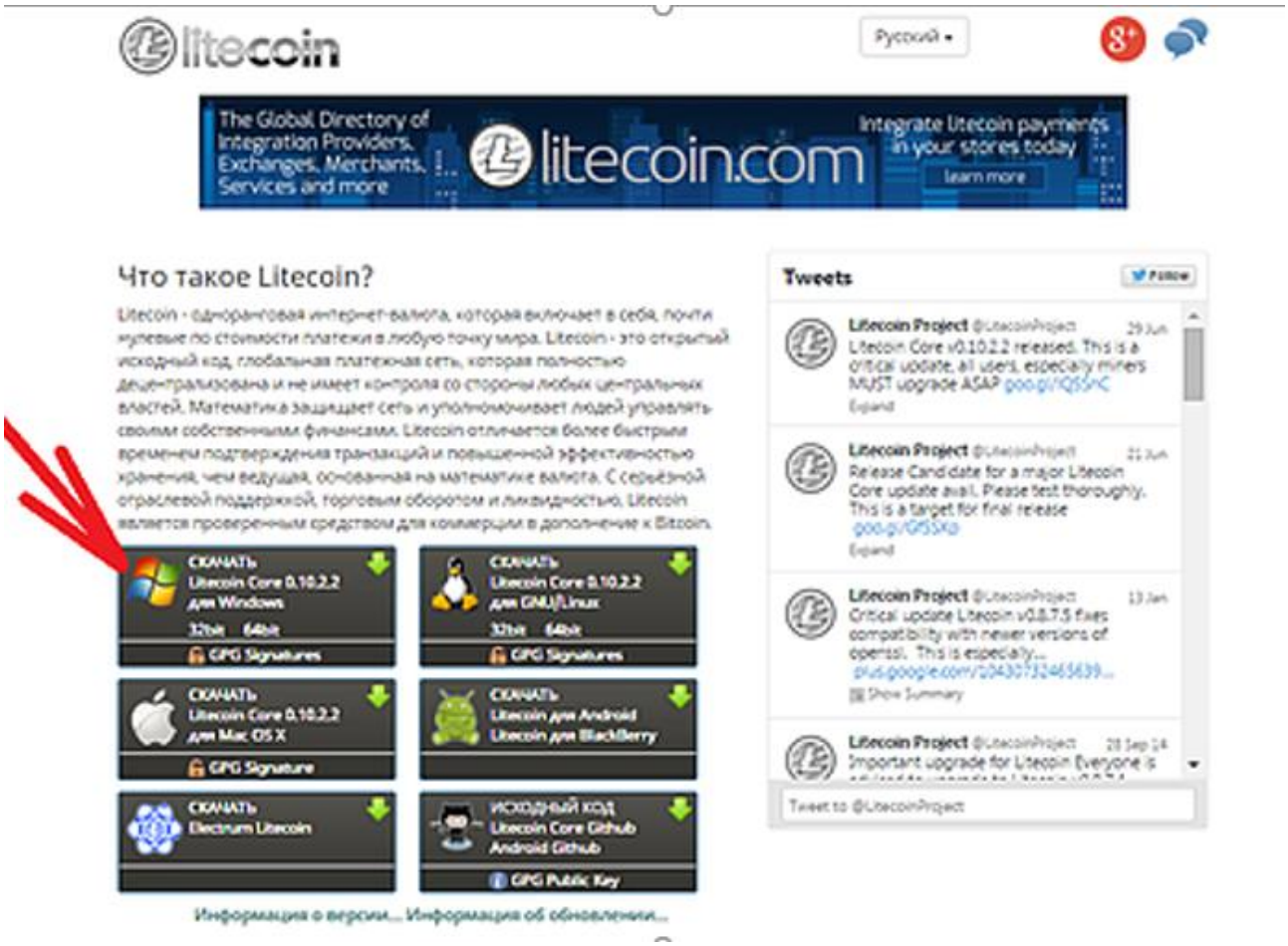
Емісія Litecoin алгоритмічно обмежена. Максимальна кількість litecoin, яке увійде в обіг, перевищує максимальне число bitcoin в 4 рази (84 мільйони проти 21). Первісна нагорода за кожен блок дорівнює 50 litecoin. Швидкість генерації litecoin зменшується вдвічі за кожні 840 000 блоків, що в 4 рази більше блоків, ніж з Bitcoin. Оскільки блоки litecoin формуються в 4 рази швидше, ніж блоки bitcoin, це означає, у них темпи емісії та винагороди будуть подібні. Наприклад, до 2020 року близько 3/4 всіх litecoin будуть згенеровані.

1.5.4. Атака 'Time Warp'

За вимогу до хешу блоків в мережі Litecoin відповідає параметр, званий «складність». Через те, що обчислювальні потужності мережі непостійні, цей параметр перераховується клієнтами мережі таким чином, щоб один блок генерувався приблизно 2,5 хвилини. Атака 'Time Warp' заснована на помилці, притаманній Bitcoin і всім його аналогам (у тому числі Litecoin). Помилка полягає в тому, що під час перерахунку складності неправильно обробляється останній блок. Зловмисник може неодноразово спробувати дозволити останній блок перед перерахунком, приписавши йому тимчасову позначку, що на 2:00 перевищує поточний час, тим самим зменшуючи складність приблизно на 0,5 %. Через таку помилку ці додаткові 2:00 не враховуються при наступному перерахунку. Як тільки складність досить сильно впаде, можна приступати до видобування «швидких» litecoin. Таким чином, зловмисник, що володіє 51 % обчислювальної потужності мережі, може знизити складність до одиниці та почати видобувати нові монети, утворивши власну гілку (форк) блокчейну. Для мережі Bitcoin дана атака практично нездійсненна, тому що ймовірність підтвердити останній блок перед перерахунком кожні два тижні при поточній потужності мережі та складності мала і нею можна знехтувати.

1.5.5. Гаманець криптовалюти Litecoin

Цей електронний гаманець можна встановити собі на комп'ютер щоб гроші були захищені в разі проблем з сайтом.



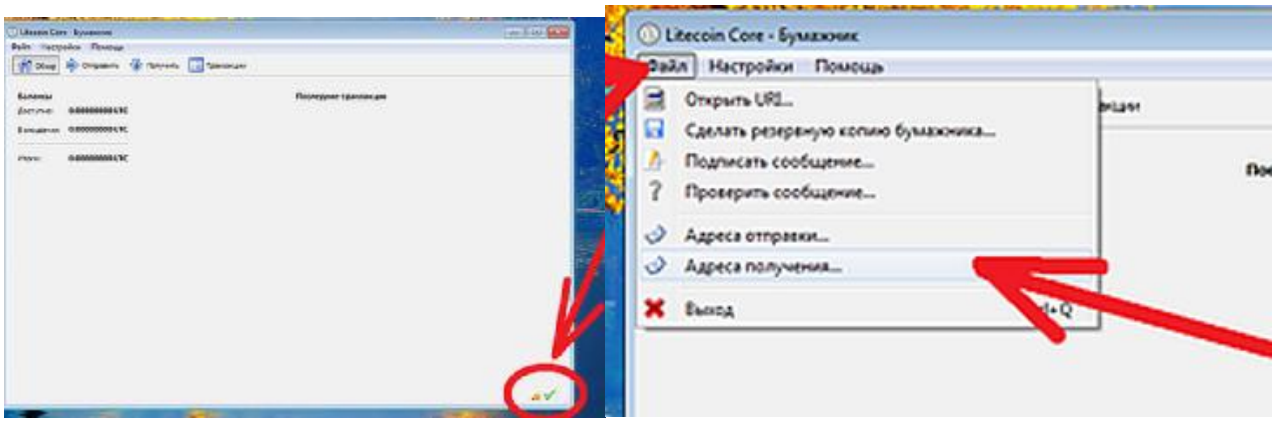
Після завантаження файлу на ваш комп'ютер, натисніть на нього, щоб встановити гаманець Litecoin. Ви побачите наступну послідовність вікон.

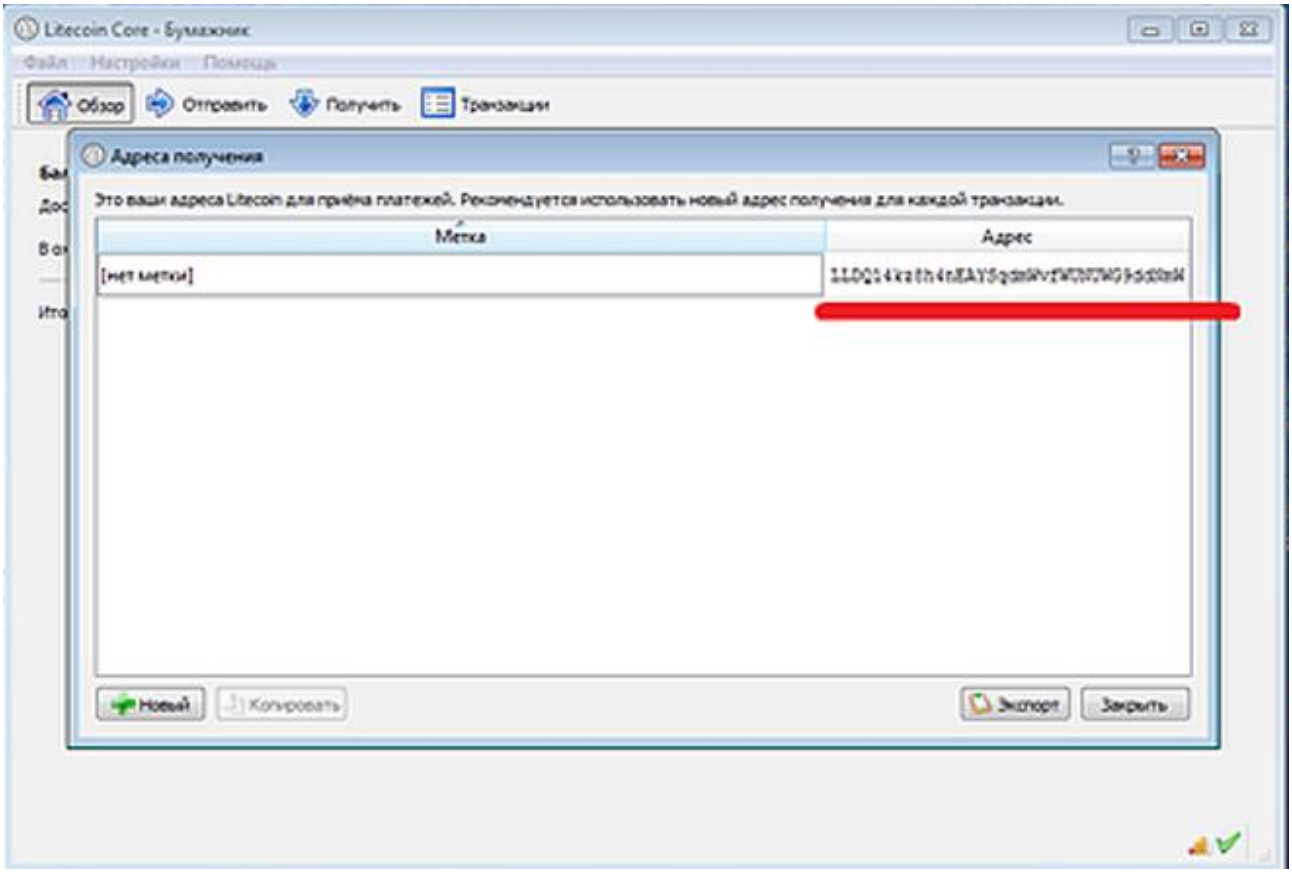
Дочекайтеся, поки завантажитися повністю Синхронізація. Увага! Синхронізація може зайняти до 10 годин. Наступного разу при вході в гаманець, цей процес буде проходити дуже швидко.

Ознакою закінчення синхронізації є зелена галочка внизу праворуч вікна гаманця.

Відкрийте "Файл" - "Адреса одержувача"

В даному вікні, буде написаний ваш номер гаманця Litecoin. Його потрібно вказувати, щоб отримувати лайткоін.





• Можливості гаманця : Оплачувати, отримувати та переказувати.

• Комісія: від 0,00170000 Ltc (від 30 грн).

• Ліміти: немає.

1.6. Криптовалюта Namecoin

Namecoin (англ. Name – «ім’я», Coin – «монета») – заснована на технології Bitcoin система зберігання довільних комбінацій виду «ім’я-значення», найбільш відомим застосуванням якої є система альтернативних кореневих DNS-серверів. Namecoin не є однією організацією. Кожен вузол мережі Namecoin має повну копію розподіленої бази даних.

Принцип тимчасової мережі і відсутність адміністративного центру унеможливлює вилучення імені. Для обчислення блоків використовується стандартне програмне забезпечення для Майнінгу у Bitcoin, перенаправлення на сервер і порт, де працює Namecoind. Можливий одночасне видобування Namecoin і Bitcoin без зниження ефективності видобування Bitcoin за рахунок використання технології «Merged Mining». Також була можливість придбати домен за криптовалюту Bitcoin через посередника. Термін реєстрації імені вважається закінченим після обчислення 36000 нових блоків.

Зараз у Namecoin реєструються домени тільки в зоні .bit, для яких використовується простір імен «d /» (наприклад, запис домена «bitcointalk.bit» використовує ім'я «d / bitcointalk»). Потужність розподіленої обчислювальної мережі гарантує, що не з'явиться двох однакових імен і що ваше відповідність «ім'я-значення» («домен-адреса» в окремому випадку) не зможе бути присвоєно і змінено жодним сторонньою особою.

Щоб отримати доступ до доменів, розташованим в доменній зоні .bit можна використовувати проксі-сервер або DNS-сервери, зазначені в вікі проекту. До недоліку Namecoin можна віднести те, що імена неможливо юридично захистити від кіберсквотерів, тобто крадіїв імен.

1.7. Криптовалюта Mastercoin

Omni (раніше Mastercoin) є цифровим протоколом валюти і зв'язку побудований на принципах блокчейн валюти біткоїн. Є одним з багатьох нових криптовалют, що дозволяють виконувати складні фінансові функції. Її можливості включають в себе розвиток децентралізованого обміну і реалізації інтелектуальних властивостей і ощадних гаманців.

Віллетт опублікував перший проект протоколу Mastercoin в січні 2012 року в вигляді білого паперу, в якому він висловив пропозицію, що існуючий протокол Bitcoin «може бути використаний в якості протокольного рівня, на вершині якої нові типи валюти з новими правилами можуть бути побудовані без зміни фундаменту».

Проект Mastercoin офіційно запущений 31 липня 2013 року, з місячним збором коштів, в якому хтось може купити Mastercoins, відправивши біткоїни на спеціальну «Вихідну адресу». Ідея полягала в тому, що в якості нова платформа поступово стане ціннішою і інвестори можуть продати свої Mastercoins реалізувати повернення іншої валюти. Некомерційна організація під назвою «Фонд Mastercoin» була створена для обробки грошових коштів, відправлених за адресою сайту мастеркоїн. Незважаючи на попередження, що Mastercoin може бути просто складна афера, близько 500 людей вклали свої гроші, відправивши в цілому близько 5000 Bitcoins вартості близько $ 500000 на той час.

Станом на січень 2014 року Віллет використовує повний робочий день по «Фонду Mastercoin» в якості «головного архітектора». Це означає, що ця криптовалюта стала теж прибутковою – її торгують на різних електронних біржах.

Станом на лютий 2014 року, Mastercoin був в світі сьомою за величиною криптовалюта за ринковою капіталізацією по coinmarketcap.com.

У квітні 2014 року MaidSafe використавши масовий продаж, зібрала понад $7000000 в Mastercoins і Bitcoins.

У липні MaidSafe COO Нік Ламберт був одним з багатьох людей, які приєднуються до Mastercoin в якості спостерігачів.

У березні 2015 Mastercoin поміняв назву і став Omni. Роль Omni в Bitcoin екосистеми оголошений як платформа для децентралізованих протоколів, таких як Factom і MaidSafe. «Поширена аналогія, яка використовується для опису відносини багатофункціонального шару до Bitcoin є те саме, що і HTTP для TCP / IP: HTTP, як і Omni Layer, є застосуванням можливостей блокчейн для більш фундаментального транспорту і інтернет шару TCP / IP, як Bitcoin».

На сайті http://www.omnicoin.org/ можна отримати вичерпну інформацію щодо цієї валюти за закладкою “Service” (Рис. 2.20).

Рис. 2.20. Перелік служб криптоалюти Омнікоін
Exchanges – валютні біржі
Pools – пули для створення нових блоків
Online Wallets – онлайн гаманці
Faucets – сайти, що надають змішані послуги.
Payment Processors – Платіжні системи
Calculators – розрахунок крос-курсів для різних типів криптовалют
Statistics – статистика
Block Explorers – системи оперування блоками в системах блокчейн.

1.8. Криптовалюта PPcoin

Peercoin, також відомий як PPCoin або КПП, є криптовалюта рівноправних вузлів ЛВС з використанням як доказ правильності пакету і доказ правильності роботи системи.

Peercoin заснована у серпні 2012 року документ, в якому перераховані автори – Скотт Надаля і Санні Кінг. Санні Кінг також створив Primecoin. Участь

Надаля зменшилася до листопада 2013 року, залишивши Кінга в якості єдиного розробника ядра Peercoin.

Peercoin був натхненний Bitcoin, і він розділяє більшу частину вихідного коду і технічної реалізації Bitcoin. Вихідний код Peercoin поширюється під ліцензією MIT / X11.

Peercoin є четвертою за величиною криптовалютою за ринковою капіталізацією. Peercoin має ринкову капіталізацію в $ 30 млн доларів США за станом на 20 липня 2014 г. На відміну від Bitcoin, Namecoin і Litecoin, Peercoin не має жорстких обмежень на кількість можливих монет, але в кінцевому підсумку досягне річного рівня інфляції в розмірі 1%. Існує дефляційний аспект Peercoin як плата за операцію у розмірі 0,01 PPC / кбайт.

Мережа є одноранговою і обробляє транзакції Peercoin через хеш-код SHA-256, що є доказом правильності роботи схеми Peercoin.

Peercoins в даний час торгуються на інтернет-біржах криптовалют, Bitcoins і інших. Реверсивні угоди (наприклад, ті, що здійснюються за допомогою кредитних карт) як правило, не використовуються для покупки Peercoins, оскільки операції Peercoin є незворотними, що не дуже зручно.

Платежі в мережі Peercoin зроблені за адресами, які засновані на цифрових підписах. Вони являють собою рядки з 34 букв і цифр, які завжди починаються з літери P. Можна створювати скільки завгодно адрес, в міру необхідності, не витрачаючи ніяких Peercoins. Часто використовують одну адресу тільки з однією метою, щоб легко побачити, хто насправді послав Peercoins.

Операції відображаються в blockchain Peercoin (цього дотримуються більшість клієнтів), новий блок додається до blockchain всього за 10 хвилин (у випадку, коли знайдено досить мале значення хеша), транзакція зазвичай вважається завершеною після 6 блоків, або 60 хвилин, хоча для невеликих угод, достатньо буде 6 блоків.

1.8.1. Створення нових монет PPcoin

Нові монети можуть бути створені двома різними способами; «видобуток» і «карбування». «Видобуток» використовує алгоритм SHA-256 для захисту мережі. Карбування монет користувачів пропорційне монетам, які вони вже мають (орієнтовані на 1% в рік). Є довгострокові плани щодо поступового скороченню обсягу «видобутку», користувачі більше покладаються на «карбування». Це повинно створити справедливий розподіл і може привести до збільшення винагороди від «карбування».

Головна відмінна риса Peercoin є те, що вона використовує гібридну коректуру через акції «видобувачів» пулу і/або докази правильності роботи системи. Система коректури була розроблена для усунення вразливостей, які можуть виникнути в системі, щоб служити доказом правильності роботи. З Bitcoin, наприклад, існує ризик атак в результаті монополії на «видобуток» нових монет. Це відбувається тому, що вигоди від видобутку запрограмовано знижуватися в геометричній прогресії, що може зменшити стимул до «видобування». Якщо «видобуток» почне спадати, вірогідність монопольності збільшується, що робить мережу вразливою для 51% атаки на біткоїни інших власників (51% атаки, коли один суб'єкт має більше половини видобутку частки, яка б дозволила двічі провести операцію з монетами, які тримає цей власник). За допомогою системи коректури, нові монети генеруються на основі наявних у окремих осіб. Іншими словами, коли хтось тримає 1% від валюти, той буде генерувати 1% всіх монет блоків. Це має ефект створення монополії і відокремлює ризик монополії від коректури за рахунок діяльності «видобувачів».

Вся мережа використовує алгоритм SHA-256. Для кожного 16-кратного збільшення в мережі, докази правильності роботи зменшується в два рази.

Для коректури акцій система Peercoin була розроблена для вирішення високого споживання енергії Bitcoin. Наприклад, станом на квітень 2013 року покоління Bitcoins використовує приблизно $ 150. 000 доларів США в день до ви-

трат на споживання електроенергії. Метод коректури через акції генеруючих монет вимагає дуже мінімальне споживання енергії; вона вимагає тільки енергії для запуску клієнтського програмного забезпечення на комп'ютері, на відміну від запуску ресурсоємних функцій криптографічного хешування. Під час ранніх стадій зростання, більшість Peercoins буде генеруватися доказом правильності роботи як і у Bitcoin, проте з плином часу доказ правильності роботи буде припинено оскільки блок винагороди зменшується. У доказі правильності пакету стає основним джерелом генерації монет, споживання енергії (по відношенню до ринкової капіталізації) зменшується з плином часу. Станом на січень 2014 року, приблизно 90% нових монет генерується ще з доказом правильності роботи і споживання енергії Peercoin використовує приблизно 30% споживання енергії Bitcoin (масштабування ринкової капіталізації – у вартісному вираженні, забезпечених на GH / с).

1.8.2. Врахування інфляції

Peercoin розроблена таким чином, що вона буде теоретично відчувати постійну 1% інфляції на рік, що дає необмежену кількість монет. Це комбінований результат процесу «карбування» нових акцій, а також масштабування труднощів «видобування». Хоча Peercoin технічно має максимум у 2 мільярди монет, це тільки для перевірки несуперечності, а максимум навряд чи буде досягнутий в осяжному майбутньому. Якщо максимум буде досягнутий, він легко може бути підвищений, отже, для всіх практичних цілей Peercoin можна вважати засобом, що має інфляцію на рівні 1% в рік, з безмежною грошовою масою. Цей тип криптовалют був частково розроблений для зростаючого задоволення населення.

Peercoin розроблений таким чином, що змінні і додаткові операційні витрати видаляються на користь протоколу визначається плата за угоду (в даний час 0.01 КПП / кБ). Вартість угоди фіксується на рівні протоколу і не йде до «ви-

добувачів», замість цього ці суми списуються. Це придумано для компенсації інфляцію при зниженні пропозиції грошей і служить для самостійного регулювання обсягу транзакцій.

1.9. Криптовалюта Dogecoin

Dogecoin - клон Bitcoin, який зміг досягти успіху завдяки правильному підходу до маркетингу. За останні роки з'явилися сотні криптовалют, але лише деякі з них відразу ж змогли стати відомими. Dogecoin виступав спонсором на багатьох відомих заходах, включаючи гонки Nascar і Зимові Олімпійські ігри. Ця монета не знайшла широкого застосування, але стала фактично валютою для дачі чайових в інтернеті. На кінець 2014 року було випущено 100 мільярдів одиниць монети, а зараз випускається приблизно по 5 мільярдів на рік.

Курс: 1 DOGE = 0.00000024 BTC -4.49 %

Капіталізація: $ 114 779 880

Об’єм емісії: 100,000,000,000

Алгоритм: Scrypt

Метод захисту: PoW

Рік заснування: 2013

Dogecoin був створений програмістом Біллі Маркусом, який хотів створити криптовалюту, яка була б ближче до більшої демографічної групи, а також дистанціюватися від історії біткоїнів, зокрема пов'язаної з продажем наркотиків.

Dogecoin був заснований на існуючій криптовалюті Luckycoin, яка в свою чергу була заснована на Litecoin, яка заснована на Bitcoin. Як і в Luckycoin, розмір нагороди за кожен блок в Dogecoin встановлювалася випадковим чином. У березні 2014 року ця положення було змінено і розмір нагороди став фіксованим. Спочатку замислювалося, що розмір емісії складе 100 млрд, але пізніше було оголошено, що виробництво DOGE буде необмеженим.

У грудні 2013 року Dogecoin поставив рекорд по добовій кількості транзакцій, перевершивши в сукупності всі інші криптовалюти у 2,5 рази.

1.10. Криптовалюта Pinkcoin

PinkCoin (PINK) - це децентралізована валюта, що використовувала PoW з алгоритмом X11 в перші 7 днів свого існування, після чого перейшла на чистий PoS, річна процентна ставка на що складає 1%. Об'єм емісії обмежено 380 млн. монет. Час блоку складає 30 секунд. Премайну, (тобто попереднього видобутку валюти перед початком операцій по ній) не було (розробники, як правило, в цілях самозбагачення прагнуть досягти значного процентного від усіх монет шляхом добутку монет до публічного анонсу).

Курс: 1 PINK = 0.00000568 BTC -4.15 %

Капіталізація: $ 8 667 192

Об’эм емісії: 380,000,000

Алгоритм: X11

Новим словом в програмуванні коїнів став перехід на алгоритми X11, X13, X15. За своєю суттю - це апгрейд (удосконалення) PoW. Число після X позначає кількість послідовних функцій, які використовуються при обчисленні в блоці. Даний вид алгоритмів займає друге місце за популярністю після scrypt.

Метод захисту: PoW/PoS

PoW (Proof-of-work - доказ виконання роботи) - система захисту систем від DoS-атак або зловживання послугами. PoS (Proof-of-Stake - «доказ збереження») - принцип доказу зберігання коштів користувача за певний термін.

Рік заснування: 2014

Ціль: спроба надати інвесторам, які останнім часом постраждали від нестабільності ринків, нову валюту, забезпечену чимось максимально надійним (кольоровими діамантами).

У березні 2016 року канадська компанія BitGem, що запустила в продаж криптовалюту PinkCoin, повідомила про закриття проекту. Ця валюта була забезпечена кольоровими фантазійними діамантами на $ 5 млн. В оборот було випущено 5 млн токенів PinkCoin за ціною $ 1. Примітно, що більшу частину токенів - близько 80% - придбали росіяни і громадяни колишніх радянських республік.

Кольорові фантазійні діаманти протягом останніх 18 років росли в ціні на 10-15% в рік, і ці камені є «найбільш концентрованою формою накопичення багатства», повідомляла компанія при запуску проекту.

Однак в процесі реалізації проекту компанія зіткнулася зі значними труднощами у вирішенні юридичних питань, що відносяться до законності операцій з PinkCoin у різних юрисдикціях.

1.11. Сайти, що обслуговують оборот криптовалюти

Подібних сайтів, які описані нижче, в Інтернеті незліченна кількість. Достатньо написати в пошуковому вікні: «Обмін біткоїн» або “Exchange bitcoin” (або назву іншої криптовалюти), як знайдеться сила-силенна посилань. Те саме стосується розрахунку курсів (Calculate) зі вказанням криптовалюти, створення електронних гаманців (e-wallet), участі у пулах «видобувачів» (pool miners) і всього іншого, пов’язаного з критовалютами.

Тому в цьому пункті були описані тільки деякі з них, вибрані як такі, що зустрічаються першими у переліку знайдених адрес.

Кожен сайт має достатньо зрозумілі підказки та інструкції, в яких легко розбереться непідготовлений користувач.

1.11.1. Біржі криптовалюти

Обмінник http://kurses.com.ua/ (рис. 1.21) пропонує посилання різні платформи, на яких здійснюється обмін криптовалют на звичайні валюти, причому веде статистику ефективності діяльності цих платформ.

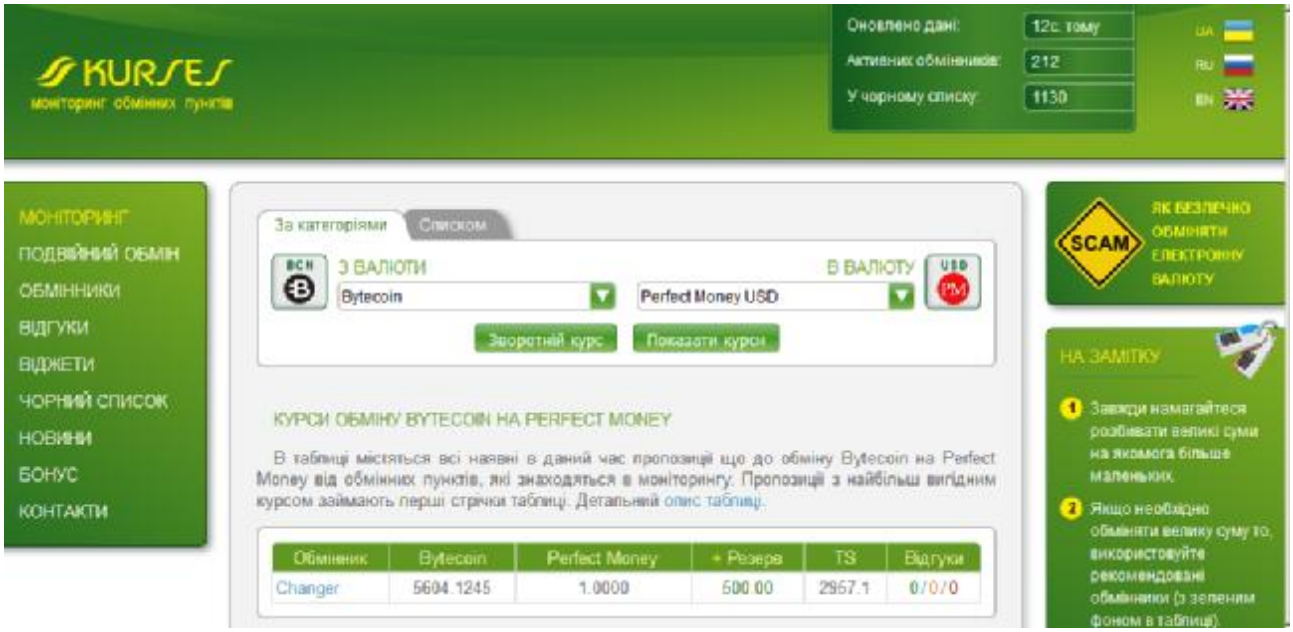


Рис. 1.21. Головна сторінка сайту http://kurses.com.ua/

Нижче знаходиться детальний опис стовпців таблиці в порядку їх розміщення (зліва направо).

Стовпець "Обмінник" містить посилання на головну сторінку обмінного пункту або ж сторінку з обраним вами напрямком обміну (в залежності від налаштувань в обміннику).

В цьому стовпці також можуть міститися різноманітні інформаційні іконки, які дозволяють обрати підходящий для вас обмінний пункт, а саме:

– повідомляє про те, що обмін здійснюється в ручному або напівавтоматичному режимі;

– під час наведення на неї мишкою з'являється спливаюча підказка з важливою інформацією.

Назва обмінного пункту виділяється *червоним* кольором, якщо він не підтримує передачу даних по захищеному SSL протоколу (https).

Стовпець №2 містить кількість одиниць вказаної валюти, яку вам потрібно буде віддати. Окрім цього, іноді вказується кількість одиниць валюти, яку необхідно буде віддати у якості комісії обмінника за здійснення операції обміну. Цю кількість одиниць валюти буде виділено червоним кольором, та перед нею буде знаходитись знак +.

Стовпець №3 містить кількість одиниць вказаної валюти, яку ви отримаєте. Окрім цього, іноді вказується кількість одиниць валюти, яку буде вирахувано з суми, яку ви маєте отримати у якості комісії обмінника за здійснення операції обміну. Цю кількість одиниць валюти буде виділено червоним кольором, та перед нею буде знаходитись знак -.

Стовпець "Резерв" містить максимальну кількість одиниць валюти, яку можна буде отримати під час обміну. За замовчуванням, обмінники, в яких кількість одиниць валюти рівна 0 або менше встановленого мінімального значення не показуються, щоб їх побачити, натисніть на + в заголовку цього стовпця.

Для зручності сприйняття, кількість одиниць валюти виділяється різноманітними кольорами. Якщо кількість одиниць валюти рівна 0 то, вона виділяється червоним кольором. Якщо для валюти вказано мінімальне значення кількості її одиниць, а поточна кількість менша або дорівнює цьому значенню то, вона виділяється помаранчевим кольором. В усіх інших випадках кількість одиниць валюти виділяється зеленим кольором.

Валюта	**Мінімальне значення**
Долар США	10
Російський рубль	500
Євро	10
Українська гривня	300

Обмінник http://bestexchangers.ru/ru/index.html (рис. 2.22) працює аналогічно попередньому, тобто сканує обрані ним платформи з метою надати користувачеві можливість вибрати найвигідніший обмінний курс криптовалюти на звичайну валюту.

Недоліком цього обмінника можна чітку орієнтацію на операції з російським рублем. Тобто, він орієнтований на громадян РСФСР.

Іншою особливістю обмінника можна вважати використання тільки двох типів криптовалют – Новакоін та Монеро. З попередніх розділів відомо, що більшість нових криптовалют створена для шахрайських оборудок, тому, хто бажає уникнути надмірного ризику, варто уникати невідомих криптовалют, якими б привабливими вам не здавалися крос-курси обміну.

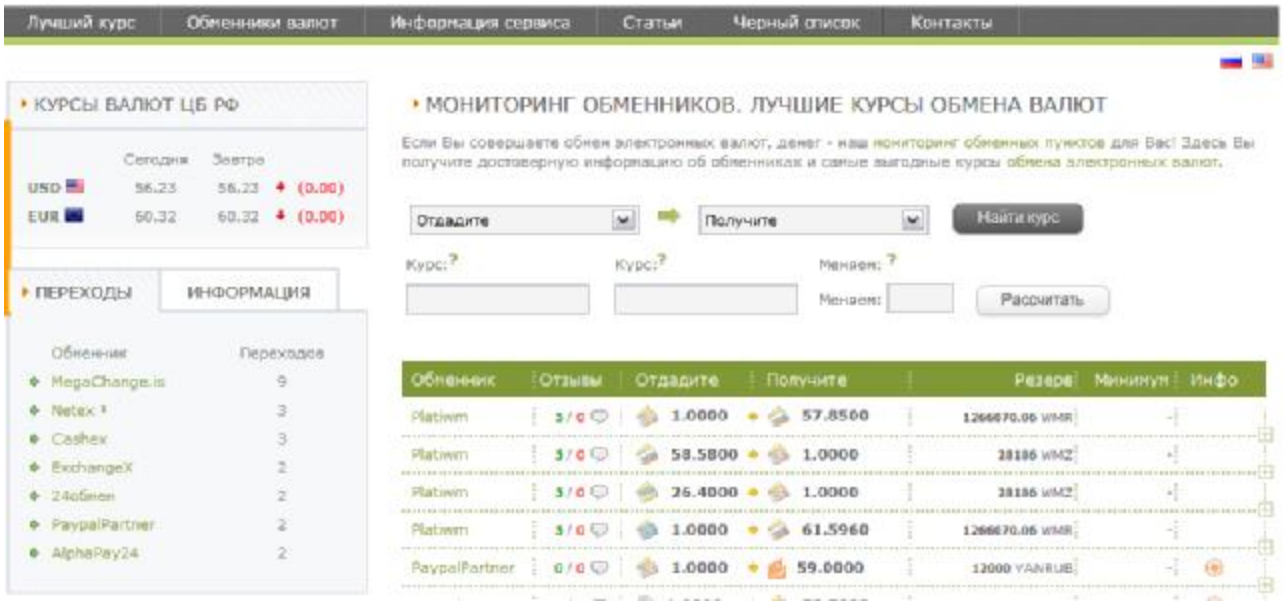


Рис. 1.22. Головна сторінка сайту http://bestexchangers.ru/ru/index.html

1.11.2. Сайти для «видобування» криптовалюти

«Майнінг» криптовалют можна розпочати з сайту https://bitmakler.com/, інтерфейс якого представлено на рис. 1. 23. Сайт русифіковано, отже тим, хто не володіє англійською легко розібратися в його можливостях, представлених у меню, що розташоване у лівій колонці сторінки.

Сайт забезпечує інформацію як про крос-курси будь-якої пари валют на всіх біржах криптовалют, так і крос-курси для всіх пар на одній біржі.

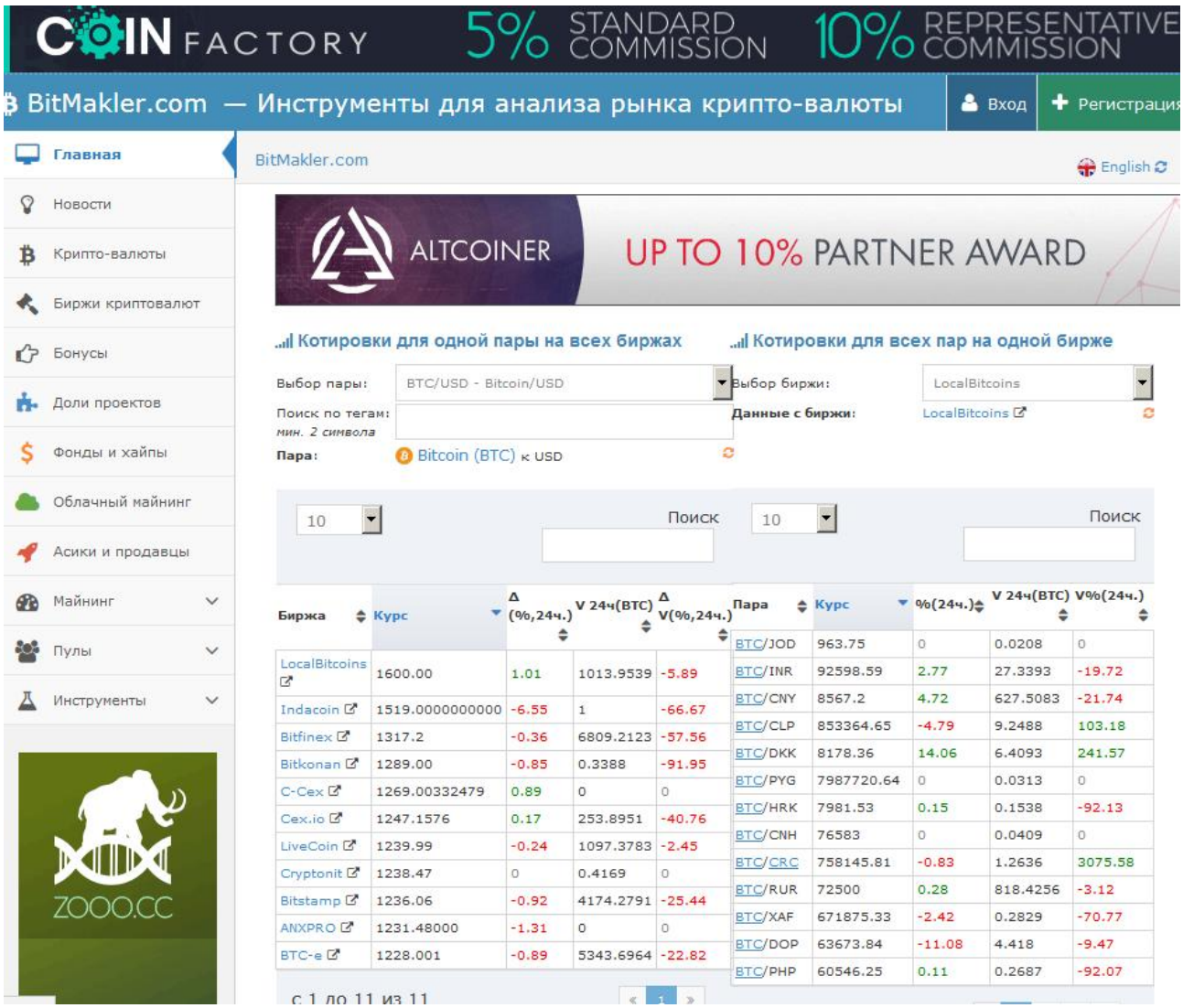


Рис. 1.23. Опис крос-курсів валют на сайті https://bitmakler.com

При виборі пункту «Пули» головного меню, відкривається список груп, які об’єднані у «видобутку» нових блоків криптовалют (рис. 1.24).

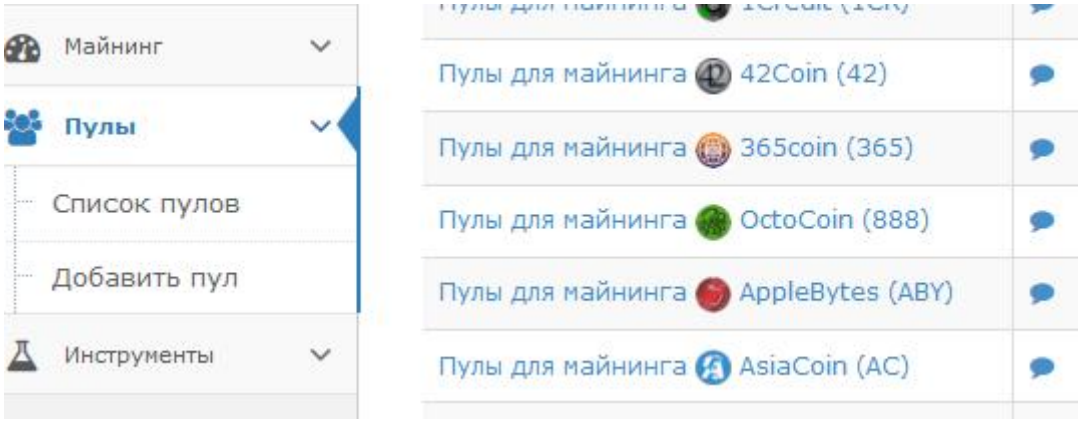


Рис. 1.24. Список доступних «пулів» криптовалют

Пункт меню «Біржі» (рис. 1.25) показує всі існуючі обмінники криптовалют та типи валют, що там торгуються, а також мови, які підтримує ця біржа. Мови позначаються прапорцем країни. Дата реєстрації біржі дозволяє зорієнтуватися щодо того, наскільки надійною є та чи інша біржа, адже ненадійні довго не існують.

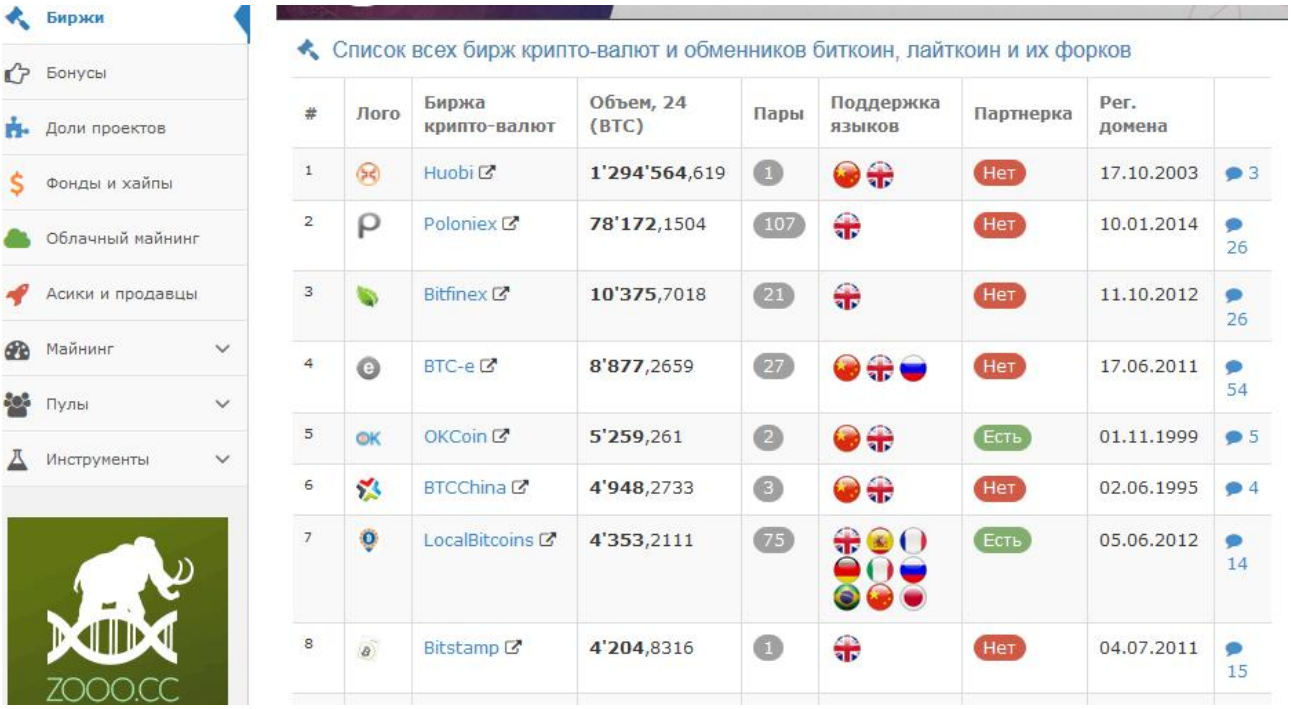


Рис. 1.25. Фрагмент списку бірж криптовалют

1.11.3. Статистика обороту та крос-курсів криптовалюти

На сайті https://coinmarketcap.com/currencies/ (рис. 1.26) подано детальну статистику для різних валют, не розділяючи їх на криптовалюти і звичайні. На рисунку показано відношення валют до долара США, але якщо натиснути синю кнопку з написом USD, то можна вибрати будь-яку іншу валюту, відносно якої будуть розраховані крос-курси.

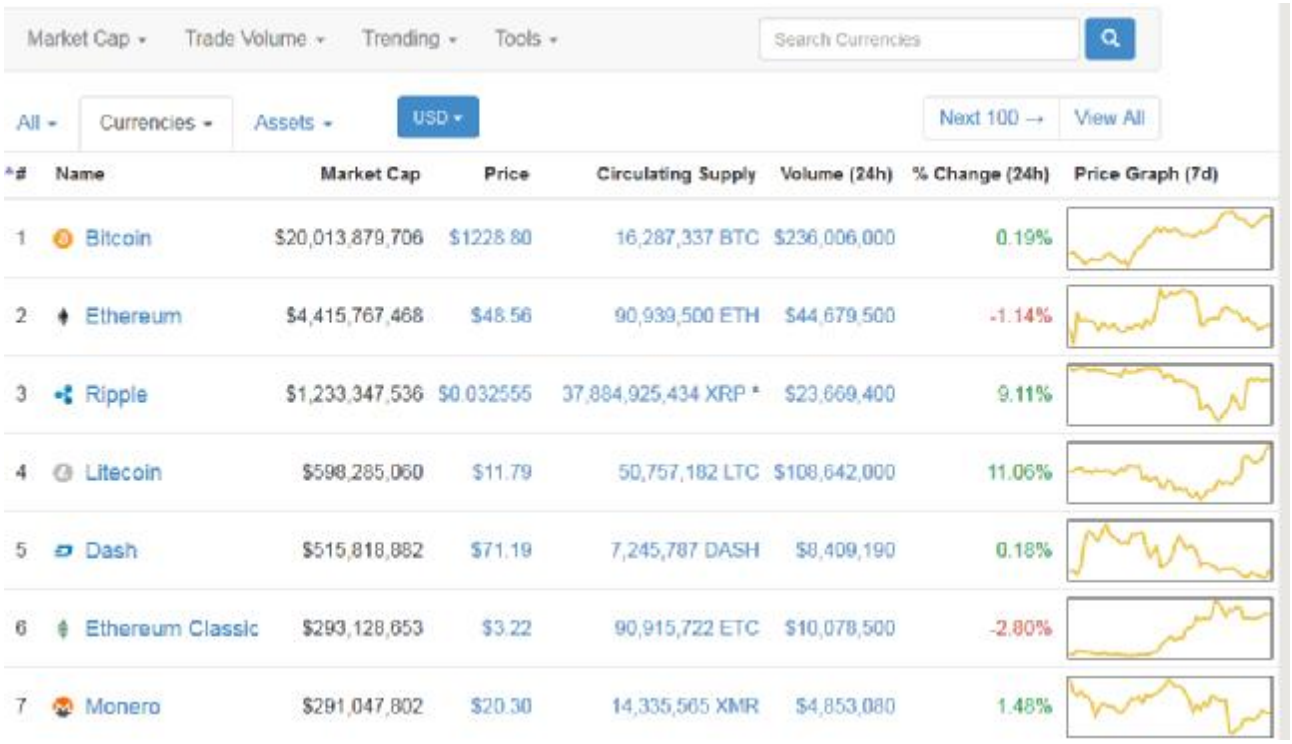


Рис. 1.26. Статистка поточних крос-курсів валют

Якщо клацнути лівою кнопкою мишки по графіку навпроти обраної валюти, то отримаємо детальний графік зміни крос-курсів (рис. 1.27).

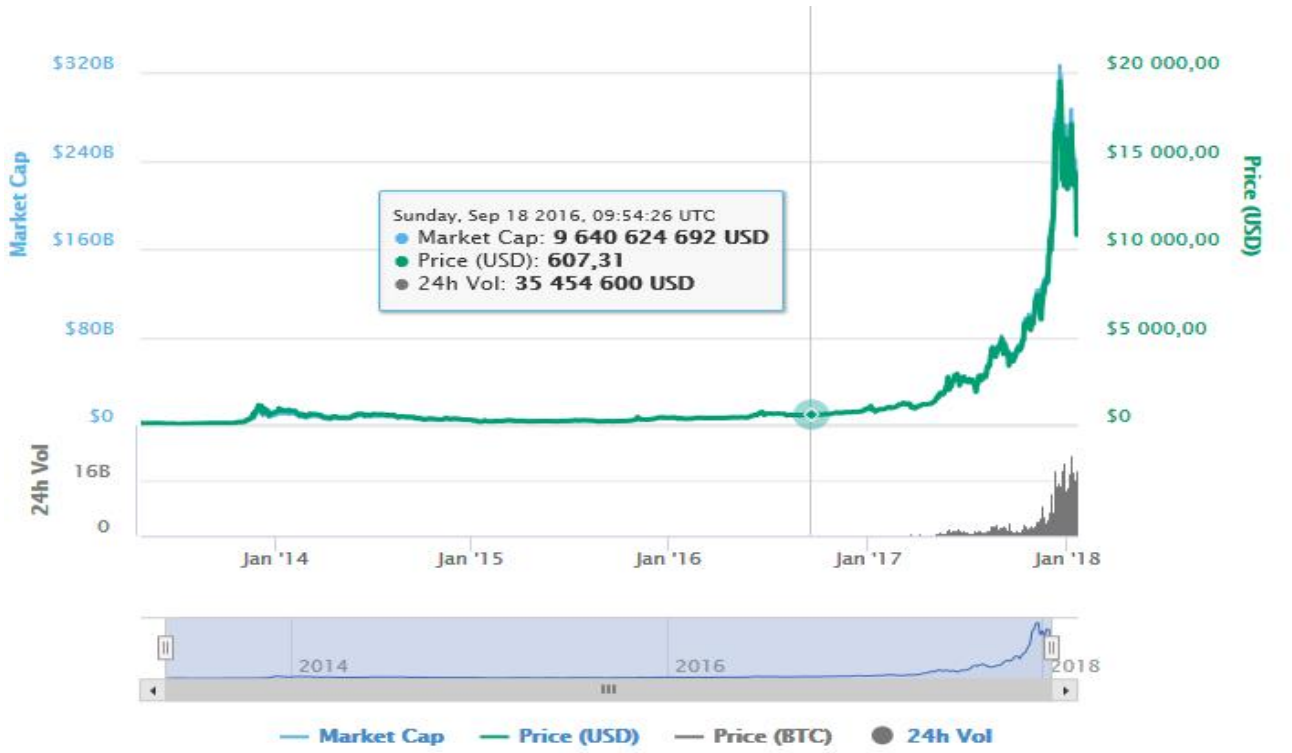
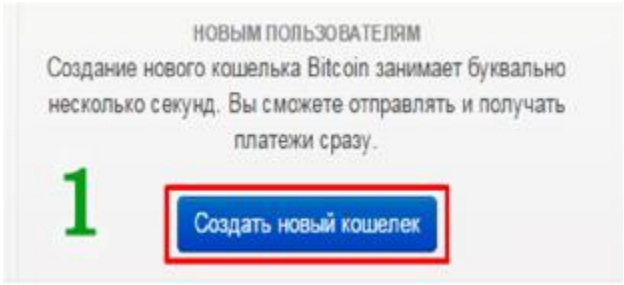


Рис. 1.27. Поточний перебіг крос-курсів біткоїну відносно долара США

1.11.4. Он-лайн гаманці криптовалюти

Найкращий опис роботи з цим сайтом подано на сайті bitcoin-crypto.blogspot.com. На рис. 1. 28 подано порядок реєстрації на сайті blockckhain.com/



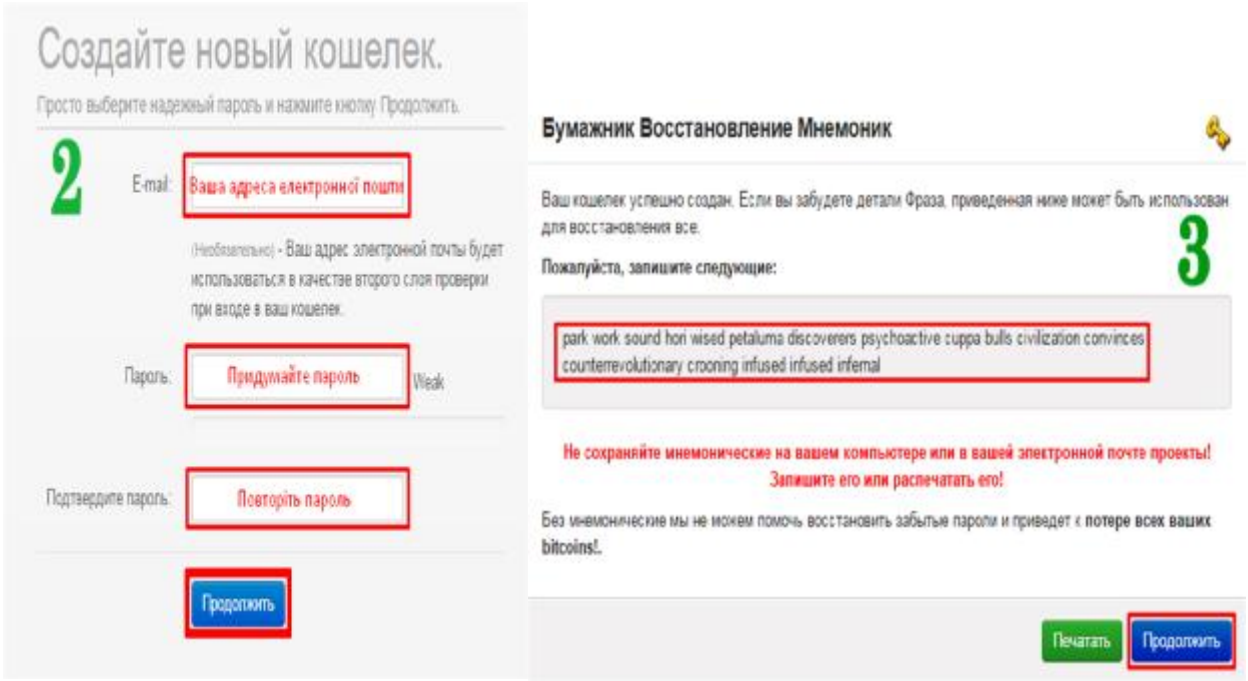


Рис. 1.28. Порядок реєстрації web-гаманця на сайті blockckhain.com/

Приклад інтерфейсу гаманця https://blockchain.info/ru/wallet/ після реєстрації, яка включає написання вашої електронної адреси та паролю, подано на рис. 1.29.

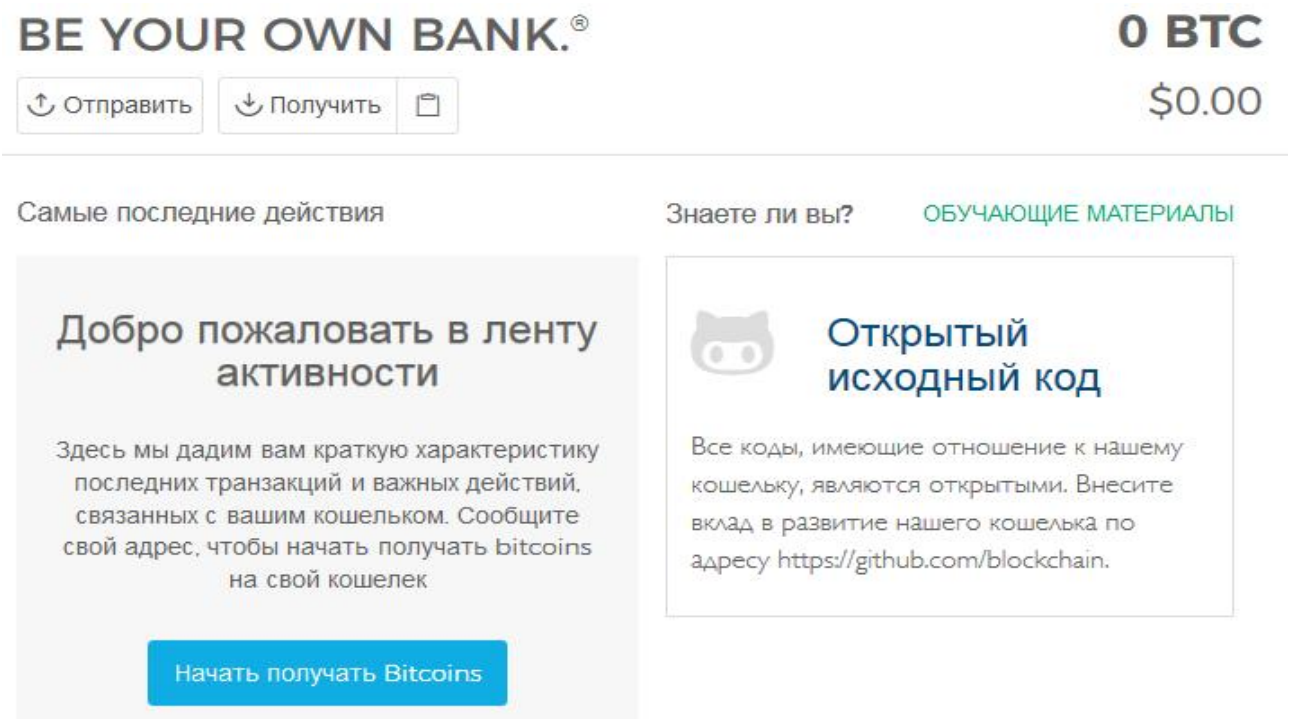


Рис. 1. 29. Стартова сторінка он-лайн гаманця

Для отримання біткоїнів відправника потрібна ваша адреса Bitcoin (рис. 1.30). Гаманець автоматично генерує нову адресу для кожної транзакції, яку бажано провести. Натисніть «Отримати» і скопіюйте адресу, щоб поділитися нею з відправником. Для відправки біткоїнів натисніть «Отправить», введіть Bitcoin адресу одержувача в полі "Кому" і вкажіть суму, яку хочете відправити.

Користування гаманцем абсолютно безкоштовно, а включена у транзакції невелика комісія призначена для «Майнерів», які допомагають посилити потік транзакції в мережі Bitcoin. Для забезпечення послідовного і надійного підтвердження ваших транзакцій гаманця, автоматично включається відповідна комісія з урахуванням розмагнічування транзакції і рівню мережевого трафіку в поточний момент. Перед відправкою транзакції можна подивитися додану рекомендовану комісію. Можна вказати власну комісію, яку роблять розділі «Розширені параметри відправки».

Транзакції з'являється майже моментально у вашій стрічці транзакцій, яка знаходиться в лівій панелі навігації гаманця. Ваша транзакція вважається виконаною, коли вона отримала 3 мережевих підтвердження. Це зазвичай займає близько 30 хвилин, але термін може змінюватися. До цього моменту ваша транзакція буде відображатися як така, що знаходиться на розгляді.

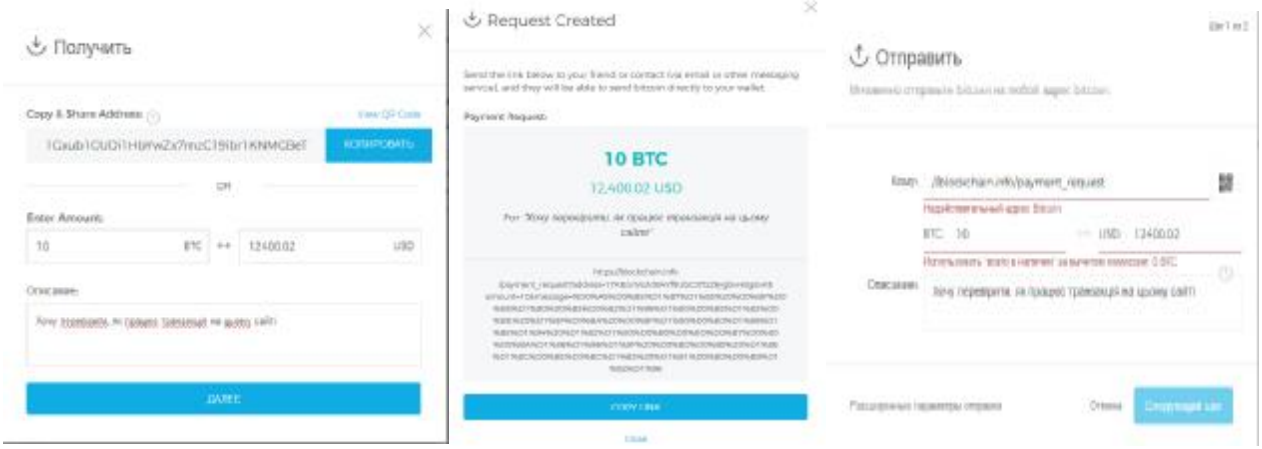


Рис. 1.30. Послідовні вікна операцій з он-лайн гаманцем

Продовжимо опис прикладів роботи з криптовалютами за описом, наданим сайтом http://bitcoin-crypto.blogspot.com/.

Отримати Bitcoin можна абсолютно безкоштовно на спеціальних сайтах роздачі криптовалют, які називають "кранами". До майнінгу такі сайти ніякого відношення не мають.

Bitcoin-кран – це рекламний сайт який розміщує рекламу на своїх сторінках за Біткоїни.

Частину прибутку від показу реклами такі сайти залишають собі, а частину розподіляють між користувачами, як винагороду за перегляд реклами. Для цього користувач виконує якусь просту дію, як правило потрібно ввести капчу, після чого Біткоїни зараховуються на внутрішній рахунок. У найпоширенішому варіанті **CAPTCHA** від користувача потрібно ввести символи, зображені, як правило, в спотвореному вигляді на пропонованому малюнку, іноді з додаванням шуму або напівпрозорості. Коли набирається мінімальна кількість для виплати,

монети виводяться на Біткоїн-гаманець. Всі крани зазвичай виплачують гроші автоматично за розкладом. На деяких сайтах виплати можна замовляти вручну.

Для прикладу зареєструємось на одному з найпопулярніших і перевірених сайтів http://freebitco.in (рис. 1.31). Зараз цей кран трохи оновився, але принцип реєстрації не змінився. Також тут багато різних бонусів, детальніше дивіться в https://vk.com/free___bitcoin цій спільноті ВК. Для завершення реєстрації натискаємо кнопку «Sign UP».

Рис. 1.31. Приклад реєстрації на сайті http://freebitco.in

На більшості кранів для того, щоб почати працювати достатньо ввести Bitcoin-адресу. Приклад роботи на http://freebitco.in. , Входимо на сайт, увівши свої дані. Набираємо капчу (тобто символи, вказані у графічному форматі) у формі і тиснемо кнопку «Roll» (рис. 1.32).

Рис. 1.32. Кінець реєстрації на сайті http://freebitco.in

Випадає комбінація цифр, яка означає ваш виграш (рис. 1.33). У даному випадку виграш склав 0,00000690 біткоїнів. Скільки це буде у гривнях? Помножимо на поточний курс біткоїну до долара – 1250 USD/BTS – та на курс гривні відносно долара – 26,89 UAH/USD. Тобто

0,00000690 x 1250 x 26,89 = 0,23 UAH.

Рис. 1.33. Виграш на сайті http://freebitco.in

Отже, виграш вийшов зовсім малим навіть у гривневому еквіваленті. Більше того, спроба перевести таку мізерну суму нічого не дасть, оскільки більшість бірж бере плату за транзакцію в межах 4 USD!

Дані в таблиці постійно змінюються, бо на цьому проекті можна отримати кожної години 200$ в BTC, то і кількість виграшних монет залежить тільки від курсу BTC/USD.

Основний принцип роботи на всіх сайтах практично однаковий. Відмінності можуть бути в кількості монет, часі через який знову можна збирати BTC, графічному оформленні.

На сьогоднішній день існує дуже багато Bitcoin-кранів, але більшість з них є не дуже надійними, тому тут будуть представлені лише перевірені і платоспроможні сайти (рис. 1.34).

Рис. 1.34. Опис популярних біткоїн-кранів

1.12. Індивідуальне завдання №1

Тема роботи: Вивчення основних прийомів криптування текстів, понять блокчейну, типів криптовалют та операцій з ними.

Мета роботи: Навчитися використовувати Інтернет ресурси для створення щоденників, які неможливо підробити, заведення електронних гаманців для операцій з криптовалютою.

Завдання:

А) 1. Створити серію записів щоденника типу: «1. Отримав 100 грн. 2. Перерахував 50 грн Петру…»

2. Зашифрувати записи щоденника кодами MD5 та SHA-256.

3. Зробити спробу змінити текст таким чином, щоб контрольна сума не змінилася. Зробити висновки щодо такої можливості

Б) 1. Зареєструватися на сайті, що надає послуги електронного гаманця. Відкрити два електронних гаманці.

2. Зайти на біткоїн-кран і заробити трохи BTC за адресами обох гаманців.

3. Перерахувати отриману суму, якою малою вона б не була, на інший біткоїн-гаманець.

4. Збільшену суму використати для придбання будь-якої звичайної валюти.

5. Створити звіт з роботи у текстовому форматі, вставляючи активні вікна з коментарями по ходу звіту. Для цього скористайтеся кнопкою Alt + Print Screen або інструментом «Ножиці».

В) Написати реферат на одну з тем, вказаних у наступній таблиці. Реферат має містити: заголовок, зміст, вступ, основну частину, висновки, список використаних джерел, додатки.

Всі завдання виконуються в електронному вигляді і вміщаються на компакт-диску, який і слугує засобом контролю за виконанням завдань.

№ п/п	Тема реферату
1.	Порядок відсікання розгалужень у блокчейні. Приклади конфліктів в різних криптовалютах щодо можливості вести майнінг паралельно.
2.	Порівняльний аналіз криптовалют Litecoin, Dogecoin, Pinkcoin.
3.	Принципи генерування монет та підтвердження блоків для криптовалют Dogecoin, Pinkcoin.
4.	Опишіть алгоритми створення хеш-коду, відмінні від описаних у розділі.
5.	Опишіть і наведіть приклад реального блоку транзакцій криптовалюти біткоїн.
6.	Прийміть участь у майнінгу і опишіть процес роботи.
7.	Описати зміну курсів 10 криптовалют з моменту їх заснування, виключивши біткоїн та лайткоїн.
8.	Опишіть, які криптовалюти мають дефляційний ефект.
9.	Знайдіть і опишіть електронні гаманці офф-лайн та онлайн, що не описані у цьому розділі.
10.	Придумайте порядок дій по залученню коштів з різних біткоїн-кранів, використайте його і покажіть ефективність.

Контрольні запитання

1. Що таке криптовалюта?
2. Чому кодування відкритих повідомлень робить надійним їх зберігання?
3. Поясність, які типи кодування ви знаєте?
4. Наведіть приклад закодованого щоденника.
5. Створіть власний закодований щоденник.
6. У чому принципова різниця кодування хеш-кодами SHA-2 та base 64?
7. Що таке хеш-код?

8. Поясніть принцип блокчейну
9. Які криптовалюти ви знаєте?
10. Які криптовалюти є найбільш популярними?
11. В чому полягає популярність криптовалют?
12. Коли і ким була створена перша криптовалюта?
13. Що є показником надійності блоку?
14. Як визначається цінність криптовалют?
15. Як створити власний електронний гаманець?
16. На яких пристроях можна створити електронний гаманець?
17. Що таке адреса електронного гаманця?
18. Чи потрібно вводити пароль, коли уже введена адреса електронного гаманця?
19. Як можна генерувати криптовалюту?
20. Які є обмеження на обсяг генерації нових монет криптовалют?
21. Що таке «пули видобутку»?
22. Як можна заробляти криптовалюту?
23. Де можна узнати про крос-курси криптовалют?
24. Чи можна перевести суму у криптовалюті у звичайні гроші?
25. Як можна придбати криптовалюту?
26. Що таке біткоїн-кран?

У розділі студенти вивчили поняття криптування записів у щоденниках, принципи криптування, основні види криптовалют та порядок користування ними.

РОЗДІЛ 2. BIG DATA. ЗБИРАННЯ, ОБРОБКА, АНАЛІЗ ВЕЛИКИХ МАСИВІВ ДАНИХ, РОЗМІЩЕНИХ В ІНТЕРНЕТІ

Засвоївши матеріал цього розділу студент усвідомить поняття, особливості обробки та застосування big data в економічних дослідженнях. Також розглянуто приклад прогнозування обсягу продажів на Kaggle – провідній платформі для змагань з Data Science.

«Зараз краще, ніж будь-коли раніше,
почати вивчення машинного навчання і
штучного інтелекту»

***Ben Hamner**, співзасновник та CTO Kaggle*

Переведення даних у цифрову форму даних в багатьох галузях створило безпрецедентний обсяг масивів, які визначаються великим обсягом, швидкістю обробки, різноманітністю та достовірністю. Найновіші технології, такі як хмарні обчислення та розподілені системи в поєднанні з останніми розробками програмного забезпечення та підходами аналізу дозволили використовувати дані всіх типів для повного та змістовного аналізу для отримання додаткової вартості.

2.1 Методи машинного навчання

Машинне навчання (англ., *machine learning*) – це галузь, що надзвичайно швидко розвивається в останні роки, знаходиться на стику прикладної статистики, чисельних методів оптимізації, дискретного аналізу, і за останні

50 років оформилася в самостійну математичну дисципліну. Для практичного застосування методів машинного навчання експерти розробили високоякісне та ефективне програмне забезпечення з відкритим вихідним кодом та бібліотеками. Нові онлайн-курси і блоги для популяризації та поширення досвіду використання методів машинного навчання з'являються щодня [1-5]. Застосування методів машинного навчання принесло мільярди доларів доходу в різних галузях промисловості, забезпечуючи безпрецедентні ресурси і величезні можливості для працевлаштування [6].

Машинне навчання – клас методів штучного інтелекту, характерною рисою яких є не пряме розв’язання задачі, а навчання в процесі застосування вирішення безлічі подібних завдань. Для побудови таких методів використовуються засоби математичної статистики, чисельних методів, методів оптимізації, теорії ймовірностей, теорії графів, різні техніки роботи з даними в цифровій формі [7].

Розрізняють два типи навчання:

- *Навчання за прецедентами*, або індуктивне навчання, засноване на виявленні закономірностей в емпіричних даних.
- *Дедуктивне навчання* передбачає формалізацію знань експертів і їх перенесення в комп'ютер у вигляді бази знань.

Дедуктивне навчання прийнято відносити до області експертних систем, тому терміни машинне навчання і навчання по прецедентах можна вважати синонімами.

Багато методів машинного навчання розроблялися як альтернатива класичним статистичним підходам. Багато методів тісно пов'язано з витягуванням інформації (*Information extraction*), інтелектуальним аналізом даних (*Data mining*).

Загальна постановка задачі навчання за прецедентами: є множина об'єктів (ситуацій) та множина можливих відповідей (відгуків, реакцій). Існує деяка залежність між відповідями і об'єктами, але вона невідома. Відома тільки кінцева сукупність прецедентів – пар «об'єкт, відповідь», звана *навчальної*

вибіркою. На основі цих даних потрібно встановити залежність, тобто побудувати алгоритм, здатний для будь-якого об'єкта знайти досить правильну відповідь. Для вимірювання точності відповідей певним чином вводиться функціонал якості.

Дана постановка є узагальненням класичних задач апроксимації функцій. У класичних задачах апроксимації об'єктами є дійсні числа або вектори. У реальних прикладних задачах вхідні дані про об'єкти можуть бути неповними, неточними, нечисловими, текстовими, графічними, аудіо та різнорідними. Ці особливості призводять до великої різноманітності методів машинного навчання.

Методи машинного навчання. Так як розділ машинного навчання, з одного боку, утворився в результаті поділу науки про нейромережі на методи навчання мереж і види топологій архітектури мереж, а з іншого, увібрав в себе методи математичної статистики, то наведені нижче способи машинного навчання походять від нейромереж. Тобто базові види нейромереж, такі як перцептрон і багатошаровий перцептрон (а також їх модифікації) можуть навчатися як з учителем, без учителя, з підкріпленням, і активно. Але деякі нейромережі і більшість статистичних методів можна віднести тільки до одного зі способів навчання. Тому якщо потрібно класифікувати методи машинного навчання залежно від способу навчання, то правильніше класифікувати алгоритми навчання нейронних мереж.

- *Навчання з учителем* – для кожного прецеденту задається пара «ситуація, необхідну рішення»;

1. *Метод корекції помилки;*
2. *Метод зворотного поширення помилки;*

- *Навчання без вчителя* – для кожного прецеденту задається тільки «ситуація», потрібно згрупувати об'єкти в кластери, використовуючи дані про попарну схожість об'єктів, і / або знизити розмірність даних:

1. *Альфа-система підкріплення*
2. *Гамма-система підкріплення*

3. *Метод найближчих сусідів*

- *Навчання з підкріпленням* – для кожного прецеденту є пара «ситуація, прийняте рішення»:

1. *Генетичний алгоритм*.

- *Активне навчання* – відрізняється тим, що алгоритм, який навчається має можливість самостійно призначати наступну досліджувану ситуацію, на якій стане відома правильна відповідь:

- *Навчання з частковим залученням вчителя* (*Semi-supervised learning*) – для частини прецедентів задається пара «ситуація, необхідне рішення», а для частини – тільки «ситуація»

- *Трансдуктивне навчання* (англ. *Transduction machine learning*) – навчання з частковим залученням вчителя, коли передбачається робити прогноз тільки для прецедентів з тестової вибірки.

- *Багатозадачне навчання* (англ. *multi-task learning*) – одночасне навчання групи взаємопов'язаних завдань, для кожної з яких задаються свої пари «ситуація, необхідне рішення».

- *Багатоприкладове навчання* (*multiple-instance learning*) – навчання, коли прецеденти можуть бути об'єднані в групи, в кожній з яких для всіх прецедентів є «ситуація», але тільки для одного з них (причому, невідомо якого) є пара «ситуація, необхідне рішення».

Класичні задачі, які вирішуються за допомогою машинного навчання.

- *Класифікація* як правило, виконується за допомогою навчання з учителем на етапі власне навчання.

- *Кластеризація* як правило, виконується за допомогою навчання без учителя.

- *Регресія* як правило, будується за допомогою навчання з учителем на етапі тестування, є окремим випадком задач прогнозування.

- *Зниження розмірності даних і їх візуалізація* виконується за допомогою навчання без учителя.
- *Відновлення щільності розподілу ймовірності* по набору даних.
- *Однокласова класифікація і виявлення новизни.*
- *Побудова рангової залежності.*

Типи вхідних даних при навчанні.

- Ознаковий опис об'єктів – найбільш поширений випадок.
- Опис взаємин між об'єктами, найчастіше відносини попарної подібності, що виражаються за допомогою матриці відстаней, ядр або графа даних.
- Часовий ряд або сигнал.
- Зображення або відеоряд.

Типи функціоналів якості.

При навчанні з учителем – функціонал якості може визначатися як середня помилка відповідей. Передбачається, що шуканий алгоритм повинен його мінімізувати. Для запобігання перенавчання в функціонал якості в явному або неявному вигляді додають регуляризатор.

При навчанні без учителя – функціонали якості можуть визначатися по-різному, наприклад, як відношення середніх міжкластерних і внутрішньо кластерних відстаней.

При навчанні з підкріпленням – функціонали якості визначаються фізичним середовищем, яке б показало якість пристосування агенту.

Практичні сфери застосування. Метою машинного навчання є часткова або повна автоматизація вирішення складних професійних завдань в самих різних областях людської діяльності.

- Розпізнавання мови;
- Розпізнавання жестів;
- Функція розпізнавання рукописного тексту;
- Розпізнавання образів;

- Технічна діагностика;
- Медична діагностика;
- Прогнозування часових рядів;
- Біоінформатика;
- Виявлення шахрайства;
- Виявлення спаму;
- Категоризація документів;
- Біржовий технічний аналіз;
- Фінансовий нагляд;
- Оцінка кредитоплатіжності;
- Передбачення витрат клієнтів;
- Хемоінформатика
- Навчання ранжируванню в інформаційному пошуку.

Сфера застосувань машинного навчання постійно розширюється. Повсюдна інформатизація призводить до накопичення величезних обсягів даних в науці, виробництві, бізнесі, транспорті, охороні здоров'я. Виникаючі при цьому завдання прогнозування, управління та прийняття рішень часто зводяться до навчання по прецедентах. Раніше, коли таких даних не було, ці завдання або взагалі не ставилися, або вирішувалися зовсім іншими методами.

На рис. 2.1 наведено узагальнену методологію, що використовується при машинному навчанні.

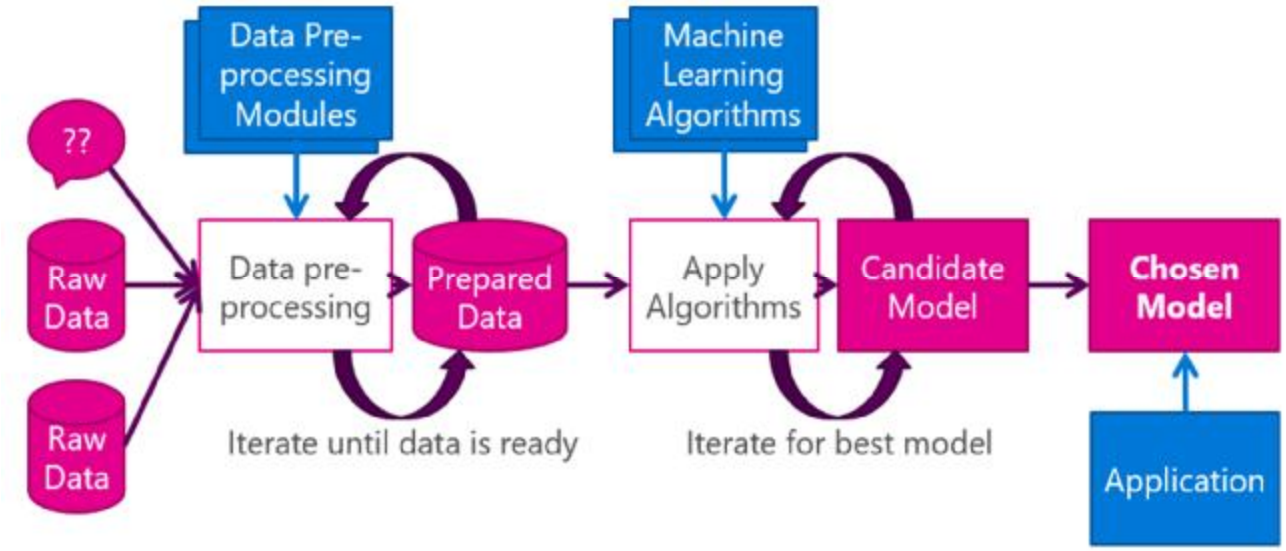


Рис. 2.1 Методологія, що використовується при машинному навчанні

Згідно рис. 2.1 першим етапом аналізу даних виступає попередня обробка даних заповнення відсутніх та видалення аномальних значень, шуму, визначення значимих факторів, розподілу даних, створення тестового, навчального та перевірочного набору даних, попередній аналіз даних. Підготовка даних та створення грамотних ознак – це часто (але не завжди) важливіше, ніж складні моделі. Наступний етап застосування до даних методів машинного навчання та обрання найкращої моделі згідно обраної метрики оцінки якості моделей. Обрати модель та застосувати її до перевірочного набору даних.

2.2. Поняття *big data*, як складової процесу машинного навчання

Big data відноситься до даних, які через їх розмір, швидкість або формат, не можна зберігати, обробляти або аналізувати за допомогою традиційних методів, таких як електронні таблиці, реляційні бази даних або загальне статистичне програмне забезпечення. Розглянемо практичне визначення *big data*, як воно відноситься до областей науки про дані, статистики та програмування і різноманітність людей і навичок, пов'язаних з великими даними.

Термін «*Big data*» «*Великі дані*» введено Кліффордом Лінчем у 2008 році, в червні 2011 року консалтингова компанія McKinsey випустила доповідь [9], в якій оцінила потенційний ринок великих даних в мільярди доларів.

Google Trends показує початок активного росту вживання словосполучення «*Big data*» у світі починаючи з 2012 року, в Україні з 2014 року (рис.2.2) :

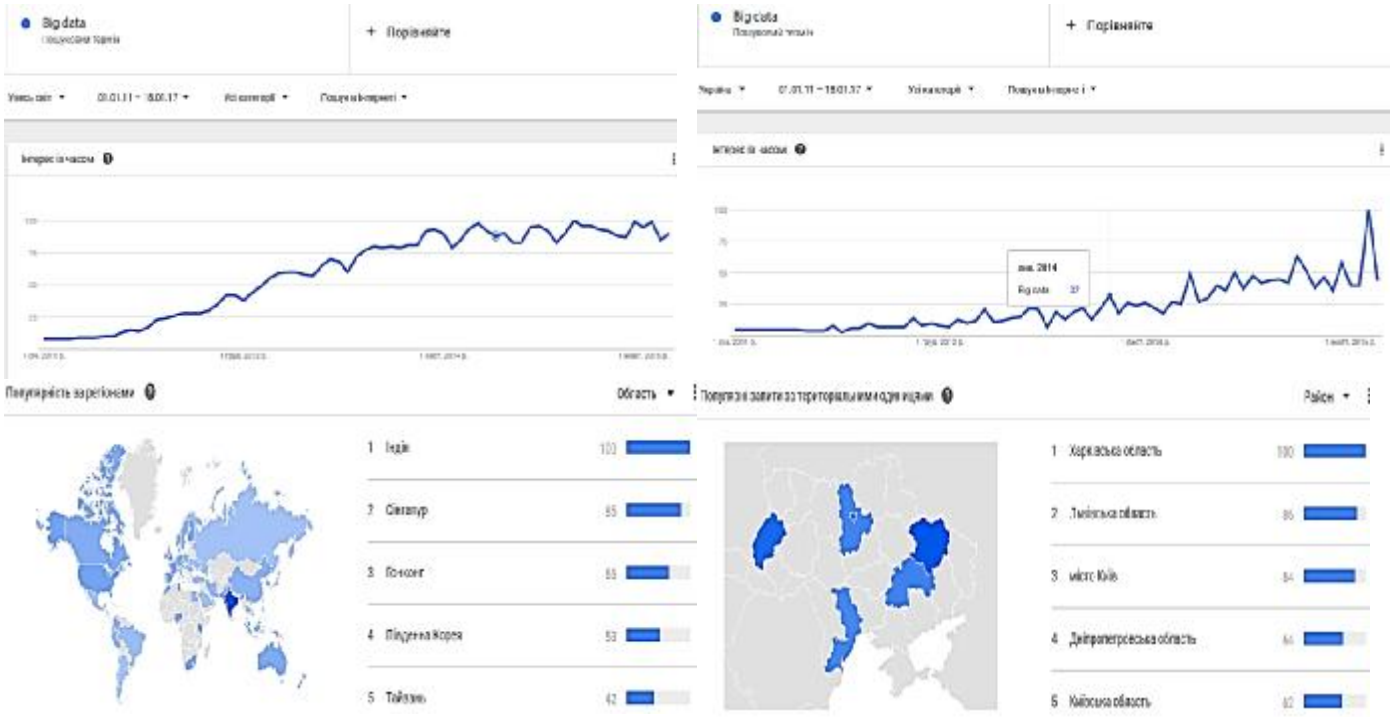


Рис. 2.2 Статистика вживання словосполучення «*Big data*» в Україні та світі.

Як видно з рис. 2.2, в Україні та світі, термін «*Big data»* найчастіше застосовують в осередках розвитку IT індустрії, в Харківській, Львівській, Київській та Дніпропетровській областях.

Вели́кі да́ні (англ. *Big Data*) – в інформаційних технологіях, це серія підходів, інструментів і методів обробки структурованих і неструктурованих даних величезних обсягів і значного різноманіття, ефективних в умовах безперервного приросту, розподілу по численних вузлах обчислювальної

мережі, альтернативних традиційним системам управління базами даних і рішенням класу *Business Intelligence* [7].

Враховуючи обсяг, природу та джерела отримання даних традиційні способи та підходи, здебільшого засновані на рішеннях класу задач бізнес-аналітики та системах управління базами даних, не можуть бути застосовані до цих даних [10] .

Тому системи для накопичення *big data* мають задовольняти вимогам:

1. *Горизонтальна масштабованість*. Оскільки обсяг даних постійно зростає – будь-яка система, для обробки великих даних, повинна бути розширюваною.

2. *Відмовостійкість*. Принцип горизонтальної масштабованості передбачає, що машин в кластері може бути багато. Це означає, що частина цих машин буде гарантовано виходити з ладу. Методи роботи з великими даними повинні враховувати можливість таких збоїв і переживати їх без будь-яких значущих наслідків та втрат.

3. *Локальність даних*. У великих розподілених системах дані розподілені по великій кількості машин. Якщо дані фізично знаходяться на одному сервері, а обробляються на іншому – витрати на передачу даних можуть перевищити витрати на саму обробку. Тому одним з найважливіших принципів проектування *BigData*-рішень є принцип локальності даних – по можливості обробляємо дані на тій же машині, на якій вони зберігаються. Технологічно це реалізується за допомогою механізму *MapReduce*, який в свою чергу реалізовано в *Hadoop* [11].

Великі дані відіграватимуть величезну роль в електронній комерції в найближчому майбутньому. Інтернет-магазини вже використовують аналітичні інструменти для аналізу кошика покупок або для відображення індивідуального контенту на основі IP-адреси через CMS. Застосування *big data* буде розширювати ці технічні функції в короткостроковій перспективі. Мета полягає в тому, щоб надати клієнтам індивідуальний і оптимізований торговий досвід в режимі реального часу.

Обговоримо, як *big data* використовується в таких областях, як маркетинг і наукові дослідження, як *big data* впливає на обслуговування клієнтів, рекомендації і етичні питання, що постають з використанням *big data*. Нарешті, розглянемо більш загальні методи для створення або накопичення великих обсягів даних, зберігання і обробки *big data* і аналізу та візуалізації великих обсягів даних, в тому числі інтелектуального аналізу даних і прогностичної аналітики [8].

Базова ідея, що лежить за фразою *big data* полягає в тому, що все що ми робимо залишає оцифрований слід (чи дані), які ми (чи інші) можуть використовувати чи аналізувати. Тому методи *big data* відносяться до можливості використовувати все, що збільшує обсяг даних.

Використання *big data* для споживачів. В більшості випадків, коли чуєте, що говорять про *big data*, то говорять про це в комерційних умовах про те, як компанії можуть використовувати великі дані в рекламних або маркетингових стратегіях. Але водночас важливо, що *big data* використовуються також для широкого загалу споживачів, а також цікавим є те, що має місце неймовірно складна обробка даних і алгоритмів майже непомітно. Результати настільки переконливі, що вони дають вам тільки маленький шматочок інформації, а саме той, що вам потрібно.

Нижче наведено приклади, що показують як деякі загальні додатки використовують *big data* для споживачів, навіть якщо ті не обізнані про складність аналізу *big data*, що відбувається в цих додатках. Перш за все, якщо у вас є *Apple*, *iPhone* або *IPad*, додаток *Siri* знає, де ви перебуваєте, в який час ви говорите, він може, знайти ресторани певного виду їжі і подивитися, якщо в них є доступні місця.

Аккаунт в додатку *Spotify* знає, що користувач слухає коли він застосовує *Spotify*, і що слухає коли ви не в додатку, які композиції додаєте та вилучаєте, що дозволяє зробити специфічні саме для даного користувача пропозиції, які допоможуть знайти нових виконавців, про яких в іншому випадку він би не взнав. Таким же чином, *Amazon.com* робить рекомендації для

книг. Наприклад, якщо в пошуковику Amazon ввести: «Принципи великих даних», автора Джулс Берман, то він видає список інших книг, які рекомендуються за цією темою. Вони породжені рекомендаціями двигуна Amazon, і ви побачите кілька інших книг про великі обсяги даних.

Багато людей використовують *Netflix* для пошуку фільмів. Netflix створює конкретні пропозиції інших фільмів, які б могли сподобатись користувачу. Кілька років тому компанія проводила великий конкурс під назвою *Netflix Prize*, вони хотіли визначити, чи хтось може поліпшити точність їх прогнозів. Якби хтось зміг поліпшити ці прогнози на 10% отримав би приз мільйон доларів. Відбувався надзвичайно складний аналіз, здобутки якого увійшли в кінцевий додаток, але результат знову проста річ, користувач отримує повний список рекомендованих фільмів, і зазвичай вибирає один, який подобається.

В іншому контексті працює додаток під назвою *Neighborland*, який розроблено, щоб допомогти людям співпрацювати з метою зробити де працюють місто кращим. Ціль завдання проста, але для цього Neighborland використовує фотографії, а також дані і API, з Twitter і Google Maps, і Instagram, а також установ, які повідомляють про власників нерухомості, він використовує транзитні системи, величезний набір даних, який дійсно підкреслює різноманітність великих обсягів даних. Інші додатки, наприклад, Spotify і Yelp, також використовують величезний обсяг, але цей додаток показує різноманітність інтеграції даних з різних місць і так різних форматів, щоб допомогти людям працювати разом над поліпшенням свого оточення.

І нарешті як працює додаток *Google Now* (використовує обробку природної мови для відповідей на питання, створення рекомендацій та виконання різних дій). Відповідаючи на різні запити користувача, Google Now відображає інформацію залежно від уподобань користувача, пророкуючи їх на основі його звичок і режиму дня, як *Google Now* насправді дає рекомендації, по-перше вони пов'язані з календарем, і пов'язані з визначенням місця розташування з використанням телефону. Він знає, де ви знаходитесь, він знає,

де ви повинні бути, і він може сказати вам про дорожній рух або погоду, перш ніж ви навіть просите про це, і це засноване на, знову ж таки, величезній кількості інформації про пошук інформації для людей.

Таким чином, для споживачів, великі дані відіграють величезну роль в забезпеченні цінних послуг, але знову ж таки, з іронією можна сказати, що вони працюють невидимо, обробляючи величезну кількість інформації з різних джерел для використання в двох або трьох аспектах, які дають вам те, що вам необхідно [12].

2.3. Перспективи, недоліки, складність / особливість обробки *big data*

Інтернет-магазини можуть поліпшити свій бізнес з великими даними в наступних областях:

1. *Оптимізований портфель продуктів*. Аналіз великих обсягів структурованих даних клієнта дозволяє провести детальний аналіз цільової групи. На підставі результатів, портфель інтернет-магазину може бути адаптовано під потреби та вимоги конкретного користувача. Особливо великі інтернет-продавці можуть масштабувати свої пропозиції з *big data* краще для задоволення потреб конкретних клієнтів. З іншого боку *big data* також дозволяють прогнозувати потреби клієнтів і надають можливість майбутньої оптимізації продуктового портфеля. Таким чином, з використанням великих даних можна оптимізувати витрати на складі.

2. *Оптимізація ціни*. Завдяки *big data*, стає можливим інтелектуальний аналіз даних в реальному масштабі часу. Інтернет-магазин може налаштувати динамічну ціну продукту. Завдяки високій прозорості в Інтернеті, необхідно мати конкурентів завжди під наглядом і коригувати власну ціну для того, щоб залишатися конкурентоспроможними. *Big data* пропонують всебічний аналіз ринку для динамічної цінової політики.

3. *Оптимізований інтернет-магазин*. Завдяки використанню великих обсягів даних і швидких технологій веб-серверів, можна забезпечити динамічні веб-сайти. Різні стартові сторінки або цільові сторінки можуть відображатися в залежності від регіону або цільової групи. Крім того, можуть бути відображені різні переваги щодо асортименту продукції для чоловіків і жінок. Завдяки аналізу великих обсягів даних не існує практично ніяких обмежень можливих преференцій для оптимізації онлайн-магазину.

4. Оптимізація інтернет реклами. У минулому показ інтернет реклами не був ефективним. На даний час з використанням *big data* онлайн роздрібні торгівці можуть орієнтувати цільові рекламу та пропозиції для своїх клієнтів, що дозволить залучити нових клієнтів. У режимі реального часу реклама дешевша і ефективніша, тому з великими даними інтернет-магазини можуть скоротити витрати на рекламу.

5. *Оптимізація обслуговування клієнтів*. Якщо клієнт не задоволений продуктом і скаржився по телефону, а не по електронній пошті на обслуговування, то з часом така інформації буде втрачена. Великою перевагою буде, якщо співробітник служби підтримки або маркетингу зможе використовувати повну історію клієнта, збагачену будь-якою соціальною інформацією, з засобів масової інформації під час спілкування, цей сценарій можливо реалізувати з *big data*. Різноманітність цінної довідкової інформації про клієнта дозволить службі підтримки істотно поліпшити ставлення до клієнтів. Якість підтримки також є конкурентною перевагою [13].

Природа та походження *big data* обумовлюють використання як традиційних методів обробки даних так і специфічних для *big data*:

- *Data Mining* – навчання асоціативним правилам, класифікація (методи категоризації нових даних на основі принципів, раніше застосованих до вже наявних даних), кластерний аналіз, регресійний аналіз;
- *Краудсорсінг* – категоризація та збагачення даних силами широкого, невизначеного кола осіб;

- *Змішання і інтеграція даних* – набір технік, що дозволяють інтегрувати різнорідні дані з різноманітних джерел для можливості глибинного аналізу;

- *Машинне навчання*, включаючи навчання з учителем і без учителя, а також використання моделей, побудованих на базі статистичного аналізу або машинного навчання для отримання комплексних прогнозів на основі базових моделей;

- *Штучні нейронні мережі, мережевий аналіз, оптимізація*, в тому числі генетичні алгоритми;

- *Просторовий аналіз* – використання топологічної, геометричній і географічної інформації в даних;

- *Статистичний аналіз*: *α/β*-тестування і аналіз часових рядів;

- *Візуалізація аналітичних даних* – подання інформації у вигляді малюнків, графіків, схем і діаграм з використанням інтерактивних можливостей та анімації як для результатів, так і для використання в якості вихідних даних для подальшого аналізу.

Відмінність методів, що застосовують для аналізу *big data* від методів бізнес-аналізу (рис.2.3) обумовлена не лише обсягом даних, що обробляються, а насамперед їх неструктурованістю та швидкістю надходження:

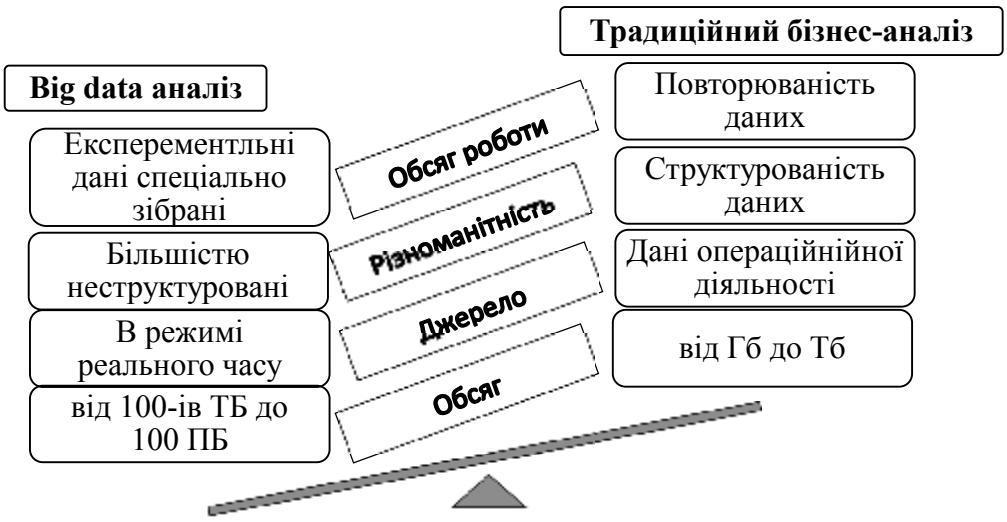


Рис. 2.3 Відмінність методів *big data* від бізнес-аналізу

Ще декілька років тому не аналізували електронні листи, PDF-файли або відеоматеріали. Розподілені обчислення з'явилися не вчора, проте можливість миттєво розподіляти і масштабувати систему – і при менших витратах – це нововведення. У самому бажанні передбачати майбутнє немає нічого нового, однак новою є можливість використовувати і зберігати всі створювані дані.

Поряд з широкими перспективами використання *big data*, існують певні організаційні, технічні та технологічні проблеми, що потребують усвідомлення та врахування при застосуванні *big data*.

1) Актуальними є питання про захист приватних даних при застосування хмарних технологій

2) Візуалізація даних

3) Аналіз аномальних значень, з’ясування причин їх виникнення

Відмінність *big data* від малих даних. *Big data* відрізняються від малих даних не лише характеристиками наведеними в пункті 2.4. Нижче наведено 10 практичних відмінностей за Дж. Берманом [14]: підготовка, надання доступу та комплекс аналізу.

1) **Ціль**. «Малі» дані зазвичай збирають для конкретної мети. Малі дані зазвичай збираються для конкретної мети, розв’язання конкретної задачі. Аналіз великих даних, може мати початково одну мету, яка може розвитися або прийняти несподівані напрямки.

2) **Розташування**. Малі дані, як правило, знаходяться в одному місці, і часто в одному комп'ютерному файлі. Великі дані з іншого боку, можуть бути в декількох файлах на декількох серверах на комп'ютерах в різних географічних точках.

3) **Структура даних і зміст**. Малі дані, як правило, добре структуровані як таблиця Excel, складаються з рядків і стовпців даних. Великі дані з іншого боку, можуть бути неструктурованими, можуть мати безліч форматів в файлах, а також містити посилання на інші ресурси.

4) **Підготовка даних**. Малі дані зазвичай отримує кінцевий користувач для своїх цілей у підготовленому для обробки вигляді, а великі дані часто готує одна група людей, аналізуються другою групою людей, а потім використовуються третьою групою людей, які можуть мають різні цілі, різні галузі знань.

5) **Довговічність**. Малі дані зазвичай зберігаються протягом певного часу після того, як проект закінчений, тому що для нього є чітка кінцева точка. У науковому світі це може бути, п'ять або сім років, а потім дані можна викинути, проте великі дані перетікають з одного проекту в інший можуть залишатися там протягом дуже довгого часу.

Вони можуть бути використані як нові вхідні дані, або контекстні дані, до вже існуючих проектів, або додаткові змінні, або як зв'язуючи між різними файлами. Тому вони мають набагато довший цикл життя в порівнянні з невеликим набором даних.

6) **Вимірювання**. Малі дані, як правило, вимірюються за допомогою простих одиниць вимірювання, і це зазвичай робиться одночасно для всього масиву. Тому що джерела можуть надходити з різних місць, в різний час, від різних організацій та країн, великі дані з іншого боку можуть вимірюватися в різних одиницях вимірювання, і можливо, доведеться зробити чимало перетворень, щоб отримати дані придатні для аналізу.

7) **Надійність**. Малі набори даних зазвичай можуть бути відтворені в повному обсязі, якщо щось піде не так в процесі обробки. Великі набори даних, тому що вони надходять в різних формах і з різних джерел, не можуть бути відтворені в разі збою, якщо щось пішло не так. Зазвичай краще, що ви можете, принаймні, визначити, які частини даних проекту є проблематичними і мати це на увазі, коли ви працюєте з такими даними.

8) **Вартість**. З невеликими даними, якщо справи йдуть погано, витрати обмежені, це не велика проблема, а з великими даними, проекти можуть коштувати сотні мільйонів доларів, а втрати даних або пошкодження

даних можуть приректи проект, можливо, навіть кар'єру дослідника чи існування організації.

9) **Самоаналіз**, і це означає, що дані описують себе з важливого боку. При невеликих даних, ідеально, наприклад, те, що називається трійка, яка використовується в декількох мовах програмування, по-перше, визначаєте об'єкт, який вимірюється. Наприклад, Солт-Лейк-Сіті, штат Юта, США. По-друге, визначаєте, що вимірюється, дескриптор для визначення даних. У цьому випадку середня висота в футах. Потім третє, ви даєте значення даних безпосередньо, 4,226 футів над рівнем моря. У невеликому наборі даних, як правило, дані добре організовані, окремі точки даних ідентифіковані, як правило, ясно, що показники означають. У великому наборі даних, однак, тому що все може бути настільки складним, з великою кількістю файлів і в багатьох форматах, можете в кінцевому підсумку отримати неясну, нецілісну інформацію, або просто безглузду.

Очевидно, це ставить під загрозу корисність великих обсягів даних в таких ситуаціях.

10) **Аналіз**. При невеликих даних зазвичай можливо аналізувати всі дані відразу в одній процедурі з одного комп'ютерного файлу. З великими даними, однак, з огляду на величезний обсяг, та розкиданість по багатьох різних файлах і серверах, можливо, доведеться пройти через витягування, підготовку, зниження розмірності, нормалізацію, перетворення й інші дії, з окремими частинами даних, щоб зробити їх більш керованими, а потім в решті зібрати в один набір даних результати.

Таким чином, стає ясно, що *big data* відрізняються від звичайних даних більше ніж просто за обсягом, і швидкістю, і різноманітністю. Є цілий ряд практичних питань, які роблять обробку та використання більш складними з великими даними, ніж з невеликими даними.

2.4. Основні властивості - 5V.

В якості визначальних характеристик для великих даних відзначають «п'ять V» (рис. 2.4) – *volume* (обсяг), *velocity* (швидкість), *variety* (різноманіття) (рис.2.4), останнім часом додано ще дві важливі характеристики – *veracity* (достовірність) і *value* (*цінність*):

- *обсяг* (*volume*) – в сенсі величини фізичного обсягу,
- *швидкість* (*velocity*) – в сенсі як швидкості приросту, так і необхідності високошвидкісної обробки і отримання результатів,
- *різноманіття* (*variety*) – в сенсі можливості одночасної обробки різних типів структурованих і неструктурованих даних,
- *достовірність* – відноситься до зміщеності, шуму і аномалій в даних, невимірної невизначеності і правдивості та достовірності даних,
- *цінність* – в сенсі отримання користі та/або прибутку з цих даних.

На рис. 2.4. представлено основні характеристики великих даних:

Рис. 2.4 Основні характеристики *big data*.

Характеристика ***обсяг*** відноситься до величезної кількості даних, що генеруються кожну секунду. Подумайте про обсяг всіх повідомлень електронної пошти, повідомлень твіттеру, фотографій, відеокліпів, даних датчиків і т.д., що ми генеруємо і надсилаємо один одному. На одному лише Facebook відправляється 10 мільярдів повідомлень в день, натискається кнопка «Подобається» 4,5 мільярди разів і завантажується 350 мільйонів нових фотографій кожен день. Якщо взяти всі дані, згенеровані в світі між початком часу і 2008 роком, таж сама кількість даних, незабаром буде генеруватися кожну хвилину! Все частіше створюються набори даних занадто великого обсягу для зберігання і аналізу з використанням традиційної технології баз даних. З технологією *big data* тепер ми можемо зберігати і використовувати ці дані набори за допомогою розподілених систем, де частини даних зберігаються в різних місцях і обробляються за допомогою спеціального програмного забезпечення.

Характеристика ***швидкість*** відноситься до швидкості, з якою генеруються нові дані, і швидкості, з якою дані обробляються. Врахуйте швидкість розсилки соціальних медіа повідомлень, що йдуть в вірусних повідомленнях, швидкість, з якою операції з кредитними картами перевіряються на наявність шахрайської діяльності, або мілісекунди за які торгові системи аналізують діяльність в соціальних мережах, щоб отримати сигнали, які спонукають рішення про покупку або продаж акцій або товарів. Технології *big data* дозволяють аналізувати дані, під час генерації, ніколи не поміщаючи їх в базу даних.

Характеристика ***різноманітність*** відноситься до різних типів даних, що аналізуються. У минулому обробляли структуровані дані, акуратно поміщені в таблиці або реляційні бази даних, такі як фінансові дані (наприклад продажу за видами продукції або регіону). Насправді, 80% даних в світі тепер неструктуровані, і, отже, їх не можна помістити в таблиці (наприклад, фотографії, відеофрагменти або оновлення в соціальних медіа). З технологією

big data тепер можемо використовувати при обробці дані, що розрізняються типами даних (структуровані і неструктуровані) в тому числі повідомлення, соціальні медіа розмови, фотографії, дані датчиків, відео або голосових записів і зібрати їх разом з більш традиційними, структурованими даними.

Характеристика **достовірність** відноситься до характеристики наявності шуму або достовірності даних. Для багатьох форм великих обсягів даних якість і точність мало керовані (тільки подумайте Twitter повідомлення з хеш-тегів, абревіатур, помилок і розмовної мови, а також надійність і точність змісту), але технології аналітики великих обсягів даних тепер дозволяють працювати такими типами даних.

Достовірність відноситься до зміщення, шуму і аномалій в даних, невимірної невизначеності і правдивості та достовірності даних.

Характеристика **цінність** даних: добре, мати доступ до великих обсягів даних, але, якщо не можна отримати користь/прибуток з цих даних, то накопичення, зберігання та обробка цих даних – марно витрачені кошти. Таким чином, можна стверджувати, що «цінність» є найбільш важливою характеристикою *big data*. Важливо, отримувати переваги чи важелі впливу в результаті використання *big data*. Легко потрапити в пастку на ідеї застосування *big data* без чіткого розуміння витрат і вигоди [16].

Перетворення *big data* в цінність рис. 2.5:

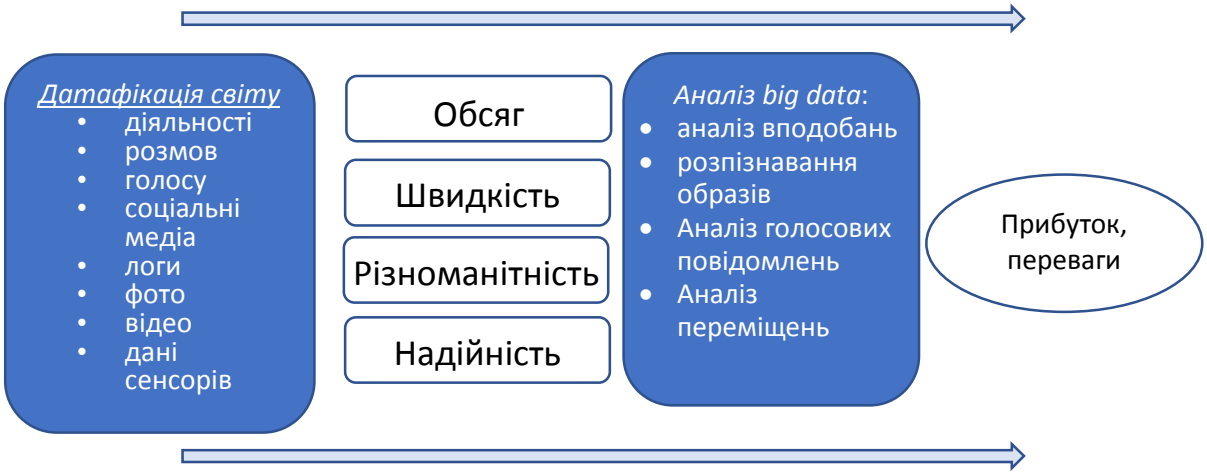


Рис. 2.5 Процес перетворення *big data* в цінність

Джерело [17].

Важливо відзначити, що мета обробки великих даних, полягає в підтримці прийняття рішень. Недостатньо лише збирати і зберігати дані, необхідно збирати і обробляти величезні обсяги складних даних, щоб зрозуміти тенденції, виявити приховані закономірності, виявити аномалії і т.д., так якщо у вас є краще розуміння проблеми та її аналіз, то стає можливим прийняття більш обґрунтованого рішення, керованого даними.

Для вирішення проблеми великих обсягів даних необхідні інноваційні технології. Паралельні, розподілені обчислювальні парадигми, алгоритми машинного навчання масштабовані, в режимі реального часу виконання запиту є ключовими для аналізу великих обсягів даних. Розподілені файлові системи, обчислювальні кластери, хмарні обчислення і сховища даних, що підтримують різні даних і маневреність також необхідні забезпечити інфраструктуру для обробки великих обсягів даних. Робочі процеси забезпечують інтуїтивний, багаторазовий, масштабований і відтворений спосіб обробки великих даних, щоб отримати перевірені результати з нього [17].

Основні задачі для розв’язання з використанням *big data*:

1. Моделювання ризиків.
2. Аналіз плинності клієнтів.
3. Механізм рекомендацій.
4. Спеціальне таргетування, позіціонування.
5. Аналіз транзакцій.
6. Аналіз мережевих даних для прогнозування банкрутства.
7. Аналіз загроз.
8. Торговельні спостереження.
9. Дослідження якості.
10. Для кращого розуміння потреб та сегметування споживачів, компанії розширюють свої бази даних даними соціальних медіа, браузерів, текстовою аналітикою та сенсорними даними для отримання більш повної картини вимог своїх споживачів. Для цього застосовують прогностичні моделі.

Використовуючи *big data* компанії можуть точніше спрогнозувати вибір споживачів, а роздрібні торговці, які товари будить користуватися попитом, страхувальники розуміти чи належним чином поводяться їх клієнти.

Обчислювальна здатність аналізу *big data* дозволяє якісніше діагностувати та виявити хвороби, напрями лікування та прогнозування захворюваності пацієнтів. Використовують всілякі дані з «розумних» пристроїв, годинників для встановлення зв'язків між стилем життя та захворюваністю.

Аналітики *big data* також дозволяють відслідковувати та прогнозувати спалахи епідемій та стрибків захворюваності, лише слухаючи, що люди говорять, наприклад, «Мене лихоманить», «У ліжко з ознобом» чи з пошукових запитів «ліки від грипу» може свідчити про початок епідемії.

11. Для покращення безпеки та правозастосування. Служби безпеки використовують аналіз *big data*, щоб завадити терористичним планам. Сили поліції використовують інструментарій *big data*, для прогнозування криміногенної ситуації. Компанії, що працюють з кредитними картками застосовують аналіз *big data* для попередження шахрайства по транзакціям.

12. Для вдосконалення та оптимізації інфраструктури міст та країн. Наприклад, дозволяє містам оптимізувати трафік дорожнього руху враховуючи інформацію про інтенсивність руху в режимі реального часу, також з соціальних медіа та метеорологічні дані. В більшості міст на даний час використовують аналітику *big data* для перетворення в «Розумне місто», в якому транспортна інфраструктура та допоміжні процеси поєднані. В якому автобус буде чекати на затриманий поїзд, сигнали світлофора враховують інтенсивність руху та мінімізують наявність заторів на дорогах.

Вище наведено лише короткий, початковий огляд трансформації в економіку *big data.* Будь який бізнес, без використання *big data*, ризикує стати аусайдером на ринку.

Процес перетворення даних в знання для отримання цінності з великих даних (рис. 2.6):

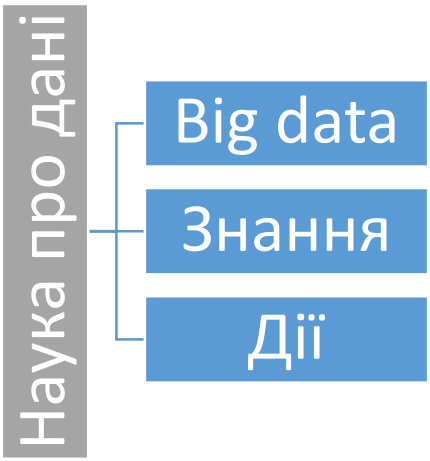


Рис. 2.6 Перетворення даних в знання

Наука про дані перетворює дані в знання чи навіть дії. Що це означає на практиці? Науку про дані, можна розглядати як основу імперичного дослідження, де дані використовують для пояснення інформації про фактичні спостереження. Ці спостереження в основному дані, в нашому випадку *big data*, пов’язані з бізнесом чи наукою. «Знання» є терміном, для позначення продуктів з наукових даних.

Ці спостереження в основному дані, в нашому випадку, великі дані, пов'язані з бізнесом або науковими цілями. Знання є терміном, який використовують для позначення продуктів наукових даних. Вони витягуються з різноманітної кількості джерел даних шляхом поєднання дослідницького аналізу даних і моделювання. Питання іноді конкретні, а іноді вимагають розгляду даних і алгоритмів для відповіді на конкретне питання.

Також важливо, усвідомлювати, що наука про дані не є статичною. Це не одноразовий аналіз. Він включає процес, в якому модель згенеровано, щоб привести до знання через постійне вдосконалення, через нові емпіричні дані, або просто дані.

Наука про дані будується на перетині комп’ютерних наук, математики та бізнес-експертизи рис. 2.7.

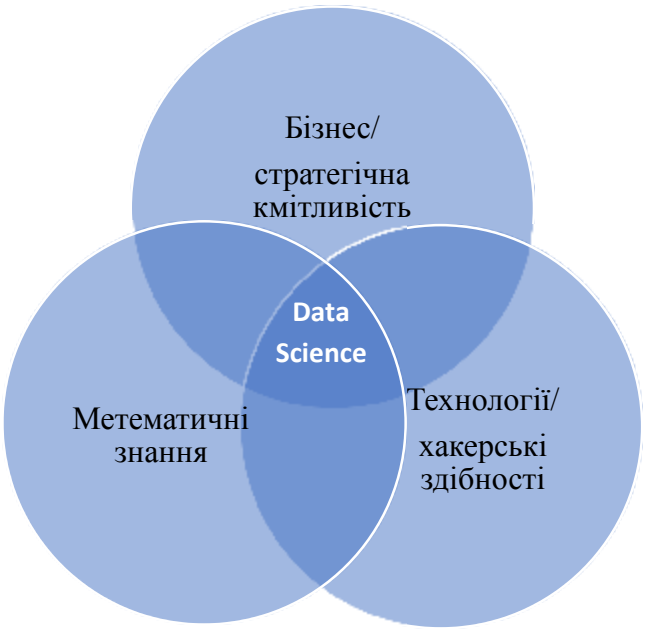


Рис. 2.7 Місце науки про дані в середовищі інформаційних технологій

Всі ці кола вимагають глибоких знань та навичок у таких сферах, як експертиза домену, інжиніринг даних, статистика та обчислення. Ще глибший аналіз цих навичок приведе вас до навиків, таких як машинне навчання, статистичне моделювання, реляційна алгебра, ділової хватки, вирішення проблем та візуалізації даних.

Процес обробки даних науки містить рекомендації щодо впровадження рішень великих на основі *big data*, оскільки це допомагає організувати зусилля та гарантує, що всі критичні кроки, виконані відповідно до попередньо визначених та узгоджених показників.

- **Мета**: Мета стосується проблеми чи набору завдань, визначених стратегією для великих даних. Мета може бути пов'язана з науковим аналізом з гіпотезою або бізнес-метрикою, яка потребує аналізу, заснованого часто на b*ig data*.
- **Люди**: науковці, що працюють з *big data* часто розглядаються як людей, які володіють навичками з різних галузей, включаючи: знання в галузі науки або бізнесу; аналіз з використанням статистики, машинного навчання та математичних методів; управління даними, програмування та обчислення. На

практиці це, як правило, група дослідників, що складаються з людей з додатковими навичками.

- **Процес**: Як тільки попередньо визначена команда з метою, необхідно організувати процес роботи, з певними ітераціями. Процес наукової обробки даних включає в себе методи статистики, машинного навчання, програмування, обчислення та керування даними. Процес концептуальний спочатку і визначено як набір кроків і кожен може вкласти свій внесок у загальний процес. Зауважте, що подібні багаторазові процеси можуть бути застосовані для багатьох додатків з різними цілями, коли вони використовуються в різних робочих процесах. Робочі процеси науки про дані об'єднують такі кроки у виконуваних графіках. Вважаємо, що процесно-орієнтоване мислення є трансформаційним способом ведення наукових даних, щоб зв'язати людей і техніку з додатками. Виконання такого процесу обробки даних вимагає доступу до багатьох наборів даних, великих та малих, що призводить до нових можливостей і труднощів для *Data Science*. Є багато етапів або завдань *Data Science*, таких як збір даних, очищення даних, обробка даних / аналіз, візуалізація результатів, що призвело до робочого процесу *data science*. Процеси обробки даних можуть потребувати взаємодії користувача та інших ручних операцій або бути повністю автоматизованими. Задачі для процесу обробки даних включають: 1) як легко інтегрувати всі необхідні завдання для створення такого процесу; 2) як знайти найкращі обчислювальні ресурси і ефективно запланувати застосування процедур до ресурсів на основі визначення процесу, параметрів та параметрів користувача.
- **Платформи**: виходячи з потреб цільової програми та обсягу даних та обчислень, необхідних для виконання поставленого завдання, різноманітні обчислювальні та інформаційні платформи можуть використовуватися як частина процесу обробки даних. Ця масштабованість повинна бути частиною будь-якої архітектури рішень для науки про данні.
- **Програмовість**. Визначення масштабованого процесу обробки даних вимагає допомоги застосування мов програмування, наприклад, *R, Python*

та шаблонів, наприклад, *MapReduce*. Інструменти, що забезпечують доступ до таких методів програмування, є ключовими елементами для того, щоб зробити програмування даних інформацією на різних платформах.

Підсумовуючи, можна зробити висновок, що наука про дані може бути визначена як майстерність використання п'яти частин, визначених вище. Маючи процеси взаємодії між людьми, які більше керуються завданнями бізнесу і досягненням поставлених цілей, а також більш технічні платформи та платформи та їх програмування, це призводить до оптимального підходу, який починається і закінчується певною діловою цінністю, відповідальністю перед командою та співпрацею [19].

Існують три типи завдань пов'язаних з *big data*:

1. Зберігання і управління. Обсяг даних в сотні терабайт або петабайт не дозволяє легко зберігати і управляти ними за допомогою традиційних реляційних баз даних.

Big data зазвичай зберігаються і організовуються в розподілених файлових системах. У загальних рисах, інформація зберігається на декількох (іноді тисячах) жорстких дисках, на стандартних комп'ютерах. Так звана «карта» (map) відстежує, де (на якому комп'ютері і / або диску) зберігається конкретна частина інформації. Для забезпечення стійкості до відмов та надійності, кожну частину інформації зазвичай зберігають кілька разів, наприклад – тричі. Так, наприклад, припустимо, що зібрано дані про індивідуальні транзакції у великій роздрібній мережі магазинів. Детальна інформація про кожну транзакцію буде зберігатися на різних серверах і жорстких дисках, а «карта» (map) індексує, де саме зберігаються відомості про відповідну угоду.

За допомогою стандартного устаткування і відкритих програмних засобів для управління цією розподіленою файловою системою (наприклад, *Hadoop*), порівняно легко можна реалізувати надійні сховища даних в масштабі петабайт.

2. Неструктурована інформація

Більшість даних *big data* є неструктурованими, тобто велика частина зібраної інформації в розподіленій файловій системі складається з неструктурованих даних, таких як текст, зображення, фотографії або відео.

Це має свої переваги і недоліки. Перевага полягає в тому, що можливість зберігання великих даних дозволяє зберігати "всі дані", не турбуючись про те, яка частина даних актуальна для подальшого аналізу і прийняття рішення. Недоліком є те, що в таких випадках для отримання корисної інформації потрібна подальша обробка цих величезних масивів даних. Хоча деякі з цих операцій можуть бути простими (наприклад, прості підрахунки, і т.д.), інші вимагають більш складних алгоритмів, які повинні бути спеціально розроблені для ефективної роботи на розподіленій файловій системі.

Отже, в той час як обсяг даних може рости в геометричній прогресії, можливості отримувати інформацію і діяти на основі цієї інформації, обмежені і будуть асимптотично досягати межі. Це дійсно велика проблема, пов'язана з аналізом неструктурованих даних *big data*.

Map-Reduce. При аналізі сотні терабайт або петабайт даних, не представляється можливим помістити дані в будь-яке окреме місце для аналізу.

Процес перенесення даних по каналах на окремий сервер або сервера (для паралельної обробки) займе дуже багато часу і вимагає занадто великого трафіку. Замість цього, аналітичні обчислення повинні бути виконані фізично близько до місця, де зберігаються дані.

Алгоритм *Map-Reduce* є моделлю для розподілених обчислень. Принцип його роботи полягає в наступному: відбувається розподіл вхідних даних на робочі вузли (*individual nodes*) розподіленої файлової системи для попередньої обробки (*map-крок*) і, потім, згортка (об'єднання) вже попередньо оброблених даних (reduce-крок). Таким чином, скажімо, для обчислення підсумкової суми, алгоритм буде паралельно обчислювати проміжні суми в кожному з вузлів розподіленої файлової системи, і потім підсумовувати ці проміжні значення.

Прості статистики, *Business Intelligence* (*BI*). Для складання простих звітів *BI*, існує безліч продуктів з відкритим кодом, що дозволяють обчислювати суми, середні, пропорції і т.п. за допомогою *map-re*duce.

Прогнозне моделювання, поглиблені статистики. На перший погляд може здатися, що побудова прогностичних моделей в розподілені файловій системі складніша, однак це зовсім не так. Розглянемо попередні етапи аналізу даних.

Підготовка даних. Деякий час назад StatSoft провів серію великих і успішних проектів за участю дуже великих наборів даних, що описують щохвилинні показники процесу роботи електростанції. Мета проведеного аналізу полягала в підвищенні ефективності діяльності електростанції і зниженні кількості викидів (Electric Power Research Institute, 2009). Важливо, що, незважаючи на те, що набори даних можуть бути дуже великими, інформація, що міститься в них, має значно меншу розмірність.

Наприклад, в той час як дані накопичуються щомиті або щохвилини, багато параметрів (температура газів і печей, потоки, положення заслінок і т.д.) залишаються стабільними на великих інтервалах часу. Інакше кажучи, дані, що записуються кожну секунду, є в основному повтореннями однієї і тієї ж інформації. Таким чином, необхідно проводити "розумне" агрегування даних, отримуючи для моделювання та оптимізації дані, які містять тільки необхідну інформацію про динамічні зміни, що впливають на ефективність роботи електростанції і кількість викидів.

Класифікація текстів і попередня обробка даних. Проілюструємо ще раз, як великі набори даних можуть містити набагато менше корисної інформації.

Наприклад, StatSoft брав участь в проектах, пов'язаних з аналізом текстів (*text mining*) з твітів, що відбивають, наскільки пасажири задоволені авіакомпаніями і їх послугами.

Незважаючи на те, що щогодини і щодня було вилучено велику кількість відповідних твітів, настрої, виражені в них, були досить простими і одноманітними. Більшість повідомлень – скарги і короткі повідомлення з

одного речення про "поганий досвід". Крім того, число і "сила" цих настроїв відносно стабільні в часі і в конкретних питаннях (наприклад, втрачений багаж, погане харчування, скасування рейсів).

Таким чином, скорочення фактичних твітів щодо оцінки настрою, використовуючи методи *text mining*, призводить до набагато меншого об'єму даних, які потім можуть бути легко зіставлені з існуючими структурованими даними (фактичним продажом квитків, або інформацією про часто літаючих пасажирів). Аналіз дозволяє розбити клієнтів на групи і вивчити їх характерні скарги.

Існує безліч інструментів для проведення такого агрегування даних (наприклад, оцінки настроїв) в розподіленій файловій системи, що дозволяє легко здійснювати даний аналітичний процес.

Побудова моделей. Часто завдання полягає в тому, щоб швидко побудувати точні моделі для даних, що зберігаються в розподіленій файловій системі.

Існують реалізації *map-reduce* для різних алгоритмів *data mining*/ прогностичної аналітики, придатних для масштабної паралельної обробки даних в розподіленій файловій системі.

Однак, саме через те, що оброблено дуже велику кількість даних, чи впевнені, що підсумкова модель є дійсно більш точною? Важливо, щоб методи і процедури для побудови, оновлення моделей, а також для автоматизації процесу прийняття рішень були розроблені разом з системами зберігання даних, щоб гарантувати, що такі системи є корисними і вигідними для підприємства.

3. Аналіз *Big data*

Як аналізувати неструктуровану інформацію? Як на основі big data складати прості звіти, будувати і впроваджувати поглиблені прогностичні моделі?

Насправді, швидше за все, зручніше будувати моделі для невеликих сегментів даних в розподіленій файловій системі. Як говориться в недавньому звіті Forrester: «Два плюс два дорівнює 3,9 - це зазвичай досить добре» [14].

Статистична та математична точність полягає в тому, що модель лінійної регресії, що включає, наприклад, 10 предикатів, заснованих на правильно обраній ймовірнісній вибірці з 100 000 спостережень, буде так само точною, як модель, побудована на 100 мільйонах спостережень. У ймовірнісній вибірці кожен елемент сукупності має певну, заздалегідь задану ймовірність бути обраним. Причому для кожного елемента сукупності ймовірність попадання у вибірку однакова.

На противагу цьому, деякі постачальники в області *big data*, часто заради реклами, заявляють, що "всі дані повинні бути оброблені". Насправді, точність моделі залежить від якості вибірки (кожне спостереження в популяції має мати відому ймовірність вибору) і її розмір пов'язаний зі складністю моделі. Розмір популяції не має принципового значення.

Саме з цієї причини, наприклад, вибірка, що складається всього з декількох тисяч голосів, може дозволити побудувати дуже точні прогнози реальних результатів голосування. Отже, реальна значимість *big data* в розподілених файлових системах полягає не в тому, щоб побудувати прогностичні моделі на основі всіх даних; точність моделей не буде вище.

Найбільш значним є використання всього обсягу даних для сегментації і кластеризації, що дозволить ефективно будувати велику кількість моделей для невеликих кластерів.

Наприклад, можна очікувати, що моделі, засновані на широкій сегментації (20-30 років), будуть менш точними, ніж велике число моделей, побудованих на більш детальній сегментації (наприклад, 20-21-річні студенти, які проживають в гуртожитку, і навчаються на факультеті бізнесу).

Таким чином, один із способів отримання переваг з *big data* полягає в тому, щоб використовувати доступну інформацію для побудови великої

кількості моделей для великого числа сегментів і, потім, за відповідною моделлю будувати прогнози.

У граничному випадку, кожен окремий «об'єкт» у великому сховищі даних клієнтів може мати свою власну модель для прогнозування майбутніх покупок.

Це означає, що аналітична платформа, що підтримує сховища даних, повинна бути в змозі управляти сотнями або навіть тисячами моделей, і мати можливість перенастроювати їх, коли це необхідно.

Ризики при використанні *big data*:

- Ризик переоцінки отримання можливих результатів: для отримання достовірних результатів необхідні досвідчені фахівці.
- Вартість зростає надто швидко.
- Багато джерел *big data* є приватними, доступ до даних потребує правового регулювання.

2.5 Програмне забезпечення для збору, накопичення, обробки та візуалізації *big data*.

Інтегрована платформа для бізнес-аналітики і аналізу великих даних – це особлива система. На початку необхідно зробити вибір – розробити її самостійно або придбати. Повинні врахувати існуючі системи, сценарії використання, а також рівень особистих якостей та компетентності ваших співробітників. Деякі компанії можуть обрати розробку системи повністю на базі відкритого вихідного коду, не використовуючи нічого, крім Hadoop (Hadoop Distributed File System [HDFS] і MapReduce) (http://hadoop.apache.org/releases.html#25+January%2C+2016%3A+Release+2.7.2+%28stable%29+available), Zookeeper, Solr, Sqoop, Hive, HBase, Nagios і Cacti, а іншим може знадобитися активна підтримка і створення системи з використанням IBM® InfoSphere® BigInsights ™ і IBM Netezza. Одні компанії

можуть побажати розділити структуровані і неструктуровані дані і створити шари графічних інтерфейсів призначених для користувачів різних рівнів: для звичайних користувачів, кваліфікованих користувачів і адміністраторів.

Розробка інтегрованої платформи ніколи не буває простою. Витягування, перетворення і завантаження (ETL) завжди є найтривалішим етапом в проектах по розгортанню сховищ даних. Існують різні оптимальні методики ETL, іноді вони працюють, а іноді недуже. Якщо процес ETL не працюватиме належним чином, то у вас раптово опиняться невірні і/або сумнівні дані. Ненадійність даних веде до ненадійного використання системи.

Інтегровані платформи для бізнес-аналітики і аналізу великих даних можуть зберігати неструктуровані дані з електронних листів. Вони можуть включати неповністю структуровані дані з реєстраційних журналів. Системи електронної пошти можуть бути розосереджені по різних базах даних в безлічі центрів обробки даних по всьому світу. Додайте кілька між мережевих екранів, - і раптово переміщення даних з одного місця в інше стає логічним кошмаром, що вимагає окремого проекту. Системні журнали можуть бути неформатними, напівформатними або повною плутаниною - ось і ще один проект.

Технології великих даних, такі як Apache Hadoop, передбачають переміщення системи туди, де знаходяться дані, замість того щоб переміщати дані в систему, і тому є причина. Для переміщення даних по мережах через міжмережеві екрани потрібен час. Ви втрачаєте дані, пакети, файли. Великою проблемою стає надійність.

Ключова концепція *noSQL* і *Hadoop* – це перемістити програму до даних, однак і це не так просто. Якщо у вас 100 різних систем, то треба додавати 100 примірників одного і того ж додатка в кожну систему? Хоча деякі можуть вважати, що вони довели MDM до досконалості, але цього не зробив поки ніхто. Якщо у вас одна система MDM для продуктів, ще одна для продажів і ще одна для клієнтів, і при цьому вони не інтегровані або не можуть бути легко пов'язані, то додавання додатків в кожну з систем не забезпечить їх

інтеграції або зв'язування. В результаті залишиться система з безліччю ізольованих масивів даних, які неможливо пов'язати.

Навіть якщо підприємство розгорнуло додаток для великих даних на найвищому рівні, в платформі, яка інтегрує і пов'язує всілякі види даних, можуть виникнути серйозні проблеми. Не можна просто взяти і запустити складні алгоритми на системі, з якої працюють користувачі – вона може цього не витримати. Її продуктивність може впасти. Можуть бути зіпсовані дані. Можуть виникнути проблеми з безпекою. Щоб встановити програму з високими вимогами до дискового простору, оперативної пам'яті і продуктивності може привести до відмови старої системи. Додаток може навіть не працювати належним чином на таких старих системах.

Платформа для бізнес-аналітики і аналізу великих даних повинна бути інноваційною. Це повинна бути платформа нового покоління. Необхідно використовувати технології обробки в оперативній пам'яті або конфігурувати систему для використання таких інструментів, як Hadoop і Apache Cassandra, як проміжної області, пісочниці, системи зберігання, щоб вона стала новою, більш досконалої ETL-системою. Платформа повинна інтегрувати структуровані, неструктуровані та напівструктуровані дані.

Hadoop – це набір технологій, які використовуються для зберігання та обробки величезної кількості даних. У цьому розділі розглянуто використання Hadoop для обробки даних, а не на налаштування та адміністрування. Екосистема Hadoop опрацьовує великі набори даних для підприємств та інших організацій.

Що таке Hadoop? Він складається з двох компонентів, а також часто використовується разом з іншими проектами. Які ці компоненти? Перша з них - відкрите джерело даних або HDFS, що означає файлову систему Hadoop. Друга - це API обробка, яка називається MapReduce. Найчастіше у професійних поставках Hadoop входять інші додатки або бібліотеки, і це багато, багато різних бібліотек, на теперішній час їх більше 25.

Програмні додатки, що використовується найчастіше детально розглянуто в цьому розділі – **HBase, Hive і Pig**. Окрім розуміння основних компонентів Hadoop важливо розуміти, що називається *Hadoop Distributions*. Давайте подивимось на них. Базовий дистрибутив – це 100% відкрите джерело, їх можна знайти в рамках Apache Foundation *(hadoop.apache.org),* називається Apache Hadoop, і існує багато версій.

Цикл випуску версії Hadoop досить швидкий. І тому, коли встановлюєте актуальну версію Hadoop, більшість підприємств використовують на одну-дві версії позаду випущеної на даний момент версії, оскільки вони вважають програмне забезпечення з відкритим кодом незрілим і не готовим до використання в професійному середовищі. Через це існує декілька комерційних додатків, і найчастіше вони працюють з комерційними клієнтами. Вони відрізняються від додатків з відкритим вихідним кодом – охоплюють деяку версію додатку з відкритим вихідним кодом, і вони забезпечать додаткові інструменти та моніторинг та управління разом з іншими бібліотеками.

Hadoop є проектом верхнього рівня організації *Apache Software Foundation (hadoop.apache.org)*, тому основним дистрибутивом і центральним репозиторієм для всіх напрацювань вважається саме Apache Hadoop. Однак цей же дистрибутив є основною причиною більшості перегрітих нервових кліток при знайомстві з даним інструментом: за замовчуванням установка Hadoop на кластер вимагає попередньої настройки машин, встановлення пакетів вручну, редагування безлічі файлів конфігурації та інших дій. При цьому ця документація частіше всього неповна або просто застаріла. Тому в практиці найчастіше використовуються дистрибутиви від однієї з трьох компаній: ***Cloudera*, *Hortonworks* та *MapR*** (рис. 2.8). Розглянемо всі три найбільш популярних комерційних дистрибутиви в цьому розділі. На додаток до цього, підприємства досить часто використовують кластери Hadoop у хмарі. Розподіл хмар, які найчастіше використовують, представлено веб-службами Amazon або Microsoft з Windows Azure HDInsight.

Рис. 2.8 Основні постачальники екосистеми Hadoop

Cloudera. Ключовий продукт – CDH (Cloudera Distribution, включаючи Apache Hadoop) – зв'язок найпопулярніших інструментів з інфраструктури Hadoop під керуванням *Cloudera Manager* (рис. 2.9) (https://www.cloudera.com/products/open-source/apache-hadoop.html). Менеджер бере на себе відповідальність за розгортання кластера, встановлення всіх компонентів і їх подальший моніторинг. Крім CDH компанія розвиває і інші свої продукти, наприклад *Impala* (про це нижче). Відмінною рисою Cloudera також є прагнення першими представляти на ринку обновлення, навіть якщо і в шкоду стабільності. Також, творець Hadoop – Дуг Крейтінг – працює в Cloudera.

Hortonworks. Так само, як і Cloudera, вони надають єдине рішення у вигляді HDP (Hortonworks Data Platform). Їх відмінною рисою є те, що замість розробки власних продуктів вони більше вкладають у розвиток продуктів Apache. Наприклад, замість диспетчера Cloudera вони використовують Apache Ambari, замість *Impala* – розвивають *Apache Hive*.

Коли ми говоримо про Hadoop, то в першу чергу маємо на увазі його файлову систему – HDFS (*Hadoop Distributed File System*). Найпростіший спосіб

думати про HDFS – це уявити звичайну файлову систему, тільки більшу. Звичайна файлова система, за великим рахунком, складається з таблиці файлових дескрипторів і області даних. У HDFS замість таблиці використовується спеціальний сервер – сервер імен (*NameNode*), а дані розкидані по серверам даних (*DataNode*).

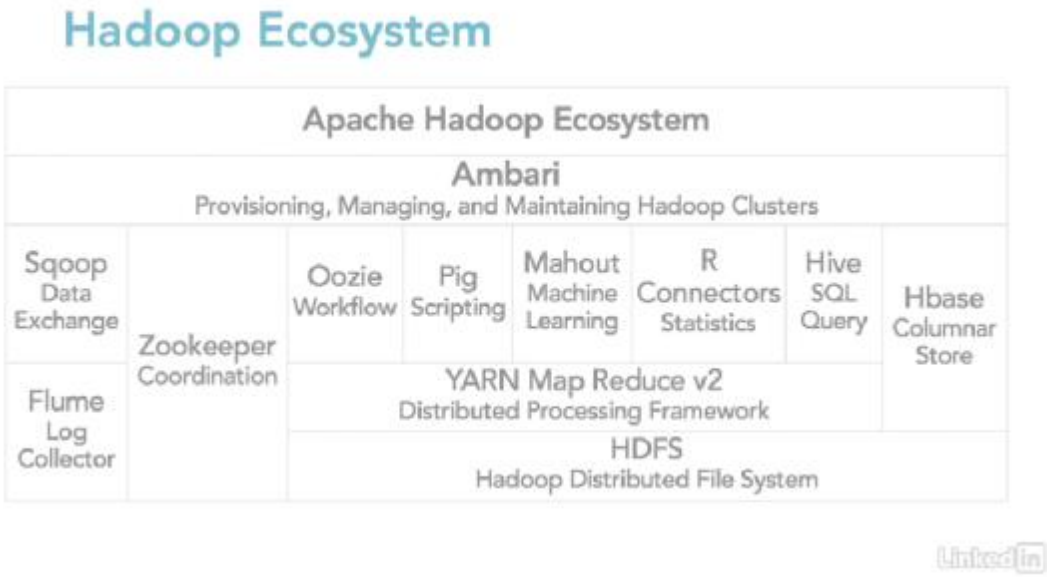


Рис. 2.9 Компоненти екосистеми *Hadoop* від *Cloudera*

В іншому відмінностей не так багато: дані розбиті на блоки (зазвичай по 64Мб або 128Мб), для кожного файлу сервер імен зберігає його шлях, список блоків і їх реплік. HDFS має класичну unix-івську деревоподібну структуру директорій, користувачів з кодонів прав, і навіть схожий набір консольних команд:

```
# просмотреть корневую директорию: локально и на HDFS
 ls /
 hadoop fs -ls /
# оценить размер директории
du -sh mydata
hadoop fs -du -s -h mydata
# вывести на экран содержимое всех файлов в директории
cat mydata/*
hadoop fs -cat mydata/*
```

Чому HDFS так цікава? По-перше, тому що вона надійна: якось при перестановці обладнання відділ IT випадково знищив 50% серверів, при цьому безповоротно було втрачено всього 3% даних. А по-друге, що навіть більш важливо, сервер імен розкриває для всіх бажаючих розташування блоків даних на машинах. Чому це важливо, буде пояснено далі.

Доступ до інформації здійснюється за допомогою так званих двигунів, що дозволяють переміщувати дані або програми для їх обробки. Найбільш широко використовують двигуни**: *MapReduce, Spark, Tez*.**

При правильній архітектурі додатків, інформація про те, на яких машинах розташовані блоки даних, дозволяє запустити на них же обчислювальні процеси (називається англіцизмом «Воркер») і *виконати більшу частину обчислень локально, тобто без передачі даних по мережі*. Саме ця ідея лежить в основі парадигми **MapReduce** і її конкретної реалізації в Hadoop.

Класична конфігурація кластера Hadoop складається з одного сервера імен, одного майстра MapReduce (т.зв. *JobTracker*) і набору робочих машин, на кожній з яких одночасно крутиться сервер даних (*DataNode*) і Воркер (*TaskTracker*). Кожна MapReduce робота складається з двох фаз (рис. 2.10):

1. ***map*** – виконується паралельно і (по можливості) локально над кожним блоком даних. Замість того, щоб доставляти терабайти даних до програми, невелика, визначена користувачем програма копіюється на сервера з даними і робить з ними все, що не вимагає перемішування і переміщення даних (shuffle).

2. ***reduce*** – доповнює **map** агрегуючими операціями.

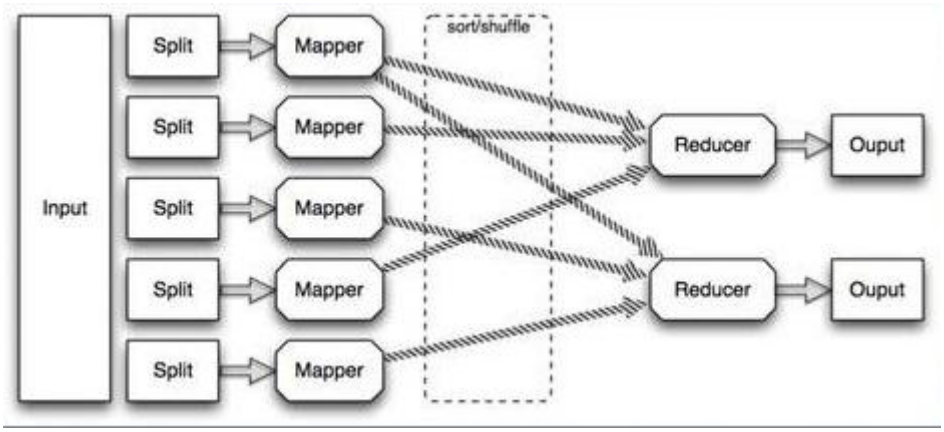


Рис. 2.10 Схема роботи *MapReduce*

Насправді між цими фазами є ще фаза ***combine***, яка робить те ж саме, що і ***reduce***, але над локальними блоками даних. Наприклад, уявімо, що у нас є 5 терабайт логів поштового сервера, які потрібно розібрати і витягти повідомлення про помилки. Рядки незалежні один від одного, тому їх розбір можна перекласти на завдачу ***map***. Далі за допомогою ***combine*** можна відфільтрувати рядки з повідомленням про помилку на рівні одного сервера, а потім за допомогою ***reduce*** зробити те ж саме на рівні всіх даних. Все, що можна було розпаралелити, ми розпаралелили, і крім того мінімізували передачу даних між серверами. І навіть якщо якась задача з якоїсь причини «впаде», Hadoop автоматично перезапустить її, піднявши з диска проміжні результати.

MapReduce працює наступним чином:

map (f, c) – приймає функцію *f* і колекцію *c*; повертає колекцію, створену шляхом застосування функції *f* до кожного елементу колекції *c*

reduce (f, c) – приймає функцію *f* і колекцію *c*; повертає об'єкт (в загальному випадку - складний об'єкт), утворений через згортку колекції *c* функцією *f*.

Для *map* – вхідна колекція складається з пар (*key, value*), причому і *key*, і *value* можуть бути скільки завгодно складним об'єктом; на виході кожен елемент вхідної колекції породжує довільно кількість вихідних пар (*key, value*) – причому формат вихідних *key* і *value* може абсолютно ніяк не залежати від формату вхідних даних, між *map* і *reduce* відбуватиметься сортування, угруповання і розподіл.

Для *reduce* – вхідна колекція буде являти собою зріз всього потоку даних з *map*, побудований якимось чином – як правило, з єдиним значенням *key*; функція застосовується до елементів такої колекції, колекція, знову ж таки, складається з (*key, value*) пар, на виході може бути будь-яку кількість агрегатів, знову ж таки, в довільному форматі (*key, value*).

Проблема в тому, що більшість реальних завдань набагато складніше одного циклу роботи MapReduce. У більшості випадків необхідно робити паралельні операції, потім послідовні, потім знову паралельні, потім комбінувати кілька джерел даних і знову робити паралельні і послідовні операції. Стандартний MapReduce спроектований так, що всі результати - як кінцеві, так і проміжні – записуються на диск. В результаті час зчитування і запису на диск, помножене на кількість разів, яку воно робиться при вирішенні завдання, часто в кілька (та що там в кілька, до 100 разів!) Перевищує час самих обчислень.

І тут з'являється Spark використовує ідею локальності даних, проте виносить більшість обчислень в пам'ять замість диска. Ключовим поняттям в Spark є *RDD (resilient distributed dataset)* – покажчик на розподілену колекцію даних. Більшість операцій над RDD не приводить до яких-небудь обчислень, а лише створює чергову обгортку, обіцяючи виконати операції тільки тоді, коли вони знадобляться. Втім, це простіше показати, ніж розказати. Нижче наведено

скрипт на Python (Spark з коробки підтримує інтерфейси для Scala, Java і Python) для вирішення задачі про логи:

```
sc = ...                          # создаём контекст (SparkContext)
rdd = sc.textFile("/path/to/server_logs") # создаём указатель на данные
rdd.map(parse_line) \             # разбираем строки и переводим их в удобный формат
       .filter(contains_error) \ # фильтруем записи без ошибок
       .saveAsTextFile("/path/to/result") # сохраняем результаты на диск
```

У цьому прикладі реальні обчислення починаються лише на останній сходинці: Spark бачить, що потрібно матеріалізувати результати, і для цього починає застосовувати операції до даних. При цьому тут немає ніяких проміжних стадій – кожен рядок піднімається в пам'ять, розбирається, перевіряється на ознака помилки в повідомленні і, якщо така ознака є, тут же записується на диск.

Така модель виявилася настільки ефективною і зручною, що проекти з екосистеми Hadoop почали один за іншим переводити свої обчислення на Spark, а над самим двигуном зараз працює більше людей, ніж над морально застарілим MapReduce.

Але не Spark-ом єдиним. Компанія Hortonworks вирішила зробити акцент на альтернативний движок – **Tez**. Tez представляє задачу у вигляді спрямованого ациклічного графа (DAG) компонентів-обробників. Планувальник запускає обчислення графа і при необхідності динамічно переконфігуровує його, оптимізуючи під дані. Це дуже природна модель для виконання складних запитів до даних, таких як SQL-подібні скрипти в **Hive**, куди Tez приніс прискорення до 100 разів. Втім, крім Hive цей движок поки мало де використовується, тому сказати, наскільки він придатний для більш простих і поширених завдань, досить складно.

Нижче наведено опис основних компонентів екосистеми Hadoop (рис. 2.10), що підтримують SQL запити: **Hive, Impala, Shark, Spark SQL, Drill.**

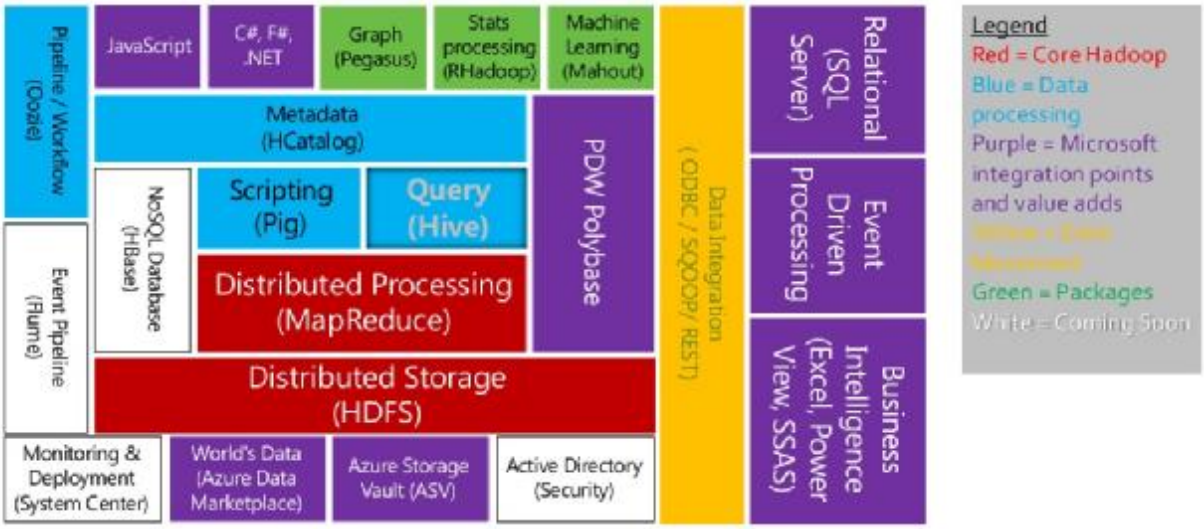


Рис 2.10 Компоненти екосистеми Hadoop

Незважаючи на те, що Hadoop є повноцінною платформою для розробки будь-яких додатків, найчастіше він використовується в контексті зберігання даних і конкретно SQL рішень. Власне, в цьому немає нічого дивного: великі обсяги даних майже завжди означають аналітику, а аналітику набагато простіше робити над табличними даними. До того ж, для SQL баз даних набагато простіше знайти і інструменти, і людей, ніж для NoSQL рішень. В інфраструктурі Hadoop-а є кілька SQL-орієнтованих інструментів:

Hive – найперша і до сих пір одна з найпопулярніших СУБД на цій платформі. В якості мови запитів використовує **HiveQL** – урізаний діалект SQL, який, тим не менш, дозволяє виконувати досить складні запити над даними, збереженими в HDFS. Тут треба провести чітку лінію між версіями Hive <= 0.12 і поточною версією 0.13: як вже було зазначено, в останній версії Hive переключився *з класичного MapReduce на новий движок Tez*, багаторазово прискоривши його і зробивши придатним для інтерактивної аналітики. Тобто тепер вам не треба чекати 2 хвилини, щоб порахувати кількість записів в одній

невеликій партії або 40 хвилин, щоб згрупувати дані по днях за тиждень. Крім того, як Hortonworks, так і Cloudera надають ODBC-драйвер, дозволяючи підключити до Hive такі інструменти як Tableau, Micro Strategy і навіть Microsoft Excel.

Impala – продукт компанії Cloudera і основний конкурент Hive. На відміну від останнього, Impala ніколи не використовувала класичний MapReduce, а спочатку виконувала запити на своєму власному движку (написаному, до речі, на нестандартному для Hadoop-а C++). Крім того, останнім часом Impala активно використовує кешування часто використовуваних блоків даних і стовпчик формати зберігання, що дуже добре позначається на продуктивності аналітичних запитів. Так само, як і для Hive, Cloudera пропонує до свого дітища цілком ефективний ODBC-драйвер.

Shark. Коли в екосистему Hadoop увійшов Spark з його революційними ідеями, природним бажанням було отримати SQL-движок на його основі. Це вилилося в проект під назвою Shark, створений ентузіастами. Однак у версії Spark 1.0 команда Spark-а випустила першу версію свого власного SQL-движка - Spark SQL; з цього моменту Shark вважається зупиненим.

Spark SQL – нова гілка розвитку SQL на базі Spark. Чесно кажучи, порівнювати його з попередніми інструментами не зовсім коректно: в Spark SQL немає окремої консолі і свого сховища метаданих, SQL-парсер поки досить слабкий, а партіції, судячи з усього, зовсім не підтримуються. По всій видимості, на даний момент його основна мета – вміти читати дані зі складних форматів (таких як Parquet, див. нижче) і висловлювати логіку у вигляді моделей даних, а не програмного коду. І, чесно кажучи, це не так і мало! Дуже часто конвеєр обробки складається з чергування SQL-запитів і програмного коду; Spark SQL дозволяє безболісно зв'язати ці стадії.

Hive on Spark – є і таке, але, запрацює не раніше версії 0.14.

Drill. Для повноти картини потрібно згадати і Apache Drill. Цей проект поки перебуває в інкубаторі ASF і мало поширений, але судячи з усього, основний упор в ньому буде зроблений на напівструктуровані і вкладені дані. У

Hive і Impala також можна працювати з JSON-рядками, проте продуктивність запиту при цьому значно падає (часто до 10-20 разів). До чого призведе створення ще однієї СУБД на базі Hadoop, сказати складно, але давайте почекаємо і подивимося [15].

NoSQL: HBase

ZooKeeper – головний інструмент координації для всіх елементів інфраструктури Hadoop. Найчастіше використовується як сервіс конфігурації, хоча його можливості набагато ширше. Простий, зручний, надійний.

Hue – веб-інтерфейс до сервісів Hadoop, частина Cloudera Manager. Працює погано, з помилками і за настроєм. Придатний для показу нетехнічним фахівцям, але для серйозної роботи краще використовувати консольні аналоги**.**

Flume – сервіс для організації потоків даних. Наприклад, можна налаштувати його для отримання повідомлень з syslog, агрегації і автоматичного скидання в директорію на HDFS. На жаль, вимагає дуже багато ручної конфігурації потоків і постійного розширення власними Java класами.

Sqoop – утиліта для швидкого копіювання даних між Hadoop і RDBMS. Швидкого в теорії. На практиці Sqoop 1 виявився, по суті, однопоточним і повільним, а Sqoop 2 на момент останнього тесту просто не заробив.

Oozie – планувальник потоків завдань. Спочатку спроектований для об'єднання окремих MapReduce робіт в єдиний конвеєр і запуску їх за розкладом. Додатково може виконувати Hive, Java і консольні дії, але в контексті Spark, Impala і ін., Цей список виглядає досить марним. Дуже крихкий, заплутаний і практично не піддається налагодженні.

Azkaban – цілком придатна заміна Oozie. Є частиною Hadoop-інфраструктури компанії LinkedIn. Підтримує декілька типів дій, головне з яких – консольна команда (а що ще треба), запуск за розкладом, логи додатків, оповіщення про що впали роботах і ін. З мінусів – деяка вогкуватість і не завжди зрозумілий інтерфейс (спробуйте здогадатися, що роботу потрібно не створювати через UI, а заливати у вигляді zip-архіву з текстовими файлами).

Зберігання даних є найважливішим фактором, а також вимагає використання різних технологій. В системі Hadoop є HBase. Однак деякі компанії використовують Cassandra, Neo4j, Netezza, HDFS і інші технології, в залежності від потреб. HDFS – це система файлового зберігання. HBase – це стовпчикова база даних, подібна до Cassandra. Багато компаній використовують Cassandra для аналізу, більш наближеного до реального часу. Однак HBase вдосконалюється [16, 17].

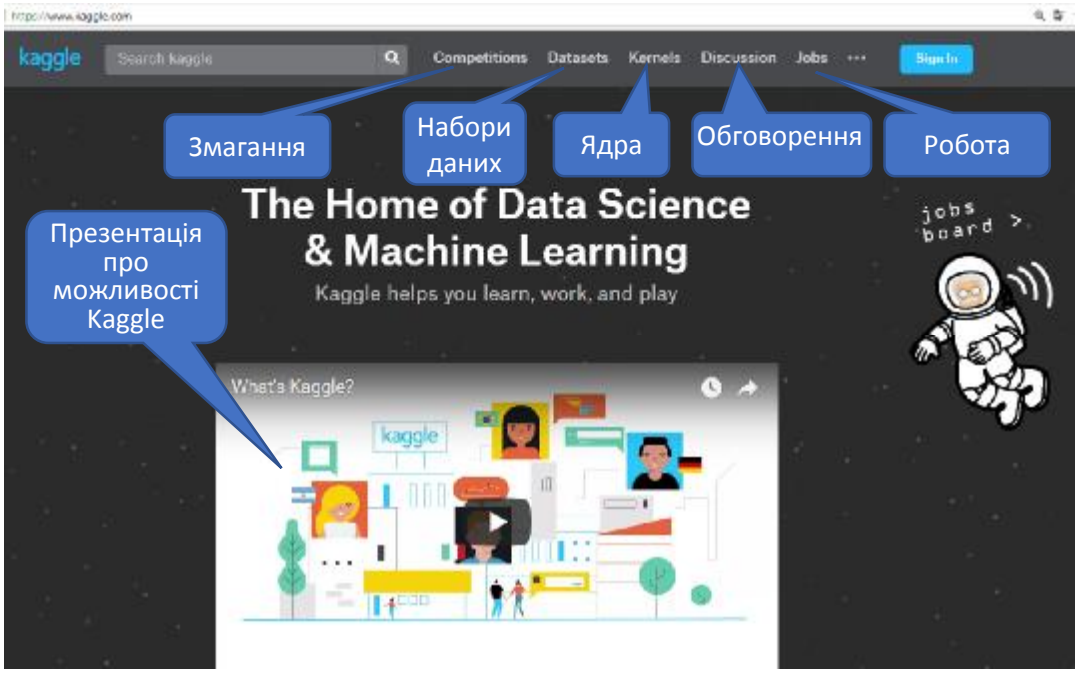
2.7 Застосування аналізу великих даних на прикладах Kaggle

Kaggle – це платформа для дослідників різних рівнів, де вони можуть випробувати свої моделі аналізу даних на серйозних і актуальних завданнях (www.kaggle.com). Суть такого ресурсу – не тільки в можливості отримати непоганий грошовий приз у разі, якщо саме ваша модель виявиться кращою, але і в тому (а, це, мабуть, набагато важливіше), щоб набратися досвіду і стати фахівцем в області аналізу даних і машинного навчання. Адже найважливіше питання, часто стоїть перед такого роду фахівцями – де знайти реальні завдання? Тут їх досить.

Kaggle – платформа для проведення конкурсів по машинному навчанню і обробці даних. Велика компанія публікує якісь свої дані і завдання – наприклад, по анонімним даним про користувачів та їхніх діях треба передбачити відтік покупців, зміну динаміки продажів, а тисячі фахівців з усього світу намагаються побудувати кращу інтелектуальну модель. Платформа *Kaggle* автоматично перевіряє якість запропонованих моделей, застосовуючи їх до неопублікованої частини вихідних даних, а в призначений день зупиняє прийом варіантів, і лідер перетворюється в переможця.

Грошову складову призу призначає компанія – автор завдання і це абсолютно довільна сума без будь-яких обмежень і правил, відмінних від "чим більше, тим краще". Рекордне змагання йде прямо зараз – за поліпшення

алгоритму діагностики раку пропонують мільйон доларів, але це абсолютно нетипова сума. Конкурсів на сто тисяч або більше за історію сервісу було трохи більше двох десятків, а медіанний розмір призу – порядку 10 тисяч доларів.



Але не грошима єдиними, Kaggle це не тільки заробіток, це ще і рекрутинг. Частина конкурсів без всякого маскування розігрують квиток на співбесіду в провідні компанії – і учасників в середньому більше, ніж в боротьбі за типові $ 10 000. Це, до речі, абсолютно раціонально – прибавка в зарплаті дуже швидко разовий кеш обжене, навіть якщо інші переваги нової роботи не враховувати. Є й просто розділ вакансій, куди можна написати що-небудь запальне, заплативши суму близько $ 1000 за одне оголошення. Однак така прямота – тільки вершина айсберга, результати конкурсів відкриті і успішний профіль на Kaggle – шикарний пункт в резюме, причому такий прозорий, що навіть самому простому *hr*-менеджеру можна пояснити, як поганого відрізнити від хорошого, а хороший від провідного.

За весь час існування сервісу проведено трохи більше двохсот конкурсів, розіграно мільйонів п'ять доларів (інвестицій в проект було в три рази більше), зареєстровано понад 800 000 «датасайтистів». Реальна кількість справжніх фахівців оцінюється тисяч в п'ятнадцять-двадцять, якщо фахівцем вважати

того, хто може побудувати модель, що виграє у Random, і готовий вкласти в конкурси необхідний для такої моделі час. Крім власне змагань у Kaggle є величезний і популярний форум, серед своїх є що обговорити, а за хороші топіки можна отримати медальку в профіль, не так круто як за змагання, але для компанії другого ряду зійде.

Загалом, Kaggle вирішив заробити, продавши базу фахівців цілком найбільшому роботодавцю, і віддався Google в березні 2017 року [18].

Розглянемо приклад змагання: «Аналіз ринку корзини Instacart». Набір даних для цього конкурсу – реляційний набір файлів, що описують замовлення клієнтів з часом. Мета конкурсу – передбачити, які продукти будуть в наступному замовленні користувача. Набір даних анонімний та містить зразок з більш ніж 3 мільйонів замовлень продуктів харчування більш ніж 200 000 користувачів Instacart. Для кожного користувача надано від 4 до 100 замовлень, з послідовністю продуктів, придбаних у кожному замовленні, також надано тиждень і годину дня замовлення, а також відносний показник часу між замовленнями.

Опис вихідних даних:

Файл **aisles.csv** містить інформацію:

aisle_id – ідентифікатор групи товарів;

aisle – група товарів;

```
aisle_id, aisle
1, prepared soups salads
2, specialty cheeses
3, energy granola bars
...
```

Файл **departments.csv** містить інформацію:

department_id – ідентифікатор відділу магазину;

department – відділ магазину;

```
department_id, department
1, frozen
2, other
3,bakery
...
```

order_products__*.csv

Ці файли вказують, які продукти були придбані в кожному замовленні. Order_products__prior.csv містить попередній зміст замовлення для всіх клієнтів. "*Reorder*" означає, що клієнт має попереднє замовлення, яке містить даний продукт. Зауважте, що деякі замовлення не матимуть змінених продуктів.

order_*id* – ідентифікатор замовлення;

product_id – ідентифікатор товару;

add_to_cart_order – порядок в замовленні;

reordered – повторне замовлення товару;

```
order_id, product_id, add_to_cart_order, reordered
1, 49302, 1, 1
1, 11109, 2, 1
1, 10246, 3, 0
...
```

orders.csv – цей файл повідомляє, до якого набору даних (попереднього (*prior*), тренувального (*train*), тестового (*test*)) належить замовлення. Ви прогнозуєте повторно впорядковані елементи лише для замовлень тестового набору. "Order_dow" - це день тижня.

order_id – ідентифікатор замовлення;

user_id – ідентифікатор споживача;

eval_set – тип набору даних;

order_number – номер замовлення;

order_dow – день тижня;

order_hour_of_day – час тижня;

days_since_prior_order – днів з останнього замовлення.

order_id, user_id, eval_set, order_number, order_dow, order_hour_of_day, days_since_prior_order

2539329, 1, prior, 1, 2, 08,

2398795, 1, prior, 2, 3, 07, 15.0

473747, 1, prior, 3, 3, 12, 21.0

...

products.csv – база даних товарів.

product_id – ідентифікатор товару;

product_name – назва товару

aisle_id – ідентифікатор групи товарів;

department_id – ідентифікатор відділу магазину.

product_id, product_name, aisle_id, department_id

1, Chocolate Sandwich Cookies, 61, 19

2, All-Seasons Salt, 104, 13

3, Robust Golden Unsweetened Oolong Tea, 94, 7

...

sample_submission.csv

order_id – ідентифікатор замовлення;

products – товари.

order_id, products

17, 39276

34, 39276

137, 39276

...

Нижче наведено скрипти мовою *Python,* які спочатку досліджують основні характеристики, зменшують розмірність набору даних Instacart. Як зазначено в описі, набір даних анонімний та містить зразок з більш ніж 3

мільйонів замовлень продуктів харчування з більш ніж 200 000 користувачів Instacart. Мета полягає в тому, щоб передбачити, які раніше придбані продукти будуть в наступному замовленні користувача.

Файл подання результатів. Для кожного замовлення *order_id* в тестовому наборі даних слід передбачити список продуктів-замовлень *product_ids*, розділених пробілом, для цього замовлення. Якщо хочете передбачити порожнє замовлення, вам слід надіслати явне значення "*None*". Ви можете поєднати "*None*" з *product_ids*. Запис правопису " *None* " є чутливим до регістру в метриці оцінок. Файл повинен мати заголовок і виглядати наступним чином:

order_id,products

17, 1 2

34, None

137, 1 2 3

Для реалізації поставленого завдання вводимо набір команд на мові програмування *Python*:

In [1]:

```
#імпорт необхідних бібліотек
import numpy as np # лінійної алгебри
import pandas as pd # обробки даних, CSV file I/O (e.g. pd.read_csv)
%matplotlib inline
import matplotlib.pyplot as plt  # Matlab-style графіків
import seaborn as sns
color = sns.color_palette()
import warnings
warnings.filterwarnings('ignore') # Стримування непотрібних попереджень
# для читабельності та чистоти презентації

pd.set_option('display.float_format', lambda x: '%.3f' % x) #Обмеження виводу дробових
                                                            # чисел 3 цифрами після крапки

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))#перевірка наявності #файлів в директорії
```

Результат:

aisles.csv

departments.csv

order_products__prior.csv

order_products__train.csv

orders.csv

products.csv

sample_submission.csv

In [2]:

```
#Завантажимо дані в pandas dataframe

order_products_train = pd.read_csv('../input/order_products__train.csv')
order_products_prior = pd.read_csv('../input/order_products__prior.csv')
orders = pd.read_csv('../input/orders.csv')
products = pd.read_csv('../input/products.csv')
aisles = pd.read_csv('../input/aisles.csv')
departments = pd.read_csv('../input/departments.csv')
```

По-перше, давайте розглянемо файли *order_products_train* та *order_products_prior*. Ці файли містять інформацію, які продукти були придбані в кожному замовленні. Більш конкретно, *order_products__prior* містить попереднє зміст замовлення для всіх клієнтів і *order_products_train* містить останні замовлення лише для деяких клієнтів.

In [3]:

```
print("The order_products_train size is : ", order_products_train.shape)
print("The order_products_prior size is : ", order_products_prior.shape)
```

Результат:

The order_products_train size is : (1384617, 4)

The order_products_prior size is : (32434489, 4)

In [4]:

```
#Виведення на екран перших п'яти рядків набору train.
order_products_train.head(5)
```

Out [4]:

	order_id	product_id	add_to_cart_order	reordered

0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0
3	1	49683	4	0
4	1	43633	5	1

In [5]:

```
# Виведення на екран перших п'яти рядків набору prior.
order_products_prior.head(5)
```

	order_id	product_id	add_to_cart_order	reordered
0	2	33120	1	1
1	2	28985	2	1
2	2	9327	3	0
3	2	45918	4	1
4	2	30035	5	0

Далі об'єднаємо в єдину order_products базу даних.

In [6]:

```
order_products_all = pd.concat([order_products_train, order_products_prior], axis=0)
print("The order_products_all size is : ", order_products_all.shape)
```

Результат:

The order_products_all size is : (33819106, 4)

In [7]:

```
# Виведення на екран перших п'яти рядків об'єднаного набору.
order_products_all.head(5)
```

Результат:

Out [7]:

	order_id	product_id	add_to_cart_order	reordered
0	1	49302	1	1
1	1	11109	2	1
2	1	10246	3	0

3	1	49683	4	0
4	1	43633	5	1

Перевірка на пропущені дані:

In [8]:

```
total = order_products_all.isnull().sum().sort_values(ascending=False)
percent = (order_products_all.isnull().sum()/order_products_all.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total Missing', 'Percent'])
missing_data
```

Out [8]:

	Total Missing	Percent
reordered	0	0.000
add_to_cart_order	0	0.000
product_id	0	0.000
order_id	0	0.000

Тобто немає пропущених даних в *order_products_all* наборі даних.

In [9]:

```
# Перевірка кількості унікальних замовлень та унікальних товарів
orders_Unique = len(set(order_products_all.order_id))
products_Unique = len(set(order_products_all.product_id))
print("There are %s orders for %s products" %(orders_Unique, products_Unique))
```

Результат:

There are 3346083 orders for 49685 products

In [10]:

```
grouped = order_products_all.groupby("order_id")["add_to_cart_order"].aggregate("max").reset_index()
grouped = grouped.add_to_cart_order.value_counts()

sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(15, 12))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)

plt.ylabel('Number of Orders', fontsize=13)
plt.xlabel('Number of products added in order', fontsize=13)
plt.show()
```

Графік визначення кількості товарів, що клієнти звичайно замовляють:

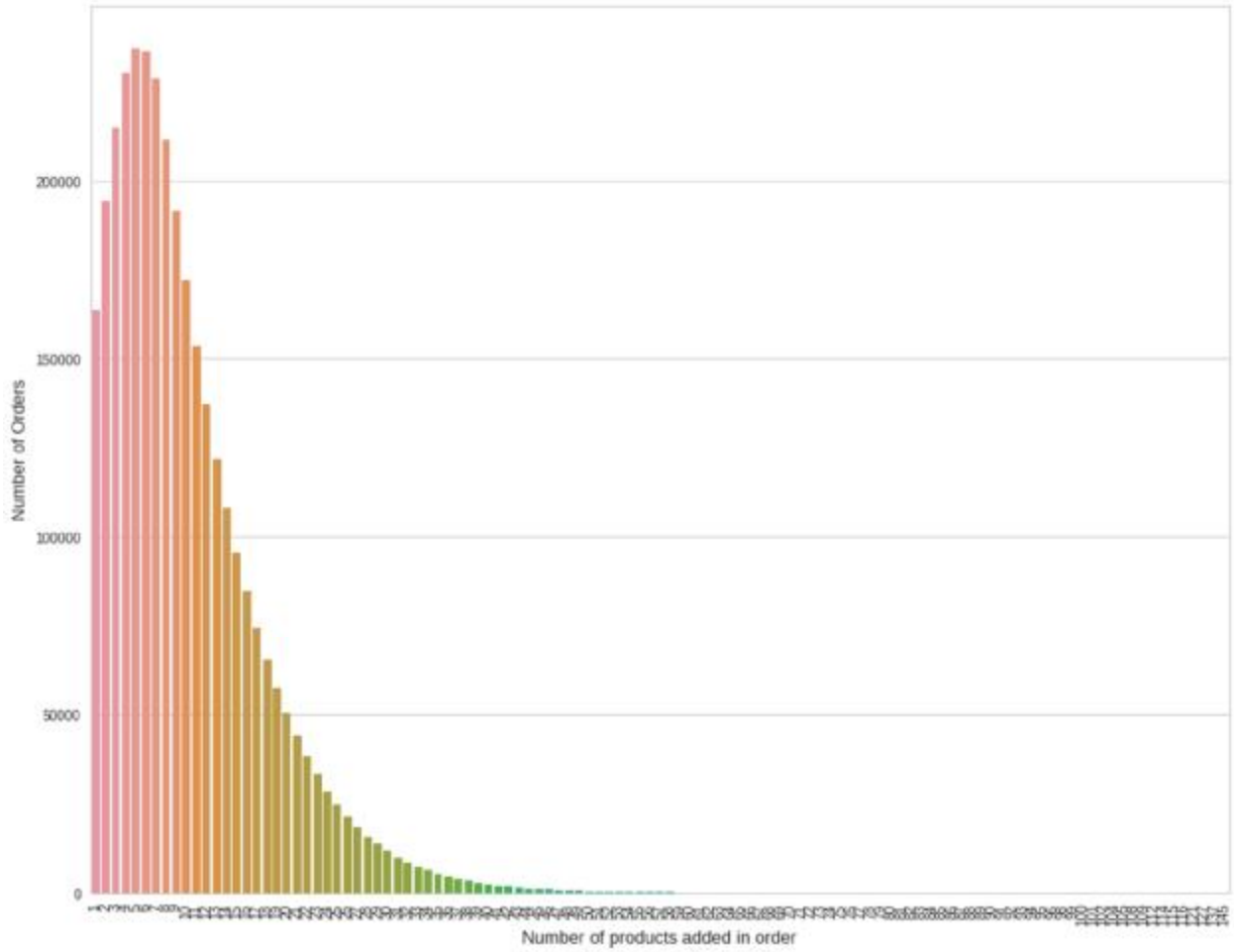


Рис. 2.11 Ранжування товарів за популярністю.

Як видно з рис 2.11 найбільше користуються попитом близько 5 товарів.

Визначення товарів, що найбільше користуються попитом.

In [11]:

	product_id	Total_reorders	product_name
24849	24852	491291	Banana
13173	13176	394930	Bag of Organic Bananas
21134	21137	275577	Organic Strawberries
21900	21903	251705	Organic Baby Spinach
47205	47209	220877	Organic Hass Avocado
47762	47766	184224	Organic Avocado
47622	47626	160792	Large Lemon
16794	16797	149445	Strawberries
26206	26209	146660	Limes

27842	27845	142813	Organic Whole Milk

Фрукти, такі як банан, полуниця найбільше користуються попитом.

In [12]:

```
grouped  = grouped.groupby(['product_name']).sum()['Total_reorders'].sort_values(ascending=False)


sns.set_style('darkgrid')
f, ax = plt.subplots(figsize=(12, 10))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of Reorders', fontsize=13)
plt.xlabel('Most ordered Products', fontsize=13)
plt.show()
```

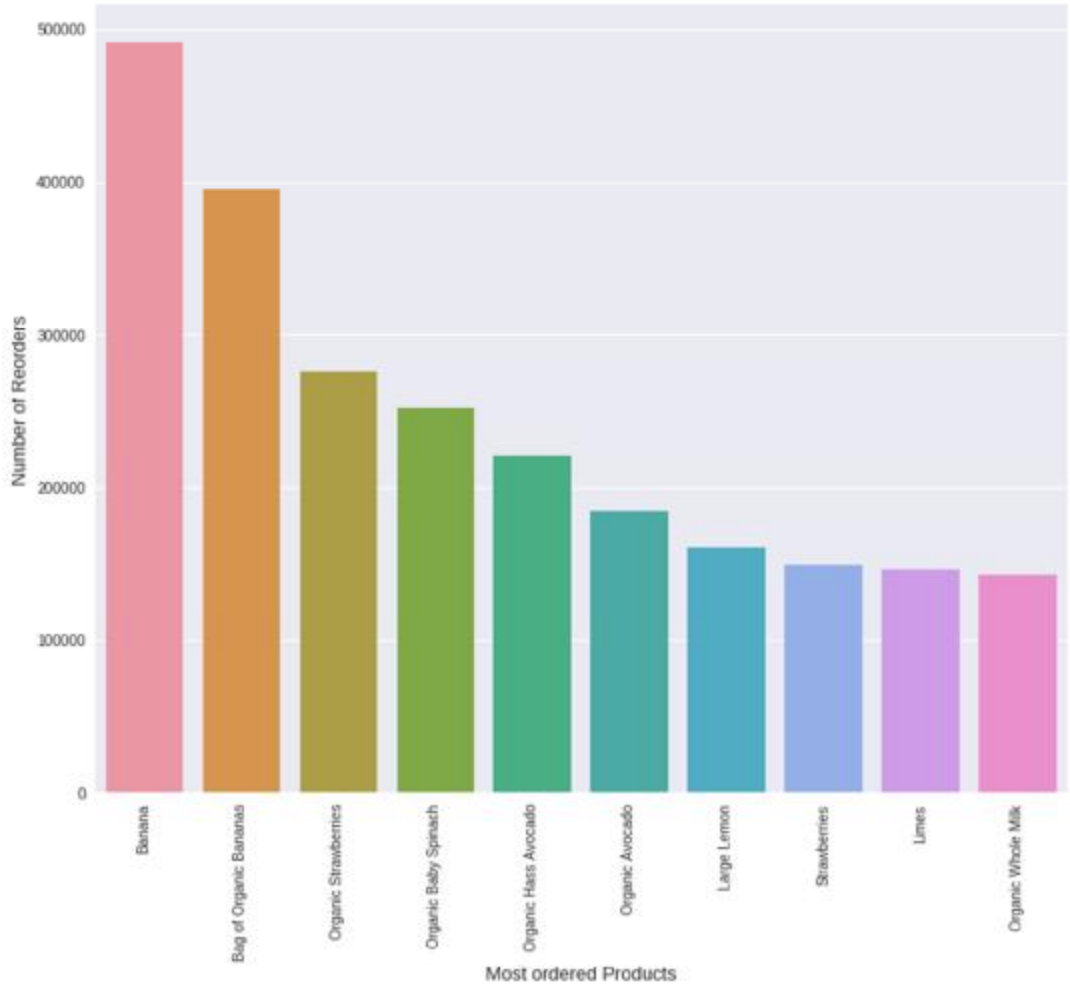


Рис. 2.12 Продукти, що найчастіше та найбільше за обсягом замовляють клієнти

На рис. 2.12 наведено продукти, що найчастіше та найбільше за обсягом замовляють клієнти.

Повторюваність замовлень:

Чи замовляють клієнти повторно продукти, які вже замовляли попередньо ?

In [13]:

```
grouped = order_products_all.groupby("reordered")["product_id"].aggregate({'Total_products':
'count'}).reset_index()
grouped['Ratios'] = grouped["Total_products"].apply(lambda x: x /grouped['Total_products'].sum())
grouped
```

Out [13]:

	reordered	Total_products	Ratios
0	0	13863746	0.410
1	1	19955360	0.590

Тобто 59% замовлених продуктів вже були попередньо замовлені клієнтами.

In [14]:

```
grouped  = grouped.groupby(['reordered']).sum()['Total_products'].sort_values(ascending=False)


sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(5, 8))
sns.barplot(grouped.index, grouped.values, palette='RdBu_r')
plt.ylabel('Number of Products', fontsize=13)
plt.xlabel('Reordered or Not Reordered', fontsize=13)
plt.ticklabel_format(style='plain', axis='y')
plt.show()
```

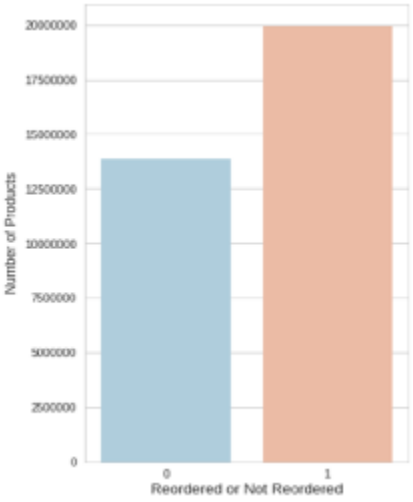


Рис. 2.13 Співвідношення повторно замовлених та повторно незамовлених товарів

Визначимо товари, що найчастіше повторно замовляють.

In [15]:

```
grouped = order_products_all.groupby("product_id")["reordered"].aggregate({'reorder_sum': sum,'reorder_total': 'count'}).reset_index()
grouped['reorder_probability'] = grouped['reorder_sum'] / grouped['reorder_total']
grouped = pd.merge(grouped, products[['product_id', 'product_name']], how='left', on=['product_id'])
grouped = grouped[grouped.reorder_total > 75].sort_values(['reorder_probability'], ascending=False)[:10]
grouped
```

Out [15]:

	product_id	reorder_sum	reorder_total	reorder_ probability	product_name
2074	2075	84	90	0.933	Serenity Ultimate Extrema Overnight Pads
27737	27740	94	102	0.922	Chocolate Love Bar
35601	35604	93	104	0.894	Maca Buttercups
38248	38251	99	111	0.892	Benchbreak Chardonnay
36798	36801	88	99	0.889	Organic Blueberry B Mega
10233	10236	114	131	0.870	Fragrance Free Clay with Natural Odor

					Eliminat...
20595	20598	99	114	0.868	Thousand Island Salad Snax
455	5457	78	90	0.867	Classic Carbonated Natural Mineral Water
35493	35496	394	457	0.862	Real2 Alkalized Water 500 ml
9289	9292	2580	2995	0.861	Half And Half Ultra Pasteurized

In [16]:

```
grouped  = grouped.groupby(['product_name']).sum()['reorder_probability'].sort_values(ascending=False)


sns.set_style('darkgrid')
f, ax = plt.subplots(figsize=(12, 10))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylim([0.85,0.95])
plt.ylabel('Reorder probability', fontsize=13)
plt.xlabel('Most reordered products', fontsize=12)
plt.show()
```

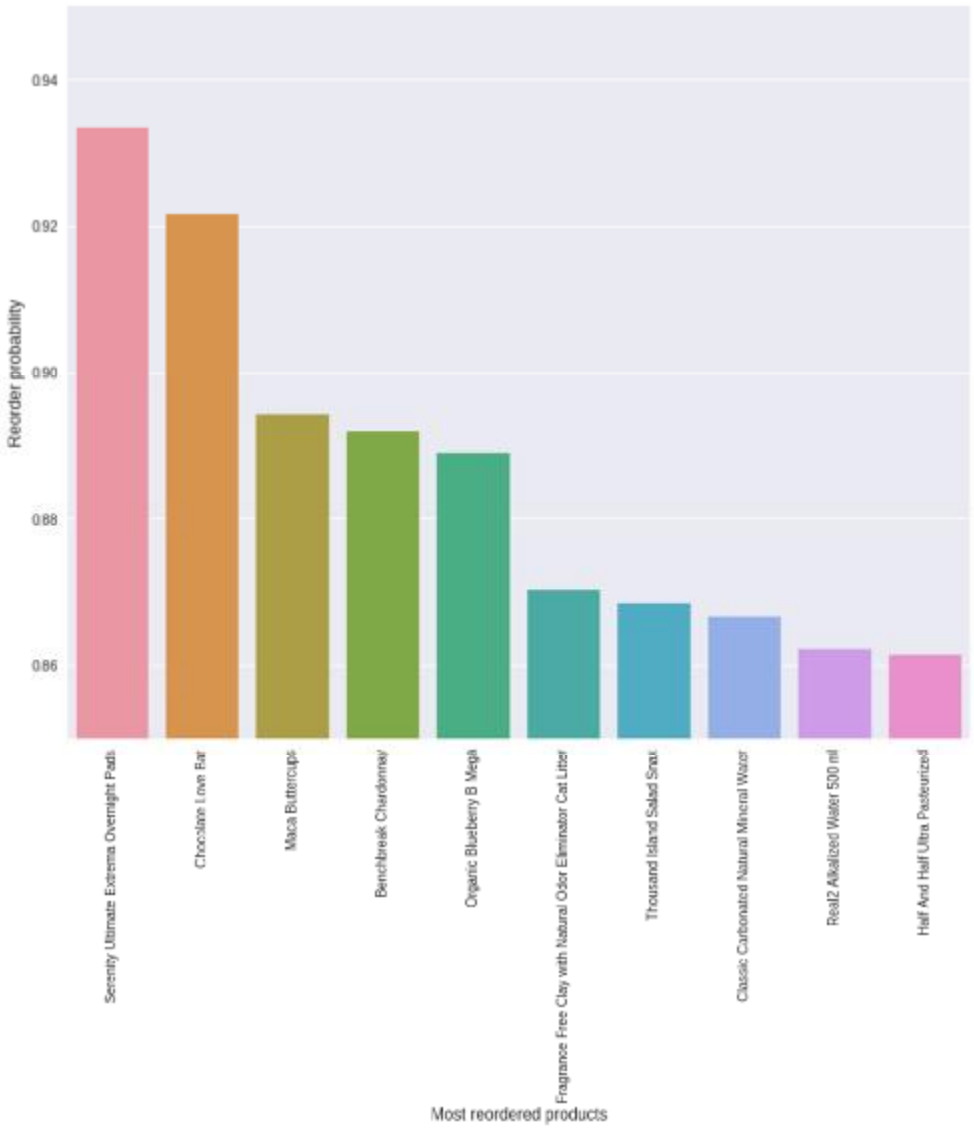


Рис. 2.14 Товари, що найчастіше повторно замовляють

In [17]:

```
print("The orders data size is : ", orders.shape)
print("Columns in orders data  are : ",  orders.columns.values)
```

Результат:

```
The orders data size is :  (3421083, 7)
Columns in orders data  are :  ['order_id' 'user_id' 'eval_set' 'order_number' 'order_dow'
 'order_hour_of_day' 'days_since_prior_order']
```

In [18]:

```
#display first five rows of our dataset.


orders.head(5)
```

Out [18]:

	order_id	user_id	eval_set	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	2539329	1	prior	1	2	8	nan
1	2398795	1	prior	2	3	7	15.000
2	473747	1	prior	3	3	12	21.000
3	2254736	1	prior	4	4	7	29.000

Пропущені дані:

In [19]:

```
orders_na = (orders.isnull().sum() / len(orders)) * 100
orders_na = orders_na.drop(orders_na[orders_na == 0].index).sort_values(ascending=False)
orders_na
```

Out [19]:

```
days_since_prior_order   6.028
dtype: float64
```

В базах вихідних даних лише один атрибут з пропущеними даними *days_since_prior_order* , що містить 6.028% пропущених даних.

Визначимо час замовлень:

Визначимо час, коли клієнти роблять замовлення.

In [20]:

```
grouped = orders.groupby("order_id")["order_hour_of_day"].aggregate("sum").reset_index()
grouped = grouped.order_hour_of_day.value_counts()


sns.set_style('darkgrid')
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of orders', fontsize=13)
plt.xlabel('Hours of order in a day', fontsize=13)
plt.show()
```

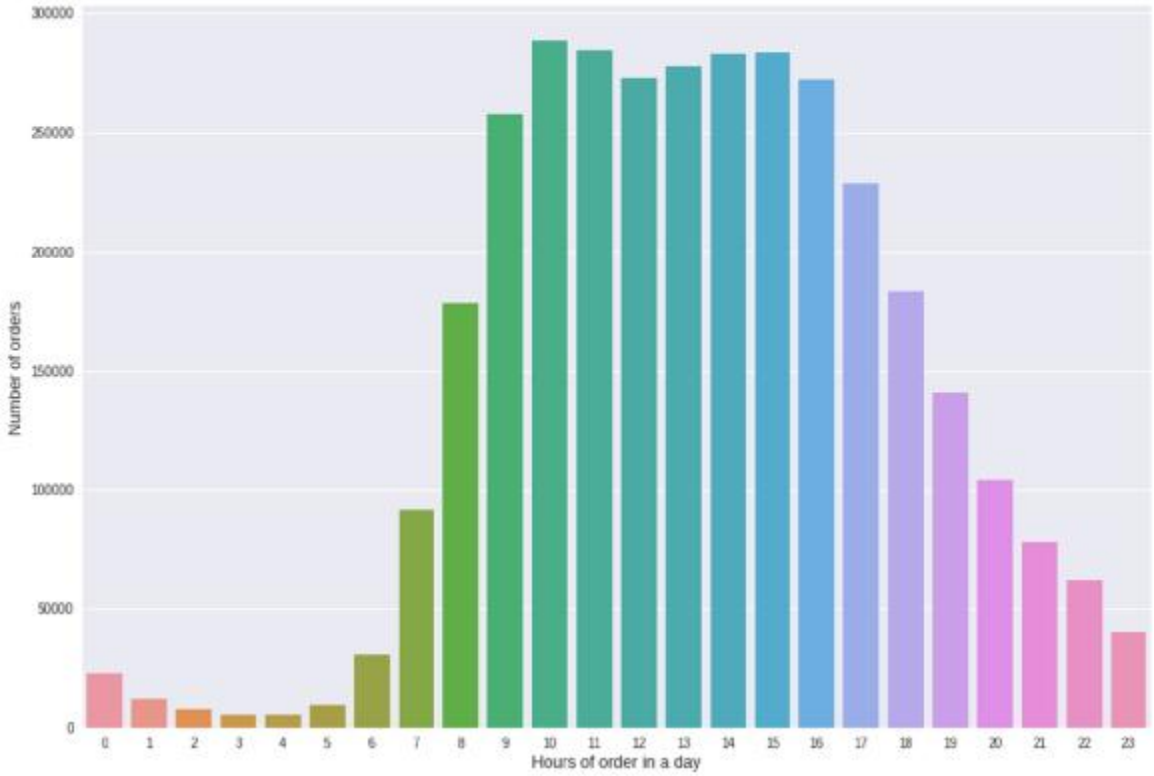


Рис. 2.15 Час замовлень

Згідно рис. 2.15 люди в основному замовляють товари з 8 до 19 години.

Визначимо дні замовлень за тиждень.

In [21]:

```
grouped = orders.groupby("order_id")["order_dow"].aggregate("sum").reset_index()
grouped = grouped.order_dow.value_counts()

f, ax = plt.subplots(figsize=(10, 10))
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of orders', fontsize=13)
plt.xlabel('Days of order in a week', fontsize=13)
plt.show()
```

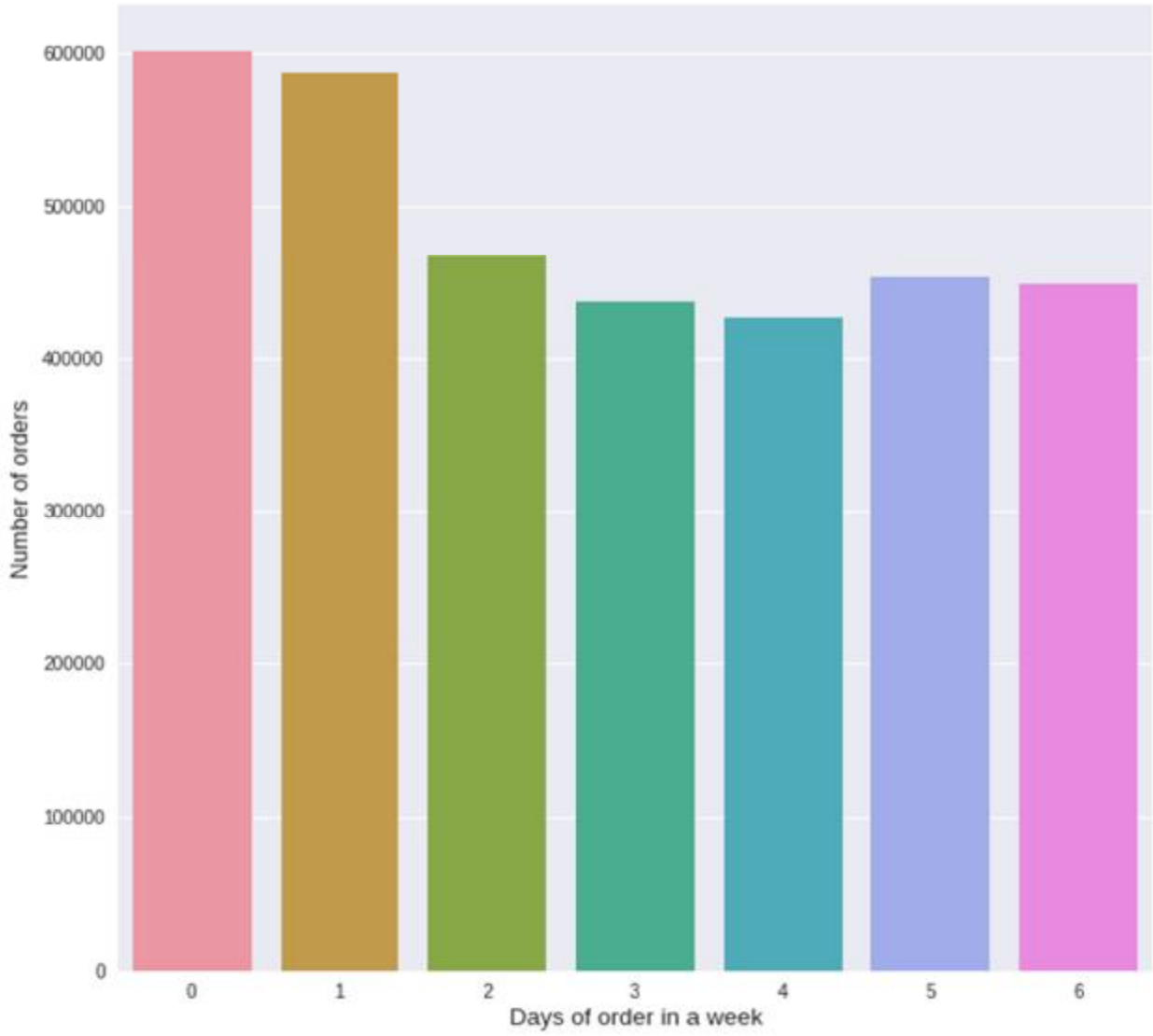


Рис. 2.16 Кількість замовлень по дням тижня

Згідно рис. 2.16 люди зазвичай замовляють в 0-й та 1-й дні тижня (дні анонімні і, можливо, це вихідні).

Періодичність повторних замовлень:

In [22]:

```
grouped = orders.groupby("order_id")["days_since_prior_order"].aggregate("sum").reset_index()
grouped = grouped.days_since_prior_order.value_counts()

from matplotlib.ticker import FormatStrFormatter
f, ax = plt.subplots(figsize=(15, 10))
sns.barplot(grouped.index, grouped.values)
ax.xaxis.set_major_formatter(FormatStrFormatter('%.0f'))
plt.ylabel('Number of orders', fontsize=13)
plt.xlabel('Period of reorder', fontsize=13)
```

```
plt.show()
```

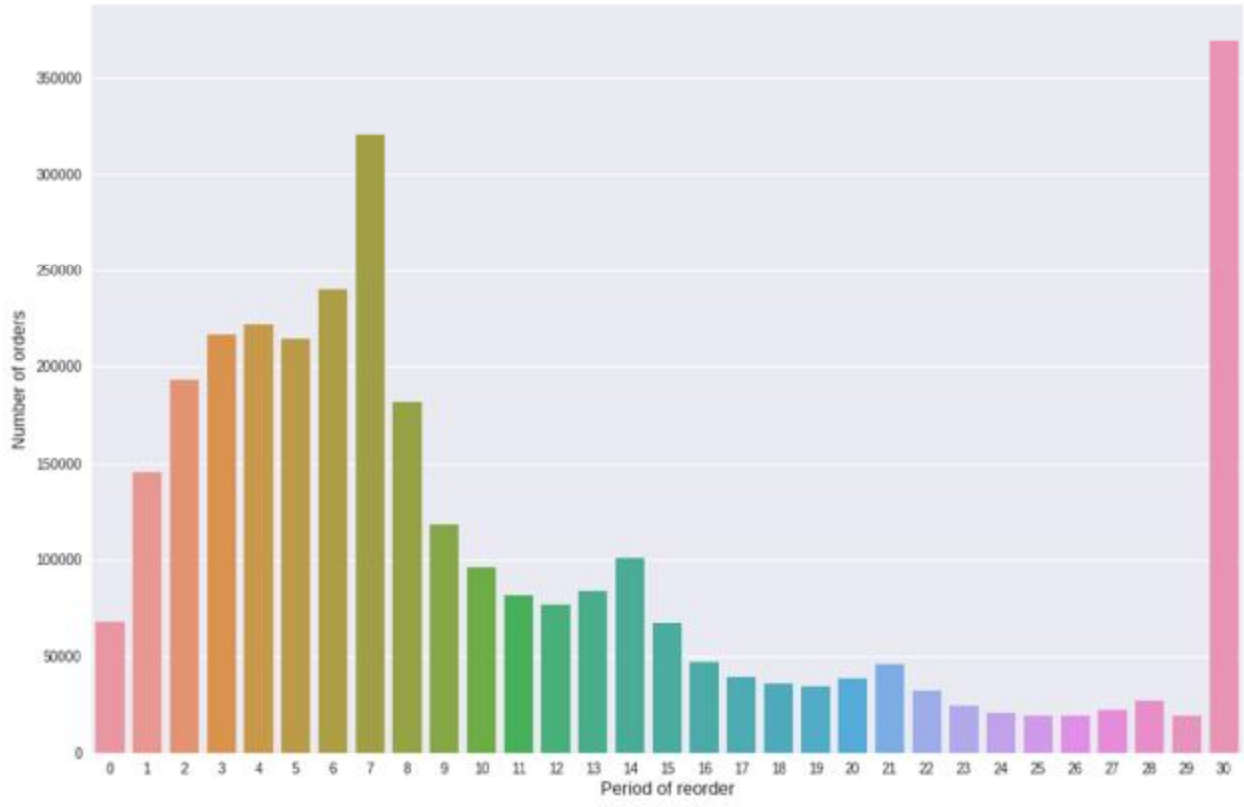


Рис. 2.17 Періодичність повторних замовлень

Згідно рис. 2.17 люди зазвичай повторюють замовлення або через 1 тиждень, або через 1 місяць.

Визначимо кількість та співвідношення замовленнь по всіх наборах даних

Кількість та співвідношення замовлень з трьох наборів даних (*prior, train, test*).

In [23]:

```
grouped = orders.groupby("eval_set")["order_id"].aggregate({'Total_orders': 'count'}).reset_index()
grouped['Ratio'] = grouped["Total_orders"].apply(lambda x: x /grouped['Total_orders'].sum())
grouped
```

Out [23]:

	eval_set	Total_orders	Ratio
0	prior	3214874	0.940
1	test	75000	0.022
2	train	131209	0.038

In [24]:

```
grouped  = grouped.groupby(['eval_set']).sum()['Total_orders'].sort_values(ascending=False)


sns.set_style('whitegrid')
f, ax = plt.subplots(figsize=(8, 8))
sns.barplot(grouped.index, grouped.values, palette='coolwarm')
plt.ylabel('Number of Orders', fontsize=13)
plt.xlabel('datasets', fontsize=13)
plt.show()
```

Побудуємо гістограму.

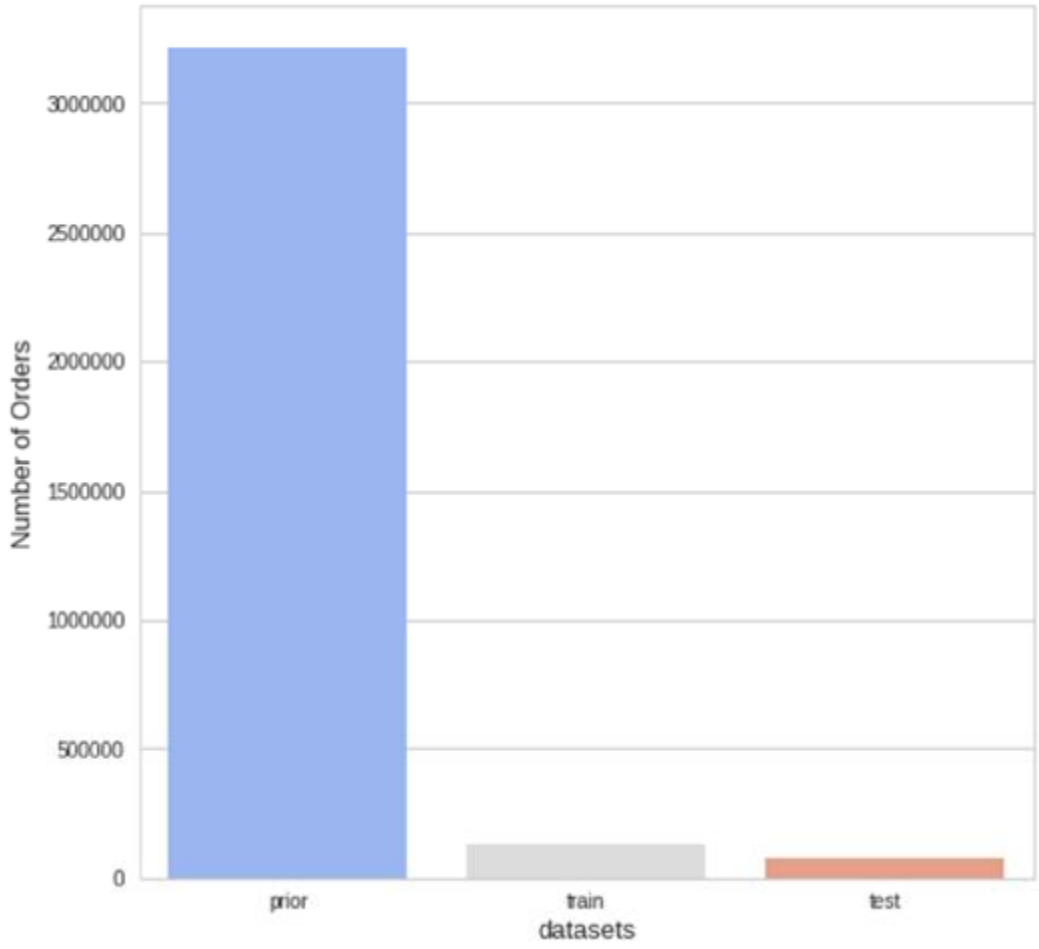


Рис. 2.18 Кількість та співвідношення замовлень з трьох наборів даних (*prior, train, test*)

Визначення унікальних клієнтів з всього набору даних

Давайте перевіримо загальну кількість унікальних клієнтів у трьох наборах даних (*prior, train, test*).

In [25]:

```
print("Number of unique customers in the whole dataset: ",len(set(orders.user_id)))
```

Результат:

```
Number of unique customers in the whole dataset:  206209
```

Загальна кількість клієнтів 206 209.

Порахуємо кількість унікальних клієнтів у кожному наборі даних.

In [26]:

```
grouped = orders.groupby("eval_set")["user_id"].apply(lambda x: len(x.unique()))
plt.figure(figsize=(7,8))
sns.barplot(grouped.index, grouped.values, palette='coolwarm')
plt.ylabel('Number of users', fontsize=13)
plt.xlabel('Eval set', fontsize=13)
plt.title("Number of unique customers in each dataset")
plt.show()
```

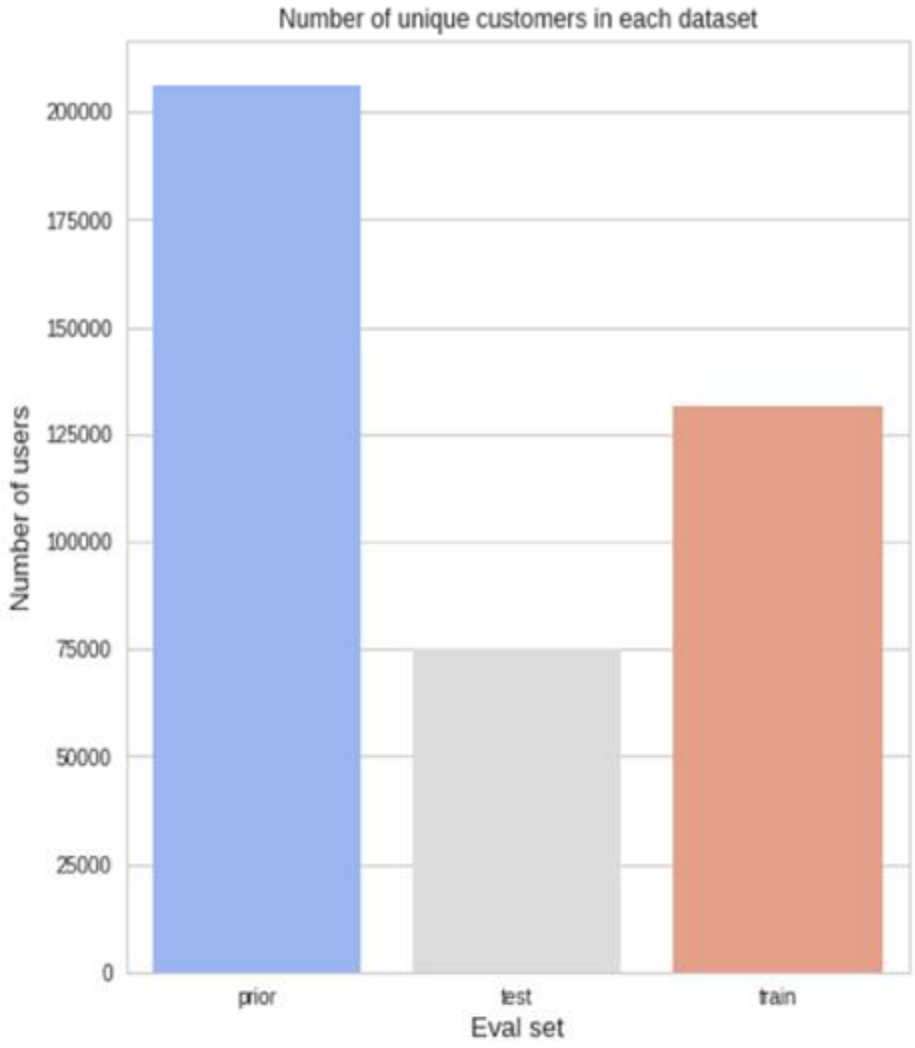


Рис. 2.19 Розподіл клієнтів по наборам даних

Визначення кількості замовлень, зроблених кожним споживачем:

Давайте порахуємо кількість замовлень, зроблених кожним клієнтом у цілому наборі даних.

In [27]:

```
grouped = orders.groupby('user_id')['order_id'].apply(lambda x: len(x.unique())).reset_index()
grouped = grouped.groupby('order_id').aggregate("count")

sns.set_style("whitegrid")
f, ax = plt.subplots(figsize=(15, 12))
sns.barplot(grouped.index, grouped.user_id)
plt.ylabel('Numbers of Customers')
plt.xlabel('Number of Orders per customer')
plt.xticks(rotation='vertical')
plt.show()
```

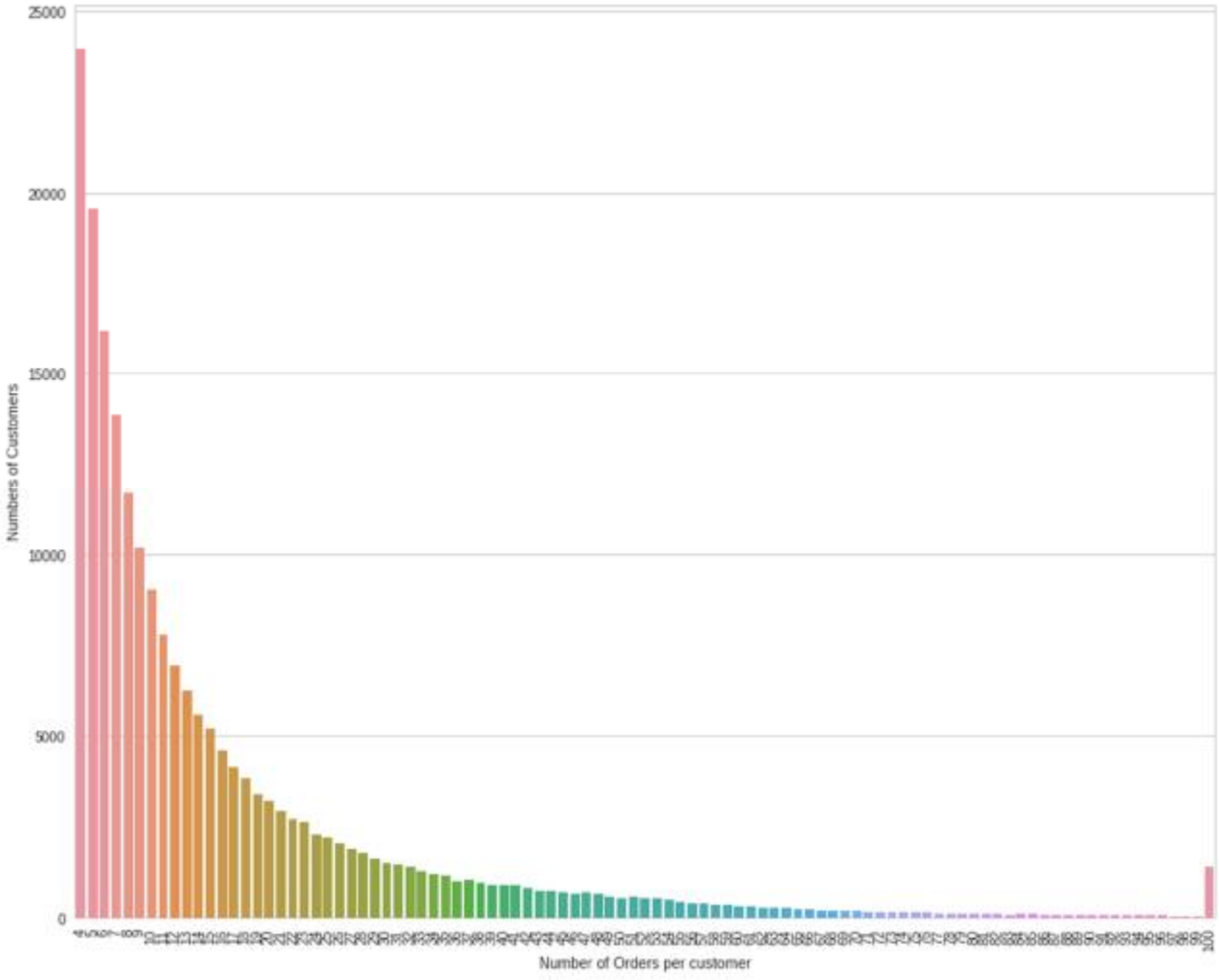


Рис. 2.20 Кількість замовлень, зроблених кожним споживачем

Згідно рис. 2.20 можемо спостерігати, що більшість замовників зробили 4 замовлення.

Тепер давайте розглянемо набори даних (продукти, відділи та групи товарів).

In [28]:

```
# Відобразити перші п'ять рядків нашого набору даних.
products.head(5)
```

Out [28]:

	product_id	product_name	aisle_id	department_id
0	1	Chocolate Sandwich Cookies	61	19
1	2	All-Seasons Salt	104	13
2	3	Robust Golden Unsweetened Oolong Tea	94	7
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1

In [29]:

```
#display first five rows of our dataset.
departments.head(5)
```

Out [29]:

	department_id	department
0	1	frozen
1	2	other
2	3	bakery
3	4	produce
4	5	alcohol

In [30]:

```
#display first five rows of our dataset.
aisles.head(5)
```

Out [30]:

	aisle_id	aisle
0	1	prepared soups salads
1	2	specialty cheeses
2	3	energy granola bars
3	4	instant foods
4	5	marinades meat preparation

Тепер давайте об'єднаємо результати у єдиному фреймі даних.

In [31]:

```
items  = pd.merge(left =pd.merge(left=products, right=departments, how='left'), right=aisles, how='left')
items.head()
```

Out [31]:

	product_id	product_name	aisle_id	department_id	department	aisle
0	1	Chocolate Sandwich Cookies	61	19	snacks	cookies cakes
1	2	All-Seasons Salt	104	13	pantry	spices seasonings
2	3	Robust Golden Unsweetened Oolong Tea	94	7	beverages	tea
3	4	Smart Ones Classic Favorites Mini Rigatoni Wit...	38	1	frozen	frozen meals
4	5	Green Chile Anytime Sauce	5	13	pantry	marinades meat preparation

Визначимо найважливіші відділи (за кількістю продуктів)

In [32]:

```
grouped = items.groupby("department")["product_id"].aggregate({'Total_products': 'count'}).reset_index()
grouped['Ratio'] = grouped["Total_products"].apply(lambda x: x /grouped['Total_products'].sum())
grouped.sort_values(by='Total_products', ascending=False, inplace=True)
grouped
```

Out [32]:

	department	Total_products	Ratio
7	personal care	6563	0.132
20	snacks	6264	0.126
16	pantry	5371	0.108
3	beverages	4365	0.088
10	frozen	4007	0.081
7	dairy eggs	3449	0.069
11	household	3085	0.062
6	canned goods	2092	0.042
9	dry goods pasta	1858	0.037
19	produce	1684	0.034
2	bakery	1516	0.031
8	deli	1322	0.027
14	missing	1258	0.025
12	international	1139	0.023
4	breakfast	1115	0.022
1	babies	1081	0.022
0	alcohol	1054	0.021
18	pets	972	0.020
13	meat seafood	907	0.018
15	other	548	0.011
5	bulk	38	0.001

In [33]:

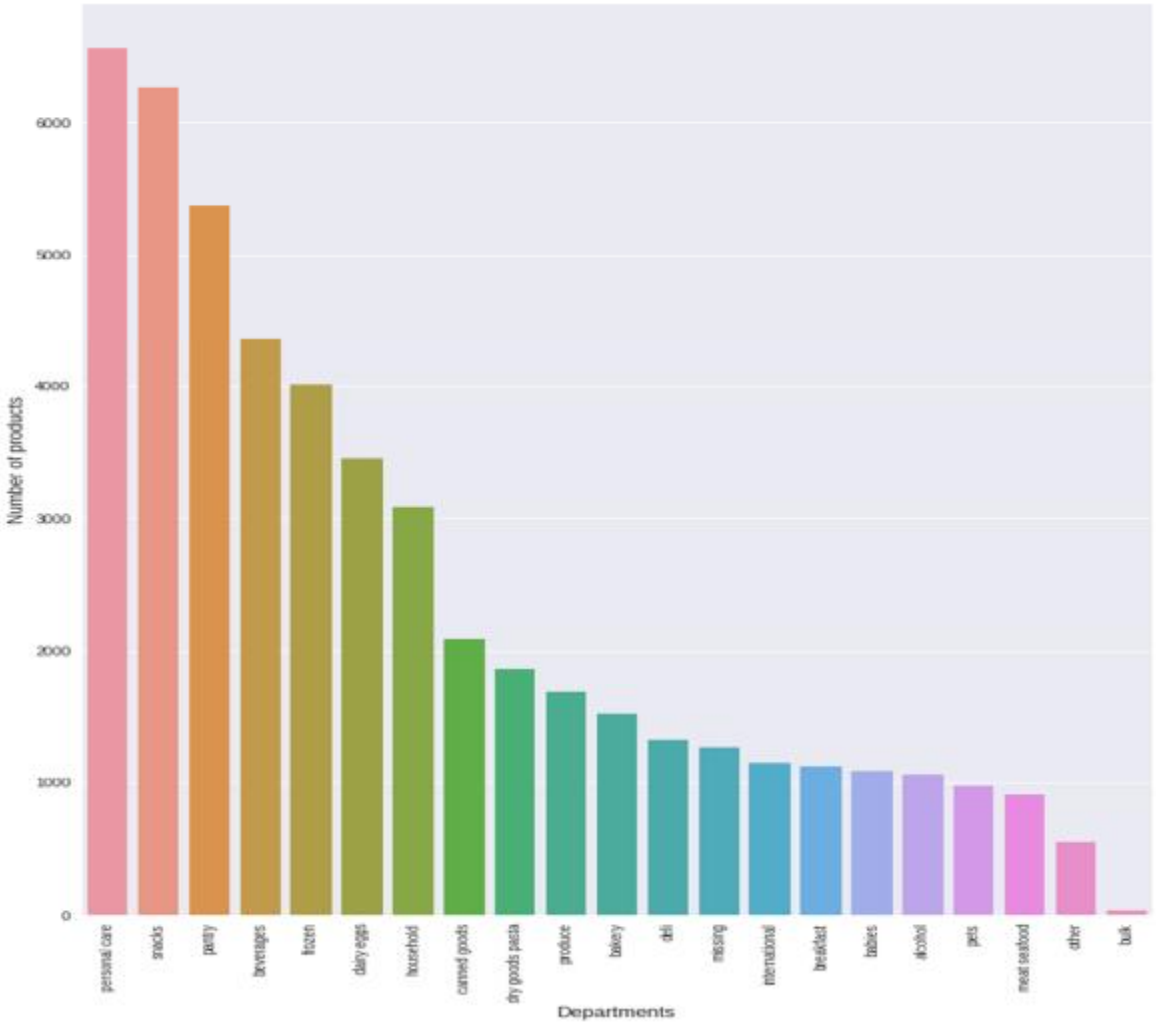


Рис. 2.21 Найважливіші відділи (за кількістю продуктів)

Визначимо найважливіші групи товарів в кожному відділі (за кількістю продуктів)

In [34]:

```
grouped = items.groupby(["department", "aisle"])["product_id"].aggregate({'Total_products':
'count'}).reset_index()
grouped.sort_values(by='Total_products', ascending=False, inplace=True)
fig, axes = plt.subplots(7,3, figsize=(20,45), gridspec_kw =  dict(hspace=1.4))
for (aisle, group), ax in zip(grouped.groupby(["department"]), axes.flatten()):
    g = sns.barplot(group.aisle, group.Total_products , ax=ax)
    ax.set(xlabel = "Aisles", ylabel=" Number of products")
    g.set_xticklabels(labels = group.aisle,rotation=90, fontsize=12)
    ax.set_title(aisle, fontsize=15)
```

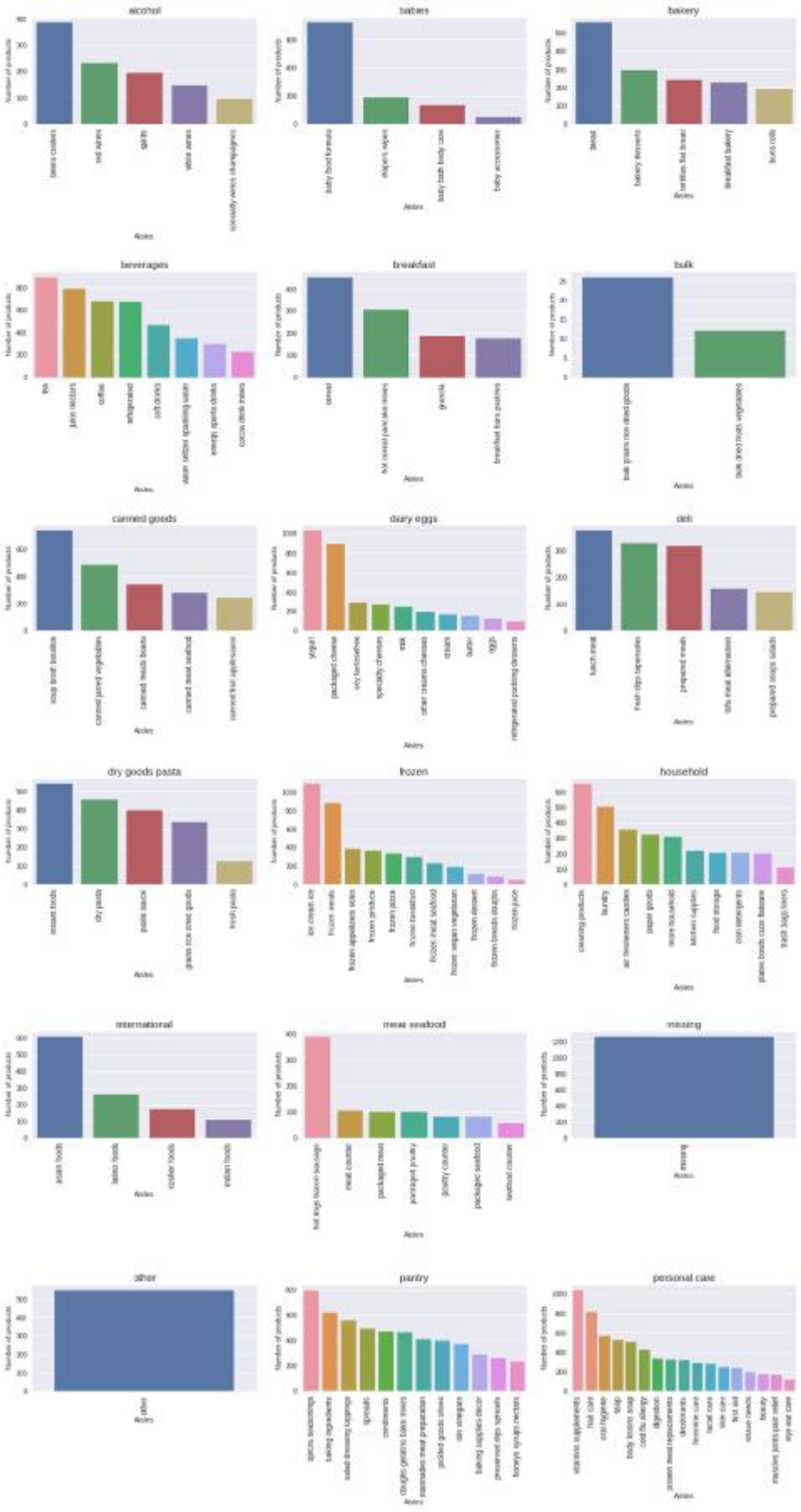


Рис. 2.22 Найважливіші групи товарів в кожному відділі

Визначимо найважливіші групи товарів по всіх відділах (за кількістю продуктів)

In [35]:

```
grouped = items.groupby("aisle")["product_id"].aggregate({'Total_products': 'count'}).reset_index()
grouped['Ratio'] = grouped["Total_products"].apply(lambda x: x /grouped['Total_products'].sum())
grouped = grouped.sort_values(by='Total_products', ascending=False)[:20]
grouped
```

Out [35]:

	aisle	Total_products	Ratio
85	missing	1258	0.025
18	candy chocolate	1246	0.025
71	ice cream ice	1091	0.022
130	vitamins supplements	1038	0.021
133	yogurt	1026	0.021
25	chips pretzels	989	0.020
125	tea	894	0.018
93	packaged cheese	891	0.018
59	frozen meals	880	0.018
31	cookies cakes	874	0.018
42	energy granola bars	832	0.017
67	hair care	816	0.016
122	spices seasonings	797	0.016
75	juice nectars	792	0.016
32	crackers	747	0.015
118	soup broth bouillon	737	0.015
4	baby food formula	718	0.014
28	coffee	680	0.014
110	refrigerated	675	0.014
26	cleaning products	655	0.013

In [36]:

```
grouped  = grouped.groupby(['aisle']).sum()['Total_products'].sort_values(ascending=False)

f, ax = plt.subplots(figsize=(12, 15))
plt.xticks(rotation='vertical')
```

```
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of products', fontsize=13)
plt.xlabel('Aisles', fontsize=13)
plt.show()
```

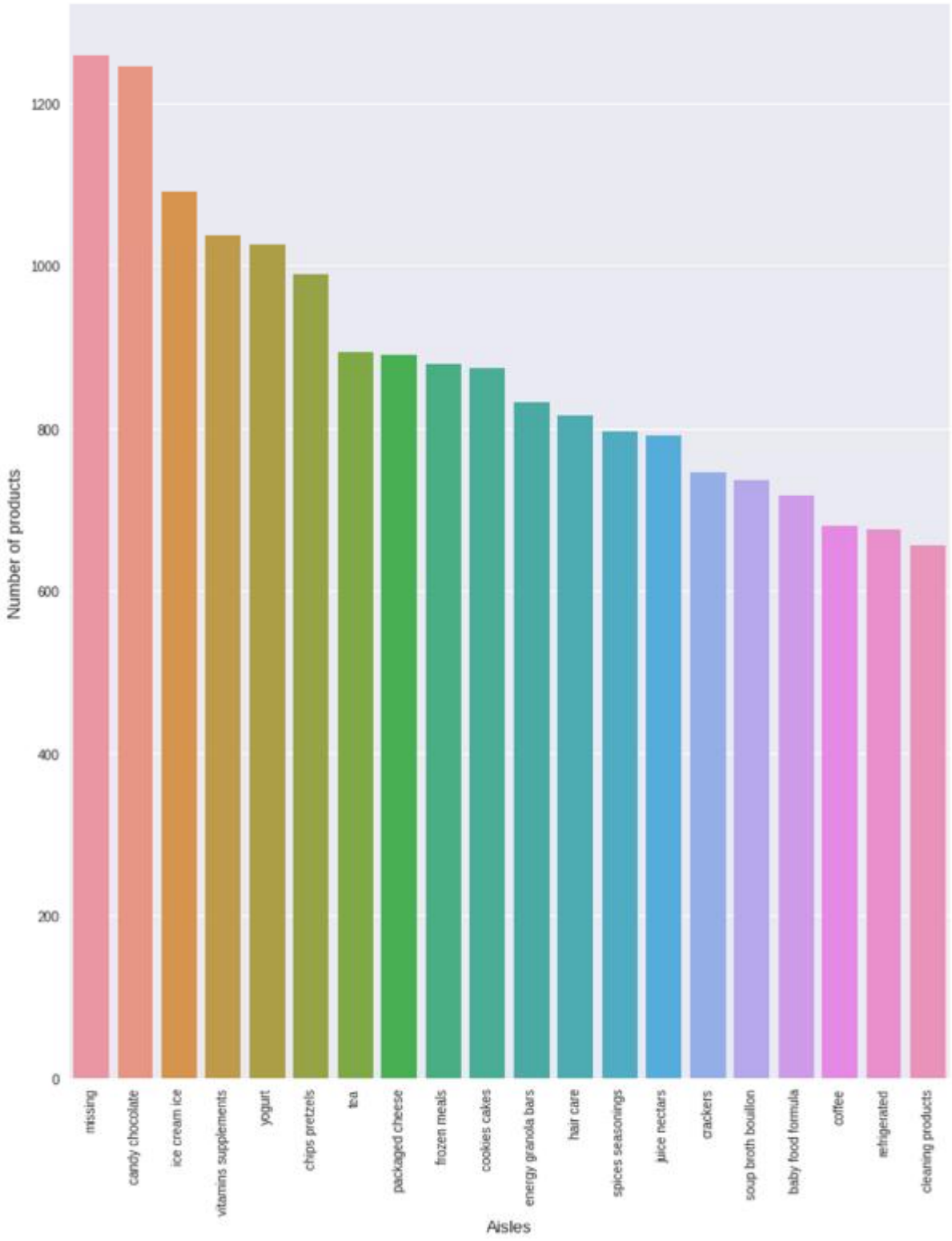


Рис. 2.23 Найважливіші групи товарів по всіх відділах

Які улюблені відділи та набори продуктів клієнтів?

In [37]:

```
users_flow = orders[['user_id', 'order_id']].merge(order_products_train[['order_id', 'product_id']],
                                    how='inner', left_on='order_id', right_on='order_id')


users_flow = users_flow.merge(items, how='inner', left_on='product_id',
                                    right_on='product_id')
```

Визначимо найкращі відділи продажу (кількість замовлень)

In [38]:

```
grouped = users_flow.groupby("department")["order_id"].aggregate({'Total_orders': 'count'}).reset_index()
grouped['Ratio'] = grouped["Total_orders"].apply(lambda x: x /grouped['Total_orders'].sum())
grouped.sort_values(by='Total_orders', ascending=False, inplace=True)
grouped
```

Out [38]:

	department	Total_orders		Ratio
19	produce	409087		0.295
7	dairy eggs	217051		0.157
20	snacks	118862		0.086
3	beverages	114046		0.082
10	frozen	100426		0.073
16	pantry	81242		0.059
2	bakery	48394		0.035
6	canned goods	46799		0.034
8	deli	44291		0.032
9	dry goods pasta	38713		0.028
11	household	35986		0.026
13	meat seafood	30307		0.022
4	breakfast	29500		0.021
17	personal care	21570		0.016
1	babies	14941		0.011
12	international	11902		0.009
14	missing	8251		0.006
0	alcohol	5598		0.004
18	pets	4497		0.003
15	other	1795		0.001
5	bulk	1359		0.001

In [39]:

```
grouped  = grouped.groupby(['department']).sum()['Total_orders'].sort_values(ascending=False)


f, ax = plt.subplots(figsize=(12, 15))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of Orders', fontsize=13)
plt.xlabel('Departments', fontsize=13)
plt.show()
```

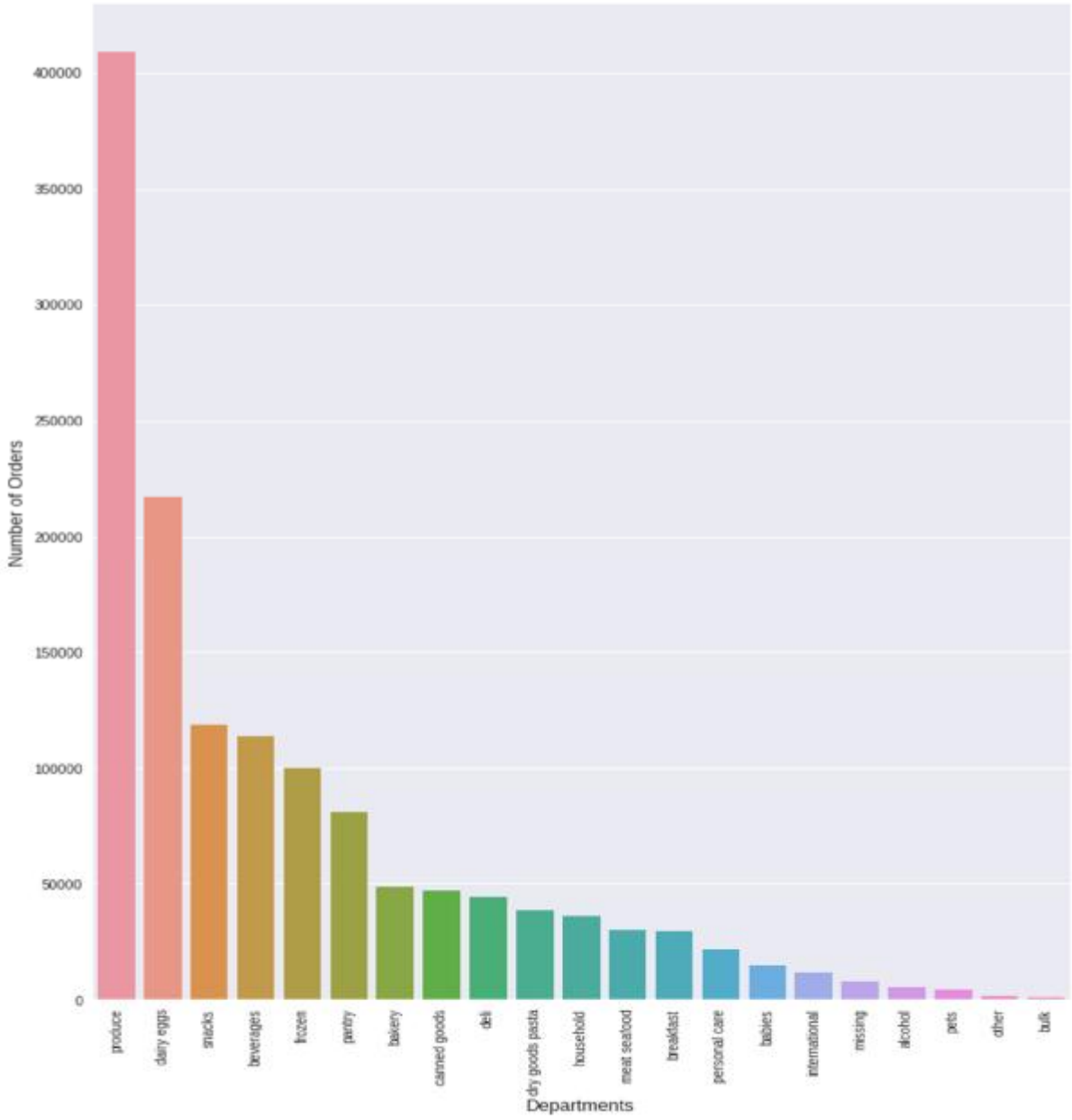


Рис. 2.24 Найкращі відділи продажу (кількість замовлень)

Найкращі групи товарів по кожному відділу (за кількістю замовлень)

In [40]:

```
grouped = users_flow.groupby(["department", "aisle"])["order_id"].aggregate({'Total_orders':
'count'}).reset_index()
grouped.sort_values(by='Total_orders', ascending=False, inplace=True)
fig, axes = plt.subplots(7,3, figsize=(20,45), gridspec_kw =  dict(hspace=1.4))
for (aisle, group), ax in zip(grouped.groupby(["department"]), axes.flatten()):
    g = sns.barplot(group.aisle, group.Total_orders , ax=ax)
    ax.set(xlabel = "Aisles", ylabel=" Number of Orders")
    g.set_xticklabels(labels = group.aisle,rotation=90, fontsize=12)
    ax.set_title(aisle, fontsize=15)
```

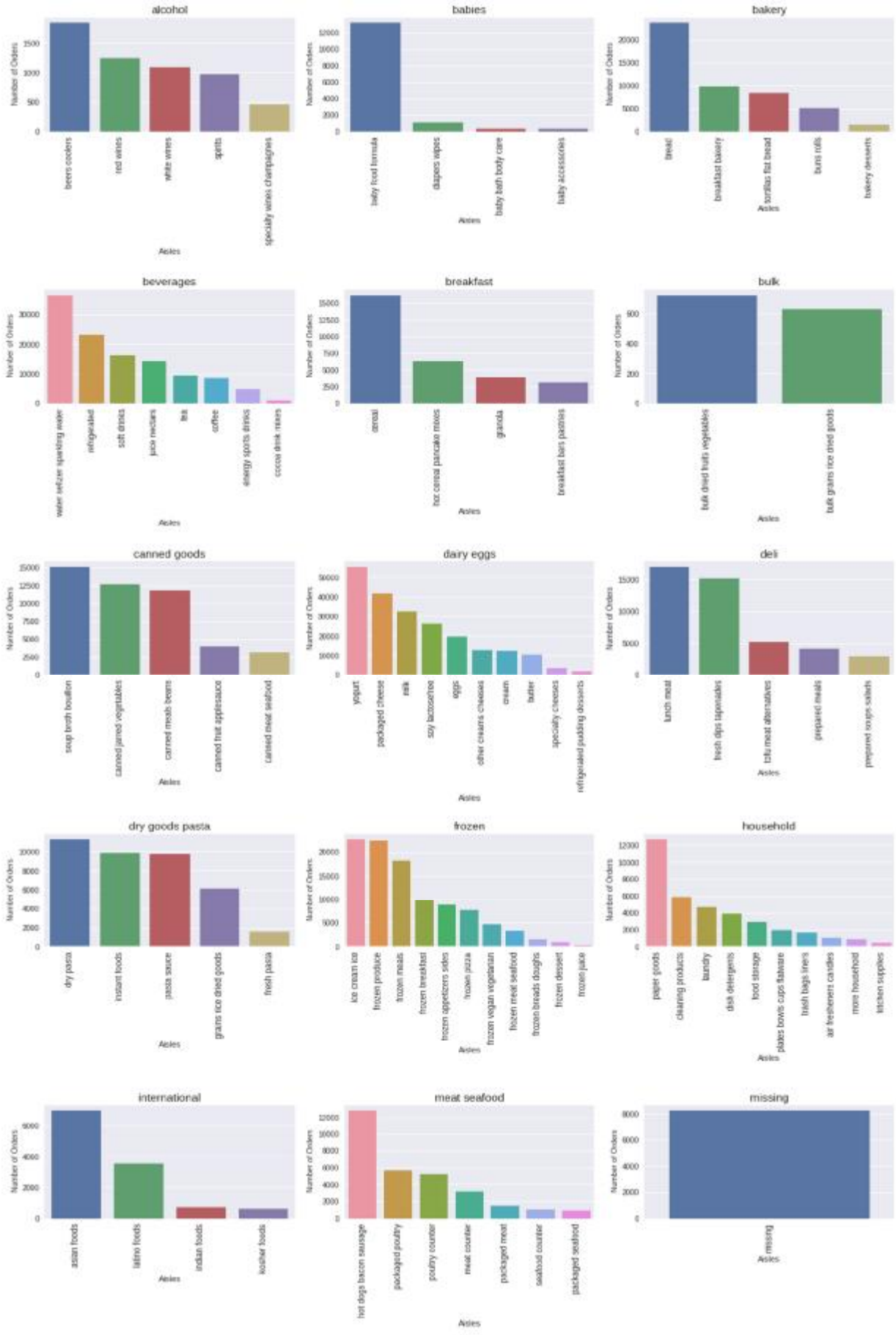


Рис. 2.25 Найкращі групи товарів по кожному відділу (за кількістю замовлень)

Найкращі групи товарів по всім відділам (за кількістю замовлень)

In [41]:

```
grouped = users_flow.groupby("aisle")["order_id"].aggregate({'Total_orders': 'count'}).reset_index()
grouped['Ratio'] = grouped["Total_orders"].apply(lambda x: x /grouped['Total_orders'].sum())
grouped.sort_values(by='Total_orders', ascending=False, inplace=True )
grouped.head(10)
```

Out [41]:

	aisle	Total_orders	Ratio
53	fresh vegetables	150609	0.109
50	fresh fruits	150473	0.109
98	packaged vegetables fruits	78493	0.057
133	yogurt	55240	0.040
93	packaged cheese	41699	0.030
131	water seltzer sparkling water	36617	0.026
83	milk	32644	0.024
25	chips pretzels	31269	0.023
119	soy lactosefree	26240	0.019
11	bread	23635	0.017

In [42]:

```
grouped  = grouped.groupby(['aisle']).sum()['Total_orders'].sort_values(ascending=False)[:15]


f, ax = plt.subplots(figsize=(12, 15))
plt.xticks(rotation='vertical')
sns.barplot(grouped.index, grouped.values)
plt.ylabel('Number of Orders', fontsize=13)
plt.xlabel('Aisles', fontsize=13)
plt.show()
```

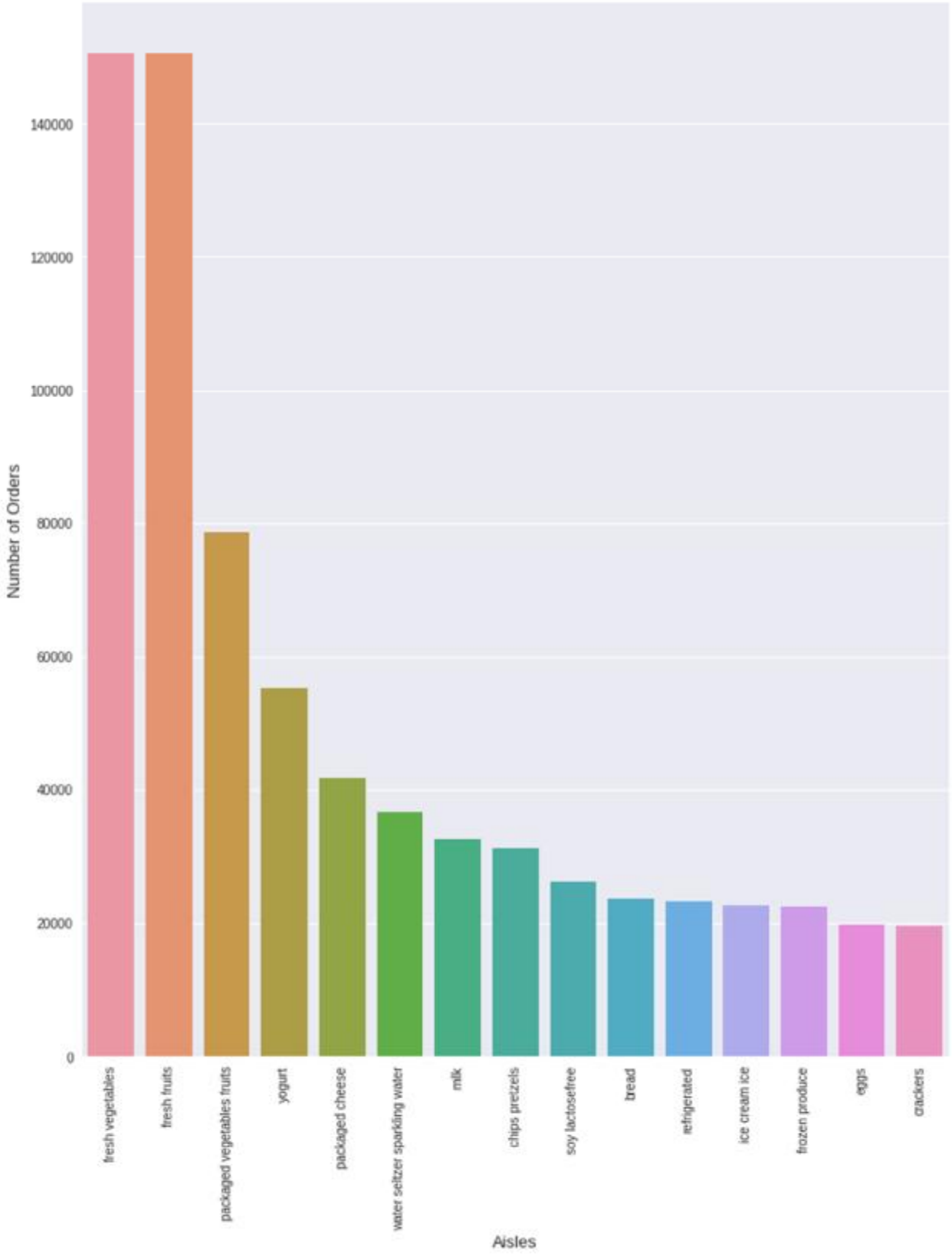


Рис. 2.26 Найкращі групи товарів по всім відділам (за кількістю замовлень)

2.7. Програмний комплекс Python для обробки великих даних

Основні властивості мови Python:

• гнучка - підходить для розробки програм самого різного призначення і складності: від простих навчальних програм до професійних проектів.

• складається з нескладних для освоєння конструкцій – на відміну від інших мов, про які ви могли чути (C, PHP, Pascal).

• не вимагає компіляції, тобто програми на Python можна запускати на будь-яких платформах (Windows, Linux, MacOS, ...).

• підтримує безліч стилів програмування, на її прикладі можна освоїти будь-яку парадигму (ООП, структурне програмування, функціональне програмування).

• містить ще безліч корисних можливостей.

Мова програмування Python використовується при розробці таких відомих інтернет-сервісів, як YouTube, Google, Yahoo !; Python застосовують при розробці програм в NASA (космічне агентство США) і CERN (Європейська організація з ядерних досліджень), а також, добре підходить для обробки економічних даних, отриманих з мережі Internet.

2.7.1. Установка Python на Windows

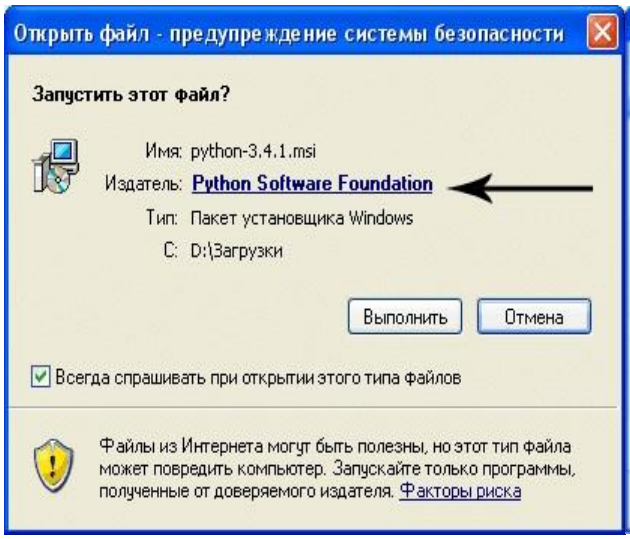
Завантажувати Python треба з офіційного сайту. Не рекомендується завантажувати інтерпретатор Python з інших сайтів або через торрент, в них можуть бути віруси. Програма безкоштовна. Потрібно зайти на *https://python.org/downloads/windows/*, та обрати "*latest python release*" і взяти найсвіжішу версію Python.

З'являється сторінка з описом даної версії Python (англійською). Потім крутимо в самий низ сторінки та відкриваємо "*download page*", як це показано на рисунках внизу.

Ви побачите список файлів, які можна завантажити. Нам потрібен Windows x86 MSI installer (якщо система 32-х бітна), або Windows x86-64 MSI installer (якщо система 64-х бітна). Більше з файлів нічого не потрібно.

Version	Operating System	Description	Date	MD5 Sum	File Size
Mac OS X 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later		316a2f83edff73bbbcb2c84390bee2db	22776248
Mac OS X 32-bit i386/PPC installer	Mac OS X	for Mac OS X 10.5 and later		534f8ec2f5ad5539f9165b3125b5e959	22692757
XZ compressed source tarball	Source release			6cafc183b4106476dd73d5738d7f616a	14125788
Gzipped source tarball	Source release			26695450087f8587b26d0b6a63844af5	19113124
Windows debug information files	Windows			9ce29e8356cf13f88e41f7595c2d7399	36744364
Windows x86 MSI installer	Windows			4940c3fad01ffa2ca7f9cc43a005b89a	24408064
Windows debug information files for 64-bit binaries	Windows			44a2d4d3c62a147f5a9f733b030490d1	24129218
Windows help file	Windows			6ff47ff938b15d2900f3c7311ab629e5	7297786
Windows x86-64 MSI installer	Windows	for AMD64/EM64T/x64, not Itanium processors		25440653f27ee1597fd6b3e15eee155f	25104384

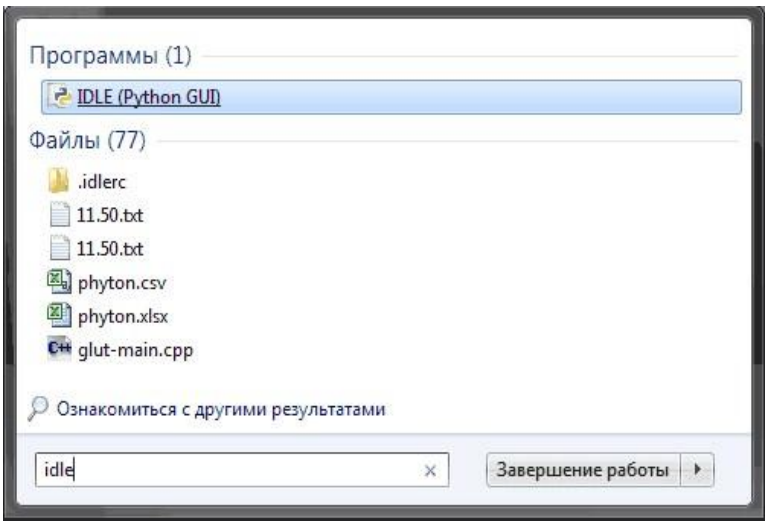
Чекаємо, поки *Python* завантажиться. Потім відкриваємо завантажений файл. Якщо він підписаний *Python Software Foundation*, значить файл скачано вірно. Встановлюємо доступ для всіх користувачів або тільки для одного (на ваш розсуд).



Вибираємо папку для установки. Вибираємо компоненти, які будуть встановлені. Залиште компоненти за замовчуванням, якщо не впевнені.

2.7.2. Середовище розробки IDLE

Після завантаження та установки Python відкриваємо IDLE (середовище розробки на мові Python, що поставляється разом з дистрибутивом).

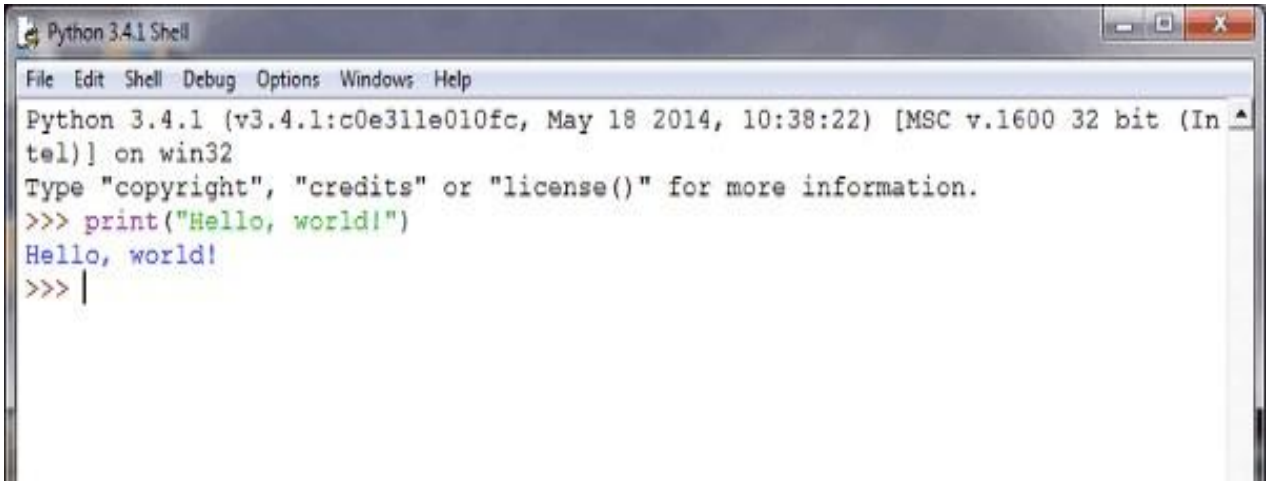


Запускаємо IDLE (спочатку запускається в інтерактивному режимі), після чого вже можна починати писати першу програму. Традиційно, першою програмою у нас буде "hello world".

Щоб написати "hello world" на python, достатньо всього одного рядка:

```
print ("Hello world!")
```

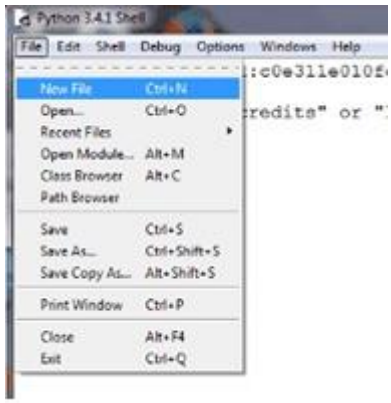
Вводимо цей код в IDLE і натискаємо Enter. Результат видно на зображенні нижче.



З інтерактивним режимом ми трохи познайомилися, можете з ним ще попрактикувася, наприклад, написати:

```
print (3 + 4)
print (3 * 5)
print (3 ** 2)
```

Але, все-таки, інтерактивний режим не буде основним. В основному, ви будете зберігати програмний код у файлі та запускати вже файл.



Для того, щоб створити нове вікно, в інтерактивному режимі IDLE виберіть File → New File (або натисніть Ctrl + N).

У вікні, введіть наступний код:

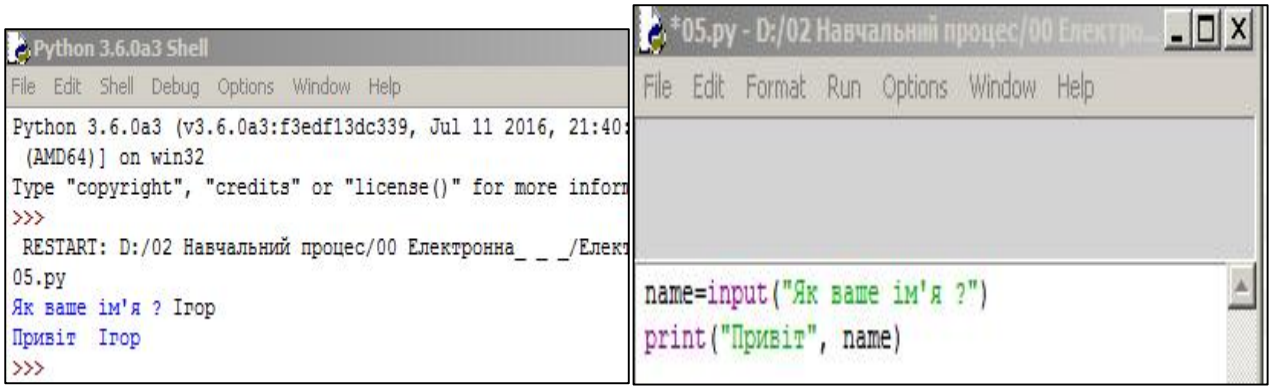
```
name = input ( "Як Вас звати?")
print ( "Привіт,", name)
```

Перший рядок друкує питання ("Як Вас звати?"), очікує, поки ви не надрукуєте що-небудь і не натиснете *Enter* і зберігає введене значення в змінній *name*.

У другому рядку ми використовуємо функцію *print* для виведення тексту на екран, в даному випадку для виведення "Привіт," і того, що зберігається в змінній "name".

Тепер натиснемо F5 (або виберемо в меню IDLE Run → Run Module) і переконаємося, що те, що ми написали, працює. Перед запуском IDLE запропонує нам зберегти файл. Збережемо туди, куди вам буде зручно, після чого програма запуститься.

Ви повинні побачити щось на зразок цього (на скріншоті зліва – файл з написаної вами програмою, праворуч – результат її роботи).



2.7.3. Синтаксис

Кінець рядка є кінцем інструкції (крапка з комою не потрібно).

Вкладені інструкції об'єднуються в блоки по величині відступів. Відступ може бути будь-яким, головне, щоб в межах одного вкладеного блоку відступ був однаковий. І про зручність читання коду не забувайте. Відступ в 1 пробіл, наприклад, не найкраще рішення. Використовуйте 4 пробіли (або знак табуляції).

Вкладені інструкції в Python записуються відповідно до одного й того ж шаблону, коли основна інструкція завершується двокрапкою, слідом за яким розташовується вкладений блок коду, зазвичай з відступом під рядком основний інструкції.

Основна інструкція:

Вкладений блок інструкцій

Іноді можливо записати кілька інструкцій в одному рядку, розділяючи їх крапкою з комою:

a = 1; b = 2; print (a, b)

Але не робіть це занадто часто! Пам'ятайте про зручність читання коду.

Припустимо записувати одну інструкцію в декількох рядках. Досить її укласти в пару круглих, квадратних або фігурних дужок:

```
if (a == 1 and b == 2 and c == 3 and d == 4): # Не забуваємо про двокрапку
    print('spam' * 3)
```

Тіло складової інструкції може розташовуватися в тому ж рядку, що і тіло основної, якщо тіло є складової інструкції і не містить складових інструкцій.. Наприклад:

if x> y: print(x)

Знак # означає коментар, що не обробляється Pyton і потрібен тільки для того, щоб описати складні елементи програм

2.7.4. Інструкція if-elif-else, перевірка істинності, тримісний вираз if / else

Умовна інструкція *if-elif-else* (її ще іноді називають оператором розгалуження) – основний інструмент вибору в *Python*. Простіше кажучи, вона вибирає, яку дію слід виконати, в залежності від значення змінних в момент перевірки умови.

Спочатку записується частина if з умовним виразом, далі можуть слідувати одна або більше необов'язкових частин elif, і, нарешті, необов'язкова частина else. Загальна форма запису умовної інструкції if виглядає наступним чином:

```
if test1:
   state1
elif test2:
   state2
else:
   state3
```

Простий приклад в якому на виході надрукується 'true', так як 1 – істина):

```
if 1:
print ( 'true')
else: print ( 'false')

true
```

Трохи складніший приклад (його результат буде залежати від того, що ввів користувач):

```
a = int(input())
if a <-5:
   print( 'Low')
elif -5 <= a <= 5:
   print( 'Mid')
else:
```

```
    print( 'High')
```

Конструкція з декількома elif може також служити заміною конструкції switch-case в інших мовах програмування.

Взагалі в опеораторі if діє простий принцип:

Будь-яке число, не рівне 0, або непорожній об'єкт – істина.

Числа, рівні 0, порожні об'єкти і значення None – брехня.

Операції порівняння застосовуються до структур даних рекурсивно та повертають *True* або *False*.

Логічні оператори *and* та *or* повертають істинний або помилковий об'єкт:

X and Y

Істина, якщо обидва значення X і Y істинні.

X or Y

Істина, якщо хоча б одне зі значень X або Y істинно.

not X

Істина, якщо X хибне.

Тримісний вираз if / else працює за наступною інструкцією:

```
if X:
   A = Y
else:
   A = Z
```

Хоч ця інструкці досить коротка, але, тим не менше, займає цілих 4 рядки. Спеціально для таких випадків і було придумано вираз if / else:

A = Y if X else Z

У даній інструкції інтерпретатор виконає вираз Y, якщо X істинно, в іншому випадку виконається вираз Z.

```
>>> A = 't' if 'spam' else 'f'’
>>> A
' t '
```

2.7.5. Цикли for і while, оператори break і continue, команда else

While – один з найбільш універсальних циклів в Python, тому досить повільний. Виконує тіло циклу до тих пір, поки умова циклу істинно. Наприклад,

i = 5

while i <15:

print (i)

i = i + 2

5

7

9

11

13

При виконанні циклу while спочатку перевіряється умова. Якщо вона помилкова, то виконання циклу припиняється і керування передається на наступну інструкцію після тіла циклу while. Якщо умова істинно, то виконується інструкція, після чого умова перевіряється знову і знову виконується інструкція. Так триває до тих пір, поки умова буде істинно. Як тільки умова стане помилково, робота циклу завершиться і управління передасться наступної інструкції після циклу. Це єдиний оператор, після якого можуть йти декілька вкладений операторів.

Наприклад, наступний фрагмент програми надрукує на екран квадрати всіх цілих чисел від 1 до 10. Видно, що цикл while може бути замінений циклом for ... in range (...):, який буде описано пізніше.

```
i = 1
while i <= 10:
    print(i ** 2)
    i += 1
```

У цьому прикладі змінна і всередині циклу змінюється від 1 до 10. Така змінна, значення якої змінюється з кожним новим проходом циклу, називається

лічильником. Зауважимо, що після виконання цього фрагмента значення змінної i дорівнюватиме 11, оскільки саме при i == 11 умова i <= 10 вперше перестане виконуватися.

Ось ще один приклад використання циклу while для визначення кількості цифр натурального числа n:

```
n = int(input())
length = 0
while n > 0:
    n //= 10  # Це еквівалентно n = n // 10
    length += 1
print(length)
```

У цьому циклі ми відкидаємо по одній цифрі числа, починаючи з кінця, що еквівалентно целочисленному діленню на 10 (n // = 10), при цьому в змінній length визначаємо, скільки разів це було зроблено.

У мові Пітон є й інший спосіб вирішення цього завдання: length = len (str (i)).

Після тіла циклу можна написати слово else: і після нього блок операцій, який буде виконаний один раз після закінчення циклу, коли перевіряється умова стане невірною.

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Цикл закінчено, i =', i)
```

Здавалося б, ніякого сенсу в цьому немає, адже цю ж інструкцію можна просто написати після закінчення циклу. Сенс з'являється тільки разом з інструкцією break. Якщо під час виконання Пітон зустрічає інструкцію break всередині циклу, то він відразу ж припиняє виконання цього циклу і виходить з нього. При цьому гілка else виконуватися не буде. Зрозуміло, інструкцію break має сенс викликати тільки всередині інструкції if, тобто вона повинна виконуватися тільки при виконанні якоїсь особливої умови.

Наведемо приклад програми, яка зчитує числа до тих пір, поки не зустріне від'ємне число. При появі від'ємного числа програма завершується. У першому варіанті послідовність чисел завершується числом 0 (при зчитуванні якого треба зупинитися).

```
a = int(input())
while a != 0:
    if a < 0:
        print('Зустрілось від’ємне число', a)
        break
    a = int(input())
else:
    print('Жодного від’ємного числа не зустрілося')
```

У другому варіанті програми спочатку на вхід подається кількість елементів послідовності, а потім і самі елементи. В такому випадку зручно скористатися циклом for. Цикл fortакже може мати гілку else і містити інструкції break всередині себе.

```
n = int(input())
for i in range(n):
    a = int(input())
    if a < 0:
        print('Зустрілось від’ємне число', a)
        break
else:
    print('Жодного від’ємного числа не зустрілося')
```

Цикл for вже трішки складніший, трохи менше універсальний, але виконується набагато швидше циклу while. Цей цикл проходиться по будь-якому об'єкту, що піддається повтору, або як кажуть ітерації (рядку, списку, ...), і під час кожного проходу виконує тіло циклу.

```
for i in 'hello world': print (i * 2, end = '')
hheellllloo wwoorrlldd
```

Оператор *continue* починає наступний прохід циклу, минаючи тіло циклу, що залишилося (for або while) і виконання циклу продовжується з наступної ітерації.

```
>>> for i in 'hello world':
if i == 'o': continue
print (i * 2, end = '')
hheellll wwrrlldd
```

Оператор *break* достроково перериває цикл.

```
for i in 'hello world':
    if i == 'o':
    break
    print (i * 2, end = '')
hheellll
```

Слово else, застосоване в циклі for або while, перевіряє, чи був проведений вихід з циклу інструкцією break, або ж "природним" чином. Блок інструкцій всередині else виконається тільки в тому випадку, якщо вихід з циклу стався без допомоги break.

```
for i in 'hello world':
    if i == 'a':
        break
    else:
        print ( 'Букви а в рядку немає')
Букви а в рядку немає
```

З прикладів добре видно, що результати роботи програми з’являються у тому ж вікні, що й сам текст програми.

Інша інструкція управління циклом – continue (продовження циклу). Якщо ця інструкція зустрічається десь посередині циклу, то пропускаються всі інструкції, що залишилися до кінця циклу.

В операторі for існує поняття внутрішнього циклу або якогось іншого оператора. Для цього цей оператор потірбно розмістити зі ссувовм управо, натиснувши кнопку TAB. В стандартних налаштуваннях зміщення виконується на три позиції.

Якщо інструкції break і continue містяться всередині декількох вкладених циклів, то вони впливають лише на виконання самого внутрішнього циклу. Наведемо приклад, який це демонструє:

```
for i in range(3):
    for j in range(5):
        if j > i:
            break
        print(i, j)
```

А ось приклад, коли вводиться матриця *a*, для якої розраховуєтьсмя сума елементів по рядках, починаючи з другої колонки. Ця сума множиться на квадрат номеру колонки і записується в матрицю *q*.

```
a=[[1,2,3],[4,5,6],[7,8,9],[10,11,12]]
q=[]
c=0
for i in range(3):
    for j in range(2):
        for k in range(2):
            if k<1:
                continue
            else:
                c+=k**2*a[i][j]
    q.append(c)
print(q)
```

2.7.6. Вбудовані функції Pyton

abs (x) – повертає абсолютну величину (модуль числа).

all(послідовність) – повертає True, якщо всі елементи істині (або, якщо послідовність порожня).

any(послідовність) – повертає True, якщо хоча б один елемент - істина. Для порожньої послідовності повертає False.

ascii(object) – як repr (), повертає рядок, що містить представлення об'єкта, але замінює не ASCII символи на екрановані послідовності.

bin(x) – перетворення цілого числа в двійкову рядок.

bool(x) – перетворення до типу bool, що використовує стандартну процедуру перевірки істинності. Якщо x є помилковим або опущений, повертає значення False, в іншому випадку вона повертає True.

bytearray ([джерело [, кодування [помилки]]]) – перетворення до bytearray. Bytearray – змінна послідовність цілих чисел в діапазоні 0 <= X < 256. Викликана без аргументів, повертає порожній масив байтів.

bytes ([джерело [, кодування [помилки]]]) – повертає об'єкт типу bytes, який є незмінною послідовністю цілих чисел в діапазоні 0 <= X < 256. Аргументи конструктора інтерпретуються як для bytearray ().

callable (x) – повертає True для об'єкта, що підтримує виклик (як функції).

chr(x) – повертає односимвольних рядок, код символу якої дорівнює x.

classmethod (x) – створює зазначену функцію методом класу.

compile (source, filename, mode, flags = 0, dont_inherit = False) – компіляція в програмний код, який згодом може виконатися функцією eval або exec. Рядок не повинен містити символів повернення каретки або нульові байти.

delattr (object, name) – видаляє атрибут з ім'ям 'name'.

complex([real [, imag]]) – перетворення до комплексного числа.

dict ([object]) – перетворення до словника.

dir ([object]) – список імен об'єкта, а якщо об'єкт не вказано, список імен в поточній локальній області видимості.

divmod (a, b) – повертає часткове і залишок від ділення a на b.

enumerate (iterable, start = 0) – повертає ітератор, при кожному проході надає кортеж з номера і відповідного члена послідовності.

eval(expression, globals = None, locals = None) – виконує рядок програмного коду.

exec(object [, globals [, locals]]) – виконує програмний код на Python.

filter(function, iterable) – повертає ітератор з тих елементів, для яких function повертає істину.

float ([X]) – перетворення до числа з плаваючою крапкою. Якщо аргумент не вказано, повертається 0.0.

format(value [, format_spec]) – форматування (зазвичай форматування рядка).

getattr (object, name, [default]) – витягує атрибут об'єкта або default.

frozenset ([послідовність]) – повертає незміну множину.

globals () – словник глобальних імен.

hasattr (object, name) – чи має об'єкт атрибут з ім'ям 'name'.

hash (x) – повертає хеш зазначеного об'єкта.

help ([object]) – виклик вбудованої довідкової системи.

hex (x) – перетворення цілого числа в шістнадцяткову рядок.

id (object) – повертає "адреса" об'єкта. Це ціле число, яке гарантовано буде унікальним і постійним для даного об'єкта протягом терміну його існування.

input ([prompt]) – повертає введений користувачем рядок.

int ([object], [підставу системи числення]) – перетворення до цілого числа.

isinstance (object, ClassInfo) – істина, якщо об'єкт є екземпляром ClassInfo або його підкласом. Якщо об'єкт не є об'єктом даного типу, функція завжди повертає брехня.

issubclass(клас, ClassInfo) – істина, якщо клас є підкласом ClassInfo. Клас вважається підкласом себе.

iter(x) – повертає об'єкт ітератора.

len(x) – повертає число елементів в зазначеному об'єкті.

list([object]) – створює список.

locals () – словник локальних імен.

map(function, iterator) – ітератор, що вийшов після застосування до кожного елементу послідовності функції function.

max(iter, [args ...] * [, key]) – максимальний елемент послідовності.

memoryview ([object]) – створює об'єкт memoryview.

min(iter, [args ...] * [, key]) – мінімальний елемент послідовності.

next (x) – повертає наступний елемент ітератора.

object() – повертає безликий об'єкт, який є базовим для всіх об'єктів.

oct(x) – перетворення цілого числа в вісімковий формат.

open(file, mode = 'r', buffering = None, encoding = None, errors = None, newline = None, closefd = True) – відкриває файл і повертає відповідний потік.

ord (c) – код символу.

pow(x, y, [r]) - (x ** y)%

range ([start = 0], stop, [step = 1]) – арифметична прогресія від start до stop з кроком step.

repr (obj) – представлення об'єкта.

print ([object, ...], *, sep = "", end = '\ n', file = s

set ([object]) – створює множину.

slice ([start = 0], stop, [step = 1]) – об'єкт зрізу від start до stop з кроком step.

str([object], [кодування], [помилки]) – строкове представлення об'єкту. Використовує метод __str__.

tuple (obj) – перетворення до кортежу.

Нижче наведено перелік функцій, які діють на масиви:

Abs(), fabs() – Обчислити абсолютне значення цілих, дійсних або комплексних елементів масиву. Для речових даних fabs працює швидше.

Sqrt() – обчислити квадратний корінь з кожного елемента. Еквівалентно arr**0.5

Square()– обчислити квадрат кожного елемента. Еквівалентно arr ** 2

Exp() – обчислити експоненту e кожного елемента.

Log(), logl0(), log2(), loglp() – натуральний (за основою *e*), десятковий, двійковий логарифм і функція log (1 + x) відповідно.

Sign() – обчислити знак кожного елемента: 1 (для додатних чисел), 0 (для нуля) або -1 (для від'ємних чисел).

Ceil() – обчислити для кожного елемента найменше ціле число, що не менше його.

Floor() – обчислити для кожного елемента найбільше ціле число, не більше його.

Rint() – округлити елементи до найближчого цілого зі збереженням dtype.

Modf() – повернути дробові і цілі частини масиву у вигляді окремих масивів

Isnan() – повернути логічний масив, який показує, які значення є NaN (не числом).

Isfinite(), isinf() – повернути логічний масив, який показує, які елементи є кінцевими (НЕ inf і не NaN) або нескінченними відповідно.

Cos(), cosh(), sin(), tan(), tanh(), sinh() – звичайні і гіперболічні тригонометричні функції.

arccos(), arccosh(), arcsinh(), arctan(), arcsin(), arctanh() – зворотні тригонометричні функції.

logical_not() – обчислити значення істинності not x для кожного елемента. Еквівалентно – arr.

add() – скласти відповідні елементи масивів.

subtract() – відняти елементи другого масиву з відповідних елементів першого.

multiply() – перемножити відповідні елементи масивів.

divide(), floor_divide() – ділення і ділення з відкиданням залишку.

Найбільш вживані функції з модуля numpy. linaig, який підключається до програми на Pyton, якщо вказати його назву:

diag() – повертає діагональні елементи квадратної матриці у вигляді одновимірного масиву або перетворює одновимірний масив в квадратну матрицю, в якій всі елементи, крім що знаходяться на головній діагоналі, дорівнюють нулю.

dot() – обчислює добуток матриць.

trace() – Обчислює слід матриці – суму діагональних елементів.

det() – Обчислює визначник матриці.

Eig() – обчислює власні значення і власні вектори квадратної матриці.

inv() – Обчислює зворотну матрицю.

pinv() – Обчислює псевдообернену матрицю Мура-Пенроуза для квадратної матриці.

qr() – Обчислює QR-розкладання.

svd() – Обчислює сингулярне розкладання (SVD).

solve() – Вирішує лінійну систему Ax = B, де A – квадратная матриця.

lstsq() – Обчислює рішення рівняння $y = xb$ за методом найменших квадратів.

2.7.7. Числа: цілі, дійсні, комплексні

Цілі числа мають тип (int).

Числа в Python 3 нічим не відрізняються від звичайних чисел. Вони підтримують набір звичайнісіньких математичних операцій:

x + y	Додавання
x - y	Віднімання
x * y	Множення
x / y	Ділення
x // y	Ціла частина від ділення
x % y	Решта від ділення
-x	Зміна знаку числа
abs(x)	Модуль числа
divmod(x, y)	Пара (x // y, x % y)
x ** y	Зведення в степінь
pow(x, y[, z])	x^y по модулю (якщо модуль задано)

Також потрібно відзначити, що числа в Python 3, на відміну від багатьох інших мов, підтримують довгу арифметику (проте, це вимагає більше пам'яті). Наприклад:

```
>>> 255 + 34
289
>>> 5 * 2
10
>>> 20/3
6.666666666666667
>>> 20 // 3
6
>>> 20% 3
2
>>> 3 ** 4
81
>>> pow (3, 4)
```

```
81
>>> pow (3, 4, 27)
0
>>> 3 ** 150
369988485035126972924700782451696644186473100389722973815184405301748249
```

Над цілими числами також можна виконувати бітові операції

x | y	Побітове або
x ^ y	Побітове або, що виключає
x & y	Побітове і
x << n	Бітовий зсув уліво
x >> y	Бітовий зсув управо
~x	Інверсія бітів

Функція int.bit_length () – кількість біт, необхідних для представлення числа в двійковому вигляді, без урахування знака і лідируючих нулів.

```
>>> n = -37
>>> bin (n)
'-0b100101'
>>> n.bit_length ()
6
```

Функція int.to_bytes (length, byteorder, *, signed = False) повертає рядок байтів, що представляють це число.

```
>>> (1024) .to_bytes (2, byteorder = 'big') b '\ x04 \ x00'
>>> (1024) .to_bytes (10, byteorder = 'big') b '\ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x00 \ x04 \ x00'
>>> (-1024) .to_bytes (10, byteorder = 'big', signed = True) b '\ xff \ xff \ xff \ xff \ xff \ xff \ xff \ xff \ xfc \ x00'
>>> x = 1000
>>> x.to_bytes ((x.bit_length () // 8) + 1, byteorder = 'little') b '\ xe8 \ x03'
classmethod int.from_bytes (bytes, byteorder, *, signed = False) - повертає число з цього рядка байтів.
>>> int.from_bytes (b '\ x00 \ x10', byteorder = 'big') 16
>>> int.from_bytes (b '\ x00 \ x10', byteorder = 'little') 4096
```

>>> int.from_bytes (b '\ xfc \ x00', byteorder = 'big', signed = True)

Цілі числа мають позначення int, на приклад -1024.

>>> int.from_bytes (b '\ xfc \ x00', byteorder = 'big', signed = False)

64512

>>> int.from_bytes ([255, 0, 0], byteorder = 'big') 16711680

Системи числення у мові Pyton: двійкова, вісімкова, десятична та шістнадцяткова. Якщо потрібно переводити числа з однієї системи числення в іншу, то Python для цього надає кілька функцій:

int ([object], [основа системи числення]) – перетворення до цілого числа в десятковій системі числення. За замовчуванням система числення десяткова, але можна задати будь-яку основу від 2 до 36 включно.

bin(x) – перетворення цілого числа в двійковий рядок.

hex(x) – перетворення цілого числа в шістнадцятковий рядок.

oct(x) – перетворення цілого числа в вісімковий рядок.

Приклади:

```
>>> a = int ('19 ') # Переводимо рядок в число
>>> b = int ('19 .5 ') # Рядок не є цілим числом Traceback (most recent call last): File "", line 1, in
ValueError: invalid literal for int () with base 10: '19 .5 '
>>> c = int (19.5) # Застосована до числа з плаваючою точкою, відсікає дробову частину >>> print (a, c)
19 19 >>> bin (19)
'0b10011'
>>> oct (19)
'0o23'
>>> hex (19)
'0x13'
>>> 0b10011 # Так теж можна записувати числові константи 19
>>> int ( '10011', 2)
19
>>> int ( '0b10011', 2)
19
```

Дійсні числа мають тип float. Дійсні числа підтримують ті ж операції, що і цілі. Однак (через представлення чисел в комп'ютері) дійсні числа неточні, і це може привести до помилок:

```
>>> 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1 + 0.1
0.9999999999999999
```

Для високої точності використовують інші об'єкти (наприклад decimal і fraction).

Також дійсні числа не підтримують більшу розрядність, аніж завжди:

```
>>> a = 3 ** тисячі
>>> a + 0.1
Traceback (most recent call last): File "", line 1, in
OverflowError: int too large to convert to float
```

Простенькі приклади роботи з числами:

```
>>> c = 150
>>> d = 12.9
>>> c + d
162.9
>>> p = abs (d - c) # Модуль числа
>>> print (p)
137.1
>>> round (p) # Округлення 137
```

Додаткові методи роботи з числами представлені наступними функціями:

float.as_integer_ratio() – пара цілих чисел, чиє відношення одне до одного дорівнює цьому числу.

float.is_integer () – чи є значення цілим числом.

float.hex() – переводить float в hex (шістнадцятирічну систему числення).

classmethod float.fromhex(s) – перетворення float з шістнадцяткового числа.

```
>>> (10.5) .hex ()
'0x1.5000000000000p + 3'
>>> float.fromhex ( '0x1.5000000000000p + 3')
10.5
```

Крім стандартних виразів для роботи з числами (а в Python їх не так вже й багато), в складі Python є кілька корисних модулів.

Модуль **math** надає більш складні математичні функції.

```
>>> import math
>>> math.pi
3.141592653589793
>>> math.sqrt (85)
9.219544457292887
```

Модуль **random** реалізує генератор випадкових чисел і функції випадкового вибору.

```
>>> import random
>>> random.random ()
```

2.7.8. Робота з рядками в Python. Літерали

Рядки в Python – впорядковані послідовності символів, що використовуються для зберігання і представлення текстової інформації, тому за допомогою рядків можна працювати з усім, що може бути представлено в текстовій формі.

Рядки можуть бути представлені в апострофах і в лапках

```
S = 'spam "s'
S =" spam's "
```

Рядки в апострофах і в лапках – одне і те ж. Причина наявності двох варіантів в тому, щоб дозволити вставляти в літерали рядків символи лапок і апострофів, не використовуючи екранування.

Екрановані послідовності дозволяють вставити символи, які складно ввести з клавіатури.

Екранована послідовність	Призначення
\n	Перевод рядка
\a	Дзвінок
\b	Повернення курсора рядку на одну позицію ліворуч
\f	Перевод сторінки
\r	Повернення каретки
\t	Горизонтальна табуляція
\v	Вертикальна табуляція
\N{id}	Ідентифікатор ID бази даних Юнікоду
\uhhhh	16-бітовий символ Юнікоду в 16-річному представленні
\Uhhhh. . .	32-бітовий символ Юнікоду в 32-річному представленні
\xhh	16-річне значення символу
\ooo	8- річне значення символу
\0	Символ Null (не є ознакою кінця рядку)

Якщо перед лапками, що відкривають рядок символів, стоїть символ 'r' (в будь-якому регістрі), то механізм подачі результатів роботи команди на екран відключається.

S = r'C:\newt.txt '

Але, незважаючи на призначення, "сирий" рядок не може закінчуватися символом зворотного слеша. Шляхи вирішення:

S = r '\ n \ n \\' [: - 1]
S = r '\ n \ n' + '\\'
S = '\\ n \\ n'

Головна перевага рядків в потрійних лапках в тому, що їх можна використовувати для запису багаторядкових блоків тексту. Усередині такого рядка можлива присутність лапок і апострофів, головне, щоб не було трьох лапок поспіль.

```
>>> c = " 'це дуже великий рядок, багатостроковий блок тексту' "
>>> c
'Це дуже великий \ nрядок, багатостроковий \ nблок тексту'
>>> print (c) це дуже великий рядок, багатостроковий блок тексту
```

Основні операції з рядками тексту наступні:

Конкатенація (додавання)

```
>>> S1 = 'spam'
>>> S2 = 'eggs'
>>> print (S1 + S2)
'Spameggs'
дублювання рядки
>>> print ( 'spam' * 3) spamspamspam
Довжина рядка (функція len)
>>> len ( 'spam') 4
Доступ за індексом
>>> S = 'spam'
>>> S [0]
'S'
>>> S [2]
'A'
>>> S [-2]
'A'
```

Як видно з прикладу, в Python є можливість доступу по від'ємному індексу, при цьому відлік йде від кінця рядка.

З текстового рядка можні витягнути якусь його частину, або як кажуть, «витягнути зріз рядка».

Оператор вилучення зрізу: [X: Y]. X – початок зрізу, а Y – закінчення; символ з номером Y в зріз не входить. За замовчуванням перший індекс дорівнює 0, а другий – довжині рядка.

```
>>> s = 'spameggs'
>>> s [3: 5]
'Me'
>>> s [2: -2]
'Ameg'
>>> s [: 6]
'Spameg'
>>> s [1:]
'Pameggs'
>>> s [:]
'Spameggs'
```

Крім того, можна задати крок, з яким потрібно витягувати зріз.

```
>>> s [:: - 1]
```

```
'Sggemaps'
>>> s [3: 5: -1]
''
>>> s [2 :: 2]
'Aeg'
```

При виклику методів необхідно пам'ятати, що рядки в Python відносяться до категорії незмінних послідовностей, тобто всі функції і методи можуть лише створювати новий рядок.

```
>>> s = 'spam'
>>> s [1] = 'b'
Traceback (most recent call last):
File "", line 1, in s [1] = 'b'
TypeError: 'str' object does not support item assignment
>>> s = s [0] + 'b' + s [2:]
>>> s
'Sbam'
```

Повний набір функцій та методів обробки рядків представлено у наступній таблиці

Функція або метод	Призначення
S = ‘str’; S = “str”; S = ‘’‘str’‘’; S = “”“str”“”	Літерали рядків
S = “s\np\ta\nbbb”	Екрановані послідовності
S = r”C:\temp\new”	Неформатованні рядки (пригнічують екранування)
S = b”byte”	Рядок байтів
S1 + S2	Конкатенація (складання рядків)
S1 * 3	Повтор рядків
S[i]	Звернення по індексу
S[i:j:step]	Витяг зрізу
len(S)	Довжина рядка
S.find(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого вхождення або 1
S.rfind(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього вхождення або 1
S.index(str, [start],[end])	Пошук підрядка в рядку. Повертає номер першого вхождення або викликає ValueError

Функція або метод	Призначення
S.rindex(str, [start],[end])	Пошук підрядка в рядку. Повертає номер останнього вхождення або викликає ValueError
S.replace(шаблон, замена)	Заміна шаблону
S.split(символ)	Разбиття рядку по розділителю
S.isdigit()	Визначає, чи складається рядок із цифр
S.isalpha()	Визначає, чи складається рядок із букв
S.isalnum()	Визначає, чи складається рядок із цифр чи з букв
S.islower()	Визначає, чи складається рядок із символів в нижньому регістрі
S.isupper()	Визначає, чи складається рядок із символів в верхньому регістрі
S.isspace()	Визначає, чи складається рядок із символів, що не відображаються в тексті (пробіл, символ переводу сторінки (‘\f’), “новий рядок” (‘\n’), “перевод каретки” (‘\r’), “горизонтальна табуляція” (‘\t’) и “вертикальна табуляція” (‘\v’))
S.istitle()	Визначає, чи починаються слова в рядку з заглавної букви
S.upper()	Перетворення рядку до верхнього регістру
S.lower()	Перетворення рядку до нижнього регістру
S.startswith(str)	Визначає, чи починаються рядок S з шаблону str
S.endswith(str)	Визначає, чи закінчується рядок S з шаблону str
S.join(список)	Збирання рядку зі списку з розділювачем S
ord(символ)	Перевод символу у код ASCII
chr(число)	Перевод коду ASCII у символ
S.capitalize()	Перевод першого символу рядка у верхній регистр, а всі інших – у нижній
S.center(width, [fill])	Повертає відцентрований рядок, по краям якого стоїть символ fill (пробіл за замовчуванням)
S.lstrip([chars])	Видаляє символи пробілів на початку рядка
S.rstrip([chars])	Видаляє символи пробілів у кінці рядка
S.strip([chars])	Видаляє символи пробілів на початку та у кінці рядка
S.swapcase()	Переводить символи нижнього регистру у верхній, а верхнього – у нижній
S.title()	Першу букву кожного слова переводить у верхній регистр, а всі інші – у нижній

Досить часто виникають ситуації, коли потрібно зробити рядок, підставивши в нього деякі дані, отримані в процесі виконання програми (призначене для користувача введення, дані з файлів і т.д.). Підстановку даних можна зробити за допомогою форматування рядків. Форматування можна зробити за допомогою оператора %, і методу format.

Форматування рядків за допомогою методу *format* виконується, якщо для підстановки потрібно тільки один аргумент, то значення – сам аргумент:

```
>>> 'Hello, {}!'. Format ( 'Vasya')
'Hello, Vasya!'
```

А якщо кілька, то значеннями будуть усі аргументи з рядками підстановки (звичайних або іменованих):

```
>>> '{0}, {1}, {2}'. Format( 'a', 'b', 'c')
'A, b, c'
>>> '{}, {}, {}'. Format( 'a', 'b', 'c')
'A, b, c'
>>> '{2}, {1}, {0}'. Format( 'a', 'b', 'c')
'C, b, a'
>>> '{2}, {1}, {0}'. Format(* 'abc')
'C, b, a'
>>> '{0} {1} {0}'. Format( 'abra', 'cad')
'Abracadabra'
>>> 'Coordinates: {latitude}, {longitude}'. Format (latitude = '37
.24N ', longitude =' - 115.81W ')
'Coordinates: 37.24N, -115.81W'
>>> coord = { 'latitude': '37 .24N ',' longitude ':' -115.81W '}
>>> 'Coordinates: {latitude}, {longitude}'. Format (** coord)
'Coordinates: 37.24N, -115.81W'
```

Однак метод format вміє більше. Ось його синтаксис:

поле заміни :: = "{" [ім'я поля] ["!" перетворення] [":" специфікація] "}"

ім'я поля :: = arg_name ("." ім'я атрибута | "[" індекс "]") *

перетворення :: = "r" (внутрішнє уявлення) | "S" (людське уявлення) специфікація :: =

Наприклад:

```
>>> "Units destroyed: {players [0]}". Format (players = [1, 2, 3])
'Units destroyed: 1'
>>> "Units destroyed: {players [0]! R}". Format (players = [ '1', '2',
'3'])
"Units destroyed: '1'"
```

Тепер специфікація формату:

специфікація :: = [[fill] align] [sign] [#] [0] [width] [,] [. precision] [type]
заповнювач :: = символ крім '{' або '}' вирівнювання :: = "<" | ">" | "=" | "^"
знак :: = "+" | "-" | "" Ширина :: = integer точність :: = integer
тип :: = "b" | "C" | "D" | "E" | "E" | "F" | "F" | "G" | "G" |"N" | "O" | "S" | "X" | "X" | "%"

Вирівнювання проводиться за допомогою символу-заповнювача. Доступними параметрами вирівнювання є:

Флаг	Значення
‘<’	Символи, що заповнюють, будуть праворуч (вирівнювання об’єкту по лівому краю) (за замовчуванням).
‘>’	Вирівнювання об’єкту по правому краю.
‘=’	Заповновач буде після знаку, але перед цифрами. Працює тільки з числовими типами.
‘^’	Вирівнювання по центру.

Опція “знак” використовується тільки для чисел і може приймати наступні значення:

Флаг	Значення
‘+’	Знак має бути використаний для всіх чисел.
‘-‘	‘-‘ для від’ємних, нічого для для додатних,
‘Пробіл’	‘-‘для від’ємних, пробіл для для додатних,

Поле “тип” може приймати наступні значення:

Тип	Значення
‘d’, ‘i’, ‘u’	Десяткове число.
‘o’	Число в восьмеричній системі счислення.
‘x’	Число в шестнадцятковій системі числення (букви у нижньому регістрі).
‘X’	Число в шестнадцятковій системі числення (букви у верхньому регістрі).

‘e’	Число з плаваючою точкою з експонентою (експонента у нижньому регістрі).
‘E’	Число з плаваючою точкою з експонентою (экспонента у верхньому регістрі).
‘f’, ‘F’	Число з плаваючою точкою (зивчайний формат).
‘g’	Число з плаваючою точкою з експонентою (експонента у нижньому регістрі), якщо воно меньше, за -4 або точність, інакше – звичайний формат.
‘G’	Число з плаваючою точкою з експонентою (экспонента у верхньому регістрі, якщо воно меньше, за -4 або точність, інакше – звичайний формат.
‘c’	Символ (рядок із из одного символу або число – код символу).
‘s’	Рядок.
‘%’	Число множиться на 100, відображається число з плаваючою точкою, а за ним знак %.

Наведемо декілька прикладів, для пояснення того, як використаовувати прийомии форматування тесту:

```
>>> coord = (3, 5)
>>> 'X: {0 [0]}; Y: {0 [1]} '. Format (coord)
'X: 3; Y: 5 '
>>> "repr () shows quotes: {! R}; str () does not: {! S}". Format
( 'test1', 'test2') "repr () shows quotes: 'test1'; str () does not: test2
"
>>> '{: <30}'. Format ( 'left aligned')
'Left aligned'
>>> '{:> 30}'. Format ( 'right aligned')
'Right aligned'
>>> '{: ^ 30}'. Format ( 'centered')
'Centered'
>>> '{: * ^ 30}'. Format ( 'centered') # use '*' as a fill char
'********** centered **********'
>>> '{: + f}; {: + F} '. Format (3.14, -3.14) # show it always
'+3.140000; -3.140000 '>>>' {: f}; {: F} '. Format (3.14, -3.14) #
show a space for positive numbers
'3.140000; -3.140000 '>>>' {: -f}; {: -f} '. Format (3.14, -3.14) #
show only the minus - same as' {: f}; {: F} '
'3.140000; -3.140000 '
>>> # format also supports binary numbers
>>> "int: {0: d}; hex: {0: x}; oct: {0: o}; bin: {0: b}". Format
(42)
```

```
'Int: 42; hex: 2a; oct: 52; bin: 101010 '
>>> # with 0x, 0o, or 0b as prefix:
>>> "int: {0: d}; hex: {0: #x}; oct: {0: #o}; bin: {0: #b}". Format
(42)
'Int: 42; hex: 0x2a; oct: 0o52; bin: 0b101010 '
>>> points = 19.5
>>> total = 22

>>> 'Correct answers: {: .2%}'. Format (points / total)
'Correct answers: 88.64%'
```

2.7.9. Функції і методи списків

Списки в Python – це впорядковані змінювані колекції об'єктів довільних типів (майже як масив, але типи можуть відрізнятися).

Щоб використовувати списки, їх потрібно створити. Створити список можна декількома способами. Наприклад, можна обробити будь-який об'єкт (наприклад, рядок) вбудованою функцією list:

```
>>> list( 'список')
['список']
```

Список можна створити і за допомогою литералу:

```
>>> s = [] # Порожній список
>>> l = [ 's', 'p', [ 'isok'], 2]
>>> s
[]
>>> l
[ 'S', 'p', [ 'isok'], 2]
```

Як видно з прикладу, список може містити будь-яку кількість будь-яких об'єктів (в тому числі і вкладені списки), чи не містити нічого.

І ще один спосіб створити список – це генератори списків. Генератор списків – це спосіб побудувати новий список, застосовуючи вираз до кожного елементу послідовності. Генератори списків дуже схожі на цикл for.

```
>>> c = [c * 3 for c in 'list']
```

```
>>> c
[ 'Lll', 'iii', 'sss', 'ttt']
```

Можлива і більш складна конструкція генератора списків:

```
>>> c = [c * 3 for c in 'list' if c! = 'I']
>>> c
[ 'Lll', 'sss', 'ttt']
>>> c = [c + d for c in 'list' if c! = 'I' for d in 'spam' if d! = 'A']
>>> c
[ 'Ls', 'lp', 'lm', 'ss', 'sp', 'sm', 'ts', 'tp', 'tm']
```

Але в складних випадках краще користуватися звичайним циклом for для генерації списків.

Для списків доступні основні вбудовані функції, а також методи списків:

Метод	Що робить
list.append(x)	Додає елемент у кінець списку
list.extend(L)	Разширяє список list, додаючи в кінець всі елементи списку L
list.insert(i, x)	Вставляє на *i*-ий элемент значеннння *x*
list.remove(x)	Видаляє перший елемент у списку, що має значення *x*
list.pop([i])	Видаляє *i*-ий елемент і повертає його. Якщо індекс не вказано, видаляеться останній елемент
list.index(x, [start [, end]])	Повертає положення першого елементу від start до end зі значением *x*
list.count(x)	Повертає кількість елементів зі значенням *x*
list.sort([key = функция])	Сортує список на основі функції
list.reverse()	Рогортає список
list.copy()	Поверхнева копія списку
list.clear()	Очищає список

Потрібно відзначити, що методи списків, на відміну від рядкових методів, змінюють сам список, а тому результат виконання не потрібно записувати в цю змінну. Наприклад:

```
>>> l = [1, 2, 3, 5, 7]
>>> l.sort ()
>>> l
```

```
[1, 2, 3, 5, 7]
>>> l = l.sort ()
>>> print (l) None
>>> a = [66.25, 333, 333, 1, 1234.5]
>>> print (a.count (333), a.count (66.25), a.count ('x'))
2 1 0
>>> a.insert (2, -1)
>>> a.append (333)
>>> a
[66.25, 333, -1, 333, 1, 1234.5, 333]
>>> a.index (333)
1
>>> a.remove (333)
>>> a
[66.25, -1, 333, 1, 1234.5, 333]
>>> a.reverse ()
>>> a
[333, 1234.5, 1, 333, -1, 66.25]
>>> a.sort ()
>>> a
[-1, 1, 66.25, 333, 333, 1234.5]
```

2.7.10. Індекси і зрізи

Як і в інших мовах програмування, взяття за індексом означає, вказати номер рядка та стовпця у масиві:

```
>>> a = [1, 3, 8, 7]
>>> a [0]
1
>>> a [3]
7
>>> a [4]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Як і в багатьох інших мовах, нумерація елементів починається з *нуля*. При спробі доступу до неіснуючого індексу виникає виняток *IndexError*.

В даному прикладі змінна *a* була списком, однак взяти елемент за індексом можна і у інших типів: рядків, кортежів.

В Python також підтримуються від'ємні індекси, при цьому нумерація йде з кінця, наприклад:

```
>>> a = [1, 3, 8, 7]
>>> a [-1]
7
>>> a [-4]
1
>>> a [-5]
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

В Python, крім індексів, існують ще й зрізи:

item [START: STOP: STEP] – бере зріз від номера START, до STOP (не включаючи його), з кроком STEP. За замовчуванням START = 0, STOP = довжині об'єкта, STEP = 1. Відповідно, якісь (а можливо, і всі) параметри можуть бути опущені.

```
>>> a = [1, 3, 8, 7]
>>> a [:]
[1, 3, 8, 7]
>>> a [1:]
[3, 8, 7]
>>> a [: 3]
 [1, 3, 8]
>>> a [:: 2]
[1, 8]
```

Також всі ці параметри можуть бути і від'ємними:

```
>>> a = [1, 3, 8, 7]
>>> a [:: - 1]
[7, 8, 3, 1]
>>> a [: - 2]
[1, 3]
>>> a [-2 :: - 1]
[8, 3, 1]
```

```
>>> a [1: 4: -1]
 []
```

В останньому прикладі вийшов порожній список, так як START <STOP, а STEP від’ємний. Те ж саме відбудеться, якщо діапазон значень виявиться за межами об'єкта:

```
>>> a = [1, 3, 8, 7]
>>> a [10:20]
 []
```

Також за допомогою зрізів можна не тільки отримувати елементи, але і додавати і видаляти елементи (зрозуміло, тільки для змінних послідовностей).

```
>>> a = [1, 3, 8, 7]
>>> a [1: 3] = [0, 0, 0]
>>> a
[1, 0, 0, 0, 7]
>>> del a [: - 3]
>>> a
[0, 0, 7]
```

2.7.11. Кортежі (tuple)

Кортеж, по суті – це незмінний список.

Навіщо потрібні кортежі, якщо є списки? Ось приклади з поясненнями:

1. Захист від дурня. Тобто кортеж захищений від змін, як навмисних (що погано), так і випадкових (що добре).

2. Менший розмір:

```
>>> a = (1, 2, 3, 4, 5, 6)
>>> b = [1, 2, 3, 4, 5, 6]
>>> a .__ sizeof __ ()
36
>>> b .__ sizeof __ ()
44
```

3. Можливість використовувати кортежі в якості ключів словника:

```
>>> d = {(1, 1, 1): 1}
```

```
>>> d
{(1, 1, 1): 1}
>>> d = {[1, 1, 1]: 1}
Traceback (most recent call last):
File "", line 1, in d = {[1, 1, 1]: 1}
TypeError: unhashable type: 'list'
```

З кортежами працювати приблизно так само, як і зі списками.

Створюємо порожній кортеж:

```
>>> a = tuple () # За допомогою вбудованої функції
tuple ()
>>> a
()
>>> a = () # За допомогою літералу кортежу
>>> a
()
>>>
```

Створюємо кортеж з одного елемента:

```
>>> a = ( 's')

>>> a

's'
```

Але в цьому випадку вийшов рядок. Як отримати кортеж? Наступним чином:

```
>>> a = ( 's',)
>>> a
( 'S',)
```

Вся справа у комі після символу *s*. Самі по собі дужки нічого не значать, точніше, означають те, що всередині них знаходиться одна інструкція, яка може бути відділена пробілами, перенесенням рядків та ін. Кортеж можна створити і так:

```
>>> a = 's',

>>> a
( 'S',)
```

Але все-таки не захоплюйтеся, і ставте дужки, тим більше, що бувають випадки, коли дужки необхідні.

Ну і створити кортеж з об'єкту можна за допомогою все тієї ж функції tuple()

```
>>> a = tuple( 'hello, world!')
>>> a
( 'H', 'e', 'l', 'l', 'o', ',', '', 'w', 'o', 'r', 'l', 'd', '! ')
```

Всі операції над списками, що не змінюють список (додавання, множення на число, методи index() і count() і деякі інші операції). Можна також по-різному змінювати елементи місцями і так далі.

Наприклад, гордість програмістів на Python – поміняти місцями значення двох змінних:

```
a, b = b, a
```

2.7.12. Словники (dict) і робота з ними. Методи словників

Словники в Python – це невпорядковані колекції довільних об'єктів з доступом по ключу. Їх іноді ще називають асоціативними масивами або хеш-таблицями. На відміну від списків, які є впорядкованими послідовностями елементів довільного типу, елементи словника - це невпорядковані послідовності пар **{ключ: значення}.** Іноді словники називають асоціативними масивами, іноді відображеннями (мається на увазі відображення безлічі ключів словника на безліч його значень). Як і списки, словники мають змінну довжину, довільну вкладеність і можуть зберігати значення довільних типів.

Щоб працювати зі словником, його потрібно створити. Створити його можна кількома способами. Поперше, за допомогою літералу:

```
>>> d = {}
>>> d
{}
>>> d = { 'dict': 1, 'dictionary': 2}
>>> d
{ 'Dict': 1, 'dictionary': 2}
```

По-друге, за допомогою функції dict:

```
>>> d = dict (short = 'dict', long = 'dictionary')
>>> d
```

```
{ 'Short': 'dict', 'long': 'dictionary'}
>>> d = dict ([(1, 1), (2, 4)])
>>> d
{1: 1, 2: 4}
```

По-третє, за допомогою методу fromkeys:

```
>>> d = dict.fromkeys ([ 'a', 'b'])
>>> d
{ 'A': None, 'b': None}
>>> d = dict.fromkeys ([ 'a', 'b'], 100)
>>> d
{ 'A': 100, 'b': 100}
```

По-четверте, за допомогою генераторів словників, які дуже схожі на генератори списків.

```
>>> d = {a: a ** 2 for a in range (7)}
>>> d
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36}
```

Тепер спробуємо додати записів в словник і витягти значення ключів:

```
>>> d = {1: 2, 2: 4, 3: 9}
>>> d [1]
2
>>> d [4] = 4 ** 2
>>> d
{1: 2, 2: 4, 3: 9, 4: 16}
>>> d [ '1']
Traceback (most recent call last):
File "", line 1, in d [ '1']
KeyError: '1'
```

Ось короткий фрагмент програми, в якому ввождиться слово запиту, і даєтьтся пояснення цього терміну зі словника.

d={'Сюрвейр':'експерт, що здійснює на прохання чи страхувальника страховика огляд судів і вантажів і, той, хто дає висновок про їхній стан','Тант*єма':'Комісія (винагорода) із прибутку перестрахувальника перестрахувальнику за надання участі в перестрахувальних договорах. Ціна страхового ризику й інших витрат, адекватне грошове вираження зобов*язань страховика за укладеним договором страхування.','Ризик':'це конкретне явище чи

сукупність явищ (страховачи подія чи сукупність подій), потенційна можливість заподіяння збитку об*єкту страхування.'}

```
q=input('Введіть запитання')
print(d[q])
```

Як видно з прикладів, присвоєння по новому ключу розширює словник, присвоєння за існуючим ключем перезаписує його, а спроба вилучення неіснуючого ключа породжує виключення. Для уникнення виключення є спеціальний метод або можна перехоплювати виняток.

Методи словників описані наступними функціями:

dict.clear() – очищає словник.

dict.copy() – повертає копію словника.

classmethod dict.fromkeys(seq [, value]) – створює словник з ключами з seq і значенням value (за замовчуванням None).

dict.get(key [, default]) – повертає значення ключа, але якщо його немає, не видає виняток, а повертає default (за замовчуванням None).

dict.items() – повертає пари (ключ, значення).

dict.keys() – повертає ключі в словнику.

dict.pop(key [, default]) – видаляє ключ і повертає значення. Якщо ключа немає, повертає default (за замовчуванням видає виняток).

dict.popitem() – видаляє і повертає пару (ключ, значення). Якщо словник порожній, видає виняток KeyError. Пам'ятайте, що словники невпорядковані.

dict.setdefault(key [, default]) – повертає значення ключа, але якщо його немає, не видає виняток, а створює ключ з значенням default (за замовчуванням None).

dict.update ([other]) – оновлює словник, додаючи пари (ключ, значення) з other.

dict.values () – повертає значення в словнику.

Іще декілька прикладів роботи зі словниками.

При переборі значень словників, ключі та відповідні значення можуть отримуватися за допомогою методу `iteritems()`:

```
>>> knights = {'Ґаллагад': 'Чистий', 'Робін': 'Хоробрий'}
>>> for k, v in knights.iteritems():
...     print k, v
...
Ґаллагад Чистий
Робін Хоробрий
```

Для словників визначена операція злиття, яка реалізована в методі ***update:***

```
phone1 = { ’ Dan ’ : ’ 7926765431 ’ , ’ John ’ : ’ 74881234567 ’ }
phone2 = { ’ Daniel ’ : ’ 7926765431 ’ , ’ John ’ : ’ 74775566221 ’ }
phone1. update ( phone2 )
print ( phone1 )
>>> { ’ Daniel ’ : ’79267654 31’ , ’ John ’: ’ 74775566221 ’ , ’Dan’: ’ 7926765431’ }
```

Як бачимо з цього прикладу, при злитті дані, що повторюються (для імені John), зливаються в один елемент.

Інший корисний метод при роботі зі словниками - метод ***get (key, default).*** Сенс його в наступному: при зверненні до значення неіснуючого ключа можна передбачити значення за замовчуванням, яке можна використовувати.

```
res = { ’ Dan ’ : 5 , ’ John ’:4 , ’ Mary ’ : 4 , ’ Claire ’:3 , ’ Maggie ’:5 }
print ( res.get ( ’ Dan ’ , 2 ))  # 5
print ( res.get ( ’Mike ’, 2))   # 2
```

2.7.13. Множини (set і frozenset)

Множина в Python – "контейнер", що містить елементи, які не повторюються і розташовані у випадковому порядку.

Створюємо множини:

```
>>> a = set ()
>>> a set ()
>>> a = set ( 'hello')
>>> a
{ 'H', 'o', 'l', 'e'}
>>> a = { 'a', 'b', 'c', 'd'}
>>> a
{ 'B', 'c', 'a', 'd'}
>>> a = {i ** 2 for i in range (10)} # генератор множин
>>> a
{0, 1, 4, 81, 64, 9, 16, 49, 25, 36}
>>> a = {} # А так не можна!
>>> type (a)
<Class 'dict'>
```

Як видно з прикладу, множини мають той же літерал, що і словники, але порожню множина за допомогою літералу створити не можна.

Множини зручно використовувати для видалення повторюваних елементів:

```
>>> words = [ 'hello', 'daddy', 'hello', 'mum']
>>> set (words)
{ 'Hello', 'daddy', 'mum'}
```

З множинами можна виконувати безліч операцій: знаходити об'єднання, перетин, тощо. Ось перелік осмновних функцій, що обробляють множини:

len(s) – число елементів у множині (розмір множини).

x in s – перевірка, чи належить *x* множині *s*.

set.isdisjoint (other) – істина, якщо set і other не мають спільних елементів. Якщо set = other – всі елементи set належать other, інакше, всі елементи other належать set.

set.issubset (other) або set <= other – всі елементи set належать other.

set.issuperset (other) або set> = other – аналогічно. set.union (other, ...) або set | other | ... – об'єднання декількох множин. set.intersection (other, ...) або set & other & ... – перетин.

set.difference (other, ...) або set – other – ... – множина усіх елементів set, які не належать жодному з other.

set.symmetric_difference (other);

set.other – множина з елементів, що зустрічаються в одній множин, але не зустрічаються в обох.

set.copy () – копія множини.

set.update (other, ...); set | = other | ... – об'єднання.

set.intersection_update (other, ...); set & = other & ... – перетин.

set.difference_update (other, ...); set - = other | ... – віднімання.

set.symmetric_difference_update (other); set ^ = other set.add (elem) – додавання елемента в множину.

set.remove (elem) – видаляє елемент із множини або видає. KeyError, якщо такого елемента не існує.

set.discard (elem) – видаляє елемент, якщо він знаходиться у множині.

set.pop () – видаляє перший елемент з множини. Так як множини не впорядковані, не можна точно сказати, який елемент буде першим.

set.clear () – очищення множини.

Єдина відмінність set від frozenset полягає в тому, що set – змінюваний тип даних, а frozenset – ні. Приблизно схожа ситуація зі списками і кортежами.

```
>>> a = set ( 'qwerty')
>>> b = frozenset ( 'qwerty')
>>> a == b
True
>>> type (a - b)
>>> type (a | b)
>>> a.add (1)
  >>> b.add (1)
Traceback (most recent call last):
File "", line 1, in
b.add (1)
```

AttributeError: 'frozenset' object has no attribute 'add'

2.7.14. Функції та їх аргументи

Функція в Python – це об'єкт, який приймає аргументи і повертає значення Зазвичай функція визначається за допомогою інструкції def.

Визначимо найпростішу функцію:

```
def add (x, y):
    return x + y
```

Інструкція return каже, що потрібно повернути якесь значення. В нашому випадку функція повертає суму x та y. Зверніть увагу на те, що після оператора def стоїть двокрапка, тобто, всі оператори, які виконуються в межах фукнкції мають бути зсунуті праворуч, щоб позначити те, що вони є внутрішнімі для цієї функції. Це стосується і оператора return.

Іще однією особливістю використання функцій в Pyton є вимога розсташування функцій перед основними операторами, що їх використовують, наприклад

```
def a(f):
   f+=1
   return f
z=2
z=z+a(z)
print (z)
```

Тут спочатку визначається функція a(f), що закінчується операторм return. Потім ідуть основні оператори. В результаті ми отримаємо число 5.

Тепер ми її можемо викликати:

```
>>> add (1, 10)
11
>>> add ( 'abc', 'def')
'Abcdef'
```

Функція може бути будь-якої складності і повертати будь-які об'єкти (списки, кортежі, і навіть функції!):

```
>>> def newfunc (n):
def myfunc (x): return x + n return myfunc
>>> new = newfunc (100)
>>> new (200)
300
>>> # new - це функція
```

Функція може і не закінчуватися інструкцією return, при цьому функція поверне значення None:

```
>>> def func (): pass
>>> print (func ())
None
>>>
```

Функція може приймати будь-яку кількість аргументів чи не приймати їх зовсім. Також поширені функції з довільним числом аргументів, функції з позиційними і іменованими аргументами, обов'язковими і необов'язковими.

```
>>> def func(a, b, c = 2): # c – необов'язковий аргумент
return a + b + c
>>> func (1, 2) # a = 1, b = 2, c = 2 (за замовчуванням)
5
>>> func (1, 2, 3) # a = 1, b = 2, c = 3
6
>>> func (a = 1, b = 3) # a = 1, b = 3, c = 2
6
>>> func (a = 3, c = 6) # a = 3, c = 6, b не визначений Traceback
(most recent call last):
File "", line 1, in func (a = 3, c = 6)
TypeError: func () takes at least 2 arguments (2 given)
```

Функція також може приймати змінну кількість позиційних аргументів, тоді перед ім'ям ставиться *:

```
>>> def func(* args): return args
>>> func(1, 2, 3, 'abc')
(1, 2, 3, 'abc')
>>> func ()
```

```
()
>>> func(1) (1,)
```

Як видно з прикладу, args – це кортеж з усіх переданих аргументів функції, і зі змінною можна працювати так само, як і з кортежем.

Функція може приймати і довільне число іменованих аргументів, тоді перед ім'ям ставиться **:

Приклад аргументів функції:

```
>>> def func(** kwargs): return kwargs
>>> func(a = 1, b = 2, c = 3)
{ 'A': 1, 'c': 3, 'b': 2}
>>> func ()
{}
>>> func(a = 'python')
{ 'A': 'python'} >>>
```

В змінній kwargs у нас зберігається словник, з яким ми, знову-таки, можемо робити все, що нам заманеться.

Анонімні функції можуть містити лише один вислів, але виконуються вони швидше. Анонімні функції створюються за допомогою інструкції lambda.

Крім цього, їх не обов'язково привласнювати змінній, як робили ми інструкцією def func ():

```
>>> func = lambda x, y: x + y
>>> func(1, 2)
3
>>> func( 'a', 'b')
'Ab'
>>> (lambda x, y: x + y) (1, 2)
3
>>> (lambda x, y: x + y) ( 'a', 'b')
'Ab'
```

lambda функції, на відміну від звичайної, не потрібно інструкція return, а в іншому, поводиться точно так же:

```
>>> func = lambda * args: args
>>> func(1, 2, 3, 4)
```

(1, 2, 3, 4) >>>

2.7.15. Файли. Робота з файлами

Перш, ніж працювати з файлом, його треба відкрити. З цим чудово впорається вбудована функція open:

f = open ('text.txt', 'r')

У функції open багато параметрів, нам поки важливі три аргументи: перший, це ім'я файлу. Шлях до файлу може бути відносним або абсолютним. Другий аргумент, це режим, в якому ми будемо відкривати файл.

Ре-жим	Позначення
‘r’	Відкривання для читання (є значенням за замовчуванням).
‘w’	Відкривання для запису, зміст файлу видаляється, якщо файлу не існує, тоді створюється новий.
‘x’	Відкривання для запису, якщо файлу не існує, інакше – виключення.
‘a’	Відкривання для дозапису, інформація додається у кінець файлу.
‘b’	Відкривання для перегляду у двоїчному режимі.
‘t’	Відкривання для перегляду в текстовому режимі (є значенням за замовчуванням).
‘+’	Відкривання для читання й запису

Режими можуть бути об'єднані, тобто, наприклад, 'rb' – читання в двійковому режимі. За замовчуванням режим дорівнює 'rt'.

І останній аргумент, encoding, потрібен тільки в текстовому режимі читання файлу. Цей аргумент задає кодування.

Відкрили ми файл, а тепер ми хочемо прочитати з нього інформацію. Для цього є кілька способів, але великого інтересу заслуговують лише два з них.

Перший – метод read, читає весь файл цілком, якщо був викликаний без аргументів, і *n* символів, якщо був викликаний з аргументом (цілим числом *n*).

>>> f = open ('text.txt')

```
>>> f.read (1)
'H'
>>> f.read ()
'Ello world! \ NThe end. \ N \ n'
```

Ще один спосіб зробити це – прочитати файл через підрядник, скориставшись циклом for:

```
>>> f = open ( 'text.txt')
>>> for line in f:
line
'Hello world! \ N'
'\ N'
'The end. \ N'
'\ N'
```

Тепер розглянемо запис в файл. Спробуємо записати в файл ось такий список:

```
>>> l = [str (i) + str (i-1) for i in range (20)]
>>> l
[ '0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110' ,
'1211', '1312', '1413']
```

Відкриємо файл на запис:

```
>>> f = open ( 'text.txt', 'w')
```

Запис в файл здійснюється за допомогою методу write:

```
>>> for index in l:
f.write (index + '\ n')
4
3
3
3
3
```

Для тих, хто не зрозумів, що це за цифри, поясню: метод write повертає число записаних символів.

Після закінчення роботи з файлом його обов'язково потрібно закрити за допомогою методу close:

```
>>> f.close ()
```

Тепер спробуємо відтворити цей список з отриманого файлу. Відкриємо файл на читання і прочитаємо рядки.

```
>>> f = open ( 'text.txt', 'r')
>>> l = [line.strip () for line in f]
>>> l
[ '0-1', '10', '21', '32', '43', '54', '65', '76', '87', '98', '109', '1110'
, '1211', '1312', '1413
>>> f.close ()
',
```

Ми отримали той же список, що і був. У більш складних випадках (словниках, вкладених кортежів і т. д.), алгоритм запису придумати складніше. Але це і не потрібно. У Python вже давно придумали засоби такі як pickle або json, що дозволяють зберігати у файлі складні структури.

Конструкція with ... as використовується для повернення виконання блока інструкцій менеджером контексту. Інколи це більш зручна конструкція, ніж try...except...finally.

Синтаксис конструкции with ... as:

"with" expression ["as" target] ("," expression ["as" target])* ":" suite

Теперь по порядку того, что відбувається при виконаннні даного блоку:

1. Виконується вираз в конструкції with ... as.

2. Завантажуется спеціальний метод __exit__ для подальшого використання.

3. Виконується метод __enter__. Якщо конструкція включає в себе слово as, то методом __enter__ значення, що повебртається, записується в змінну.

4. Виконується комплект.

5. Викликається метод __exit__, причому не має значення, чи виконана Suite чи сталося вимкнення. В цей метод передаються параметри вимкнення, якщо воно відбулося, або у всіх аргументах значення Non, якщо не було виключень.

Якщо в конструкції with - as було декілька виразів, то це еквівалентно декільком вкладеним конструкціям:

```
with A() as a, B() as b:
suite
```

Еквівалентно:

```
with A() as a:
with B() as b:
suite
```

Для чого застосовується конструкція with - as? Для гарантування того, що критичні функції виконуються в будь-якому випадку. Самий розповсюджений приклад використання цієї конструкції – відкриття файлів. зпзвичай відкриття файлів здійснюється за допомогою функції open, але конструкція with - as, як правило, є більш зручною і гарантує закриття файлу в будь-якому випадку.

Наприклад:

```
with open('newfile.txt', 'w', encoding='utf-8') as g: d = int(input())
print('1 / {} = {}'.format(d, 1 / d), file=g)
```

І ви можете бути впевнені, що файл буде закритий незалежно від того, що вводив користувач.

Недоліком роботи з файлом у Pyton є те, що файл може мати тільки тестовий формат.

2.7.16. Інсталяція бібліотек

Мова Pyton цікава іще й тим, що більшівсть операцій в ній не потрібно програмувати. Для цього існують так звані бібліотеки підпрограм (Extension Packages). На сайті https://pypi.python.org/pypi їх вже налічується 127091.

Бльш зручним для для отримання цих бібліотек є сайт http://www.lfd.uci.edu/%7Egohlke/pythonlibs/ звідкіля можна взяти потрібну

бібліотеку для того виду операційної системи, яка встановлена на вашому комп’ютері.

Ось так виглядає частина списку бібліотек:

Index by date: scikit-umfpack jupyter matplotlib scikit-image lxml pandas pyrsistent rasterio psutil gensim pillow peewee cffi pillow-simd cobra gpy openpiv chaco enable indexed_gzip spglib yarl mod_wsgi dulwich mercurial cython tifffile numpy orange scandir ode openexr discretize scipy pypmc chainer netcdf4 pymatgen regex twisted zodbpickle zope.interface fiona gdal brotli psycopg javabridge sfepy pyopengl fastcache bsddb3 ruamel.yaml multidict python-ldap faulthandler debug-information-files pymol babel czifile numexpr bokeh mkl-service spyder astropy kiwisolver numba llvmlite sqlalchemy scs veusz scikit-learn mysqlclient shapely vtk h5py cx_oracle fabio fastparquet spectrum tornado chompack mpi4py pycairo pymongo moderngl minepy cvxpy fastcluster pywin32 tensorflow quantlib wrapt yt pycuda wordcloud gr opencv fisx gvar ad3 aiohttp pulp

А так – список конкретної бібліотеки, який ви побачите, обравши якусь зі списку.

NumPy, a fundamental package needed for scientific computi
Numpy+MKL is linked to the Intel® Math Kernel Library
numpy.core directory.

numpy-1.13.3+mkl-cp27-cp27m-win32.whl

numpy-1.13.3+mkl-cp27-cp27m-win_amd64.whl

numpy-1.13.3+mkl-cp34-cp34m-win32.whl

numpy-1.13.3+mkl-cp34-cp34m-win_amd64.whl

numpy-1.13.3+mkl-cp35-cp35m-win32.whl

numpy-1.13.3+mkl-cp35-cp35m-win_amd64.whl

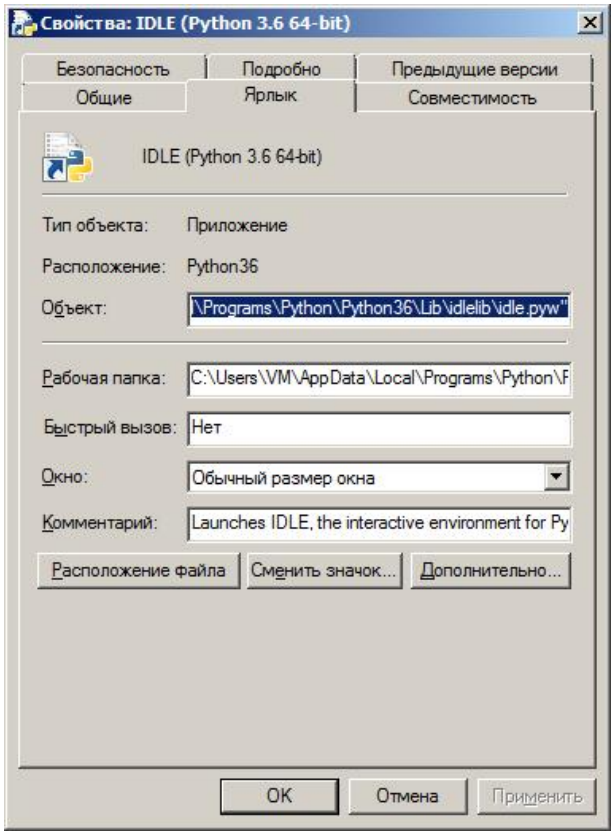
numpy-1.13.3+mkl-cp36-cp36m-win32.whl

numpy-1.13.3+mkl-cp36-cp36m-win_amd64.whl

У назві файлів можна побачити тип операційної системи та її розрядність, наприклад win_32 або win_64.

Всі бібліотеки маюить свій опис, що саме вони роблять. Опис можна прочитати наприклад на сайті https://pypi.python.org/pypi.

Далі будуть подіні описи деяких із бібліотек.



Кожен пакет бібілотеки потрібно підключити до основного пакету Pyton, але пряме підключення робиться не завжди коректно.

Для того, щоб ця бібліотека діяла, необхідно скористатися спеціальною програмою- інсталятором.

Для цього з сайту https://pip.pypa.io/en/latest/installing.html необхідно завантажити файл get-pip.py. Потім клацніть по ньому, щоб він спрацював.

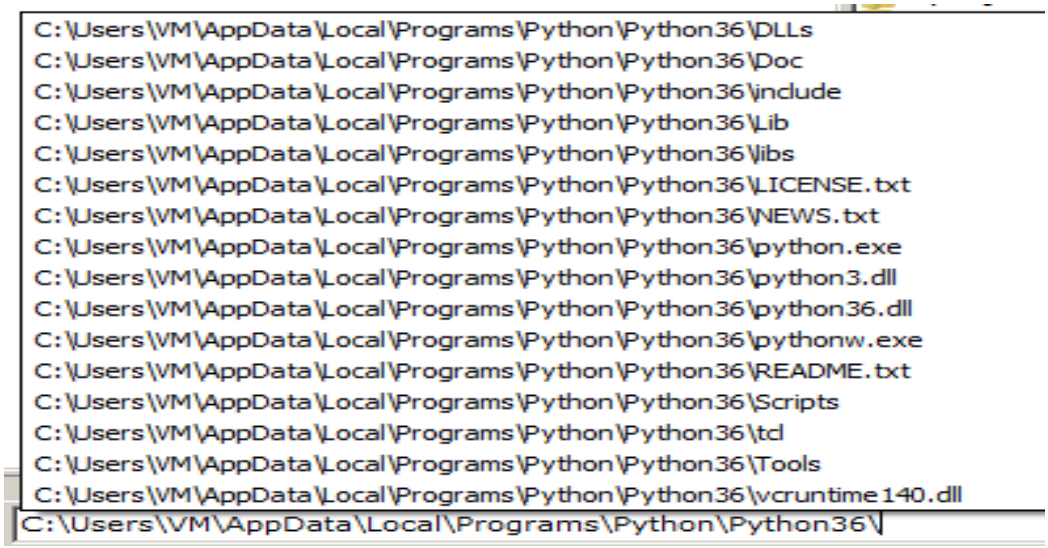
Далі, треба мати на увазі, що всі файли Пітону встановлюються на прихованій теці на диску C:. Тому для подальшої роботи необхідно узнати адресу цієї теки. Для цього за кнопкою «Пуск» знаходимо програму IDLE, клацнемо лівою кнопкою, щоб викликати контекстне меню і вибираємо пункт «Свойства».

Відкриється вікно з повною адресою знаходження цього файлу, так як показано на рисунку.

Вставимо цю адресу в командний рядок. Наприклад, такий:

C:\Users\VM\AppData\Local\Programs\Python\Python36\

В цьому випадку відкриється вікно зі списком внутрішніх папок папки \Piton36, ось так:



Необхідно вибрати той рядок, що закінчується на слово \Scripts. Саме у цій папці міститься файл pip3.exe, який здійснює завантаження на ваш комп'ютер потрібної бібліотеки. Тоді, ваш командний рядок матиме вигляд, наприклад: C:\Python34\Tools\Scripts\pip3.exe install numpy. У цьому рядку слово install означає вказівку для програми PIP почати встановлення бібліотеки, а слово numpy – ім'я бібліотеки. У цьому випадку почнеться завантаження бібліотеки з Інтернету.

Якщо ви вже скачали потрібну бібліотеку з вказаного вище сайту, то команда буде іншою, бо замість простого імені бібліотеки, ви маєте вказати повний шлях до неї, наприклад:

D:\Install 1\PYTON\pandas-0.21.0rc1-cp36-cp36m-win_amd64.whl ,

це якщо ви інсталюєте бібліотеку pandas, файл якої ви вже скачали на власний комп'ютер.

Тобто загальний вигляд команди буде такий:

C:\Users\VM\AppData\Local\Programs\Python\Python36\Scripts\pip3.exe install pandas-0.21.0rc1-cp36-cp36m-win_amd64.whl

Коли інсталяція пакету закінчена, ви можете скористатися його можливоястями, вказавши на початку програми команду, наприклад для пакету «numpy»: «import numpy as np». При цьому додаток до команди «import» – «as np» означає, що до цього пакету можна звертатися за скороченою назвою «np», що зручно при написанні великих програм.

Тепер, коли необхідно наприклад, використтати якусь функцію з цього пакету, спочатку вказується ім’я пакету (повне чи скорочене), ставіиться крапка і дали пишеться ім’я фукнції з пакету. Наприклад, у команді a=np.log(10) змінній *a* присвоюється значення логарифма 10 за основю 10. Використання пакету «numpy» в цьому випадку було потрібно тому, що в стандартному Pyton’і існує тільки натуральний логарифм.

2.7.17. Бібліотека turtle

Бібліотека turtle – це розширення мови Пітон, що дозволяє малювати на екрані нескладні малюнки.

Для цього потрібно підключити бібліотеку turtle. Програму, яка використовує цей графічний модуль треба починати командами (не забудьте про перші два рядки!):

```
import turtle turtle.reset ()
```

Для того, щоб затримати графічне вікно на екрані, необхідно закінчувати всі програми, які використовують модуль turtle командою:

```
turtle._root.mainloop ()
```

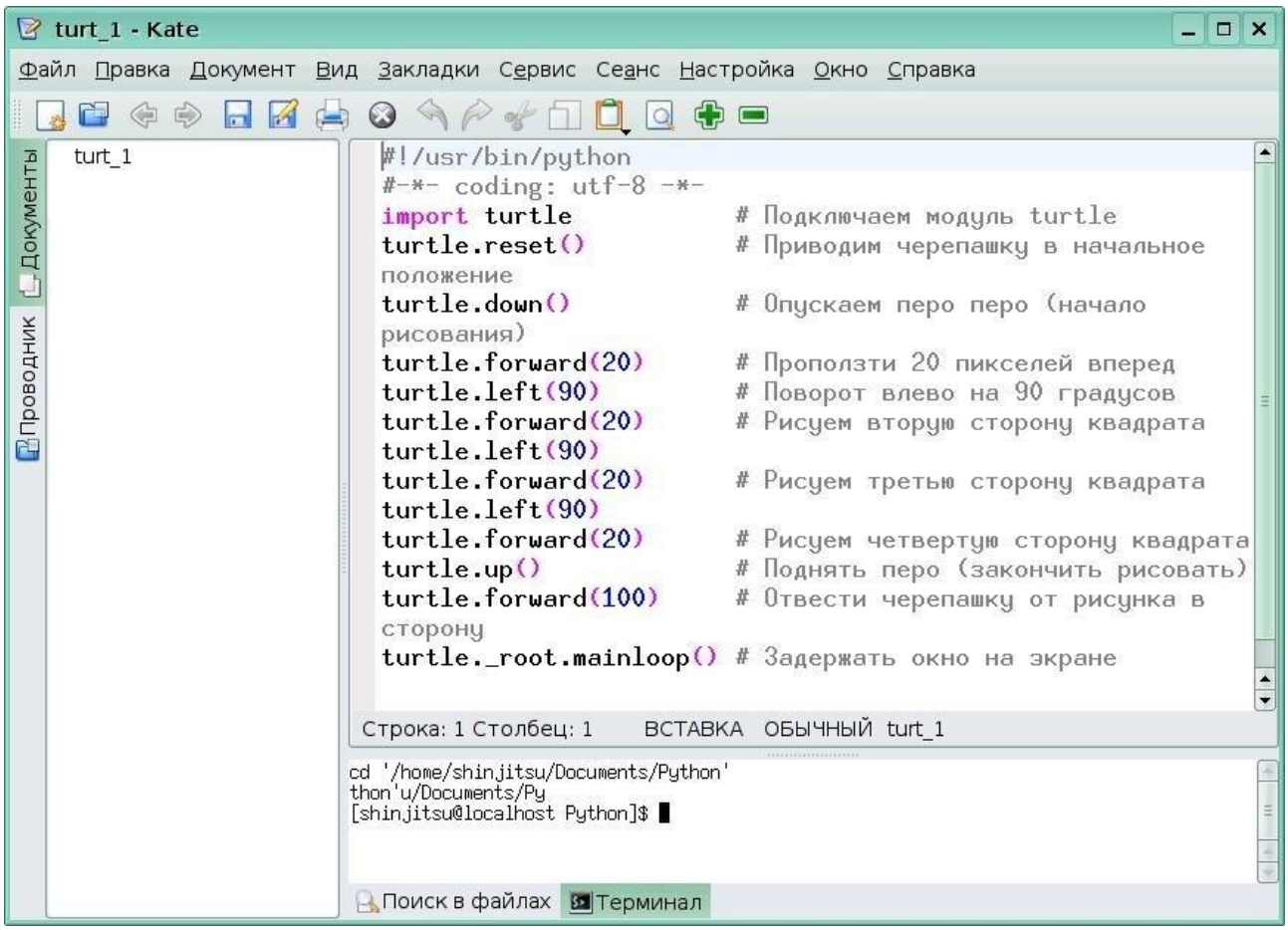
Треба відзначити, що на екрані ми побачимо трикутник.

У Пітоні, як і в інших мовах програмування, необхідно використовувати коментарі. Вони теж позначаються знаком #, коментарі можна писати не тільки в окремому рядку, а й в тому ж рядку, правіше команди.

Пронумерувати рядки можна клавішею F11. Команди для малювання пишуться англійською мовою. Ось необхідний набір команд.

forward (a)	уперед на a кроків
backward (a)	назад на a кроків
left(ß)	ліворуч на ß градусів
right (ß)	праворуч на ß градісов
circle (r)	Намалювати окружність радіусом r, центр ліворуч - r>0, праворуч - r>0
circle(r,ß)	Намалювати дугу радіуса r та градусної міри ß
goto (x,y)	"перейти" в точку з координатами (x,y)
down()	перо опусти
up()	перо підніми
width(a)	нова ширина пера
color(s)	колір пера, "red" - червоний, "green" - зелений, "blue" - синій, "white" - білий, "black" - чорний, "yellow" - жовтий, "pink" - рожевий
fill(f)	Зафарбовує замкнуту область, перед роботою дайте команду *turtle.fill(1)*, а як закінчите розфарбування – *turtle.fill(0)*.
reset()	сброс
clear()	очистка екрану
write(s)	вивід тексту (=напиши)

На малюнку нижче показано приклад застосування цих команд.



2.7.18. Бібліотека pandas

Це основна бібліотека для аналізу даних. Вона входить у так званий NumPy – метамову Pyton, пристосовану для аналізу даних.

2.7.18.1. Структури даних pandas

Щоб почати роботу з pandas, ви повинні освоїти дві основні структури даних: Series і DataFrame. Вони, звичайно, не є універсальним рішенням будь-якого завдання, але все ж утворюють солідну і просту для використання основу більшості додатків.

Series – одновимірний схожий на масив об'єкт, що містить масив даних (будь-якого типу, підтримуваного NumPy) і асоційований з ним масив міток, який називається індексом. Найпростіший об'єкт Series складається тільки з масива даних:

```
In [4]: obj = Series ([4, 7, -5, 3])
In [5]: obj
Out [5]:
0 4
1 7
2 -5
3 3
```

У строковому поданні Series, який відображається в інтерактивному режимі, індекс знаходиться зліва, а значення праворуч. Оскільки ми не задали індекс для даних, то за замовчуванням створюється індекс, що складається з цілих чисел від 0 до N - 1 (де N – довжина масиву даних). Маючи об'єкт Series, отримати уявлення самого масиву і його індексу можна за допомогою атрибутів values і index відповідно:

```
In [6]: obj.values
OuU6]: array ([4, 7, -5, 3])
In [7]: obj.index
Out [7]: Int64Index ([0, 1, 2, 3])
```

Часто бажано створити об'єкт Series з індексом, що ідентифікує кожен елемент даних:

```
In [8]: obj2 = Series ([4, 7, -5, 3], index = [ 'd', 'b', 'a', 'c'])
In [9]: obj2 Out [9]: d 4 b 7
a -5 з 3
In [10]: obj2.index
Out [lO]: Index ([d, b, a, c], dtype = object)
```

На відміну від звичайного масиву NumPy, для вибірки одного або декількох елементів з об'єкта Series можна використовувати значення індексу:

```
In [11]: obj2 [ 'a']
Out [11]: -5
In [12]: obj2 [ 'd‘] = 6
In [13]: obj2 [[ 'c', 'a', 'd']]
```

Out [13]: c 3 a -5 d 6

Операції з масивом NumPy, наприклад фільтрація за допомогою булева масива, скалярне множення або застосування математичних функцій, зберігають зв'язок між індексом і значенням:

In [14]: obj2 Out [14]: d 6 b 7 a -5 c 3
In [15]: obj2 [obj2> Про] Out [15]: d 6 b 7 c 3
In [16]: obj2 * 2 Out [16]: d 12 b 14 a -10 c 6
In [17]: np.exp (obj2) Out [17]: d 403.428793 b 1096.633158 a 0.006738 c 20.085537

Об'єкт Series можна також уявляти собі як упорядкований словник фіксованої довжини, оскільки він відображає індекс на дані.

Об'єкт DataFrame представляє табличну структуру даних, що складається з впорядкованої колекції стовпців, причому типи значень (числовий, рядковий, логічний і т. д.) В різних стовпчиках дані можуть відрізнятися. В об'єкті DataFrame зберігаються два індексу: по рядках і по стовпчиках. Можна вважати, що це словник об'єктів Series. У порівнянні з іншими схожими на DataFrame структурами, які вам могли зустрічатися раніше (наприклад, data. frame в мові R), операції з рядками і стовпцями в DataFrame в першому наближенні симетричні. Усередині об'єкту дані зберігаються у вигляді одного або декількох двовимірних блоків, а не у вигляді списку, словника або ще який-небудь колекції одновимірних масивів.

Хоча в DataFrame дані зберігаються в двовимірному форматі, у вигляді таблиці, неважко уявити і дані більш високої розмірності, якщо скористатится ієрархічним індексуванням. Цю тему ми обговоримо в наступному розділі, вона лежить в основі багатьох просунутих механізмів обробки даних в pandas.

Є багато способів сконструювати об'єкт DataFrame, один з найбільш розповсюджекних – на основі словника списків однакової довжини або масивів NumPy:

data = { 'state': ['Ohio', 'Ohio 1,' Ohio ',' Nevada ',' Nevada '],
'Year': [2000, 2001, 2002 2001, 2002],

'Pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame (data)

Для отриманого DataFrame автоматично буде побудований індекс, як і в разі Series, і стовпці розташуються по порядку:

In [38]: Out [38]: frame
pop state year
0 1.5 Ohio 2000
1 1.7 Ohio 2001
2 3.6 Ohio 2002
3 2.4 Nevada 2001
4 2.9 Nevada 2002

Якщо задати послідовність стовпців, то стовпці DataFrame розташуються строго в зазначеному порядку:

In [39): DataFrame (data, columns = ['year', 'state', 'pop'])
Out [39]
year state pop
0 2000 Ohio 1.5
1 2001 Ohio 1.7
2 2002 Ohio 3.6
3 2001 Nevada 2.4
4 2002 Nevada 2.9

Як і в разі Series, якщо запросити стовпець, якого немає в data, то він буде заповнений значеннями NaN:

In [40]: frame2 = DataFrame (data, columns = ['year', 'state', 'pop', 'debt'], ...: index = ['one', 'two', 'three' , 'four', 'five'])
Out [41]:
year state pop debt
про 2000 Ohio 1.5 NaN
1 2001 Ohio 1.7 NaN
2 2002 Ohio 3.6 NaN
3 2001 Nevada 2.4 NaN
4 2002 Nevada 2.9 NaN
In [42]: frame2.columns
Out [42]: Index ([year, state, pop, debt], dtype = object)

Стовпець DataFrame можна витягти як об'єкт Series, скориставшись нотацією словників, або за допомогою атрибута:

In (43]: frame2 ['state'] In [44]: frame2.year
Out [43]: Out [44]:
one Ohio one 2 0 00

two Ohio two 2 001
three Ohio three 2002
four Nevada four 2001
five Nevada five 2 002
Name: state

Відзначимо, що повернутий об'єкт Series має той же індекс, що і DataFrame, а його атрибут name встановлено відповідним чином. Рядки також можна витягти з позиції або по імені, для чого є два методи, один з них – із зазначенням індексного поля:

In (45): frame2.ix ['three']
Out (45): year 2002
state Ohio pop 3.6
debt NaN
Name: three

Стовпці можна модифікувати шляхом привласнення. Наприклад, порожньому стовпчику 'debt 1 можна було б присвоїти скалярний значення або масив значень:

In (46): frame2 ['debt'] = 16.5 In (47): frame2
Out [47]:
year state pop debt
one 2000 Ohio 1.5 16.5
two 2001 Ohio 1.7 16.5
three 2002 Ohio 3.6 16.5
four 2001 Nevada 2.4 16.5
five 2002 Nevada 2.9 16.5
In (48]: frame2 ['deЬt'] - np.arange (5.)
In (49]: frame2
Out [49]:
year state pop debt
one 2 000 Ohio 1.5 0
two 2 001 Ohio 1.7 1
three 2002 Ohio 3.6 2
four 2001 Nevada 2.4 3
five 2002 Nevada 2.9 4

2.7.18.2. Редукція і обчислення описових статистик

Об'єкти pandas оснащені набором стандартних математичних і статистичних методів. Велика їх частина потрапляє в категорію редукцій, або зведених статистичних методів, які обчислюють єдине значення (наприклад, суму або середнє) для Series або об'єкт Series – для рядків або стовпців DataFrame. У порівнянні з еквівалентними методами масивів NumPy, всі вони ігнорують відсутні значення. Розглянемо невеликий об'єкт DataFrame:

```
In [198]: df = DataFrarne ([[l.4, np.nan], [7.1, -4.5],
[Np.nan, пр.nan], [npo .75, -1.3]],
: Index = [ 'a', 'b', 'c', 'd'],
: Columnscc [ 'one', 'two'])
Out [199]:
one two
a 1.40 NaN
b 7.10 -4.5
c NaN NaN
d 0.75 -1.3
```

Метод sum об'єкта DataFrame повертає Series, що містить суми по стовпцях:

```
In [200]: df.sum ()
Out [200]: one 9.2 5 two -5.80
```

Якщо передати параметр axis = l, то підсумовування буде проводитися по рядкам:

```
In [201]: df.sum (axis = l)
Out [201]:
a 1.40
b 2. 60
c NaN
d -0.55
```

Повний список зведених статистик:

argmin, argmax – Обчислює позицію в індексі (цілі числа), при якому досягається мінімальне або максимальне значення відповідно.

idxmin, idxmax – Обчислює значення індексу, при якому досягається мінімальне або максимальне значення відповідно.

quantile – Обчислює вибірковий квантиль в діапазоні від 0 до 1

sum – Сума значень.

mean – Середнє значення.

median – Медіана (50% -ий квантиль).

mad – Середнє абсолютне відхилення від середнього.

var – Вибіркова дисперсія.

std – Вибіркове стандартне відхилення.

skew – Асиметрія (третій момент).

kurt – Куртозіс (четвертий момент).

cumsum – Наростаюча сума.

cummin, cummax – Наростаючий мінімум або максимум відповідно.

cumprod – Наростання множення.

diff – Перша арифметична різниця (корисно для часових рядів).

pct_change – Обчислює процентну зміну.

Деякі зведені статистики, наприклад кореляція і ковариация, обчислють по парам аргументів. Розглянемо об'єкти DataFrame, що містять ціни акцій і обсяги біржових угод:

```
import pandas.io.data as web all_data = {}
for ticker in [ 'AAPL', 'IBM', 'MSFT', 'GOOG']:
    all_data [ticker] = web.get_data_yahoo (ticker, '1/1/2000', '1/1/2010')
price = DataFrame ({tic: data [ 'Adj Close']
for tic, data in all_data.iteritems ()})
    volume = DataFrame ({tic: data [ 'Volume']
for tic, data in all_data.iteritems ()})
```

Тепер обчислимо процентні зміни цін:

```
In [209]: returns = price.pct_change ()
Out [210]: Date
2009-12-24
2009-12-28
2009-12-29
2009-12-30
2009-12-31
AAPL
0.034339
```

0.012294
-0. 011861
0.012147
-0.004300
GOOG
0.011117
0.007098
-0.005571
0.005376
-0.004416
IBM
0.004420
0.013282
-0.003474
0.005468
-0.012609
MSFT
0.002747
0.005479
0.006812
-0.013532
-0.015432

Метод corr об'єкта Series обчислює кореляцію відмінних від NA, вирівняних за індексом значень в двох об'єктах Series. Відповідно, метод cov обчислює коваріацию:

In [211]: returns.MSFT.corr (returns.IBM)
Out [211]: O.49609291822168838
In [212]: returns.MSFT.cov (returns.IBM)
Out [212]: O.00021600332437329015

З іншого боку, методи corr і cov об'єкта DataFrame повертають відповідно повну кореляційний або ковариационную матрицю у вигляді DataFrame:

In [213] I: returns. .corr ()
Out [213] I:
AAPL GOOG IBM MSFT
AAPL 1.000000 0.470660 0.410648 0.424550
GOOG 0.470660 1.000000 0.390692 0.443334
IBM 0.410648 0.390692 1.000000 0.496093
MSFT 0.424550 0.443334 0.496093 1.000000
In [214] I: returns. . cov ()
Out [214] 1:
AAPL GOOG IBM MSFT
AAPL 0.001028 0.000303 0.000252 0.000309

GOOG 0.000303 0.000580 0.000142 0.000205
IBM 0.000252 0.000142 0.000367 0.000216
MSFT 0.000309 0.000205 0.000216 0.000516

За допомогою методу corrwith об'єкта DataFrame можна обчислити попарні кореляції між стовпцями або рядками DataFrame і іншим об'єктом Series або DataFrame. Якщо передати йому об'єкт Series, то буде повернуто Series, що містить значення кореляції, обчисленої для кожного стовпця:

In [215]: returns.corrwith (returns.IBM)
Out (215):
AAPL 0.410648 GOOG 0.3 9 06 92 IBM 1.000000 MSFT 0.496093

Якщо передати об'єкт DataFrame, то будуть обчислені кореляції стовпців з відповідними іменами. Нижче обчислено кореляції процентних зв’язків з обсягом угод:

In [216]: returns.corrwith (volume)
Out [216]:
AAPL -0.057461 GOOG 0.062644 IBM -0.007900 MSFT -0.014175

Якщо передати axis = l, то будуть обчислені кореляції рядків. У всіх випадках перед початком обчислень дані вирівнюються по мітках.

2.7.19. Інтеграція MS Excel та Python

Для роботи з Excel файлами з Python варто використти бібліотеки: xlrd, xlwt, xlutils або openpyxl

Для прикладу будемо використовувати готовий файл excel з якого ми спочатку завантажуємо дані і з першої клітинки, а потім запишемо їх в другу. Треба також мати на увазі, що всі ці бібліотеки працюють тільки з такими файлами, що мають імена, написані латиницею. Те саме стосується й повного шляху до потрібного файлу Excel – всі теки і підтеки потрібно іменувати теж латиницею, інакше, програмf не працюватиме, а видаватиме помилку.

Для початку завантажимо потрібні бібліотеки і відкриємо файл xls на читання і виберемо потрібний лист з даними:

```
import xlrd, xlwt
# Відкриваємо файл
rb = xlrd.open_workbook ( '../ ArticleScripts / ExcelPython / xl.xls', formatting_info = True)
# Вибираємо активний лист
sheet = rb.sheet_by_index (0)
```

Тепер давайте подивимося, як завантажити значення з потрібних клітинок:

```
# Отримуємо значення першої клітинки A1
val = sheet.row_values (0) [0]
# Клітинки в Excel нумеруються так: номер рядку в круглих дужках, номер стовпця – у квадратних.

# Отримуємо список значень з усіх записів
vals = [sheet.row_values (rownum) for rownum in range (sheet.nrows)]
```

Як видно читання даних не складає труднощів. Тепер запишемо їх в інший файл. Для цього створю новий excel файл з новою робочою книгою:

```
wb = xlwt.Workbook ()
ws = wb.add_sheet ( 'Test')
```

Запишемо в новий файл отримані раніше дані і збережемо зміни:

```
# В A1 записуємо значення з комірки A1 з іншого файлу
ws.write (0, 0, val [0])
```

```
# В стовпець В запишемо нашу послідовність зі стовпця А вихідного файлу
i = 0
for rec in vals:
ws.write (i, 1, rec [0])
   i = + i

# Зберігаємо робочу книгу
wb.save ( '../ ArticleScripts / ExcelPython / xl_rec.xls')
```

З прикладу вище видно, що бібліотека xlrd відповідає за читання даних, а xlwt – за запис, тому немає можливості внести зміни в уже створену книгу без її копіювання в нову. Крім цього зазначені бібліотеки працюють тільки з файлами формату xls (Excel 2003) і у них немає підтримки нового формату xlsx (версії Excel 2007 і вище).

Щоб успішно працювати з форматом xlsx, знадобиться бібліотека openpyxl. Для демонстрації її роботи виконаємо дії, які були показані для попередніх бібліотек.

Для початку завантажимо бібліотеку і виберемо потрібну книгу і робочий лист:

```
import openpyxl
wb = openpyxl.load_workbook (filename = '../ArticleScripts/ExcelPython/openpyxl.xlsx')
sheet = wb [ 'test']
```

Як видно з вищенаведеного лістингу зробити це не складно. Тепер подивимося як можна завантажити дані:

```
# Зчитуємо значення певної клітинки
val = sheet [ 'A1']. value
```

```
# Зчитуємо заданий діапазон
vals = [v [0] .value for v in sheet.range ( 'A1: A2')]
```

Відмінність від попередніх бібліотек в тому, що openpyxl дає можливість звертатися до клітинок і їх груп через їх імена,такі ж, як і в самому Excel що досить зручно і зрозуміло при читанні програми.

Тепер подивимося як нам зробити запис і зберегти дані:

```
# Записуємо значення в певну комірку
sheet [ 'B1'] = val

# Записуємо послідовність
i = 0
for rec in vals:
   sheet.cell (row = i, column = 2) .value = rec
   i = + 1

# Зберігаємо дані
wb.save ( '../ ArticleScripts / ExcelPython / openpyxl.xlsx')
```

У випадку, якщо один лист електронної таблиці містить одну таблицю, можна скористатися простішим кодом, наприклад:

```
import xlrd
xls_file = pd.ExcelFile('table_D1_transp.xls')
table = xls_file.parse('Sobol_tr')
print(table)
```

Тут спочатку відкриваються потрібна бібліотека «xlrd», потім відкривається файл «table_D1_transp.xls». Далі, в масив «table» вносяться дані з листа, що має ім’я «Sobol_tr'». Але цей прийом діє тільки для електронних таблиць версії «*.xls».

2.7.20. XML і HTML: як із них вибирати дані

На Python написано багато бібліотек для читання і запису даних, що містяться у форматах HTML і XML. Зокрема, бібліотека lxml (http://lxml.de) відома високою продуктивністю при розборі дуже великих файлів. Для lxml є кілька програмних інтерфейсів; спочатку продемонстрємпо інтерфейс lxml. html для роботи з HTML, а потім розберемо XML-документ за допомогою lxml.obj ectify.

Багато сайтів показують дані у вигляді HTM-таблиць, зручних для перегляду в браузері, але не пропонують їх в таких машинозчитуваних форматах, raJSON або XML. Так, наприклад, йде справа з даними про біржових опціонах на сайті Yahoo! Finance. Опціон – це похідний фінансовий інструмент (дериватив), який дає право купувати (опціон на придбання, або колл-опціон) або продавати (опціон на продаж, або пут-опціон) акції компанії за деякою ціною (ціною виконання) в проміжку часу між поточним моментом і деяким фіксованим моментом в майбутньому (кінцевою датою). Колл- і пут-опціони торгуються з різними цінами виконання і кінцевими датами; ці дані можна знайти в таблицях на сайті Yahoo! Finance.

Для початку вирішіть, з якої URL-адреси ви хочете завантажувати дані, потім відкрийте його за допомогою засобів з бібліотеки urllib2 і розберіть потік, користуючись xml:

```
from lxml.html import parse
from urllib2 import urlopen. .
parsed = parse (urlopen (
'http://finance.yahoo.com/q/op?s=AAPL+Op
tions'))
doc = parsed.getroot ()
```

Маючи цей об'єкт, ми можемо вибрати всі HTML-теги зазначеного типу, наприклад, теги table, всередині яких знаходяться дані, що нас цікавлять. Для прикладу отримаємо список усіх активних посилань в документі, вони представляються в HTML тегом A. Викличемо метод findall кореневого елемента

документа, передавши йому вираз XPath (це мова, на якому записуються «запити» до документу):

```
In [906]: links = doc.findall ( '.// a')
In [907]: links [l5: 20] Out [907]:
[<Element a at 0x6c488f0>,
<Element a at 0x6c48950>,
<Elernent a at 0x6c489Ь0>,
<Elernent a at 0x6c48a10>,
<Elernent a at 0x6c48a70>]
```

Але це об'єкти, що представляють HTML-елементи; щоб отримати URL і текст посилання, нам потрібно скористатися методом get елемента (для отримання URL) або методом text_content (для отримання тексту):

```
In [908]: lnk = links [28]
In [909]: lnk
0ut [909]: <Element a at 0x6c48dd0>
In [910]: lnk.get ( 'href')
0ut [910]: 'http://biz.yahoo.com/special.html'
In (911]: lnk.text_content ()
Out [911]: 'Special Editions'
```

Таким чином, отримання усіх активних посилань в документі зводиться до включення за списком:

```
In (912]: urls = [lnk.get ( 'href') for lnk in doc.findall ( '.// a')]
In (913]: urls [-10:]
Out [913]:
[ 'Http://info.yahoo.com/privacy/us/yahoo/finance/details.html',
'Http://info.yahoo.com/relevantads/',
'Http://docs.yahoo.com/info/terms/',
'Http://docs.yahoo.com/info/copyright/copyright.html',
'http://help.yahoo.com/l/us/yahoo/finance/forms_index.html', 'http:
// help .yahoo.eom / l / us / yahoo / finance / quotes / fitadelay.html
',' http://help.yahoo.com/l/us/yahoo/finance/quotes/fitadelay.html ',
'Http://www.capitaliq.com',
'Http://-www.csidata.com',
'Http: //w^.morningstar.com/']
```

Що стосується відшукання потрібних таблиць в документі, то це робиться методом проб і помилок; на деяких сайтах рішення цього завдання спрощується, тому що таблиця має атрибут id. Я знайшов, які таблиці містять дані про колл- і пут-опціони:

```
tables = doc.findall ( '.// table') calls = tables [9] puts = tables [13]
```

У кожній таблиці є рядок-заголовок, а за нею йдуть рядки з даними:

```
In [915]: rows = calls.findall ( '.// tr')
```

Для всіх рядків, включаючи заголовок, ми хочемо отримати текст з кожного осередку; в разі заголовка осередками є елементи TR, а для рядків даних – елементи TD:

```
def _unpack (row, kind = 'td'):
elts = row.findall ( './/% s'% kind) return [val.text_content () for val in elts]
```

Таким чином, отримуємо:

```
In [917]: _unpack (rows [O], kind = 'th')
Out [917]: [ 'Strike', 'Syi ^ iol', 'Last', 'Chg', 'Bid', 'Ask', 'Vol', 'Openlnt'
In [918]: _unpack (rows [l], kind = 'td')
Out [918]:
[ '295.00',
'AAPL12 0818C002 95 000',
'310.40',
'0.00',
'289.80',
'290.80',
'1',
'169']
```

Тепер для перетворення даних в об'єкт DataFrame залишилося об'єднати всі описані кроки разом. Оскільки числові дані як і раніше записані у вигляді рядків, можливо, буде потрібно перетворити деякі, але не всі стовпці в формат з плаваючою точкою. Це можна зробити і вручну, але, на щастя, в бібліотеці pandas є клас TextParser, який використовується для здійснення read_ csv і іншими функціями розбору для автоматичного перетворення типів:

```
from pandas.io.parsers import TextParser
def parse_options_data (table): rows = table.findall ( '.// tr') header = _unpack (rows [O], kind = 'th') data = [_unpack (r) for r in rows [l:]] return TextParser (data, names = header) .get_chunk ()
```

Нарешті, викликаємо цю функцію розбору для табличних об'єктів lxml і отримуємо результат у вигляді DataFrame:

```
In [920]: call_data = parse_options_data (calls)
In [921]: put_data = parse_options_data (puts)
In [922]: call_data [: 10]
Out[922]:
```

Strike Symbol Last Chg Bid Ask Vol Open Int
o 295 AAPL120818C00295000 310.40 o.o 289.80 290.80 1 169
1 300 AAPL120818C00300000 277.10 1.7 284.80 285.60 2 478
2 305 AAPL120818C00305000 300.97 o.o 279.80 280.80 10 316
3 310 AAPL120818C00310000 267.05 o.o 274.80 275.65 6 239
4 315 AAPL120818C00315000 296.54 o.o 269.80 270.80 22 88
5 320 AAPL120818C00320000 291.63 o.o 264.80 265.80 96 173
6 325 AAPL120818C00325000 261.34 o.o 259.80 260.80 N/A 108
7 330 AAPL120818C00330000 230.25 o.o 254.80 255.80 N/A 21
8 335 AAPL120818C00335000 266.03 o.o 249.80 250.65 4 46
9 340 AAPL120818C00340000 272.58 o.o 244.80 245.80 4 30

XML (розширювана мова розмітки) – ще один популярний формат представлення структурованих даних, що підтримує ієрархічно вкладені дані, забезпечені метаданими.

Продемонструємо альтернативний інтерфейс, зручний для роботи з XML-даними, – lxml .objectify.

Управління міського транспорту Нью-Йорка (MTA) публікує тимчасові ряди з даними про роботу автобусів і електричок (http: j / www.mta.info / developers / download.html). Ми зараз розглянемо дані про якість обслуговування, зберігається у вигляді XML-файлів. Для кожної автобусної та залізничної компанії існує свій файл (наприклад, Performance_MNR. Xml для компанії MetroNorth Railroad), що містить дані за один місяць в вигляді послідовності таких XML-повідомлень

<INDICATOR>
<INDICATOR_SEQ> 3 73 889 </ INDICATOR_SEQ>
<PARENT_SEQ> </ PARENT_SEQ>
<AGENCY_N ^ E> Metro-North Railroad </ AGENCY_NAME>
<INDICATOR_NAME> Escalator Availability </ INDICATOR_NAME> <DESCRIPTION> Percent of the time that escalators are operational systemwide. The availability rate is based on physical observations performed the morning of regular business days only. This is a new indicator the agency began reporting in 2009. </ DESCRIPTION>
<PERIOD_YEAR> 20ll </ PERIOD_YEAR>
<PERIOD_MONTH> l2 </ PERIOD_MONTH>
<CATEGORY> Service Indicators </ CATEGORY>
<FREQUENCY> M </ FREQUENCY>

```
<DESIRED_CHANGE> U </ DESIRED_CHANGE>
<INDICATOR_UNIT>% </ INDICATOR_UNIT>
<DECIMAL_PLACES> l </ DECIMAL_PLACES>
<YTD_TARGET> 97.00 </ YTD_TARGET>
<YTD_ACTUAL> </ YTD_ACTUAL>
<MONTHLY_TARGET> 97.00 </ MONTHLY_TARGET>
<MONTHLY_ACTUAL> </ MONTHLY_ACTUAL>
</ INDICATOR>
```

Використовуючи lxml. obj ecti fy, ми розбираємо файл і отримуємо посилання на кореневий вузол XML-документа від методу getroot:

```
from lxml import objectify
path = 'Performance_MNR.xml' parsed = objectify.parse (open (path))
root = parsed.getroot ()
```

Властивість root. INDICATOR повертає генератор, послідовно віддає всі елементи <INDICATOR>. Для кожного запису ми заповнюємо словник імен тегів (наприклад, YTD_ACTUAL) значеннями даних (деякі теги пропускаються):

```
data = []
skip_fields = [ 'PARENT_SEQ', 'INDICATOR_SEQ',
'DESIRED_CHANGE', 'DECIHAL_PLACES']
for elt in root.INDICATOR:
el_data = {}
for child in elt.getchildren (): if child.tag in skip_fields: continue
el_data [child.tag] = child.pyval data.append (el_data)
```

Нарешті, перетворимо цей список словників в об'єкт DataFrame:

```
In [927]: perf = DataFrame (data)
In [928]: perf Out [928]:
Empty DataFrame
Columns: array ([], dtype = int64) Index: array ([], dtype = int64)
```

XML-документи можуть бути набагато складніше, ніж в цьому прикладі. Зокрема, в кожному елементі можуть бути метадані. Розглянемо тег гіперпосилання в форма¬те HTML, який є окремим випадком XML:

```
from StringlO import StringlO
tag = '<a href = "http: //^www.google.com"> Google </a>'
root = objectify.parse (StringIO (tag)). getroot ()
```

Тепер ми можемо звернутися до будь-якого атрибуту тега (наприклад, href) або до тексту посилання:

```
In [930]: root
0ut [930]: <Element a at 0x88bd4b0>
In [931]: root.get ( 'href')
```

```
Out [931]: 'http: //^www.google.com'
In [932]: root.text
Out [932]: 'Google'
```

У pandas є також підтримка для читання табличних даних в форматі Excel 2003 (і більш пізніх версією) за допомогою класу ExcelFiie. На внутрішньому рівні ExcelFile користується пакетами xlrd і openpyxl, тому їх потрібно попередньо встановити. Для роботи з ExcelFile створіть його екземпляр, передавши конструктору шлях до файлу з розширенням xls або xlsx:

```
xls_file = pd.ExcelFile ( 'data.xls')
```

Прочитати дані з робочого листа в об'єкт DataFrame дозволяє метод parse:

```
table = xls_file.parse ( 'Sheetl')
```

Взаємодія з HTML і Web AP /

Багато сайтів надають відкритий API для отримання даних в форматі JSON або якомусь іншому. Отримати доступ до таких API з Python можна різними способами; рекомендується простий пакет requests (http.j / docs.python-requests. org). Для пошуку за словами «python pandas» в Твіттері ми можемо відправити такий HTTP-запит GET:

```
In [944]: import requests
In [945]: url = 'http://search.twitter.com/search.json?q=python%20pandas'
In [946]: resp = requests.get (url)
In (947): resp
Out [947]: <Response [200)>
```

У об'єкта Response має атрибут text, в якому зберігається вміст відповіді на запит GET. Багато API в веб повертають JSON-рядок, яку слід завантажити в об'єкт Python:

```
In [948): import json
In [949]: data = json.loads (resp.text)
In [950]: data.keys ()
Out [950]:
[U'next_page ', u'completed_in', u'max_id_str ', u'since_id_str', u'refresh_url ', u'results', u 'since_id', u'results_per_page ', u'query', u'max_id ' , u'page ']
```

Поле відповіді results містить список твітів, кожен з яких представлений таким словником Python:

{U'created_at ': u'Mon, 25 Jun 2012 17:50:33 +0000', u1from_user ': u'wesmckinn', u'from_user_id ': 115494880, u'from_user_id_str': u'l15494880 ', u'from_user_name ': u'Wes McKinney •, u'geo': None,
u'id ': 217313849177686018,
u'id_str ': u'217313849177686018',
u'iso_language_code ': u'pt',
u'metadata ': {u'result_type': u'recent '},
u'source ': u' web ',
u'text ': u'Lunchtime pandas-fu http://t.co/SI70xZZQ #pydata',
u'to_user ': None,
u'to_user_id ': 0,
u'to_user_id_str ': u'O',
u'to_user_name ': None}

Далі ми можемо побудувати список полів твіту, що нас цікавлять та передати його конструктору DataFramc:

```
In [951]: tweet_fields = [ 'created_at', 'from_user', 'id', 'text']
In [952]: tweets = DataFrame (data [ 'results'], columns = tweet_fields)
In [953]: tweets Out [953]:
<Class 'pandas.core.frame.DataFrame'>
Int64Index: 15 entries, Про to 14 Data columns:
created_at 15 non-null values from_user 15 non-null values
id 15 non-null values
text 15 non-null values
dtypes: int64 (1), object (3)
```

Тепер в кожному рядку DataFrame знаходяться дані, витягнуті з одного твіту:

```
Out [121]: created_at frorn_user id
In [121]: tweets.ix [7]
Thu, 23 Jul 2012 9:54:00 +0000
deblike
227419585803059201
text pandas: powerful Python data analysis toolkit Name: 7
```

2.7.21. Побудова графіків та візуалізація

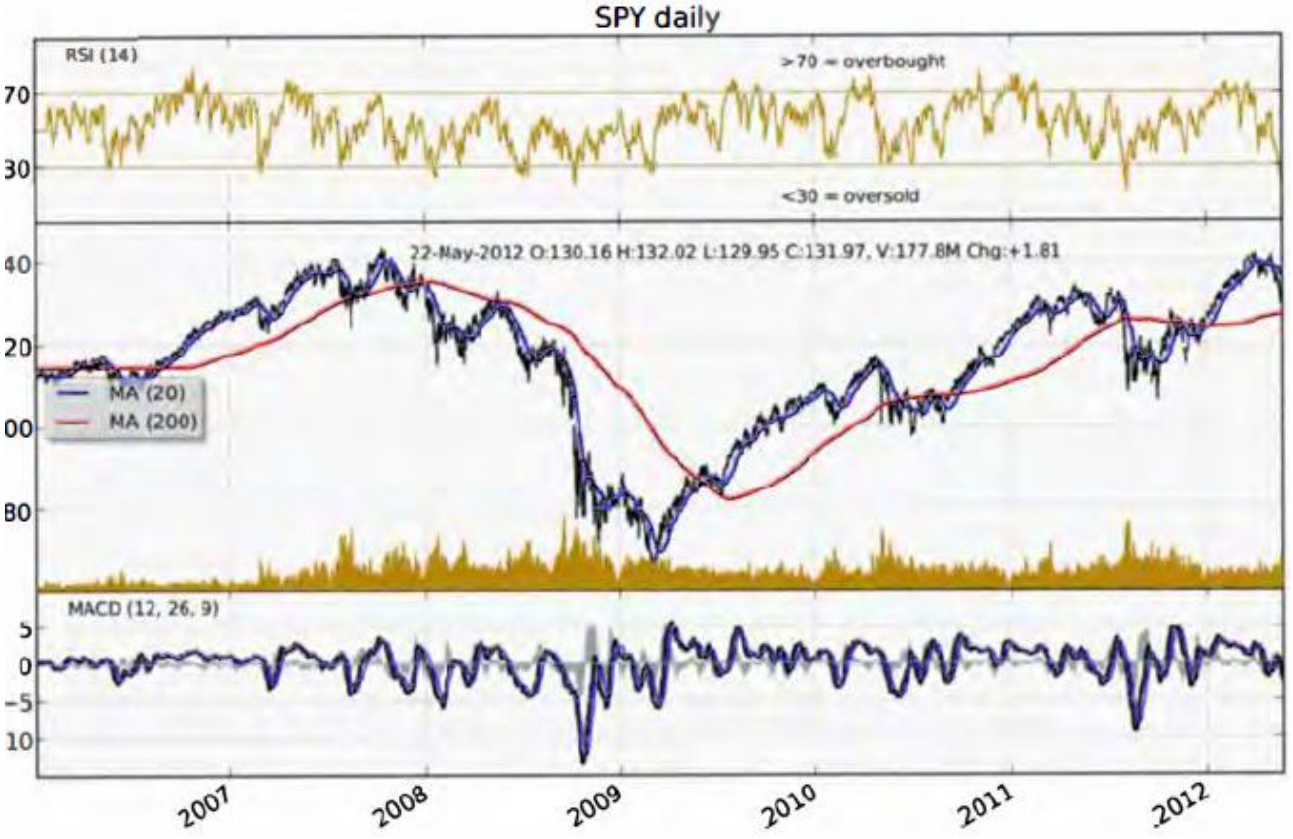
Побудова графіків, а також статична або інтерактивна візуалізація – одне з найважливіших завдань аналізу даних. Вони можуть бути частиною процесу дослідження, наприклад, застосовуватися для виявлення викидів, ухвали необхідних перетворень даних або пошуку ідей для побудови моделей. В інших випадках побудова інтерактивної візуалізації для веб-сайту, наприклад за допомогою бібліотеки d3.js (http: //d3js.or-g/), може бути кінцевою метою. Для Python є багато інструментів візуалізації.

Matplotlib – це пакет для побудови графіків (головним чином, двовимірних) поліграфічної якості. При використанні в поєднанні з якою-небудь бібліотекою ГІП (наприклад, всередині IPython), matplotlib набуває інтерактивні можливості: панорамування, масштабування і інші. Цей пакет підтримує різноманітні системи ГІП у всіх операційних системах, а також вміє експортувати графічні дані у всіх векторних і растрових форматах: PDF, SVG, JPG, PNG, BMP, GIF і т. Д.

Для matplotlib є цілий ряд додаткових бібліотек, наприклад mpiot3d для побудови тривимірних графіків і basemap для побудови карт і проекцій. Для опрацювання наведених в цьому пункті прикладів коду, не забудьте завантажити IPython в режимі pylab (ipython --pylab) або включити інтеграцію з циклом обробки подій ГІП за допомогою функції% gui.

Взаємодіяти з matplotlib можна декількома способами. Самий поширенням странения – запустити IPython в режимі pylab за допомогою команди ipython --pylab. В результаті IPython конфігується для підтримки обраної системи ГІП (Tk, wxPython, PyQt, платформний ГІП OS X, GTK). Для більшості користувачів мається на увазі за замовчуванням системи ГІП. У режимі pylab в IPython також імпортується багато модулів і функцій, щоб інтерфейс був більше схожий на MATLAB. Переконатися, що все працює, можна, побудувавши простийграфік:

plot (np.arange (lO))



Якщо все налаштовано правильно, то з'явиться нове вікно з лінійним графіком. Його можна закрити мишею або ввівши команду close (). Всі функції matplotlib API, зокрема plot і close, знаходяться в модулі matplotlib.pyplot, при імпорті якого зазвичай дотримуються наступної угоди:

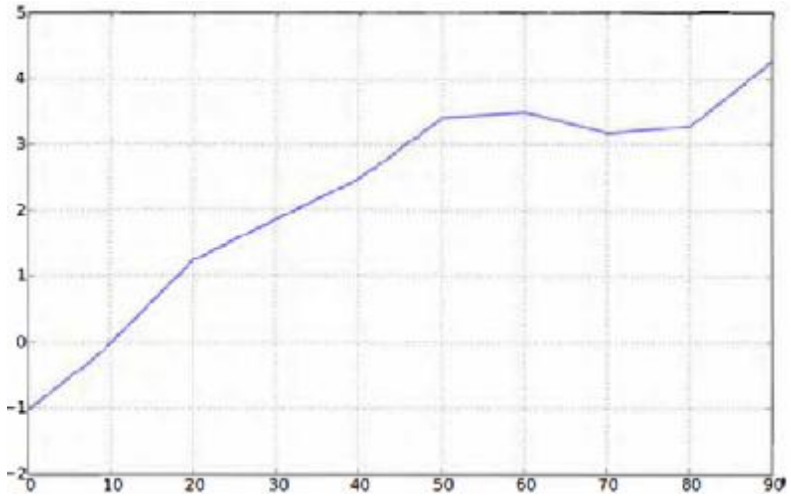
import matplotlib.pyplot as plt

Як ви могли переконатися, matplotlib – бібліотека досить низького рівня. Графік в ній складається з базових компонентів: спосіб відображення даних (тип графіка: лінійний графік, стовбчаста діаграма, коробчаста діаграма, діаграма розсіювання, контурний графік і т. д.), Пояснювальий напис, назва, мітки рисок і інші анотації. Почасти так зроблено тому, що в багатьох випадках дані, необхідні для побудови повного графіка, розкидані по різних об'єктах. У бібліотеці pandas у нас вже є мітки рядків, мітки стовців і, можливо, інформація про угруповання. Це означає, що багато графіків, для побудови яких засобами matplotlib довелося б писати багато коду, в pandas можуть бути побудовані за допомогою одного-двох коротких команд. Тому в pandas є багато високорівневих методів побудови графіків для стандартних типів візуалізації, в

яких використовується інформація про вну¬тренней організації об'єктів DataFrame.

У об'єктів Series і DataFrame є метод plot, який вміє будувати графіки різних типів. За замовчуванням він будує лінійні графіки:

```
In [55]: s = Series
(np.random.randn
(lO) .cumsum (),
index = np.arange
(O, 100, 10)) In
[56]: s.plot ()
```



Індекс об'єкту Series передається matplotlib для нанесення рисок на вісь X, але це можна відключити, задавши параметр use_index = False. Ризики і діапазон значень на осі X можна налаштовувати за допомогою параметрів xticks і xlim, а на осі Y – за допомогою параметрів yticks і y1im. Повний перелік параметрів методу plot наведено нижче:

label – Мітка для пояснювальної написи на графіку.

ax – Об'єкт подграфіка matplotlib, всередині якого будувати графік. Якщо параметр не заданий, то використовується активний подграфік.

Style – Рядок стилю, наприклад 'ko--', що передається у matplotlib.

alpha – Рівень непрозорості графіка (число від 0 до 1).

kind – Може приймати значення 'line', 'bar' barh '; «kde».

logy – Використовувати логарифмічний масштаб по осі Y.

use_index – Брати мітки рисок з індексу об'єкта.

rot – Кут повороту міток рисок (від 0 до 360).

xticks – Значення рисок на осі X.

yticks – Значення рисок на осі Y.

xlim – Межі по осі X (наприклад, [0, 10])

ylim – Межі по осі Y.

grid – Відображати координатну сітку (за замовчуванням включено).

2.7.22. Алгоритми знайдення оптимального рішення

Для прикладу, візьмемо класичну задачу групової подорожі.

Наш приклад відноситься до планування групової подорожі дружній компанії з різних частин світу. Всякий, хто намагався планувати поїздку групи людей або навіть однієї людини, розуміє, що потрібно враховувати безліч вихідних даних, наприклад: яким рейсом повинен летіти кожна людина, скільки потрібно орендувати машин і від якого аеропорту найзручніше добиратися. Також слід взяти до уваги ряд вихідних змінних: повна вартість, час очікування в аеропортах і непродуктивно витрачений час.

Планування подорожі групи людей (в даному прикладі сімейства Джонс), які, вирушаючи з різних місць, повинні прибути в одне і те ж місце.

Члени сім'ї живуть в різних кінцях країни і хочуть зустрітися Нью-Йорку. Всі вони повинні вилетіти в один день і в один день полетіти і при цьому з метою економії хотіли б виїхати з аеропорту і приїхати в нього на одній орендованій машині. Щодня, до міста Нью Йорк, аеропорт Ла Гуардіа (LGA), з місць проживання будь-якого члена сім'ї відправляються десятки рейсів, всі в різний час. Ціна квитка і час у дорозі для кожного рейсу різні.

Почнемо зі списку самих Джонс:

```
# Тут перше слово – це ім’я, а друге – місто, звідкіля ці люди відправляються
peoples = [( 'Seymour', 'BOS'),
   ( 'Franny', 'DAL'),
   ( 'Zooey', 'CAK'),
   ( 'Walt', 'MIA'),
```

```
    ( 'Buddy', 'ORD'),
    ( 'Les', 'OMA')]


    # місце призначення
    destination = 'LGA'

    # Словник рейсів
    flights = {( 'LGA', 'CAK'): [( '6:58', '9:01', 238), ( '8:19', '11: 16 ', 122), (' 9: 58 ',
'12: 56', 249), ('10: 32 ', '13: 16', 139), ('12: 01 ', '13: 41', 267), ('13: 37 ' , '15: 33 ', 142),
('15: 50', '18: 45 ', 243), ('16: 33', '18: 15 ', 253), ('18: 17', ' 21:04 ', 259), ('19: 46', '21:
45 ', 214)], (' DAL ',' LGA '): [(' 6:12 ', '10: 22', 230 ), ( '7:53', '11: 37 ', 433), (' 9:08 ',
'12: 12', 364), ('10: 30 ', '14: 57', 290), ('12: 19 ', '15: 25', 342), ('13: 54 ', '18: 02', 294),
('15: 44 ', '18: 55', 382), ( ' 16:52 ', '20: 48', 448), ('18: 26 ', '21: 29', 464), ('20: 07 ', '23:
27', 473)], ( 'LGA ',' BOS '): [(' 6:39 ',' 8:09 ', 86), (' 8:23 ', '10: 28', 149), ( '9:58', '11:
18 ', 130), ('10: 33', '12: 03 ', 74), ('12: 08', '14: 05 ', 142), ('13: 39', '15: 30 ' , 74), ('15:
25 ', '16: 58', 62), ('17: 03 ', '18: 03', 103), ('18: 24 ', '20: 49', 124 ), ('19: 58 ', '21: 23',
142)], ( 'LGA', 'MIA'): [( '6:33', '9:14', 172), ( '8: 23 ', '11: 07', 143), ( '9:25', '12: 46 ',
295), ('11: 08', '14: 38 ', 262), ('12: 37' , '15: 05 ', 170), ('14: 08', '16: 09 ', 232), ('15:
23', '18: 49 ', 150), ('16: 50', ' 1 9:26 ', 304), ('18: 07', '21: 30 ', 355), ('20: 27', '23: 42 ',
169)], (' LGA ',' OMA ') : [( '6:19', '8:13', 239), ( '8:04', '10: 59 ', 136), (' 9:31 ', '11: 43',
210), ('11: 07 ', '13: 24', 171), ('12: 31 ', '14: 02', 234), ('14: 05 ', '15: 47', 226), ( ' 15:07
', '17: 21', 129), ('16: 35 ', '18: 56', 144), ('18: 25 ', '20: 34', 205), ('20: 05 ', '21: 44',
172)], ( 'OMA', 'LGA'): [( '6:11', '8:31', 249), ( '7:39', '10: 24 ', 219), (' 9:15 ', '12: 03',
99), ('11: 08 ', '13: 07', 175), ('12: 18 ', '14: 56' , 172), ('13: 37 ', '15: 08', 250), ('15: 03
', '16: 42', 135), ('16: 51 ', '19: 09', 147 ), ('18: 12 ', '20: 17', 242), ('20: 05 ', '22: 06',
261)], ( 'CAK', 'LGA'): [( '6: 08 ',' 8:06 ', 224), (' 8:27 ', '10: 45', 139), ( '9:15', '12: 14 ',
247), ('10: 53' , '13: 36 ', 189), ('12: 08', '14: 59 ', 149), ('13: 40', '15: 38 ', 137), ('15:
23', ' 17:25 ', 232), ('17: 08', '19: 08 ', 262), ('18: 35', '20: 28 ', 204), ('20: 30', '23: 11 ',
114)], (' LGA ',' DAL '): [(' 6:09 ',' 9:49 ', 414), (' 7:57 ', '11: 15', 347), ( '9:49', '13: 51
```

', 229), ('10: 51', '14: 16 ', 256), ('12: 20', '16: 34 ', 500), (' 14:20 ', '17: 32', 332), ('15: 49 ', '20: 10', 497), ('17: 14 ', '20: 59', 277), ('18: 44 ', '22: 42 ', 351), ('19: 57', '23: 15 ', 512)], (' LGA ',' ORD '): [(' 6:03 ',' 8:43 ', 219), ('7:50', '10: 08 ', 164), (' 9:11 ', '10: 42', 172), ('10: 33 ', '13: 11', 132) , ('12: 08 ', '14: 47', 231), ('14: 19 ', '17: 09', 190), ('15: 04 ', '17: 23', 189), ('17: 06 ', '20: 00', 95), ('18: 33 ', '20: 22', 143), ('19: 32 ', '21: 25', 160)], (' ORD ',' LGA '): [(' 6:05 ',' 8:32 ', 174), (' 8:25 ', '10: 34', 157), ('9:42', '11 : 32 ', 169), ('11: 01', '12: 39 ', 260), ('12: 44', '14: 17 ', 134), ('14: 22', '16: 32 ', 126), ('15: 58', '18: 40 ', 173), ('16: 43', '19: 00 ', 246), ('18: 48', '21: 45 ', 246), ('19: 50 ', '22: 24', 269)], ('MIA', 'LGA'): [('6:25', '9:30', 335), ('7 : 34 ',' 9:40 ', 324), (' 9:15 ', '12: 29', 225), ('11: 28 ', '14: 40', 248), ('12: 05 ', '15: 30', 330), ('14: 01 ', '17:24', 338), ('15:34', '18:11', 326), ('17:07', '20:04', 291), ('18:23', '21:35', 134), ('19:53', '22:21', 173)], ('BOS', 'LGA'): [('6:17', '8:26', 89), ('8:04', '10:11', 95), ('9:45', '11:50', 172), ('11:16', '13:29', 83), ('12:34', '15:02', 109), ('13:40', '15:37', 138), ('15:27', '17:18', 151), ('17:11', '18:30', 108), ('18:34', '19:36', 136), ('20:17', '22:22', 102)]}

Словник рейсів має такий вигляд: {('LGA', 'CAK'): [('6:58', '9:01', 238), ('8:19', '11: 16 ', 122), ...], ...} - (аеропорт вильоту, прильоту), як ключ і (час вильоту, час прильоту, вартість), як значення. Заповніть його довільним способом, можна пошукати на сайтах авіа і заповнити реальною ситуацією))

Для подібних завдань необхідно визначитися зі способом подання потенційних рішень. Функції оптимізації, з якими ви незабаром ознайомитеся, досить загальні і застосовні до різних завдань, тому так важливо вибрати просте уявлення, яке не було б прив'язане до конкретного завдання про груповому подорожі. Дуже часто для цієї мети вибирають список чисел. Кожне число позначає рейс, яким вирішив летіти учасник групи.

Наприклад, в рішенні, представленому списком [1,4,3,2,7,3,6,3,2,4,5,3]

Сеймур (Seymour) летить першим рейсом з Бостона в Нью-Йорк і четвертим рейсом з Нью-Йорка в Бостон, а Френні (Franny) – третім рейсом з Далласа в Нью-Йорк і другим назад.

Цільова функція. Ключем до вирішення будь-якої задачі оптимізації є цільова функція, і саме її зазвичай найважче знайти. Мета оптимізаційного алгоритму полягає в тому, щоб знайти такий набір вхідних змінних, який мінімізує цільову функцію. Тому цільова функція повинна повертати значення, що показує, наскільки дане рішення незадовільно. Значення, що повертається повинно бути тим більше, чим гірше рішення.

Розглянемо кілька параметрів, які можна виміряти в прикладі з груповим подорожжю:

Ціна. Повна вартість усіх квитків або, можливо, середнє значення, зважене з урахуванням фінансових можливостей.

Час в дорозі. Сумарний час, проведений всіма членами сім'ї в польоті.

Час очікування. Час, проведений в аеропорту в очікуванні прибуття інших членів групи.

Час вильоту. Якщо літак вилітає рано вранці, це може збільшувати загальну вартість через те, що мандрівники не виспляться.

Час оренди автомобілів. Якщо група орендує машину, то повернути її слід до тієї години, коли вона була орендована, інакше доведеться платити за зайвий день.

Визначившись з тим, які змінні впливають на вартість, потрібно вирішити, як з них скласти одне число. У нашому випадку можна, наприклад, висловити в грошах час у дорозі або час очікування в аеропорту. Скажімо, кожна хвилина в повітрі еквівалентна $ 1 (інакше гово- ря, можна витратити зайві $ 90 на прямий рейс, що економить півтори години), а кожна хвилина очікування в аеропорту еквівалентна $ 0,50. Можна було б також приплюсувати вартість зайвого дня оренди машини, якщо для всіх має сенс повернутися в аеропорт до більш поз- днем годині.

Вказана нижче цільова функція shedule_cost бере до уваги повну вартість поїздки і загальний час очікування в аеропорту усіма членами сім'ї. Крім того, вона додає штраф $ 50, якщо машина повернута в більш пізній час, ніж орендована.

```
# Час в хвилинах
def get_minutes (t):
 x = time.strptime (t, '% H:% M')
 return x [3] * 60 + x [4]
def schedule_cost (sol):
 totalprice = 0
 latestarrival = 0
 earliestdep = 24 * 60
 for d in xrange (len (sol) / 2):
  # Отримати список прибувають і відбувають PEIC
  origin = peoples [d] [1]
  outbound = flights [(origin, destination)] [int (sol [d])]
  returnf = flights [(destination, origin)] [int (sol [d + 1])]

  # Повна ціна дорівнює сумі цін на квиток туди і назад
  totalprice + = outbound [2]
  totalprice + = returnf [2]

  # Знаходимо сами пізній приліт і сами ранніі виліт
  if latestarrival <get_minutes (outbound [1]): latestarrival = get_minutes
(outbound [1])
  if earliestdep> get_minutes (returnf [0]): earliestdep = get_minutes (returnf [0])

 # Все повинні чекати в аеропорту прибуття останнього учасника групи.
 # Зворотно все прибувають одночасно і повинні чекати свої PEIC.
 totalwait = 0
 for d in xrange (len (sol) / 2):
  origin = peoples [d] [1]
  outbound = flights [(origin, destination)] [int (sol [d])]
```

```
returnf = flights [(destination, origin)] [int (sol [d + 1])]
totalwait + = latestarrival - get_minutes (outbound [1])
totalwait + = get_minutes (returnf [0]) - earliestdep

# Для цього рішення потрібно оплачувати дополнітельниі день оренди?
# Якщо так, це обоідется в зайві $ 50!
if latestarrival> earliestdep: totalprice + = 50
return totalprice + totalwait
```

Логіка цієї функції дуже проста, але суть питання вона відображає. Поліпшити її можна декількома способами. Так, в поточній версії передбачається, що всі члени сім'ї виїжджають з аеропорту разом, коли прибуває найостанніший, а повертаються в аеропорт до моменту вильоту самого раннього рейсу. Можна вчинити по-іншому: якщо людині доводиться чекати дві години або довше, то він орендує окрему машину, а ціни і час очікування відповідно коригуються.

Випадковий пошук

Випадковий пошук – не найкращий метод оптимізації, але він дозволить нам ясно зрозуміти, чого намагаються досягти всі алгоритми, а також послужить еталоном, з яким можна буде порівнювати інші алгоритми.

Відповідна функція приймає два параметри. Domain – це список значень, що визначають кількість варіантів рейсів кожного з учасників подорожі. Важливо що вони відсортовані в тому ж порядку, що і самі учасники, спочатку йдуть в Урюпінськ, потім звідти. Довжина рішення збігається з довжиною цього списку.

Другий параметр, costf, – це цільова функція; в нашому прикладі в цій якості використовується schedule_cost. Вона передається у вигляді параметра, щоб алгоритм можна було використовувати повторно і для інших завдань оптимізації. Алгоритм випадковим чином генерує 1000 гіпотез і для кожної

викликає функцію costf. Повертається найкраща гіпотеза (з мінімальною вартістю).

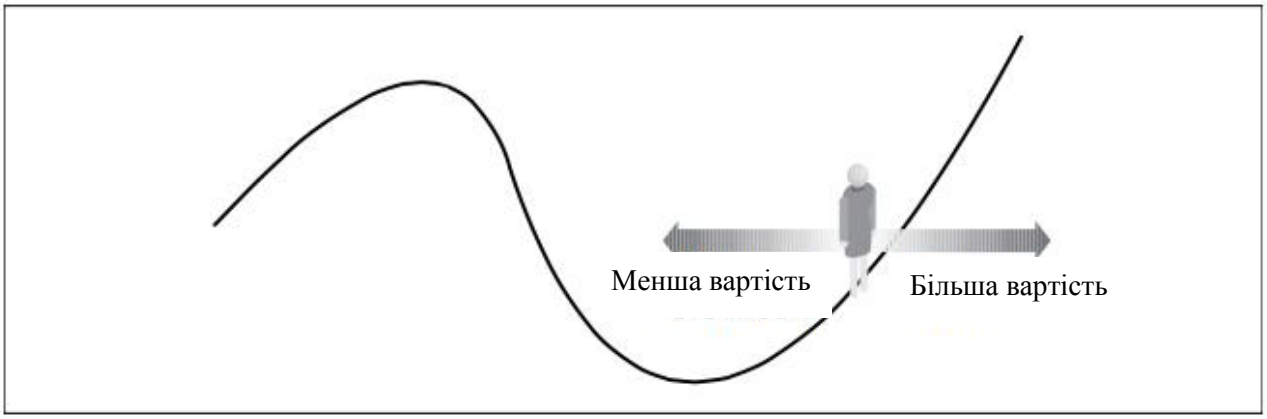
```
domain = []
for people in peoples:
 domain.append (len (flights [(people [1], destination)]) - 1)
 domain.append (len (flights [(destination, people [1])]) - 1)
print domain

def random_optimize (domain, costf):
 best = 999999999
 bestr = None

 for i in xrange (0 1000):
  # Вибрати випадкові рішення
  r = [random.randint (0, domain [i]) for i in xrange (len (domain))]

  # Get the cost
  cost = costf (r)

  # Порівняти з вартістю найкращого знайденного до цього моменту рішення
  if cost <best:
   best = cost
   bestr = r
 return r, best

result, score = random_optimize (domain, schedule_cost)
print result, score
```

З 10 запусків по 1000 варіантів була отримана найкраща сума у 3208 $, яка слугуватиме для порівняння з іншими алгоритмами.

Алгоритм спуску з гори

Випадкове апробування рішень дуже неефективно, тому що нехтує вигодами, які можна отримати від аналізу вже знайдених оптимальних рішень. У нашому прикладі можна припустити, що розклад з низькою повною вартістю схоже на інші розклади з низькою вартістю.

Альтернативний метод випадкового пошуку називається алгоритмом спуску з гори (hill climbing). Він починає з випадкового рішення і шукає кращі рішення (з меншим значенням цільової функції) по сусідству. Можна провести аналогію зі спуском з гори.



Уявіть, що чоловічок на малюнку - це ви і виявилися в цьому місці з волі випадку. Ви хочете дістатися до найнижчої точки, щоб знайти воду. Для цього ви, напевно, озирніться і направитеся туди, де схил найкрутіший. І будете рухатися в напрямку найбільшої крутизни, поки не дійдете до точки, де місцевість стає рівною або починається підйом.

Застосуємо цей підхід. Почнемо з випадково обраного розкладу і переглянемо всі розклади в його околиці. В даному випадку це означає перегляд таких розкладів, для яких одна людина вибирає рейс, що вилітає трохи раніше чи трохи пізніше. Для кожного з сусідніх розкладів обчислюється вартість, і

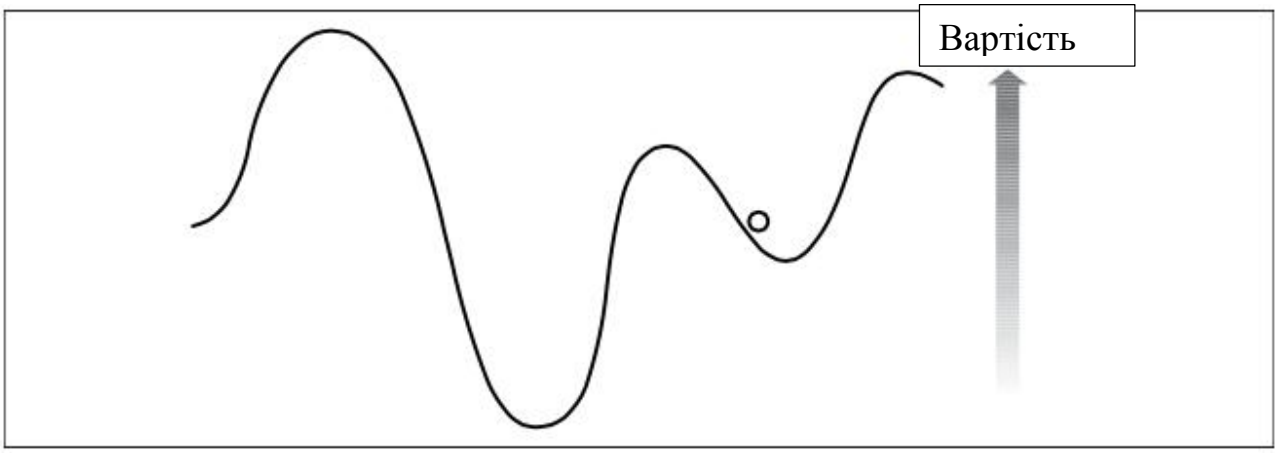
розклад з найменшою вартістю стає новим рішенням. Цей процес повторюється і завершується, коли жодна з сусідніх розкладів не дає поліпшення вартості.

```
def hill_climb (domain, costf):

 # Вибрати випадковій рішення
 sol = [random.randint (0, domain [i]) for i in xrange (len (domain))]
 best = costf (sol)

 # Главниі цикл
 is_stop = False
 while not is_stop:
  # Створити список сусідніх вирішенні
  neighbors = []
  for j in xrange (len (domain)):

   # Відходимо на один крок в кожному напрямку
   if 0 <sol [j] <9:
    neighbors.append (sol [0: j] + [sol [j] +1] + sol [j + 1:])
    neighbors.append (sol [0: j] + [sol [j] -1] + sol [j + 1:])

   if 0 == sol [j]:
    neighbors.append (sol [0: j] + [sol [j] +1] + sol [j + 1:])

   if sol [j] == domain [j]:
    neighbors.append (sol [0: j] + [sol [j] -1] + sol [j + 1:])

  # Шукаємо найкраще з сусідніх вирішенні

  is_stop = True
```

```
 for j in xrange (len (neighbors)):
  cost = costf (neighbors [j])
  if cost <best:
   is_stop = False
   best = cost
   sol = neighbors [j]

 return sol, best

result, score = hill_climb (domain, schedule_cost)
print result, score
```

Для вибору початкового рішення ця функція генерує випадковий список чисел із заданого діапазону. Сусіди поточного рішення шукаються шляхом відвідування кожного елемента списку та створення двох нових списків, в одному з яких цей елемент збільшений на одиницю, а в іншому зменшений на одиницю. Найкраще з сусідніх рішень стає новим рішенням.

Насправді, в даному випадку це рішення дасть результати так собі. Хоча, в більшості випадків, краще ніж 1000 випадкових варіантів. Однак у алгоритму спуску з гори, є один суттєвий недолік.



З малюнка видно, що, спускаючись по схилу, ми обов'язково знайдемо найкраще можливе рішення. Знайдене рішення буде локальним мінімумом,

тобто найкращим з усіх в найближчій околиці, але це не означає, що воно взагалі краще. Рішення, краще серед усіх можливих, називається глобальним мінімумом, і саме його хочуть знайти всі алгоритми оптимізації. Один з можливих підходів до вирішення проблеми називається спуском з гори з випадковим перезапуском. У цьому випадку алгоритм спуску виконується кілька разів з випадковими початковими точками в надії, що якесь із знайдених рішень буде близько до глобального мінімуму. Так само, всі посилюється тим, що у такій функції як наша, велика імпульсивність (багато локальних мінімумів і максимумів), дуже НЕ лінійна швидкість росту в різних точках, в загальному дуже багато чинників, з урахуванням того, що ми розглядаємо функцію в 12 вимірному просторі і в кожному з них вона поводиться по-різному ...

Алгоритм імітації відпалу

Алгоритм імітації відпалу навіяний фізичними аналогіями. Відпалом називається процес нагрівання зразка з подальшим повільним охолодженням. Оскільки спочатку атоми змушують пострибати, а потім поступово «відпускають віжки», то вони переходять в нову низько-енергетичну конфігурацію.

Алгоритмічна версія відпалу починається з вибору випадкового рішення задачі. У ній використовується змінна, що інтерпретується як температура, початкове значення якої дуже велике, але поступово зменшується. На кожній ітерації випадковим чином вибирається один з параметрів рішення та змінюється в певному напрямку.

Тут є важлива деталь: якщо нова вартість нижче, то нове рішення стає поточним, як і в алгоритмі спуску з гори. Але, навіть якщо нова вартість вище, нове рішення все одно може стати поточним з певною ймовірністю. Так ми намагаємося уникнути ситуації скочування в локальний мінімум.

Іноді необхідно перейти до гіршого рішенням, щоб знайти найкраще. Алгоритм імітації відпалу працює, тому що завжди готовий перейти до кращого рішення, але ближче до початку процесу погоджується прийняти і найгірше. У

міру того як процес розвивається, алгоритм погоджується на найгірше рішення все з меншим бажанням, а в кінці роботи приймає тільки краще. Імовірність того, що буде прийнято рішення з більш високою вартістю, розраховується за формулою:

$$p = EXP\left(-\frac{\text{висока вартість+низька вартість}}{\text{температура}}\right)$$

Оскільки температура (готовність прийняти найгірше рішення) спочатку дуже висока, то показник ступеня близький до 0, тому ймовірність дорівнює майже 1. У міру зменшення температури різниця між високою вартістю і низькою вартістю стає більш значущою, а чим більше різниця, тим нижча ймовірність, тому алгоритм з усіх найгірших рішень буде вибирати ті, що лише небагато чим гірше поточного.

```
def annealing_optimize (domain, costf, T = 10000.0, cool = 0.99, step = 1):

 # Вибрати випадковій рішення
 vec = [random.randint (0, domain [i]) for i in xrange (len (domain))]

 while T> 0.1:
  # Вибрати один з індексів
  i = random.randint (0, len (domain) -1)

  # Вибрати напрямок зміни
  dir = random.randint (-step, step)

  # Створити нові список, в якому одне значення змінено
  vecb = vec [:]
  vecb [i] + = dir
  if vecb [i] <0: vecb [i] = 0
  elif vecb [i]> domain [i]: vecb [i] = domain [i]
```

```
    # Обчислити поточну і нову вартість
    ea = costf (vec)
    eb = costf (vecb)
    p = pow (math.e, (-eb-ea) / T)

    # Нове рішення краще? Якщо немає, Метн кістки
    if (eb <ea or random.random () <p):
     vec = vecb

    # Зменшити температуру
    T = T * cool
   return vec, eb

result, score = annealing_optimize (domain, schedule_cost)
print result, score
```

На кожній ітерації як і випадково вибирається одна з змінних рішення, а dir випадково вибирається між -step і step. Обчислюється вартість поточного рішення і вартість рішення, що виходить зміною змінної на величину step.

В p, обчислюється ймовірність, яка зменшується зі зменшенням T. Якщо величина, випадково обрана між 0 і 1, виявляється менше обчисленої ймовірності або якщо нове рішення краще поточного, то нове рішення стає поточним. Цикл триває, поки температура майже не досягне нуля, причому кожен раз температура множиться на швидкість охолодження.

З приводу краще чи ні результати, сказати не можу, швидше за все, приблизно однакові.

Генетичні алгоритми

Батьком генетичних алгоритмів прийнято вважати вченого-теоретика Джона Холланда (John Holland), який в 1975 році написав книгу «Adaptation in Natural and Artificial Systems» (видавництво Мічиганського університету). Але коріння цих робіт сягають до біологів 1950-х років, які намагалися моделювати еволюцію на комп'ютерах. З тих пір генетичні алгоритми та інші методи оптимізації використовувалися для вирішення найширшого кола завдань, в тому числі:

Знаходження форми концертного залу з оптимальними акустичними характеристиками.

Проектування оптимальної форми крила надзвукового літака.

Складання оптимальної бібліотеки хімічних речовин для синтезу потенційних ліків.

Автоматичного проектування мікросхем для розпізнавання мови.

Рішення цих задач можна представити у вигляді списків чисел. Це спрощує застосування до них генетичних алгоритмів або методу імітації відпалу.

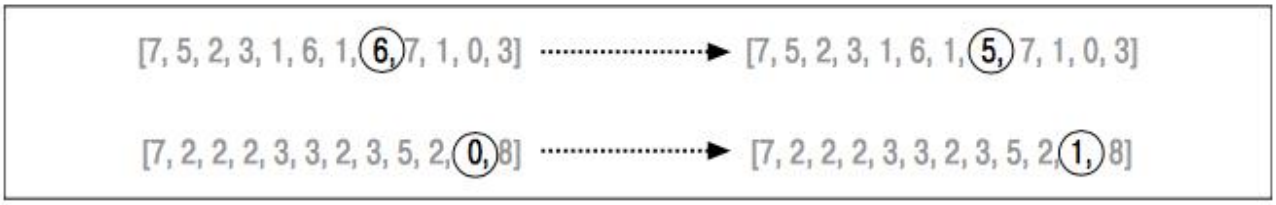
Ще один клас методів оптимізації, також навіяний природою, називається генетичними алгоритмами. Принцип їх роботи полягає в тому, щоб створити набір випадкових рішень, який називається популяцією. На кожному кроці оптимізації цільова функція обчислюється для всієї популяції, в результаті чого виходить ранжований список рішень.

Рішення	**Вартість**
[7, 5, 2, 3, 1, 6, 1, 6, 7, 1, 0, 3]	4394
[7, 2, 2, 2, 3, 3, 2, 3, 5, 2, 0, 8]	4661
...	...
[0, 4, 0, 3, 8, 8, 4, 4, 8, 5, 6, 1]	7845
[5, 8, 0, 2, 8, 8, 8, 2, 1, 6, 6, 8]	8088

Проранжирувавши рішення, ми створюємо нову популяцію, яка називається наступним поколінням. Спочатку в нову популяцію включаються

найкращі рішення з поточної. Цей процес називається елітизмом. Крім них, в наступну популяцію входять абсолютно нові рішення, що виходять шляхом модифікації найкращих.

Модифікувати рішення можна двома способами. Простіший називається мутацією; зазвичай це невелике, просте, випадкове зміна існуючого рішення. У нашому випадку для мутації досить вибрати одну з змінних рішення і зменшити або підвищити її.



Інший спосіб називається схрещуванням (або кросовером). Полягає він у тому, що ми беремо якісь два з кращих рішень і якось комбінуємо їх. У нашому прикладі досить взяти випадкове число елементів з одного рішення, а інші - з іншого, як показано.

Розмір нової популяції зазвичай збігається з розміром попередньої, а створюється вона шляхом випадкових мутацій і схрещувань кращих рішень. Потім цей процес повторюється – нова популяція ранжируется і створюється чергове покоління. Так триває задане число раз або до тих пір, поки на протязі декількох поколінь не спостерігається ніяких поліпшень.

Pop_size - розмір популяції

Mutprob - чим менше, тим рідше беремо мутацію і частіше схрещування

Elite - Частка особеq в популяції, що вважаються хорошими рішеннями і переходять в наступне покоління

Maxiter - Кількість поколінь

def genetic_optimize (domain, costf, pop_size = 50, step = 1, mutprob = 0.2, elite = 0.2, maxiter = 100):

Мутація

```
def mutate (vec):
 i = random.randint (0, len (domain) -1)
 if vec [i] <domain [i]:
  return vec [0: i] + [vec [i] + step] + vec [i + 1:]
 else:
  return vec [0: i] + [vec [i] -step] + vec [i + 1:]

# Схрещування
def crossover (r1, r2):
 i = random.randint (1, len (domain) -2)
 return r1 [0: i] + r2 [i:]

# Створюємо першу популяцію
pop = []
for i in xrange (pop_size):
 vec = [random.randint (0, domain [i]) for i in xrange (len (domain))]
 pop.append (vec)

# Скільки кращих залишаємо з кожного покоління
topelite = int (elite * pop_size)

for i in xrange (maxiter):
 scores = [(costf (v), v) for v in pop]
 scores.sort ()
 ranked = [v for (s, v) in scores]

 # Спочатку включаємо тільки переможці
 pop = ranked [0: topelite]
```

```
# Додаємо особин, отриманих мутацією і схрещуванням перемогли батьків
while len (pop) <pop_size:
 if random.random () <mutprob:
  # Мутація
  c = random.randint (0, topelite)
  pop.append (mutate (ranked [c]))
 else:
  # Схрещування
  c1 = random.randint (0, topelite)
  c2 = random.randint (0, topelite)
  pop.append (crossover (ranked [c1], ranked [c2]))
return scores [0] [1], scores [0] [0]
```

Загалом, ці методи вже запрограмовані в різних бібліотеках, опис яких є на сайті https://wiki.python.org/moin/PythonForOperationsResearch.

Дамо описи деяких бібілотек:

- APM Python – це безкоштовна програма оптимізації через веб-сервіс. Задача нелінійного програмування надсилається на сервер APMonitor і результати повертаються до локального скрипту Python. Автоматичне завантаження веб-інтерфейсу допомагає візуалізувати рішення, зокрема проблеми динамічної оптимізації, що включають диференціальні та алгебраїчні рівняння. Типовими розв'язками є APOPT, BPOPT та IPOPT. Попередньо налаштовані режими включають оптимізацію, оцінку параметрів, динамічне моделювання та нелінійний контроль.
- Coopr - Програмний продукт Coopr інтегрує різні пакети, пов'язані з оптимізацією Python.
- CVOXPT – це безкоштовний пакет програм для опуклої оптимізації на основі мови програмування Python. Він може бути використаний

з інтерактивним інтерпретатором Python, у командному рядку шляхом виконання сценаріїв Python або інтегрований в інше програмне забезпечення через модулі розширення Python. Його основна мета полягає в тому, щоб зробити розробку програмного забезпечення для опуклих додатків оптимізації простими, будуючи велику стандартну бібліотеку Python та сильні сторони Python як мову програмування високого рівня.

- OpenOpt (ліцензія: BSD) містить з'єднання з десятками розв'язувачів та має деякі власні Python-написані, наприклад нелінійний розв'язувач з визначеною точністю: interalg, графічний вихід збіжності та деяка більш чисельна оптимізація "ПОВИННІ БУТИ". Також OpenOpt може вирішити проблеми з функціями FuncDesigner з автоматичною диференціацією, які зазвичай працюють швидше і дають більш точні результати, ніж апроксимація похідних від звичайних відмінностей.
- PuLP – це оптимізація методом лінійного прорамування моделювач, написаний на python. PuLP може генерувати файли MPS або LP і викликати GLPK, COIN CLP / CBC, CPLEX та GUROBI для вирішення лінійних проблем.
- Pyomo – Пакет оптимізації Python для моделювання об'єктів (Pyomo) – це інструмент з відкритим кодом для моделювання додатків для оптимізації в Python. Pyomo може бути використаний для визначення символічних проблем, створення конкретних випадків проблеми та вирішення цих випадків з використанням стандартних розв'язувачів. Pyomo надає можливості, яка зазвичай пов'язана з мовами алгебраїчного моделювання, такими як AMPL, AIMMS та GAMS, але об'єкти моделювання Pyomo вбудовані в повнофункціональну мову програмування високого рівня з насиченим набором допоміжних бібліотек. Pyomo використовує можливості бібліотеки програмного забезпечення Coopr, яка

інтегрує пакети Python для визначення оптимізаторів, моделювання додатків для оптимізації та керування обчислювальними експериментами.

- scipy.optimize – деякі розв'язувачі, написані або пов'язані розробниками SciPy.
- pyOpt – це пакет для формулювання та вирішення нелінійних обмежених оптимізаційних завдань у ефективній, багаторазовій і портативній формі (ліцензія: LGPL).

Зверніть увагу на те, що слова «ліцензія» в опису, означають, що використання цих пакетів можливо тільки з дозволу конкретної фірми. А отже, може бути платним.

Для прикладу розберемо детальніше використання функції solve з пакету pulp. Для цього вирішимо наступну оптимізаційну задачу.

Витрати на рекламу в місяць не повинні перевищувати 10 000 грошових одиниць (ГО). Хвилина радіореклами коштує 5 ГО, а телереклами 90 ГО Фірма має намір використовувати радіорекламу в три рази частіше ніж телерекламу. Практика показує, що 1 хвилина телереклами забезпечує обсяг продажів в 30 разів більший ніж 1 хвилина радіореклами. Знайти оптимальні витрати на кожен вид реклами.

Позначимо як x1 – витрати на телерекламу, x2 – витрати на радіорекламу.

Тоді, цільова функція, що максимізує ефективність від реклами прийме вигляд $30x1+x2 \rightarrow \max$

а обмеження:

– за обсягом витрат: $90x1+5x2 \leq 10\ 000$

– за часом реклами: $x2=3x1$

На наступному рисунку покажемо роботу функції solve. Коментарі в тексті програми пояснюють особливості її використання.

```
from pulp import *
import time
start = time.time() # Визначаємо час на початок оптимального розрахунку

# Встановлюємо відповідність змінних у програмі їх умовним позначенням
# в нашому випадку вони однакові, але можна було написати в лапках
# замість x1 - витрати на радіорекламу
# замість x2 - витрати на телерекламу
# параметр LowBound встановлює нижню межу значень змінних
x1 = pulp.LpVariable("x1", lowBound=0)
x2 = pulp.LpVariable("x2", lowBound=0)

#  Вмикаємо програму знайдення максимально можливого значення
problem = pulp.LpProblem('0', pulp.LpMaximize)


problem += 30*x1 + x2, '-->max' #"Цільова функція*"
# Текст в лапках після формули та коми обов'язковий

# Обмеження. Цифра в лапках піцля формули та коми показують номер обмежегння
problem += 90*x1 + 5*x2 <= 10000, '1' #"Перше обмеження"
problem += x2 == 3*x1, '2' #"Друге обмеження"

# запускаємо процес вирішення оптимальної лінійної задачі
problem.solve ()

print ("Результат оптимального розрахунку: ")

# Друкуємо оптимальні значення змінних
for variable in problem.variables():
        print (variable.name, "=", variable.varValue)

print ("Прибуток:")
print (value(problem.objective))

stop = time.time ()
print ("Час на вирішення оптимізаційної задачі:")

# Друкуємо різнюцю в часі між почсатком розрахунку та його закінченням
print(stop - start)
```

Результат розрахунку показано нижче

```
 RESTART: D:\02 Навчальний процес\00 Еле
оптиміз2.py
Результат оптимального розрахунку:
x1 = 95.238095
x2 = 285.71429
Прибуток:
3142.85714
Час на вирішення оптимізаційної задачі:
0.09360027313232422
```

Розглянемо тепер транспортну задачу, представлену в наступній таблиці. Як видно з неї, це задача відкритого типу з перевищенням запитів на товар.

			Споживачі		
			1	2	3
		Запаси\ Запити на постачання	74	40	36
Постачальники	1	20	7	3	6
	2	45	4	8	2
	3	30	1	5	9

Введемо умовні позначення для обсягів поставки від першого постачальника до 3-х стоживачів, як x11, x12, x13. Відповідно для другого та третього ці позначення будуть x21, x22, x23, x31, x32, x33.

Тоді вартість перевезення всіх товарів буде знайдена як

F(x)

→ min

Обмеження для постачальників означають, що весь товар має бути вивезений зі складів:

$$x11 + x12 + x13 = 20;$$

$$x21 + x22 + x23 = 45;$$

$$x31 + x32 + x33 = 30.$$

Обмеження для споживачів, навпаки, визначають, що запит на товари може виконаний не повністю:

$$x11 + x21 + x31 \leq 74;$$

$$x12 + x22 + x32 \leq 40;$$

$$x13 + x23 + x33 \leq 36.$$

Варто не забувати, що функція solve вирішує тільки задачі максимізації, тому цільову функцію потрібно записувати зі знаком мінус.

Нижче на рисунку показано програму, що заходить оптимальне рішення.

```
from pulp import *
import time
start = time.time()

x11 = pulp.LpVariable("x11", lowBound=0)
x12 = pulp.LpVariable ("x12", lowBound=0)
x13 = pulp.LpVariable("x13", lowBound=0)
x21 = pulp.LpVariable("x21", lowBound=0)
x22 = pulp.LpVariable("x22", lowBound=0)
x23 = pulp.LpVariable("x23", lowBound=0)
x31 = pulp.LpVariable ("x31", lowBound=0)
x32 = pulp.LpVariable("x32", lowBound=0)
x33 = pulp.LpVariable("x33", lowBound=0)

problem = pulp.LpProblem('0',pulp.LpMaximize)

problem += - (7*x11 + 3*x12 + 6*x13 + 4*x21 + 8*x22 + 2*x23 + 1*x31 + 5*x32 + 9

problem +=x11 + x12 + x13  == 20, '1'
problem +=x21 + x22 + x23  == 45, '2'
problem +=x31 + x32 + x33  == 30, '3'

problem +=x11 + x21 + x31 <= 74, '4'
problem +=x12 + x22 + x32  <= 40, '5'
problem +=x13 + x23 + x33  <= 36, '6'

problem.solve()

print ("Чисельні значення змінних")
for variable in problem.variables():
        print (variable.name, "=", variable.varValue)
print ("Оптимальна вартість перевезень:")
print (abs(value(problem.objective)))
stop = time.time()
print ("Час на вирішення оптимізаційної задачі")
print(stop - start)
```

Результат оптимального розрахунку показав, що деякі замовлення не будуть виконані

```
 RESTART: D:\02 Навчальний процес\00 Електрс
Оптиміз 1.py
Чисельні значення змінних
x11 = 0.0
x12 = 20.0
x13 = 0.0
x21 = 9.0
x22 = 0.0
x23 = 36.0
x31 = 30.0
x32 = 0.0
x33 = 0.0
Оптимальна вартість перевезень:
198.0
Час на вирішення оптимізаційної задачі
0.171600341796875
```

2.7.23. Компіляція текстів програм у *.exe файл

Компіляція програми на Python під Windows може знадобиться, коли програмою буде користуватися людина, у якого на комп'ютері не встановлено Python. Недоліком даного способу є те, що крім самої програми, в папці буде присутній багато файлів, необхідних для її запуску на машинах без Python. Проте, компіляція в .exe дозволяє запускати програму так, як звикли користувачі Windows.

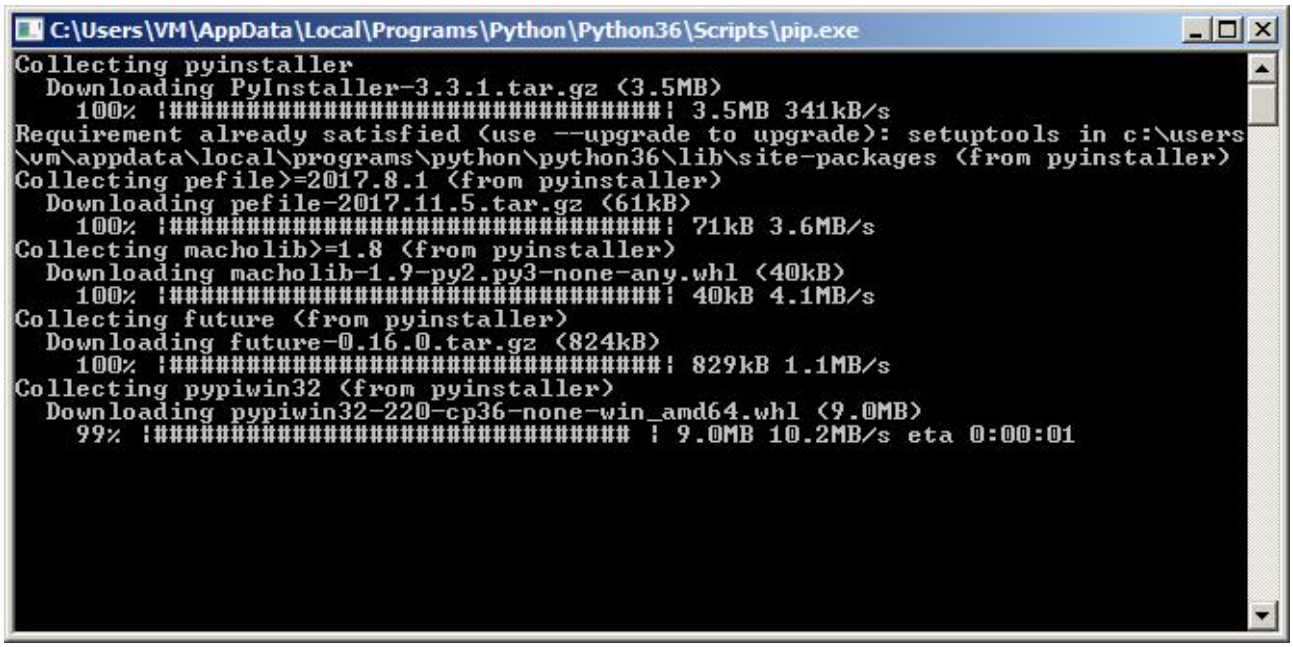
Для цього є кілька способів. Спочатку розглянемо найбільш простий – за допомогою бібліотеки бібліотеки pyinstaller

Дамо в командному рядку, як це описано у п. 2.2.16, дві команди:

pip install wheel

pip install pyinstaller

На екрані з’явиться протокол процесу встановлення цих програм.



Після установки можна скомпілювати програму, яка поки що знаходиться у файлі з розширенням *.py. Створимо, наприклад, на диску C: \ якусь папку з простим ім'ям. Нехай, це буде 1. Помістимо в цю папку скрипт,

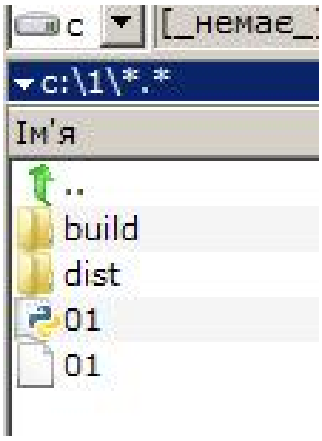
який ми хочемо скомпілювати – наприклад, 01.py. Тепер зайдемо в командний рядок і дамо команду:

pyinstaller --onedir --onefile --name=01 "C:\1\01.py"

Тут слово name означає те ім'я, яке матиме файл з розширенням exe, а текст в лапках означає повний шлях до файлу та ім'я файлу з розширенням py, який необхідно скомпілювати. Якщо ця команда не спрацює, необхідно вказати повний шлях до цієї команди ось так:

C:\Users\VM\AppData\Local\Programs\Python\Python36\Scripts\pyinstaller --onedir --onefile --name=01 "C:\1\01.py"

Після закінчення роботи pyinstaller в теці, де була тільки одна програма з'явиться декілька нових тек. Файл з прикладу 01.exe буде знаходитися у теці **dist.**



Інколи pyinstaller не справляється з компіляцією складних скриптів. Тому розглянемо альтернативний спосіб компіляції Python скриптів за допомогою cx_freeze. Для цього отримуємо потрібну версію програми за адресою http://www.lfd.uci.edu/~gohlke/pythonlibs/#cx_freeze .

Щоб почати отримання файлу, досить тільки клацнути на необхідну версію. Варто також звернути увагу на розрядність вашої операційної системи: 32 чи 64 біта?

cx_Freeze, a set of scripts and modules for freezing Python scripts into executable.

cx_Freeze-5.1.1-cp27-cp27m-win32.whl
cx_Freeze-5.1.1-cp27-cp27m-win_amd64.whl
cx_Freeze-5.1.1-cp34-cp34m-win32.whl
cx_Freeze-5.1.1-cp34-cp34m-win_amd64.whl
cx_Freeze-5.1.1-cp35-cp35m-win32.whl
cx_Freeze-5.1.1-cp35-cp35m-win_amd64.whl
cx_Freeze-5.1.1-cp36-cp36m-win32.whl
cx_Freeze-5.1.1-cp36-cp36m-win_amd64.whl
cx_Freeze-5.1.1-cp37-cp37m-win32.whl
cx_Freeze-5.1.1-cp37-cp37m-win_amd64.whl

Перед установкою програми cx_Freeze, необхідно встановити програму wheel, як це описано вище.

Для початку установки варто скопіювати скачаний файл в корінь диска D, щоб легко прописати шлях до нього. Далі пройдіть в командний рядок Windows, і наберіть команду pip install D:\ cx_Freeze-5.1.1-cp37-cp37m-win32.whl замість імені файлу з розширенням whl підставте ваш варіант. Ця команда встановить cx_Freeze на ваш комп'ютер. Тепер в каталозі з програмою яку потрібно скомпілювати створіть файл "setup.py", з таким змістом

```
from cx_Freeze import setup, Executable
setup (
    name = "Freelance",
    version = "1.0",
    description = "Freelance Parser",
    executables = [Executable ( "01.py")]    )
```

Тут замість 01.py потрібно вставити ім'я вашого скрипта на Python який ви компілюєте. Перейшовши в командному рядку в каталог з вашим скриптом дайте команду python.exe setup.py build.

Після виконання програми з'явиться новий каталог build, в якому, серед купи необхідних для роботи бібліотек з’явиться файл 01.exe.

1.8. Індивідуальне завдання №2

Тема роботи: Вирішення економічних задач з обробки великих масивів економічних даних, що містяться в Інтернеті, застосовуючи можливості програмного комплексу «Python».

Мета роботи: Набуття знань щодо використання програмного комплексу «Python» при вирішенні економічних задач.

Задача №1. Банківська позика

Потрібно розрахувати вартість кредиту, тобто треба обчислити, скільки доведеться платити в місяць по позиці та скільки всього віддати грошей банку за весь період.

Місячна виплата по позиці обчислюється за такою формулою:

$m = (s * p * (1 + p)^n) / (12 * ((1 + p)^n - 1))$.

де:

m – розмір місячної виплати;

s – сума позики (кредиту) (№ залікової книжки);

p – відсоток банку, виражений в частках одиниці (тобто якщо 20%, то буде 0.2) (поточний рік-2000).

n – кількість років, на які береться позика (№ по списку у журналі).

Параметри s, p та n – вводяться користувачем.

Результат: m – розмір місячної виплати; *sum* – загальна сума за весь період. Округлити нецілe число до двох знаків після коми.

Задача №2. Логічні оператори, булевий тип даних (Python)

Тема: Робота з текстовими даними та умовними операторами

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

Напишіть програму, що визначає:

1) який з двох введених рядків довший,
2) чи введено порожній рядок,
3) чи рядки однакові за довжиною,
4) чи рядки однакові за вмістом,
5) яке з двох введених чисел більше,
6) чи буде від'ємною сума введених чисел,

за допомогою логічних функцій.

Користувачу подається запит, наприклад:

```
Введіть перший рядок: ty
Введіть другий рядок: kl;
Введіть перше число: 6.8
Введіть друге число: 8
----------
Перший рядок більший за другий: True
Числа не рівні: True
Перший рядок більший другого та числа не рівні: True
Сума чисел більше нуля АБО ні один з рядків не пустий: True
```

В Python є прості логічні оператори (=,! =, <,>, <=,> =) і складні (*and, or, not*). Всі логічні оператори, за винятком *not*, є бінарними. Це означає, що зліва і праворуч від них повинні стояти вирази. За допомогою логічних операторів ці вирази так чи інакше порівнюються між собою.

Результат логічних операцій має булевий тип даних (вбудований *class 'bool'* в Python), тобто може приймати лише два значення – "істина" та "неправда". Потрібно бути обережним, порівнюючи між собою різні типи даних, так як це не завжди можливо. Наприклад, не можна порівнювати числа і рядки, але дробові та цілі числа - можна.

У складних логічних виразах потрібно враховувати послідовність операцій. Якщо немає впевненості, яка операція має пріоритет, то краще використовувати дужки.

Задача №3. Конкатенація і повторення рядків (Python)

Тема: Робота з текстовими даними та форматування

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

В Python над двома рядками можна виконати операцію, що позначається знаком +. Однак, на відміну від чисел, виконується не складанням (що для рядків в принципі неможливо), а з'єднання, тобто до кінця першого рядка додається другий. По-іншому операція з'єднання рядків називається *конкатенацією*.

Крім того, в Python є операція повторення (мультиплікації) рядків. Вона позначається знаком * (також як операція множення для чисел). При повторенні рядки з одного боку від знаку * ставиться рядок, а з іншого число, що позначає кількість повторів. При цьому не важливо, який об'єкт з якого боку знаходиться (зліва від знаку можна писати число, а праворуч – рядок).

В одному вираженні можна поєднувати операції конкатенації і мультиплікація. При цьому більш високий пріоритет у операції повторення рядка.

Напишіть програму, в якій:

1) Користувач вводить два рядка та кількість повторів (Прізвище, Ім’я, По-батькові) та шифр групи,
2) Виконується конкатенація рядків (Прізвище, Ім’я, По-батькові+ “студент/ка групи”+шифр групи),
3) Пошук підрядка в рядку,
4) Повтор об’єднаних рядків задану кількість раз, з нового рядка;
5) Надрукувати символ “*” задану кількість раз у вигляді:

```
    *
   * *
  * * *
 * * * *
```

6) Надрукувати ініціали ПІБ, наприклад: Михайло Сергійович Нестеренко

```
M          M     CCCC     H          H
M M    M M     C          H          H
M    M     M     C          HHHHHHHH
M          M     C          H          H
M          M     CCCC     H          H
```

Задача №4. Обмін значень змінних.

Тема: Створення найпростіших алгоритмів, кортежі

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

Обмін значень двох змінних – це "дія", в результаті якого одна змінна приймає значення, рівне другою змінною величиною, а друга – першої. Припустимо, є дві змінні *a* і *b*. При цьому *a* = 5 і *b* = 6. В результаті обміну має стати *a* = 6 і *b* = 5.

Зрозуміло, що якщо спробувати зробити такий обмін "по простому", тобто спочатку першій змінній привласнити значення другої, а другій – значення першої, то нічого не вийде. Якщо виконати вираз *a* = *b*, то змінна *a* буде посилатися на число 6, також як і *b*. Число 5 буде втрачено, так як на нього вже не буде посилатися жодна змінна, і вираз *b* = *a* безглуздо, так як *b* буде присвоєно його ж поточне значення (6 в даному випадку).

Тому в багатьох мовах програмування (наприклад, Pascal) доводиться вводити третю змінну, що грає роль буфера (її іноді називають буферною змінної). У цій змінній зберігають значення першої змінної, потім першій змінній привласнюють значення другої, в нове значення для другої змінної беруть з буфера. Існує і інший варіант обміну без додаткової змінної, реалізуйте і його.

Також необхідно реалізувати обмін значень через кортежі.

Задача № 5. Програма "Вгадай число" з використанням тільки конструкції if-else (Python)

Тема: Робота з логічними операторами

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

Користувач загадує число від 1 до 5. Потрібно його відгадати, задавши якомога менше питань, і обмежитися тільки використання оператора розгалуження (зазвичай подібні завдання вирішують за допомогою циклу).

Щоб користувачеві задати менше питань, треба "розділити" діапазон чисел на дві по можливості рівні частини і визначити, в якій із них знаходиться шукане число.

Задача № 6. Програма "Вгадай число" з використанням тільки конструкції if-else (Python)

Тема: Створення алгоритмів для взаємодії з користувачем

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

Користувач загадує число від 1 до 20. Потрібно його відгадати, для пошуку використайте цикл та підрахуйте кількість спроб.

Задача № 7. Визначити індекси елементів масиву (списку), значення яких належать заданому діапазону (Python)

Тема: робота зі списками

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

Необхідно визначити індекси елементів списку, значення яких не менше заданого мінімуму і не більше заданого максимуму.

Нехай досліджуваний масив (список в Python) заповнюється випадковими числами в діапазоні від 0 до 99 (включно) і складається з 100 елементів.

Далі мінімум і максимум для пошуку значень задається користувачем.

Задача № 8. Робота з файлами

Тема: запис/зчитування даних в/з текстовий файл, *.csv файл.

- Робота з файлами:
  1) Створити файл *Resume.txt*
  2) Ввести у файл інформацію про претендента: ПІБ, кваліфікацію, телефон.
  3) Закрити файл *Resume.txt*
  4) Вивести на екран інформацію з файлу *Resume.txt*

- Скласти картотеку з файлів зі структурою:
  5) Запросити у користувача ввести шлях до файлу (*user_url*).
  6) Запросити у користувача ввести ПІБ (*user_name*).
  7) Запросити у користувача ввести кваліфікацію.
  8) Запросити у користувача ввести бажаний розмір заробітної платні.
  9) Запросити у користувача ввести контактний телефон.
  10) Створити файл з іменем *user_url_ user_name.txt* (Наприклад, *Lab8_Petrenko.txt*)
  11) Записати у файл інформацію про претендента.
  12) Закрити файл.

- Зчитати/вивести на екран інформацію про потрібного претендента за введеним ім’ям:
  13) Запросити з екрану ім’я претендента.
  14) Вивести про нього інформацію.

- Робота з csv файлами:
  15) Зчитати дані з файлу Data.csv
  16) Вивести на екран перші 10 записів.

УВАГА! Слідкуйте за режимами доступу до файлу, своєчасно закривайте відкриті файли.

Задача № 9. Малювання з використанням графіки Turtle

Тема: Використання бібліотеки turtle, що дозволяє малювати на екрані нескладні малюнки.

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

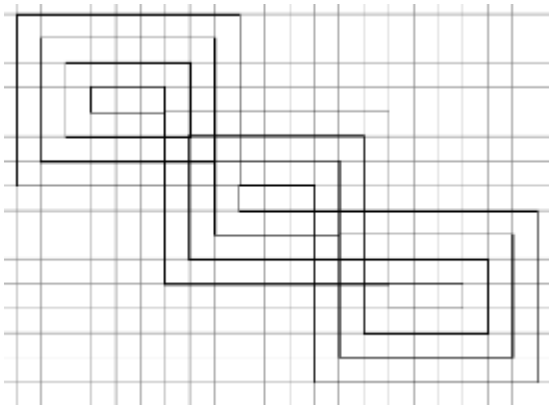
1) З використанням команд бібліотеки turtle намалювати багатокутник з N кутами, де N – число від 1 до 9 (остання цифра номеру за списком).

2) намалюйте на екрані рівносторонній трикутник. Намалюйте жовтий рівносторонній трикутник. Намалюйте зафарбований червоний рівносторонній трикутник.

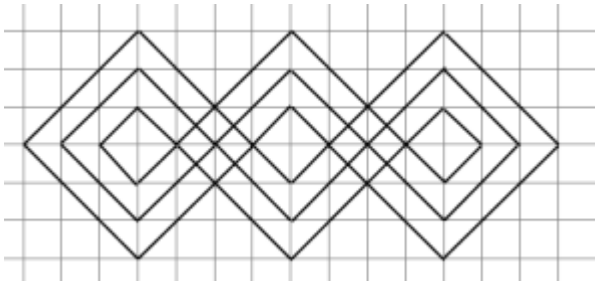
3) намалюйте на екрані квадрат з діагоналями. Намалюйте тільки діагоналі квадрата (два пересічних відрізка). Намалюйте квадрат, сторони якого не паралельні осям координат.

4) намалюйте:

Непарні номери за списком:



Парні номери за списком:



Всі програми, що працюють з черепашкою, повинні починатися з команд

```
import turtle
turtle.reset ()
```

а закінчуватися рядком

turtle._root.mainloop ()

Задача № 10. Робота з списками в Pyton

Тема: Списки.

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

Масиви в Пітоні називаються списками, тому що вони підтримують ряд додаткових операцій, що не властивих стандартним масивів. Заповніть список випадковими числами від 0 до *N*, де *N* – номер за списком у журналі (N>=5). Деякі елементи списку можна змінити.

import random

random.random ()

Потім:

1) Знайдіть найбільший елемент в списку.
2) Знайдіть найменший елемент в списку.
3) Знайдіть другий за величиною елемент у списку.
4) Знайдіть кількість елементів списку, рівних найбільшому.

Створити список студентів групи, відсортувати, додати ще одного студента. Перетворити в кортеж.

Задача № 11. Створення GUI додатку

Тема: Створити програмний додаток для перевірки валідності номеру кредитної/платіжної картки.

Обов'язкова наявність коментарів з умовами завдання перед написанням коду.

1) Тип проекту – Windows Form.

Користувач має ввести 16 цифр номеру картки, за натисканням кнопки «Check» має бути видано повідомлення або «The number *0123456789101112* is correct!!!» або «The number *0123456789101112* is NOT correct!!!». Приклад інтерфейсу програмного додатку наведено на рис. 1.

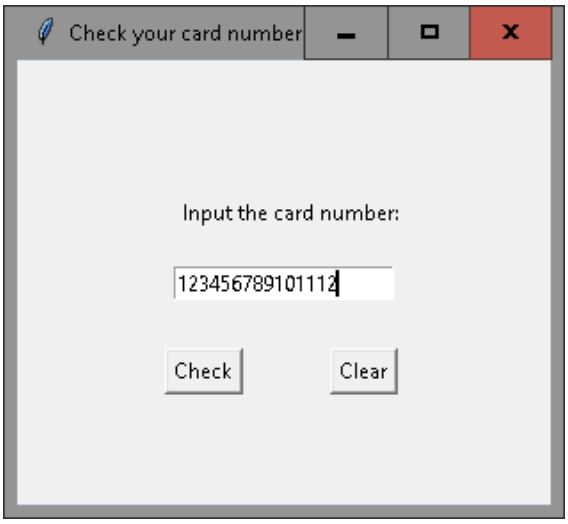


Рис. 1 Головна форма

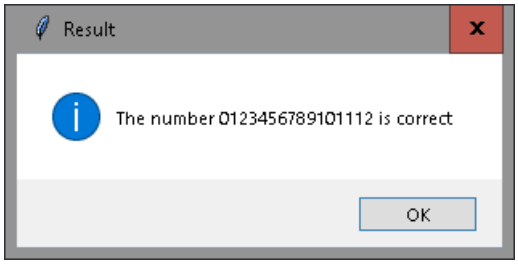


Рис. 2 Результат роботи програми

Перевірка введеного номера здійснюється за алгоритмом Луна (https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9B%D1%83%D0%BD%D0%B0).

Також, визначити платіжну систему:

VISA – 4
American Express – 3
MasterCard – 5
Maestro - 3 або 6
JCB International - 3
China UnionPay – 6
https://metanit.com/python/tutorial/9.2.php

2) Тип проекту – Windows Form

Запуск виконання завдань здійснювати при натисканні на відповідні кнопки.

Уважно тестувати задачі.

Завдання для парних номерів у списку групи в проекті з іменем ElectroenergiyaВашеПрізвище

Завдання для непарних номерів у списку групи виконати у проекті з іменем ConverterВашеПрізвище,

Практична робота. **Конвертер**

Постановка задачі. Програма **Конвертер** перераховує ціну з доларів в гривні. Демонструє використання компонентів TextBox и Label для введення і відображення числових даних. Форму програми приведено на рис. 1

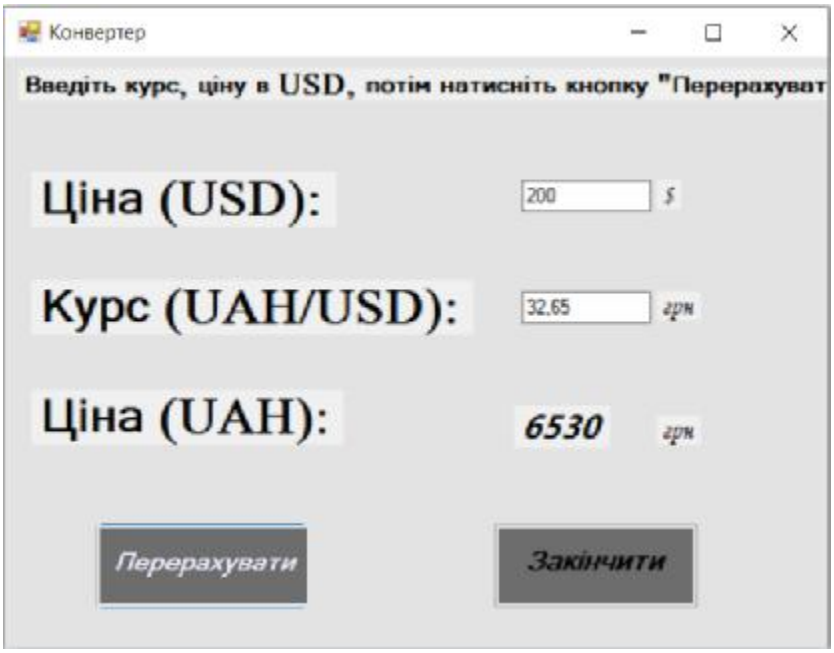


Рис.1. Інтрефейс додатку Конвертер

Програма **Електроенергія** визначає суму, яку потрібно заплатити користування електроенергією виходячи з показань лічильника. Форму програми наведено на рис. 2

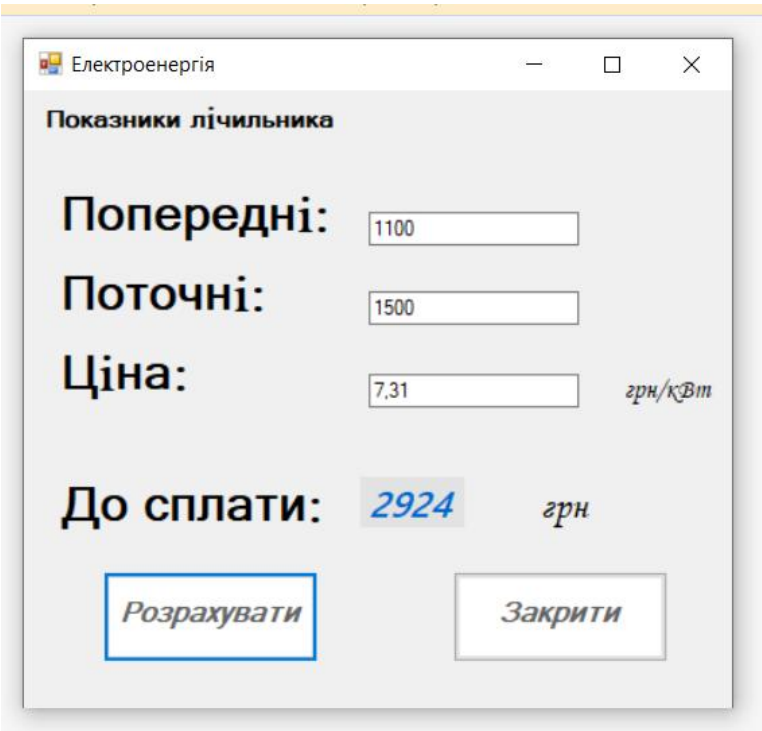


Рис.2. Інтрефейс додатку Електроенергія

Контрольні запитання

1. Що таке машинне навчання ?
2. Назвіть два підходи до машинного навчання.
3. Наведіть постановку задачі навчання за прецедентами.
4. Наведіть постановку задачі дедуктивного навчання.
5. Визначте місце науки про дані в середовищі інформаційних технологій.
6. Які задачі, вирішуються за допомогою машинного навчання ?
7. Наведіть особливості обробки та застосування big data в економічних дослідженнях.
8. Наведіть визначення *big data.*
9. Яким вимогам мають задовольняти системи для накопичення *big data* ?
10. Відмінність методів *big data* від бізнес-аналізу.
11. Відмінність *big data* від малих даних.
12. В чому полягає складність обробки *big data* ?
13. Основні властивості *big data.*

14. Назвіть основні мови програмування для обробки *big data.*

15. Яке програмне забезпечення використовують для збору, накопичення, обробки та візуалізації *big data*.

16. В чому полягає ключова концепція *noSQL* і *Hadoop* про обробці *big data* ?

17. Назвіть основні компоненти екосистеми *Hadoop.*

18. Що таке HDFS?

19. Поясніть схему роботи *MapReduce.*

20. Що таке *Kaggle?*

21. Наведіть основні принципи роботи з *Kaggle.*

22. Що таке цикли в програмі?

23. Чим тримісний вираз if / else у пакеті Pyton відрізняється від звичайної конструкції if / else?

24. Як працювати з бібліотеками у пакеті Pyton?

25. Чим множини відрізняються від словників у пакеті Pyton?

26. Яка бібліотека дозволяє робити статистичні обчислення у пакеті Python?

27. Як вибирати дані зі сторінок в Інтернеті, написаних у кодах XML і HTML за допомогою команд пакету Python?

У розділі студенти вивчили поняття методології обробки даних з застосуванням методів машинного навчання. Інструменти для обробки big data та застосування аналізу великих даних на прикладі змагання: «Аналіз ринку корзини Instacart». Розглянуто можливість застосування програмних пакетів Python для збору даних та розрахунків статистичних характеристик економічної інформації, доступ до якої забезпечує Інтернет, та забезпечення діяльності в галузі електронної комерції.

ВИСНОВКИ

Описані в посібнику електронні новинки такі як криптовалюти або методи отримання економічних даних з Інтернету говорять про швидку зміну самого поняття економіка в епоху тотального застосування електронних засобів її ведення.

В наш час кількість різних типів криптовалют, які нічим не забезпечені, а отже, є фікцією, а не грошима, швидко зростає. Національний Банк України прийняв рішення відмовитися від будь-яких операцій з біткоїнами та іншими криптовалютами. І застерігає всіх, хто намагається на цьому заробити, що в разі, коли їх ошукають, вони не зможуть відшкодувати збиток законним шляхом, бо криптовалюта не є грошима.

В той же час один університет у Швейцарії готовий приймати оплату за навчання у цій фіктивній валюті. Отже, у світі формуються різні підходи до цього електронного феномену. Але сам принцип блок-чейну виявився таким вдалим, що уряд України почав переводити всі реєстри на цю систему збереження інформації, яка гарантує стовідсоткову гарантію її не ушкодження.

Big Data – теж є феноменом, що виник як реакція економістів на тотальне розповсюдження Інтернету та перенесення всіх даних туди, на веб-сторінки. З’явилися системи отримання цих даних, їх переробки, аналізу. І все це напряму, без додаткового перекодування.

Ми зараз знаходимося на початку докорінної зміни економічних стосунків, вплив на суспільство яких передбачити важко, але знати про них, вміти їх використовувати є обов’язком усіх сучасних економістів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Adams M. J.. Chemometrics in Analytical Spectroscopy, RSC, Cambridge, UK, 1995
2. Beebe K.R., R.J. Pell, M.B. Seasholtz. Chemometrics: a Practical Guide, Willey, N.Y., 1998
3. Brereton R.G.. Applied Chemometrics for Scientists. Wiley, Chichester, UK, 2007
4. Gemperline P.. Practical Guide To Chemometrics, Taylor & Francis, 2006
5. http://hostinfo.ru/articles/web/rubric48/rubric55/rubric59/1358 /
6. http://lib.qrz.ru/book/export/html/1644
7. http://progs.biz/csharp/csharp01.aspx
8. http://py-algorithm.blogspot.com/2014/10/blog-post_21.html
9. http://sleepy.cs.surrey.sfu.ca/cmpt/courses/cmpt120/notes /
10. http://www.chemometrics.ru/materials/textbooks/matlab.htm
11. http://www.colinfahey.com/dotnet/dotnet_ru.html
12. http://www.python.org/doc/2.5.2/lib/module-turtle.html
13. http://www.python.ru /
14. http://www.twirpx.com/files/informatics/languages/cs/
15. https://netbeans.org/downloads/
16. https://www.mathworks.com/help/matlab/?requestedDomain=www.mathworks.com
17. https://www.visualstudio.com/ru/vs/support/#!articles/816-6458-hello-world-in-c-using-visual-studio-2015
18. J. J. Faraway, Practical Regression and Anova using R, 2002
19. J. Verzani, Using R for Introductory Statistics,CHAPMAN & HALLCRC,2005
20. Kramer R.. Chemometric Techniques for Quantitative Analysis, Marcel-Dekker, 1998
21. M.J. Crawley, The E Book, Wiley, 2007.
22. Mark H., J. Workman. Chemometrics in Spectroscopy, Elsevier, 2007
23. R Journal, http://journal.r-project.org/, 2001-2010
24. W. N. Venables and B. D. Ripley,Modern Applied Statistics with S , Springer, 2002.

25. W. N. Venables, D. M. Smith and the R Development Core Team, An Introduction to R, http://cran.r-project.org/doc/manuals/E-intro.pdf, 2009
26. Wes McKinney & PyData Development Team. Pandas: powerful Python data analysis toolkit / Pandas.DataFrame.describe [Электронный ресурс], 2017. – Режим доступа: https://pandas.pydata.org/pandas-docs/stable/pandas.pdf.
27. Дьяконов А. Презентация: Pandas. Обзор основных функций [Электронный ресурс]. – Москва, 2015. – Режим доступа: https://alexanderdyakonov.files.wordpress.com/2015/04/ama2015_pandas.pdf.
28. Златопольський, Д. М. Сборник задач по программированию – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2007.
29. Использование пакета Pandas в Python [Электронный ресурс], 2017. – Режим доступа: https://www.8host.com/blog/ispolzovanie-paketa-pandas-v-python-3/.
30. K. Hornik, The R FAQ, http://cran.r-project.org/doc/FAQ/E-FAQ.html, 2009
31. Караванова,Т.П. Інформатика: основи алгоритмізації та програмув.: 777 задач з рек. та прикл.:Навч. посіб. для 8-9 кл. із поглибл. вивч. інф-ки/ За заг. ред. М.З. Згуровського.-К : Гєнеза, 2006. - 286 с.
32. Луне Педро Коэльо, Вилл Ричарт. Построение снстем машишюго обучения на языке Pytho11. 2-е издание/ Пер. с англ. Слинют А. А. – М.: ДМК Пресс, 2016. – 302 с.
33. Лутц Марк. Pyton. Карманный справочник, 5-е изд. : Пер. с англ. – М.: И.Д.Вильямс, 2015. – 320 с.
34. Маккинли Уэс. Python и анализ данных/ Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2015. - 482 с.
35. Мусин Дмитрий. Самоучитель Python. Выпуск 0.1 //http://pythonworld.ru
36. Чаплыгин А. Н. Учимся программировать вместе с Питоном// – https://www.visualstudio.com/ru/vs/support
37. 3. http://wombat.org.ua/AByteOfPython/AByteofPythonRussian-2.01.pdf
38. 4. http://window.edu.ru/resource/825/76825/files/python_structured_programming.pdf

ДОДАТОК

Словник спеціальних термінів

Автентифікація (authentification) – певний засіб захисту безпеки роботи в мережі, який визначає особистість користувача і законність його дій. Перевірка заснована на імені користувача, паролі, а також обмеженнях облікового запису та обмеженнях за часом.

Архітектура Інтернет – багаторівнева архітектура мережі, при якій всі мережні функції розділені на сім рівнів. Кожному рівню відповідають певні мережні операції, устаткування і протоколи. Інтерфейс визначає послуги, які нижній рівень надає верхньому, і спосіб доступу до них. Проте реальна передача даних відбувається на самому нижньому – фізичному рівні, де знаходиться фізичне середовище передачі (мережний кабель).

Атрибут тегу – відповідний параметр тегу, якому призначається конкретне значення. Так, тег опису шрифту тексту, має наступні атрибути: face="..." – задає гарнітуру шрифту (Arial, Times New Roman,…); size= "..." – встановлює розмір шрифту (1-7); color="..." – задає колір шрифту.

Безперервне отримання інформації (“викачування”) – процес перезапису файлів, що мають значний розмір, з сервера, де вони знаходяться, на комп’ютер користувача. Програми, призначені для викачування файлів з FTP і HTTP серверів, називаються «менеджерами викачувань»: ***Net Wampire, ReGet, Opera.***

Бекбон (backbone) – загальна, первинна лінія зв’язку первинних провайдерів.

Браузер, переглядач (browser) – програма, що дозволяє користувачеві переглядати зміст Web-сторінок, розміщених на серверах мережі Інтернет, завдяки чому відбувається подорож по «Всесвітній павутині» документів. У даний момент найбільш поширені типи браузерів, особливості яких повинні

враховуватися при створенні Web-сторінок: ***Chrom, Mozilla Firefox, Opera.***

Вбудований об'єкт – будь-який об'єкт, який підтримується реалізацією мови JavaScript незалежно від середовища виконання і існуючий на момент початку виконання сценарію. Всі вбудовані об'єкти є об'єктами мови.

Веб-сайт (web-site) – сукупність Web-сторінок, об'єднаних однією загальною темою. Сайт розташовується, як правило, на одному вузловому комп'ютері і контролюється однією людиною чи групою людей.

Веб-сторінка (web-page) – складова частина Web-сайту, що фізично являє собою файл, підготовлений мовою HTML. Може містити текст, зображення, JAVA-аплети й інші Web-елементи. Сторінка може бути статичною або динамічно генерованою. Перегляд Web-сторінки здійснюється за допомогою програми-браузера.

Всесвітня павутина (World Wide Web) – служба Internet для роботи з гіпертекстом і мультимедіа. WWW – найбільше сховище інформації в електронному вигляді, мільйони пов'язаних між собою документів, які розміщені на комп'ютерах по всій земній кулі. Дозволяє одержувати доступ до інформації по заданій темі незалежно від місця її розташування. Користувачі автоматично переходять від однієї веб-сторінки до іншої за допомогою «гіперпосилань» з використанням інтерактивних мультимедійних засобів.

Гіпермедія (hypermedia) – метод надання інформації у вигляді тексту, графіки, звукозаписів, відеозаписів, анімації тощо, з'єднаних з допомогою посилань у WWW–системі.

Гіперпосилання (hypertext, гіперлінк) – будь-який текст, символ чи рисунок, зв'язаний з URL-адресою документа (фрагменту, сторінки чи серверу), на який встановлюється посилання, та реагуючий на клік миші завантаженням цього документу. Це дає можливість легко і

швидко переміщуватися між документами, розташованими на різних серверах у WWW-системі.

Гіпертекст – текст, у якому містяться посилання на інші текстові документи.

Глобальна обчислювальна мережа (Wide Area Network, WAN) - комп'ютерна мережа, яка надає послуги багатьом кінцевим користувачам, розміщеним на великій території. Глобальні мережі не мають точно визначеної кількості робочих станцій, не мають і чіткої ієрархії. До глобальних мереж може приєднатися і від'єднатися невизначена кількість комп`ютерів. Фізичне місцезнаходження кожного комп`ютера в глобальній мережі теж не визначено.

Домен (domain) – сукупність хостів Internet, об'єднаних за якоюсь ознакою (наприклад, за територіальною). Кожний домен має унікальне ім'я.

Доменне ім'я (domain name) – унікальне алфавітно-цифрове ім'я, що ідентифікує конкретний Web-вузол. Доменні імена звичайно складаються з двох і більше частин, відокремлених крапкою. Ліва частина доменного імені відповідає кінцевому вузлу мережі (тобто є найбільш специфічною), тоді як права вказує на домени, до яких він належить. Доменне ім'я виглядає, наприклад, так: minagro.kiev.ua.

Елемент одного з типів Undefined, Null, Boolean, Number або String. Примітивні значення – це дані, які представляються безпосередньо і на самому нижньому рівні реалізації мови.

Елемент типу Object – є неврегульованим набором властивостей, кожне з яких може бути примітивним значенням, об'єктом або функцією. Властивість, що є функцією, називається методом.

Елемент типу String – являє собою рядок символів, т. е. упорядкований масив з нуля або більше символів Unicode (т. е. 16-бітових цілих чисел без знака).

Закладка (bookmark) – "мітка" користувача, що дозволяє записати у пам'ять комп'ютера адреси Web-сторінок, які він найчастіше відвідує. Закладки є такими ж файлами, як і інші у Windows, тому порядок

роботи з переліком закладок тотожний з порядком роботи з файлами. В програмі наперед створені деякі папки з закладками, але можна створювати власні.

Конструктор – функція, яка створює і ініціалізує об'єкти. Кожен конструктор має відповідний прототип, який використовується для наслідування і поділу властивостей.

Логічний об'єкт – елемент типу Object, який є екземпляром вбудованого логічного об'єкта. Іншими словами, логічний об'єкт створюється виразом new Boolean (value), де value - логічне значення. Результуючий об'єкт має неявне (безіменне) властивість типу Boolean.

Мова JavaScript – інструмент, що дозволяє завантаженій в браузер сторінці динамічно управляти своїм вмістом, а заразом і власне браузером. Скрипти (script – сценарії) дозволяють відкривати і закривати вікна браузера, завантажувати в них документи, управляти фреймами і взаємодіяти з полями форм (наприклад, перевіряючи правильність введених в них значень).

Мова розмітки гіпертекстових документів (Hyper Text Markup Language, HTML) – являє собою набір команд, що описують структуру та оформлення документу. Усі Web-сторінки створюються на основі мови HTML. Команди HTML забезпечують з'єднання сайтів Всесвітньої павутини за допомогою гіперпосилань.

Невизначене значення – примітивне значення, яке означає, що змінної не присвоєно ніякого значення.

Об'єкт мови – будь-який об'єкт, який підтримується реалізацією мови JavaScript, а не середовищем виконання сценаріїв. Частина об'єктів мови є вбудованими; інші створюються в процесі виконання сценарію.

Об'єкт середовища – будь-який об'єкт, який не є об'єктом мови, а підтримується середовищем виконання сценаріїв.

Обліковий запис пошти – установка параметрів для роботи з електронною поштою: ім'я відправника, його адреса електронної пошти, характеристики серверів вхідних та вихідних повідомлень, ім'я облікового запису та пароль.

Початкова Web-сторінка (home page) – перша сторінка, яку одержує користувач, потрапивши до Web-серверу, містить загальні відомості про вміст сайту.

Поштовий агент – додаток до браузера, який дозволяє користатися можливостями електронної пошти.

Пошуковий сервер (search engines) – система, призначена для пошуку і доставки інформації. Коли користувач вводить в цю систему яке-небудь ключове слово, що відповідає потрібній інформації, воно шукається в базі даних пошукового серверу, і користувачу видається низка посилань на ті сервери, на яких задане слово зустрічається. Основні пошукові сервери: ***BIGMIR.NET (http://www.bigmir.net/), META (http://www.meta.ua), TOPPING (http://www.topping.com.ua), ЯNDEX (http://www.yandex.ru), RAMBLER (http://www.rambler.ru), APORT(http://www.aport.ru), YAHOO! (http://www.yahoo.com), ALTAVISTA (http://www.altavista.com), GOOGLE (http://www.google.com).***

Проксі-сервер (proxy server) – компонент брандмауера, який керує вхідним і вихідним трафіком Internet в локальній мережі. Визначає безпеку передачі повідомлень або файлів у мережу організації, управляє доступом до мережі, фільтрує і відхиляє запити згідно із заданими параметрами, включаючи запити на несанкціонований доступ до конфіденційних даних.

Протокол – правила-домовленості про сигнали, якими обмінюються комп'ютери під час встановлення зв'язку між собою і приймання чи передавання інформації. Цих правил передачі даних у мережі

повинні дотримуватися всі компанії, щоб забезпечити сумісність апаратного і програмного забезпечення.

Прототип – об'єкт, який використовується в JavaScript для реалізації успадкування структури, стану та поведінки. Коли конструктор створює об'єкт, останній містить неявну посилання на прототип конструктора, що дозволяє вирішувати посилання на властивості даного об'єкта. Властивості прототипу поділяються усіма об'єктами, створеними на його основі.

Редактори html-кодів – програми для розробки Web-публікацій професійної якості, що виконують багато функцій, зручних для користувача: автоматизують введення коду, перевіряють помилки, здійснюють підтримку технологій створення динамічних і інтерактивних сторінок та ін. Серед редакторів html-текстів найбільш відомі ***Homesite та HotDog***.

Редактори html-сторінок типу WYSIWYG (What You See Is What You Get – „що бачите, те і отримаєте") – візуальні редактори, завдяки яким користувач має справу не з кодом документа, а з графічними образами елементів HTML. Тобто створює не код, а оформлення сторінки, після чого програма автоматично підбирає для неї код. До редакторів цього типу можна віднести: ***Microsoft FrontPage, Macromedia Dreamveawer, Netscape Composer, Star Office.***

Робоча станція (workstation, client) – комп'ютер, що споживає мережні ресурси, які надає сервер. У мережах з сервером робочі станції виступають як клієнти мережі, тому про них кажуть – мережі типу „клієнт-сервер".

Сервер (server) – по-перше, комп'ютер, що надає свої ресурси в спільне користування в мережі (сервер баз даних, файловий сервер, поштовий сервер, веб-сервер) та відрізняється вищою продуктивністю, більшими обсягами оперативної пам'яті та жорстких дисків; по-друге, програма, яка надає клієнтам доступ до

мережевих ресурсів та використовується для визначення користувачів, розподілу доступу до ресурсів, встановлення черги на доступ до інформації, дотримання режиму секретності.

Скрипт (script) – невелика програма-сценарій, що, звичайно, складається на мові JavaScript для забезпечення інтерактивності сторінки. Основне призначення скриптів полягає в організації діалогу з користувачем – анкетування, отримання даних з сайту із заповнених користувачем форм. Вони також дозволяють відкривати і закривати вікна браузера, завантажувати в них документи, управляти фреймами.

Смайл (smile – посмішка) – спеціальні значки, що надають емоційного забарвлення поштовому повідомленню, мають вигляд смішних облич, якщо розглядати їх, повернувши на кут у 90 градусів за годинниковою стрілкою.

Спам (spam) – розсилка якого-небудь повідомлення (частіше рекламного чи комерційного характеру) багатьом адресатам, для яких це повідомлення небажане.

Таблиця стилів (Cascading Style Sheets – CSS) – каскадна таблиця стилів представляє собою набір правил форматування HTML-тегів і є інструментом впливу на зовнішній вигляд Web-сторінок. З появою CSS з’явилась можливість відокремити структуру HTML-документа від його формату. Стилі форматування – це сукупність властивостей, які можна привласнити багатьом елементам (таким як таблиці, текст посилання і т.п.) і які впливають на відображення об’єкта у вікні браузера.

Теги (tags), дескриптори – спеціальні команди розмітки мови HTML, вкладені в кутові дужки (< і >). Теги мови HTML задають правила, за якими браузер відображає документ на екрані: розміщення тексту у вікні, представлення графічних об'єктів (малюнків), а також виведення звукових файлів, відеокліпів і т.д.

Тип – Набір значень даних.

Тип Boolean – тип, що складається рівно з двох значень: true (істина) і false (брехня).

Тип Number – тип, що складається з усіх можливих числових значень. Точніше кажучи, це набір 64-бітових числових значень формату IEEE 754, що включає спеціальні значення NaN (не числом), позитивна нескінченність і негативна нескінченність.

Тип String – цей тип складається з усіх можливих значень тексту.

Тип Undefined – Цей тип складається з одного значення undefined, яке є невизначеним.

Топологія мереж – фізична топологія визначає правила фізичних з'єднань вузлів, логічна – напрямок потоків даних між вузлами мережі. Логічна та фізична топології незалежні одна від одної. Кожна мережна технологія має характерну тільки для неї топологію з'єднання вузлів мережі та метод доступу до середовища передачі даних.

Трафік (traffic) – потік інформації переданий по мережі.

Універсальний покажчик на ресурс (Uniform Resource Locator, URL) – унікальне ім'я, що однозначно визначає документ у мережі Internet. Складається з таких частин: протоколу, імені комп'ютера, шляху до каталогу та імені файлу (наприклад, http://www.kneu.kiev.ua/index.html).

Унікальна адреса – кожен комп'ютер, підключений до Internet, має свою IP-адресу. Вона має довжину 32 біта і складається з чотирьох частин по 8 бітів, що відокремлюються крапкою. Наприклад, 147.120.3.28 – це IP-адреса.

Форма – об'єкт мови HTML, що використовується для розміщення на Web-сторінці діалогового вікна для отримання від користувача різноманітної інформації. Після заповнення форми і запуску процесу її обробки інформація з неї потрапляє до програми, що працює на сервері. Таким чином користувач може інтерактивно спілкуватись з

Web-сервером через Internet.

Формати GIF (Graphics Interchange Format) та ***JPEG*** (*Joint Picture Expert Group)* – стандартні формати для представлення графіки в Internet.

Форум – тематичні електронні дискусії, призначені для обміну інформацією між людьми через електронні повідомлення. Найчастіше форуми влаштовуються для того, щоб опитати читачів якоїсь статті про їх думку. Інколи форуми призначені для постійно обговорюваної теми, наприклад, ремонт комп'ютерів чи антивіруси. В будь який час ви можете прочитати всі послання, адресовані на цей форум.

Фрейми (frames – кадр) – незалежні частини вікна Web-браузера, в кожну з яких завантажується своя власна Web-сторінка. Наприклад, для організації зручної навігації по сайту в одному фреймі розміщується перелік розділів сайту у вигляді гіперпосилань, а в іншій завантажуються Web-сторінки розділів, що були вибрані користувачем у попередньому фреймі.

Хост **(**host), вузол – комп'ютер, постійно підключений до Internet. Він призначений для забезпечення входу і роботи кінцевих користувачів в мережі, а також для розташування і збереження інформації. Крім комп'ютерів, це можуть бути спеціальні мережні пристрої – маршрутизатори (router) та інші, що підтримують протоколи TCP/IP.

Хостінг (hosting) – послуга розміщення Web-сайтів на серверах, які підключені до мережі Internet і забезпечують цілодобовий доступ до сайту.

Цифровий підпис (digital signature) – засіб підтвердження авторства зашифрованого повідомлення, файла або будь-якої іншої зашифрованої цифрової інформації. Цифрові підписи застосовуються в середовищах з відкритими ключами і надають функції забезпечення цілісності та запобігання неавторизованій зміні інформації, яка передається.

Цифровий сертифікат (digital certificate) – електронний документ, підписаний за допомогою цифрового підпису, який встановлює, що заданий відкритий ключ відповідає об'єкту, що має певне ім'я. Видачу цифрових сертифікатів здійснюють державні чи комерційні сертифікаційні центри.

Чат (chat – бесіда, розмова) – проведення Internet-конференції в режимі реального часу. При цьому кожен з кореспондентів цієї конференції може бачити листи, що надходять від усіх інших кореспондентів. Можна також об’єднатися з деякими кореспондентами і зробити своє листування невидимим для інших учасників конференції.

Числове значення – елемент типу Number. Є безпосереднім поданням числа.

Шлюз – з'єднання між двома локальними мережами, а також апаратне забезпечення для встановлення з'єднання.