Creating Numpy Array

1. **using np.array() //1-D and 2-D**
2. **using np.zeros/onces/empty/random()**
3. **using np.arange()**
4. **using np.linspace()**
5. **using Copy()**
6. **using Identity()**

In [2]:

```
import numpy as np
```

In [3]:

```
list=[1,2,3,4,5]
print(list)
type(list)
```

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

Out[3]:

```
list
```

In [4]:

```
# 1-D
array = np.array([1,2,3,4,5])
print(array)
print("Array Shape : ", array.shape)
type(array)
# [1 2 3 4 5] : Vectors
```

```
[1 2 3 4 5]
Array Shape :  (5,)
```

Out[4]:

```
numpy.ndarray
```

In [5]:

```
# 2-D
array2 = np.array([[1,2,3],[4,5,6]])
print(array2)
print("Array Shape : ", array2.shape)
array2
#[[1 2 3]
# [4 5 6]] : matrices
```

```
[[1 2 3]
 [4 5 6]]
Array Shape :  (2, 3)
```

Out[5]:

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

In [6]:

```
array_2d= np.array([1,2,3,4,5],ndmin=2)
print(array_2d)
print("Array Shape : ",array_2d.shape)
```

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

```
Array Shape :  (1, 5)
```

np.zeros

In [7]:

```
np.zeros(5)
```

Out[7]:

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

In [8]:

```
np.zeros((5,4))
```

Out[8]:

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

In [9]:

```
np.zeros((5,4),dtype=int)
```

Out[9]:

```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

np.onces

In [10]:

```
np.ones((5,10))
```

Out[10]:

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

In [11]:

```
np.ones((2,4),dtype=complex)
```

Out[11]:

```
array([[1.+0.j, 1.+0.j, 1.+0.j, 1.+0.j],
       [1.+0.j, 1.+0.j, 1.+0.j, 1.+0.j]])
```

np.identity()

In [12]:

```
np.identity(5)
```

Out[12]:

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

```
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [13]:

```
np.identity(5,dtype=int)
```

Out[13]:

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

np.arange() ----------

In [14]:

```
np.arange(5)
```

Out[14]:

```
array([0, 1, 2, 3, 4])
```

In [15]:

```
np.arange(5,20)
```

Out[15]:

```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

In [16]:

```
np.arange(5,20,5)
```

Out[16]:

```
array([ 5, 10, 15])
```

np.linspace()

In [17]:

```
np.linspace(1,2,50)
```

Out[17]:

```
array([1.        , 1.02040816, 1.04081633, 1.06122449, 1.08163265,
       1.10204082, 1.12244898, 1.14285714, 1.16326531, 1.18367347,
       1.20408163, 1.2244898 , 1.24489796, 1.26530612, 1.28571429,
       1.30612245, 1.32653061, 1.34693878, 1.36734694, 1.3877551 ,
       1.40816327, 1.42857143, 1.44897959, 1.46938776, 1.48979592,
       1.51020408, 1.53061224, 1.55102041, 1.57142857, 1.59183673,
       1.6122449 , 1.63265306, 1.65306122, 1.67346939, 1.69387755,
       1.71428571, 1.73469388, 1.75510204, 1.7755102 , 1.79591837,
       1.81632653, 1.83673469, 1.85714286, 1.87755102, 1.89795918,
       1.91836735, 1.93877551, 1.95918367, 1.97959184, 2.        ])
```

In [18]:

```
np.linspace(1,10)
```

Out[18]:

```
array([ 1.        ,  1.18367347,  1.36734694,  1.55102041,  1.73469388,
        1.91836735,  2.10204082,  2.28571429,  2.46938776,  2.65306122,
        2.83673469,  3.02040816,  3.20408163,  3.3877551 ,  3.57142857,
```

```
       3.75510204,  3.93877551,  4.12244898,  4.30612245,  4.48979592,
       4.67346939,  4.85714286,  5.04081633,  5.2244898 ,  5.40816327,
       5.59183673,  5.7755102 ,  5.95918367,  6.14285714,  6.32653061,
       6.51020408,  6.69387755,  6.87755102,  7.06122449,  7.24489796,
       7.42857143,  7.6122449 ,  7.79591837,  7.97959184,  8.16326531,
       8.34693878,  8.53061224,  8.71428571,  8.89795918,  9.08163265,
       9.26530612,  9.44897959,  9.63265306,  9.81632653, 10.        ])
```

In [19]:

```
np.linspace(1,10,10)
```

Out[19]:

```
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10.])
```

Copy()

In [20]:

```
array2
```

Out[20]:

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

In [21]:

```
array_3 = array2.copy()
```

In [22]:

```
array_3
```

Out[22]:

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

np.empty()

In [23]:

```
np.empty((2,5))
```

Out[23]:

```
array([[ 1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10.]])
```

In [24]:

```
np.empty((2,10),dtype=int)
```

Out[24]:

```
array([[94497381540176,              0,              0,              0,
                     0,              0,              0,              0,
                     0,              0],
       [             0,              0,              0,              0,
                     0,              0,              0,              0,
                     0,              0]])
```

np.random()

In [25]:

```
np.random(default_rng())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
/tmp/ipykernel_3932/235175187.py in <module>
----> 1 np.random(default_rng())

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

In [ ]:

```

```

Properties and Attributes

1. **shape**
2. **ndim**
3. **size**
4. **itemsize**
5. **dtype**
6. **astype()**

ndim : Give the dim

In [26]:

```
print(array)
print(array.shape)
print("Dim is:", array.ndim)
```

```
[1 2 3 4 5]
(5,)
Dim is: 1
```

In [27]:

```
print(array2)
print()
print(array2.shape)
print("Dim is:", array2.ndim)
```

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

(2, 3)
Dim is: 2
```

In [28]:

```
array_3d =  np.array([[[1,2],[3,4],[5,6]]])
print(array_3d)
array_3d
```

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

Out[28]:

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

In [29]:

```
array_3d.shape
```

Out[29]:

Out[29]:

```
(1, 3, 2)
```

In [30]:

```
print("Dim is:", array_3d.ndim)
```

```
Dim is: 3
```

size

In [31]:

```
array2
```

Out[31]:

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

In [32]:

```
array2.size
```

Out[32]:

```
6
```

Itemsize : Give based upon data-types

In [33]:

```
array2.itemsize
```

Out[33]:

```
8
```

In [34]:

```
array.itemsize
```

Out[34]:

```
8
```

In [35]:

```
array_3d.itemsize
```

Out[35]:

```
8
```

In [36]:

```
a=np.array([1,2,3,4,5],dtype=float)
print(a)
```

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

In [37]:

```
a.itemsize
```

Out[37]:

```
8
```

In [38]:

```
b=np.array([1,2,3,4,5],dtype=complex)
print(b)
```

```
[1.+0.j 2.+0.j 3.+0.j 4.+0.j 5.+0.j]
```

In [39]:

```
b.itemsize
```

Out[39]:

```
16
```

dtype

In [40]:

```
array
```

Out[40]:

```
array([1, 2, 3, 4, 5])
```

In [41]:

```
array.dtype
```

Out[41]:

```
dtype('int64')
```

In [ ]:

```

```

astype : Convert pre-created one array data-type to another data-type array.astype("float")

In [42]:

```
print(array.dtype)
```

```
int64
```

In [43]:

```
array
```

Out[43]:

```
array([1, 2, 3, 4, 5])
```

In [44]:

```
array.astype("float")
```

Out[44]:

```
array([1., 2., 3., 4., 5.])
```

In [45]:

```
array.astype("float").dtype
```

Out[45]:

```
dtype('float64')
```

List Vs Numpy Array

1. Faster
2. Convenient
3. Less Memory

Less Memory

In [46]:

```
import sys
```

In [47]:

```
lista=range(100)
```

In [48]:

```
len(lista)
```

Out[48]:

```
100
```

In [49]:

```
narray = np.arange(100)
```

In [50]:

```
len(narray)
```

Out[50]:

```
100
```

In [51]:

```
# 88: one element size * total element size == 2800 bytes
print(sys.getsizeof(88)*len(lista))
```

```
2800
```

In [52]:

```
print(narray.itemsize*narray.size)
```

```
800
```

In [53]:

```
print("Same thing is happen but memory difference is : ",sys.getsizeof(88)*len(lista)-nar
ray.itemsize*narray.size,"bytes")
```

```
Same thing is happen but memory difference is :  2000 bytes
```

Faster

In [54]:

```
import time
```

In [55]:

```
time.time()
```

Out[55]:

Out[55]:

```
1638780427.0125968
```

In [59]:

```
x=range(1000000)
y=range(1000000,2000000)

start_time=time.time()

z=[(x+y) for x,y in zip(x,y)]

time.time()-start_time
```

Out[59]:

```
0.1595139503479004
```

In [60]:

```
a = np.arange(1000000)
b = np.arange(1000000,2000000)

start_time=time.time()

c = a+b

time.time()-start_time
```

Out[60]:

```
0.005863189697265625
```

In [61]:

```
t1=np.arange(5)
t2=np.arange(0,5)
print(t2+t1)
```

```
[0 2 4 6 8]
```

In [62]:

```
l1=range(5)
l2=range(0,5)
l3=[(l1+l2) for (l1,l2) in zip(l1,l2)]
print(l3)
```

```
[0, 2, 4, 6, 8]
```

In [ ]:

```

```

Indexing, Slicing and Iteration

reshape()

In [64]:

```
arr1 = np.arange(24).reshape(6,4)
```

In [67]:

```
arr1
```

Out[67]:

```
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]])
```

In [68]:

```
arr1[3:5,2:]
```

Out[68]:

```
array([[14, 15],
       [18, 19]])
```

In [72]:

```
arr1[1]
```

Out[72]:

```
array([4, 5, 6, 7])
```

In [102]:

```
arr1[:,1]
```

Out[102]:

```
array([ 1,  5,  9, 13, 17, 21])
```

In [73]:

```
np.arange(16)
```

Out[73]:

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

In [81]:

```
arr2 = np.arange(16).reshape(2,8)
```

In [82]:

```
arr2
```

Out[82]:

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

In [87]:

```
arr2[:,4:6]
```

Out[87]:

```
array([[ 4,  5],
       [12, 13]])
```

In [90]:

```
arr1
```

Out[90]:

```
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]])
```

In [91]:

```
# iterate over array
```

In [96]:

```
for i in arr1:
    print(i)
```

```
[0 1 2 3]
[4 5 6 7]
[ 8  9 10 11]
[12 13 14 15]
[16 17 18 19]
[20 21 22 23]
```

Iteration

In [100]:

```
for i in np.nditer(arr1):
    print(i)
```

```
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 Operations

1. **Basic Operations #### a. A-B #### b. B*2 #### c. A.dot(B)**
2. **Unary Opeartions #### a. A.min() #### b. A.sum()**
3. **Universal Function #### a. np.exp(B) #### b. np.aqrt() #### c. np.sin(A)**
4. **Convertion Types #### a. arr1.astype("int")**

In [103]:

```
arr3 =np.array([1,2,3,4,5,6,7,8,9,10])
arr4 =np.array([11,12,13,14,15,16,17,18,19,20])
```

In [104]:

```
arr3+arr4
```

Out[104]:

```
array([12, 14, 16, 18, 20, 22, 24, 26, 28, 30])
```

In [105]:

```
# Vector Multiplicatiom
arr3*arr4
```

Out[105]:

```
array([ 11,  24,  39,  56,  75,  96, 119, 144, 171, 200])
```

In [106]:

```
# Scaler Multiplication
arr3*5
```

Out[106]:

```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

In [111]:

```
arr3>5
```

Out[111]:

```
array([False, False, False, False, False,  True,  True,  True,  True,
        True])
```

Dot

In [128]:

```
arr5 = np.arange(6,12).reshape(3,2)
arr6 = np.arange(6,18,2).reshape(2,3)
```

In [129]:

```
arr5
```

Out[129]:

```
array([[ 6,  7],
       [ 8,  9],
       [10, 11]])
```

In [130]:

```
arr6
```

Out[130]:

```
array([[ 6,  8, 10],
       [12, 14, 16]])
```

In [131]:

```
arr5.dot(arr6)
```

Out[131]:

```
array([[120, 146, 172],
       [156, 190, 224],
       [192, 234, 276]])
```

In [132]:

```
arr6
```

Out[132]:

```
array([[ 6,  8, 10],
       [12, 14, 16]])
```

In [133]:

```
arr6.max()
```

Out[133]:

```
16
```

In [134]:

```
arr6.min()
```

Out[134]:

```
6
```

In [135]:

```
arr6.mean()
```

Out[135]:

```
11.0
```

In [137]:

```
arr6
```

Out[137]:

```
array([[ 6,  8, 10],
       [12, 14, 16]])
```

In [139]:

```
arr6.max(axis=1)
```

Out[139]:

```
array([10, 16])
```

In [140]:

```
arr6.min(axis=0)
```

Out[140]:

```
array([ 6,  8, 10])
```

In [141]:

```
arr6.sum()
```

Out[141]:

```
66
```

In [149]:

```
arr6.sum(axis=1)
```

Out[149]:

```
array([24, 42])
```

Universal Function

In [150]:

```
np.sin(arr6)
```

Out[150]:

```
array([[-0.2794155 ,  0.98935825, -0.54402111],
       [-0.53657292,  0.99060736, -0.28790332]])
```

In [151]:

```
np.log(arr6)
```

Out[151]:

```
array([[1.79175947, 2.07944154, 2.30258509],
       [2.48490665, 2.63905733, 2.77258872]])
```

In [152]:

```
np.median(arr6)
```

Out[152]:

```
11.0
```

Reshaping Numpy Array

1. **Revel**
2. **Reshape**
3. **Transpose**
4. **Stacking**
5. **Spliting**

Revel : Convert High Dim To 1-D

In [154]:

```
arr1
```

Out[154]:

```
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]])
```

In [156]:

```
arr1.ndim
```

Out[156]:

```
2
```

In [161]:

```
# Convert 2-D To 1-D
arr1.ravel()
```

Out[161]:

```
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])
```

In [163]:

```
arr1.ravel().ndim
```

Out[163]:

```
1
```

Transpose

In [170]:

```
arr2
```

Out[170]:

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

In [168]:

```
arr2.transpose()
```

Out[168]:

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

Stacking : Adding two row of column of an array

In [171]:

```
arr7= np.arange(6,12).reshape(3,2)
arr8= np.arange(12,18).reshape(3,2)
```

In [179]:

```
print(arr7.shape)
arr7
```

```
(3, 2)
```

Out[179]:

```
array([[ 6,  7],
       [ 8,  9],
       [10, 11]])
```

In [181]:

```
print(arr8.shape)
arr8
```

```
(3, 2)
```

Out[181]:

```
array([[12, 13],
       [14, 15],
       [16, 17]])
```

In [177]:

```
# arr7+arr8
```

In [182]:

```
print("shape is : ",np.hstack((arr7,arr8)).shape)
np.hstack((arr7,arr8))
```

```
shape is :  (3, 4)
```

Out[182]:

```
array([[ 6,  7, 12, 13],
       [ 8,  9, 14, 15],
       [10, 11, 16, 17]])
```

In [183]:

```
print("shape is : ",np.vstack((arr7,arr8)).shape)
np.vstack((arr7,arr8))
```

```
shape is :  (6, 2)
```

Out[183]:

```
array([[ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15],
       [16, 17]])
```

In [185]:

```
np.concatenate((arr7,arr8))
```

Out[185]:

```
array([[ 6,  7],
       [ 8,  9],
       [10, 11],
       [12, 13],
       [14, 15],
       [16, 17]])
```

In [186]:

```
arr8
```

Out[186]:

```
array([[12, 13],
       [14, 15],
       [16, 17]])
```

In [188]:

```
np.hsplit(arr8,2)
```

Out[188]:

```
[array([[12],
        [14],
        [16]]),
 array([[13],
        [15],
        [17]])]
```

In [189]:

```
np.vsplit(arr8,3)
```

Out[189]:

```
[array([[12, 13]]), array([[14, 15]]), array([[16, 17]])]
```

Fancy indexing

In [197]:

```
arr8
```

Out[197]:

```
array([[12, 13],
       [14, 15],
       [16, 17]])
```

In [198]:

```
arr8[[0]]
```

Out[198]:

```
array([[12, 13]])
```

In [209]:

```
arr8[[1,2]]
```

Out[209]:

```
array([[14, 15],
       [16, 17]])
```

Random Array generate

In [269]:

```
arr9 = np.random.randint(low=1,high=100,size=10).reshape(5,2)
```

In [270]:

```
arr9
```

Out[270]:

```
array([[86, 79],
       [37, 75],
       [10, 12],
       [16, 12],
       [70,  3]])
```

In [271]:

```
arr9>50
```

Out[271]:

```
array([[ True,  True],
       [False,  True],
       [False, False],
       [False, False],
       [ True, False]])
```

Indexing using boolean array

In [273]:

```
arr9[arr9>50]
```

Out[273]:

```
array([86, 79, 75, 70])
```

In [285]:

```
# greater the 50 and odd do equal to zero
arr9[(arr9>50) & (arr9%2!=0)] = 0
```

In [286]:

```
arr9
```

Out[286]:

```
array([[86,  0],
       [37,  0],
       [10, 12],
       [16, 12],
       [70,  3]])
```

Graph Plot

In [308]:

```
x = np.linspace(-50,50,50)
```

In [309]:

```
x.size
```

Out[309]:

```
50
```

In [310]:

```
y=np.sin(x)
```

In [311]:

```
y.size
```

Out[311]:

```
50
```

In [312]:

```
import matplotlib.pyplot as plt
%matplotlib inline  # This is similar to getch in c language
```

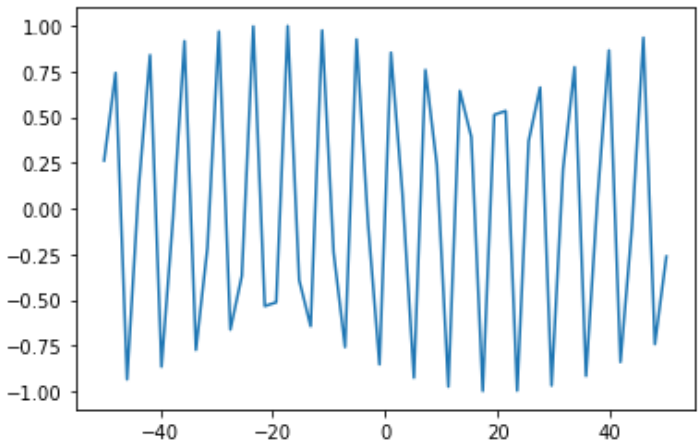
```
UsageError: unrecognized arguments: # This is similar to getch in c language
```

In [313]:

```
plt.plot(x,y)
```

Out[313]:

```
[<matplotlib.lines.Line2D at 0x7f891221d9a0>]
```



In [314]:

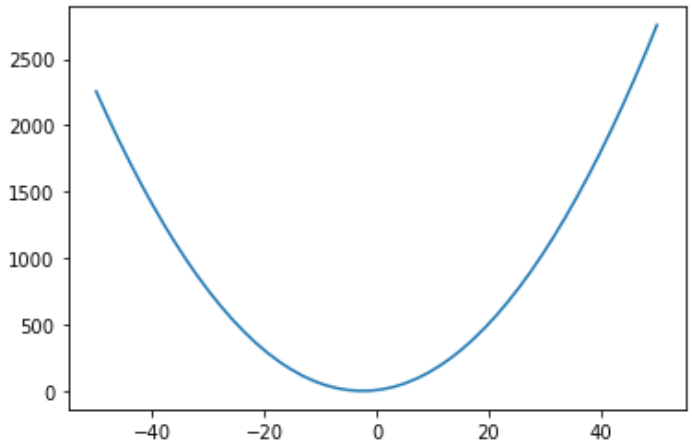
```
# quedetric equation
y=x*x+5*x+5
```

In [322]:

```
plt.plot(x,y)
```

Out[322]:

```
[<matplotlib.lines.Line2D at 0x7f8911f37a00>]
```



Broadcasting

if the dimensions of two array are dissimilar, elements-to-elemnts operations are not possible. However operations on arrays of non-similar shapes is still possible in Numpy, Because of the broadcasting capability.

In [323]:

```
x1 = np.arange(6).reshape(3,2)
y1 = np.arange(6,12).reshape(3,2)
```

In [324]:

```
x1+y1
```

Out[324]:

```
array([[ 6,  8],
       [10, 12],
       [14, 16]])
```

In [329]:

```
x2 = np.arange(2).reshape(1,2)
y2 = np.arange(6).reshape(3,2)
```

In [330]:

```
x2+y2
```

Out[330]:

```
array([[0, 2],
       [2, 4],
       [4, 6]])
```

Important Function In Numpy

In [347]:

```
np.random.random()
```

Out[347]:

```
0.5156712377826581
```

In [370]:

```
# Random values generate in float in range 0.0-0.1
np.random.random(10)
```

Out[370]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

In [363]:

```
# Random Values in integer
np.random.randint(1,100,10)
```

Out[363]:

```
array([13, 73, 10, 76,  6, 80, 65, 17,  2, 77])
```

In [371]:

```
np.random.seed(1)
np.random.random()
```

Out[371]:

```
0.417022004702574
```

In [372]:

```
# Random values in float
np.random.uniform(1,100,10)
```

Out[372]:

```
array([72.31212485,  1.01132311, 30.93092469, 15.52883319, 10.14152088,
       19.43976093, 35.21051198, 40.27997995, 54.34285667, 42.50025693])
```

In [375]:

```
np.random.uniform(1,100,10).reshape(5,2)
```

Out[375]:

```
array([[42.68965488, 95.83106348],
       [53.78336321, 69.49583428],
       [32.23604747, 68.96359184],
       [83.62794152,  2.81053946],
       [75.26428718, 98.8972478 ]])
```

In [399]:

```
array1 = np.random.randint(1,10,6)
```

In [400]:

```
array1
```

Out[400]:

```
array([6, 8, 1, 4, 2, 5])
```

In [401]:

```
array1.max()
```

Out[401]:

```
8
```

In [404]:

```
# To find max element index
array1.argmax()
```

Out[404]:

```
1
```

In [414]:

```
array5 = np.random.randint(1,10,6)
```

In [415]:

```
array5
```

Out[415]:

```
array([9, 2, 2, 9, 8, 1])
```

In [421]:

```
# Replace odd with -1
array5[(array5%2!=0)]=-1
```

In [422]:

```
array5
```

Out[422]:

```
array([-1,  2,  2, -1,  8, -1])
```

In [423]:

```
# where
```

In [428]:

```
array6 = np.random.randint(1,50,6)
```

In [442]:

```
array6
```

Out[442]:

```
array([11, 19, 29, 29, 31, 32])
```

In [443]:

```
np.where(array6%2!=0,-1,array6)
```

Out[443]:

```
array([-1, -1, -1, -1, -1, 32])
```

In [448]:

```
np.sort(array6)
```

Out[448]:

```
array([11, 19, 29, 29, 31, 32])
```

In [ ]:

```

```