FIT9136 Assignment 1

Semester 2 2020

Deep Mendha
Teaching Associate, Faculty of IT
Email: deep.mendha@monash.edu

Assignment Structure by Shirin Ghaffarian Maghool

Date: 18 Aug 2020

Table of Contents

1. Key Information

Format:	Individual
Due date:	7th Sept '20
Weight:	15% of unit mark

1.1. Learning outcomes

1. design, construct, test and document Python programs;
2. demonstrate how basic data types/structures function;
3. evaluate different algorithms and analyse their complexity;
4. translate problems into algorithms with appropriate implementations by investigating different strategies for the algorithm development

1.2. Do and Do NOT

Do	Do NOT
• Maintain academic integrity[1] • Get support early from this unit and other services in the university • Apply for special consideration for extensions[2]	• Leave your assignment in draft mode • Submit late (10% daily penalty applies)[3] • Submission is not accepted after 5 days of the due date, unless you have special consideration. • Import any libraries except for **math**

1.3. Marking Criteria

Your work will be marked on

Functionality	Correctly working program	**60%**
Code Architecture	Algorithms, data types, control structures and use of libraries	**10%**
Code Style	Variable names, readability, clear logic	**10%**
Documentation	Program comments, clarity and connection to code	**20%**

[1] https://www.monash.edu/rlo/research-writing-assignments/referencing-and-academic-integrity/academic-integrity
[2] https://www.monash.edu/exams/changes/special-consideration
[3] eg: original mark was 70/100, submitting 2 days late results in 56/100 (14 marks off). This includes weekends

1.4. Submission details

Submit to “Assignment 1 Submission” on moodle:

- A1_studentID.zip[4]
- Make different .py file for each question and name it as question_name.py.[5]

Containing each of the five (5) tasks covered in Section 3. Key tasks (100 marks)

[4] studentID is your student ID. E.g. if your Id was 12345678, you would submit “A1_12345678.py”

[5] Question_name is a question name of the task. E,g, for first question “Fake Passport”, you would name it as “fake_passport.py”.

2. Getting help

2.1. English language skills

if you don’t feel confident with your English.

- Talk to English Connect: https://www.monash.edu/english-connect

2.2. Study skills

If you feel like you just don’t have enough time to do everything you need to, maybe you just need a new approach

- Talk to a learning skills advisor: https://www.monash.edu/library/skills/contacts

2.3. Things are tough right now

Everyone needs to talk to someone at some point in their life, no judgement here.

- Talk to a counsellor: https://www.monash.edu/health/counselling/appointments (friendly, approachable, confidential, free)

2.4. Things in the unit don’t make sense

Even if you’re not quite sure what to ask about, if you’re not sure you won’t be alone, it’s always better to ask.

- Ask in Ed: https://lms.monash.edu/course/view.php?id=78169&section=4
- Attend a consultation: https://lms.monash.edu/course/view.php?id=78169#section-3
- Email your tutor: https://lms.monash.edu/course/view.php?id=78169&section=1

2.5. I don’t know what I need

Everyone at Monash University is here to help you. If things are tough now they won’t magically get better by themselves. Even if you don’t exactly know, come and talk with us and we’ll figure it out. We can either help you ourselves or at least point you in the right direction.

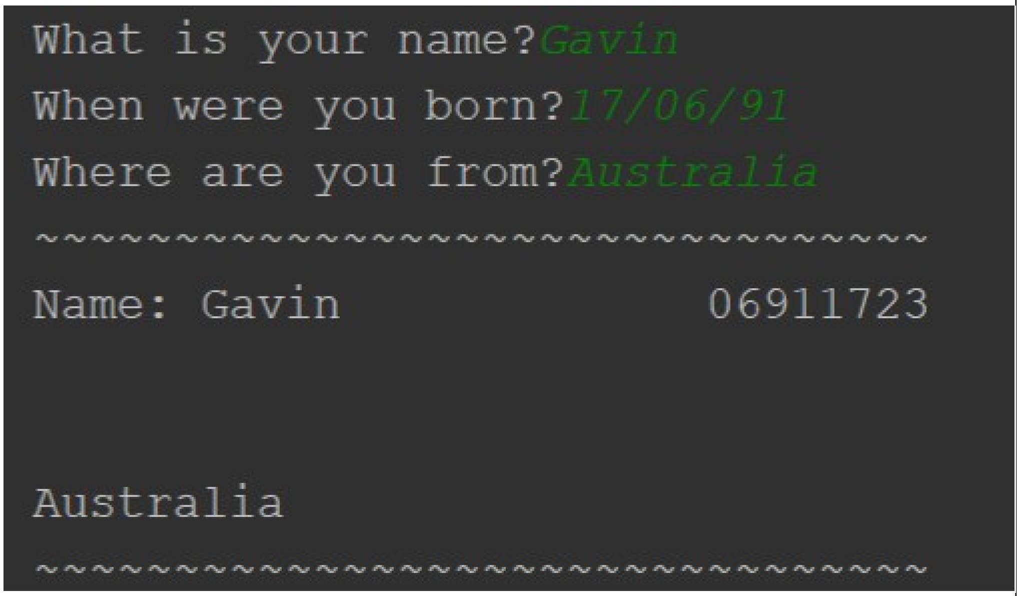
3. Key tasks (100 marks)

This assignment is broken into five parts. Each part is attempting to assess a particular aspect of programming. Read each section carefully and attempt to solve them using the techniques we have explored during the unit (Week 1 - 6).

3.1. Fake Passport (10 marks)

Write a function with the signature generate_passport(). This function asks the user to input their name, their Date of Birth (formatted as DD/MM/YY), and their country of origin. The function will then generate a passport string. More details for the task is provided below:

Function name	generate_passport()
Inputs	**USER INPUT**: • name – the user's name • country – their country of birth • date of birth – their birth date (formatted as DD/MM/YY)
Output	**PRINT:** a string as per *sample behaviour* containing Passport name, country and passport ID number **FILE OUTPUT:** None
Details	Create the passport ID using the following format: [Month] [Year] [Day] [Day + Month] e.g. for 17/06/91 it would be [06] [91] [17] [17+6 = 23] which becomes 06911723

Sample behaviour	
	```
	What is your name?Gavin
	When were you born?17/06/91
	Where are you from?Australia
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	Name: Gavin             06911723


	Australia
	~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	```

Validation	Required (**Note: Try to do as many validation as you can think of,**

of user input	**but make sure when teaching staff checks it should not give an error.**)

3.2. Common Statistics (15 marks)

Write a function with the signature statistics(numbers) where numbers is a list of numbers (e.g. [11,14,91,0,3]). Each number may be a oat or integer. Calculate the mean, median, mode, and range for these numbers. More details are provided below:

Function name	statistics(numbers)
Inputs	**ARGUMENTS**: • numbers – a list of numbers (e.g. [11,14.2,91,0,3]) **floats** and **integers** allowed
Output	**RETURN VALUE:** a list in the order of [mean, median, mode range]
Details	Calculate the: • mean – the sum of all the items divided by the number of numbers • median – the middle-most number in the sorted version of the list *if there is no middle element, use the average of the 2 middle elements* • mode – the most common number in the list • range – the difference between the smallest and largest number in the list of the list named "numbers"
Sample behaviour	>>> statistics([1,1,1,2,3,2,1,4,5,6]) [2.6,2,1,5]
Validation of user input	Required (**Note: Try to do as many validation as you can think of, but make sure when teaching staff checks it should not give an error.**)

3.3 Batch statistics (20 Marks)

Write a function with signature batch_statistics(filename) where filename is the name of a file in the directory of your Python script. Each line in the file is a list of numbers that you must pass to the function written in section 3.2 (statistics(numbers)). The output of running statistics(numbers) should be written to a file called "output.txt" where filename is the input to this function. More details are provided below.

Function name	batch_statistics(filename)
Inputs	**ARGUMENTS**: • filename – the name of a file in your python directory each line in 'filename' represents a list of numbers
Output	**FILE OUTPUT:** results of batch_statistics – one line per result
Details	1. Call on the ***statistics*** function for each line in 'filename' 2. Write the output of ***statistics*** to the corresponding line in the output file
Sample behaviour	**Demo Input File** **Demo Output File**
Validation of user input	Required (**Note: Try to do as many validation as you can think of, but make sure when teaching staff checks it should not give an error.**)

3.4. Valid Sandwich (25 marks)

Sandwiches are serious business. Write a function with signature validate_sandwich (sandwich). There are three types of bread: white, rye, and pumpernickel. For a sandwich to be valid, it must have two pieces of the same type of bread and not be obstructed by unpaired pieces of bread. Every sandwich must have a buttered piece of bread, and an un-buttered piece of bread. The buttered piece must be on top and un-buttered at bottom. More details are provided below:

Function name	validate_sandwich(sandwich)
Inputs	**ARGUMENTS**: • sandwich – a string of letters representing a sandwich
Output	**RETURN VALUE:** True/False – True if valid, False otherwise

Details	For a sandwich to be valid, it must have • two pieces of the same type of bread • not be interrupted by unpaired pieces of bread • one piece of bread must be buttered and another un-buttered • buttered bread must be on top Types of bread • 'w' – white bread • 'r' – rye bread • 'p' – pumpernickel Butter state • 'b' next to a bread type (i.e. bw, br, bp) is buttered • 'n' next to a bread type (i.e. nw, nr, np) is **not** buttered
Sample behaviour	**1. Input:** validate_sandwich("bw nw") **Result:** **TRUE** **Reason:** valid -- the classic white **2. Input:** validate_sandwich("bw br nw") **Result:** **FALSE** **Reason:** invalid – the rye bread does not have a pair **3. Input:** validate_sandwich("bw br nr nw") **Result:** **TRUE** **Reason:** valid – a sandwich inside a sandwich. **4. Input:** validate_sandwich("bw np") **Result:** **FALSE** **Reason:** invalid -- the white and pumpernickel do not match
Validation of user input	Required. As part of function, sandwiches are always lower case, and space-separated. (**Note: Try to do as many validation as you can think of, but make sure when teaching staff checks it should not give an error.**)

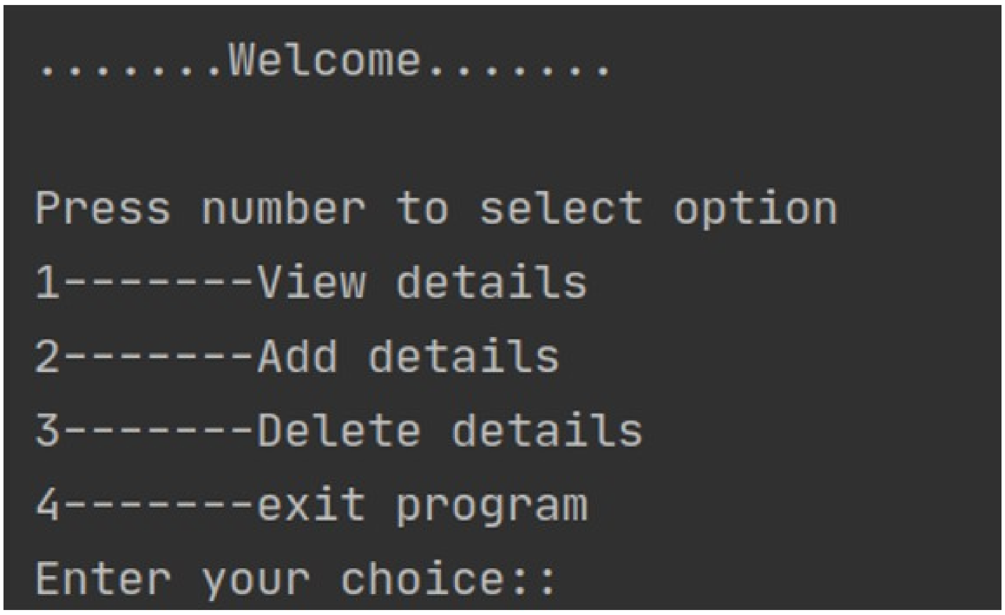
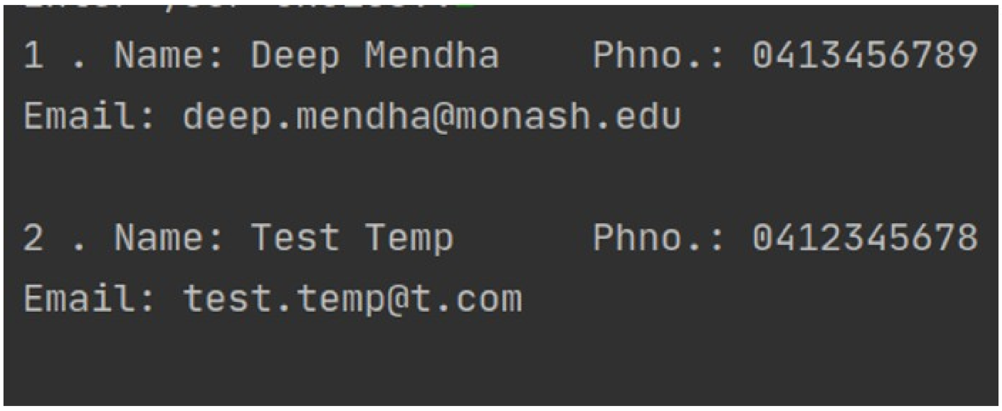
3.5. Contact Book (30 marks)

Write at-least 3 function with signature AddData(), DeleteData(), and ViewData(). All the function should be done on file called "contactBook.txt", which can be created beforehand. All the operation, that are carried out should reflect in the text file. More details are provided below:

Function name	AddData(filename)
	DeleteData(ID, filename)

	ViewData(filename)
Inputs	**USER INPUT:** • **Main menu** o Choice – the option chosen by the user to move through the menu • **AddData:** o **First Name –** the given name of the contact to add o **Last Name** – the family name of the contact to add o **Phone number** – the phone number of the contact to add ▪ No more than 10 digits ▪ Must follow Australian mobile number format (04xxxxxxxx) o **Email address** – the email address of the contact to add ▪ Should follow standard email format. **ARGUMENTS**: • **AddData**: o filename – the file to write to • **DeleteData**: o ID – the unique ID for the contact to delete o filename – the file to update • **ViewData**: o filename – the file to read from
Output	**PRINT:** • **Main menu** o Print the options available to the user at any point • **ViewData** o Print the entire contents of the contacts list (see sample behaviour) **FILE OUTPUT:** • AddData o Writes the new contact details as a line in “contactBook.txt” • DeleteData o Rewrite/update “contactBook.txt” to remove the deleted contact
Details	1. Main menu • Show options for the user to choose from • Validate their choice • Run the option chosen • Continue running the menu unless the user chooses to quit 2. AddData

	• Get the first name, last name, phone number and email address from the user • Generate a unique ID • Write the data to the file 3. DeleteData • Access the file • Find the line for the contact with a matching ID • Remove that contact from the file 4. ViewData • Load up the file • Print each record in the file to the screen

Sample behaviour	**Main menu**
	
	`.......Welcome.......`
	
	`Press number to select option`
	`1-------View details`
	`2-------Add details`
	`3-------Delete details`
	`4-------exit program`
	`Enter your choice::`
	
	ViewData
	
	`1 . Name: Deep Mendha    Phno.: 0413456789`
	`Email: deep.mendha@monash.edu`
	
	`2 . Name: Test Temp      Phno.: 0412345678`
	`Email: test.temp@t.com`

Validation of user input	• The main menu should request input again if given an invalid choice and should only exit when exit is selected • Phone number must meet requirements • Email address need to be verified (**Note: Try to do as many validation as you can think of, but make**

	sure when teaching staff checks it should not give an error.)