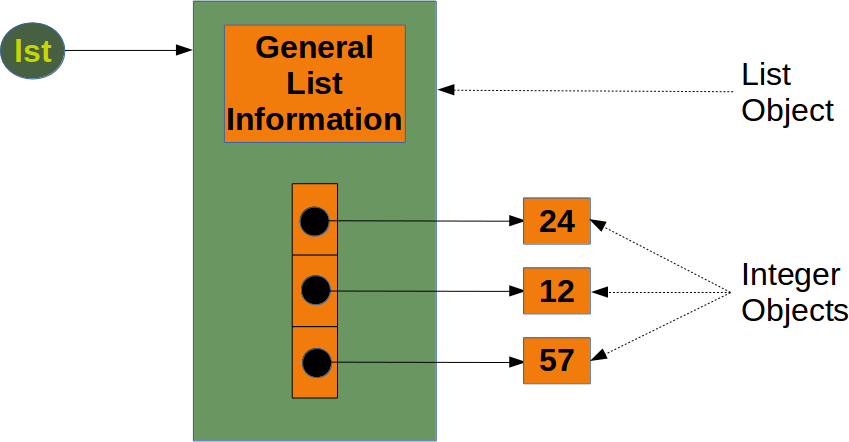
**Memory Consumption: ndarray and list**

The main benefits of using numpy arrays should be smaller memory consumption and better runtime behaviour. We want to look at the memory usage of numpy arrays in this subchapter of our turorial and compare it to the memory consumption of Python lists.



To calculate the memory consumption of the list from the above picture, we will use the function getsizeof from the module sys.

from sys import getsizeof as size

lst = [24, 12, 57]

size\_of\_list\_object = size(lst) # only green box

size\_of\_elements = len(lst) \* size(lst[0]) # 24, 12, 57

total\_list\_size = size\_of\_list\_object + size\_of\_elements

print("Size without the size of the elements: ", size\_of\_list\_object)

print("Size of all the elements: ", size\_of\_elements)

print("Total size of list, including elements: ", total\_list\_size)

Size without the size of the elements: 88

Size of all the elements: 84

Total size of list, including elements: 172

The size of a Python list consists of the general list information, the size needed for the references to the elements and the size of all the elements of the list. If we apply sys.getsizeof to a list, we get only the size without the size of the elements. In the previous example, we made the assumption that all the integer elements of our list have the same size. Of course, this is not valid in general, because memory consumption will be higher for larger integers.

We will check now, how the memory usage changes, if we add another integer element to the list. We also look at an empty list:

lst = [24, 12, 57, 42]

size\_of\_list\_object = size(lst) # only green box

size\_of\_elements = len(lst) \* size(lst[0]) # 24, 12, 57, 42

total\_list\_size = size\_of\_list\_object + size\_of\_elements

print("Size without the size of the elements: ", size\_of\_list\_object)

print("Size of all the elements: ", size\_of\_elements)

print("Total size of list, including elements: ", total\_list\_size)

lst = []

print("Emtpy list size: ", size(lst))

Size without the size of the elements: 96

Size of all the elements: 112

Total size of list, including elements: 208

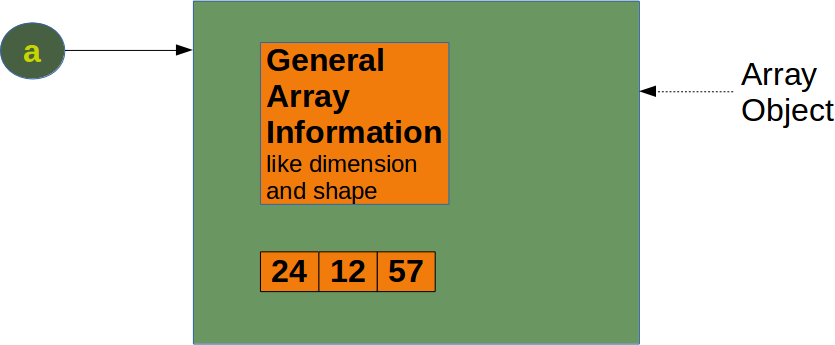
Emtpy list size: 64

We can conclude from this that for every new element, we need another eight bytes for the reference to the new object. The new integer object itself consumes 28 bytes. The size of a list "lst" without the size of the elements can be calculated with:

64 + 8 \* len(lst)

To get the complete size of an arbitrary list of integers, we have to add the sum of all the sizes of the integers.

We will examine now the memory consumption of a numpy.array. To this purpose, we will have a look at the implementation in the following picture:



We will create the numpy array of the previous diagram and calculate the memory usage:

a = np.array([24, 12, 57])

print(size(a))

120

We get the memory usage for the general array information by creating an empty array:

e = np.array([])

print(size(e))

96

We can see that the difference between the empty array "e" and the array "a" with three integers consists in 24 Bytes. This means that an arbitrary integer array of length "n" in numpy needs

96 + n \* 8 Bytes

whereas a list of integers needs, as we have seen before

64 + 8 *len(lst) + len(lst)*28

This is a minimum estimation, as Python integers can use more than 28 bytes.

When we define a Numpy array, numpy automatically chooses a fixed integer size. In our example "int64". We can determine the size of the integers, when we define an array. Needless to say, this changes the memory requirement:

a = np.array([24, 12, 57], np.int8)

print(size(a) - 96)

a = np.array([24, 12, 57], np.int16)

print(size(a) - 96)

a = np.array([24, 12, 57], np.int32)

print(size(a) - 96)

a = np.array([24, 12, 57], np.int64)

print(size(a) - 96)

3

6

12

24

### Time Comparison between Python Lists and Numpy Arrays

One of the main advantages of NumPy is its advantage in time compared to standard Python. Let's look at the following functions:

import time

size\_of\_vec = 1000

def pure\_python\_version():

t1 = time.time()

X = range(size\_of\_vec)

Y = range(size\_of\_vec)

Z = [X[i] + Y[i] for i in range(len(X)) ]

return time.time() - t1

def numpy\_version():

t1 = time.time()

X = np.arange(size\_of\_vec)

Y = np.arange(size\_of\_vec)

Z = X + Y

return time.time() - t1

Let's call these functions and see the time consumption:

t1 = pure\_python\_version()

t2 = numpy\_version()

print(t1, t2)

print("Numpy is in this example " + str(t1/t2) + " faster!")

0.00025272369384765625 7.367134094238281e-05

Numpy is in this example 3.43042071197411 faster!

Attention: Whereas slicings on lists and tuples create new objects, a slicing operation on an array creates a view on the original array. So we get another possibility to access the array, or better a part of the array. From this follows that if we modify a view, the original array will be modified as well.

## Data Type Objects, dtype

The data type object 'dtype' is an instance of numpy.dtype class. It can be created with numpy.dtype. With the aid of dtype we are capable to create "Structured Arrays", - also known as "Record Arrays". The structured arrays provide us with the ability to have different data types per column. It has similarity to the structure of excel or csv documents.

Structured Array: Structured arrays are ndarrays whose datatype is a composition of simpler datatypes organized as a sequence of named fields. Whereas, ndarrays are homogeneous data objects, i.e. all elements of an array have to be of the same data type. The data type dytpe on the other hand allows as to define separate data types for each column.

**Matrices vs. Two-Dimensional Arrays:**

There are "real" matrices in Numpy. They are a subset of the two-dimensional arrays. We can turn a two-dimensional array into a matrix by applying the "mat" function. The main difference shows, if you multiply two two-dimensional arrays or two matrices. We get real matrix multiplication by multiplying two matrices, but the two-dimensional arrays will be only multiplied component-wise.