CompSci 101 - Assignment 01

Due: 4:30pm, Friday 14th August 2015.

Worth: This assignment is marked out of 10 and is worth 3% of your final mark.

Topics covered:

- Arithmetic operators
- Printing output
- Manipulating string objects
- Generating random numbers
- Getting user input

NOTE: Each of your files must include a docstring at the top of the file containing your name, UPI, ID number and a description of the program.

Submit the files containing your exercises using the Assignment Dropbox:

https://adb.auckland.ac.nz/Home/

QUESTION 1 (2MARKS)

Write a program which, given a start time and an end time, calculates the elapsed time (in hours, minutes and seconds). The times in this program use a 24 hour clock, e.g., 15:23:15 is 23 minutes and 15 seconds past 3pm. You can assume that the end time is always later than the start time. The first two lines of your program initialise the variables in the following way (you will need to change the values assigned to these two variables when you test your program):

```
start = "06:55:55"
end = "07:04:57"
```

The time elapsed should always be a time between 00:00:00 and 23:59:59. The start time, the end time and time elapsed always have the form "hh:mm:ss", i.e., a single digit hour, minute or second has a "0" in front of it. Below are four example outputs from the completed program (using different variable values). Your program must give the correct output in the same format as the outputs in the four examples below:

```
Start time:  15:26:12   End time:  21:40:36
Total time elapsed:  06:14:24
```

```
Start time:  06:25:03   End time:  07:16:03
Total time elapsed:  00:51:00
```

```
Start time:  06:55:12   End time:  07:04:01
Total time elapsed:  00:08:49
```

```
Start time:  06:05:12   End time:  11:04:01
Total time elapsed:  04:58:49
```

Include this exercise in a module (file) named 'YourUPIA1Q1.py', e.g., afer023A1Q1.py.

QUESTION 2 (2 MARKS)

Write a program which simulates four transactions on a bank account. Initially the bank account has a balance of $11568. Each transaction is either a negative amount (a withdrawal) or a positive amount (a deposit). The amount of each transaction (withdrawal or deposit) is a random integer between 100 and 150 inclusive. Below is the statement which initialises the variable, `balance`:

```
balance = 11568
```

In the output, the amount of each transaction is displayed inside parentheses following the balance (after the transaction has been applied). The final line of output shows the overall sum of the four transactions.

Hint: in order to create a transaction which is either a withdrawal or a deposit you will find it useful to generate a random number which is either -1 or 1. The random transaction amount is then multiplied by this number.

Below are two example outputs using the completed program. Your program must give a correct output in the same format as the outputs in the two examples below:

```
Initial balance: $11568
1: $11674 (106)
2: $11816 (142)
3: $11926 (110)
4: $11815 (-111)
==========================
Sum of transactions: $247
==========================
```

```
Initial balance: $11568
1: $11673 (105)
2: $11538 (-135)
3: $11663 (125)
4: $11537 (-126)
==========================
Sum of transactions: $-31
==========================
```

Include this exercise in a module (file), named 'YourUPIA1Q2.py', e.g., afer023A1Q2.py

QUESTION 3 (2 MARKS)

Write a program which prints a 5 letter word (I have used names in my examples) in the form of a parallelogram as in the examples below. Nine lines of the output print sections of the word, the first line prints the first letter of the word, the next line prints the next two letters of the word, etc. , the fifth line prints the whole word and the next three lines contract the word until the last line which prints just the last letter of the word. Each of these 9 lines is indented by two spaces and the whole nine lines are enclosed inside four rows of stars.

```
word = "MARIA"
```

Below are two example outputs using the completed program (using two different 5 letter words). Your program must give the correct output in the same format as the outputs in the two examples below:

```
*********
*********
      M
     AR
    ARI
   ARIA
  MARIA
  ARIA
  RIA
  IA
  A
*********
*********
```

```
*********
*********
      J
     AM
    AME
   AMES
  JAMES
  AMES
  MES
  ES
  S
*********
*********
```

Include this exercise in a module (file), named 'YourUPIA1Q3.py', e.g., afer023A1Q3.py.

QUESTION 4 (2 MARKS)

Write a program which encrypts (or decrypts) a message. The message is always 25 letters in length. The last two lines of code below ensure that the message is always 25 letters in length.

```
original_message = "Pagg rm rfomieugisanrn t!"
#Makes sure the message is 25 letters in length
message = original_message * 25
message = message[0: 25]
```

Copy the above statements which initialise the variables into your program. The program uses the following encryption method. First the 25 letter message is broken up into five 5 letter words, i.e., a five by five square of letters. Then the message is recreated by going down the five by five square of letters from left to right. For example, the message `"abcdefghijklmnopqrstuvwxy"` is reshaped into five rows:

```
abcde
fghij
klmno
pqrst
uvwxy
```

and then the message is reconstructed going down each column starting from the first column to the last column: `"afkpubglqvchmrwdinsxejoty"`

Below are three example outputs using the completed program (using three different starting messages). Your program must give the output in the same format as the outputs in the three examples below.

```
Original message: E  l ntbidjhraaoiinyyslt!
Encrypted message: Enjoy this brilliant day!
```

```
Original message: Meet at 11 in the library
Encrypted message: Ma hbetiere n at1 lr 1tiy
```

```
Original message: Ma hbetiere n at1 lr 1tiy
Encrypted message: Meet at 11 in the library
```

Include this exercise in a module (file), named 'YourUPIA1Q4.py', e.g., afer023A1Q4.py.

QUESTION 5 (2 MARKS)

Write a program which creates a word chain. The word chain starts with a word (of any length) and allows the player to create a new word by making one change (which affects two consecutive letters) to the current word. In this way the user creates four new words in the chain. The user is first prompted for the position of the first of the two letters they wish to change (the first letter is position 0), then prompted for the replacement two letters, and then the program prints the changed word. This process is repeated four times. Below is the statement which initialises the variable, `word`:

```
word = "frank"
```

Below are two example outputs using the completed program (using two different starting words). Your program must give the output in the same format as the outputs in the two examples below. Note that the string of "=" symbols is the same length as the final combined chain of words.

```
Starting word: frank
Position: 0
Replacement letters: bl
1 Word is now: blank
Position: 3
Replacement letters: me
2 Word is now: blame
Position: 3
Replacement letters: st
3 Word is now: blast
Position: 2
Replacement letters: un
4 Word is now: blunt
==================================
frank->blank->blame->blast->blunt->
==================================
```

```
Starting word: bird
Position: 2
Replacement letters: ns
1 Word is now: bins
Position: 0
Replacement letters: ta
2 Word is now: tans
Position: 1
Replacement letters: ip
3 Word is now: tips
Position: 2
Replacement letters: lt
4 Word is now: tilt
==============================
bird->bins->tans->tips->tilt->
==============================
```

Include this exercise in a module (file), named 'YourUPIA1Q5.py', e.g., afer023A1Q5.py.