In [3]:

```
import array as ar # импорт модуля для работы с масивами
arr1 = ar.array ('i', [100, 23, 189, 3456]) # создаем массив: не забываем указавать тип мас
#указываем элементы массива в []
#обратимся к элементам массива
arr1[0], arr1[3]
```

Out[3]:

```
(100, 3456)
```

In [37]:

```
# измерим размер массивов, для этого подключаем модуль sys
import sys
import array as ar
arr0 = ar.array ('i',[])
arr1 = ar.array ('i',[1,2,9,13,45,23,23,24,23,24])
arr2 = ar.array ('i',[1, 2,3, 4])
print('массив в памяти', arr0)
print('размер пустого массива', sys.getsizeof(arr0))
print('массив в памяти', arr1)
print ('размер массива в памяти', sys.getsizeof(arr1))
print('массив в памяти', arr2)
print ('размер массива в памяти', sys.getsizeof(arr2))
# измерим размер списков
sp0 = []
sp1 = [1,2, 9, 13,45,23,23,24,23,24]
sp2 = [1,2, 3, 4]
print('список в памяти sp0', sp0)
print ('размер ссылок в памяти списка sp0', sys.getsizeof(sp0))
# помним из лекции, что мы измерили только размер ссылок на объекты, которые хранятся в спи
# для понимания полноты масштаба трагедии воспользуемся следующей формулой: размер ссылок +
# выполните программную реализацию расчета размера списка не зависимо от количества элемент
r = sys.getsizeof(sp0)
print ('размер списка sp0 в памяти', r)
print('список в памяти sp1', sp1)
print ('размер ссылок в памяти списка sp1', sys.getsizeof(sp1))
r = sys.getsizeof(sp1) + sys.getsizeof(sp1[0]) + sys.getsizeof(sp1[1])+ sys.getsizeof(sp1[2
print ('размер списка sp1 в памяти', r)
print('список в памяти sp2', sp2)
print ('размер ссылок в памяти списка sp2', sys.getsizeof(sp2))
r = sys.getsizeof(sp2) + sys.getsizeof(sp2[0]) + sys.getsizeof(sp2[1])+ sys.getsizeof(sp2[2
print ('размер списка sp2 в памяти', r)
```

```
массив в памяти array('i')
размер пустого массива 64
массив в памяти array('i', [1, 2, 9, 13, 45, 23, 23, 24, 23, 24])
размер массива в памяти 104
массив в памяти array('i', [1, 2, 3, 4])
размер массива в памяти 80
список в памяти sp0 []
размер ссылок в памяти списка sp0 64
список в памяти sp1 [1, 2, 9, 13, 45, 23, 23, 24, 23, 24]
размер ссылок в памяти списка sp1 144
список в памяти sp2 [1, 2, 3, 4]
размер ссылок в памяти списка sp2 96
реальные размеры списков в памяти
размер списка sp0 в памяти 64
размер списка sp1 в памяти 424
размер списка sp2 в памяти 208
```

In [42]:

```
#создание двумерного массива с использованием библиотеки numpy
#Функция array() трансформирует вложенные последовательности в многомерные массивы.
#Тип элементов массива зависит от типа элементов исходной последовательности (но можно и не
import numpy as num
arrdoub = num.array ([[1,2,2,3,4], [1,2,2,3,4]])
print (arrdoub)
```

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

In [46]:

```
#Можно также переопределить тип массива в момент создания:
import numpy as num
b = num.array([[1.5, 2, 3], [4, 5, 6]], dtype=num.complex)
print(b)
```

```
[[1.5+0.j 2. +0.j 3. +0.j]
 [4. +0.j 5. +0.j 6. +0.j]]
```

Функция array() не единственная функция для создания массивов. Обычно элементы массива вначале неизвестны, а массив, в котором они будут храниться, уже нужен. Поэтому имеется несколько функций для того, чтобы создавать массивы с каким-то исходным содержимым (по умолчанию тип создаваемого массива — float64).

Функция zeros() создает массив из нулей, а функция ones() — массив из единиц. Обе функции принимают кортеж с размерами, и аргумент dtype: Функция eye() создаёт единичную матрицу (двумерный массив) Функция empty() создает массив без его заполнения. Исходное содержимое случайно и зависит от состояния памяти на момент создания массива (то есть от того мусора, что в ней хранится):

In [56]:

```
import numpy as num
print ('создадим двумерный массив 3X5')
print(num.zeros((3, 5)))
print ('создадим трехмерный массив 2X2X2')
print (num.ones((2, 2, 2)))
print ('создадим единичную матрицу, она же двумерный массив, например 5X5')
print(num.eye(5))
print ('создадим пустую матрицу 5X5')
print(num.empty((5,5))) # помним, что в ней может оказаться мусор из памяти, так как она не
```

```
создадим двумерный массив 3X5
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
создадим трехмерный массив 2X2X2
[[[1. 1.]
  [1. 1.]]

 [[1. 1.]
  [1. 1.]]]
создадим единичную матрицу, она же двумерный массив, например 5X5
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
создадим пустую матрицу 5X5
[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]
```

Для создания последовательностей чисел, в NumPy имеется функция arange(), аналогичная встроенной в Python range(), только вместо списков она возвращает массивы, и принимает не только целые значения.

In [58]:

```
import numpy as num
k = num.arange(10, 30, 5)
print (k)
l = num.arange(0, 1, 0.1)
print (l)
#как видите, arange позволяет создавать не только целочисленные, но и вещественные массивы
```

```
[10 15 20 25]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9]
```

Вообще, при использовании arange() с аргументами типа float, сложно быть уверенным в том, сколько элементов будет получено (из-за ограничения точности чисел с плавающей запятой). Поэтому, в таких случаях обычно лучше использовать функцию linspace(), которая вместо шага в качестве одного из аргументов принимает число, равное количеству нужных элементов.

In [59]:

```
import numpy as num
s = num.linspace(0, 3, 9)  # 9 чисел от 0 до 2 включительно
print (s)
```

```
[0.    0.375 0.75  1.125 1.5   1.875 2.25  2.625 3.   ]
```

Для понимания длительности процессов целесообразно выполнять расчет временных затрат. Для этого возможно использование библиотеки timeit.

In [86]:

```
import array as ar
import timeit
ar1 = ar.array ('i', [1,2,2,3,4])
ar2 = ar.array ('i', [1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4
sp1 = [1,2,2,3,4]
sp2 = [1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,2,3,4,1,2,
a = timeit.default_timer()
len(ar1)
print('время на измерение длины массива из 5 элементов', timeit.default_timer()-a)
a = timeit.default_timer()
len(ar2)
print('время на измерение длины массива из 45 элементов', timeit.default_timer()-a)
a = timeit.default_timer()
len(sp1)
print('время на измерение длины списка из 5 элементов', timeit.default_timer()-a)
a = timeit.default_timer()
len(sp2)
print('время на измерение длины списка из 45 элементов', timeit.default_timer()-a)

# выполните расчет временных затрат для  5 функций работы со списками и работы с массивами,
# и выскажите обоснованное предположение полученной разницы во времени исполнения
```

```
время на измерение длины массива из 5 элементов 7.190899850684218e-05
время на измерение длины массива из 45 элементов 7.355299931077752e-05
время на измерение длины списка из 5 элементов 6.903199937369209e-05
время на измерение длины списка из 45 элементов 7.067699880280998e-05
```

```
Методы для работы с массивами, которые могут быть вам полезны.
array.typecode - TypeCode символ, использованный при создании массива.
array.itemsize - размер в байтах одного элемента в массиве.
array.append(x) - добавление элемента в конец массива.
array.buffer_info() - кортеж (ячейка памяти, длина). Полезно для низкоуровневых операций.
array.byteswap() - изменить порядок следования байтов в каждом элементе массива. Полезно
при чтении данных из файла, написанного на машине с другим порядком байтов.
array.count(x) - возвращает количество вхождений x в массив.
array.extend(iter) - добавление элементов из объекта в массив.
array.frombytes(b) - делает массив array из массива байт. Количество байт должно быть
кратно размеру одного элемента в массиве.
array.fromfile(F, N) - читает N элементов из файла и добавляет их в конец массива. Файл
должен быть открыт на бинарное чтение. Если доступно меньше N элементов, генерируется
исключение EOFError , но элементы, которые были доступны, добавляются в массив.
array.fromlist(список) - добавление элементов из списка.
array.index(x) - номер первого вхождения x в массив.
array.insert(n, x) - включить новый пункт со значением x в массиве перед номером n.
Отрицательные значения рассматриваются относительно конца массива.
array.pop(i) - удаляет i-ый элемент из массива и возвращает его. По умолчанию удаляется
последний элемент.
```

```
array.remove(x) - удалить первое вхождение x из массива.
array.reverse() - обратный порядок элементов в массиве.
array.tobytes() - преобразование к байтам.
array.tofile(f) - запись массива в открытый файл.
array.tolist() - преобразование массива в список.
```

В качестве заданий используются задания по работе со списками из прошлого семестра, требуется выполнить их программную реализацию с применением массивов. Выполните расчет времени исполнения и количества памяти при списочной реализации и при реализации с помощью массивов (на основе лекции). В отчете предложите обоснованный подход к программной реализациии с учетом предлагаемой Вами структуре данных. Для заданий 12,13, 14, 15, 16 выполниет обоснование используемой структуры данных, проведите сравнительный анализ двух програмных реализаций.

Задача 1. В массиве, содержащем положительные и отрицательные целые числа, вычислить сумму четных положительных элементов.

Задача 2:
В массиве найти максимальный элемент с четным индексом.
Другая формулировка задачи: среди элементов массива с четными индексами, найти тот, который имеет максимальное значение.

Задача 3:
Выполните обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Все элементы имеют целый тип. Дано целое число H. Определите, какие столбцы имеют хотя бы одно такое число, а какие не имеют.

Задача 4:
Найти в массиве те элементы, значение которых меньше среднего арифметического, взятого от всех элементов массива.
Задача 5:
В одномерном массиве целых чисел определить два наименьших элемента. Они могут быть как равны между собой (оба являться минимальными), так и различаться.
Задача 6:
Сжать массив, удалив из него все элементы, величина которых находится в интервале [a, b]. Освободившиеся в конце массива элементы заполнить нулями.
Задача 7:
Вычислить сумму модулей элементов массива, расположенных после первого отрицательного элемента.

Например, в массиве [5, 3, -1, 8, 0, -6, 1] первый отрицательный элемент является третьим по счету, а сумма модулей стоящих после него элементов массива будет составлять 8 + 0 + 6 + 1 = 15.

Задача 8:
Пользователь вводит упорядоченный список/массив книг (заданной длины по алфавиту). Добавить новую книгу, сохранив упорядоченность списка по алфавиту

Задача 9:
Дан список/массив целых чисел. Упорядочьте по возрастанию только:
а) положительные числа;
б) элементы с четными порядковыми номерами в списке.

Задача 10:
Даны два списка/массива. Определите, совпадают ли множества их элементов.

Задача 11:
Дан список/массив. После каждого элемента добавьте предшествующую ему часть списка.

Задача 12:

Пусть элементы списка/массива хранят символы предложения. Замените каждое вхождение слова "itmathrepetitor" на "silence".

Задача 13*:
Дан текстовый файл. Создайте двусвязный список (или массив), каждый элемент которого содержит количество символов в соответствующей строке текста.

Задача 14*:
Создайте двусвязный список (или массив) групп факультета. Каждая группа представляет собой односвязный список (или массив) студентов.

Задача 15*:
Дан список студентов. Элемент списка содержит фамилию, имя, отчество, год рождения, курс, номер группы, оценки по пяти предметам. Упорядочите студентов по курсу, причем студенты одного курса располагались в алфавитном порядке.
Найдите средний балл каждой группы по каждому предмету. Определите самого старшего студента и самого младшего студентов.
Для каждой группы найдите лучшего с точки зрения успеваемости студента. Выберете структуру данных для программной реализации.

Задача 16*:
Выполнить программную реализацию калькулятора матриц.
Задача 17:
Сдвинуть элементы массива в указанном направлении (влево или вправо) и на указанное число шагов. Освободившиеся ячейки заполнить нулями. Выводить массив после каждого шага.
Задача 18:
Заполнить массив случайными положительными и отрицательными целыми числами. Вывести его на экран. Удалить из массива все отрицательные элементы и снова вывести.
Задача 19:
Заполнить вводом с клавиатуры численный массив за исключением последнего элемента, вывести его на экран. Запросить еще одно значение и его позицию в в массиве. Вставить указанное число в заданную позицию, подвинув элементы после него.
Задача 20:
Заполнить один массив случайными числами, другой - введенными с клавиатуры числами, в ячейки третьего записать суммы соответствующих ячеек первых двух. Вывести содержимое массивов на экран.
Задача 21:
Сгенерировать 2000 случайных целых чисел в диапазоне от -5 до 4, записать их в ячейки массива. Посчитать сколько среди них положительных, отрицательных и нулевых значений. Вывести на экран элементы массива и посчитанные количества. Рассмотрите возможность использования numpy.set_printoptions для полноценного вывода занчений массива на экран.