|  |  |
| --- | --- |
| **LAB 4: Math in Python** | Name: Lex Baker  Date: 9/13/2022 |

**Late Explanation - I got caught up with the Hackathon and thought our labs could be turned in before the next lab, not the day before. Here is my Word document and code for Lab 4.**

Any programming language can also be used as a calculator and Python has some libraries that also make it almost as powerful as the commonly used programs Mathematica and Matlab. However, unlike those programs, Python and it's libraries are free. Python + Numpy + Matplotlib has all the functionality of a graphing calculator plus more. This LAB will get you familiar with some of the basic as well as some of the more advanced ways to use Python for math.

1. What are the built in math operators that you've seen already? Type in the following lines of code in the interactive prompt and explain what each of these symbols do in python.

>>> 2\*\*10; **# The \*\* operator is for exponentials. This is 2 to the power of 10**

>>> 100/3; **# Normal division, will return a float**

>>> 100//3; **# Integer division, will floor (round down) the result and return an int**

>>> 100%3; **# Modulus operation, will return the remaining whole number after dividing the first number by the last**

1. Type the following lines of code into the editor, save, and run them. Play around with this until you can explain what augment assignment operators do. Why might this be useful?

date = 16;

print("Today is June", date);

date +=1;

print("Tomorrow is June", date);

weight = 30;

print("My dog weighs", weight, "pounds on earth.");

weight /= 6;

print("My dog weights", weight, "pounds on the moon.");

**# Augment assignment operators set the variable equal to the original value (plus/minus/multiplied by/whatever operator is used) the value on the right side of the augment assignment operator.**

1. As you saw in the example code on LAB 2, you can also import that math module that comes with every version of python to have many more mathematical functions at your disposal.
   1. import the math module in your interpreter:  
         
       >>> import math;

Now try the following lines:

>>> print("The value of e is", math.e);

>>> math.log(math.exp(2));

>>> math.sin(math.pi/2);

>>> math.sin(math.pi);

Why is this not zero? **Because math.pi is an approximation stored in a float, and even though pi is infinite, the float is not.**

* 1. A comprehensive list of functions in the math library can be found here: <https://docs.python.org/3.3/library/math.html>. Look around there and familiarize yourself with what functions are available. Here are some more useful ones:

>>> math.atan(1.73205);

>>> math.hypot(3,4);

>>> math.degrees(math.pi/4);

>>> math.asinh(1);

>>> math.factorial(5);

>>> math.sqrt(8);

>>> math.ceil(3.8);

>>> math.pow(2,10);

>>> math.log(8,2);

>>> math.log10(100);

1. Now import the random module. Here is the example you saw in LAB 2.

>>> import random;  
 >>> firstRandom = random.randint(1,10);

What does randint do? Will it ever generate the number 1? What about 10?

**It generates a random number between 1 and 10, inclusive, so it could return both 1 and 10.**

What does the following line do? Will randrange ever generate the number 10?

>>>random.randrange(2,10,2);

**It will not ever generate 10, as it generates a random even number between 2 and 10, right-exclusive. (So 2, 4, 6, 8)**

Now look at the documentation online (<https://docs.python.org/3.3/library/random.html>) to try and find how to generate random numbers as listed below:

1. Pick a random floating point number between 0 and 1.

**random.random()**

1. Pick a random integer between 5 and 9.

**random.randint(5, 9)**

1. Pick a random floating point number from a uniform distribution from 2 to 3.

**random.random() + 2**

1. Pick a random floating point number from a Gaussian distribution with a standard deviation of 3 and a mean of 1.

**random.gauss(1, 3)**

**Quick Check** (before moving on to the libraries)

1. Use the factorial function in the math library to answer the following question:

What is the probability of drawing a hand of 5 cards from a standard 52 card deck that contains the 2 of clubs?

# Chances of pulling the two of clubs when drawing 5 cards from a deck

# There is 1 card out of 52 total cards that we want

# There are 51 cards we don't care about

# We are drawing 5 cards, 1 of which needs to be the one we want

# For non-replacement probability, we use hypergeometric distribution

# Math is (1 choose 1) \* (51 choose 4) / (52 choose 5)

num\_1 = math.factorial(1) / (math.factorial(1) \* math.factorial(1 -1))

num\_2 = math.factorial(51) / (math.factorial(4) \* math.factorial(51 - 4))

denom = math.factorial(52) / (math.factorial(5) \* math.factorial(52 - 5))

print(num\_1, num\_2, denom)

prob = (num\_1 \* num\_2) / denom

print("Probability of drawing the two of clubs when drawing 5 cards from a deck is ", round(prob \* 100, 4), "%", sep="")

**The probability of drawing the two of clubs in a 5 card hand out of a standard deck of cards is 9.6154%**

1. **Symbolic math.** As you know, there are times when you want the exact answer instead of just a decimal approximation (remember how sin(pi) didn't evaluate to exactly 0 in exercise 3?). The library SymPy can help with this. Many of the commands are similar to what you would find with Mathematica. SymPy (and NumPy which we'll see in the next exercise) has its own version of many of the math functions from the math module. However, the SymPy versions must be used when using other functions in SimPy to achieve the desired result.  
   1. Let's try to fix the problem from exercise 3. Enter the following lines of code:

>>> import sympy as sp;

>>> sp.sin(math.pi);

>>> sp.sin(sp.pi);

Which version of pi allows you to calculate sine exactly?

**The sp.pi version**

* 1. Now type in the following lines of code and see what each of them do:

>>> sp.sqrt(8)

**Finds the square root of a number**

>>> x,y = sp.symbols('x y')

**Turns python variables into sympy variables**

>>> expr = x + 2\*y

**Creates an expression which equals x + 2\*y**

>>> expr + 1 - 2\*y

**Nothing changes**

>>> sp.expand(x\*(x+1))

**Calculates and expands the function to (x^2 + x)**

>>> sp.factor(x\*\*2 + 2\*x\*y + y\*\*2)

**Factors the given equation**

>>> sp.simplify(sp.sin(x)\*\*2+sp.cos(x)\*\*2)

**Simplifies the equation (this equation simplifies to 1)**

1. Now look up the documentation (<http://docs.sympy.org/latest/tutorial/index.html>) to figure out how to differentiate, integrate, take limits, and solve differential equations symbolically. Write down one line of code showing an example of each.

**Differentiate: diff(cos(x), x)**

**Integrate: integrate(cos(x), x)**

**Limits: limit(sin(x)/x, x, 0)**

**Solve: f = symbols('f', cls=Function)**

**dsolve(Eq(f(x).diff(x, x) - 2\*f(x).diff(x) + f(x), sin(x)), f(x))**

You may have noticed that I keep asking you to look up documentation online. Although it might seem annoying, this is what you'll need to do when coding on your own. You often need to search through libraries to find one that has the exact commands you want and syntax is important.

1. Probably the most useful library you will use with python is NumPy. Is is made to help with large vectors and matrices. Here is some documentation that you can reference as needed (<http://www.engr.ucsb.edu/~shell/che210d/numpy.pdf>). Don't forget to use NumPy's versions of common functions (sin, pi, etc.) when working with NumPy objects.
   1. You can create something similar to mathematical vectors by using the array object in NumPy. There are even pre-built functions to calculate the dot and cross products.

>>> import numpy as np

>>> x = np.array([1,5,2])

>>> y = np.array([7,4,1])

>>> x+y

>>> np.dot(x,y)

>>> np.cross(x,y)

* 1. Many operations on arrays are done element-wise. Write your best guess for how you would create a vector where each element is the square root of the corresponding element in another vector. Test this out and don't forget to call np.sqrt instead of math.sqrt. Can you square each element of an array? What about taking the sine of it?

**print(np.sqrt(nparr))**

**print(np.square(nparr))**

**print(np.sin(nparr))**

* 1. There are multiple ways to create matrices and many built-in functions that you can apply to them. Run the following lines on your command line and make sure you understand the syntax for each line.

>>> np.zeros((3,3));

>>> m1 = np.array([[1,3],[5,7]]);

>>> m2 = np.identity(2);

>>> np.linalg.det(m1);

>>> np.linalg.eig(m1);

>>> np.linalg.inv(m1));

>>> m1\*m2;

What does m1\*m2 do? Notice that using the multiplication sign triggers elementwise multiplication as well. You can get around this in two ways. Either use the dot function on two multidimensional arrays or create matrix objects.

**m1 \* m2 multiplies each element by its corresponding place in the second array**

>>> m1.dot(m2);

>>> a = np.matrix([[1,3],[5,7]]);

>>> b = np.matrix([[1,0],[0,1]]);

>>> a\*b

1. Use NumPy to calculate the mean, standard deviation, and variance of the following data set: 2, 3, 5, 9, 9, 10, 11

**nums = np.array([2, 3, 5, 9, 9, 10, 11])**

**print("average =", np.average(nums))**

**print("variance =", np.var(nums))**

**print("std =", np.std(nums))**

1. What do the following two commands do?

>>> np.poly([2,3])

**Finds the coefficients of equation of the roots given**

**Returns: 1, -5, 6**

>>> np.roots([1,2,1])

**Finds the roots of an equation by the coefficients given**

**Returns: -1, -1**

**8. Plotting points.** Matplotlib is a library that creates high quality plots. This link has some examples of what can be done with Matplotlib: <http://matplotlib.org/users/screenshots.html>. Create a new file and import NumPy and matplotlib as shown below. The last two words of the import statements allow you to use the functions without typing the entire library name in front of them.

import numpy as np

import matplotlib.pyplot as plt

1. Save and run the following code.

xVals = np.array([1,2,3,4]);

yVals = xVals\*\*2;

plt.plot(yVals);

plt.show();

The show command is necessary to actually see the plot. Try running the program again without that line. Now replace the plot command you already have with a new one.

plt.plot(xVals, yVals);

As you saw, the default x values are integers beginning with 0. You can alter those however you want as you showed with the line above. Now suppose you want the plot to look different. Replace the plot line in your code with each of the following commands one at a time and try to figure out what each of the characters in the third argument does. There are lots of possibilities; see if you can find more!

plt.plot(xVals, yVals, 'r');

plt.plot(xVals, yVals, 'ro');

plt.plot(xVals, yVals, 'rx');

plt.plot(xVals, yVals, 'g+');

plt.plot(xVals, yVals, 'c\*')

**The first letter sets the color, the second letter sets the shape of the point and removes the lines between the points.**

1. You can also adjust the axes, add labels, add a title, and a legend, and just about anything else you'd want to do. Add label='Points' inside your plot command (see first line below) and add the following lines to your program after the plot command. What does each line do?

plt.plot(xVals, yVals, 'c\*', label='Points')

**Sets up values, point shape and color, and label**

plt.axis([0,5,0,18]);

**Sets x-axis from 0 to 5 and y-axis from 0 to 18**

plt.title("Quadratic");

**Sets title**

plt.grid(True);

**Enables gridlines**

plt.legend(loc='upper left');

**Moves legend to top left**

plt.xlabel('x axis title');

plt.ylabel('y axis title');

**These two set the axis labels**

plt.show();

1. Go to the matplotlib documentation (<http://matplotlib.org/api/pyplot_summary.html>), search for figtext and figure out how to add a line of text somewhere on your figure.

**plt.figtext(0.5, 0.5, "test")**

**9. Plotting a function**

1. **Many x values.** Say you want to plot 100 points without writing out the numbers from 1-100. NumPy has a function called arange() that will do this for you. We'll discuss more about a similar function called range() later, but for now just know that you enter the first number, last number, and step size and it will give you all the number in between. Comment out your previous plots and type the following expressions.   
     
    x = np.arange(0, 5, 0.2)

plt.plot(x,x\*\*2)

What happens as you change the step size in np.arange?

**It changes the increment in between each number from 0 to 5**

Now add the following line after your current plot statement.

plt.plot(x,x)

Can you add x^3 as well? What about Cos(x)?

**plt.plot(x, x\*\*3)**

**plt.plot(x, np.cos(x))**

1. **Subplots and other functions.** There's an easy way to see multiple plots at once. Open a new file and type in the following code to begin to explore subplots.

import numpy as np

import matplotlib.pyplot as plt

t = np.arange(0., 5., 0.2)

plt.subplot(311) #num rows, num columns, fig number

plt.plot(t, t, 'r--')

plt.title("linear")

plt.subplot(312)

plt.plot(t, t\*\*2, 'bs')

plt.plot(t, t\*\*3,'g^')

plt.title("quadratic and cubic")

plt.ylabel("subplot 2 y axis")

plt.subplot(313)

plt.plot(t, np.cos(t),'k')

plt.axis([0,np.pi,-1.5, 1.5])

plt.grid(True)

plt.show()

Now alter this code to make a figure with 4 subplots, one showing a sine function, the second showing a cosine function, the third showing an exponential function, and the fourth showing a logarithm. Play around with the axes, grid, and titles until the plot looks the way you want it.

**t = np.arange(0., 5., 0.2)**

**plt.subplots\_adjust(left=0.1,bottom=0.1,right=.9,top=.9,hspace=1)**

**plt.subplot(4,1,1) #num rows, num columns, fig number**

**plt.plot(t, 2\*\*t, 'bs')**

**plt.title("exponential")**

**plt.subplot(4,1,2)**

**plt.plot(t, np.sin(t), 'r--')**

**plt.title("sine")**

**plt.subplot(4,1,3)**

**plt.plot(t, np.cos(t),'k')**

**plt.title("cosine")**

**plt.subplot(4,1,4)**

**plt.plot(t, np.log(t),'k')**

**plt.title("logarithmic")**

**plt.show()**

1. Extension: if you have extra time after doing the next problem, go back and figure out how to change your code from part b to print out 4 figures instead of 4 subplots

**10. Histograms.** You can also plot histograms. Here is code to show a histogram of a normal distribution:

import numpy as np

import matplotlib.pyplot as plt

mu, sigma = 100, 15

x = mu + sigma \* np.random.randn(10000)

plt.hist(x)

plt.axis([0, 200, 0, 5000])

plt.grid(True)

plt.show()

The function np.random.randn(10000) returns 10,000 random numbers from the "standard normal" distribution.

1. Adjust the axes as you'd like and add x and y axis labels and a title to the histogram.

**mu, sigma = 100, 15**

**x = mu + sigma \* np.random.randn(10000)**

**plt.hist(x)**

**plt.title("Graphing Numpy Random Frequency")**

**plt.axis([0, 200, 0, 5000])**

**plt.xlabel("Values")**

**plt.ylabel("Frequency")**

**plt.grid(True)**

**plt.show()**

1. Now create your own histogram given the following data. Calculate and then print out the standard deviation and mean on the graph itself.

scores = [71, 98, 80, 85, 85, 93, 74, 70, 88, 80, 91, 83, 82, 84, 84, 84, 84, 82, 80, 88, 79, 95, 87, 85, 90]

**scores = [71, 98, 80, 85, 85, 93, 74, 70, 88, 80, 91, 83, 82, 84, 84, 84, 84, 82, 80, 88, 79, 95, 87, 85, 90]**

**plt.hist(scores)**

**plt.title("Graphing Histogram")**

**plt.axis([60, 100, 0, 10])**

**plt.xlabel("Values")**

**plt.ylabel("Frequency")**

**plt.grid(True)**

**plt.figtext(0.2, 0.8, "mean = " + str(np.average(scores)))**

**plt.figtext(0.2, 0.7, "std = " + str(np.std(scores)))**

**plt.show()**