Using Python for Simulations in Mathematics

Thomas Everth everth t@mbas.ac.nz

In the past students would have used calculators or MS-Excel or Google Sheets for the generation of random numbers for the purpose of simulations. In the "real world" coding in Python or other programming languages is used for simulations.

Using Python for simulations is ideal. The language is easy to learn and with a few lines of code simulations can be programmed by students. It not only teaches them simple coding in Python but also procedural thinking. Buy programming in Python students can also produce evidence of using technology for simulations and their programming code and the implicit mathematical concepts can form part of the evidence of their assessed work.

Python and Repl.it

In order to make Python coding accessible to all students on all devices I selected the online coding framework Repl.it

Website: www.repl.it

Repl.it offers coding a many programming languages. Python is the easiest for students to learn and also is supported by a wide ranging and fast growing programming community.

There is a plethora of information on the Internet about Python programming.

Before showing some sample simulations, and tips on using repl.it first two important libraries that you can use within Repl.it

numpy

Python offers also many excellent deep libraries that are of interest to mathematicians. Foremost this is the numpy library. The documentation for numpy is here:

Numpy documentation https://docs.scipy.org/doc/numpy/index.html

Of particular interest within the vast array of functions in numpy for simulations are the random sampling functions: https://docs.scipy.org/doc/numpy/reference/routines.random.html

In the coding examples that follow three of these functions are used:

```
#draw one uniform random number n between 1 and 30
n = np.random.random_integers(1,30)
```

```
#draw one normal distributed random number as integer of mean 15 and standard deviation of 5
n = int(np.random.normal(15, 5))
# without the int(…) function around this, the result would be floating point numbers
```

```
#draw one Poisson distributed random number around the expectation interval of 5
n = np.random.poisson(5)
```

See the numpy documentation for all the parameter that are available with these functions.

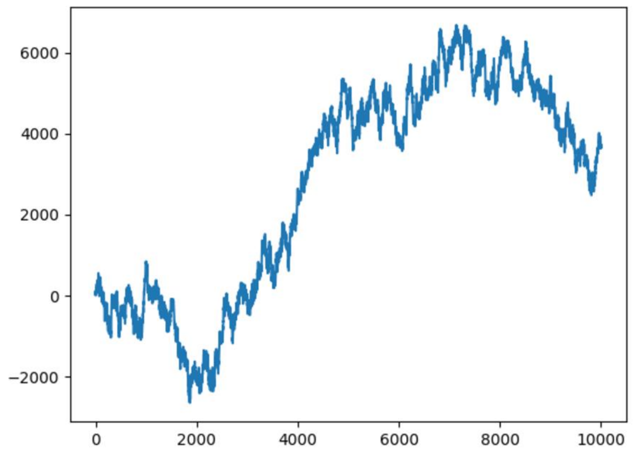
mathplotlib

The mathplotlib library is a great addition for making plots from python data.

See: https://matplotlib.org/

Here is an example of a plot generated by mathplotlib in Repl.it with a single line of code
The ypos positions are an array of 10,000 integers generated by a random walk.

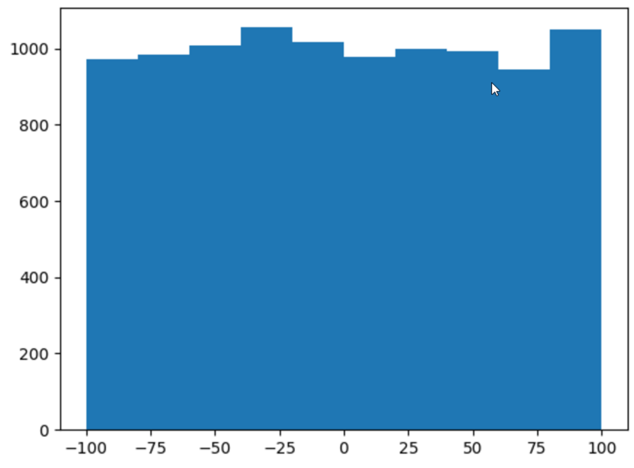
Code: ax.plot(ypos)



A host of parameters can be used to add elements

Here is a one line code that makes a histogram from an array with 10,000 random steps that were done to make a graph like the one above with 10 buckets

Code: bx.hist(steps_done, 10)

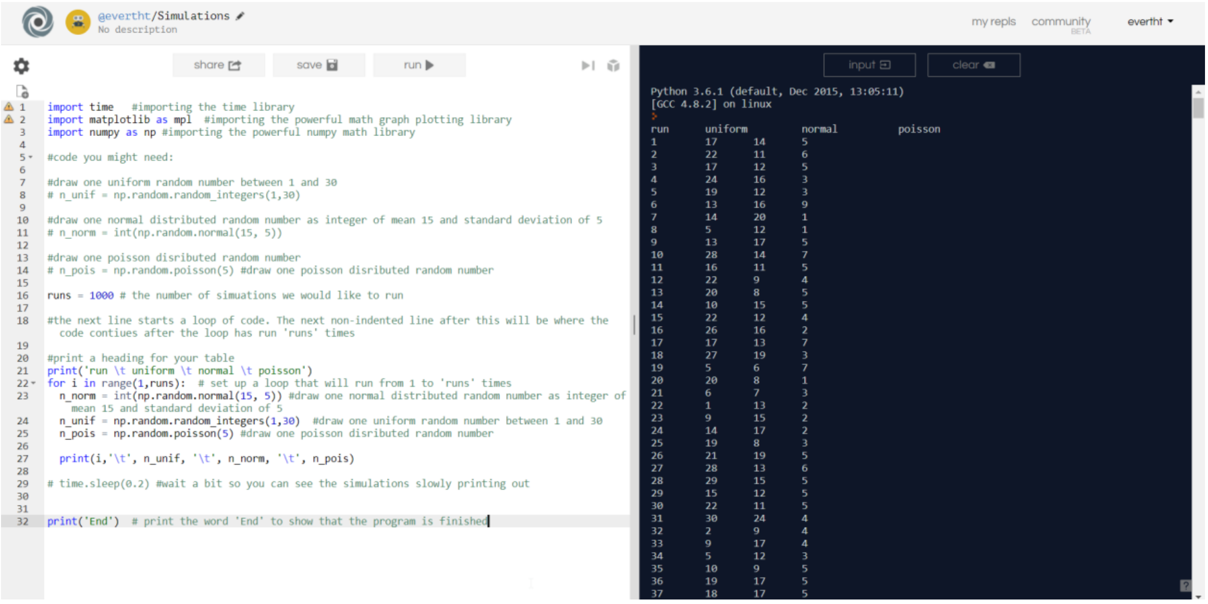


Tips for using Repl.it

When you or your students start using Repl.it you must first make a free user account. This allows you to get back to previous projects.

When you make a new project you should chose the Python3 programming language in order to be compatible with the examples here.

This is the Repl.it window:



On the right, the back screen, is the “console”. Here you see standard output that your program prints and you can also enter values if your program asks for an input during its execution.

On the left is your program. It can have one or more files. All your work is saved whenever you press CTRL-S or the save button.

Results of your program such as the table of numbers there that are printed to the console, can be copied and then pasted into a local document, for example a spreadsheet.

When printing tabular data from your program use the TAB field delimiter. Then your output can be directly pasted into an Excel spreadsheet.

To print output from Python to your console use the print command.

Here are two examples of printing:

print(‘Hello World’)   #will print the infamous Hello World message

print(x,’\t’,y)   #will print the value of variables x and y delimited with a TAB character between

Output files and graphics

You can also write to and read from files that are kept in your project directory

Click the Add new file button above your code top left corner to put your project into a multi file mode. You can then from within your code make new files. These could also be images of plots that can made with one line of code from your mathplotlib.

To make a png file of a graph in mathplotlib you use the savefig function in the mathplotlib:

These four lines of code will generate a plot of the array ypos and a histogram of the array steps_done and then save these plots as two separate png image files. They will appear as tabs in your Repl.it project. From there you can right click and use the “save image as….” command.

```
ax.plot(ypos) #plot a graph of the ypos array into the plot ax
bx.hist(steps_done, 10) #plot a histogram of the steps into the plot bx
fig.savefig('graph.png')  # save the ypos plot fig as graph.png
fig2.savefig('hist.png')  # save the histogram of steps as hist.png
```

Examples for simple python programs

Random sampling demo

```
1  import numpy as np
2
3  print('run \t uniform \t normal \t poisson')
4
5  for i in range(1,100):
6    n_norm = int(np.random.normal(15, 5))
7    n_unif = np.random.random_integers(1,30)
8    n_pois = np.random.poisson(5)
9
10   print(i,'\t', n_unif, '\t', n_norm, '\t', n_pois)
11
12 print('Finished')
13
```

This program makes a table of 100 random samples for three different distributions.

The table can be copied and directly pasted in the Excel or into NZGrapher for making distribution plots to look at the distributions.

Line 1 imports the numpy library

Line 3 prints a header for the table with ‘\t’ tab delimiters

Line 5 starts a for loop. The iteration variable i is iterating through the values 1 to 99

The range construct that python uses makes a list of all integers between the two values given, including the fist but excluding the last. You can also use the shortcut range(100) to make a list of values from zero to 99.

The next three lines draw one random number each from three different distributions.

Line 10 prints these out.

Line 12 Lets you know that the program is finished.

Lego toy collection

This is the simulation of the Lego toy collection task. Nulake Simulations 2.13 page 12

Visits to a shop are made that hands out one of 6 different lego toys at random to each shopper.

How many visits will it take on average to collect all 6 toys.

```
1  import numpy as np
2
3  # Lego toys collection
4
5  sumvisits = 0
6
7  print ('run \t tries \t average')
8
9  for gamenum in range(1,101):
10
11   collected = [0 for x in range(6)]
12   visits = 0
13
14   while True:
15     visits += 1
16     toy = np.random.random_integers(0,5)
17     collected[toy]=1
18     if (0 not in collected):
19       break
20
21   sumvisits += visits
22
23   average = float(sumvisits)/gamenum
24   average_str = format(average, '.2f')
25
26   print (gamenum,'\t',visits,'\t',average_str)
27
28
29
30 print('Finished')
31
```

Line 9 starts a loop over 100 games from 1 to 100 to try to collect all 6 toys

Line 11 sets up an array of 6 fields and fills these with zeros. The index into this array can have values from zero to 5.

Line 14 sets up an infinite while loop to wait until we have all toys collected

Line 15 the visits variable counts how many visits it will take us to collect all 6 toys

Line 16 draws a random integer from zero to 5.

Line 17 sets that toys field to ‘1’ and marks it has been collected

Line 18 tests if there is still a zero in the toy collection, meaning that there is still at least one toy missing in the collection. If not, the infinite while loop from line 14 is broken.

The final number of visits for this game is added to a grand total of all visits. Then the average number of visits is calculated for all games so far.

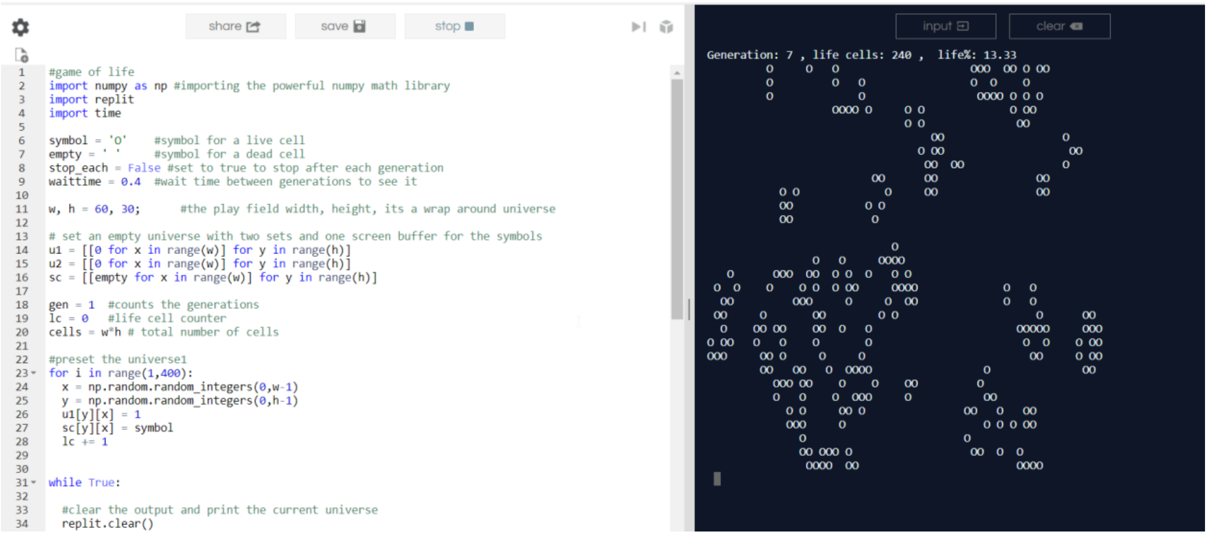
Line 26 prints the current state of the game.

The final table is tab delimited and can be exported for graphing or further work.

Other simulations

I have made a few other simulations that you can share by using the URL provided.

Game of life: https://repl.it/JZ4V/35



Random walker in one dimension https://repl.it/JPJu/8

This uses the mathplotlib to make graphs directly in Repl.it

main.py | graph.png | hist.png

```python
import matplotlib.pyplot as plt
from random import randint

# first figure for the graph of the points
fig = plt.figure()
ax = fig.add_subplot(111)

#second figure and subplot for the histogram
fig2, bx = plt.subplots()

#this program observes the path of a random walking 'animal' in 1 dimension (Y axis)
#the animal walks in each iteration of the simulation a random number of steps along the Y axis
#the steps are: steps = randint(-bound, bound)
ypos = []   # array to store the points where the animal was last
steps_done = []  #array were we remember the steps that were taken
last = 0  #the last known position of the animal
bound = 100  #the size of each step is a random number between -bound to bound

for i in range(0, 10000):  #main loop, run 10000 times
  steps = randint(-bound, bound)  #draw a random number of steps
  last += steps   #modify the current position of the animal from the last
  ypos.append(last)  #append this position to the array of positions
  steps_done.append(steps)  #remember the steps we took

ax.plot(ypos) #plot a graph of the ypos array into the plot ax

bx.hist(steps_done, 10) #plot a historgam of the steps into the plot bx

fig.savefig('graph.png')  # save the ypos plot fig as graph.png
fig2.savefig('hist.png')  # save the historgam of steps as hist.png

print('Finished, check the grapg.png on the panel on the left')
```

Distributions: https://repl.it/JUou/14

This program simply demonstrates three useful random sampling distributions

```python
import time   #importing the time library
import matplotlib as mpl  #importing the powerful math graph plotting library
import numpy as np #importing the powerful numpy math library

#code you might need:

#draw one uniform random number between 1 and 30
# n_unif = np.random.random_integers(1,30)

#draw one normal distributed random number as integer of mean 15 and standard deviation of 5
# n_norm = int(np.random.normal(15, 5))

#draw one poisson disributed random number
# n_pois = np.random.poisson(5) #draw one poisson disributed random number

runs = 1000 # the number of simuations we would like to run

#the next line starts a loop of code. The next non-indented line after this will be where the
  code contiues after the loop has run 'runs' times

#print a heading for your table
print('run\tun\tnrm\tps')
for i in range(1,runs):  # set up a loop that will run from 1 to 'runs' times
  n_norm = int(np.random.normal(15, 5)) #draw one normal distributed random number as integer of
    mean 15 and standard deviation of 5
  n_unif = np.random.random_integers(1,30)  #draw one uniform random number between 1 and 30
  n_pois = np.random.poisson(5) #draw one poisson disributed random number

  print(i,'\t', n_unif, '\t', n_norm, '\t', n_pois)

# time.sleep(0.2) #wait a bit so you can see the simulations slowly printing out


print('End')  # print the word 'End' to show that the program is finished

```

```
Python 3.6.1 (default, Dec 2015, 13:05:11)
[GCC 4.8.2] on linux

run     un      nrm     ps
1       18      12      8
2       4       11      4
3       29      18      8
4       11      13      8
5       19      26      3
6       15      10      5
7       25      10      4
8       23      8       4
9       24      22      7
10      23      12      9
11      23      12      9
12      14      11      2
13      1       16      5
14      16      8       6
15      23      14      2
16      21      15      8
17      15      12      2
18      9       14      3
19      2       15      4
20      9       18      5
21      2       19      1
22      29      10      3
23      12      19      1
24      27      10      5
25      12      13      5
26      15      13      5
27      16      12      1
28      1       11      4
29      9       17      3
30      8       4       4
31      27      9       4
32      7       20      5
```

I hope this collection of Python code will provide some ideas for your own simulations

Best!

Thomas Everth

evertht@mbas.ac.nz