NumPy Arrays advantages

1.Size - Numpy data structures take up less space

2.Performance - they have a need for speed and are faster than lists

Type *Markdown* and LaTeX: α^2

1. NumPy uses much less memory to store data

The NumPy arrays takes significantly less amount of memory as compared to python lists. It also provides a mechanism of specifying the data types of the contents, which allows further optimisation of the code.

In [1]:

```
import numpy as np
import sys

py_arr = [1,2,3,4,5,6]

numpy_arr = np.array([1,2,3,4,5,6])

sizeof_py_arr = sys.getsizeof(1) * len(py_arr)

sizeof_numpy_arr = numpy_arr.nbytes # numpy_arr.itemsize * numpy_arr.size
```

In [2]:

```
sys.getsizeof(1), sizeof_py_arr
```

Out[2]:

```
(28, 168)
```

In [3]:

```
sizeof_numpy_arr
```

Out[3]:

```
24
```

In the example above, NumPy by default considers these integers as 8 Bytes integers, however, we can provide data types with NumPy arrays if we know the maximum range of the data. For example, we can use 1 Byte integer for storing numbers upto 255 and 2 Bytes integer for numbers upto 65535

In [4]:

```
# For NumPy arrays elements limited to 1 Byte / 8 Bits
numpy_arr = np.array([1,2,3,4,5,6], dtype = np.int8)
sizeof_numpy_arr = numpy_arr.nbytes # numpy_arr.itemsize * numpy_arr.size   # Size = 6
print("int8 -", sizeof_numpy_arr)

# For NumPy arrays elements limited to 2 Bytes / 16 Bits

numpy_arr = np.array([1,2,3,4,5,6], dtype = np.int16)
sizeof_numpy_arr = numpy_arr.nbytes # numpy_arr.itemsize * numpy_arr.size   # Size = 12
print("int16 -", sizeof_numpy_arr)
```

```
int8 - 6
int16 - 12
```

2)NumPy Arrays are faster than Python List

In [5]:

```
# importing required packages
import numpy
import time
```

In [6]:

```
# declaring lists
list1 = [i for i in range(100)]
list2 = [i for i in range(100)]
```

In [7]:

```
# declaring arrays
array1 = numpy.arange(100)
array2 = numpy.arange(100)
```

In [8]:

```
initialTime = time.time()
list1 = list1 + list2

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

```
Time taken by Lists : 0.0 seconds
```

In [9]:

```
initialTime = time.time()
array = numpy.concatenate((array1, array2),
                          axis = 0)
# calculating execution time
print("Time taken by NumPy Arrays :",
      (time.time() - initialTime),
      "seconds")
```

```
Time taken by NumPy Arrays : 0.0 seconds
```

In [10]:

```
# Dot Product
dot = 0
print("\nDot Product:")

# list
initialTime = time.time()
for a, b in zip(list1, list2):
        dot = dot + (a * b)

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

# NumPy array
initialTime = time.time()
array = numpy.dot(array1, array2)

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

```

Dot Product:
Time taken by Lists : 0.0 seconds
Time taken by NumPy Arrays : 0.008822202682495117 seconds
```

In [54]:

```
# Scalar Addition
print("\nScalar Addition:")

# list
initialTime = time.time()
list1 =[i + 2 for i in range(100)]

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

# NumPy array
initialTime = time.time()
array1 = array1 + 2

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

```
Scalar Addition:
Time taken by Lists : 0.0 seconds
Time taken by NumPy Arrays : 0.0 seconds
```

In [ ]: