SMU SINGAPORE MANAGEMENT UNIVERSITY

School of Information Systems
Academic Year 2019/20
Term I

IS111: Introduction to Programming

Lab Exercise 07: While Loops

Optional submission deadline: **Sun 06 Oct 2019 @ 2359 hrs**

INSTRUCTIONS

You are expected to read through the accompanying lesson 07 notes before attempting lab exercise 07. You should attempt the questions using **Visual Studio Code**. Each question should be submitted on a different file.

Do not submit Level 1 questions; use the discussion forum for that.

To submit, you should:

1. Name your 2 files as **week7-qn2-1.py**, **week7-qn2-2.py** and **week7-qn3-1.py**.
2. Submit the 2 .py files to **eLearn → Assignments** by the stipulated deadline.

Level 1 Questions (no submission required; use the discussion forum)

1.1 Random Generator

Write a Python function `random_generator` that takes in three integer parameters - *a*, *b* and *c*. The function should continuously generate a random integer between *a* and *b*, and return None when the sum of the random numbers that are generated is at least *c* (i.e., greater than or equal to *c*).

Here is a sample output when `random_generator(3,10,15)` is called.

```
adding 3, total is 3
adding 5, total is 8
adding 6, total is 14
adding 9, total is 23 # stops here because 23 > 15
```

1.2 Multiplication Table

Write a Python function `multiplication_table` that takes in a single positive integer *n*. The function should use a **while** loop to generate the first 10 lines of the multiplication table for *n*.

Here is a sample output when `multiplication_table(7)` is called.

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

1.3 Binary to Decimal

Write a Python function `binary_to_decimal` that takes in a single string parameter *s*. The string s is non empty, and contains a sequence of binary digits (i.e., either '1' or '0'). The function should use a **while** loop to compute and return the decimal number representation of the binary string *s*.

The following table shows some examples of binary strings and their decimal representations.

Binary string	Computation	Decimal
'1011'	$1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$	11
'01001100'	$0x2^7 + 1x2^6 + 0x2^5 + 0x2^4 + 1x2^3 + 1x2^2 + 0x2^1 + 0x2^0$	76

You can use the following code snippet to test your code:

```
binary_to_decimal('1011') # returns 11
binary_to_decimal('01001100') # returns 76
```

Level 2 Questions

2.1 Decimal to Binary

(This is the reverse of Qn 1.3; hence, you *should* attempt Qn 1.3 first.)

Write a Python function `decimal_to_binary` that takes in a single **positive** integer parameter *d*. The function should use a **while** loop to compute and return the binary number representation of integer *d*, as a string.

Here is an example of how you can convert from a decimal number to its binary number representation (through repeated division by 2, until the quotient is 0).

Decimal	Computation	Binary string
11	11/2 = 5 **R 1** 5/2 = 2 **R 1** 2/2 = 1 **R 0** 1/2 = 0 **R 1** (stop here since quotient is 0) Hence, if you read the remainders **from bottom up**, you get the binary 1011.	'1011'

You can use the following code snippet to test your code:

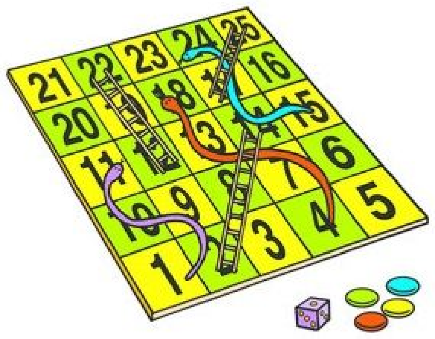
```
decimal_to_binary(11) # returns '1011'
decimal_to_binary(76) # returns '01001100'
```

📌 **You must return your binary number as a string!**

2.2 Snakes and Ladders

Write a simplified Snake and Ladder board game using a Python function `snakes_and_ladders`. The function takes in the following parameter:

- `snakeladder_list` (type: `list`): This is a list of tuples, where each tuple is in the format `(board_num, step)`. Each tuple therefore represents either a snake or a ladder. The `board_num` refers to the board number of a snake or ladder, and `step` refers to the number of steps/hop your counter will need to take. The number of `steps` can be positive (where your counter climbs up a ladder) or `negative` (where it slides down a snake). The board_num is always between 1 and the board size.

	In this example, `snakeladder_list = [(2,12), (12,10), (17,8), (11,-10), (18,-14), (24,-9)]`

You are to play the game continuously by moving your counter forward by the dice number, which is a randomly generated number between 1 to 6 (inclusive). If your resulting new board number matches any of the snakes and ladders' board numbers (ie. `board_num`), you are required to make an additional move, based on the corresponding `step` in the matching tuple.

For example, if your new board number after rolling the dice is 10 and there is a tuple `(10,5)` in the `snakeladder_list`, your counter will make an additional move from 10 to 15 (ie. take 5 steps forward), and your new counter is now 15.

In this simplified version, we assume that the size of the board is 20. Hence, the player wins when the counter first reaches (or exceeds) 20.

Here is a sample output when `snakes_and_ladders([(6,-3), (10,5), (13,-2), (14,5), (18,-6)])` is called.

```
Welcome to Snake and Ladder board game. Your board number is 0.
Enter 'r' to roll the dice, or type 'No' to quit: r
The dice number is 6 and you land on a space with (6,-3). Your new
board number is 3.
Enter 'r' to roll the dice, or type 'No' to quit: r
The dice number is 4. Your new board number is 7.
Enter 'r' to roll the dice, or type 'No' to quit: r
The dice number is 3 and you land on a space with (10,5). Your new
board number is 15.
Enter 'r' to roll the dice, or type 'No' to quit: r
The dice number is 6. Your new board number is 20.
You win the game!
```

```
Welcome to Snake and Ladder board game. Your board number is 0.
Enter 'r' to roll the dice, or type 'No' to quit: r
The dice number is 5. Your new board number is 5.
Enter 'r' to roll the dice, or type 'No' to quit: No
Thank you for playing.
```

3 Brownie Points 🍪

3.1 Computing sin(x)

The sine is 'a trigonometric function of an angle'. The sine of an angle *x* (where *x* is in radians), may be computed based on the following formula:

$$sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Write a Python function `compute_sine` that **uses while loop(s)** to compute the sine value of any given *x*. The function takes in two parameters:

- *x* (type: `float`): This is the value of *x* for which `sin(x)` should be computed. The value of *x* is in radians, which means that $-2\pi \leq x \leq 2\pi$. Recall that π=3.142.
- *n* (type: `int`): This is a positive **odd-numbered** integer.

The function should compute the sine of *x* using all the terms in the given formula (as shown above), up to and including the term x^n. The function should return the value of `sin(x)`, rounded off to 5 decimal places.

Notice that the accuracy of `sin(x)` increases with larger *n* values.

For instance, `compute_sine(5,29)` should give the result:

$$sin(5) = 5 - \frac{5^3}{3!} + \frac{5^5}{5!} - \frac{5^7}{7!} + \ldots + \frac{5^{29}}{29!} \approx \; -0.95892$$

You can use the following code snippet to test your code:

```
compute_sine(5, 29) # returns -0.95892
compute_sine(0.5, 21) # returns 0.47943
compute_sine(-0.2, 19) # returns -0.1987
```

📌 You may need to use two (2) while loops for this solution.

📌 Suppose `sign` is a variable with initial value of 1. You can alternate `sign` between 1 and -1 by multiplying `sign` with -1 within each loop. For example, `1x(-1) = -1`, and `-1x(-1) = 1`.

💾 30 Sep 2019