Types of objects (data)

- number(integer, floating,
- string "squence of letters, ordered, immutable" *list [ordered, mutable]*tuple: x = (5, 6) *sets: unordered, no repetition* dict {"key": value pair, "key2": value}
- numpy arrays supports scientific and mathamtical operations *panda* boolian

In [ ]:

```
```

In [1]:

```
#random module
import random
```

In [4]:

```
# Stimulating dice throw
random.choice([1,2,3,4,5,6])
```

Out[4]:

```
5
```

In [3]:

```
random.choice(range(1,7))
```

Out[3]:

```
5
```

In [9]:

```
# stimulate pickingn one of four dice, and the coutcome of that dice:
random.choice(random.choice([range(1,6), range(1,4), range(1,10)]))
```

Out[9]:

```
3
```

In [10]:

```
# now let's repeat the upbove 10 times and save the numbers we get:
draws = []
for i in range(10):
    draws.append(random.choice(random.choice([range(1,6), range(1,4), range(1,10)])))
```

In [11]:

```
draws
```

Out[11]:

```
[2, 7, 4, 3, 2, 5, 1, 3, 5, 4]
```

In [12]:

```
len(draws)
```

Out[12]:

```
10
```

In [13]:

```
# mutable and immutable objects
x =4
y = x
y = 5
```

In [14]:

```
x
```

Out[14]:

```
4
```

In [15]:

```
x = [ 1, 2,3, 4]
y = x
y[0] = 5
```

In [18]:

```
# chaning x changes y and changing y changes X because both of them point to the same object (mutable list). Whil
e when the object is immutable, python creates a new object and changes the reference of y only to that new objec
t.
print(y)
print(x)
```

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

In [19]:

```
x[3]= 500
```

In [20]:

```
print(y)
print(x)
```

```
[5, 2, 3, 500]
[5, 2, 3, 500]
```

In [21]:

```
# same goes for numpy array. Slicing is only a reference view to the same object, while indexing creates a new co
py of portion of the original array

import numpy as np
```

In [30]:

```
X = np.array([[1,2,3,4,5,6], [7,8,9,10,11,12]])
```

In [31]:

```
Y = X[:,1:3]
```

In [32]:

```
Y
```

Out[32]:

```
array([[2, 3],
       [8, 9]])
```

In [34]:

```
X[1,2] = 500
print(X)
print("notice how Y[1,1] also changes because it points to X[1,2]")
print(Y)
```

```
[[  1   2   3   4   5   6]
 [  7   8 500  10  11  12]]
notice how Y[1,1] also changes because it points to X[1,2]
[[  2   3]
 [  8 500]]
```

In [40]:

```
from colorama import Fore
```

In [59]:

```
# changing Y affects X because both point to the same object
Y[0,0] = 88
print(Y)
print("\n")
print(Fore.RED + "Note that Y[0,0] is [0,1] in X")
print(Fore.BLACK)
print(X)
```

```
[[ 88   3]
 [  8 500]]


Note that Y[0,0] is [0,1] in X

[[  1  88   3   4   5   6]
 [  7   8 500  10  11  12]]
```

In [62]:

```
# law of large number:

rolls = []
for i in range(100000):
    rolls.append(random.choice(range(1,7)))
```

In [63]:

```
len(rolls)
```

Out[63]:

```
100000
```

In [65]:

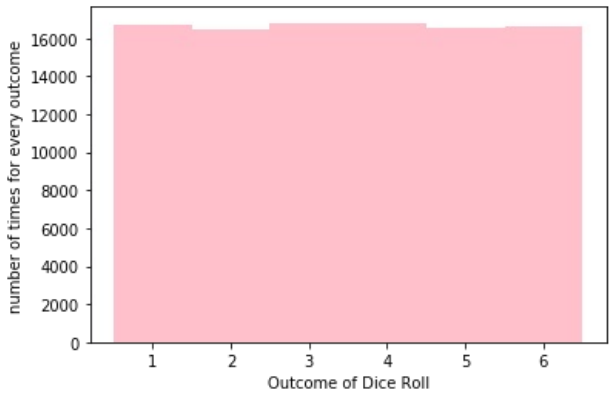
```
import matplotlib.pyplot as plt
```

In [81]:

```
plt.hist(rolls, bins=np.linspace(0.5,6.5,7), color='pink');
plt.xlabel('Outcome of Dice Roll')
plt.ylabel("number of times for every outcome")
```

Out[81]:

```
Text(0, 0.5, 'number of times for every outcome')
```



In [108]:

```
# law of summing two numbers approaches normal distribution
ys=[]
for k in range(1000):
    y= 0
    for i in range(10):
        x= random.choice(range(1,7))
        y= x+y
    ys.append(y)
```

In [110]:

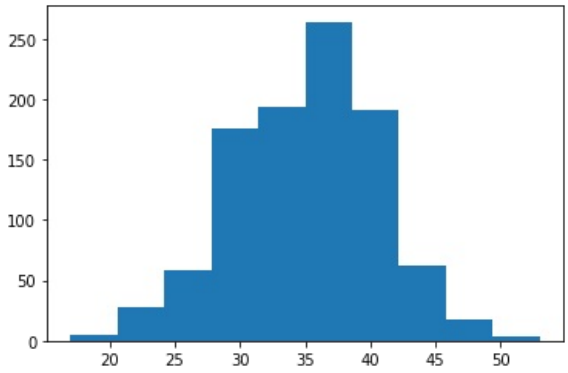
```
len(ys)
```

Out[110]:

```
1000
```

In [111]:

```
plt.hist(ys);
```



In [82]:

```
#numpy also has varous random generating methods
```

In [83]:

```
# np.random.random() generates random numbers between 0 and 1
np.random.random()
```

Out[83]:

```
0.44119608516328024
```

In [84]:

```
np.random.random(3)
```

Out[84]:

```
array([0.43098012, 0.48259731, 0.60112541])
```

In [85]:

```
np.random.random((2,2))
```

Out[85]:

```
array([[0.23607777, 0.88929927],
       [0.77138334, 0.73742458]])
```

In [86]:

```
# to generate random numbers from the standard normal distributions, wheremean is 0 and std is 1
#np.random.normal(mean, std, D1, D2)
np.random.normal(0,1)
```

Out[86]:

```
0.15581574999725617
```

In [87]:

```
np.random.normal(0,1,3)
```

Out[87]:

```
array([-0.43292755,  0.04418779, -0.10689396])
```

In [ ]:

```

```

In [89]:

```
np.random.normal(0,1,(3,3))
```

Out[89]:

```
array([[ 1.76781452, -1.07855452, -1.18197585],
       [ 1.73006627,  0.20843029, -0.07859246],
       [-1.08924229,  2.22685778,  1.01323829]])
```

In [92]:

```
# np.random.randint() generates random from range of numbers provided, third argument can provide size of array
np.random.randint(1,7)
```

Out[92]:

```
2
```

In [95]:

```
Y=np.random.randint(1,7, (3,3))
```

In [97]:

```
np.sum(Y, axis=1)
```

Out[97]:

```
array([14, 13, 10])
```

In [101]:

```
print(Y)
print('\n check your answer')
sum(Y[:,1])
```

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

 check your answer
```

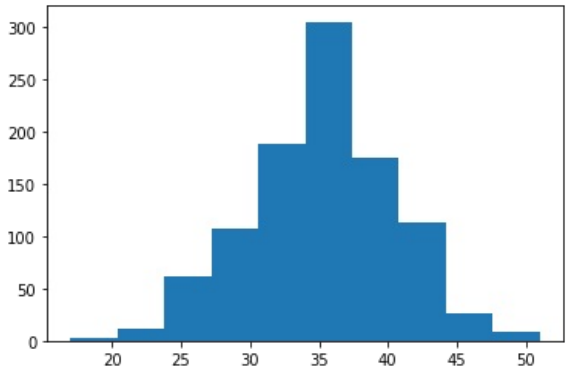
Out[101]:

```
10
```

In [103]:

```
# plotting sum of 10 die rolls

X= np.random.randint(1,7, (1000,10))
Y = np.sum(X, axis=1)
plt.hist(Y);
```



In [114]:

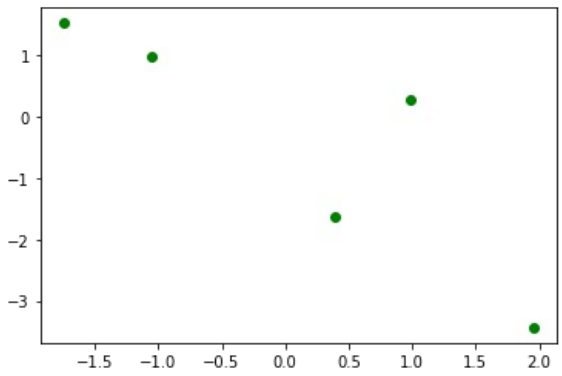
```
#random walks
delta_x = np.random.normal(0,1, (2,5))
```

In [115]:

```
plt.plot(delta_x[0], delta_x[1], "go")
```

Out[115]:

```
[<matplotlib.lines.Line2D at 0x1c8c6efae48>]
```

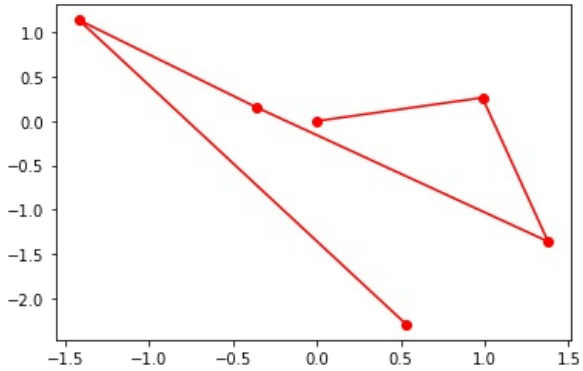


In [116]:

```
#let's have the random walk starts at the origin (0,0)
x_0 = np.array([[0],[0]])
X = np.concatenate( (x_0, np.cumsum(delta_x, axis= 1)) , axis =1)
plt.plot(X[0], X[1], 'ro-')
```

Out[116]:

```
[<matplotlib.lines.Line2D at 0x1c8c6f54888>]
```



In [117]:

```
X
```

Out[117]:

```
array([[ 0.        ,  0.98649828,  1.37820388, -0.36370411, -1.41752859,
         0.53414882],
       [ 0.        ,  0.26234359, -1.35925317,  0.15794043,  1.13861763,
        -2.29416954]])
```

In [ ]:

```

```