

# ПРАКТИЧНА РОБОТА №1. ОЗНАЙОМЛЕННЯ З КЕРУЮЧИМИ КОНСТРУКЦІЯМИ

**Мета роботи** – освоєння основ роботи із простими керуючими конструкціями. Вивчення понять «цикл та розгалуження», робота зі списками у Python.

## Основні теоретичні відомості

### Встановлення Python

Python можна скачати з [python.org](https://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.

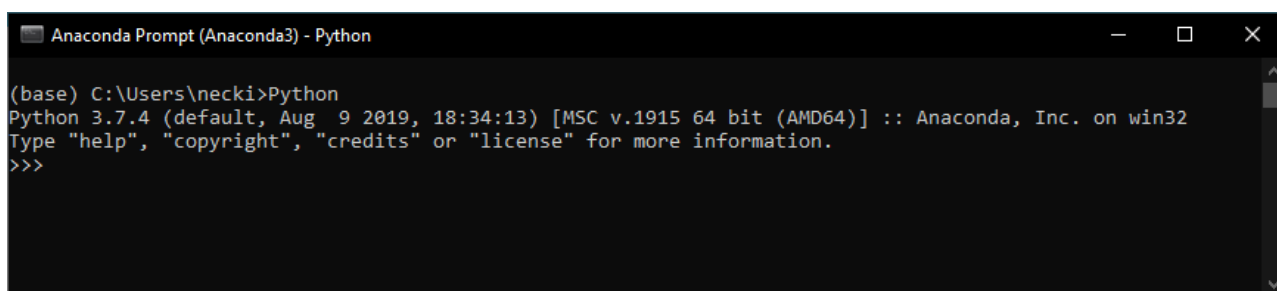


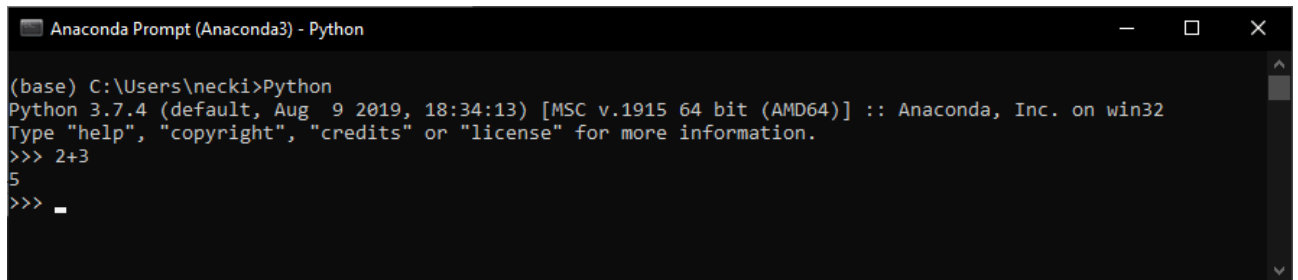
Рисунок 1.1 – Демонстрація вікна Anaconda Prompt

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`.

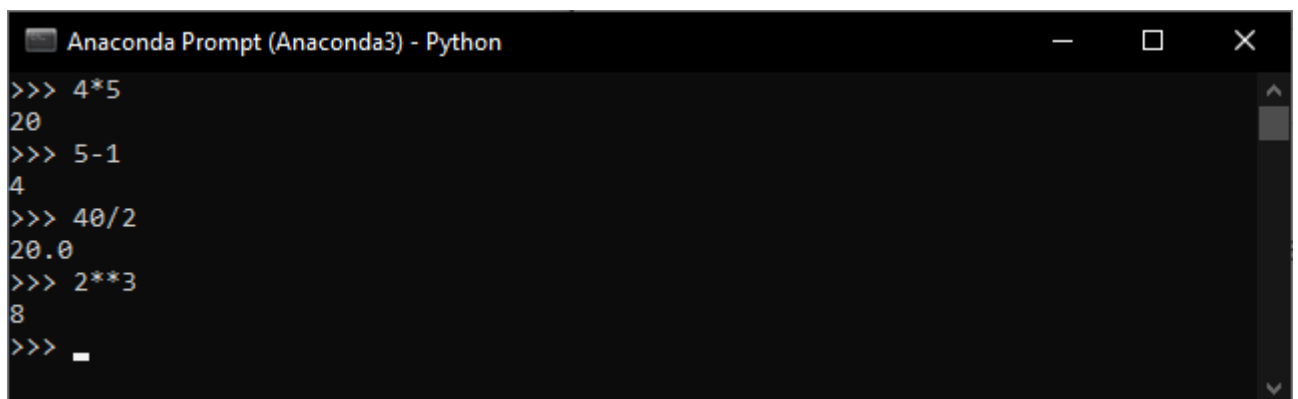


```
Anaconda Prompt (Anaconda3) - Python
(base) C:\Users\neeki>Python
Python 3.7.4 (default, Aug  9 2019, 18:34:13) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+3
5
>>> _
```

Рисунок 1.2 – Результат виконання математичної операції

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 вводимо:  $2^{**}3$ .

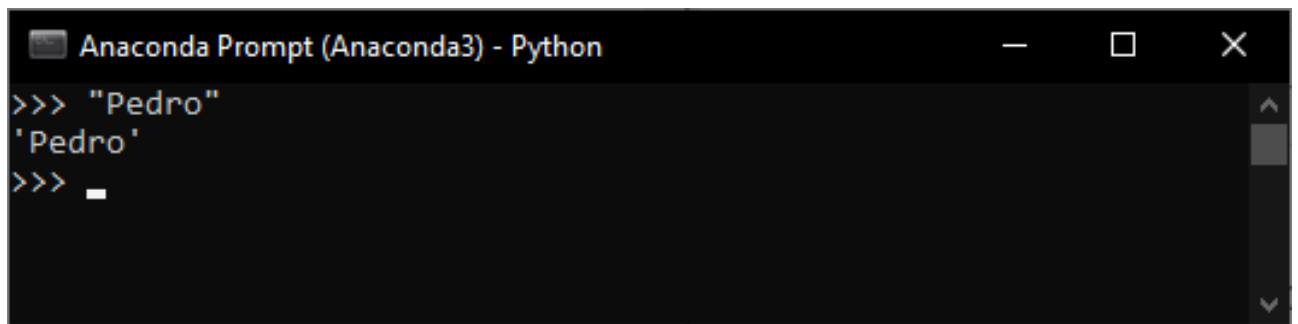


```
Anaconda Prompt (Anaconda3) - Python
>>> 4*5
20
>>> 5-1
4
>>> 40/2
20.0
>>> 2**3
8
>>> _
```

Рисунок 1.3 – Результат виконання математичної операції

## Рядки

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

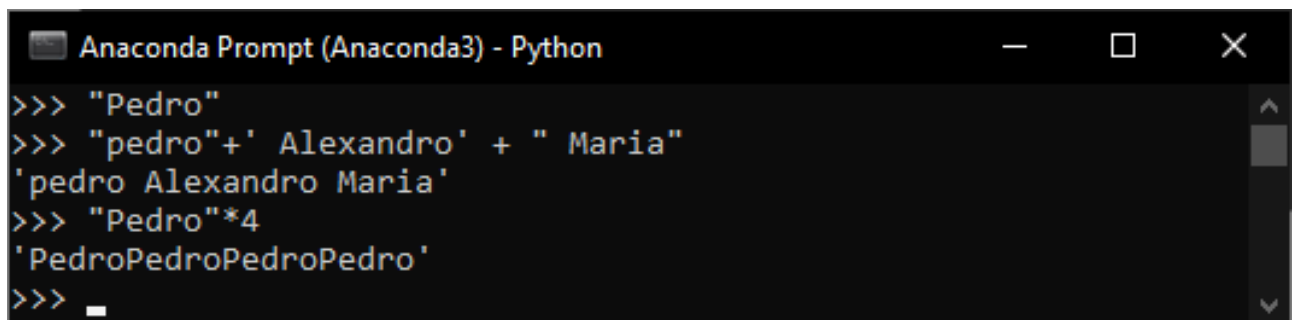


```
Anaconda Prompt (Anaconda3) - Python
>>> "Pedro"
'Pedro'
>>> _
```

Рисунок 1.4 – Результат ведення рядка в консолі

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

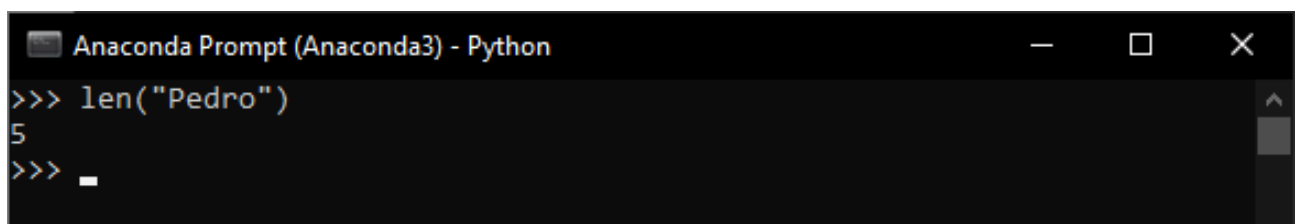
Над рядками також можуть проводитися арифметичні операції:



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

Рисунок 1.5 – Результат виконання математичних операцій над рядками

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



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

Рисунок 1.6 – Результат виклику методу

## Встановлення **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>.

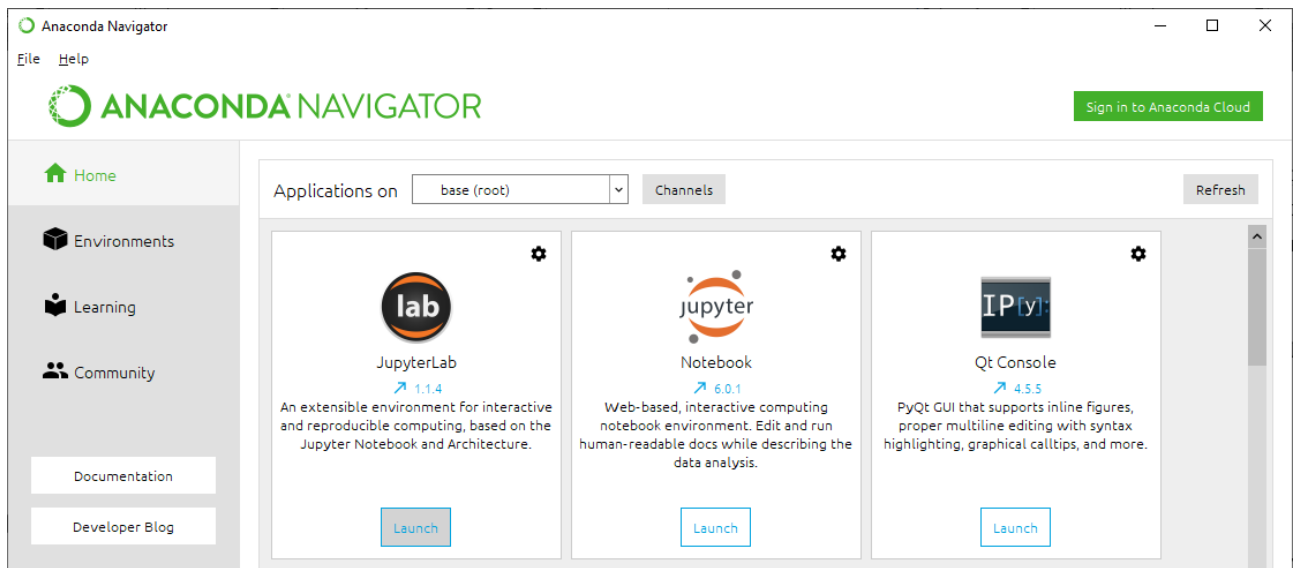


Рисунок 1.7 – Вигляд програми

## Структура програми Python

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 " )
```

Рисунок 1.8 – Демонстрація використання відступів

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

Рисунок 1.9 – Приклад багатослівного виразу

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]]

Рисунок 1.10 – Демонстрація структури програми

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

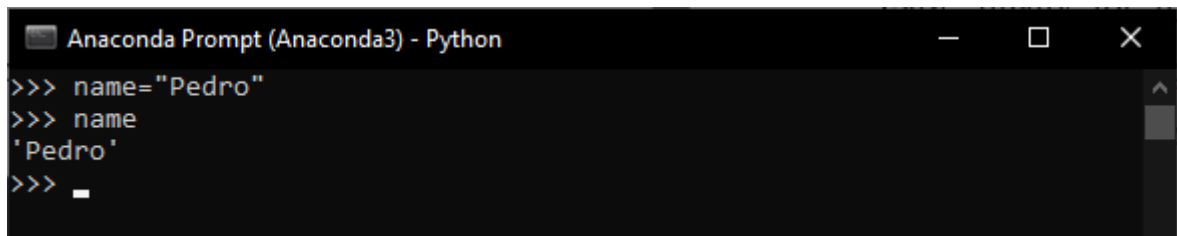
Рисунок 1.11 – Демонстрація продовження оператора на наступному рядку

## Базові відомості щодо мови Python

### Змінні

Змінна зберігає певні дані. Назва змінної в Python має починатися з алфавітного символу або зі знаку 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:

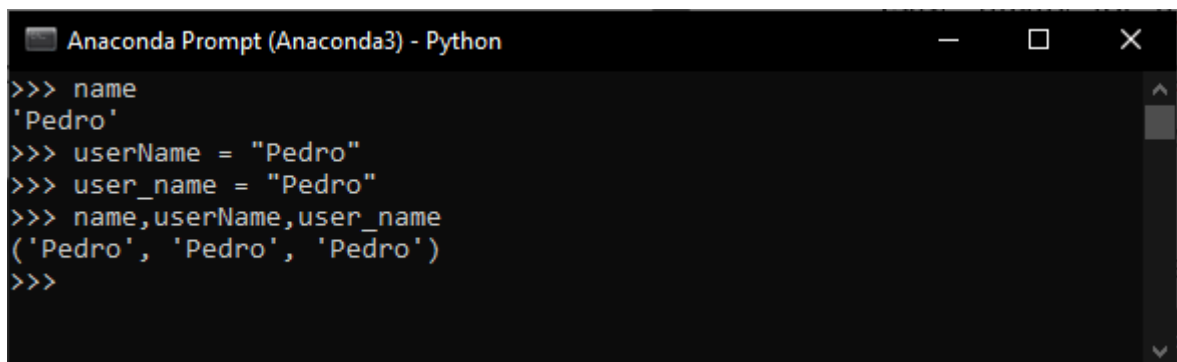


```
Anaconda Prompt (Anaconda3) - Python
>>> name="Pedro"
>>> name
'Pedro'
>>> _
```

Рисунок 1.12 – Створення змінної

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

У Python 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:



```
Anaconda Prompt (Anaconda3) - Python
>>> name
'Pedro'
>>> userName = "Pedro"
>>> user_name = "Pedro"
>>> name,userName,user_name
('Pedro', 'Pedro', 'Pedro')
>>>
```

Рисунок 1.13 – Демонстрація типів найменування

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

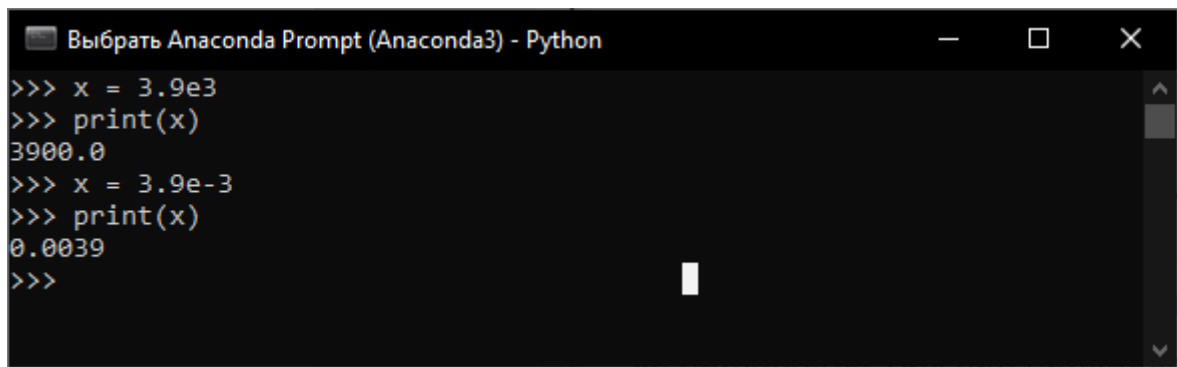
**tuple** - кортеж

**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:

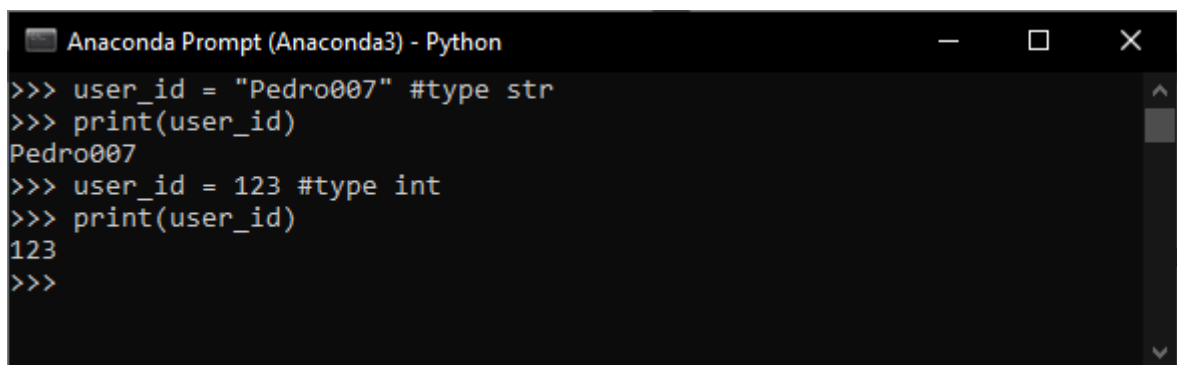


```
Выбрать Anaconda Prompt (Anaconda3) - Python
>>> x = 3.9e3
>>> print(x)
3900.0
>>> x = 3.9e-3
>>> print(x)
0.0039
>>>
```

Рисунок 1.14 – Демонстрація експоненційного запису

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
>>>
```

Рисунок 1.15 – Зміна типу даних змінної

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'>
>>> _
```

Рисунок 1.16 – Визначення типу даних

## 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
```

Рисунок 1.17 – Демонстрація роботи зі списком

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 елемента = 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]
```

Рисунок 1.18 – Отримання доступу до n-го елемента списку

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]

Рисунок 1.19 – Демонстрація «нарізки» списків

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

```

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

```

Рисунок 1.20 – Перевірка належності елемента списку

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}

```

Рисунок 1.21 – Зчіплення списку

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]

Рисунок 1.22 – Складання списку

## 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

Рисунок 1.23 – Імпортування модуля

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`

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

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

Рисунок 1.24 – Використання псевдоніма модуля

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()
```

Рисунок 1.25 – Отримання конкретних значень з модуля

## 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

Рисунок 1.26 – Використання операторів розгалуження

### 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
```

Рисунок 1.27 – Використання циклу while

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
```

Рисунок 1.28 – Використання циклу for

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
```

Рисунок 1.29 – Використання операторів continue та break

### Істинність

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

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

Рисунок 1.30 – Демонстрація булевих змінних

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);
-

## Завдання до виконання

Практична робота складеться з 2-х частин. Перша буде виконуватися в консолі і необхідна для знайомства з базовими функціями Python. Друга частина буде виконуватися в середовищі JupyterLab, і необхідна для закріплення вивчення базових керуючих конструкцій мови Python.

### Перше завдання (варіанти див. таб. 1.1)

- Створити 2 змінні  $X$ ,  $Y$  типу рядок, з власним ім'ям та прізвищем;
- Поєднати рядки через прогалину та зберегти результат у змінній  $Z$ ;
- Підключити модуль `math` та виконати розрахунок:

$$(N + \text{func}(N \bmod 5)) * (N \bmod 10)$$

Де:

$N$  – номер студента у журналі;

`func` - визначається згідно з варіантом.

- Всі проміжні дані надрукувати.

### Друге завдання (варіанти див. таб. 1.2)

- Створити рядок довжиною  $(10 + N \bmod 5)$ , де  $N$  – номер студента у журналі;
- Виконати завдання згідно власного варіанту (таб. 1.2).

Таблиця 1.1 – Варіанти завдань

№	Завдання	№	Завдання
1.	<code>fabs</code>	9.	<code>cos</code>
2.	<code>factorial</code>	10.	<code>sin</code>
3.	<code>log10</code>	11.	<code>tan</code>
4.	<code>log2</code>	12.	<code>atanh</code>
5.	<code>sqrt</code>	13.	<code>asinh</code>
6.	<code>acos</code>	14.	<code>tanh</code>
7.	<code>asin</code>	15.	<code>sqrt</code>
8.	<code>atan</code>	16.	<code>sinh</code>

Таблиця 1.2 – Варіанти завдань

№	Завдання
1.	Масив $X=(x_1, x_2, \dots, x_n)$ містить велику кількість нульових елементів. Визначити положення і розмір найбільш довгої серії таких елементів.
2.	Заданий масив $X=(x_1, x_2, \dots, x_n)$ , в якому можуть бути однакові числа. Знайти максимальний і мінімальний елементи серед неповторюваних чисел.
3.	З масиву чисел $X=(x_1, x_2, \dots, x_n)$ вилучити всі парні за значенням елементи.
4.	У масиві $X=(x_1, x_2, \dots, x_n)$ поміняти місцями перший і другий негативні елементи, третій і четвертий негативні елементи тощо.
5.	Елементи масиву $X=(x_1, x_2, \dots, x_n)$ – це послідовність цифр цілого числа. Переставити цифри числа у зворотному порядку
6.	Відомо, що в цілочисельному масиві $X=(x_1, x_2, \dots, x_n)$ три і тільки три числа, що є рівними між собою. Знайти ці числа
7.	За однократний перегляд масиву знайти його максимальний позитивний елемент $X_{\max}$
8.	Перетворити масив $X$ , розташувавши спочатку його негативні, а потім позитивні елементи, зберігши при цьому в групі негативних та позитивних елементів їх вихідний відносний порядок.
9.	У масиві $X=(x_1, x_2, \dots, x_n)$ поміняти місцями перший і другий позитивні елементи, третій і четвертий позитивні елементи тощо.
10.	Заданий масив цілих чисел $X=(x_1, x_2, \dots, x_n)$ . Сформувати масив $Y=(y_1, y_2, \dots, y_m)$ , помістивши в нього в порядку убутання всі позитивні числа, що входять у масив $X$ .
11.	Заданий цілочисельний масив $X=(x_1, x_2, \dots, x_n)$ , у якому можуть бути однакові числа. Підрахувати кількість повторюваних чисел у масиві.
12.	Виконати циклічне зрушення масиву $X=(x_1, x_2, \dots, x_n)$ на 5 елементів вліво.
13.	Виконати циклічне зрушення масиву $X=(x_1, x_2, \dots, x_n)$ на 3 елементів вправо.
14.	Масив $X=(x_1, x_2, \dots, x_n)$ містить велику кількість нульових елементів. Визначити положення і розмір найбільш довгої серії ненульових елементів.
15.	Заданий масив цілих чисел $X=(x_1, x_2, \dots, x_n)$ . Сформувати масив $Y=(y_1, y_2, \dots, y_m)$ , помістивши в нього в порядку убутання всі негативні числа, що входять у масив $X$ .
16.	Заданий масив цілих чисел $X=(x_1, x_2, \dots, x_n)$ . Сформувати масив $Y=(y_1, y_2, \dots, y_m)$ , помістивши в нього в порядку убутання всі різні (неповторювані) числа, що входять у масив $X$ .



## Приклад виконання

Варіант № 16.

```
C:\Users\aharo>python
Python 3.10.11 (tags/v3.10.11:7d4cc5a, Apr 5 2023, 00:38:17) [MSC v.1929 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> X="Ivan"
>>> Y="Ivanov"
>>> Z=X+" "+Y
>>> Z
'Ivan Ivanov'
>>> import math
>>> func=math.sinh
>>> X=(16+func(1))*6
>>> X
103.0512071618628
>>> |
```

Рисунок 1.31 – Виконання першої частини

```
[1]: x=[1,2,2,4,5,8,4,3,18,11,11] # вхідний масив
k=0 # лічильник
imax=0 # номер максимального елемента
max=x[0] # значення максимального елемента
# сортування масиву
while k<len(x)-1:
    j=k
    imax=k
    max=x[k]
    while j<len(x):
        if max<x[j]:
            max=x[j]
            imax=j
        j+=1
    x[imax]=x[k]
    x[k]=max
    k+=1
print(x) # демонстрація відсортованого масиву
y=[] # створення нового масиву
# перенесення неповторюваних елементів до нового масиву
for i in range(len(x)):
    buf=x[i]
    num=0
    for j in range(len(x)):
        if x[i]==x[j]:
            num+=1
    if num==1:
        y+=x[i]
print(y) # демонстрація результуючого масиву

[18, 11, 11, 8, 5, 4, 4, 3, 2, 2, 1]
[18, 8, 5, 3, 1]
```

Г 1.

Рисунок 1.32 – Виконання другої частини

## **Контрольні запитання**

1. Що таке змінна і для чого вони використовуються?
2. Що таке тип даних, які існують типи даних у Python.
3. Як отримати поточний тип даних змінної у Python?
4. Операції розгалуження, як та для чого застосовуються.
5. Як відокремлюються блоки коду у Python.
6. Пробільні символи у Python та їх застосування.
7. Як задаються рядки у Python, для чого використовуються, можливі типи даних у межах одного рядка.
8. Булеві змінні у Python, для чого використовуються та які значення можуть приймати.
9. Цикли у Python, для чого використовуються, відмінності циклів While та For.
10. Складна логіка управління циклом, оператори continue та break.

## **Оформлення звіту**

- 1) Титульний лист;
- 2) Мета роботи;
- 3) Номер варіанту та умови завдання;
- 4) Програмний код з коментарями;
- 5) Скріни роботи програми;
- 6) Висновки