# Речници в Python

д-р Филип Андонов

10 юни 2022 г.

# Съдържание

# Речници

Един недостатък на списъците е, че за да намерим дадена стойност ние трябва или да претърсваме списъка докато я намерим или да знаем позицията ѝ, която обикновено няма логическа връзка със стойността. Затова в Python има друг съставен тип, наречен речник.
Можем да мислим за асоциативните масиви (друго наименование на речниците) като за списъци, в които ключовете могат да бъдат (почти) всякаква стойност, а не само целочислена.

# Речници

Речниците се състоят от двойки (ключ-стойност). На всеки ключ отговаря някаква стойност и по зададен ключ добиваме съответната му стойност или установяваме, че такава няма. Ключовете не могат да се повтарят, а стойностите могат. Както и при списъците, обръщението към елемент с даден ключ връща съответстващата стойност. Трябва да имаме предвид, че в речниците няма подредба на ключовете.

Listing 1: Телефонна книга със списък

```
1 Contacts = ['S klasata','Audi quatro', '
    princesa92', 'gazara666']
2 Phones = ['0888696562','0898007392','0878666555',
    '0888929493']
3 Phones[Contacts.index('princesa92')]
4 print(Phones[Contacts.index('princesa92')])
```

Единственото нещо, което свързва двата списъка, са общите индекси. На индекс 1 в първия списък е името, чийто телефон е на индекс 1 във втория. Python обаче не знае за тази връзка и няма да попречи тя да бъде нарушена. Достатъчно е да изтрием някой елемент от единия списък и в другия се получава осиротял запис.

Listing 2: Телефонна книга с речник

```
1 phonebook = {'S klasata':'0888696562', 'Audi
    quatro' : '0898007392', 'princesa92':'
    0878666555', 'gazara666':'0888929493'}
2 print(phonebook['princesa92'])
```

Тук кодът е по-изчистен. Данните, които са обвързани логически, са обвързани и физически.

# Речници

Речниците в Python са обекти, поради което имат методи. Създаването на речници обаче се извършва от функция.
Функцията dict се използва, за да създаде речници от други съответствия (например други речници) или от редици от двойки (ключ, стойност).

Listing 3: Функция dict

```
>>> constants = [("pi",3.14159),("g",9.18)]
>>> d = dict(constants)
>>> d
{'pi': 3.14159, 'g': 9.18}
>>> d["pi"]
3.14159
>>> d = dict(pi=3.14159,G=9.18)
>>> d["pi"]
3.14159
```

# Речници

Трябва да имаме предвид, че двойките ключ-стойност в речник не са подредени по никакъв начин. Не е задължително ключовете на речниците да са целочислени стойности (въпреки че е допустимо). Ключовете могат да са от всеки неизменим тип – числа с плаваща запетая, символни низове, кортежи. Можем да присвояваме стойност на ключ, който не съществува в речника – това ще създаде нова двойка ключ-стойност. Не можем да присвояваме стойност извън размера на списъка.

Listing 4: Създаване на нов елемент в речник

```
1 >>> x = {}
2 >>> x[42] = 'Foobar'
3 >>> x
4 {42: 'Foobar'}
```

# Методи

Методът clear премахва всички елементи от речник. Това е операция на място, така че не връща нищо.

Listing 5: Метод clear

```
>>> d = dict(pi = 3.14159,G = 9.18)
>>> d
{'pi': 3.14159, 'G': 9.18}
>>> d.clear()
>>> d
{}
```

Когато искаме да изтрием речник, трябва да използваме именно този метод. Може да ни се стори добра идея просто да присвоим празен речник на променлива – речник, но това просто ще доведе до това тя да сочи към друг речник. Нека разгледаме двата варианта по-долу.

Listing 6: Присвояване на празен речник

```
1 >>> x = {}
2 >>> y = x
3 >>> x['key'] = 'value'
4 >>> y
5 {'key': 'value'}
6 >>> x = {}
7 >>> y
8 {'key': 'value'}
```

Вместо да присвояваме празен речник, в следващия вариант ще използваме метода clear. В първия вариант имаме две променливи, които сочат към един и същ речник. Пренасочвайки едната да сочи към празен списък по никакъв начин не променя съдържанието на речника и той е достъпен през другата променлива, тъй като разбира се и двете са просто референции.

Listing 7: Използване на clear

```
>>> x = {}
>>> y = x
>>> x['key'] = 'value'
>>> y
{'key': 'value'}
>>> x.clear()
>>> y
{}
```

# Методи

Методът copy връща ново плитко копие на речника със същите двойки ключ-стойност. Разликата между плитко и дълбоко копие е от значение само за съставни обекти (като списъци или инстанции на класове). Плиткото копие съставя нов обект и след това до колкото е възможно вмъква референции към обектите от оригинала. Дълбокото копие от друга страна съставя нов обект и след това рекурсивно копира всички обекти от оригинала.

Listing 8: Метод copy

```
1 >>> d = {'username':'filip','aliases':['phil','
      fil','fuf']}
2 >>> d2 = d.copy()
3 >>> d2['username'] = 'Mike'
4 >>> d2['aliases'].remove('fuf')
5 >>> d
6 {'username': 'filip', 'aliases': ['phil', 'fil']}
7 >>> d2
8 {'username': 'Mike', 'aliases': ['phil', 'fil']}
```

Функцията deepcopy от модула copy връща ново дълбоко копие на речника със същите двойки ключ-стойност.

Listing 9: Функция deepcopy

```
>>> from copy import deepcopy
>>> d = {'username':'filip','aliases':['phil','fil','fuf']}
>>> d2 = d.copy()
>>> d3 = deepcopy(d)
>>> d['aliases'].append('Felipe')
>>> d2
{'username': 'filip', 'aliases': ['phil', 'fil', 'fuf', 'Felipe']}
>>> d3
{'username': 'filip', 'aliases': ['phil', 'fil', 'fuf']}
```

# Методи

Методът fromkeys създава нов речник от зададени ключове, всеки със съответна стойност None. Ако не искаме стойността по подразбиране да е None, можем да предоставим своя.

Listing 10: Метод fromkeys

```
>>> dict.fromkeys(['name', 'age'])
{'age': None, 'name': None}
>>> dict.fromkeys(['name', 'age'], '(unknown)')
{'age': '(unknown)', 'name': '(unknown)'}
```

Методът get е прощаващ грешките начин за достъп до елементите на речник. Обикновено когато се обърнем към ключ, който не съществува в речника, възниква грешка. Когато обаче поискаме стойност с get, ако ключът не съществува, методът просто ни връща None.

Listing 11: Метод get

```
>>> d = {}
>>> print(d["name"])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'name'
>>> d.get('name')
>>>
```

За да проверим дали даден ключ k е наличен в даден речник d използваме оператора in.

Listing 12: Наличие на ключ

```
>>> d = {}
>>> 'name' in d
False
>>> d = {'name': 'Ivan'}
>>> 'name' in d
True
```

# Методи

Методът items връща изглед към всички елементи от речник като списък, като всеки елемент на списъка е във вида (ключ, стойност). Изгледите са подобни на итераторите, но в допълнение допускат промени по речника по време на итериране. Елементите не са подредени по никакъв специален начин.

Listing 13: Метод items

```
1 >>> d = {'title': 'NBU page', 'url': 'http://www.
      nbu.bg', 'spam': 0}
2 >>> d.items()
3 [('url', 'http://www.nbu.bg'), ('spam', 0), ('
      title', 'NBU page')]
```

В Python 3 методът keys връща изглед към ключовете в речника. Това означава, че върху този обект-изглед може да се итерира, и промените в речника по време на итериране ще се отразят коректно. Елементите в резултата не са подредени по никакъв специален начин.

Listing 14: Метод keys

```
1 >>> d = {'title': 'Official NBU page', 'url': '
      http://www.nbu.bg', 'spam': 0}
2 >>> d.keys()
3 ['url', 'spam', 'title']
```

# Методи

Методът pop връща стойност, съответстваща с даден ключ и след това изтрива двойката ключ-стойност от речника:

Listing 15: Метод pop

```
>>> d = {'title': 'Official NBU page', 'url': '
    http://www.nbu.bg', 'spam': 0}
>>> d.pop("spam")
0
>>> d
{'url': 'http://www.nbu.bg', 'title': 'Official
    NBU page'}
```

Методът popitem е подобен на list.pop. За разлика от list.pop обаче, popitem връща навън случайна двойка ключ-стойност и я изтрива, защото речниците нямат подредба и в резултат нямат последен елемент. Това е полезно когато искаме да премахваме и обработваме елементите по ефективен начин.

Listing 16: Метод popitem

```
1 >>> d = {'title': 'Official NBU page', 'url': '
      http://www.nbu.bg', 'spam': 0}
2 >>> d.popitem()
3 ('url', 'http://www.nbu.bg')
4 >>> d
5 {'spam': 0, 'title': 'Official NBU page'}
```

Методът setdefault е подобен на get, но в допълнение на get, setdefault установява стойността, съответстваща със зададения ключ, ако все още не е в речника. На ред 2 от програма се обръщаме към съществуващ ключ и setdefault ни връща стойността, съответстваща с него, а на редове 4 и 7 използваме setdefault, за да въведем нов елемент с дадена от нас стойност по подразбиране.

Listing 17: Метод setdefault

```
>>> mails = {'Ivan':'vankata@abv.bg','Gergana':'
    hubavatageri@yahoo.com'}
>>> mails.setdefault('Ivan')
>>> 'vankata@abv.bg'
>>> mails.setdefault('Joro')
>>> mails
{'Ivan': 'vankata@abv.bg', 'Joro': None, 'Gergana
    ': 'hubavatageri@yahoo.com'}
>>> mails.setdefault('Paco','N/A')
'N/A'
>>> mails
{'Ivan': 'vankata@abv.bg', 'Joro': None, 'Paco':
    'N/A', 'Gergana': 'hubavatageri@yahoo.com'}
```

Методът update допълва един речник с елементите на друг. Съществуващите ключове се презаписват.

Listing 18: Метод update

```
>>> x = {1:'a',2:'b',3:'c'}
>>> y = {3:'q',4:'d',5:'e'}
>>> x.update(y)
>>> x
{1: 'a', 2: 'b', 3: 'q', 4: 'd', 5: 'e'}
```

В Python 3 методът values връща изглед към стойностите в речника, които са итерируеми и динамични (промените в речника по време на итериране се отразяват коректно). За разлика от keys, резултатът може да съдържа повторения.

Listing 19: Метод values

```
>>> D = {}
>>> D[0] = 'Zero'
>>> D[1] = 'One'
>>> D[2] = 'Two'
>>> D[3] = 'Three'
>>> D.values()
['Zero', 'One', 'Two', 'Three']
```

# Итериране

Върху съдържанието на речници може да се итерира с цикъл for. Управляващата променлива получава стойността на текущия ключ, а с негова помощ можем винаги да достъпим и кореспондиращата му стойност.

Listing 20: Итериране

```
1 d = {'a':1,'b':2,'c':3,'d':4}
2 for key in d:
3     print(key,d[key])
```

# Речници VS списъци

Нека направим сравнение между речници и списъци.
Речниците не са подредени по никакъв начин.
Те нямат „първа“ и „последна“ стойност, както е при списъците.
Речниците не могат да се конкатенират с оператор +. Ако искаме да добавим нов елемент, просто трябва да използваме индекс с нов ключ.
Списъците имат индекси, които са целочислени стойности в интервала от 0 до тяхната дължина без едно. Речниците могат да имат всякакъв ключ. Ако имаме речник с име spam, можете да съхраним нещо в spam[3] без да съществуват spam[0], spam[1] или spam[2].
Списъците се индексират по ключ, който задължително е целочислена стойност.

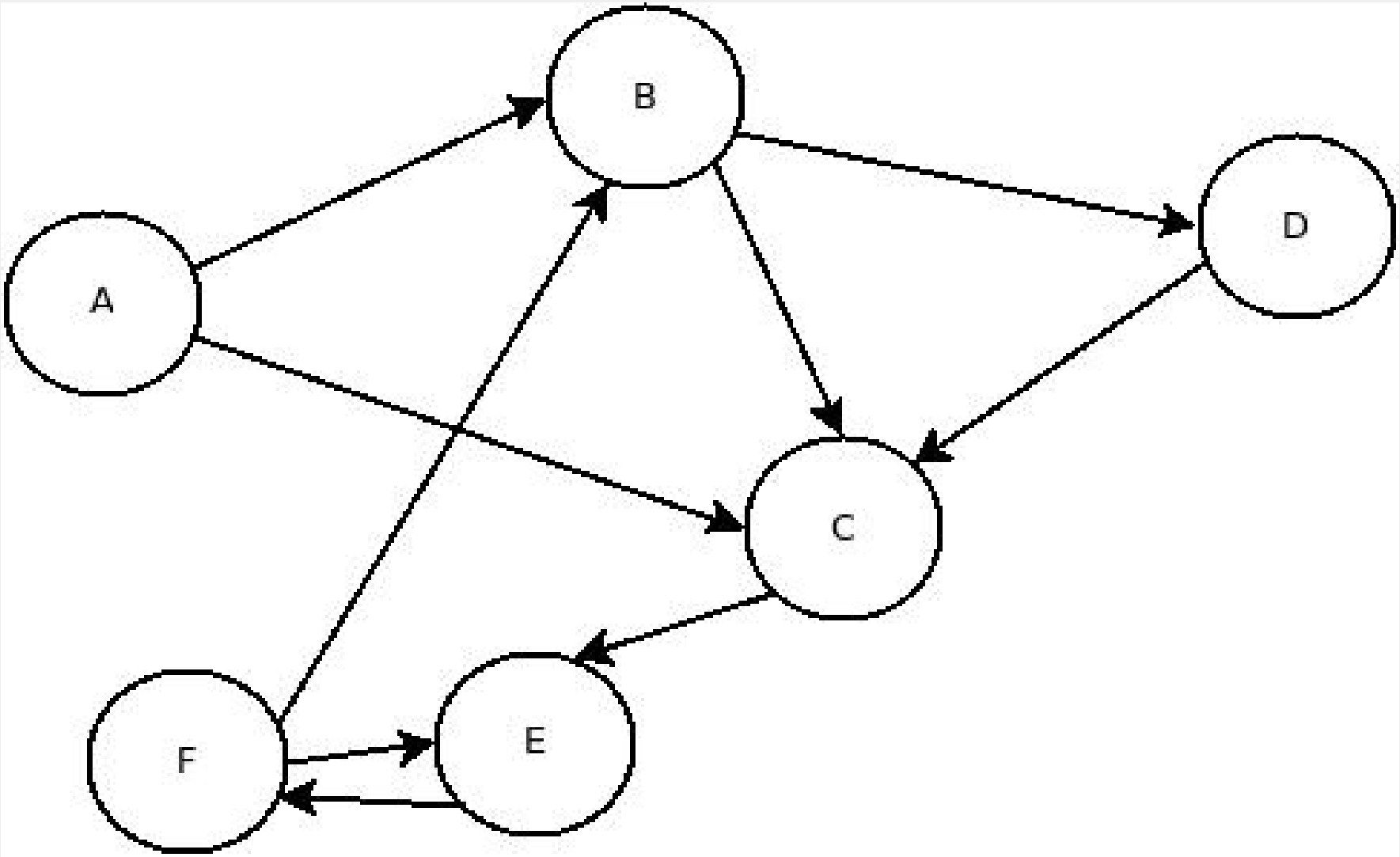
# Пример

Нека реализираме ранна версия на PageRank алгоритъма за оценяване на уеб-страници чрез метода на случайния посетител. За целта трябва да опишем коя страница към кои други страници сочи. Това на практика е насочен граф, който можем да реализираме в Python с речник. В него всеки ключ е възел, от който излизат ребра, а стойността му е списък с възлите, към които има ребра.

# Пример

Ребрата на графа ще опишем със следния набор от двойки:
A->B
A->C
B->C
B->D
C->E
D->C
E->F
F->B
F->E

# Пример



# Пример

Алгоритъмът се състои в следното: Започваме със случайна страница. В 5 от 6 случая посещаваме случайна страница, сочена от текущата. В останалия 1 път избираме да посетим случайна страница. PageRank е честотата, с която случен потребител посещава всяка страница.

# Пример

```
1  import random
2
3  graph = {'A': ['B', 'C'], 'B':['C', 'D'],
4  'C':['E'], 'D':['C'], 'E':['F'],
5  'F':['E', 'B']}
6  pages = graph.keys()
7  visitations = {}
8  for page in pages:
9      visitations[page] = 0
10 print(visitations)
11 current = random.choice(pages)
12 counter = 0
```

# Пример

```
13 while counter<2000:
14     counter += 1
15     choice = random.randint(0, 6)
16     if choice == 6:
17         current = random.choice(pages)
18     else:
19         if graph.has_key(current) and graph[current
               ]:
20             current = random.choice(graph[current])
21     visitations[current] += 1
22 print(visitations)
```

# Задача

В речник при зададен ключ намирането на съответната му стойност е тривиално, в крайна сметка това е предназначението му. Интересна е, обаче, обратната задача – търсенето на стойност. Да се създаде програма, която връща списък от всички ключове, които сочат към дадена стойност, а ако няма нито една такава стойност, то да се връща празен списък.

# Задача

Да се създаде хистограма на честотата на срещане на символите в даден текст. Например, ако текстът е „brontosaurus“, то резултатът трябва да е
'a': 1, 'b': 1, 'o': 2, 'n': 1, 's': 2, 'r': 2, 'u': 2, 't': 1

Благодаря за вниманието!

Въпроси?

# References