Programowanie Python 1

(CP1S02005)

Politechnika Białostocka - Wydział Elektryczny

Cyfryzacja przemysłu, sem. II, studia stacjonarne I stopnia

Rok akademicki 2024/2025

Wykład nr 1 (04.03.2025)

dr inż. Jarosław Forenc

Dane podstawowe

- dr inż. Jarosław Forenc
- Politechnika Białostocka, Wydział Elektryczny,
  Katedra Elektrotechniki, Energoelektroniki i Elektroenergetyki
  ul. Wiejska 45D, 15-351 Białystok
  WE-204
- e-mail: j.forenc@pb.edu.pl
- tel. (0-85) 746-93-97
- http://jforenc.prv.pl
  - Dydaktyka - slajdy z wykładu

- konsultacje:
  - poniedziałek, 08:30-10:00, WE-204
  - środa, 08:30-10:00, WE-204

Program wykładu

1. Ogólna struktura programu w języku Python. Typy danych, słowa kluczowe i nazwy zmiennych. Operatory i wyrażenia arytmetyczne, priorytet operatorów, funkcje matematyczne. Komentarze. Operatory porównania i logiczne, wyrażenia logiczne. Instrukcja warunkowa if/elif/else.
2. Instrukcje iteracyjne for i while, funkcja range, instrukcje break i continue.
3. Ciągi tekstowe (typ napisowy), operacje na napisach (metody). Listy, krotki, słowniki i zbiory.
4. Funkcje, definiowanie funkcji, argumenty i parametry funkcji, zasięg zmiennych.
5. Operacje na plikach, wyjątki.
6. Elementy programowania obiektowego, obiekty, klasy, dziedziczenie.
7. Biblioteka standardowa, biblioteki NumPy, Matplotlib, SciPy. Środowisko Jupyter Notebook.
8. Sprawdzian zaliczeniowy.

Literatura

1. Sarbicki G., „Python. Kurs dla nauczycieli i studentów. Wydanie II”. Helion, Gliwice, 2022.
2. Matthes E., „Python. Instrukcje dla programisty. Wydanie III”. Helion, Gliwice, 2023.
3. Sweigart A., „Automatyzacja nudnych zadań z Pythonem. Nauka programowania. Wydanie II”. Helion, Gliwice, 2021.
4. McKinney W., „Python w analizie danych. Przetwarzanie danych za pomocą pakietów pandas i NumPy oraz środowiska Jupyter. Wydanie III”. Helion, Gliwice, 2023.
5. Miles R., „Python. Zacznij programować!”. Helion, Gliwice 2018.
6. https://docs.python.org/pl/3/ - Python, dokumentacja.

Efekty uczenia się

Podstawę do zaliczenia przedmiotu (uzyskanie punktów ECTS) stanowi stwierdzenie, że każdy z założonych efektów uczenia się został osiągnięty.

Student, który zaliczył przedmiot, zna i rozumie:

EU1	podstawowe mechanizmy języka Python umożliwiające programowanie strukturalne i obiektowe w tym języku
EU2	podstawowe konstrukcje programistyczne stosowane w języku Python

- Szczegóły: http://jforenc.prv.pl/dydaktyka.html lub system USOS

Zaliczenie wykładu

- Zaliczenie wykładu odbędzie się na podstawie wyników sprawdzianu pisemnego
- Sprawdzian odbędzie się na ostatnim wykładzie w semestrze
- Za sprawdzian można otrzymać od 0 do 100 pkt.
- Prowadzący zajęcia może przyznawać dodatkowe punkty za aktywność na wykładzie
- Ocena końcowa wystawiana jest na podstawie otrzymanych punktów:

Punkty	Ocena	Punkty	Ocena
91 - 100	5,0	61 - 70	3,5
81 - 90	4,5	51 - 60	3,0
71 - 80	4,0	0 - 50	2,0

Terminy zajęć

- Wykład nr 1: 04.03.2025
- Wykład nr 2: 18.03.2025
- Wykład nr 3: 01.04.2025
- Wykład nr 4: 15.04.2025
- Wykład nr 5: 29.04.2025
- Wykład nr 6: 13.05.2025
- Wykład nr 7: 27.05.2025
- Wykład nr 8: 10.06.2025 (1h, 11:15-12:00, zaliczenie)

Plan wykładu nr 1

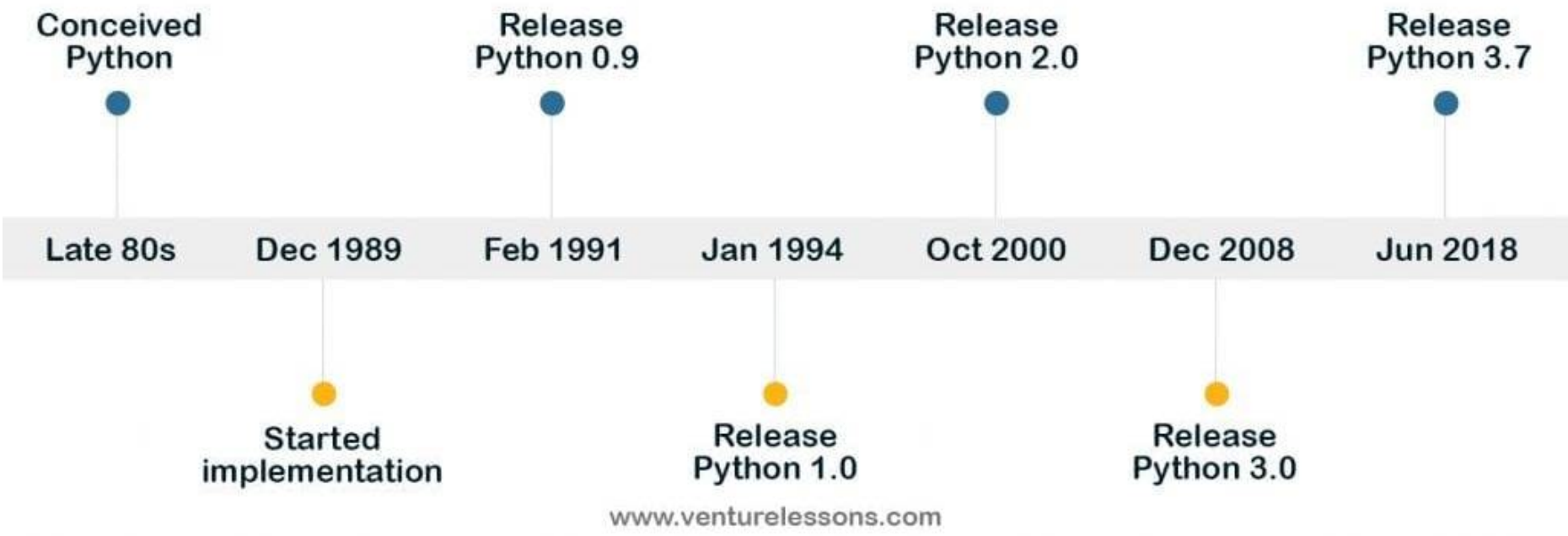
- Historia języka Python, podstawowe informacje
- Pierwszy program
- Zmienne, słowa kluczowe, typy zmiennych
- Funkcje print() i input()
- Operatory arytmetyczne, priorytet operatorów
- Stałe i funkcje matematyczne
- Komentarze
- Operatory porównania i logiczne
- Instrukcje if, if-else, if-elif-else
- Operator warunkowy

Python - historia

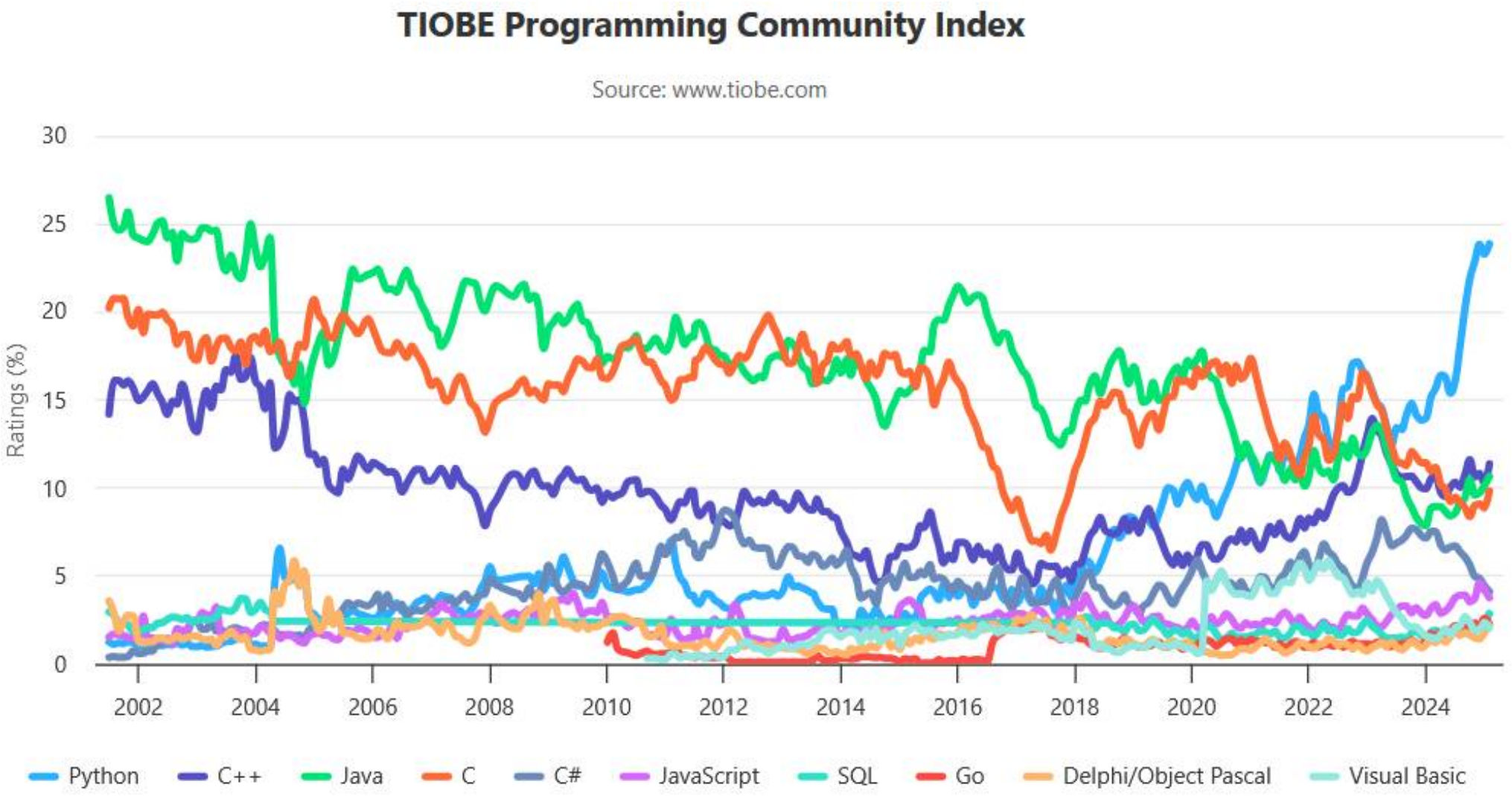
- Twórcą Pythona jest holenderski programista Guido van Rossum
- Stworzył go we wczesnych latach 90-tych jako następcę języka ABC, opracowanego w Centrum Wiskunde & Informatica (CWI, Amsterdam)
- Nazwa języka pochodzi od serialu „Latający cyrk Monty Pythona”
- Pierwsza wersja języka został przedstawiona publicznie w lutym 1991 roku (wersja 0.9.0)
- Wersja 1.2 była ostatnią wydaną przez CWI
- Od 1995 roku Van Rossum kontynuował pracę nad Pythonem w Corporation for National Research Initiatives (CNRI) w Reston w Wirginii, gdzie wydał kilka wersji Pythona, do 1.6 włącznie
- W 2000 roku van Rossum wraz z zespołem przenieśli się do BeOpen.com, by założyć zespół BeOpen PythonLabs - pierwszą i jedyną wersją wydaną przez BeOpen.com był Python 2.0

Python - historia

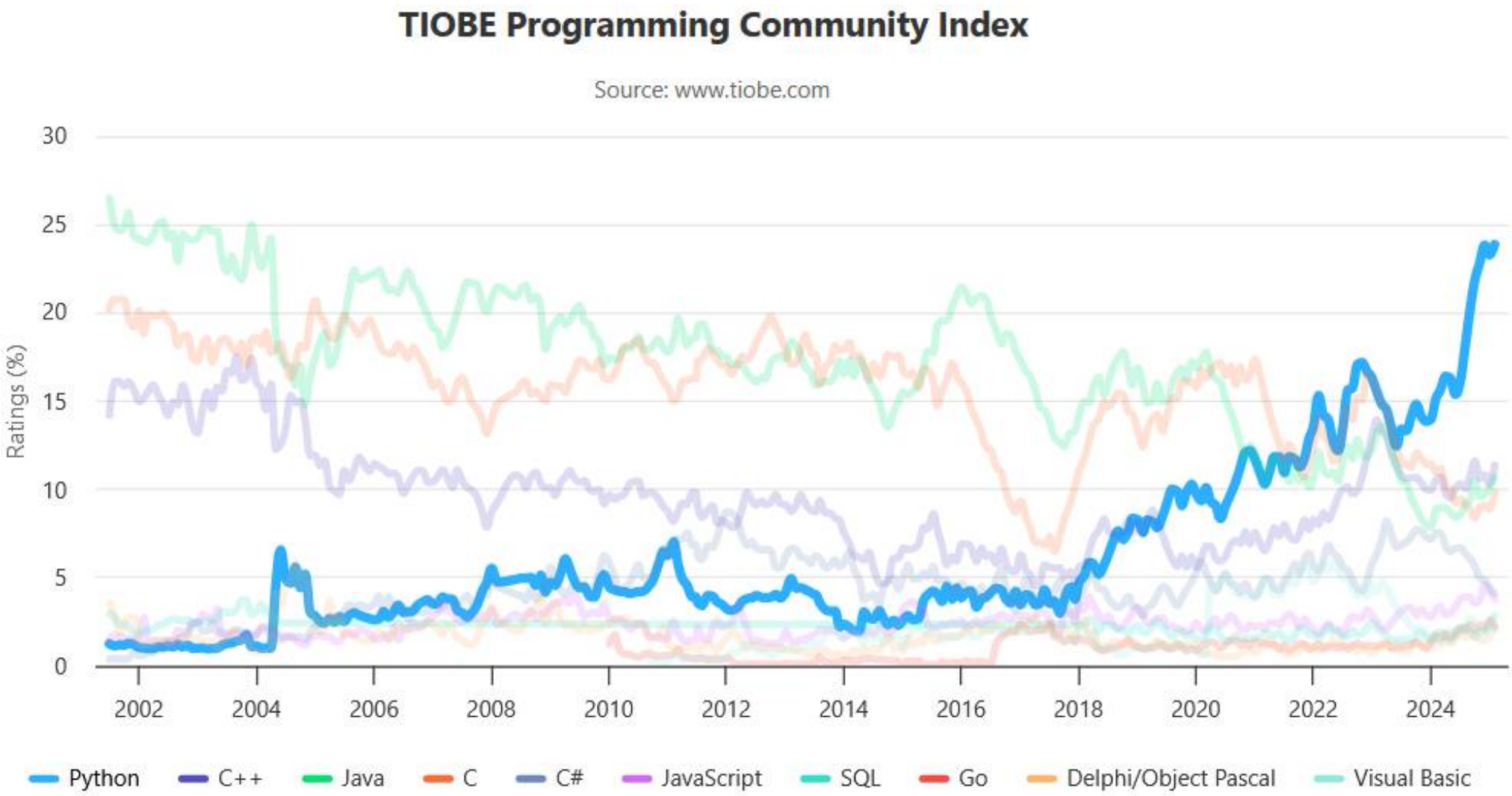
- Po wydaniu Pythona 2.0 przez BeOpen.com Guido van Rossum i inni programiści z PythonLabs przeszli do Digital Creations
- Cała własność intelektualna dodana od tego momentu, począwszy od Pythona 2.1 jest własnością Python Software Foundation (PSF), niedochodowej organizacji (non-profit)



Python - TIOBE Programming Community Index



Python - TIOBE Programming Community Index

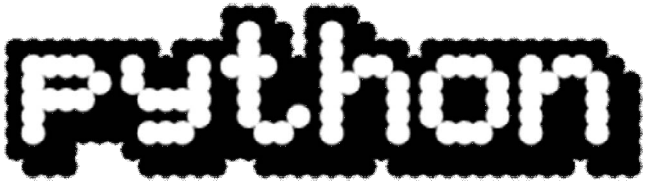


Python - TIOBE Programming Community Index

Feb 2025	Feb 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.88%	+8.72%
2	3	↑	C++	11.37%	+0.84%
3	4	↑	Java	10.66%	+1.79%
4	2	↓	C	9.84%	-1.14%
5	5		C#	4.12%	-3.41%
6	6		JavaScript	3.78%	+0.61%
7	7		SQL	2.87%	+1.04%
8	8		Go	2.26%	+0.53%
9	12	↑	Delphi/Object Pascal	2.18%	+0.78%
10	9	↓	Visual Basic	2.04%	+0.52%

Python - informacje

- Strona internetowa: http://python.org
- Aktualna wersja stabilna: 3.13.2 (04.02.2025)
- Systemy operacyjne: Windows, macOS, Linux, Android, Unix, BSD i inne
- Implementacje: CPython, PyPy, Stackless Python, MicroPython, CircuitPython, IronPython, Jython
- Logo:



1997 - 2006 2006 - now

Python - cechy charakterystyczne

- Czytelność i prostota składni - łatwy do nauki i zrozumienia, co przekłada się na szybszy rozwój i utrzymanie kodu
- Dynamiczne typowanie - nie trzeba deklarować typu zmiennej, co pozwala na szybszy rozwój i bardziej elastyczny kod
- Obszerna biblioteka standardowa - zapewnia dostęp do wielu przydatnych funkcji i modułów, co ułatwia rozwijanie aplikacji bez konieczności pisania kodu od zera
- Wieloplatformowość - kod napisany w Pythonie może być uruchamiany na różnych systemach operacyjnych, takich jak Windows, macOS i Linux
- Wszechstronność - znajduje zastosowanie w różnych dziedzinach, takich jak web development, data science, machine learning, sztuczna inteligencja, analiza danych, automatyzacja zadań

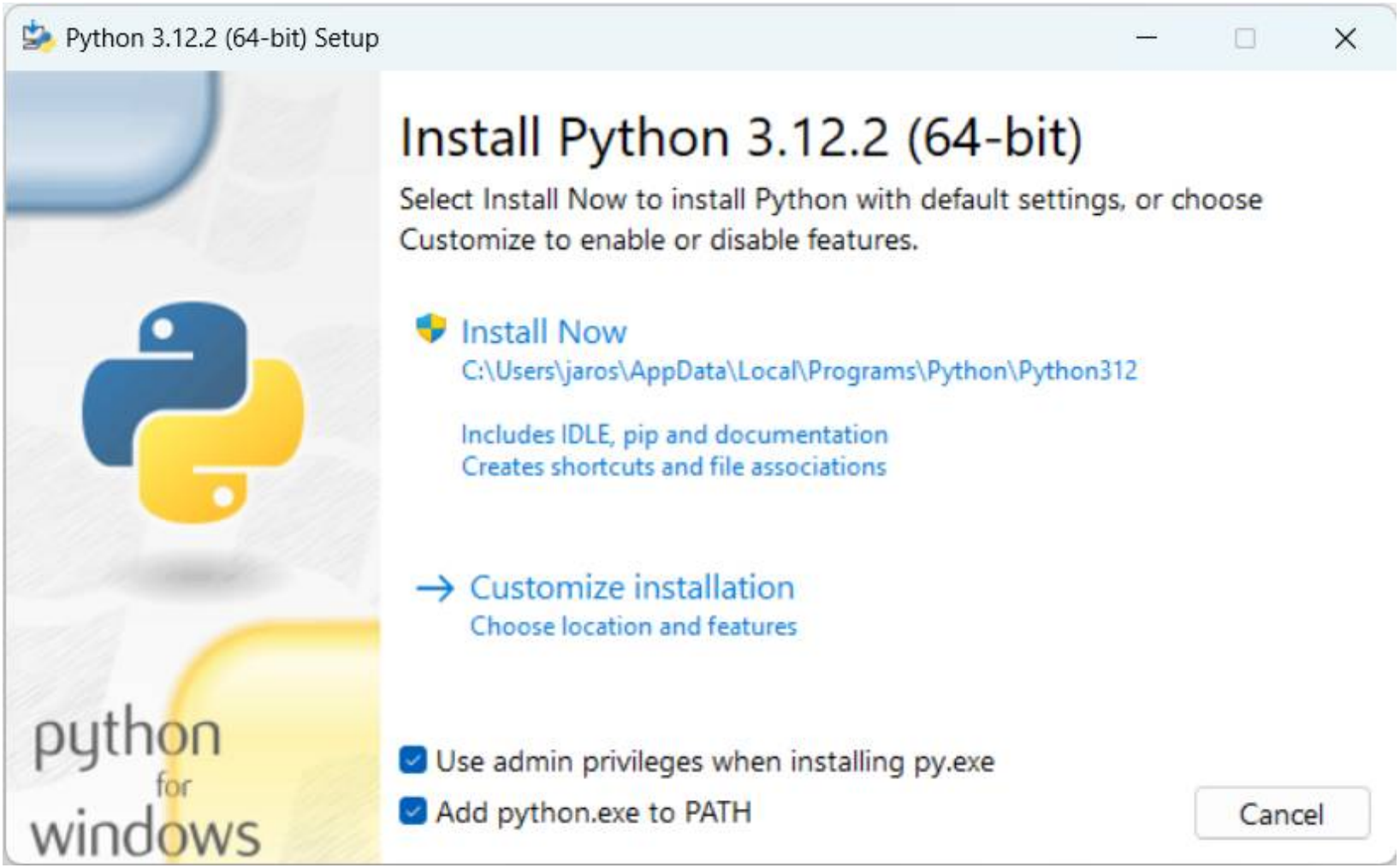
Python - pierwszy program

- Niesformatowany plik tekstowy o rozszerzeniu .py
- Kod najprostszego programu:

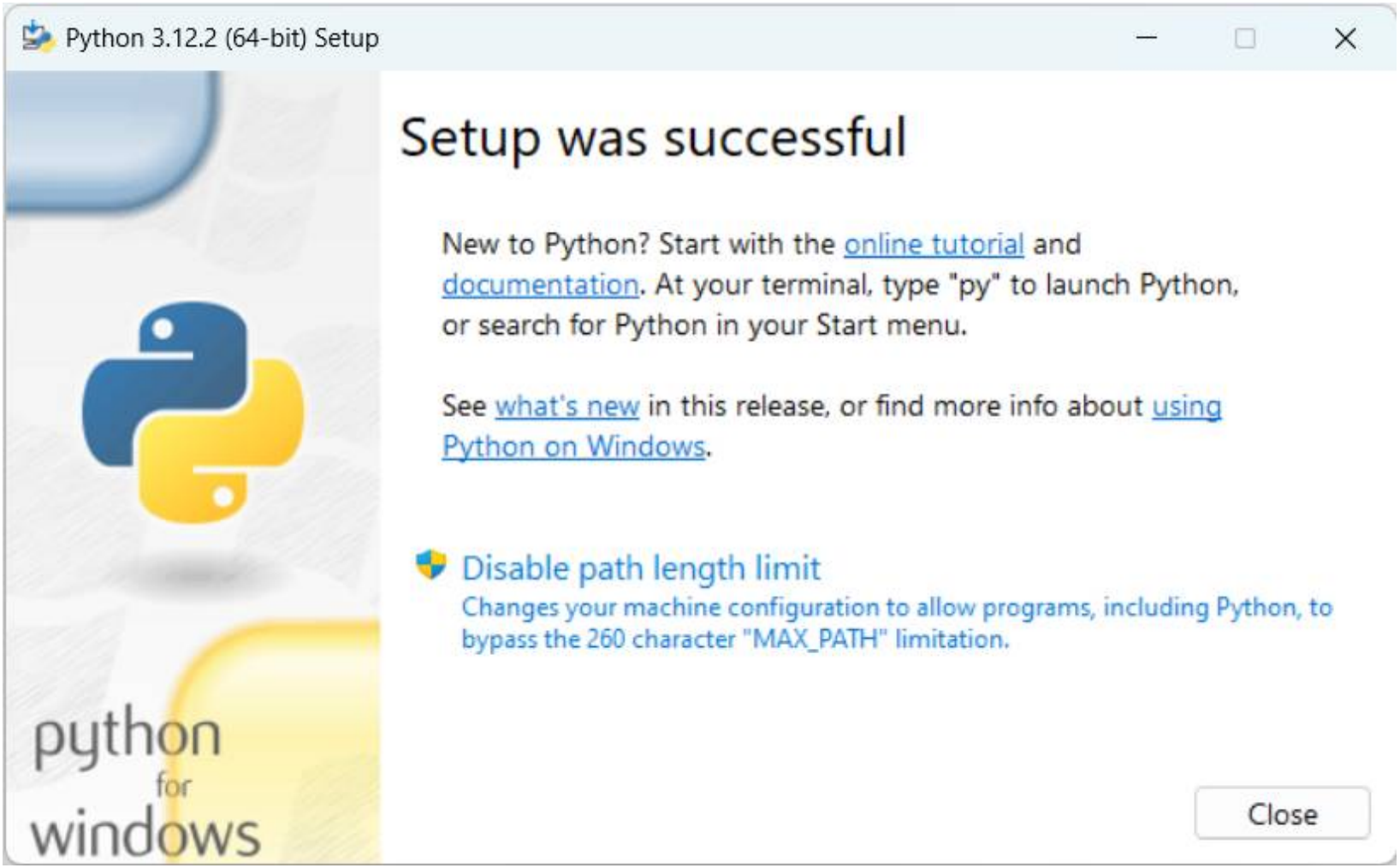
```
print("Witaj swiecie")
```

- Funkcja print() służy do wyświetlania tekstu lub innych danych na standardowym wyjściu (zazwyczaj konsola lub terminal)
- Uruchomienie programu wymaga zainstalowania Pythona
  - https://www.python.org/downloads/windows/
  - Python 3.13.2: https://www.python.org/downloads/release/python-3132/
  - Windows installer (64-bit) - plik 27.3 MB

Python - instalacja

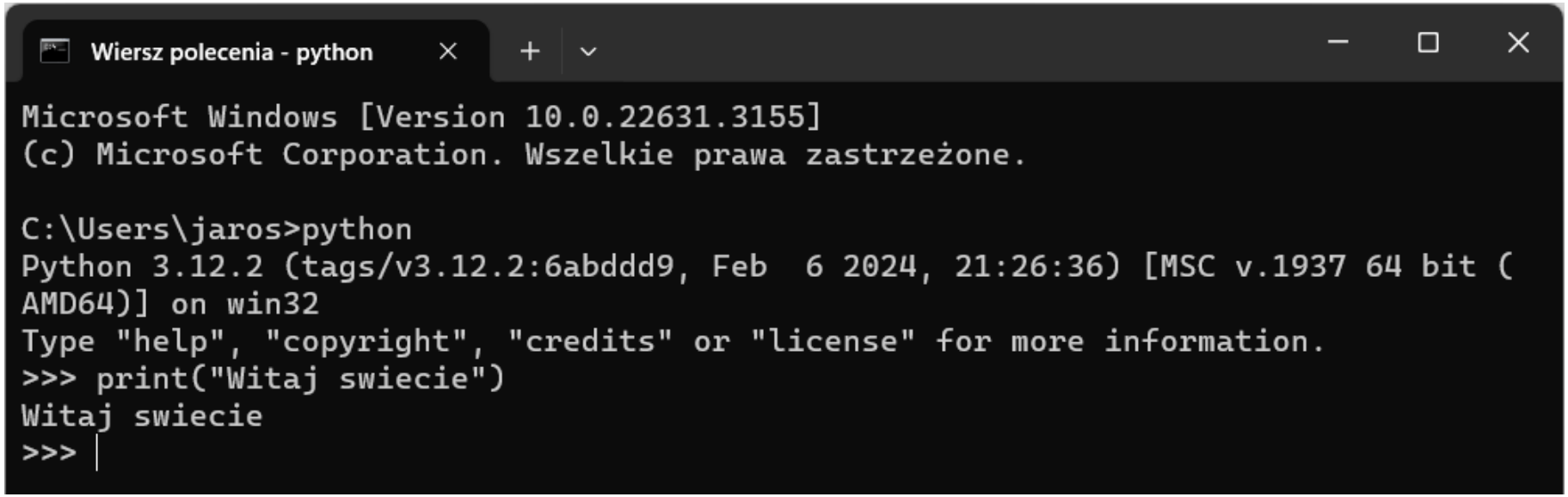


Python - instalacja



Python - wykonanie kodu w Wierszu polecenia

- W oknie Wiersza polecenia wprowadzamy polecenie python uruchamiające interpreter pythona
- Po znaku zachęty (>>>) wprowadzamy jedną linię kodu do wykonania
- Wynik wykonania kodu wyświetla się w tym samym oknie

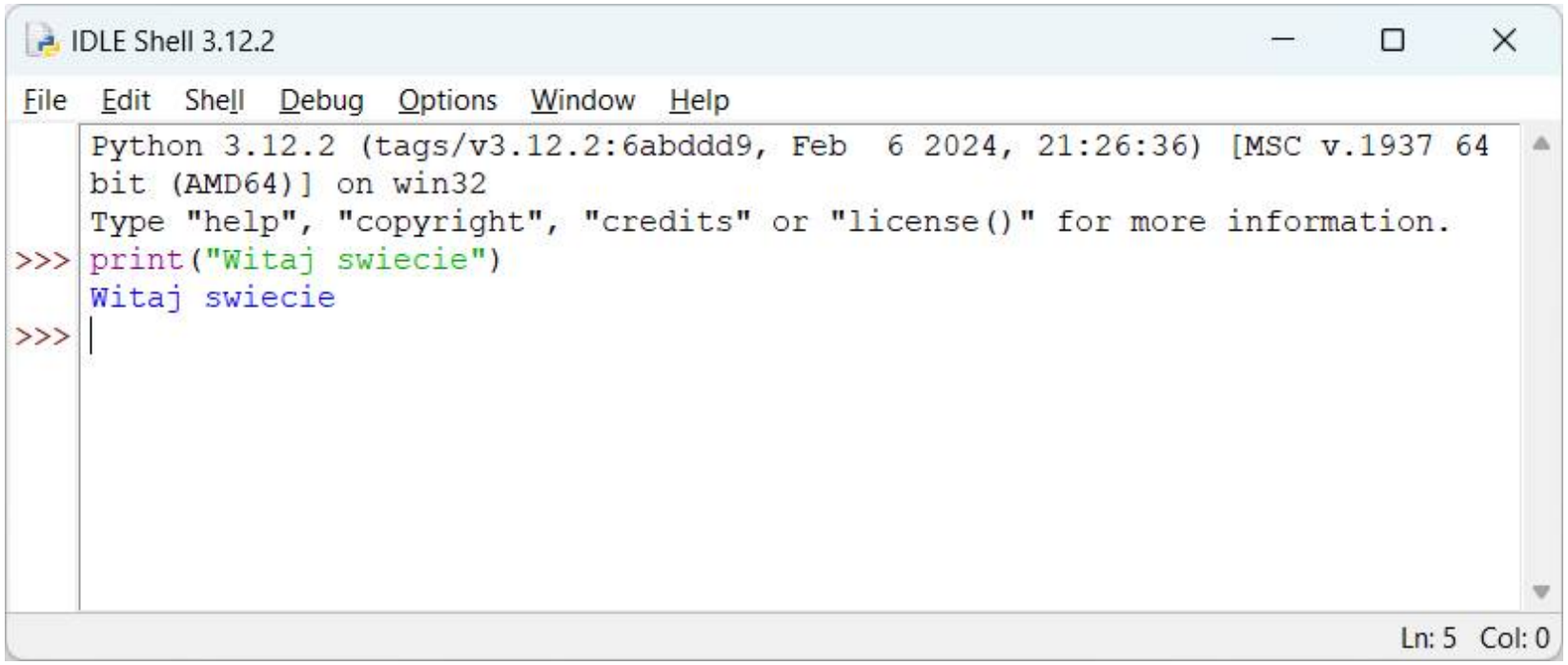


Python - IDLE Shell 3.12.2

- IDLE Shell (Integrated Development and Learning Environment)
  - zintegrowane środowisko programistyczne dla języka Python
  - stworzone przez twórców Pythona
  - zapewnia wiele przydatnych funkcji dla programistów
  - domyślnie instalowane wraz z większością dystrybucji Pythona
- IDLE zawiera:
  - interaktywną powłokę Pythona (Python Shell, Python Interpreter) - interaktywne środowisko, w którym można natychmiastowo wykonywać i testować pojedyncze linie kodu Pythona
  - edytor kodu - pisanie, edycja i zapisywanie kodu Pythona
  - debuger - narzędzia do debugowania kodu Pythona, w celu wykrywania błędów i śledzenie wykonywania programu

Python - IDLE Shell 3.12.2

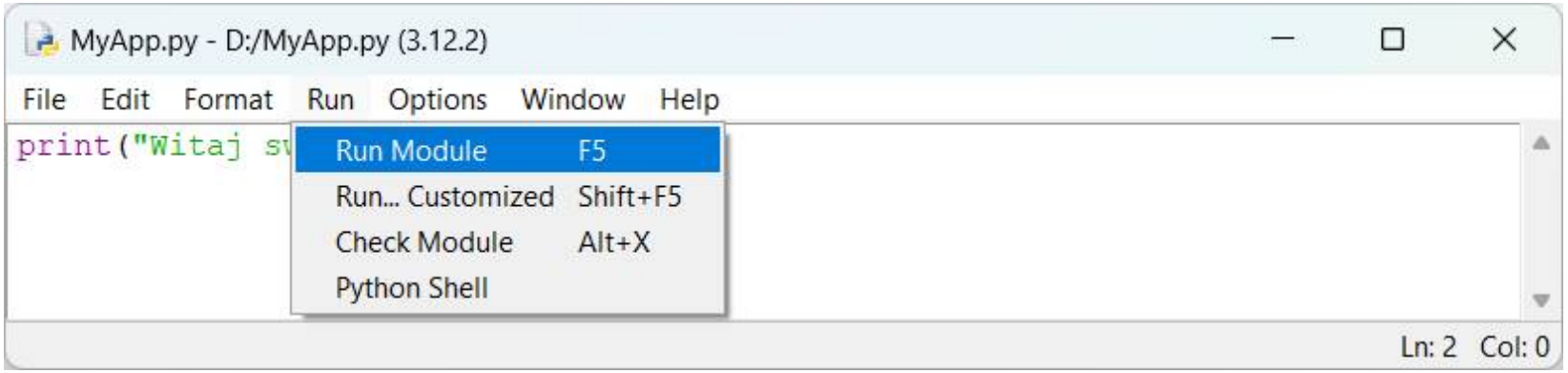
- Interaktywna powłoka Pythona



Python - IDLE Shell 3.12.2

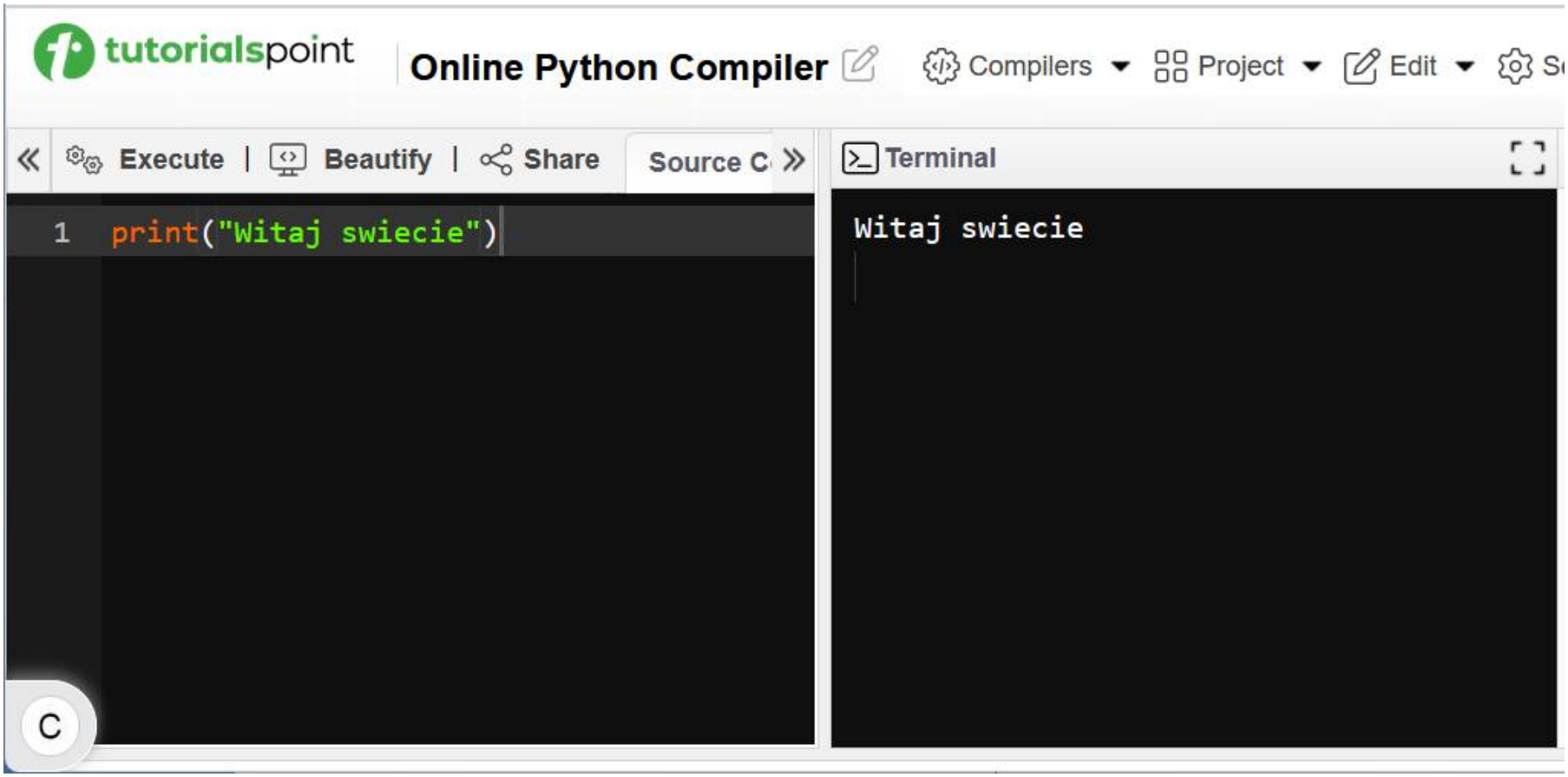
- Edytor kodu

- Uruchomienie programu



Kompilatory on-line

- https://www.tutorialspoint.com/python/online-python-compiler.php

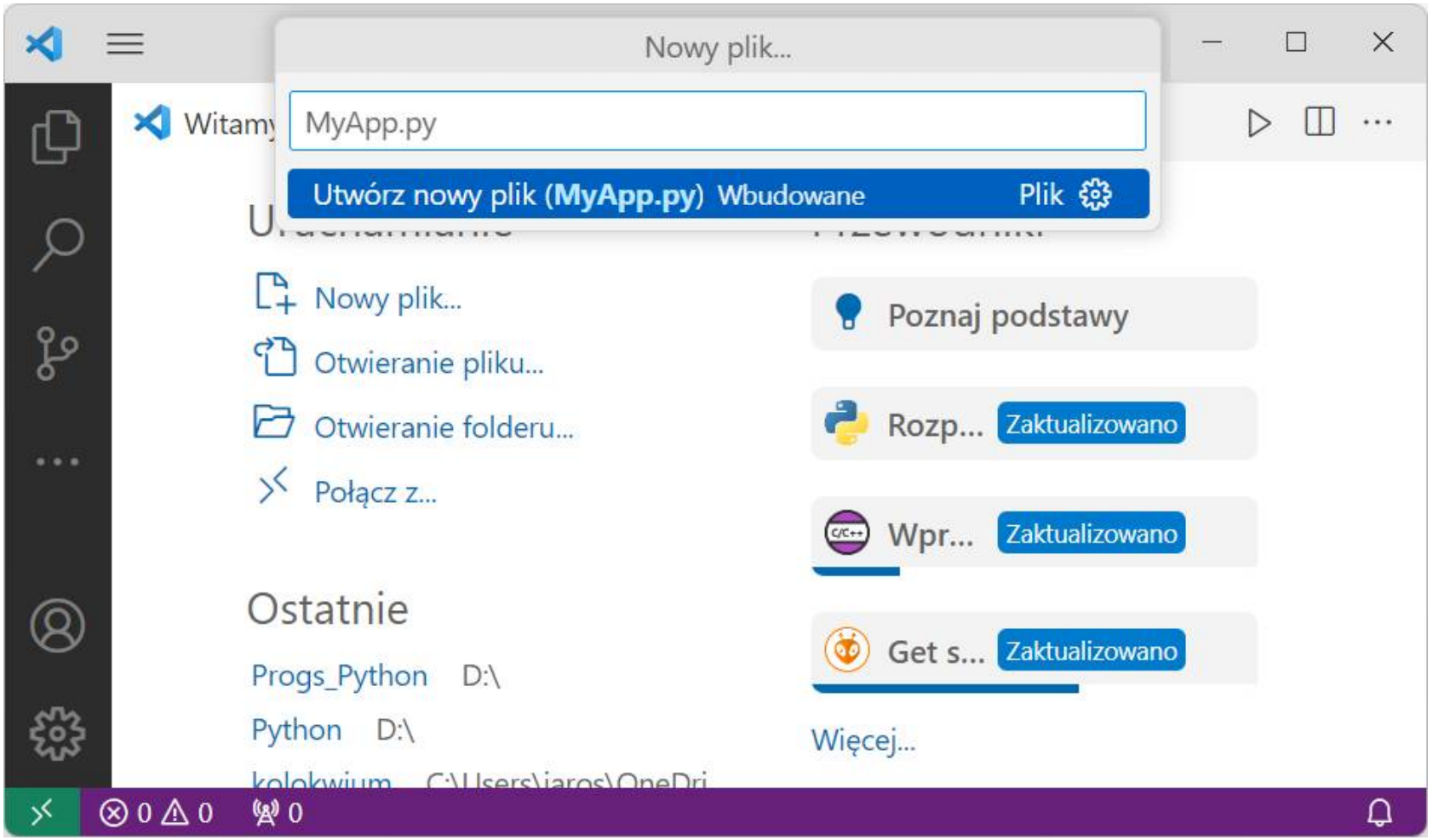


Microsoft Visual Studio Code

- Instalacja rozszerzenia Python (Microsoft)

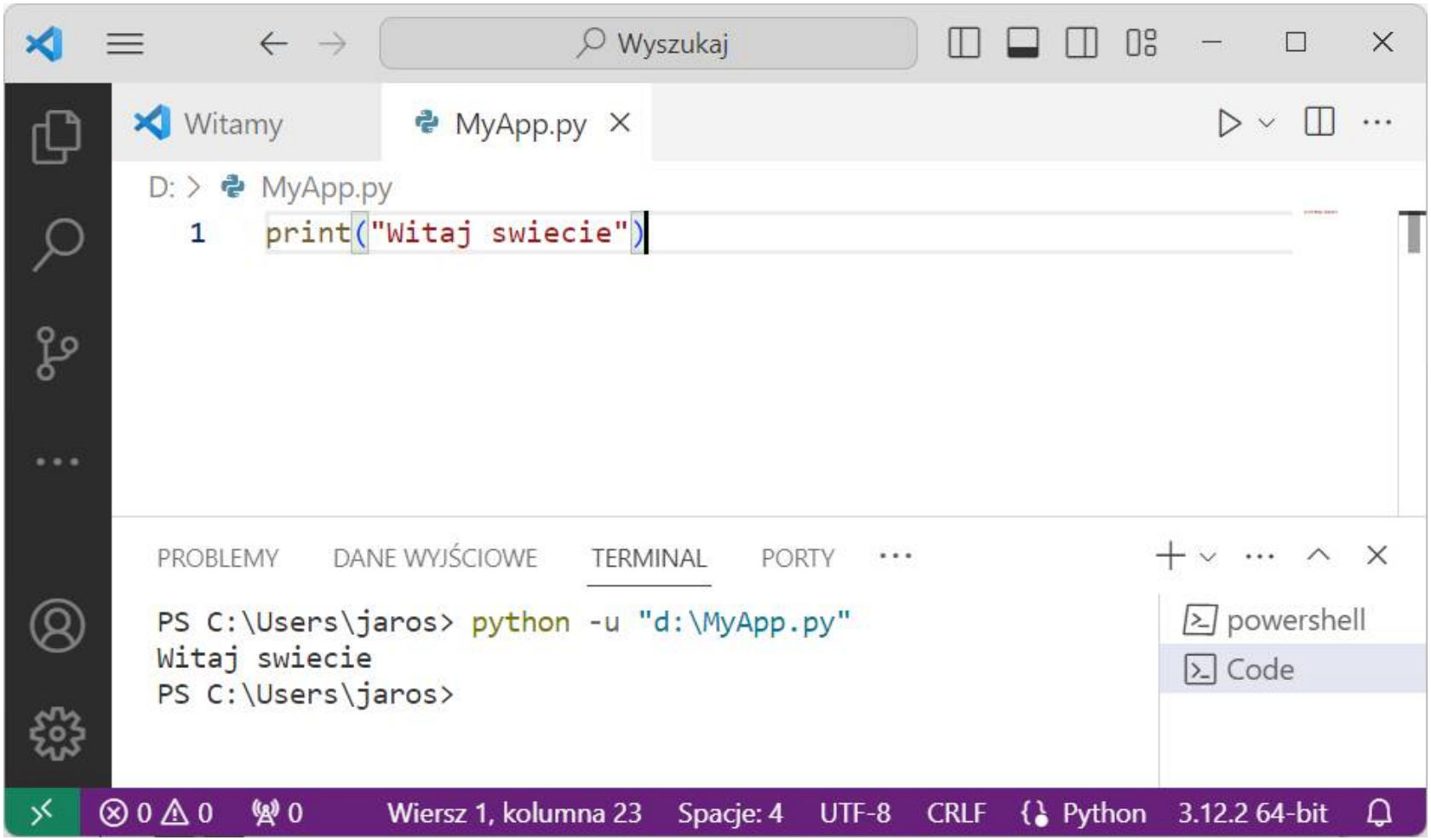
Microsoft Visual Studio Code

- Dodajemy nowy plik (z rozszerzeniem .py)



Microsoft Visual Studio Code

- Wprowadzamy kod, uruchamiamy program



Python - zmienne

- Deklarowanie zmiennych polega na przypisaniu wartości do określonego identyfikatora
- Typ zmiennej jest określany dynamicznie na podstawie wartości przypisanej do zmiennej (nie ma potrzeby jawnej deklaracji typu)

```
a = 10
b = "Witaj"
c = 3.14
d = 5.2e-6
is_true = True
```

- Wielokrotne przypisanie wartości do zmiennych w jednym wierszu

```
a, b, c = 10, 20, 30
```

Python - nazwy zmiennych

- Reguły i zalecenia:
  - nazwy zmiennych mogą zawierać jedynie litery, cyfry i znak podkreślenia
  - nazwa zmiennej może rozpoczynać się od litery lub znaku podkreślenia, ale nie od cyfry

```
temp2    _variable4
```

```
2temp    my-variable
```

  - nie można stosować spacji w nazwach zmiennych (ale można zastosować podkreślenie do separacji słów)

```
my_variable    pole_koła    temp_celsjusz
```

  - unikamy stosowania w nazwach zmiennych słów kluczowych Pythona

```
print    class    input    pass
```

Python - słowa kluczowe

- Słowa kluczowe pisane są małymi literami

```
False      class      finally    is         return
None       continue   for        lambda     try
True       def        from       nonlocal   while
and        del        global     not        with
as         elif       if         or         yield
assert     else       import     pass
break      except     in         raise
```

Python - nazwy zmiennych

- Reguły i zalecenia:
  - nazwa zmiennej powinna być krótka, ale czytelna

```
tc      tempc
```

```
temperatura_w_skali_celsjusza
```

  - ostrożnie używamy małej litery l i wielkiej litery O, bo mogą być łatwo pomylone z cyframi 1 i 0
  - nazwy zmiennych piszemy małymi literami, gdyż przyjęło się, że wielkie litery są używane do nazw stałych
- Nie ma specjalnego sposobu zapisywania / definiowania stałych

```
MAX = 9999
```

```
(constant) MAX: Literal[9999]
MAX = 9999
```

```
(variable) max: Literal[9999]
max = 9999
```

Python - funkcja print()

```
print(*objects, sep=' ', end='\n', file=sys.stdout,
      flush=False)
```

- objects - obiekty do wyświetlenia; jeden lub wiele argumentów (tekst, liczby, listy, zmienne itp.)
- sep (opcjonalny) - separator między obiektami; domyślnie jest to pojedyncza spacja
- end (opcjonalny) - znak końca linii; domyślnie jest to '\n' - każde wywołanie print() kończy się przeniesieniem do nowej linii; przypisując end='', można wywołać print() bez przeniesienia kursora do nowej linii
- file (opcjonalny) - obiekt pliku, na który ma być wypisany tekst; domyślnie jest to standardowe wyjście (sys.stdout)
- flush (opcjonalny) - wartość boolowska określająca, czy należy wymusić opróżnienie bufora; domyślnie jest to False

Python - funkcja print()

```
print("Witaj swiecie!")
```

```
Witaj swiecie!
```

```
print("Witaj", "swiecie", sep=", ", end="!\n")
```

```
Witaj, swiecie!
```

```
x = 10
y = 20
print("Wartosc x to", x, "wartosc y to", y)
```

```
Wartosc x to 10 wartosc y to 20
```

Python - funkcja print(), f-stringi

```
imie = "Anna"
wiek = 30
print(f"Nazywam sie {imie} i mam {wiek} lat.")
```

```
Nazywam sie Anna i mam 30 lat.
```

```
x = 1.23456789
print(f"Wartosc x = {x:.3f}")
```

```
Wartosc x = 1.235
```

```
x = 1.2345
print(f"Wartosc x = [{x:10.2f}]")
```

```
Wartosc x = [      1.23]
```

Python - funkcja print(), f-stringi

```
x = 1.2345
print(f"Wartosc x = [{x:>10.2f}]")
```

```
Wartosc x = [      1.23]
```

```
x = 1.2345
print(f"Wartosc x = [{x:<10.2f}]")
```

```
Wartosc x = [1.23      ]
```

```
x = 1.2345
print(f"Wartosc x = [{x:^10.2f}]")
```

```
Wartosc x = [   1.23   ]
```

Python - typy

- Liczby całkowite:
  - stosowane systemy liczbowe: dziesiętny, dwójkowy, ósemkowy, szesnastkowy

```
w, x, y, z = 10, 0b10, 0o10, 0x10
print(w,x,y,z)
```

```
10 2 8 16
```

  - wyświetlenie liczb w różnych systemach liczbowych

```
x = 255
print(f"Dwójkowy: {x:b}")
print(f"Ósemkowy: {x:o}")
print(f"Szesnastkowy: {x:x}")
print(f"Szesnastkowy: {x:X}")
```

```
Dwójkowy: 11111111
Ósemkowy: 377
Szesnastkowy: ff
Szesnastkowy: FF
```

Python - typy

- Liczby całkowite:
  - przy dzieleniu liczb całkowitych wynik jest zawsze liczbą zmiennoprzecinkową

```
x = 4/2
print(x)
```

```
2.0
```

  - przy zapisywaniu liczb składających się z wielu cyfr można grupować je za pomocą znaków podkreślenia

```
ludnosc = 8_019_000_000
print(ludnosc)
```

```
8019000000
```

Python - typy

- Liczby zmiennoprzecinkowe:
  - w przypadku operacji na liczbie całkowitej i rzeczywistej wynik jest zawsze zmiennoprzecinkowy

```
x = 2 + 2.0
print(x)
```

```
4.0
```

  - czasem wyniki operacji na liczbach zmiennoprzecinkowych może być zaskakujący

```
x = 0.2 + 0.1
print(x)
```

```
0.30000000000000004
```

Python - typy

- Ciągi tekstowe:
  - ciąg tekstowy może być ujęty w cudzysłów lub apostrofy

```
txt1 = "Ala ma laptopa"
txt2 = 'Janek ma tablet'
```

- Wbudowana funkcja type() pozwala sprawdzić typ przekazanego argumentu

```
print(type(32))
print(type(2.5))
print(type(3.1e-4))
print(type(True))
print(type("napis"))
print(type('napis'))
```

```
<class 'int'>
<class 'float'>
<class 'float'>
<class 'bool'>
<class 'str'>
<class 'str'>
```

Python - funkcja input()

```
input(prompt)
```

- funkcja input() służy do pobierania danych od użytkownika za pomocą klawiatury
- prompt (opcjonalny) - tekst wyświetlany przed oczekiwaniem na wprowadzenie danych przez użytkownika; jest to informacja dla użytkownika, co powinien wpisać
- funkcja input() zwraca wprowadzone przez użytkownika dane w postaci łańcucha znaków (typu str)
- w przypadku wprowadzania liczb konieczne jest użycie funkcji konwersji, np. int() lub float()

Python - funkcja input()

```
imie = input("Podaj swoje imie: ")
print("Witaj,", imie)
```

```
Podaj swoje imie: Paweł
Witaj, Paweł
```

```
rok = input("Podaj rok urodzenia: ")
wiek = 2024 - rok
print("Masz", wiek, "lat")
```

```
Podaj rok urodzenia: 2003
Traceback (most recent call last):
  File "d:\myapp.py", line 2, in <module>
    wiek = 2024 - rok
           ~~~~~^~~~~
TypeError: unsupported operand type(s) for -: 'int' and 'str'
```

Python - funkcja input()

```
rok = int(input("Podaj rok urodzenia: "))
wiek = 2024 - rok
print("Masz", wiek, "lat")
```

```
Podaj rok urodzenia: 1997
Masz 27 lat
```

```
Tf = float(input("Podaj temperaturę [F]: "))
print(f"{Tf} [F] to {5/9*(Tf-32)} [C]")
```

```
Podaj temperaturę [F]: 45.5
45.5 [F] to 7.5 [C]
```

Python - operatory arytmetyczne

- Dodawanie (+): dodaje dwie liczby lub łączy dwa ciągi znaków

```
x = 5
y = 3
print(x + y)  # Wyświetli: 8
```

- Odejmowanie (-): odejmuje jedną liczbę od drugiej

```
x = 5
y = 3
print(x - y)  # Wyświetli: 2
```

Python - operatory arytmetyczne

- Mnożenie (*): mnoży dwie liczby

```
x = 5
y = 3
print(x * y)  # Wyświetli: 15
```

- Dzielenie (/): dzieli jedną liczbę przez drugą, zwraca liczbę zmiennoprzecinkową

```
x = 5
y = 3
print(x / y)  # Wyświetli: 1.6666666666666667
```

Python - operatory arytmetyczne

- Dzielenie całkowite (//): dzieli jedną liczbę przez drugą i zwraca wynik jako liczbę całkowitą, zaokrąglając w dół

```
x = 5
y = 3
print(x // y)  # Wyświetli: 1
```

- Reszta z dzielenia (%): zwraca resztę z dzielenia jednej liczby przez drugą

```
x = 5
y = 3
print(x % y)  # Wyświetli: 2
```

Python - operatory arytmetyczne

- Potęgowanie (**): Podnosi pierwszą liczbę do potęgi drugiej

```
x = 5
y = 3
print(x ** y)  # Wyświetli: 125
```

Python - priorytet operatorów

- Priorytet operatorów od najwyższego do najniższego:
  - wyrażenia w nawiasach - wyrażenia w nawiasach są zawsze wykonywane jako pierwsze
  - potęgowanie (**) - operator potęgowania ma najwyższy priorytet
  - jednoargumentowe operatory: ujemny (-) i dodatni (+) - operatory wykonywane przed innymi operatorami arytmetycznymi
  - mnożenie (*), dzielenie (/), dzielenie całkowite (//) i reszta z dzielenia (%) - operatory wykonywane przed dodawaniem i odejmowaniem
  - dodawanie (+) i odejmowanie (-) - najniższy priorytet mają operatory dodawania i odejmowania
- Operatory arytmetyczne są lewostronnie łączne

```
x = 5 - 2 + 3      # (-) → (+) → (=)
```

Python - operatory rozszerzonego przypisania

Operator	Przykład instrukcji	Instrukcja równoważna
`+=`	`x += 10`	`x = x + 10`
`-=`	`x -= 10`	`x = x - 10`
`*=`	`x *= 10`	`x = x * 10`
`/=`	`x /= 10`	`x = x / 10`
`//=`	`x //= 10`	`x = x // 10`
`%=`	`x %= 10`	`x = x % 10`
`**=`	`x **= 10`	`x = x ** 10`

Python - stałe i funkcje matematyczne

- Stałe i funkcje matematyczne dostępne w module o nazwie math
- Aby korzystać z tych funkcji i stałych, należy najpierw zaimportować ten moduł

```
import math
```

- Stałe matematyczne:
  - math.pi - stała pi (π)
  - math.e - stała e (podstawa logarytmu naturalnego)
  - math.inf - nieskończoność (dodatnia)
  - math.nan - NaN (Not a Number) - reprezentuje wartość nieokreśloną lub niemożliwą do reprezentacji numerycznej

Python - stałe i funkcje matematyczne

- Funkcje matematyczne:
  - sin(x) - oblicza sinus kąta x (w radianach)
  - cos(x) - oblicza cosinus kąta x (w radianach)
  - tan(x) - oblicza tangens kąta x (w radianach)
  - atan(x) - oblicza arcus tangens x w radianach, zwracając wynik w zakresie $[-\pi/2, \pi/2]$
  - sqrt(x) - oblicza pierwiastek kwadratowy z x
  - exp(x) - oblicza wartość wykładniczą e podniesioną do potęgi x
  - log(x) - oblicza logarytm naturalny z x
  - log10(x) - oblicza logarytm o podstawie 10 z x
  - pow(x, y) - oblicza x podniesione do potęgi y
  - fabs(x) - zwraca wartość bezwzględną |x|

Python - przykład (pole i obwód koła)

```
import math

# Wczytanie promienia koła z klawiatury
promien = float(input("Podaj promień koła: "))

# Obliczenie pola koła
pole = math.pi * math.pow(promien,2)

# Obliczenie obwodu koła
obwod = 2 * math.pi * promien

# Wyświetlenie wyników
print(f"Pole koła: {pole:.2f}")
print(f"Obwód koła: {obwod:.2f}")
```

```
Podaj promień koła: 10
Pole koła: 314.16
Obwód koła: 62.83
```

Python - komentarze

- Komentarze są używane do dodawania opisów lub wyjaśnień w kodzie źródłowym programu i są ignorowane podczas wykonywania programu
- Rozpoczynają się od znaku # i obejmują całą linię
- Wszystko po znaku # traktowane jest jako komentarz

```
# Kod najprostszego programu
print("Witaj swiecie!")
```

Python - operatory porównania

Operator	Znaczenie	Przykład	Wynik
==	równa się	`2 == 2`	`True`
!=	nie równa się	`2 != 2`	`False`
>	większe niż	`2 > 5`	`False`
<	mniejsze niż	`2 < 5`	`True`
>=	mniejsze lub równe	`2 >= 5`	`False`
<=	większe lub równe	`2 <= 2`	`True`

- w wyniku porównania otrzymujemy wartość True (prawda) lub False (fałsz)
- wartości True i False można przypisywać zmiennym

```
koniec = True
warunek = False
```

Python - operatory porównania (przykłady)

```
wiek = 17
print(wiek == 18)
```

```
False
```

```
wiek = 17
print(wiek < 18)
```

```
True
```

```
wiek = 17
print(wiek >= 15)
```

```
True
```

Python - operatory porównania (przykłady)

```
imie = "Jan"
print(imie != "Jan")
```

```
False
```

```
imie = "Jan"
print(imie == "Jan")
```

```
True
```

```
imie = "Jan"
print(imie > "Ela")
```

```
True
```

Python - operatory porównania (przykłady)

```
imie = "Jan"
print(imie == "jan")
```

```
False
```

```
imie = "Jan"
print(imie.lower() == "jan")
```

```
True
```

- metoda lower() zwraca ciąg znaków, w którym wszystkie wielkie litery zostały zamienione na małe
- metoda ta nie zmienia wartości zmiennej imie

Python - operatory logiczne

Operator	Znaczenie	Przykład
`and`	zwraca wartość True jeśli oba warunki są prawdziwe	`x > 5 and x <= 7`
`or`	zwraca wartość True jeśli przynajmniej jeden z warunków jest prawdziwy	`x < 3 or x >= 4`
`not`	zwraca odwrotność wartości logicznej wyrażenia	`not x`

```
wiek1 = 19
wiek2 = 17
print(wiek1 >= 18 and wiek2 >= 18)
```

```
False
```

Python - operatory logiczne (przykłady)

- w warunku można zastosować dodatkowe nawiasy, ale nie jest to konieczne

```
wiek1 = 19
wiek2 = 17
print((wiek1 >= 18) and (wiek2 >= 18))
```

- przykład zastosowania operatora or

```
wiek1 = 19
wiek2 = 17
print(wiek1 >= 18 or wiek2 >= 18)
```

```
True
```

Python - instrukcja if

- Najprostsza postać instrukcji if:

```
if test_warunkowy:
    dowolna_akcja
```

  - jeśli wynikiem testu_warunkowego jest True to akcja jest wykonywana; jeśli zaś False - to akcja nie jest wykonywana
  - wcięty blok kodu określa akcje wykonywane po if

```
wiek = 19
if wiek >= 18:
    print("Jesteś pełnoletni")
```

  - blok kodu to jeden lub kilka kolejnych wierszy kodu z taką samą wielkością wcięcia
  - wcięcie to najczęściej: 4 spacje, 2 spacje, tabulator

Python - instrukcja if

- tam, gdzie kończy się wcięcie, tam też kończy się blok warunkowy:

```
wiek = 19
if wiek >= 18:
    print("Jesteś pełnoletni")
    print("Możesz iść na wybory")
print("Koniec")
```

Python - instrukcja if-else

- Składnia instrukcji if-else:

```
if test_warunkowy:
    akcja1
else:
    akcja2
```

  - jeśli wynikiem testu_warunkowego jest True to wykonywana jest tylko akcja1; jeśli zaś False - to wykonywana jest tylko akcja2

```
liczba = int(input("Podaj liczbe: "))
if liczba % 2 == 0:
    print(f"{liczba} - liczba parzysta")
else:
    print(f"{liczba} - liczba nieparzysta")
```

Python - przykład (pierwiastek kwadratowy)

```
import math

x = float(input("Podaj liczbę: "))
if x >= 0:
    y = math.sqrt(x)
    print("Pierwiastek liczby:", y)
else:
    print("Błąd! Liczba ujemna")
```

```
Podaj liczbę: 5
Pierwiastek liczby: 2.23606797749979

Podaj liczbę: -3
Błąd! Liczba ujemna
```

Python - przykład (pierwiastek kwadratowy)

```
import math

x = float(input("Podaj liczbę: "))
if x >= 0:
    y = math.sqrt(x)
    print("Pierwiastek liczby:", y)
else:
    print("Błąd! Liczba ujemna")
```

```
Podaj liczbę: a
Traceback (most recent call last):
  File "d:\MyApp.py", line 3, in <module>
    x = float(input("Podaj liczbę: "))
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: could not convert string to float: 'a'
```

Python - przykład (pierwiastek kwadratowy)

```
import math

try:
    x = float(input("Podaj liczbę: "))
    if x >= 0:
        y = math.sqrt(x)
        print("Pierwiastek liczby:", y)
    else:
        print("Błąd! Liczba ujemna")
except ValueError:
    print("Błąd! Wprowadź poprawną liczbę")
```

```
Podaj liczbę: a
Błąd! Wprowadź poprawną liczbę
```

Python - instrukcja if-elif-else

- Składnia instrukcji if-elif-else:

```
if test1:
    akcja1
elif test2:
    akcja2
else:
    akcja3
```

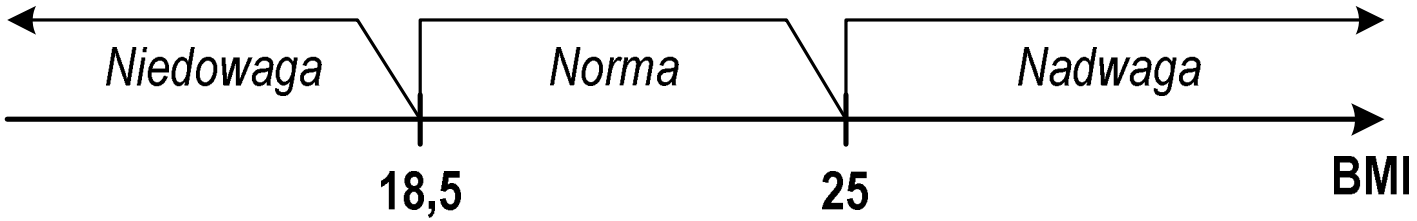
  - jeśli wynikiem test1 jest True to wykonywana jest akcja1
  - jeśli wynikiem test1 jest False to sprawdzany jest test2
  - jeśli wynikiem test2 jest True to wykonywana jest akcja2
  - jeśli wynikiem test2 jest False to wykonywana jest akcja3
  - tylko jedna akcja może być wykonana

Python - przykład (BMI)

- BMI - współczynnik powstały przez podzielenie masy ciała podanej w kilogramach przez kwadrat wzrostu podanego w metrach

$$BMI = \frac{masa}{wzrost^2}$$

- Dla osób dorosłych:
  - BMI < 18,5 - wskazuje na niedowagę
  - BMI ≥ 18,5 i BMI < 25 - wskazuje na prawidłową masę ciała
  - BMI ≥ 25 - wskazuje na nadwagę

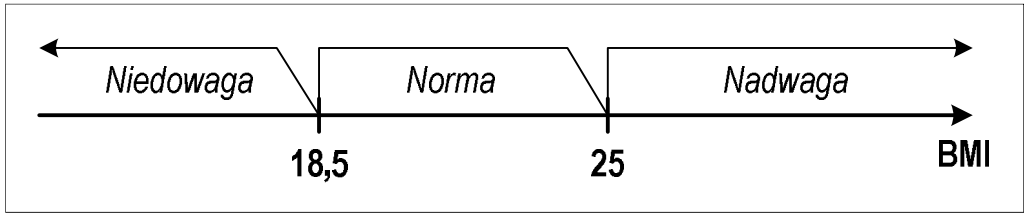


Python - przykład (BMI)

```
masa = float(input("Podaj masę [kg]: "))
wzrost = float(input("Podaj wzrost [m]: "))
bmi = masa / (wzrost * wzrost)
print("BMI:", "{:.2f}".format(bmi))

if bmi < 18.5:
    print("Niedowaga")
elif bmi >= 18.5 and bmi < 25:
    print("Norma")
else:
    print("Nadwaga")
```

```
Podaj masę [kg]: 84
Podaj wzrost [m]: 1.85
BMI: 24.54
Norma
```



Python - przykład (BMI)

```
masa = float(input("Podaj masę [kg]: "))
wzrost = float(input("Podaj wzrost [m]: "))
bmi = masa / (wzrost * wzrost)
print("BMI:", "{:.2f}".format(bmi))

if bmi < 18.5:
    print("Niedowaga")
elif 18.5 <= bmi < 25:
    print("Norma")
else:
    print("Nadwaga")
```

- warunek sprawdzający czy bmi znajduje się w przedziale można zapisać także w inny sposób

`bmi >= 18.5 and bmi < 25` → `18.5 <= bmi < 25`

Python - przykład (BMI)

```
masa = float(input("Podaj masę [kg]: "))
wzrost = float(input("Podaj wzrost [m]: "))
bmi = masa / (wzrost * wzrost)
print("BMI:", "{:.2f}".format(bmi))

if bmi < 18.5:
    print("Niedowaga")
else:
    if bmi < 25:
        print("Norma")
    else:
        print("Nadwaga")
```

- instrukcje if można zagnieżdżać

Python - przykład (oceny)

```
pkt = int(input("Podaj liczbę punktów: "))

if pkt < 51:
    ocena = 2.0
elif pkt < 61:
    ocena = 3.0
elif pkt < 71:
    ocena = 3.5
elif pkt < 81:
    ocena = 4.0
elif pkt < 91:
    ocena = 4.5
else:
    ocena = 5.0

print(f"Twoja ocena: {ocena:.1f}")
```

- blok elif może występować wielokrotnie
- ostatni else może być pominięty

Python - operator warunkowy

- Składnia operatora warunkowego (wyrażenia trójargumentowego):

```
wartość_prawda if warunek else wartość_fałsz
```

  - warunek jest warunkiem logicznym, który ma być sprawdzany
  - wartość_prawda jest wartością zwracaną, jeśli warunek jest spełniony
  - wartość_fałsz jest wartością zwracaną, jeśli warunek nie jest spełniony

```
x = int(input("Podaj liczbe: "))
txt = "Parzysta" if x % 2 == 0 else "Nieparzysta"
print(txt)
```

Koniec wykładu nr 1

Dziękuję za uwagę!