Assignment 1 (15 Marks)

Due: Friday 29 March 2019 at 20:30

Introduction

Welcome to Travel Inspiration!

We are going to be building a travel recommendation program. Out of ideas where to travel? Travel Inspiration has got you covered.

The program will ask the user questions about their preferences in a travel destination. It will use these to recommend the best match in a database of potential travel destinations.

You are provided with a file `travel.py`, which serves as a template for the assignment. This file is where you should write the code for your solution. The file also provides an example of how to read the data from the database, one destination at a time. You need to implement the functionality to question the user and perform the matching. The database is in the form of a text file, `destinations.csv`. You may edit this text file to add more destinations for testing.

Overview

Tasks

This assignment is broken down into six main tasks, grouped into two categories:

- **Core**
  1. Questions & Inputs
  2. First Exact Match
  3. Climate & Season Factor
  4. Interests
- **Advanced**
  5. Input Validation
  6. Multiple Inputs

Do **not** attempt the advanced tasks unless you have implemented all the core tasks. Most of the marks for this assignment are assigned to the core tasks.

Getting Started

The archive `a1_files.zip` contains all the necessary files to start this assignment. It contains:

- `travel.py`: The main assignment file. **Add your code to this file.**
- `destinations.csv`: The destinations database, in Comma Separated Value (CSV) format.
- `destinations.py`: Code to help Python read from a database. You are not expected to understand the code in the `destinations.py` file. The provided `travel.py` demonstrates everything you need to do to use this code.

Core Tasks

1. Questions & Inputs

The first task involves creating the questionnaire and prompting the user for input. Your program must duplicate the functionality demonstrated below.

The final line of output is the recommended destination. For now, this should be None, but in the following tasks this should be replaced with the recommended destination.

You may assume that the user will only enter valid input.

Required Output

```
Welcome to Travel Inspiration!

What is your name? Dora

Hi, Dora!

Which continent would you like to travel to?
  1) Asia
  2) Africa
  3) North America
  4) South America
  5) Europe
  6) Oceania
  7) Antarctica
> 4

What is money to you?
  $$$) No object
  $$) Spendable, so long as I get value from doing so
  $) Extremely important; I want to spend as little as possible
> $$$

How much crime is acceptable when you travel?
  1) Low
  2) Average
  3) High
> 3

Will you be travelling with children?
  1) Yes
  2) No
> 2

Which season do you plan to travel in?
  1) Spring
  2) Summer
  3) Autumn
  4) Winter
> 1

What climate do you prefer?
  1) Cold
  2) Cool
  3) Moderate
  4) Warm
  5) Hot
> 4

Now we would like to ask you some questions about your interests, on a scale
of -5 to 5. -5 indicates strong dislike, whereas 5 indicates strong interest,
and 0 indicates indifference.

How much do you like sports? (-5 to 5)
> -5
```

```

How much do you like wildlife? (-5 to 5)
> -2

How much do you like nature? (-5 to 5)
> -4

How much do you like historical sites? (-5 to 5)
> -1

How much do you like fine dining? (-5 to 5)
> -3

How much do you like adventure activities? (-5 to 5)
> -4

How much do you like the beach? (-5 to 5)
> -2

Thank you for answering all our questions. Your next travel destination is:
None
```

User input is ***`bold, italic and blue`***.

Your program must match this output exactly. Including spaces, spelling and punctuation. Your programs will be tested using an automated system that will match your program's output with the expected output. A mismatch in the output will result in the test failing. Note: Option lines, i.e. "`  1) Spring`", should begin with **two** spaces.

Input Requirements

For name, anything is valid. For general questions, the user input will be the value to the left of the parenthesis (i.e. 1-7 for continent; $, $$ or $$$ for money; etc.). For interest questions, the user input will be an integer between -5 and 5, inclusive. For tasks 1 to 4, you may assume that the user input is always valid.

2. First Exact Match

Extend your program to output a recommended destination on the final line, instead of None.

For this task, the recommended destination should be the **first** destination that meets the following criteria:

- The continent must match exactly (`destination.get_continent()`).
- If the user will be travelling with children, it must be kid friendly (`destination.is_kid_friendly()`).
- Cost must be less than or equal to the user's response to the money question (`destination.get_cost()`).
- Crime cannot be greater than is acceptable to the user (`destination.get_crime()`).

If no exact match is found, the recommended destination should be None.

There is no difference between varying levels of cost and crime, provided they meet the above requirements. For example, if the user's money response is $$, then a cost of either $ or $$ would be equally valid.

3. Climate & Season Factor

In addition to the requirements from 2. First Exact Match, the program must look for the best match from all destinations, according to the following criteria:

- Matches the user's climate preference exactly (`destination.get_climate()`).
- Has the greatest `season_factor`, for the user's selected season (e.g. `destination.get_season_factor('spring')`).

As above, if no exact match is found, the recommended destination should be None.

4. Interests

Extend the matching criteria from task 3 to consider the user's interests as well as the season factor. To do this, let `score = season_factor * interest_score`, where `season_factor` is defined above (e.g. `destination.get_season_factor('summer')`) and the interest score is the sum of the user's response multiplied by the destination's score, for each interest (sports, wildlife, nature, historical, cuisine, adventure, beach). More specifically:

$$\begin{aligned}\text{interest\_score} = {} & \text{response}_{sports} * \text{score}_{sports} \\ & + \text{response}_{wildlife} * \text{score}_{wildlife} \\ & + \text{response}_{nature} * \text{score}_{nature} \\ & + \text{response}_{historical} * \text{score}_{historical} \\ & + \text{response}_{cuisine} * \text{score}_{cuisine} \\ & + \text{response}_{adventure} * \text{score}_{adventure} \\ & + \text{response}_{beach} * \text{score}_{beach}\end{aligned}$$

Where $\text{response}_{interest}$ is the user's response for the given interest, and $\text{score}_{interest}$ is the destination's score for the given interest (e.g. `destination.get_interest_score('beach')`).

Advanced Tasks

5. Input Validation

For this task you may no longer assume that the user will enter valid input.

If, for any question, the user enters invalid input, they should be asked the question again until they enter valid input, as demonstrated in the example below.

Input Validation Example

```
...

How much crime is acceptable when you travel?
  1) Low
  2) Average
  3) High
> 5

I'm sorry, but 5 is not a valid choice. Please try again.

How much crime is acceptable when you travel?
  1) Low
  2) Average
  3) High
> 1

...
```

6. Multiple Inputs

Extend the continent and season questions to accept multiple inputs, so that the user can type in multiple options separated by a comma, as shown below.

A destination is now considered a match if it's continent matches one of the user's choices (there is no ordering; each choice is considered equally).

The season factor used in calculating the score is now the greatest factor of those matching the user's choice.

Extend your input validation to handle multiple inputs.

- Values can be repeated (i.e. "`2,3,2`" is a valid input).
- Numbers can have spaces around them (i.e. "`2,  3  ,4`" is a valid input).
- Numbers can be unordered (i.e. "`1,4,3`" is a valid input).

You should first implement this task to handle the simple case, where the input does not have repeated values, spaces or is unordered. Once this is working, then implement each type of validation in turn. Marking will test to determine how much of the validation your implement, and you can obtain part marks for partially implementing the validation for this section.

Multiple Inputs Example

```
...

Which continents would you like to travel to?
  1) Asia
  2) Africa
  3) North America
  4) South America
  5) Europe
  6) Oceania
  7) Antarctica
> 1,2,7,3

...

Which seasons do you plan to travel in?
  1) Spring
  2) Summer
  3) Autumn
  4) Winter
> 1, 3

...
```

Note that the words "continents" and "seasons" are now plural.

Utility Functions

To implement the assignment you will need to use some utility functions provided in the `destinations.py` file. The specific functions that you need to use for each task have been identified in the description above. The initial code inside `travel.py` provides an example of using all of these functions. The following table indicates the type of the value returned by each of the functions.

Function	Parameter	Return Type
`destination.get_name()`	no parameter	`string`
`destination.get_crime()`	no parameter	`string`
`destination.is_kid_friendly()`	no parameter	`boolean`
`destination.get_cost()`	no parameter	`string`
`destination.get_climate()`	no parameter	`string`
`destination.get_continent()`	no parameter	`string`
`destination.get_interest_score()`	`string` that is the name of the interest	`int`
`destination.get_season_factor()`	`string` that is the name of the season	`float`

Interview

In addition to providing a working solution to the assignment problem, the assessment will involve discussing your code submission with a tutor. This discussion will take place in week 6, during the practical session to which you have signed up. You **must** attend your allocated practical session, swapping to another session is not possible. In preparation for your discussion with a tutor you should consider:

- any parts of the assignment that you found particularly difficult, and how you overcame them to arrive at a solution; or, if you did not overcome the difficulty, what you would like to ask the tutor about the problem;
- whether you considered any alternative ways of implementing a given function;
- where you have known errors in your code, their cause and possible solutions (if known).

It is important that you can explain to your tutor the algorithmic logic of your program and how each of the functions that you have written operates. (For example, if you have used a while loop in a function, why this was an appropriate choice).

In the interview you must demonstrate understanding of your code. If you cannot demonstrate understanding of your code, this may affect the final mark you achieve for the assignment. A technically correct solution may not achieve a pass mark unless you can demonstrate that you understand its operation.

Submission

You must submit your assignment electronically via the assignment one submission link on Blackboard. For information on submitting through Blackboard, please read:
https://web.library.uq.edu.au/library-services/it/learnuq-blackboard-help/learnuq-assessment/blackboard-assignments

You must submit your assignment as a single Python file called `travel.py` (use this name – all lower case), and nothing else. Do **not** submit the `destinations.csv` or `destinations.py` files. Do **not** submit **any** sort of archive file (e.g. zip, rar, 7z, etc.).

Late submissions of the assignment will **not** be marked. Do not wait until the last minute to submit your assignment, as the time to upload it may make it late. Multiple submissions are allowed, so ensure that you have submitted an almost complete version of the assignment well before the submission deadline of 20:30. Your latest, on time, submission will be marked. Ensure that you submit the correct version of your assignment.

In the event of exceptional personal or medical circumstances that prevent you from handing in the assignment on time, you may submit a request for an extension. See the course profile for details of how to apply for an extension.

Requests for extensions **must** be made no later than 48 hours prior to the submission deadline. The application and supporting documentation (e.g. medical certificate) **must** be submitted via my.UQ. You **must** retain the original documentation for a minimum period of six months to provide as verification, should you be requested to do so.

Assessment and Marking Criteria

This assignment assesses course learning objectives:

1. apply program constructs such as variables, selection, iteration and sub-routines,
3. read and analyse code written by others,
5. read and analyse a design and be able to translate the design into a working program,
6. apply techniques for testing and debugging

Criteria	Mark
Programming Constructs	
• Program is well structured and readable	1
• Identifier names are meaningful and informative	1
• Algorithmic logic is appropriate	1
• Functions are used to split logic into meaningful and useful blocks, with data passed as parameters and returned appropriately	1
• Functions are well-designed, simple cohesive blocks of logic	1.5
Sub-Total	5.5
Functionality	
1. Questions and Inputs	1
2. First Exact Match	2
3. Climate and Season Factor	1
4. Travel Interests	1
5. Input Validation	1
6. Multiple Inputs	1
Sub-Total	7
Documentation	
• Comments are clear and concise, without excessive or extraneous text	0.5
• Program, and all functions having informative docstring comments	1.5
• Significant blocks of program logic are clearly explained by comments	0.5
Sub-Total	2.5
Total	**/ 15**

Your total mark will be constrained if your program does not implement all the core functionality (tasks 1 to 4). The following table indicates a multiplier that will be applied to your programming constructs and documentation marks based on how much of the core functionality your program correctly implements.

Functionality Correctly Implemented	Multiplier
Task 1 does not execute at all	25%
Task 1 is partially working	35%
Task 1 is completely working	50%
Task 2 is mostly working	75%
Task 3 is mostly working	85%
Task 4 is mostly working	100%

It is your responsibility to ensure that you have adequately tested your program to ensure that it is working correctly.

A partial solution will be marked. If your partial solution causes problems in the Python interpreter please comment out the code causing the issue and we will mark what is working. Python 3.7.2 will be used to test your program. If your program works correctly with another version of Python but does not work correctly with Python 3.7.2, you will lose **at least** all the marks for the functionality criteria.

Please read the section in the course profile about plagiarism. Submitted assignments will be electronically checked for potential plagiarism.

Detailed Marking Criteria

Criteria	**Mark**		
Programming Constructs	**1**	**0.5**	**0**
• Program is well structured and readable	Code structure highlights logical blocks and is easy to understand. Code does not employ global variables. Constants clarify code meaning.	Code structure corresponds to some logical intent and does not make the code too difficult to read. Code does not employ global variables.	Code structure makes the code difficult to read.
• Identifier names are meaningful and informative	All variable and function names are clear and informative, increasing readability of the code.	Most identifier names are informative, aiding code readability to some extent.	Most identifier names are not clear or informative, detracting from code readability.
• Algorithmic logic is appropriate	Algorithm design is simple, appropriate, and has no logical errors. Control structures are well used to implement expected logic.	Algorithm design is not too complex or has minor logical errors. A few control structures are a little convoluted.	Algorithm design is overly complex or has significant errors. Many control structures are used in a convoluted manner (e.g. unnecessary nesting, multiple looping, …).
• Functions are used to split logic into some meaningful and useful blocks, with data passed as parameters and returned appropriately	Functions represent useful logical functionality and parameters and return values are appropriate.	Functions represent some useful logical functionality and parameters and return values are usually appropriate (e.g. too functions or they are too large).	Functions are not used or are not useful logical blocks. Parameters or return values are not used or are not appropriate.

Programming Constructs	**1.5**	**1**	**0.5**	**0**
• Functions are well-designed, simple cohesive blocks of logic	Program is well-designed, splitting the logic into an appropriate number of general functions, where each function performs a single cohesive logical task.	Program is split into a reasonable number of general functions. None are too complex or large.	Program may use functions correctly, but some functions are large blocks of logic that implement multiple tasks. Or, parameters are not used well.	Program makes little or no use of functions. Functions implemented are often large and complex blocks of logic. Parameters may be poorly used.

Functionality	**1**	**0.5**	**0**
• Task 1: Questions & Inputs	All prompts are correctly displayed and can accept correct inputs.	Most prompts are displayed and they can accept correct input	Few prompts are displayed or most cannot accept input.
• Task 2: First Exact Match	In all cases finds the correct destination match. (2 marks)	In all simple cases finds the correct destination match. (1 mark)	In most cases does not find the correct destination match. (0.5 or 0 marks)
• Task 3: Climate and Season Factor	In all cases finds the correct destination match.	In all simple cases finds the correct destination match.	In most cases does not find the correct destination match.
• Task 4: Travel Interests	In all cases finds the correct destination match.	In all simple cases finds the correct destination match.	In most cases does not find the correct destination match.
• Task 5: Input Validation	Handles all invalid input strings	Handles simple invalid inputs	Does not handle any invalid inputs.
• Task 6: Multiple Inputs	In all cases finds the correct destination match.	In all simple cases finds the correct destination match.	In most cases does not find the correct destination match.

Documentation	**1.5**	**1**	**0.5**	**0**
• Comments are clear and concise, without excessive or extraneous text			Comments provide useful information that elaborates on the code. These are useful in understanding the logic and are not verbose.	Many comments are irrelevant or do not provide any detail beyond what is already obvious in the code. Excessive length of some comments obscures the program logic.
• All functions having informative docstring comments	Program docstring is a clear summary of its purpose. All functions have docstrings that provide a complete, unambiguous, description of how it is to be used.	Program docstring fairly clearly summarises its purpose. All functions have docstrings that provide a complete and fairly clear description of how it is to be used.	Most docstrings provide a complete and fairly clear description.	Some docstrings provide an inaccurate description, or there are functions without docstrings.
• Significant blocks of program logic are clearly explained by comments			In-line comments are used to explain logical blocks of code (e.g. significant loops or conditionals).	In-line comments are missing in places where they would have been useful. Or, in-line comments are irrelevant or repeat what is already clear in the code.