Typy obiektów Pythona

- Programy napisane w Pythonie można rozbić na moduły, instrukcje, wyrażenia i obiekty w następujący sposób

✓Program składa się z modułów

✓Moduły zawierają instrukcje

✓Instrukcje zawierają wyrażenia

✓*Wyrażenia tworzą i przetwarzają obiekty*

Moduły uwzględniają najwyższy poziom w tej hierarchii, obiekty i wyrażenia odwołują się do najniższego poziomu.

W dalszej części zostanie przedstawiony przegląd wbudowanych obiektów i wyrażeń, które tworzy się w celu korzystania z tych obiektów.

Typy obiektów Pythona

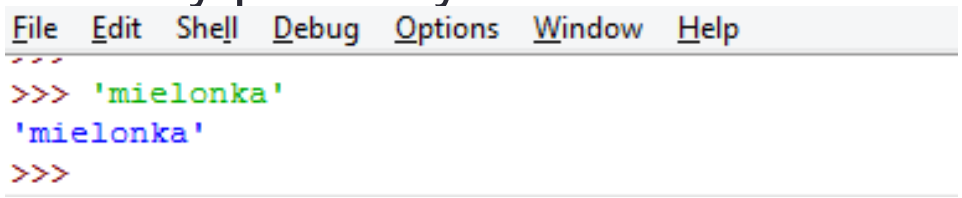
- Obiekty wbudowane sprawiają, że programy łatwo się pisze
- Obiekty wbudowane są standardową częścią języka
- Są często bardziej wydajne od własnych struktur danych
- Obiekty wbudowane są komponentami rozszerzeń
- Wszystko co przetwarzamy w Pythonie jest tak naprawdę rodzajem obiektu
- Bez względu na to, czy zdecydujemy się implementować nowe typy obiektów, obiekty wbudowane stanowią podstawę każdego programu napisanego w Pythonie
- *Literał* – wyrażenie generujące obiekt

Typy obiektów Pythona

Przegląd obiektów wbudowanych Pythona	
Typ obiektu	Przykładowy literał
Liczby	1234, 3.1415, 3+4j
Łańcuch znaków	'mielonka', "Brian"
Listy	[1, [2, 'trzy'], 4]
Słowniki	{'jedzenie': 'mielonka', 'smak':'mniam'}
Krotki	(1, 'mielonka', 4, 'U')
Pliki	myfile = open('jajka','r')
Zbiory	set('abc'),{ 'a', 'b', 'c'}
Inne typy podstawowe	Wartość Boolean, typy, None
Typy jednostek programu	Funkcje, moduły, klasy
Typy powiązane z implementacją	Kod skompilowany

Typy obiektów Pythona

- Kiedy wykonamy poniższy kod



Z technicznego punktu widzenia wykonujemy właśnie wyrażenie z literałem, które generuje i zwraca obiekt łańcucha znaków

- W Pythonie nie istnieje deklarowanie typu, jednak składnia wykonywanego wyrażenia określa typy tworzonych i wykorzystywanych obiektów
- Kiedy tworzymy jakiś obiekt wiążemy go z określonym zbiorem operacji. Np. na łańcuchach znaków można wykonywać tylko operacje dostępne dla łańcuchów znaków
- Oznacza to, że Python jest językiem z *typami dynamicznymi,* jednak jego typy są *silne* (to znaczy że na obiekcie można wykonać tylko te operacje, które są dozwolone dla określonego typu)

Liczby

- Zbiór podstawowych obiektów Pythona obejmuje typowe rodzaje liczb:

✓Całkowite(liczby bez części ułamkowej)

✓Zmiennoprzecinkowe(liczby z częścią dziesiętną)

- A także:

✓Liczby zespolone

✓Liczby wymierne

✓Liczby stałoprzecinkowe

Liczby obsługują podstawowe działania matematyczne jak:

Znak + wykonuje dodawanie

Znak – odejmowanie

Znak * mnożenie

Znak ** potęgowania

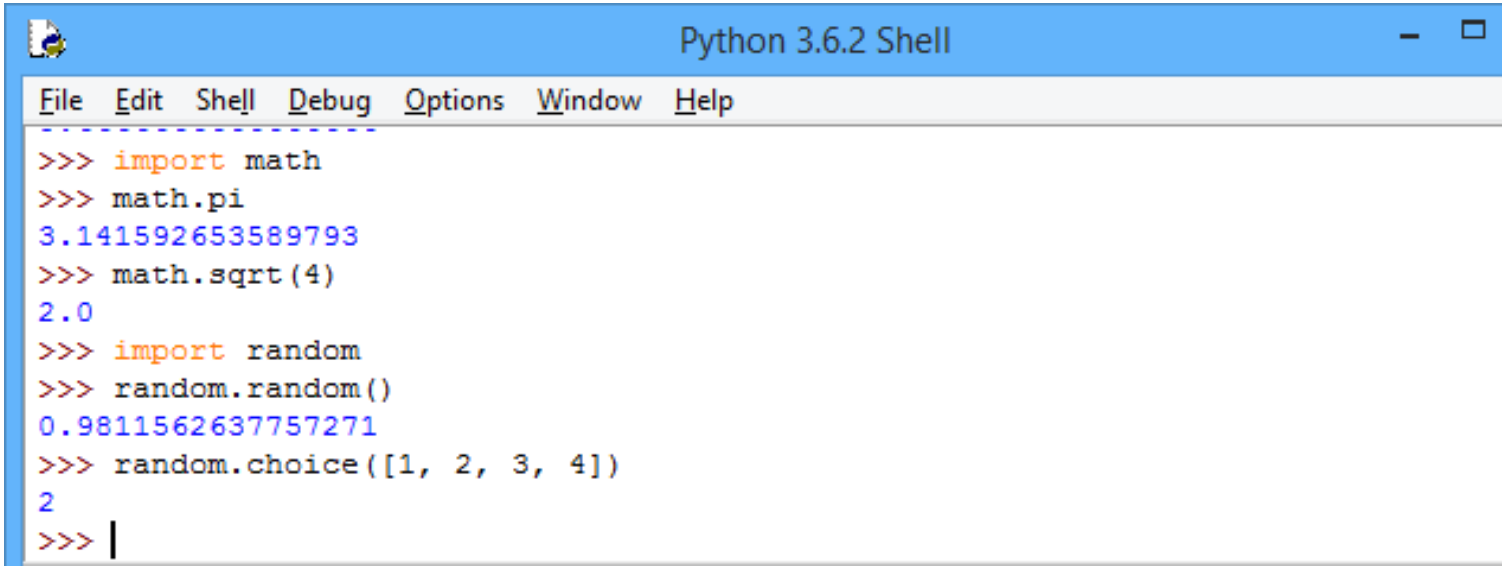
Znak / dzielenie

Znak // dzielenie całkowite

Znak % reszta z dzielenia całkowitego

Liczby

- Poza wyrażeniami w Pythonie znajduje się kilka przydatnych modułów liczbowych. *Moduły* są to pakiety dodatkowych narzędzi, które musimy zaimportować by móc z nich korzystać
- Moduł *math* zawiera dodatkowe narzędzia liczbowe w postaci funkcji, natomiast moduł *random* wykonuje generowanie liczb losowych



Łańcuchy znaków

- Łańcuchy znaków (ang. *string*) wykorzystywane są do przechowywania informacji tekstowych a także dowolnych zbiorów bajtów
- Są przykładem tego, co w Pythonie znane jest pod nazwą *sekwencji* – czyli uporządkowanych zbiór innych obiektów
- Sekwencje zachowują porządek zawieranych elementów od lewej do prawej
- Łańcuchy znaków to inaczej sekwencja składająca się z pojedynczych znaków
- Pozostałe typy sekwencji to: listy i krotki (o których będzie mowa później)

Łańcuchy znaków

- Operacje na sekwencjach

✓Pobieranie długości za pomocą wbudowanej funkcji *len*

✓Pobieranie elementów za pomocą *wyrażeń indeksujących*

```
>>> S = 'Mielonka'
>>> len(S)
8
>>> S[0]
'M'
>>> S[1]
'i'
>>> S[-1]
'a'
>>> S[-2]
'k'
```

- W Pythonie indeksy zakodowane są jako wartość przesunięcia od początku łańcucha, dlatego rozpoczynają się od zera
- Indeksy dodatnie odliczane są od lewej strony, natomiast ujemne od prawej do lewej

Łańcuchy znaków

- Poza prostym indeksowaniem sekwencje obsługują również ogólniejszą formę indeksowania znaną jako *wycinki* (ang. *slice*), polega na ekstrakcji części łańcucha
- Ogólna forma *X[i:j]* oznacza wycinek z łańcucha *X* wszystkiego od przesunięcia *i* do *j* ale bez *j*
- Lewą granicą jest zero natomiast prawą długość sekwencji

```
>>> S = 'Mielonka'
>>> S[1:]
'ielonka'
>>> S[0:7]
'Mielonk'
>>> S[:7]
'Mielonk'
>>> S[:-1]
'Mielonk'
>>> S[:]
'Mielonka'
>>>
```

Łańcuchy znaków

- Tak jak każda sekwencja, łańcuchy znaków obsługują również konkatenację z użyciem znaku + , a także powtórzenie za pomocą operatora *

```
>>> S + 'xyz'
'Mielonkaxyz'
>>> S*3
'MielonkaMielonkaMielonka'
>>>
```

- Znak + jak i * oznacza coś innego w przypadku różnych obiektów – dodawanie mnożenie dla liczb a konkatenację lub powtórzenie dla łańcucha znaków
- Jest to właściwość którą nazywamy *polimorfizmem* co oznacza że każda operacja uzależniona jest od typu obiektów, na jakich się ją wykonuje
- Każda operacja na łańcuchu znaków zwraca nowy łańcuch znaków, nie zmieniając oryginalnego – łańcuchy w Pythonie są *niezmienne*

Listy

- Obiekt *Listy* jest w Pythonie najbardziej ogólnym rodzajem sekwencji
- *Listy* to uporządkowane pod względem pozycji zbiory obiektów dowolnego typu; nie mają one ustalonej wielkości
- Listy w przeciwieństwie do łańcuchów znaków są *zmienne,* co oznacza że można je modyfikować w miejscu
- Ponieważ listy są sekwencjami, obsługują wszystkie operacje przedstawione przy okazji omawiania łańcuchów znaków
- Jedyną różnicą jest to że wynikiem zazwyczaj są listy a nie łańcuchy znaków
- Listy Pythona powiązane są z tablicami w innych językach programowania, jednak mają większe możliwości
- Przede wszystkim nie mają żadnych ograniczeń co do typów danych

Listy

- Przykład listy trzyelementowej o różnych typach
- Możemy ją zindeksować, sporządzić z niej wycinki i wykonać pozostałe operacje zaprezentowane na przykładzie łańcuchów znaków
- Ale możemy zmieniać w oryginalnej liście wartość elementów

```
>>> L = [123, 'mielonka', 1.23]
>>> L[0]
123
>>> L[:-1]
[123, 'mielonka']
>>> L + [4, 5, 6]
[123, 'mielonka', 1.23, 4, 5, 6]
>>> L
[123, 'mielonka', 1.23]
>>> L*2
[123, 'mielonka', 1.23, 123, 'mielonka', 1.23]
>>> L[0]=2
>>> L
[2, 'mielonka', 1.23]
>>>
```

Listy

- Operacje specyficzne dla typu
- Dodawanie obiektu na końcu listy *append*
- Usunięcie obiektu ze środka listy *pop*
- Wstawienie elementu na dowolnym miejscu *insert*
- Usunięcie elementu o podanej wartości *remove*
- Odwrócenie kolejności elementów *reverse*
- Porządkowanie listy *sort*

```
>>> L
[2, 'mielonka', 1.23]
>>> L.append('NI')
>>> L
[2, 'mielonka', 1.23, 'NI']
>>> L.pop(2)
1.23
>>> L
[2, 'mielonka', 'NI']
>>> M=['bb', 'aa', 'cc']
>>> M.sort()
>>> M
['aa', 'bb', 'cc']
```

```
>>> M.reverse()
>>> M
['cc', 'bb', 'aa']

>>> M.remove('bb')
>>> M
['cc', 'aa']
>>> M.insert(1,'bb')
>>> M
['cc', 'bb', 'aa']
```

Słowniki

- *Słowniki* w Pythonie zwane *odwzorowaniami* są zbiorami obiektów, jednak słowniki przechowują je po kluczu a nie ich pozycji względnej jak to ma miejsce w sekwencjach
- Odwzorowania nie zachowują żadnej kolejności od lewej do prawej strony, po prostu łączą klucze z powiązanymi z nimi wartościami
- Słowniki są zmienne, i mogą być modyfikowane w miejscu, podobnie jak listy
- Literał słownika składa się z serii par *klucz: wartość* umieszczonych w nawiasach klamrowych
- Słowniki są przydatne jeżeli chcemy powiązać zbiór wartości z kluczami – np. opisać właściwości czegoś
- Przykład słownika składającego się z trzech elementów (których kluczami są: jedzenie, ilość i kolor)

```
>>> D = {'jedzenie': 'Mielonka', 'ilość': 4, 'kolor' : 'różowy'}
>>> D
{'jedzenie': 'Mielonka', 'ilość': 4, 'kolor': 'różowy'}
```

Słowniki

- W celu zmiany wartości powiązanych słownik możemy indeksować po kluczu

```
>>> D = {'jedzenie': 'Mielonka', 'ilość': 4, 'kolor' : 'różowy'}
>>> D['jedzenie']
'Mielonka'
>>> D['ilość'] = 5
>>> D['ilość']
5
```

- Forma literału z nawiasami klamrowymi nie jest jedynym sposobem tworzenia słowników, możemy to zrobić z pomocą instrukcji przypisania po jednym kluczu

```
>>> D={}
>>> D['imię'] = 'Robert'
>>> D['zawód'] = 'programista'
>>> D['wiek'] = 40
>>> D
{'imię': 'Robert', 'zawód': 'programista', 'wiek': 40}
>>> D['wiek']+=1
>>> D['wiek']
41
```

Krotki

- *Krotki* to w przybliżeniu listy, których nie można modyfikować. Krotki są sekwencjami, podobnie do list, jednak są też niezmienne – tak jak łańcuchy znaków
- Krotki obsługują dowolne typy danych, zagnieżdżanie i zwykłe operacje na sekwencjach

```
>>> T = (1, 2, 3, 'aa')
>>> T
(1, 2, 3, 'aa')
>>> len(T)
4
>>> T + ('bb', 6)
(1, 2, 3, 'aa', 'bb', 6)
>>> T[0]
1
```

- Krotki mają jedynie dwie metody wywoływalne specyficzne dla tego typu: *index* i *count*

```
>>> T.index('aa')
3
>>> T.count(3)
1
```

Krotki

- Krotki w praktyce są dużo rzadziej używane niż listy, jednak ich niezmienność jest ich zaletą tzn. nie mogą rosnąć ani kurczyć się
- Kiedy w programie przekazujemy zbiór obiektów w postaci listy, obiekty te mogą się w dowolnym momencie zmienić. W przypadku krotki zmienić się nie mogą

```
>>> T[0]=2
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    T[0]=2
TypeError: 'tuple' object does not support item assignment
```

- W związku z tym nakładają pewne ograniczenia w zakresie integralności, które mogą sią przydać w większych programach