



Python - część 1

Tematyka zajęć

- ▶ Składnia i formatowanie
- ▶ Typy zmiennych
- ▶ Operacje wejścia/wyjścia
- ▶ Formatowanie napisów
- ▶ Obsługa błędów i wyjątków

Python

- ▶ Język programowania wysokiego poziomu
- ▶ Interpretowany
- ▶ Wspiera różne paradygmaty programowania: obiektowy, strukturalny, częściowo funkcyjny
- ▶ Dynamicznie typowany
- ▶ Posiada automatyczne zarządzanie pamięcią
- ▶ Może być używany jako język skryptowy
- ▶ Posiada interaktywny interpreter
- ▶ Język ogólnego przeznaczenia (aplikacje webowe, aplikacje użytkowe, data science i sztuczna inteligencja)
- ▶ Duża liczba modułów standardowych oraz dodatkowych bibliotek
- ▶ Kompaktowość kodu: grupowanie instrukcji poprzez wcięcia w kodzie (zamiast nawiasów), niepotrzebne deklaracje zmiennych i argumentów, użycie struktur wysokiego poziomu
- ▶ Daną rzecz można zrobić tylko na jeden sposób



Guido van Rossum, twórca Pythona

By Daniel Stroud - Praca własna, CC BY-SA 4.0,

<https://commons.wikimedia.org/w/index.php?curid=52040846>

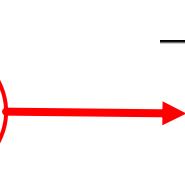
Interpreter Pythona (cpython)

W formie interaktywnej:

```
air-zuzanna:~ noemi$ python3.7
Python 3.7.1 (default, Nov 28 2018, 11:51:47)
[Clang 10.0.0 (clang-1000.11.45.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>>
>>> a = 5
>>> 2 + a
7
>>> quit()
```

W formie wsadowej:

```
air-zuzanna:~ noemi$ python demo.py
7
```



```
a = 5
b = a + 2
print(b)
```

Zmienne - konwencja nazewnicza

- Nazwa zmiennej/funkcji/obiektu może zaczynać się od: liter (A...Z, a..z) lub znaku podkreślenia (_) po którym mogą następować: litery, cyfry, _
- Wielkość liter ma znaczenie
- Nazwy zmiennych nie mogą być takie same jak słowa kluczowe np. and, def, while, if
 - 7_krasnolodkow
 - krasnolodkow_7
 - \$suma
 - Suma
 - Suma != suma

Przyjmujemy, że identyfikatory zaczynamy:

- Klas - wielką literą
- pozostałe - małą literą
- zmienne prywatne - pojedynczym lub podwójnym podkreśleniem („_nazwa”, „__nazwa”)
- __nazwa__ - nazwa specjalna określona przez język

Zmienne

- ▶ Podlegają tzw. leniwej inicjalizacji. Pamięć jest rezerwowana dopiero w chwili wywołania przypisania.
- ▶ Nie ma potrzeby deklarowania typu zmiennej.
- ▶ Do tworzenia zmiennych używamy operatora przypisania
- ▶ Typ zmiennej możemy podejrzeć przy pomocy wbudowanej funkcji `type()`
- ▶ Do jednej nazwy zmiennych możemy przypisać wiele typów zmiennych

```
[In [30]: zmienna = 1
```

```
[In [31]: type(zmienna)
```

```
Out[31]: int
```

```
[In [32]: zmienna = 'Ala ma kota'
```

```
[In [33]: type(zmienna)
```

```
Out[33]: str
```

```
[In [34]: zmienna = 1.234
```

```
[In [35]: type(zmienna)
```

```
Out[35]: float
```

Zmienne - rzutowanie

- ▶ Możliwa jest bardzo prosta zmiana typu danych dla typów podstawowych. Wystarczy odnosić się po prostu do typów w połączeniu z operatorem (), tj. operacjami

- ▶ int()
- ▶ float()
- ▶ str()

```
[In [46]:] zmienna = 1

[In [47]:] float(zmienna)
Out[47]: 1.0

[In [48]:] str(zmienna)
Out[48]: '1'

[In [49]:] int(zmienna)
Out[49]: 1
```

Uwaga na obcięcie dokładności

```
[In [51]:] zmienna = 3.14

[In [52]:] int(zmienna)
Out[52]: 3
```

Zmienne - rzutowanie

```
In [54]: zmienna = "Ala"
```

```
In [55]: int(zmienna)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-55-a7bd95ff52ea> in <module>()  
----> 1 int(zmienna)
```

```
ValueError: invalid literal for int() with base 10: 'Ala'
```

```
In [56]: █
```

Zmiana bazy z domyślnej 10-tnej:

```
In [57]: x = '101'
```

```
In [58]: int(x)
```

```
Out[58]: 101
```

```
In [59]: int(x, base=2)
```

```
Out[59]: 5
```

```
In [60]: int(x, base=16)
```

```
Out[60]: 257
```


Operator

- ▶ arytmetyczne,
- ▶ relacyjne (porównawcze),
- ▶ logiczne,
- ▶ bitowe,
- ▶ przypisania,
- ▶ identycznościowe,
- ▶ przynależności.

Operatory arytmetyczne

- ▶ * dodawanie '+'
- ▶ * odejmowanie '-'
- ▶ * mnożenie '*'
- ▶ * dzielenie '/'
- ▶ * dzielenie modulo '%'
- ▶ * potęgę '**'
- ▶ * dzielenie całkowitoliczbowe '//'

```
int x = 3;  
int y = 2;  
cout << x/y;
```

```
▶ x = 3  
  y = 2  
  print(x/y)  
  print(x//y)
```

```
↳ 1.5  
   1
```

Operatory relacyjne

- ▶ równość '=='
- ▶ nierówność '!='
- ▶ mniejsze oraz większe ('<' i '>')
- ▶ mniejsze lub równe oraz większe lub równe ('<=' i '>=')

```
▶ if 3 < 5:  
    print('3 jest mniejsze od 5')
```

3 jest mniejsze od 5

```
▶ liczba = 4  
if 3 < liczba <= 5:  
    print('Liczba ta należy do przedziału (3,5]')
```

Liczba ta należy do przedziału (3,5]

Operatory logiczne

- ▶ logiczne i 'and',
- ▶ logiczne lub 'or',
- ▶ logiczne nie 'not'.

```
[ ] liczba = 14
    if liczba % 3 == 0 and liczba % 2 == 0 :
        print('Liczba jest podzielna przez 3 oraz 2')
    else :
        print('Liczba nie jest podzielna przez 3 oraz 2')
```




Liczba nie jest podzielna przez 3 oraz 2


Operatory bitowe

- ▶ Bitowe i ,&' - gdzie dany bit zostaje ustalony na 1 tylko jeśli oba operandy miały 1 na tym bicie
- ▶ Bitowe lub '|' - gdzie dany bit zostaje ustalony na 1 tylko jeśli dowolny operand miał 1 na tym bicie
- ▶ Bitowe xor '^' - gdzie dany bit zostaje ustalony na 1 tylko jeśli dokładnie jeden operand miał 1 na tym bicie
- ▶ Bitowe not '~' - gdzie dany bit zostaje ustalony na 1 tylko jeśli operand (jedeny operand) miał 0 na tym bicie
- ▶ Bitowe przesunięcie w lewo '<<' - gdzie bity są przesuwane zgodnie w reprezentacji na miejsca bardziej po lewo, bity wysunięte poza zakres są tracone, a od lewej strony reprezentacja jest uzupełniana zerami
- ▶ Bitowe przesunięcie w prawo '>>' - gdzie bity są przesuwane zgodnie w reprezentacji na miejsca bardziej po prawo, bity wysunięte poza zakres są tracone, a od lewej strony reprezentacja jest uzupełniana zerami

```
[ ] x = 0b0011 #tak można zadawać liczby w postaci dwójkowej  
    y = 0b1110  
    print('x ma wartość', x, 'y ma wartość', y)
```


 x ma wartość 3 y ma wartość 14

```
▶ z = x & y  
  print('Wynik {0:{fill}4b}'.format(z,fill='0'))
```

 Wynik 0010

Zaprezentujemy jeszcze działanie przesunięcia

```
[ ] print('Wynik {0:{fill}4b}'.format(x<<1,fill='0'))
```

 Wynik 0110

Operatory przypisania

- ▶ Kolejną grupą operatorów są operatory przypisania. Do grupy tej zaliczamy
- ▶ operator '=',
- ▶ operatory przypisania arytmetycznego '+=', '-=', '=', '/=', '%=', '//=', '*=',
- ▶ operatory przypisania logicznego '&=', '|=', '^=',
- ▶ operatory przypisania przesuniętego '<<=' i '>>='

```
[In [75]: x = 0
```

```
[In [76]: x = x + 1
```

```
[In [77]: print(x)  
1
```

```
[In [78]: x = 0
```

```
[In [79]: x += 1
```

```
[In [80]: print(x)  
1
```

Operatory identycznościowe i przynależności

- ▶ is
- ▶ is not
- ▶ in
- ▶ not in

```
[In [83]: a = [5]
```

```
[In [84]: b = [5]
```

```
[In [85]: a is b
```

```
Out[85]: False
```

```
[In [86]: c = b
```

```
[In [87]: c is b
```

```
Out[87]: True
```

```
[In [88]: 5 in [1, 2, 3, 4, 5]
```

```
Out[88]: True
```

```
[In [89]: -5 in [1, 2, 3, 4, 5]
```

```
Out[89]: False
```

Wbudowane typy danych

Typ	Opis	Przykład
int	Liczba całkowita	123
float	Liczba zmiennoprzecinkowa	3.1415927
complex	Liczba zespolona	3 + 1.5j
bool	Prawa/Fałsz	True False
bytes	Sekwencja bajtów	b'Ala' b"Ala"
str	Ciąg znaków	'Ala', "Ala"
list	Lista (zmienna)	[1,2,2]
tuple	Krotka (niezmienna)	(1, 2, 1)
set	Zbiór	{1, "Ala", 3}
frozenset	Zbór (niezmienny)	
dict	Słownik/ tablica asocjacyjna	{'klucz1':1, 'klucz2':2}

Łańcuchy znaków

- ▶ Obowiązuje zasada, że znak który otwierał łańcuch musi go zamknąć. Stąd w łańcuchach tworzonych przed `"""` można np. umieścić `'` bez obawy o to że uszkodzą skrypt
- ▶ Rezerwuje się znak `"""` jako znak komentarza dokumentującego i stąd nie zaleca się go tworzenia słów
- ▶ Potrójne cudzysłowy można używać do tekstów zajmujących wiele linijek kodu

```
[16] o_ali = "Ala" + " " + "ma" + " kota"
```

```
▶ print(o_ali)
```

```
☞ Ala ma kota
```

Przykład konkatencji

```
▶ print("Witaj świecie!")  
print('Witaj świecie!')  
print(''''Witaj świecie!''')  
print("""Witaj świecie!""")
```

```
☞ Witaj świecie!  
Witaj świecie!  
Witaj świecie!  
Witaj świecie!
```

Łańcuchy znaków - przykładowe operacje

```
# Operacje na łańcuchach znaków
s = "Dzień dobry"
print(len(s)) # długość łańcucha
print(s.lower()) # zamiana znaków na małe litery
print(s.split()) # rozdzielenie napisów, domyślny separator " "
print("i" in s) # sprawdzamy czy w napisie jest literka i
print("\nDzień dobry \ndo widzenia \n\t***")
print("\\")
```

```
11
dzień dobry
['Dzień', 'dobry']
True

Dzień dobry
do widzenia
    ***
\
```

```
s = "Dzień dobry"
print(s[0])
print(s[0:5])
```

```
D
Dzień
```

Formatowanie łańcuchów znaków

```
▶ zmienna = 123.123456789
print(f'Wartość zmiennej to {zmienna:10.5f}.')
```

```
print("Wartość zmiennej to {}".format(zmienna))
print("Wartość zmiennej to {:10.5f}".format(zmienna))
ala = "Ala"
ola = "Ola"
print("Cześć {0} i {1}".format(ala, ola))
print("Cześć {1} i {0}".format(ala, ola))
```

```
☞ Wartość zmiennej to 123.12346.
Wartość zmiennej to 123.123456789
Wartość zmiennej to 123.12346
Cześć Ala i Ola
Cześć Ola i Ala
```

Odczyt i zapis do pliku

```
f = open('test.txt', 'w')
```

Nazwa pliku

uprawnienia: w - do zapisu, r - do odczytu

- f.write(...) - zapis do pliku
- f.read() - odczyt całej zawartości pliku
- f.readline() - odczyt linii z pliku
- f.close() - zamknięcie pliku

```
▶ with open('test.txt') as f:  
    line1 = f.readline()  
    line2 = f.readline()  
    print(line1)  
    print(line2)
```

📄 Drugi zjazd CMI

Zajęcia z podstaw pythona

Struktura programu: funkcje

- ▶ Funkcja: blok kodu uruchamiany tylko wtedy gdy jest wywołany
- ▶ Do funkcji można przekazywać parametry
- ▶ Funkcja może zwracać rezultat obliczeń

```
[52] def funkcja(powitanie):  
    | print(powitanie)
```

Definicja

```
▶ funkcja("Dzień dobry")
```

Wywołanie

```
☐➔ Dzień dobry
```

Struktura programu moduły

- ▶ Moduły w Pythonie to katalogi oraz pliki
- ▶ `__init__.py`

```
def powiedz_hello():  
    print('Hello')
```

```
[ ] from paczka.plik_w_paczce import powiedz_hello  
  
    print('Wykorzystajmy kod z modułu paczka')  
    powiedz_hello()
```

Moduły import

Sposób importu modułów

```
import plik
import moduł.plik
import moduł.submoduł.plik
from plik import *
from plik import nazwa
from moduł.plik import *
from moduł.plik import nazwa1, nazwa2
```

Aby uniknąć duplikacji nazw

```
import nazwa_modułu as przezwanie
from moduł import nazwa_w_pliku as przezwanie
```

Podstawowe pakiety Pythona:

- Math
- Random
- os

Wyjątki

- ▶ Sytuacja w działaniu programu, która uniemożliwia poprawne wykonanie części skryptu
- ▶ Kod który może nie zakończyć się poprawnym wykonaniem należy zgromadzić w bloku try

```
def dzielenie(x, y):  
    try:  
        print("Dzielenie "+str(x)+"/"+str(y))  
        result= x/y  
    except ZeroDivisionError:  
        print("Błąd dzielenia")  
    else:  
        print("Wynik to "+str(result))  
    finally:  
        print("A ten blok wykona się zawsze")  
  
dzielenie(2, 3)  
dzielenie(1, 0)
```


Ćwiczenie

Napišmy program, który:

- ▶ Oblicza pole okręgu o zadanym promieniu
- ▶ Oblicza obwód okręgu o zadanym promieniu
- ▶ Zapisuje wyniki do pliku z dokładnością do dwóch miejsc po przecinku