

# АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

*и.о. доцент Муханова А.А.*



**Лекция 5-6. Массивы. Сортировка.  
Двумерные массивы**

## План лекции

Использование списков (массивов)

Заполнение списка значениями

Вывод списка на экран

Особенности копирования списков в Python

Двумерные списки

Типовые задачи обработки списков

Расчеты

Поиск и отбор нужных элементов

Работа с максимальными/минимальными элементами списка

Перестановки элементов

Проверка соответствия списка в целом некоторому условию

Задача.«Слияние (объединение) списков»

# Использование списков

## Общие вопросы

**Массив** — это пронумерованная последовательность величин одинакового типа, обозначаемая одним именем. Элементы **массива** располагаются в последовательных ячейках памяти, обозначаются именем **массива** и индексом. Каждое из значений, составляющих **массив**, называется его компонентой (или элементом **массива**). Пример: `a: array [1..15] of integer; //на языке Паскаль.`

**Список** — структура данных, предназначенная для хранения группы значений. Список позволяет использовать для всех, например 20 значений роста общее имя `r`. Для всех семи цветов радуги можно использовать список с именем `Raduga`.

Каждое отдельное значение списка называется «элемент списка».

**Элементы списка** при хранении списка в памяти имеют уникальные порядковые номера — «индексы» (нумерация непрерывная и начинается с 0). Чтобы обратиться в программе к значению некоторого элемента списка, надо указать имя списка и в квадратных скобках — индекс элемента.

**Например:**

```
print(Raduga[3])
print(Raduga[k])
if r[13] > r[12]:
```

...

Размер списка (количество элементов в нем) определяется с помощью функции `len()`:

```
n = len(a)
```

## Заполнение списка значением

Если на момент написания программы известны значения всех элементов списка, то эти значения могут быть записаны в список при его объявлении – указании его имени и перечня значений в квадратных скобках:

```
V = [20, 61, 2, 37, 4, 55, 36, 7, 18, 39]  
Raduga = ['Красный', 'Оранжевый', 'Желтый', 'Зеленый', 'Голубой', 'Синий', 'Фиолетовый']
```

В списке V – 10 элементов, являющихся числами, в списке Raduga – 7 элементов, являющихся строками. Обратим внимание на то, что, в отличие от большинства языков программирования, в Python совсем не обязательно, чтобы элементы списка были одного типа.

Значения элементов списка можно также ввести в программу с помощью инструкций присваивания:

```
a[0] = ...  
a[1] = ...
```

...

Можно использовать множественное присваивание:

```
a[0], a[1], ... = ...
```

хотя, конечно, первый способ удобнее.

## Заполнение списка значениям

Если известен закон, которому подчиняются значения элементов, то заполнить список можно с помощью так называемых «генераторов списка». Проиллюстрируем их использование на примере случая, когда значение каждого элемента равно его индексу. Пусть размер списка равен 10. Можем так оформить фрагмент:

`a[0] = 0`

`a[1] = 1`

...

`a[9] = 9`

Обозначив меняющийся при повторении индекс элемента списка так, как это традиционно делается в программировании, – именем *i*, можем оформить все в виде цикла `for`:

**`for i in range(10):`**

**`a[i] = i`**

**или короче:**

**`a = [i for i in range(10)]`**

**Последний вариант в общем случае при *n* элементах имеет вид:**

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

## Заполнение списка значением

Списки можно «складывать» с помощью знака «+»:

```
a = [20, 61, 2, 37]
```

```
b = [55, 36, 7, 18]
```

```
c = a + b
```

```
d = c + [100, 101]
```

и умножать на целое число:

```
a = [0] * 10 #a == [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
b = [""]; k = 8
```

```
b2 = b * k #b2 == ["", "", "", "", "", "", "", ""]
```

## Заполнение списка значением

В таких случаях нужно:

1) указать с помощью инструкции присваивания или ввести с помощью инструкции `input()` размер (количество элементов)

списка:

```
n = int(input())
```

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

```
print(a)
```

2) заполнить список «пустыми» (") значениями. Это можно сделать с использованием инструкции `for`:

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

или короче:

```
a = [' ' * n]
```

## Заполнение списка значениями

3) ввести в программу (с помощью инструкции `input()`) значения всех элементов и записать их в список. Это также можно сделать с использованием инструкции `for`:

```
for i in range(n):
```

```
    el = input('Введите значение очередного элемента списка ')
```

```
    a[i] = el
```

или короче:

```
for i in range(n):
```

```
    a[i] = input('Введите значение очередного элемента списка ')
```

Подсказка для ввода в инструкции `input()` может быть записана «символически»:

```
for i in range(n):
```

```
    print('a[', i + 1, '] = ', sep = ", end = ")
```

```
    a[i] = input()
```



## Заполнение списка значениями

Можно также подсказку сделать подробной:

```
for i in range(1, n + 1):
```

```
    a[i - 1] = input('Введите значение элемента списка номер',  
                    i)
```

**Внимание!** Нужно помнить, что инструкция `input()`

возвращает строку символов, поэтому для заполнения списка числами нужно использовать функции `int()` или `float()`.

Если в ходе выполнения программы необходимо добавлять в имеющийся список новые элементы, это можно сделать с помощью метода **append()**: **(добавляет элементы в конец списка)**

```
values.append(5) #Добавлен новый элемент списка values
#со значением 5
values.append(a) #Добавлен еще один элемент списка values
#со значением, равным значению переменной a
...
```

Можно также добавлять элементы со значениями, вводимыми с клавиатуры с помощью инструкции `input()`:

```
e1 = input('Введите значение следующего элемента списка ')
values.append(e1)
или короче:
values.append(input('Введите значение следующего элемента списка '))
```

В последнем случае при необходимости можно использовать инструкцию `for`:

```
for i in range(...)
e1 = input('Введите значение следующего элемента списка ')
values.append(e1)
```

Конечно, и здесь надо тоже при необходимости применять функции `int()` или `float()`.

В Python имеется возможность очень просто записать в список все слова предложения. Для этого используется метод **split()**. Например, в результате выполнения следующей инструкции:

```
L = fraza.split()
```

из слов предложения `fraza` будет сформирован список `L`.

Обратим внимание на то, что метод `split()` может быть применен, когда между всеми словами предложения имеются одни и те же разделители.

Когда решаются условные задачи учебного назначения, удобно заполнять список случайными целыми числами с помощью функции `randint()`. Например, заполнение списка `L_rand` двадцатью случайными числами, которые могут принимать значения 100,

101, ..., 200, проводится так:

```
from random import randint
```

```
for i in range(20):
```

```
    L_rand[i] = randint(100, 200)
```

или с помощью генератора списка:

```
from random import randint
```

```
L_rand = [randint(100, 200) for i in range(20)]
```

## Вывод списка на экран

Заполненный значениями список можно вывести на экран с использованием в инструкции `print()` имени списка:

**`print(a)`**

В результате выполнения инструкции на экран будут выведены значения всех элементов списка через запятую в квадратных скобках:

`[20, 61, 2, 37, 4, 55, 36, 7, 18, 39]`

Можно также вывести часть списка. Для этого используется «срез» списка.

Программа	Результат ее выполнения
<pre>a =[20, 61, 2, 37, 4, 55, 36, 7, 18, 39] print(a[1:5])</pre>	<code>[61, 2, 37, 4]</code>

Ясно, что так как есть повторяющиеся действия, то лучше использовать инструкцию цикла for:

```
for i in range(n):  
    print(a[i], end = ' ')
```

где n – количество элементов в списке. Это значение знать необязательно (особенно в случаях, когда размер списка меняется с помощью метода `append()`), – вместо него можно записать функцию `len()`, возвращающую количество элементов в списке:

```
for i in range(len(a)):  
    print(a[i], end = ' ')
```

К каждому отдельному элементу списка (при рассмотрении всех элементов) можно также обратиться без записи индексов:

```
for el in a:  
    print(el, end = ' ')
```

## Особенности копирования списков в Python

Если попытаться получить список `b` путем присваивания ему значений всех элементов списка `a`:

```
a = [...]
```

```
b = a
```

то две переменные `a` и `b` будут связаны с одним и тем же списком, поэтому при изменении одного списка будет изменяться и второй (ведь это фактически один и тот же список, к которому можно обращаться по двум разным именам). Эту особенность Python нужно учитывать при работе со списками.

Если в программе нужна именно копия списка (а не еще одна ссылка на него), можно использовать срез списка, формирующий его полную копию:

```
b = a[0:len(a)]
```

В результате `a` и `b` будут независимыми списками, и изменение одного из них не изменит второго.

# Двумерные списки

Часто в программах приходится хранить прямоугольные таблицы с данными. В программировании такие таблицы называют «двумерными массивами», или «матрицами». В языке программирования Python таблицу можно представить в виде списка, каждый элемент которого является, в свою очередь, списком. Такой список будем называть «двумерным», или «вложенным».

# Двумерные списки

Например, числовую таблицу можно представить в программе в виде двумерного списка:

$S_p = [[12, 7, 8], [21, 4, 55], [7, 22, 12], [54, 45, 31]]$

(списка из четырех элементов, каждый из которых является списком из трех элементов).

12	7	8
21	4	55
7	22	12
54	45	31



Чтобы использовать в программе какое-то число этого списка, надо после имени двумерного списка указать в квадратных скобках два индекса:

- 1) индекс соответствующего «внутреннего» списка ([12, 7, 8], [21, 4, 55], [7, 22, 12] или [54, 45, 31]); можно сказать, что это номер строки таблицы при нумерации с нуля;
- 2) индекс числа во «внутреннем» списке (номер столбца таблицы)

Например, `Sp = [[12, 7, 8], [21, 4, 55], [7, 22, 12], [54, 45, 31]]`

для числа 55 использование выглядит так: **`Sp[1, 2]`**.

Можно также применить такой вариант описания двумерного списка:

`per = [12, 7, 8]`

`vt = [21, 4, 55]`

`tr = [7, 22, 12]`

`ch = [54, 45, 31]`

`Sp = [per, vt, tr, ch]`

В этом случае для вывода, например, числа 55 необходимо использовать несколько необычную запись:

`print(Sp[1][2])`

запустить

выполнить пошагово

```
1 a = [[1, 2, 3], [4, 5, 6]]
2 print(a[0])
3 print(a[1])
4 b = a[0]
5 print(b)
6 print(a[0][2])
7 a[0][1] = 7
8 print(a)
9 print(b)
10 b[2] = 9
11 print(a[0])
12 print(b)
13
```

Здесь первая строка списка `a[0]` является списком из чисел `[1, 2, 3]`. То есть `a[0][0] == 1`, значение `a[0][1] == 2`, `a[0][2] == 3`, `a[1][0] == 4`, `a[1][1] == 5`, `a[1][2] == 6`.

Для обработки всего двумерного списка в программе применяется вложенная инструкция for:

```
for stroka in range(4)
```

```
for stol in range(3):
```

```
#Используется значение Sp[stroka, stol]
```

**или**

```
for stol in range(3):
```

```
for stroka in range(4):
```

```
#Используется (также) значение Sp[stroka, stol]
```

## Типовые задачи обработки списков

### Суммирование элементов списка

Для решения необходимо последовательно обратиться ко всем элементам списка и учесть их значения в уже рассчитанной ранее сумме (с использованием переменной-сумматора).

Соответствующие фрагменты программы решения задачи:

```
S = 0
```

```
for i in range(n):
```

```
    S = S + a[i]
```

**или**

```
S = 0
```

```
for el in a:
```

```
    S = S + el
```

Суммирование элементов списка – это очень распространенная операция, поэтому для этого в Python предусмотрена стандартная

**функция `sum()`:**

```
S = sum(a)
```

Если вместо имени списка использовать генератор списка с функцией `input()`:

```
S = sum([int(input('Введите следующий элемент списка ')) \
for i in range(n)])
```

то сумму всех элементов списка можно получить при вводе значений элементов. Правда, при этом сам список не будет сохранен в памяти.

С другой стороны, такая запись может быть использована для решения задачи, нахождения суммы всех чисел некоторого набора:

```
S = sum([int(input('Введите очередное число ')) for i in range(n)])
```

## Нахождение суммы элементов списка с заданными свойствами (удовлетворяющих некоторому условию)

Отличие задач данного типа от предыдущей задачи в том, что добавлять значение элемента списка к уже рассчитанной ранее сумме следует только тогда, когда элемент обладает заданными свойствами:

$S = 0$

for  $i$  in range( $n$ ):

#Если элемент обладает заданными свойствами

if  $\langle \text{условие} \rangle$ :

$S = S + a[i]$

где  $\langle \text{условие} \rangle$  – заданное условие для суммирования. Это условие может определяться значением элемента списка  $a[i]$  или/и его индексом  $i$ .

Можно также применить функцию sum:

```
S = sum([a[i] for i in range(n) if a[i] > 0 and i % 2 == 1])
```

В приведенном примере находится сумма положительных элементов списка с нечетным индексом. Для отбора таких чисел используется генератор списка с условием (видно, что условие записывается в конце генератора).

Если <условие> определяется только значением элемента (например, при решении задачи определения суммы четных чисел), то может быть применен и второй вариант рассмотрения всех элементов:

```
S = 0
```

```
for el in a:
```

```
if <условие>:
```

```
S = S + el
```

или с использованием функции sum()(для указанного примера):

```
S = sum([el for el in a if el % 2 == 0])
```

**Примечание.** Вычислив сумму всех элементов списка S, можно определить их среднее значение:

```
sred = S/n
```



## Нахождение количества элементов списка с заданными свойствами

Решение, аналогичное применяемому в большинстве других языков программирования:

```
kol = 0 #Начальное значение искомого количества
for i in range(n):
#Если элемент обладает заданными свойствами
if <условие>:
#Учитываем его в искомом количестве
kol = kol + 1
```

Решения, возможные только в программе на языке Python3:

1) kol = len([a[i] for i in range(n) if <условие>])

Пример: количество положительных элементов списка с нечетным индексом определяется так:

```
kol = len([a[i] for i in range(n) if a[i]> 0 and i % 2 == 1])
```

2) `kol = len([el for el in a if <условие>])`

Пример: количество четных элементов списка:

```
kol = len([el for el in a if el % 2 == 0])
```

В рассмотренной задаче `<условие>` определяется значением элемента списка `a[i]` или одновременно значениями `a[i]` и `i`. Количество элементов, зависящих только от значения индекса `i`, может быть найдено без использования инструкции цикла.

Задача нахождения количества элементов списка, равных некоторому значению `X`, может быть решена с использованием функции **`count()`**:

```
kol = a.count(X)
```

## Нахождение среднего арифметического значения элементов списка с заданными свойствами

Здесь тоже возможны «традиционный» вариант решения

```
S = 0
```

```
kol = 0
```

```
for i in range(n):
```

```
if <условие>: #Если элемент списка
```

```
#удовлетворяет заданному условию
```

```
S = S + a[i] #Учитываем его значение в сумме
```

```
kol = kol + 1 #и увеличиваем на 1 значение kol
```

```
#Подсчет и вывод результата
```

```
if kol > 0:
```

```
sred = S/kol
```

```
print('Среднее значение: ', '% 5.1f' % sred)
```

```
else:
```

```
print('Таких чисел в списке нет')
```

и варианты, характерные только для языка Python (с функцией `sum()` и генератором списка):

1)

```
S = sum([a[i] for i in range(n) if <условие>])
kol = len([a[i] for i in range(n) if <условие>])
if kol > 0:
    sred = S/kol
    print('Среднее значение: ', '% 5.1f '% sred) else:
    print('Таких чисел в списке нет')
```

2)

```
S = sum([el for el in a if <условие>])
kol = len([el for el in a if <условие>])
if kol > 0:
```

...

Пример, увеличить на 1 все четные элементы списка  
можно следующим образом:

```
for i in range(n):
```

```
    if a[i] % 2 == 0:
```

```
        a[i] = a[i] + 1
```

# Сортировка

- 0 s=[.....]
- 0 s.sort () # по возрастанию
- 0 s.sort (reverse=True) # по убыванию
- 0 s.sort(Key=f)
- 0 Key=len
- 0 Пример:  
Mynumber\_list = list(range(0,10))  
Print (mynumber\_list)  
Mynumber\_list.sort(reverse=True)  
Print (mynumber\_list)

# Max/Min

- 0 Print("Max number is:" + str (max(mynumber\_list)))
- 0 Print("Min number is:" + str (min(mynumber\_list)))
- 0 Print("Sum of list is:" + str (sum(mynumber\_list)))

**o Контрольные вопросы:**

- o** 1. Какие преимущества дает использование списка?
- o** 2. Как можно обратиться к отдельному элементу списка?
- o** 3. Какие возможны способы заполнения списка?
- o** 4. Чем удобно заполнение списка случайными числами?
- o** 5. В каких случаях применяется метод append
- o** 6. Как вывести на экран все элементы списка?



## Задачи для разработки программ

1. Заполнить список двадцатью символами «#».
2. Заполнить список из элементов случайными целыми числами из интервала от  $a$  до  $b$ .
3. Заполнить список двадцатью пятью первыми натуральными числами (1, 2, ..., 25), после чего добавить в него числа 100 и 200.
5. Дан список  $a$  из десяти элементов с числами, среди которых есть отрицательные. Записать все отрицательные числа во второй список. Разработать два варианта программы:
  - 1) без использования генератора списка;
  - 2) с использованием генератора списка.
6. Дан список. Получить новый список, в котором будут все элементы заданного списка, кроме элемента с индексом  $k$ . Разработать два варианта программы:
  - 1) без использования генератора списка;
  - 2) с использованием генератора списка.
7. Дан список, в котором есть числа 13. Получить новый список, в котором будут все элементы заданного списка, кроме числа 13. Разработать два варианта программы:
  - 1) без использования генератора списка;
  - 2) с использованием генератора списка.
8. Заполнить список десятью первыми числами последовательности Фибоначчи
9. Начиная поиск с числа 100, найти первые 10 простых чисел и записать их в список.  
Разработайте также программу, которая проверяет знание таблицы умножения. В ней на экран по одному выводятся 20 вопросов типа:  
Чему равно произведение 5 by 4?