Тема: Python. Основы. Массивы / Списки

Массивы

- Массив (в некоторых языках программирования также таблица, ряд, матрица, вектор, список) — структура данных, хранящая набор значений (элементов массива), идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.
- Каждая ячейка в массиве имеет уникальный номер (индекс).

Массив

Индекс	0	1	2	3	4	5	6	7
Значение	1	2	3	4	5	6	7	8

Массивы = Списки

- В языке Python нет такой структуры данных, как массив.
- Для хранения группы однотипных объектов используют списки (тип данных `list`).
- В отличие от массивов в других языках, у списков нет никаких ограничений на тип переменных, поэтому в них могут храниться объекты разного типа.

- Списки являются упорядоченными последовательностями, которые состоят из различных объектов (значений, данных), заключающихся в квадратные скобки [ ] и отделяющиеся друг от друга с помощью запятой.

- Пример:

```
list1 = [‘физика', ’химия', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
list3 = ["a", "b", "c", "d"]
```

Создание списков

Создание списков на Python

1. Получение списка через присваивание конкретных значений
2. Списки при помощи функции List()
3. Создание списка при помощи функции Split()
4. Генераторы списков

1. Получение списка через присваивание конкретных значений

- Так выглядит в коде Python пустой список:

```
s = []  # Пустой список
```

- Примеры создания списков со значениями:

```
l = [25, 755, -40, 57, -41]   # список целых чисел
l = [1.13, 5.34, 12.63, 4.6, 34.0, 12.8] # список из дробных чисел
l = ["Sveta", "Sergei", "Ivan", "Dasha"]#список из строк
l = ["Москва", "Иванов", 12, 124]# смешанный список
l = [[0, 0, 0], [1, 0, 1], [1, 1, 0]] # список,
состоящий из списков
l = ['s', 'p', ['isok'], 2] # список из значений и списка
```

1. Получение списка через присваивание конкретных значений

- **Ввод значений с клавиатуры**

```
N = 5          # размер массива
B = [0] * N # заполнение массива нулями
print ("Введите", N, "элементов массива:")
for i in range(N):  # перебор индексов
    B[i] = int(input())
    # ввод числа с клавиатуры
```

2. Списки при помощи функции List()

- **Получаем список при помощи функции List()**

```
empty_list = list() # пустой список

l = list ('spisok')  # 'spisok' – строка
print(l) #['s', 'p', 'i', 's', 'o', 'k’] # результат
– список
```

2. Списки при помощи функции List()

```
# Создание списка из непрерывной
последовательности целых чисел

L = list(range(1,10))
print("L = ",L)

# L = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3. Создание списка при помощи функции Split()

- Используя функцию split в Python можно получить из строки список.
- Рассмотрим пример:

```
stroka ="Hello, world"  # stroka – строка
lst=stroka.split(",") # lst – список
lst # ['Hello', ' world']
```

4. Генераторы списков

- В python создать список можно при помощи генераторов.
- Первый простой способ. Сложение одинаковых списков заменяется умножением:

```
# список из 10 элементов, заполненный единицами
l = [1]*10
# список l = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

- Второй способ сложнее.

```
l = [i for i in range(10)]
# список l = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

или такой пример:

```
c = [c * 3 for c in 'list']
print (c) # ['lll', 'iii', 'sss', 'ttt']
```

Доступ к элементам списка

Доступ к значениям в списках

Чтобы получить доступ к значениям в списках, используйте квадратные скобки для нарезки вместе с индексом или индексами, чтобы получить значение, доступное по этому индексу.

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5, 6, 7 ];
print("list1[0]: ", list1[0])
print("list2[1:5]: ", list2[1:5])
```

```
list1[0]:  physics
list2[1:5]:  [2, 3, 4, 5]
```

Доступ к значениям в списках

- 1 самый простой способ

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

```
[1, 2, 3, 4, 5]
```

- 2 способ с помощью цикла

```
for i in range(len(mylist)):
    print(mylist[i], end = " ")
```

```
1 2 3 4 5
```

Доступ к значениям в списках

```
myList=[2.5, 8, "Hello"]
myList[0]
myList[1]
myList[2]

# вывести элемент списка по индексу 0
print(myList[0])
# вывести элемент списка по индексу 1
print(myList[1])
# вывести элемент списка по индексу 2
print(myList[2])
```

```
2.5
8
Hello
```

Простейшие операции над списками

Обновление значений в списке

```
list = ['Физика', 'Химия', 1997, 2000]
print ("Значение 2 значения в списке : ")
print (list[2])

list[2] = 2020;

print ("Новое значение 2 значения в списке: ")
print (list[2])
```

```
Значение 2 значения в списке :
1997
Новое значение 2 значения в списке:
2020
```

Удаление элементов списка - del

```
list = ['Физика', 'Химия', 1997, 2000]
print ("Значение 2 индекса в списке : ")
print (list[2])

del list[2];

print ("Новое значение 2 индекса в списке: ")
print (list[2])
```

```
Значение 2 индекса в списке :
1997
Новое значение 2 индекса в списке:
2000
```

Сложение списков

```
l = [1, 3] + [4, 23] + [5]
# Результат:  l = [1, 3, 4, 23, 5]


[33, -12, 'may'] + [21, 48.5, 33]
# [33, -12, 'may', 21, 48.5, 33]


a=[33, -12, 'may']
b=[21, 48.5, 33]
print(a+b)
# [33, -12, 'may', 21, 48.5, 33]
```

Использование элемента списка в выражении

```
# использование списка в выражении
L=[2,3,4]
x=5
y=x+L[1]  # y=5+3=8
print("y = ",y)

LS = ["456", 7, 3.1415]
s = "123"
s += LS[0]  # s="123456"
print("s = ", s)
```

```
y =  8
```

```
s =  123456
```

Основные операции со списком

Python Expression	Результаты	Описание
`len([1, 2, 3])`	`3`	Length – длина
`[1, 2, 3] + [4, 5, 6]`	`[1, 2, 3, 4, 5, 6]`	Concatenation – конкатенация. Сложение
`['Hi!'] * 4`	`['Hi!', 'Hi!', 'Hi!', 'Hi!']`	Repetition – Повторение
`3 in [1, 2, 3]`	`True`	Membership – членство
`for x in [1, 2, 3]: print x,`	`1 2 3`	Iteration – итерация

Встроенные функции и методы списка

№	Функция с описанием
1	**cmp(list1, list2)** Сравнивает элементы обоих списков.
2	**len(list)** Дает общую длину списка.
3	**max(list)** Возвращает элемент из списка с максимальным значением.
4	**min(list)** Возвращает элемент из списка с минимальным значением.
5	**list(seq)** Преобразует кортеж в список.

Встроенные функции и методы списка

№	Методы с описанием
1	**list.append(obj)** Добавляет объект obj в список
2	**list.count(obj)** Возвращает количество раз, сколько obj встречается в списке
3	**list.extend(seq)** Добавляет содержимое seq в список
4	**list.index(obj)** Возвращает самый низкий индекс в списке, который появляется obj
5	**list.insert(index, obj)** Вставляет объект obj в список по индексу смещения
6	**list.pop(obj=list[-1])** Удаляет и возвращает последний объект или объект из списка
7	**list.remove(obj)** Удаляет объект obj из списка
8	**list.reverse()** Переворачивает объекты списка на месте
9	**list.sort([func])** Сортирует объекты списка, используйте функцию сравнения, если дано

Как определить длину списка? Операция len.

```
# Определение длины списка операцией len
A = [ 3.5, 2.8, 'abc', [ 2, 3, False]]
length = len(A)
print("length = ", length) # length = 4

B = [ "Hello world!" ]
length = len(B)
print("length = ", length) # length = 1

C = [ 0, 3, 2, 4, 7 ]
length = len(C)
print("length = ", length) # length = 5
```

```
length =  4
length =  1
length =  5
```

Пример создания списка, содержащего другие сложные объекты

В примере создается список с именем D, который содержит другие сложные объекты: два списка с именами A, B; // кортеж с именем C; // строку с именем STR.

```
# Пример списка, содержащего сложные объекты
# объявляются списки, кортеж и строка символов
A = [] # пустой список
B = [ 1, 3, -1.578, 'abc' ] # список з разнотипными объектами
C = ( 2, 3, 8, -10 ) # кортеж
S = "Hello world!"

# список, содержащий разные сложные объекты
D = [ A, B, C, S ]
print("D = ", D)
```

```
D =  [[], [1, 3, -1.578, 'abc'], (2, 3, 8, -10), 'Hello world!']
```

Методы списков - list.append(x)

- `list.append(x)- Добавляет элемент в конец списка.`
- `Ту же операцию можно сделать так a[len(a):] = [x]:`

```
a = [1, 2]
a.append(3)
print(a)
```

```
[1, 2, 3]
```

Методы списков - list.extend(L)

- list.extend(L) – Расширяет существующий список за счет добавления всех элементов из списка L.
- Эквивалентно команде a[len(a):] = L:

```
a = [1, 2]
b = [3, 4]
a.extend(b)
print(a)
```

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

Методы списков - list.insert(i, x)

- list.insert(i, x) – Вставить элемент x в позицию i. Первый аргумент – индекс элемента после которого будет вставлен элемент x:

```
a = [1, 2]
a.insert(0, 5)
print(a)
```

```
[5, 1, 2]
```

```
a.insert(len(a), 9)
print(a)
```

```
[5, 1, 2, 9]
```

Методы списков - list.remove(x)

- **list.remove(x) – Удаляет первое вхождение элемента x из списка:**

```
a = [1, 2, 3]
a.remove(1)
print(a)
```

```
[2, 3]
```

Методы списков - list.pop([i])

- list.pop([i]) – Удаляет элемент из позиции i и возвращает его. Если использовать метод без аргумента, то будет удален последний элемент из списка:

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

```
3
5
[1, 2, 4]
```

Методы списков - list.clear()

- `list.clear() – Удаляет все элементы из списка. Эквивалентно del a[:]:`

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

a.clear()
print(a)
```

```
[1, 2, 3, 4, 5]
[]
```

Методы списков - list.index(x[, start[, end]])

- `list.index(x[, start[, end]])` – Возвращает индекс элемента:

```
a = [1, 2, 3, 4, 5]
a.index(4)
```

```
3
```

Методы списков - list.count(x)

- **list.count(x)- Возвращает количество вхождений элемента x в список:**

```
a=[1, 2, 2, 3, 3]
print(a.count(2))
```

```
2
```

Методы списков - list.sort

- `list.sort(key=None, reverse=False)` – Сортирует элементы в списке по возрастанию.
- Для сортировки в обратном порядке используйте флаг `reverse=True`.
- key (необязательный параметр) если указать ключ, то сортировка будет выполнена по функции этого ключа:

```
a = [1, 4, 2, 8, 1]
a.sort()
print(a)
```

```
[1, 1, 2, 4, 8]
```

Методы списков - list.reverse()

- **list.reverse() – Изменяет порядок расположения элементов в списке на обратный:**

```
a = [1, 3, 5, 7]
a.reverse()
print(a)
```

```
[7, 5, 3, 1]
```

Методы списков - list.copy()

- **list.copy()- Возвращает копию списка. Эквивалентно a[:]:**

```
a = [1, 7, 9]
b = a.copy()
print(a)
print(b)

b[0] = 8
print(a)
print(b)
```

```
[1, 7, 9]
[1, 7, 9]

[1, 7, 9]
[8, 7, 9]
```

Функция zip()

- Функция zip() в Python создает итератор, который объединяет элементы из нескольких источников данных.
- Функция zip() принимает итерируемый объект, например, список, кортеж, множество или словарь в качестве аргумента. Затем она генерирует список кортежей, которые содержат элементы из каждого объекта, переданного в функцию.

```
employee_numbers = [2, 9, 18, 28]
employee_names = ["Дима", "Марина", "Андрей", "Никита"]
zipped_values = zip(employee_names, employee_numbers)
zipped_list = list(zipped_values)
print(zipped_list)
```

```
[('Дима', 2), ('Марина', 9), ('Андрей', 18),
('Никита', 28)]
```

Функция zip с циклом for

```
employee_numbers = [2, 9, 18, 28]
employee_names = ["Дима", "Марина", "Андрей", "Никита"]

for name, number in zip(employee_names, employee_numbers):
  print(name, number)
```

```
Дима 2
Марина 9
Андрей 18
Никита 28
```

List Comprehensions

- List Comprehensions чаще всего на русский язык переводят как “абстракция списков” или “списковое включение”, является частью синтаксиса языка, которая предоставляет простой способ построения списков.
- Проще всего работу list comprehensions показать на примере. Допустим вам необходимо создать список целых чисел от 0 до n, где n предварительно задается.
- Классический способ решения данной задачи выглядел бы так:

```
n = int(input())
a=[]
for i in range(n):
  a.append(i)
  print(a)
```

```
5
[0]
[0, 1]
[0, 1, 2]
[0, 1, 2, 3]
[0, 1, 2, 3, 4]
```

List Comprehensions

- Использование list comprehensions позволяет сделать это значительно проще:

```
n = int(input())

a = [i for i in range(n)]
print(a)
```

```
5
[0, 1, 2, 3, 4]
```

List Comprehensions

- или вообще вот так, в случае если вам не нужно больше использовать n:

```
a = [i for i in range(int(input()))]
print(a)
```

```
5
[0, 1, 2, 3, 4]
```

List Comprehensions как обработчик списков

- В языке Python есть две мощные функции для работы с коллекциями: `map` и `filter`.
- Они позволяют использовать функциональный стиль программирования, не прибегая к помощи циклов, для работы с такими типами как list, tuple, set, dict и т.п. Списковое включение позволяет обойтись без этих функций.
- Приведем несколько примеров для того, чтобы понять о чем идет речь.

Задача со списком

- Пример с заменой функции map.
- Пусть у нас есть список и нужно получить на базе него новый, который содержит элементы первого, возведенные в квадрат.
- Решим эту задачу с использованием циклов:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = []
for i in a:
  b.append(i ** 2)
print('a = {}\nb = {}'.format(a, b))
```

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Задача со списком – map

- Та же задача, решенная с использованием map, будет выглядеть так:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = list(map(lambda x: x**2, a))
print('a = {}\nb = {}'.format(a, b))
```

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Задача со списком

- Через списковое включение эта задача будет решена так:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [i**2 for i in a]
print('a = {}\nb = {}'.format(a, b))
```

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [1, 4, 9, 16, 25, 36, 49]
```

Пример с заменой функции filter

- **Построим на базе существующего списка новый, состоящий только из четных чисел:**

```
a = [1, 2, 3, 4, 5, 6, 7]
b = []
for i in a:
    if i%2 == 0:
        b.append(i)
print('a = {}\nb = {}'.format(a, b))
```

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Пример с заменой функции filter

- Построим на базе существующего списка новый, состоящий только из четных чисел.
- Решим эту задачу с использованием filter:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = list(filter(lambda x: x % 2 == 0, a))
print('a = {}\nb = {}'.format(a, b))
```

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Пример с заменой функции filter

- Построим на базе существующего списка новый, состоящий только из четных чисел.
- Решение через списковое включение:

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [i for i in a if i % 2 == 0]
print('a = {}\nb = {}'.format(a, b))
```

```
a = [1, 2, 3, 4, 5, 6, 7]
b = [2, 4, 6]
```

Слайсы / Срезы

- Слайсы (срезы) являются очень мощной составляющей Python, которая позволяет быстро и лаконично решать задачи выборки элементов из списка.

- Слайс задается тройкой чисел, разделенных запятой: start:stop:step.
- Start –позиция с которой нужно начать выборку,
  stop – конечная позиция, step – шаг.
- При этом необходимо помнить, что выборка не включает элемент, определяемый stop.

Слайсы / Срезы

```
a = [i for i in range(10)]
a[:]
  # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
a[0:5] # Получить первые пять элементов списка
  # [0, 1, 2, 3, 4]
a[2:7] # Получить элементы с 3-го по 7-ой
  # [2, 3, 4, 5, 6]
a[::2] # Взять из списка элементы с шагом 2
  # [0, 2, 4, 6, 8]
a[1:8:2] # Взять из списка элементы со
2-го по 8-ой с шагом 2
  # [1, 3, 5, 7]
```

Слайсы / Срезы

- Слайсы можно сконструировать заранее, а потом уже использовать по мере необходимости.
- Это возможно сделать, в виду того, что слайс – это объект класса slice.
- Ниже приведен пример, демонстрирующий эту функциональность:

```
a = [i for i in range(10)]
a[:]
s = slice(0, 5, 1)
a[s]
  # [0, 1, 2, 3, 4]
s = slice(1, 8, 2)
a[s]
  # [1, 3, 5, 7]
```