



Основы Python

Кортежи, списки, множества

Юдинцев В. В.

Кафедра теоретической механики

2 ноября 2018 г.



САМАРСКИЙ УНИВЕРСИТЕТ
SAMARA UNIVERSITY

Содержание

- 1 Кортежи
- 2 Операторы и списки
- 3 Строки
- 4 Атомарные и ссылочные типы
- 5 Множество (set)
- 6 Задание

Кортежи

Кортеж – неизменяемый список

Кортежи (`tuple`) – это **неизменяемые** списки

Кортеж создается перечислением элементов в круглых скобках

```
1 | a = (1, 2, 3, 4, 5)
```

или функцией `tuple`

```
1 | a = tuple(range(5))  
2 | a  
3 | (0, 1, 2, 3, 4)
```

Срезы

```
1 a = ( 'Меркурий' , 'Венера' , 'Земля' , 'Марс' )
```

Первые три элемента (с 0 по 2)

```
1 a[0:3]
2 ( 'Меркурий' , 'Венера' , 'Земля' )
```

Каждый второй (или с шагом 2), начиная с первого

```
1 a[::2]
2 ( 'Меркурий' , 'Земля' )
```

Каждый второй, начиная со второго

```
1 a[1::2]
2 ( 'Венера' , 'Марс' )
```

tuple или list

- Кортеж (tuple) – это **неизменяемый** список (list): удаление и добавление элементов невозможно:
 - `a.append(element)`
 - `a.extend(['Юпитер', 'Сатурн'])`
 - `a.insert(4, 'Фазтон')`
 - `del a[2]`
 - `a.remove('Земля')`
- Кортежи “работают” **быстрее** списков
- Кортежи могут использоваться как ключи словаря

```
1 colony = [ (1,1) , (1,2) , (1,3) ]
2
3 x = 2
4 y = 3
5
6 isOccupied = False
7
8 for cell in colony :
9     if x==cell[0] and y==cell[1]:
10         isOccupied = True
11         break
12
13 print(isOccupied)
```

```
1 colony = [ (1,1) , (1,2) , (1,3) ]
2
3 x = 2
4 y = 3
5
6 isOccupied = False
7
8 for cell in colony :
9     if (x, y) == cell :
10        isOccupied = True
11        break
12
13 print(isOccupied)
```



```
1 colony = [ (1,1) , (1,2) , (1,3) ]
2
3 x = 2
4 y = 3
5
6 isOccupied = (x,y) in colony
7
8 print(isOccupied)
```

или так

```
1 colony = [ [1,1] , [1,2] , [1,3] ]
2
3 x = 2
4 y = 3
5
6 isOccupied = [x,y] in colony
7
8 print(isOccupied)
```

Операторы и списки

Оператор +

```
1 x = (1, 2)
2
3 y = x + x
4
5 print(y)
```

Оператор +

```
1 x = (1, 2)
2
3 y = x + x
4
5 print(y)
```

```
(1, 2, 1, 2)
```

Оператор *

```
1 x = (1, 2)
2
3 y = x * 2
4
5 print(y)
```

```
(1, 2, 1, 2)
```

Строки

Строка – последовательность символов

```
1 | message = 'Quid est veritas?'
```

Первый элемент строки

```
1 | message[0]
2 | 'Q'
```

Отсчёт с конца строки

```
1 | message[-3]
2 | 'a'
```

Символы с 5 по 7

```
1 | message[5:8]
2 | 'est'
```

Движение в обратном порядке

```
1 | message[-2:-5:-1]
2 | 'sat'
```

Многострочный текст

Для создания много длинного **многострочного** текста используются тройные кавычки

```
1 poem = '''Над небом голубым –  
2 Есть город золотой,  
3 С прозрачными воротами  
4 И с яркою стеной.  
5  
6 А в городе том – сад:  
7 Всё травы да цветы.  
8 Гуляют там Животные  
9 Невиданной красоты... '''
```


Преобразование типов в строку

```
1 str(192.008)
2 '192.008'
3
4 str(192.36)*2
5 '192.008192.008'
```

Операции со строками

Сложение

```
1 a= 'Quid '  
2 b= 'est '  
3 a+ ' '+b  
4 'Quid est '
```

Умножение

```
1 a= 'B '  
2 b= 'z '  
3 a+b*4  
4 'Bzzzz '
```

Операции со строками

Разделение строки на список подстрок, разделяемых заданным символом – метод **строка.split(разделитель)**

```
1 a = '2-12-85-06'  
2  
3 a.split('-')  
4 ['2', '12', '85', '06']
```

Обратная операция объединения списка строк выполняется методом **разделитель.join(список)**

```
1 a = ['2', '12', '85', '06']  
2 '*'.join(a)  
3  
4 '2*12*85*06'
```

Операции со строками

Разделение строки на список подстрок, разделяемых заданным символом – метод **строка.split(разделитель)**

```
1 a = '2-12-85-06'  
2  
3 a.split('-')  
4 ['2', '12', '85', '06']
```

Обратная операция объединения списка строк выполняется методом **разделитель.join(список)**

```
1 a = ['2', '12', '85', '06']  
2 '*'.join(a)  
3  
4 '2*12*85*06'
```

Атомарные и ссылочные типы

Атомарные и ссылочные типы

Атомарные типы `bool`, `int`, `float`, `complex` копируются при присвоении

```
1 a = 10
2 b = a
3 b = b + 1
4 print(a)
5 10
6 print(b)
7 11
```

Переменная `b` содержит копию значения `a`

Атомарные и ссылочные типы

Ссылочные типы: списки, строки, кортежи, словари, множества

```
1 a = [1, 2, 3, 4, 5]
2 b = a
```

Атомарные и ссылочные типы

Ссылочные типы: списки, строки, кортежи, словари, множества

```
1 a = [1, 2, 3, 4, 5]
2 b = a
```

Заменяем **третий** элемент списка

```
1 b[2] = 0
2 print(b)
3 [1, 3, 0, 4, 5]
```


Атомарные и ссылочные типы

Ссылочные типы: списки, строки, кортежи, словари, множества

```
1 a = [1, 2, 3, 4, 5]
2 b = a
```

Заменяем **третий** элемент списка

```
1 b[2] = 0
2 print(b)
3 [1, 3, 0, 4, 5]
```

Объект (список), на который ссылается переменная **a** тоже изменился

```
1 print(a)
2 [1, 3, 0, 4, 5]
```

Переменная **b** ссылается на тот же объект, что и **a**.

Псевдоним и копия

Переменная `b` — это **имя**, которое ссылается на объект в памяти (`list`). На этот же объект ссылается и **имя** `a`:

```
1 a = [1, 2, 3, 4, 5]
2 b = a
```

Для того, чтобы создать копию, необходимо явно вызвать функцию-конструктор `list`

```
1 a = [1, 2, 3, 4, 5]
2 b = list(a)
3 b[0] = 0
4 a
5 [1, 2, 3, 4, 5]
6 b
7 [0, 2, 3, 4, 5]
```

Изменяемые и неизменяемые типы

Список – изменяемый тип

```
1 a = [1, 2, 3, 4, 5]
2 a[0] = 0
3 print(a)
4 [1, 2, 0, 4, 5]
```

Строка – неизменяемый тип

```
1 word = 'мир'
2 print(word[1])
```

и

```
1 word[0] = 'i'
```

TypeError: 'str' object does not support item assignment

Множество (set)

Множества

Определение

Неупорядоченный набор **неповторяющихся** элементов

Создание

- При помощи функции `set`

```
1 | s1 = set((1, 2, 3, 4, 5, 6))  
2 | s2 = set(('d', 'abc', 3, 4, 5, 6))
```

- Перечисление элементов множества в `{}` скобках

```
1 | s1 = {'d', 'abc', 3, 4, 5, 6}
```

Создание множества

Множество из элементов строки (букв)

```
1 a=set('окно')
2 print(a)
3
4 {'о', 'к', 'н'}
```

Множество из списка

```
1 a=set([1,2,3,4,6,4,6])
2 print(a)
3
4 {1,2,3,4,6}
```

Добавление элемента к множеству

```
1 a=set( [ 'Меркурий' , 'Венера' , 'Земля' , 'Марс' ] )  
2  
3 a.add( 'Юпитер' )  
4  
5 print(a)
```

{'Венера', 'Земля', 'Марс', 'Меркурий', 'Юпитер'}

Есть ли элемент в множестве?

```
1 a=set ( [ 'Меркурий' , 'Венера' , 'Земля' , 'Марс' ] )
```

Существует ли элемент в множестве - оператор **in**

```
1 print ( 'Юпитер' in a )  
2 False  
3 print ( 'Марс' in a )  
4 True  
5 print ( 'Юпитер' not in a )  
6 True
```


Пересечение множеств

A & B или **A.intersection(B)**

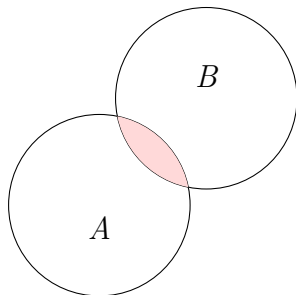
```
1 a=set ([1 , 2, 3, 4])  
2 b=set ([3 , 4, 5, 6])
```

Оператор **&**

```
1 print (a & b)  
2  
3 {3, 4}
```

Метод **intersection**

```
1 print (a.intersection (b))  
2  
3 {3, 4}
```



Объединение множеств

$A \mid B$ или `A.union(B)`

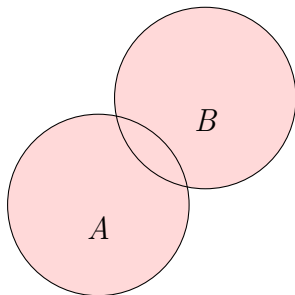
```
1 a=set([1, 2, 3, 4])
2 b=set([3, 4, 5, 6])
```

Оператор `|`

```
1 print(a | b)1, 2, 3, 4, 5, 6
```

Метод `union`

```
1 print(a.union(b))
2
3 {1, 2, 3, 4, 5, 6}
```



Разность множеств

Элементы, принадлежащие **A**, но не принадлежащие **B**

A - B или **A.difference(B)**

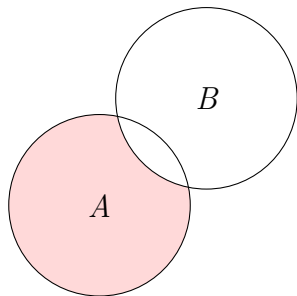
```
1 a=set ([1, 2, 3, 4])
2 b=set ([3, 4, 5, 6])
```

Оператор -

```
1 print (a - b)
2
3 {1, 2}
```

Метод **difference**

```
1 print (a.difference (b))
2
3 {1, 2}
```



Исключающее ИЛИ

Элементы, принадлежащие **A** или **B**, но не общие

$A \wedge B$ или **`A.symmetric_difference(B)`**

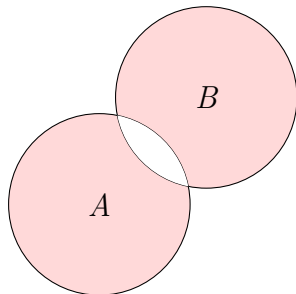
```
1 a=set([1, 2, 3, 4])
2 b=set([3, 4, 5, 6])
```

Оператор **\wedge**

```
1 print(a ^ b)
2
3 {1, 2, 5, 6}
```

Метод **`symmetric_difference`**

```
1 print(a.symmetric_difference(
    b))
2
3 {1, 2, 5, 6}
```



Подмножество

Все элементы, принадлежащие **A**, принадлежат **B**

A <= B или **A.issubset(B)**

```
1 a=set ([3 , 4])
2 b=set ([3 , 4 , 5 , 6])
```

Оператор **<=**

```
1 print (a <= b)
2
3 True
```

Метод **issubset**

```
1 print (a . issubset (b))
2
3 True
```

Подмножество

A >= B или **A.issuperset(B)**

```
1 a=set ([1, 2, 3, 4, 5, 6])
2 b=set ([3, 4, 5, 6])
```

Оператор **>=**

```
1 print (a >= b)
2
3 True
```

Метод **issuperset**

```
1 print (a.issuperset (b))
2
3 True
```

Задание

Координаты смежных клеток

	-1,1	0,1	1,1	
	-1,0	x,y	1,0	
	-1,-1	0,-1	1,-1	

	$x-1$ $y+1$	x $y+1$	$x+1$ $y+1$	
	$x-1$ y	x,y	$x+1$ y	
	$x-1$ $y-1$	x $y-1$	$x+1$ $y-1$	

Список координат смежных клеток

```
1 x, y = 2, 3
2
3 dxdy = [ ( 1, 1), ( 0, 1), (-1, 1),
4           (-1, 0), (-1,-1), ( 0,-1),
5           ( 1,-1), ( 1, 0) ]
6
7 nearest = list()
8
9 for (dx,dy) in dxdy:
10     nearest.append((x+dx, y+dy))
11
12 print(nearest)
```

[(3, 4), (2, 4), (1, 4), (1, 3), (1, 2), (2, 2), (3, 2), (3, 3)]

Множество координат смежных клеток

```
1 x, y = 2, 3
2
3 dxdy = [ ( 1, 1), ( 0, 1), (-1, 1),
4           (-1, 0), (-1,-1), ( 0,-1),
5           ( 1,-1), ( 1, 0) ]
6
7 nearest = set()
8
9 for (dx, dy) in dxdy:
10     nearest.add((x+dx, y+dy))
11
12 print(nearest)
```

{(1, 2), (3, 2), (1, 3), (3, 3), (1, 4), (2, 2), (3, 4), (2, 4)}

Координаты соседей (для списков)

```
1 colony = [ (1, 1), (1, 2), (1, 3) ]
2
3 x, y = 2, 3
4
5 neighbours = list()
6
7 for cell in nearest:
8     if cell in colony:
9         neighbours.append(cell)
10
11 print(neighbours)
```

[(1, 3), (1, 2)]

Координаты соседей (для множества)

```
1 colony = { (1, 1), (1, 2), (1, 3) }  
2  
3 x, y = 2, 3  
4  
5 neighbours = nearest & colony  
6  
7 print(neighbours)
```

```
{ (1, 2), (1, 3) }
```

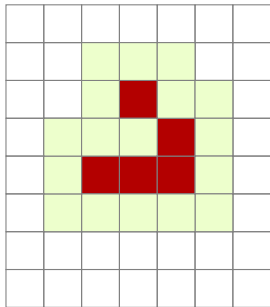
Количество соседей

Количество пар координат соседей – количество соседей

```
1 | len(neighbours)
```

```
1 | print("У клетки с координатами (", x, y,  
2 |       ") всего ", len(neighbours), " соседей")
```

Ареал колонии



Клетки, граничащие с колонией



Клетки, принадлежащие колонии

Ареал колонии

```
1 colony = [ (1, 1), (1, 2), (1, 3) ]
```

В ареал входят все живые клетки

```
2 area = list(colony)
```

и все клетки, граничащие с живыми

```
3 for cell in colony:
4     for (dx, dy) in dx dy:
5         dx dy_cell = (cell[0]+dx, cell[1]+dy)
6         if dx dy_cell not in area:
7             area.append(dx dy_cell)
8
9 print(area)
```

[(1, 1), (1, 2), (1, 3), (2, 2), (0, 2), (0, 1), (0, 0), (1, 0), (2, 0),
(2, 1), (2, 3), (0, 3), (2, 4), (1, 4), (0, 4)]

Ареал колонии – множество

```
1 colony = { (1, 1), (1, 2), (1, 3) }
```

В ареал входят все живые клетки

```
2 area = set(colony)
```

и все клетки, граничащие с живыми

```
3 for cell in colony:  
4     for (dx, dy) in dx dy:  
5         dx dy_cell = (cell[0]+dx, cell[1]+dy)  
6         area.add(dx dy_cell)  
7  
8 print(area)
```

{(1, 2), (0, 1), (1, 3), (0, 0), (0, 2), (1, 0), (2, 1), (1, 4), (2, 0), (2, 4),
(2, 3), (0, 4), (2, 2), (0, 3), (1, 1)}

Следующее поколение

```
1 colony = [ (1, 1), (1, 2), (1, 3) ]
2
3 next_generation = []
4 for cell in area:
5     cont_neighbours = 0
6     for (dx,dy) in dxdy:
7         nearest_cell = (cell[0]+dx, cell[1]+dy)
8         if nearest_cell in colony:
9             cont_neighbours+=1
10    if cont_neighbours == 3 or \
11    cont_neighbours == 2 and cell in colony:
12        next_generation.append(cell)
13 print(next_generation)
```

[(1, 2), (2, 2), (0, 2)]

Список источников

- 1 Простой Python. Современный стиль программирования. — СПб.: Питер, 2016. — 480 с.: ил. — (Серия «Бестселлеры O'Reilly»).
- 2 Python 3 для начинающих <https://pythonworld.ru>
- 3 Python documentation <https://docs.python.org>