PYTHON LISTS VS NUMPY ARRAYS

Numpy array

What is a Numpy array?

NumPy is the fundamental package for scientific computing in Python. Numpy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences. Numpy is not another programming language but a Python extension module. It provides fast and efficient operations on arrays of homogeneous data.

Some important points about Numpy arrays:

- We can create an N-dimensional array in Python using Numpy.array().
- The array is by default Homogeneous, which means data inside an array must be of the same Datatype. (Note You can also create a structured array in Python).
- Element-wise operation is possible.
- Numpy array has various functions, methods, and variables, to ease our task of matrix computation.
- Elements of an array are stored contiguously in memory. For example, all rows of a two-dimensioned array must have the same number of columns. A three-dimensional array must have the same number of rows and columns on each card.

Numpy array

Representation of Numpy array

Single Dimensional Numpy Array

```
import numpy as np

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

Multi-dimensional Numpy Array

```
import numpy as np

a = np.array([(1, 2, 3), (4, 5, 6)])
print(a)
```

Python list

What is python list?

A Python list is a collection that is ordered and changeable. In Python, lists are written with square brackets.

Some important points about Python Lists:

- The list can be homogeneous or heterogeneous.
- Element-wise operation is not possible on the list.
- Python list is by default 1-dimensional. But we can create an N-Dimensional list. But then too it will be 1 D list storing another 1D list
- Elements of a list need not be contiguous in memory.

Representation of Python List

Here we are creating Python List using [ ]

```
Var = ["Geeks", "for", "Geeks"]
print(Var)
```

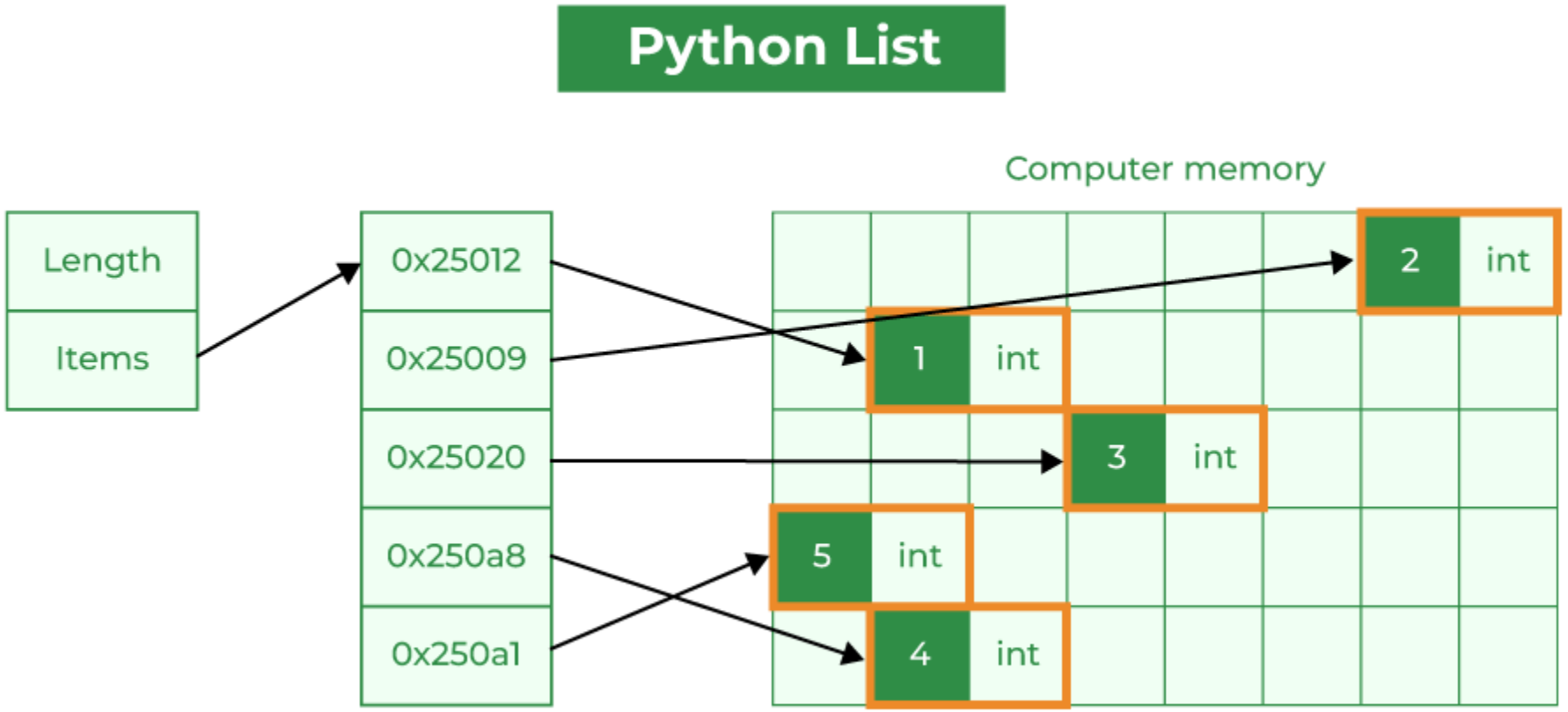
Comparison

Python List

1. Element Overhead: Lists in Python store additional information about each element, such as its type and reference count. This overhead can be significant when dealing with a large number of elements.
2. Datatype: Lists can hold different data types, but this can decrease memory efficiency and slow numerical operations.
3. Memory Fragmentation: Lists may not store elements in contiguous memory locations, causing memory fragmentation and inefficiency.
4. Performance: Lists are not optimized for numerical computations and may have slower mathematical operations due to Python's interpretation overhead. They are generally used as general-purpose data structures.
5. Functionality: Lists can store any data type, but lack specialized NumPy functions for numerical operations.

Comparison

Python List



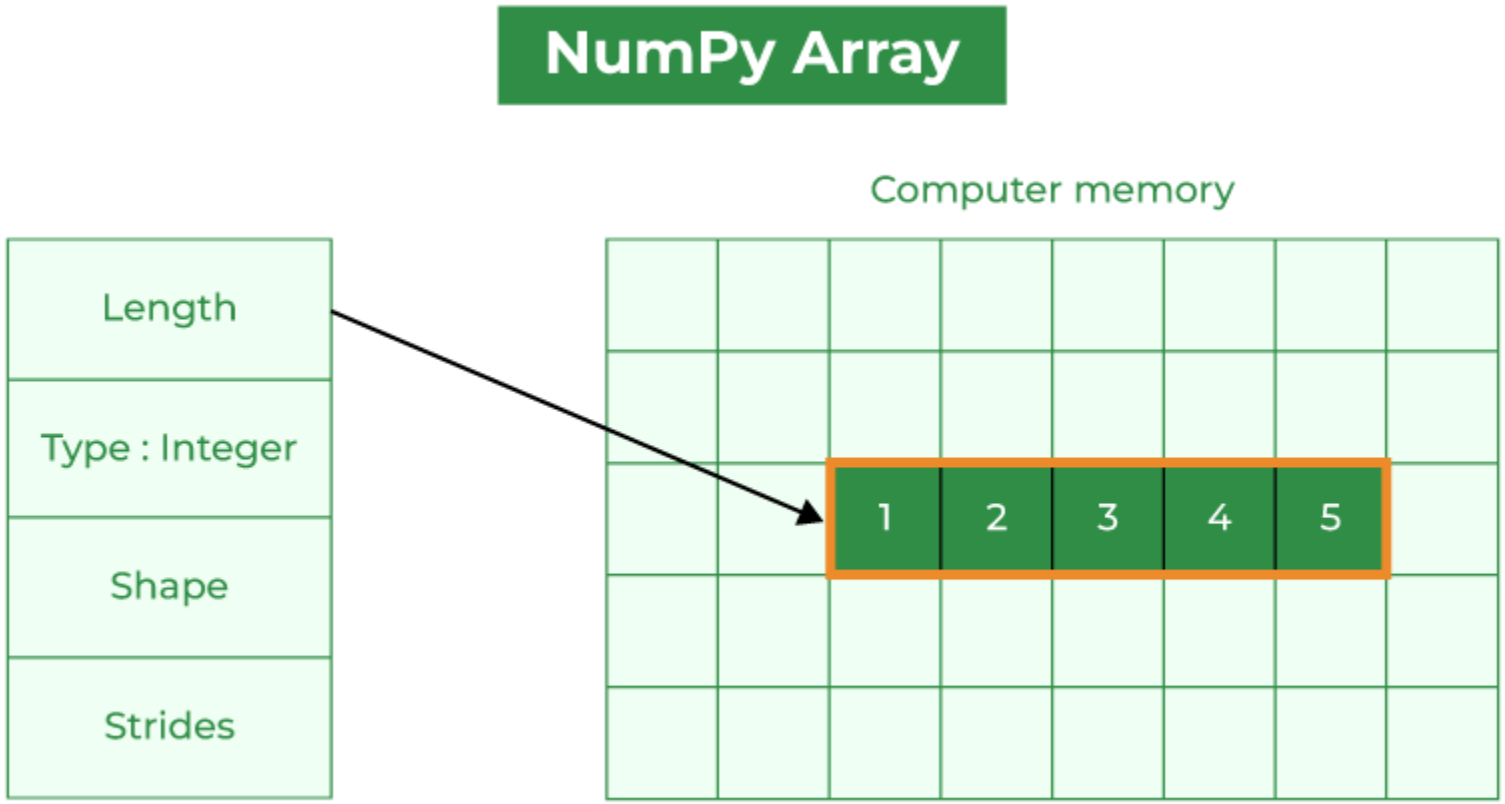
Comparison

Numpy arrays

1. Homogeneous Data: NumPy arrays store elements of the same data type, making them more compact and memory-efficient than lists.
2. Fixed Data Type: NumPy arrays have a fixed data type, reducing memory overhead by eliminating the need to store type information for each element.
3. Contiguous Memory: NumPy arrays store elements in adjacent memory locations, reducing fragmentation and allowing for efficient access.
4. Array Metadata: NumPy arrays have extra metadata like shape, strides, and data type. However, this overhead is usually smaller than the per-element overhead in lists.
5. Performance: NumPy arrays are optimized for numerical computations, with efficient element-wise operations and mathematical functions. These operations are implemented in C, resulting in faster performance than equivalent operations on lists.

Comparison

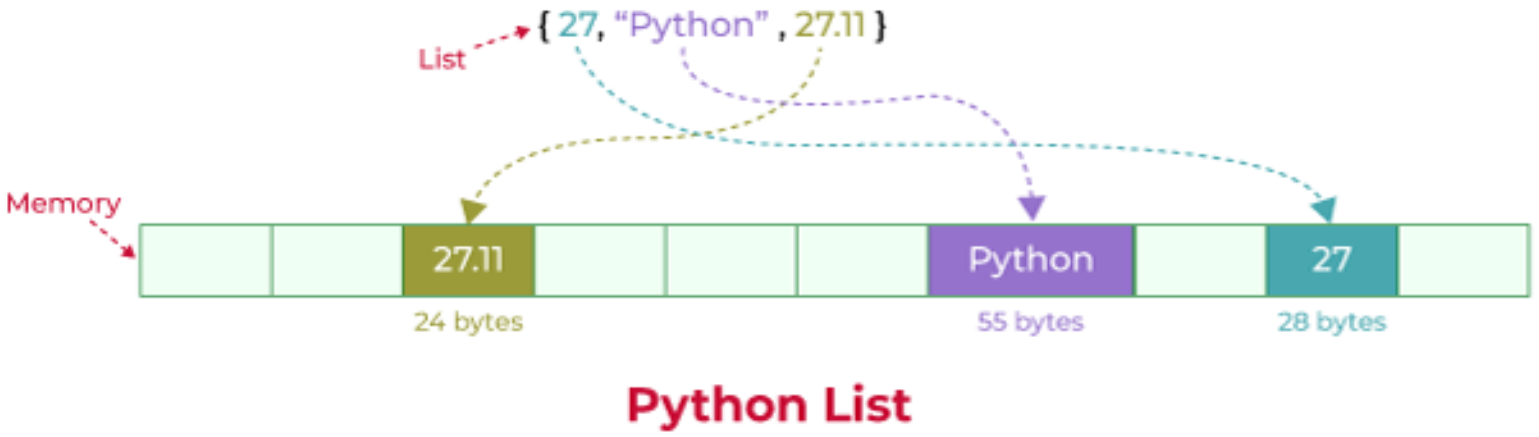
Numpy arrays



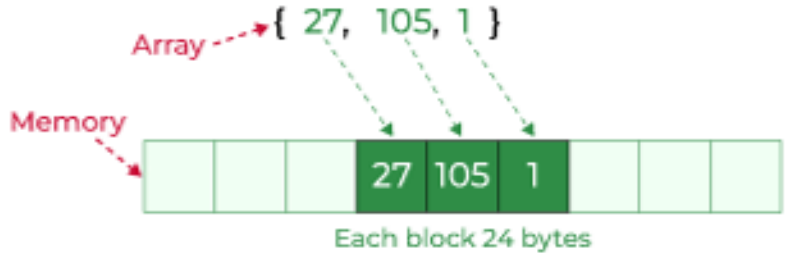
Comparison

Memory consumption between Numpy array and lists

In Python, a list is a built-in data structure that can hold elements of varying data types. However, the flexibility of lists comes at the cost of memory efficiency.
Python's NumPy library supports optimized numerical array and matrix operations.



Python List



NumPy Array

- Memory size in bytes are in Mac OS X.

Comparison

Memory consumption between Numpy array and lists

In this example, a Python list and a Numpy array of size 1000 will be created. The size of each element and then the whole size of both containers will be calculated and a comparison will be done in terms of memory consumption.

```
# importing numpy package
import numpy as np

# importing system module
import sys

# declaring a list of 1000 elements
S= range(1000)

# printing size of each element of the list
print("Size of each element of list in bytes: ",sys.getsizeof(S))
```

Comparison

```
# printing size of the whole list
print("Size of the whole list in bytes: ",sys.getsizeof(S)*len(S))

# declaring a Numpy array of 1000 elements
D= np.arange(1000)

# printing size of each element of the Numpy array
print("Size of each element of the Numpy array in bytes: ",D.itemsize)

# printing size of the whole Numpy array
print("Size of the whole Numpy array in bytes: ",D.size*D.itemsize)
```

Comparison

Time comparison between Numpy array and Python lists

In this example, here two Python lists and two Numpy arrays will be created and each container has 1000000 elements. Multiplication of elements in both the lists and Numpy arrays respectively will be carried out and the difference in time needed for the execution for both the containers will be analyzed to determine which one takes less time to perform the operation.

```
# importing required packages
import numpy
import time

# size of arrays and lists
size = 1000000

# declaring lists
list1 = range(size)
list2 = range(size)
```

Comparison

```
# declaring arrays
array1 = numpy.arange(size)
array2 = numpy.arange(size)

# capturing time before the multiplication of Python lists
initialTime = time.time()

# multiplying elements of both the lists and stored in another list
resultantList = [(a * b) for a, b in zip(list1, list2)]

# calculating execution time
print("Time taken by Lists to perform multiplication:",
            (time.time() - initialTime),
            "seconds")

# capturing time before the multiplication of Numpy arrays
initialTime = time.time()
```

Comparison

```
# multiplying elements of both the Numpy arrays and stored in another Numpy array
resultantArray = array1 * array2

# calculating execution time
print("Time taken by NumPy Arrays to perform multiplication:", (time.time() - initialTime), "seconds")
```

Comparison

Effect of operations on Numpy array and Python Lists

In this example, the incapability of the Python list to carry out a basic operation is demonstrated. A Python list and a Numpy array having the same elements will be declared and an integer will be added to increment each element of the container by that integer value without looping statements. The effect of this operation on the Numpy array and Python list will be analyzed.

```
# importing Numpy package
import numpy as np

# declaring a list
ls =[1, 2, 3]

# converting the list into a Numpy array
arr = np.array(ls)

try:
        # adding 4 to each element of list
        ls = ls + 4
```

Comparison

```
except(TypeError):
        print("Lists don't support list + int")

# now on array
try:
        # adding 4 to each element of Numpy array
        arr = arr + 4

        # printing the Numpy array
        print("Modified Numpy array: ",arr)

except(TypeError):
        print("Numpy arrays don't support list + int")
```

Conclusion

Advantages of using Numpy Arrays Over Python Lists:

- ✓ Consumes less memory.
- ✓ Fast as compared to the python List.
- ✓ Convenient to use.

Reference

- ✓ Numpy https://numpy.org/
- ✓ 나무위키 https://namu.wiki/w/NumPy
- ✓ 위키독스 https://wikidocs.net/193543
- ✓ Geeksforgeeks https://www.geeksforgeeks.org/
- ✓ 자료 분석을 위한 파이썬 https://compmath.korea.ac.kr/appmath/NumpyBasics.html
- ✓ 텐서플로우 블로그 https://tensorflow.blog/%ED%95%B8%EC%A6%88%EC%98%A8-%EB%A8%B8%EC%8B%A0%EB%9F%AC%EB%8B%9D-1%EC%9E%A5-2%EC%9E%A5/%EB%84%98%ED%8C%8C%EC%9D%B4-%ED%8A%9C%ED%86%A0%EB%A6%AC%EC%96%BC/
- ✓ 데이터 사이언스 스쿨 https://datascienceschool.net/01%20python/03.01%20%EB%84%98%ED%8C%8C%EC%9D%B4%20%EB%B0%B0%EC%97%B4.html
- ✓ kim taewan https://taewan.kim/post/numpy_cheat_sheet/
- ✓ 유투브 https://www.youtube.com/playlist?list=PLBHVuYlKEkULZLnKLzRq1CnNBOBlBTkqp