COMP 10001 - Programming Fundamentals
Assignment 1

Mohawk College
Department of Engineering Technology

Due: Friday February 17, 2017 at 11:59 PM

Abstract

In lectures we have discussed that the best way to learn computer programming and problem solving is to get hands on and solve problems. In the labs you have investigated sequential execution, selection structures and repetition. The following problems require you to apply the define, design, implement and test approach to problem solving.

It is very important that you not leave this assignment for the last minute. The problems are supposed to challenge you and help you learn more about problem solving. They cannot be easily completed in a single sitting.

Regardless of your level of success on the problems be sure to submit whatever work you do, part marks will be awarded where possible.

1 Instructions

1. This assignment is individual work. Groups and sharing are not permitted.
2. This assignment is out of 100 but there are 105 marks available. No bonus will be awarded above 100%.
3. All source code submitted as part of this assignment **must** contain a statement of authorship or a grade of 0 will be awarded for the entire problem.
4. You will be submitting several source code files as well as an assignment report. Your report must be named ***COMP 10001 - Assignment 1 - <your student number>.docx***, where ***<your student number>*** is replaced by your Mohawk College student ID number.
5. Your assignment report should contain a title page which includes the title of the assignment, your name, student number, your professor's name, the date, and a statement of authorship.

6. Each problem requires evidence of the DDIT approach: an IPO chart, pseudocode or flow chart and testing traces. You may include pseudocode in your Python solutions as comments but you should make a copy of this in your report as well, do not use a screen shot.
7. Each implementation must use the style rules defined in labs 1-3.
8. Late assignments will be penalized 10% for the first 24 hours, 50% up to 48 hours and will not be accepted more than 48 hours late without a doctor's note or similar circumstance.

2 Problem: Guessing Game

2.1 Problem Statement

Create a program that picks a random number from 1 to 100 (inclusive) and then asks the user to guess it. If the user doesn't guess it correctly tell them if the number is higher or lower and give them another chance. Let the user keep guessing until they get the correct answer. Keep track of how many guesses it takes to get the right number. You may assume the user will enter reasonable numbers only.

In order to do this you will need to use Python's random library. You may use the following as starter code:

```
import random

randomNumber = random.randint( 1, 100 )

# Your code goes here
```

You are required to submit your IPO chart, pseudocode or flow chart, design trace, source code and screen shots of your program execution with the same values as your design trace.

Sample program run:

```
I picked a number between 1 and 100 inclusive, can you guess it?  50
Sorry, my number is higher than that.  Try again!
I picked a number between 1 and 100 inclusive, can you guess it?  75
Sorry, my number is lower than that.  Try again!
I picked a number between 1 and 100 inclusive, can you guess it?  67
Yes!  That's correct!  It took you 3 guesses to find my number.
```

2.2 Grading Scheme and Other Requirements

To be eligible for grading your solution must be named **guessingGame.py** and must contain a statement of authorship. Penalties will be assigned if either of these conditions are not met.

DDIT Evidence	IPO Chart	3
	Pseudocode/Flow Chart	5
	Design Trace	3
	Final Tests	2
Style Rules	spacing	3
	variable naming	3
Implementation	random number	2
	user input	3
	integer conversion	3
	output similar to sample	5
	guess accumulator	3
	loop until correct	5
Total		40

3 Problem: String Centering

3.1 Problem Statement

Ask the user to enter a short message. If the message is longer than 30 characters reject the string with a message that tells the user that the message was too long and end the program.

If the user enters a short message your program will center the message output on an 80 character line and display the centered message as output.

Use *len(string)* to obtain the length of the input string. For example: `len( "asdf" )` would result in the value 4. Centering this string would require inserting 38 spaces before the `"asdf"`. **Note:** Loops are neither required nor suggested to complete this question.

Sample program run:

```
Please enter a string less than 30 characters:  1234567890123456789012345678901
I'm sorry, your string is too long, good bye.
```

Sample program run:

```
Please enter a string less than 30 characters: 1234567890123
Your centered string is:
                                 1234567890123
```

3.2 Grading Scheme and Other Requirements

To be eligible for grading your solution must be named **centeredMessage.py** and must contain a statement of authorship. Penalties will be assigned if either of these conditions are not met.

DDIT Evidence	IPO Chart	2
	Pseudocode/Flow Chart	2
	Design Trace	2
	Final Tests	2
Style Rules	spacing	1
	variable naming	1
Implementation	user input	2
	spacing correct	6
	output similar to sample	2
Total		20

4 Problem: Plinko!

4.1 Problem Statement

On the popular TV game show "The Price is Right" one of the games involves letting a flat disk slide down a triangular board that has pegs positioned to push the disk left or right into a slot that indicated how many dollars you had won. To earn up to 5 disks you had to play a pricing game.

You are going to write a version of this game that uses random numbers to simulate a lot of the game play such as the number of disks available and the left or right motion of the disk.

Your program should pick a random number between 1 and 5 inclusive to determine how many times the player gets to try their luck. Once the number of disks is chosen the user will drop a disk onto the game board. Show a message to indicate this is happening.

As the disk moves down the board it will hit 5 pins. At each of the pins simulate the hit by generating a random number between 1 and 100 inclusive. If the number is between 1 and 50 inclusive print the message "tick" which will mean the disk moved left and if it is between 51 and 100 inclusive print the message "tock" which will mean it moved right. Keep track of the total number of left and right movements.

After the 5 pin hits have been simulated use the total number of left and right to determine what prize the player has won. Add this number to the total prize winnings and repeat for each disk the player has.

Figure 1: The Plinko Game

Hint: You may wish to use 0 to mean the center, negative numbers to mean left and positive numbers to mean right. For example:

-5	-4	-3	-2	-1	0	1	2	3	4	5
0	100	500	1000	0	10000	0	1000	500	100	0

Sample program run:

```
Congratulations you have 3 disks to play with.
Dropping disk 1...
-tick-
-tock-
-tick-
-tick-
-tick-
The disk landed in $500, congratulations!

Dropping disk 2...
-tock-
-tock-
-tock-
-tick-
-tick-
The disk landed in $0, aww, too bad!
Dropping disk 3...
```

```
-tock-
-tock-
-tick-
-tock-
-tock-
The disk landed in $500, congratulations!

Your total prize winning today is $1000!
```

You may, if you wish, display all the ticks and tocks for each disk on a single line for neatness.

4.2 Grading Scheme and Other Requirements

To be eligible for grading your solution must be named **plinko.py** and must contain a statement of authorship. Penalties will be assigned if either of these conditions are not met.

DDIT Evidence	IPO Chart	2
	Pseudocode/Flow Chart	8
	Design Trace	5
	Final Tests	2
Style Rules	spacing	2
	variable naming	2
Implementation	random number of disks	2
	5 pin hits	5
	bonus: loop used for pins	5
	correct prize award	5
	loops for each turn	5
	output similar to sample	2
Total		40