03. Въведение 2.0

02 март 2026

Какво видяхте в предишната лекция?

- Приключихме с колекциите `list` и `tuple`
- Започнахме с колекциите `dict` и `set`
- Приключихме с колекциите `dict` и `set`

Въпроси?

Подай Snickers-а там

```
>>> things = {'baba', 'dyado', 'vuicho', 'lelya', 'strinka'}
>>> things.pop(), things.pop(), things

('vuicho', 'lelya', {'dyado', 'baba', 'strinka'})
```

Нали помните, множествата не са подредени.

А речниците?

По темата с подредените речници

[Python-Dev] Guarantee ordered dict literals in v3.7?

Guido van Rossum guido at python.org
Fri Dec 15 10:53:40 EST 2017

- Previous message (by thread): [Python-Dev] Guarantee ordered dict literals in v3.7?
- Next message (by thread): [Python-Dev] Guarantee ordered dict literals in v3.7?
- **Messages sorted by:** [date] [thread] [subject] [author]

Make it so. "Dict keeps insertion order" is the ruling. Thanks!

On Fri, Dec 15, 2017 at 2:30 AM, INADA Naoki <songofacandy at gmail.com> wrote:

> > That's interesting information - I wasn't aware of the different
> > performance goals.
>
> FYI, performance characteristic of my POC implementation of
> OrderedDict based on dict order are:
>
> * 50% less memory usage
> * 15% faster creation
> * 100% (2x) faster iteration
> * 20% slower move_to_end
> * 40% slower comparison
>
> (copied from https://bugs.python.org/issue31265#msg301942)
>
> Comparison is very unoptimized at the moment and I believe it can be
> more faster.
> On the other hand, I'm not sure about I can optimize move_to_end() more.
>
> If OrderDict is recommended to be used for just keeping insertion order,
> I feel 1/2 memory usage and 2x faster iteration are more important than
> 20% slower move_to_end().
>
> But if either "dict keeps insertion order" or "dict keeps insertion order
> until
> deletion" is language spec, there is no reason to use energy and time for
> discussion of OrderedDict implementation.
>
> Regards,
>
> INADA Naoki <songofacandy at gmail.com>
> _____
> Python-Dev mailing list
> Python-Dev at python.org
> https://mail.python.org/mailman/listinfo/python-dev
> Unsubscribe: https://mail.python.org/mailman/options/python-dev/
> guido%40python.org
>

По темата с подредените речници

Guido van Rossum guido at python.org
Fri Dec 15 10:53:40 EST 2017

Make it so. "Dict keeps insertion order" is the ruling. Thanks!

По темата с подредените речници

- И въпреки, че все още можете да срещнете следното из документацията на езика:

> *Changed in version 3.7:* Dictionary order is guaranteed to be insertion order. This behavior was an implementation detail of CPython from 3.6.

- На този етап можете да разчитате, че редът е този на инициализиране / добавяне на елементи в речника
- *Забележка: Ако промените стойността зад даден ключ, двойката си остава на същото място*
- Единственото, което трябва да се запитате е - “Очаквам ли кодът ми да бъде пускан на Python < 3.7 и искам ли да е compatible?”
- В общия случай отговорът ще бъде “не”

Подай Snickers-а там (2)

```
>>> lectures_location = {'Monday': '200', 'Wednesday': '229'}
>>> lectures_location.get('Tuesday', 'Thursday')
```

```
'Thursday'
```

Опа, умишлено се опитахме да ви подведем.
Вторият аргумент е default стойност, която се връща, ако не се намери елемент с ключ *(в случая)* `'Tuesday'`

Подай Snickers-а там (3)

Какъв ще е резултатът от следния код?

```
>>> my_favourite_things = ('spam') * 100

'spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspams
pamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamsp
amspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspa
mspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspamspam
spamspamspamspamspamspamspamspam'
```

Малко код от предизвикателството

Къде е проблемът?

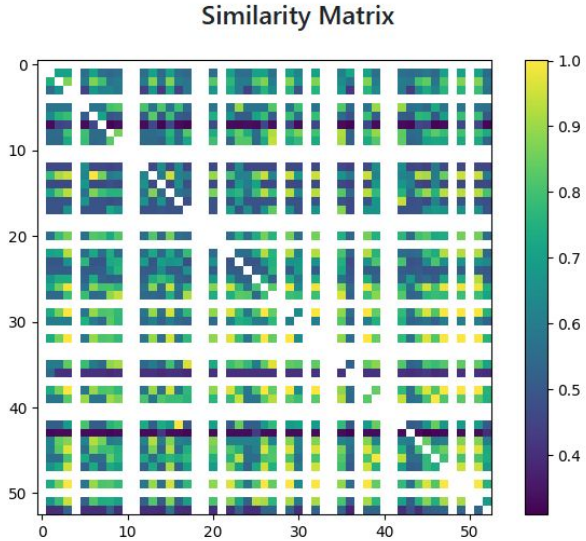
```
>>> my_stuff = ['ball', 'ukulele', 'lube']
>>> your_stuff = ['headphones', 'backpack']
>>> our_stuff = your_stuff
>>> our_stuff.extend(my_stuff)

>>> your_stuff
['headphones', 'backpack', 'ball', 'ukulele', 'lube']

>>> your_stuff is our_stuff
True
```

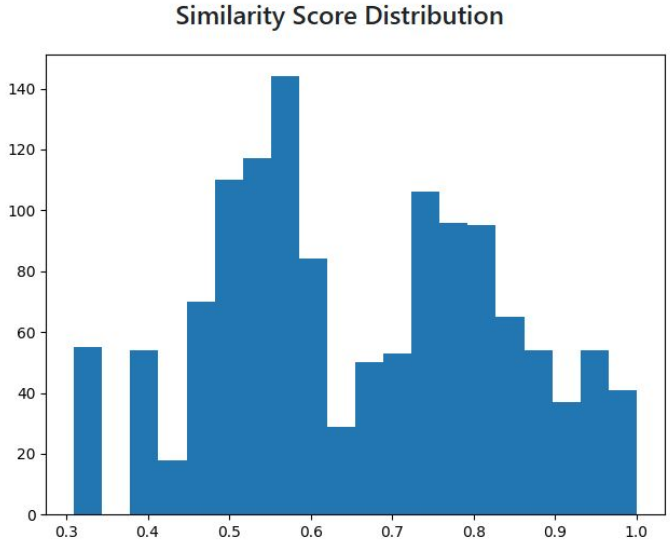
Малко за предизвикателството

- Не сме оставили кой знае колко обратна връзка, защото предизвикателството беше няколко сравнително прости реда - не много поле за коментари
- А не върви да ви спамим нотификациите с “Изглежда супер”
- Не се притеснявайте, на първото домашно ще си получите коментарите
- Грешките на повечето хора са или от недочитане на условието или от грешки при копиране на стринговете
- Това е единственото домашно, в което няма удобен начин да копирате, така че в бъдеще не би следвало да имате такива проблеми
- Все пак, четете условието **внимателно** и внимавайте за подобни дребни грешки

Още малко за предизвикателството

- Поради същата причина не гледаме и следното:





Но ви индикираме, че съществува и ако до момента не сме ви казали да не преписвайте, сега е добър момент да го направим - Не преписвайте!

И последно за предизвикателството

- Няма нужда от `print`-ове
- В същото време не е и функционален проблем
- Но пък е концептуален такъв - желателно е да ги няма в production код *(освен ако изрично не им е там мястото)*
- Съвсем скоро ще си говорим как работят unit тестовете
- За любопитните:

Но така и така си говорим за print

- Можете (желателно е) да използвате така наречените f-strings:

```
>>> main_character = "Captain Jack Sparrow"
>>> print(f"This is the tale of {main_character}")
"This is the tale of Captain Jack Sparrow"
```

- Имат и интересни опции за форматиране:

```
heights = {"Mount Everest": 8849, "Musala": 2925, "Shaq": 2.16}
for name, height in heights.items():
    print(f"{name:15} ==> {height:10}")
"Mount Everest   ==>       8849"
"Musala          ==>       2925"
"Shaq            ==>       2.16"
```

Повече тук и тук.

Какво със сигурност ще видите днес?

- Контролни структури - `if`, `while`, `for`, `switch` (`break`, `continue`)
- Как се дефинират блокове код в Python
- Дефиниране на функции и аргументи на функции
- Супер готина новина

Какво може би ще видите днес?

- Голи снимки
- Функции, които генерират колекции
- Функции, които използват колекции
- Функции, които едновременно използват и генерират колекции
- Бикове и крави

Контролни структури

- `if .. elif .. else`
- `while`
- `for`

if

```
if a == 5:
    print("a is five")
elif a == 3 and not b == 2:
    print("a is three, but b is not two")
else:
    print("a is something else or b is two")
```

- Точно каквото очаквахте.
- Не слагайте скоби около условията.
- `and`, `or` и `not`
- **НЕ** &&, ||, !

if (с булеви променливи)

```
a = True

if a:
    print("a is True")
if not a:
    print("a is not True")
```

Истина и лъжа

В контекста на булевите операции като лъжа се интерпретират следните стойности:

- `False`
- `None`
- числото `0` независимо от типа числа (например `0, 0.0, 0j`)
- празният низ
- празни контейнери (`tuple, list, dict, set`)
- наши типове могат да дефинират как да бъдат оценявани като булеви променливи

Всички останали стойности се интерпретират като истина.

В този ред на мисли...

- Вероятно помните, че можем да “cast”-ваме стойности към различни типове.
- В кавички, защото не е баш кастване и определено не е като в статично типизираните езици.

```
int('5') # 5

b = 10
str(b) # '10'

nums = (1, 2, 3)
nums = list(nums)
nums # [1, 2, 3]

str(['baba', 2]) # "['baba', 2]"
```

Разбира се в някакви граници

```
int('диевиетстотин и пидисе')

Traceback (most recent call last):
  File "<pyshell#950>", line 1, in <module>
    int('диевиетстотин и пидисе')
ValueError: invalid literal for int() with base 10: 'диевиетстотин и
пидисе'
```

Тогава - въпрос

На базата на последните 3 слайда, до какво ще се оцени долното:

```
>>> question = "Питона искаш ли да ти покажа?"
>>> bool(question)

True
```

В почивката.

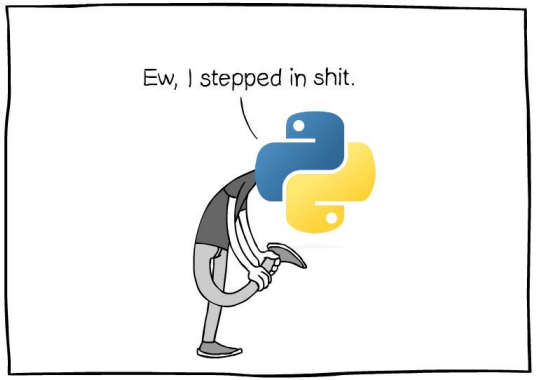
Също така, цитат от преди 3 слайда:
Всички останали стойности се интерпретират като истина.

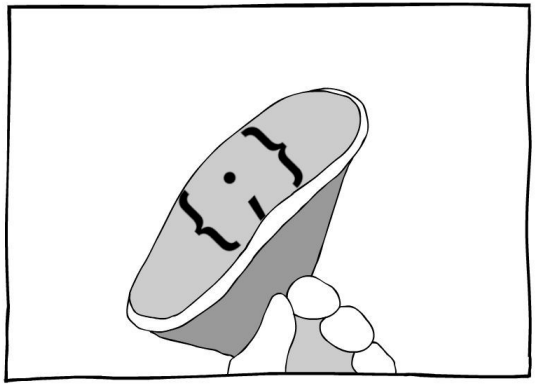
if (тестове за принадлежност)

```
my_list = [1, 2, 3, 4]

if 1 in my_list:
    print('1 is in my list')

if 5 not in my_list:
    print('5 is not in my list')
```

Индентация

- Къде са къдравите скоби?!
- Всеки блок код (тяло на `if`, тяло на функция, и т.н.) се определя с индентацията му спрямо обгръщащия го блок.
- Всеки блок код започва само след двоеточие в края на предишния ред.
- Блокът свършва, когато се върнете към предишната индентация.
- 4 празни места = нов блок.
- **Не 2, не 3, не 8, не табулация**
- Дресирайте редактора си да слага 4 празни места когато натиснете <Tab>





Индентация


C#
Java
C++
New Programmer
Missing ;
Python
IdentationError:


TabError: inconsistent use of tabs and spaces in indentation
Python Programmer

while

```
a = 10
while a > 5:
    a -= 1
    print(f"a is {a}")
```

- Елементарно
- Въпроси?

Какво е колекция?

- В Python всички колекции са итерируеми (iterable)
- Един итерируем обект може да бъде обхождан последователно (поне веднъж)
- Някои могат да бъдат обхождани многократно

for

```
primes = [3, 5, 7, 11]
for e in primes:
    print(e ** 2) # 9 25 49 121

people = {'bob': 25, 'john': 22, 'mitt': 56}
for name, age in people.items():
    print("{} is {} years old".format(name, age))
    # bob is 25 years old
    # john is 22 years old
    # ...
```

- `for` е като `foreach` в другите езици
- Няма инициализация, стъпка и проверка, не е fancy `while`
- Обхожда структури от данни

Един пример от предна лекция

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

# I tend to like coffee
# I tend to like cheese
# ...
```

for като в C

```
for i in range(0, 20):
    print(i)
    # 0 1 2 3 4 5 6 .. 19

for i in range(0, 20, 3):
    print(i)
    # 0 3 6 9 12 15 18
```

Може и наобратно

```
for i in range(20, 0, -1):
    print(i)
    # 20 19 18 17 16 15 .. 1

for i in range(20, 0, -3):
    print(i)
    # 20 17 14 11 8 5 2
```

break и continue

- Работят, както очаквате във `for` и `while`
- Афектират само най-вътрешния цикъл

switch/case

- Няма...
- Добре де, нямаше...
- Вече има (Python >= 3.10)
- Все още не сме решили дали е добра идея
- Засега можете да си поиграете с него
- И в Python е `match`/`case`

match/case

```
http_status = 400

match http_status:
    case 400:
        print("Bad request")
    case 401 | 403: # 401 OR 403
        print("Authentication error")
    case 404:
        print("Not found")

# Bad request
```

match/case

```
http_status = 9001

match http_status:
    case 400:
        print("Bad request")
    ...
    case _: # Default
        print("Other error")

# Other error
```

Има и още хиляда синтактични конструкции свързани с `match`/`case`, за момента толкоз.

Функции

```
def say_hello(name, side):
    return "Hello… It's me…"
```

- Функцията приема аргументи
- Функцията може да върне нещо с `return`, а ако няма `return`, връща `None`
- Не се описват типовете на аргументите, нито типа на резултата

Аргументи на функции - позиционни

- Параметрите в предният пример са позиционни
- Ако функцията има 3 позиционни параметъра - трябва да я извикате с 3 аргумента
- Иначе - грешка
- Параметри vs аргументи?!

```
def multiply(a, b):
    return a * b

multiply(5, 10) # 50

multiply(5)
# TypeError: multiply() missing 1 required positional argument: 'b'
```

Аргументи на функции - именувани

```
def multiply(a, b=2):
    return a * b

multiply(5) # 10
multiply(5, 10) # 50

def is_pythagorean(a=2, b=3, c=4):
    return a * a + b * b == c * c

is_pythagorean(b=5, a=3) # c = 4
is_pythagorean(1, c=3) # a = 1, b = 3
```

- Именувани, по подразбиране, опционални
- Могат да бъдат изрично упоменати, ако не - взимат стойността по подразбиране
- Стига позиционните да са подадени в правилен ред, именуваните могат да бъдат в какъвто и да е ред

Променлив брой аргументи

```
def varfunc(some_arg, *args, **kwargs):
    #...

varfunc('hello', 1, 2, 3, name='Bob', age=12)
    # some_arg == 'hello'
    # args = (1, 2, 3)
    # kwargs = {'name': 'Bob', 'age': 12}
```

- Функциите могат да приемат произволен брой аргументи
- Позиционните аргументи (тези без име) отиват в `args`, което е `tuple` от аргументи
- Именуваните аргументи отиват в `kwargs`, което е `dict` от имена на аргументи и съответните им стойности
- Имената `args` и `kwargs` не са специални, **но са наложена конвенция**
- **Редът е важен!**


To Be Continued

Въпроси?