Разработка на софтуер

Лекция 5 – Python collections

Милен Спасов

Стил

- PEP8 – https://www.python.org/dev/peps/pep-0008/

- Индентация с 4 интервала и без табове
- 79 символа на ред
- snake_case – променливи, параметри, функции и методи
- SCREAMING_SNAKE_CASE – константи
- _private – за частни методи и променливи
- class_, range_ – при използване на запазени думи, но не е добра практика

- интервал след "," при изброяване и ":" при конструиране на dict: [1, 2, 3] {1: 2, 2: 4, 13: 26}
- без интервали след (, [, { и след),], }: func(1, 2, 3), [5, 6, 7, 8]
- по един интервал около оператори: a == b; 3 > 4; "abra" + "-" + "kadabra"
- без интервали, когато задаваме стойност по подразбиране: def my_func(a, b, option=True):
- и при подаване на именовани параметри при извикване my_func(option=False, b=13, a=666)
- без скоби около условията на while/for/if/elif/return: while True: ...

- Конфигурирайте editor-ите си
- Може да използвате style checker-и

Структури от данни

- list (array, масив) = подредена последователност от стойности
- tuple = непроменяема по състав подредена последователност от обекти (~списък, но не съвсем)
- set = стойности без повтаряне и без подредба (множество в математическия смисъл)
- dict = ключове/имена, зад които стоят стойности (без подредба)

- **Какво е колекция?**
  - Всички колекции са итерируеми (iterable)
  - Един итерируем обект може да бъде обхождан последователно (поне веднъж)
  - Някои могат да бъдат обхождани многократно или непоследователно

List (1)

```
nice_things = ['coffee', 'cheese', 'crackers', 'tea']
for thing in nice_things:
    print('I tend to like {}'.format(thing))

print(nice_things[1]) # cheese
print(nice_things[-1]) # tea

-----------------------------------------------------------------------------------------------------------
cute_animals = ['cat', 'raccoon', 'panda', 'red panda', 'marmot']
cute_animals[1:3]  # ['raccoon', 'panda']
cute_animals[-1]  # 'marmot'
cute_animals[1:-1]  # ['raccoon', 'panda', 'red panda']
cute_animals[::-1]  # ['marmot', 'red panda', 'panda', 'raccoon', 'cat']
cute_animals[-1:0:-1]  # ['marmot', 'red panda', 'panda', 'raccoon']
cute_animals[-1:0:-2]  # ['marmot', 'panda’]

-----------------------------------------------------------------------------------------------------------
coffee, cheese, crackers, tea = 'coffee', 'cheese', 'crackers', 'tea' # unpacking
things_i_like = [coffee, cheese, crackers]
things_you_like = [crackers, coffee, tea]

things_i_like[0] == things_you_like[1] # True
things_i_like[0] is things_you_like[1] # True
```

List (2)

Това позволява някои интересни неща

```
cheeses = ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto']
cheeses.append(cheeses)

cheeses[-1] is cheeses # True
print(cheeses) # ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto', [...]]
```

```
cheeses = ['brie', 'bergkäse', 'kashkaval', 'leipäjuusto']
teas = ['chai', 'earl grey', 'jasmine', 'oolong']

breakfast = [cheeses, teas]
print(breakfast[0][1]) # bergkäse

breakfast[1][2] = [‘pancakes’, ‘eggs’, ‘chocolate']
print(teas) # ?

# teas = ['chai', 'earl grey', [‘pancakes’, ‘eggs’, ‘chocolate'], 'oolong']
```

List (3)

Методи за списъци:

- .index(element) - Индекса на първото срещане на element в списъка или гърми с ValueError
- .count(element) - Броят срещания на element в списъка
- .append(element) - Добавя element в края на списъка
- .extend(elements) - Добавя елементите на elements в списъка
- .sort() - Сещате се

За всички други методи и при въпроси какво се поддържа ползвайте help().

Range

range връща итерируемо за интервал от числа

```
numbers = range(3)

for number in numbers:
    print('We can count to ' + str(number))
```

range интервалът може да не започва от нула

```
numbers = range(10, 13)
```

range може и в обратен ред

```
numbers = range(13, 0, -1)
```

Tuple (1)

Като списък, но с постоянен състав

```
people = ('Niki', ‘Vladi', ‘Georgi')
people[2] # Georgi
people[1] # Vladi
people[0] # Niki

people[1] = ‘Kaloyan'

Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Последователността от елементи не може да се променя, но самите елементи може да изменят вътрешната си структура

```
change_me = ([1, 2, 3], [4, 5, 6], [7, 8, 9])
change_me[1][1] = 0
change_me[2][0] = 'c'

print(change_me) # ([1, 2, 3], [4, 0, 6], ['c', 8, 9])
```

Tuple (2)

Алтернативен синтаксис:

```
people = 'Niki', ‘Vladi', ‘Georgi‘
```

Ако имате tuple, съдържащ само имена от лявата страна на присвояване, може да постигнете интересни ефекти:

```
(a, b) = 1, 2
print(a)  # 1
```

Скобите изобщо не са задължителни:

```
a, b = 1, 2
print(a)  # 1
```

Друг интересен запис:

```
numbers = (1, 2, 3)
a, b, c = numbers
```

Сравняване на lists и tuples

Сравняват се лексикографски:

```
>>> (1, 2) < (1, 3)
True

>>> (1, 2) < (1, 2)
False

>>> (1, 2) < (1, 2, 3)
True

>>> [1, 2] < [1, 3]
True

>>> (1, 2) < [1, 3] # tuple vs. list
# поражда грешка:
#    TypeError: unorderable types: tuple() < list()
```

Популярни структури от данни

Опашка (queue, FIFO buffer) - можете да ползвате списък.

```
adjectives = []

def add_adjective(items):
    adjectives.append(items)

def get_adjective():
    return adjectives.pop(0)

add_adjective('Magic')
add_adjective('Woody Allen')
add_adjective('Zombie')
add_adjective('Superhero')

print(' '.join(adjectives) + ' Jesus!') # Magic Woody Allen Zombie Superhero Jesus!
```

Set (1)

Множества(за всякакви практически цели неразличими от математическата абстракция със същото име)

```
favourite_numbers = set()
favourite_numbers.add(13)
favourite_numbers.add(73)
favourite_numbers.add(32)
favourite_numbers.add(73)
favourite_numbers.add(1024)
favourite_numbers.add(73)

print(favourite_numbers) # {32, 73, 666, 13, 1024}
```

Множествата са итеруеми и НЕподредени:

```
for num in favourite_numbers:
    print('I really like the number ' + str(num))
```

Можем да проверяваме за принадлежност:

```
73 in favourite_numbers # True
```

Set (2)

```
>>> {1, 2, 3} | {2, 3, 4}
{1, 2, 3, 4}
>>> {1, 2, 3} & {2, 3, 4}
{2, 3}
>>> {1, 2, 3} - {2, 3, 4}
{1}
>>> {1, 2, 3} ^ {2, 3, 4}
{1, 4}
>>> {1, 2, 3} < {2, 3, 4}
False
>>> {2, 3} < {2, 3, 4} # < - подмножество
True
>>> {2, 3} == {2.0, 3}
True
>>> {1, 2}.isdisjoint({3, 4})
True
```

Статия за сравняване на set-ове:
https://betterprogramming.pub/a-visual-guide-to-set-comparisons-in-python-6ab7edb9ec41

Dictionary

Индексите не винаги са достатъчно информативни

```
artist_names = {
   'Eddie': 'Vedder',
   'Maynard': 'Keenan',
   'Matthew': 'Bellamy',
   'James': 'LaBrie',
}

print('Eddie\'s last names is ' + artist_names['Eddie’])
```

{} е празен речник, по простата причина, речниците са доста по-често използвана структура от множествата

Можем да добавяме нови стойности във вече създаден речник

```
names['Devin'] = 'Townsend'

print(names) # {'Devin': 'Townsend', 'Matthew': 'Bellamy',
         #  'Eddie': 'Vedder', 'James': 'LaBrie', 'Maynard': 'Keenan’}
```

Речникът също е неподреден.


Thank
You
Asante
Shukria
Dhanyavadagalu
Vinaka
köszönöm
Kiitos
Maake
Manana
Dankon
Matondo
감사합니다
Kam Sah Hammida
Dankscheen
спасибо
Dank Je
Mauruuru
Biyan
Terima Kasih
Taiku
Blagodaram
Ngiyabonga
Dziekuje
Chokrane
Diolch i Chi
Tack
Grazie
Juspaxar
Arigato
Mochchakkeram
Gracias
Tingki
நன்றி
धन्यवाद
cảm ơn bạn
Gràzas
Dàkujem
Bedankt
Ua Tsaug Rau Koj
Paldies
Gratias Tibi
Khap
Obrigado
Děkuji
Nirringrazzjak
Di Ou Mèsi
Kia Ora
Kop Khun
ありがとう
Suksama
Hvala
Djiere Dieuf
Welalin
Matur Nuwun
Rahmat
谢
谢
хвала
Danke
Merci
Go Raibh Maith Agat
Misaotra
ขอขอบคุณคุณ
Salamat
Najis Tuke
Eskerrik Asko