School of Information Systems
Academic Year 2019/20
Term I

IS111: Introduction to Programming

Lab Exercise 10: Dictionaries

Optional submission deadline: **Sun 27 Oct 2019 @ 2359 hrs**

INSTRUCTIONS

You are expected to read through the accompanying lesson 10 notes before attempting lab exercise 10. 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 3 files as **week10-qn2-1.py**, **week10-qn2-2.py** and **week10-qn3-1.py**.
2. Submit the 3 .py files to **eLearn → Assignments** by the stipulated deadline.

Guidelines for Forum Discussion

You can refer to this link for guidelines on how to ask questions in a discussion forum. For instance, if someone has posted a solution that is exactly the same as yours (**or very similar to yours**), then you need not post your solution again. You are highly encouraged to help your coursemates with their questions, so that everyone can learn together.

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

1.1 Practical Exam Grades

The IS111 class has just finished the programming practical exam. You are given a dictionary containing everyone's grades, for example, in the form:

```
grades = {'A':9, 'B':8.5, 'C':7, 'D':3.5, 'E':1.5, 'F':7}
```

Write a Python function `compute_statistics`, which takes in a single dictionary parameter `grades` containing information about everyone's grades, and then returns a

dictionary containing some statistics of the grades. Specifically, the output dictionary should contain the min (minimum score), max (maximum score), and mean (average score, correct to 2 decimal places), as follows:

```
{'min':1.5, 'max':9, 'mean': 6.08}
```

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

```
compute_statistics({'A':9, 'B':8.5, 'C':7, 'D':3.5, 'E':1.5, 'F':7})
# returns {'min': 1.5, 'max': 9, 'mean': 6.08}
compute_statistics({'A':22, 'B':33, 'C':44, 'D':55, 'E':66})
# returns {'min': 22, 'max': 66, 'mean': 44.0}
```

1.2 Personal Demographics

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

- `weight_dict` (type: `dict`): This is a dictionary with key-value pairs as names and weights (in kg) of persons. An example is `weight_dict = {'Anne':53, 'Bob':37, 'Carol':83, 'Dave':44}`.
- `height_dict` (type: `dict`): This is a dictionary with key-value pairs as names and heights (in meters) of persons. An example is `height_dict = {'Anne':1.56, 'Bob':1.34, 'Carol':1.8, 'Dave':1.7}`.
- `age_dict` (type: `dict`): This is a dictionary with key-value pairs as names and age (in years) of persons. An example is `age_dict = {'Anne':15, 'Bob':17, 'Carol':21, 'Dave':14}`.

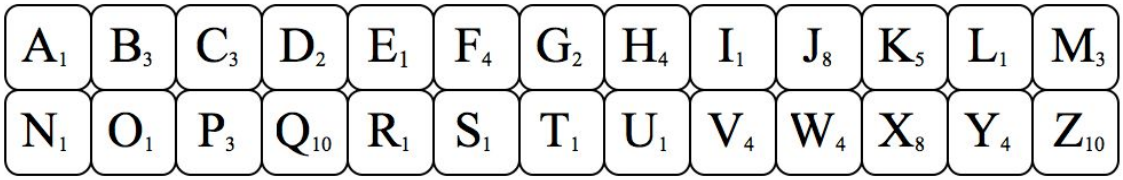
The function should return a single dictionary with the key as the name of the person, and value as a tuple in the form `(age, weight, height)`.

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

```
weight_dict = {'Anne':53, 'Bob':37, 'Carol':83, 'Dave':44}
height_dict = {'Anne':1.56, 'Bob':1.34, 'Carol':1.8, 'Dave':1.7}
age_dict = {'Anne':15, 'Bob':17, 'Carol':21, 'Dave':14}
combine_demographics(weight_dict, height_dict, age_dict)
# returns {'Anne':(15,53,1.56), 'Bob':(17,37,1.34),
'Carol':(21,83,1.8), 'Dave':(14,44,1.7)}
```

1.3 Scrabble

In the game of Scrabble, players get points by spelling (English) words on a crossword puzzle grid, using letter tiles. Each letter tile has a value. The score of a word is obtained by the total value of each individual letter tile.



For instance, the score of a word 'apple' is given by 1+3+3+1+1 = 9.

Write a Python function `compute_scrabble_score` that takes a single string input parameter *word* (representing the English word), and returns the equivalent scrabble score for that word. The function should make use of a **dictionary**. Note that the input word may contain both upper cases and lower cases.

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

```
compute_scrabble_score('apple') # returns 9
compute_scrabble_score('Programming') # returns 19
```

Level 2 Questions

2.1 NRIC Formula

The Singapore NRIC (National Registration Identity Card) number is in the form of `#0000000@` (e.g., `S1234567D`), where:

- `#` is a character that can be `S`, `T`, `F` or `G`, depending on the status of the person;
- `0000000` is a 7-digit serial number; and
- `@` is the checksum letter calculated with respect to `#` and `0000000`.

A checksum is usually a small-sized data, that is computed based on the actual data, to ensure that the actual data is correct (and not corrupted during transmission).

Although the algorithm for computing the checksum (i.e., last character) of the Singapore NRIC number is not officially released, there are several online sources (such as this and this) that provide details on how the checksum is computed.

Write a Python function `compute_checksum` that takes in a single string parameter `partial_nric` containing a partial NRIC string (e.g., `S1234567`), and returns the checksum (e.g., `D`).

Here are the steps taken to compute the checksum, given the partial NRIC `S1234567`.

1. Compute the weighted sum, which is given by the following table.

partial NRIC	S	1	2	3	4	5	6	7	checksum
multiplication_weight	-	2	7	6	5	4	3	2	?

   weighted sum = (1*2) + (2*7) + (3*6) + (4*5) + (5*4) + (6*3) + (7*2)

   = 2 + 14 + 18 + 20 + 20 + 18 + 14

   = 106

2. If the first character of the NRIC is `T` or `G`, add 4 to the weighted sum.

   Since the first letter is `S` in this case, we do not need to add 4 to the weighted sum, and it remains at 106.

3. Obtain the modulo 11 of the adjusted weighted sum (after Step 2).

   The modulo 11 of the adjusted weighted sum (after Step 2) is the remainder when the adjusted weighted sum is divided by 11. In this example, 106 mod 11 = 9 with remainder 7.

4. Use the table below to obtain the checksum, depending on the first character of the NRIC (i.e., whether it is `S`, `T`, `F` or `G`) and modulo 11 result (computed from Step 3).

modolo 11 result	10	9	8	7	6	5	4	3	2	1	0
S, T	A	B	C	D	E	F	G	H	I	Z	J
F, G	K	L	M	N	P	Q	R	T	U	W	X

In this example, our modulo result from Step 3 is 7, and the first character of the NRIC is S. Hence, the checksum is D.

You are not allowed to use lists in this question. You are allowed only to use dictionary.

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

```
compute_checksum('S1234567') # returns 'D'
compute_checksum('G1234567') # returns 'X'
compute_checksum('T7654321') # returns 'B'
compute_checksum('F7654321') # returns 'Q'
```

2.2 ISBN-10 Checksum Algorithm

The International Standard Book Number (ISBN) is a unique numeric commercial book identifier. The ISBN may either be 10 digits long, or 13 digits; it is 13 digits long if assigned on or after 1 January 2007, and 10 digits long if assigned before 2007.

The ISBN-10 number comprises a 9-digit number, followed by a 1-digit checksum. For instance, if we consider an ISBN-10 number `a-bcde-fghi-j`, then *j* is the checksum.

Here are the steps to check if a ISBN-10 number `a-bcde-fghi-j` is valid or not:

1. Compute the weighted sum, which is given by:

   `weighted = a*10 + b*9 + c*8 + d*7 + e*6 + f*5 + g*4 + h*3 + i*2`

2. Obtain the modulo 11 of the weighted sum (from Step 1):

   `temp1 = (weighted) mod 11`

3. Subtract `temp1` (from Step 2) from 11:

   `temp2 = 11 - temp1`

4. The checksum is given by the modulo 11 of `temp2` (from Step 3).

   `checksum = (temp2) mod 11`

If the `checksum` obtained from Step 4 is the same as *j*, then the ISBN-10 is valid number; otherwise, it is not a valid number.

Write a Python function `validate_isbn10` which takes in a single input string `input_isbn`, and returns a boolean `True` or `False`, depending on whether the `input_isbn` is valid or not. Note that `input_isbn` may contain different numbers of hyphens (-), which may appear at different positions. However, a valid ISBN-10 should contain exactly 10 digits (regardless of whether there are hyphens or not, in the input).

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

```
validate_isbn10('0-3064-0615-2') # returns True
validate_isbn10('03064-06152') # returns True
validate_isbn10('0-545-01022-5') # returns True
validate_isbn10('0-3064-0615-20') # returns False (has 11 digits)
validate_isbn10('0-545-01022-4') # returns False (incorrect checksum)
```

3 Brownie Points 🍪

2.2 (Simplified) Grab Driver Allocation Algorithm

Whenever a passenger makes a GrabCar booking request, the company has algorithm(s) in place, that will assign the best driver to the passenger.

A simplified version of the GrabCar booking assignment flow is reproduced below:

```
for each booking request:
    find nearby available GrabCar drivers
    if there are available GrabCar drivers
        assign the best driver to the passenger
    else
        recycle booking to next round of broadcasting and assignment
```

For the purpose of this practice set, we will assume that the best driver is defined to be the one who is ***nearest*** (based on some distance) to the passenger's current location.

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

- `passenger_loc` (type: `tuple`): This is a tuple containing the passenger's location in the form of `(latitude,longitude)` - for example `(1.297490, 103.849600)`.
- `driver_loc` (type: `dict`): This is a dictionary containing each driver's name and his/her current location in the form of a tuple `(latitude,longitude)`. For example, `{'A':(1.318350,103.843100),'B':(1.290210,103.859512)}` You may assume that each driver's name is unique, and that the dictionary has a minimum length of 1.

The function should return a tuple in the form `(driver_name, distance)` containing the name of the driver who is nearest to the passenger, as well as the corresponding distance (in km, rounded off to 2 decimal places) from the passenger. If there is more than one driver who is nearest to the passenger, choose any one of the nearest drivers *randomly*.

You may use the Great Circle Distance to estimate the distance between any two latitude and longitude points; the formula is reproduced below for your convenience:

Let lat_1, lon_1 and lat_2, lon_2 be the geographical latitudes and longitudes of two points 1 and 2 (**in radians**). Then, the distance between them is given by:

$$d = 6371 \times acos(sin(lat_1) \times sin(lat_2) + cos(lat_1) \times cos(lat_2) \times cos(lon_1 - lon_2))$$

You can check out Python implementations of the Great Circle Distance by doing some Googling. It is important to note that $lat_1, lon_1, lat_2, lon_2$ are in radians; you can convert a latitude and longitude coordinate to radians using `math.radians(lat)` and `math.radians(lon)`.

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

```
get_nearest_driver((1.297490, 103.849600),
{'A':(1.318350,103.843100),'B':(1.290210,103.859512)})
# returns ('B', 1.37)
```

💾 14 Oct 2019