

Programowanie w Python

od podstaw do zaawansowanego
programowania

MGR INŻ. SZYMON GUZIK

Kontakt:
e-mail: szguzik@wsb.gda.pl



Odpisuję po godzinie 18-tej lub z samego rana (ok. 6-tej)

Przerwa (15 min.)



Po przeprowadzeniu 1 h 30 m zajęć

Po kawę i do toalety można wyjść w każdym momencie ☺

Spis treści

- Co to jest Python ?
- Zastosowanie Python'a
- Przygotowanie Środowiska pracy
- Anaconda
- Jupyter Notebook
 - Tworzenie katalogu
 - Tworzenie pliku
- Operatory wyrażeń Pythona
- Operacje arytmetyczne
- Deklaracja zmiennych
- Wypisywanie na ekran
- Tworzenie listy (analogiczne do tablicy w innych językach programowania)
- Wypisywanie z listy
- Odwracanie listy
- Dodawanie do listy
- Edytowanie listy
- Usuwanie z listy
- Operatory logiczne
- Porównywanie wartości / Instrukcje warunkowe
- Predefiniowane funkcje
- Pętle
- Zadania (7 zadań)

Spis treści

- Listy
- Tuples
- Dictionary
- Set
- OOP
 - Klasa
 - Inicjalizer
 - Metody
 - Atrybuty klasy
 - Dziedziczenie
 - Mutowalność
 - Dekoratory
 - Generatory
- Zadania (7 zadań)

Spis treści

- Break /Continue
- Konwersja typów
- Formatowanie string
- Funkcja Range
- Funkcje Python
 - Argumenty
 - Argumenty kluczowe
 - Return
 - Funkcje anonimowe - Lambda
 - Scope (zasięgi)
- Własny moduł
- Błędy / wyjątki
- Instrukcja Raise
- Input w konsoli
- Zadania (7 zadań)

Python

- - język programowania ogólnego przeznaczenia
- nie jest powiązany z jedną platformą czy z jednym zastosowaniem - tylko jest wszechstronny
- - bogaty zestaw bibliotek standardowych
- bardzo dużo rozwiązań, bibliotek do implementacji powtarzających się funkcjonalności
- np.
 - - otwieranie, czytanie, zapisywanie plików
 - - wysyłanie rzędu HTTP
 - - agregacja danych
 - - obliczenia naukowe
- - czytelny i zwykle przejrzysty kod źródłowy
- - brak nawiasów klamrowych
- - brak "dekoracyjnych" elementów

Python

- - jest to język interpretowany
- - przeciwnieństwo języka komplikowanego (C# czy Java)
- - dynamicznie uruchamiany przez interpreter
- - dostępny dla wielu platform
 - - Windows
 - - Linux
 - - macOS
- - jeden z najpopularniejszych języków na świecie
 - - Obecnie znajduje się w pierwszej 5 (w zależności od sprawdzanych statystyk)
 - - Wzrastająca popularność, zainteresowanie
- - niski próg wejścia dla początkujących
 - - prosta przejrzysta składnia, którą można relatywnie szybko opanować
 - - wystarczy zainstalować interpreter co pozwala na rozpoczęcie pracy z językiem Python.

Zastosowanie

- automatyzacja zadań
- obliczenia naukowe
- obliczenia statystyczne
- analiza danych
- aplikacje dekstopowe
- sztuczna inteligencja
- uczenie maszynowe
- aplikacje webowe

Przygotowanie środowiska pracy

- <https://www.python.org/>

Przygotowanie środowiska pracy



The screenshot shows the Python.org website's "Downloads" section. On the left, there's a sidebar with a code snippet demonstrating a for loop that calculates the product of a list of numbers. Below the code are links for "All releases", "Source code", "Windows", "macOS", "Other Platforms", "License", and "Alternative Implementations". The main content area is titled "Download for Windows" and features a large button for "Python 3.11.0". A note below it states: "Note that Python 3.9+ cannot be used on Windows 7 or earlier." It also mentions that Python can be used on many other operating systems and provides a link to "View the full list of downloads".

For loop on a list of numbers

```
>>> numbers = [1, 2, 3, 4]
>>> product = 1
>>> for number in numbers:
...     product *= number
...
>>> print('The product is:', product)
```

The product is: 24

All releases

Source code

Windows

macOS

Other Platforms

License

Alternative Implementations

Download for Windows

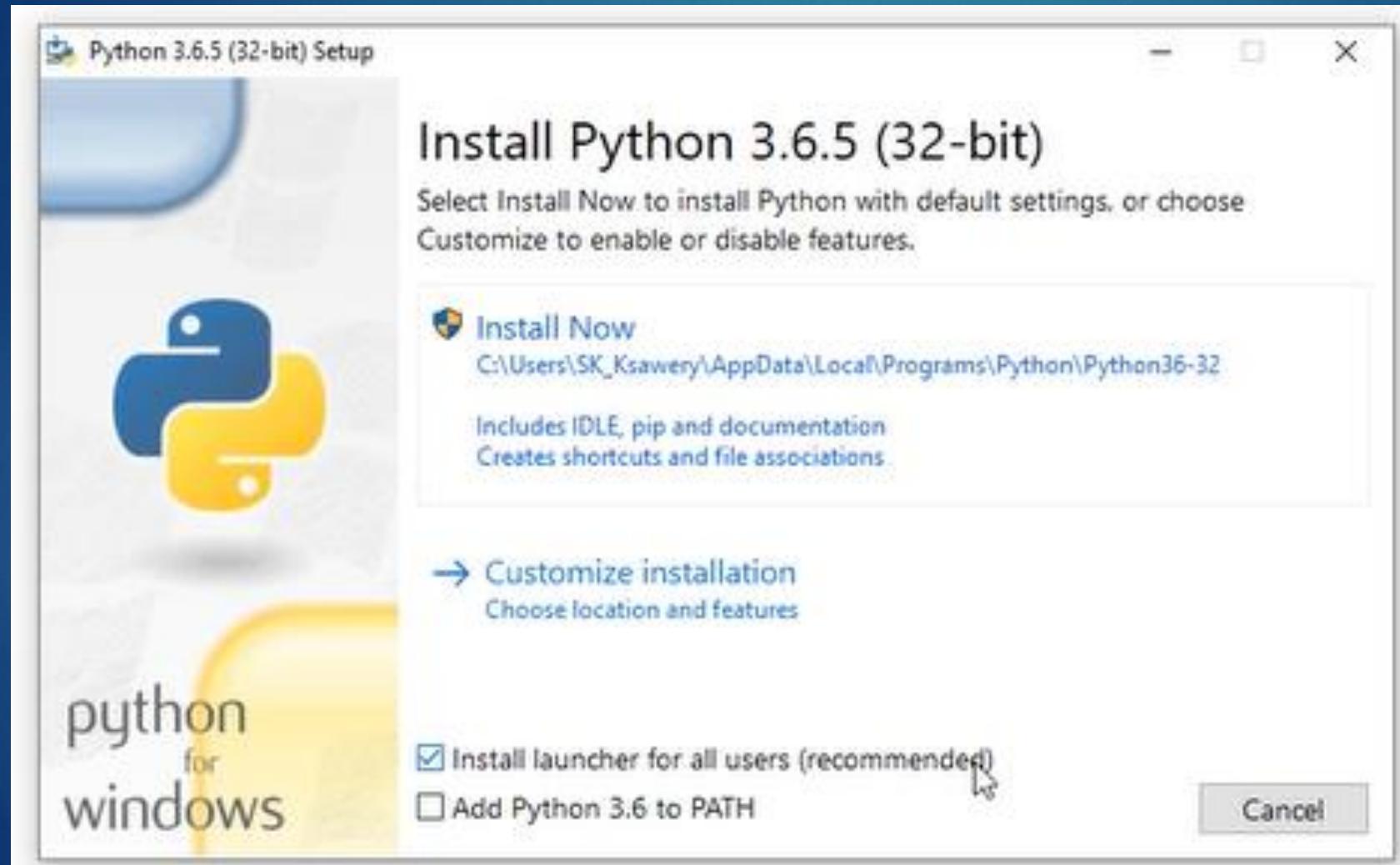
Python 3.11.0

Note that Python 3.9+ cannot be used on Windows 7 or earlier.

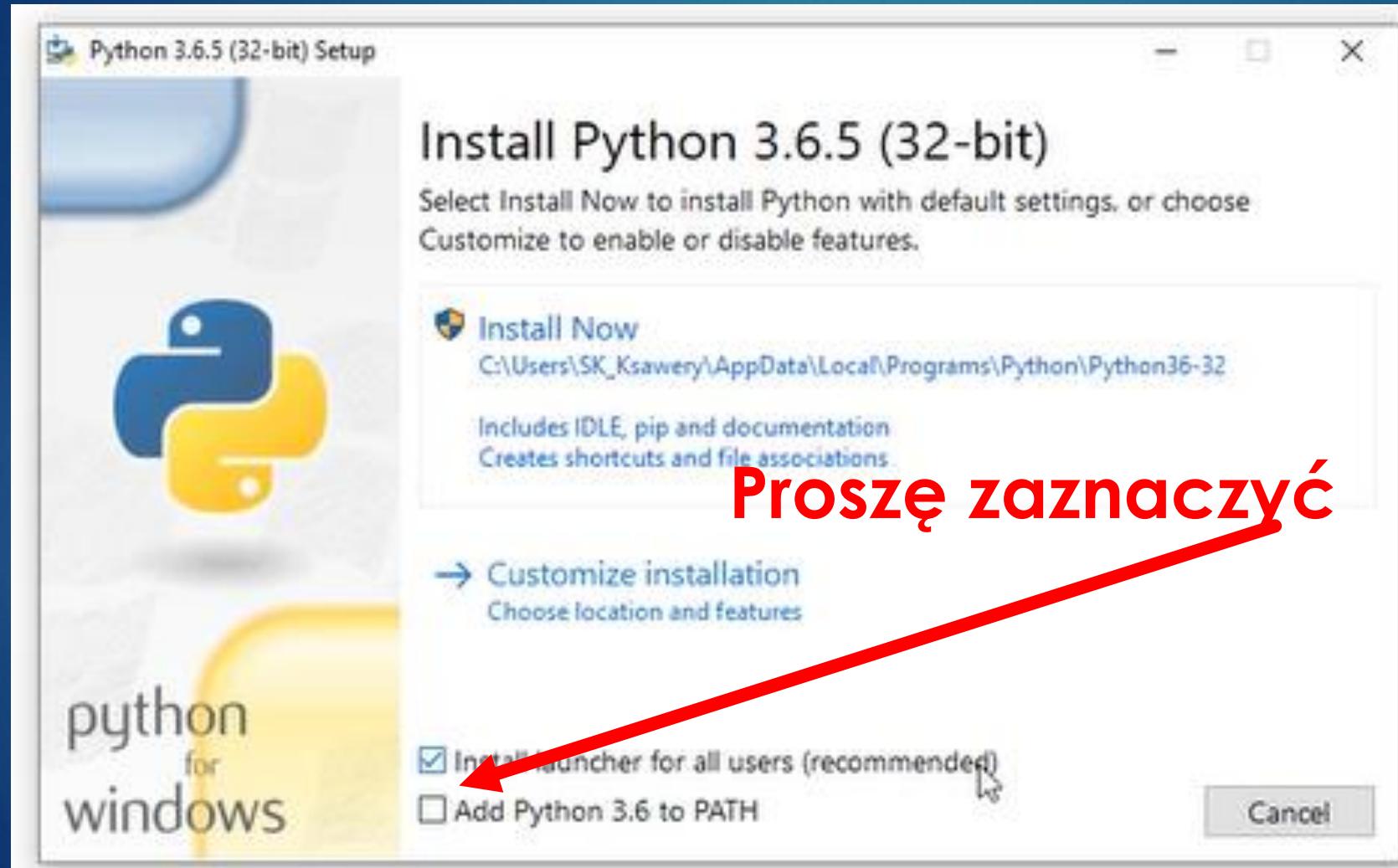
Not the OS you are looking for? Python can be used on many operating systems and environments.

[View the full list of downloads.](#)

Przygotowanie środowiska pracy



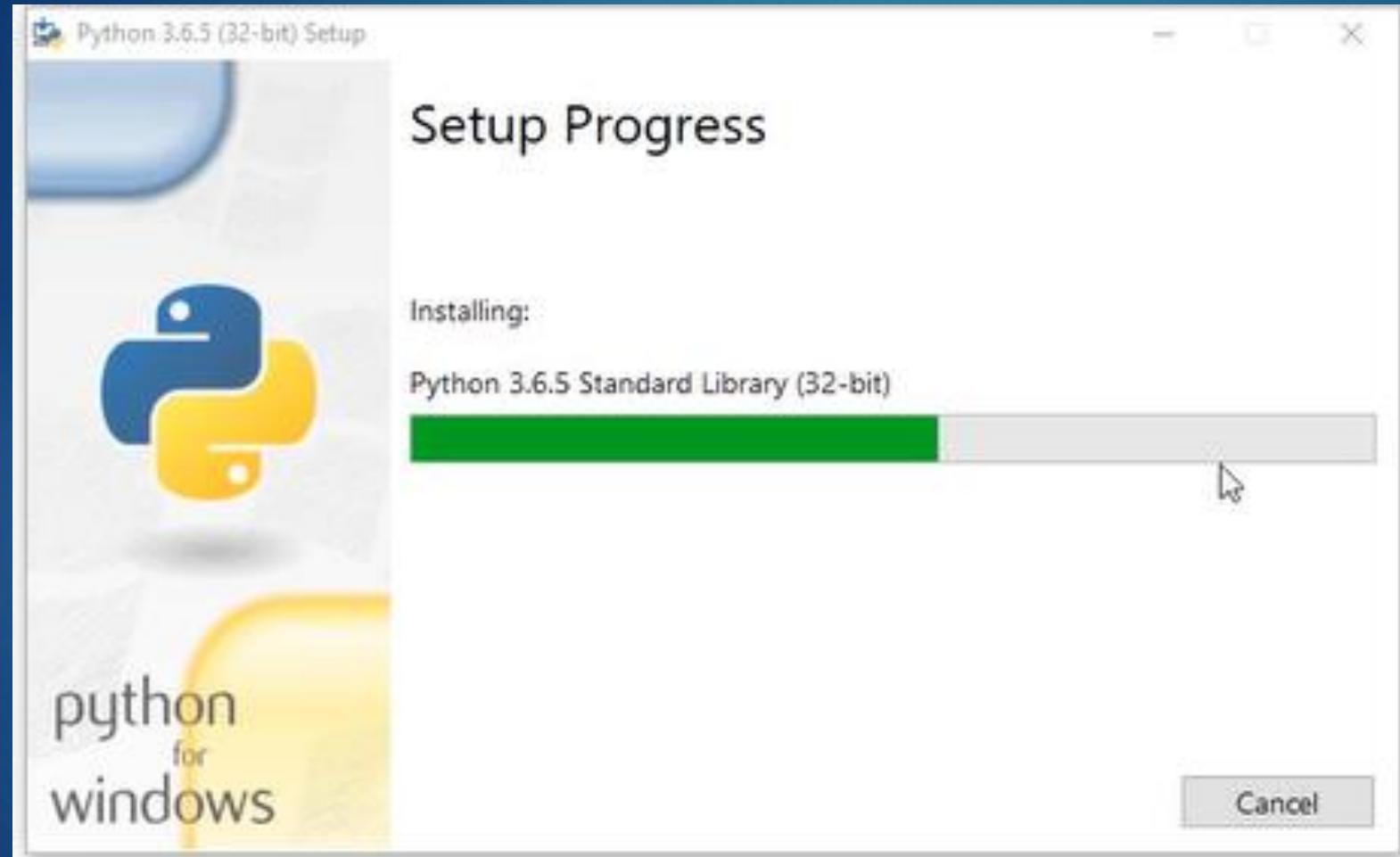
Przygotowanie środowiska pracy



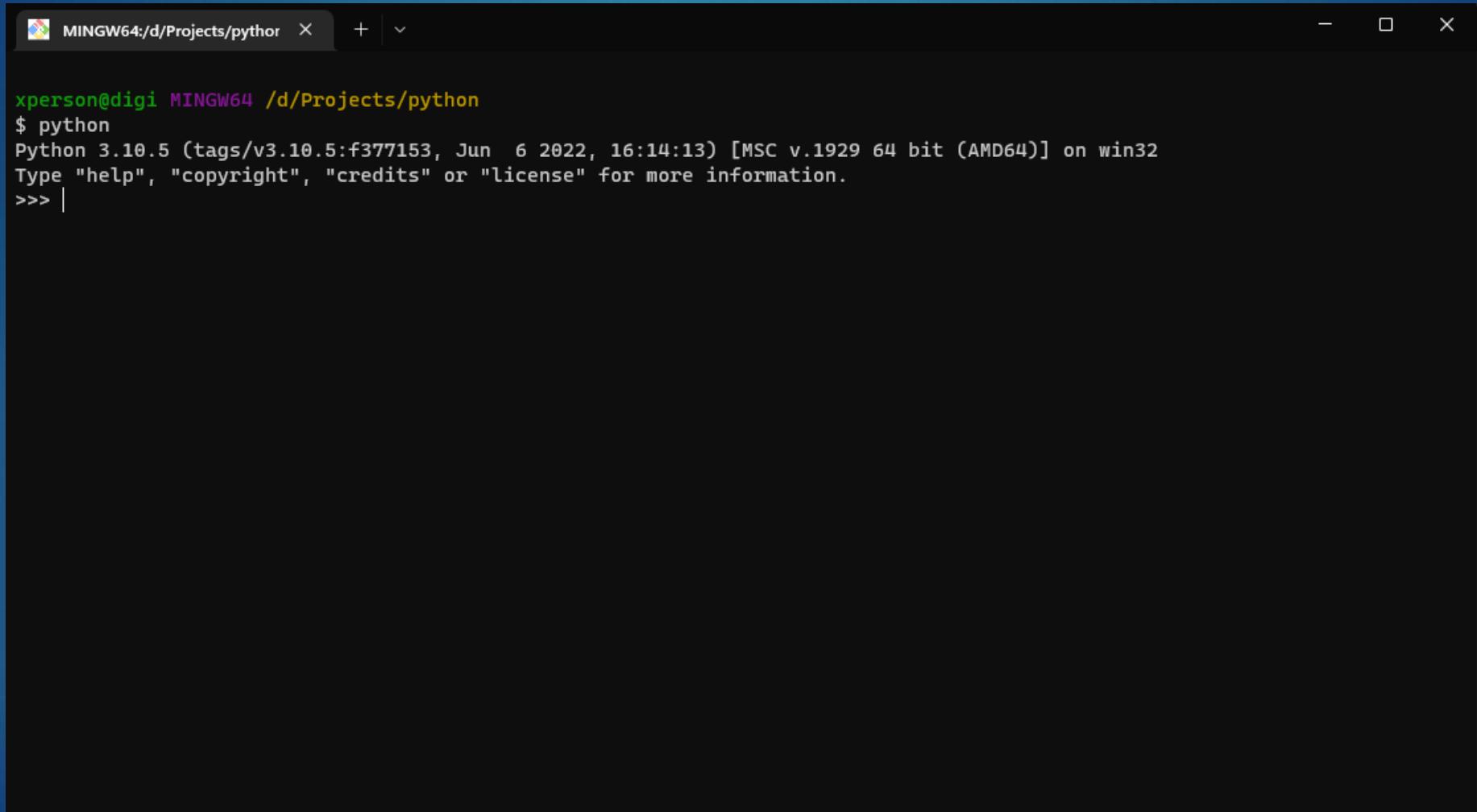
Dodanie Python
do zmiennej środowiskowej

Proszę zaznaczyć

Przygotowanie środowiska pracy



Przygotowanie środowiska pracy



A screenshot of a terminal window titled "MINGW64:/d/Projects/python". The window shows a Python 3.10.5 command-line interface. The output includes the Python version information and a prompt for further input.

```
xperson@digi MINGW64 /d/Projects/python
$ python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> |
```

Anaconda

Anaconda to bezpłatna dystrybucja Open Source szeroko stosowanych języków programowania Python i R obliczenia naukowe (nauka o danych, nauka o danych, uczenie maszynowe, nauka, inżynieria, analiza predykcyjna, duże zbiory danych itp.).

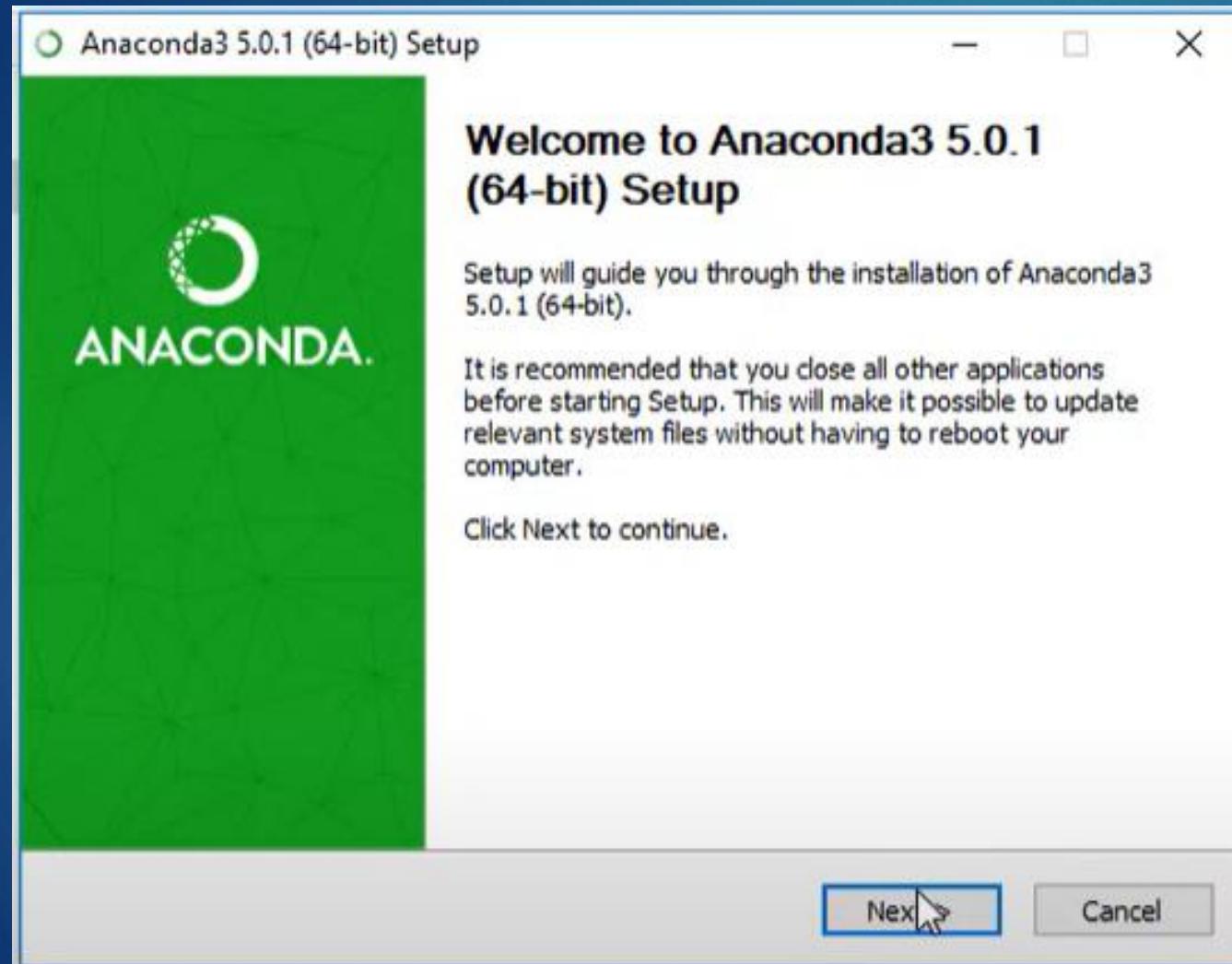


Anaconda

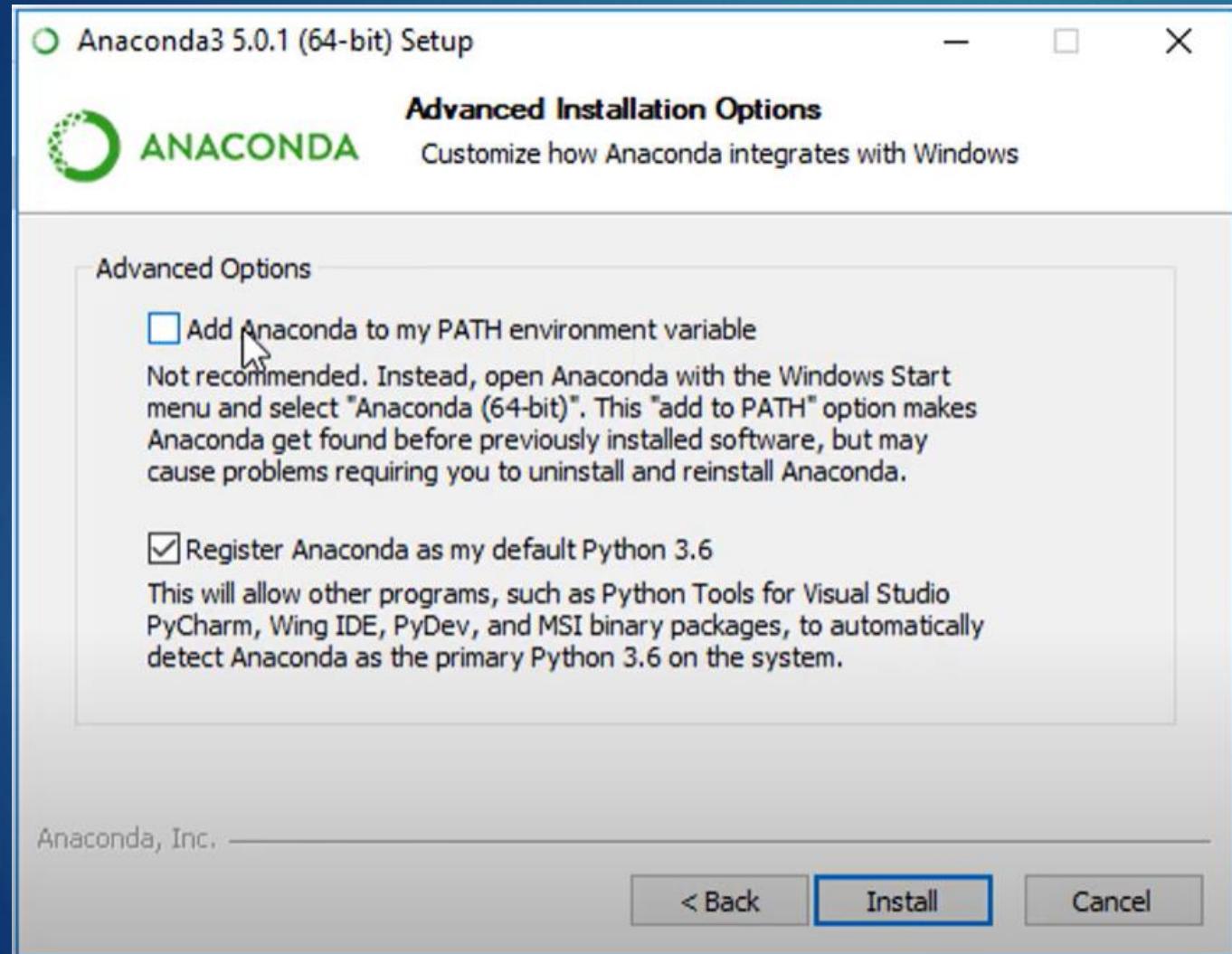
<https://www.anaconda.com/>



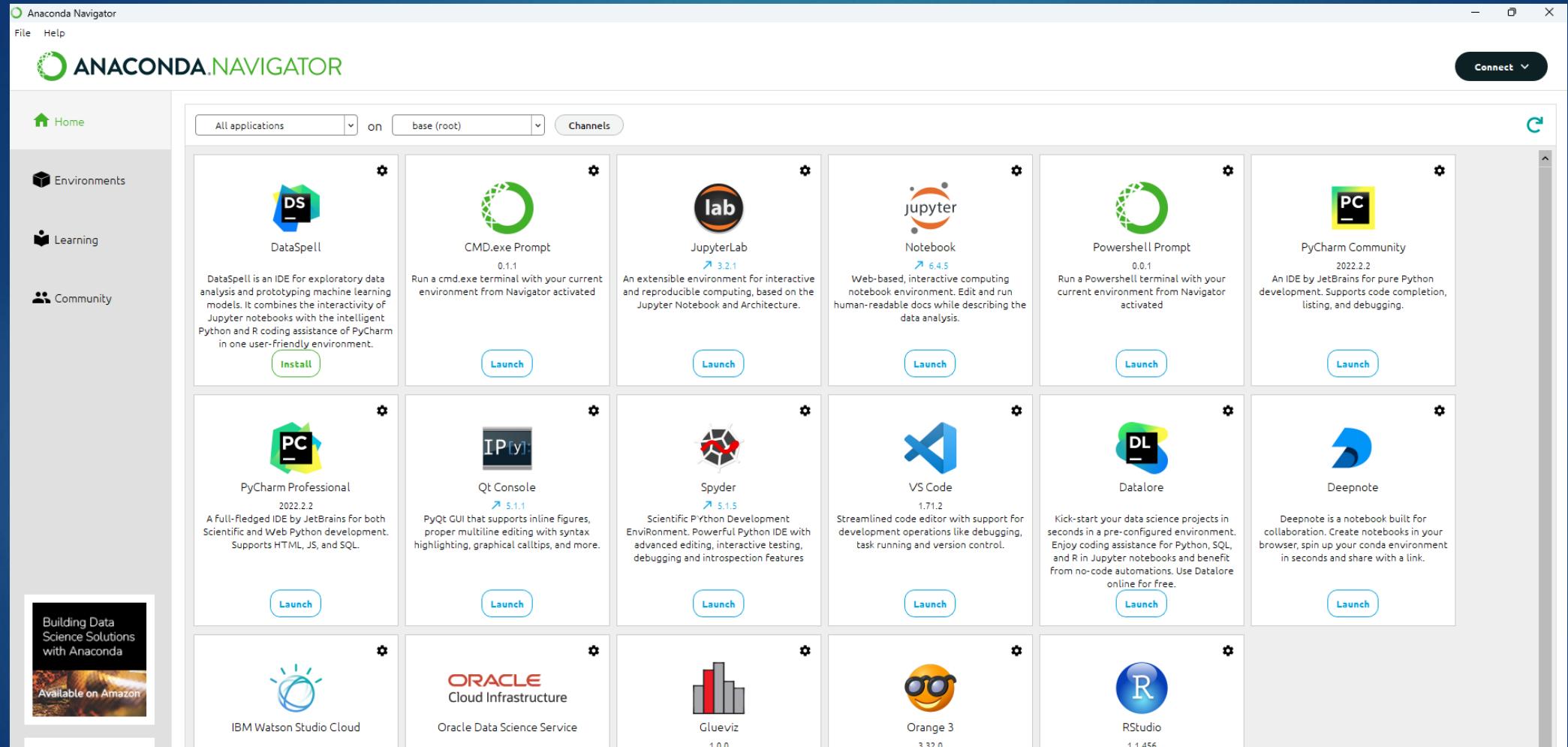
Anaconda



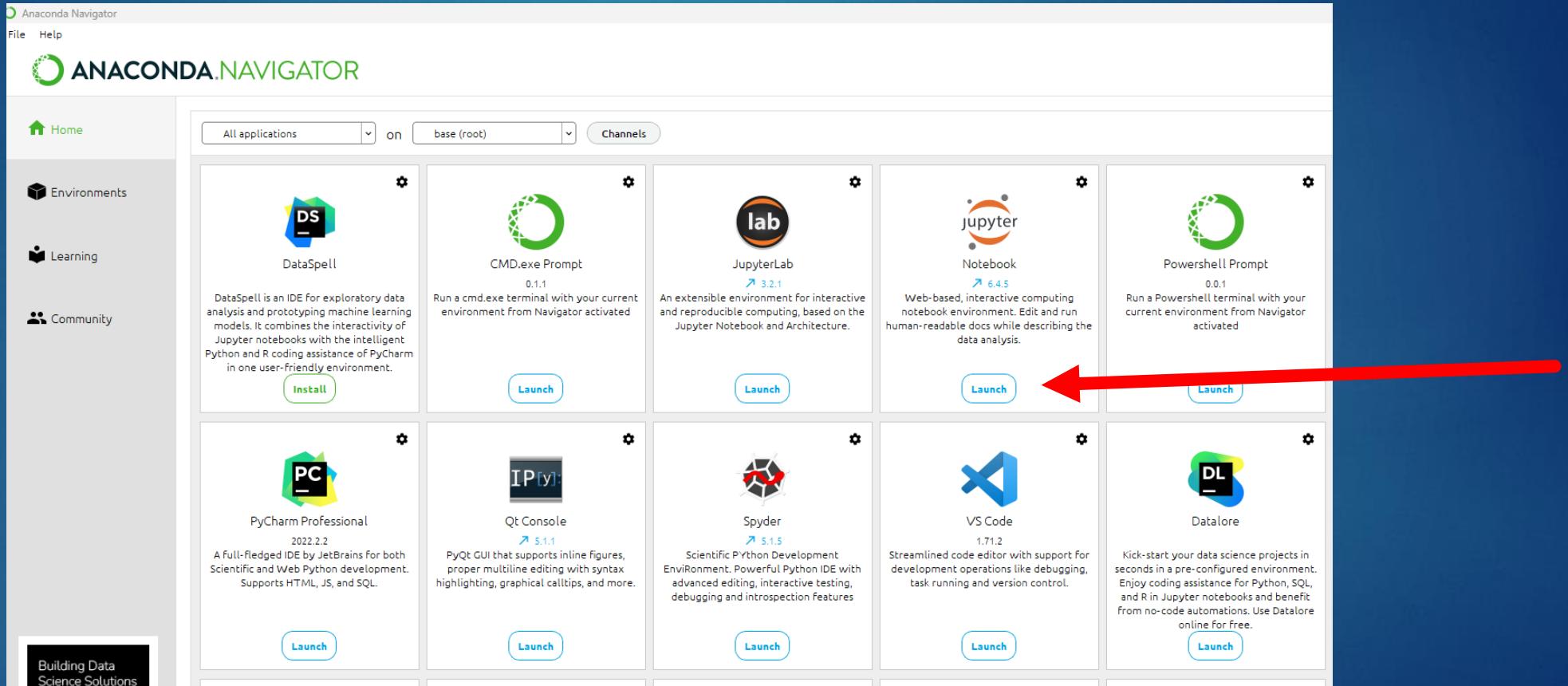
Anaconda



Anaconda



Jupyter Notebook



Jupyter Notebook

localhost:8890/tree

jupyter

Files Running Clusters

Select items to perform actions on them.

Upload New 

	Name	Last Modified	File size
<input type="checkbox"/> 0	/		
<input type="checkbox"/> anaconda3		11 minut temu	
<input type="checkbox"/> Contacts		miesiąc temu	
<input type="checkbox"/> Desktop		6 godzin temu	
<input type="checkbox"/> Documents		6 dni temu	
<input type="checkbox"/> Downloads		3 godziny temu	
<input type="checkbox"/> Favorites		miesiąc temu	
<input type="checkbox"/> Intel		miesiąc temu	
<input type="checkbox"/> Links		miesiąc temu	
<input type="checkbox"/> Music		19 dni temu	
<input type="checkbox"/> OneDrive		3 miesiące temu	
<input type="checkbox"/> Pictures		3 dni temu	
<input type="checkbox"/> Postman		3 miesiące temu	
<input type="checkbox"/> Saved Games		miesiąc temu	
<input type="checkbox"/> Searches		miesiąc temu	
<input type="checkbox"/> Videos		6 dni temu	
<input type="checkbox"/> Zaj1AnalizaDanych		3 godziny temu	
<input type="checkbox"/> PUTTY.RND		2 miesiące temu	128 B
<input type="checkbox"/> Sti_Trace.log		3 miesiące temu	0 B

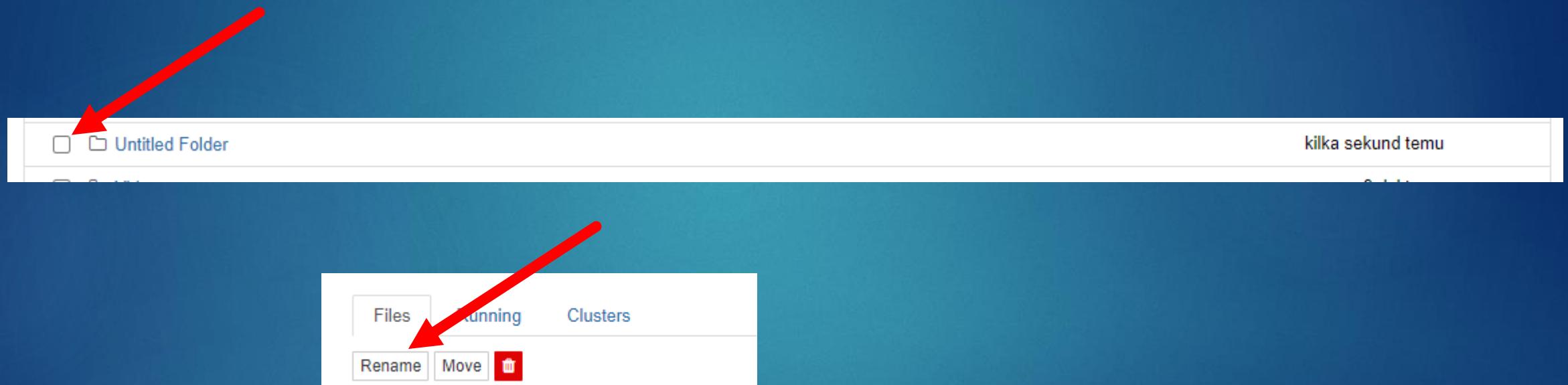
Quit Logout

Jupyter Notebook

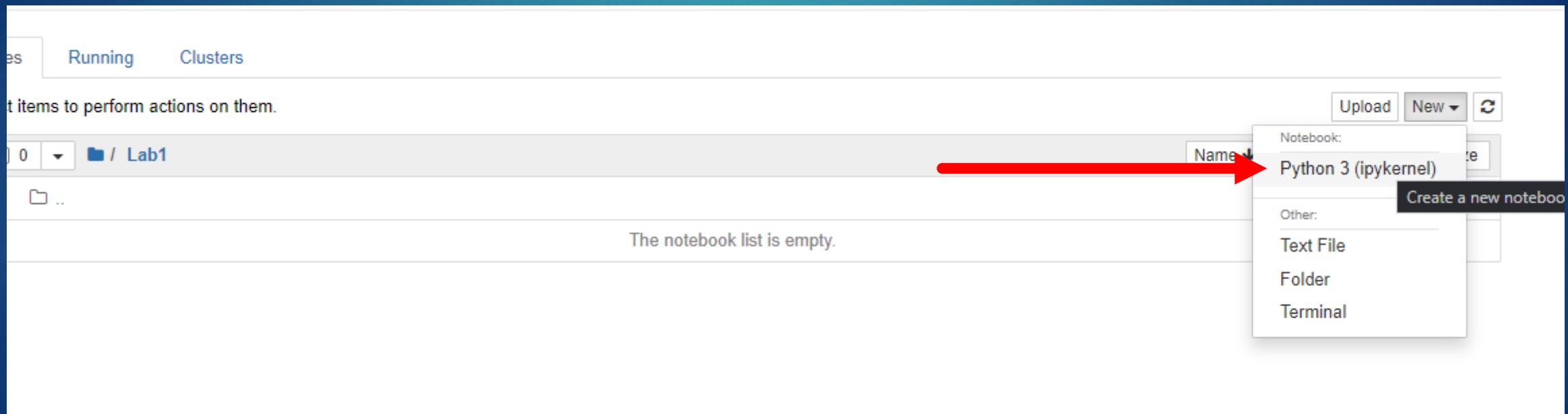


The screenshot shows the Jupyter Notebook interface. At the top, there are three tabs: "Files" (selected), "Running", and "Clusters". Below the tabs, a message says "Select items to perform actions on them." A sidebar on the left lists directory contents: 0, anaconda3, Contacts, Desktop, Documents, and Downloads. On the right, there's a toolbar with "Upload", "New", and a refresh icon. A dropdown menu is open under "New", showing options: "Notebook: Python 3 (ipykernel)", "Other: Text File", "Folder", and "Terminal". Two red arrows point to the "New" button and the "Folder" option in the dropdown menu.

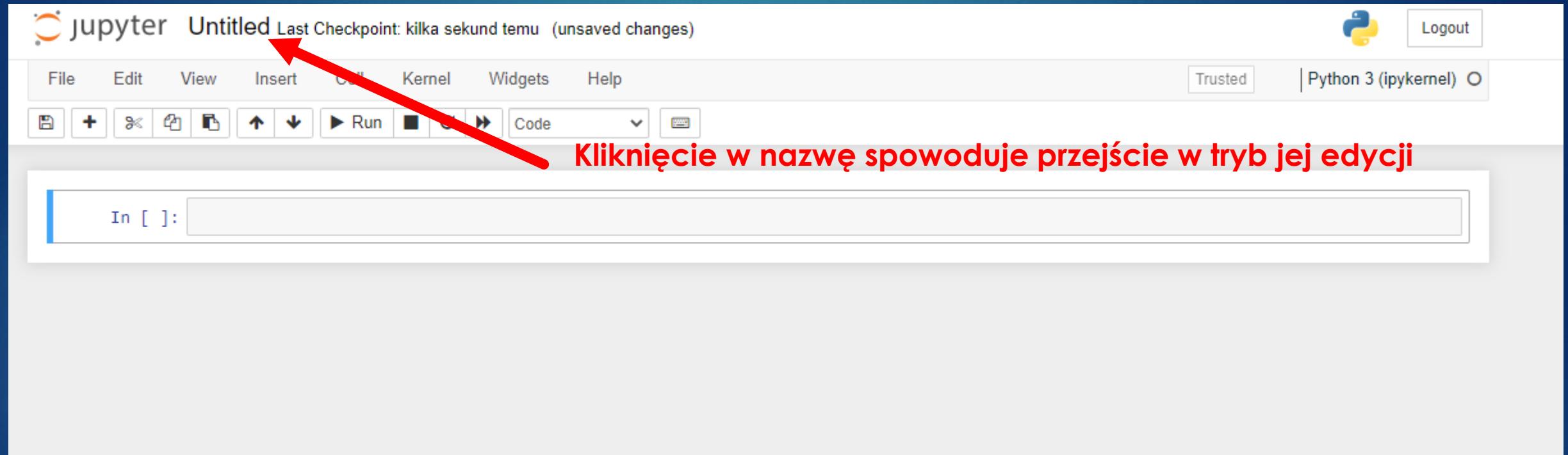
Jupyter Notebook



Jupyter Notebook



Jupyter Notebook



Operatory wyrażeń Pythona

Operatory	Opis
<code>yield x</code>	Protokół send funkcji generatora
<code>lambda argumenty: wyrażenie</code>	Generowanie funkcji anonimowej
<code>x if y else z</code>	Trójargumentowe wyrażenie wyboru (<code>x</code> jest obliczane jedynie wtedy, gdy <code>y</code> jest prawdziwe)
<code>x or y</code>	Logiczne OR (y jest obliczane tylko wtedy, gdy <code>x</code> jest fałszywe)
<code>x and y</code>	Logiczne AND (y jest obliczane tylko wtedy, gdy <code>x</code> jest prawdziwe)
<code>not x</code>	Logiczna negacja
<code>x in y, x not in y</code>	Przynależność (obiekty, po których można iterować, zbiory)
<code>x is y, x is not y</code>	Testy identyczności obiektów
<code>x < y, x <= y, x > y, x >= y</code>	Operatory porównania, podzbiory i nadzbiory zbiorów;
<code>x == y, x != y</code>	Operatory równości wartości
<code>x y</code>	Bitowe OR, suma zbiorów
<code>x ^ y</code>	Bitowe XOR, symetryczna różnica zbiorów
<code>x & y</code>	Bitowe AND, część wspólna zbiorów
<code>x << y, x >> y</code>	Przesunięcie <code>x</code> w lewo lub prawo o <code>y</code> bitów

Operatory wyrażeń Pythona

Operatory	Opis
<code>x + y</code>	Dodawanie, konkatenacja;
<code>x - y</code>	Odejmowanie, różnica zbiorów
<code>x * y</code>	Mnożenie, powtóżenie;
<code>x % y</code>	Reszta z dzielenia, format;
<code>x / y, x // y</code>	Dzielenie: prawdziwe i bez reszty
<code>-x, +x</code>	Negacja, identyczność
<code>~x</code>	Bitowe NOT
<code>x ** y</code>	Potęga
<code>x[i]</code>	Indeksowanie (sekwencje, odwzorowania, inne)
<code>x[i:j:k]</code>	Wycinki
<code>x(...)</code>	Wywołanie (funkcji, metod, klasy, innych obiektów, które można wywoływać)
<code>x.atrybut,</code>	Odwołanie do atrybutu
<code>(...)</code>	Krotka, wyrażenie, wyrażenie generatora
<code>[...]</code>	Lista, lista składana
<code>{...}</code>	Słownik, zbiór, a także zbiór i słownik składany

Operacje arytmetyczne

Suma

```
a = 3  
b = 6  
c = a + b  
print(c)
```

Różnica

```
a = 3  
b = 6  
c = a - b  
print(c)
```

Iloczyn

```
a = 3  
b = 6  
c = a * b  
print(c)
```

Iloraz

```
a = 3  
b = 6  
c = a / b  
print(c)
```

Modulo

```
a = 3  
b = 6  
c = a % b  
print(c)
```

Potęgowanie

```
a = 10  
b = 3  
c = a ** b  
print(c)
```

Deklaracja zmiennych

Zmienna - to obiekt w programowaniu, który przechowuje różnego rodzaju dane niezbędne do działania programu. Zmienna podczas działania programu może zmieniać swoje wartości (jak wskazuje nazwa). Tworząc zmienną musimy nadać jej nazwę oraz typ, który określa co nasza zmienna będzie przechowywać. Nadając nazwę trzymamy się następujących reguł:

- zmienna jest jednym ciągiem znaków bez spacji
- nie zaczynamy nazwy od cyfry
- nie używamy polskich liter takich jak q, ę itp.
- nazwa zmiennej powinna kojarzyć się z przeznaczeniem tej zmiennej
- nazwa nie może być słowem kluczowym języka programowania

[<https://www.algorytm.edu.pl/wstp-do-c/typy-zmiennych.html>]

Deklaracja zmiennych

```
print(a)
a = 2
```

```
a = 1
b = 3
c = a + b
print(a, b, c)
```

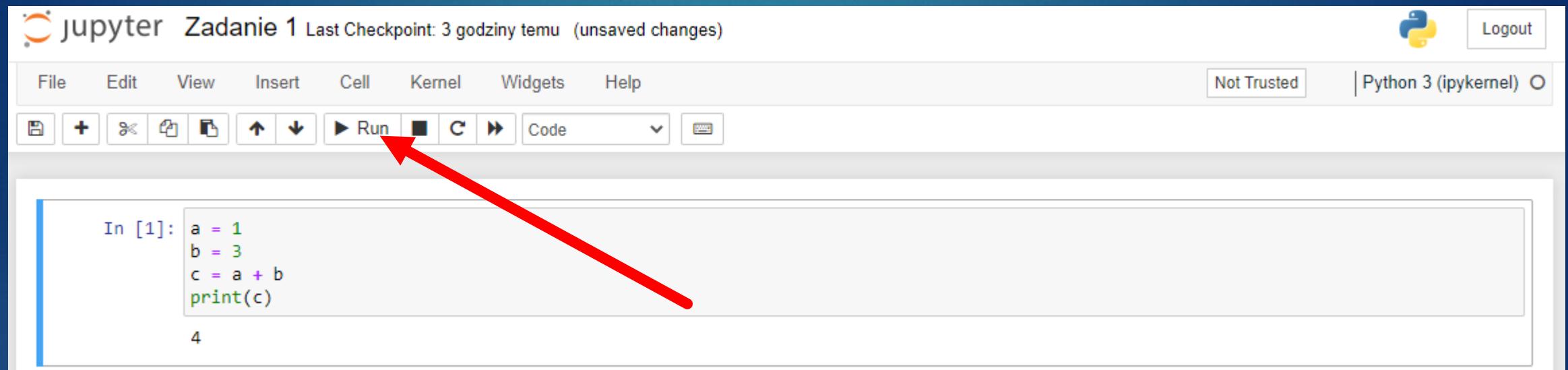


Wypisywanie na ekran (w konsoli)

```
print(a,b,c)
```

```
python program.py
```

Wypisywanie na ekran (Jupyter Notebook)



Lista

(analogiczne do tablicy w innych językach programowania)

- Zbiór wartości
- Index rozpoczyna się od zera
- Łatwe odczytywanie
- Łatwe dodawanie
- Łatwe usuwanie wartości



Tworzenie listy

(analogiczne do tablicy w innych językach programowania)

```
first_list = [1,2,3,4,5, ...n]
```

Wypisywanie z listy

(analogiczne do tablicy w innych językach programowania)

first_list[index]

first_list[0] // 1 element listy

first_list[19] // 20 element listy

Odwracanie listy

(analogiczne do tablicy w innych językach programowania)

```
first_list = [1,2,3,4,5, ...n]
```

```
print(first_list[::-1])
```

Dodawanie do listy

(analogiczne do tablicy w innych językach programowania)

```
first_list = [1,2,3,4,5, ...n]
```

```
first_list.append(y)
```

```
first_list = [1,2,3,4,5, ...n,y]
```

Edytowanie listy

(analogiczne do tablicy w innych językach programowania)

```
first_list = [1,2,3,4,5, ... 100]
```

n-1 // bo liczymy od 0!

Index[n-1] //99
Wartość 100

```
first_list[99] = 200
```

```
first_list = [1,2,3,4,5, ... 200]
```

Index[n-1] //99
Wartość 200

Usuwanie z listy

(na podstawie numeru indexu)

Przed

```
first_list = [1,2,3,4,5, ...100]
```

```
del first_list[1] ← n-1 // numer indexu
```

Po

```
first_list = [1,3,4,5, ...100]
```

Usuwanie z listy

(Usuwanie pierwszego występującego elementu listy)

Przed

```
first_list = [1,'Kasia',3,4,5,...100]
```

```
first_list.remove('Kasia')
```

Wartość indexu

Po

```
first_list = [1,3,4,5,...100]
```

Operatory logiczne

Operator	Opis
and	oraz
or	lub
not	zaprzeczenie

Operatory logiczne

Operator	Opis
and	oraz
or	lub
not	zaprzeczenie

Operacja	Wynik
prawda i prawda	prawda
prawda i fałsz	fałsz
fałsz i prawda	fałsz
fałsz i fałsz	fałsz
prawda lub prawda	prawda
prawda lub fałsz	prawda
fałsz lub prawda	prawda
fałsz lub fałsz	fałsz

Operatory logiczne

Założenie - oba spełnione:

- A = 3
- B = 4

```
a = 3
b = 4

if a == 3 and b == 4:
    print('Warunek spełniony')
```

python program.py

Założenie – przynajmniej jeden:

- A = 3
- B = 10

```
a = 3
b = 10

if a == 3 or b == 4:
    print('Warunek spełniony')
```

python program.py

Warunek spełniony

Operatory logiczne

Założenie - oba spełnione:

- A = 3
- B = 4
- C = 5

```
a = 3
b = 4
c = 1

if a == 3 and b == 4 and c == 5:
    print('Warunek spełniony')
```

python program.py

Warunek spełniony

Operatory logiczne

Założenie:

- A = False
- B = 4
- C = 10

```
a = False
b = 4
c = 10

if not a:
    print('Warunek spełniony')
```

python program.py

Warunek spełniony

Instrukcje warunkowe

Def: Kontrola przepływu programu !

Symbol	Opis
>	Większy
\geq	Większy lub równy
<	Mniejszy
\leq	Mniejszy lub równy
\equiv	Równy
\neq	Nie równy (negacja)

Porównywanie wartości / Instrukcje warunkowe

```
print(1 == 1)
```

```
True
```

```
print(1 == 3)
```

```
False
```

```
$ python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1 == 1)
True
>>>
```

```
$ python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit
 (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1 == 3)
False
'
```

Porównywanie wartości / Instrukcje warunkowe

```
print(1 <= 1)
```

```
True
```

```
print(1 <= 3)
```

```
True
```

```
$ python
Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print(1 <= 1)
True
>>> |
```

```
>>> print(1 <= 3)
True
>>>
```

Porównywanie wartości / Instrukcje warunkowe

```
print(1 >= 1)
```

```
True
```

```
print(1 >= 3)
```

```
False
```

```
>>> print(1 >= 1)  
True  
>>> |
```

```
>>> print(1 >= 3)  
False  
>>> |
```

Porównywanie wartości / Instrukcje warunkowe

```
print(1 != 1)
```

```
False
```

```
a = 1
b = 3
print(a != b)
```

```
True
```

```
>>> print(1 != 1)
False
>>> |
```

```
>>> a = 1
>>> b = 3
>>> print (a != b)
True
>>> |
```

Porównywanie wartości / Instrukcje warunkowe

```
if warunek:  
    print("To jest prawda")
```

W instrukcji warunkowej należy pamiętać o wcięciach
(4 spacje)

```
if 1 == 1:  
    print('To jest prawda')
```

```
if 1 == 0: I  
    print('To jest prawda')
```

Porównywanie wartości / Instrukcje warunkowe

wcięcia(4 spacje)

```
a = 200
if a == 200:
    print("To jest prawda")
else:
    print("To jest fałsz")
print("Koniec programu")
```

Porównywanie wartości / Instrukcje warunkowe

wcięcia(4 spacje)

```
a = 200
if a == 200:
    print("a = 200")
elif a == 100:
    print('a = 100')
else:
    print("Nic nie pasuje")
print("Koniec programu")
```

Predefiniowane funkcje (przykładowe)

Funkcje `max()` i `min()` dają nam odpowiednio największe i najmniejsze wartości występujące w liście:

```
>>> max('Hello world')
'w'
>>> min('Hello world')
' '
>>>
```

Funkcja `len()`, która mówi nam, ile elementów znajduje się w przekazanym argumencie. Jeśli argument przekazany do funkcji `len()` jest napis, to zwraca liczbę znaków w tym napisie

```
>>> len('Hello world')
11
>>>
```

Funkcja `type()` zwraca typ wartości

```
>>> type(32)
<class 'int'>
```

Funkcja `input()` pobiera dane wprowadzone np. przez użytkownika

```
x = input('Podaj liczbę:')
print(x)

Podaj liczbę:x
x
```

Pętle

Pętle (ang. loop) to konstrukcje językowe służące do zdefiniowania szeregu instrukcji, które będą powtarzane wielokrotnie. Najbardziej podstawowe rodzaje pętli: while, for

[MIROSŁAW ZELENT]

Zadania do wykonania



Zadanie 1

Napisz program, który będzie sprawdzał długość imienia.

- jeżeli warunek jest spełniony program ma wypisać w konsoli "Masz długie imię"
- w przeciwnym wypadku "Masz krótkie imię"

Po zakończeniu sprawdzania program ma również zwrócić ciąg znaków "Koniec programu"

Zadanie 2

Napisz program, który będzie sprawdzał długość imienia.

- imię będzie podawane w konsoli
- jeżeli warunek nie jest spełniony program ma wypisać w konsoli "Masz krótkie imię"
- w przeciwnym wypadku "Masz długie imię"

Po zakończeniu sprawdzania program ma również zwrócić ciąg znaków "Koniec programu"

Zadanie 3 – praca samodzielna (30 min)

Poczytaj o importowaniu w Python, następnie wyszukaj informację o metodzie służącej do losowego wypisywania liczb z podanego przedziału (jako parametru) oraz konkatenacji. Samodzielnie zdobądź wiedzę na temat tworzenia funkcji w Python

Napisz w oparciu od 2 funkcje program, który będzie zawierał pustą listę. Następnie stwórz pierwszą funkcję, której zadaniem jest sprawdzenie czy ilość elementów na liście jest równa 50.

Pozytywny scenariusz wypisuje na ekranie „Wypisano 50 pozycji do tablicy” Negatywny scenariusz uruchamia drugą funkcję, której zadaniem jest wstrzyknięcie do listy kolejnej pozycji. Do argumentu tej funkcji przekazać należy już losową wartość.

Druga funkcja oprócz dodania przekazanej wartości powinna wyświetlać na ekranie bieżącą listę oraz wypisywać z boku listy sumę jej elementów. Ostatnim zadaniem tej funkcji jest wywołanie pierwszej metody w celu weryfikacji wielkości listy

Po zakończeniu programu należy wypisać również „Koniec programu”

Zadanie 3 – rozwiązańie

```
import random

array = []

def addToArray(item):
    array.append(item)
    print(array, ' ilosc el w tablicy = ', len(array))
    checkArray()

def checkArray():
    if len(array) == 50:
        print("Wypisano 50 pozycji do tablicy")
    else:
        addToArray(random.randint(0,9))

checkArray()
print("Koniec programu")
```

```
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 35
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 36
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 37
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 38
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 39
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 40
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 41
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 42
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 43
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 44
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 45
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 46
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 47
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 48
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 49
[7, 8, 7, 9, 6, 9, 5, 5, 7, 3, 1, 1, 0, 4, 2, 0, 3, 8, 6, 5, 9, 5, 1, 8, 0, 4, 9, 3, 3, 3, 4, 3, 2, 0, 0, 0] ilosc el w tablicy = 50
Wypisano 50 pozycji do tablicy
Koniec programu
```

Zadanie 4

W oparciu o instrukcje warunkowe sprawdź czy podana liczba przez użytkownika jest podzielna przez 3 lub 5 a może przez obie wartości.

W zależności od wyniku zwróć odpowiednią informację

Zadanie 5

Napisz program na obliczanie delty

Sprawdź czy wszystkie warianty zostały rozpatrzone. W przypadku braku wariantu dopisz jego logikę

Liczymy deltę:

$$\Delta = b^2 - 4ac = 2^2 - 4 \cdot 1 \cdot (-3) = 4 + 12 = 16$$

Zatem $\Delta > 0$ czyli mamy dwa rozwiązania:

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a} = \frac{-2 - \sqrt{16}}{2 \cdot 1} = \frac{-6}{2} = -3$$

$$x_2 = \frac{-b + \sqrt{\Delta}}{2a} = \frac{-2 + \sqrt{16}}{2 \cdot 1} = \frac{2}{2} = 1$$

Zadanie 6

Wyszukaj wzór na pole kwadratu o trójkąta

Daj użytkownikowi możliwość wyboru pola do obliczenia gdzie

- Wybranie 1 oznacza kwadrat
- Wybranie 2 oznacza trójkąt

W zależności od decyzji użytkownika pobierz odpowiednią ilość zmiennych niezbędnych do obliczenia wybranej figury. Następnie oblicz pole wybranej pozycji i zwróć użytkownikowi dane, które podał odpowiednio opisane oraz pole figury.

Zadanie 7

Stwórz kalkulator na

- Dodawanie
- Odejmowanie
- Mnożenie
- Dzielenie
- Resztę z dzielenia

Listy

- Zbiór wartości
- Index rozpoczyna się od zera
- Łatwe odczytywanie
- Łatwe dodawanie
- Łatwe usuwanie wartości

```
my_list = [1, 2, 3, 4]
```

```
my_list[0] # 1
```

```
my_list[0] = 20
```

```
my_list = [1, 2, 3.5, 's', True]  
print(my_list)
```

Listy

- Iterowanie listy
- Odwracanie listy
- Sortowanie listy
- Konkatenacja listy
- Dodawanie do listy
- Długość listy
- Sprawdzanie wystąpienia w liście

```
my_list = [10, 20, 30, 40]
```

```
for v in my_list:  
    print(v)
```

```
my_list.reverse()  
print(my_list)
```

```
my_list.sort()  
print(my_list)
```

```
my_list = [10, 20, 30, 40]  
my_list_2 = [50, 60]  
main_list = my_list + my_list_2 * 2  
print(main_list)
```

```
my_list = [10, 20, 30, 40]  
my_list.append(50)  
print(my_list)
```

```
my_list = [10, 20, 30, 40]  
my_list_2 = [50, 60]  
main_list = my_list + my_list_2 * 2  
print(len(main_list))
```

```
my_list = [10, 20, 30, 40]  
my_list_2 = [50, 60]  
main_list = my_list + my_list_2 * 2  
print(10 in main_list)
```

Lista

Przed

```
first_list = [1,2,3,4,5, ...100]
```

```
del first_list[1] ← n-1 // numer indexu
```

Po

```
first_list = [1,3,4,5, ...100]
```

Lista

(Usuwanie pierwszego występującego elementu listy)

Przed

```
first_list = [1,'Kasia',3,4,5,...100]
```

```
first_list.remove('Kasia')
```

Wartość indexu

Po

```
first_list = [1,3,4,5,...100]
```

Tuples

- Zbiór wartości
- Wartości mogą być różnych typów
- Index
- Wartości nie można zmodyfikować

```
values = (1, 2, 3, 'abc', True)  
print(values)
```

```
(1, 2, 3, 'abc', True)
```

```
my_tuple = (1, 2, 3, 4)
```

```
my_tuple = (1, 2, 3, 4)
```

```
my_tuple[0] # 1
```

```
my_tuple[0] = 20 # błąd!
```

Tuples

- Iterowanie tuples
- Odwoływanie do elementu tuples
- Wycinanie (3 pierwsze wartości)
- Czy element znajduje się w tuples ?
- Powtarzanie elementów tuples
- Konkatenacja tuples
- Sprawdzanie ilości elementów w tuples

```
values = (1, 2, 3, 'abc', True)
for v in values:
    print(v)
```

```
values = (1, 2, 3, 'abc', True)
print(values[2])
```

```
values = (1, 2, 3, 4, 5, 6, 7)
new_values = values[:3]
print(new_values)
```

```
values = (1, 2, 3, 4, 5, 6, 7)
new_values = values[:3]
print( in new_values)
```

```
values = (1, 2, 3, 4, 5, 6, 7)
new_values = values[:3]
print(new_values * 2)
```

```
values = (1, 2, 3, 4, 5, 6, 7)
new_values = values[:3]
print(new_values + (10, 20))
```

```
values = (1, 2, 3, 4, 5, 6, 7)
new_values = values[:3]
print(len(new_values + (10, 20)))
```

Dictionary

- Zbiór wartości
- Nie ma indexu
- Zawiera pary klucz – wartość
- Mutowalne

```
options = {  
    'env': 'production',  
    'db': 'mysql',  
    'version': 1.0  
}
```

```
print(options)
```

```
my_d = {'a': 1, 'b': 2}
```

```
my_d = {'a': 1, 'b': 2}
```

```
my_d['a'] # 1
```

```
my_d['a'] = 20
```

Dictionary

```
options = {
    'env': 'production',
    'db': 'mysql',
    'version': 1.0,
    'show_errors': True
}

options.update({
    'user': 'admin',
    'app': 0
})
print(options.get('version'))
```

```
options = {
    'env': 'production',
    'db': 'mysql',
    'version': 1.0,
    'show_errors': True
}

options['user'] = 'admin'
print(options)
```

- Dodawanie pojedynczego elementu
- Dodawanie wielu elementów
- Usuwanie elementu
- Wypisywanie wartości
- Wypisywanie klucza
- Iterowanie słownika

```
for key in options:
    print(options[key])
```

```
options = {
    'env': 'production',
    'db': 'mysql',
    'version': 1.0,
    'show_errors': True
}

del options['db']
print(options)

print(options.values())
```

```
print(options.keys())
```

Set (Zbiór)

Część wspólna zbioru a i b

```
a = set([1, 2, 3, 4])
b = set([1, 3, 7, 9])
print(a.intersection(b))
```

Łączenie wartości zbioru a i b

```
a = set([1, 2, 3, 4])
b = set([1, 3, 7, 9])
print(a.union(b))
```

Wypisywanie

```
a = set([1, 2, 3, 4, 2, 4, 3])
print(a)
```

Elementy zbioru a, które są
inne niż w zbiorze b

```
a = set([1, 2, 3, 4])
b = set([1, 3, 7, 9])
print(a.difference(b))
```

Elementy zbioru a, oraz zbioru b,
które są unikalne

```
a = set([1, 2, 3, 4])
b = set([1, 3, 7, 9])
print(a.symmetric_difference(b))
```

Wypisywanie obu zbiorów

```
a = set([1, 2, 3, 4])
b = set([1, 3, 7, 9])
print(a)
print(b)
```

OOP

- Zmienna (`a = 1`)
- Wartość (1)
- Obiekt – zmienna jest obiektem (`print(id(a))`)

```
def fn():
    return 2

a = fn
print(dir(a))
```

a JEST REFERENCJĄ FUNKCJI

```
a = 1
print(id(a))
a = 2
print(id(a))
```

1862132496
1862132512

```
a = 10
print(dir(a))
```

WSZYSTKO JEST OBIEKTEM

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

OOP - Klasa

- Wszystko jest obiektem
- Klasa to wzorzec, na podstawie którego tworzymy obiekt

```
class Nazwa:  
    dane oraz zachowania
```

OOP - Klasa

```
1 class Point:  
2     pass  
3  
4 p1 = Point()  
5 p2 = Point()  
6 print(type(p1))    <class '__main__.Point'>
```

```
print(type(p2))    <class '__main__.Point'>
```

OOP - Inicjalizer

- Specjalna metoda, która pozwala zainicjalizować obiekt
- Jest uruchamiana automatycznie w momencie tworzenia nowej instancji
- Używamy jej w celu ustawienia początkowego stanu instancji (początkowej wartości atrybutów)

Self – jako koncepcja

- Referencja do instancji (konkretniej instancji)
- Umożliwia odwoływanie do atrybutu wskazanej instancji

```
class Point:  
    def __init__|
```

Metoda `__init__` jest istotą OOP i jest wymagana do tworzenia obiektów.

OOP - Inicjalizer

```
class Point:  
    def __init__(self):  
        self.x = 0  
        self.y = 0
```

Ustawienie inicjalizatora

```
p1 = Point()  
print(p1.x, p1.y)
```

Odwołanie do x,y za pomocą inicjalizatora

Statycznie – w tym przykładzie przy każdej inicjalizacji koordynaty x oraz y mają wartość 0

OOP - Inicjalizer

```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y
```

Ustawienie inicjalizatora

```
p1 = Point(3, 5)  
p2 = Point(2, 7)  
print(p1.x, p1.y)  
print(p2.x, p2.y)
```

Odwołanie do x,y za pomocą inicjalizatora

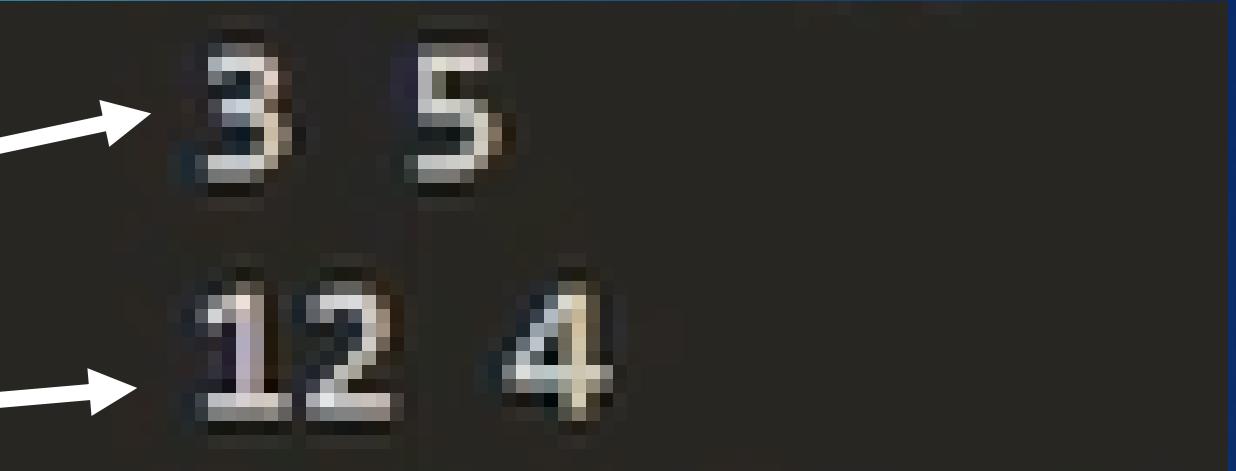
Dynamicznie – w tym przykładzie przy każdej inicjalizacji koordynaty x oraz y muszą zostać podane

OOP - Metody

```
def add(a, b):  
    return a + b  
  
def multiply(a, b):  
    return a * b  
  
def apply(fn, a, b):  
    return fn(a, b)  
  
r1 = apply(add, 4, 5)  
r2 = apply(multiply, 4, 8)  
print(r1, r2)
```

OOP - Metody

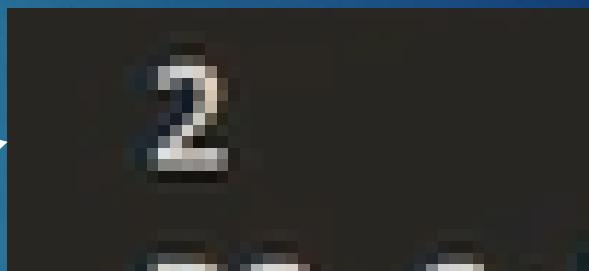
```
class Point:  
    def __init__(self, x=0, y=0):  
        self.x = x  
        self.y = y  
  
    def move_to_new_coords(self, x=0, y=0):  
        self.x = x  
        self.y = y  
  
p1 = Point(3, 5)  
print(p1.x, p1.y)  
p1.move_to_new_coords(12, 4)  
print(p1.x, p1.y)
```



OOP – Atrybuty klasy

```
class Point:  
    points_counter = 0 ← Atrybut klasy (zmienna klasy)  
    def __init__(self, x=0, y=0):  
        self.x = x  
        self.y = y  
        Point.points_counter += 1  
  
    def move_to_new_coords(self, x=0, y=0):  
        self.x = x  
        self.y = y  
  
p1 = Point(3, 5)  
p2 = Point(4, 9)  
print(Point.points_counter)
```

Atrybut klasy (zmienna klasy)



OOP – Dziedziczenie

```
class Widget:  
    def __init__(self, Label):  
        self.label = label  
  
class Button(Widget):  
    def __init__(self, Label, size):  
        super().__init__(label)  
        self.size = size  
  
b = Button('my button', 'large')  
print(b.label, b.size)
```



my button large

OOP – Dziedziczenie

```
class Widget:  
    def __init__(self, label):  
        self.label = label  
  
class Button(Widget):  
    def __init__(self, label, size):  
        super().__init__(label)  
        self.size = size  
  
b = Button('my button', 'large')  
print(b.label, b.size)
```

Klasa Widget

Ustawienie inicjalizatora

Klasa Button dziedziczy po klasie Widget

Ustawienie inicjalizatora

Wywołanie klasy Button

my button large

OOP – Dziedziczenie

```
class Widget:  
    def __init__(self, label):  
        self.label = label  
  
class Button(Widget):  
    def __init__(self, label, size):  
        super().__init__(label)  
        self.size = size  
  
    def handle_click(self):  
        return 'Klik!'  
  
b = Button('my button', 'large')  
print(b.label, b.size)  
print(b.handle_click())
```

my button large
Klik!

Mutowalność

```
a = [1, 2]
a[0] = 1000
print(a)
```

```
a = (1, 2)
a[0] = 1000
print(a)
```

```
my_list = [3, 4]
my_tuple = (1, 2, my_list)
print(my_tuple)
my_tuple[2].append(5)
print(my_tuple)
```



Dekoratory

Dekorator – „coś” co „dekoruje” funkcje – zmienia jej zachowanie. Dekoratorem jest przeważnie inna funkcja (wywołana z argumentami i zwróceniem nowego zachowania)

```
def my_decorator(fn):
    def wrapper():
        print('Początek odliczania')
        fn()
        print('Koniec odliczania')
    return wrapper

def get_5_values():
    for v in range(1,6):
        print(v)

get_5_values_decorated = my_decorator(get_5_values)
get_5_values_decorated()
```

Początek odliczania

1
2
3
4
5

Koniec odliczania

1
2
3
4
5

Dekoratory

```
def my_decorator(fn):
    def wrapper():
        print('Początek odliczania')
        fn()
        print('Koniec odliczania')
    return wrapper

@my_decorator
def get_5_values():
    for v in range(1,6):
        print(v)

# get_5_values_decorated = my_decorator(get_5_values)
# get_5_values_decorated()
get_5_values
```

Początek odliczania

1
2
3
4
5

Koniec odliczania

Generatory

Generator – Funkcja, na której możemy „zrobić pause” – zastosowanie do czasochłonnych operacji

```
def number_generator(end): ← Generator (funkcja)
    n = 1
    while n < end:
        yield n
        n += 1

values = number_generator(1000000) ← Referencja generatora
print(next(values)) ← Pierwsza wartość
print('Cos innego') ← Druga (kolejna) wartość
print(next(values)) ← Kolejna wartość
print(next(values))
print(next(values))
```

Generatory

```
def number_generator(end):
    n = 1
    while n < end:
        yield n
        n += 1

values = number_generator(100)
for v in values:
    print(v)
```

92
93
94
95
96
97
98
99

Break / continue

Przerwanie pętli

```
1 counter = 1
2 while counter <= 10:
3     counter += 1
4     if counter == 5:
5         break
6
7     print(counter)
8
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python breakContinue.py
2
3
4
```

Pomija akcje w ramach jednego przejścia
(nie przerywa pętli – tylko pomija przejście
kiedy warunek jest spełniony)

```
1 counter = 1
2 while counter <= 10:
3     counter += 1
4     if counter == 5:
5         continue
6
7     print(counter)
8
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python breakContinue.py
2
3
4
6
7
8
9
10
11
```

Konwersja typów

```
q = input('Wprowadz liczbę: ') # Wartość zawsze jest traktowana jak string
q = int(q) # Konwersja na Integer analogicznie dla float itd.
print("suma to " + str(q + 1)) # Konwersja na string + konkatenacja + wypisanie.
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python conversionTypes.py
Wprowadz liczbę: 2
suma to 3
```

Formatowanie string

```
c = 12
s = 'W magazynie ' + str(c) + ' szt.'
print(s)

c1 = 13
s1 = 'W magazynie {} szt.'.format(c1)
print(s1)

c2 = 14
c3 = 15
s2 = 'W magazynie {} {} szt.'.format(c2, c3)
print(s2)
```

```
zperQ@gidi MINGW64 /d/wsb_p
$ python formatString.py
W magazynie 12 szt.
W magazynie 13 szt.
W magazynie 14 15 szt.
```

Funkcja Range

```
1  for v in range(1, 10):
2      print(v)
3  print('koniec')
4  for v in range(1, 10, 2):
5      print(v)
6  print('koniec')
7  for v in range(1, 10, 3):
8      print(v)
9  print('koniec')
```

```
zperQ@gidi MINGW64 /d/wsb_p
$ python funRange.py
1
2
3
4
5
6
7
8
9
koniec
1
3
5
7
9
koniec
1
4
7
koniec
```

Funkcja Range

```
for v in range(20):
    print('+' * v)
print('koniec')
```

Funkcje Python - Argumenty

```
def fn(a, b):  
    print(a + b)
```

```
fn(3, 4)
```

```
def fn2(a=0, b=0):  
    print(a + b)
```

💡
`fn2(3)`

```
zperQ@gidi MINGW64 /d/wsb_projects/python  
$ python funkcje/args.py
```

```
7
```

```
3
```

Funkcje Python – Argumenty kluczowe

```
1 def fn(a, b=0, c=0):
2     print("a:", a, "b:", b, "c:", c)
3     print(a + b + c)
4
5
6 fn(3, c=4)
7
8 fn(3, c=4, b=1)
9
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python funkcje/argsKey.py
a: 3 b: 0 c: 4
7
a: 3 b: 1 c: 4
8
```

Funkcje Python – ciąg dalszy Argumenty

```
def fn(a, *args): # zmienna ilość argumentów (tuple) (jedna gwiazdka)
    print(a)
    print(args)

fn('pierwszy', 'drugi', 3, True)

def fn2(a, *args, **dic): # zmienna ilość argumentów (tuple)(jedna gwiazdka) oraz zmienna ilosc argumentow (dictionary) (dwie gwiazdki)
    print(a)
    print(args)
    print(dic)

fn2('pierwszy', 'drugi', 3, True, learn=True, name="Szymon", number=10, )
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python funkcje/argsMore.py
pierwszy
('drugi', 3, True)
pierwszy
('drugi', 3, True)
{'learn': True, 'name': 'Szymon', 'number': 10}
```

Funkcje Python – ciąg dalszy Argumenty

```
1 def fn2(a, *args, **dic):
2     for arg in args:
3         print(arg)
4
5     for item in dic:
6         print(dic[item])
7
8
9 fn2('pierwszy', 'drugi', 3, True, learn=True, name="Szymon", number=10, )
10
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python funkcje/argsMoreI.py
drugi
3
True
True
Szymon
10
```

Funkcje Python - Return

```
1      def fn(x, z):  
2          return x + z, x * z, x - z  
3  
4  
5      result = fn(5, 2)  
6      print(result)  
7      print(result[0])  
8      print(result[2])  
9
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python  
$ python funkcje/return.py  
(7, 10, 3)  
7  
3
```

Funkcje Python – anonimowe (lambda)

```
double = lambda a: a * 2  
  
print(double(2))
```

```
zperQ@gidi MINGW64 /d/wsb_projects/python  
$ python funkcje/Lambda.py  
4
```

Syntax

lambda arguments : expression

Funkcje Python – scope (zasięgi)

```
1  x = 34
2  print('-----zasieg globalny wywołany bezpośrednio przed modyfikacją w funkcji-----')
3  print(x)    # zasięg globalny
4
5
6  def fn():
7      global x
8
9      x += 1
10     z = 2  # na czas życia funkcji
11     y = 3  # na czas życia funkcji
12     print('-----zasieg lokalny wewnątrz funkcji-----')
13     print(y)
14     print(z)
15     print('-----modyfikacja globalnej wewnątrz funkcji-----')
16     print(x)
17
18
19     fn() # na czas życia funkcji
20     print('-----zasieg globalny wywołany bezpośrednio po modyfikacji w funkcji-----')
21     print(x)    # zasięg globalny
22     # print(y)    # zasięg globalny - nie działa
```

Funkcje Python – scope (zasięgi)

```
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python funkcje/Scope.py
-----zasieg lokalny-----
2
3
3
-----zasieg globalny-----
-----modyfikacja globalnej wewnatrz funkcji-----
35
-----zasieg globalny-----
35
```

Moduł

**Definiujemy 1 raz i korzystamy z niego wielokrotnie
– zestaw funkcjonalności – kolekcja**

- Zbiór zmiennych
- Zbiór funkcji

Moduł – importowanie i analiza

```
import sys  
  
print(sys)
```

```
PS C:\python-kurs> python program.py  
<module 'sys' (built-in)>
```

```
import sys  
  
print(dir(sys))
```

```
'_git', '_home', '_xoptions', 'api_version', 'argv', 'base_exec_prefix', 'base_prefix', 'builtin_module_names', 'byteorder', 'call_tracing',  
'callstats', 'copyright', 'displayhook', 'dllhandle', 'dont_write_bytecode', 'exc_info', 'excepthook', 'exec_prefix', 'executable', 'exit',  
'flags', 'float_info', 'float_repr_style', 'get_asyncgen_hooks', 'getCoroutine_wrapper', 'getallocatedblocks', 'getcheckinterval',  
'getdefaultencoding', 'getfilesystemencodeerrors', 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount', 'getsizeof',  
'getswitchinterval', 'gettrace', 'getwindowsversion', 'hash_info', 'hexversion', 'implementation', 'int_info', 'intern', 'is_finalizing',  
'maxsize', 'maxunicode', 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform', 'prefix', 'set_asyncgen_hooks',  
'setCoroutine_wrapper', 'setcheckinterval', 'setprofile', 'setrecursionlimit', 'setswitchinterval', 'settrace', 'stderr', 'stdin',  
'stdout', 'thread_info', 'version', 'version_info', 'warnoptions', 'winver']
```

Moduł – importowanie i analiza

```
import random

print(dir(random))
```

```
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', 'SG_MAGICCONST', 'SystemRandom', 'TWOPI', '_BuiltinMethodType', '_MethodType', '_Sequence', '_Set', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__', '__acos', '__bisect', '__ceil', '__cos', '__e', '__exp', '__inst', '__itertools', '__log', '__pi', '__random', '__sha512', '__sin', '__sqrt', '__test', '__test_generator', '__urandom', '__warn', 'betavariate', 'choice', 'choices', 'expovariate', 'gammavariate', 'gauss', 'getrandbits', 'getstate', 'lognormvariate', 'normalvariate', 'paretovariate', 'randint', 'random', 'randrange', 'sample', 'seed', 'setstate', 'shuffle', 'triangular', 'uniform', 'vonmisesvariate', 'weibullvariate']
```

```
import random

print(random.randint(1,5))
```

```
PS C:\python-kurs> python program.py
5
PS C:\python-kurs> python program.py
5
PS C:\python-kurs> python program.py
4
PS C:\python-kurs> python program.py
5
```

Moduł – importowanie i analiza

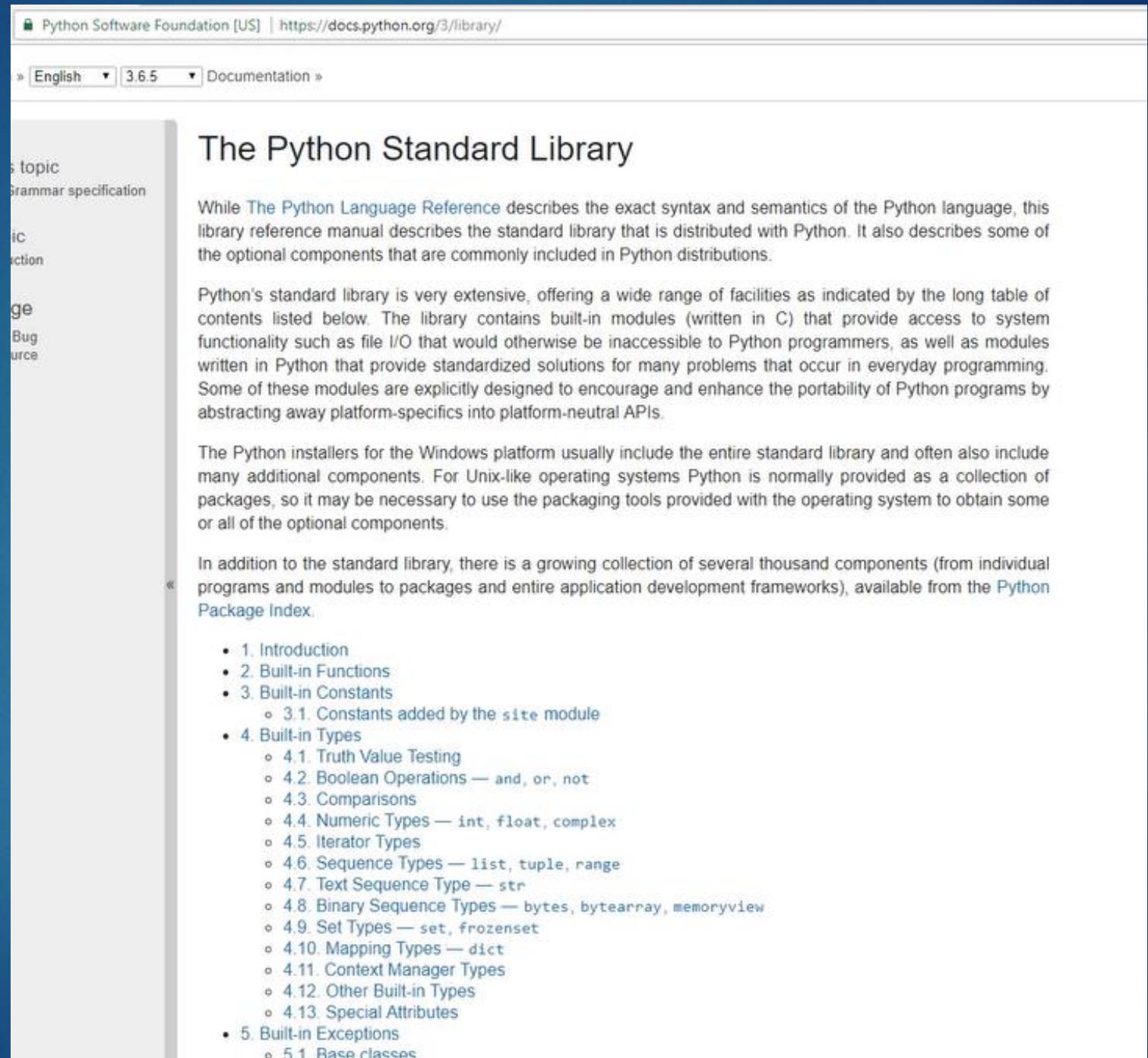
```
from random import randint
from sys import version

print(randint(1,5))
print(version)
```



```
PS C:\python-kurs> python program.py
5
3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 16:07:46) [MSC v.1900 32 bit (Intel)]
```

Moduł



The screenshot shows a web browser displaying the Python Standard Library documentation. The URL in the address bar is <https://docs.python.org/3/library/>. The page title is "The Python Standard Library". The left sidebar contains a table of contents with sections like "1. Introduction", "2. Built-in Functions", "3. Built-in Constants", "4. Built-in Types", "5. Built-in Exceptions", and "6. Built-in Functions". The main content area describes the standard library and its modules. A green vertical bar is visible on the right side of the slide.

Python Software Foundation [US] | <https://docs.python.org/3/library/>

» English ▾ 3.6.5 ▾ Documentation »

The Python Standard Library

While [The Python Language Reference](#) describes the exact syntax and semantics of the Python language, this library reference manual describes the standard library that is distributed with Python. It also describes some of the optional components that are commonly included in Python distributions.

Python's standard library is very extensive, offering a wide range of facilities as indicated by the long table of contents listed below. The library contains built-in modules (written in C) that provide access to system functionality such as file I/O that would otherwise be inaccessible to Python programmers, as well as modules written in Python that provide standardized solutions for many problems that occur in everyday programming. Some of these modules are explicitly designed to encourage and enhance the portability of Python programs by abstracting away platform-specifics into platform-neutral APIs.

The Python installers for the Windows platform usually include the entire standard library and often also include many additional components. For Unix-like operating systems Python is normally provided as a collection of packages, so it may be necessary to use the packaging tools provided with the operating system to obtain some or all of the optional components.

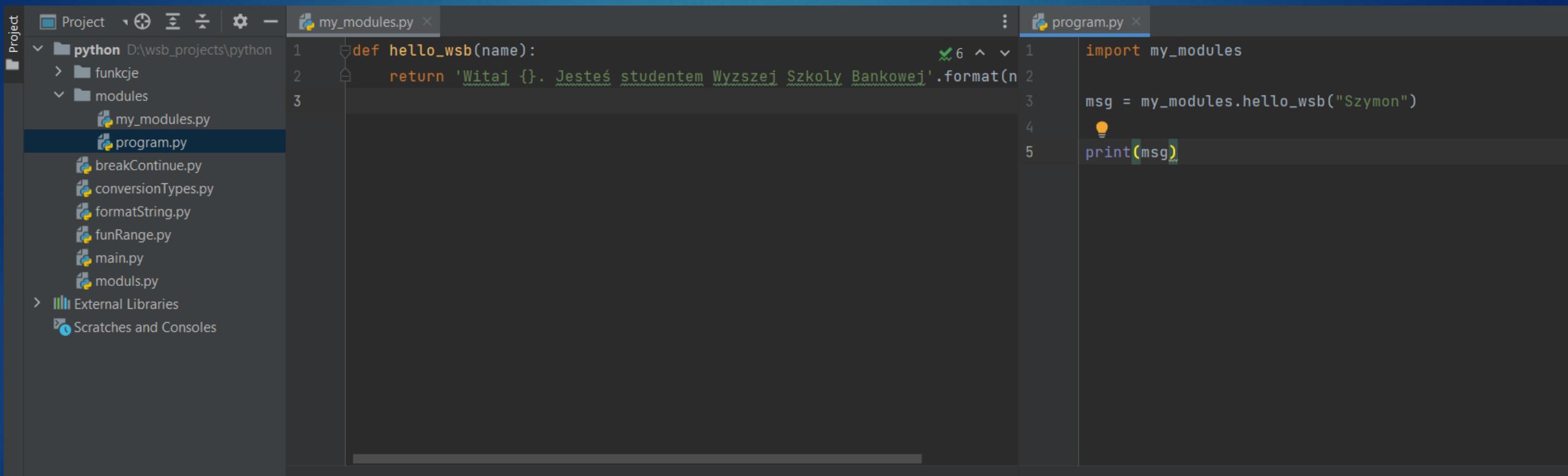
In addition to the standard library, there is a growing collection of several thousand components (from individual programs and modules to packages and entire application development frameworks), available from the [Python Package Index](#).

- 1. Introduction
- 2. Built-in Functions
- 3. Built-in Constants
 - 3.1. Constants added by the `site` module
- 4. Built-in Types
 - 4.1. Truth Value Testing
 - 4.2. Boolean Operations — `and`, `or`, `not`
 - 4.3. Comparisons
 - 4.4. Numeric Types — `int`, `float`, `complex`
 - 4.5. Iterator Types
 - 4.6. Sequence Types — `list`, `tuple`, `range`
 - 4.7. Text Sequence Type — `str`
 - 4.8. Binary Sequence Types — `bytes`, `bytearray`, `memoryview`
 - 4.9. Set Types — `set`, `frozenset`
 - 4.10. Mapping Types — `dict`
 - 4.11. Context Manager Types
 - 4.12. Other Built-in Types
 - 4.13. Special Attributes
- 5. Built-in Exceptions
 - 5.1. Base classes

Moduł

<https://docs.python.org/3/library/>

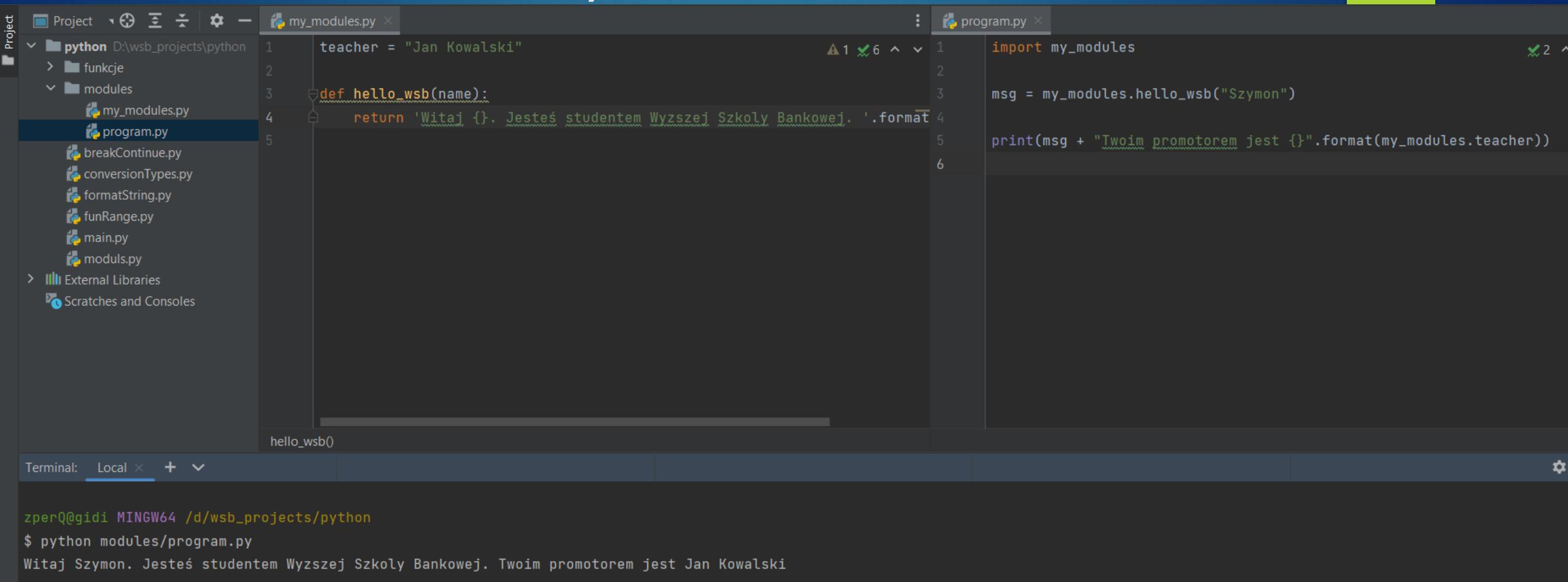
Moduł – własny moduł



```
Project my_modules.py x program.py x
python D:\wsb_projects\python
  ↘ funkcje
  ↘ modules
    ↘ my_modules.py
    ↘ program.py
    breakContinue.py
    conversionTypes.py
    formatString.py
    funRange.py
    main.py
    modulus.py
External Libraries
Scratches and Consoles

Terminal: Local x + ▾
35
zperQ@gidi MINGW64 /d/wsb_projects/python
$ python modules/program.py
Witaj Szymon. Jesteś studentem Wyzszej Szkoły Bankowej
```

Moduł – własny moduł



The screenshot shows a Python development environment with two open files:

- my_modules.py**:

```
1 teacher = "Jan Kowalski"
2
3 def hello_wsb(name):
4     return 'Witaj {}. Jesteś studentem Wyzszej Szkoly Bankowej. '.format
5
```
- program.py**:

```
1 import my_modules
2
3 msg = my_modules.hello_wsb("Szymon")
4
5 print(msg + "Twoim promotorem jest {}".format(my_modules.teacher))
```

In the terminal below, the command `$ python modules/program.py` is run, resulting in the output:
`Witaj Szymon. Jesteś studentem Wyzszej Szkoly Bankowej. Twoim promotorem jest Jan Kowalski`

Moduł – własny moduł

```
import my_module
```

```
print(my_module.x)
```

```
from my_module import my_function
```

```
from my_module import my_function
msg = my_function('Ksawery')
print(msg)
print(my_module.x)
```

```
from my_module import *
```

```
from my_module import *
msg = my_function('Ksawery')
print(msg)
print(x)
```

Zadanie 1

Wczytywanie danych

- Imię
- Nazwisko
- Wiek
- Semestr
- Zwrócić po podaniu wszystkich wartości

Obliczanie Delty

- Delta
- X₁,X₂
- Warunek dla Delty = 0

Kalkulator

- Dodawanie
- Odejmowanie
- Mnożenie
- Dzielenie

Wywołanie przez rekurencję po zakończeniu obliczenia

Zadanie 2

Wypisać całą tabliczkę mnożenia

Przykład formatu

$$1 * 1 = 1$$

$$10 * 10 = 100$$

Zadanie 3

Zbudować klasę Pojazd

- ma zawierać cechy dla wszystkich pojazdów

Zbudować klasę samochód osobowy (dziedziczenie z klasy pojazd podstawowych cech)

- ma zawierać cechy , które są specyficzne tylko dla samochodu osobowego

Zbudować klasę autobus (dziedziczenie z klasy pojazd podstawowych cech)

- ma zawierać cechy , które są specyficzne tylko dla autobusu

Zadanie 4

Zbudować klasę człowiek

- ma zawierać informacje
 - Płci
 - Wieku
 - Imię
 - Nazwisko

Powołać małżeństwo (dwa obiekty)

Połączyć z poprzednim zadaniem

WYNIK: Żona {obiekt żona} kupiła pojazd {obiekt pojazd}

Błędy i wyjątki

```
print(x)
```

```
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

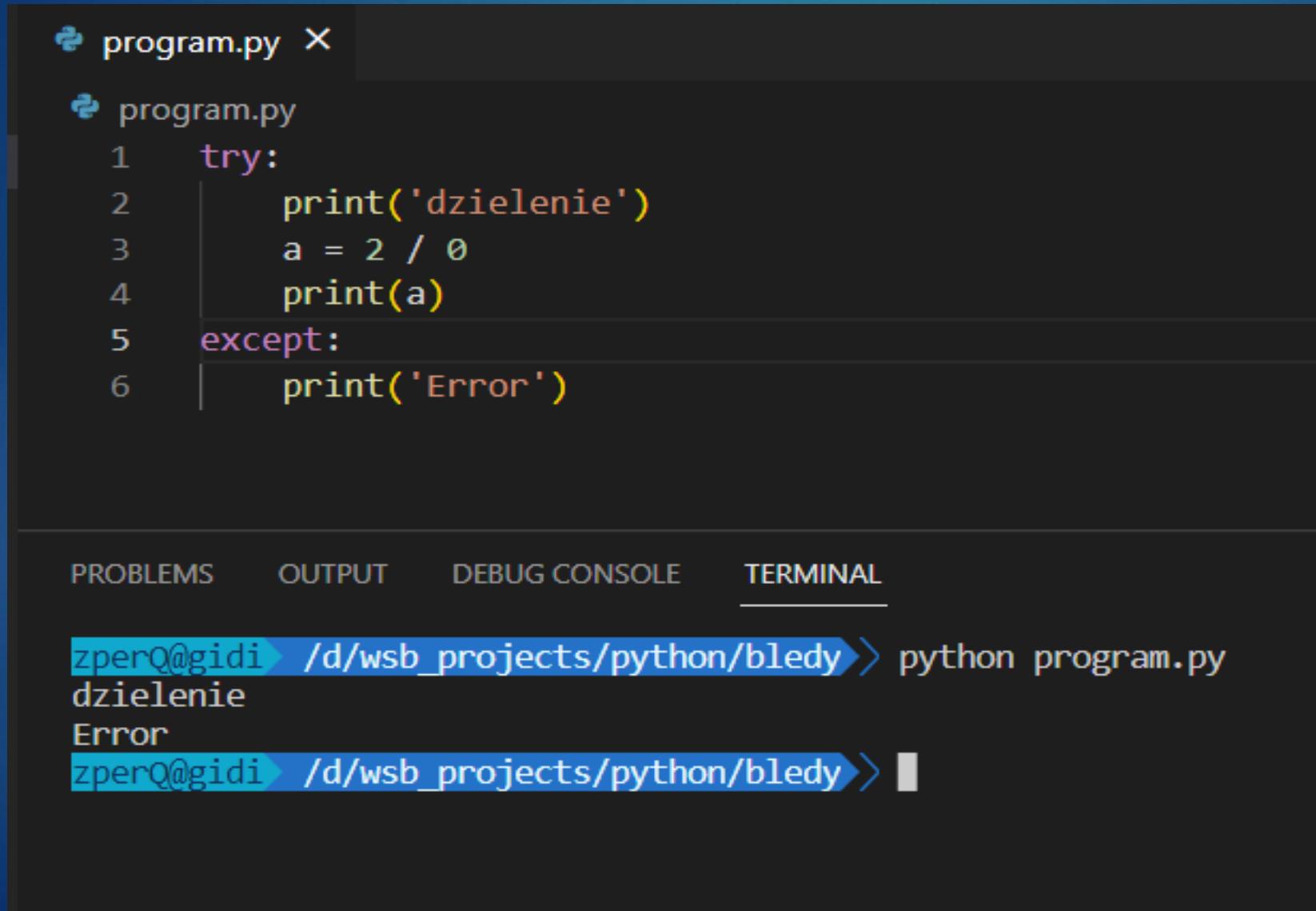
```
print(1 + 'x')
```

```
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    print(1 + 'x')
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
print(2 / 0)
```

```
Traceback (most recent call last):
  File "program.py", line 1, in <module>
    print(2 / 0)
ZeroDivisionError: division by zero
```

Błędy i wyjątki



The screenshot shows a code editor window with a dark theme. At the top, there are two tabs: "program.py" (the active tab) and another "program.py". The code in the editor is:

```
1  try:
2      print('dzielenie')
3      a = 2 / 0
4      print(a)
5  except:
6      print('Error')
```

Below the code editor, there is a navigation bar with four tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is currently selected. The terminal window displays the following output:

```
zperQ@gidi ~ % cd /d/wsb_projects/python/bledy
zperQ@gidi ~/d/wsb_projects/python/bledy % python program.py
dzielenie
Error
zperQ@gidi ~/d/wsb_projects/python/bledy %
```

Błędy i wyjątki

program.py

```
1  try:
2      print('dzielenie')
3      a = 2 / 1
4      print(a)
5  except:
6      print('Error')
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

```
zperQ@gidi > /d/wsb_projects/python/bledy > python program.py
dzielenie
2.0
zperQ@gidi > /d/wsb_projects/python/bledy > |
```

Błędy i wyjątki

program.py

```
1  try:
2      print('dzielenie')
3      a = 2 / 'x'
4      print(a)
5  except TypeError:
6      print('Error - niewlasciwy typ')
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

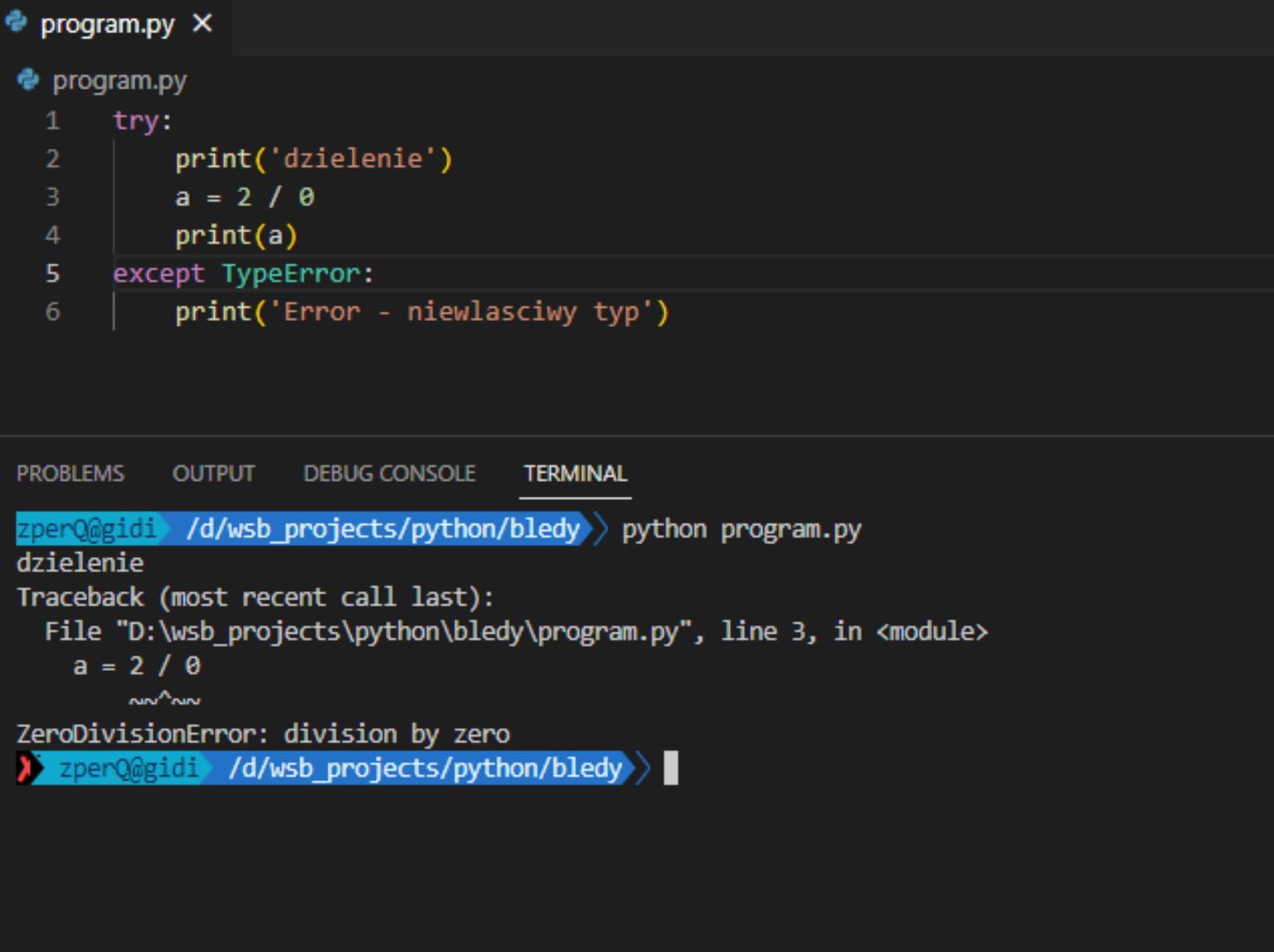
```
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
```

```
dzielenie
```

```
Error - niewlasciwy typ
```

```
zperQ@gidi ~ /d/wsb_projects/python/bledy > 
```

Błędy i wyjątki



The screenshot shows a code editor window with a dark theme. At the top, there are two tabs: "program.py X" and "program.py". The "program.py" tab is active, displaying the following Python code:

```
program.py
try:
    print('dzielenie')
    a = 2 / 0
    print(a)
except TypeError:
    print('Error - niewlasciwy typ')
```

Below the code editor is a navigation bar with four tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab is selected, indicated by an underline. The terminal window displays the following output:

```
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
dzielenie
Traceback (most recent call last):
  File "D:\wsb_projects\python\bledy\program.py", line 3, in <module>
    a = 2 / 0
           ^
ZeroDivisionError: division by zero
zperQ@gidi ~ /d/wsb_projects/python/bledy >
```

Błędy i wyjątki

```
python program.py > ...
1  try:
2      print('dzielenie')
3      a = 2 / 0
4      print(a)
5  except ZeroDivisionError:
6      print('Error -Dzielenie przez zero')
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
dzielenie
Error -Dzielenie przez zero
zperQ@gidi ~ /d/wsb_projects/python/bledy > |
```

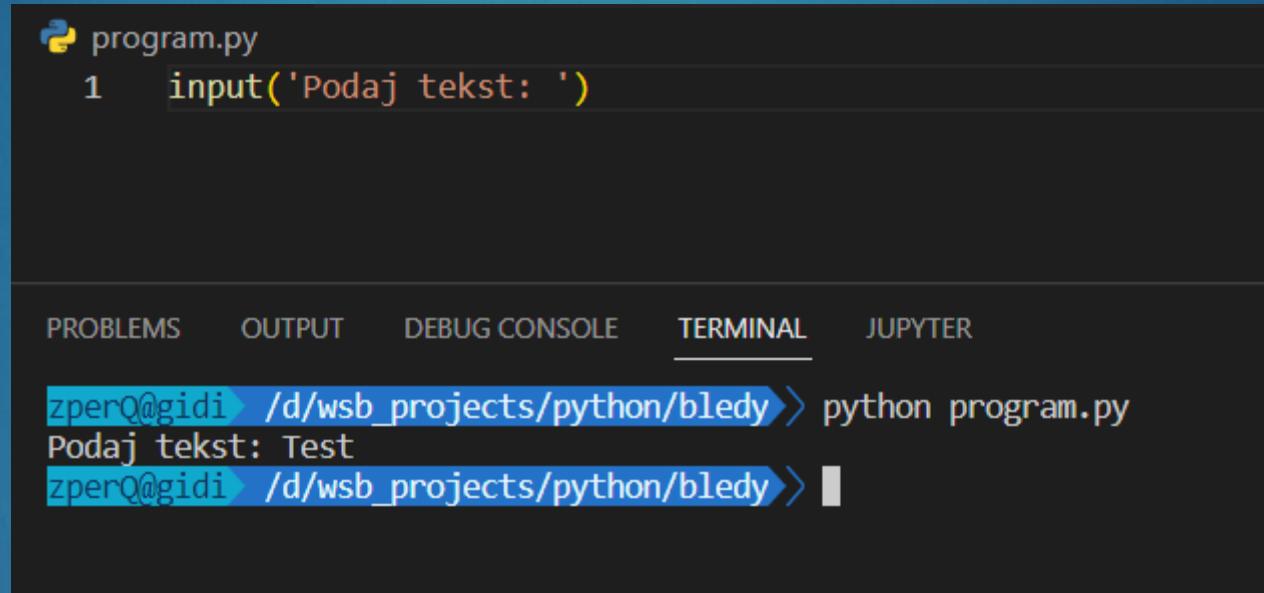
Instrukcja Raise

```
py program.py > ...
1 def divide(a,b):
2     if b == 0:
3         raise RuntimeError('Blad ogolny')
4
5     return a // b
6
7 try:
8     r = divide(4,0)
9     print(r)
10 except RuntimeError:
11     print('RuntimeError')
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

```
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
RuntimeError
zperQ@gidi ~ /d/wsb_projects/python/bledy > |
```

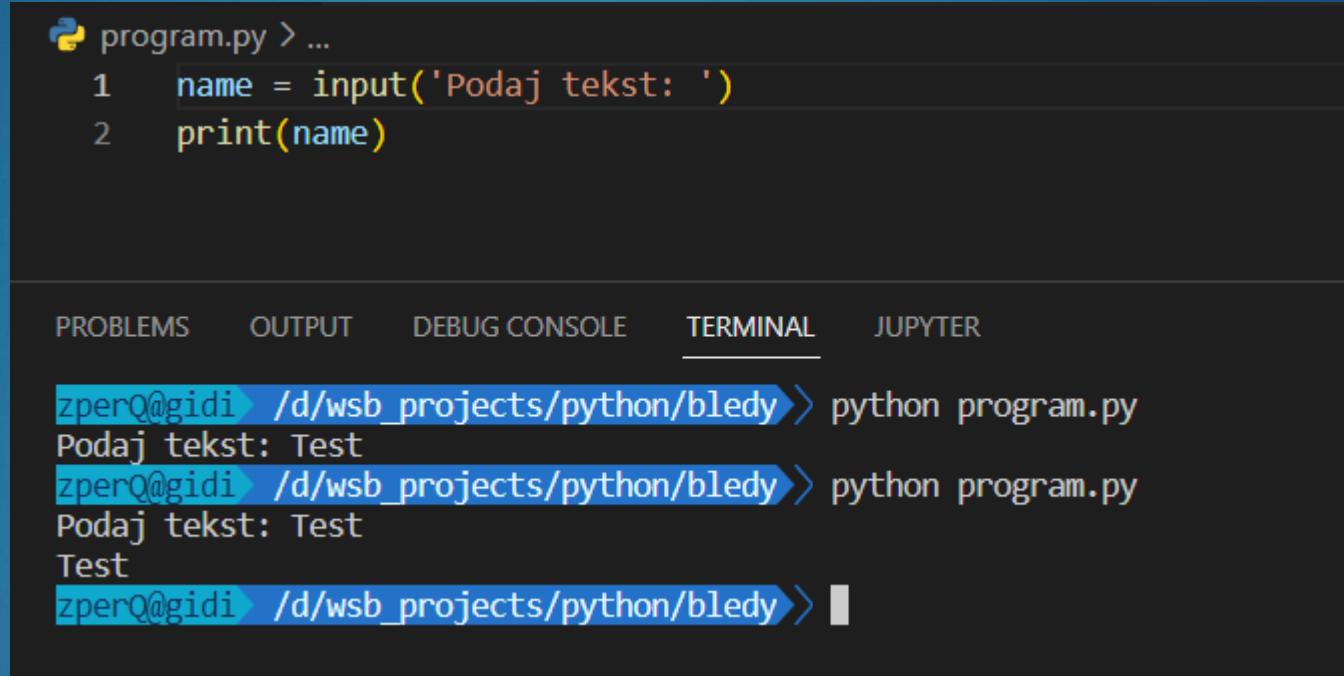
Input w konsoli



A screenshot of a code editor showing a Python script named `program.py`. The code contains a single line: `1 input('Podaj tekst: ')`. Below the code editor is a terminal window with the following output:

```
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
Podaj tekst: Test
zperQ@gidi ~ /d/wsb_projects/python/bledy >
```

Input w konsoli



The screenshot shows a terminal window with the following content:

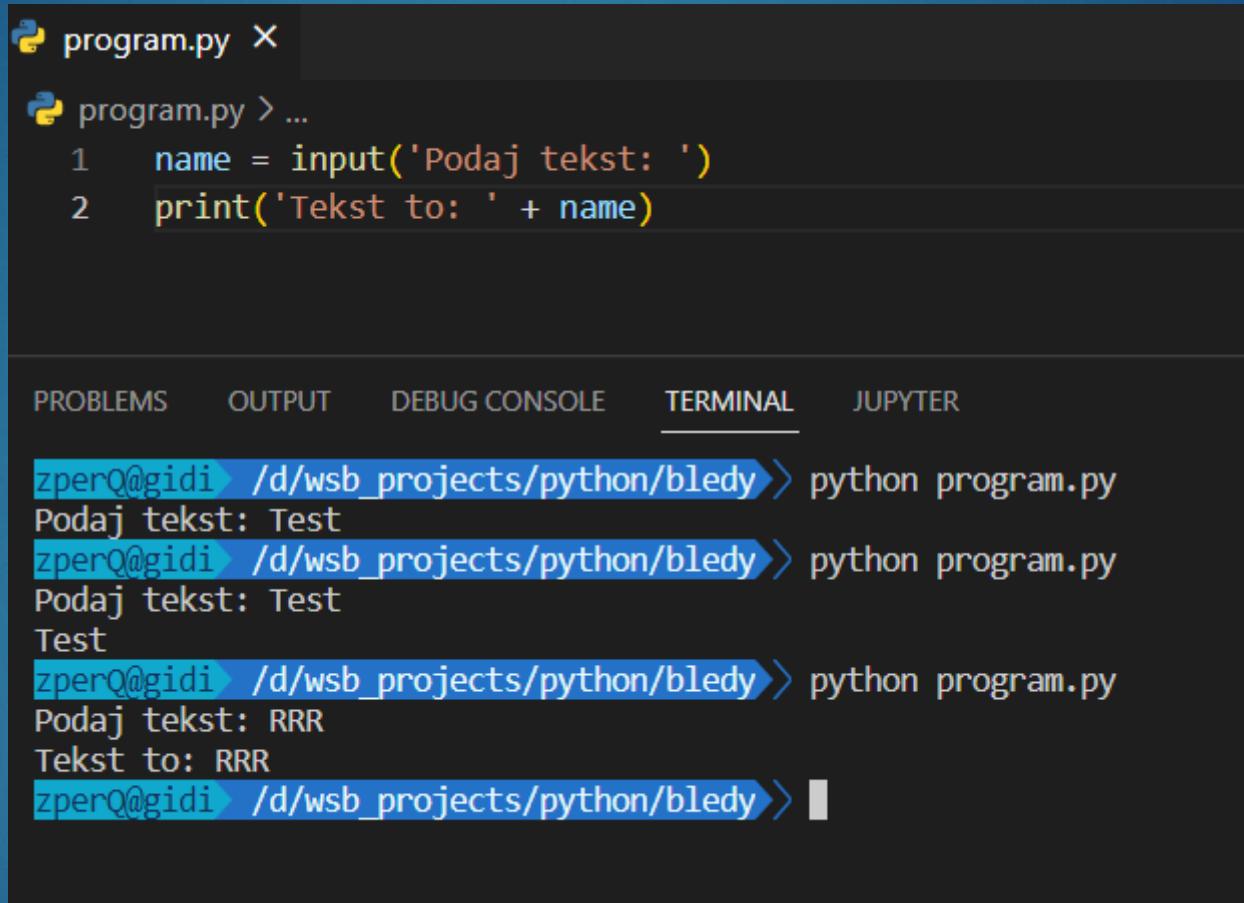
```
program.py > ...
1 name = input('Podaj tekst: ')
2 print(name)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

zperQ@gidi > /d/wsb_projects/python/bledy > python program.py
Podaj tekst: Test
zperQ@gidi > /d/wsb_projects/python/bledy > python program.py
Podaj tekst: Test
Test
zperQ@gidi > /d/wsb_projects/python/bledy > |
```

The terminal window has tabs at the top: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is underlined), and JUPYTER. The command `python program.py` is run twice, each time prompting for input with the message "Podaj tekst: ". The user types "Test" both times, and the output "Test" is displayed. The final prompt "`zperQ@gidi > /d/wsb_projects/python/bledy > |`" indicates the terminal is ready for the next command.

Input w konsoli



```
program.py X
program.py > ...
1 name = input('Podaj tekst: ')
2 print('Tekst to: ' + name)

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
Podaj tekst: Test
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
Podaj tekst: Test
Test
zperQ@gidi ~ /d/wsb_projects/python/bledy > python program.py
Podaj tekst: RRR
Tekst to: RRR
zperQ@gidi ~ /d/wsb_projects/python/bledy > |
```

Realizacja własnych projektów

Dziękuję za uwagę!