Лабораторна робота №1
БАЗОВІ ПОНЯТТЯ ПО ПРОГРАМУВАННЮ МОВОЮ PYTHON

0. Про курс

За основу даного курс взято курс Массачусетського технологічного інституту (MIT) «Introduction to Computer Science and Programming». Курс MIT супроводжується відео лекціями, які рекомендовані для перегляду, а також завданнями, які є рекомендованим, як додаткові, для самостійного виконання.

1. Базові поняття

Спершу доведеться розібратися з певними базовими поняттями. Не варто їх заучувати – достатньо їх зрозуміти хоча би на інтуїтивному рівні. Згодом, під час практичної роботи, все встане на свої місця.

1.1. Алгоритми і програми

Поняття алгоритму є одним їх центральних понять всієї комп'ютерної дисципліни. Слово «алгоритм», по суті, є синонімом слів «спосіб» або «рецепт». Можна говорити, в цьому сенсі, про алгоритм знаходження кореня рівняння за його коефіцієнтами, або про алгоритм розкладання натурального числа на прості множники. Якщо в основі алгоритмів лежать прості обчислення, то такі алгоритми називають чисельними. Втім, досить часто розглядаються і не чисельні алгоритми.

Наприклад, в ролі початкових даних і результатів можуть виступати послідовності символів: тексти, формули і т.д. В ролі операцій – не звичні операції складання, множення і подібні до них, а операції зчеплення рядків або операції заміни одних символів на інші за деякою таблицею відповідностей. Прикладом може служити кодування тексту азбукою Морзе. Існують алгоритми побудови складних графічних об'єктів і їх перетворення.

Для того, щоб навчити комп'ютер щось робити, потрібно заздалегідь скласти алгоритм. Якщо Ви не зможете скласти (запропонувати) алгоритм та представити його у вигляді послідовності простих кроків (підтримуваних команд мови програмування), то Ви його потім не зможете реалізувати у вигляді програми. Наприклад, якщо Ви не уявляєте як ефективно та однозначно відсортувати за зростанням нагрудні номери для видачі учасникам деяких спортивних змагань, то навряд чи Ви зможете потім реалізувати це у вигляді програми. Тепер, якщо у Вас є коробки з відсортованими номерами, і учасник змагань просить видати йому нагрудний номер 2431, то Вам також необхідно запропонувати ефективний алгоритм пошуку цього номеру, щоб не довелось перебрати номери всіх учасників для знаходження потрібного.

Алгоритм – це описаний зі всіма подробицями спосіб отримання результатів, що задовольняють поставленим умовам, за початковими даними. **Програма** – це послідовність машинних інструкцій, що описує алгоритм. Зрозуміло, що для того, щоб написати програму, потрібно придумати алгоритм. Причому алгоритм не обов’язково буде єдиним. Різні алгоритми можуть відрізнятись своєю ефективністю (час виконання, кількість необхідної пам’яті та процесорного часу).

Комп'ютерні програми зазвичай складаються на спеціальних мовах програмування.

1.2. Мови програмування і рівні абстракції

Існує кілька підходів до програмування. Спочатку обчислення описувалися на рівні машинних команд в двійковому коді. Логіку подібних програм було важко відслідкувати через те, що програмістові доводилося приділяти увагу таким питанням, як, наприклад, скільки елементів пам'яті необхідно виділити для зберігання того або іншого значення. Для складання двох чисел необхідно було заздалегідь обчислити адреси елементів пам'яті, в яких зберігалися отримані значення, і лише після цього провести операцію складання двійкових чисел. Такий підхід до програмування іноді називають адресним.

Прочитати і розібратися, як працює програма, що написана в двійковому коді, було дуже складно, не говорячи вже про те, щоб знайти і виправити в ній помилку. Тому для спрощення своєї роботи програмісти придумали мнемо-коди або мнемоніки (від греч. Mnemonikos < mnemon – запам'ятати) – буквені позначення машинних двійкових команд, які простіше запам'ятати, ніж послідовності нулів і одиниць. Для спрощення роботи з елементами пам'яті стали використовувати поняття змінної.

Змінна – в програмуванні це буквене позначення області пам'яті, в якій зберігається деяке значення.

Для перекладу мнемокодів у машинні інструкції та імен змінних в адреси комірок пам'яті використовувалася спеціальна програма – транслятор. Мови мнемо-кодів отримали назву асемблерів.

Технології розробки продовжували розвиватися, фахівці шукали нові підходи і незабаром стали викристалізовуватися ідеї, які згодом лягли в основу так званого структурного підходу.

Було відмічено, що всі обчислення зводяться до наступних елементарних дій:

- Введення даних з клавіатури, з файлу або з певного пристрою;
- Виведення даних на екран, у файл, на принтер або інший пристрій;
- Виконання деяких операцій над числами, рядками або іншими об'єктами;
- Вибір гілки виконання програми на основі ухваленого рішення (наприклад, за результатами порівняння двох значень);
- Повторення групи операцій найчастіше із зміною одного або декількох параметрів.

Паралельно почали з'являтися нові транслятори, які перетворювали в машинні команди програми, що написані на мовах, які засновані на цих базових операціях. Такі мови стали називати структурними або **мовами високого рівня**. Програми на високорівневих мовах описують обчислювальний процес на більш високому рівні абстракції (тобто, не у вигляді машинних команд, а за допомогою мов близьких до людської), що дозволяють програмісту абстрагуватися від особливостей машинної реалізації.

Найчастіше програми, що написані на мовах високого рівня працюють повільніше, оскільки транслятори будують не найоптимальніший машинний код за текстом програми.

Але, мови високого рівня мають багато переваг:

- Їх легко читати і розуміти
- Їм притаманна переносимість. Це означає те, що високо рівневі програми можуть виконуватися на різних типах комп'ютерів або під управлінням різних операційних систем, причому з мінімальними змінами або без них, тоді як низькорівневі програми зазвичай пишуться під певний тип комп'ютерів або операційну систему, і для перенесення таких програм на іншу платформу їх доводиться переписувати.

Класичними структурними мовами є C і Pascal. Вони дозволяють описувати обчислювальні процеси на більш високому рівні абстракції, ніж асемблери, але, тим не менш, їм притаманні певні недоліки. Зокрема, програмісту, що пише на цих мовах все ж таки доводиться явно вказувати тип змінних, з якими працюватимуть його програми, для того, щоб транслятор знав, скільки пам'яті виділяти під їх зберігання.

Крім того, в змінних, що призначені для зберігання цілих чисел неможливо зберегти, наприклад, рядок.

Останнім часом стали набувати популярності так звані мови дуже високого рівня. Вони дозволяють абстрагуватися від типів змінних, що відкриває нові можливості. До таких мов відноситься мова Python.

1.3. Формальні і природні мови

Розберемося, чим мови програмування відрізняються від природних мов. Існує два види мов: природні і формальні.

До **природних** відносяться мови, на яких розмовляють люди: українська, російська, англійська та інші. Вони виникли природним шляхом, і є досить складними, хоча часто цього просто не помітно (але ми вчимо їх по 5-6 років). Для того, щоб спростити вивчення іноземних мов, люди придумали правила і словники, в яких до слів однієї мови приводяться відповідності з інших мов.

Формальними називають мови, що вигадані людьми для вирішення специфічних завдань. Наприклад, формальною мовою є набір спеціальних знаків і правил для запису математичних формул. Хіміки так само використовують свою формальну мову для записів хімічної структури речовин. Мови програмування – формальні мови, що призначені для опису алгоритмів.

Формальні мови характерні тим, що мають чіткі синтаксичні правила. Наприклад, 3+3=6 є синтаксично правильним математичним записом, а 3=+6$ – ні. H_2O – синтаксично правильна хімічна формула речовини, а $_2Zz$ – ні.

Коли ви читаєте пропозицію природною мовою або вираз на формальній мові, ви визначаєте його структуру, часто не усвідомлено. Цей процес називається синтаксичним аналізом або синтаксичним розбором. Еквівалентний англомовний термін – parsing.

Наприклад, коли ви читаєте фразу «Мама мила раму», ви по пропусках визначаєте початок та кінець слів і лише після цього знаходите підмет («мама») і присудок («мила»). Розібравши синтаксичну структуру, ви можете зрозуміти її сенс – семантику.

Будь-який транслятор перед тим, як перетворити програму в зрозумілий для комп'ютера вигляд, виконує синтаксичний аналіз. При синтаксичному аналізі транслятор розбирає синтаксичну структуру виразів, і знаходить так звані символи (tokens) – синтаксично неподільні частини. У даному контексті символами можуть бути назви змінних, числа, знаки операцій, ключові слова, позначення хімічних елементів.

Використання символу $ у формулі 3=+6$ не має сенсу, і це є однією з причин, чому воно невірне з погляду математики. Та ж проблема в «хімічній» формулі $_2$Zz: у таблиці Менделєєва немає елементу з позначенням Zz.

Другий тип синтаксичних помилок пов'язаний з неправильною структурою виразів, тобто послідовністю символів. Вираз 3=+6$ має невірну структуру, оскільки відразу після знаку рівності не може слідувати знак додавання. Аналогічно, в молекулярних формулах використовуються нижні індекси, але вони не можуть йти перед позначенням хімічного елементу.

Хоча формальні і природні мови мають багато загального, вони мають ряд важливих відмінностей:

1. Однозначність

У природних мовах безліч ідіом і метафор; часто люди визначають значення фраз залежно від ситуації, в якій вони використовуються. Формальні мови розроблені так, щоб виключити неоднозначність виразів. Це означає, що вираз повинен мати тільки одне значення контексту.

2. Надмірність

Для того, щоб позбавитися від неоднозначності і уникнути нерозумінь у природних мовах використовується надмірність: визначення і доповнення.

Формальні мови короткі і максимально виразні. Більшість людей, звиклих до природних мов, зазвичай насилу звикають до формальних мов (і програмам зокрема). Тому варто пам'ятати, що щільність сенсу у таких мовах є більшою, тому вони повільніше читаються і розуміються.

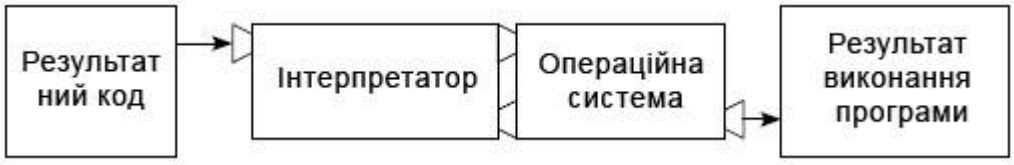
У формальних мовах дуже важливою є структура, тому читання справа наліво або від низу до верху – не кращий спосіб їх зрозуміти. І, нарешті, дрібниці є важливими. Маленькі орфографічні помилки чи невірна пунктуація, які в природних мовах можуть бути проігноровані, у формальних мовах матимуть велике значення, аж до зміни сенсу на протилежне.

В математиці є цілий розділ, присвячений теорії формальних мов. Саме на математичному апараті цієї теорії засновані синтаксичні аналізатори трансляторів.

1.4. Інтерпретатори і компілятори

Існує два типи трансляторів, що перетворюють початковий код програм у машинні команди: інтерпретатори і компілятори.

Інтерпретатор читає високорівневу програму (або початковий код) і, безпосередньо у взаємодії з операційній системою, її виконує. Перетворення і виконання програми виконується по рядках.



На відміну від інтерпретаторів, **компілятори** повністю перетворюють результатний код програми в машинний код (або так званий об'єктний код), який операційна система може виконати самостійно. Це дозволяє виконувати скомпільовані програми навіть на тих комп'ютерах, на яких немає компілятора.

Крім того, такі програми виконуються швидше за рахунок того, що комп'ютеру не доводиться кожного разу перед запуском програми виконувати її розбір і перетворення у зрозумілий для себе вигляд.

Втім, сучасні інтерпретатори теж здатні зберігати проміжний код, на виконання якого витрачається менше часу за рахунок економії на синтаксичному розборі початкового коду. Проте, такий проміжний код є зрозумілим тільки інтерпретатору, тому для запуску програми його наявність на комп'ютері все рівно потрібна.

При сучасних потужностях комп'ютерів і об'ємах пам'яті різниця у швидкості виконання програми інтерпретаторами і компіляторами вже майже непомітна але процес розробки і відлагодження програм на інтерпретаційних мовах є набагато простішим.

Мова Python є інтерпретованою, оскільки написані на ній програми виконує інтерпретатор.

2. Перша програма

Настав час запустити інтерпретатор Python і написати першу програму.

Існує два способи використання інтерпретатора:

- Командний режим
- Режим виконання програм з файлів

Для запуску інтерпретатора Python у командному рядку необхідно набрати команду *python*

```
$ python
```

При запуску інтерпретатора ми бачимо інформацію про його версію, додаткову інформацію і запрошення >>> вводити команди та оператори мови Python.

Після цього в командному рядку інтерпретатора Python ви можете набирати команди, і інтерпретатор одразу буде виводити результати їх виконання:

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

Ми викликали функцію *print("Hello world!")*, тобто дали вказівку вивести на екран рядок Hello world!, і в наступному рядку інтерпретатор вивів те, що ми просили.

Ми також можемо записати програму у текстовий файл і використовувати інтерпретатор для того щоб її виконати. Такий файл називають сценарієм або **скриптом** (від англ. script – сценарій). Наприклад, використовуючи довільний текстовий редактор, створимо файл *prog1.py* з наступним змістом: print("Hello world!")

Назви файлів, що містять програми на Python, прийнято завершувати розширенням *.py* (англійськими літерами). Для того, щоб виконати програму, ми маємо передати інтерпретатору як параметр назву скрипта:

```
$ python prog1.py
Hello world!
```

У інших програмних середовищах метод запуску програм може відрізнятися, але принцип виклику інтерпретатора залишиться таким же.

- ***Вправа.*** Спробуйте змінити свою першу програму так, щоб в ній з'явилися синтаксичні помилки: спочатку помилка, пов'язана з нерозпізнаною синтаксичною одиницею (тобто незрозумілим словом), а потім – з неправильною структурою програми (можна спробувати поміняти місцями синтаксичні одиниці).

2.1. Використання інтерпретатора Python, як калькулятора

Спробуємо використати Python, як калькулятор.

```
>>> 2+5
7
>>> 3*6
18
```

Після натиснення Enter виконуються дії, інтерпретатор видає результат і чекає на введення наступного оператора. Операція множення виконана вірно з додержанням пріоритету виконання арифметичних дій.

Можемо спробувати виконати інші операції множення і ділення.

```
>>> 3/3
1
>>> 1/3
0
```

В другому випадку отримали нуль, бо ділення в цьому випадку є цілочисленним. Якщо ввести вираз без змісту то інтерпретатор видає повідомлення про помилку з вказуванням місця помилки і її типу.

```
>>> 1+
SyntaxError: invalid syntax
```

2.2. Представлення тексту

Мова Python названа на честь «Монті Пайтона» (Monty Python) - комік-групи з Великобританії, колективне ім'я авторів комедійного шоу «Літаючий Цирк Монті Пайтона», британського телевізійного комедійного скетч-шоу.

Спробуємо працювати з текстом, його можна безпосередньо вводити в інтерпретатор.

```
>>> Hello World
SyntaxError: invalid syntax
```

Отримали помилку. Текст або частини тексту в програмах на `Python` представляються за допомогою *стрічок* (*string*) і повинен відділятися від решти програми лапками (одинарними(1), подвійними(2) або потрійними).

```
>>> 'Monty Python'                      (1)
'Monty Python'

>>> "Monty Python's Flying Circus"     (2)
"Monty Python's Flying Circus"

>>> 'Monty Python\'s Flying Circus'    (3)
"Monty Python's Flying Circus"

>>> 'Monty Python's Flying Circus'      (4)
  File "<stdin>", line 1
    'Monty Python's Flying Circus'
                   ^
SyntaxError: invalid syntax
```

Якщо стрічка містить одинарні лапки необхідно використовувати лівий слеш перед апострофом (3) для того, щоб символ апострофа не розглядався як символ завершення стрічки або використовувати подвійні лапки (2). Якщо цього не зробити, то отримаємо помилку (4).

```
>>> "Hello World"
'Hello World'
>>> 'Hello World'
'Hello World'
```

Деколи стрічки можуть складатися з кількох рядків. Python забезпечує декілька способів роботи з ними. В наступному прикладі послідовність з двох стрічок об'єднується в одну. Потрібно використовувати лівий слеш (1) або круглі дужки (2) для того щоб інтерпретатор знав що ввід стрічки ще не завершився після введення першого рядка.

```
>>> "Shall I compare thee to a Summer's day?"\

          "Thou are more lovely and more temperate:"
Shall I compare thee to a Summer's day?Thou are more lovely and more
temperate:
>>>
>>> ("Rough winds do shake the darling buds of May,"

           "And Summer's lease hath all too short a date:")
Rough winds do shake the darling buds of May,And Summer's lease hath
all too short a date:
```

Застосування потрійних лапок дозволяє представляти стрічку так як вона введена з розділенням окремих рядків:

```
>>> """Shall I compare thee to a Summer's day?
Thou are more lovely and more temperate:"""

Shall I compare thee to a Summer's day?
Thou are more lovely and more temperate:

>>> '''Rough winds do shake the darling buds of May,
And Summer's lease hath all too short a date:'''

Rough winds do shake the darling buds of May,
And Summer's lease hath all too short a date:
```

Стрічку також можна вивести за допомогою функції print. Зверніть увагу, що застосовано потрійні лапки і текст виводиться в тому ж самому форматуванні.

```
>>> print("""
Usage: thingy [OPTIONS]
     -h                        Display this usage message
```

```
     -H hostname               Hostname to connect to
""")


Usage: thingy [OPTIONS]
     -h                        Display this usage message
     -H hostname               Hostname to connect to
```

Спробуємо використати оператори додавання і множення для роботи з стрічкою.

```
>>> 'Hello'+ 'World'
'HelloWorld'
```

Оператор додавання виконує операцію поєднання. Він дозволяє створити нову стрічку на основі двох існуючих, але він не додає пробіл між словами. Спробуємо поєднати три однакові стрічки за допомогою операторів множення та додавання.

```
>>> 'Hi'+'Hi'+'Hi'
'HiHiHi'
>>> 'Hi'*3
'HiHiHi'
```

3. Завдання та порядок виконання роботи

1. Ознайомитися з наведеними вище теоретичними відомостями.
2. Виконати приклади, які приводяться в теоретичних відомостях.
3. Виконати наступні завдання:

   3.1. Використовуючи інтерпретаторі `Python` вивести:

   - у першому рядку назву курсу та номер лабораторної роботи
   - у другому - ваше ім’я та прізвище, а також номер Вашої заліковки.

     Приклад результату:

     ‘Introduction to programming’:Task 1

     Andrii Rodionov, FI-9119

   - у третьому – сорок п’ять разів через кому з пробілом ім’я викладача цих практичних занять (подумайте, як зробити так, щоб останнє ім’я не закінчувалось комою)

   3.2. Використовуючи інтерпретаторі Python обчислити наступні вирази згідно варіанта індивідуального завдання:

1) $\frac{11+2*3+4.1}{12.4-221.3}+4.8;$ 2) $\frac{(123-23.1)*(2^3+2.2)}{127.24-21.1}-3.2;$ 3) $\frac{11.2}{11.2-211.3}+\frac{1.2}{15.2+11.3};$

4) $123.8+\frac{11-21.1/2}{87-32.2};$ 5) $\frac{113.2}{21.2+11.3}+\frac{22.2}{15.2-11.3};$ 6) $7.8-\frac{1+2.1/3}{78+21.3};$

7) $\frac{15+11.1/3}{33+12.3}-123.1;$ 8) $\frac{(12^2+17.1)*(73-254.2)}{17.4-211.3}-4.2;$ 9) $\frac{11.2}{11.2-211.3}-\frac{15.2+11.3}{123.1};$

10) $\frac{15-13.1/3}{33^2+19.3}+3.3;$ 11) $\frac{195+1.3/8}{18.2-3.8^2+19.3};$ 12) $\frac{176-9.3/7}{18.2+179.3}-323.8^2;$

4. Записати усі команди що ви вводили до інтерпретатора у текстовий файл та зберегти у вигляді скрипта (програми) на мові Python. Назва файлу має бути наступною <*Surname*>*_Task1.py* (наприклад *Rodionov_Task1.py*)

5. Запустити даний скрипт за допомогою інтерпретатора та переконатись у його працездатності та правильності результатів. Для того, щоб у режимі виконання програм з файлів, результати виконання команд виводилась на екран, перед ними необхідно писати функцію *print*. Наприклад:

   print((2.2-3)/(12.1+4))

6. Спробувати здати та захистити лабораторну роботу у викладача практичних занять

Література

1. A Byte of Python (Russian) (http://wombat.org.ua/AByteOfPython/#id14)
2. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с..
3. Computational Physics with Python by M. Newman (http://www-personal.umich.edu/~mejn/computational-physics/)

Інтернет посилання

- http://ru.wikipedia.org/wiki/Python
- MIT: Introduction to Computer Science and Programming
- http://python.org