Laboratorium – Python – PPY – PJATK 2023

Zajęcia 2

15.03.2023

1 Podstawy programowania

- Normy dotyczące języka Python znajdują się w *Python Enhancment Proposals* (**PEPs**);
- Każdy może dorzucić własny pomysł.
- PEP 20 – "*The Zen of Python*"jest niezwykle ciekawym przykładem:

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

- Aby go wywołać, należy wpisać komendę:

```
import this
```

- Python jest językiem interpretowanym. Co to oznacza?
- Jak sprawdzić typ danego obiektu?

```
print(type(47))  # Zwrócenie: "<class 'int'>"
```

1.1 Wprowadzenie – piersze "kody"za płoty!

- Pierwszy program w stylu "Hello World!":

```
print("Max lubi gry komputerowe") #dowolne wyrażenie w ""
```

- Dodawanie:

```
print(1+1)
```

Wynik: **2**. Jednakże:

```
print("1"+"1")
```

Wynik: **11**. A:

```
print("1"+1)
```

Error! Błąd. Jak Państwo myślą – dlaczego?

1.2 Słowa kluczowe

W języku Python, jak w każdym języku programowania, występują słowa, których NIE używamy jako nazw zmiennych, typ klas itp, a są to:

```
and       del       for       is        raise
assert    elif      from      lambda    return
break     else      global    not       try
class     except    if        or        while
continue  exec      import    pass      yield
def       finally   in        print
```

1.3 Przykłady nazywania zmiennych

Zmiennie w Pythonie możemy nazywać np. tak:

```
jankowalki = 0
jankowalski2 = 0
jankowalski_3 = 0
jan_kowalski = 0
_jankowalki = 0
_jan_kowalski = 0
janKowalski = 0
JanKowalski = 0
JANKOWALSKI = 0
JAN_KOWALSKI = 0
```

Nie możemy dac '1jankowalski' !

1.4 Przypisywanie zmiennych

Zmienne można przypisać następująco:

```
x = 1
x, y, z = 1, 2, 3
x, y = z, 1
x = y = z = 1
```

- Wskazówki dotyczące typów można znaleźć w **PEP 484** (`https://peps.python.org/pep-0484/`):

```
x: int
x: int = 1
x = 1  # type: int
x = "jeden"  # Ostrzeżenie: "Expected type 'int', got 'str' instead"
```

1.5 Typy zmiennych

Używane typy zmiennych:

- Ciąg znaków – string (str):

  ```
  x = "Raz, dwa, trzy! Baba Jaga patrzy"
  x = 'a'
  ```

- Liczba całkowita – integer (int):

  ```
  x = 20
  ```

- Wartość zmniennoprzecinkowa – floating point (float):

  ```
  x = 20.5
  ```

- Wartość logiczna (boolean):

  ```
  x = True
  ```

- PUSTE (NoneType):

  ```
  x = None
  ```

1.6 Operatory

- Dodawanie, odejmowanie, mnożenie i dzielenie:

```
x = y + 1 -1 * 1 / 1
```

- Reszta z dzielenia (modulo):

```
x = y % 1
```

- Dzielenie całkowite:

```
x = y // 1
```

- Potęgowanie:

```
x = y ** 2
```

1.6.1 Pozostałe operatory

x += y $\iff$ x = x + y
x -= y $\iff$ x = x - y
x *= y $\iff$ x = x * y
x /= y $\iff$ x = x / y
x %= y $\iff$ x = x % y
x //= y $\iff$ x = x // y
x **= y $\iff$ x = x ** y

- **x** = **True**, jeżeli **x** jest równe **y**:

```
x = x == y
```

- **x** = **False**, jeżeli **x** nie jest równe **y**:

```
x = x != y
```

2 Podstawowe operacje

Jak deklarujemy **zmienne**?

- Deklaracja jednej zmiennej

```
zm = 5
print(zm)
```

lub

```
zm = "dowolny napis"
print(zm)
```

lub

```
zm = "Testujemy " + "1"
print(zm)
```

- Deklaracja wielu zmiennych:

```
a, b, c = 2, 4, 6
print(c)
```

lub

```
k = (1, 2, 3)
(x, y, z) = k
print(x)
```

- Usuwanie zmiennej (musimy ją uprzednio zdefiniować):

```
del x
print(x)
```

- Powielanie znaków:

```
print("P" + "y" * 3 + "thon rzadzi!")
```

- Konkatenacja:

```
print("Czerwony " + "rower")
print('Row' 'er')
```

- Apostrof:

```
napis = 'I\'m Max'
print(napis)
```

- Tabulator oraz znak nowego wiersza:

```
napis2 = "Tekst z tabulatorem\ti znakiem\n nowego wiersza"
print(napis2)
```

- Nowe wiersze:

```
napis3 = '''wiersz1
wiersz2
wiersz3'''
print(napis3)
```

2.1 Operacje na zmiennych

- Przypisanie zmiennej do nowego obiektu typu int:

```
x = 1000  # Ekwiwalent operacji 'x = int(1000)'
x = 1_000  # Inny zapis '1000' - Znak '_' zwiększa czytelność dużych liczb
```

- Wykorzystanie operatorów arytmetycznych:

```
x = (x + 1 - 2) * 3  # Dodawanie, odejmowanie, mnożenie
x = x // 10  # Dzielenie całkowite
x = (x % 7) ** 2  # Dzielenie modulo, potęgowanie
x = x / 3  # Dzielenie - Domyślnie wynik jest zawsze wyrażany typem 'float'
print(x, type(x))
```

- Zmiana typu liczby przy dzieleniu:

```
x = int(x / 3)  # Zmiana na int.
print(x, type(x))
```

- Zapisy liczb zmiennoprzecinkowych:

```
q, x = 1000.0, 1_000.0
y, z = 1e3, 1e-3  # Zapis z wykorzystaniem notacji naukowej, '1e3' == '1*10**3'

print(q, x, y, z)
print(type(q), type(x), type(y), type(z))

x = 1000000
y = 1_000_000
print(x, y)
print(type(x), type(y))

q, x, y, z = 1000.0, 1_000.0, ie3, 1e_3
print(q, x, y, z)
print((type(q), type(x), type(y), type(z))
```

2.2 Przykłady ciągów i ich znaków

- Znaki, łańcuchy można zamykać pomiędzy (') lub ("):

```
x = 'x'
y = "x"
print(x, y, type(x), type(y))
```

- Można wykorzystywać zarówno apostrofy, jak i cudzysłowy w tych samych łańcuchach:

```
x = "I'm Piotr!"
print(x)
x = 'Piotr powiedział: "Uczcie się, a będzie wam zdane!"'
print(x)
```

- Można użyć nawet potrójnego apostrofu lub cudzysłowu dla wielowierszowych łańcuchów:

```
x = '''"Szaleństwem jest robić wciąż to samo i oczekiwać różnych rezultatów."
- Albert Einstein'''
print(x)
%
```

- Można także tak zamykać łańcuchy:

```
x = '''Powiedział, "Dziękuję! Naprawdę!"'''
print(x)
```

- Operacje na łańcuchach znakowych, formatowanie i konwersja typów:

```
napis = "Wiek: " + str(38)
print(napis)
```

- Zamiana znaków w tekście:

```
print(napis.replace("W", "w"))

print(napis.lower())

print(napis.upper())
```

- Formatowanie tekstu (dla Python 3!):

```
napis = "Liczba {0} to {1}".format(3.14, "pi")
print(napis)
```

- Łączenie łańcuchów z użyciem operatora dodawania:

```
x, y = '123', '456'
z = x + y
print(x, y, z)
```

- Łączenie łańcuchów z użyciem operatora mnożenia:

```
q = z * 2
print(q)
```

- Wybieranie konkretnego elementu z łańcucha:

```
element = q[0]  # Wybranie pierwszego elementu łańcucha 'q'
elements = q[0:3]  # Wybranie pierwszych trzech elementów łańcucha 'q'
print(element, elements)

vector = "1234" + " 4567"
print(vector type(vector))
new_vector_ = vector * 2
print(new_vecotr, type(new_vector))
new_vector = "USER_ID:" + vector * 2
print(new_vecotr, type(new_vector))
element = new_vector[12]
print(element, type(element))
```

- Podmienianie znaków w łańcuchu:

```
name = "Marian Kowalski"
new_name = name.replace("Marian", "Kowalski")
```

- Wycinanie części łańcuchów:

```
IP = "IP: 192.168.1.1"
raw_IP = IP[3:]
print(raw_IP, type(raw_IP))
```

- Dzielenie łańcuchów z wykorzystaniem metody 'split()':

```
IP = "IP: 192.168.1.1"
raw_IP = IP.split(' ')
print(raw_IP, type(raw_IP))

splitted = q.split('4')  # Podzielenie łańcucha 'q' w miejscu występowania '4'
print(splitted, type(splitted))  # Metoda 'split()' zwraca nowy typ danych - listę
```

- Konwersja typu tekstowego na liczbowy:

```
a = '3'
b = 9
print(int(a) + b)
```

- Konwersja typu liczbowego na tekstowy:

```
a = '8'
b = 10
print(a + repr(b))
print(a + str(b))
```