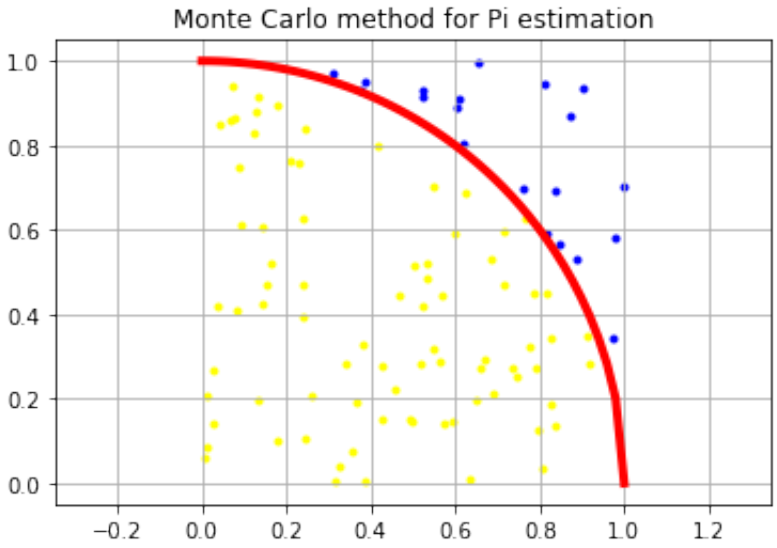
Use the section “Applying the Monte Carlo method for Pi estimation” on Page 91 of the book and use Python codes to calculate Pi. NOTE: Instead of ¼ of the circle (depicted in Figure 4.2), use half (1/2) of the circle and redo the exercise accordingly. a. Repeat the Python code using N = 100, 1000, 10000. Explain about the initial value and the role of Large Numbers in the calculation of Pi number. Explain all the codes, all functions, and different methodology in Python. b. Re-do the plot of the Pi estimation in Figure 4.3 for the half circle. Provide three figures for different cases where N = 100, 1000, 10000. Explain all the codes. Explain how the increasing number of N changes the figures.

In [ ]:
```
import math
import random
import numpy as np
import matplotlib.pyplot as plt
```

In [4]:
```
N = 100 #change here
M = 0

XCircle = []
YCircle = []
XSquare = []
YSquare = []

for p in range(N):
    x = random.random()
    y = random.random()
    if (x ** 2 + y ** 2 <= 1):
        M += 1
        XCircle.append(x)
        YCircle.append(y)
    else:
        XSquare.append(x)
        YSquare.append(y)

Pi = 4 * M / N

print("N=%d M=%d Pi=%.2f" % (N, M, Pi))

XLin = np.linspace(0, 1)
YLin = []
# NYLin = []

for x in XLin:
    YLin.append(math.sqrt(1 - x ** 2))
    # NYLin.append(-math.sqrt(1 - x ** 2))

plt.axis("equal")
```

```
plt.grid(which="major")
plt.plot(XLin, YLin, color="red", linewidth="4")
# plt.plot(XLin, NYLin, color="red", linewidth="4")
plt.scatter(XCircle, YCircle, color="yellow", marker=".")
plt.scatter(XSquare, YSquare, color="blue", marker=".")
plt.title("Monte Carlo method for Pi estimation")

plt.show()
```

```
N=100 M=80 Pi=3.20
```

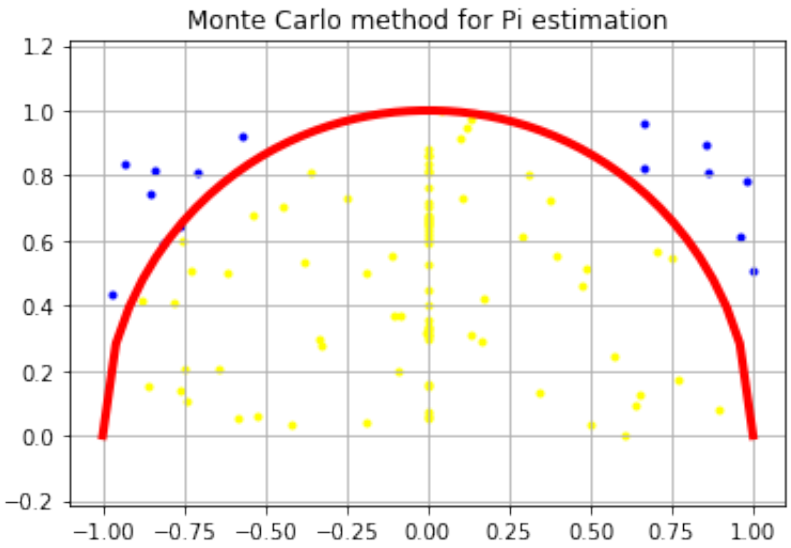


In [5]:

```
N = 100 #change here
M = 0

XCircle = []
YCircle = []
XSquare = []
YSquare = []

for p in range(N):
    x = random.random()*random.randint(-1,1)
    y = random.random()
    if (x ** 2 + y ** 2 <= 1):
        M += 1
        XCircle.append(x)
        YCircle.append(y)
    else:
        XSquare.append(x)
        YSquare.append(y)

Pi = 4 * M / N
```

```
print("N=%d M=%d Pi=%.2f" % (N, M, Pi))

XLin = np.linspace(-1, 1)
YLin = []
# NYLin = []

for x in XLin:
    YLin.append(math.sqrt(1 - x ** 2))
    # NYLin.append(-math.sqrt(1 - x ** 2))

plt.axis("equal")
plt.grid(which="major")
plt.plot(XLin, YLin, color="red", linewidth="4")
# plt.plot(XLin, NYLin, color="red", linewidth="4")
plt.scatter(XCircle, YCircle, color="yellow", marker=".")
plt.scatter(XSquare, YSquare, color="blue", marker=".")
plt.title("Monte Carlo method for Pi estimation")

plt.show()
```

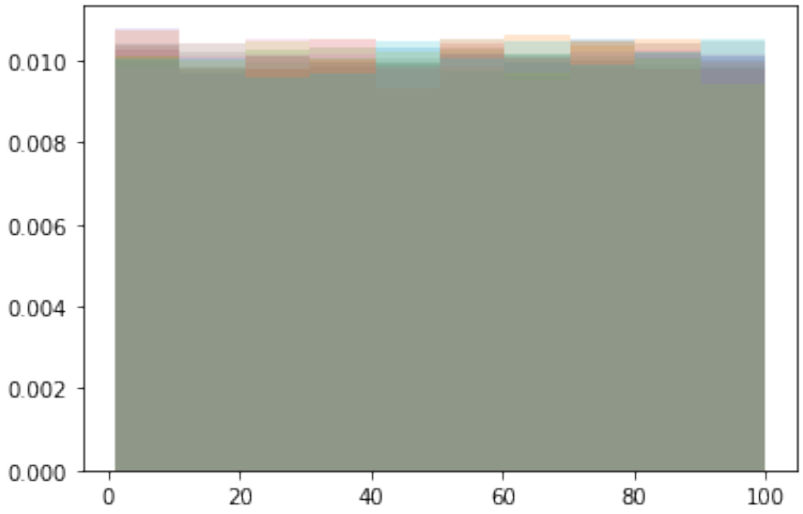
```
N=100 M=85 Pi=3.40
```

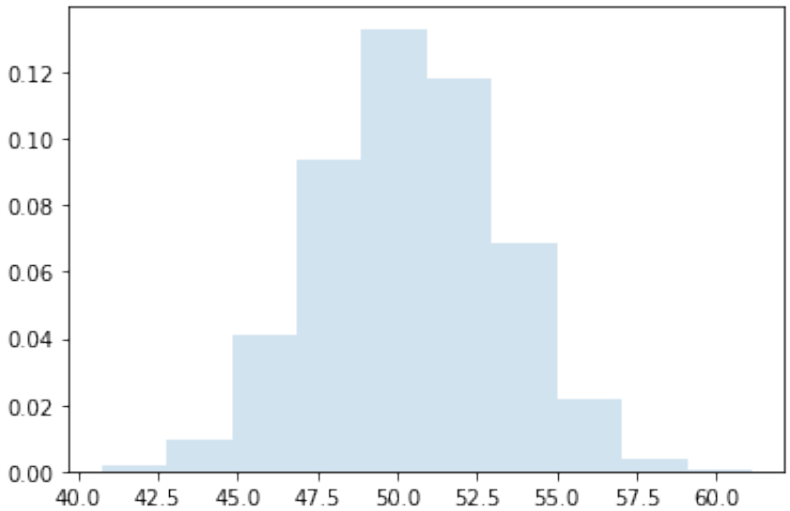


Estimate the expected value of a random variable as established by the law of large numbers in Page 97. To this end, generate 10,000 random numbers with a uniform distribution and then extract 100 samples from this population, also taken randomly (resulting in samples with 1000 numbers). (Note that in the book we had vice versa and had 1000 samples). Repeat this operation for a consistent number of times and store the results in a vector. Finally draw a histogram of the distribution that we have obtained. Compare your result with the figures in the book and explain which approach better explains the central limit theorem.

In [6]:
```
import random
import numpy as np
```

```
import matplotlib.pyplot as plt
a=1
b=100
N=10000
SamplesMeans = []
for j in range(0,10):
    DataPop=list(np.random.uniform(a,b,N))
    plt.hist(DataPop, density=True, histtype='stepfilled', alpha=0.2)



    for i in range(0,1000):
        DataExtracted = random.sample(DataPop,k=100)
        DataExtractedMean = np.mean(DataExtracted)
        SamplesMeans.append(DataExtractedMean)

plt.figure()
plt.hist(SamplesMeans, density=True, histtype='stepfilled', alpha=0.2)
plt.show()
```





Use the integration methodology using Monte Carlo in Page 104 to calculate the following two integrals. In addition, use Visual representation method in Page 112 to draw the Plot of numerical integration results.

$$\text{a)} \quad I = \int_0^5 2x/(2-x)\, dx$$

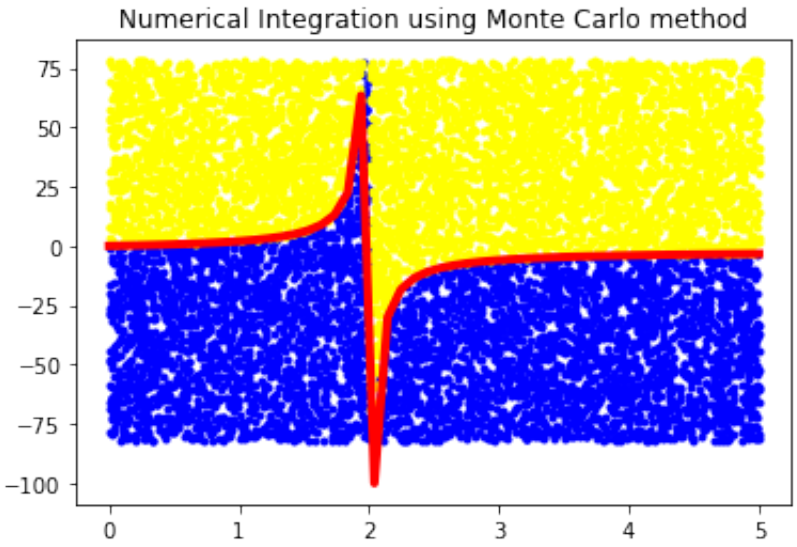
$$\text{b)} \quad I = \int_1^3 x^3\, dx$$

In [11]:
```
import random
import numpy as np
import matplotlib.pyplot as plt

random.seed(2)
f = lambda x: 2*x/(2-x)
a = 0.0

b = 5.0
NumSteps = 100
XIntegral=[]
YIntegral=[]
XRectangle=[]
YRectangle=[]

ymin = f(a)
ymax = ymin

for i in range(NumSteps):
    x = a + (b - a) * float(i) / NumSteps
    if (2-x)!=0:
        y = f(x)
        if y < ymin: ymin = y
        if y > ymax: ymax = y

print(f'ymin={ymin}, ymax={ymax}')
A = (b - a) * (ymax - ymin)
print(f'A={A}')
N = 10000
M = 0
for k in range(N):
    x = a + (b - a) * random.random()
    y=random.choice([ymax* random.random(),ymin*random.random()])
    # print(f'f(x)={f(x)}, y={y}')
```

```
    if y <= f(x):
            M += 1
            XIntegral.append(x)
            YIntegral.append(y)
    else:
            XRectangle.append(x)
            YRectangle.append(y)

NumericalIntegral = M / N * A
print ("Numerical integration = " + str(NumericalIntegral))

XLin=np.linspace(a,b)
YLin=[]
for x in XLin:
    YLin.append(f(x))
print(XLin)
print(len(XLin))
print(YLin)
print(len(YLin))

# plt.axis   ([0,b, -50,50])
plt.plot   (XLin,YLin, color="red" , linewidth="4")
plt.scatter(XIntegral, YIntegral, color="blue", marker   =".")
plt.scatter(XRectangle, YRectangle, color="yellow", marker   =".")
plt.title  ("Numerical Integration using Monte Carlo method")
plt.show()
```

```
ymin=-82.00000000000028, ymax=77.99999999999993
A=800.0000000000011
Numerical integration = 394.8000000000006
[0.         0.10204082 0.20408163 0.30612245 0.40816327 0.51020408
 0.6122449  0.71428571 0.81632653 0.91836735 1.02040816 1.12244898
 1.2244898  1.32653061 1.42857143 1.53061224 1.63265306 1.73469388
 1.83673469 1.93877551 2.04081633 2.14285714 2.24489796 2.34693878
 2.44897959 2.55102041 2.65306122 2.75510204 2.85714286 2.95918367
 3.06122449 3.16326531 3.26530612 3.36734694 3.46938776 3.57142857
 3.67346939 3.7755102  3.87755102 3.97959184 4.08163265 4.18367347
 4.28571429 4.3877551  4.48979592 4.59183673 4.69387755 4.79591837
 4.89795918 5.        ]
50
[0.0, 0.10752688172043011, 0.22727272727272727, 0.3614457831325302, 0
.5128205128205129, 0.684931506849315, 0.8823529411764706, 1.111111111
1111112, 1.3793103448275863, 1.6981132075471697, 2.0833333333333335,
2.5581395348837215, 3.1578947368421053, 3.9393939393939386, 5.0, 6.52
1739130434785, 8.88888888888889, 13.076923076923073, 22.5000000000000
07, 63.3333333333335, -99.99999999999989, -30.000000000000014, -18.33
3333333333314, -13.529411764705877, -10.909090909090908, -9.259259259
25926, -8.125000000000002, -7.2972972972972965, -6.666666666666666, -
6.170212765957447, -5.769230769230768, -5.438596491228069, -5.1612903
22580645, -4.925373134328359, -4.722222222222222, -4.545454545454545,
```

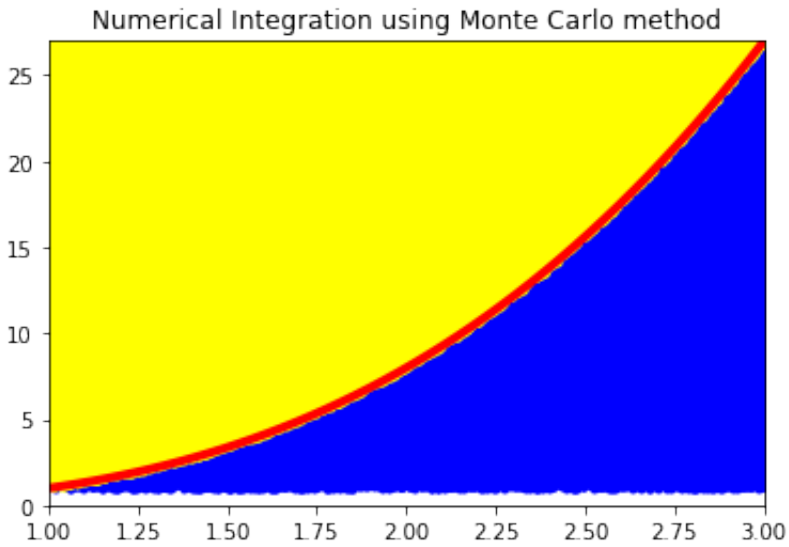
```
-4.390243902439024, -4.252873563218391, -4.130434782608695, -4.020618
556701031, -3.9215686274509802, -3.8317757009345796, -3.75, -3.675213
6752136755, -3.60655737704918, -3.543307086614173, -3.484848484848485
, -3.4306569343065694, -3.380281690140845, -3.3333333333333335]
50
```



In [14]:
```
import random
import numpy as np
import matplotlib.pyplot as plt

random.seed(2)
f = lambda x: x**3
a = 1.0

b = 3.0
NumSteps = 1000000
XIntegral=[]
YIntegral=[]
XRectangle=[]
YRectangle=[]

ymin = f(a)
ymax = ymin
for i in range(NumSteps):
    x = a + (b - a) * float(i) / NumSteps
    y = f(x)
    if y < ymin: ymin = y
    if y > ymax: ymax = y

A = (b - a) * (ymax - ymin)
N = 100000
M = 0
for k in range(N):
    x = a + (b - a) * random.random()
```

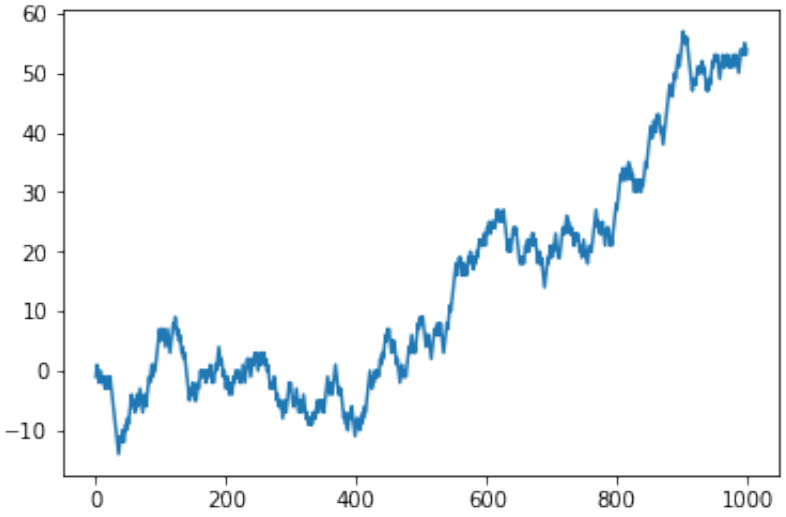
```
    y = ymin + (ymax - ymin) * random.random()
    if y <= f(x):
            M += 1
            XIntegral.append(x)
            YIntegral.append(y)
    else:
            XRectangle.append(x)
            YRectangle.append(y)
NumericalIntegral = M / N * A
print ("Numerical integration = " + str(NumericalIntegral))

XLin=np.linspace(a,b)
YLin=[]
for x in XLin:
    YLin.append(f(x))


plt.axis   ([a, b, 0, f(b)])
plt.plot   (XLin,YLin, color="red" , linewidth="4")
plt.scatter(XIntegral, YIntegral, color="blue", marker   =".")
plt.scatter(XRectangle, YRectangle, color="yellow", marker   =".")
plt.title  ("Numerical Integration using Monte Carlo method")
plt.show()
```

```
Numerical integration = 18.019522574784947
```



Using P=0.45, P=0.5, and P=0.55 and redo the simple random walk in Page 130-132 for each value of P. Draw the trend plot of the random walk and provide explanations about each trend and the impact of P in the trend.

In [19]:
```
from random import seed
from random import random
from matplotlib import pyplot
seed(1)
RWPath = list()
#change here
P=0.5
# P=0.45
# P=0.55
RWPath.append(-1 if random() < P else 1)

for i in range(1, 1000):
    ZNValue = -1 if random() < P else 1
    XNValue = RWPath[i-1] + ZNValue
    RWPath.append(XNValue)

pyplot.plot(RWPath)
pyplot.show()
```



Simulating weather forecast according to the following probability values. P(Sunny| Sunny)=0.75, P(Sunny| Rainy) = 0.25 P(Rainy| Sunny)=0.3 P(Rainy| Rainy) = 0.7 a) Plot the weather forecast for the next 365 days.

b) Draw the histogram of the weather forecast.

c) Draw a histogram for the weather forecast for different days of the week. You can draw one histogram for the number of sunny days during different days of the week and another histogram for the number of rainy days during days of the week. You can also combine and use two bars (one for rainy and one for sunny days) during a week. Is there any difference among different days?

Given that Monday is sunny find the following probability values:

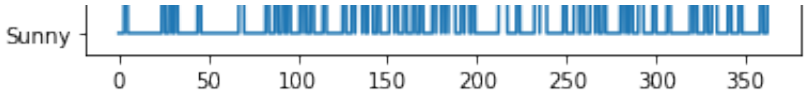
d) Tuesday is rainy

e) Tuesday and Wednesday are both Sunny

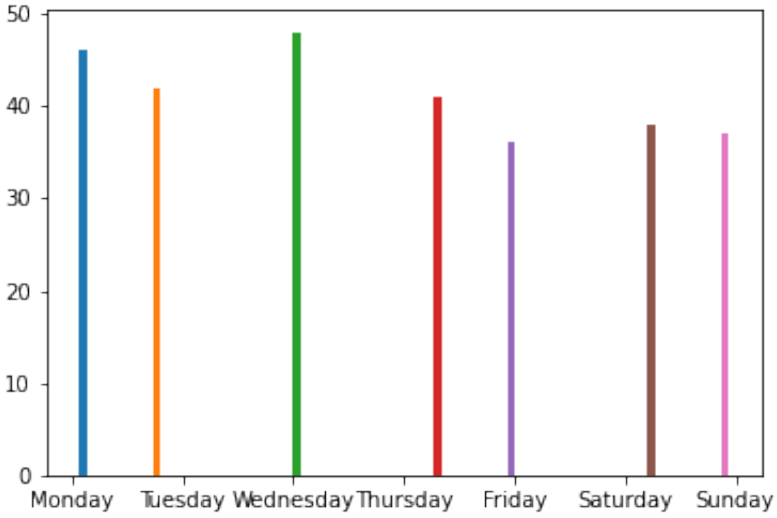
In [23]:
```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(3)
StatesData = ["Sunny", "Rainy"]

TransitionStates = [["SuSu", "SuRa"], ["RaRa", "RaSu"]]
TransitionMatrix = [[0.75, 0.25], [0.30, 0.70]]

WeatherForecasting = list()
NumDays = 365
TodayPrediction = StatesData[0]

print("Weather initial condition =", TodayPrediction)
week_count = 0

for i in range(1, NumDays):

    if TodayPrediction == "Sunny":
        TransCondition = np.random.choice(TransitionStates[0], replace
        if TransCondition == "SuSu":
            pass
        else:
            TodayPrediction = "Rainy"



    elif TodayPrediction == "Rainy":
        TransCondition = np.random.choice(TransitionStates[1], replace
        if TransCondition == "RaRa":
```

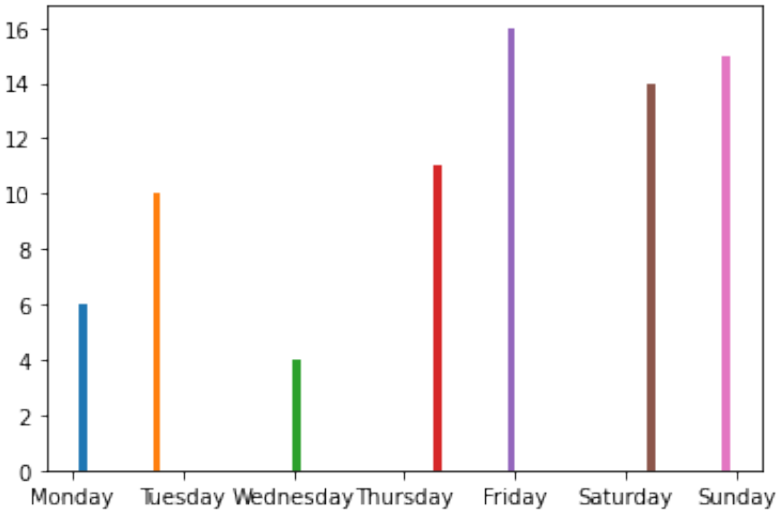
```
            pass
        else:
            TodayPrediction = "Sunny"

    WeatherForecasting.append(TodayPrediction)
WeatherForecastingWeekly= [[],[],[],[],[],[],[]]
WeatherForecastingWeekly_SUNNY = [[],[],[],[],[],[],[]]
WeatherForecastingWeekly_RAINY = [[],[],[],[],[],[],[]]
# c=['r','g','b','c','m','y','k']
for i in range(len(WeatherForecasting)):
    WeatherForecastingWeekly[i % 7].append(WeatherForecasting[i])
    if WeatherForecasting[i]=='Sunny':
        WeatherForecastingWeekly_SUNNY[i%7].append(week[i%7])
    else:
        WeatherForecastingWeekly_RAINY[i%7].append(week[i%7])
plt.plot(WeatherForecasting)
plt.show()
plt.figure()
plt.hist(WeatherForecastingWeekly_SUNNY)
plt.show()
plt.hist(WeatherForecastingWeekly_RAINY)
plt.show()
rainydayonTuesday=0
for i in WeatherForecastingWeekly[1]:
    if i=='Rainy':
        rainydayonTuesday+=1
P_RainyOnThusday= rainydayonTuesday/len(WeatherForecastingWeekly[1])
print(f'P_RainyOnThusday={P_RainyOnThusday}')


BothsunnyonThuesdayandWensday=0
for i in range(len(WeatherForecastingWeekly[1])):
    if WeatherForecastingWeekly[1][i] == 'Sunny' and WeatherForecastir
        BothsunnyonThuesdayandWensday+=1
P_BothsunnyonThuesdayandWensday=BothsunnyonThuesdayandWensday/(365//7)
print(f'P_BothsunnyonThuesdayandWensday={P_BothsunnyonThuesdayandWensc
```

```
Weather initial condition = Sunny
```




```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: Vis
ibleDeprecationWarning: Creating an ndarray from ragged nested sequen
ces (which is a list-or-tuple of lists-or-tuples-or ndarrays with dif
ferent lengths or shapes) is deprecated. If you meant to do this, you
must specify 'dtype=object' when creating the ndarray
  return array(a, dtype, copy=False, order=order)
```





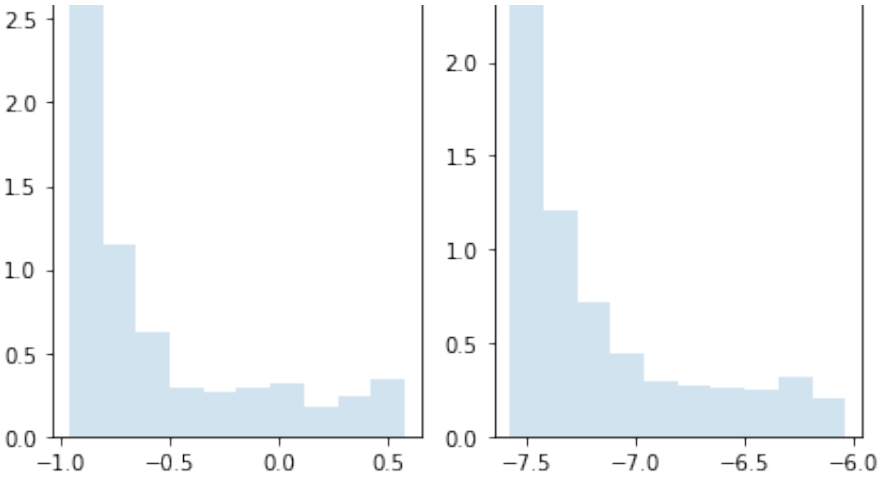

```
P_RainyOnThusday=0.19230769230769232
P_BothsunnyonThuesdayandWensday=0.75
```

(Jackknife resampling) Do the resampling of 250 and 500 randomly generated values and calculate the coefficient of variation. Report the correlation of variation in different stages of the resampling. Use two different number of samples (N) in the starting distribution (according to the number of values). Explain what Pseudo value is in the resampling method described in the Jackknife resampling method. Draw the distribution of pseudo values

In [28]:
```
import random
import statistics
import matplotlib.pyplot as plt

PopData = list()
PopData2=[]
random.seed(5)

for i in range(250):
    DataElem = 10 * random.random()
    PopData.append(DataElem)

for i in range(500):
    DataElem = 10 * random.random()
    PopData2.append(DataElem)

def CVCalc(Dat):
    CVCalc = statistics.stdev(Dat) / statistics.mean(Dat)
    return CVCalc


CVPopData = CVCalc(PopData)
CVPopData2 = CVCalc(PopData2)
print(f'CVPopData={CVPopData}')
print(f'CVPopData2={CVPopData2}')

N = len(PopData)
JackVal = [0]*N
PseudoVal = [0]*N



for i in range(N):
    for j in range(N):
        if (j < i):
            JackVal[j] = PopData[j]
        else:
            if (j > i):
                JackVal[j - 1] = PopData[j]
    PseudoVal[i] = N * CVCalc(PopData) - (N - 1) * CVCalc(JackVal)
```

```
N2 = len(PopData2)
JackVal2 = [0]*N2
PseudoVal2 = [0]*N2
for i in range(N2):
    for j in range(N2):
        if (j < i):
            JackVal2[j] = PopData2[j]
        else:
            if (j > i):
                JackVal2[j - 1] = PopData2[j]
    PseudoVal2[i] = N2 * CVCalc(PopData) - (N2 - 1) * CVCalc(JackVal2)


plt.subplot(121)
plt.title('250')
plt.hist(PseudoVal, density=True, histtype='stepfilled', alpha=0.2)
plt.subplot(122)
plt.title('500')
plt.hist(PseudoVal2, density=True, histtype='stepfilled', alpha=0.2)
plt.tight_layout()


MeanPseudoVal = statistics.mean(PseudoVal)
print(f'MeanPseudoVal={MeanPseudoVal}')
VariancePseudoVal = statistics.variance(PseudoVal)
print(f'VariancePseudoVal={VariancePseudoVal}')
VarJack = statistics.variance(PseudoVal) / N
print(f'VarJack={VarJack}')

MeanPseudoVal2 = statistics.mean(PseudoVal2)
print(f'MeanPseudoVal 2={MeanPseudoVal2}')
VariancePseudoVal2 = statistics.variance(PseudoVal2)
print(f'VariancePseudoVal 2={VariancePseudoVal2}')
VarJack2 = statistics.variance(PseudoVal2) / N2
print(f'VarJack 2={VarJack2}')

plt.show()
```

```
CVPopData=0.5800698843053581
CVPopData2=0.5933261337134711
MeanPseudoVal=-0.5702694171088855
VariancePseudoVal=0.1977172786901032
VarJack=0.0007908691147604128
MeanPseudoVal 2=-7.17333888113138
VariancePseudoVal 2=0.17969504069693995
VarJack 2=0.0003593900813938799
```



Using the Gradient Descent algorithm in Python calculate the maximum/minimum value of

a) Explain about the learning rate variable. Try 0.1, 0.5, and 0.01 and compare the results.

$$y = -3x^2 - 5x + 2$$

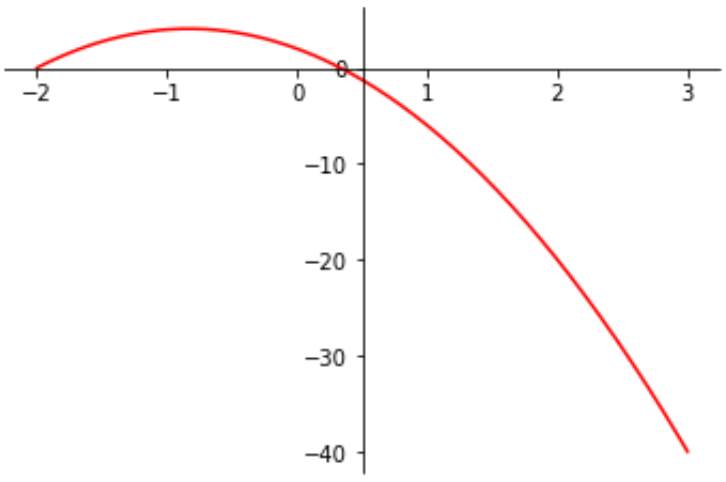
b) List the last 10 values of y in the iterations.

c) Now imposing a restriction on x, and having a new feasible set, calculate the following optimization problem.

$$y = -3x^2 - 5x + 2$$
$$s.t. x \geq 0$$

In [29]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 3, 10000000)
y = -3*x ** 2 - 5 * x + 2

fig = plt.figure()
axdef = fig.add_subplot(1, 1, 1)
axdef.spines['left'].set_position('center')
axdef.spines['bottom'].set_position('zero')
axdef.spines['right'].set_color('none')
```

```
axdef.spines['top'].set_color('none')
axdef.xaxis.set_ticks_position('bottom')
axdef.yaxis.set_ticks_position('left')

plt.plot(x, y, 'r')
plt.show()

Gradf = lambda x: -6 * x - 5

ActualX = 3
LearningRate = 0.01 # change here
PrecisionValue = 0.000001
PreviousStepSize = 1
MaxIteration = 10000
IterationCounter = 0
last_ten_list=np.array([])

while PreviousStepSize > PrecisionValue and IterationCounter < MaxIter
    PreviousX = ActualX
    ActualX = ActualX + LearningRate * Gradf(PreviousX)
    PreviousStepSize = abs(ActualX - PreviousX)
    IterationCounter = IterationCounter + 1
    if len(last_ten_list)==10:
        last_ten_list=np.append(last_ten_list[1::],ActualX)
    else:
        last_ten_list=np.append(last_ten_list,ActualX)
    print("Number of iterations = ", IterationCounter, "\nActual value

print("X value of f(x) maximum = ", ActualX)
last_ten_list=-3 *last_ten_list**2-5*last_ten_list+2
print(last_ten_list)
```




```
Number of iterations =  1
Actual value of x  is =  2.77
Number of iterations =  2
Actual value of x  is =  2.5538
```

```
Number of iterations =  3
```

In [36]:
```
#s.t.x >= 0
ActualX = 3
LearningRate = 0.01
PrecisionValue = 0.000001
PreviousStepSize = 1
MaxIteration = 10000
IterationCounter = 0
last_ten_list=np.array([])
while PreviousStepSize > PrecisionValue and IterationCounter < MaxIter
    PreviousX = ActualX
    ActualX = ActualX + LearningRate * Gradf(PreviousX)
    if ActualX <0:
        ActualX=PreviousX
        break
    PreviousStepSize = abs(ActualX - PreviousX)
    IterationCounter = IterationCounter + 1
    if len(last_ten_list)==10:
        last_ten_list=np.append(last_ten_list[1::],ActualX)
    else:
        last_ten_list=np.append(last_ten_list,ActualX)
    print("Number of iterations = ", IterationCounter, "\nActual value
print("X value of f(x) maximum = ", ActualX)
last_ten_list=-3 *last_ten_list**2-5*last_ten_list+2
print(last_ten_list)
```

```
Number of iterations =  1
Actual value of x  is =  2.77
Number of iterations =  2
Actual value of x  is =  2.5538
Number of iterations =  3
Actual value of x  is =  2.3505719999999997
Number of iterations =  4
Actual value of x  is =  2.1595376799999997
Number of iterations =  5
Actual value of x  is =  1.9799654191999996
Number of iterations =  6
Actual value of x  is =  1.8111674940479996
Number of iterations =  7
Actual value of x  is =  1.6524974444051197
Number of iterations =  8
Actual value of x  is =  1.5033475977408126
Number of iterations =  9
Actual value of x  is =  1.3631467418763639
Number of iterations =  10
Actual value of x  is =  1.231357937363782
Number of iterations =  11
Actual value of x  is =  1.107476461121955
Number of iterations =  12
```

```
Actual value of x  is =  0.9910278734546376
Number of iterations =  13
Actual value of x  is =  0.8815662010473594
Number of iterations =  14
Actual value of x  is =  0.7786722289845178
Number of iterations =  15
Actual value of x  is =  0.6819518952454467
Number of iterations =  16
Actual value of x  is =  0.59103478153072
Number of iterations =  17
Actual value of x  is =  0.5055726946388768
Number of iterations =  18
Actual value of x  is =  0.42523833296054414
Number of iterations =  19
Actual value of x  is =  0.3497240329829115
Number of iterations =  20
Actual value of x  is =  0.2787405910039368
Number of iterations =  21
Actual value of x  is =  0.21201615554370057
Number of iterations =  22
Actual value of x  is =  0.14929518621107854
Number of iterations =  23
Actual value of x  is =  0.09033747503841383
Number of iterations =  24
Actual value of x  is =  0.034917226536109
X value of f(x) maximum =  0.034917226536109
[-2.80493464 -2.00314025 -1.29467472 -0.66867458 -0.11554086  0.37320
809
  0.80506667  1.18665691  1.52383005  1.82175623]
```

In one paragraph explain how does the Newton-Raphson method work a) Using this method find the optimal value of the same function as the above example. Explain the details.

$$y = -3x^2 - 5x + 2$$

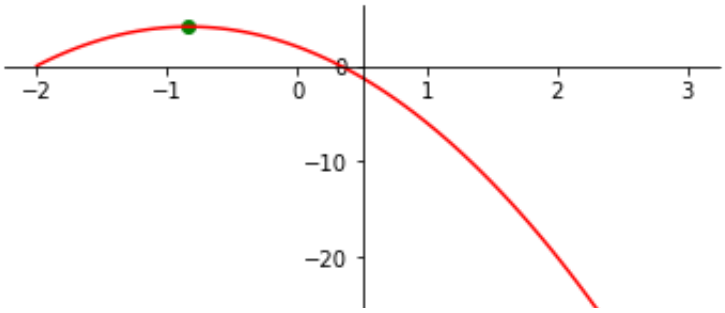
b) Draw the maximum/minimum point of the function.

c) In how many iterations did you find the results. Compare the performance with the Gradient descent algorithm in Problem 7

In [38]:
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-2, 3, 100)
```

```
y = -3*x ** 2 - 5 * x + 2

print('Value of x at the minimum of the function', x[np.argmin(y)])

FirstDerivative = lambda x: -6 * x - 5
SecondDerivative = lambda x: -6

ActualX = 3
PrecisionValue = 0.000001
PreviousStepSize = 1
MaxIteration = 10000
IterationCounter = 0

while PreviousStepSize > PrecisionValue and IterationCounter < MaxIter
    PreviousX = ActualX
    ActualX = ActualX - FirstDerivative(PreviousX) / SecondDerivative(
    PreviousStepSize = abs(ActualX - PreviousX)
    IterationCounter = IterationCounter + 1
    print("Number of iterations = ", IterationCounter, "\nActual value

print("X value of f(x) minimum = ", ActualX)

fig = plt.figure()
axdef = fig.add_subplot(1, 1, 1)
axdef.spines['left'].set_position('center')
axdef.spines['bottom'].set_position('zero')
axdef.spines['right'].set_color('none')
axdef.spines['top'].set_color('none')
axdef.xaxis.set_ticks_position('bottom')
axdef.yaxis.set_ticks_position('left')
plt.scatter(ActualX,-3*ActualX**2 -5*ActualX + 2,c='g')

plt.plot(x,y, 'r')
plt.show()
```

```
Value of x at the minimum of the function 3.0
Number of iterations =  1
Actual value of x  is =  -0.8333333333333335
Number of iterations =  2
Actual value of x  is =  -0.8333333333333334
X value of f(x) minimum =  -0.8333333333333334
```




−30
−40