In [1]:
```python
import numpy as np
print(np.__version__)
```
```
1.20.1
```
In [ ]:
```python
#  print(np.info(np.add))  if will give information how to add the function
```

Array creation

In [49]:
```python
import numpy as np

list1 = [1, 2, 3, 4, 5, 6]
list2 = [10, 9, 8, 7, 6, 5]
a1 = np.array(list1)
a2 = np.array(list2)
print(a1*a2)
```
```
[10 18 24 28 30 30]
```
In [2]:
```python
d = np.full((3, 3), 6, dtype = 'complex')
print(d)
```
```
[[6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]
 [6.+0.j 6.+0.j 6.+0.j]]
```
In [3]:
```python
x= np.arange(2,11).reshape(3,3) # reshape the vale 3*3 matrix
print(x)
```
```
[[ 2  3  4]
 [ 5  6  7]
 [ 8  9 10]]
```
In [46]:
```python
array = np.arange(8).reshape(2, 2, 2)
print(array)
```
```
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```
In [44]:
```python
c = np.empty([3, 3])
print("\nMatrix c : \n", c)
```
```
Matrix c :
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```
In [8]:
```python
a = np.arange(10,1,-2)
print(a)
newarr = a[np.array([3, 1, 2 ,4])]   # give the index value
print(newarr)
```
```
[10  8  6  4  2]
[4 8 6 2]
```

indexing

In [9]:
```python
x = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
arr = x[np.array([1, 3, -3])]        # give the index value
print(arr)
```
```
[2 4 7]
```
In [11]:
```python
a = np.arange(20)
print(a[-8:17:1])
```
```
[12 13 14 15 16]
```
In [28]:
```python
a = np.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],
              [24, 25, 26, 27, 28, 29],
              [30, 31, 32, 33, 34, 35]])
print(a)
print("1st", a[0, 3:5])     # 0 is row and 3:5 is (3 to 4)column, get value is both intersect
print("2nd", a[4:, 4:])     # 4 to end row & 4 to end column
print("3rd", a[:, 2])       # 0 to end row & 2 is cloum
print("4th", a[2::2, ::2])
```
```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
1st [3 4]
2nd [[28 29]
 [34 35]]
3rd [ 2  8 14 20 26 32]
4th [[12 14 16]
 [24 26 28]]
```
In [31]:
```python
b = np.array([[[1, 2, 3],[4, 5, 6]],
              [[7, 8, 9],[10, 11, 12]]])
print(b.ndim)
print(b[...,1])
```
```
3
[[ 2  5]
 [ 8 11]]
```
In [37]:
```python
a = np.array([[1 ,2 ],[3 ,4 ],[5 ,6 ]])

print(a[[0 ,1 ,2 ],[0 ,0 ,1]])  # this case value will select (0,0) (1,0) (2,1)
```
```
[1 3 6]
```
In [35]:
```python
a = np.array([[0 ,1 ,2],[3 ,4 ,5 ],
              [6 ,7 ,8],[9 ,10 ,11]])
print(a)
print()
print(a[1:2 ,1:3 ])    # 1 row & 1 to 3 column intersect
print(a[1:2 ,[1,2]])
```
```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

[[4 5]]
[[4 5]]
```
In [38]:
```python
# Boolean Array Indexing:
a = np.array([10, 40, 80, 50, 100])
print(a[a>50])
```
```
[ 80 100]
```
In [39]:
```python
a = np.array([10, 40, 80, 50, 100])
print(a[a%40==0]**2)
```
```
[1600 6400]
```
In [40]:
```python
b = np.array([[5, 5],[4, 5],[16, 4]])
sumrow = b.sum(-1)
print(b[sumrow%10==0])
```
```
[[ 5  5]
 [16  4]]
```

data type object

In [50]:
```python
print(np.dtype(np.int16))
```
```
int16
```
In [65]:
```python
a = np.zeros(5)
a.dtype
```
Out[65]:
```
dtype('float64')
```
In [68]:
```python
b = np.zeros(6,dtype=np.int8)
b
```
Out[68]:
```
array([0, 0, 0, 0, 0, 0], dtype=int8)
```
In [69]:
```python
c = np.zeros(6,dtype = int)
c
```
Out[69]:
```
array([0, 0, 0, 0, 0, 0])
```
In [71]:
```python
x = np.array([1,2,3], dtype = 'f')
x
```
Out[71]:
```
array([1., 2., 3.], dtype=float32)
```
In [74]:
```python
a = np.zeros(3)
a                 # by default it takes float
```
Out[74]:
```
array([0., 0., 0.])
```
In [75]:
```python
a.astype(int)     #we changethe already taken the default value
```
Out[75]:
```
array([0, 0, 0])
```
In [58]:
```python
import numpy as np
a = np.array([2])
print(type(a))  # type is:
print(a.dtype)  # dtype is:
```
```
<class 'numpy.ndarray'>
int32
```
In [62]:
```python
dt = np.dtype([('name', np.unicode_, 16), ('grades', np.float64, 2)])
print(dt['grades'])
print(dt['name'])
```
```
('<f8', (2,))
<U16
```
In [63]:
```python
x = np.array([('Sarah', (8.0, 7.0)), ('John', (6.0, 7.0))], dtype=dt)   # dt = data type
print(x[1])

print("Grades of John are: ", x[1]['grades'])
print("Names are: ", x['name'])
```
```
('John', [6., 7.])
Grades of John are:  [6. 7.]
Names are:  ['Sarah' 'John']
```

Iterating

In [7]:
```python
import numpy as np
a = np.arange(12).reshape(3,4)
print(a)

for row in a:
    for cell in row:
        print(cell,end=" ")
```
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
0 1 2 3 4 5 6 7 8 9 10 11
```
In [20]:
```python
b = np.array([[1,2,3], [4,5,6], [7,8,9]])
b.flatten()
```
Out[20]:
```
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```
In [22]:
```python
for x in np.nditer(b,order='c'):     # C order
    print(x, end=" ")
```
```
1 2 3 4 5 6 7 8 9
```
In [23]:
```python
for x in np.nditer(b,order='F'):     # Fortran order
    print(x, end=" ")
```
```
1 4 7 2 5 8 3 6 9
```
In [24]:
```python
for x in np.nditer(b,order='F',flags=['external_loop']):
    print(x)
```
```
[1 4 7]
[2 5 8]
[3 6 9]
```
In [40]:
```python
d = np.arange(12).reshape(3,4)
print(d)
```
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
```
In [53]:
```python
for x in np.nditer(d, op_flags = ['readwrite']):
    x[...] = 5*x
print(d)
```
```
[[     0  62500 125000 187500]
 [250000 312500 375000 437500]
 [500000 562500 625000 687500]]
```
In [59]:
```python
c = np.arange(12).reshape(3,4)
e = np.array([5, 6, 7, 8], dtype = int)
print(c)
print(e)
```
```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]
[5 6 7 8]
```
In [57]:
```python
for x,y in np.nditer([c,e]):
    print(x,y)
```
```
0 5
1 6
2 7
3 8
4 5
5 6
6 7
7 8
8 5
9 6
10 7
11 8
```

Bitwise operator & math

In [60]:
```python
in_num1 = 10
in_num2 = 11

print ("Input  number1 : ", in_num1)
print ("Input  number2 : ", in_num2)

out_num = np.bitwise_and(in_num1, in_num2)
print (out_num)
```
```
Input  number1 :  10
Input  number2 :  11
10
```
In [69]:
```python
in_arr1 = [2, 8, 125]
in_arr2 = [3, 3, 115]

out_arr = np.bitwise_or(in_arr1, in_arr2)
out_arr1 = np.bitwise_xor(in_arr1, in_arr2)
out_arr2 = np.bitwise_and(in_arr1, in_arr2)
print (out_arr2)      # Output array after bitwise_and:
print(out_arr)    # Output array after bitwise_or:
print(out_arr1)   # Output array after bitwise_xor:
```
```
[  2   0 113]
[  3  11 127]
[ 1 11 14]
```
In [68]:
```python
in_arr = [2, 0, 25]
out_arr = np.invert(in_arr)
print (out_arr)
```
```
[ -3  -1 -26]
```
In [73]:
```python
in_num = 5
bit_shift = 3

out_num = np.left_shift(in_num, bit_shift)
print (out_num)               #  5*(2^2) get
```
```
40
```
In [72]:
```python
in_arr = [2, 8, 15]
bit_shift =[3, 4, 5]

out_arr = np.left_shift(in_arr, bit_shift)
print (out_arr)          # Output array after left shifting:
```
```
[ 16 128 480]
```
In [76]:
```python
import numpy as np
import math

in_array = [0, math.pi / 2, np.pi / 3, np.pi]
print ("Input array : \n", in_array)

cos_Values = np.cos(in_array)
print (cos_Values)  #  Cosine values :

Sin_Values = np.sin(in_array)
print(Sin_Values)    # sine value
```
```
Input array :
 [0, 1.5707963267948966, 1.0471975511965976, 3.141592653589793]
[ 1.000000e+00  6.123234e-17  5.000000e-01 -1.000000e+00]
[0.00000000e+00 1.00000000e+00 8.66025404e-01 1.22464680e-16]
```
In [78]:
```python
in_array = [.5, 1.5, 2.5, 3.5, 4.5, 10.1]

round_off_values = np.round_(in_array)
print (round_off_values)      # Rounded values :
```
```
[ 0.  2.  2.  4.  4. 10.]
```
In [82]:
```python
in_array = [1, 3, 5]
out_array = np.exp(in_array)
print (out_array)  # exponential value

out_array = np.log(in_array)
print(out_array)
```
```
[  2.71828183  20.08553692 148.4131591 ]
[0.         1.09861229 1.60943791]
```
In [83]:
```python
in_num = 2.0
out_num = np.reciprocal(in_num)
print ("Output number : ", out_num)
```
```
Output number :  0.5
```

String

In [87]:
```python
import numpy as np

print(np.char.lower(['GEEKS', 'FOR']))
print(np.char.split('geeks, for, geeks', sep = ','))
print(np.char.join('-', 'geeks'))
print(np.char.join(['-', ':'], ['geeks', 'for']))
```
```
['geeks' 'for']
['geeks', ' for', ' geeks']
g-e-e-k-s
['g-e-e-k-s' 'f:o:r']
```
In [3]:
```python
import numpy as np
a = np.array(['geeks', 'for', 'geeks'])

print(np.char.count(a,'geek'))  # get the value then print 1 otherwise 0
print(np.char.count(a, 'fo'))
```
```
[1 0 1]
[0 1 0]
```
In [4]:
```python
import numpy as np

a=np.array(['geeks', 'for', 'geeks'])

# counting a substring
print(np.char.rfind(a,'geek'))  #  If not found then it returns -1
print(np.char.rfind(a, 'fo'))
```
```
[ 0 -1  0]
[-1  0 -1]
```
In [6]:
```python
print(np.char.isnumeric('12'))
print(np.char.isnumeric('12geeks'))
```
```
True
False
```
In [13]:
```python
a=np.char.equal('geeks','for') # comparision both string
print(a)

# b = np.char.unequal('geeks','for')
# print(b)
a=np.char.greater('geeks','for')

print(a)
```
```
False
True
```

Linear Algebra

In [45]:
```python
a = [[2,3], [4,5]]
b = [[2,3], [1,3]]
result = np.dot(a, b)
print(result)
```
```
[[ 7 15]
 [13 27]]
```
In [46]:
```python
a = [[2,3], [4,5]]
b = [[2,3], [1,3]]
result2 = np.outer(a, b)
print(result2)
```
```
[[ 4  6  2  6]
 [ 6  9  3  9]
 [ 8 12  4 12]
 [10 15  5 15]]
```
In [47]:
```python
p = [[2,3], [4,5]]
q = [[2,3], [1,3]]
result1 = np.cross(p, q)
print(result1)
```
```
[0 7]
```
In [49]:
```python
a = np.array([1,2,3])
b = np.array([0,1,0])
result = np.kron(a, b)
print("Kronecker product  of the said arrays:")
print(result)
```
```
Kronecker product  of the said arrays:
[0 1 0 0 2 0 0 3 0]
```
In [14]:
```python
a = np.array([[1, -2j], [2j, 5]])

print("Array is :",a)
```
```
Array is : [[ 1.+0.j -0.-2.j]
 [ 0.+2.j  5.+0.j]]
```
In [16]:
```python
import numpy as np

A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])
 print("\nTrace of A:", np.trace(A))
```
```
Trace of A: 11
```
In [17]:
```python
A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])

print(("\nDeterminant of A:", np.linalg.det(A)))
```
```
('\nDeterminant of A:', -306.0)
```

Sorting, searching, counting

In [19]:
```python
a = np.array([[12, 15], [10, 1]])
print(a)
arr1 = np.sort(a, axis = 0)    # every column sort individually (Lower to Upper)
print ("Along first axis : \n", arr1)
```
```
[[12 15]
 [10  1]]
Along first axis :
 [[10  1]
 [12 15]]
```
In [23]:
```python
a = np.array([[10, 15], [12, 1]])
print(a)
arr2 = np.sort(a, axis = -1)
print("\nAlong first axis : \n", arr2)
```
```
[[10 15]
 [12  1]]

Along first axis :
 [[10 15]
 [ 1 12]]
```
In [22]:
```python
a = np.array([[12, 15], [10, 1]])
print(a)
arr1 = np.sort(a, axis = None)
print("\nAlong none axis : \n", arr1)
```
```
[[12 15]
 [10  1]]

Along none axis :
 [ 1 10 12 15]
```
In [24]:
```python
# Not completed
```

numpy |random. randint() function

In [27]:
```python
# rand()       # uniformly distributed  values
# randint()    # uniformly distributed integer values
# ranf()       # uniformly distributed integer values
#randn()       # normally distributed  values
```
In [31]:
```python
Z = np.random.rand(5)
print(Z)
```
```
[0.27680007 0.88326205 0.59534184 0.41446208 0.51584913]
```
In [32]:
```python
arr1 = np.random.rand(2,3)
print(arr1)
```
```
[[0.65176244 0.17289039 0.66776259]
 [0.15723444 0.18434748 0.56969789]]
```
In [28]:
```python
A = np.round(10*np.random.rand(2,3))
print(A)
```
```
[[5. 3. 8.]
 [0. 3. 4.]]
```
In [29]:
```python
import numpy as np
A = np.random.permutation(np.arange(10))
print(A)
```
```
[9 0 6 3 4 2 5 1 8 7]
```
In [27]:
```python
import numpy as np

#numpy.random.randint(low, high=None, size=None, dtype='l')

out_arr = np.random.randint(low = 0, high = 3, size = 5)
print ("Output 1D Array filled with random integers : ", out_arr)
```
```
Output 1D Array filled with random integers :  [1 0 1 2 0]
```
In [ ]:
```python

```
In [29]:
```python
out_arr = np.random.randint(low = 4, size =(2, 3))  # 2 row 3 colum
print ("Output 2D Array filled with random integers : ", out_arr)
```
```
Output 2D Array filled with random integers :  [[1 2 3]
 [1 3 1]]
```
In [30]:
```python
out_arr = geek.random.randint(2, 10, (2, 3, 4))  # low = 2, high = 10, matrix= 2,3,4
print ("Output 3D Array filled with random integers : ", out_arr)
```
```
Output 3D Array filled with random integers :  [[[4 5 5 4]
  [8 6 9 9]
  [6 4 9 6]]

 [[7 4 9 2]
  [3 3 6 7]
  [5 9 9 7]]]
```

Random sampling in numpy | random_sample() function

In [ ]:
```python
# numpy.random.random_sample() is one of the function for doing
# random sampling in numpy. It returns an array of specified shape and
# fills it with random floats in the half-open interval [0.0, 1.0).
```
In [8]:
```python
out_val = np.random.random_sample()
print ("Output random float value : ", out_val)
```
```
Output random float value :  0.9387280557142557
```
In [10]:
```python
out_arr = np.random.random_sample(size =(1, 3))
print ("Output 2D Array filled with random floats : ", out_arr)
```
```
Output 2D Array filled with random floats :  [[0.64501426 0.25308157 0.20693803]]
```
In [11]:
```python
out_arr = np.random.random_sample((3, 2, 1))
print ("Output 3D Array filled with random floats : ", out_arr)
```
```
Output 3D Array filled with random floats :  [[[0.25710716]
  [0.19422148]]

 [[0.96387745]
  [0.59005443]]

 [[0.2337291 ]
  [0.11847503]]]
```

Random sampling in numpy | ranf() function

In [ ]:
```python
# numpy.random.ranf() is one of the function for doing random
# sampling in numpy. It returns an array of specified shape and
# fills it with random floats in the half-open interval [0.0, 1.0).
```
In [13]:
```python
#numpy.random.ranf(size=None)

out_val = np.random.ranf()
print ("Output random float value : ", out_val)
```
```
Output random float value :  0.9875587409371372
```
In [14]:
```python
out_arr = np.random.ranf(size =(2, 1))
print ("Output 2D Array filled with random floats : ", out_arr)
```
```
Output 2D Array filled with random floats :  [[0.74422996]
 [0.96595141]]
```
In [16]:
```python
out_arr = np.random.ranf((3, 3, 2))
print ("Output 3D Array filled with random floats : ", out_arr)
```
```
Output 3D Array filled with random floats :  [[[0.03098791 0.05576051]
  [0.57836732 0.1970554 ]
  [0.95666585 0.12014111]]

 [[0.60550149 0.37175724]
  [0.63132599 0.94975108]
  [0.37040909 0.7173222 ]]

 [[0.65481222 0.50169867]
  [0.0491669  0.98636306]
  [0.92909105 0.690225  ]]]
```

Random sampling in numpy | random_integers() function

In [ ]:
```python
# numpy.random.random_integers() is one of the function for doing random
# sampling in numpy. It returns an array of specified shape and fills it with
# random integers from low (inclusive) to high (exclusive), i.e. in the interval [low, high).

# Syntax : numpy.random.random_integers(low, high=None, size=None)
```
In [22]:
```python
import numpy as np
out_arr = np.random.random_integers(low = 0, high = 5, size = 4)
print ("Output 1D Array filled with random integers : ", out_arr)
```
```
Output 1D Array filled with random integers :  [5 3 5 5]
<ipython-input-22-981ed3fc5b89>:2: DeprecationWarning: This function is deprecated. Please call randint(0, 5 + 1) instead
  out_arr = np.random.random_integers(low = 0, high = 5, size = 4)
```
In [23]:
```python
out_arr = np.random.random_integers(low = 3, size =(3, 3))
print ("Output 2D Array filled with random integers : ", out_arr)
```
```
Output 2D Array filled with random integers :  [[1 2 2]
 [3 1 2]
 [1 2 2]]
<ipython-input-23-332c808dbc83>:1: DeprecationWarning: This function is deprecated. Please call randint(1, 3 + 1) instead
  out_arr = np.random.random_integers(low = 3, size =(3, 3))
```
In [24]:
```python
out_arr = np.random.random_integers(1, 6, (2, 2, 3))
print ("Output 3D Array filled with random integers : ", out_arr)
```
```
Output 3D Array filled with random integers :  [[[2 6 2]
  [6 6 6]]

 [[4 2 5]
  [6 6 5]]]
<ipython-input-24-03f512aa788a>:1: DeprecationWarning: This function is deprecated. Please call randint(1, 6 + 1) instead
  out_arr = np.random.random_integers(1, 6, (2, 2, 3))
```

Universal Function

In [36]:
```python
arr = np.array([0, 30, 45, 60, 90, 180])
radians = np.deg2rad(arr)   # conver radian into degree
#degree = np.rad2deg(arr)
print(arr)
```
```
[  0  30  45  60  90 180]
```
In [37]:
```python
sine_value = np.sin(radians)
print(sine_value)
```
```
[0.00000000e+00 5.00000000e-01 7.07106781e-01 8.66025404e-01
 1.00000000e+00 1.22464680e-16]
```
In [40]:
```python
weight = np.array([50.7, 52.5, 50, 58, 55.63, 73.25, 49.5, 45])
print(np.amin(weight), np.amax(weight))
```
```
45.0 73.25
```
In [42]:
```python
# percentile
print('Weight below which 70 % student fall: ')
print(np.percentile(weight, 70))

print('Mean weight of the students: ')
print(np.mean(weight))

# median
print('Median weight of the students: ')
print(np.median(weight))

# standard deviation
print('Standard deviation of weight of the students: ')
print(np.std(weight))
```
```
Weight below which 70 % student fall:
55.317
Mean weight of the students:
54.3225
Median weight of the students:
51.6
Standard deviation of weight of the students:
8.052773978574091
```
In [43]:
```python
even = np.array([0, 2, 4, 6, 8, 16, 32])
odd = np.array([1, 3, 5, 7, 9, 17, 33])

# bitwise_and
print('bitwise_and of two arrays: ')
print(np.bitwise_and(even, odd))

# bitwise_or
print('bitwise_or of two arrays: ')
print(np.bitwise_or(even, odd))

# bitwise_xor
print('bitwise_xor of two arrays: ')
print(np.bitwise_xor(even, odd))

# invert or not
print('inversion of even no. array: ')
print(np.invert(even))

# left_shift
print('left_shift of even no. array: ')
print(np.left_shift(even, 1))

# right_shift
print('right_shift of even no. array: ')
print(np.right_shift(even, 1))
```
```
bitwise_and of two arrays:
[ 0  2  4  6  8 16 32]
bitwise_or of two arrays:
[ 1  3  5  7  9 17 33]
bitwise_xor of two arrays:
[1 1 1 1 1 1 1]
inversion of even no. array:
[ -1  -3  -5  -7  -9 -17 -33]
left_shift of even no. array:
[ 0  4  8 12 16 32 64]
right_shift of even no. array:
[ 0  1  2  3  4  8 16]
```

End

In [ ]:
```python

```