Python

Programowanie w języku Python

Radosław Roszczyk

27 luty 2022

radek@people.pl

O czym będzie

1. O języku
2. Proste typy danych
3. Złożone typy danych
4. Instrukcje sterujące
5. Obsługa wyjątków w Python

O języku

Początki języka Python

Python powstał we wczesnych latach 90 jako następca języka ABC. Pierwotna wersja języka stworzona została w Centrum Matematyki i Informatyki w Amsterdamie.

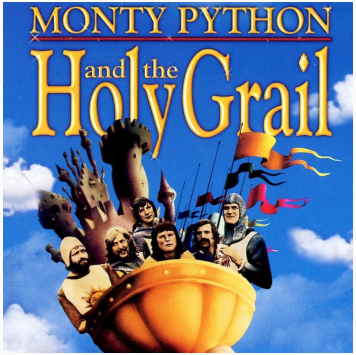
Bieżący rozwój

- Python Software Foundation (PSF)

Wersje języka

- **3.9 – październik 2025**
- 3.8 – październik 2024
- 3.7 – czerwiec 2023
- 3.6 – grudzień 2021
- *2.7 – styczeń 2020*

Rysunek 1: Guido van Rossum - szef zespołu tworzącego Pythona



Rysunek 2: Fragment plakatu promującego film Monty Python

Proste typy danych

Typy liczbowe

Typ całkowity (integer)

- Stałe: 256, 0x100, 0o400, 0b100000000
- Maksymalna wartość: 9223372036854775807*

Typ logiczny (boolean)

- **Prawda**: True, > 0, niepusty element
- **Fałsz**: False, 0, None, [],

Liczba zespolona (complex)

- Stałe: 30+30j, a+30j,
- Konstruktor: complex(30, 30), complex(a, b)

Liczba zmiennopozycyjna (float)

- Stałe: 3.141592, 100., .001, 10e4
- Maksymalna wartość: 1.7976931348623157e+308*

Typy znakowe

Typ znakowy (string)

- Stałe:
  - "tekst",
  - 'tekst',
  - """bardzo długi tekst
    w kilku linijkach"""
- Napisy indeksujemy od 0
- s[0], s[1], s[−1], s[1 : 4], s[3 : *len*(s)]

Podstawowe funkcje

- *chr*() – konwertuje liczbę na znak
- *ord*() – konwertuje znak na liczbę
- *len*() – zwraca długość napisu w znakach
- *str*() – reprezentuje obiekt w postaci znakowej

Konwersja między typami

Użycie nazwy typu jako operatora rzutowania

- $int(3.141592)$
- $float(7)$
- $complex(5, 15)$
- $bool(5)$
- $str(4543)$

Konwersja podczas wykonania

- $1/4 * 3 \rightarrow 0,75$
- $int(1/4 * 3) \rightarrow 0$
- $1 * 4//3 \rightarrow 1$
- $1 * 4/3 \rightarrow 1.3333333333333333$
- $int("1234") * 10 + 5 \rightarrow 12345$
- $int("15") * 4 \rightarrow 60$
- $"15" * 4 \rightarrow 15151515$

Złożone typy danych

Złożone typy danych

Kolekcje

- Listy: [1, 2, 3]
- Krotki: (7, ”napis”, ”drugi”, (1, 2, 0x7), 30+30j)
- Słowniki: ”jeden”: 1, ”dwa”: 2, ”trzy”: 3
- Zbiory
- Obiekty

Listy

Lista jest podstawową strukturą danych służącą do reprezentacji danych dynamicznych.

Przykładowe listy

```
1 lista1 = [1, 2, 3]
2 lista2 = ["a", "b", "c"]
3 lista3 = ["4", lista2, "napis", 5]
```

Alternatywne tworzenie list

```
1 lista_znakow = list(("1", "2", "3"))
2 lista_liczb = list((1, 2, 3))
```

Operacje na listach

Inicjalizacja listy

```
1 kolekcja = [i for i in range(0, 10, 2)]
2 kolekcja = [i * i + 2 for i in range(0, 10, 2)]
3 parzyste = [i for i in range(0, 20) if i % 2 == 0]
```

Przekształcenie list

```
1 kolekcja_orig = ['kot', 'pies', 'chomik']
2 nowa_kolekcja = ['mój '+i for i in kolekcja_orig]
```

Krotki

Krotka jest strukturą przechowują stałe wartości o różnych typach danych. Krotki są strukturami niemodyfikowalnymi.

Przykładowe krotki

```
krotka1 = ('jeden', 'dwa', 'trzy', 'cztery', 'siedem', 'osiem')
krotka2 = (1, 2, 3, 4, 5 );
krotka3 = "a", "b", "c", "d"
krotka4 = (1, 'dwa', [3, 4.5])
krotka5 = ()
krotka6 = (50,)
```

Odwoływanie się do poszczególnych elementów

```
print("krotka1[0]: ", krotka1[0])
print("krotka1[1:5]: ", krotka1[1:5])
```

Operacje na krotkach

Funkcje wbudowane

```
tup = ('jeden', 'dwa', 'trzy', 'cztery', 'piec', 'szesc')

# len() zwraca liczbe elementów krotki
print(len(tup))

# index() zwraca indeks pierwszego wystapienia elementu w krotce
print(tup.index('dwa'))

# count() zwraca liczbe wystapien elementu w krotce
print(tup.count('dwa'))

# min() i max() zwracaja najmniejszy i najwiekszy element krotki
# sum() zwraca ęsum elementów krotki
tupN = (1,2,3)   # dla tych metod krotka musi zawierac tylko liczby
print(min(tupN))
print(max(tupN))
print(sum(tupN))
```

Po co stosować krotki

Zalety

- Krotki stosujemy dla danych tylko do odczytu, np: lista dni tygodnia
- Krotki mogą przechowywać różne typy danych
- Krotki są znacznie szybsze w obliczeniach niż listy

Porównanie wydajności krotek i list

```
roszczyr@laptop ~ → python3 -m timeit "x=(1,2,3,4,5,6,7,8)"
50000000 loops, best of 5: 8 nsec per loop
roszczyr@laptop ~ → python3 -m timeit "x=[1,2,3,4,5,6,7,8]"
5000000 loops, best of 5: 42.1 nsec per loop
roszczyr@laptop ~ →
roszczyr@laptop ~ → python3 -m timeit -s "x=(1,2,3,4,5,6,7,8)" "y=x[3]"
20000000 loops, best of 5: 19.8 nsec per loop
roszczyr@laptop ~ → python3 -m timeit -s "x=[1,2,3,4,5,6,7,8]" "y=x[3]"
10000000 loops, best of 5: 20.8 nsec per loop
```

Słowniki

Słownik jest jednym z wbudowanych typów danych i przechowuje dane w postaci pary: klucz – wartość.

Przykładowe słowniki

```
1 slownik_plen = { "kon": "horse", "pies": "dog", "matematyka": "mathematics" }
2 moj_slownik = {'klucz1': 'wartosc1', 'klucz2': 'wartosc2'}
3 slow1 = {'klucz1': 'napis', 'klucz2': 123, 'klucz3': ['i1', 'i2', 'i3']}
```

Alternatywne tworzenie słowników

```
1 moj_slownik1 = {}
2 moj_slownik1['klucz1'] = 'wartosc1'
3 moj_slownik1['klucz2'] = 'wartosc2'
```

Operacje na słownikach

Usuwanie elementów

```
del moj_slownik['klucz1']
moj_slownik.pop('klucz2')
moj_slownik.clear()
```

Funkcje wbudowane

```
dict1 = {'k1': 'w1', 'k2': 'w2'} # zdefiniowanie slownika

# Sprawdzanie liczby kluczy w slowniku
len(dict1)

# Zwracanie listy elementów slownika
dict1.items()   # wyswietlenie wszystkich elementow slownika: par klucz - wartosc
dict1.keys()    # wyswietlenie wszystkich kluczy slownika
dict1.values()  # wyswietlenie wszystkich świartoci slownika
```

Instrukcje sterujące

Instrukcja warunkowa

Instrukcja warunkowa I

```
x = 3
if x > 0:
    print('dodatnia')
elif x < 0:
    print('ujemna')
else : print('zero')
```

Instrukcja warunkowa II

```
if punkty >= 90:
    print("ocena 5")
elif punkty >=80:
    print("ocena 4")
elif punkty >=70:
    print("ocena 3")
elif punkty >= 65:
    print("ocena 2")
else:
    print("ocena 1")
```

Zastosowanie instrukcji warunkowej do inicjalizacji listy

```
tp = ['L: ' + str(i) + ' parzysta' if i % 2 == 0 else
      'L: ' + str(i) + ' nieparzysta' for i in range(0, 20)]
print(tp)
```

Instrukcje pętli

Instrukcja for I

```
1 a = [1, 2, 3, 4]
2 for e in a:
3   print(e)
4 print('koniec')
```

Instrukcja for II

```
1 suma = 0
2 for i in range(100):
3   suma = suma + i
4 print('suma = ', suma)
```

Zastosowanie instrukcji for do inicjalizacji listy

```
1 parzyste = [i for i in range(0, 20) if i % 2 == 0]
2 print(parzyste)
```

Instrukcje pętli

Instrukcja while-else I

```
liczba = 0
while liczba < 5:
  print("W petli")
  liczba+=1
else:
  print("Koniec przez 'else'")
```

Instrukcja while-else II

```
liczba = 3
while liczba < 20 :
  print("W ęptli")
  if (liczba <= 0):
    break
  liczba += 1
  liczba *= liczba
else:
  print("Koniec przez 'else'")
print("Koniec petli")
```

Pętla pusta

```
while (True): pass  # brak mozliwosci tworzenia pustych bloków
```

Instrukcje pętli

Instrukcja while-else II

```
liczba = 3
while liczba < 20 :
  print("W ęptli")
  if (liczba <= 0):
    break
  liczba += 1
  liczba *= liczba
else:
  print("Koniec przez 'else'")
print("Koniec petli")
```

Instrukcja while-else III

```
liczba = -3
while liczba < 20 :
  print("W ęptli")
  if (liczba <= 0):
    break
  liczba += 1
  liczba *= liczba
else:
  print("Koniec przez 'else'")
print("Koniec petli")
```

Pętle while-else wykorzystujemy gdy chcemy wykonać określoną liczbę powtórzeń pętli oraz zakończyć je pewnym kodem. Dopuszczamy jednak możliwość, że w trakcie wykonywania pętli dojdzie do sytuacji, kiedy ta dodatkowa instrukcja nie powinna się wykonać.

Klauzula with

Klauzula with służy do pracy z tzw. kontekstami. Najprościej można je rozumieć jak tworzenie bloku do pracy z pewnym zewnętrznym względem Pythona narzędziem. Przykładem może tu być praca z otwartym plikiem na dysku, zasobem internetowym, czy innym programem systemu operacyjnego.

Przykład

```
1 with open('readme.txt', 'r') as plik:
2     dane = plik.read()
3     print('Czy plik jest ęzamknity ', plik.closed)
4 print(dane)
5 print('Czy plik jest ęzamknity ', plik.closed)
```

Obsługa wyjątków w Python

Obsługa wyjątków

Python podobnie jak większość języków programowania pozwala na tworzenie, zgłaszanie i obsługiwanie wyjątków.

Czym jest wyjątek? Jest sytuacją w działaniu programu, która uniemożliwia poprawne wykonanie części skryptu. Podobnie jak w Javie, kod który może nie zakończyć się poprawnym wykonaniem instrukcji w bloku try

Przykład

```
1 tekst = "nie liczba"
2 try :
3   x = int(tekst)
4 except ValueError:
5   print("Zla wartosc, ale")
6 print("Ale mimo to udalo sie")
```

Definiowanie wyjątków

```
1 class MyException(Exception):
2     pass
3 def call_exception():
4     raise MyException()
5 try:
6     call_exception()
7 except MyException:
8     print("Przechwycono wyjatek")
```

Obsługa wyjątków

Python posiada podobnie do Javy klauzulę finally, czyli blok który musi się wykonać niezależnie do tego czy wyjątek został czy nie został zgłoszony.

Przykład

```
1  def dzielenie(x, y):
2      try:
3          print("Dzielenie " + str(x) + "/" + str(y))
4          result= x/y
5      except ZeroDivisionError:
6          print("Blad dzielenia")
7      else:
8          print("Wynik to " + str(result))
9      finally:
10         print("A ten blok wykona sie zawsze")
11
12 dzielenie(2, 3)
13 dzielenie(1, 0)
```

Pytania?