Linspace()

```
- used for creating sequence of numbers
```

```
linspace(start,stop,num,retstep=True/False)
when retstep is True it returns evenly spaced values
by default value for num is 50
```

In [1]:
```
import numpy as np
arr = np.linspace(0,1)
arr
```

Out[1]:
```
array([0.        , 0.02040816, 0.04081633, 0.06122449, 0.08163265,
       0.10204082, 0.12244898, 0.14285714, 0.16326531, 0.18367347,
       0.20408163, 0.2244898 , 0.24489796, 0.26530612, 0.28571429,
       0.30612245, 0.32653061, 0.34693878, 0.36734694, 0.3877551 ,
       0.40816327, 0.42857143, 0.44897959, 0.46938776, 0.48979592,
       0.51020408, 0.53061224, 0.55102041, 0.57142857, 0.59183673,
       0.6122449 , 0.63265306, 0.65306122, 0.67346939, 0.69387755,
       0.71428571, 0.73469388, 0.75510204, 0.7755102 , 0.79591837,
       0.81632653, 0.83673469, 0.85714286, 0.87755102, 0.89795918,
       0.91836735, 0.93877551, 0.95918367, 0.97959184, 1.        ])
```

In [11]:
```
arr = np.linspace(1,10,10,dtype='int',retstep='True')
arr
```

Out[11]:
```
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10]), 1.0)
```

In [9]:
```
np.arange(1,10)
```

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

In [10]:
```
np.arange(0,1,0.02040816)
```

In [17]:
```
dict1 = {'amount':np.ones(6,dtype='int')*np.arange(100,700,100),
        'intrest':np.linspace(3.5,12.5,6)[::-1],
        'month':np.linspace(6,36,6)}
import pandas as pd
df = pd.DataFrame(dict1)
df['Principla amount']=((df['amount']*df['intrest'])/100)+df['amount']
df
```

Out[17]:

	amount	intrest	month	Principla amount
0	100	12.5	6.0	112.5
1	200	10.7	12.0	221.4
2	300	8.9	18.0	326.7
3	400	7.1	24.0	428.4
4	500	5.3	30.0	526.5
5	600	3.5	36.0	621.0

Genration of Random numbers

In [18]:
```
import random
x = random.random()
x
```

Out[18]: 0.5011604787984737

In [20]:
```
x = random.randint(2,10)
x
```

Out[20]: 9

```
in numpy there are similar random no. generation functions:
1. rand()
2. randf()
3. random()
4. randint()
5. randn()
```

rand()

```
it will genrate random values b/w 0-1
```

In [21]:
```
rand_no = np.random.rand(1)
rand_no
```

Out[21]: array([0.25524239])

In [23]:
```
1 rand_no = np.random.rand(5)
2 rand_no
```

Out[23]:
```
array([0.24792151, 0.46937577, 0.77748655, 0.11106505, 0.15805265])
```

In [25]:
```
1 rand_no = np.random.rand(18).reshape(2,3,3)
2 rand_no
```

Out[25]:
```
array([[[3.82976045e-01, 9.12418218e-02, 9.32313305e-01],
        [2.94830523e-01, 8.92371124e-01, 3.17500554e-01],
        [5.88299411e-04, 3.71308983e-01, 3.12422957e-02]],

       [[5.64691653e-01, 8.30844964e-01, 8.64968224e-01],
        [1.46492108e-01, 2.73652907e-01, 2.29478257e-01],
        [9.76220287e-01, 3.20992440e-01, 6.26292005e-01]]])
```

In [26]:
```
1 rand_no = np.random.rand(2,3,3)
2 rand_no
```

Out[26]:
```
array([[[0.54400785, 0.19630573, 0.20854064],
        [0.51912287, 0.88820739, 0.11673352],
        [0.22579109, 0.9084854 , 0.8656821 ]],

       [[0.20776188, 0.54127004, 0.91287548],
        [0.57869077, 0.35534917, 0.78092241],
        [0.17510996, 0.34973913, 0.15015993]]])
```

Random

```
1 # random takes only 1 positional arguement
2 # it gives values b/1 0-1
```

In [27]:
```
1 np.random.random()
```

Out[27]:
```
0.5271224578175167
```

In [28]:
```
1 np.random.random(10)
```

Out[28]:
```
array([0.01482659, 0.01720794, 0.4510251 , 0.59788859, 0.86556057,
       0.07677139, 0.44505082, 0.10369318, 0.9090535 , 0.34424089])
```

In [29]:
```
1 np.random.random(2,3)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-29-80b6e3664a11> in <module>
----> 1 np.random.random(2,3)

mtrand.pyx in numpy.random.mtrand.RandomState.random()

TypeError: random() takes at most 1 positional argument (2 given)
```

In [30]:
```
np.random.random(6).reshape(2,3)
```

Out[30]:
```
array([[0.21102673, 0.09660731, 0.52809704],
       [0.35391781, 0.35888053, 0.52064731]])
```

Ranf()

```
# It takes only 1 positional arguement
# it gives values b/1 0-1
```

In [31]:
```
np.random.ranf()
```

Out[31]:
```
0.11145681054494283
```

In [32]:
```
np.random.ranf(10)
```

Out[32]:
```
array([0.60627295, 0.01180789, 0.57785077, 0.32418727, 0.96564298,
       0.17809661, 0.96717409, 0.86300325, 0.87899447, 0.22817299])
```

In [33]:
```
np.random.ranf(12).reshape(2,3,2)
```

Out[33]:
```
array([[[0.16315813, 0.01418154],
        [0.40124441, 0.39822791],
        [0.14042011, 0.11631344]],

       [[0.31757219, 0.73801284],
        [0.14409891, 0.0453212 ],
        [0.53378024, 0.31543574]]])
```

In [34]:
```
np.random.ranf(2,3,2)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-34-7f0aeb5c4733> in <module>
----> 1 np.random.ranf(2,3,2)

mtrand.pyx in numpy.random.mtrand.ranf()

mtrand.pyx in numpy.random.mtrand.RandomState.random_sample()

TypeError: random_sample() takes at most 1 positional argument (3 given)
```

randn

```
It will generate normally disctributed values around (0,0) cordinate
```

In [36]:
```
np.random.randn()
```

Out[36]:
```
-0.4317950215008157
```

In [38]:
```
np.random.randn(5)
```

Out[38]:
```
array([-4.38511970e-04, -8.82984516e-01, -3.29678784e-01,  1.89645270e+00,
        1.75050598e+00])
```

In [39]:
```
np.random.randn(5,4)
```

Out[39]:
```
array([[-0.46368472, -0.72715155,  0.94362846, -0.17535447],
       [-0.33449028, -0.31634258, -0.59433391, -1.19770638],
       [ 0.10907424, -1.40362446,  0.28531412, -0.26545481],
       [ 0.97370794, -2.21128089,  0.39092081,  0.53957696],
       [-1.50108131,  0.63405549, -1.93140128,  0.38401849]])
```

randint()

```
generates random numbers
Syntax:
    randint(low,high,size,dtype)
```

In [54]:
```
np.random.randint(10)             # 10 is considered as high & default value of low is 0.
```

Out[54]:
```
9
```

In [62]:
```
np.random.randint(0,10)           # default value for size is 1
```

Out[62]:
```
2
```

In [50]:
```
np.random.randint(low=0,high=10,size=5)
```

Out[50]:
```
array([7, 3, 1, 8, 7])
```

In [46]:
```
np.random.randint(937,1000,12)
```

Out[46]:
```
array([995, 957, 946, 952, 984, 977, 975, 937, 975, 978, 975, 950])
```

In [60]:
```
np.random.randint(-100,-37,12).reshape(3,4)
```

Out[60]:
```
array([[-38, -75, -50, -91],
       [-82, -75, -99, -92],
       [-67, -38, -55, -82]])
```

sort()

Ascending()

In [66]:
```
ran_nums = np.random.randint(10,100,12)
print(ran_nums)
sorted_nums = sorted(ran_nums)
sorted_nums
```

```
[49 66 58 90 40 90 39 75 23 39 77 99]
```

Out[66]:
```
[23, 39, 39, 40, 49, 58, 66, 75, 77, 90, 90, 99]
```

In [67]:
```
print(ran_nums)
np.sort(ran_nums)
```

```
[49 66 58 90 40 90 39 75 23 39 77 99]
```

Out[67]:
```
array([23, 39, 39, 40, 49, 58, 66, 75, 77, 90, 90, 99])
```

descending

In [64]:
```
sorted_nums = sorted(ran_nums)[::-1]
sorted_nums
```

Out[64]:
```
[98, 87, 72, 55, 53, 47, 42, 19, 16, 13, 12, 12]
```

In [65]:
```
sorted_nums = sorted(ran_nums,reverse=True)
sorted_nums
```

Out[65]:
```
[98, 91, 59, 58, 56, 55, 42, 25, 20, 16, 14, 13]
```

In [76]:
```
print(ran_nums)
np.flip(np.sort(ran_nums))     # flip reverses the order of the array
```

```
[49 66 58 90 40 90 39 75 23 39 77 99]
```

Out[76]:
```
array([99, 90, 90, 77, 75, 66, 58, 49, 40, 39, 39, 23])
```

Argmax()

```
returns the index of maximum value in the array
```

In [82]:
```
ran_nums
```

Out[82]:
```
array([49, 66, 58, 90, 40, 90, 39, 75, 23, 39, 77, 99])
```

In [83]:
```
np.argmax(ran_nums)
```

Out[83]:
```
11
```

Argmin()

```
returns the index of minimum value in the array
```

In [84]:
```
np.argmin(ran_nums)
```

Out[84]: 8

Argsort()

```
# return the indexes for the sorted values in the array
```

In [135]:
```
import numpy as np
arr = np.array([9,5,3,1,7,8,4])
np.argsort(arr)
```

Out[135]: array([3, 2, 6, 1, 4, 5, 0], dtype=int64)

In [136]:
```
arr[np.argsort(arr)]
```

Out[136]: array([1, 3, 4, 5, 7, 8, 9])

sorting on Matrix

row-wise sorting > vertical sorting

In [86]:
```
arr = np.array([[49, 66, 58], [90, 40, 90], [39, 75, 23], [39, 77, 99]])
arr
```

Out[86]:
```
array([[49, 66, 58],
       [90, 40, 90],
       [39, 75, 23],
       [39, 77, 99]])
```

In [91]:
```
np.sort(arr,axis=0)              # axis = 0 is for sorting along rows, vertical sorting
```

Out[91]:
```
array([[39, 40, 23],
       [39, 66, 58],
       [49, 75, 90],
       [90, 77, 99]])
```

In [100]:
```
np.flip(np.sort(arr,axis=0))      # Descesnding sorting for axis = 0
```

Out[100]:
```
array([[99, 77, 90],
       [90, 75, 49],
       [58, 66, 39],
       [23, 40, 39]])
```

column-wise sorting > Horizaontal sorting

In [92]:
```
arr = np.array([[49, 66, 58], [90, 40, 90], [39, 75, 23], [39, 77, 99]])
arr
```

Out[92]:
```
array([[49, 66, 58],
       [90, 40, 90],
       [39, 75, 23],
       [39, 77, 99]])
```

In [93]:
```
np.sort(arr,axis=1)              # axis = 1 is for sorting along columns, horizontal sorting
```

Out[93]:
```
array([[49, 58, 66],
       [40, 90, 90],
       [23, 39, 75],
       [39, 77, 99]])
```

In [101]:
```
np.flip(np.sort(arr,axis=1))      # Descesnding sorting for axis = 1
```

Out[101]:
```
array([[99, 77, 39],
       [75, 39, 23],
       [90, 90, 40],
       [66, 58, 49]])
```

sorting along both rows & columns

In [94]:
```
matrix = np.array([[49, 66, 58], [90, 40, 90], [39, 75, 23], [39, 77, 99]])
matrix
```

Out[94]:
```
array([[49, 66, 58],
       [90, 40, 90],
       [39, 75, 23],
       [39, 77, 99]])
```

In [95]:
```
np.sort(np.sort(matrix,axis=0),axis=1)
```

Out[95]:
```
array([[23, 39, 40],
       [39, 58, 66],
       [49, 75, 90],
       [77, 90, 99]])
```

In [97]:
```
total_sort = np.sort(np.sort(matrix,axis=1),axis=0)
total_sort
```

Out[97]:
```
array([[23, 39, 66],
       [39, 58, 75],
       [40, 77, 90],
       [49, 90, 99]])
```

In [98]:
```
 np.flip(total_sort)        # descending sorting
```

Out[98]:
```
array([[99, 90, 49],
       [90, 77, 40],
       [75, 58, 39],
       [66, 39, 23]])
```

replacing any item in array

In [102]:
```
arr = np.array([2,6,9,3,5,0])
arr[1]=100
arr
```

Out[102]:
```
array([  2, 100,   9,   3,   5,   0])
```

In [105]:
```
matrix = np.array([[49, 66, 58], [90, 40, 90], [39, 75, 23], [39, 77, 99]])
print(matrix)
matrix[2][0]=100
matrix
```

```
[[49 66 58]
 [90 40 90]
 [39 75 23]
 [39 77 99]]
```

Out[105]:
```
array([[ 49,  66,  58],
       [ 90,  40,  90],
       [100,  75,  23],
       [ 39,  77,  99]])
```

In [106]:
```
matrix[2,0]=1000
matrix
```

Out[106]:
```
array([[  49,   66,   58],
       [  90,   40,   90],
       [1000,   75,   23],
       [  39,   77,   99]])
```

In [110]:
```
matrix = np.array([[[49, 66, 58], [90, 40, 90]], [[39, 75, 23], [39, 77, 99]]])
matrix[0,1,2]=100
matrix
```

Out[110]:
```
array([[[ 49,  66,  58],
        [ 90,  40, 100]],

       [[ 39,  75,  23],
        [ 39,  77,  99]]])
```

replacing slice from the array

In [118]:
```
matrix = np.array([[[49, 66, 58], [90, 40, 90]], [[39, 75, 23], [39, 77, 99]]])
matrix[:1,:,1:]=[[11,22],[33,44]]
matrix
```

Out[118]:
```
array([[[49, 11, 22],
        [90, 33, 44]],

       [[39, 75, 23],
        [39, 77, 99]]])
```

In [116]:
```
matrix[:1,:,1:]=[[[10, 20],
                  [30, 40]]]
matrix
```

Out[116]:
```
array([[[49, 10, 20],
        [90, 30, 40]],

       [[39, 75, 23],
        [39, 77, 99]]])
```

inserting item into array()

```
# insert item at any specific index in an array
```

In [ ]:
```
Syntax:
    np.insert(array,index,value)
```

In [122]:
```
arr = np.array([2,6,9,3,5,0])
arr = np.insert(arr,1,100)
arr
```

Out[122]:
```
array([  2, 100,   6,   9,   3,   5,   0])
```

In [134]:
```
# matrix = np.array([[49, 66, 58], [90, 40, 90], [39, 75, 23], [39, 77, 99]])
# print(matrix)
# matrix = np.insert(matrix,3,[100,200,300])
# len(matrix)
# matrix.reshape(5,3)
```

In [ ]:
```

```