Stałe, zmienne i typy danych

Stałe i zmienne

Deklarowanie zmiennych

```
my_variable = 10
my_variable = 'ehlo world'
```

Deklarowanie stałych

```
MY_CONSTANT = 10
MY_CONSTANT = 'ehlo world'
```

Różnica między stałymi i zmiennymi

Jedyną różnicą jest konwencja nazewnicza:

- stałe zapisujemy dużymi literami
- zmienne zapisujemy małymi literami

Zasięg widoczności

- `globals()`
- `locals()`

Numeryczne typy danych

`int` - Liczba całkowita

Jednym z najbardziej podstawowych typów danych jest `int`. `int()` jest funkcją wbudowaną, która zamieni swój argument na liczbę całkowitą.

```
>>> age = 10

>>> int(10)
10

>>> int(10.0)
10

>>> int(10.9)
10
```

`float` - Liczba zmiennoprzecinkowa

`float` w Pythonie reprezentuje liczbę zmiennoprzecinkową. Ciekawą własnością tego typu jest możliwość reprezentacji nieskończoności za pomocą `Infinity` oraz minus nieskończoności `-Infinity`. Więcej szczegółów dostępnych jest w dokumentacji dla tego typu

Podobnie jak pozostałe typy `float()` jest funkcją, która konwertuje swój argument na liczbę zmiennoprzecinkową.

```
>>> float(10)
10.0

>>> float('+1.23')
1.23

>>> float('   -12345\n')
-12345.0

>>> float('1e-003')
0.001

>>> float('+1E6')
1000000.0

>>> float('-Infinity')
-inf
```

`complex` - liczba zespolona

`complex` reprezentuje typ liczby zespolonej posiadającej część rzeczywistą oraz urojoną. Należy zwrócić uwagę, że argument powinien być ciągiem znaków niezawierającym spacji. W przeciwnym przypadku otrzymamy `ValueError`.

```
>>> complex('1+2j')
(1+2j)

>>> complex('1 + 2j')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: complex() arg is a malformed string
```

Tekstowe typy danych

`str` - Ciąg znaków

Obiekt typu `str` przechowuje łańcuch znaków. `str()` jest także funkcją, która zwraca ciąg znaków z argumentu.

```
>>> name1 = 'José'
'José'

>>> name2 = "Ivan"
'Ivan'

>>> print("""
... Max Peck
... """)
'\nMax Peck\n'

>>> str(10)
'10'
```

Escape'owanie znaków

```
"""
\xac
\u7723
\n
\b
\r
\t
\'
"""
```

Znaki przed stringiem

```
u'zażółć gęślą jaźń'
r'(?P<foo>)\n' # escapes does not matters
r'C:\Users\Admin\Desktop\foobar.txt'
f'hello {first_name}, how are you?'
b'this is text'
```

Niemutowalność

Ważną cechą ciągów znakowych jest tzw. niemutowalność. Gdy wykonujemy operację na stringu tworzona jest jego nowa kopia.

Pojedynczy czy podwójny cudzysłów

Python nie rozróżnia czy stosujemy pojedyncze znaki cudzysłowiu czy podwójne. Ważne jest aby wybrać jedną konwencję i się jej konsekwentnie trzymać.

Interpreter Pythona domyślnie stosuje pojedyncze znaki cudzysłowia, z tego powodu w tym materiale będziemy trzymać się tej konwencji.

Operacje na stringach

- `strip()`, `lstrip()`, `rstrip()`

- `join()`
- `startswith()`
- `title()`
- `replace()`

Wycinanie części stringów

```
>>> text = "Lorem ipsum"

>>> text[2]
'r'

>>> text[:2]
'Lo'

>>> text[0:3]
'Lor'

>>> text[-3]
's'

>>> text[-3:]
'sum'

>>> text[-3:-1]
'su'

>>> text[:-2]
'Lorem ips'
```

`io`

`io` to biblioteka do obsługi strumienia wejściowego i wyjściowego. StringIO jest wtedy traktowany jak plik wejściowy.

```
import io

io.StringIO
```

Logiczne typy danych

`bool` - Wartość logiczna

Obiekt typu `bool` może przyjąć dwie wartości logiczne:

- `True`
- `False`

Zwróć uwagę na wielkość liter!

`bool()` to także funkcja wbudowana w język Python, która zwraca wartość logiczną wyrażenia.

`None` - Wartość pusta

Ważne: nie jest to wartość `False` ani `0`. Wyobraź sobie, że masz bazę danych z użytkownikami. Gdy użytkownik nie poda wieku, to jest to wartość `None`.

```
wiek = None

if wiek is None:
    print('użytkownik nie podał wieku')
```

Przykłady praktyczne

Dla każdego z poniższych przykładów wykonano funkcję `type(what)` i wynik pokazano poniżej. Dla czytelności przykładu pominięto tę linijkę.

```
>>> what = 'foo'
<class 'str'>

>>> what = 'foo',
<class 'tuple'>

>>> what = ('foo')
<class 'str'>

>>> what = ('foo',)
<class 'tuple'>
```

```
>>> what = 10
<class 'int'>

>>> what = 10.5
<class 'float'>

>>> what = .5
<class 'float'>

>>> what = 10.
<class 'float'>

>>> what = 10,
<class 'tuple'>

>>> what = 10, 20
<class 'tuple'>

>>> what = (10, 20)
<class 'tuple'>

```

```
>>> what = (10,)
<class 'tuple'>
```

```
>>> what = {}
<class 'dict'>

>>> what = {'id'}
<class 'set'>

>>> what = {'id': 1}
<class 'dict'>


>>> a = {}

>>> isinstance(a, dict)
True

>>> isinstance(a, set)
False

>>> isinstance(a, (set, dict))
True
```

Zadania kontrolne

Zmienne i typy

Napisz program, który poprosi użytkownika o imie i ładnie go przywita wyświetlając 'hello IMIE'. Zamiast spacji użyj przecinka

Podpowiedź

: - Użyj podawania stringów po przecinku `print(str, str)` oraz parametru `sep` - Użyj f-string formatting dla Python >= 3.6

Zmienne i wczytywanie ciągu od użytkownika

Napisz program, który poprosi użytkownika o wiek i wyświetli wartość. Następnie sprawdzi pełnoletność i wyświetli informację czy osoba jest "dorosła" czy "niepełnoletnia".

Liczby całkowite

Napisz program, który wczyta od użytkownika liczbę i wyświetli informację, czy jest to liczba całkowita, czy niecałkowita.

Podpowiedź

: Liczba całkowita to taka, której część dziesiętna nie występuje (`int`) lub jest równa zero `float` .