Data Analysis with Python

Intro to NumPy

Aim: learn why **NumPy** is an important library for the data-processing world in Python

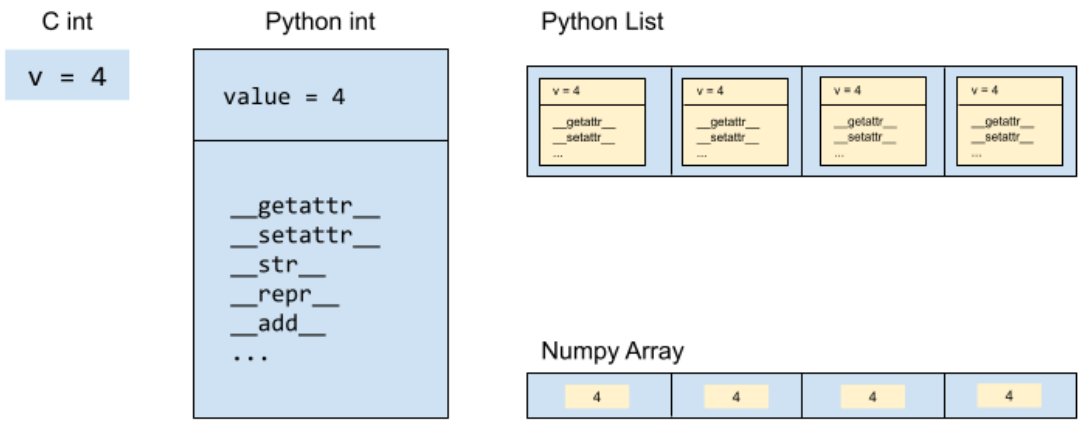
- NumPy provides the following:
  - Computations, memory storage, illustrates how Excel will always be limited when processing large volumes of data and more

Numpy: Numeric computing library

NumPy (Numerical Python): one of the core packages for numerical computing in Python, Pandas, Matplotlib, Statmodels

- many other scientific libraries rely on NumPy
- Major contributions:
  - Efficient numerical computation with C primitives
  - Efficient collections with vectorized operations
  - An integrated and natural Linear Algebra API
  - A C API for connecting NumPy with libraries written in C, C++, or FORTRAN

Let's develop on efficiency. In Python, **everything is an object**, which means that even simple ints are also objects, with all the required machinery to make object work. We call them "Boxed Ints". In contrast, NumPy uses primitive numeric types (floats, ints) which makes storing and computation efficient.



```
In [ ]:
import sys
import numpy as np
```

import sys
import numpy as np

Basic Numpy Arrays

```
In [3]:
# Basic Numpy Arrays
np.array([1, 2, 3, 4])

array([1, 2, 3, 4])
```

np.array([1, 2, 3, 4])

```
In [4]:
a = np.array([1, 2, 3, 4])

In [5]:
b = np.array([0, .5, 1, 1.5, 2])
```

a = np.array([1, 2, 3, 4])
b = np.array([0, .5, 1, 1.5, 2])

```
In [6]:
a[0], a[1]

(1, 2)
```

a[0], a[1]

```
In [7]:
a[0:]

array([1, 2, 3, 4])
```

a[0:]

- Provides all entries from 0 index up to AND including the max index.
- Note: compared to the output above (*In [6]:*), *In[7]:* has a **range**
    - Specifically '**:**' results in output containing array([...])
    - While *In[6]:*, extracting entries within certain indices provides (1, 2)
        - generally , (...)

```
In [8]: a[1:3]
        # Indices from 1 up to 3, excluding index 3.

        array([2, 3])
```

a[1:3]

```
In [9]: a[1:-1]

        array([2, 3])
```

a[1:-1]

```
In [10]: a[::2]

         array([1, 3])
```

a[::2]

- The index entries right before AND after *2*

a[::int]

- Generally, the entries right before and after the specified int

```
In [11]: b

         array([0. , 0.5, 1. , 1.5, 2. ])

In [12]: b[0], b[2], b[-1]

         (0.0, 1.0, 2.0)
```

b[0], b[2], b[-1]

```
In [14]: b[[0, 2, -1]]

         array([0., 1., 2.])
```

b[[0, 2, -1]]

- Compared to the code above (*In[12]*), *In[14]* provides array([...])
- While *In[12]* provides (...)
- Generally, b[index]
  - Provides (...)

- And b[[...]]
    - Provides array([...])

Array Types

```
In [16]: a
Out[16]: array([1, 2, 3, 4])

In [17]: a.dtype
Out[17]: dtype('int32')
```

a.dtype

```
In [18]: b
Out[18]: array([0. , 0.5, 1. , 1.5, 2. ])

In [19]: b.dtype
Out[19]: dtype('float64')
```

b.dtype

```
In [21]: np.array([1, 2, 3, 4], dtype=float)
Out[21]: array([1., 2., 3., 4.])
```

np.array([1, 2, 3, 4], dtype=float)

```
In [23]: np.array([1, 2, 3, 4], dtype=np.int8)
Out[23]: array([1, 2, 3, 4], dtype=int8)
```

```
In [24]: c = np.array(['a', 'b', 'c'])
```

c = np.array([‘a’, ‘b’, ‘c’])

```
In [25]: c.dtype

dtype('<U1')
```

c.dtype

```
In [26]: d = np.array([{'a': 1}, sys])

In [27]: d.dtype

dtype('O')
```

d = np.array([{'a': 1}, sys])
d.dtype

Dimensions and shapes

```
In [28]: # Dimensions and shapes
         A = np.array([
             [1, 2, 3],
             [4, 5, 6]
         ])
```

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

```
In [30]: A.shape
         # (2 rows, 3 columns)

(2, 3)
```

A.shape

```
In [40]: A.ndim
         # 2 sets of [[?

2
```

A..ndim

```
In [34]:
B = np.array([
    [
        [12, 11, 10],
        [9, 8, 7]
    ],
    [
        [6, 5, 4],
        [3, 2, 1]
    ]
])
```

B = np.array([
    [
        [12, 11, 10],
        [9, 8, 7]
    ],
    [
        [6, 5, 4],
        [3, 2, 1]
    ]
])

```
In [35]:
B

array([[[12, 11, 10],
        [ 9,  8,  7]],

       [[ 6,  5,  4],
        [ 3,  2,  1]]])
```

```
In [37]:
B.shape
# 2 groups, 2 rows in each group, 3 columns in each group

(2, 2, 3)
```

B.shape

```
In [39]: B.ndim

3
```

B.ndim

```
In [42]: B.size
         # 12 individual numbers within the array.

12
```

B.size

```
In [44]: C = np.array([
             [
                 [12, 11, 10],
                 [9, 8, 7],
             ],
             [
                 [6, 5, 4]
             ]
         ])
```

```
In [45]: C.dtype

dtype('O')
```

C.dtype

```
In [46]: C.shape

(2,)
```

C.shape

- Take a look at what *C* looks like when prompted

```
In [167]: C

          array([list([[12, 11, 10], [9, 8, 7]]), list([[6, 5, 4]])], dtype=object)
```

```
In [47]: C.size

         2
```

C.size

- *2* b/c C consists of 2 lists

```
In [48]: type(C[0])

         list
```

type(C[0])

---

tbc...

References

https://www.youtube.com/watch?v=r-uOLxNrNk8

https://github.com/ine-rmotr-curriculum/freecodecamp-intro-to-numpy/blob/master/2.%20NumPy.ipynb

https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.ndarray.html#array-methods