|  |  |
| --- | --- |
| **NAME** | **DUE** |
| **TOMOKI KOIKE** | **OCT 20TH 2019** |

**Every Boiler Engineering Code – Intermediate Level Programming**

**Week 5 – Programming Exercises**

|  |  |
| --- | --- |
| **QUESTION** | **POINTS** |
| **#1** | **/20** |
| **#2** | **/15** |
| **#3** | **/15** |
| **#4** | **/20** |
| **TOTAL** | **/70** |

1. (**20 points, Data Analysis**) In the attachment, you’ll find a text file named ‘1994\_weekly\_Gas\_Average.txt’. The file contains the average gas price for each week in the year 1994. (There are 52 lines in the file.) Using *matplotlib*, write two Python programs that reads the contents of the file then plots the data as a line graph and a bar chart. Be sure to display meaningful labels along the X and Y axes, as well as the tick marks.

**>>CODE**

###  
# AUTHOR : Tomoki Koike  
# DATE : Oct 20th, 2019  
# DESCRIPTION: This program reads a file and creates a bar and line graph based on  
# the data from the text file.  
###  
  
# Main Function  
**def main():** # Reading the file  
 data **=** read\_file**('1994\_Weekly\_Gas\_Averages.txt')** # Creating the line graph for the data  
 create\_lineGraph**(**data**)** # Creating the bar chart for the data  
 create\_barChart**(**data**)**# Import modules  
**import** matplotlib.pyplot **as** plt  
  
# Functions  
# Function to read the data from the file  
**def read\_file(**file**):** # Opening the flle  
 subject\_file **=** open**(**file, **'r')** # Reading the first line of the file  
 line **=** subject\_file.readline**()** # Preallocate the list to store the data  
 data **= []** # Loop to the read all the data from the file  
 **while** line **!= '':** # Appending the data read from the file to the list  
 data.append**(**line**)** # Reading the next line  
 line **=** subject\_file.readline**()** # Closing the file  
 subject\_file.close**()  
 return** data  
  
# Function to create an optimal tick list  
**def create\_tick\_list(**data, x\_tick**):** tick\_list **= []** counter **=** 0  
 sort\_data **=** sorted**(**data**)  
 for** x **in** x\_tick**:** tick\_list.append**(**sort\_data**[**x**-**1**])  
 return** tick\_list  
  
**def create\_lineGraph(**data**):** # Creating the x-values corresponding to the input data  
 x\_vals **=** range**(**1, len**(**data**)+**1**)** x\_tick **= [**1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50**]** x\_tick\_count **= [**1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50**]** # The list for ticks  
 # most decimals  
 y\_tick\_list **=** create\_tick\_list**(**data, x\_tick**)** y\_tick\_list\_count **=** x\_tick\_count  
 # Plotting  
 plt.figure**(**1**)** plt.plot**(**x\_vals, data, **'ro-')** # Setting the title  
 plt.title**('The Average Weekly Price of the Gas in 1994')** # Putting x labels  
 plt.xlabel**('Week')** plt.ylabel**('Price ($)')** # Putting the ticks on the graph  
 plt.xticks**(**x\_tick\_count, x\_tick**)** plt.yticks**(**y\_tick\_list\_count, y\_tick\_list**)** # Putting the grids on the graph  
 plt.grid**(True)** # Saving the graph as  
 plt.savefig**('gas\_price\_1994\_line.png')** # Showing the graph  
 plt.show**()  
 return**# Function to create a bar chart  
**def create\_barChart(**data**):** # Creating the x-values corresponding to the input data  
 x\_vals **=** range**(**1, len**(**data**) +** 1**)** x\_tick **= [**1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50**]** x\_tick\_count **= [**1, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50**]** # The list for ticks  
 # most decimals  
 y\_tick\_list **=** create\_tick\_list**(**data, x\_tick**)** y\_tick\_list\_count **=** x\_tick\_count  
 # Plotting  
 bar\_width **=** 0.5  
 plt.figure**(**1**)** plt.bar**(**x\_vals, data, bar\_width**)** # Setting the title  
 plt.title**('The Average Weekly Price of the Gas in 1994')** # Putting x labels  
 plt.xlabel**('Week')** plt.ylabel**('Price ($)')** # Putting the ticks on the graph  
 plt.xticks**(**x\_tick, x\_tick\_count**)** plt.yticks**(**y\_tick\_list\_count, y\_tick\_list**)** # Saving the chart as a file  
 plt.savefig**('gas\_price\_1994\_bar.png')** # Showing the chart  
 plt.show**()  
 return**# Calling out the main function  
**if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

**FIG1**

**A screenshot of a cell phone

Description automatically generated**

**FIG2**

A screenshot of a cell phone

Description automatically generated

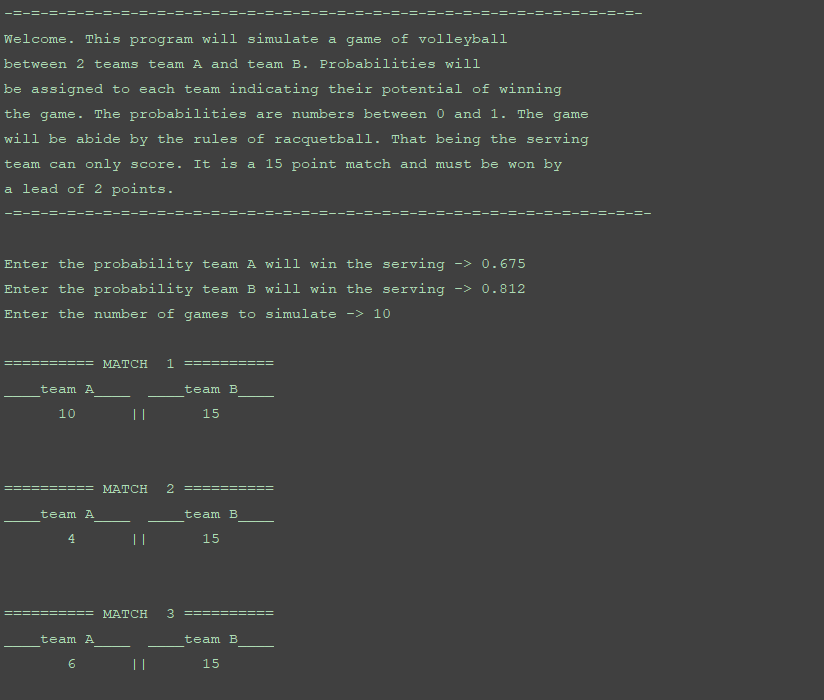
1. (**15 points, Games**) Design and implement a simulation of the game of volleyball. Normal volleyball is played like racquetball, in that a team can only score points when it is serving. Games are played to 15, but **must be won by at least two points**.

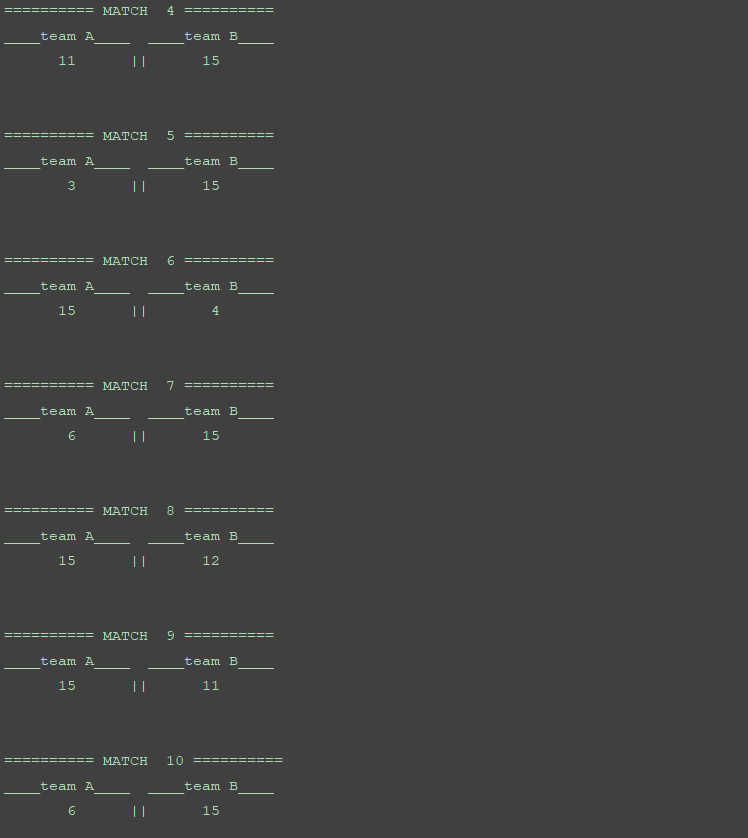
**>>CODE**

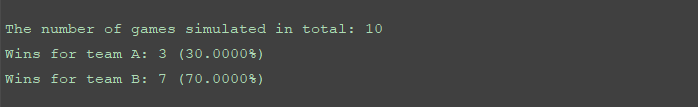
###  
# AUTHOR: Tomoki Koike  
# DATE: Oct. 20, 2019  
# DESCRIPTION: This program is designed to simulate a game of volleyball using probability.  
###  
  
# Main function  
**def main():** # Printing the introduction  
 introPrint**()** # Obtaining the inputs from the user  
 **[**probA, probB, num**] =** getInputs**()** # Simulating the volleyball games  
 **[**winA, winB**] =** allGamesSim**(**num, probA, probB**)** print**()** # Printing out the results  
 printOutput**(**winA, winB, num**)**# Import modules  
**import** random **as** rand  
**import** re  
  
# Functions  
# Input validation function  
**def getValid(**prompt**):** basis **=** re.compile**('[^0-9.]')** # Basis to check REGEX inside input  
 **while True:  
 try:** this **=** input**(**prompt**)** # Obtain input from user  
 **except** ValueError**:** # Invalid input  
 print**('Sorry, could not understand input. Please try again.')  
 continue  
 if** basis.search**(**this**):** # Invalid input  
 print**('Please enter an appropriate input. Please try again.')  
 continue  
 else:** # Valid input  
 **break  
 return** this  
  
# Function to print out introduction  
**def introPrint():** print**('-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-')** print**('Welcome. This program will simulate a game of volleyball')** print**('between 2 teams ''team A'' and ''team B''. Probabilities will')** print**('be assigned to each team indicating their potential of winning')** print**('the game. The probabilities are numbers between 0 and 1. The game')** print**('will be abide by the rules of racquetball. That being the serving')** print**('team can only score. It is a 15 point match and must be won by')** print**('a lead of 2 points.')** print**('-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=--=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-')** print**()  
 return**# Function that accepts the input from the user  
**def getInputs():** # The probabilities for each team  
 probA **=** getValid**('Enter the probability team A will win the serving -> ')** probB **=** getValid**('Enter the probability team B will win the serving -> ')** # Number of games to simulate  
 num **=** getValid**('Enter the number of games to simulate -> ')  
 return** float**(**probA**)**, float**(**probB**)**, int**(**num**)**# Function deciding first serving team  
**def chooseServing():** # Generating number 1 or 0  
 chip **=** rand.randint**(**0, 1**)  
 if** chip **==** 0**:** server **= 'A'  
 else:** server **= 'B'  
 return** server  
  
# Function simulating one game  
**def oneGameSim(**probA, probB, firstServe**):** serving **=** firstServe # Which team starts serving  
 scoreA **=** scoreB **=** 0 # Initialize the scores  
 # Loop to simulate the game  
 **while not** endGame**(**scoreA, scoreB**):  
 if** serving **== 'A':  
 if** rand.random**() <** probA**:** scoreA **+=** 1  
 **else:** serving **= 'B'  
 else:  
 if** rand.random**() <** probB**:** scoreB **+=** 1  
 **else:** serving **= 'A'  
 return** scoreA, scoreB  
  
# Function simulating all games  
**def allGamesSim(**num, probA, probB**):** winA **=** winB **=** 0 # Initializing winning numbers  
 # Loop to simulate all games  
 **for** x **in** range**(**num**):  
 [**scoreA, scoreB**] =** oneGameSim**(**probA, probB, chooseServing**())** # print()  
 # print('========== MATCH {0} =========='.format(x+1))  
 # print('\_\_\_\_team A\_\_\_\_ \_\_\_\_team B\_\_\_\_')  
 # print('{0:8} ||{1:8}'.format(scoreA, scoreB))  
 # print()  
 # Determine winner  
 **if** scoreA **>** scoreB**:** winA **+=** 1  
 **else:** winB **+=** 1  
 **return** winA, winB  
  
# Function that defines the end of the game  
**def endGame(**scoreA, scoreB**):** # One team must be over 15 points  
 cond1 **=** scoreA **>=** 15 **or** scoreB **>=** 15  
 # The team to win must be leading the game by 2 points  
 cond2 **= (**scoreA **-** scoreB **>=** 2**) or (**scoreB **-** scoreA **>=** 2**)  
 return** cond1 **and** cond2  
  
# Function to print out output  
**def printOutput(**winA, winB, num**):** print**('The number of games simulated in total: {0}'**.format**(**num**))** print**('Wins for team A: {0} ({1:.4f}%)'**.format**(**winA, winA**/**num\*100**))** print**('Wins for team B: {0} ({1:.4f}%)'**.format**(**winB, winB**/**num\*100**))**# Executing main function  
**if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

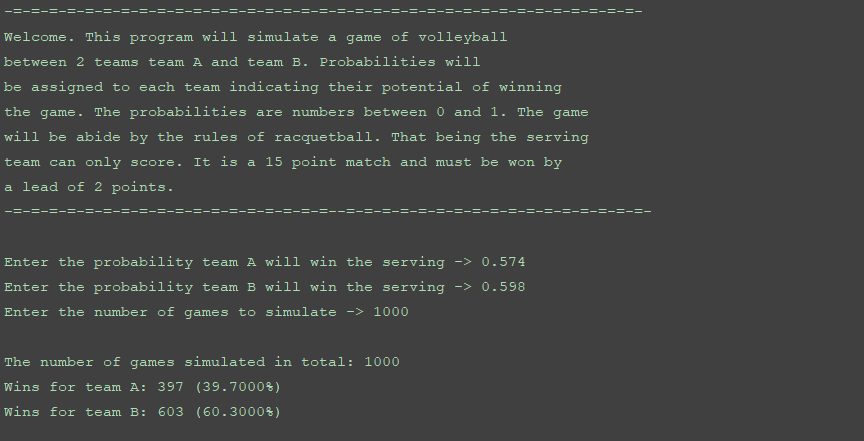
Small simulation N=10 (\*\*\*showing each game scores)







Large Simulation N=1000 (\*\*\*without displaying each game score)



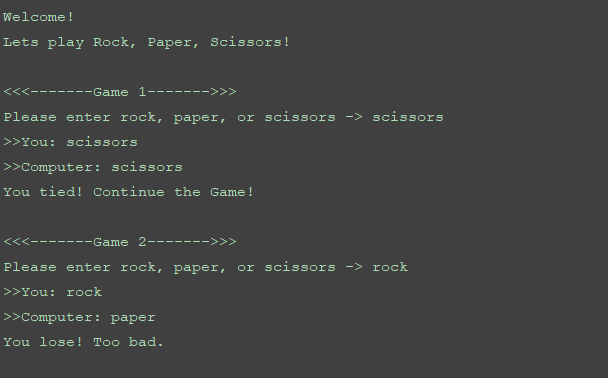
1. (**15 points**, **Games**) Write a program that lets user play the game of Rock, Paper, Scissors against the computer. The program should work as follows:
   1. When the program begins, a random number in the range of 1 through 3 is generated. If the number is 1, then the computer has chosen rock. If the number is 2, then the computer has chosen paper. If the number is 3, then the computer has chosen scissors. (Don’t display the computer’s choice yet.)
   2. The user enters his or her choice of “rock”, “paper”, or “scissors” at the keyboard.
   3. The computer’s choice is displayed.
   4. A winner is selected acceding to the following rules:
      * If one player chooses rock and other player chooses scissors, then rock wins. (Rock smashes scissors.)
      * If one player chooses scissors and the other player choose paper, then scissors wins. (scissors cut paper.)
      * If one player chooses paper and the other play chooses rock, then paper wins. (Paper wraps rock.)
      * If both players make the same choice, the game must be played again to determine the winner.

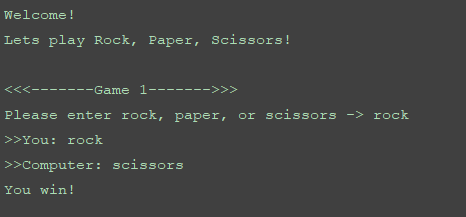
**>>CODE**

###  
# AUTHOR : Tomoki Koike  
# DATE : Oct. 20, 2019  
# DESCRIPTION: This program lets the user play the game of rock, paper, scissor with a computer  
###  
  
# Main  
**def main():** # Starting game  
 startGame**()** # Initiate game loop  
 gameLoop**()**# Modules  
**import** random **as** rand  
**import** re  
  
# Matrix with winning, losing, and tying combinations (player perspective)  
# combos = {  
# 'win':[13, 32, 21],  
# 'lose':[12, 23, 31],  
# 'tie':[11, 22, 33]  
# } The matrix indices  
combos **= [  
 ['tie'**, **'lose'**, **'win']**,  
 **['win'**, **'tie'**, **'lose']**,  
 **['lose'**, **'win'**, **'tie']  
]**# Matrix on the moves  
moves **= ['rock'**, **'paper'**, **'scissors']**# Functions  
# Function for input validation  
**def getValid(**prompt**):** base **=** re.compile**('[^a-z]')  
 while True:  
 try:** # Getting input from player  
 this **=** input**(**prompt**)  
 except** ValueError**:** # Inappropriate input  
 print**('Sorry, could not understand. Please try again.')  
 if** base.search**(**this**):** # Inappropriate input  
 print**('Your input was not accepted. Please enter only lower case alphabets with no space.')  
 elif** this **!= 'rock' and** this **!= 'paper' and** this **!= 'scissors':** # Invalid input  
 print**('You probably had a typo. Please try again.')  
 continue  
 else:** # Valid input  
 **break  
 return** this  
  
# Function to start the game  
**def startGame():** print**('Welcome!')** print**('Let''s play Rock, Paper, Scissors!')  
 return**# Function that chooses the computers move  
**def computerHand():** hand **=** rand.randint**(**1,3**)  
 return** hand  
  
**def playerHand():** hand **=** getValid**('Please enter rock, paper, or scissors -> ')** # Assigning numbers depending on move  
 **if** hand **== 'rock':** handCode **=** 1  
 **elif** hand **== 'paper':** handCode **=** 2  
 **else:** handCode **=** 3  
 **return** handCode  
  
# Function that shows the result of the game  
**def playGame(**computer, player**):** # Getting result of the game  
 result **=** combos**[**player**-**1**][**computer**-**1**]  
 return** result  
  
**def gameLoop():** # Counter for number of games played  
 counter **=** 1  
 **while True:** print**()** print**('<<<-------Game {0}------->>>'**.format**(**counter**))** # Generating the compture's move  
 computerMove **=** computerHand**()** # Generating the player's move  
 playerMove **=** playerHand**()** print**('>>You: ' +** moves**[**playerMove**-**1**])** print**('>>Computer: ' +** moves**[**computerMove**-**1**])** # Obtaining the result  
 result **=** playGame**(**computerMove, playerMove**)** # Determing if the game ends or continues  
 **if** result **== 'win':** # If player wins  
 print**('You win!')  
 break  
 elif** result **== 'lose':** # If player loses  
 print**('You lose! Too bad.')  
 break  
 else:** # If player ties  
 print**('You tied! Continue the Game!')** counter **+=** 1  
 **continue  
 return**# Executing main function  
**if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

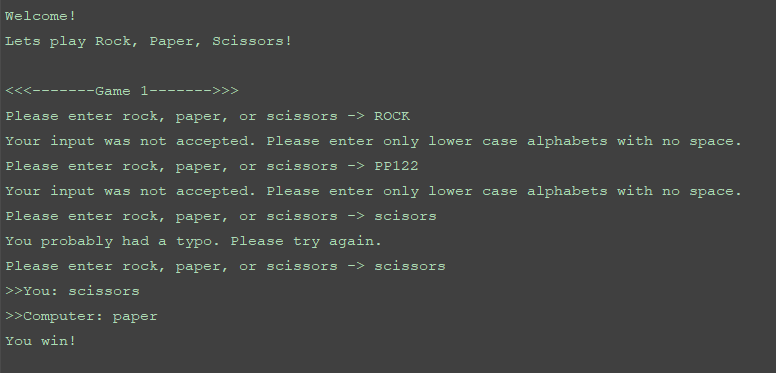
**>>OUTPUT**

**W/O INPUT VALIDATION**





**W/ INPUT VALIDATIONS**



1. (**20 points, Data Analysis**) In the attachments, there is a file named ‘Particle\_Morphology\_Data.txt’. This file contains three kinds of measurements of a lot of solid particles. There are 15133 lines which corresponds to the 15133 particles. In each line, there are three numbers, which are the equivalent diameter (unit: μm), circularity (dimensionless) and aspect ratio (dimensionless), respectively.

Write a Python program that reads this file and creates three lists for the three properties. For each list, the program should display the maximum and minimum values and divide the value range into 12 equally spaced segments. Then the program should count the number of data in each segment and plot a histogram with the bar function. In total, there should be three histograms.

(Please read https://en.wikipedia.org/wiki/Histogram if you don’t know what histogram is.)

Note: Try to make your plots tidy and professional. No too wide or too narrow bars. Appropriate upper and lower limits for the axis. Make sure you have proper labels with units and titles for all plots.

**>>CODE**

###  
# AUTHOR : Tomoki Koike  
# DATE : Oct. 20 2019  
# DESCRIPTION: This program is designed to manipulate a data text file and break the  
# specific data by relevance. And then break each of them into equal numbers  
# of data and create a histogram for each data.  
###  
  
# Main Function  
**def main():  
 [**diam, circ, aspr**] =** createList**('Particle\_Morphology\_Data.txt')** # For diameters  
 diam\_count **=** segmentcount**(**diam**)** diam\_barwidth **=** getbarwidth**(**diam**)** diam\_left\_edge **=** getleftedge**(**diam\_barwidth**)** fig1 **=** plt.figure**()** histplot**(**diam\_count, diam\_left\_edge, diam\_barwidth, **'blue')** plt.xlabel**('Diameter')** plt.ylabel**('Count')** plt.title**('Count of Diameters in 12 Equally Spaced Segments')** plt.show**(**block**=False)** fig1.savefig**('Diameter.png')** time.sleep**(**1**)** plt.close**('all')** # For circularity  
 circ\_count **=** segmentcount**(**circ**)** circ\_barwidth **=** getbarwidth**(**circ**)** circ\_left\_edge **=** getleftedge**(**circ\_barwidth**)** fig2 **=** plt.figure**()** histplot**(**circ\_count, circ\_left\_edge, circ\_barwidth, **'red')** plt.xlabel**('Circularity')** plt.ylabel**('Count')** plt.title**('Count of Circularities in 12 Equally Spaced Segments')** plt.show**(**block**=False)** fig2.savefig**('Circularity.png')** time.sleep**(**1**)** plt.close**('all')** # For aspect ratio  
 aspr\_count **=** segmentcount**(**aspr**)** aspr\_barwidth **=** getbarwidth**(**aspr**)** aspr\_left\_edge **=** getleftedge**(**aspr\_barwidth**)** fig3 **=** plt.figure**()** histplot**(**aspr\_count, aspr\_left\_edge, aspr\_barwidth, **'green')** plt.xlabel**('Aspect Ratio')** plt.ylabel**('Count')** plt.title**('Count of Aspect Ratios in 12 Equally Spaced Segments')** plt.show**(**block**=False)** fig3.savefig**('AspectRatio.png')** time.sleep**(**1**)** plt.close**('all')**# Imported Modules  
**import** matplotlib.pyplot **as** plt  
**import** time  
  
# Functions  
# Reading the file and creating lists  
**def createList(**afile**):** data\_file **=** open**(**afile, **'r')** # Open the file  
 aline **=** data\_file.readline**()** # Reading the first line  
 # Preallocating the lists  
 diam **= []** # List of diameters  
 circ **= []** # List of circularities  
 aspr **= []** # List of aspect ratios  
 # Loop through the file and read all the lines  
 **while** aline **!= '':** aline **=** aline.rstrip**('\n')** # Striping off the end of the readline  
 temp **=** list**(**aline.split**('\t'))** # Splitting the read line by spaces  
 diam.append**(**float**(**temp**[**0**]))** # Appending the diameter value in the read line  
 circ.append**(**float**(**temp**[**1**]))** # Appending the circularity value in the read line  
 aspr.append**(**float**(**temp**[**2**]))** # Appending the aspect ratio value in the read line  
 aline **=** data\_file.readline**()** # Reading the next line  
 data\_file.close**()** # Closing the opened file  
 **return** diam, circ, aspr  
  
# Function that gives the maximum and the minimum  
**def max\_min(**alist**):  
 return** max**(**alist**)**, min**(**alist**)**# Function that retrieves the range of the list  
**def getrange(**alist**):  
 [**amax, amin**] =** max\_min**(**alist**)** # Obtaining the maximum and minimum values of the list  
 **return** amax **-** amin  
  
# Function that gives the left edge of the bar graph or histogram  
**def getleftedge(**bar\_width**):** left\_edge **= []** # Initializing list  
 **for** x **in** range**(**1, 13**):** left\_edge.append**(**bar\_width**\***x**)  
 return** left\_edge  
  
# Function to find bar\_width  
**def getbarwidth(**alist**):** list\_range **=** getrange**(**alist**)** chop **=** list\_range **/** 12  
 **return** chop  
  
# Function that segments the lists  
**def segmentcount(**alist**):** chop **=** getbarwidth**(**alist**)** # Dividing the range into 12 equal segments  
 sorted\_list **=** sorted**(**alist**)** # Sorting the list  
 count\_list **= []** # Initializing the count list  
 **for** x **in** range**(**1, 13**):** i **=** 0 # Counter  
 sliced\_list **= []** # Initializing the sliced list  
 **if** x **!=** 12**:  
 while** sorted\_list**[**i**] <= (**min**(**alist**)+**chop**\***x**):** sliced\_list.append**(**sorted\_list**[**i**])** # Appending value  
 i **+=** 1 # Incrementing counter  
 **else:** sliced\_list **=** sorted\_list  
 **del** sorted\_list**[**0**:**i**]** # Delete the values that are unnecessary  
 count\_list.append**(**len**(**sliced\_list**))** # Appending to the count list  
 **return** count\_list  
  
# Function to plot histogram  
**def histplot(**count\_list, left\_edge, bar\_width, iro**):** plt.bar**(**left\_edge, count\_list, bar\_width, edgecolor**='black'**, color**=**iro**)  
 return  
  
if** \_\_name\_\_ **== '\_\_main\_\_':** main**()**

**>>OUTPUT**

**A screenshot of a cell phone

Description automatically generated**

**A close up of a logo

Description automatically generated**

**A picture containing screenshot

Description automatically generated**