

ОСНОВНІ ТЕОРЕТИЧНІ ВІДОМОСТІ

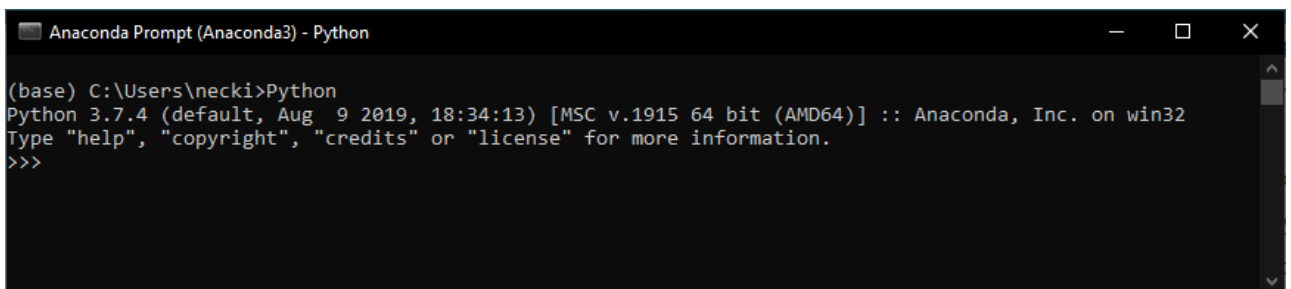
Початок роботи з Python

Встановлення

Python можна скачати з python.org. Однак якщо він ще не встановлений, то замість нього рекомендується встановити дистрибутивний пакет Anaconda, який вже включає в себе більшість бібліотек, необхідних для роботи в області науки про дані.

Консоль Python

Щоб почати працювати з Python, потрібно відкрити командний рядок на комп'ютері. Щоб відкрити консоль Python, необхідно ввести `python`, для Windows, або `python3` для Mac OS / Linux, і натиснути `enter`. Якщо ви встановили Anaconda, треба відкрити вікно Anaconda Prompt, та ввести Python у консолі що з'явилася.

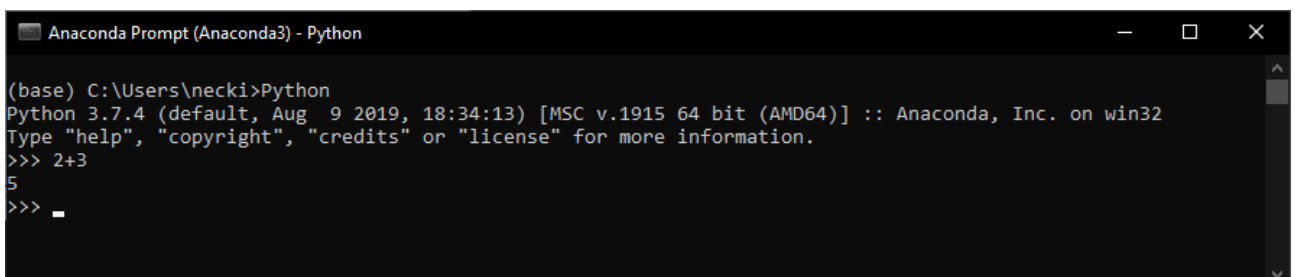


```
(base) C:\Users\necki>Python
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Після запуску Python командний рядок змінилася на `>>>`. Це означає, що зараз ми можемо використовувати тільки команди на мові Python. Вводити `>>>` - не потрібно, Python буде робити це самостійно. Для виходу з консолі Python, в будь-який момент - необхідно ввести `exit ()` або використовувати поєднання клавіш `Ctrl + Z` для Windows і `Ctrl + D` для Mac / Linux.

Калькулятор в консолі.

Спробуємо набрати простий математичний вираз, $2 + 3$, і натиснути `enter`.



```
(base) C:\Users\necki>Python
Python 3.7.4 (default, Aug 9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> _
```

Як можна переконатися Python знає математику! Також можливе виконання і інших арифметичних дій, як:

$4 * 5$; $5 - 1$; $40/2$;

Щоб обчислити ступінь числа, наприклад, 2 у кубі, ми вводим: $2^{**}3$.

```
Anaconda Prompt (Anaconda3) - Python
>>> 4*5
20
>>> 5-1
4
>>> 40/2
20.0
>>> 2**3
8
>>> _
```

Рядки

Для прикладу роботи з рядками спробуємо ввести своє ім'я. Рядки необхідно вводити в лапках:

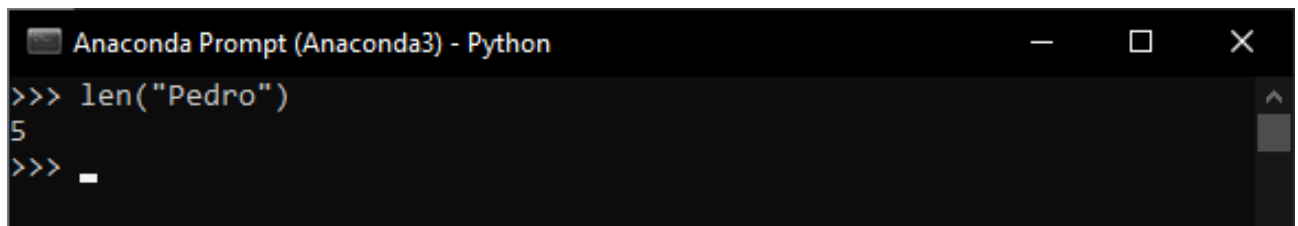
```
Anaconda Prompt (Anaconda3) - Python
>>> "Pedro"
'Pedro'
>>> _
```

Рядок повинен завжди починатися і закінчуватися однаковими символами. Їми можуть бути одинарні (') або подвійні (") лапки.

Над рядками також можуть проводитися арифметичні операції:

```
Anaconda Prompt (Anaconda3) - Python
>>> "Pedro"
'Pedro'
>>> "pedro" + ' Alexandro' + " Maria"
'pedro Alexandro Maria'
>>> "Pedro"*4
'PedroPedroPedroPedro'
>>> _
```

Для обчислення кількості символів у рядку використовується метод `len()`;



```
Anaconda Prompt (Anaconda3) - Python
>>> len("Pedro")
5
>>> _
```

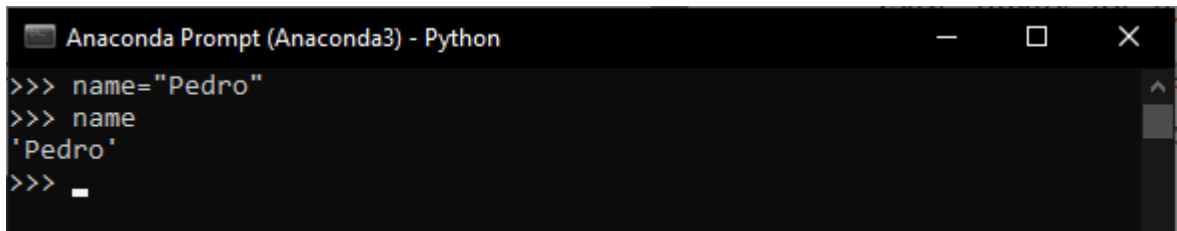
Змінні і типи даних

Змінні

Змінна зберігає певні дані. Назва змінної в Python має починатися з алфавітного символу або зі знака підкреслення і може містити алфавітно-цифрові символи і знак підкреслення. Крім того, назва змінної не повинна збігатися з назвою ключових слів мови Python. Ключових слів не так багато, їх легко запам'ятати: and, as, assert, break, class, continue, def, del, elif, else, except, False, finally, for, from, global, if, import, in, is, lambda, None, nonlocal, not, or, pass, raise, return, True, try, while, with, yield.

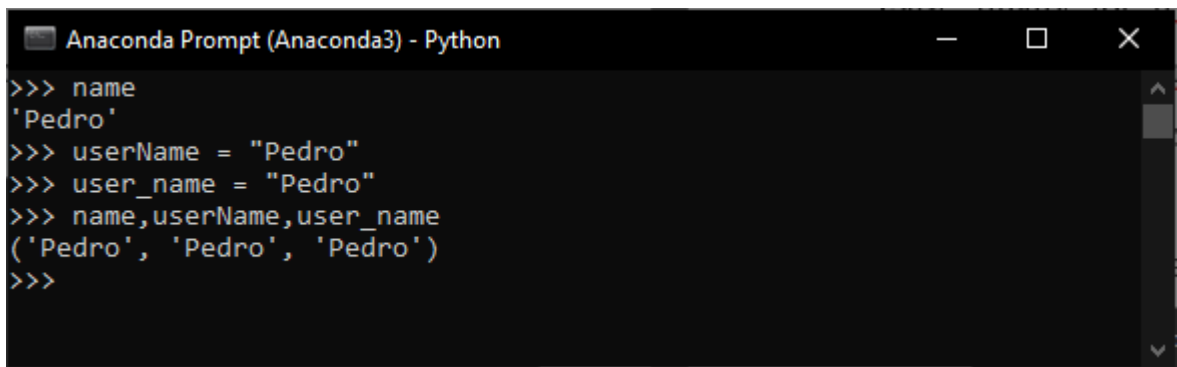
Наприклад, створимо змінну:

Тут визначена змінна name, яка зберігає рядок "Pedro".



```
Anaconda Prompt (Anaconda3) - Python
>>> name="Pedro"
>>> name
'Pedro'
>>> _
```

У Пайтон застосовується два типи найменування змінних: camel case і underscore notation. Camel case має на увазі, що кожне нове підсловом в найменуванні змінної починається з великої літери. Underscore notation має на увазі, що підслова в найменуванні змінної поділяються знаком підкреслення. наприклад:

A screenshot of an Anaconda Prompt window titled "Anaconda Prompt (Anaconda3) - Python". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The terminal shows the following Python code and its output:

```
>>> name
'Pedro'
>>> userName = "Pedro"
>>> user_name = "Pedro"
>>> name, userName, user_name
('Pedro', 'Pedro', 'Pedro')
>>>
```

І також треба враховувати що назви змінних чутливі до регістру, тому змінні `name` і `Name` представлятимуть різні об'єкти.

Типи даних

Змінна зберігає дані одного з типів даних. В Python існує безліч різних типів даних, які поділяються на категорії: числа, послідовності, словники, набори:

boolean - логічне значення `True` або `False`.

int - представляє ціле число, наприклад, 1, 4, 8, 50.

float - представляє число з плаваючою точкою, наприклад, 1.2 або 34.76

complex - комплексні числа

str - рядки, наприклад "hello". В Python 3.x рядки представляють набір символів в кодуванні Unicode

bytes - послідовність чисел в діапазоні 0-255

byte array - масив байтів, аналогічний `bytes` з тим відмінністю, що може змінюватися

list - список

tuple - кортеж

set - неупорядкована колекція унікальних об'єктів

frozen set - те ж саме, що і `set`, тільки не може змінюватися (`immutable`)

dict - словник, де кожен елемент має ключ і значення

Python є мовою з динамічною типізацією. Він визначає тип даних змінної виходячи із значення, яке їй присвоєно. Так, при присвоєнні рядки в подвійних або одинарних лапках змінна має тип `str`. При присвоєнні цілого числа Python автоматично визначає тип змінної як `int`. Щоб визначити змінну як об'єкт `float`, їй присвоюється дробове число, в якому роздільником цілої і дробової частини є точка. Число з плаваючою крапкою можна визначати в експоненційній запису:

```
Выбрать Anaconda Prompt (Anaconda3) - Python
>>> x = 3.9e3
>>> print(x)
3900.0
>>> x = 3.9e-3
>>> print(x)
0.0039
>>>
```

Число float може мати тільки 18 значущих символів. Так, в даному випадку використовуються тільки два символи - 3.9. І якщо число занадто велике або занадто мало, то ми можемо записувати число у подібній нотації, використовуючи експоненту. Число після експоненти вказує ступінь числа 10, на яке треба помножити основне число - 3.9.

При цьому в процесі роботи програми ми можемо змінити тип змінної, присвоївши їй значення іншого типу:

```
Anaconda Prompt (Anaconda3) - Python
>>> user_id = "Pedro007" #type str
>>> print(user_id)
Pedro007
>>> user_id = 123 #type int
>>> print(user_id)
123
>>>
```

За допомогою функції `type ()` можна динамічно дізнатися поточний тип змінної:

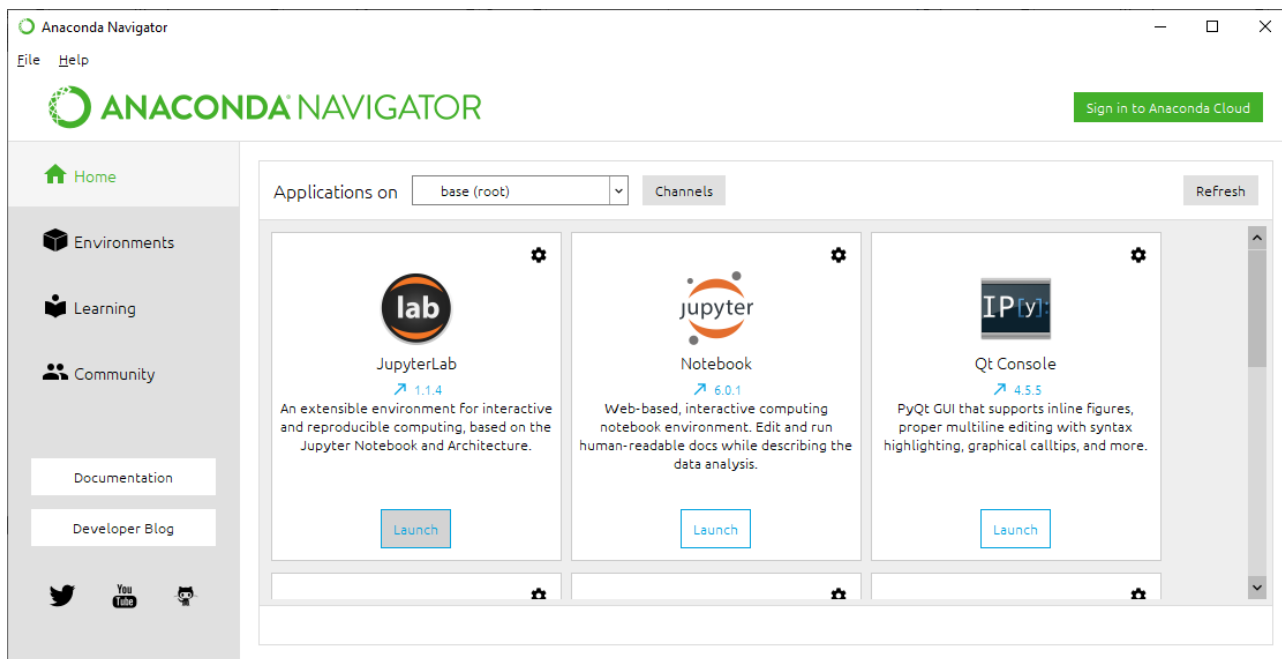
```
Anaconda Prompt (Anaconda3) - Python
>>> user_id = 123 #type int
>>> print(user_id)
123
>>> print(type(user_id))
<class 'int'>
>>>
```

Jupyter lab

JupyterLab - це інтерактивне середовище розробки для роботи з блокнотами, кодом і даними. Проект Jupyter існує для розробки програмного забезпечення з

відкритим вихідним кодом, відкритих стандартів і сервісів для інтерактивних та відтворюваних обчислень.

Використовувати JupyterLab ми будемо для виконання лабораторних, а саме написання, збереження компіляції і запуску коду. Відкрити JupyterLab можна з пакету Anaconda.



Більш детально ознайомитися з JupyterLab можна на сайті <https://proglab.io/p/jupyter>

Списки

Однією з найважливіших структур даних в Python є список. Це просто впорядкована сукупність (або колекція), схожа на масив в інших мовах програмування, але з додатковими функціональними можливостями.

```
integer_list = [1, 2, 3] # список цілих чисел
heterogeneous_list = ["строка", 0.1, True] # різnorідний список
list_of_lists = [integer_list, heterogeneous_list, []] # список списків

list_length = len(integer_list) # довжина списку = 3
list_sum = sum(integer_list) # сума значень у списку = 6
```

Встановлювати значення і отримувати доступ до n-го елемента списку можна за допомогою квадратних дужок:

```

buf = list(range (10)) # створює список {0, 1 , . . . , 9}
zero = buf [0] # = 0 , списки нуль-індексні, тобто индекс 1-го елементу = 0
one = buf [1] # = 1
nine = buf [-1] # = 9, отримати останній елемент
eight = buf [-2] # = 8, отримати передостанній елемент
buf [0] = -1 # тепер x = { - 1 , 1 , 2, 3, . . . , 9}
print(buf)

```

[-1, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Крім цього, квадратні дужки застосовуються для «нарізки» списків:

```

first_three = buf[:3] # перші три = [-1 , 1, 2]
three_to_end = buf[3:] #з третього до кінця = {3, 4, ... , 9}
one_to_four = buf[1:5] # з першого по четвертий = {1 , 2, 3, 4}
last_three = buf[-3:] # останні три = { 7, 8, 9}
without_first_and_last = buf[1:-1] # без першого та останнього = {1 , 2, ... , 8}
copy_of_x = buf[:] # копія списку x= [ -1, 1, 2, ... , 9]
print(first_three,three_to_end,one_to_four,last_three,without_first_and_last,copy_of_x)

```

[-1, 1, 2] [3, 4, 5, 6, 7, 8, 9] [1, 2, 3, 4] [7, 8, 9] [1, 2, 3, 4, 5, 6, 7, 8] [-1, 1, 2, 3, 4, 5, 6, 7, 8, 9]

В Python є оператор `in`, який перевіряє належність елемента списку:

```

: 1 in [1, 2, 3] #True
0 in [1, 2, 3] #False

```

Перевірка полягає в почерговому перегляді всіх елементів, тому користуватися ним стоїть тільки тоді, коли точно відомо, що список невеликий або неважливо, скільки часу піде на перевірку.

Списки легко зчіплювати один з одним:

```

In [ ]: x = [1, 2, 3]
        x. extend ( [ 4, 5, 6] ) # тепер x = {1, 2, 3, 4, 5, 6}

```

Якщо потрібно залишити список `x` без змін, то можна скористатися складанням списків:

```

buf = [1, 2, 3]
z = buf + [4, 5, 6] #z= (1, 2, 3, 4, 5, 6) ; buf не змінився
print(buf,z)

```

[1, 2, 3] [1, 2, 3, 4, 5, 6]

Структура програми Python

Пробільні символи

У багатьох мовах програмування для розмежування блоків коду використовуються фігурні дужки. В Python використовуються відступи:

```
# приклад відступів у вкладених циклах for
for i in [ 1, 2, 3, 4, 5 ] :
    print (i) # перший рядок у блоці for i
    for j in [1, 2, 3, 4, 5 ] :|
        print ( j ) # перший рядок у блоці for j
        print (i + j) # останній рядок у блоці for j
    print (i) # перший рядок у блоці for i
print ( "цикли закінчились ")
```

Це робить код легким для читання, але в той же час змушує стежити за форматуванням. Пропуск всередині круглих і квадратних дужок ігнорується, що полегшує написання багатослівних виразів:

```
# приклад багатослівного виразу
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 +
11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20)
print(long_winded_computation)
```

210

і дозволяє легко читати код:

```
# список списків
list_of_lists = [ [ 1 , 2, 3 ] , [4, 5, 6 ] , [ 7 , 8, 9 ] ]
# такий список списків легше читається
easy_to_read_list_of_lists = [[1, 2, 3 ] ,
                               [4, 5, 6 ] ,
                               [7, 8, 9 ] ]
print(easy_to_read_list_of_lists)
```

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

Для продовження оператора на наступному рядку використовується зворотна коса риса, втім, такий запис буде застосовуватися рідко:


```
two_plus_three = 2 + \
3
print(two_plus_three);|
```

5

Модулі (Імпорт бібліотек)

Деякі бібліотеки середовища програмування на основі Python не завантажуються за замовчуванням. Для того щоб ці інструменти можна було використовувати, необхідно імпортувати модулі, які їх містять.

Один з підходів полягає в тому, щоб просто імпортувати сам модуль:

```
import math
sn=math.sin; cs=math.cos; p=math.pi
uno=sn(0.3*p)**2 + cs(0.3*p)**2;
print(uno)
```

1.0

Тут math - це назва модуля, що містить функції і константи для 'роботи з регулярними виразами. Імпортувавши таким способом весь модуль, можна звертатися до функцій, випереджаючи їх префіксом math.

Функції модуля math

acos(x)	cosh(x)	ldexp(x,y)	sqrt(x)
asin(x)	exp(x)	log(x)	tan(x)
atan(x)	fabs(x)	sinh(x)	frexp(x)
atan2(x,y)	floor(x)	pow(x,y)	modf(x)
ceil(x)	fmod(x,y)	sin(x)	
cos(x)	log10(x)	tanh(x)	

Якщо в коді змінна з ім'ям math вже є, то можна скористатися псевдонімом модуля:

```
import math as math_lib
sn=math_lib.sin; cs=math_lib.cos; p=math_lib.pi
uno=sn(0.5*p)**2 + cs(0.5*p)**2;
print(uno)
```

1.0

Ім'я користувача використовують також в тих випадках, коли імпортований модуль має громіздке ім'я або коли в коді відбувається часте звертання до модуля.

Якщо з модуля потрібно отримати кілька конкретних значень, то їх можна імпортувати в явному вигляді і використовувати без обмежень:

```
from collections import defaultdict, Counter
lookup = defaultdict(int)
my_counter = Counter()
```

Керуючі конструкції

Оператори розгалуження (if, else, elif)

Як і в більшості інших мов програмування, дії можна виконувати за умовою, застосовуючи оператори розгалуження, такі як:

if – якщо + умова,

elif – інакше якщо + умова,

else – інакше.

```
x1=1
y1=2
y2=3
if x1 > y1:
    message= "якщо 1 було б більше 2"
elif x1 > y2:
    message ="коли попередня умова була невиконана, а нова умова виконалася"
else:
    message = " коли всі попередні умови не виконуються, використовується else "
print(message)
```

коли всі попередні умови не виконуються, використовується else

Цикли (while, for)

В Python є цикл while, який працює як і в інших мовах програмування.

while <умова>: - тіло циклу

Тобто всі операції записані в тілі циклу будуть повторно виконуватися до тих пір, доки умова циклу не перестане бути істинною.

```

t = 0
while t < 5:
    print (t, "не більше 5")
    t += 1 #збільшуємо x
print("кінець виконання циклу")

```

```

0 не більше 5
1 не більше 5
2 не більше 5
3 не більше 5
4 не більше 5
кінець виконання циклу

```

Однак частіше буде використовуватися цикл for спільно з оператором in:

for <змінна> **in** <діапазон>: блок

Блок коду після заголовка виконується Доті, поки змінна належить діапазону (причому цим діапазоном може бути список, числова послідовність, рядок, інша послідовність якихось проіндексованих значень):

```

for c in range (10) : # доки c належить діапазону 0-10
    print (c, "менше 10" )
print("кінець циклу")

```

```

0 менше 10
1 менше 10
2 менше 10
3 менше 10
4 менше 10
5 менше 10
6 менше 10
7 менше 10
8 менше 10
9 менше 10
кінець циклу

```

Якщо потрібно більш складна логіка управління циклом, то можна скористатися операторами **continue** та **break**:

```

for f in range (10) :
    if f == 3:
        continue # перейти відразу до наступної ітерації
    if f == 5:
        break
    print (f)
print ("Кінець циклу")

```

```

0
1
2
4
Кінець циклу

```

Істинність

Булеві змінні в Python працюють так само, як і в більшості інших мов програмування лише з одним винятком - вони пишуться з великої літери:

```
one_is_less_than_two = 1 < 2 #True  
true_equals_false = True == False #False|
```

В Python може використовуватися будь-яке значення там, де очікується логічний тип Boolean. Всі наступні елементи мають логічне значення False:

- False; .
- None;
- set() (множина);
- [] (пустий список);
- {} (пустий словник);