**ENGR 102 Sect 508 Lab 7a**

**100 points**

**Reading assignment:**

|  |  |
| --- | --- |
| **Lecture Slides** | **L07** |
| **zyBook chapter 7** | **Complete all participation and challenge activities** |
| **Hoffman book handout** | **Chapter 9 Lists** |

*Attention!!*

*Team submission. one submission per team.*

*Submit* *your Py-files together with your word/pdf file with screenshots of your tests outputs. Include any derivations, comments and supplemental notes in your word/pdf files.*

*No pictures by the phone – it is impossible to read. You will be allowed to resubmit and reupload HW as many times as you want to within the due date/time, only last submission will be graded. No late submissions. For submission you may use this file as template: rename file including your name. Do not forget to put your name inside of this file as well. If it is a team work, include the team number and all team members. For this submission use Team Header, include all team members into the list of participants. Submit 1 assignment per team.*

**[35 points] Activity #1: Comparing array values – to be done as a team**

This activity is meant to help you learn to read data, store it in a list, and then process it by looping through values.

Imagine that you are managing a plant, and measure production each day. Production is in number of widgets, and you want a program that will let you enter in the number of widgets, and then tell whether for periods of time ranging from 1 day intervals to the maximum interval, whether the production is rising or falling. You want this to work for an arbitrary number of days. That is, you should be able to enter as many days as you want, and get a report as to days increasing or decreasing.

**For the input**, make sure your program can collect the number of widgets for an arbitrary number of days. You can stop taking input when a user enters a negative number. Be sure to give a descriptive message telling what to input.

**For the output**, you want to report, for each possible interval from 1 day to the maximum interval, what percentage of differences in widget production were increasing, and what percentage of intervals were decreasing (some could be the same). ***Print the output with 1 digit after the decimal.***

**For example**, if the widget production was entered for 5 consecutive days, and the production levels were 13, 15, 17, 15, 18, you might output something like:

For 1-day intervals 75.0% were increasing and 25.0% were decreasing

For 2-day intervals 66.7% were increasing and 0.0% were decreasing

For 3-day intervals 100.0% were increasing and 0.0% were decreasing

For 4-day intervals 100.0% were increasing and 0.0% were decreasing

The first line is because out of the four one-day intervals, 13-15, 15-17, and 15-18 were increasing, while 17-15 was decreasing. The second line is because out of the three two-day intervals, 13-17 and 17-18 were increasing, while 15-15 was neither increasing nor decreasing. The third line is because both four-day intervals, 13-15 and 15-18, were increasing, and the final line is because the one four-day interval 13-18 was increasing.

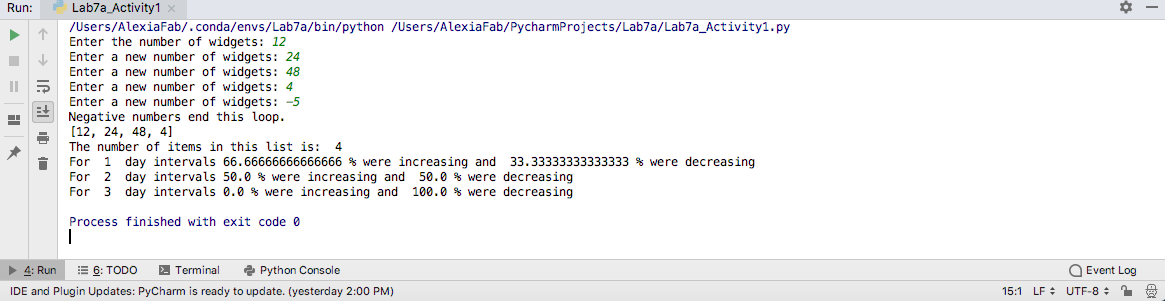
There is more than one way to get one decimal place after the number. One option is for your team to look up and learn the command for formatting floating-point output. Another option is for your team to come up with a set of steps to create this string, yourselves.

Before your team starts coding, consider exactly how you will make these computations. The looping in this problem is trickier than what you have encountered previously, and formatting the output may be a challenge for you (work together as a team to figure out how you might print the percentages with exactly 1 place after the decimal). Remember to use the methods we have discussed for testing, and using incremental development.

Input:

*# By submitting this assignment, all team members agree to the following:  
# “Aggies do not lie, cheat, or steal, or tolerate those who do”  
# “I have not given or received any unauthorized aid on this assignment”  
#  
# Names: Alexia Perez  
# Bethany Gawalis  
# Nicolas Martinez  
# Sam Lyzzaik  
# Tyler Scataglia  
# Section: 508  
# Assignment: Lab 7a  
# Date: 09-10-2018***import** numpy  
**from** math **import** \*  
  
*# In this program we will measure production of a plant (each day).  
# Production will be expressed as number of widgets,  
# the program will let a user enter the number of widgets,  
# and then output whether the production is rising or falling.  
# for periods of time ranging from 1 day intervals to the maximum interval.*widgets = int(input(**"Enter the number of widgets: "**))  
production\_list= []  
  
**while** widgets >= 0: *# This loop will stop if the value enetered for a widget is negative.* production\_list.append(widgets)  
 widgets = int(input(**"Enter a new number of widgets: "**))  
**else**:  
 print(**"Negative numbers end this loop."**)  
  
*# This prints the whole list (except the negative value entered last, as it is assumed  
# that a negative value will only be inputted to stop the loop.*print(production\_list)  
items = len(production\_list) *# gives number of items in list*print(**"The number of items in this list is: "**,items)  
  
number\_intervals = 0  
increasing\_intervals = 0  
decreasing\_intervals = 0  
max\_intervals = items  
i = 1  
j = 0  
  
*# This loop runs as many times as the maximum size interval***for** i **in** range (1,max\_intervals):  
 *# This loop iterates through the array* **for** j **in** range (0,max\_intervals-i):  
  
 **if** production\_list[j]<production\_list[j+i]:  
 increasing\_intervals+=1  
 **elif** production\_list[j]>production\_list[j+i]:  
 decreasing\_intervals+=1  
 number\_intervals = number\_intervals+1  
 increasing\_percentage= increasing\_intervals/(number\_intervals) \* 100  
 decreasing\_percentage = decreasing\_intervals/(number\_intervals) \* 100  
 print(**"For "**,i,**" day intervals"**,increasing\_percentage,**"% were increasing and "**,  
 decreasing\_percentage,**"% were decreasing"**)  
 number\_intervals = 0  
 increasing\_intervals = 0  
 decreasing\_intervals = 0

Output:

****

**[35 points] Activity #2: Making the cut in golf**

The purpose of this activity is to get your team, as a team, to think through a computational problem and come up with an interesting computational solution. There are multiple possible ways to solve this problem.

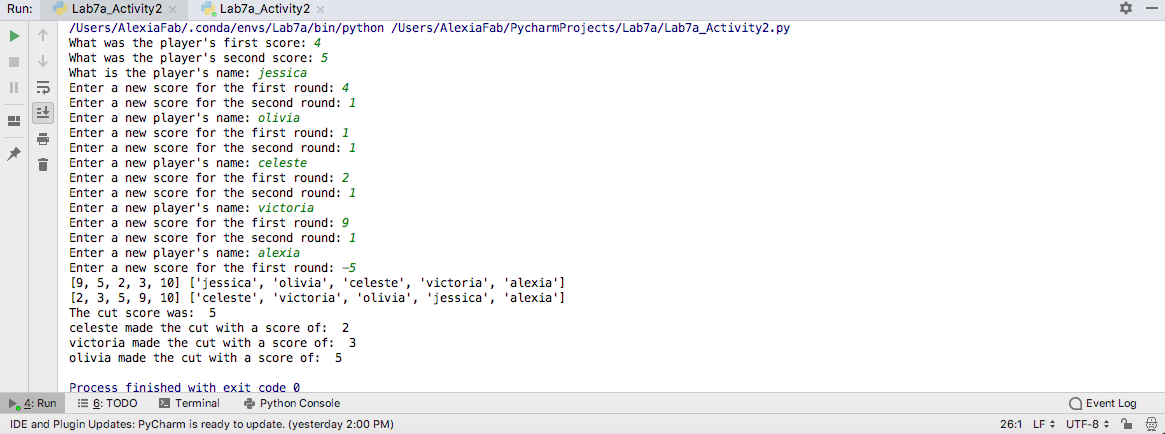
In professional golf tournaments, a tournament typically lasts four rounds. A player’s score is the sum of the scores in the rounds (the lower the better). It is common for all players to play the first two rounds of golf. However, only a fraction of the players are allowed to continue on to the third and fourth rounds. The way this is done is that a “cut” score is determined, and those whose scores are better (i.e. lower) than the cut are allowed to play the remainder of the tournament (they “made the cut”) while the rest do not.

In this case, the cut will be the median score among the golfers. You are to print out the names of golfers who made the cut (whose score was below the median), and those who did not make the cut (score above the median). Thus, about half the golfers should make the cut. Here are some details:

* You are to write a program that reads in golfers’ names, and their first and second round scores. Specifically, they should enter the first round score on one line, then the second round score on another line, then the player’s name on a third line. This should continue for an arbitrary number of players. The user should indicate that they are done entering players by giving a negative score for the first round – at this point, the reading-in of data should stop (without reading a second round or player name).
* In practice, a common way to find the median is typically to sort the numbers from smallest to largest, and then to find the median from there directly (either the middle element if an odd number of elements., or the average of the two middle elements if an even number of elements). There is a simple sort routine built in to Python, ***but you are not to use it (the built-in sort command) for this problem***.
* There are many ways to find a median. A major part of this problem is to figure out a method for finding the median, yourselves. There are solutions involving just multiple loops, there are solutions that will use more than one list, and so on. Also, you do not technically have to find a median – you could just find a way to report both those above and those below the median.
* You’ll be outputting two sets of names. Be sure it is clear which is which.
* Do this project as a team. The idea is that you should talk through the problem and develop a solution together. Be sure to use good code development, as discussed before.

Input:

*# By submitting this assignment, all team members agree to the following:  
# “Aggies do not lie, cheat, or steal, or tolerate those who do”  
# “I have not given or received any unauthorized aid on this assignment”  
#  
# Names: Alexia Perez  
# Bethany Gawalis  
# Nicolas Martinez  
# Sam Lyzzaik  
# Tyler Scataglia  
# Section: 508  
# Assignment: Lab 7a  
# Date: 09-10-2018***import** numpy  
**from** math **import** \*  
  
score\_one= int(input(**"What was the player's first score: "**))  
score\_two= int(input(**"What was the player's second score: "**))  
score= score\_one+score\_two  
name = str(input(**"What is the player's name: "**))  
array\_scores=[]  
array\_names=[]  
  
**while** score\_one >= 0:  
 array\_names.append(name)  
 array\_scores.append(score)  
 score\_one = int(input(**"Enter a new score for the first round: "**))  
 **if** score\_one < 0:  
 **break** score\_two = int(input(**"Enter a new score for the second round: "**))  
 score = score\_one + score\_two  
 name = str(input(**"Enter a new player's name: "**))  
print(array\_scores, array\_names)  
array\_size1 = len(array\_scores)  
median\_score=0  
i = 0  
j = 0  
  
**for** i **in** range(0,array\_size1-1):  
 **for** j **in** range(0,array\_size1-1):  
 **if** array\_scores[j]>array\_scores[(j+1)]:  
 temp = array\_scores[j]  
 array\_scores[j]=array\_scores[(j+1)]  
 array\_scores[(j+1)]=temp  
 median\_score = array\_scores[int(array\_size1/2)]  
 temp\_two = array\_names[j]  
 array\_names[j] = array\_names[(j + 1)]  
 array\_names[(j + 1)] = temp\_two  
print(array\_scores,array\_names)  
print(**"The cut score was: "**, median\_score)  
**for** i **in** range(0,array\_size1-1):  
 **if** array\_scores[i] <= median\_score:  
 print(array\_names[i], **"made the cut with a score of: "**, array\_scores[i])  
  
Output:



**[30 points] Activity #3: Chessboard moves**

The purpose of this activity is to get you used to using lists of lists, in a 2-D matrix-like format.

Write a program that sets up a chess board with chess pieces, and lets people make moves (of one piece at a time) on the chess board. Here are the details:

* The chessboard is an 8x8 board.
* Display the chess board before every move. Each empty square should just be a period. Each square with a piece should have the piece’s identifier.
* For identifiers, use lower-case for the white pieces, upper-case for the black pieces.
  + Use P/p for pawn, R/r for rook, N/n for knight, B/b for bishop, Q/q for queen, and K/k for king.
* When a piece is moved, the location it moves from is then an empty square, and it then occupies the square it is moving to. If there was already a piece in the square it is moving to, then that other piece is eliminated from the game.
* **Important:** You do **NOT** need to enforce any rules of chess or verify moves, with one exception:
  + If someone puts in a move from a position where there is not a piece, report an error and exit the program.
  + Other than that, you don’t need to worry about the sides alternating turns, about pieces moving in ways they aren’t allowed, about landing on your own pieces, etc.
  + Only one piece moves at a time, and it can move to any position on the board.
* You can specify the system you want the user to use to specify the starting and ending location.
  + In reality, chess uses a standard system, where the columns are labeled a-h from left to right and the rows are labeled 1-8 from bottom to top. You do not have to use this system, but can if you wish.

As an example, here is what the board would look like at the very beginning of the game:

RNBQKBNR

PPPPPPPP

........

........

........

........

pppppppp

rnbqkbnr

Remember to discuss this as a team and think through exactly what you will do before developing it!

Code:

*# By submitting this assignment, all team members agree to the following:  
# “Aggies do not lie, cheat, or steal, or tolerate those who do”  
# “I have not given or received any unauthorized aid on this assignment”  
#  
# Names: Alexia Perez  
# Bethany Gawalis  
# Nicolas Martinez  
# Sam Lyzzaik  
# Tyler Scataglia  
# Section: 508  
# Assignment: Lab 7a  
# Date: 09-10-2018***from** numpy **import** \*  
**from** math **import** \*  
  
chess\_setup= array([[**'R'**,**'N'**,**'B'**,**'Q'**,**'K'**,**'B'**,**'N'**,**'R'**],  
 [**'P'**,**'P'**,**'P'**,**'P'**,**'P'**,**'P'**,**'P'**,**'P'**],  
 [**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**],  
 [**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**],  
 [**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**],  
 [**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**,**'.'**],  
 [**'r'**,**'n'**,**'b'**,**'q'**,**'k'**,**'b'**,**'n'**,**'r'**],  
 [**'p'**,**'p'**,**'p'**,**'p'**,**'p'**,**'p'**,**'p'**,**'p'**]])  
chessboard = reshape(chess\_setup,(8,8))  
print(chessboard)  
*# the coordinates are (row,column)  
# print(chessboard[0][4])  
# print(chessboard[6][5])  
  
  
  
  
# Ask user for input of the coordinates of the piece they want to move, +  
# and the coordinates of the grid where they want to move it.*x\_initial = int(input(**"In which row is the piece you want to move,"  
 "the first row is 0 and the last one is 7: "**))  
y\_initial = int(input(**"In which column is the piece you want to move,"  
 "the first column is 0 and the last one is 7: "**))  
  
  
*# This next block is extra, we made it just to confirm that the user used the right coordinates for the piece  
# that they want to move.*print(**"You're going to move "**,chessboard[x\_initial][y\_initial])  
confirm = int(input(**"Is this correct? Input 1 for yes and 0 for no: "**))  
  
**while** confirm == 0:  
 x\_initial = int(input(**"Re-enter in which row is the piece you want to move,"  
 "the first row is 0 and the last one is 7: "**))  
 y\_initial = int(input(**"Re-enter in which column is the piece you want to move,"  
 "the first column is 0 and the last one is 7: "**))  
 print(**"You're going to move "**, chessboard[x\_initial][y\_initial])  
 confirm = int(input(**"Is this correct? Input 1 for yes and 0 for no: "**))  
  
*# Now we ask for the coordinates of the box where they want to move the piece.*print(**"Recall that the first row/column is 0."**)  
x\_final = int(input(**"In which row do you want to place your piece: "**))  
y\_final = int(input(**"In which column do you want to place your piece: "**))  
  
*# Now we check that there is a piece in the box they selected  
# If there is, we move it to the new position and replace the initial with a dot  
# if there's not a piece we print an error  
# This code will run over and over asking the user for new input  
# until the user inputs a negative number (meaning they want to stop playing)***while** x\_initial >= 0:  
 pieces=[**"R"**,**"N"**,**"B"**,**"Q"**,**"K"**,**"P"**,**"r"**,**"n"**,**"b"**,**"q"**,**"k"**,**"p"**]  
 **if** str(chessboard[x\_initial][y\_initial]) **in** pieces:  
 swap = (chessboard[x\_initial][y\_initial])  
 chessboard[x\_initial][y\_initial] = chessboard[x\_final][y\_final]  
 chessboard[x\_final][y\_final]= swap  
 print(chessboard)  
 x\_initial = int(input(**"Re-enter in which row is the piece you want to move,"  
 "the first row is 0 and the last one is 7: "**))  
 **if** x\_initial < 0:  
 print(**"Thank you for playing our chess game."**)  
 **elif** x\_initial >=0:  
 y\_initial = int(input(**"Re-enter in which column is the piece you want to move,"  
 "the first column is 0 and the last one is 7: "**))  
 x\_final = int(input(**"In which row do you want to place your piece: "**))  
 y\_final = int(input(**"In which column do you want to place your piece: "**))  
 **else**:  
 print(**"Sorry, there's not a piece in the box you have selected"**)  
 x\_initial = int(input(**"Re-enter in which row is the piece you want to move,"  
 "the first row is 0 and the last one is 7: "**))  
 y\_initial = int(input(**"Re-enter in which column is the piece you want to move,"  
 "the first column is 0 and the last one is 7: "**))  
 x\_final = int(input(**"In which row do you want to place your piece: "**))  
 y\_final = int(input(**"In which column do you want to place your piece: "**))  
  
  
Output:

