Lesson 2

Ryan Branagan

January 22, 2019

Problem 1

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw2_p1.py
#1/10/19

n=5 #This value determines the first N integers to square and add together
a=0 #We need a to start at a value of zero but it DOES NOT go in the loop
for i in range(1,n+1):
    a = a + i**2
    #print(a) #I used this for troubleshooting and don't want to get rid of it
print('The sum of the first',n,'integers squared is',a)
```

This code does output the correct answer for the sum of the first N integers squared. I had some difficulty at first because I put a = 0 in the loop which kept on clearing the value of a in each iteration. Once it was taken out of the loop, the code worked.

Problem 2

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw2_p1.py
#1/10/19

a=0 #This is needed to start with 1 and be able to even run the code with "a" assigned
#to a value that will not affect the output
for i in range(1,100):
    if i % 3 == 0 or i % 5 == 0:
        a = a + i
print('The sum of the numbers less than 100 divisible by 3 or 5 is',a)
```

The answer I should get is 2318 which is the answer that this routine outputs.

Problem 3

```
#Ryan Branagan
#Collaborators: I got help from Jack Featherstone
#Branagan_hw2_p3.py
#1/15/19

#import what I need
import numpy as np
import pylab as p

#Initial conditions
v0 = 300. #Initial speed (m/s)
theta = 30 #Angle with respect to the x axis
tf = 31 #How long in seconds it should plot (s)

#Finding the x and y components of the initial velocity (m/s)
vx = v0 * np.cos(theta * (np.pi/180))
vy = v0 * np.sin(theta * (np.pi/180))

#Equations for our x and y positions as a funciton of time (m)
#This is what Jack helped me with
g = 9.81
def xp(t):
    return vx * t
def yp(t):
    return vy * t - 0.5 * g * t**2

#Making an array of times to plug into our equation (s)
times = np.linspace(0,tf,2*tf+1) #I want to check every half second so 2*tf+1 is necessa

#Make an array of x positions over time (m)
x = np.zeros(2*tf+1)
for a in np.arange(len(x)):
    x[a] = xp(times[a])
#print(x)

#Make an array of y positions over time (m)
y = np.zeros(2*tf+1)
for a in np.arange(len(y)):
    y[a] = yp(times[a])
#print(y)

#Make a plot of the position of the projectile over time
```

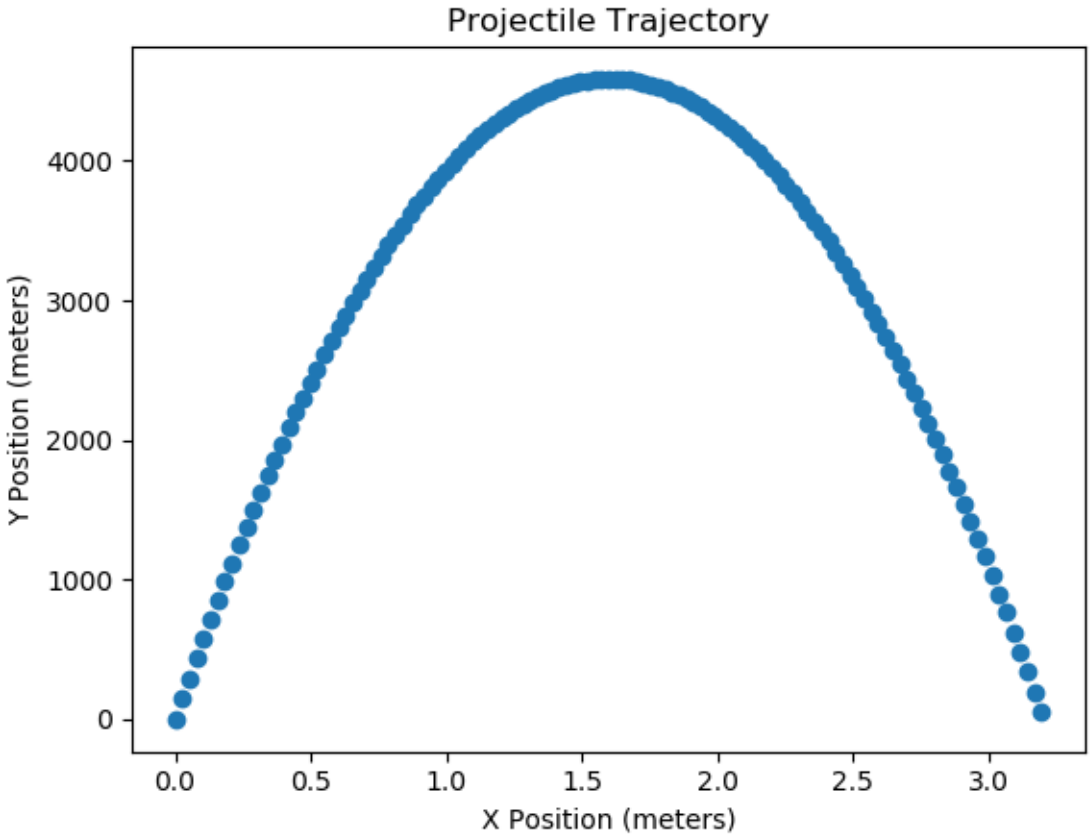

Figure 1: Output for $v_0 = 300(m/s)$,$\theta = 89.99$ degrees, and is plotted over 61 seconds

```
p.plot(x,y,'o')
p.xlabel('X Position (meters)')
p.ylabel('Y Position (meters)')
p.title('Projectile Trajectory')
p.show()
```

This routine takes in three input parameters and outputs a graph of projectile motion. The input parameters are initial speed, angle above the x axis that the projectile is shot at, and the time interval that should be plotted. In the code I picked reasonable parameters but for the graph I used the conditions of a bullet being shot upwards. I was curious if a bullet being shot upwards could actually hit someone next to the shooter. If the person is sufficiently close to the shooter and the shooter has the right amount of error then it is possible it seems.

Problem 4

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw2_p4.py
#1/15/19

N = 25 #This is the input integer

a = [] #This is an empty list

for i in range(N):
    if i % 3 == 0 or i % 5 == 0:
        a.append(i) #Instead of using a list full of zeroes, I wanted to just append the

print(a)
```

This was fairly simple, test which numbers are divisible by three or five and add them to the list up to N-1. This outputs a list of all the numbers divisible by three or five in order.

Problem 5

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw2_p5.py
#1/16/19

N = 10 #This is the input integer

a = [] #This is an empty list
y = 0 #Just like in previous routines this is needed to start our adding at first to sta

for i in range(1,N+1):
    for x in range(1,i+1): #Nested for loop that takes input from the first for loop to
       y = y + x
    a.append(y) #Once the nested for loop has added the first N_i integers together we a
    y=0 #We need to clear y in each step so we can start over from 1 instead of the resu

print(a)
```

This routine basically takes something we've done previously, adding together the first N integers, and combining it with the idea of putting that into a list. The output is as expected, a list where each successive number in the list is the first N_i integers added together up to the first N integers added together.

Problem 6

```
#Ryan Branagan
#Collaborators: N/A
#Branagan_hw2_p6.py
#1/17/19

import numpy as np
import pylab as p

#Input values
a = 0.1
b = 2.
points = 1000

#Define our function of sin(1/x)
def sinoinv(x):
    y = np.sin(x**(-1))
    return y

#Create an array of x values
xarray = np.linspace(a,b,points)

#Method with loop
output1 = np.zeros(points)
for i,x in enumerate(xarray):
    output1[i] = sinoinv(x)

#Method without loop
output2 = sinoinv(xarray)

#Plotting
p.plot(xarray,output1,label="With a for loop")
p.plot(xarray,output2,label="Without a for loop")
p.legend()
p.show()
```

This routine outputs a graph of $\sin(1/x)$ using two different methods that give the same output. I prefer the method of directly using lists to do math, it is more familiar for me because it is similar to how I did things in Mathematica. With a sufficiently high number of points the output looks like a clean graph of $\sin(1/x)$.

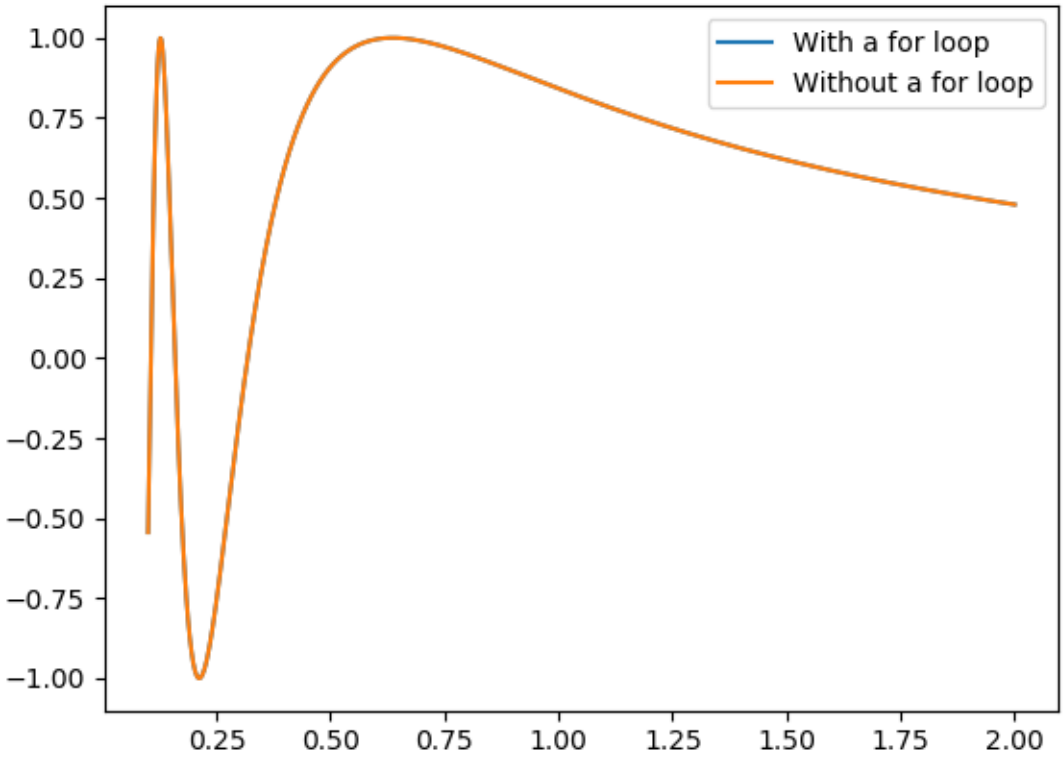

Figure 2: Plotting 1000 points between 0.1 and 2.