

Лекция 6

Язык программирования Python.

Хайрулин Сергей Сергеевич

email: s.khairulin@g.nsu.ru, s.khayrulin@gmail.com

Ссылка на [материалы](#)

План

- Лекции/практические занятия
 - Тест
- Дифференцированный зачет в конце семестра
 - Защита задания

Литература

Начальный уровень

- Mark Pilgrim. Dive into Python - <http://www.diveintopython.net/>
- Марк Лутц. Изучаем Python, 4-е издание // Символ-Плюс 2011.
- ...

Стандарт/Документация

- PEP-8 - <https://www.python.org/dev/peps/pep-0008/>
- <https://www.python.org/>
- <https://github.com/python/cpython>

Экспертный уровень

- Лучано Рамальо: Python. К вершинам мастерства
- Mitchell L. Model. Bioinformatics Programming Using Python // O'Reilly 2010.

Версии Python

- Python 2 вышел 2010 году последняя версия 2.7.16 - исправлялись только баги(ошибки) с января 2020 года поддержка прекращена.
- Python 3 появился в 2008, является актуальной версией языка. Текущая стабильная версия 3.8.5 -> в предрелиз 3.9, в разработке 3.10
 - Python 3 не гарантирует совместимости кода с Python 2

План занятия

- Общее представление
- Hash function
- Словарь - Dict
 - definition
- Множество - Set
- Генераторы
- Практика

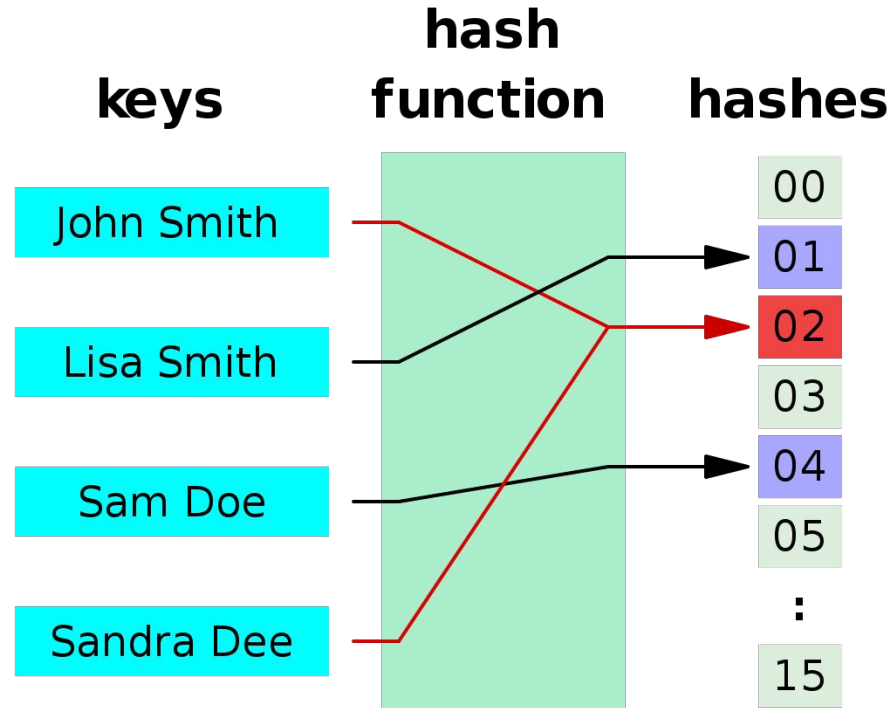
Hash function

Неоднозначная функция принимающая на вход некоторые данные и возвращающая некоторое целое число $\in [0, \dots, N]$. Хорошая хеш-функция должна удовлетворять двум свойствам:

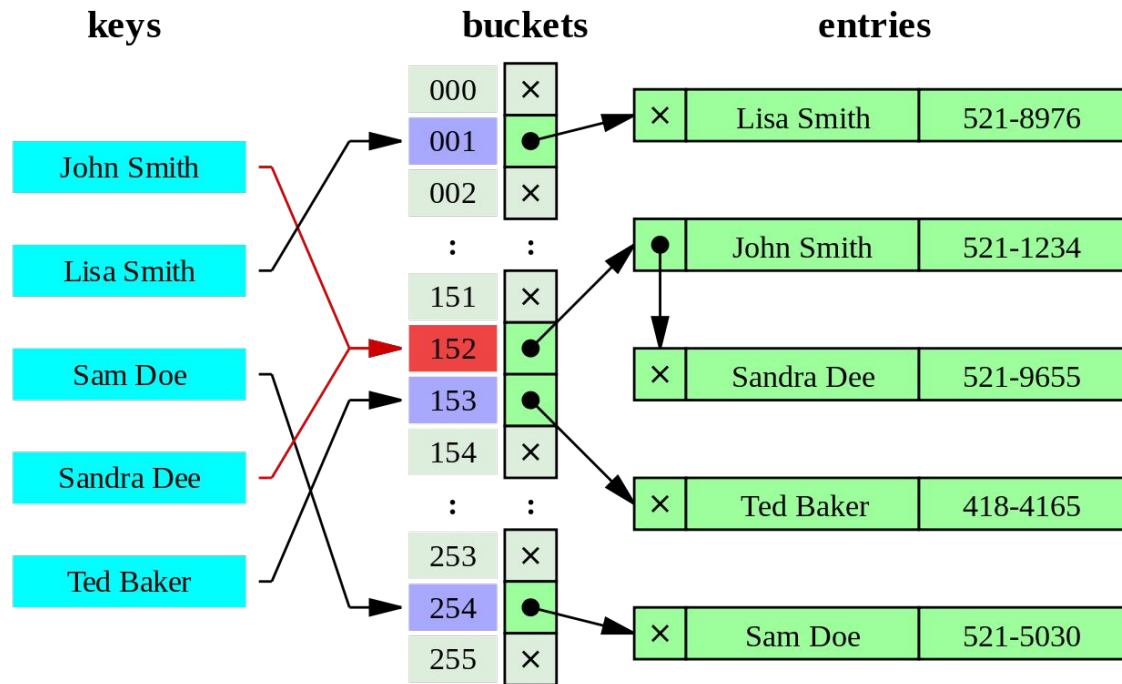
- быстрое вычисление
- минимально количество коллизий

Hash function

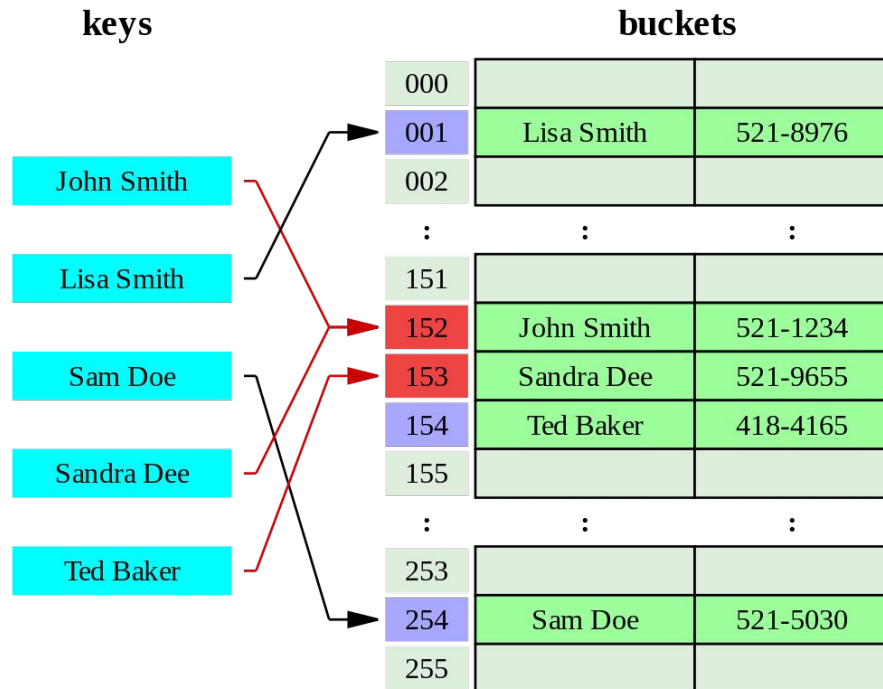
Hash function



Hash function (коллизия)



Hash function (коллизия)



Hash table

Еще называют хеш таблицей(дословный перевод), ассоциативный массив, словарь... Структура, которая не сохраняет порядок элементов при вставки, НО обеспечивает при этом быстрый доступ, быстрое удаление и вставку. В худшем и среднем случае работы этих функций производится за константное время в худшем за линейное.

Hash table

Python, как и почти все языки программирования дает возможность программисту создавать такие структуры. Ключами при этом могут быть только те типы данных, которые поддерживают операцию хеширования - значениями могут быть любые типы данных.

Hash table (python)

Словарь можно определить несколькими способами

- Явно
- Dict comprehension
- через цикл
- ключевое слово dict

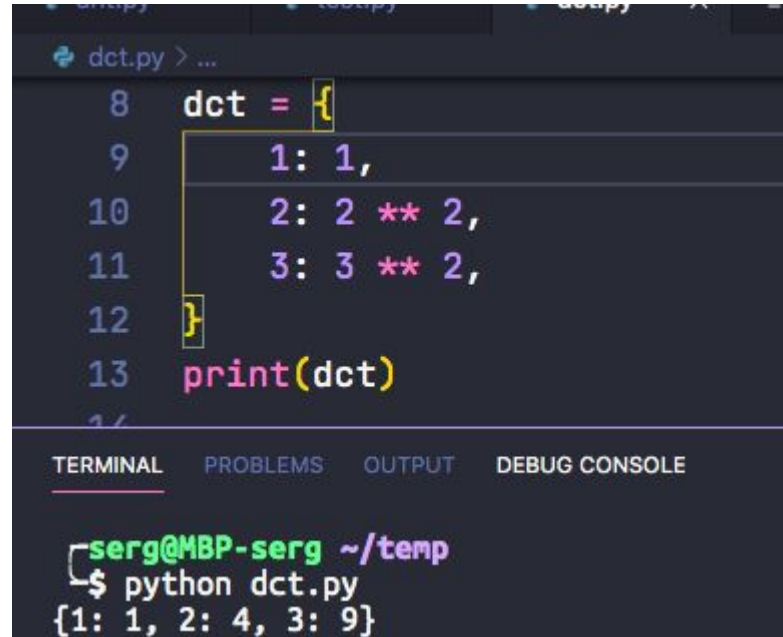
Hash table (python)



The image shows a code editor window with a dark theme. The top part displays a Python script named `dct.py` with three lines of code: `hash_table = {}`, `hash_table["key1"] = "value"`, and `print(hash_table)`. The bottom part shows a terminal window with the command `python dct.py` being executed, resulting in the output `{'key1': 'value'}`.

```
dct.py > ...  
1 hash_table = {}  
2 hash_table["key1"] = "value"  
3 print(hash_table)  
  
TERMINAL PROBLEMS 2 OUTPUT DEBUG CONSOLE  
serg@MBP-serg ~/temp  
$ python dct.py  
{'key1': 'value'}  
serg@MBP-serg ~/temp  
$
```

Python **dict** obj definition 1



The image shows a code editor window with a dark theme. The editor displays a Python script named `dct.py` with the following code:

```
8  dct = {  
9      1: 1,  
10     2: 2 ** 2,  
11     3: 3 ** 2,  
12 }  
13 print(dct)
```

Below the code editor, there is a terminal window. The terminal shows the command `python dct.py` being executed, and the output is `{1: 1, 2: 4, 3: 9}`.

```
serg@MBP-serg ~/temp  
$ python dct.py  
{1: 1, 2: 4, 3: 9}
```


Python **dict** obj definition 2

```
dct.py > ...
14
15  dct = {i: i**2 for i in range(1,4)}
16  print(dct)
17
```

TERMINAL PROBLEMS OUTPUT DEBUG CONSOLE

```
serg@MBP-serg ~/temp
$ python dct.py
{1: 1, 2: 4, 3: 9}
```

Python **dict** obj definition 3

```
21  dct = {}
22  for i in range(1, 4):
23      key = i
24      value = i ** 2
25      dct[key] = value
26  print(dct)
```

TERMINAL

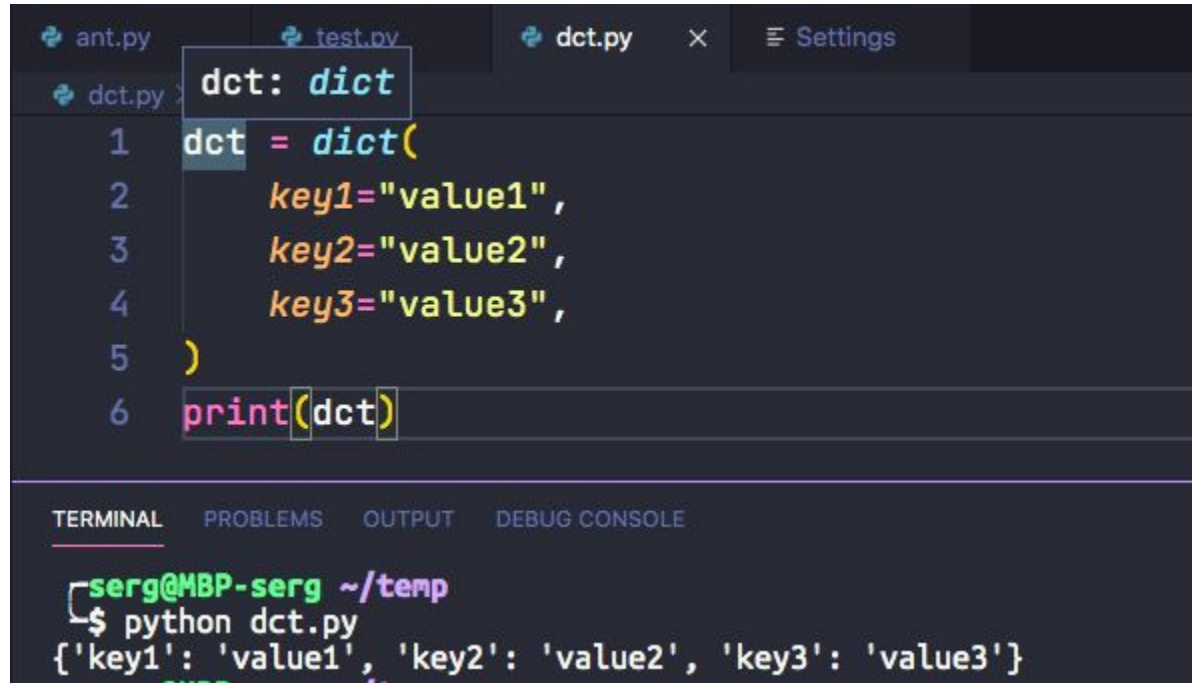
PROBLEMS

OUTPUT

DEBUG CONSOLE

```
serg@MBP-serg ~/temp
$ python dct.py
{1: 1, 2: 4, 3: 9}
```

Python **dict** obj definition 3



The image shows a code editor with a dark theme. At the top, there are tabs for 'ant.py', 'test.py', and 'dct.py'. The 'dct.py' tab is active. The code in 'dct.py' is as follows:

```
1 dct = dict(  
2     key1="value1",  
3     key2="value2",  
4     key3="value3",  
5 )  
6 print(dct)
```

A tooltip is visible over the word 'dict' on line 1, showing 'dct: dict'. Below the code editor is a terminal window with the following output:

```
serg@MBP-serg ~/temp  
$ python dct.py  
{'key1': 'value1', 'key2': 'value2', 'key3': 'value3'}
```

Python **dict** obj definition 4

```
>>> a = dict(one=1, two=2, three=3)
>>> b = {'one': 1, 'two': 2, 'three': 3}
>>> c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
>>> d = dict([('two', 2), ('one', 1), ('three', 3)])
>>> e = dict({'three': 3, 'one': 1, 'two': 2})
>>> f = dict({'one': 1, 'three': 3}, two=2)
>>> a == b == c == d == e == f
True
```

<https://docs.python.org/3/library/stdtypes.html#typesmapping>

Dict - операции

dict.clear() - очищает словарь.

dict.copy() - возвращает копию словаря.

dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).

dict.items() - возвращает пары (ключ, значение).

dict.keys() - возвращает ключи в словаре.

dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).

dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение `KeyError`. Помните, что словари неупорядочены.

Dict - операции

dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).

dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).

dict.values() - возвращает значения в словаре.

Dict (проход)

```
dct.py > ...
1  dct = {i: i**2 for i in range(1,4)}
2  print("-----")
3  for k in dct:
4      print(k, dct[k])
5  print("-----")
6  for k in dct.keys():
7      print(k, dct[k])
8  print("-----")
9  for v in dct.values():
10     print(v)
11 print("-----")
12 for k, v in dct.items():
13     print(k, v)
```

serg@MBP-serg ~/temp
\$ python dct.py

```
-----
1 1
2 4
3 9
-----
1 1
2 4
3 9
-----
1
4
9
-----
1 1
2 4
3 9
```

Dict - ключи

- immutable objects
- hashable
- comparable \Rightarrow obj1 == obj2

Dict - ключи

Class	Description	Immutable?
bool	Boolean value	✓
int	integer (arbitrary magnitude)	✓
float	floating-point number	✓
list	mutable sequence of objects	
tuple	immutable sequence of objects	✓
str	character string	✓
set	unordered set of distinct objects	
frozenset	immutable form of set class	✓
dict	associative mapping (aka dictionary)	

Dict - ключи

```
In [2]: d = {[1,2,3]: 1}
```

```
-----  
TypeError                                Traceback (most recent call last)  
<ipython-input-2-7e7847a84a44> in <module>()  
----> 1 d = {[1,2,3]: 1}
```

```
TypeError: unhashable type: 'list'
```

```
In [3]: d = {(1,2,3): 1}
```

```
In [4]: d
```

```
Out[4]: {(1, 2, 3): 1}
```

Set - множество

Структура данных очень похожая на словарь, но в отличие от словаря хранит только ключи. Ключи уникальны. Сам тип данных реализует теоретико-множественные операции.

Множество (Set)

```
In [13]: s1 = set((1,2,3))
```

```
In [14]: s2 = {1,2,3}
```

```
In [15]: s1
```

```
Out[15]: {1, 2, 3}
```

```
In [16]: s2
```

```
Out[16]: {1, 2, 3}
```

```
In [17]: s1 == s2
```

```
Out[17]: True
```

<https://docs.python.org/3/tutorial/datastructures.html#sets>

set операции

len(s) - вернет длину множества

x in s - проверить наличие элемента в множестве

x not in s - проверить отсутствие элемента в множестве

isdisjoint(other) - Верните **True**, если множество не имеет общих элементов с другими. Множества не пересекаются тогда и только тогда, когда их пересечение - пустое множество.

issubset(other) - наоборот

set операции

set <= other - Является ли каждый элемент в наборе состоит из другого.

set < other - Тоже самое но строго

set >= other

set > other

union(*others) ⇒ set | other | ... - Объединение множеств в одно результат
МНОЖЕСТВО

intersection(*others) \Rightarrow **set & other & ...** Пересечение всех множеств, то есть результат это множество из элементов общих для всех

difference(*others) \Rightarrow **set - other - ...**

set ^ other - Вернет новое множество элементов, которые содержатся только в одном из множеств, но не в обоих

copy() Верните неглубокую копию множества.

Множество (Set) операции

```
In [31]: 1 in s1
```

```
Out[31]: True
```

```
In [32]: s2.add(4)
```

```
In [33]: s2
```

```
Out[33]: {1, 2, 3, 4}
```

```
In [34]: s1 ^ s2
```

```
Out[34]: {4}
```

```
In [35]: s1 | s2
```

```
Out[35]: {1, 2, 3, 4}
```

```
In [36]: s1 & s2
```

```
Out[36]: {1, 2, 3}
```


Определение

Generator functions allow you to declare a function that behaves like an iterator, i.e. it can be used in a for loop (<https://wiki.python.org/moin/Generators>).

При создании генератора нет необходимости выделять такое количество памяти, которое необходимо для хранения всего массива данных.

Создание

```
In [1]: gen = (i for i in range(10))
```

```
In [2]: type(gen)
```

```
Out[2]: generator
```

```
In [3]: for i in gen:  
...:     print(i)  
...:
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Создание

```
1 class Arifmetic:
2     def __init__(self, start=0, step=1):
3         self.start = start
4         self.step = step
5
6     def __next__(self):
7         self.start += self.step
8         return self.start
9
10    def __iter__(self):
11        return self
12
13
14    a = Arifmetic(2, 2)
15    for an in a:
16        print(an)
17        if an >= 100:
18            break
```

Создание

```
class Arifmetic:
    def __init__(self, end, start=0, step=1):
        self.start = start
        self.n = end
        self.step = step
        self.cnt = 1

    def __next__(self):
        if self.cnt < self.n:
            self.cnt += 1
            self.start += self.step
            return self.start
        else:
            raise StopIteration()

    def __iter__(self):
        return self

a = Arifmetic(100, 2, 2)
print(sum(a))
```

Ключевое слово yield

```
home > serg > tmp > test.py > gen
1  def gen(start, end, step):
2      while start < end:
3          yield start
4          start += step
5
6
7  g = gen(1, 10, 1)
8  print(sum(g))
9
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[13:07:23] serg :: serg-pc → ~/tmp»
python test.py
45
[13:07:25] serg :: serg-pc → ~/tmp»
```

Ключевое слово yield

```
1 def gen(start, end, step):
2     while start < end:
3         yield start
4         start += step
5
6
7 g = gen(1, 10, 1)
8 for i in g:
9     print(i)
10
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[13:10:56] serg :: serg-pc → ~/tmp»
python test.py
```

```
1
2
3
4
5
6
7
8
9
```

```
[13:10:57] serg :: serg-pc → ~/tmp»
```

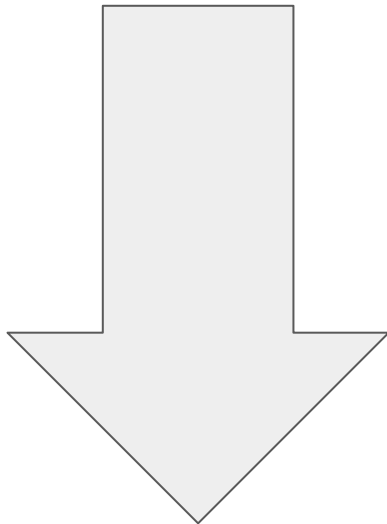
Ключевое слово yield

```
1  def gen(start, end, step):
2      while start < end:
3          yield start
4          start += step
5
6
7  g = gen(1, 10, 1)
8  for i in g:
9      print(i)
10
11 for i in g:
12     print(i)
13
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
[13:09:18] serg :: serg-pc → ~/tmp»
python test.py
1
2
3
4
5
6
7
8
9
[13:10:31] serg :: serg-pc → ~/tmp»
```

Практическая Часть



1. Постройте список с повторяющимися значениями и создайте из него множество
2. Определите основные теоретико множественные операции, объединение, пересечение, разность двух множеств
3. Расширьте ваши функции для работы с бесконечно большим количеством множеств.
4. Как можно просто в этой строке
'rewlkdfsklgjdfklkjglkdsfjgkldfsjgliiiiiiiiierwtsj;kldfjg;lksdfjgl;ksdjfl;gj;l sdfjg;lk' - удалить все повторяющиеся элементы
5. Предположим вы вводите строку, используя стандартную функцию `input()` посчитайте и выведите какое количество разных символов в этой строке
6. напишите функцию `concatenate(dict1, dict2)`, которая объединяет два словаря и возвращает результат, выведете результат
7. Реализуйте программу заполняющую телефонный справочник. При этом заполнение справочника должно осуществляться из строки ввода, учтите, что у человека может быть несколько телефонов. Добавьте специализированную команду, с помощью которой можно выводить справочник на экран.

8. Создайте генератор числовых значений от 0 до 100. Переберите все значения этого генератора в цикле
9. Создайте генератор квадратов элементов предыдущего списка. Переберите все значения этого генератора в цикле
10. Создайте генератор, состоящий из четных элементов предыдущего списка. Переберите все значения этого генератора в цикле