



ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ
МГТУ им. Н.Э. Баумана

Программирование на языке Python. Уровень 1.

Коллекции и функции.

Коллекции в Python

- ❑ Коллекции – группы типов данных, которые содержат в себе другие данные и поддерживают:
 - ✓ проверку на вхождения элементов `in` и `not in` (`True/False`);
 - ✓ определение размера `len()`;
 - ✓ возможность выполнения итераций (перемещения по элементам последовательности) – из-за этого коллекции также называются итерируемыми объектами.

- ❑ Среди коллекций выделяют 3 группы:
 - ✓ последовательности: строка, список, кортеж, числовой диапазон;
 - ✓ множества;
 - ✓ отображения: словарь.

Работа со списками

- ❑ Список (list) – это структура данных для хранения объектов различных типов.
- ❑ Список является изменяемым типом данных.
- ❑ Переменная, определяемая как список, содержит ссылку на структуру в памяти, которая в свою очередь хранит ссылки на какие-либо другие объекты или структуры.
- ❑ Python размещает элементы списка в памяти, затем размещает указатели на эти элементы. Список в Python – это массив указателей на элементы, размещенные в памяти.

Работа со списками. Объявление списка.

❑ Для объявления списка в Python существует несколько способов.

❑ Вариант №1: Через выражение, создающее объект:

```
>>> elements = [1, 3, 5, 6]
```

```
>>> print(elements)
```

```
[1, 3, 5, 6]
```

Если нужен пустой список, в квадратных скобках ничего не указывается:

```
>>> elements = [].
```


❑ Вариант №2: Через функцию list():

```
>>> elements = list()
```

```
>>> print(elements) []
```

В данном примере создается пустой список.

[1, 2, "str", True, [7, 2]]



Обращение к элементу списка в Python

- ❑ Индексом элемента называют его порядковый номер в списке.
- ❑ Нумерация элементов списка в направлении слева направо начинается с нуля.
- ❑ Отрицательными индексами нумеруются элементы в списке справа налево.
- ❑ Отрицательным индексом удобно пользоваться, когда необходимо обратиться к последнему элементу в списке, не вычисляя его номер.
- ❑ Любой конечный элемент в направлении справа налево будет нумероваться индексом, равным -1.
- ❑ Пример.

```
elements = [1, 2, 3, "word"]
```

- ❑ Индексы (позиции в списке) соответственно будут: 0, 1, 2, 3.

```
print(elements[1]) # 2
print(elements[3]) # word
print(elements[-1]) # word
print(elements[-3]) # 2
```

elements[elements]-1]

elements[-1]

Обращение к элементу списка в Python

❑ Список может содержать объекты разных типов: числовые, буквенные, а также списки (вложенные списки).

❑ Пример. Список списков:

```
>>> elements = [[1, 2, 3], ['a', 'b', 'c'], [0.1, 0.2, 0.3]]
```

❑ Для обращения к элементу вложенного списка нужно использовать два индекса: первый указывает на индекс главного списка, второй – индекс элемента во вложенном списке.

❑ Пример.

```
elements = [{"яблоки", 50}, {"апельсины", 190}, {"киви", 100}]
```

```
>>> print(elements[0])
```

```
['яблоки', 50]
```

```
>>> print(elements[1][0])
```

```
апельсины
```

элемент[0][0]
киви → элемент[2][0]

Методы списков.

Метод	Действие
<u>list.append(x)</u>	Добавляет элемент в конец списка
list.extend(L)	Расширяет список list, добавляя в конец все элементы списка L
list.insert(i, x)	Вставляет на i-ый элемент значение x
list.remove(x)	Удаляет первый элемент в списке, имеющий значение x.
list.pop([i])	Удаляет i-ый элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(x, [start [, end]])	Возвращает положение первого элемента со значением x (при этом поиск ведется от start до end)
list.count(x)	Возвращает количество элементов со значением x
list.sort([key=функция])	Сортирует список на основе функции
list.reverse()	Разворачивает список
list.copy()	Поверхностная копия списка
list.clear()	Очищает список

Кортежи в Python

❑ Кортежи, как и списки, являются упорядоченными последовательностями элементов. Кортеж – неизменяемый тип данных.

❑ Кортеж – это список, доступный только для чтения.

❑ Создать кортеж можно следующими способами:

➤ с помощью функции `tuple(<Последовательность>)`. Функция позволяет преобразовать любую последовательность в кортеж. Если параметр не указан, создается пустой кортеж:

```
>>> tuple() # Создаем пустой кортеж
>>> tuple("String") # Преобразуем строку в кортеж
('S', 't', 'r', 'i', 'n', 'g')
# Преобразуем список в кортеж:
>>> tuple([1, 2, 3, 4, 5])
(1, 2, 3, 4, 5)
```

➤ указав все элементы через запятую внутри круглых скобок:

```
t1 = () # Создаем пустой кортеж
t2 = (5,) # Создаем кортеж из 1-го элемента
t3 = (1, "str", (3, 4)) # Кортеж из трех элементов
```

Множества в Python

❑ **Множество** – это неупорядоченная последовательность уникальных элементов.

❑ Объявить множество можно с помощью функции `set()`:

```
>>> s = set()
```

```
>>> s
```

```
set([])
```

❑ Функция `set()` позволяет преобразовать элементы последовательности во множество:

```
>>> set("string") # Преобразуем строку
```

```
set(['g', 'i', 'n', 's', 'r', 't'])
```

```
>>> set([1, 2, 3, 4, 5]) # Преобразуем список
```

```
set([1, 2, 3, 4, 5])
```

```
>>> set((1, 2, 3, 4, 5)) # Преобразуем кортеж
```

```
set([1, 2, 3, 4, 5])
```

```
>>> set([1, 2, 3, 1, 2, 3])
```

```
>>> # Остаются только уникальные элементы
```

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

Методы работы со множествами

Название	Назначение
len	Получение размера
add	Добавление элемента
remove	Удаление элемента
clear	Очистка
union	Объединение
update	Добавление всех элементов одного множества в другое
intersection	Нахождение множества, элементы которого находятся на пересечении двух множеств
difference	Нахождение множества, элементы которого входят в первое, но не входят во второе множество
issubset	Проверка, является ли множество подмножеством
issuperset	Проверка, является ли множество надмножеством

Отображения. Словари.

- ❑ **Отображение** - это неупорядоченная коллекция пар элементов «ключ-значение».
- ❑ Отображения в Python представлены единственным типом: `dict` (словарь).
- ❑ **Словари** – наборы объектов, доступ к которым осуществляется не по индексу, а по ключу. В качестве ключа можно указать неизменяемый объект, например: число, строку или кортеж.
- ❑ **Элементы словаря** могут содержать объекты произвольного типа данных и иметь неограниченную степень вложенности!
- ❑ Элементы в словарях располагаются в произвольном порядке. Чтобы получить элемент, необходимо указать ключ, который использовался при сохранении значения.
- ❑ **Функции и методы**, предназначенные для работы с последовательностями, к словарям не применимы!
- ❑ Словари относятся к изменяемым типам данных.
- ❑ Существует возможность не только получить значение по ключу, но и изменить его.

Создание словарей

- ❑ Создать словарь можно следующими способами:

- С помощью функции `dict()`.

Форматы функции:

`dict(<Ключ1>=<Значение1>[, ..., <КлючN>=<ЗначениеN>])`

`dict(<Словарь>)`

`dict(<Список кортежей с двумя элементами (Ключ, Значение)>)`

`dict(<Список списков с двумя элементами [Ключ, Значение]>)`

Если параметры не указаны, то создается пустой словарь.

- Указав все элементы словаря внутри фигурных скобок.
- Заполнив словарь поэлементно. В этом случае ключ указывается внутри квадратных скобок.
- С помощью метода:

`dict.fromkeys (<последовательность> [, <значение>]).`

Метод создает новый словарь, **ключами** которого будут элементы последовательности, переданной **первым параметром**, а их **значениями** – величина, переданная **вторым параметром**.

Если второй параметр не указан, то значением элементов словаря будет значение *None*.

Методы и операции словарей

❑ Операции:

- **d[key]** - возвращает значение словаря для ключа **key**. Если ключ не существует, возникает ошибка.
- **d[key] = value** - устанавливает значение словаря по ключу **key**. Если ключ не существует, он создается.

❑ Методы:

- **get(key[, default])** - возвращает значение словаря для ключа **key**. Если ключ не существует возвращается значение **default** или **None**.
- **items()** - возвращает набор пар «ключ-значение» для словаря **d**.
- **keys()** - возвращает набор ключей для словаря **d**.
- **values()** - возвращает набор значений для словаря **d**.
- **clear()** - удаляет из словаря все элементы.
- **del d[key]** - удаляет из словаря пару «ключ-значение» на основании ключа **key**.

➤ * **d** - словарь, на котором вызывается метод.

Понятие функции

- **Функции** – это самостоятельные единицы программы, разработанные для решения конкретных задач, обычно повторяющихся несколько раз.
- **Функция** является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке).
- **Сигнатура** функции определяет правила использования функции. Сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров и тип возвращаемого значения.
- **Семантика** функции определяет способ реализации функции. Представляет собой тело функции.

Определение функций в Python

- Основная форма определения (definition) функции:

проверить с чем про?

```
def <Имя функции>([<Параметры>]) :  
ТАК <Тело функции>  
    [return <Результат>]
```

(a, b, item) :

- Имя функции должно быть уникальным идентификатором, состоящим из латинских букв, цифр и знаков подчеркивания, причем имя функции не может начинаться с цифры.
- В качестве имени функции нельзя использовать ключевые слова, кроме того, следует избегать совпадений с названиями встроенных идентификаторов.
- Регистр символов в имени функции имеет значение.

Сохранение ссылки на функцию в переменной

- Инструкция **def** создает объект, имеющий тип **function** и сохраняет ссылку на него в идентификаторе, указанном после инструкции **def**.
- Таким образом существует возможность сохранить ссылку на функцию в другой переменной — для этого название функции указывается без круглых скобок.

- **Пример.**

Сохранение ссылки в переменной и вызов функции через неё.

```
def summa(x, y):  
    return x + y
```

```
f = summa          # Сохранение ссылки в переменной
```

```
v = f(10, 20)      # Вызов функции через переменную
```

summa(5) → error

Параметры функции по умолчанию

- Для того, чтобы сделать некоторые параметры функции необязательными, следует в определении функции присвоить этому параметру начальное значение (значение по умолчанию).
- **Пример.**

В функции суммирования двух чисел – второй параметр необязательный (задан по умолчанию).

```
def summa(x, y=2): # y – необязательный параметр
```

```
    return x + y
```

Переменной **a** будет присвоено значение 7:

```
a = summa(5) x=5, y=2
```

Переменной **b** будет присвоено значение 60:

```
b = summa(10, 50) x=10, y=50
```

- Необязательные параметры должны следовать после обязательных!

Переменное число параметров в функции

- Если перед параметром в определении функции указать символ *, то функции можно будет передать произвольное количество параметров.
- Все переданные параметры сохраняются в кортеже.
- **Пример.**

Функция суммирования произвольного количества чисел.

```
def summa(*t):  
    """ Функция принимает произвольное количество параметров """  
    res = 0  
    for i in t: # Перебор кортежа с переданными параметрами  
        res += i  
    return res  
  
print(summa(10, 20)) # Выведет: 30  
print(summa(10, 20, 30, 40, 50, 60)) # Выведет: 210
```

Переменное число параметров в функции

- Допускается вначале указать несколько обязательных параметров и параметров, имеющих значения по умолчанию.

- Пример.**

Функция с комбинацией параметров разных типов.

```
def summa(x, y=5, *t): # Комбинация параметров
    """ Функция принимает комбинацию параметров """
    res = x + y
    for i in t: # Перебор кортежа с переданными параметрами
        res += i
    return res
print(summa(10)) # Выведет: 15
print(summa(10, 20, 30, 40, 50, 60)) # Выведет: 210
```

Встроенная функция map()

- Встроенная функция `map()` позволяет применить заданную в параметре функцию к каждому элементу последовательности.

- Формат:

```
map(<Функция>, Последовательность1> [,...,<ПоследовательностьN>])
```

- Функция `map()` возвращает объект, поддерживающий итерации.
- В качестве параметра `<Функция>` указывается ссылка на функцию (название функции без круглых скобок), которой будет передаваться текущий элемент последовательности. Внутри этой функции необходимо вернуть новое значение.

- Пример.** Увеличение каждого элемента списка на 100.

```
def func(elem):  
    """ Увеличение значения каждого элемента списка """  
    return elem + 100 # Возврат нового значения  
  
arr = [1, 2, 3, 4, 5]  
print(list(map(func, arr)))  
# Результат выполнения: [101, 102, 103, 104, 105]
```

Встроенная функция zip()

- Встроенная функция `zip()` на каждой итерации возвращает кортеж, содержащий элементы последовательностей, которые расположены на одинаковом смещении.

- Формат:

`zip(Последовательность1> [, ..., <ПоследовательностьN>])`

- Функция возвращает объект, поддерживающий итерации.

- Пример.

list

```
>>> print(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
```

→ `<zip object at 0x0000000002E4C100>`

```
>>> print(list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9])))
```

```
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
```

```
>>> print(list(zip([1, 2, 3], [4, 6], [7, 8, 9, 10])))
```

```
[(1, 4, 7), (2, 6, 8)]
```

Встроенная функция filter()

- Встроенная функция filter() позволяет выполнить проверку элементов последовательности.

- Формат:

filter(<Функция>, <Последовательность>)

- Функция возвращает объект, поддерживающий итерации.
- Если в первом параметре вместо названия функции указать значение None, то каждый элемент последовательности будет проверен на соответствие значению True.
- Если элемент в логическом контексте возвращает значение False, то он не будет добавлен в возвращаемый результат.

- Пример.

```
>>> filter(None, [1, 0, None, [], 2])  
<filter object at 0x000000002ECD970>  
>>> list(filter(None, [1, 0, None, [], 2]))  
[ 1 , 2 ]
```

Handwritten diagram:
A red bracket is drawn under the list [1, 0, None, [], 2] in the code above. Below the list, there are handwritten red symbols: a large 'S' followed by a sequence of '+' and '-' signs, likely representing a filter operation or a sequence of elements being processed.

Handwritten code:
def s(x):
 return x >= 2

Анонимные функции

- Кроме обычных, язык Python позволяет использовать анонимные функции, которые также называются лямбда-функциями.
- Анонимная функция не имеет имени и описывается с помощью ключевого слова `lambda`.
- Формат:
`lambda [<Параметр1>[, ..., <ПараметрN>]]: <Возвращаемое значение>`
- После ключевого слова `lambda` допускается указать передаваемые параметры.
- В качестве параметра `<возвращаемое значение>` указывается выражение, результат выполнения которого будет возвращен функцией.
- В качестве значения анонимная функция возвращает ссылку на объект-функцию, которую можно сохранить в переменной или передать в качестве параметра другой функции.

```
f1 = lambda: 10 + 20          # Функция без параметров
f2 = lambda x, y: x + y      # Функция с двумя параметрами
f3 = lambda x, y, z: x + y + z # Функция с тремя параметрами
print(f1())                  # Выведет: 30
print(f2(5, 10))             # Выведет: 15
print(f3(5, 10, 30))         # Выведет: 45
```

Функция-генератор

- Функция-генератор представляет собой особую разновидность функции, которую можно использовать для определения собственных итераторов.
- При определении функций-генераторов значение каждой итерации возвращается ключевым словом `yield`.
- Генератор перестает возвращать значения, когда итераций больше нет, при достижении пустой команды `return` или конца функции.
- Локальные переменные в функции-генераторе сохраняются между вызовами (в отличие от обычных функций)!
- Функция-генератор, например, может содержать цикл `while`, ограничивающий количество выполнений генератора.
- В зависимости от способа использования генератор, в котором не предусмотрено условие остановки, может создать бесконечный цикл в программе.



**ЦЕНТР
ДОПОЛНИТЕЛЬНОГО
ОБРАЗОВАНИЯ**
МГТУ им. Н.Э. Баумана



do.bmstu.ru