Python Lists vs. Numpy Arrays - What is the difference?

Numpy (http://www.numpy.org/) is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension.

The Python core library provided **Lists** (https://webcourses.ucf.edu/courses/1249560/pages/lists). A list is the Python equivalent of an array, but is resizeable and can contain elements of different types.

A common beginner question is what is the real difference here. The answer is performance. Numpy data structures perform better in:

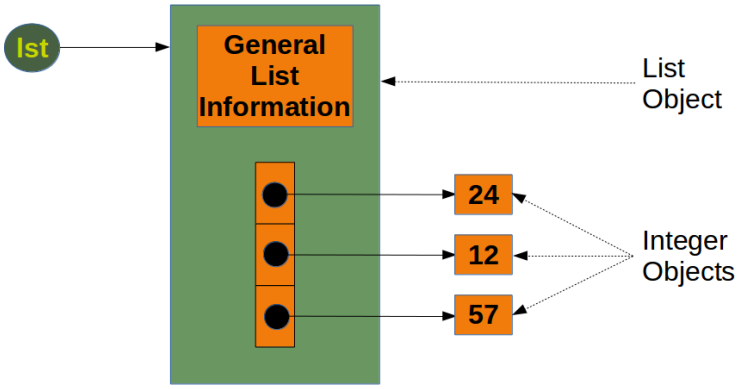
- **Size** - Numpy data structures take up less space
- **Performance** - they have a need for speed and are faster than lists
- **Functionality** - SciPy and NumPy have optimized functions such as linear algebra operations built in.

Memory

The main benefits of using NumPy arrays should be smaller memory consumption and better runtime behavior.

For Python Lists - 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:

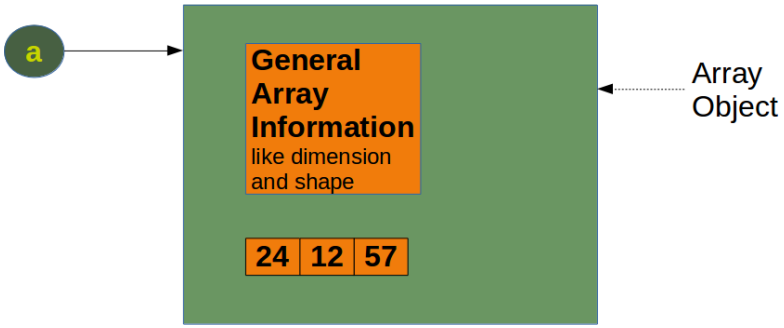
64 + 8 * len(lst) + + *len(lst)* * 28



NumPy takes up less space. This means that an arbitrary integer array of length "n" in numpy needs

96 + n * 8 Bytes

whereas a list of integer



So the more numbers you need to store - the better you do.

Speed

This shows some performance numbers of operations between Python and Numpy. Notice how the 2nd set of numbers (NumPy) are always smaller - meaning they have much better performance than their Python List core library conterparts.

script.py IPython Shell

```
import time
import numpy as np

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


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

```

Hint Run

This shows some performance numbers of operations between Python and Numpy. Notice how the 2nd set of numbers (NumPy) are always smaller - meaning they have much better performance than their Python List core library conterparts.

script.py IPython Shell

```
import numpy as np
from timeit import Timer

size_of_vec = 1000
X_list = range(size_of_vec)
Y_list = range(size_of_vec)
X = np.arange(size_of_vec)
Y = np.arange(size_of_vec)

def pure_python_version():
    Z = [X_list[i] + Y_list[i] for i in range(len(X_list)) ]
```

```
12
13  def numpy_version():
14      Z = X + Y
15
16  timer_obj1 = Timer("pure_python_version()",
17                     "from __main__ import pure_python_version")
18  timer_obj2 = Timer("numpy_version()",
19                     "from __main__ import numpy_version")
20
21  print(timer_obj1.timeit(10))
22  print(timer_obj2.timeit(10))  # Runs Faster!
23
24  print(timer_obj1.repeat(repeat=3, number=10))
25  print(timer_obj2.repeat(repeat=3, number=10)) # repeat to prove it!
26
27
```

Hint Run

References:

1. Python Course - NumPy Tutorial (https://www.python-course.eu/numpy.php)
2. NumPy Documentation (http://www.numpy.org/) Page