In [ ]:
```
            ############    ##########   Python Numpy   ###########     ###########
```

In [ ]:
```
=>  Numpy is a general-purpose array-processing package.

=>  It provides a high-performance multidimensional array object, and tools for working with these arrays.

=>  It is the fundamental package for scientific computing with Python.

=>  Numpy can also be used as an efficient multi-dimensional container of generic data.
```

In [ ]:
```
          ###########    ##########    Numpy arrays   #########    ##############

=>  Array in Numpy is a table of elements (usually numbers), all of the same type,
      indexed by a tuple of positive integers.

=>  In Numpy, number of dimensions of the array is called rank of the array.

=>  A tuple of integers giving the size of the array along each dimension is known as shape of the array.

=>  An array class in Numpy is called as ndarray.

=>   Elements in Numpy arrays are accessed by using square brackets and can be initialized
        by using nested Python Lists.
```

In [ ]:
```
       #############    ###########    Creating a Numpy Array   ##########     ##############

=>  Numpy arrays are created by using array() function, which comes under the numpy package.

=>  array(object, dtype=None, copy=True, order='K', subok=False, ndmin=0)


     *  object ->  We can pass a sequence as an object like as list, tuple, etc. .

     *  dtype ->  This is an optional attribute & stands for datatype of the resulting array.
                    If not given, it takes desired type by default.

     *  ndmin ->  This is also an optional attribute and takes integers. This specifies the required dimension
                    for the resulting array.

     *  copy :  This is an otional attribute and takes boolean values.
                 If true (default), then the object is copied.  Otherwise, a copy will only be made
                   if __array__ returns a copy,

     *  order : {'K', 'A', 'C', 'F'} ->  This is an optional attribute and specifies the memory layout of array.

     *  subok (optional) ->  If True, then sub-classes will be passed-through, otherwise
                                 the returned array will be forced to be a base-class array (default)
```

In [ ]:
```
       ##########    ##########    Datatypes of Numpy arrays    ###########   #############

=>  'dtype = complex 'is used to create a ndarray of Complex type.

=>  Specifying datatype of more than one elements.

     >>>  x = np.array([(1, 2, 3),(3, 4, 5)],dtype=[('a','<i4'),('b','<i8'), ('c', '<i2')])


=>  'i2' means int 16 bit   ,  'i4' means int 32 bit   , 'i8' means int 64 bit
```

In [ ]:
```
       ############   ############    Dimensions of Numpy arrays    ###########    ###############

=>  'ndmin' attribute is used to specify the dimension of resulting Array.

=>  A Numpy Array of N-dimension consists of 'n' no. of square brackets around them.

=>  'array.ndim' is used to return the no. of dimensions of an array.
```

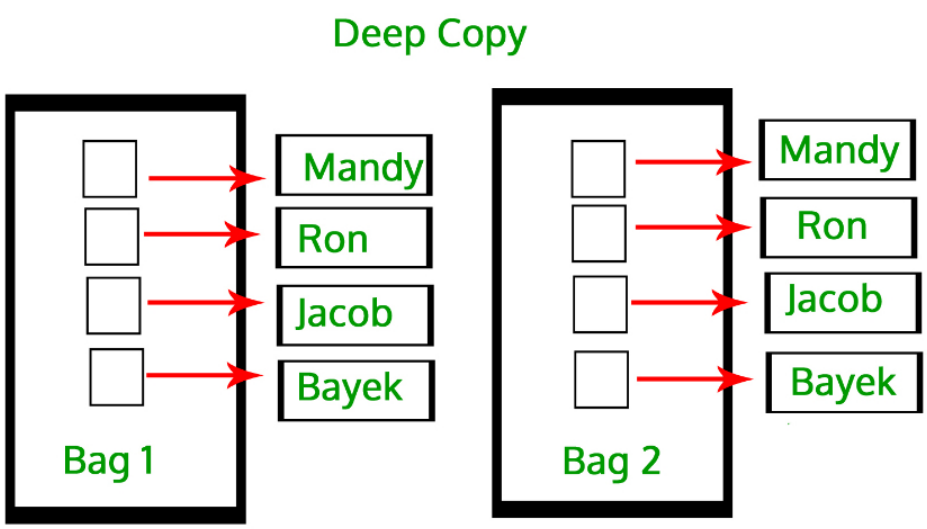
In [ ]:
```
    ############    ############    Shape and size of Numpy arrays    #############    #############

=>  'array.shape' is used to return the shape of Numpy array in the form of a Tuple (m, n).

=>  (m & n)  are no. of rows and columns of the array.

=>  'array.size' returns the total number of elements in a Numpy array.
```

In [ ]:
```
    ##########     #############   Upcasting of elements in Numpy arrrays   ##########    ############

=>  If there some elements of an array are float, and remaining are integers, then Creation of array upcasts all
       elements into float.

    >>>  np.array([1, 5, 8, 12, 45, 75, 2.0, 3.0, 4.0])

    >>>  array([ 1.,  5.,  8., 12., 45., 75.,  2.,  3.,  4.])
```

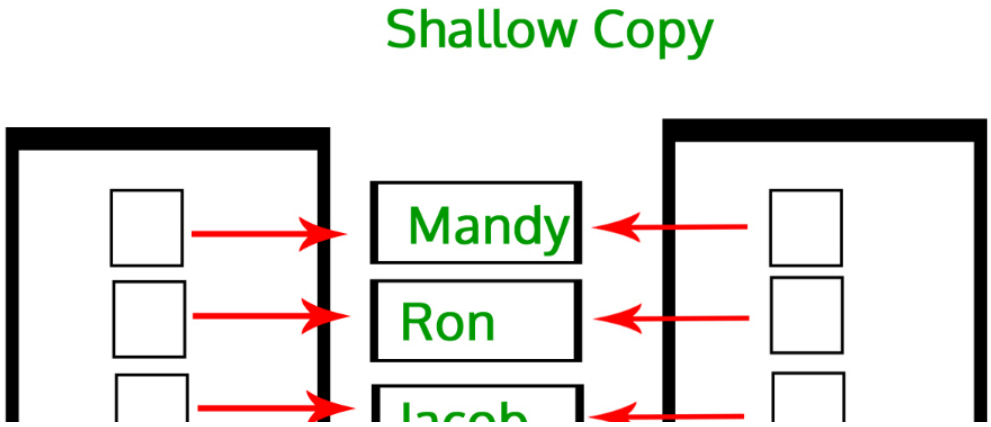
In [ ]:
```
    ###########    ##########    Deep Copy vs shallow Copy    ############    ############
```

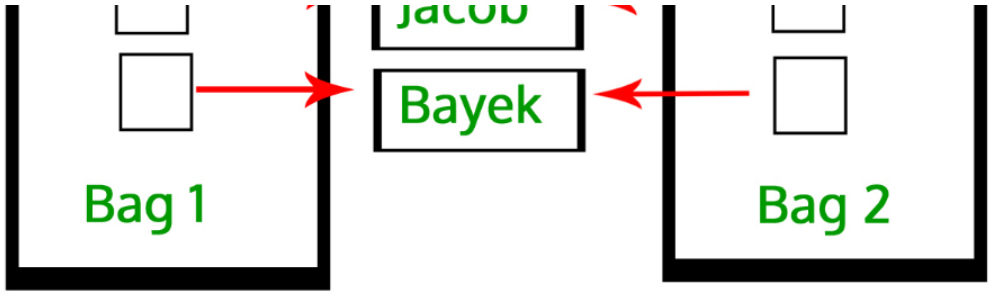
In [ ]:
```
=>  Using '=' operator only creates a new variable that shares the reference of the original object.

=>  In python, There are 2 different types of copies.

        *  Shallow copy            *  Deep copy

=>  'copy' modules is used to making these copies and need to bwe imported .

=>  copy.copy(x) is used to make a Shallow copy of an object.

=>  copy.deepcopy(x) is used to makes a Deep copy of an object.
```

In [ ]:
```
          ###########    ##########    Deep Copy    ############    ############

=>  Deep copy is a process in which the copying process occurs recursively.

=>  It means first constructing a new collection object and then recursively populating it with copies
      of the child objects found in the original.

=>  In case of deep copy, a copy of object is copied in other object.

=>  It means that any changes made to a copy of object do not reflect in the original object.
```



In [ ]:
```
       ###########    ##########    Shallow Copy    ############    ############

=>  A shallow copy means constructing a new collection object and then populating it with references to
      the child objects found in the original.

=>  The copying process does not recurse and therefore won't create copies of the child objects themselves.

=>  In case of shallow copy, a reference of object is copied in other object.

=>  It means that any changes made to a copy of object do reflect in the original object.
```





In [ ]:

Deep Copy vs shallow copy

In [ ]:
```
       ############    ##########   Matrix from a Numpy array   ############     #############

 =>  Matrix is a datatype in Numpy package. and a sub-class or subset of 'array' super class.

 =>  np.mat(data, dtype=None)  ->  This is used to return a matrix from a Numpy array.

       *  data ->  we can pass a (arrray-like) sequence as data.

 =>  np.asmatrix(data, dtype=None)  ->  This is similar to matrix but doesnot returns a matrix,
                                          if data is alreday a matrix
```

In [ ]:
```
     ############     ###########   Numpy arrays from Functions    ############     ##############

 =>  np.fromfunction(function, shape, dtype, **kwargs)  ->  This generates an array from the given function.

 =>  dtype = By-default 'float'
```

In [3]:
```
#  Python code to show the generation of arrrays from a function

import numpy as np

arr = np.fromfunction(lambda x,y :  x==y, (3,3))

arr
```

Out[3]:
```
array([[ True, False, False],
       [False,  True, False],
       [False, False,  True]])
```

In [ ]:
```
v[0][0]
```

In [ ]:
```
v.shape
```

In [1]:
```
np.array([4,5,6,7],ndmin= 20)
```

```
---------------------------------------------------------------
NameError                 Traceback (most recent call last)
<ipython-input-1-3663c7809970> in <module>
----> 1 np.array([4,5,6,7],ndmin= 20)

NameError: name 'np' is not defined
```

In [ ]:
```
np.array([4,5,6,7] , dtype = complex)
```

In [ ]:
```
np.array([(3,4) , (5,6)])
```

In [ ]:
```
n = np.array([(3,4) , (5,6)],dtype = [('a' , '<i4'),('b' , '<i2')])
```

In [ ]:
```
type(n[0][0])
```

In [ ]:
```
type(n[0][1])
```

In [ ]:
```
m = np.mat([1,2,3,4])
m
```

In [ ]:
```
np.array([5,6,7,7])
```

In [ ]:
```
np.array(m)
```

In [ ]:
```
np.asanyarray([5,6,7,78])
```

In [ ]:
```
np.asanyarray(m)
```

In [ ]:
```
np.asanyarray()
```

In [ ]:

In [ ]:
```
a = np.array([4,5,6,7,8])
```

In [ ]:
```
b = a
```

In [ ]:
```
b
```

In [ ]:
```
b[0] = 535
```

In [ ]:
```
b
```

In [ ]:
```
a
```

In [ ]:
```
c = np.copy(a)
```

In [ ]:
```
c
```

In [ ]:
```
a
```

In [ ]:
```
c[0] = 343
```

In [ ]:
```
c
```

In [ ]:
```
a
```

In [ ]:
```
a[0] = 435354345
a
```

In [ ]:
```
c
```

In [ ]:
```
np.fromfunction(lambda x,y : x==y , (3,4))
```

In [ ]:
```
np.fromfunction(lambda x,y : x - y*6 , (3,4))
```

In [ ]:
```
np.fromstring('6,7', dtype = int ,sep= ',')
```

In [ ]:
```
a = [[3,4,5] ,[6,7,8] ,[6,7,9]]
```

In [ ]:
```
n = np.array(a)
```

In [ ]:
```
n
```

In [ ]:
```
n.ndim
```

In [ ]:
```
n.size
```

In [ ]:
```
n.shape
```

In [ ]:
```
n.dtype
```

In [ ]:
```
m = np.array([])
```

In [ ]:
```
m.ndim
```

In [ ]:
```
range(7.,5.,.0)
```

In [ ]:
```
 n = np.arange(2.5 ,9.87 , .5)
```

In [ ]:
```
list(n)
```

In [ ]:
```
np.linspace(3,7.9, axis = 0 )
```

In [ ]:
```
np.zeros((2,5,6,2))
```

In [ ]:
```
np.ones((2,3,4)) +1
```

In [ ]:
```
np.empty((2,4))
```

In [ ]:
```
np.eye(4)
```

In [ ]:
```
np.logspace(5.6 , 7.8 ,6 ,base=2)
```

In [ ]:
```
m = np.random.rand(4,5)
```

In [ ]:
```
m
```

In [ ]:
```
m.reshape(5,4)
```

In [ ]:
```
m.reshape(3,10)
```

In [ ]:
```
np.random.randn(5,6)
```

In [ ]:
```
a = np.random.randint(3,8 , (5,6))
```

In [ ]:
```
a
```

In [ ]:
```
a[0:2,0:2]
```

In [ ]:
```
a[a>5]
```

In [ ]:
```
a[0][0]=90
```

In [ ]:
```
a
```

In [ ]:
```
m = np.array([[3,4,5],[5,6,7]])
```

In [ ]:
```
m
```

In [ ]:
```
 n= np.array([[67,4,5],[5,0,5]])
```

In [ ]:
```
n
```

In [ ]:
```
m+n
```

In [ ]:
```
m-nm =
```

In [ ]:
```
m*n
```

In [ ]:
```
m1 = np.array([[0,1],[2,3]])
```

In [ ]:
```
m1
```

In [ ]:
```
m2 = np.array([[2,3],[1,8]])
```

In [ ]:
```
m2
```

In [ ]:
```
m1*m2
```

In [ ]:
```
m1@m2
```

In [ ]:
```
m1
```

In [ ]:
```
m1**2
```

In [ ]:
```
pow(m1,2)
```

In [ ]:
```
m = np.zeros((3,4)).reshape((1,2,-1))
```

In [ ]:
```
np.linspace(3,7,.45)
```

In [ ]:
```
m+2
```

In [ ]:
```
n1  = np.array([1,3,4,2])
```

In [ ]:
```
n1
```

In [ ]:
```
m
```

In [ ]:
```
m+n1
```

In [ ]:
```
n2 = np.array([[3,4,5]])
```

In [ ]:
```
n2.shape
```

In [ ]:
```
n2.ndim
```

In [ ]:
```
m = np.random.rand(4,5).reshape(10,2)
```

In [ ]:
```
m
```

In [ ]:
```
m1 = np.random.rand(4,5).reshape(10,2)
```

In [ ]:
```
m1
```

In [ ]:
```
np.sqrt(m1)
```

In [ ]:
```
np.exp(m1)
```

In [ ]:
```
np.log10(m1)
```

In [ ]:
```
m1
```

In [ ]:
```
m
```

In [ ]:
```
np.fmod(m,m1)
```

In [ ]:
```
n = np.array([[3,4,5],[1,2,3],[3,6,7]],ndmin = 10 )
```

In [ ]:
```
n
```

In [ ]:
```
n.shape
```

In [ ]:
```
n1 = np.array([[3,4,5],[1,2,3],[3,6,7]])
```

In [ ]:
```
n1.reshape(3,1,3)
```

In [ ]:
```
n1.shape
```

In [ ]:
```
n1.ndim
```

In [ ]:
```

```