

Python #4: Typy i zmienne

Typy

Do tej pory we wpisach przewijały się typy liczbowe (int i float) oraz stringi zawierające tekst, ale do dyspozycji mamy więcej typów takich jak:

- Liczby (Numbers)
- Teksty (String)
- Typ logiczny (Bool)
- Listy (List)
- Krotki (Tuple)
- Słowniki (Dictionary)

Liczby

Liczby w Pythonie mogą być dwójakiego typu – **int** (*integer* = l. całkowita) i **float** (*floating point number* l. zmiennoprzecinkowa).

Np.

Liczbą typu int będzie 4, która jest liczbą całkowitą, natomiast 4.4 jest już liczbą z przecinkiem, co nie powinno budzić niczych wątpliwości. Liczbą typu float jest też 62.1E+6, gdzie E odpowiada notacji do potęgi 10 czyli 62.1E+6 oznacza $62.1 * 10^6$, ten zapis jest w Polsce popularniejszy.

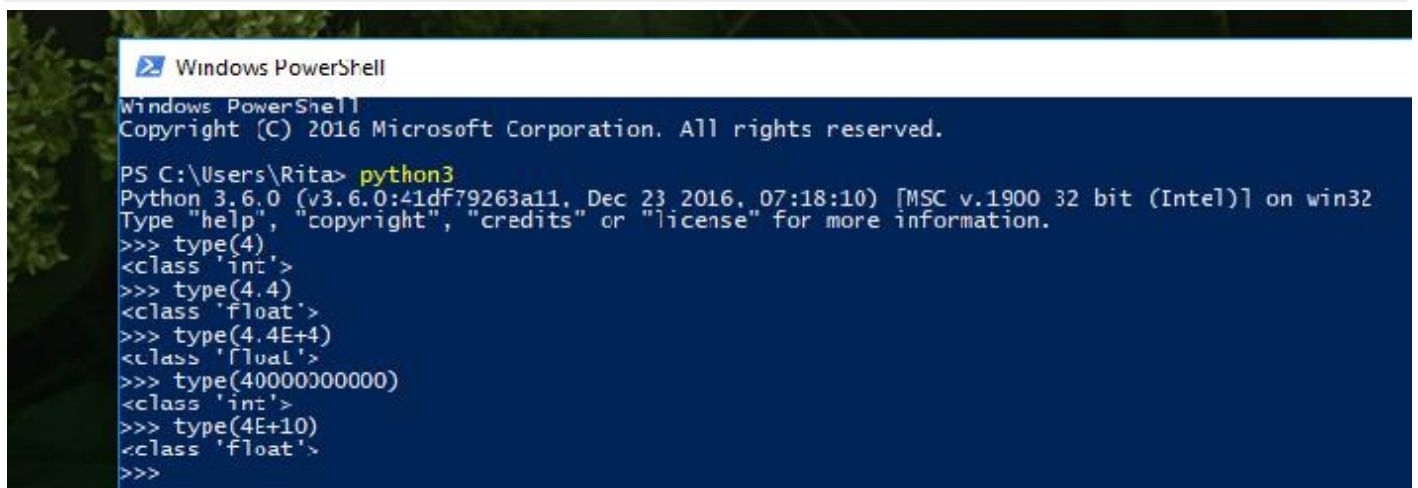
Z innych języków programowania:

W Pythonie nie ma osobnego typu long. Typ int może „pomieścić” liczby całkowite dowolnej wielkości.

Szybkie sprawdzenie czy nie kłamię:

```
>>> type(4)
>>> type(4.4)
```

```
...
```



```
Windows PowerShell
Copyright (C) 2016 Microsoft Corporation. All rights reserved.

PS C:\Users\Rita> python3
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> type(4)
<class 'int'>
>>> type(4.4)
<class 'float'>
>>> type(4.4E+4)
<class 'float'>
>>> type(400000000000)
<class 'int'>
>>> type(4E+10)
<class 'float'>
>>>
```

Teksty

String jest sekwencją znaków, a zazwyczaj przechowują słowa, zdania. Nie ma skryptu, w którym nie użyjecie stringów, dlatego warto poświęcić im więcej uwagi.

Cudzysłów pojedynczy

String możemy zapisać umieszczając w pojedynczym cudzysłowie i wygląda to tak `'Hej, zawieram tekst'`. Wszystkie białe znaki tzn. spacje czy tabulatory są wyświetlane dokładnie tak, jak zostały zapisane.

Cudzysłów podwójny

Działa tak samo jak pojedynczy, należy tylko pilnować, by początek i koniec rozpoczynał się tym samym znakiem `"Dzień dobry!"`.

Cudzysłów potrójny

Możesz też tworzyć teksty wielolinijkowe za pomocą trzech cudzysłów – (`'''` lub `'''`). Jeśli chcesz, możesz użyć pojedynczych cudzysłówów i podwójnych cudzysłówów wewnątrz tekstu znajdującego się w potrójnym cudzysłowie.

Przykład:

```
>>> quote='''To jest wielolinijkowy tekst
zawierający cytāt
"Always code as if the guy who ends up maintaining your code will be a
violent psychopath who knows where you live"
-John Wood'''
>>> print(quote)
```

Stringi są niemutowalne

Niemutowalne = niezmiennie. Oznacza to, że po utworzeniu tekstu nie można go bezpośrednio zmodyfikować. Osobom, dla których Python jest pierwszym językiem nie powinno to sprawiać problemu.

Dla programistów C/C++:

W Pythonie nie spotkacie typu `char`. Tak naprawdę, to nawet nie ma powodu, by był. Nie będziecie za nim tęsknić.

Dla programistów Perl i PHP:

Łańcuchy znaków w pojedynczych i podwójnych cudzysłowach są takie same. Nie, niczym się nie różnią, można używać je zamiennie.

Formatowanie stringów

Czasem możemy chcieć zbudować treść w oparciu o inne informacje. W takim wypadku przydaje się metoda `format()`.

Skoro jesteśmy już przy formatowaniu, warto zwrócić uwagę na jeszcze jeden niuans.

Zauważmy, że `print` zawsze zakończony jest „niewidzialnym” znakiem nowej linii (`\n`).

Oczywiście, można się go po prostu pozbyć.

Jeśli w pliku zapiszemy:

```
print('czeko', end='')  
print('lada', end='')
```

To na wyjściu otrzymamy

```
czekolada
```

Dodatkowo znak nowej linii można zamienić na np. spację:

```
print('czeko', end=' ')  
print('lada', end='')
```

To na wyjściu otrzymamy

```
czeko lada
```

Stringi mogą być raw

Jeśli chcesz wyświetlić czy wykorzystać tekst, w którym nie będzie obsługi znaków specjalnych, takich jak sekwencje escapowane, warto użyć surowego łańcucha znaków poprzedzonego `r` lub `R`.

```
>>> txt = r"Nowe linie zapisujemy jako \n a tabulatory za pomocą \t"  
>>> print(txt)  
Nowe linie zapisujemy jako \n a tabulatory za pomocą \t
```

Typ logiczny

Wartości logiczne – Prawda / Fałsz są przechowywane w typie logicznym. O tym jak go używać dowiesz się już wkrótce.

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

Listy i krotki

Listy i krotki to złożone typy danych.

Listy przechowują różne element poprzedzielane przecinkiem umieszczone w nawiasie kwadratowym. Elementy na liście mogą mieć różne typy.

```
>>> Lista = [3,5,"herbata", False]
>>> Lista
[3,5,'herbata', False]
>>> type(Lista)
<class 'list'>
```

Listy pozwalają na modyfikacje elementów, możemy podmienić element na liście. Listy są indeksowane (numerowane) od 0 – oznacza to, że żeby dostać się do pierwszego elementu musimy wpisać `Lista[0]`.

Aby podmienić pierwszy element na liście wystarczy `Lista[0] = 4`.

Krotki to listy, które są **niezmienne**. Nie można zmodyfikować elementów zawartych w krotce. Krotka, tak jak stringi jest niemutowalna. Krotki umieszczamy w nawiasach okrągłych.

```
>>> Krotka = (3,5,"kawa", True)
>>> Krotka
(3,5,'kawa', True)
>>> type(Krotka)
<class 'tuple'>
```

Możemy wyświetlić element w krotce wg jego indeksu `Krotka[2]`, ale nie mamy możliwości jego bezpośredniej modyfikacji `Krotka[2] = 12` (dostajemy błąd).

Słowniki

Słowniki są bardzo przydatnym typem danych. W innych językach programowania pojawiają się jako *tablica haszująca*, *tablica asocjacyjna* czy *mapa*. Przechowuje dane w parach jako **klucz : wartość**, z czego klucz musi być niemutowalnego typu oraz unikalny w obrębie słownika (klucz nie może się powtarzać!).

Przykład słownika:

```
>>> Słownik = { "kawa": "czarna", "herbata" : "zielona", "cukier kostka": 4, 0: "zero"}
>>> Słownik["herbata"]
zielona
>>> Słownik[0]
zero
>>> Słownik[1] = "jeden"
>>> Słownik = { "kawa": "czarna", "herbata" : "zielona", "cukier kostka": 4, 0: "zero", 1:
"jeden" }
>>> type(Słownik)
<class 'dict'>
```

Słownik jest strukturą nieuporządkowaną, w związku z czym nie można odwołać się do elementów słownika po indeksie. Jeśli istnieje klucz o wartości 0, otrzymamy wartość słownika jak w przykładzie powyżej niezależnie, którym z kolei elementem w słowniku jest para **0 : "zero"**.

Warto wiedzieć, że jeśli spróbujemy **przypisać wartości do nieistniejącego klucza**, **spowoduje to automatyczne powiększenie słownika** – stworzenie takiego klucza i przypisanie mu wartości.

Sprawdź też działanie:

Słownik.keys() – zwraca klucze w słowniku

Słownik.values() – zwraca wartości w słowniku

"herbata" in Słownik – zwróci prawdę jeśli klucz występuje w słowniku.

Porównaj wynik **Słownik.keys()** i **list(Słownik.keys())**.

list() spowoduje przekształcenie obiektu dict_keys na listę.

Zmienne i ich nazwy

Zmienne są dokładnie tym, co sugeruje nazwa – ich wartość może się zmieniać. To „konstrukty programistyczne”, służące do przechowywania danych. Zmienna w Pythonie może przechowywać dowolne wartości.

Dla programistów C/C++:

W Pythonie wszystkie wartości są przekazywane przez referencję. Typy są dynamiczne, co oznacza, że zmiennej nie deklarujemy typu. W momencie nadania wartości zmienna będzie przechowywać dany typ, dopóki, w trakcie działania kodu nie zdecydujemy się przypisać do niej innej wartości (wraz z jej typem).

Zmienne są przechowywane w pamięci komputera. Najprościej można je sobie wyobrazić pudełka, do których coś wkładamy. W pudełkach zapisywane pewne informacje, a aby otrzymać do nich dostęp (do zmiennych) musimy odwołać się do nich po nazwie.

Jak prawidłowo nazywać zmienne w Pythonie?

Zmienne trzeba jakoś zidentyfikować. Jak? – Najprościej, po nazwie!

Wracam znowu do zdania: *zmienne mogą mieć prawie dowolne nazwy*. Istnieją pewne reguły, których należy przestrzegać przy nazywaniu zmiennych:

- pierwszym znakiem **musi być litera alfabetu**, mała bądź wielka litera (ASCII lub Unicode – co oznacza, że możecie używać nie tylko znaków alfabetu łacińskiego, ale również polskich znaków, znaków z innych języków, czego mimo wszystko nie polecam!) lub **podkreślenie** (`_`).
- reszta może składać się z liter, podkreślenia (`_`), a także cyfry (0-9)
- nazwy zmiennych są „case-sensitive” – ważna jest wielkość liter.
Dlatego `myvariable`, `myVariable` i `MyVariable` to 3 różne nazwy zmiennych

Dobre przykłady nazw zmiennych: `i`, `my_name`, `name_2_3`, `_variable`.

Przykładowo błędne nazwy: `4Ucode`, `3words`, `my-name`, `oh space`, `>zmienna_z_dziubkiem`, `-sthNew`.

To wartość, a nie zmienna, posiada typ, dlatego też zmienna nie może istnieć bez wartości.

```
>>> myVariable
Traceback (most recent call last):
File „<stdin>”, line 1, in
NameError: name 'myVariable' is not defined
>>> myVariable = 5
>>> myVariable
5
>>> type(myVariable)
<class 'int'>
>>> myVariable='tekst'
```

```
>>> myVariable  
,tekst'
```

Kolejną ciekawostką w Pythonie jest możliwość przypisania kilku zmiennym ich wartości w jednej linii.

```
>>> varA, varB, varC = 4, 14, 'kawa'
```

Wyświetl zawartość varA, varB, varC.

Komentarze

Komentowanie kodu to dobry zwyczaj na początek nauki. Komentarze ułatwiają nam (czy innemu programiście) analizę naszego kodu. Są to uwagi, teksty, których nie zobaczy użytkownik programu.

Wraz z nabraniem praktyki przestaniesz komentować swój kod. Dobrze ponazywane zmienne i metody nie wymagają komentarza – **kod jest samokomentujący się**.

Komentarze w Pythonie mogą być jednowierszowe – rozpoczynać się znakiem # i ciągnąć do końca linii lub blokowe przechowywane w cudzysłowie potrójnym.

```
myName = Rita # Zmienna przechowująca imię  
myAge = 23 # Zmienna przechowująca wiek  
""" A to jest  
wielolinijkowy komentarz blokowy.  
Fajny prawda? """
```

Komentarz blokowy wygląda jak jeden ze stringów, jednak dopóki go nie przypiszemy do zmiennej, to interpreter będzie ignorował tę część kodu – traktował jako komentarz.

Zadanie 1

Utwórz i wyświetl dowolne zmienne przechowujące wartości wg. typów:

Liczby (Numbers) – całkowite/zmiennoprzecinkowe

Teksty (String)

Typ logiczny (Bool)

Listy (List)

Krotki (Tuple)

Słowniki (Dictionary)

Zadanie 2

Utwórz spis oglądanych seriali.

Każdy serial powinien mieć przypisaną ocenę w skali od 1-10.

Zapytaj użytkownika jaki serial chce obejrzeć.

W odpowiedzi podaj jego ocenę.

Zapytaj użytkownika o dodanie kolejnego serialu i oceny.

Dodaj serial do spisu.