School of Information Systems
Academic Year 2019/20
Term I

IS111: Introduction to Programming

Lab Exercise 09: Revision

Optional submission deadline: **Sun 20 Oct 2019 @ 2359 hrs**

INSTRUCTIONS

You should attempt the questions using **Visual Studio Code**. Each question should be submitted in a different file.

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

To submit, you should:

1. Name your 3 files as **week9-qn2-1.py**, **week9-qn2-2.py** and **week9-qn3-1.py**.
2. Submit the 3 .py files to **eLearn → Assignments** by the stipulated deadline.

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

1.1 CamelCasing

In Python, variable names that contain multiple words can be concatenated using underscores (e.g., `long_word`) or using camel casing (e.g., `longWord`).

Write a Python function `count_words` that takes in a single string parameter *s* that is written in camel casing, and returns the number of words that it contains. You may assume that:

- String *s* contains at least one word.
- The first word in string *s* is written in lowercase.
- All subsequent words in string *s* have the first letter in uppercase, and the rest of the letters in lowercase.

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

```
count_words('alphabets') # returns 1
count_words('camelCasing') # returns 2
```

1.2 Perfect Squares

A perfect square is an integer that is a square of an integer. For example, 4, 9, 16 and 25 are all perfect squares, since 2x2=4, 3x3=9, 4x4=16 and 5x5=25.

Write a Python function `count_perfect_sq` that takes in two positive integer parameters *a* and *b*, and returns the number of perfect squares that fall within the range *a* and *b* (both inclusive).

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

```
count_perfect_sq(1,3) # returns 1
count_perfect_sq(4,30) # returns 4
count_perfect_sq(37,47) # returns 0
```

1.3 Robot Moves

A robot can move up, down, left or right in each time unit. If the robot moves right/left, its x-position is changed positively/negatively. If the robot moves up/down, its y-position is changed positively/negatively.

For example, if the robot was originally at (0,0) and moves left, followed by up, then its coordinates become (0,0) → (-1,0) → (-1,1).

Write a Python function `robot_dest` that takes in single string parameter *s* containing the sequence of moves (either `'U'`, `'D'`, `'L'` or `'R'`), and then returns a single tuple `(x,y)` containing the *x* and *y* coordinates of the robot's final coordinates. You may assume that the robot will always start from (0,0).

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

```
robot_dest('UULRDRDDDD') # returns (1,-3)
robot_dest('URLD') # returns (0,0)
```

Level 2 Questions

2.1 Spotify Playlist

You have decided to subscribe to Spotify so that you can play music during your long commute between your home and school. Your favorite playlist has *n* songs; each of these *n* songs has a song title and a song duration.

Write a Python function `play_music` that takes in the following two parameters:

- `playlist` (type: `list`): This is a list of *n* tuples (as illustrated above), whereby each of the *n* tuples is in the form (t_i, d_i), where t_i is the title of the i^{th} song, and d_i is the duration (in seconds) of the i^{th} song.
- `commute_duration` (type: `int`): This is an integer, which gives the duration of the commute (in seconds).

The function should compute and return the number of unique songs that are played completely during the commute duration, when the songs in the playlist are played in order (i.e., not randomized). You should assume that if the commute duration is longer than your entire playlist, the playlist will repeat from the beginning again.

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

```
play_music([('a',360),('b',300),('c',220),('d',400)],0) # returns 0
play_music([('a',360),('b',300),('c',220),('d',400)],40) # returns 0
play_music([('a',360),('b',300),('c',220),('d',400)],290) # returns 0
play_music([('a',360),('b',300),('c',220),('d',400)],660) # returns 2
play_music([('a',360),('b',300),('c',220),('d',400)],900) # returns 3
play_music([('a',360),('b',300),('c',220),('d',400)],2000) # returns 4
```

2.2 Diagonal Difference of Square Matrix

A square matrix has *n* rows and *n* columns; here is an example of a 3x3 square matrix:

1	2	3
4	5	6
8	8	9

This 3x3 square matrix can be represented as a list of lists, in the following form:

```
[[1,2,3], [4,5,6], [8,8,9]]
```

Write a Python function `compute_diagonal_diff` that takes in a single parameter:

- `matrix_list` (type: `list`): This is a list of *n* lists, that represents the square matrix (as given by the example above). Each of the *n* lists in `matrix_list` comprises *n* elements.

The function should compute and return the absolute difference of the diagonals in the given matrix_list. For instance, given the 3x3 matrix above, the left-to-right diagonal is 1+5+9=15, and the right-to-left diagonal is 3+5+8=16. Hence, the absolute difference in the diagonals is given by |15-16|=1.

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

```
compute_diagonal_diff([[1,2,3], [4,5,6], [8,8,9]]) # returns 1
compute_diagonal_diff([[11,2,4,4],[10,4,-4,3],[1,1,5,5],[3,3,-8,4]])
# returns 20
```

3 Brownie Points 🍪

3.1 Magic Square Checker

A magic square is a square grid with *n* rows and *n* columns (i.e., a *n*x*n* grid), such that:

- each of the *n*x*n* square grids is filled with a distinct positive integer in the range 1, 2, 3, ..., n^2; and
- the sum of the integers in each row, column and diagonal is equal.

For example, the following 3x3 square grid is a magic square because each of the rows, columns and diagonals add up to 15.

2	7	6
9	5	1
4	3	8

This 3x3 square matrix can be represented as a list of lists, in the following form:

```
[[2,7,6], [9,5,1], [4,3,8]]
```

Write a Python function `check_magic_square` that takes in a single parameter:

- `matrix_list` (type: `list`): This is a list of *n* lists, that represents the square matrix. Each of the *n* lists in `matrix_list` comprises *n* elements.

The function returns `True` if the square grid represented by `matrix_list` is a magic square, and `False` otherwise.

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

```
check_magic_square([[2,7,6], [9,5,1], [4,3,8]]) # returns True
check_magic_square([[1,2,3], [4,5,6], [7,8,9]]) # returns False
check_magic_square([[16,3,2,13], [5,10,11,8], [9,6,7,12],
[4,15,14,1]]) # returns True
```

💾 12 Oct 2019