toPython- 200521

May 21, 2020

1 Topics:

Recursive Programing
Lists and an Intruduction to numpy Arrays
Plot some arrays

2 Recursive Programing

For recursive furmulas

2.1 Factorial

$5! = 5 * 4! = 5 * 4 * 3! = ...$

[1]:
```
def factorial (n):
    if n== 1 or n==0:
        return 1
    else:
        out= n*factorial(n-1)
        return out

for n in range (7):
    print (factorial  (n))
```

```
1
1
2
6
24
120
720
```

2.2 Fibonacci sequence

[2]:
```
def Fibonacci(n):
    if n<0:
        print("Incorrect input")
    # First Fibonacci number is 1
    elif n==1 or n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)


FibList= []
for n in range (1, 20):
    FibList.append(Fibonacci(n))

print(FibList)
```

```
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181]
```

3 Lists and Arrays

3.1 Python List

[3]:
```
L1= [1, 2]
print (L1)
print (L1*5)
```

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

[4]:
```
type (L1)
```

[4]:
```
list
```

3.2 Numpy Arrays

[5]:
```
import numpy as np
```

[6]:
```
a1= np.array(L1)
a1*3
```

[6]:
```
array([3, 6])
```

[7]:
```
type (a1)
```

[7]:
```
numpy.ndarray
```

[8]:
```
a1.append (1)
```

```
 ␣
↪---------------------------------------------------------------------------

    AttributeError                            Traceback (most recent call␣
↪last)

    <ipython-input-8-37475cee8308> in <module>
  ----> 1 a1.append (1)


    AttributeError: 'numpy.ndarray' object has no attribute 'append'
```

[9]:
```
# np.append (a1, [], axis= )
```

[10]:
```
T= np.linspace (0, 20, 20, endpoint= False)
T
```

[10]:
```
array([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11., 12.,
       13., 14., 15., 16., 17., 18., 19.])
```

[11]:
```
a2= np.linspace (5.5, 17.3, 30)
a2
```

[11]:
```
array([ 5.5       ,  5.90689655,  6.3137931 ,  6.72068966,  7.12758621,
        7.53448276,  7.94137931,  8.34827586,  8.75517241,  9.16206897,
        9.56896552,  9.97586207, 10.38275862, 10.78965517, 11.19655172,
       11.60344828, 12.01034483, 12.41724138, 12.82413793, 13.23103448,
       13.63793103, 14.04482759, 14.45172414, 14.85862069, 15.26551724,
       15.67241379, 16.07931034, 16.4862069 , 16.89310345, 17.3       ])
```

[12]:
```
np.arange (5,20)
```

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

3.3 The projectile motion problem for different fire angles

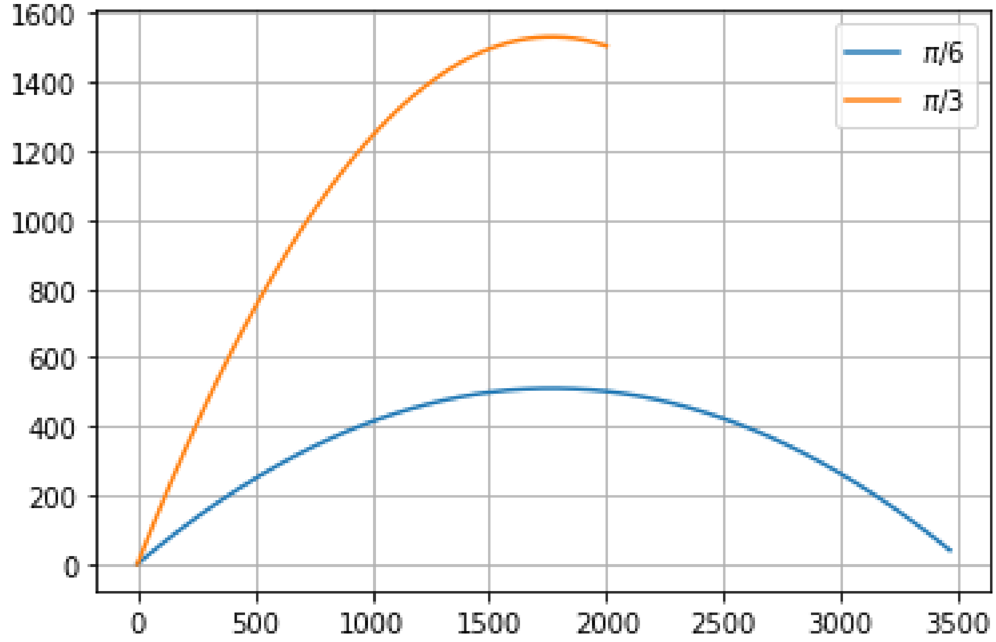
[13]:
```
from numpy import sin, cos, pi, exp
import matplotlib.pyplot as plt
```

[14]:
```
def r(t, theta):
    v0= 200
    v0x= v0*cos (theta)
    v0y= v0*sin (theta)
    a= -9.8
    y0= 0.1
    x= v0x*t
```

```python
    y= 0.5*a*t**2 + v0y*t + y0
    return [x, y]

x, y= r(10, pi/3)
print (x)
print (y)
```

```
1000.0000000000002
1242.150807568877
```

[15]:
```python
N= 200          # Number of time steps
tLim= 20        # total time in seconds
T= np.linspace (0, tLim, N)

theta1= pi/6
theta2= pi/3

X1= np.zeros (N)
Y1= np.zeros (N)
X2= np.zeros (N)
Y2= np.zeros (N)

for i in range (len (T)):
    X1[i], Y1[i]= r(T[i], theta1)
    X2[i], Y2[i]= r(T[i], theta2)

plt.plot (X1, Y1, label= '$\pi/6$')
plt.plot (X2, Y2, label= '$\pi/3$')
plt.legend ()

plt.grid ()
plt.show ()
```



3.4 More about numpy arrays

[16]:
```
pi* np.eye (5)
```

[16]:
```
array([[3.14159265, 0.        , 0.        , 0.        , 0.        ],
       [0.        , 3.14159265, 0.        , 0.        , 0.        ],
       [0.        , 0.        , 3.14159265, 0.        , 0.        ],
       [0.        , 0.        , 0.        , 3.14159265, 0.        ],
       [0.        , 0.        , 0.        , 0.        , 3.14159265]])
```

[17]:
```
print (np.zeros (5))
print (np.zeros ([5,5]))
```

```
[0. 0. 0. 0. 0.]
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

[18]:
```
a5=np.ones (3)
a6= np.array ([a5])
print(a5)
print(a6)
np.shape (a6)
```

```
[1. 1. 1.]
[[1. 1. 1.]]
```

[18]:
```
(1, 3)
```

[19]:
```
a4= np.ones ([2,3,4])
print (a4)
np.shape (a4)
```

```
[[[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]

 [[1. 1. 1. 1.]
  [1. 1. 1. 1.]
  [1. 1. 1. 1.]]]
```

[19]:
```
(2, 3, 4)
```

[20]:
```
np.random.random ()
```

[20]:
```
0.41566409616770117
```

[21]:
```
np.random.random ([5,5])
```

[21]:
```
array([[0.17585609, 0.65077008, 0.98652707, 0.47027098, 0.93997942],
       [0.13931773, 0.53441043, 0.27371766, 0.26109756, 0.10959959],
       [0.11818547, 0.25323387, 0.57839836, 0.82063839, 0.25345605],
       [0.64490134, 0.45701519, 0.95287655, 0.44507076, 0.82233405],
       [0.56219659, 0.61200035, 0.13466123, 0.80476123, 0.00449203]])
```

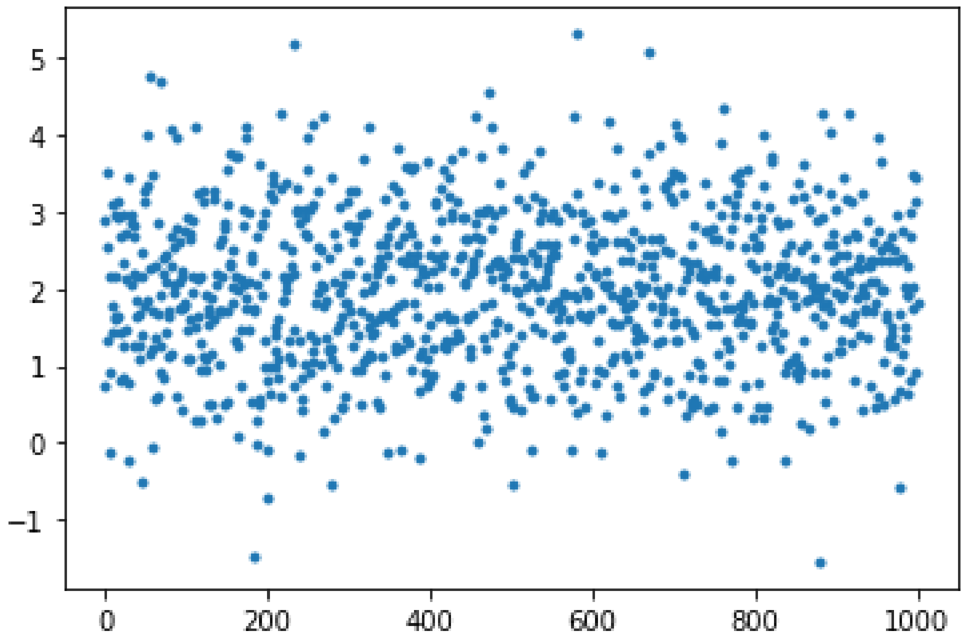
[22]:
```
np.random.randint (10, 15)
```

[22]:
```
10
```

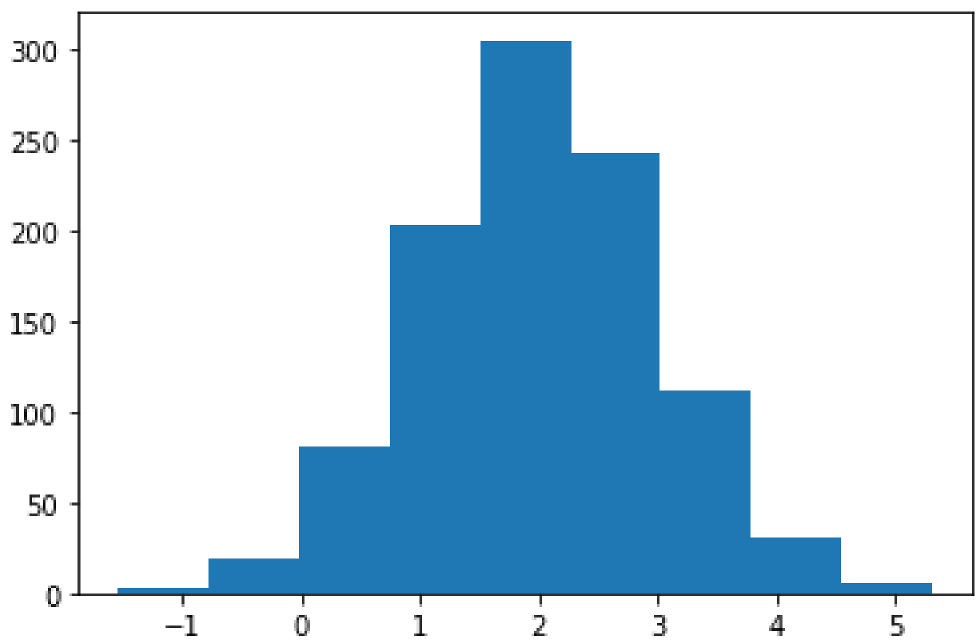
[23]:
```
np.random.randint (10, 15, 100)
```

[23]:
```
array([14, 13, 12, 14, 11, 14, 11, 12, 13, 12, 12, 13, 10, 13, 12, 12, 14,
       14, 14, 11, 14, 12, 14, 13, 14, 13, 11, 11, 10, 12, 10, 14, 14, 10,
       14, 14, 10, 11, 10, 12, 13, 14, 14, 14, 14, 12, 10, 13, 13, 14, 14,
       12, 12, 11, 12, 10, 11, 10, 14, 12, 14, 12, 10, 11, 10, 10, 12, 11,
       13, 14, 10, 14, 12, 11, 14, 10, 10, 12, 12, 12, 11, 13, 13, 14, 12,
       12, 11, 12, 11, 14, 13, 11, 14, 13, 13, 11, 11, 13, 13, 14])
```

[24]:
```
y= np.random.normal (2, 1, 1000)
plt.plot (y, '.')
plt.show ()
```



[25]:
```python
# histogram of normal distribution y, with 9 blocks!
plt.hist (y, 9);
```



[26]:
```python
help (np.random.randint)
```

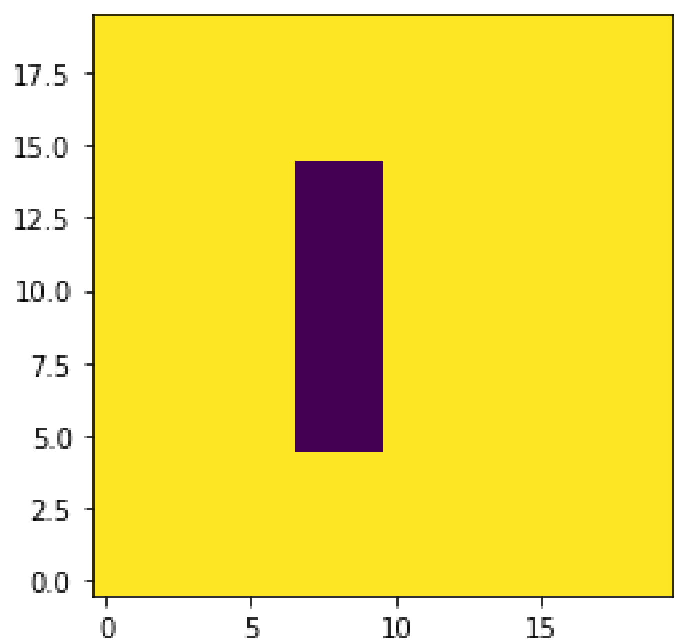
```
Help on built-in function randint:

randint(...) method of mtrand.RandomState instance
    randint(low, high=None, size=None, dtype='l')

    Return random integers from `low` (inclusive) to `high` (exclusive).

    Return random integers from the "discrete uniform" distribution of
    the specified dtype in the "half-open" interval [`low`, `high`). If
    `high` is None (the default), then results are from [0, `low`).

    Parameters
    ----------
    low : int
        Lowest (signed) integer to be drawn from the distribution (unless
        ``high=None``, in which case this parameter is one above the
        *highest* such integer).
    high : int, optional
        If provided, one above the largest (signed) integer to be drawn
        from the distribution (see above for behavior if ``high=None``).
    size : int or tuple of ints, optional
        Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
        ``m * n * k`` samples are drawn.  Default is None, in which case a
        single value is returned.
    dtype : dtype, optional
        Desired dtype of the result. All dtypes are determined by their
        name, i.e., 'int64', 'int', etc, so byteorder is not available
        and a specific precision may have different C types depending
        on the platform. The default value is 'np.int'.

        .. versionadded:: 1.11.0

    Returns
    -------
    out : int or ndarray of ints
        `size`-shaped array of random integers from the appropriate
        distribution, or a single such random int if `size` not provided.

    See Also
    --------
    random.random_integers : similar to `randint`, only for the closed
        interval [`low`, `high`], and 1 is the lowest value if `high` is
        omitted. In particular, this other one is the one to use to generate
        uniformly distributed discrete non-integers.

    Examples
    --------
    >>> np.random.randint(2, size=10)
```

```
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0])
>>> np.random.randint(1, size=10)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Generate a 2 x 4 array of ints between 0 and 4, inclusive:

>>> np.random.randint(5, size=(2, 4))
array([[4, 0, 2, 1],
       [3, 2, 2, 0]])
```

3.4.1 show a 2D array

[27]:
```python
y2= np.ones ([20, 20])
y2[5:15, 7:10]= 0
plt.imshow (y2, origin= 'bottom')
```

[27]: `<matplotlib.image.AxesImage at 0x7fb4777fe6a0>`



3.4.2 make a random 2D lattice

[28]:
```python
# Choose randomely from some inputs:
np.random.choice ([18, 13, 15, 46])
```

[28]: 46

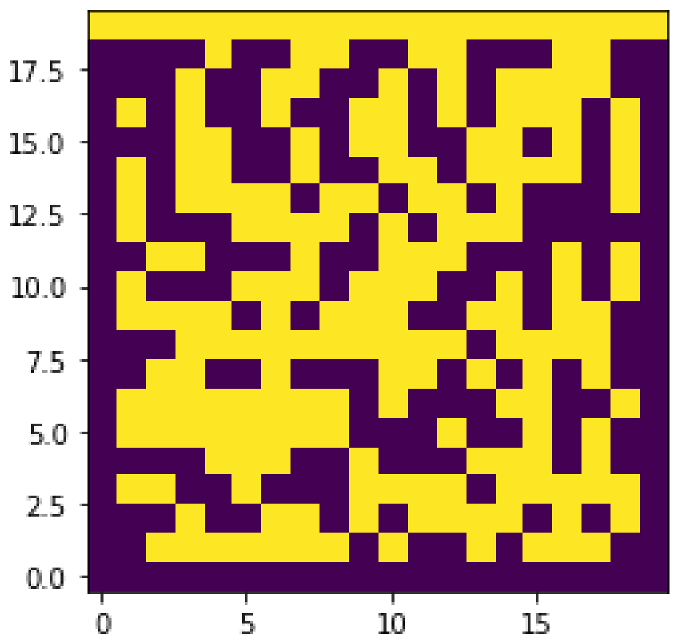
[29]:
```python
# Choose 2 objects randomely from some inputs:
np.random.choice ([18, 13, 15, 46], 2)
```

[29]:
```
array([13, 15])
```

[30]:
```python
# Make random [3, 4] array of 0 or 1:
np.random.choice ([0, 1], [3, 4])
```

[30]:
```
array([[1, 0, 0, 1],
       [0, 1, 0, 1],
       [0, 1, 0, 1]])
```

[31]:
```python
y3= np.random.choice ([0, 1], [20,20])
# plt.imshow (y3);

y3[0,:]= 0
y3[:, 0]= 0
y3[:, -1]= 0
y3[-1, :]= 1

plt.imshow (y3, origin= 'bottom');
```



[32]:
```python
# see whats in 10th row of y2:
y3[10]
```

[32]:
```
array([0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0])
```

[ ]:
```python
```