***1. Generating random numbers***

1a) Go to your working directory.

Copy the file myreport.py into it.

Start a new Python code file e.g. s6\_task1.py and insert the following code:

from myreport import html\_report

import random

# CREATE REPORT FOR OUTPUT

myreport=html\_report("s6\_task1.html")

# ADD A HEADING

myreport.add\_heading("Generating random numbers")

# ADD A SUBHEADING

myreport.add\_subheading ("1a.")

# ADD SOME TEXT

myreport.add\_text("result of calling random.random()")

Generate the value of a random number using random.random() and include the value in your report using myreport.add\_text(). After doing this add the following lines to write the report and open the resulting file:

# ADD SOURCE CODE AND WRITE REPORT

# THEN OPEN IT IN A BROWSER

# INSERT SOURCE CODE INTO REPORT

myreport.add\_subheading('Python Code')

myreport.add\_source(\_\_file\_\_)

# WRITE REPORT TO FILE

myreport.write()

# OPEN FILE IN BROWSER

myreport.view()

Run this code. Your web browser should open showing your "report"!

***As you work through the rest of the tasks store your results and comments in a report.***

b) Store 1,000,000 numbers that are generated by this function in a list rand\_num\_list.

i) Histogram the set of numbers using the code below:

# BECAUSE FIGURE IS GOING IN REPORT

# WE CREATE IT LIKE THIS:

fig1 = myreport.init\_figure()

# WE CAN THEN TREAT IT LIKE A NORMAL MATPLOTLIB FIGURE

ax=fig1.add\_subplot(1,1,1)

ax.hist(rand\_num\_list, bins=100)

# ADD FIGURE TO REPORT

myreport.add\_subheading('Histogram of 1,000,000 random numbers')

myreport.add\_figure(fig1)

ii) Describe the distribution in words.

iii) Describe what the function random.random() does.

c) What proportion of the numbers generated by random.random() are:

i) greater that 0.5

ii) less than 0.5

iii) less than 0.7

d) How could we simulate

i) a fair coin toss using random.random()?

ii) an unfair coin toss that is ‘heads’ 70% of the time?

e. i) Write functions toss\_fair\_coin() and toss\_unfair\_coin() which return ‘H’ or ‘T’ in accordance with i) and ii).

ii) Check your code works as expected by counting the number of ‘H’ after 10,000 tosses in each case.

***2. Generating a random choice***

Use a plastic bag and screw some up some paper to represent balls.

We want to simulate a system with 9 green balls and 3 red balls. In a single 'trial' or 'measurement' we draw a ball out the bag, record its colour then **put it back**.

Start a new python file: s6\_task2.py (and html\_report: s6\_task2.html)

(There is a template in the Dropbox folder).

a) What is the chance that the colour will be green?

b) Find a bag and make some paper ‘balls’. Record the result of 20 draws.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Draw | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Colour |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

%green =

%red =

How does the result compare with what you expected?

c) Write a Python function that simulates a draw, using random.random() to generate a random ball. Return a value 'G' for green and 'R' for red.

def draw\_ball():

d) Count the number of green balls drawn from 10,000 draws, and comment on your result.

e) Modify your function (if necessary) so it will work for a system with G green balls and R red balls.

e.g. function definition uses

def draw\_ball2(G,R):

where G is the number of green balls and R the number of red balls, and it returns a character ‘G’ or ‘R’

***3. Generating random trajectories***

We are going to adjust our system so that it models mRNA degradation.

* green balls represent function mRNA strands
* red balls represent non-functional mRNA strands

We are going to act as an enzyme molecule that acts to degrades a mRNA strand upon collision.

Every 10 seconds we take a ball out of the bag simulating our enzyme molecule colliding with a molecule.

* If it is green we 'degrade' the mRNA strand and replace it with a red ball.
* If it is red we put it back.

a) Simulate this system yourself using paper balls, recording your result in a table e.g.

|  |  |  |
| --- | --- | --- |
| Time | Green | Red |
| 0 | 10 | 5 |
| 10 |  |  |
| 20 |  |  |
| ... |  |  |

Plot the time series of the number of green balls you generated using Python.

Imagine doing this 100 times or 10,000 times...

Difficult by hand but possible with Python!

b) Write a Python function that performs one 'run'.

Start a new file; 's6\_task3.py'. Some code to get you started is below. Note how it uses a while loop to increment time in steps of 10 seconds.

1. **def** simulate\_system():
2. #balls
3. G=10
4. R=5
5. T=G+R
7. #time
8. t=0
10. G\_record=[]
11. R\_record=[]
12. t\_record=[]
14. G\_record.append(G)
15. R\_record.append(R)
16. t\_record.append(t)
18. **while** t<=1000:
19. # increment time
20. t=t+10
22. # select ball from bag
24. # update number of balls
26. # record system state
27. G\_record.append(G)
28. R\_record.append(R)
29. t\_record.append(t)

32. **return** (G\_record, R\_record, t\_record)
34. results= simulate\_system()
35. G\_record, R\_record, t\_record = results
36. plt.plot(t\_record, G\_record, 'g-')
37. plt.plot(t\_record, R\_record, 'r-')
38. plt.show()

b) Examine the length and contents of the three output lists:

G\_record, R\_record, t\_record

Check you understand how the code above works.

c) Use a loop similar to the one below to simulate 6 runs of the system. Store each resulting G\_record in a list called my\_runs.

1. my\_runs=[]
2. t\_record=None
3. **for** i **in** range(6):
4. results= simulate\_system()
5. G\_record, R\_record, t\_record = results
6. my\_runs.append(G\_record)

After this code runs check that:

my\_runs contains a list of lists.

t\_record contains the time increments (which were identical for each run)

i) What is the length of my\_run?

ii) What is the length of each element in my\_run?

***4. Analysing random trajectories***

This task builds on the result of the last.

To plot all the trajectories of the system we might try to do something like this:

plt.plot(t\_record, my\_runs, 'r-')

This fails because the dimensions do not match; t\_record contains 100 timepoints, but my\_runs contains 6 runs:

t\_record = [ 0, 10, 20, 30, 40, ... , 950, 960, 980, 990]

# 1 list with 100 elements

my\_run=[

[ 10, 10, 9, 9, 9, ... 0, 0, 0, 0, 0, 0],

[ 10, 10, 10, 9, 9, ... 0, 0, 0, 0, 0, 0],

[ 10, 9, 9, 8, 8, ... 0, 0, 0, 0, 0, 0],

[ 10, 9, 9, 8, 7, ... 0, 0, 0, 0, 0, 0],

[ 10, 10, 10, 9, 9, ... 0, 0, 0, 0, 0, 0],

[ 10, 9, 9, 8, 8, ... 0, 0, 0, 0, 0, 0],

] # 6 rows each with 100 elements

To be able to plot these against the time values we need to transpose the my\_runs results so that it is stored like this:

my\_run=[

[ 10, 10, 10, 10, 10, 10],

[ 10, 10, 9, 9, 10, 9],

[ 9, 10, 9, 9, 10, 9],

[ 9, 9, 8, 8, 9, 8],

...

[ 0, 0, 0, 0, 0, 0],

[ 0, 0, 0, 0, 0, 0],

[ 0, 0, 0, 0, 0, 0]

] # 100 rows each with 6 elements

a) In order to do this it is convenient to convert my\_runs into to a 2D numpy array:

my\_runs=np.array(my\_runs)

Now use the the numpy array transpose method and verify that the command below then works:

plt.plot(t\_record, my\_runs, '-')

b) Repeat this plot using 100 runs. Comment the value of this plot in helping you understand the system.

Compare your simulation of the system "by-hand" in Task 3a with the computed trajectories. Are they consistent?

c) In order to display the results of many 'runs' of a random system we can:

- plot the average of all the trajectories

- plot the distribution of the set of runs at a given time point

i) look at how to use the numpy array average function. Use it to generate the average trajectory over time. Store this as variable av\_run and plot it.

plt.plot(t\_record, av\_run, 'k-', lw='2')

ii) take a slice of the array my\_runs which contain the number of "green balls" for each run at time=100s. Histogram the result using 10 bins.

(Hint this t=100s corresponds to row 10 of my\_runs).

Comment on the distribution shown.

***Extension Task 1: Working with Probabilities***

a) Consider the roll of a single (fair) dice. What are the following probabilities?

b) Consider the result from a sequence of four such dice rolls. Write expressions for, then calculate , the following probabilities:

i)

ii)

iii)

iv)

v)

v)

c) Write code that generates:

i) the result of one random dice roll

ii) the result of a sequence of four dice rolls (so that it produces a list e.g. [1,2,3,4] or [2,3,1,2])

iii) 10,000 sets of four dice rolls. i.e. a list containing 10,000 roll sequences

d) Compare your results with the predictions made in part e).

Hint. You can compare list contents directly, e.g.

if dice\_seq==[1,2,3,4]:

seq\_count+=1

***Extension Task 2: The law of averages***

The “law of averages” or the “law of large numbers” is that if a large enough set of trials / measurements / samples of a system are recorded then the observed distribution found will tend to the true underlying distribution.

e.g.

- over a large number of spins of a a roulette wheel, the number of times that the ball ends in “red” will be approximately equal to the number of times that the ball ends in “black”.

**Test it!**

Write code to simulate the spin of a random roulette wheel, returning a value that corresponds to Red or Black (each with probability 0.5).

1) Count the number of Red and Black results after:

i) 20 spins

ii) 200 spins

iii) 2000 spins

iv) 20,000 spins

v) 200,000 spins

vi) 2,000,000 spins

2) Calculate the fractional deviation between the recorded count of “red” and the expected values of “red”.

The | | brackets mean take the absolute value e.g. |-3| = 3

Example:

Suppose for 100 spins we record 62 “red” spins (we would expect 50)

i.e. so in this case we saw 24% more “red” spins than expected.

3) Plot the results you found in (2) – what is the general trend shown?

***Extension Task 3: The central limit theorem***

The central limit theorem describes the case in which an average is calculated from a large set of measurements.

It says that the distribution of the averages will form a “normal” or Gaussian distribution. This occurs even when the distributions from which the individual measurements are taken are non-normal.

This is important because it means that as long as we average “a large number” of measurements we can assume that the resulting value comes from a normal distribution, and apply standard statistical tests to it (e.g. 95% of measurements will be within 2 standard deviations of the true average of the distribution).

**Test it!**

i) Generate a list of 10,000 values generated by the random.expovariate function, using k=0.2

ii) Histogram the resulting distribution and find its mean value. Describe the shape of the distribution.

iii) Write a function:

mean\_of\_N\_expovariate(k,N)

that averages N values generated by the random.expovariate function with value k.

iv) Generate a list of 10,000 values from your function with N=2 and k=0.2.

Find the mean value and comment on the shape of the distribution.

v) Now generate a list of 10,000 values from your function with N=100 and k=0.2.

Find the standard deviation and mean value of this distribution, and comment on its shape.

vi) Now generate a list of 10,000 values from your function with N=400 and k=0.2.

Find the standard deviation and mean value of this distribution, and comment on its shape. Comment on how these compare to your answer from (5)

vii) Check that 95% of the list generated in part (6) lie within 2 standard deviations of the mean, as expected if it is a normal distribution.