

Język Python

Python to język interpretowany, wysokiego poziomu, obiektowy, umożliwia również programowanie strukturalne i funkcyjne.

Interpreter

Interpreter Pythona to program, który wykonuje skrypty zapisane w plikach z rozszerzeniem ".py" i polecenia wprowadzane w trybie interaktywnym. W systemach Linux interpreter jest preinstalowany. W systemie MS Windows należy do zainstalować.



Tryb interaktywny

Uruchom terminal (wiersz poleceń) i wpisz polecenie `python3`. Interpreter Pythona zostanie uruchomiony w trybie interaktywnym. Po znaku zachęty `>>>` możesz wpisywać wyrażenia i instrukcje, które chcesz przetestować. Wpisz `exit()`, aby opuścić tryb interaktywny.

```
lmint@dboxh: ~  
Plik  Edycja  Widok  Wyszukiwanie  Terminal  Pomoc  
lmint@dboxh:~$ python3  
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> a = 10  
>>> b = 5  
>>> a > b  
True  
>>> exit()  
lmint@dboxh:~$
```

Wykonywanie skryptu

Uruchom terminal (wiersz poleceń) w katalogu zawierającym skrypt, który chcesz wykonać. Wydadź polecenie:

`python3 nazwa_skryptu.py` – które spowoduje wykonanie podanego skryptu przez interpreter.

```
lmint@dboxh: ~/Dokumenty  
Plik  Edycja  Widok  Wyszukiwanie  Terminal  Pomoc  
lmint@dboxh:~$ cd Dokumenty  
lmint@dboxh:~/Dokumenty$ python3 ile_liczb.py  
Podaj liczbę wprowadzanych liczb: 4  
Podaj liczbę: 12  
Podaj liczbę: 9  
Podaj liczbę: 5  
Podaj liczbę: 8  
Iloczyn liczb: 96  
Liczba parzystych: 2  
lmint@dboxh:~/Dokumenty$
```

Skrypty Pythona możesz uruchamiać również z poziomu edytora kodu, o ile oferuje taką funkcję. Na przykład w edytorze IDLE wczytany skrypt uruchomisz naciskając klawisz **F5**.

```
IDLE Shell 3.10.6  
File  Edit  Shell  Debug  Options  Window  Help  
Python 3.10.6 (main, Aug 10 2022, 11:40:04) [GCC 11.3.0] on linux  
Type "help", "copyright", "credits" or "license()" for more information.  
>>>  
===== RESTART: /home/lmint/Dokumenty/ile_liczb.py =====  
Podaj liczbę wprowadzanych liczb: 4  
Podaj liczbę: 12  
Podaj liczbę: 9  
Podaj liczbę: 5  
Podaj liczbę: 8  
Iloczyn liczb: 96  
Liczba parzystych: 2  
>>>
```

Konwencja formatowania kodu

- ciągi znaków otaczamy cudzysłowami albo pojedynczymi, albo podwójnymi

- nawiasy, o ile nie są częścią ciągu znaków, występują zawsze parami
- operatory otacza się spacjami
- dwukropek (:) zapowiada blok kodu, czyli wcięcia
- bloki kodu wcina się 4 spacjami i ich wielokrotnościami
- w wierszu powinny występować maksymalnie 72 znaki
- znak # oznacza komentarz, przy czym:
 - pierwszy wiersz skryptu może (nie musi) zawierać wskazanie interpretera (*shebang*), np.:
`#!/usr/bin/env python3`
 - skrypt może (nie musi) zawierać preferowane kodowanie znaków: `# -*- coding: utf-8 -*-`

Co zawiera kod źródłowy?

- **identyfikatory zmiennych**, tj. nazwy, które wskazuje na wartość określonego typu
- **identyfikatory słów kluczowych**, tj. nazwy zarezerwowane przez język, np.: `if`, `elif`, `else`, `for`, `while`
- **konkretne wartości** (tzw. literały), np. znakowe `"Adam"` lub liczbowe `10`, `2.345` – które mogą być przypisywane zmiennym
- **operatory**, np. arytmetyczne (`+`, `-`, `/`, `*`) lub logiczne (`<`, `>`, `==`, `!=`)
- **ograniczniki**, np. przecinek, dwukropek, nawiasy okrągłe, kwadratowe, klamrowe
- **wyrażenia**, tj. kombinacje zmiennych, wartości, operatorów i/lub wywołań funkcji, która są obliczane i dają jakąś wartość, np.: `int((a + b) / 2)`

Nazywanie zmiennych

- nazwa zaczyna się najczęściej od litery
- nazwa zawiera tylko znaki, cyfry i/lub podkreślenia, nie zawiera spacji i znaków narodowych
- rozróżniane są małe i duże litery
- używaj konwencji **snake_case** do nazywania zmiennych (także funkcji, metod, modułów i pakietów), czyli małych liter i wyrazów oddzielanych podkreśleniem, np. `moja_zmienna`

Typ danych (ang. *data type*)

Python wykorzystuje **typowanie dynamiczne** (ang. *duck typing*). Zmienne inicjuje się przez przypisanie im jakiejś wartości. Zmienne "przechowują" lub inaczej "wskazują na" wartości określonego typu.

- **ciąg znaków** (`<class 'str'>`, ang. *string*) – np. `"", ' ', " ", '12345'`
- **liczba całkowita** (`<class 'int'>`, ang. *integer*) – np. `10`,
- **liczba zmiennoprzecinkowa** (`<class 'float'>`, ang. *floating point*) – np. `1.2345`,
- **typ logiczny** (`<class 'bool'>`, ang. *boolean type*) – podtyp liczb całkowitych, zawiera dwie wartości: `True` (1, prawda) lub `False` (0, fałsz)

Inicjacja zmiennych

Inicjacja zmiennej wymaga operacji przypisania, tj. podania nazwy, operatora przypisania (=) oraz wartości określonego typu, np. ciągu znaków, liczby całkowitej itd. Wartość może być wynikiem wyrażenia lub funkcji.

- `imie = "Adam"` – inicjacja zmiennej zawierającej znaki
- `liczba = input("Podać liczbę: ")` – inicjacja zmiennej wartością zwróconą przez funkcję
- `a = 5, pi = 3.14` – inicjacja zmiennej zawierającej liczby
- `parzysta = False` – inicjacja zmiennej zawierającej wartość logiczną
- `iloczyn = (2 * a) + 1` – inicjacja zmiennej za pomocą wyrażenia

Operatory i ich priorytet

1. `**` – potęgowanie
2. `*`, `/`, `//`, `%` – mnożenie, dzielenie, dzielenie całkowite, dzielenie modulo (reszta z dzielenia)
3. `+`, `-` – dodawanie, odejmowanie
4. `<`, `<=`, `>`, `>=`, `!=`, `==` – operatory porównań, mniejsze, mniejsze lub równe, większe, większe lub równe, różne, równe
5. `not x` – operator logiczny, negacja
6. `and` – operator logiczny, koniunkcja
7. `or` – operator logiczny, alternatywa

Typowe operacje

- **łączenie ciągów znakowych** (tzw. konkatencja) za pomocą operatora `+`, np. `"10" + "1"`
- **operacje arytmetyczne** zgodne z kolejnością działań: 1. nawiasy, 2. potęgowanie, 3. mnożenie, 4. dzielenie, 5. dodawanie, 6. odejmowanie – i priorytetem operatorów, np.: `(a + 2.5) // 2**b`
- **testowanie wyrażeń logicznych**, które:
 - mają wartość **True** (1) lub **False** (0), przy czym za **False** uznaje się: `None`, `False`, `0`, puste sekwencje lub kolekcje, tj. `'`, `()`, `[]`, `{}`, `set()`, `range(0)`
 - wykorzystują operatory porównań (`<`, `<=`, `>`, `>=`, `!=`, `==`) i operatory logiczne `not`, `and`, `or`
 - są używane jako warunki, np.: `a == b` – operacja porównania
- **pobieranie danych z klawiatury** – instrukcja wejścia `input("komunikat dla użytkownika")` zwraca dane jako tekst (typ `str`!)
- **wypisywanie komunikatów** – instrukcja `print("komunikat")` wypisuje znaki lub zmienne w terminalu
 - `print("komunikat", a)` – wypisanie ciągu znaków i wartości zmiennej `a`
 - `print(*lista)` – wypisanie elementów listy (sekwencji)
 - `print(f"{a} jest większe od {b}")` – wypisanie ciągu formatowanego (tzw. *f-string*), w miejsce nawiasów klamrowych wstawiane są wartości zmiennych lub wyrażeń
- **rzutowanie typów danych**:
 - `int("123")` – zamiana tekstu na liczbę całkowitą
 - `float("2.5")` – zamiana na liczbę zmiennoprzecinkową

- `str(10)` – zamiana np. liczby na tekst

Instrukcja warunkowa

Ogólna forma:

```
if warunek1:
    instrukcja lub blok instrukcji wykonywanych gdy warunek1 jest prawda
elif warunek2:
    instrukcja lub blok instrukcji wykonywanych gdy warunek2 jest prawda
else:
    instrukcja lub blok instrukcji wykonywanych gdy warunki nie są
    prawdziwe
```

Przykłady:

```
# sprawdzamy, czy a jest parzyste
if a % 2 == 0:
    print("Liczba", a, "jest parzysta!")
else:
    print("Liczba", a, "nie jest parzysta!")
```

Forma z dodatkowym testem:

```
# sprawdzamy, czy a jest większe, czy równe b
if a > b:
    printf("Liczba {a} jest większa od {b}!")
elif a == b:
    printf("Liczby {a} i {b} są równe!")
else:
    printf("Liczba {a} jest mniejsza od {b}!")
```

Zagnieżdżanie instrukcji warunkowych:

```
# sprawdzamy, czy a jest większe od b i c
if a > b:
    # ewentualna instrukcja lub instrukcje
    if a > c:
        print("Liczba {a} jest największa.")
else:
    ewentualna instrukcja lub instrukcje
```

Zastąpienie zagnieżdżonych instrukcji warunkowych warunkiem złożonym:

```
if a > b and a > c:
    print("Liczba {a} jest największa.")
elif a == b or a == c:
    print("Liczba a jest równa b lub c.")
else:
    print("Liczba a jest mniejsza od b lub c.")
```

Instrukcje iteracji (pętle)

Przykłady pętli **for** (wykonuje się określoną liczbę razy):

```
# pętla, która wykona się 10 razy
for i in range(10):
    print(i) # powtarzana instrukcja wypisująca wartość i

# pętla, która odczyta wszystkie znaki z napisu
napis = "abcde"
for znak in napis:
    print(znak) # powtarzana instrukcja wypisująca kolejne znaki
```

Przykłady pętli **while** (wykonuje się, dopóki warunek jest prawdziwy):

```
# pętla, która wykonuje się dopóki warunek a > 0 jest prawdziwy
a = 10
while a > 0:
    a = a - 1

# pętla, która pobiera liczbę z klawiatury, dopóki liczba jest ujemna lub
równa 0
liczba = int(input("Podaj liczbę: "))
while liczba <= 0:
    liczba = int(input("Podaj liczbę: "))
```

Najczęstsze typy błędów

- **IndentationError** – błąd wcięcia, najczęściej polega na niezachowywaniu tych samych wcięć w bloku kodu
- **TypeError** – błąd typu, występuje, kiedy chcemy wykonać operację na złym typie danych
- **NameError** – błąd nazwy, występuje, kiedy używamy niezdefiniowanej nazwy, np. niezainicjowanej zmiennej, błędnie zapisanej funkcji itp.