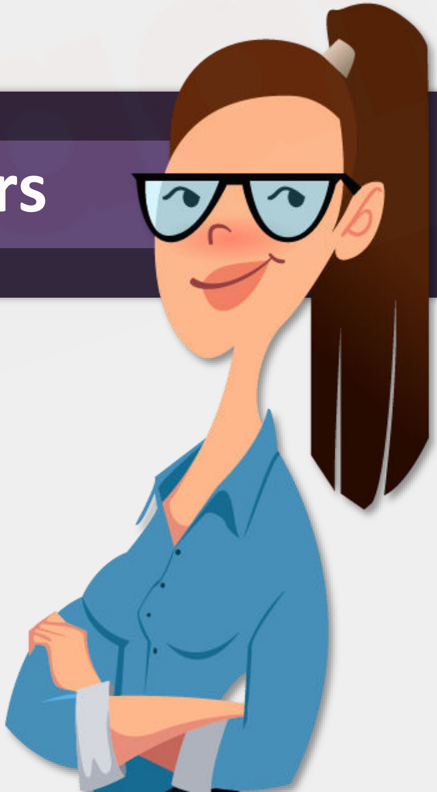
Numpy Training for Beginners



What is Numpy?

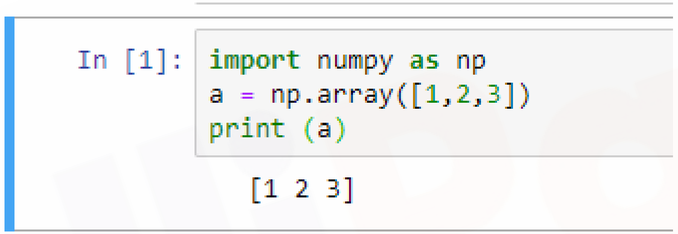
- Linear algebra library in Python
- Used for performing mathematical and logical operations on Arrays
- Provides features for operations on multi-dimensional arrays and matrices in Python

How to create Numpy Array?

Creating Numpy Array

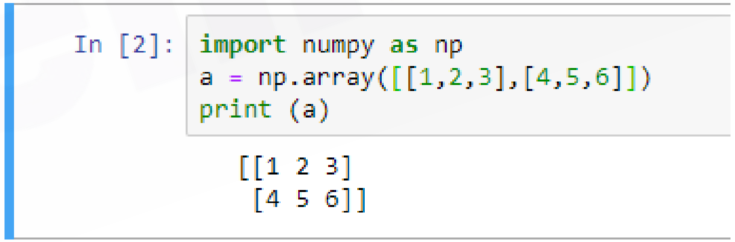
Creating Numpy Array

1D Array



```
In [1]: import numpy as np
        a = np.array([1,2,3])
        print (a)

        [1 2 3]
```

2D Array

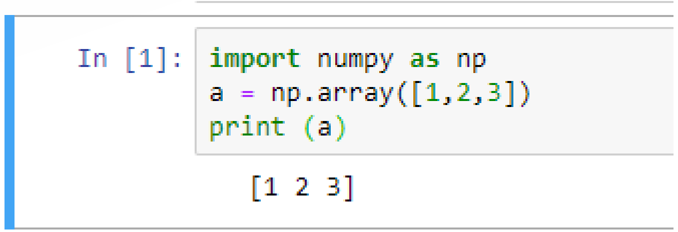


```
In [2]: import numpy as np
        a = np.array([[1,2,3],[4,5,6]])
        print (a)

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

What is Numpy Array

Ndarray Object

- Most important object defined in NumPy is an N-dimensional array type called **ndarray**
- Describes the collection of items of the same type
- Items can be accessed using a zero-based index
- Every item in an ndarray takes the same size of block in the memory.
- Each element in ndarray is an object of data-type object (called dtype).



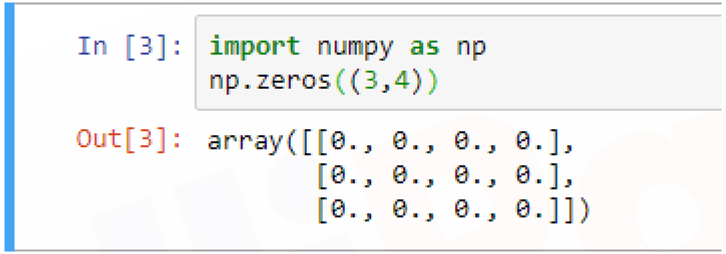
```
In [1]: import numpy as np
        a = np.array([1,2,3])
        print (a)

        [1 2 3]
```

Numpy Array Initialization

Initializing Numpy Array

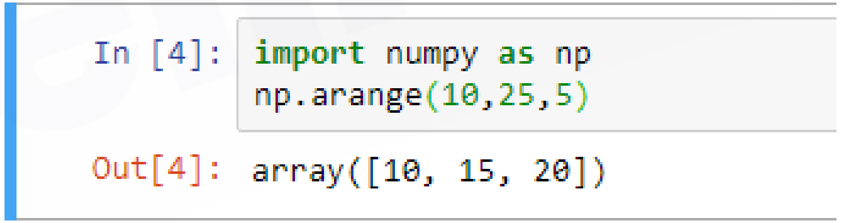
Initializing Numpy Array

Initialize an array of 'x' X 'y' dimension with 0

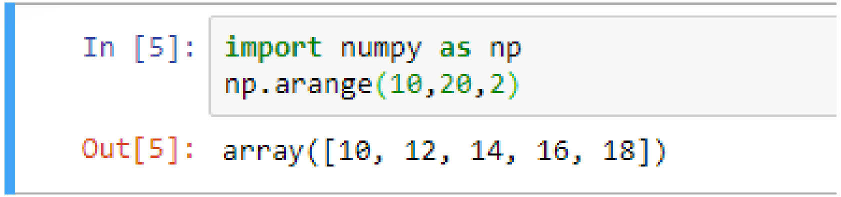


```
In [3]: import numpy as np
        np.zeros((3,4))

Out[3]: array([[0., 0., 0., 0.],
               [0., 0., 0., 0.],
               [0., 0., 0., 0.]])
```

Arranging the numbers between x and y with an interval of z



```
In [4]: import numpy as np
        np.arange(10,25,5)

Out[4]: array([10, 15, 20])
```

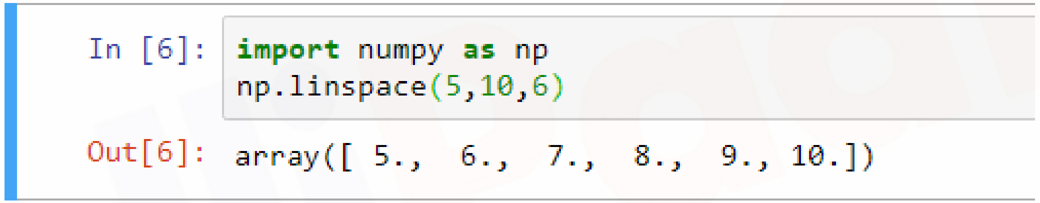


```
In [5]: import numpy as np
        np.arange(10,20,2)

Out[5]: array([10, 12, 14, 16, 18])
```

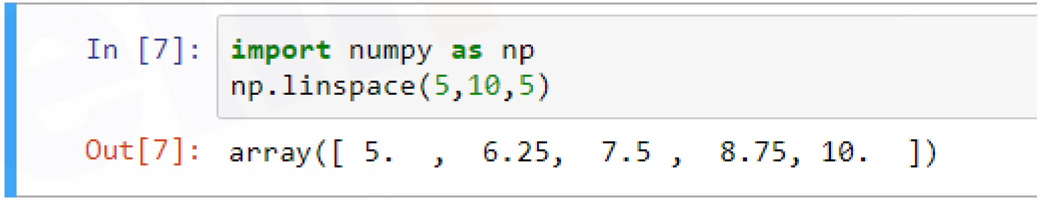
Initializing Numpy Array

Initializing Numpy Array

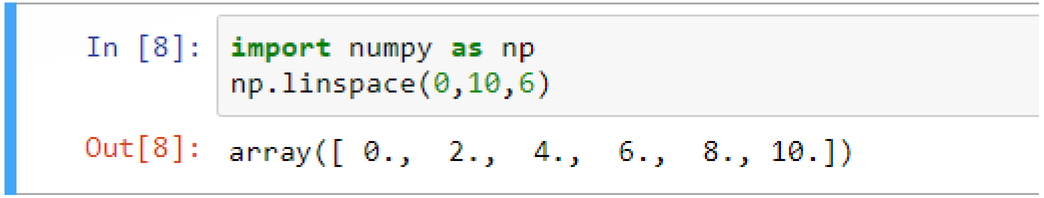
Arranging ‘z’ numbers between x and y



```
In [6]: import numpy as np
        np.linspace(5,10,6)

Out[6]: array([ 5.,  6.,  7.,  8.,  9., 10.])
```



```
In [7]: import numpy as np
        np.linspace(5,10,5)

Out[7]: array([ 5.  ,  6.25,  7.5 ,  8.75, 10.  ])
```

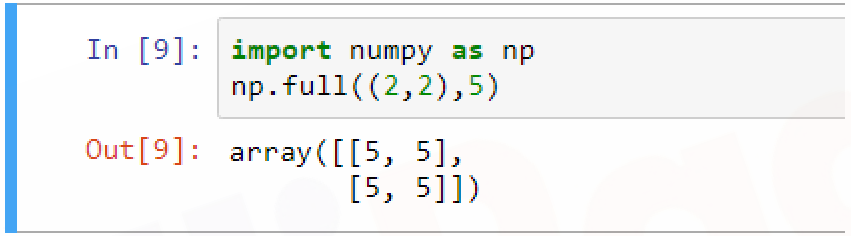


```
In [8]: import numpy as np
        np.linspace(0,10,6)

Out[8]: array([ 0.,  2.,  4.,  6.,  8., 10.])
```

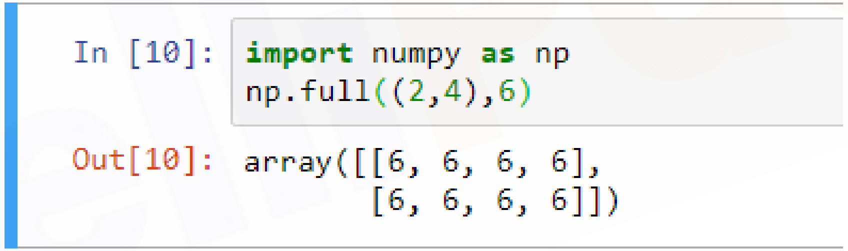
Initializing Numpy Array

Initializing Numpy Array

Filling SAME number in a array of dimension x X y

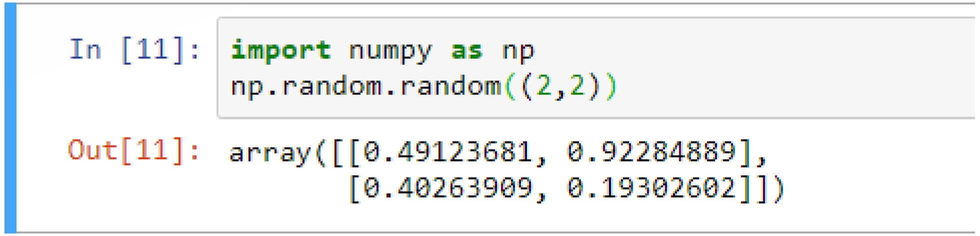


```
In [9]:  import numpy as np
          np.full((2,2),5)

Out[9]:  array([[5, 5],
                [5, 5]])
```



```
In [10]:  import numpy as np
           np.full((2,4),6)

Out[10]:  array([[6, 6, 6, 6],
                 [6, 6, 6, 6]])
```

Filling RANDOM numbers in a array of dimension x X y



```
In [11]:  import numpy as np
           np.random.random((2,2))

Out[11]:  array([[0.49123681, 0.92284889],
                 [0.40263909, 0.19302602]])
```

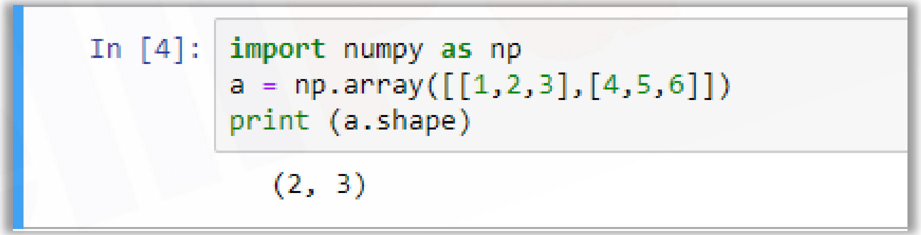
Numpy Array Inspection

Numpy Array Inspection

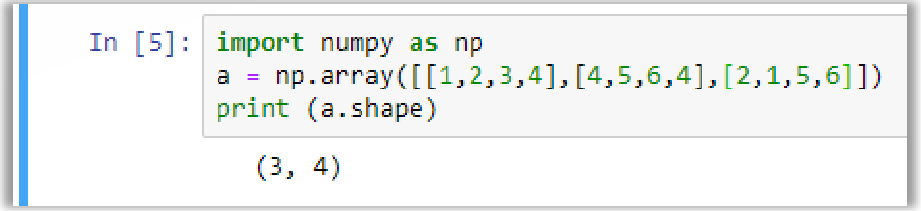
ndarray.shape

"Returns a tuple consisting of array dimensions. Can also be used to resize the array."

For Example,



```
In [4]: import numpy as np
        a = np.array([[1,2,3],[4,5,6]])
        print (a.shape)

(2, 3)
```



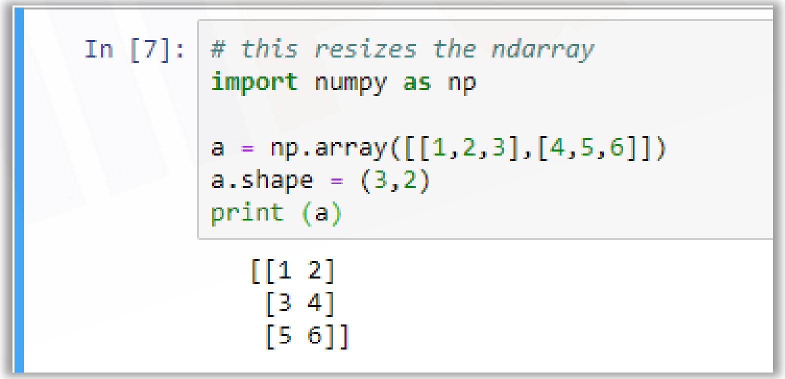
```
In [5]: import numpy as np
        a = np.array([[1,2,3,4],[4,5,6,4],[2,1,5,6]])
        print (a.shape)

(3, 4)
```

Numpy Array Inspection

ndarray.shape

“Returns a tuple consisting of array dimensions. Can also be used to **resize** the array.”

For Example,



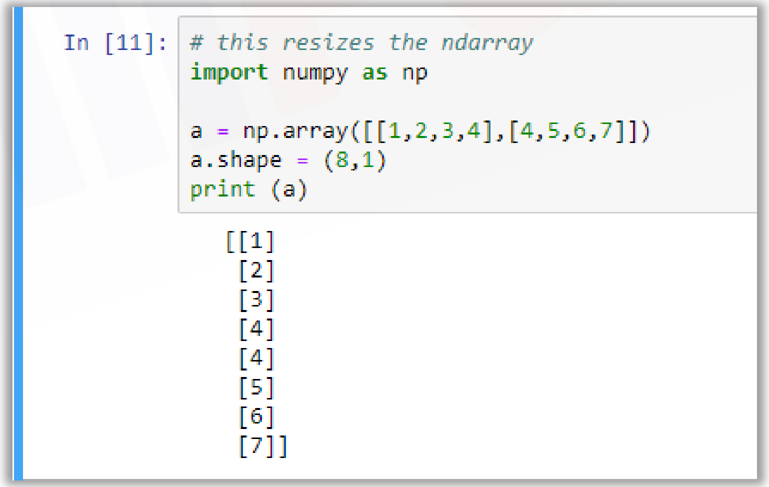
```
In [7]: # this resizes the ndarray
        import numpy as np

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

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

Numpy Array Inspection

ndarray.shape

"Returns a tuple consisting of array dimensions. Can also be used to **resize** the array."

For Example,



```
In [11]: # this resizes the ndarray
         import numpy as np

         a = np.array([[1,2,3,4],[4,5,6,7]])
         a.shape = (8,1)
         print (a)

         [[1]
          [2]
          [3]
          [4]
          [4]
          [5]
          [6]
          [7]]
```

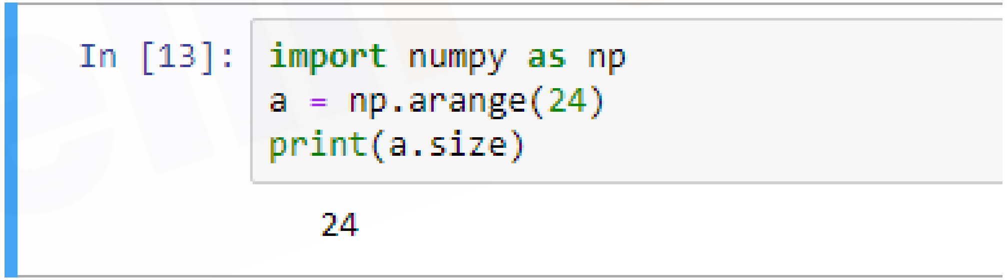
IntelliPaat

Numpy Array Inspection

ndarray.size

“Returns the count of number of elements in an array.”

For Example,



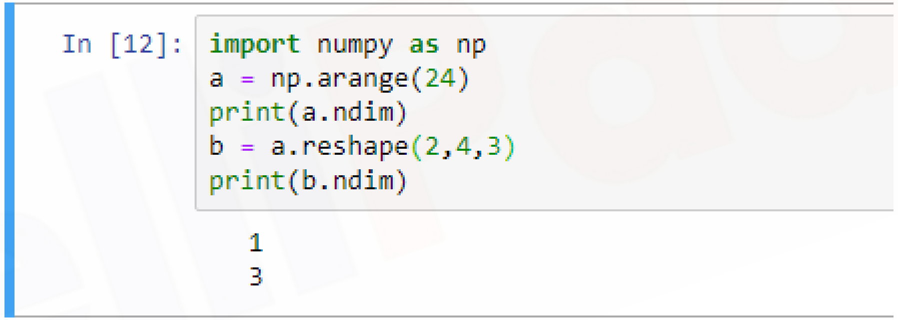
```
In [13]: import numpy as np
         a = np.arange(24)
         print(a.size)

         24
```

Numpy Array Inspection

ndarray.ndim

"Returns the dimension of the array."

For Example,



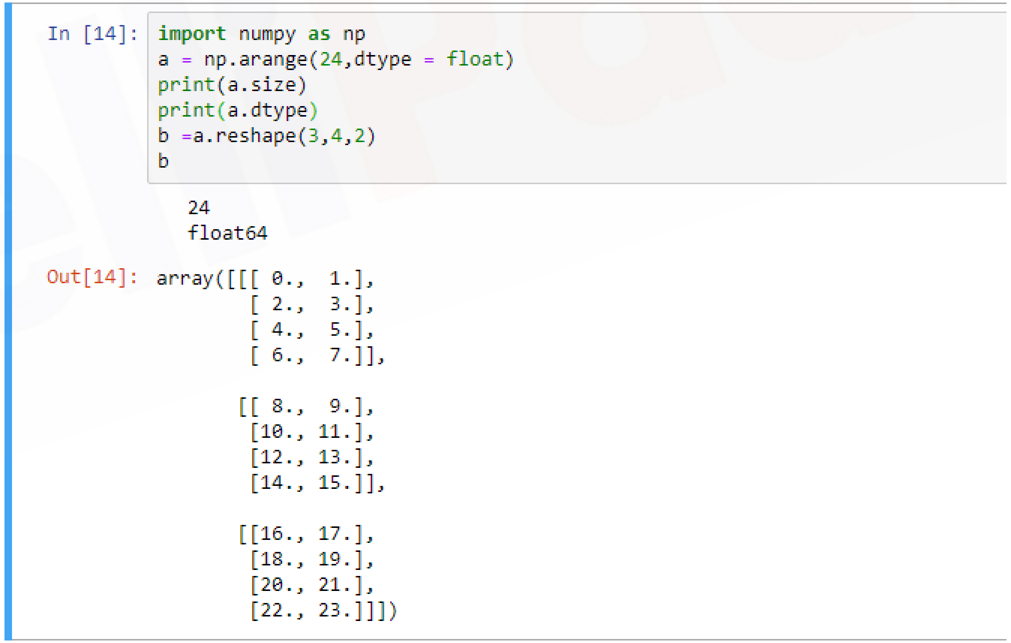
```
In [12]: import numpy as np
         a = np.arange(24)
         print(a.ndim)
         b = a.reshape(2,4,3)
         print(b.ndim)

1
3
```

Numpy Array Inspection

ndarray.dtype

“Returns datatype of an array.”

For Example,

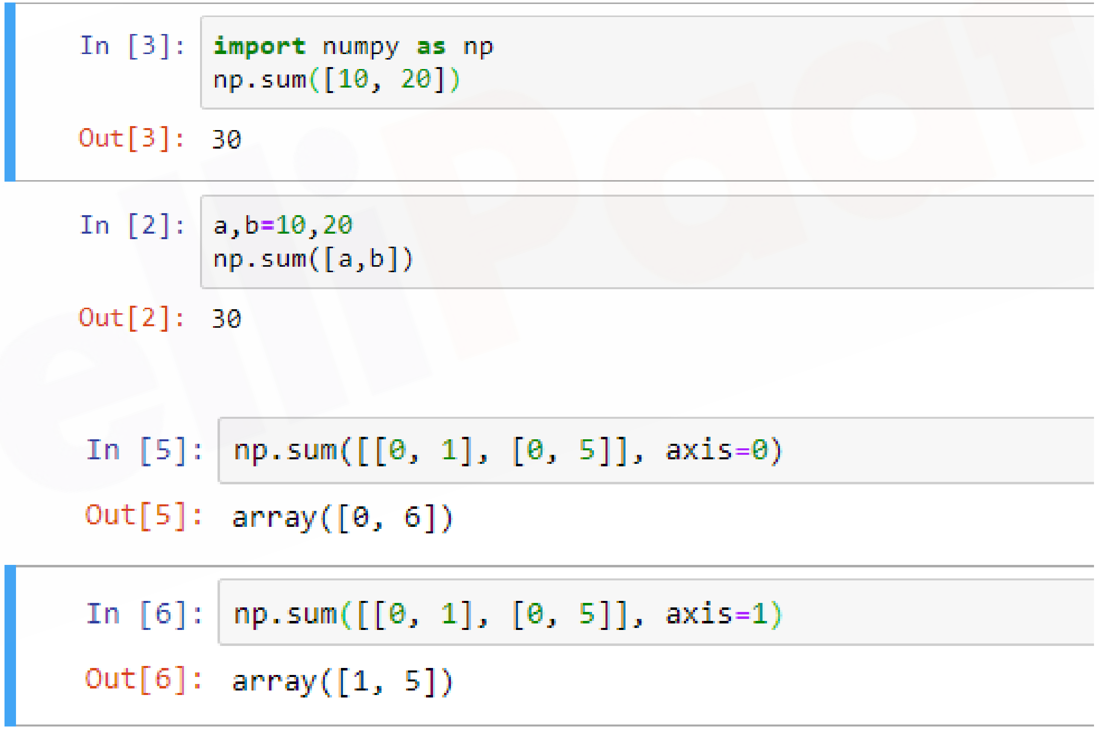


```
In [14]: import numpy as np
         a = np.arange(24,dtype = float)
         print(a.size)
         print(a.dtype)
         b =a.reshape(3,4,2)
         b

           24
           float64

Out[14]: array([[[ 0.,  1.],
                 [ 2.,  3.],
                 [ 4.,  5.],
                 [ 6.,  7.]],

                [[ 8.,  9.],
                 [10., 11.],
                 [12., 13.],
                 [14., 15.]],

                [[16., 17.],
                 [18., 19.],
                 [20., 21.],
                 [22., 23.]]])
```

Numpy Array Mathematics

Numpy Array Mathematics

Addition using Numpy



```
In [3]:  import numpy as np
          np.sum([10, 20])

Out[3]:  30

In [2]:  a,b=10,20
          np.sum([a,b])

Out[2]:  30

In [5]:  np.sum([[0, 1], [0, 5]], axis=0)

Out[5]:  array([0, 6])

In [6]:  np.sum([[0, 1], [0, 5]], axis=1)

Out[6]:  array([1, 5])
```

Numpy Array Mathematics

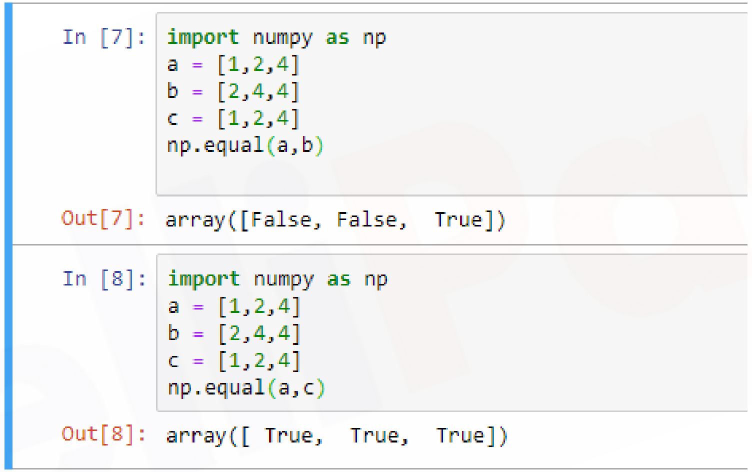
Other similar operations that you can perform:

- `np.subtract(a,b) #a-b`
- `np.divide(a,b) #a/b`
- `np.multiply(a,b) #a*b`
- `np.exp(a) #e^a`
- `np.sqrt(a)`
- `np.sin(a)`
- `np.cos(a)`
- `np.log(a)`

Numpy Array Mathematics

Array Comparison

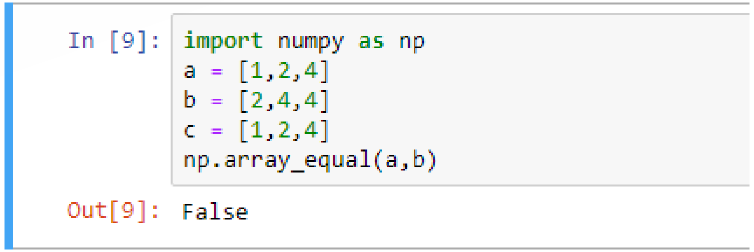
Element-wise Comparison



```
In [7]:  import numpy as np
         a = [1,2,4]
         b = [2,4,4]
         c = [1,2,4]
         np.equal(a,b)

Out[7]:  array([False, False,  True])
```

```
In [8]:  import numpy as np
         a = [1,2,4]
         b = [2,4,4]
         c = [1,2,4]
         np.equal(a,c)

Out[8]:  array([ True,  True,  True])
```

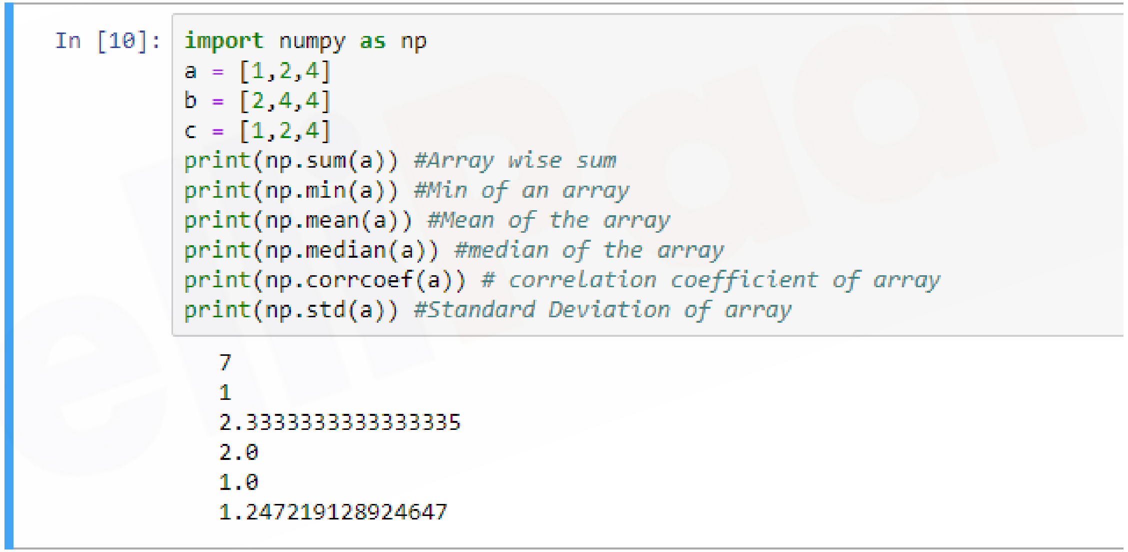
Array-wise Comparison



```
In [9]:  import numpy as np
         a = [1,2,4]
         b = [2,4,4]
         c = [1,2,4]
         np.array_equal(a,b)

Out[9]:  False
```

Numpy Array Mathematics

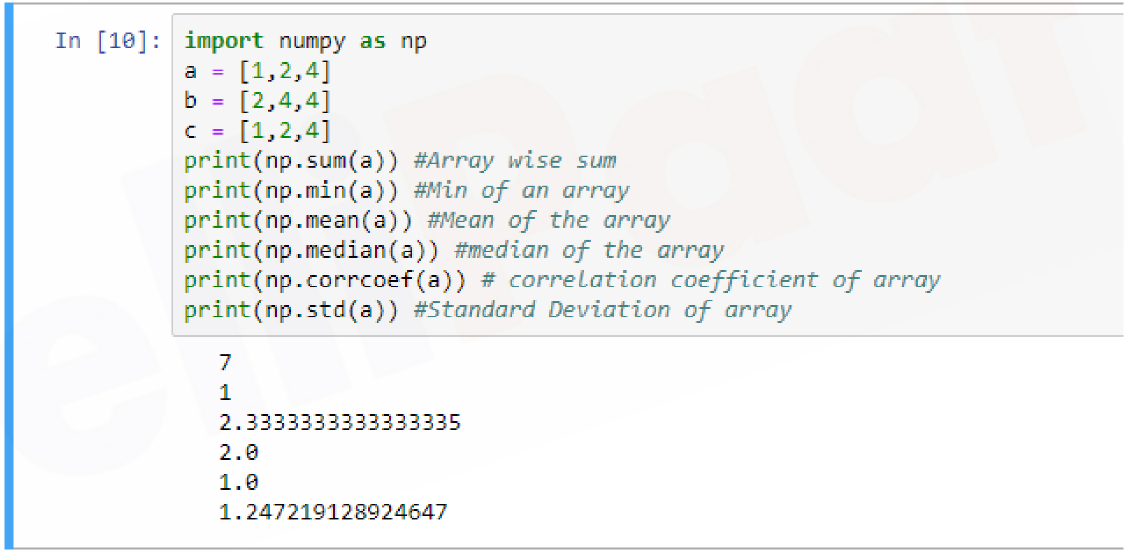
Aggregate Functions



```
In [10]: import numpy as np
         a = [1,2,4]
         b = [2,4,4]
         c = [1,2,4]
         print(np.sum(a)) #Array wise sum
         print(np.min(a)) #Min of an array
         print(np.mean(a)) #Mean of the array
         print(np.median(a)) #median of the array
         print(np.corrcoef(a)) # correlation coefficient of array
         print(np.std(a)) #Standard Deviation of array

         7
         1
         2.3333333333333335
         2.0
         1.0
         1.247219128924647
```

Numpy Array Mathematics

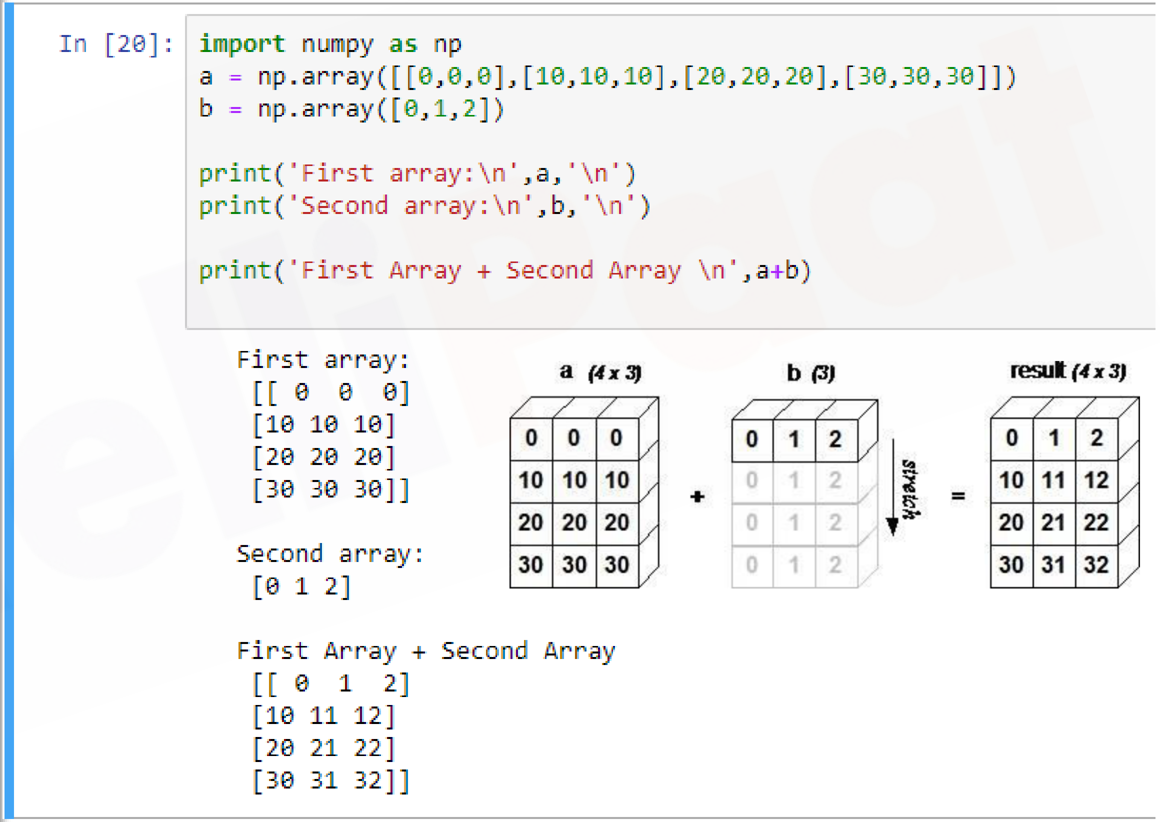
Aggregate Functions



```
In [10]: import numpy as np
         a = [1,2,4]
         b = [2,4,4]
         c = [1,2,4]
         print(np.sum(a)) #Array wise sum
         print(np.min(a)) #Min of an array
         print(np.mean(a)) #Mean of the array
         print(np.median(a)) #median of the array
         print(np.corrcoef(a)) # correlation coefficient of array
         print(np.std(a)) #Standard Deviation of array

           7
           1
           2.3333333333333335
           2.0
           1.0
           1.247219128924647
```

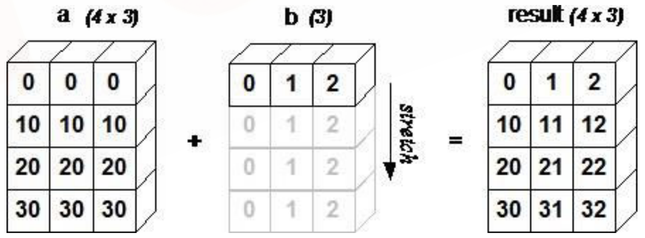
Numpy Broadcasting

Concept of Broadcasting



```
In [20]: import numpy as np
         a = np.array([[0,0,0],[10,10,10],[20,20,20],[30,30,30]])
         b = np.array([0,1,2])

         print('First array:\n',a,'\n')
         print('Second array:\n',b,'\n')

         print('First Array + Second Array \n',a+b)
```

```
First array:
 [[ 0  0  0]
 [10 10 10]
 [20 20 20]
 [30 30 30]]

Second array:
 [0 1 2]

First Array + Second Array
 [[ 0  1  2]
 [10 11 12]
 [20 21 22]
 [30 31 32]]
```



How to select certain elements from the array?

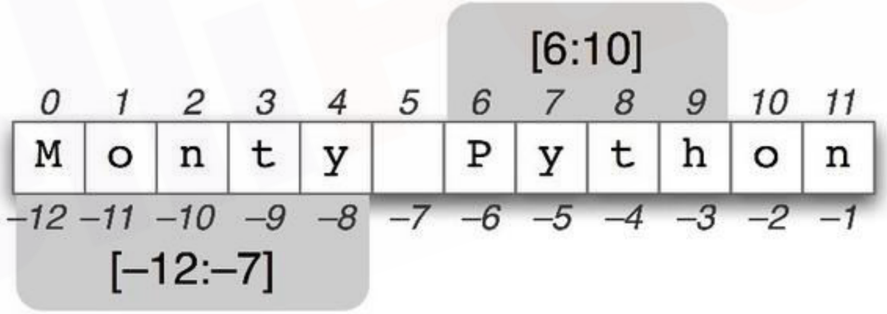
Indexing and Slicing in Python

Numpy Indexing and Slicing

Indexing in Python

“Index refers to a position .”

For Example,



Numpy Indexing and Slicing

Slicing

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

Let’s learn to extract/slice the array

Numpy Indexing and Slicing

Slicing

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

How to extract the selected element?

My selection is in 1st row = 0th index

`A[0]` ------#includes all the elements from the first row

`A[:1]` ------ #Extract first row from the array.

Numpy Indexing and Slicing

Slicing

$$\begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 2 & 3 \\ 1 & 4 & 5 & 6 \\ 2 & 7 & 8 & 9 \end{array}$$

How to extract the selected element?

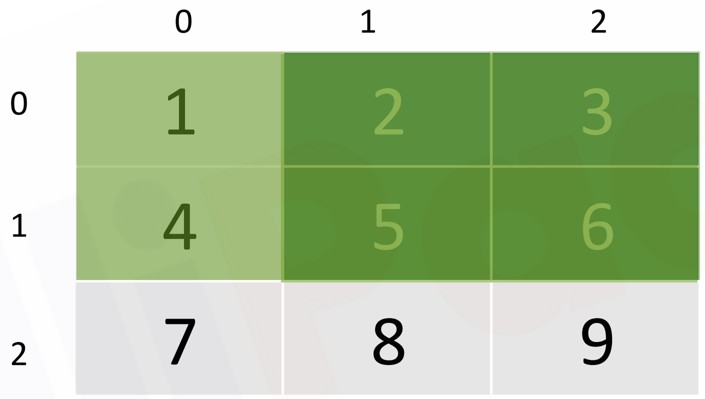
My selection is in 1st row = 0th index

`A[:1]` ------#Extracting till row = 0 (that is 0th row)

`A[:1,1:]------#Extracting till row = 0` then select the col index starting from 1 till last

Numpy Indexing and Slicing

Slicing



How to extract the selected element?

My selection is in 1st two rows = 0,1 index

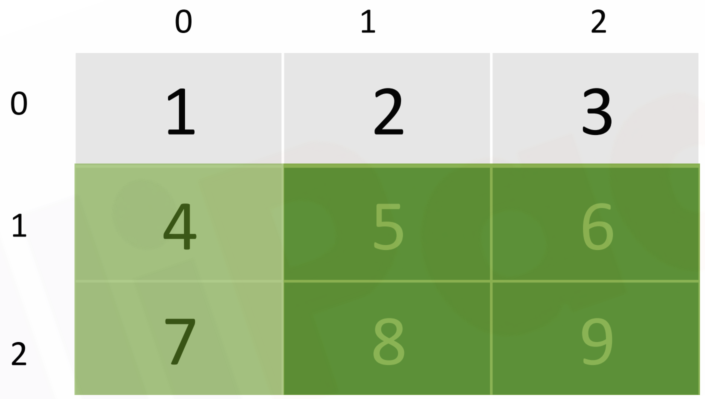
`A[:2]` ------#Extracting till row = 1 (that is 0,1)

`A[:2,1:]` ------ #Extracting till row = 1 (that is 0,1)

then select the col index starting from 1 till last

Numpy Indexing and Slicing

Slicing



How to extract the selected element?

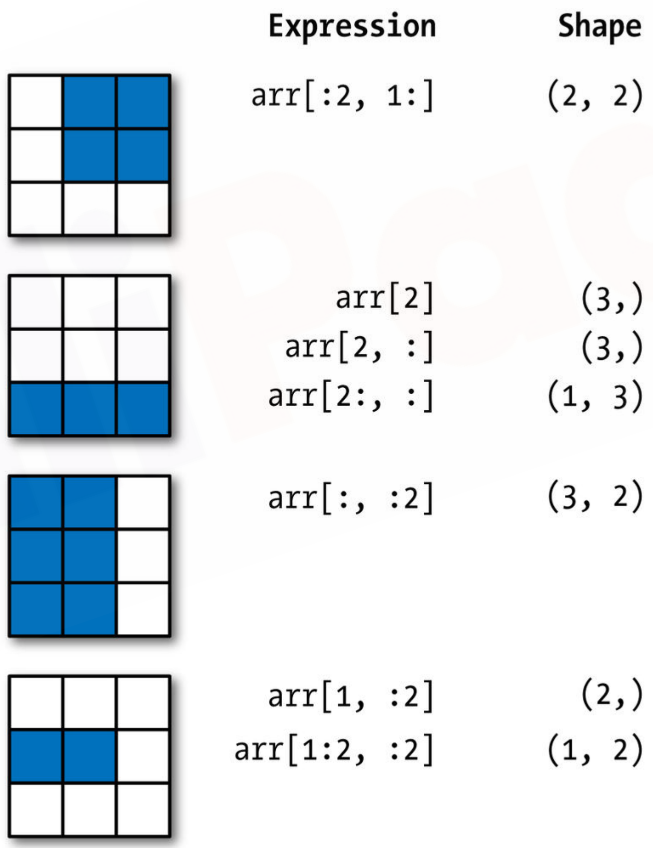
My selection is in 1st two rows = 0,1 index

`A[1:,]` ------#Extracting starts from row = 1 till end

`A[1:,1:]` ------ #Extracting starts from row = 1 till end then select col index = 1 till end

Numpy Indexing and Slicing

Slicing

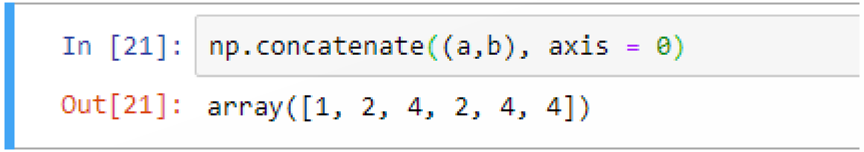


Expression	Shape
arr[:2, 1:]	(2, 2)
arr[2]	(3,)
arr[2, :]	(3,)
arr[2:, :]	(1, 3)
arr[:, :2]	(3, 2)
arr[1, :2]	(2,)
arr[1:2, :2]	(1, 2)

Array Manipulation in Python

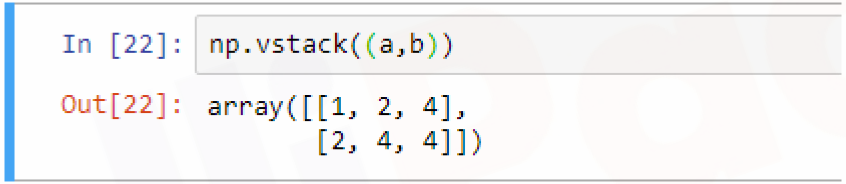
Array Manipulation

Concatenating two arrays together



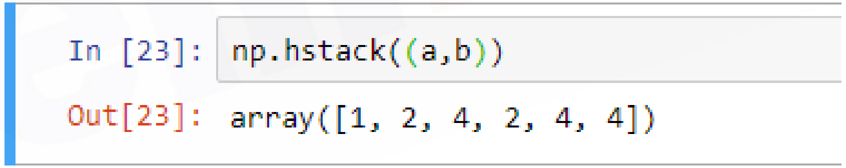
```
In [21]: np.concatenate((a,b), axis = 0)

Out[21]: array([1, 2, 4, 2, 4, 4])
```

Stack arrays row-wise(vertically)



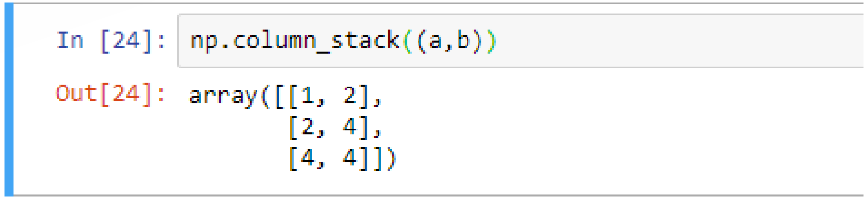
```
In [22]: np.vstack((a,b))

Out[22]: array([[1, 2, 4],
                [2, 4, 4]])
```

Stack arrays column-wise(horizontally)



```
In [23]: np.hstack((a,b))

Out[23]: array([1, 2, 4, 2, 4, 4])
```

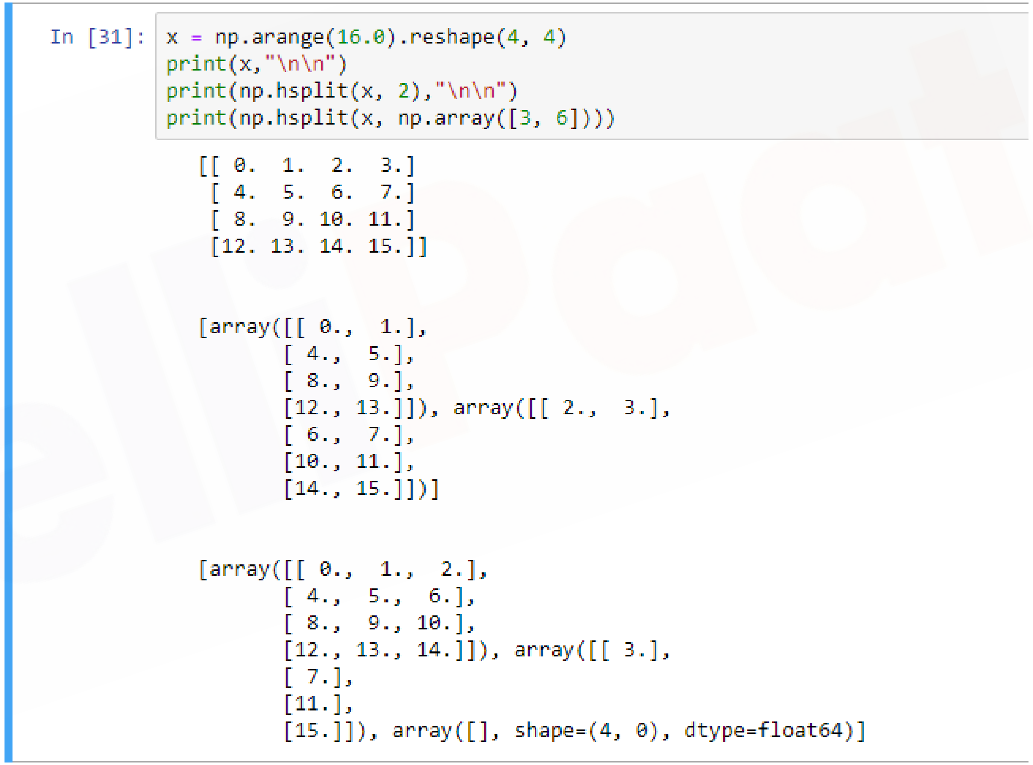
Combining column wise stacked array



```
In [24]: np.column_stack((a,b))

Out[24]: array([[1, 2],
                [2, 4],
                [4, 4]])
```

Array Manipulation

Splitting Arrays



```
In [31]: x = np.arange(16.0).reshape(4, 4)
         print(x,"\n\n")
         print(np.hsplit(x, 2),"\n\n")
         print(np.hsplit(x, np.array([3, 6])))

    [[ 0.  1.  2.  3.]
     [ 4.  5.  6.  7.]
     [ 8.  9. 10. 11.]
     [12. 13. 14. 15.]]


    [array([[ 0.,  1.],
           [ 4.,  5.],
           [ 8.,  9.],
           [12., 13.]]), array([[ 2.,  3.],
           [ 6.,  7.],
           [10., 11.],
           [14., 15.]])]


    [array([[ 0.,  1.,  2.],
           [ 4.,  5.,  6.],
           [ 8.,  9., 10.],
           [12., 13., 14.]]), array([[ 3.],
           [ 7.],
           [11.],
           [15.]]), array([], shape=(4, 0), dtype=float64)]
```

Advantages of Numpy Array over List

Why not List?

Advantages of Numpy over List

What are the advantages of Numpy over List?

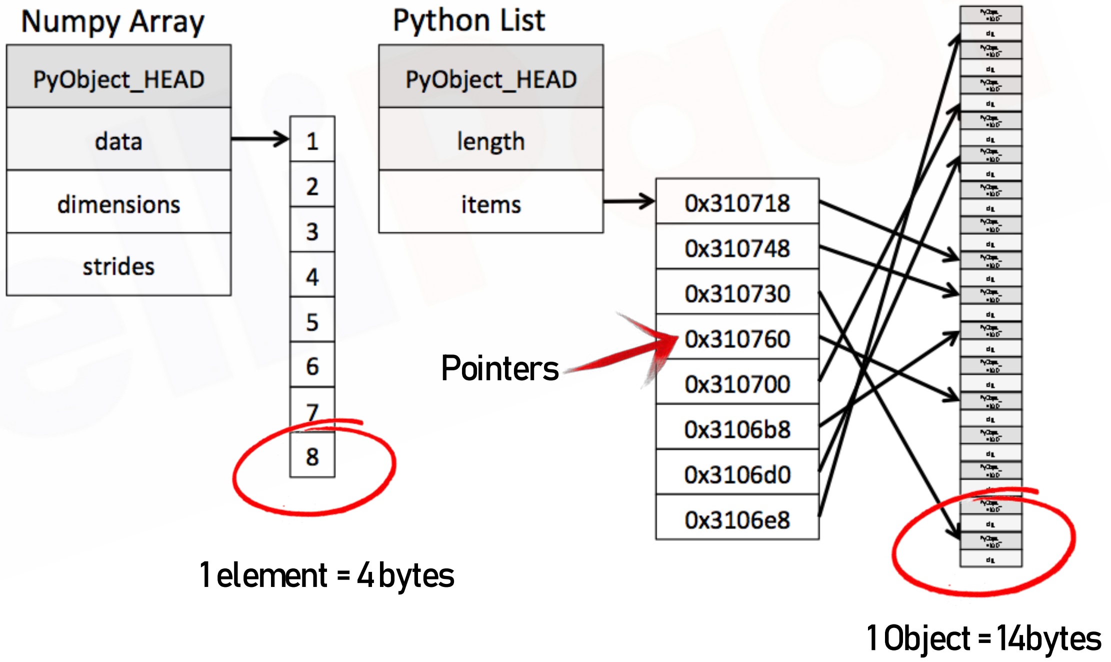
Consumes
Less Memory

Faster

More
Convenient

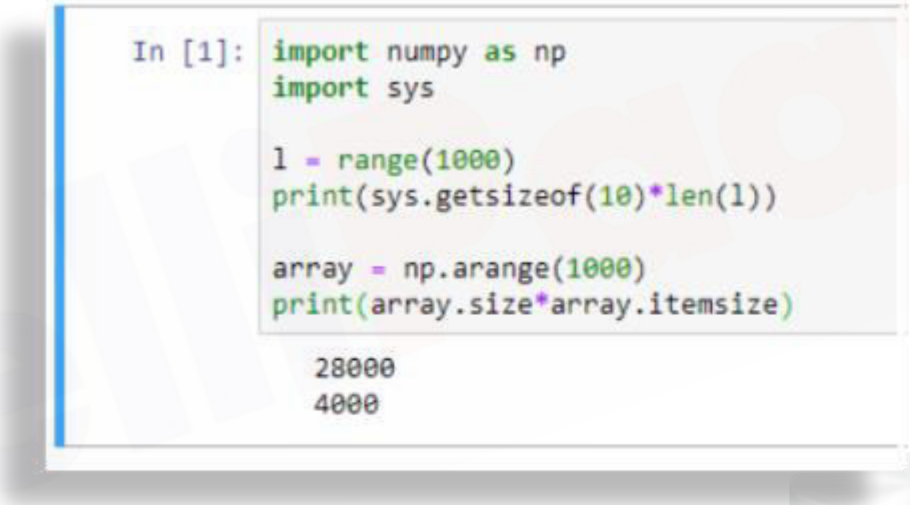
Advantages of Numpy over List

What are the advantages of Numpy over List?



Advantages of Numpy over List

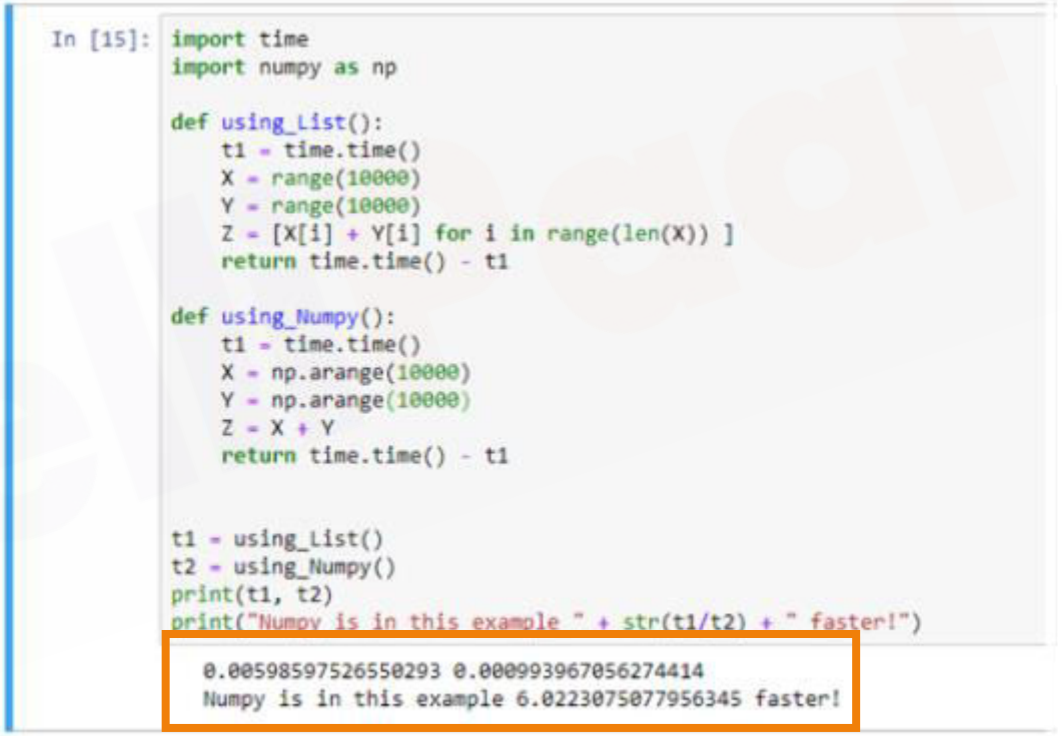
Numpy vs List: Memory Size



```
In [1]: import numpy as np
        import sys

        l = range(1000)
        print(sys.getsizeof(10)*len(l))

        array = np.arange(1000)
        print(array.size*array.itemsize)

        28000
        4000
```

Advantages of Numpy over List

Numpy vs List: Speed



```
In [15]: import time
         import numpy as np

         def using_List():
             t1 = time.time()
             X = range(10000)
             Y = range(10000)
             Z = [X[i] + Y[i] for i in range(len(X)) ]
             return time.time() - t1

         def using_Numpy():
             t1 = time.time()
             X = np.arange(10000)
             Y = np.arange(10000)
             Z = X + Y
             return time.time() - t1


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

         0.005985975265502930 0.0009939670562744141
         Numpy is in this example 6.022307507795634 5 faster!
```

Thank You

IntelliPaat

www.intellipaat.com

CALL US NOW

India : +91-7847955955

US : 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com