Assignment 3 – Due Friday July 4th , 2021 at 4:00 pm

Each program must start with a multi-line comment as shown below. For Assignment 3 question 1 replace x with 3 and y with 1. Replace each occurrence of DentStew and Stew Dent with your name. Replace yyyy/mm/dd with the date you completed the program.

```
# DentStewAxQy.py
#
# Course:      COMP 1012
# Instructor:  Ramin Soltanzadeh
# Assignment:  x Question y
# Author:      Stew Dent
# Version:     yyyy/mm/dd
#
# Purpose:     The purpose of the program.
#
# var1 - the use / meaning of each variable in the program
```

Begin each program with the following statements.

```
from time import ctime

print('\n---------------------------------------------\n')
```

End each program with the following statements.

```
Print("""
Programmed by Stew Dent
Date: %s
End of processing.""" % ctime())
```

Replace Stew Dent with your real name.

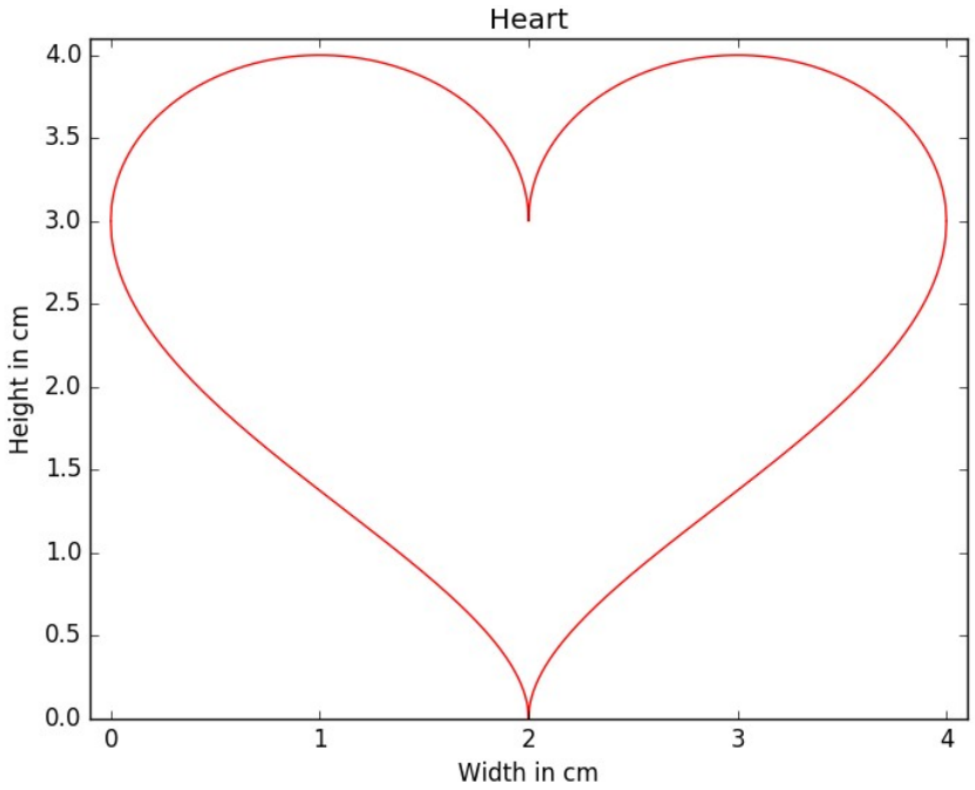
The name of each program should be of the form:
    LastnameFirstnameAnQm
If your name is Stew Dent and the program is for assignment 1 question 1 then the name of the program should be:
    DentStewA2Q1
The corresponding python file would be named:
    DentStewA2Q1.py

Upload your solution on Moodle. Write all your answers in one *.py file and separate them with appropriate comments. Do **NOT** use zip, rar or any other packaging of your files!

Page 1 of 9

In questions 1, 2 and 3 of this assignment you will solve the same problem. In question 1 you must use lists and in question 2 you must use arrays but not vector arithmetic and in question 3 you must use arrays and vector arithmetic. You will use the *timeit* module to determine how much time each program uses to calculate the area of a heart for a large number of intervals and compare the times. To do this, use the trapezoid technique.

For each programming question in this assignment you will use the function named *displayTerminationMessage* from assignment 2.



The heart shown above has a width and height of 4 cm. The points on the boundary of the heart are defined by two equations where x ranges from -2 to +2 for each equation.

$y(x) = \sqrt{1 - (|x| - 1)^2}$ *equation* 1(*for top of heart*)

$y(x) = -3\sqrt{1 - \sqrt{\frac{|x|}{2}}}$ *equation* 2(*for bottom of heart*)

where |*x*| means to take the absolute value of x.

Question 1

You must use lists in this question. Do not use arrays in this question.

The purpose of this question is to write a complete Python program that computes the approximate area of a heart.
Write a function that begins with the following header:

```
def getNonNegativeInt(prompt):
```

Call this function to get input from the user. Valid input is an *int* greater than or equal to zero. The function displays the *prompt* to tell the user what to enter. If the user does not enter valid input display the appropriate error message as described below and display the *prompt* again asking for more input from the user. Repeat this until the user enters valid input. When the user enters valid input return that input.

Error situations:
- If the user presses enter/return without entering anything display the message 'Missing input!'
- If the input causes an exception when passed to *eval* display the message 'Invalid input!'
- If the input is not an *int* display the value of the input and the message 'must be an integer!'
- If the input is less than 0 display the value of the input and the message 'must not be negative!'

Write a function that begins with the following header.

```
def computeXcoordinates(width, intervals):
```

This function is given:
- *width* - the width of the heart
- *intervals* - the number of equally sized intervals along the X axis

Given the width of the heart, and the number of intervals create a list of equally spaced X coordinates from x = -width/2 to x = width/2. Return the list of X coordinates.

Hint: You may find it useful to use list comprehension to create the list of X coordinates.

Write a function that begins with the following header.

```
def computeYcoordinates(xCoords):
```

This function is given:
- *xCoords* - the list of equally spaced X coordinates

Create a list of Y coordinates, one Y coordinate for each element of *xCoords* using equation 1.
Create a list of Y coordinates, one Y coordinate for each element of *xCoords* using equation 2.
Return the two lists of Y coordinates.

Write a function that begins with the following header.

```
def computeArea(intervals, width):
```

This function is given:

- *intervals* - the number of equally sized intervals along the X axis
- *width* - the width of the heart

Call *computeXcoordinates* to get the list of X coordinates. Call *computeYcoordinates* to get the two lists of Y coordinates. Using the two lists of Y coordinates and the trapezoid technique compute the approximate area of the heart. Display the approximate area of the heart to 14 decimal places using exponential format. See the sample run of the program for the exact format of the output.

This function does not return a value.

The main program defines the width of the heart to be 4.
Call getNonNegativeInt to get the number of intervals.
Loop as long as the number of intervals is not equal to zero. In the loop:
- use the *Timer* and *timeit* functions from the *timeit* module to calculate the time it takes to compute the area of the heart
- display the time it took to compute the area of the heart, see the sample run of the program for the exact format of the output
- call getNonNegativeInt to get the number of intervals.

Call *displayTerminationMessage*, as defined in assignment 2, to display the termination message.

To use the *Timer* and *timeit* functions from the *timeit* module to calculate the time it takes to compute the area of the heart use the following statements:

```
t = timeit.Timer('computeArea(intervals, width)',
'from __main__ import computeArea, intervals, width')
computeTime = t.timeit(1)
print("Compute time using lists is %.3f seconds." % computeTime)
```

Do NOT type in the statements given above, cut and paste them from this document!

There are two underscores before and after the word *main* in the above statements.
To use the *Timer* and *timeit* functions import the *timeit* module.

Note: There is NO function named main in this program!

The output from a sample run of the program is shown below.

```
---------------------------------------------------------------------
Enter the number of intervals (0 to quit):
Missing input!

Enter the number of intervals (0 to quit): hello
Invalid input!

Enter the number of intervals (0 to quit): 12.34
12.34 must be an integer!

Enter the number of intervals (0 to quit): -1
-1 must not be negative!

Enter the number of intervals (0 to quit): 1000
The approximate area of the heart is 9.54124947499537e+00 cm^2
Compute time using lists is 0.001 seconds.

Enter the number of intervals (0 to quit): 10000000

The approximate area of the heart is 9.54159265324541e+00 cm^2
Compute time using lists is 11.815 seconds.

Enter the number of intervals (0 to quit): 0
Programmed by Stew Dent.
Date: Fri Jun 18 10:23:30 2021
End of processing.
```

Question 2

Do NOT use vector arithmetic in your solution to question 2.

Modify your solution to question 1 to use arrays instead of lists.

Modify the function named *computeXcoordinates* so that it creates and returns an array of X coordinates. Use one of the functions from *numpy* to do this. There must NOT be any loops in this function.

Modify the function named *computeYcoordinates* so that it converts the two lists of Y coordinates to arrays and returns the arrays.

Modify a print statement in the main program so that instead of displaying "The compute time using lists is" it displays "The compute time using arrays is".

No other changes are allowed.

Test your program using the same number of intervals used for question 1.

Question 3

You must use arrays and vector arithmetic in your solution to question 3.

Modify your solution to question 2 to use vector arithmetic on the arrays.

Modify the function named *computeYcoordinates* so that it uses vector arithmetic to create the two arrays of Y coordinates. Return the two arrays of Y coordinates. There must NOT be any loops in this function.

Modify the function named *computeArea* so that it uses vector arithmetic. There must NOT be any loops in this function.

Modify a print statement in the main program so that instead of displaying "The compute time using arrays is" it displays "The compute time using vector arithmetic is".

No other changes are allowed.

Test your program using the same number of intervals used for questions 1 and 2.

Question 4

Run the three programs that are your solutions to questions 1, 2 and 3 for 10 million intervals. Hand in your answers to the following questions in a Python .py file.

Do NOT use any other type of file for your answer to this question.

a) How long did it take for your solution to question 1 to compute the area of the heart?
b) How long did it take for your solution to question 2 to compute the area of the heart?
c) How long did it take for your solution to question 3 to compute the area of the heart?
d) Which program ran the slowest?
e) Which program ran the fastest?
f) What do you conclude from this? Do not state what you observe, rather state when you think it is appropriate to use lists and when it is appropriate to use arrays in the solution to a problem.

Question 5

The purpose of this question is to write a complete Python program that is given a document and computes the weight of each word and the frequency of each weight for the words in the document *anagrams.txt* which you must download from Moodle.

Insert the following statements near the beginning of your program. Later in the course we will learn what each statement does.

```
def getWords(fileName):
''' Given the name of a file, read and return the words
in the file. '''
# fileName - the name of the file to read
# infile - the descriptor of the file to read
# words - the words read from the file
infile = open(fileName, 'r')
words = infile.read()
infile.close()
return words.splitlines()
```

Write a function that begins with the following header:

```
def computeWeights(words):
```

This function is given a list of strings where each element in the list is a word from a file. Create a list of weights that contains only the unique weights for the words. Create a second list that keeps a count of each unique weight. If a weight is not in the list of weights, add it to the list of unique weights and add a count of 1 to the list that counts the number of unique weights. If a weight is already in the list of unique weights then increment the counter for that weight by one. For example, if the weight 454 is already in the list of unique weights at position 4 then each time after that when a weight of 454 is found in the list of unique weights increase the count at position 4 of the list of counts by 1. Return the list of unique weights followed by the list of counts.

The weight of a word is the sum of the numeric values of each of the letters in the word. If *letter* is a variable that contains one letter of a word then the numeric value of the *letter* is *ord(letter)*.

HINT: In the week 3 notes there is the description of a function that can be used to find the position of a value within a list.

Write a function that begins with the following header:

```
def displayWeights(weights, counts):
```

This function is given:

- *weights* - the list of unique weights
- *counts* - the list of the counts of the number of occurrences of the weights in *weights*

Display a heading as shown in the sample run of the program. Create a new list that contains the unique weights sorted in ascending order. Display the unique weights in ascending order and their corresponding counts one pair of values to a line as shown in the sample run of the program. The weights and the counts are right justified in 6 character positions.

Write a function that begins with the following header:

```
def main():
```

This function:

- calls *input* to get the name of the file from the user
- calls *getWords* to get a list of the words from the file
- calls *computeWeights* to get the lists of unique weights and the list of counts of the number of occurrences of each unique weight
- calls *displayWeights* to display the unique weights and their corresponding counts
- calls *displayTerminationMessage*, as defined in assignment 2, to display the termination message.

The main program, not to be confused with the function named *main*, contains all import statements, the functions and a call to the function named *main*.
A sample run of the program is shown on the next page.

```
---------------------------------------------------------------------
Enter the name of the file of anagrams: anagrams.txt

There are 100 words in the file.

Weight Count
320   3
429   5
434   4
454   4
526   4
536   5
537   6
538   4
539   9
547   4
551   4
553   5
626   3
639   6
646   3
649   8
654   3
663   4
737   4
744   4
852   1
856   2
872   5

The total of the counts is 100

Programmed by Stew Dent.
Date: Fri Jun 18 12:44:09 2021
End of processing.
```