Array and NumPy Array

By Sunil Ghimire- "**I try not to clutter common sense for the illusion of knowledge**"

INTRODUCTION:

NumPy (***An open-source library that is used for working with arrays)*** is not just more efficient but also more convenient. We get a lot of vector and matrix operations for free, which sometimes allows one to avoid unnecessary work. And they are also efficiently implemented.

In the previous section, we discussed:

1. What is Data Science?
2. What is Machine Learning?
3. Types of Machine Learning with suitable example
4. Installing Python and Jupyter Notebook on your operating system
5. Brief introduction about Machine Learning useful packages (NumPy, Pandas, and Matplotlib)

Today, we are continuing the Practical example behind NumPy, Pandas, and Matplotlib. So, Python stores data in several different ways but the most popular data structure methods to store data are List, Array, and Dictionary. Also, we know Array is similar to NumPy Array.

At first, we will learn about:

1. How these Array and NumPy Array are similar?
2. How they are different?
3. Why should we use the NumPy library compared to the python library?

SIMILARITIES BETWEEN ARRAY AND NUMPY ARRAY :

Array	NumPy Array

• **Storing Data**
`import array as arr`
`python_array = arr.array('i', [1, 2, 3, 4, 5])`
	• **Storing Data**
	`import numpy as np`
	`numpy_array = np.array([1, 2, 3, 4, 5])`

• **Mutable**
`python_array.append(6)`
`python_array.remove(5)`
`python_array[3] = 9`
	• **Mutable**
	`np.append(numpy_array, [6, 7, 8])`
	`np.delete(numpy_array, [5])`

• **Can be indexed** `python_array[2]`	• **Can be indexed** `numpy_array[3]`
• **Slicing Operation** `python_array[1:3]`	• **Slicing Operation** `numpy_array[1:3]`

DIFFERENCE BETWEEN ARRAY AND NUMPY ARRAY :

Array	NumPy Array

• **Same Type**
`python_array = arr.array('i', [1.2, 2, 3, 4, 5])`
`Result: TypeError: integer argument expected, got float`
	• **Different Type**
	`numpy_array = np.array([1, 2.5, 3, 4, 5])`

• **Operation can't perform**
`python_array / 2`
`Result: TypeError: unsupported operand type(s) for /: 'array.array' and 'int'`
	• **Operation can perform**
	`numpy_array / 2`

• **Built_in** `import array as arr`	• **Install NumPy** `pip install numpy`

WHY SHOULD WE USE NUMPY ARRAY COMPARED TO PYTHON ARRAY ?

For storage purposes, NumPy array beats array.array. Here is the code for the benchmark for both comparing storage size of unsigned integer of 4 bytes. Other data types can also be used for comparison. Data of the list and tuple is also added for comparison.

CODE:

```
import numpy as np
import array as array
import sys
a = [100400, 1004000, 10040, 100400, 100400, 100400, 400, 1006000, 8, 800999]
print("size per element for list, tuple, numpy array, array.array:===============")
for i in range(1, 15):
    aa = a*i #list size multiplier
    n = len(aa)
    list_size = sys.getsizeof(aa)
    d = list_size
    tup_aa = tuple(aa)
    tup_size = sys.getsizeof(tup_aa)
    nparr = np.array(aa, dtype='uint32')
    np_size = sys.getsizeof(nparr)
    arr = array.array('L', aa) #4 byte unsigned integer
    arr_size = sys.getsizeof(arr)
    print('number of element:%s, list %.2f, tuple %.2f, np.array %.2f, arr.array %.2f' % \
          (len(aa), list_size/n, tup_size/n, np_size/n, arr_size/n))
```

OUTPUT:

```
size per element for list, tuple, numpy array, array.array:==============
number of element:10, list 13.60, tuple 12.00, np.array 13.60, arr.array 14.40
number of element:20, list 10.80, tuple 10.00, np.array 8.80, arr.array 11.20
number of element:30, list 9.87, tuple 9.33, np.array 7.20, arr.array 10.13
number of element:40, list 9.40, tuple 9.00, np.array 6.40, arr.array 9.60
number of element:50, list 9.12, tuple 8.80, np.array 5.92, arr.array 9.28
number of element:60, list 8.93, tuple 8.67, np.array 5.60, arr.array 9.07
number of element:70, list 8.80, tuple 8.57, np.array 5.37, arr.array 8.91
number of element:80, list 8.70, tuple 8.50, np.array 5.20, arr.array 8.80
number of element:90, list 8.62, tuple 8.44, np.array 5.07, arr.array 8.71
number of element:100, list 8.56, tuple 8.40, np.array 4.96, arr.array 8.64
number of element:110, list 8.51, tuple 8.36, np.array 4.87, arr.array 8.58
number of element:120, list 8.47, tuple 8.33, np.array 4.80, arr.array 8.53
number of element:130, list 8.43, tuple 8.31, np.array 4.74, arr.array 8.49
number of element:140, list 8.40, tuple 8.29, np.array 4.69, arr.array 8.46
```

Note: ***For numbers smaller than 10, lists and tuples can be used.***

Additionally,

NumPy's arrays are more compact than Python arrays-- an array of arrays as we describe, in Python, would take at least 20 MB or so, while a NumPy 3D array with single-precision floats in the cells would fit in 4 MB. Access to reading and writing items is also faster with NumPy.

Maybe we don't care that much for just a million cells, but we definitely would for a billion cells -- neither approach would fit in a 32-bit architecture, but with 64-bit builds NumPy would get away with 4 GB or so, Python alone would need at least about 12 GB (lots of pointers which double in size) -- a much costlier piece of hardware!

The difference is mostly due to "indirectness" -- a Python array is an array of pointers to Python objects, at least 4 bytes per pointer plus 16 bytes for even the smallest Python object (4 for type pointer, 4 for reference count, 4 for value -- and the memory allocators rounds up to 16). A NumPy array is an array of uniform values -- single-precision numbers take 4 bytes each, double-precision ones, 8 bytes. Less flexible, but you pay substantially for the flexibility of standard Python lists! (*Source:* ***Stackoverflow***)