

Изучение алгоритмизации и основ программирования  
на языке Python  
в курсе Информатика и ИКТ

# Одномерные массивы

Преподаватель: Гупалова А.В.  
Цветкова И.В.

**Массив** – это группа переменных одного типа, расположенных в памяти рядом (в соседних ячейках) и имеющих общее имя. Каждая ячейка в массиве имеет уникальный номер (индекс).

**В Питоне массивы - списки**



**A**

массив

0	1	2	3	4
5	10	15	20	25

НОМЕР  
элемента массива  
(ИНДЕКС)

A[0]

A[1]

ЗНАЧЕНИЕ  
элемента массива

A[4]

НОМЕР (ИНДЕКС)  
элемента массива: 2

A[2]

```
A = [1, 3, 4, 23, 5]
```

```
A = [1, 3] + [4, 23] + [5]  
[1, 3, 4, 23, 5]
```

```
A = [0] * 10
```

```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

```
A = list ( range (10) )
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# Генераторы списков

```
A = [ i for i in range(10) ]
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

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

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
from random import randint
```

```
A = [ randint(20, 100)
```

```
for x in range(10) ]
```

случайные  
числа

```
A = [ i for i in range(100)
```

```
if isPrime(i) ]
```

условие  
отбора



## Создание массива:

```
N = 5  
A = [0] * N
```

## Обработка в цикле:

```
i = 0  
while i < N:  
    # обработать A[i]  
    i += 1
```

## Цикл с переменной:

```
for i in range(N):  
    # обработать A[i]
```

## Ввод с клавиатуры:

```
for i in range(N):  
    print ( "A[" , i , "]" = "  
            sep = "" , end = "" )  
    A[i] = int( input() )
```

## Ввод без подсказок:

```
A = [ int(input()) for i in range(N) ]
```

```
data = input()      # "1 2 3 4 5"  
s = data.split()   # ["1", "2", "3", "4", "5"]  
A = [ int(x) for x in s ]  
                    # [1, 2, 3, 4, 5]
```

# Вывод массива на экран

Как список:

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

В строчку через пробел:

```
for i in range(N):  
    print ( A[i], end=" " ) 1 2 3 4 5
```

или так:

```
for x in A:  
    print ( x, end=" " ) 1 2 3 4 5
```

или так:

```
s = [ str(x) for x in A ]  
print ( " ".join( s ) )
```

записать как строку

соединить через  
пробел

или так:

```
print ( *A )
```



```
print ( 1, 2, 3, 4, 5 )
```



# Заполнение случайными числами

```
from random import randint
N = 10
A = [0]*N
for i in range(N):
    A[i] = randint(20, 100)
```

или так:

```
from random import randint
N = 10
A = [ randint(20, 100)
      for x in range(N) ]
```

случайные  
числа  
[20, 100]

# Перебор элементов

---

Общая схема (можно изменять  $A[i]$ ):

```
for i in range(N):  
    ... # сделать что-то с A[i]
```

```
for i in range(N):  
    A[i] += 1
```

Если не нужно изменять  $A[i]$ :

```
for x in A:  
    ... # сделать что-то с x
```

$x = A[0], A[1], \dots, A[N-1]$

```
for x in A:  
    print ( x )
```

# Подсчёт нужных элементов

---

**Задача.** В массиве записаны данные о росте баскетболистов. Сколько из них имеет рост больше 180 см, но меньше 190 см?



Как решать?

```
count = 0
for x in A:
    if 180 < x and x < 190:
        count += 1
```



# Перебор элементов

---

Сумма:

```
summa = 0
for x in A:
    if 180 < x and x < 190:
        summa += x
print ( summa )
```

или так:

```
print ( sum(A) )
```

# Перебор элементов

## Среднее арифметическое:

```
count = 0
summa = 0
for x in A:
    if 180 < x and x < 190:
        count += 1
        summa += x
print ( summa/count )
```

среднее  
арифметическое

## или так:

```
B = [ x for x in A
      if 180 < x and x < 190 ]
print ( sum(B)/len(B) )
```

отбираем нужные

# Максимальный элемент

```
M = A[0]
for i in range(1, N):
    if A[i] > M:
        M = A[i]
print ( M )
```



Если `range(N)` ?

Варианты в стиле Python:

```
M = A[0]
for x in A:
    if x > M:
        M = x
```



Как найти его номер?

```
M = max ( A )
```



# Максимальный элемент и его номер

```
M = A[0]; nMax = 0
for i in range(1, N):
    if A[i] > M:
        M = A[i]
        nMax = i
print( "A[" , nMax, "]=" , M, sep = " " )
```



Что можно улучшить?



По номеру элемента можно найти значение!

```
nMax = 0
for i in range(1, N):
    if A[i] > A[nMax]:
        nMax = i
print( "A[" , nMax, "]=" , A[nMax], sep = " " )
```

# Максимальный элемент и его номер

## Вариант в стиле Python:

```
M = max (A)
nMax = A . index (M)
print ( "A[" , nMax , "]" = " , M , sep = " " )
```

номер заданного  
элемента (первого из...)

# Вставка и удаление элементов

## Алгоритм удаления элемента:

1. определить номер удаляемого элемента -  $k$  (ввести с клавиатуры или найти из каких-то условий)
2. сдвинуть все элементы начиная с  $k+1$ -ого на 1 элемент влево
3. последнему элементу массива присвоить значение 0

При удалении элемента **размер массива не меняется!**  
Поэтому необходимо далее в программе указывать не до  $n-1$ , а до  $n-2$ .



дан массив A:

3 5 6 8 12 15 17 18 20 25



Элемент который нужно удалить

1.  $k=3$

2. 3 5 6 12 15 17 18 20 25 25

3. 3 5 6 12 15 17 18 20 25 0

{ВВОД массива и k}

for i in range(k,n-1):

    a[i]=a[i+1]

---

a[n-1] = 0

{ВЫВОД массива}



## Алгоритм вставки элемента: (после $k$ -ого)

1. первые  $k$  элементов остаются без изменений
2. все элементы, начиная с  $k$ -ого сдвигаются на 1 позицию назад
3. на место  $(k+1)$ -ого элемента записываем новый элемент.

Массив из  $n$  элементов, в который вставляется  $k$  элементов **необходимо определять как массив, имеющий размер  $n+k$** . Вставка перед элементом отличается только тем, что сдвигаются все элементы, начиная с  $k$ -ого и на место  $k$ -ого записываем новый



дан массив  $A$ :

0	1	2	3	4	5	6	7	8	9
3	5	6	8		15	17	18	20	25

позиция для добавления  
нового элемента

1.  $k=3$

2. 3 5 6 8 8 12 15 17 18 20 25

3. 3 5 6 8 100 12 15 17 18 20 25

## Пример:

Вставить 100 после элемента номер которого вводится с клавиатуры:

{ВВОД массива и k}

---

**for** i in range(n,k+2,-1):

    a[i+1]=a[i]

a[k+1] = 100;

{ВЫВОД массива}

# Алгоритм циклического сдвига на k позиций.

## I способ

1. определить сколько раз необходимо произвести одноэлементный сдвиг

$$k := k \% n;$$

2. k раз применить одноэлементный сдвиг  
*Алгоритм одноэлементного сдвига.*

- 1) Запомнить в дополнительной ячейке первый (или последний) элемент массива
- 2) Сдвинуть все элементы влево (вправо)
- 3) На последнее (первое) место записать тот, который запоминали.

# Сдвиг вправо и влево

```
n=int(input())
a=[5]*n
for i in range(n):
    a[i]=int(input())
print(a)
k=int(input())
k=k%n
for i in range(k):
    t=a[0]
    for j in range(n-1):
        a[j]=a[j+1]
    a[n-1]=t
print(a)
```

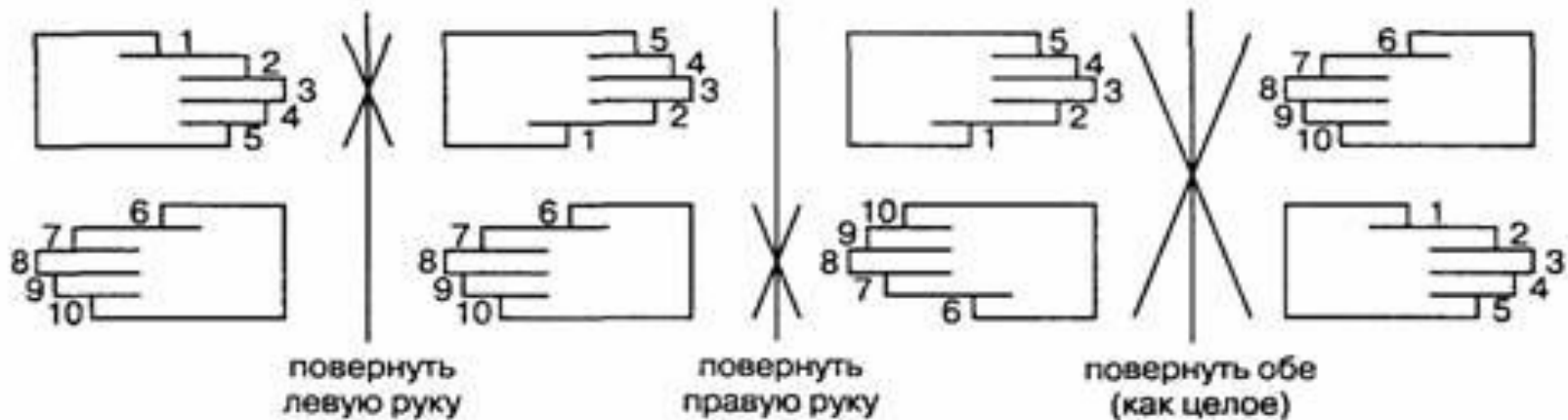
## II способ

1. Скопировать первые  $k$  элементов массива во временный массив
2. Сдвинуть оставшиеся  $n-k$  элементов влево на  $k$  позиций
3. Скопировать данные из временного массива обратно в основной массив на последние  $k$  позиций



# III способ

1. отобразить элементы массива(0, k-1)
2. отобразить элементы массива (k, n-1)
3. отобразить элементы массива (0, n-1)



**j**-сколько раз произвести обмен, **left** - левая граница отображения, **right** - правая граница отображения, **Dlina** - длина отображаемой части массива

$j=1$   $left=0$   $right=k-1$   $dlna=right-left+1$

(\*\*\*) **while**  $j \leq dlna // 2$  :

$temp=a[left]$

$a[left]=a[right]$

$a[right]=temp$

$left+=1$

$right-=1$

$j+=1$

$j=1$   $left=k$   $right=n-1$   $dlna=right-left+1$

(\*\*\*) {повторить цикл}

$j=1$   $left=0$   $right=n-1$   $dlna=right-left+1$

(\*\*\*) {повторить цикл}

# Сжатие массива.

Удаление каждого  $k$ -го элемента:

$i$  – индекс активного элемента

$l$  - индекс просматриваемого элемента

**kol** – количество элементов после всех удалений.

$i=k-1; l=k-1;$

**while**  $l \leq n-1$ :

**if**  $(l+1) \% k == 0$  :

$l+=1$

**if**  $l \leq n-1$  :

$a[i]=a[l];$

$i+=1$

$l+=1$

$kol=n-n // k$

# 1. Линейный поиск.

## Алгоритм.

Последовательно просматриваем массив и сравниваем значение очередного элемента с данным, если значение очередного элемента совпадет с  $X$ , то запоминаем его номер в переменной  $k$ .

**For**  $i$  in range( $n$ ):

**if**  $a[i] == x$  :

$k = i$ ;

Недостатки данной реализации алгоритма:

- находим только последнее вхождение элемента
- в любом случае производится  $n$  сравнений

Улучшим: будем прерывать поиск, как только найдем элемент:

```
while i <= n-1 and a[i] != x :  
    i+=1
```

В результате или найдем нужный элемент, или просмотрим весь массив.

**Недостаток данной реализации:**

в заголовке цикла сложное условие, что замедляет поиск.



## 2. Бинарный поиск

**Применяется для отсортированных массивов!!!!!!!.**

**Задача.** Дано  $X$  и массив  $A(n)$ , отсортированный по неубыванию Найти  $i$ , такой что  $a[i] = x$  или сообщить что данного элемента в массиве нет.

# Алгоритм

1. Является ли  $X$  средним элементом массива. Если да, то поиск завершен, иначе переходим к пункту 2.
2. Возможно 2 случая:
  - a)  $X$  меньше среднего, тогда так как  $A$  упорядочен, то из рассмотрения можно исключить все элементы массива, расположенные правее среднего и применить метод к левой половине массива.
  - b)  $X$  больше среднего. Значит, исключаем из рассмотрения левую половину массива и применяем метод к правой части.

$l = 0; r = n-1; \{ \text{на первом шаге рассматриваем весь массив} \}$

$f = \text{False}; \{ \text{признак того, что } X \text{ не найден} \}$

**while**  $l \leq r$  **and not**  $f$  :

$m = (l+r) // 2;$

**if**  $a[m] == x$  :

$f = \text{True} \{ \text{элемент найден! Поиск прекращаем} \}$

**elif**  $x < a[m]$  :

$r = m-1 \{ \text{отбрасываем правую часть} \}$

**else:**  $l = m + 1 \{ \text{отбрасываем левую часть} \}$

**Сортировка** - процесс упорядочения заданного множества объектов по заданному признаку.


Данные можно отсортировать:

■ **по возрастанию** - каждый следующий элемент больше предыдущего  $a[1] < a[2] < \dots < a[n]$

■ **по не убыванию** - каждый следующий элемент не меньше предыдущего  $a[1] \leq a[2] \leq \dots \leq a[n]$

■ **по убыванию** - каждый следующий элемент меньше предыдущего  $a[1] > a[2] > \dots > a[n]$

■ **по не возрастанию** - каждый следующий элемент не больше предыдущего  $a[1] \geq a[2] \geq \dots \geq a[n]$



**Степень эффективности метода** -  
количество сравнений и обменов,  
произведенных в процессе сортировки.

Наиболее часто встречаются 3 метода:  
сортировка выбором, обменом и  
вставкой.



# Сортировка методом выбора

Алгоритм (на примере сортировки по убыванию)

1. Выбрать минимальный (максимальный) элемент массива
2. Поменять его местами с последним (первым) элементом: теперь самый маленький (большой) на своем месте
3. Уменьшить количество рассматриваемых элементов на 1
4. Повторить действия 1-3 с оставшимися элементами (теми, которые еще не стоят на своих местах)

23 12 43 21 5 17

23 12 43 21 17 5

23 17 43 21 12 5

23 21 43 17 12 5

23 43 21 17 12 5

43 23 21 17 12 5



**For**  $i$  in range( $n-1, 0, -1$ ):

найти минимальный элемент из  $a[0], \dots, a[i]$

запомнить его индекс в переменной  $k$


если  $i \neq k$  то поменять местами  $a[i]$  и  $a[k]$

**end;**

```
for i in range (n-1,0,-1)
    k=0
    for j in range(1, i+1):
        if a[j]<a[k]:
            k=j
    if i!=k :
        temp=a[i]
        a[i]=a[k]
        a[k]=temp
```

Алгоритм: (на примере сортировки по убыванию)

- 1) Просматриваем массив парами  $a[0]$ ,  $a[1]$ ;  $a[2]$ ,  $a[3]$ ; ...
- 2) Если первый элемент пары меньше второго (пара расположена неправильно), то необходимо поменять их местами
- 3) Уменьшить количество рассматриваемых элементов на 1
- 4) Повторять действия 1-3 пока количество элементов в текущей части массива не уменьшится до двух.

- 
- 12 34 6 11 45
  - 34 12 6 11 45
  - 34 12 6 11 45
  - 34 12 11 6 45
  - 34 12 11 45 6



- For  $k$  in range( $n-1$ ):
- For  $i$  in range ( $n-k+1$ ):
- if  $a[i] > a[i+1]$  :
- $t = a[i]$
- $a[i] = a[i+1]$
- $a[i+1] = t$
-

# Улучшенный пузырьрек

P=True; {есть перестановка?}

K=1; {Номер просмотра}

While P

    P=false;

    For I in range(n-k+1)

        If  $X[i] > X[i+1]$

            A=X[i]

            X[i]=X[i+1]

            X[i+1]=A

            P=True;

        k=k+1;