Formatowanie napisów

W Pythonie istnieją sposoby tworzenia napisów w ogólnej postaci i podstawiania do nich konkretnych obiektów (tzn. napisów, które je reprezentują).

- Interpolacja napisów – f-stringi (dziś),
- „stary sposób" z użyciem operatora % (tylko informacyjnie),
- „nowy sposób" (później)

Przykład użycia f-stringa:

```
name, age = "Manfred", 25
txt = f"{name} is {age} years old, not {age+1}."
```

Fragmenty postaci {...} to „formatki", w których można umieszczać wyrażenia. Wyrażenia zostają wyliczone i ich napisowe reprezentacje zostają podstawione w miejsce formatek. W formatkach można umieszczać dodatkowe informacje (przykłady: `strformat.py` i `strformat.html`).

Atrybuty

Atrybut – „nazwa przywiązana (lub należąca) do obiektu".
(nazwy nazywają obiekty, więc w szczególności atrybuty nazywają obiekty „przywiązane" do danego obiektu)

Składnia dostępu do atrybutu `attribute` obiektu `obj`: `.` (kropka):

```
obj.attribute
```

Przykład: liczby zespolone:

```
z = 2 + 1j      # liczba zespolona 2 + i
print(z.real) # część rzeczywista
print(z.imag) # część urojona
```

Przykład: moduły (obiekty) i ich atrybuty (np. funkcje, stałe):

```
import math

y = math.sin(math.pi / 2)
```

Metody

Metoda – atrybut, funkcja przywiązana do obiektu.

O takiej funkcji można myśleć, że jej ukrytym parametrem jest obiekt, do którego jest przywiązana.

Przykłady:

```
z1 = 2 + 1j            # liczba zespolona
z2 = z1.conjugate()    # sprzężenie z1, bez parametrów
# z2 == 2 - 1j
```

```
lst = [1, 2, 3]
lst.append(4)    # dokłada 1 na koniec listy lst
# lst == [1, 2, 3, 4]
```

Operacje na listach

Atrybuty (w tym metody) obiektu są w znacznej części zdeterminowane przez typ tego obiektu.

Na wykładzie: niektóre metody list.

- `append`, `extend`, `insert`
- `pop`, `remove`, `clear`
- `index`, `count`
- `sort`, `reverse`
- `copy`

Oprócz tego, inne operacje na listach (nie przez metody):

- `del`, `len`
- `min`, `max`, `sum`
- `sorted`, `reversed`
- listy składane
- (extended) slicing

Listy składane

Listy składane (schematy konstrukcji list) – skrótowe wyrażenia konstruujące listy z innych list (lub napisów, etc.).
„Ręczna" konstrukcja listy kwadratów `[0, 1, 4, 9, 16, 25]`:

```
lst = [] # pusta lista
for i in range(6):
    lst.append(i ** 2)
```

Taką listę można stworzyć (i potem nazwać) pojedynczym wyrażeniem:

```
lst = [i ** 2 for i in range(6)]
```

W liście składanej mogą pojawiać się warunki (odrzucające niektóre obiekty) oraz więcej iteracji, np.:

```
lst1 = [1 / x for x in range(-3, 3) if x != 0]
lst2 = [c * i for c in "abc" for i in range(3)]
```

Przykład: `pascal.py`

Słowniki (wstępnie): `dict`

Słownik: struktura danych stowarzyszająca **klucze** z **wartościami**. Przykład użycia:

```
d = {1: 'a', 2: 'b'} # klucz: wartosc, klucz2: wartosc2
print(d[1]) # 'a'
d['abc'] = 500
print(d['abc']) # 500
```

- Kluczami mogą być tylko obiekty hashowalne (czyli $\approx$ niezmienialne), odpowiadające im wartości – dowolne.
- Kilka szczególnych sposobów konstrukcji (następny slajd).
- Dodawanie i usuwanie par klucz/wartość, podmienianie wartości stowarzyszonej z kluczem.
- Iteracja po kluczach, po wartościach, po parach klucz/wartość.
- Indeksowanie odczytuje wartość stowarzyszoną z kluczem.

Niektóre operacje na słownikach

Sposoby konstrukcji:

```
d = {1: 'a', 2: 'b'} # klucz1: wartość1, klucz2: wartość2
d = dict([(1, 'a'), (2, 'b')]) # lista par
d = dict(x=1, y=2, z=1) # nazwy parametrów to klucze
d = {n: n**2 for n in range(5)} # słownik składany
```

Podstawowe operacje:

```
d[k] = v # nowy klucz, lub podmiana wartości
del d[k] # usunięcie klucza+wartości
k in d # test, czy k jest kluczem
d.get(k, default) # (*)
```

(*) - jeśli `k` jest w słowniku, zwraca `d[k]`, w przeciwnym wypadku zwraca `default`.

Niektóre operacje na słownikach

Iteracja (d – słownik):
Po kluczach:

```
for k in d:  # równoważnie: for k in d.keys():
    ...
    # k - klucz
```

Po wartościach:

```
for v in d.values():
    ...
    # v - wartosc
```

Po parach klucz-wartość:

```
for k, v in d.items():
    ...
    # k - klucz
    # v - wartość
```