НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки та менеджменту
Кафедра економічної кібернетики

КОНСПЕКТ ЛЕКЦІЙ
навчальної дисципліни
«Аглоритмізація та програмування в економіці»

Укладач:
к.е.н., Квашук Д.М

Конспект лекцій розглянутий та схвалений на засіданні кафедри економічної кібернетики
Протокол № ____ від «___»_____2018_р.
Завідувач кафедри_____ Н. В. Кась'янова

Зміст

ВСТУП **4**
МОДУЛЬ 1 ОСНОВИ ПРОГРАМУВАННЯ 4
ЛЕКЦІЯ 1. ТЕОРЕТИЧНІ ОСНОВИ АЛГОРИТМІЗАЦІЇ 5
ЛЕКЦІЯ 2 ОСНОВНІ КОНСТРУКЦІЇ МОВИ PYTHON 9
2.1 Встановлення Python 9
2.2 Перевірка працездатності 10
2.3. Запуск програм на Python 11
2.4 Пакетний режим роботи 12
2.5 Модель даних 13
2.6 Змінні і незмінні типи даних 15
2.7 Арифметичні операції 16
2.8 Робота з комплексними числами 18
2.9 Бітові операції 19
2.10 Подання чисел в інших системах числення 19
2.11 Бібліотека (модуль) math 20
ЛЕКЦІЯ 3. УМОВНІ ОПЕРАТОРИ ТА ЦИКЛИ 21
3.1 логічний операторот if 21
3.2 Конструкція if – else 22
3.3 Конструкція if - elif – else 23
3.4 Оператор циклу while 23
3.5 Оператори break і continue 24
3.6 Оператор циклу for 24
ЛЕКЦІЯ 4. РОБОТА ЗІ СПИСКАМИ (list) 25
4.1 Що таке список (list) в Python? 25
4.2 Створення, зміна, видалення списків і робота з його елементами 26
4.3 Методи списків 28
4.4 List Comprehensions 30
ЛЕКЦІЯ 5. КОРТЕЖІ (tuple) 31
5.1 Що таке кортеж (tuple) в Python? 31
5.2 Створення, видалення кортежів і робота з його елементами 32
ЛЕКЦІЯ 6 СЛОВНИКИ (dict) 34
6.1 Що таке словник (dict) в Python? 34
6.2 Створення, зміна, видалення словників і робота з його елементами 34
6.3 Методи словників 35
ЛЕКЦІЯ 7. ФУНКЦІЇ В PYTHON 37
7.1 Що таке функція в Python? 37
7.2 Створення функцій 37
7.3 Робота з функціями 38
7.4 Lambda-функції 38
ЛЕКФЧЯЦІЯ 8 ВИЙНЯТКИ 39
8.1 Робота з винятками 39
8.2 Синтаксичні помилки в Python 40
8.3 винятки в Python 40
8.4 Ієрархія винятків в Python 40
8.5 Обробка виключень в Python 42
8.6 Використання finally в обробці виключень 44
8.7 Генерація винятків в Python 44
8.8 Користувальницькі виключення (User-defined Exceptions) в Python 44
ЛЕКЦІЯ 9 ВВЕДЕННЯ-ВИВЕДЕННЯ ДАНИХ. РОБОТА З ФАЙЛАМИ 45
9.1 Вивід даних в консоль 45
9.2 Введення даних з клавіатури 46

9.3 Робота з файлами 47
ЛЕКЦІЯ 10. МОДУЛІ ТА ПАКЕТИ 49
10.1 Модулі в Python 49
10.2 Пакети в Python 51
ЛЕКЦІЯ 11. КЛАСИ І ОБ'ЄКТИ 52
11.1 Основні поняття об'єктно-орієнтованого програмування 52
11.2 Класи в Python 52
11.3 Спадкування 54
11.4 Поліморфізм 55
ЛЕКЦІЯ 12. ІТЕРАТОРИ ТА ГЕНЕРАТОРИ 55
12.1 Ітератори в мові Python 56
12.2 Створення власних ітераторів 56
12.3 Генератори 57
МОДУЛЬ 2 ТИПОВІ АЛГОРИТМИ ОБРОБКИ ЕКОНОМІЧНОЇ ІНФОРМАЦІЇ 57
ЛЕКЦІЯ 13. НЕСТРУКТУРОВАНА ІНФОРМАЦІЯ 58
ЛЕЦІЯ 14. ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ЗБОРУ, НАКОПИЧЕННЯ, ОБРОБКИ ТА ВІЗУАЛІЗАЦІЇ BIG DATA 58
ЛЕКЦІЯ 15. СТРУКТУРИ ДАНИХ БІБЛІОТЕКИ PANDAS 62
ЛЕКЦІЯ 16. ІНТЕГРАЦІЯ MS EXCEL ТА PYTHON 67
ЛЕКЦІЯ 17. ПОБУДОВА ГРАФІКІВ ТА ВІЗУАЛІЗАЦІЯ 72
Індивідуальне завдання до модулю 2 78
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 83

Вступ

Метою викладання дисципліни є теоретична та практична підготовка студентів до вивчення систем обробки даних та принципів алгоритмізації економічних розрахунків з використанням мови програмування python.

Завданнями вивчення навчальної дисципліни є: дослідження технологій зберігання, збору та обробки економічної інофрмації; оволодіння методами програмування на мові python; дослідження процесів виявлення знань; дослідження принципів побудови сховищ даних.

У результаті вивчення даної навчальної дисципліни студент повинен:

Знати:

- методи та технології алгоритмізації економічних розрахунків;
- методи реалізації атоматизованого аналізу економічної інорфмації.

Вміти:

- самостійно застосовувати алгоритми економчної аналізу з використанням мови програмування python;
- самостійно розробляти та будувати моделі сховищ даних;
- самостійно проводити аналіз даних для виявлення знань;
- самостійно використовувати бібліотеки мови python при обробці економічних даних.

Навчальний матеріал дисципліни структурований за модульним принципом і складається з двох класичних навчальних модулів.

У результаті засвоєння навчального матеріалу навчального модуля №1 «Основи програмування на алгоритмічних мовах» студент повинен:

Знати:

- синтаксис мови python;
- принципи алгоритмізації;
- способи використання сторонніх бібліотек для аналіза економічних даних.

Вміти:

- самостійно організовувати сховище даних;
- самостійно підготовлювати дані для їх аналізу;
- самостійно застосовувати методи використання навчальної інформації.

У результаті засвоєння навчального матеріалу навчального модуля №2 «Типові алгоритми обробки економічної інформації» студент повинен:

Знати:

- структуру багатовимірної моделі даних;
- методи та задачі Data Mining;
- методи асоціативних правил.

Вміти:

- самостійно застосовувати методи Data Mining;
- самостійно використовувати методи вибірки економічних даних;
- самостійно засотсовівати методи асоціативних правил;
- самостійно використовувати алгоритми пошуку в економічних розрахунках

Знання та вміння, отримані студентом під час вивчення даної навчальної дисципліни, використовуються для закріплення отриманих знань та є базою для написання магістерського дипломного проекту (роботи).

МОДУЛЬ 1 Основи програмування на алгоритмічних мовах

ЛЕКЦІЯ 1. Теоретичні основи алгоритмізації

Перша у світі електронно-обчислювальна машина (ЕОМ) була створена в 1946 році в США групою вчених під керівництвом Дж. фон Неймана. Її робота базувалася на чотирьох основних принципах:

- принцип двійковості. Як інформаційна частина (числа, текст), так і керуюча (команди) представляються на машинному рівні у двійковій системі;
- принцип адресності. Усі дії виконуються над вмістом певних адрес. Команди також розміщені за адресами;
- принцип програмного управління. Від початку до кінця процес обчислень здійснюється під управлінням програми (набору команд);
- принцип переадресації. Над командами виконуються арифметичні дії, як і над даними. Такі сформовані команди мають змогу виконувати операції над послідовністю даних.

Ці принципи дотепер називають принципами Неймана. Хоча основні ідеї побудови й функціонування ЕОМ були висунуті ще в 1890-х роках англійським ученим Ч. Беббіджем. В 1930-х роках були сформульовані (А.Тьюрінг та ін.) принципи інформатики, які чітко окреслювали можливості майбутніх ЕОМ. Отже, приступаючи в кінці другої світової війни до створення першої ЕОМ, учені чітко усвідомлювали можливості цієї майбутньої для них техніки. Зокрема, один з основних принципів інформатики стверджує:

Будь-яка ЕОМ, побудована на принципах фон Неймана, яка має в складі своїх команд арифметичні дії, пересилку (присвоєння) та розгалуження є універсальною (алгоритмічно повною) у тому сенсі, що за допомогою цих команд можна в принципі виконати довільний алгоритм.

Звичайно, для виконання конкретного алгоритму на конкретній техніці може не вистарчити машинних ресурсів (пам'яті чи часу). Проте алгоритмічна повнота має місце принципово. Сучасні комп'ютери теж функціонують на принципах фон Неймана, отже поняття універсальності відноситься і до них.

Проте, звичайно, далеко не всі задачі можна в принципі розв'язати на комп'ютері. З науково-технічним прогресом усе нові й нові задачі підключаються до розв'язання на комп'ютерах. Проте існують і такі задачі, які в принципі ніколи на комп'ютерах розв'язані не будуть (як не буде знайдено

універсального розчинника, вічного двигуна, ліки від усіх хвороб,...). Уперше ряд чітко сформульованих математичних задач, для яких у принципі не існує алгоритму їхнього розв'язування (а, отже, і розв'язування за допомогою комп'ютера), було виявлено в 1930-х роках. При цьому неіснування таких алгоритмів було строго математично доведено (доведення у вигляді теорем). Наведемо дві з таких теорем.

Теорема Райса (Rice). Не існує алгоритму (в принципі), який по тексту програми та по вхідних даних зміг би установити, чи відбудеться зациклення.

Звичайно, ця теорема нами сформульована в перефразованому вигляді, оскільки у 1930-х роках ще не існувало термінів "програма", "вхідні дані", "зациклення".

Теорема Черча (Church). Не існує алгоритму, який би по заданих аксіомах, правилах виведення та деякому твердженню зміг би встановити, чи це твердження виводиться з аксіом, чи ні.

Алгоритм, його властивості

Під алгоритмом розуміють скінчену систему правил для розв'язування певного класу задач, яка задовольняє таким властивостям:

- масовість. Алгоритм – це не знаходження розв'язку однієї конкретної задачі, а єдиний спосіб для розв'язання класу задач. Як правило, множина вхідних даних алгоритму є нескінченною;
- результативність. Після скінченої кількості кроків алгоритм повинен видати певний результат (навіть такий результат, як "я не можу знайти розв'язку");
- детермінованість. Хто б не виконував алгоритм і коли б його не виконував, він при одних і тих же вхідних даних завжди повинен отримувати той самий результат.

Остання властивість виражає основну суть алгоритму: виконавець може не розуміти змісту. Він повинен лише вірно виконувати запропоновану йому послідовність дій.

Розглядають три основні способи задання алгоритмів:

- вербальний (словесний);
- за допомогою блок-схем;
- за допомогою програми на алгоритмічній мові.

При цьому словесний спосіб аж ніяк не є легшим, ніж написання програми. Справді, алгоритм (при будь-якому заданні) повинен бути настільки точно описаний, щоб кожен виконавець однозначно розумів усі інструкції алгоритму (без присутності автора алгоритму як консультанта).

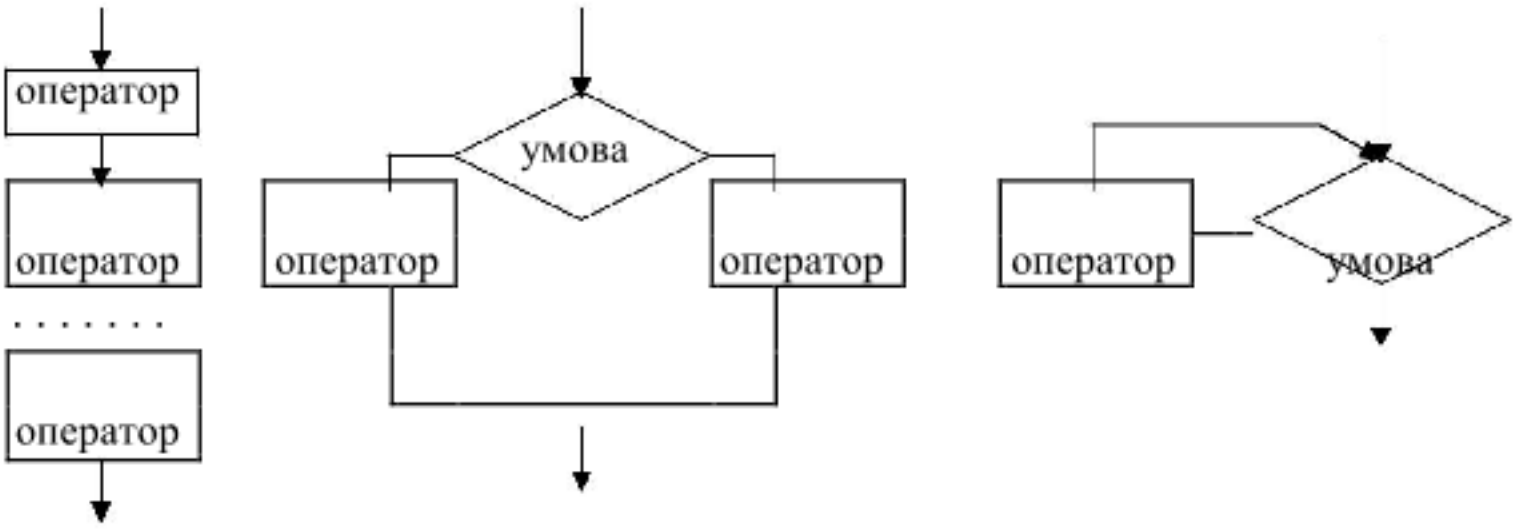
Як твердить принцип інформатики, для запису будь-якого алгоритму достатньо наявності арифметичних команд, операції розгалуження та операції пересилки. Усі алгоритмічні мови (PYTHON, PYTHON, C++ тощо) ці операції включають. Отже, із позиції універсальності усі мови програмування рівнопотужні. Проте з позиції реального написання й налагодження програм дуже суттєвою є і методика написання цих програм.

Основна трудність при розробці великих програм (як і будь-якої інтелектуальної діяльності) полягає в тому, що людина не може одночасно тримати в голові більше, ніж 7±2 об'єкти. Отже, блок-схему програми з 10 і більше блоками уже неможливо зрозуміти. Майстерність програміста (як і письменника, науковця) полягає у вмінні розбити одну велику задачу на ряд простіших, які можна розв'язувати незалежно в часі (або одночасно різними людьми). Зокрема, бажано так будувати блок-схему (чи текст програми), щоб у кожен момент часу концентрувати свою увагу на невеликому фрагменті блок-схеми (не більше 7 блоків) чи програми.

Однією з методик розробки програм, яка дозволяє поетапно будувати програму, є структурне програмування.

Теоретичною основою застосування структурного програмування є Теорема Бома-Джакопіні.

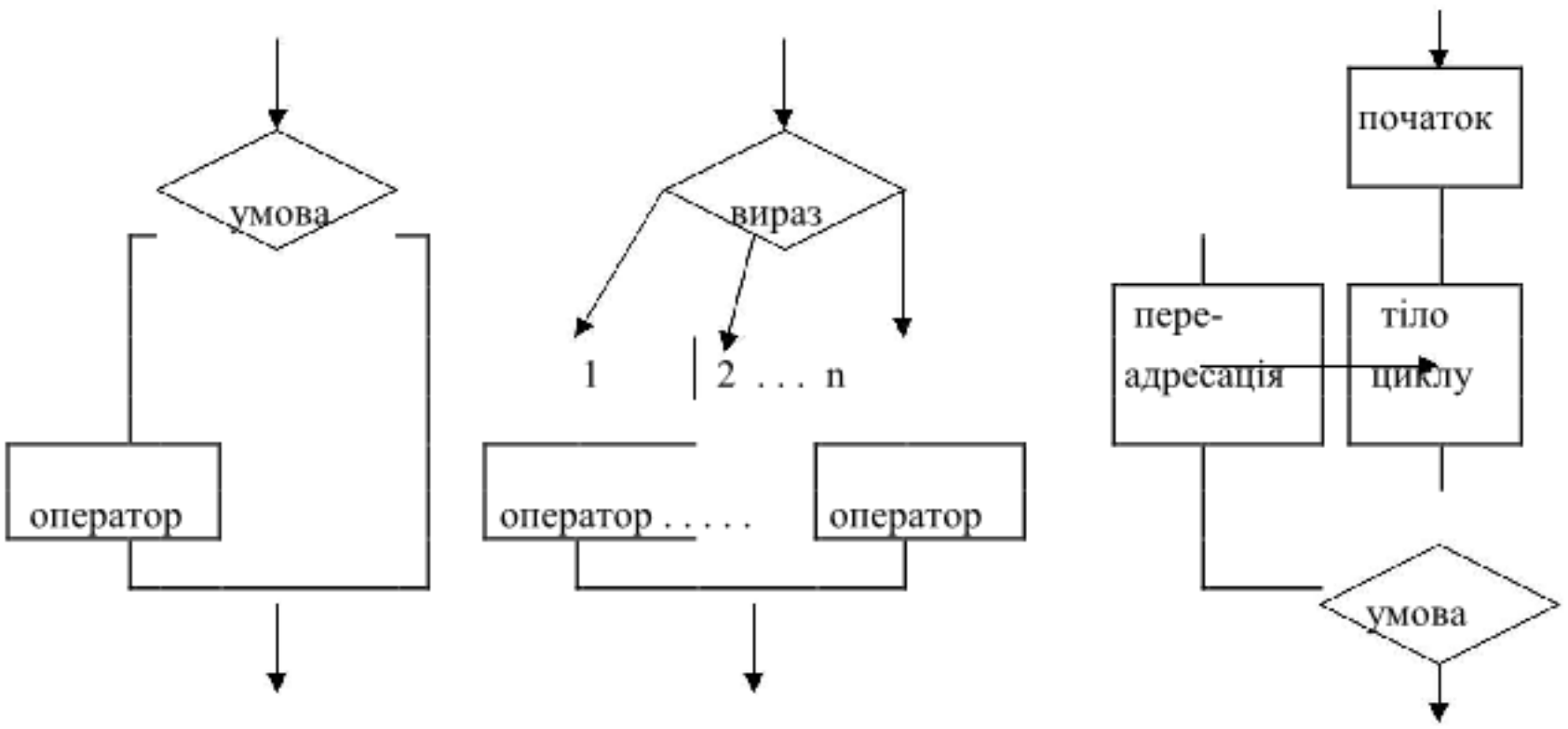
Для того щоб скласти блок-схему довільного алгоритму, достатньо трьох конструкцій: складеного оператора типу BEGIN_END, оператора розгалуження типу IF_THEN_ELSE та оператора циклу типу WHILE_DO:



begin оператор;... if умова then оператор while умова

...;оператор end else оператор do оператор

Для зручності на практиці використовують ще три такі конструкції:



If умова
Then оператор

Case вираз Of
1: оператор;
. . . . . .
n: оператор
end

FOR змінна=початкове
значення
To кінцеве значення
Do оператор

Структурне програмування рекомендує:

1. Використовувати всі шість конструкцій природнім способом (не програмувати циклічний процес оператором розгалуження тощо);
2. Не використовувати жодних інших керуючих конструкцій (зокрема, оператора безумовного переходу GOTO).

Ряд систем програмування (PYTHON, C++, шкільна Алгоритмічна Мова) мають оператори, що відповідають усім шести конструкціям. Отже, в цих системах природно використовувати структурне програмування. Структурні програми є набагато зрозуміліші і наглядніші, ніж не-

структурні. Великі програмні розробки можна виконати тільки за допомогою технології структурного програмування.

Деякі системи, наприклад, FOXPRO взагалі не мають оператора GOTO.
На них, отже, можна писати тільки структурні програми.

Проте існують системи (мова PYTHON ранніх версій, зокрема, GW-PYTHON, які іноді ще досі викладається в середніх школах), що не містять складеного оператора або інших конструкцій зі структурного набору. При використанні таких систем програміст змушений систематично звертатися до оператора GOTO. Дуже швидко програма стає зовсім незрозумілою. Розробити таким чином програму хоча б середньої складності неможливо. Неструктурні мови програмування тому є лише учбовими.

ЛЕКЦІЯ 2 Основні конструкції мови PYTHON

2.1 Встановлення Python

На сьогоднішній день існують дві версії Python - це Python 2 і Python 3, у них відсутня повна сумісність один з одним. На момент написання книги друга версія Python ще широко використовується, але, судячи зі змін, які відбуваються, з часом, вона залишиться тільки для запуску старого коду. У нашій з вами роботі, ми будемо використовувати Python 3, і, в подальшому, якщо десь буде зустрічатися слово Python, то під ним слід розуміти Python 3. Випадки застосування Python 2 будуть спеціально обумовлюватися.

Для установки інтерпретатора Python на ваш комп'ютер, перше, що потрібно зробити - це завантажити дистрибутив. Завантажити його можна з офіційного сайту, перейшовши за посиланням https://www.python.org/downloads/.

Установка Python в Windows

Для операційної системи Windows дистрибутив поширюється або у вигляді виконуваного файлу (з розширенням exe), або у вигляді архівного файлу (з розширенням zip).

Порядок установки:

1. Запустіть завантажений інсталяційний файл.

2. Визначте спосіб установки (пропонується два варіанти Install Now і Customize installation. при виборі Install Now, Python встановиться в папку за вказаним шляхом. Крім самого інтерпретатора буде встановлено IDLE (інтегроване середовище розробки), pip (пакетний менеджер) і документація, а також будуть створені відповідні ярлики і встановлені зв'язки файлів, що мають розширення .py.

3. Відзначте необхідні опції установки

На цьому кроці нам пропонується відзначити доповнення, що встановлюються разом з інтерпретатором Python. Рекомендуємо вибрати всі опції

4. Виберіть місце установки

5. Після успішної установки зявиться повідомлення про завершення процесу встановлення.

Установка Anaconda

Для зручності запуску прикладів і вивчення мови Python, радимо встановити на свій ПК пакет Anaconda. Цей пакет включає в себе інтерпретатор мови Python (є версії 2 і 3), набір найбільш часто використовуваних бібліотек і зручне середовище розробки та виконання, що запускається в браузері. Для установки цього пакета, попередньо потрібно завантажити дистрибутив https://www.continuum.io/downloads . Є варіанти під Windows, Linux і Mac OS.

Установка Anaconda в Windows

1. Запустіть завантажений інсталятор. У першому вікні необхідно натиснути "Next".

2. Далі слід прийняти ліцензійну угоду.

3. Виберіть одну з опцій установки:

– Just Me - тільки для користувача, що запустив установку;

– All Users - для всіх користувачів.

4. Вкажіть шлях, по якому буде встановлена Anaconda

5. Вкажіть додаткові опції:

– Add Anaconda to the system PATH environment variable - додати Anaconda в системну змінну PATH

– Register Anaconda as the system Python 3.5 - використовувати Anaconda, як інтерпретатор Python 3.5 за замовчуванням.

6. Для початку установки натисніть на кнопку "Install".

7. Після цього буде проведена установка Anaconda на ваш комп'ютер.

Установка PyCharm

Якщо в процесі розробки вам необхідний відладчик і взагалі ви звикли працювати в IDE, а не в текстовому редакторі, то тоді одним з кращих варіантів буде IDE PyCharm від JetBrains. Для скачування даного продукту потрібно перейти по посиланню https://www.jetbrains.com/pycharm/download/IDE доступна для Windows, Linux і Mac OS. Існують два види ліцензії PyCharm - це Professional і Community. Ми будемо використовувати версію Community, так як вона безкоштовна і її функціоналу більш ніж достатньо для навчальних цілей.

Установка PyCharm в Windows

1. Запустіть завантажений дистрибутив PyCharm
2. Виберіть шлях установки програми.
3. Вкажіть ярлики, які потрібно створити на робочому столі
4. Виберіть ім'я для папки в меню Пуск.
5. Далі PyCharm буде встановлений на ваш комп'ютер.

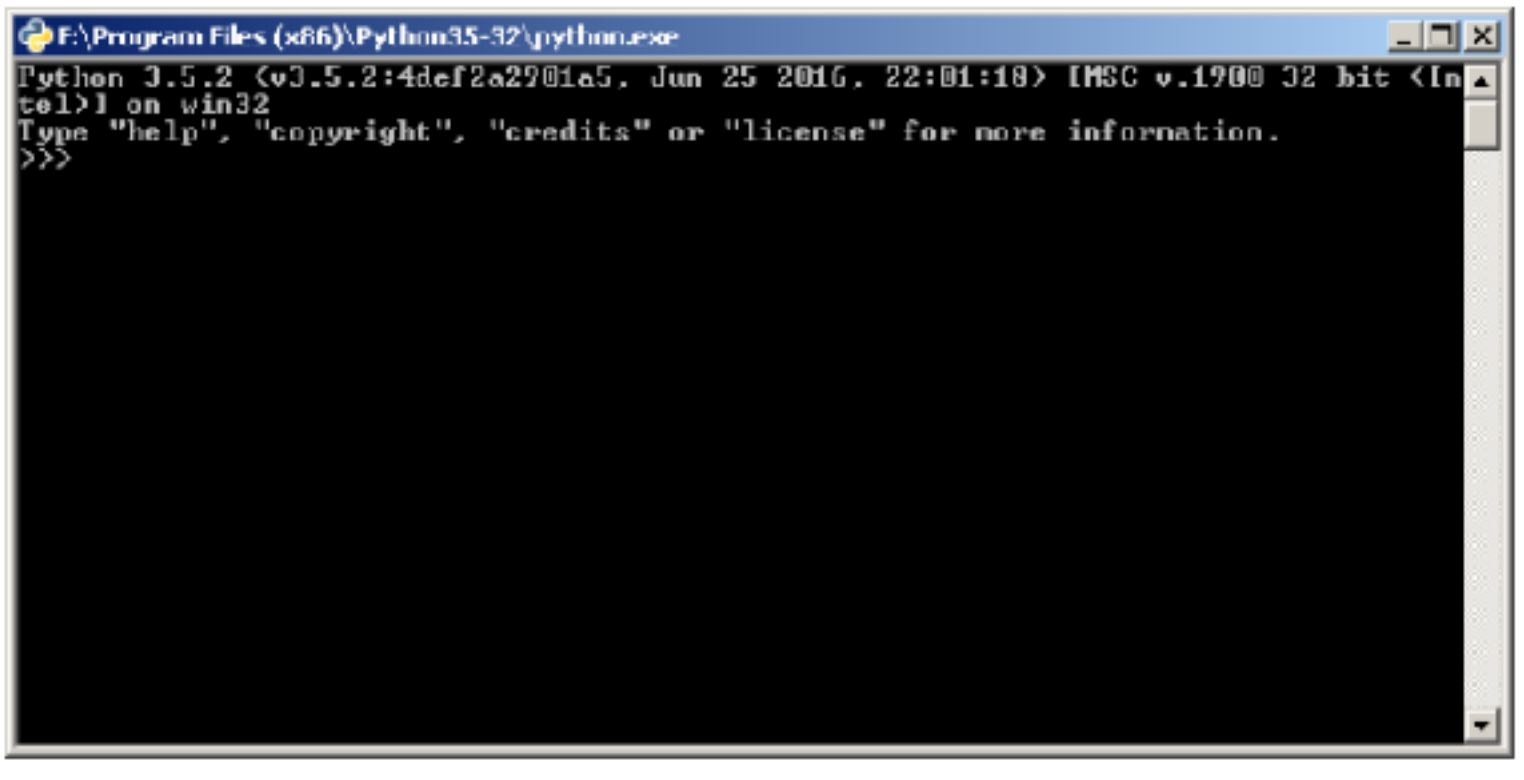
2.2 Перевірка працездатності

Тепер перевіримо працездатність всього того, що ми встановили.

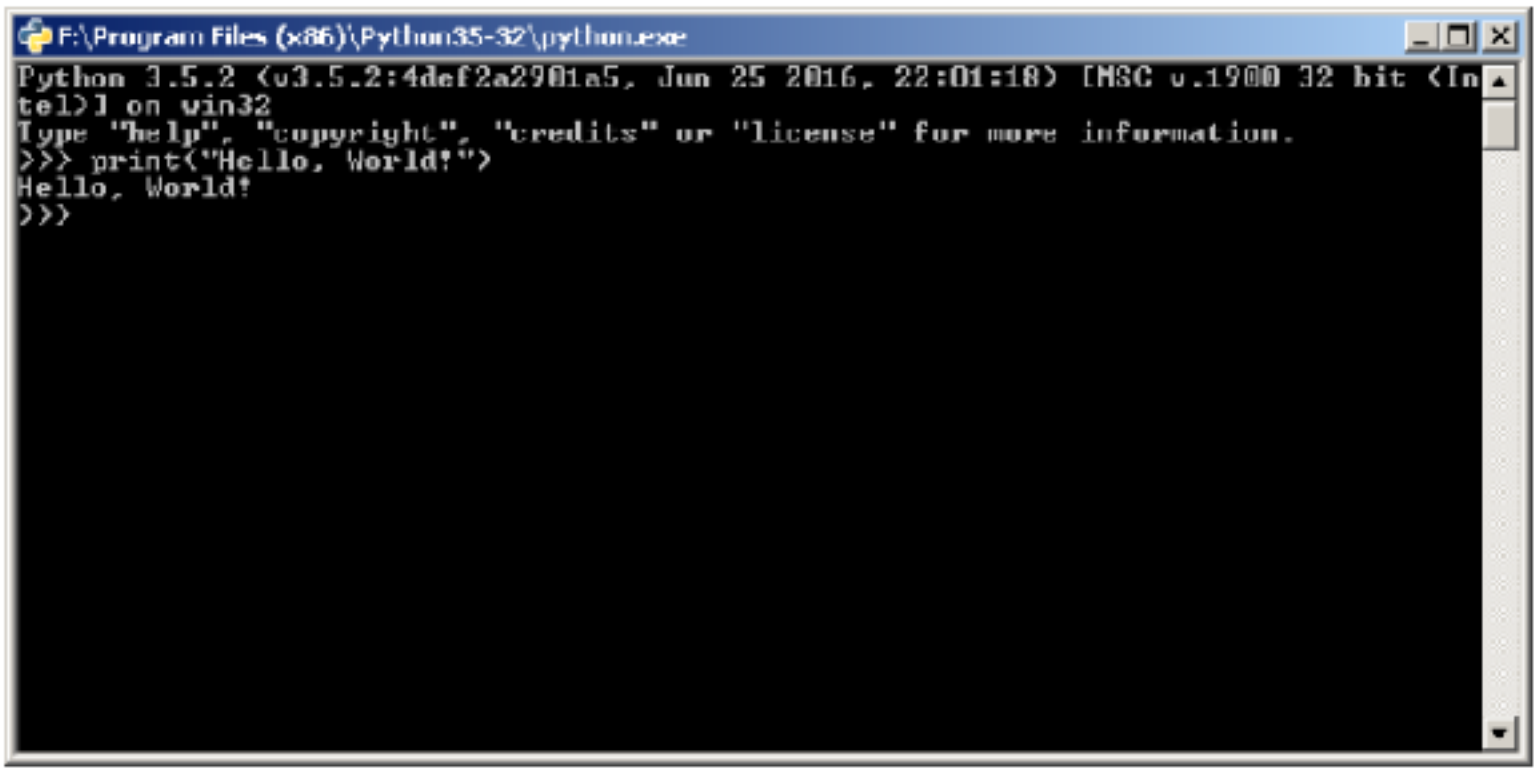
2.2.1 Перевірка інтерпретатора Python

Для початку протестуємо інтерпретатор в командному режимі. Якщо ви працюєте в Windows, то натисніть Win + R і у вікні введіть python.

В результаті Python запуститься в командному режимі, виглядати це буде приблизно так (картинка приведена для Windows, в Linux результат буде аналогічним):



У вікні введіть: print ("Hello, World!"). Результат повинен бути наступний:

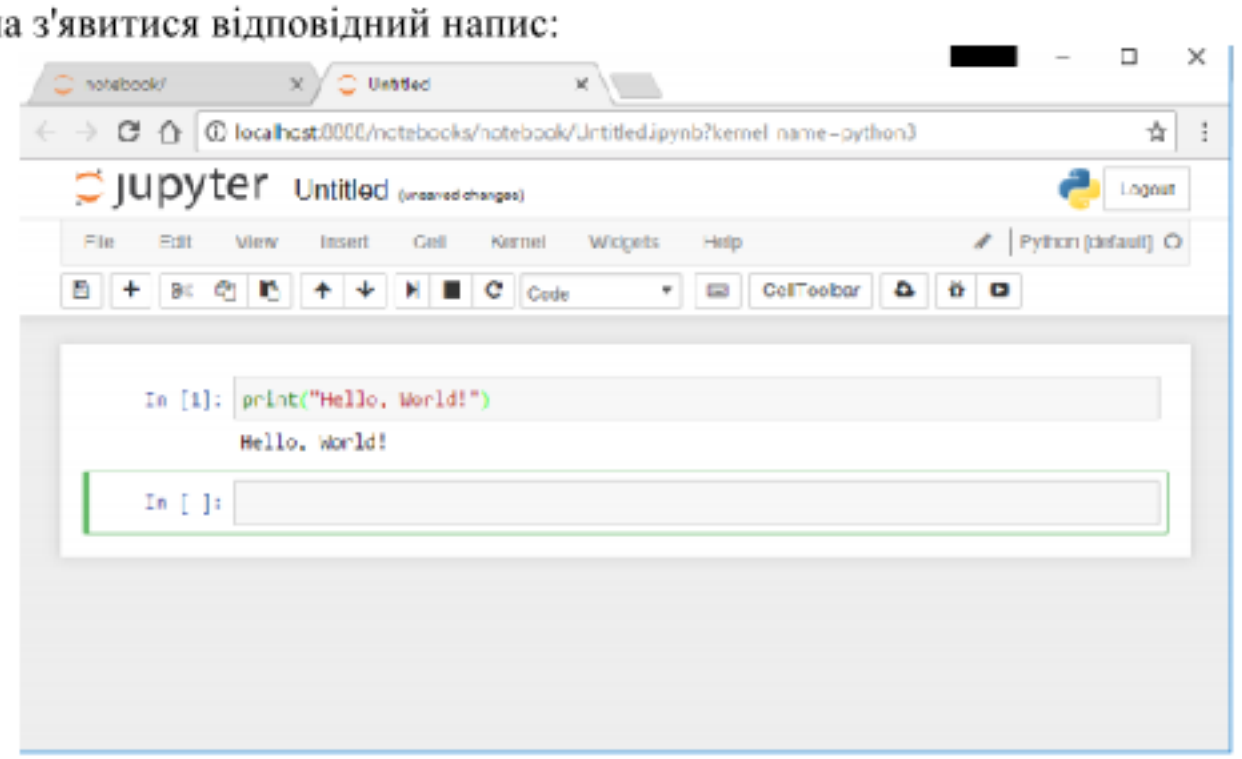


2.2.2 Перевірка Anaconda

Тут і далі будемо вважати, що пакет Anaconda встановлений в Windows, в папку C: \ Anaconda3.

Перейдіть в папку Scripts і введіть у командному рядку: > Ipython notebook Якщо ви перебуваєте в Windows і відкрили папку C: \ Anaconda3 \ Scripts через провідник, то для запуску інтерпретатора командного рядка для цієї папки в полі адреси введіть cmd.

В результаті запуститься веб-сервер і середовище розробки в браузері. Створіть ноутбук для розробки, для цього натисніть на кнопку New (в правому кутку вікна) і в списку, що з'явився виберіть Python. В результаті буде створена нова сторінка в браузері з ноутбуком. Введіть в першій клітинці команду print ("Hello, World!") і натисніть Alt + Enter на клавіатурі. Нижче осередку повинна з'явитися відповідний напис:



2.2.3 Перевірка PyCharm

Запустіть PyCharm і виберіть Create New Project.

Вкажіть шлях до проекту Python і інтерпретатор, який буде використовуватися для запуску та налагодження. Додайте Python файл в проект. Введіть код програми. Запустіть програму. В результаті має відкритися вікно з висновком програми.

2.3. Запуск програм на Python

Розглянемо два основних підходи до роботи з інтерпретатором Python - це безпосередня інтерпретація рядків коду, що вводяться з клавіатури в інтерактивному режимі і виконання фай-

лів з вихідним кодом в пакетному режимі. Також торкнемося деяких особливостей роботи з Python в Linux і MS Windows. Мова Python - це мова, що інтерпретується. Це означає, що крім безпосередньо самої програми, вам необхідний спеціальний інструмент для її запуску. Нагадаю, що існують компільовані й інтерпретовані мови програмування. У першому випадку, програма з мови високого рівня перекладається в машинний код для конкретної платформи. Надалі, серед користувачів, вона, як правило, поширюється у вигляді бінарного файлу. Для запуску такої програми не потрібні додаткові програмні засоби (за винятком необхідних бібліотек, але ці тонкощі виходять за рамки нашого предмету). Найпоширенішими мовами такого типу є C ++ і C. програми на інтерпретованих мовах, виконуються інтерпретатором і поширюються в вигляді вихідного коду.

Отже, Python може працювати в двох режимах:

1. інтерактивний;
2. пакетний.

2.3.1 Інтерактивний режим роботи

В інтерактивний режим можна увійти, набравши в командному рядку

> python

або

> python3

В результаті Python запуститься в інтерактивному режимі і буде очікувати введення команд користувача.

Якщо ж у вас є файл з вихідним кодом на Python, і ви його хочете запустити, то для цього потрібно в командному рядку викликати інтерпретатор Python і в якості аргументу передати ваш файл. Наприклад, для файлу з ім'ям test.py процедура запуску буде виглядати так:

>python test.py

Відкрийте Python в інтерактивному режимі і наберіть в ньому наступне:

print ("Hello, World!")

І натисніть ENTER.

У відповідь на це інтерпретатор виконає цього рядка і відобразить рядком нижче результат своєї роботи.

Python можна використовувати як калькулятор для різних обчислень, а якщо додатково підключити необхідні математичні бібліотеки, то за своїми можливостям він стає практично рівним таким пакетам як Matlab, Octave і т.п. Різні приклади обчислень наведені нижче. Більш докладно про арифметичні операції буде розглянуто в наступних лекціях.

Для виходу з інтерактивного режиму, наберіть команду **exit ()** і натисніть ENTER.

В комплекті разом з інтерпретатором Python йде IDLE (інтегроване середовище розробки). За своєю суттю вона подібна до інтерпретатора, запущеного в інтерактивному режимі з розширеним набором можливостей (підсвічування синтаксису, перегляд об'єктів, налагодження і т.п.). Для запуску IDLE в Windows необхідно перейти в папку Python в меню "Пуск" і знайти там ярлик з ім'ям "IDLE (Python 3.5 XX-bit)"

2.4 Пакетний режим роботи

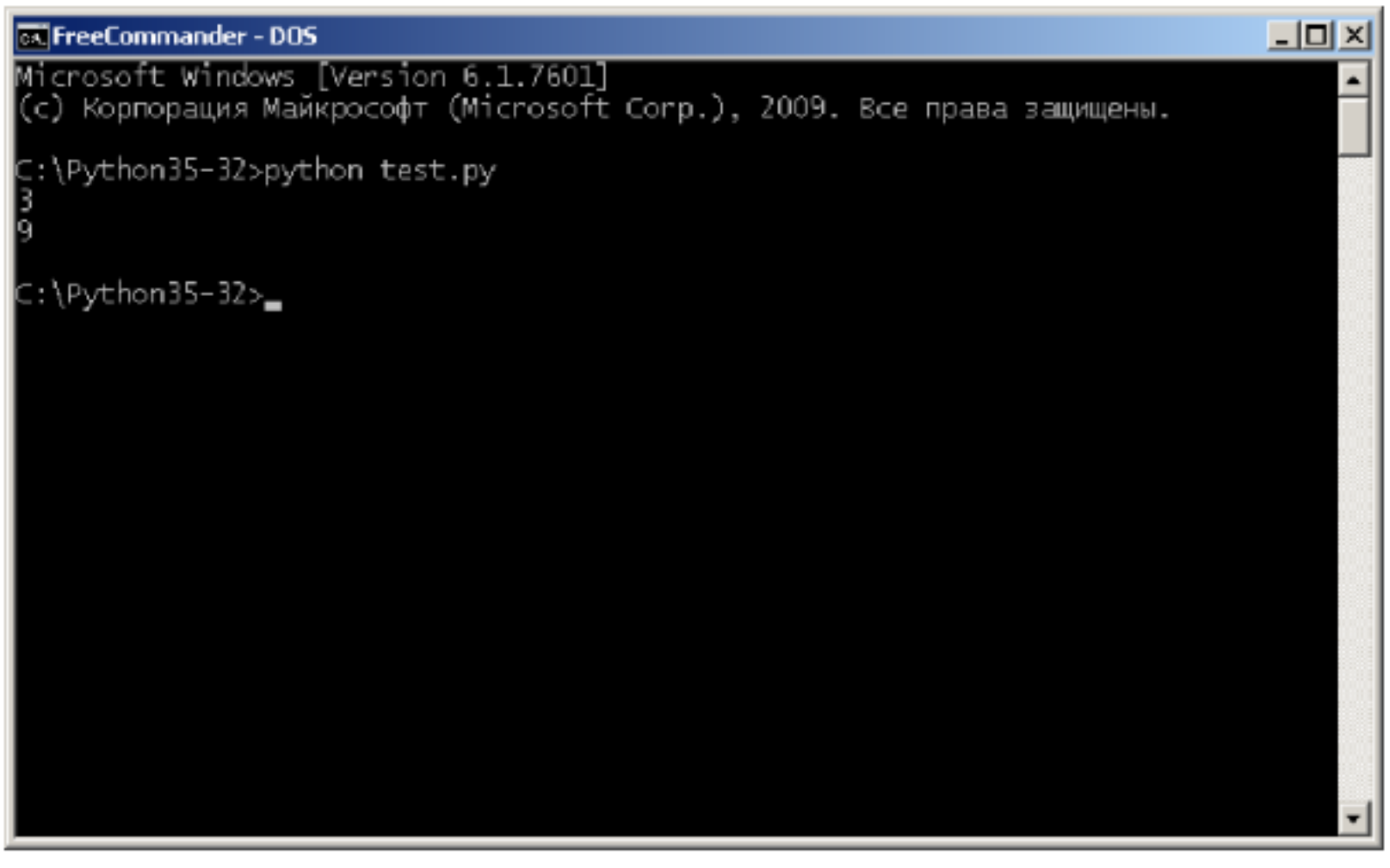
Тепер запустимо Python в режимі інтерпретації файлів з вихідним кодом (пакетний режим). Створіть файл з ім'ям test.py, відкрийте його за допомогою будь-якого текстового редактора і введіть наступний код:

a = int (input ()) print (a ** 2)

Ця програма приймає ціле число на вхід і виводить його квадрат. Для запуску, наберіть у командному рядку:

> Python test.py

Приклад роботи програми наведено нижче:



2.8. Типи і модель даних

В даній лекції розберемо як Python працює зі змінними і визначимо, які типи даних можна використовувати в рамках цієї мови. Детально розглянемо модель даних Python, а також механізми створення і зміни значення змінних.

Якщо досить формально підходити до питання про типізації мови Python, то можна сказати, що він відноситься до мов з динамічною типизацією. Неявна типізація означає, що при оголошенні змінної вам не потрібно вказувати її тип, при явній - це робити необхідно. Як приклад мов з явною типизацией можна привести Java, C ++. Ось як буде виглядати оголошення цілочисельний змінної в Java і Python.

– Java: int a = 1;

– Python: a = 1

Отже, мови програмування бувають з динамічної та статичної типізацею. В першому випадку тип змінної визначається безпосередньо при виконанні програми, у другому - на етапі компіляції (про компіляції та інтерпретації коротко розказано в лекції 1).

Як вже було сказано Python - це динамічно типізована мова, такі мови як C, C #, Java - статично типізовані. Сильна типізація не дозволяє проводити операції у виразах з даними різних типів, слабка - дозволяє. У мовах з сильною типізацією ви не можете складати наприклад рядки і числа, потрібно все приводити до одного типу. До першої групи можна віднести Python , Java , до другої - C і C ++ .

Типи даних можна розділити на вбудовані в інтерпретатор (built-in) і невбудовані, які можна використовувати при імпортуванні відповідних модулів.

До основних вбудованих типів відносяться:

1. None (невизначене значення змінної)
2. Логічні змінні (Boolean Type)
3. Числа (Numeric Type):
    a. int - ціле число
    b. float - число з плаваючою точкою
    c. complex - комплексне число
4. Списки (Sequence Type):
    a. list - список

b. tuple - кортеж
c. range – діапазон

5. Строки (Text Sequence Type):
   a. str
6. Бінарні списки (Binary Sequence Types):
   a. bytes – байти
   b. bytearray - масиви байт
   c. memoryview - спеціальні об'єкти для доступу до внутрішніх даних об'єкта через protocol buffer
7. Множини (Set Types):
   a. set – множина
   b. frozenset – неизменяемое множество
8. Словники (Mapping Types):
   a. dict – словник

2.5 Модель даних

Розглянемо як створюються об'єкти в пам'яті, їх влаштуваня, процес оголошення нових змінних та роботу операції присвоювання. Для того, щоб оголосити і відразу форматувати змінну необхідно написати її ім'я, потім поставити знак рівності і значення, з яким ця змінна буде створена.

Наприклад рядок: **b = 5** оголошує змінну b і привласнює їй значення 5. Цілочисленне значення 5 в рамках мови Python по своїй суті є об'єктом. Об'єкт, в даному випадку - це абстракція для представлення даних, дані - це числа, списки, рядки і т.п. При цьому, під даними слід розуміти як безпосередньо самі об'єкти, так і відносини між ними (про це трохи пізніше). Кожен об'єкт має три атрибути - це ідентифікатор, значення і тип.

Ідентифікатор - це унікальний ознака об'єкта, що дозволяє відрізняти об'єкти один від одного, а значення - безпосередньо інформація, що зберігається в пам'яті, якою управляє інтерпретатор.

При ініціалізації змінної, на рівні інтерпретатора, відбувається наступне:

1. створюється цілочисельний об'єкт 5 (можна уявити, що в цей момент створюється осередок і число 5 кладеться в цей осередок);
2. даний об'єкт має певний ідентифікатор, значення: 5, і тип: ціле число;
3. за допомогою оператора "=" створюється посилання між змінної b і цілочисельним об'єктом 5 (змінна b посилається на об'єкт 5).

Ім'я змінної не повинно збігатися з ключовими словами інтерпретатора Python. Список ключових слів можна отримати безпосередньо в програмі, для цього потрібно підключити модуль keyword і скористатися командою keyword.kwlist.

```
>>> import keyword
>>> print "Python keywords:", keyword.kwlist
```

Перевірити чи є ідентифікатор ключовим словом можна так:

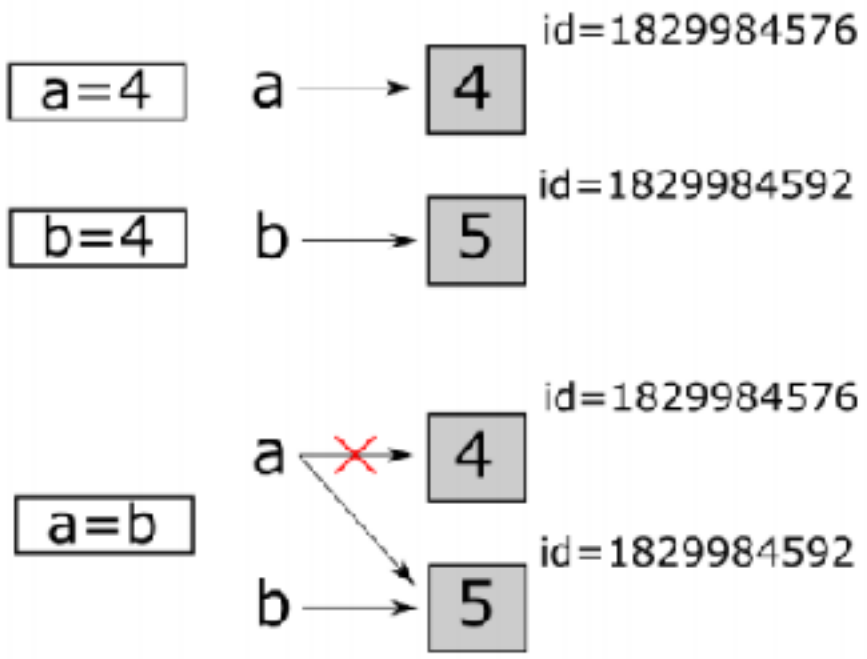
```
>>> keyword.  iskeyword ( "try")
True
>>> keyword.  iskeyword ( "b")
False
```

Для того, щоб подивитися на об'єкт з яким ідентифікатором посилається дана змінна, можна використовувати функцію id ().

```
>>> a = 4
>>> b = 5
>>> id (a)
```

```
1829984576
>>> id (b)
1829984592
>>> a = b
>>> id (a)
1829984592
```

Як видно з прикладу, ідентифікатор - це деяке цілочисельне значення, за допомогою якого унікально адресується об'єкт. Спочатку змінна a посилається на об'єкт 4 з ідентифікатором 1829984576, змінна b - на об'єкт з id = 1829984592. Після виконання операції присвоювання a = b, змінна a стала посилатися на той самий об'єкт, що і b, як представлено на рисю нижче:



Тип змінної можна визначити за допомогою функції type (). Приклад використання наведено нижче.

```
>>> a = 10
>>> b = "hello"
>>> c = (1, 2)
>>> type (a) <class 'int'>
>>> type (b) <class 'str'>
>>> type (c) <class 'tuple'>
```

2.6 Змінні і незмінні типи даних

У Python існують змінювані і незмінні типи.
До незмінних (immutable) відносяться:

–цілі числа (int);
–числа з плаваючою точкою (float);
–комплексні числа (complex);
–логічні змінні (bool);
–кортежі (tuple);
–рядки (str);
–незмінні безлічі (frozen set).

До змінних (mutable) типів відносяться:

– списки (list);
– безлічі (set);
– словники (dict).

Як вже було сказано раніше, при створенні змінної, спочатку створюється об'єкт, який має унікальний ідентифікатор, тип і значення. Після цього змінна може посилатися на створений об'єкт.

Незмінюваність типу даних означає, що створений об'єкт більше не змінюється. Наприклад, якщо ми оголосимо змінну k = 15, то буде створено об'єкт зі значенням 15, типу int і ідентифікатором, який можна дізнатися за допомогою функції id ().

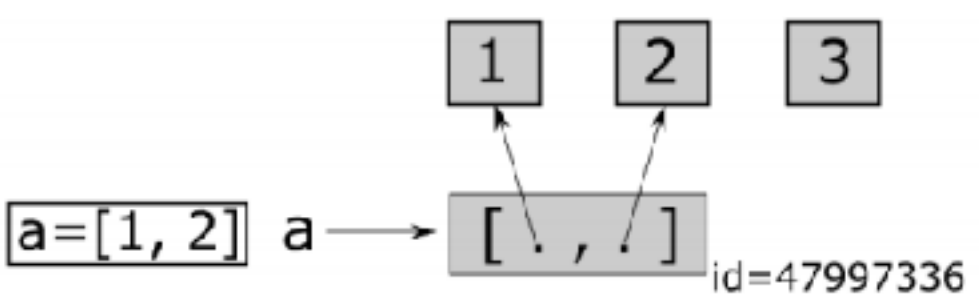
```
>>> k = 15
>>> id (k)
1672501744
>>> type (k) <class 'int'>
```

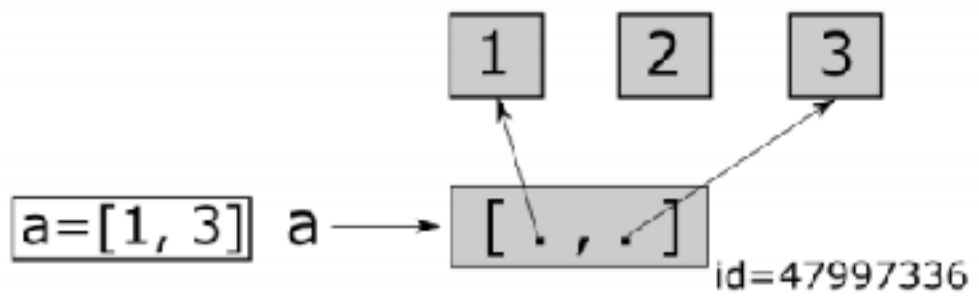
Об'єкт з id = 1672501744 матиме значення 15 і змінити його вже не можна.

Якщо тип даних змінюваний, то можна змінювати значення об'єкта. Наприклад, створимо список [1, 2], а потім замінимо другий елемент на 3.

```
>>> a = [1, 2]
>>> id (a) 47997336
>>> a [1] = 3
>>> a [1, 3]
>>> id (a) 47997336
```

Як видно, об'єкт на який посилається змінна a, був змінений. Це можна проілюструвати наступним малюнком:





Отже, у розглянутому випадку, в якості даних списку, виступають не об'єкти, а відносини між об'єктами. Тобто в змінній a зберігаються посилання на об'єкти , які містять числа 1 і 3, а не безпосередньо на самі ці числа.

2.7 Арифметичні операції

Мова Python, завдяки наявності великої кількості бібліотек для вирішення різного роду обчислювальних завдань, є конкурентом таким пакетам як Matlab і Octave. Запущений в інтерактивному режимі, він, фактично, перетворюється в потужний калькулятор. Зараз мова піде про арифметичні операції, доступних в даній мові. Як було сказано в попередньому уроці, присвяченому типами і моделі даних Python, в цій мові існує три вбудованих числових типи даних:

1. цілі числа (int);
2. з плаваючою комою числа (float);
3. комплексні числа (complex).

Якщо в якості операндів деякого арифметичного виразу використовуються тільки цілі числа, то результат теж буде ціле число. Винятком є операція ділення, результатом якої є дійсне число. При спільному використанні цілочисельних і речових змінних, результат буде число з плаваючою комою.

Всі експерименти будемо проводити в Python, запущеному в інтерактивному режимі.

Приклад:
Складати можна безпосередньо самі числа ...

```
>>> 3 +25
```

чи змінні, але вони повинні попередньо бути проініціалізовані.

```
>>> a = 3
>>> b = 2
>>> a + b
```

Результат операції додавання можна привласнити іншої змінної,

```
>>> a = 3
>>> b = 2
>>> c = a +  b
>>> print (c)
```

або їй же самій, в такому випадку можна використовувати повну або скорочену запис, повна виглядає так:

```
>>> a = 3
>>> b = 2
>>> a = a + b
>>> print (a)
```

скорочена так:

```
>>> a = 3
>>> b = 2
>>> a + = b
>>> print (a)
```

Всі перераховані вище варіанти використання операції додавання можуть бути застосовані для всіх нижченаведених операцій.

Віднімання.

```
>>> 4-2
2
>>> a = 5
>>> b = 7
>>> a – b
2
```

Множення.

```
>>> 5 * 8
40
>>> a = 4
>>> a * = 10
>>> print (a)
40
```

Ділення

```
>>> 9/3
3.0
 >>> a = 7
 >>> b = 4
 >>> a / b
 1. 75
```

Отримання цілої частини від ділення.

```
>>> 9 // 3
 3
```

Отримання дробової частини від ділення.

```
 >>> 9% 5
 4
>>> a = 7
>>> b = 4
 >>> a% b
 3
```

Піднесення до степеня.

```
 >>> 5 ** 4
 625
>>> a = 4
>>> b = 3
 >>> a ** b
 64
```

2.8 Робота з комплексними числами

Для створення комплексного числа можна використовувати функцію complex (a, b), в яку, в якості першого аргументу, передається дійсна частина, в якості другого - уявна. Або записати число у вигляді a + bj.

Розглянемо кілька прикладів. Створення комплексного числа.

```
 >>> z = 1 + 2j
>>> print (z)
(1 + 2j)
>>> x = complex (3, 2)
>>> print (x)
(3 + 2j)
```

Комплексні числа можна складати, віднімати, множити, ділити і підносити до степеня.

```
 >>> x + z
(4 + 4j)
 >>> x – z
 (2 + 0j)
 >>> x * z
 (-1 + 8j)
```

```
>>> x / z
(1. 4-0. 8j)
>>> x ** z
(-1. 1122722036363393-0. 01263518535533520 8j)
>>> x ** 3
(-9 + 46j)
```

У комплексного числа можна витягти дійсну і уявну частини.

```
>>> x = 3 + 2j
>>> x.real
3. 0
>>> x.imag
2. 0
```

Для отримання комплексносопряженного число необхідно використовувати метод conjugate ().

```
>>> x.conjugate ()
(3-2j)
```

2.9 Бітові операції

В Python доступні бітові операції, їх можна виробляти над цілими числами.
Побітове I (AND).

```
>>> p = 9
>>> q = 3
>>> p & q
1
```

Побітовое АБО (OR).

```
>>> p | q
11
```

Побітове виключаюче АБО (XOR).

```
>>> p ^ q
10
```

Інверсія.

```
>>> ~ p
-10
```

Зрушення вправо і вліво.

```
>>> p << 1
18
>>> p >> 1
4
```

2.10 Подання чисел в інших системах числення

У своєму повсякденному житті ми використовуємо десяткову систему числення, але при програмуванні, дуже часто, доводиться працювати з шістнадцятковій, двійковій і вісімковій.

Подання числа в шістнадцятковій системі.

```
>>> m = 124504
>>> hex (m)
 '0x1e658'
```

Подання числа в вісімковій системі.

```
>>> oct(m)
'0o363130'
```

Подання числа в двійковій системі.

```
>>> bin(m)
 '0b11110011001011000'
```

2.11 Бібліотека (модуль) math

У стандартну поставку Python входить бібліотека math, в якій міститься велика кількість часто використовуваних математичних функцій. Для роботи з даним модулем його попередньо потрібно імпортувати.

```
>>> import math
```

Розглянемо найбільш часто використовувані функції.

math.ceil(x) Повертає найближче ціле число більше, ніж x.

```
>>> math.ceil (3.2)
 4
```

math.fabs (x) Повертає абсолютне значення числа.

```
 >>> math.fabs (-7)
7.0
```

math.factorial (x) Обчислює факторіал x.

```
>>> math.  factorial (5)
120
```

math.floor (x) Повертає найближче ціле число менше, ніж x.

```
>>> math.floor (3. 2)
3
```

m ath.exp(x) Обчислює e ** x.

```
 >>> math.  exp (3)
20. 08553692318766
```

math.log2 (x) Логарифм по основі 2.
math.log10 (x) Логарифм по основі 10.
math.log (x [base]) За замовчуванням обчислює логарифм за основою e, додатково можна вказати підставу логарифма.

>>> math. log2 (8)
3. 0
>>> math.log10 (1000)
3. 0
>>> math. log (5)
1. 609437912434100
>>> math. log(4, 7)
0.7124143742160444

math.pow (x, y) Обчислює значення x в ступені y.

>>> math. pow (3, 4)
81.0

math.sqrt (x) Корінь квадратний від x.

>>> math. sqrt (25)
5.0

Тригонометричні функції залишимо без прикладу:
– math.cos (x)
– math.sin (x)
– math.tan (x)
– math.acos (x)
– math.asin (x)
– math.atan (x)

І наостанок пару констант:
– math.pi число пі.
– math.e число e.

Крім перерахованих, модуль math містить ще багато різних функцій, за більш детальною інформацією можете звернутися на офіційний сайт (https://docs.python.org/3/library/math.html).

Лекція 3. Умовні оператори та цикли

В даній лекції буде розглянуто оператор розгалуження if та оператори циклу while і for. Основна мета - дати загальне уявлення про ці оператори і на простих прикладах показати базові принципи роботи з ними.

3.1 логічний операт if

Оператор розгалуження if дозволяє виконати певний набір інструкцій залежно від деякої умови. Можливі наступні варіанти використання:

Синтаксис оператора if виглядає так.

if вираз:
інструкція_1
інструкція_2
. . .
інструкція_n

Після оператора if записується вираз. Якщо цей вислів істинно, то виконуються інструкції, які визначаються даними оператором.

Вираз є істинним, якщо його результатом є число не рівне нулю, непорожній об'єкт, або логічне True.

Після висловлення потрібно поставити двокрапку ":".

ВАЖЛИВО: блок коду, який необхідно виконати, в разі вислову, відділяється чотирма пробілами зліва!

Приклади:

```
if 1:
        print ( "hello 1")
```
надрукує: hello 1

```
a = 3
if a == 3:
        print ( "hello 2")
```
надруковує: hello 2

```
a = 3
if a> 1:
        print (  "hello 3")
```
надруковано: hello 3

```
lst = [1, 2, 3]
if lst:
        print ( "hello 4")
```
надруковано: hello 4

3.2 Конструкція if – else

Бувають випадки, коли необхідно передбачити альтернативний варіант виконання програми. Тобто при істинному умові потрібно виконати один набір інструкцій, при помилковому - інший. Для цього використовується конструкція if - else.

```
if вираз:
        інструкція_1
        інструкція_2
        . . .
        інструкція_n
else:
        інструкція_a
        інструкція_b
        . . .
        інструкція_x
```

Приклади.

```
a = 3
 if a> 2:
        print ( "H")
 else:
         print ( "L")
```
надруковано: H

```
a = 1
 if a> 2:
        print ( "H")
 else:
        print ( "L"  )
```
надруковано: L

Умова такого виду можна записати в рядок, в такому випадку воно буде являти собою тернарного вираз.

```
a = 17
b = True
if a> 10 else False
print (b)
```

У результаті виконання такого коду буде надруковано: **True**

3.3 Конструкція if - elif - else

Для реалізації вибору з декількох альтернатив можна використовувати конструкцію if - elif - else.

```
 if вираз_1:
        інструкції_(блок_1)
 elif вираз_2:
        інструкції_ (блок_2)
elif вираженіе_3:
        інструкції_ (блок_3)
else:
        інструкції_ (блок_4)
```

Приклад.

```
a = int (input ( "введіть число:"))
if a <0:
        print ( "Neg")
elif a == 0:
        print ( "Zero")
else:
        print ( "Pos")
```

Якщо користувач введе число менше нуля, то буде надруковано "Neg", рівне нулю - "Zero", більше нуля - "Pos".

3.4 Оператор циклу while

Оператор циклу while виконує вказаний набір інструкцій до тих пір, поки умова циклу істинно. Істинність умови визначається також як і в операторі if. Синтаксис оператора while виглядає так.

```
 while вираз:
        інструкція_1
        інструкція_2
       . . . інструкція_n
```

Здійснюється набір інструкцій називається тілом циклу. Приклад.

```
a = 0
 while a <7:
        print ( "A")
a + = 1
```

Буква "A" буде виведена сім раз в стовпчик. Приклад нескінченного циклу.

```
a = 0
while a == 0:
        print ( "A")
```

3.5 Оператори break і continue

При роботі з циклами використовуються оператори break і continue. Оператор break призначений для дострокового переривання роботи циклу while.

```
a = 0
while a> = 0:
if a == 7:
        break
a + = 1
print ( "A")
```

У наведеному вище коді, вихід з циклу відбудеться при досягненні змінної a значення 7. Якби не було цієї умови, то цикл виконувався б нескінченно.

Оператор continue запускає цикл заново, при цьому код, розташований після цього оператора, не виконується. Приклад.

```
 a = -1
 while a <10:
a + = 1
 if a> = 7:
        continue
print ( "A")
```

При запуску даного коду символ "A" буде надруковано 7 разів, незважаючи на те, що все буде виконано 11 проходів циклу.

3.6 Оператор циклу for

Оператор for виконує вказаний набір інструкцій задану кількість разів, яке визначається кількістю елементів в наборі. Приклад.

```
 for i in range (5):
        print ( "Hello")
```

В результаті "Hello" буде виведено п'ять разів.

Усередині тіла циклу можна використовувати оператори break і continue, принцип роботи їх точно такий же як і в операторі while.

Якщо у вас є заданий список, і ви хочете виконати над кожним елементом певну операцію (звести в квадрат і роздрукувати вийшло число), то за допомогою for таке завдання вирішується так.

```
 lst = [1, 3, 5, 7, 9]
for i in lst:
        print (i ** 2)
```

Також можна пройти по всіх буквах в рядку.

```
word_str = "Hello, world!"
for i in word_str:
        print (l)
```

Рядок "Hello, world!" буде надрукована в стовпчик. На цьому закінчимо короткий огляд операторів розгалуження і циклу.

Лекція 4. Робота зі списками (list)

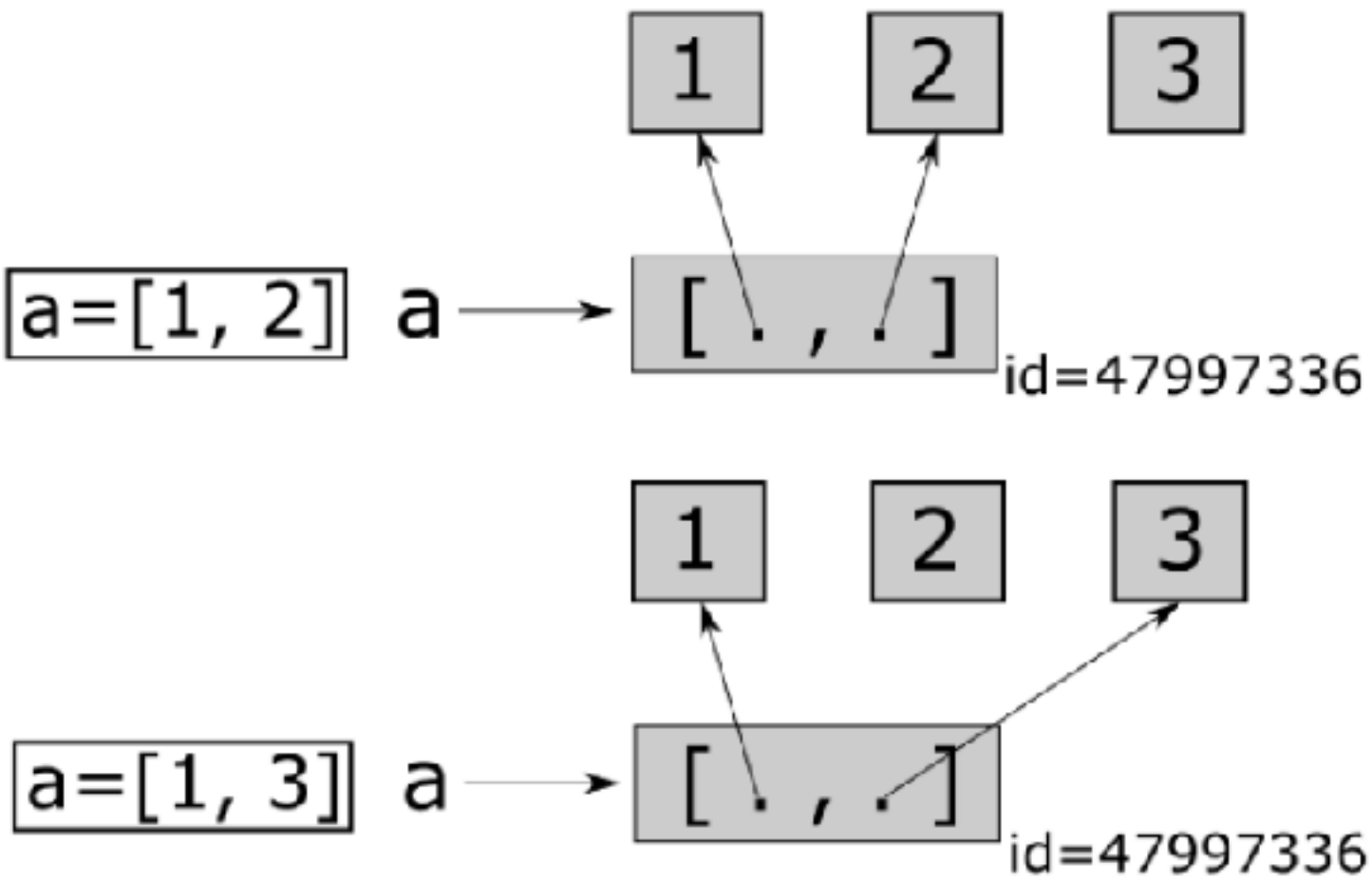
Одна з ключових особливостей Python, завдяки якій він є таким популярним - це простота. Особливо підкуповує простота роботи з різними структурами даних - списками, кортежами, словниками та множинами. Даний урок, присвячений списками.

4.1 Що таке список (list) в Python?

Список (list) - це структура даних для зберігання об'єктів різних типів. Якщо ви використовували інші мови програмування, то вам повинно бути знайоме поняття масиву. Так ось, список дуже схожий на масив, тільки, як було вже сказано вище, в ньому можна зберігати об'єкти різних типів. Розмір списку не статичний, його можна змінювати. Список за своєю природою є змінним типом даних. Про типи даних можна докладно прочитати в третьому уроці. Змінна, яка визначається як список, містить посилання на структуру в пам'яті, яка в свою чергу зберігає посилання на будь-які інші об'єкти або структури.

Як списки зберігаються в пам'яті?

Як вже було сказано вище, список є змінним типом даних. При його створенні, в пам'яті резервується область, яку можна умовно назвати деяким "контейнером", в якому зберігаються посилання на інші елементи даних в пам'яті. На відміну від таких типів даних як число або рядок, вміст "контейнера" списку можна змінювати. Для того, щоб краще візуально уявляти собі цей процес погляньте на картинку нижче.



Спочатку був створений список,що містить посилання на об'єкти 1 і 2, після операції a [1] = 3, друга посилання в списку стала вказувати на об'єкт 3.

4.2 Створення, зміна, видалення списків і робота з його елементами

Створити список можна одним з таких способів.

```
>>> a = []
 >>> type (a)
 <class 'list'>
 >>> b = list ()
>>> type (b)
 <class 'list'>
```

Також можна створити список з наперед заданим набором даних.

```
>>> a = [1, 2, 3]
>>> type (a)
<class 'list'>
```

Якщо у вас вже є список і ви хочете створити його копію, то можна скористатися наступним способом:

```
 >>> a = [1, 3, 5, 7]
 >>> b = a [:]
 >>> print (a)
 [1, 3, 5, 7]
>>> print (b)
 [1, 3, 5  , 7]
```

або зробити це так:

```
 >>> a = [1, 3, 5, 7]
>>> b = list (a)
 >>> print (a)
[1, 3, 5, 7]
 >>>  print (b)
 [1, 3, 5, 7]
```

в разі, якщо ви виконаєте просте присвоєння списків один одному, то змінної b буде присвоєна посилання на той же елемент даних в пам'яті, на який посилається a, а не копія списку a.

Тобто якщо ви будете змінювати список a, то і b теж буде змінюватися.

```
>>> a = [1, 3, 5, 7]
 >>> b = a >>>
>>>print (a) [1, 3, 5, 7]
[1, 3, 5, 7]
 >>> print (b) [1, 3, 5,  7]
[1, 3, 5, 7]
>>> a [1] = 10
 >>> print (a)
 [1, 10, 5, 7]
>>> print (b)
[1, 10, 5, 7]
```

Додавання елемента в список здійснюється за допомогою методу append().

```
>>> a = []
>>> a. append (3)
>>> a. append ( "hello")
>>> print (a)
[3, 'hello']
```

Для видалення елемента зі списку, в разі, якщо ви знаєте його значення, використовуйте метод remove (x), при цьому буде видалена перша посилання на даний елемент.

```
>>> b = [2, 3, 5]
>>> print (b) [2, 3, 5]
>>> b.  remove (3)
>>> print (b)
[2, 5]
```

Якщо необхідно видалити елемент по його індексу, скористайтеся командою **del назва_списку [індекс]**

```
>>> c = [3, 5, 1, 9, 6]
>>> print (c)
 [3, 5, 1, 9, 6]
>>> del c [2]
>>> print (c)
 [  3, 5, 9, 6]
```

Змінити значення елемента списку, знаючи його індекс, можна звернувшись безпосередньо до нього.

```
>>> d = [2, 4, 9]
>>> print (d)
[2, 4, 9]
>>> d [1] = 17
>>> print (d)
[2, 17, 9]
```

Очистити список можна просто заново його проініціалізувати, так як ніби ви його знову створюєте. Для отримання доступу до елемента списку вкажіть індекс цього елемента в квадратних дужках.

```
>>> a = [3, 5, 7, 10, 3, 2, 6, 0]
>>> a [2]
7
```

Можна використовувати негативні індекси, в такому випадку рахунок буде йти з кінця, наприклад для доступу до останнього елемента списку можна використовувати ось таку команду:

```
>>> a [-1]
0
```

Для отримання зі списку деякого подсписка в певному діапазоні індексів, вкажіть початковий і кінцевий індекс в квадратних дужках, розділивши їх двокрапкою.

>>> a [1: 4]
[5, 7, 10]

4.3 Методи списків

list.append (x)

Додає елемент в кінець списку. Ту ж операцію можна зробити так: a [len (a):] = [x].

>>> a = [1, 2]
>>> a. append (3)
>>> print (a)
[1, 2, 3]

list.extend (L)

Розширює існуючий список за рахунок додавання всіх елементів зі списку L. Еквівалентно команді a [len (a):] = L.

> >> a = [1, 2]
>>> b = [3, 4]
>>> a. extend (b)
>>> print (a)
[1, 2, 3, 4]

list.insert (i, x)

Вставити елемент x в позицію i. Перший аргумент - індекс елемента після якого буде вставлений елемент x.

>>> a = [1, 2]
>>> a. insert (0, 5)
>>> print (a) [5, 1, 2]
>>> a. insert (len (a), 9)
>>> print (a) [5, 1, 2, 9]

list.remove (x)
Видаляє перше входження елемента x зі списку.

>>> a = [1, 2, 3]
>>> a. remove (1)
>>> print (a) [2, 3]

list.pop ([i])

Видаляє елемент з позиції i і повертає його. Якщо використовувати метод без аргументу, то буде видалений останній елемент зі списку.

>>> a = [1, 2, 3, 4, 5]
>>> print (a. Pop (2))
3
> >> print (a. Pop ())
5

> >> print (a)
[1, 2, 4]

list.clear ()

Видаляє всі елементи зі списку. Еквівалентно del a [:].

>>> a = [1, 2, 3, 4, 5]
>>> print (a) [1, 2, 3, 4, 5]
>>> a. clear ()
>>> print (a) []

list.index (x [start [end]])

Повертає індекс елемента.

>>> a = [1, 2, 3, 4, 5]
>>> a. index (4)
3

list.count (x)

Повертає кількість входжень елемента x в список.

>>> a = [1, 2, 2, 3, 3]
>>> print (a. Count (2))
2

lis t.sort (key = None, reverse = False)

Сортує елементи в списку по зростанню. Для сортування в зворотному порядку використовуйте прапор reverse = True. Додаткові можливості відкриває параметр key, за більш детальною інформацією зверніться до документації.

>>> a = [1, 4, 2, 8, 1]
>>> a. sort ()
>>> print (a)
[1, 1, 2, 4, 8]

list.reverse ()

Змінює порядок розташування елементів у списку на зворотний.

>>> a = [1, 3, 5, 7]
>>> a. reverse ()
>>> print (a)
[7, 5, 3, 1]

list.copy ()

Повертає копію списку. Еквівалентно a [:].

>>> a = [1, 7, 9]
>>> b = a. copy ()
>>> print (a) [1, 7, 9]

```
>>> print (b) [1, 7, 9]
>>> b [0] = 8
>>> print (a) [1,  7, 9]
>>> print (b) [8, 7, 9]
```

4.4 List Comprehensions

List Comprehensions найчастіше на укр. мову перекладають як "абстракція списків" або "списковий включення", є частиною синтаксису мови, яка надає простий спосіб побудови списків. Найпростіше роботу list comprehensions показати на прикладі. Припустимо вам необхідно створити список цілих чисел від 0 до n, де n попередньо задається. Класичний спосіб вирішення даного завдання виглядав би так:

```
n = int (input ())
a = []
for i in range (n):
        a.  append (i)
        print (a)
```

Використання list comprehensions дозволяє зробити це значно простіше:

```
n = int (input ())
a = [i for i in range (n)]
print (a)
```

або взагалі ось так, в разі якщо вам не потрібно більше використовувати n:

```
a = [i for i in range (int (input ()))]
print (a)
```

Лекція 5. Кортежі (tuple)

Лекція присвячена кортежам (tuple) в Python. Основна увага приділена питанню використання кортежів, розглянуті способи створення і основні прийоми роботи з ними. Також торкнемося теми перетворення кортежу в список і назад.

5.1 Що таке кортеж (tuple) в Python?

Кортеж (tuple) - це незмінна структура даних, яка за своєю подобою дуже схожа на список. Тобто якщо у нас є список a = [1, 2, 3] і ми хочемо замінити другий елемент з 2 на 15, то ми може це зробити, безпосередньо звернувшись до елемента списку.

```
>>> a = [1, 2, 3]
>>> print (a)
[1, 2, 3]
>>> a [1] = 15
>>> print (a)
[1, 15, 3]
```

З кортежем ми не можемо виробляти такі операції, тому що елементи його змінювати не можна.

```
>>> b = (1, 2, 3)
>>> print (b)
(1, 2, 3)
```

```
>>> b [1] = 15
Traceback (most recent call last):
        File "<pyshell # 6>  ", line 1, in <module>
         b [1] = 15
 TypeError: 'tuple' object does not support item assignment
```

Навіщо потрібні кортежі в Python?

Існує кілька причин, за якими варто використовувати кортежі замість списків. Одна з них - це убезпечити дані від випадкового зміни. Якщо ми отримали звідкись масив даних, і у нас є бажання попрацювати з ним, але при цьому безпосередньо змінювати дані ми не збираємося, тоді, це як раз той випадок, коли кортежі припадуть як не можна до речі. Використовуючи їх в даній задачі, ми додатково отримуємо відразу кілька бонусів - по-перше, це економія місця. Справа в тому, що кортежі в пам'яті займають менший об'єм в порівнянні зі списками.

```
 >>> lst = [10, 20, 30]
>>> tpl = (10, 20, 30)
 >>> print (lst. __sizeof __ ())
32
>>> print (tpl. __sizeof __ ())
 24
```

По друге - приріст продуктивності, який пов'язаний з тим, що кортежі працюють швидше, ніж списки (тобто на операції перебору елементів і т.п. буде витрачатися менше часу). Важливо також відзначити, що кортежі можна використовувати в якості ключа у словника.

5.2 Створення, видалення кортежів і робота з його елементами

5.2.1 Створення кортежів

Для створення пустого кортежу можна скористатися однією з наступних команд.

```
>>> a = ()
 >>> print (type (a))
<class 'tuple'>
>>> b = tuple ()
>>> print (type (b))
<class 'tuple'>
```

Кортеж із заданим вмістом створюється також як список, тільки замість квадратних дужок використовуються круглі.

```
 >>> a = (1, 2, 3, 4, 5)
>>> print (type (a))
<class 'tuple'>
 >>> print (a)
(1, 2, 3, 4, 5)
```

при бажанні можна скористатися функцією tuple ().

```
>>> a = tuple ((1, 2, 3, 4))
>>> print (a)
(1, 2, 3, 4)
```

5.2.2 Доступ до елементів кортежу

Доступ до елементів кортежу здійснюється також як до елементів списку - через вказівку індексу. Але, як вже було сказано - змінювати елементи кортежу не можна!

```
>>> a = (1, 2, 3, 4, 5)
>>> print (a [0])
1
>>>
 print (a [1: 3])
 (2, 3)
 >>> a [1 ] = 3
 Traceback (most recent call last):
        File "<pyshell # 24>", line 1, in <module>
              a [1] = 3
 TypeError: 'tuple' object does not support item assignment
```

5.2.3 Видалення кортежів

Видалити окремі елементи з кортежу неможливо.

```
>>> a = (1, 2, 3, 4, 5)
>>> del a [0]
        Traceback (most recent call last):
              File "<pyshell # 26>", line 1, in <module>
                    del a  [0]
        TypeError: 'tuple' object doesn 't support item deletion
```

Але можна видалити кортеж цілком.

```
>>> del a
>>> print (a)
 Traceback (most recent call last):
        File "<pyshell # 28>", line 1, in <module>
         print (a)
 NameError: name 'a' is not defined
```

5.2.4 Перетворення кортежу в список і назад

На базі кортежу можна створити список, вірно і зворотне твердження. Для перетворення списку в кортеж досить передати його в якості аргументу функції tuple().

```
 >>> lst = [1, 2, 3, 4, 5]
>>> print (type (lst))
 <class 'list'>
 >>> print (lst)
 [1, 2, 3, 4, 5]
 >>> tpl = tuple (lst)
 >>> print (type (tpl))
<class 'tuple'>
 >>> print (tpl)
(1, 2, 3, 4, 5)
```

Зворотна операція також є коректною.

```
>>> tpl = (2, 4, 6, 8, 10)
>>> print (type (tpl)) <class 'tuple'>
>>> print (tpl) (2, 4, 6, 8, 10)
>>> lst = list (tpl)
>>> print (type (lst))
<class 'list'>
>>> print (lst)
[2, 4, 6, 8, 10]
```

Лекція 6 Словники (dict)

6.1 Що таке словник (dict) в Python?

Словник (dict) являє собою структуру даних (яка ще називається асоціативний масив), призначену для зберігання довільних об'єктів з доступом по ключу. Дані в словнику зберігаються в форматі ключ - значення. Якщо згадати таку структуру як список, то доступ до його елементів здійснюється за індексом, який являє собою ціле позитивне число, причому ми самі, безпосередньо, не беремо участі в його створенні (індексу). У словнику аналогом індексу є ключ, при цьому відповідальність за його формування лягає на програміста.

6.2 Створення, зміна, видалення словників і робота з його елементами 6.2.1 Створення словника

Порожній словник можна створити, використовуючи функцію dict (), або просто вказавши порожні фігурні дужки.

```
>>> d1 = dict ()
>>> print (type (d1))
<class 'dict'>
>>> d2 = {}
>>> print (type (d2))
<class 'dict'>
```

Якщо необхідно створити словник із заздалегідь підготовленим набором даних, то можна використовувати один з перерахованих вище підходів, але з перерахуванням груп ключ-значення.

```
>>> d1 = dict (Ivan = "manager", Mark = "worker")
>>> print (d1)
{ 'Mark': 'worker', 'Ivan': 'manager'}
>>> d2 = { "  A1 ":" 123 "," A2 ":" 456 "}
>>> print (d2)
{ 'A2': '456', 'A1': '123'}
```

6.2.2 Додавання і видалення елемента

Щоб додати елемент в словник потрібно вказати новий ключ і значення.

```
>>> d1 = { "Russia": "Moscow", "USA": "Washington"}
>>> d1 [ "China"] = "Beijing"
>>> print (d1)
{ 'Russia': 'Moscow'  , 'China': 'Beijing', 'USA': 'Washington'}
```

Для видалення елемента зі словника можна скористатися командою del. >>>

```
d2 = { "A1": "123", "A2": "456"}
 >>> del d2 [ "A1"]
>>> print (d2)
{ 'A2': '456'}
```

6.2.3 Робота зі словником

Перевірка наявності ключа в словнику проводиться за допомогою оператора in.

```
>>> d2 = { "A1": "123", "A2": "456"}
 >>> "A1" in d2
 True
 >>>
 "A3" in d2
 False
```

Доступ до елементу словника, здійснюється так же, як доступ до елементу списку, тільки в якості індексу вказується ключ.

```
 >>> d1 = {"Russia":"Moscow","USA":"Washington"}
>>> d1 [ "Russia"]
'Moscow'
```

6.3 Методи словників

У словників доступний наступний набір методів.

clear ()

Видаляє всі елементи словника.

```
 >>> d2 = { "A1": "123", "A2": "456"}
>>> print (d2) { 'A2': '456', 'A1': '123'}
>>> d2.clear ()
>>> print (d2)
 {}
```

copy ()

Створює нову копію словника.

```
>>> d2 = { "A1": "123", "A2": "456"}
>>> d3 = d2.  copy ()
>>> print (d3)
{ 'A1': '123', 'A2': '456'}
>>> d3 [ "A1"] = "789"
>>> print (d2)
{ 'A2  ':' 456 ',' A1 ':' 123 '}
 >>> print (d3)
{' A1 ':' 789 ',' A2 ':' 456 '}
```

fromkeys (seq [value])

Створює новий словник з ключами з seq і значеннями з value. За замовчуванням value присвоюється значення None.

get (key)

Повертає значення зі словника по ключу key.

```
>>> d = { "A1": "123", "A2": "456"}
>>> d.get ( "A1") '123'
```

items ()

Для отримання елементів словника (ключ, значення) в отформатированном вигляді.

```
>>> d = {"A1":"123","A2": "456"}
>>> d.items ()
dict_items ([( 'A2', '456'), ( 'A1', '123')])
```

keys ()

Повертає ключі словника.

```
>>> d = { "A1": "123", "A2": "456"}
>>> d.  keys () dict_keys ([ 'A2', 'A1'])
```

pop (key [default])

Якщо ключ key є в словнику, то даний об'єкт був видалений з словника і повертається значення по цьому ключу, інакше буде повернуто значення default. Якщо default невідомий і запитуваний ключ відсутній в словнику, то буде викликано виключення KeyError.

```
>>> d = { "A1": "123", "A2": "456"}
>>> d.pop ("A1")
'123'
>>> print (d)
 {'A2': '456'}
```

popitem ()

Видаляє і повертає пару (ключ, значення) зі словника. Якщо словник порожній, то буде викликано виключення KeyError.

```
 >>> d = { "A1": "123", "A2": "456"}
>>> d.  popitem ()
( 'A2', '456')
>>> print (d)
{ 'A1': '123'}
```

setdefault (key [default])

Якщо ключ key є в словнику, то повертається значення по ключу. Якщо такого ключа немає, то в словник вставляється елемент з ключем key і значенням default, якщо default не визначений, то за замовчуванням присвоюється None.

```
>>> d = { "A1": "123", "A2": "456"}
>>> d.setdefault ( "A3", "777")
'777'
```

```
>>> print (d)
 {'A2': '456', 'A3': '777', 'A1': '123'}
>>> d.setdefault ( "A1")
 '123'
>>> print (d)
{ 'A2': '456', 'A3': '777', 'A1': '123'}
```

update ([other])

Обновляє словник парами (key / value) з other, якщо ключі вже існують, то оновлює їх значення.

```
>>> d = {"A1": "123", "A2": "456"}
 >>> d.  update ({"A1":"333", "A3": "789"})
>>> print (d) {'A2': '456', 'A3': '789', 'A1': '333'}
```

values ()

Повертає значення елементів словника.

```
>>> d = { "A1": "123", "A2": "456"}
>>> d.values ()
 dict_values ([ '456', '123'])
```

Лекція 7. Функції в Python

Лекція присвячена створенню функцій в Python і роботі з ними (передача аргументів, повернення значення і т.п.). Також розглянуті lambda-функцій, їх особливості та використання.

7.1 Що таке функція в Python?

За своєю суттю функції в Python практично нічим не відрізняються від функцій з інших мов програмування. Функцією називають іменований фрагмент програмного коду, до якого можна звернутися з іншого місця вашої програми (але є lambda-функції, у яких немає імені, про них буде розказано в кінці). Як правило, функції створюються для роботи з даними, які передаються їй як аргументи, також функція може формувати певний значення, що повертається.

7.2 Створення функцій

Для створення функції використовується ключове слово def, після якого вказується ім'я та список аргументів у круглих дужках. Тіло функції виділяється також як тіло умови (або циклу): чотирма пробілами. Таким чином найпростіша функція, яка нічого не робить, буде виглядати так.

```
def fun ():
    pass
```

Повернення значення функцією здійснюється за допомогою ключового слова return, після якого вказується значення, що повертається.

Приклад функції повертає одиницю представлений нижче.

```
>>> def fun ():
    return 1
>>> fun ()
```

1

7.3 Робота з функціями

У багатьох випадках функції використовують для обробки даних.

Ці дані можуть бути глобальними, або передаватися в функцію через аргументи. Список аргументів визначається на етапі реалізації і вказується в круглих дужках після імені функції. Наприклад операцію складання двох аргументів можна реалізувати ось так.

```
>>> def summa (a, b):
        return a + b
>>> summa (3, 4)
7
```

Розглянем ще два приклади використання функції:
обчислення числа Фібоначчі з використанням рекурсії і обчислення факторіала з використанням циклу.

Обчислення числа Фібоначчі.

```
>>> def fibb (n):
        if n == 0:
            return 0
      elif n == 1:
            return 1
       elif n == 2:
            return 1
        else: return fibb (n-1) + fibb (n-2)
>>> print (fibb (10))
```

Обчислення факторіала.

```
>>> def factorial (n):
prod = 1
for i in range (1, n + 1):
        prod * = i
        return prod
>>> print (factorial (5))
120
```

Функцію можна привласнити змінної і використовувати її, якщо необхідно скоротити ім'я. Як приклад можна привести варіант використання функції обчислення факторіала з пакета math.

```
 >>> import math
>>> f = math. factorial
>>> print (f (5))
120
```

7.4 Lambda-функції

Lambda-функція - це безіменна функція з довільним числом аргументів і обчислює один втраз. Тіло такої функції не може містити більше однієї інструкції (або виразу). Дану функцію можна використовувати в рамках будь-яких конвеєрних обчислень (наприклад всередині filter (), map () і reduce ()) або самостійно, в тих місцях, де потрібно провести якісь обчислення, які зручно "загорнути" в функцію.

```
>>> (lambda x: x ** 2) (5)
 25
```

Lambda-функцію можна привласнити будь-якої змінної і надалі використовувати її в якості імені функції.

```
>>> sqrt = lambda x: x ** 0.  5
>>> sqrt (25)
5.0
```

Списки можна обробляти lambda-функціями всередині таких функцій як map(), filter (), reduce (). Функція map приймає два аргументи, перший - це функція, яка буде застосована до кожного елементу списку, а другий - це список, який потрібно обробити.

```
 >>> l = [1, 2, 3, 4, 5, 6, 7]
>>> list (map (lambda x: x ** 3, l))
[1, 8, 27, 64, 125, 216  , 343]
```

Лекфчяція 8

8.1 Робота з винятками

Даний урок присвячений винятків і роботі з ними.

Основну увагу приділено поняттю виключення в мовах програмування, обробці виключень в Python, їх генерації і створення призначених для користувача винятків.

Винятками (exceptions) в мовах програмування називають проблеми, що виникають в ході виконання програми, які допускають можливість подальшої її роботи в рамках основного алгоритму. Типовим прикладом винятку є поділ на нуль, неможливість зчитати дані з файлу (пристрою), відсутність доступної пам'яті, доступ до закритої області пам'яті і т.п.

Для обробки таких ситуацій в мовах програмування, як правило, передбачається спеціальний механізм, який називається обробка виключень (exception handling). Винятки поділяють на синхронні і асинхронні. Синхронні виключення можуть виникнути тільки в певних місцях програми. Наприклад, якщо у вас є код, який відкриває файл і зчитує з нього дані, то виключення типу "помилка читання даних" може відбутися тільки в зазначеному шматку коду.

Асинхронні виключення можуть виникнути в будь-який момент роботи програми, вони, як правило, пов'язані з будь-якими апаратними проблемами, або приходом даних. Як приклад можна привести сигнал відключення живлення. У мовах програмування найчастіше передбачається спеціальний механізм обробки виключень. Обробка може бути з поверненням, коли після обробки виключення, виконання програми продовжується з того місця, де воно виникло. І обробка без повернення, в цьому випадку, при виникненні виключення, здійснюється перехід до спеціального, заздалегідь підготовлений, блок коду.

Розрізняють структурну і неструктурну обробку винятків. Неструктурна обробка передбачає реєстрацію функції обробника для кожного виключення, відповідно дана функція буде викликана при виникненні конкретного винятку.

Для структурної обробки мова програмування повинненна підтримувати спеціальні синтаксичні конструкції, які дозволяють виділити код, який необхідно контролювати і код, який потрібно виконати при виникненні виняткової ситуації. В Python виділяють два різних види помилок: синтаксичні помилки і виключення.

8.2 Синтаксичні помилки в Python

Синтаксичні помилки виникають у разі якщо програма написана з порушеннями вимог Python до синтаксису. Визначаються вони в процесі парсинга програми. Нижче представлений приклад з помилковим написанням функції print.

```
>>> for i in range (10):
        prin ( "hello!")
Traceback (most recent call last):
        File "<pyshell # 2>", line 2, in <module> prin ( "hello!")
NameError: name 'prin' is not defined
```

8.3 винятки в Python

Другий вид помилок - це виключення. Вони виникають у разі якщо синтаксично програма коректна, але в процесі виконання виникає помилка (поділ на нуль і т.п.). Більш докладно про поняття виключення написано вище, в розділі "виключення в мовах програмування". Приклад виключення ZeroDivisionError, яке виникає при розподілі на 0.

```
>>> a = 10
>>> b = 0
>>> c = a / b
Traceback (most recent call last):
        File "<pyshell # 5>", line 1, in <module> c = a / b
ZeroDivisionError: division by zero
```

У Python виключення є певним типом даних, через який користувач (програміст) отримує інформацію про помилку. Якщо в коді програми виняток не обробляється, то додаток зупиняється і в консолі друкується докладний опис сталася помилки із зазначенням місця в програмі, де вона сталася і тип цієї помилки.

8.4 Ієрархія винятків в Python

Існує досить велика кількість вбудованих типів виключень в мові Python, всі вони складають певну ієрархію, яка виглядає так, як показано нижче. BaseException

```
+ - SystemExit
+ - KeyboardInterrupt
+ - GeneratorExit
+ - Exception
        + - StopIteration
        + - StopAsyncIteration
        + - ArithmeticError
        |       + - FloatingPointError
        |       + - OverflowError
        |       + - ZeroDivisionError
        + - AssertionError
         + - AttributeError
         + - BufferError
        + - EOFError
         + - ImportError
                + - ModuleNotFoundError
        + - LookupError
        |       + - IndexError
        |       + - KeyError
        + - MemoryError
        + - NameError
         |      + - UnboundLocalError
         + - OSError
        |       + - BlockingIOError
        |       + - ChildProcessError
```

```
|           + - ConnectionError
|            |        + - BrokenPipeError
|            |        + - ConnectionAbortedError
|            |        + - ConnectionRefusedError
|            |  + - ConnectionResetError
|           + - FileExistsError
 |          + - FileNotFoundError
 |          + - InterruptedError
|           + - IsADirectoryError
|           + - NotADirectoryError
|           + - PermissionError
|           + - ProcessLookupError
|           + - TimeoutError
+ - ReferenceError
+ - RuntimeError
|           + - NotImplementedError
|           + - RecursionError
 + - SyntaxError
|           + - IndentationError
|                     + - TabError
+ - SystemError
 + - TypeError
 + - ValueError
 |         + - UnicodeError
|                     + - UnicodeDecodeError
|                     + - UnicodeEncodeError
|                     + - UnicodeTranslateError
+ - Warning
          + - DeprecationWarning
           + - PendingDeprecationWarning
           + - RuntimeWarning
           + - SyntaxWarning
           + - UserWarning
           + - FutureWarning
           + - ImportWarning
           + - UnicodeWarning
           + - BytesWarning
          + - ResourceWarning
```

Як видно з наведеної вище схеми, всі винятки є підкласом виключення BaseException. Більш докладно про ієрархію винятків і їх описі можете прочитати в офіційній документації https://docs.python.org/3/library/exceptions.html.

8.5 Обробка виключень в Python

Обробка винятків потрібна для того, щоб програма не завершувалося аварійно кожен раз, коли виникає виняток. Для цього блок коду, в якому можлива поява виняткової ситуації необхідно помістити всередину синтаксичної конструкції try ... except.

```
print ( "start")
try:
        val = int (input ( "input number:"))
        tmp = 10 / val
        print (tmp)
except Exception as e:
```

```
            print ("Error!" + Str (e))
print ("stop")
```

у наведеній вище програмі можливих два види винятків - це ValueError, що виникає в разі, якщо на запит програми "введіть число", ви введете рядок, і ZeroDivisionError - якщо ви введете в якості числа 0.

При введенні нульового числа буде таким.

```
start
input number: 0
Error!
stop
```

Якби інструкцій try ... except не було, то при викиді будь-якого з виключень програма аварійно завершиться.

```
print ( "start")
val = int (input ( "input number:"))
tmp = 10 / val
print (tmp)
print ( "stop")
```

Якщо ввести 0 на запит наведеної вище програми, відбудеться її зупинка з роздруківкою повідомлення про виключення.

```
start input number: 0
 Traceback (most recent call last):
File "F: / work / programming / python / devpractice / tmp. py", line 3, in <module>
tmp = 10 / val
 ZeroDivisionError: division by zero
```

Зверніть увагу, напис stop вже не друкується в кінці виведення програми. Згідно з документом за мовою Python, що описує помилки і виключення, оператор try працює таким чином:

- Спочатку виконується код, що знаходиться між операторами try і except.
- Якщо в ході його виконання винятку не відбулося, то код в блоці except пропускається, а код в блоці try виконується весь до кінця.
- Якщо виняток відбувається, то виконання в рамках блоку try переривається і виконується код в блоці except. При цьому для оператора except можна вказати, які виключення можна обробляти в ньому. При виникненні виключення, шукається саме той блок except, який може обробити цей виняток.
- Якщо серед except блоків немає відповідного для обробки винятку, то воно передається назовні з блоку try. У разі, якщо обробник виключення так і не буде знайдений, то виключення буде необробленим (unhandled exception) і програма аварійно зупиниться.

Для вказівки набору винятків, який повинен обробляти даний блок except їх необхідно перерахувати в дужках (круглих) через кому після оператора except.

Якби ми в нашій програмі хотіли обробляти тільки ValueError і ZeroDivisionError, то програма виглядала б так.

```
print ( "start")
try:
        val = int (input ( "input number:"))
        tmp = 10 / val
        print (tmp)
 except (ValueError, ZeroDivisionError):
```

```
            print ( "Error!")
        print ("stop")
```

Або так, якщо хочемо обробляти ValueError, ZeroDivisionError по окремість, і, при цьому, зберегти працездатність при виникненні виключень відмінних від перерахованих вище.

```
print ( "start")
try:
        val = int (input ( "input number:"))
        tmp = 10 / val
        print (tmp)
except ValueError:
        print ("ValueError!")
except ZeroDivisionError:
        print ( "ZeroDivisionError!")
except:
        print ( "Error!")
print ( "stop")
```

Існує можливість передати детальну інформацію про подію виключення в код всередині блоку except.

```
print ( "start")
try:
        val = int (input ( "input number:"))
        tmp = 10 / val print (tmp)
except ValueError as ve:
        print ("ValueError! {0}". format (ve))
except ZeroDivisionError as zde:
        print ( "ZeroDivisionError! {0}". format (zde))
except Exception as ex:
        print ( "Error! {0}". format (ex))
print ( "stop")
```

8.6 Використання finally в обробці виключень

Для виконання певного програмного коду при виході з блоку try / except, використовуйте оператор finally.

```
try:
        val = int (input ("input number:"))
        tmp = 10 / val
        print (tmp)
except:
        print ("Exception")
finally:
        print ("Finally code")
```

Не залежно від того, виникне чи ні під час виконання коду в блоці try виняток, код в блоці finally все одно буде виконаний.

Якщо необхідно виконати якийсь програмний код, в разі якщо в процесі виконання блоку try не виникло винятків, то можна використовувати оператор else.

```
try:
        f = open ( "tmp. txt", "r")
```

```
        for line in f:
                print (line)
        f.close ()
except Exception as e:
        print (e)
else:
        print ( "File was readed")
```

8.7 Генерація винятків в Python

Для примусової генерації виключення використовується інструкція raise. Найпростіший приклад роботи з raise може виглядати так.

```
try:
        raise Exception ( "Some exception")
except Exception as e:
        print ( "Exception exception" + str (e))
```

Таким чином, можна "вручну" викликати виключення при необхідності.

8.8 Користувальницькі виключення (User-defined Exceptions) в Python

В Python можна створювати власні виключення. Така практика дозволяє збільшити гнучкість процесу обробки помилок в рамках тієї предметної області, для якої написана ваша програма. Для реалізації власного типу виключення необхідно створити клас, який є спадкоємцем від одного з класів винятків.

```
class NegValException (Exception):
        pass
try:
        val = int (input ( "input positive number:"))
        if val <0:
                raise NegValException ( "Neg val:" + str (val))
         print (val + 10)
except NegValException  as e:
         print (e)
```

Лекція 9 Введення-виведення даних. Робота з файлами

В уроці розглянуті основні способи введення і виведення даних в Python з використанням консолі та робота з файлами: відкриття, закриття, читання і запис.

9.1 Вивід даних в консоль

Один з найпоширеніших способів вивести дані в Python - це надрукувати їх в консолі. Якщо ви перебуваєте на етапі вивчення мови, такий спосіб є основним для того, щоб швидко переглянути результат своєї роботи. Для виведення даних в консоль використовується функція print. Розглянемо основні способи використання даної функції.

```
>>> print ( "Hello")
 Hello >>>
print ( "Hello," + "world!")
 Hello, world!
 >>> print ( "Age:" + str (23))
```

Age: 23

За замовчуванням, для поділу елементів у функції print використовується пропуск.

```
>>> print ( "A", "B", "C")
A B C
```

Для заміни роздільник необхідно використовувати параметр sep функції print.

```
print ( "A", "B", "C", sep = "#")
A # B # C
```

В якості кінцевого елемента рядка, що виводиться, використовується символ перекладу рядка.

```
>>> for i in range (3):
        print ( "i:" + str (i))
        i: 0
        i: 1
       i: 2
```

Для його заміни використовується параметр end.

```
>>> for i in range (3):
        print ( "[i:" + str (i) + "]", end = "--")
[i: 0] -- [i: 1] -- [  i: 2] --
```

9.2 Введення даних з клавіатури

Для зчитування вводяться з клавіатури даних використовується функція input ().

```
>>> input ()
 test
 'test'
```

Для збереження даних у змінній використовується наступний синтаксис. `>>> a = input ()`

```
 hello
>>> print (a)
hello
```

Якщо зчитується з клавіатури ціле число, то рядок, що отримується за допомогою функції input (), можна передати відразу в функцію int ().

```
>>> val = int (input ())
 123
 >>> print (val)
 123
>>> type (val)
<class 'int'>
```

Для виведення рядка-запрошення, використовуйте її в якості аргументу функції input().

```
>>> tv = int (input ( "input number:"))
 input number: 334
 >>> print (tv)
```

Перетворення рядка в список здійснюється за допомогою методу split (), за замовчуванням, як роздільник, використовується пропуск.

```
>>> l = input().split()
1 2 3 4 5 6 7
>>> print (l)
[ '1', '2', '3', '4', '5', '6', '7']
```

Роздільник можна замінити, вказавши його в якості аргументу методу split ().

```
>>> nl = input ().split ( "-")
1-2-3-4-5-6-7
>>> print (nl)
[ '1', '2', '3', '4', '5', '6','7']
```

Для зчитування списку чисел з одночасним приведенням їх до типу int можна скористатися ось такою конструкцією.

```
>>> nums = map(int, input().Split ())
 1 2 3 4 5 6 7
 >>> print (list (nums))
 [1, 2, 3, 4, 5, 6, 7]
```

9.3 Робота з файлами

9.3.1 відкриття та закриття файлу

Для відкриття файлу використовується функція open (), яка повертає файловий об'єкт. Найбільш часто використовуваний вид даної функції виглядає так open (имя_файла, режім_доступа).

Для вказівки режиму доступу використовується наступні символи:

'r' - відкрити файл для читання;
'w' - відкрити файл для запису;
'x' - відкрити файл з метою створення, якщо файл існує, то виклик функції open завершиться з помилкою;
'a' - відкрити файл для запису, при цьому нові дані будуть додані в кінець файлу, без видалення існуючих;
'b' - бінарний режим;
't' - текстовий режим;
'+' - відкриває файл для оновлення.

За замовчуванням файл відкривається на читання в текстовому режимі. У файлового об'єкта є такі атрибути.

file.closed - повертає true якщо файл закритий і false в іншому випадку; file.mode - повертає режим доступу до файлу, при цьому файл повинен бути відкритий;
file.name - ім'я файлу.

```
 >>> f = open ( "test. Txt", "r")
 >>> print ( "file. Closed:" + str (f. Closed))
file.closed: False
 >>> print ( "file. mode:" + f. mode)
file. mode: r
```

```
>>> print ( "file. name:" + f. name)
 file.name: test.txt
```

Для закриття файлу використовується метод close ().

9.3.2 Читання даних з файлу

Читання даних з файлу здійснюється за допомогою методів read (розмір) і readline ().

Метод read (розмір) зчитує з файлу певну кількість символів, передане в якості аргументу. Якщо використовувати цей метод без аргументів, то буде лічений весь файл.

```
 >>> f = open ( "test. Txt", "r")
>>> f.read()
‘1 2 3 4 5 \ nWork with file\n '
>>> f.close ()
```

Як аргумент методу можна передати кількість символом, яке потрібно зичитати.

```
 >>> f = open ( "test. Txt", "r")
>>> f.  read (5)
'1 2 3'
>>> f.close ()
```

Метод readline () дозволяє зчитати рядок з відкритого файлу.

```
 >>> f = open("test. Txt", "r")
 >>> f.  readline()
 ‘ 1 2 3 4 5 \ n '
>>> f.  close ()
```

Порядкове зчитування можна організувати за допомогою оператора for.

```
>>> f = open("test. Txt", "r")
>>> for line in f:
        print (line)
    . . .
1 2 3 4 5
Work with file
>>> f.close()
```

9.3.3 Запис даних в файл

Для запису даних файл використовується метод
write (рядок), при успішній запис він поверне кількість записаних символів.

```
>>> f = open ( "test. Txt", "a")
>>> f.  write ( "Test string")
11
 >>> f.  close ()
```

9.3.4 Додаткові методи для роботи з файлами

Метод tell () повертає поточну позицію "умовного курсора" у файлі. Наприклад, якщо ви зчитали п'ять символів, то "курсор" буде встановлено в позицію 5.

```
>>> f = open ( "test. Txt", "r")
>>> f.  read (5)
'1 2 3'
>>> f.tell ()
 5
>>> f.close ()
```

Метод seek (позиція) виставляє позицію в файлі.

```
 >>> f = open ( "test. Txt", "r")
>>> f.  tell ()
0
> >> f. seek (8)
 8
> >> f.  read (1)
 '5'
 >>> f. tell ()
9>>>
 f.close ()
```

Доброю практикою при роботі з файлами є застосування оператора with. При його використанні немає необхідності закривати файл, при завершенні роботи з ним, ця операція буде виконана автоматично.

```
>>> with open ( "test. Txt", "r") as f:
. . .  for line in f:
. . .          print (line)
. . .
1 2 3 4 5
Work with file
Test string
>>> f.  closed
```

True

Лекція 10. Модулі та пакети

Модулі та пакети значно спрощують роботу програміста. Класи, об'єкти, функції і константи, якими доводиться часто користуватися можна упакувати в модуль, і, в подальшому, завантажувати його в свої програми при необхідності. Пакети дозволяють формувати простори імен для роботи з модулями.

10.1 Модулі в Python

10.1.1 Що таке модуль в Python?

Під модулем в Python розуміється файл з розширенням .ру. Модулі призначені для того, щоб в них зберігати часто використовувані функції, класи, константи і т.п. Можна умовно розділити модулі та програми: програми призначені для безпосереднього запуску, а модулі для імпортування їх в інші програми. Варто зауважити, що модулі можуть бути написані не тільки на мові Python, але і на інших мовах (наприклад C).

10.1.2 Як імпортувати модулі в Python?

Найпростіший спосіб імпортувати модуль в Python це скористатися конструкцією:

import імя_модуля

Імпорт та використання модуля math, який містить математичні функції, буде виглядати ось так.

```
>>> import math
 >>> math.  factorial (5)
 120
```

За один раз можна імпортувати відразу кілька модулів, для цього їх потрібно перерахувати через кому після слова import:

```
import імя_модуля1, імя_модуля2
```

```
>>> import math, datetime
 >>> math.cos (math. pi / 4)
 0. 707106781186547
 >>> datetime. date (2017, 3, 21)
 datetime. date (2017, 3, 21)
```

Якщо ви хочете задати псевдонім для модуля у вашій програмі, можна скористатися ось таким синтаксисом:

```
import імя_модуля as новое_імя
```

```
>>> import math as m
 >>> m.  sin (m. pi / 3)
 0. 866025403784438
```

Використовуючи будь-який з перерахованих вище підходів, при виконанні функції з імпорту модуля, вам завжди доведеться вказувати ім'я модуля (або псевдонім). Для того, щоб цього уникнути робіть імпорт через конструкцію from ... import ...

```
from імя_модуля import імя_об'єкта
```

```
>>> from math import cos
>>> cos (3. 14)
0. 999998731727539
```

При цьому імпортується тільки конкретний об'єкт (у нашому прикладі: функція cos), інші функції недоступні, навіть якщо при їх виклик вказати ім'я модуля.

```
>>> from math import cos
>>> cos (3. 14)
 -0.  999998731727539
 >>> sin (3. 14)
 Traceback (most recent call last):
          File "<pyshell # 2>", line 1, in <module>
         sin (3. 14)
 NameError: name 'sin' is not defined
> >> math.sin (3. 14)
 Traceback (most recent call last):
          File "<pyshell # 3>", line 1, in <module>
          math.  sin (3. 14)
NameError: name 'math' is not defined
```

Для імпортування кількох функцій з модуля, можна перерахувати їх імена через кому.

from імя_модуля import імя_об'єкта1, імя_об'єкта2

```
>>> from math import cos, sin, pi
>>> cos (pi / 3)
0. 500000000000000
>>> sin (pi / 3)
0. 866025403784438
```

Для імпортованого об'єкту можна задати псевдонім.

from імя_модуля import імя_об'єкта as псевдонім_об'єкта

```
>>> from math import factorial as f
>>> f (4)
 24
```

Якщо необхідно імпортувати всі функції, класи і т.п. з модуля, то скористайтеся наступною формою оператора from ... import ...

from імя_модуля import *

```
>>> from math import *
>>> cos (pi / 2)
 6. 123233995736766e-17
>>> sin (pi / 4)
0. 707106781186547
> >> factorial (6)
720
```

10.2 Пакети в Python

10.2.1 Що таке пакет в Python?

Пакет в Python - це каталог, що включає в себе інші каталоги та модулі, але при цьому додатково містить файл __init__.py. Пакети використовуються для формування простору імен, що дозволяє працювати з модулями через вказівку рівня вкладеності (через точку). Для імпортування пакетів використовується той же синтаксис, що і для роботи з модулями.

10.2.2 Використання пакетів в Python

Розглянемо наступну структуру пакету:

```
fincalc
| -- __init__.py
| -- simper.py
| -- compper.py
| -- annuity.py
```

Пакет fincal містить в собі модулі для роботи з простими відсотками (simper.py), складними відсотками (compper.py) і аннуітетами (annuity.py).

Для використання функції з модуля роботи з простими відсотками, можна використовувати один з наступних варіантів:

```
import fincalc.simper
fv = fincalc.simper.fv (pv, i, n)
```

```
import fincalc.simper as sp
fv = sp.fv (pv, i, n)
from fincalc import simper
fv = simper.fv (pv, i, n)
```

Файл __init__.py може бути порожнім або може містити змінну __all__, що зберігає список модулів, який імпортується при завантаженні через конструкцію:

```
from Имя_Пакета import*
```

Наприклад для нашого випадку вміст __init__.py може бути ось таким: __all__ = ["simper", "compper", "annuity"]

Лекція 11. Класи і об'єкти

Дана лекція присвячена об'єктно-орієнтованому програмуванню в Python. Розібрані такі теми як створення об'єктів і класів, робота з конструктором, успадкування і поліморфізм в Python.

11.1 Основні поняття об'єктно-орієнтованого програмування

Об'єктно-орієнтоване програмування (ООП) є методологією розробки програмного забезпечення, в основі якої лежить поняття класу і об'єкту, при цьому сама програма створюється як деяка сукупність об'єктів, які взаємодіють один з одним і з зовнішнім світом. Кожен об'єкт є екземпляром деякого класу. Класи утворюють ієрархії. Виділяють три основних "стовпи" ООП- це інкапсуляція, успадкування і поліморфізм.

Інкапсуляція

Під инкапсуляцией розуміється об'єднання в рамках однієї сутності (класу) певних даних і методів для роботи з ними (і не тільки). Наприклад, можна визначити клас "холодильник", який буде містити наступні дані: виробник, обсяг, кількість камер зберігання, споживана потужність і т.п., і методи: відкрити / закрити холодильник, включити / виключити. При цьому клас стає новим типом даних в рамках розроблюваної програми. Можна створювати змінні цього нового типу, такі змінні називаються об'єкти.

Спадкування

Під спадкуванням розуміється можливість створення нового класу на базі існуючого. Спадкування передбачає наявність відносини "тобто є" між класом спадкоємцем і класом батьком. При цьому клас нащадок буде містити ті ж атрибути і методи, що і базовий клас, але при цьому його можна (і потрібно) розширювати через додавання нових методів і атрибутів. Прикладом базового класу, який демонструє спадкування, можна визначити клас "автомобіль", що має атрибути: маса, потужність двигуна, об'єм паливного бака і методи: завести і заглушити. У такого класу може бути нащадок - "вантажний автомобіль", він буде містити ті ж атрибути і методи, що і клас "автомобіль", і додаткові властивості: кількість осей, потужність компресора і т.п.

Поліморфізм

Поліморфізм дозволяє однаково поводитися з об'єктами, що мають однотипний інтерфейс, незалежно від внутрішньої реалізації об'єкта. Наприклад з об'єктом класу "вантажний автомобіль" можна робити ті ж операції, що і з об'єктом класу "автомобіль", тому що перший є спадкоємцем другого, при цьому зворотне твердження не вірне (у всякому разі не завжди).

Іншими словами поліморфізм передбачає різну реалізацію методів з однаковими іменами. Це дуже корисно при спадкуванні, коли в класі спадкоємця можна перевизначити методи класу батька.

11.2 Класи в Python

11.2.1 Створення класів і об'єктів

Створення класу в Python починається з інструкції class. Ось так буде виглядати мінімальний клас.

```
class C:
        pass
```

Клас складається з оголошення (інструкція class), імені класу (нашому випадку це ім'я C) і тіла класу, яке містить атрибути і методи (в нашому мінімальному класі є тільки одна інструкція pass). Для того щоб створити об'єкт класу необхідно скористатися наступним синтаксисом:

```
імя_об'єкта = ім'я_класу ()
```

Клас може містити атрибути і методи. Нижче представлений клас, що містить атрибути color (колір), width (ширина), height (висота).

```
class Rectangle:
        color = "green"
        width = 100
        height = 100
```

Доступ до атрибуту класу можна отримати наступним чином.

```
імя_об'єкта.атрібут

rect1 = Rectangle ()
 print (rect1. color)
```

Додамо до нашого класу метод. Метод - це функція знаходиться всередині класу, що виконує певну роботу, яка, найчастіше, передбачає доступ до атрибутів створеного об'єкта. Наприклад, нашому класу Rectangle, можна додати метод, який розраховує площу прямокутника. Для того, щоб метод в класі знав, з яким об'єктом він працює (це потрібно для того, щоб отримати доступ до атрибутів: ширина (width) і висота (height)), першим аргументом йому слід передати параметр self, через який він може отримати доступ до своїх даних.

```
class Rectangle:
        color = "green"
        width = 100
        height = 100
        def square (self):
                return self.width*self.height
```

Тоді, наша кінцева програма, що демонструє роботу з атрибутами і методами, буде виглядати так:

```
class Rectangle:
        color = "green"
        width = 100
        height = 100
```

```
        def square (self):
            return self.width*self.height
rect1 = Rectangle ()
print (rect1. color)
print (rect1. square ())
rect2 = Rectangle ()
rect2.width = 200
rect2.  color = "brown"
print (rect2. color)
print (rect2. square ())
```

11.2.2 Конструктор класу

Конструктор класу дозволяє задати певні параметри об'єкта при його створенні. Таким чином з'являється можливість створювати об'єкти з вже заздалегідь заданими атрибутами. Конструктором класу є метод:

```
__init __ (self)
```

Наприклад, для того, щоб мати можливість задати колір, довжину і ширину прямокутника при його створенні, додамо до класу Rectangle наступний конструктор:

```
class Rectangle:
    def __init __ (self, color = "green", width  = 100, height = 100):
        self.color = color
        self.width = width
        self.height = height

    def square (self):
        return self. width * self. height

rect1 = Rectangle ()
print (rect1. color)
print (rect1. square ())
rect1 = Rectangle ( "yellow", 23, 34)
 print (rect1. color)
 print (rect1. square ())
```

11.3 Спадкування

В організації успадкування беруть участь як мінімум два класи: клас батько і клас нащадок. При цьому можливо множинне спадкування, в цьому випадку у класу нащадка є кілька батьків. Не всі мови програмування підтримують множинне спадкування, але в Python можна його використовувати. Синтаксично створення класу з зазначенням його батька виглядає так:

```
class ім'я_класу (імя_родітеля1, [імя_родітеля2, ..., імя_родітеля_n])
```

Доопрацюємо наш приклад так, щоб в ньому було присутнє успадкування.

```
class Figure:
    def __init __ (self, color):
        self.color = color
    def get_color(self):
        return self.color
```

```
class Rectangle (Figure):
        def __init __ (self, color, width = 100, height = 100):
                super ().__init __ (color)
                self. width = width
                self.height = height
        def square (self):
                return self. width * self. height

rect1 = Rectangle ( "blue")
print (rect1. get_color ())
print (rect1. square ())
rect2 = Rectangle ( "red", 25, 70)
print (rect2. get_color ())
print (rect2. square  ())
```

11.4 Поліморфізм

Як вже було сказано у вступі в рамках ООП поліморфізм, як правило, використовується з позиції перевизначення методів базового класу в класі спадкоємця. Найпростіше це розглянути на прикладі. Додамо в наш базовий клас метод info (), який друкує зведену інформацію по об'єкту класу Figure і перевизначити цей метод в класі Rectangle, де додамо додаткові дані і висновок.

```
class Figure:
        def __init __ (self, color):
                self.color = color
        def get_color (self):
                return self. color
        def info (self):
                print ("Figure")
                print ( "Color:" + self. color)

class Rectangle (Figure):
        def __init __ (self, color, width = 100, height = 100):
                super ().__init __(color)
                self.  width = width
                self.  height = height
        def square (self):
                return self.width * self.height
        def info (self):
                print ( "Rectangle")
                print ( "Color:" + self. color)
                print ( "Width:" + str (self. width))
                print ( "Height:" + str (self. height  ))
                print ( "Square:" + str (self. square ()))

fig1 = Figure ( "green")
print (fig1.info ())
rect1 = Rectangle ( "red", 24, 45)
print (rect1.info ())
```

Таким чином спадкоємець класу може розширювати і модифікувати функціонал класу батька.

Лекція 12. Ітератори і генератори

Генератори і ітератори представляють собою інструменти, які, як правило, використовуються для потокової обробки даних. У цьому уроці розглянемо концепцію ітераторів в Python, навчимося створювати свої ітератори і розберемося як працювати з генераторами.

12.1 Ітератори в мові Python

У багатьох сучасних мовах програмування використовують такі сутності як ітератори. Основне їх призначення - це спрощення навігації по елементах об'єкта, який, як правило, являє собою деяку колекцію (список, словник і т.п.). Мова Python, в цьому випадку, не виняток і в ньому теж є підтримка ітераторів. Итератор представляє із себе обєкт пееараховувач, який для даного об'єкта видає наступний елемент, або кидає виняток, якщо елементів більше немає.

Основне місце використання ітераторів - це цикл for. Якщо ви перебирає елементи в деякому списку або символи в рядку за допомогою циклу for, то, фактично, це означає, що при кожній ітерації циклу відбувається звернення до Ітератора, що міститься в рядку / списку, з вимогою видати наступний елемент, якщо елементів в об'єкті більше немає, то итератор генерує виняток, яке обробляється в рамках циклу for непомітно для користувача. Наведемо кілька прикладів, які допоможуть краще зрозуміти цю концепцію. Для початку виведемо елементи довільного списку на екран.

```
>>> num_list = [1, 2, 3, 4, 5]
>>> for i in num_list:
        print (i)
1
2
3
4
5
```

Як вже було сказано, об'єкти, елементи яких можна перебирати в циклі for, містять в собі об'єкт итератор, для того, щоб його отримати необхідно використовувати функцію iter (), а для вилучення наступного елемента з ітератора - функцію next ().

```
>>> itr = iter(num_list)
>>> print (next(itr))
 1
>>> print (next (itr))
 2
>>> print (next (itr))
 3
> >> print (next (itr ))
 4
>>> print (next (itr))
 5
>>> print (next (itr))
Traceback (most recent call last):
        File "<pyshell # 12>", line 1, in <module>
                print (  next (itr))
 StopIteration
```

Як видно з наведеного вище прикладу виклик функції next (itr) кожен раз повертає наступний елемент зі списку, а коли ці елементи закінчуються, генерується виключення StopIteration.

12.2 Створення власних ітераторів

Якщо потрібно обійти елементи всередині об'єкта вашого власного класу, необхідно побудувати свій ітератор. Створимо клас, об'єкт якого буде ітератором, що видає певну кількість одиниць, яке користувач задає при створенні об'єкта.

Такий клас буде містити конструктор, який приймає на вхід кількість одиниць і метод __next __ (), без нього екземпляри даного класу не будуть ітераторами.

```
class SimpleIterator:
        def __init __ (self, limit):
                self.limit = limit
                self.counter = 0
        def __next __ (self):
         if self.counter<self.limit:
                self.counter + = 1
                return 1
         else:
                raise StopIteration

 s_iter1 = SimpleIterator (3)
print (next (s_iter1))
 print (next (s_iter1))
 print (next (s_iter1))
 print (next (s_iter1))
```

В нашому прикладі при четвертому виклику функції next() буде викинуто виключення StopIteration. Якщо ми хочемо, щоб з даним об'єктом можна було працювати в циклі for, то в клас SimpleIterator потрібно додати метод __iter__(), який повертає ітератор, в даному випадку цей метод повинен повертати self.

```
class SimpleIterator:
        def __iter __ (self):
                return self
         def __init __ (self, limit):
                self.  limit = limit
                self.counter = 0
        def __next __ (self):
                if self.counter <self.limit:
                self.counter + = 1
                return 1
        else:
                raise StopIteration

s_iter2 = SimpleIterator (5)
 for i in s_iter2:
         print (i)
```

12.3 Генератори

Генератори дозволяють значно спростити роботу з конструювання ітераторів. У попередніх прикладах, для побудови ітератора і роботи з ним, ми створювали окремий клас. Генератор - це функція, яка будучи викликаною в функції next() повертає наступний об'єкт згідно з алгорит-

мом її роботи. Замість ключового слова return в генераторі використовується yield. Найпростіше роботу генератора подивитися на прикладі.

Напишемо функцію, яка генерує необхідну нам кількість одиниць.

```
def simple_generator (val):
        while val> 0:
            val - = 1
            yield 1

gen_iter = simple_generator (5)
print (next (gen_iter))
print (next (gen_iter))
print (next (gen_iter))
print (next (gen_iter))
print (next (gen_iter))
print (next (gen_iter))
```

Ця функція буде працювати так само, як клас SimpleIterator з попереднього прикладу. Ключовим моментом для розуміння роботи генераторів є те, що при виклику yield функція не припиняє свою роботу, а "заморожується" до чергової ітерації, яку запускає функція next ().

Якщо ви в своєму генераторі, десь використовуєте ключове слово return, то дійшовши до цього місця буде викинуто виключення StopIteration, а якщо після ключового слова return помістити будь-яку інформацію, то вона буде додана до опису StopIteration.

МОДУЛЬ 2 Типові алгоритми обробки економічної інформації
Лекція 13 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-reduce.

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.

Леція 14 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 можезнадобитися 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 їхінтеграції 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 (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).

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.

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:

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.

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.

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 – цілкомпридатна 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.

Лекція 15. Структури даних бібліотеки 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 мас- сива 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', 'з'])
In [9]: obj2 Out [9]: d 4 b 7
a -5 з 3
In [10]: obj2.index
Out [lO]: Index ([d, b, a, з], 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]: obj 2 ['d '] = 6 In [13]: obj2 [['c', 'a', 'd']]
Out [13]: з 3 a -5 d 6

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 зна-чень:

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

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, пр.пап], [7.1, -4.5],
[Np.nan, пр.пап], [Про .75, -1.3]],
: Index = ['a', 'Ь', 'з', 'd'],
: Columnscc ['one', 'two']) Out [199]:
one two
a 1.40 NaN
ь 7.10 -4.5
з NaN NaN d 0.75 -1.3

Метод sum об'єкта DataFrame повертає Series, що містить суми по стовпцям:

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
Ь 2. 60
з NaN d -0.55

Повний список зведених статистик:
argmin, argmax – Обчислює позицію в індексі (цілі числа), при якому досягається мініма-
льне 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, обчислюва- ляють 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.

Лекція 16. 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 іменуцвати теж латиницею, інакше, програме 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]) файлу
В стовпець В запишемо нашу послідовність зі стовпця А вихідного

```
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')

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) відомеа високою продуктивністю при розборі дуже великих файлів. Для lxml є кілька програмних інтерфейсів; спочатку продемонструємпо 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+Options'))
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 = ' Google ' 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' <a href="http://twitter.com/"> web </a> ',
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 2274195858030592O1
text pandas: powerful Python data analysis toolkit Name: 7
```

Лекція 17. 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 якихвикористовується 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).

17.1. 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: 51 243', 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 .

Індивідуальне завдання до модулю 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 нецiле число до двох знаків після коми.

Задача №2. Логічні оператори, булевий тип даних (Python)

Напишіть програму, що визначає:

1) який з двох введених рядків довший,
2) чи введено порожній рядок,
3) чи рядки однакові,
4) яке з двох введених чисел більше,
5) чи буде від’ємною сума введених чисел, за допомогою логічних функцій.

Користувачу подається запит, наприклад:

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)

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:

```
*
* *
* * *

* *   * *
```

Задача №4. Обмін значень змінних.

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)

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)

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)

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 комплексом Pyton в Інтернеті

Тема: команди виймання інформації з сайтів в Інтернеті.

Із сайту Державного управління статистики http://www.ukrstat.gov.ua/ за допомогою команд Pyton:

1. Вибрати статистичну інформацію згідно номеру студента за списком групи за якнайбільший період. Конкретну таблицю в групі даних обирати довільно.
2. Зберегти цю інформацію на диску у файлі.
3. Розрахувати середнє, дисперсію та стандарт для кожної колонки таблиці.
4. Розрахувати кореляційну матрицю зв’язку усіх колонок таблиці.
5. Представити кореляційну матрицю у вигляді графіку з точками.

№ за списком групи
Статистична інформація
1. Ринок праці
2. Освіта

№ за списком групи
Статистична інформація
3. Охорона здоров'я
4. Доходи та умови життя
5. Соціальний захист
6. Населені пункти та житло
7. Правосуддя та злочинність
8. Культура
9. Макроекономічна статистика
10. Національні рахунки
11. Економічна діяльність
12. Діяльність підприємств
13. Внутрішня торгівля
14. Капітальні інвестиції
15. Основні засоби
16. Сільське, лісове та рибне господарство

17. Енергетика
18. Промисловість
19. Будівництво
20. Транспорт
21. Туризм
22. Реєстр статистичних одиниць
23. Навколишнє середовище
24. Державні фінанси, податки та публічний сектор
25. Зовнішньоекономічна діяльність
26. Ціни
27. Наука, технології та інновації
28. Інформаційне суспільство

Задача № 9. Малювання з використанням графіки Turtle

Тема: Використання бібліотеки turtle, що дозволяє малювати на екрані нескладні малюнки.

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 дві стосуються окружності. Намалюйте дві пересічні окружності.

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.

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. Робота з файлами в Pyton

Тема: работа з файлами в Pyton.

Створіть файл з назвою, що складається з Вашого прізвища, на диску в директорії \STUDENT\Назва групи\ …. Помістіть інформацію (ПІБ) про 5х людей. Закрийте файл.

Виведіть вміст файлу на екран та додайте інформацію ще про двох осіб.

УВАГА! Слідкуйте за режимами доступу до файлу, своєчасно закривайте відкриті файли.

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 Pyton?
27. Як вибирати дані зі сторінок в Інтернеті, написаних у кодах XML і HTMLза допомогою команд пакету Pyton?

Під час курсу 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.

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 Introducti- on to R, http://cran.r-project.org/doc/manuals/E-intro.pdf, 2009
26.Златопольський, Д. М. Сборник задач по программированию – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2007.
27.K. Hornik, The R FAQ, http://cran.r-project.org/doc/FAQ/E-FAQ.html, 2009 28.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.
29.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.
30.Лутц Марк. Pyton. Карманный справочник, 5-е изд. : Пер. с англ. – М.: И.Д.Вильямс, 2015. – 320 с.
31.Маккинли Уэс. Python и анализ данных/ Пер. с англ. Слинкин А. А. – М.: ДМК Пресс, 2015. - 482 с.
32.Мусин Дмитрий. Самоучитель Python. Выпуск 0.1 //http://pythonworld.ru 33.Чаплыгин А. Н. Учимся программировать вместес Питоном// https://www.visualstudio.com/ru/vs/support