**Name: Session:**

**Programming I**

**Scientific Programming**

**Lab Exercise 12.2.2020**

**DNA Matching**

In this activity, you will match a DNA sequence inside another DNA sequence.

1. Start IDLE and create a new window
2. Create two variables and initialize them with DNA sequence

strand = "AACAACTTCGTAAGTAA"

look = "TCGT"

1. Test for match and report

location = strand.find(look)

if location == -1:

print ("Sequence not found")

else:

print ("Sequence found at location ", location)

1. Run the program and a match should be found at location 7. Remember lists start counting with element 0.

Now let’s do something to make this a little more interesting. We will generate a DNA sequence of 1000000 bases and search for a sequence within that sequence.

1. Start by saving your previous program under a different name such as dna2.py
2. Since we are creating random DNA strings we need our random number generator
3. Next we will initialize some variables. The variable bases should be a list for the four bases that we will add randomly to our generated strand. Remember the four DNA bases are where genetic information is encoded as a sequence of nucleotides (guanine, adenine, thymine, and cytosine) recorded using the letters G, A, T, and C.
4. We will now build our DNA strand
5. Have the user enter the DNA sequence to search for
6. Conduct the search
7. Record the search results

Now we are going to validate the DNA sequence the user has entered (i.e. it only contains the characters ‘C’, ‘G’, ‘A’, and ‘T’)

1. Save your current file as dna3.py
2. Write a validateSequence function that will return True if all bases provided by the user are valid and False otherwise.
3. Your function should be passed a string to validate

**Submit your source code for dna3.py.**

**Curve Fitting**

You will find all of the files you require for this activity on the server in the following folder:

\\Ada\Data Files\Programming I\Starter Code\Lab Exercise 12.2.2020\Curve Fitting

In this exercise we are going to plot the data resulting from an experiment where we hang various weights from a spring and measure the displacement. Our data looks like this.

Distance(m) Mass(kg)

0.0865 0.1

0.1015 0.15

0.1106 0.2

0.1279 0.25

0.1892 0.3

0.2695 0.35

0.2888 0.4

0.2425 0.45

0.3465 0.5

0.3225 0.55

0.3764 0.6

0.4263 0.65

0.4562 0.7

0.4502 0.75

0.4499 0.8

0.4534 0.85

0.4416 0.9

0.4304 0.95

0.437 1.0

1. In order to do this I have written a program to plot this data. On the server, find the file spring.py and springData.txt.
2. Copy spring.py and springData.txt into a folder.
3. Open spring.py in IDLE and run.
4. You should get a nice plot of the data.
5. Take a screen shot of your plot and paste into a word document labeling it **spring**.

Now let us do a curve fit on this data. We are going to start with a simple linear fit that will find the line of form y = mx + b where m is the slope of the line and b is the y-intercept.

1. In order to do this I have written a program to plot this data as well as the line representing a linear curve fit. On the server, find the file spring2.py and springData.txt.
2. Copy spring2.py and springData.txt into a folder.
3. Open spring2.py in IDLE and run.
4. You should get a nice plot of the data. Note the in the caption I have reported the value of k which is the reciprocal of the slope.
5. Take a screen shot of your plot and paste into a word document labeling it **spring2**.

Sometimes a linear equation does not give us the greatest fit. So this time we will use a cubic equation to fit our data. We are going to start with a simple linear fit that will find the line of form y = ax + b where a is the slope of the line and b is the y-intercept and add a cubic fit of an equation of the form

y = ax3 + bx2 + cx = d.

1. In order to do this I have written a program to plot this data as well as the linear and cubic curve fits. On the server, find the file spring3.py and springData.txt.
2. Copy spring3.py and springData.txt into a folder.
3. Open spring3.py in IDLE and run.
4. You should get a nice plot of the data. This time you will see there is both a linear fit as well as a cubic fit.
5. Take a screen shot of your plot and paste into a word document labeling it **spring3**.

In the last run, the cubic fit looked pretty good but can we trust it. To see what is going on a little better, I am going to add a few data points to the springData and see if it still looks as good.

1. In order to do this I have written a program to plot this data as well as both linear and cubic curve fits. On the server, find the file spring4.py and springData.txt.
2. Copy spring4.py and springData.txt into a folder.
3. Open spring4.py in IDLE and run.
4. You should get a plot of the extrapolated data. What does it show?
5. Take a screen shot of your plot and paste into a word document labeling it **spring4**.

As you can see, the data sort of breaks down as the spring is stretched towards its elastic limit. As a final experiment, I will remove the last five data points from the data set and try a linear fit

1. In order to do this I have written a program to plot this data as well as the linear fit. On the server, find the file spring5.py and springData.txt.
2. Copy spring5.py and springData.txt into a folder.
3. Open spring5.py in IDLE and run.
4. You should get a plot with a much nicer fit. Notice the value of k is much smaller so I suspect that this model is much more representative of the behavior of the spring.
5. Take a screen shot of your plot and paste into a word document labeling it **spring5**.

Next we are going to analyze some ballistic flight path data. The data shows the result of 4 trials of the launch of a projectile. The data file looks like this:

Distance (yds) height (ins) height height height

30 0 0 0 0

29 2.25 3.25 4.5 6.5

28 5.25 6.5 6.5 8.75

27 7.5 7.75 8.25 9.25

26 8.75 9.25 9.5 10.5

25 12 12.25 12.5 14.75

24 13.75 16 16 16.5

23 14.75 15.25 15.5 17.5

22 15.5 16 16.6 16.75

21 17 17 17.5 19.25

20 17.5 18.5 18.5 19

15 19.5 20 20.25 20.5

10 18.5 18.5 19 19

5 13 13 13 13

0 0 0 0 0

1. In order to do this I have written a program to plot this data as well as curve that represents a linear and quadratic fit. On the server, find the file ballistic.py and launcherData.txt.
2. Copy ballistic.py and launcherData.txt into a folder.
3. Open ballistic.py in IDLE and run.
4. Which fit do you think is the most representative?
5. Take a screen shot of your plot and paste into a word document labeling it **ballistic**.

Finally, let us look at a method for finding a metric for goodness-of-fit. We are going to calculate something called the coefficient of determination (R2).

1. In order to do this I have written a program to plot this data as well as curve that represents a linear and quadratic fit. On the server, find the file ballistic2.py and launcherData.txt.
2. Copy ballistic2.py and launcherData.txt into a folder.
3. Open ballistic2.py in IDLE and run.
4. For the value of R2 will be between 0 and 1 where 1 is a perfect fit. Notice how close the quadratic fit is to the data (0.9858).
5. Take a screen shot of your plot and paste into a word document labeling it **ballistic2**.

One last aside that I would like to show you is that Python has the ability to generate a variety of probability distributions. In this example I use 3 distributions, Triangular, Gaussian, and Uniform.

1. Copy distribution.py into a folder.
2. Open distribution.py in IDLE and run
3. Take a screen shot of your plot and paste into a word document labeling it **distribution**.