Introduction to Programming (Python)

Coursework 2016-17

This document describes your final project work for this module. The overall assessment breakdown for the module is:

- Class Test: 20%
- Final Project: 80%

The grade for the module will be the sum of these two marks.

Project Specification

Your final project work is to build a small application in the Python programming language used in the module and to document the process. The development work for this coursework may be completed in pairs (recommended because it is good to have someone to discuss the problem and solution with, but you may NOT work in groups larger than two) or individually. If you work as a pair this means working on all parts of the program together (the report is an individual report, but the development and testing of the code should be done jointly) – it is not intended that you delegate different parts of the implementation work to partners individually, though it is acceptable for you to identify who will do the lead work for each function/part (as in, who types the code in to the computer once the algorithm/logic has been agreed and who sits and watches over their shoulder!), and you may wish to decide that where one person takes the lead on implementing some function that the other person takes the lead responsibility for testing it. If you are in a pair then each pair will submit a single project and documentation and an **individual critical appraisal** (for each partner) as described in *What To Submit*, below.

Project Description

You are going to design and build the UWS student 'study buddy' application. Your application should take the information supplied by students and make use of it to find a list of possible study buddies.

Your program should be interactive, and have an appropriate user interface. As ***a minimum*** it must provide the following capabilities:

1. Enter, edit and delete the details of students who use the app

2. Provide some level of anonymity and security of personal information

3. Contain an algorithm to match students based on their profiles

4. Have a means of displaying the results of the matching exercise

Data Acquisition

You must provide a method which allows new students to create matching requests on the UWS 'study buddy' service. Each new matching request should be given a randomly assigned request ID which is

in the form of a six digit number. The user should provide a password and a set of data consisting of Name (first and surname), programme, year of study, the name of the module they wish to study, campus location and available days & times (times should take the form of morning, afternoon or evening). You should use different input methods (e.g. text entry, multiple choice…). Generally, it is left to you to decide which input method to use for each data entry field. However, when matching longer strings such as the module name, to eliminate the possibility of problems when matching (due to typing/spelling errors by users), you should present the user with a list of options from which to choose.

Each student's information should be stored in a separate file which uses the user ID number as its name (e.g. 999999.txt or 999999.pck). Each user should be able to edit their own file by accessing it with the combination of user ID and password. All user files should be stored together in the same directory.

Matching Algorithm

Although it is left up to you to decide on the exact algorithm which should be used to provide matches, your algorithm must match on a minimum of module name. It should return the details of all potential matches identified, or report that no matches were found. Possible matches should be ranked according to the number of criteria which match and then displayed. All attributes except student name, user ID and password should be displayed for each match.

Minimum Standard

The specification above outlines the minimum functionality required by your program, however you can add additional functionality, which if done well and properly documented, will attract additional marks. Some examples of additional functionality might include a well-designed user interface which includes a number of Tkinter widgets, encryption of the user password (you can find examples of how to do this online – e.g. https://anaconda.org/pypi/xxhash or https://www.youtube.com/watch?v=vPpRkHUPX_Q) or well thought out matching algorithms. **Remember that any code that you find online must be properly referenced both as a comment in your code and in the accompanying documentation.**

Stepwise Development and Marking Scheme

You are ***advised*** to develop the program one step at a time and keep a copy of working versions. At each stage of the development you should design appropriate tests to ensure that your code works as expected.

Step 1 (functionality is worth 15 marks)

- Create a *Request* class with suitable attributes to hold the request Id, password, name (first and surname), programme, year of study, campus location, module name and a list of available days (Monday …Sunday) and timeslots (morning, afternoon, evening). The user must be able to enter as many unique timeslots as they wish on a single request. Use the *__init* method to accept a request ID passed from the calling object and allow the user to input the values for the other attributes. (using suitable input methods – you may use command line inputs or appropriate TKinter widgets – see section on gaining extra credit)
- Provide a *displayAll* method that displays these details

- Create *stepA.py* that contains functions to randomly generate a request ID and ensure that this ID is unique by maintaining a list of currently used request ID's. These unique ID's should be maintained by the application using a list or dictionary, N.B. the user does not input or select the request ID. It will then instantiate a new request object, passing the request ID and call the *displayAll* method.
- a typical run of the program is: (remember that you choose the input methods this is an example)

```
>>>Request ID = 123456
>>>input name: John Smith
>>>input password: 77889900
>>>input program: BSc Computing Science
>>>input year of study: 2
>>>input campus: Paisley
>>>input module: Intro to Programming
>>>input available day (9 to quit): Monday
>>>input available time: evening
Request ID = 123456, password = 77889900
John Smith, BSc Computer Science, Paisley
Intro to Programming, Monday: Evening
```

- keep a copy of this version of the programme

Step 2 (functionality is worth 5 marks)

- Extend the *Request* class as follows:
- Create a *displayRestricted* method which is similar to the *displayAll* method but does not show the student name request ID or password.
- Create *stepB.py* that contains all the functionality of *stepA.py* and which calls the *displayRestricted* method instead of the *displayAll* method. When run, this program should present the user with a welcome message and a single menu option to start a new request. (The example of the menu given below is for illustration only, you should use an appropriate input method of your choice)
- a typical run of the program is: (remember that you choose the input methods this is an example)

```
>>>Welcome to UWS Study Buddy App
>>>Press any key to make a new request
>>>Request ID = 123456
>>>input name: John Smith
>>>input password: 77889900
>>>input program: BSc Computer Science
>>>input year of study: 2
>>>input campus: Paisley
>>>input module: Intro to Programming
>>>input available day (9 to quit): Monday
>>>input available time: evening
BSc Computer Science, Paisley
Intro to Programming, Monday: Evening
```

- keep a copy of this version of the programme

Step 3 (functionality is worth 15 marks)

- Extend your program as follows:

- Create *makeFile* and *readFile* functions.
- The *makeFile* function should create a file called XXXXXX.pck where XXXXXX is the request ID. This function should write out the full contents of the current instance of the *Request* class to the file using pickle to maintain the data structure.
- The *readFile* function should, when passed a request ID, open the file named XXXXXX.pck for reading and return the data contained within that file. This function should display the contents of the file using the *displayAll* method.
- Make the program *stepC.py* (containing all the functionality of *stepB.py* which creates an instance of the *Request* class and then uses the *makeFile* function to create a persistent copy of the *Request* object. It should then retrieve the data from the file using the *readFile* function. It should also display messages to indicate the file was written and retrieved. In this program the method of checking for unique request ID's should be updated such that the list of current ID's is held in a text file named *Requests.txt.* Functions named *checkId,* and *addId* should be written to check the list and add a new entry respectively. These two functions will need mechanisms to create and update the *Request.txt* file.
- a typical run of the program is: (remember that you choose the input methods this is an example)

```
>>>Welcome to UWS Study Buddy App
>>>Press any key to make a new request
>>>Request ID = 123456
>>>input name: John Smith
>>>input password: 77889900
>>>input program: BSc Computer Science
>>>input year of study: 2
>>>input campus: Paisley
>>>input module: Intro to Programming
>>>input available day (9 to quit): Monday
>>>input available time: evening
File 123456.pck written
File 123456.pck retrieved
Request ID = 123456, password = 77889900
John Smith, BSc Computer Science, Paisley
Intro to Programming, Monday: Evening
```

- keep a copy of this version of the programme

Step 4 (functionality is worth 15 marks)

- Make the program *stepD.py* (including the functionality of *stepC.py*) which creates an instance of the *Request* class and then uses the *makeFile* method to create a persistent copy of the *Request* object. This object is the **current object** in your program.
- Add a function which loops through the file containing the list of currently used request ID's and for each request (excluding the **current object**) firstly retrieves the .pck file related to that request ID and then compares the data in that file with that of the **current object**.
- The data matching algorithm should compare each attribute (except name, request ID and password) of the data held in the file with that held in the same attribute of the **current object**. If the module name attribute matches it should count of the total number of attributes which match and add it to a list of matched requests which contains request ID and count (number of matching attributes) for each matching request.
- A *displayMatches* function should display the number of matching requests found or a message stating no matches were found. For each match found the *displayRestricted* method of the *Request* class should be used to display details of the match. Where more

than one match is found, all matches should be displayed with the highest number of matches displayed first.

- a typical run of the program is: (remember that you choose the input methods this is an example)

```
>>>Welcome to UWS Study Buddy App
>>>Press any key to make a new request
>>>Request = 123456
>>>input name: John Smith
>>>input password: 77889900
>>>input program: BSc Computer Science
>>>input year of study: 2
>>>input campus: Paisley
>>>input module: Intro to Programming
>>>input available day (9 to quit): Monday
>>>input available time: evening
1 Match found
BSc Networking, Paisley
Intro to Programming, Wednesday: Evening
```

Note: In this example one module and location were matched!

- keep a copy of this version of the programme

Step 5 (functionality is worth 10 marks)

- Make the program *stepE.py* (including the functionality of *stepD.py*) which creates an instance of the *Request* class and then uses the *makeFile* method to create a persistent copy of the *Request* object. This object is the **current object** in your program.
- Add an *editFile* method which, when passed a request ID, firstly calls the *readFile* method to retrieve the file contents and then prompts the user for the file password. If the user entered password is correct the file contents are displayed and the user is permitted to edit the contents of the file.
- Create a function that displays a menu with choices to make a new request or edit an existing request,
- Add functionality which automatically searches for matches whenever a new request is entered into the system (or an existing request is edited) and displays any matches found or an appropriate message if no match is found.
- a typical run of the program is: (remember that you choose the input methods this is an example)

```
>>>Welcome to UWS Study Buddy App
>>>Press 1 to make a new request
>>>Press 2 to edit an existing request
>>>Press 3 to exit UWS Study Buddy
>>>1
>>>input name: John Smith
>>>input password: 77889900
>>>input program: BSc Computer Science
>>>input year of study: 2
>>>input campus: Paisley
>>>input module: Intro to Programming
>>>input available day (9 to quit): Monday
>>>input available time: evening
1 Match found
BSc Networking, Paisley
Intro to Programming, Wednesday: Evening
>>>Welcome to UWS Study Buddy App
```

>>>Press 1 to make a new request
>>>Press 2 to edit an existing request
>>>Press 3 to exit UWS Study Buddy
>>>2
>>>Enter Request ID you wish to edit: 123456
>>>Enter Password: 123456
>>>Incorrect Password Try Again: 77889900
>>>Data Held for 123456 is

```
Request ID = 123456, password = 77889900
John Smith, BSc Computer Science, Paisley
Intro to Programming, Monday: Evening
>>>update name: John Smith
>>>update password: 77889900
>>>update program: BSc Computer Science
>>>update year of study: 2
>>>update campus: Ayr
>>>update module: Intro to Programming
>>>update available day (9 to quit): Monday
>>>update available time: evening
Sorry there are currently no matches for your request
```

>>>Welcome to UWS Study Buddy App
>>>Press 1 to make a new request
>>>Press 2 to edit an existing request
>>>Press 3 to exit UWS Study Buddy
>>>3
>>>Thank you for using UWS Study Buddy, Goodbye

- Keep a copy of this program (the final iteration of your development)

Programming style and organisation – 10 marks Awarded for clear, well organised and commented code with all sources referenced using comments.

Extra Credit – 10 marks Awarded for using more complex, and varied input methods (e.g. a range of TKinter widgets), providing a well-designed user interface, evidence of self-directed study of Python such as introducing features/packages/modules not covered in the lectures.

Testing plan and evidence – 10 marks Awarded for providing an effective, well documented testing plan for each step in the development supported by evidence of testing having taken place (outputs such as screenshots)

Individual critical appraisal – 10 marks

What to Submit

You will submit all coursework to the appropriate assignments page on Moodle. The project code files (. Py) and associated data files (you must include ***at least three*** request data files) should be submitted along with the project documentation (a Word document) in a single ZIP file. To do this:

- Create the project in your Python environment of choice and fully test it
- Create the documentation (see below) as a Word document or PDF. Remember to add the Banner IDs of the two participants to this document.
- Move or copy the document into a folder and copy your project source code files into the same folder. Include any data files your project may depend on

- ZIP the folder in Windows Explorer by right clicking on it and selecting Send to…→Compressed File
- Go to the Moodle Assignments and click on Final Project Submission
- Attach your ZIP file using the Add Submission button

The deadline for the project submission is **Friday 14th April 2017 at 23:59**. Projects submitted later than this will be subject to the normal university late submission rules (up to 1 week late, a reduction of the mark by 10 percentage points; later than this, a submission will receive a zero mark).

Project Documentation

The documentation for your project should be in Word or PDF format. This must contain the following sections:

- The Banner IDs of both project participants (do not include your names anywhere in the submission)
- A description of each source code file in the project, including a brief description of the purpose of any classes, methods and functions you may have used.
- A comprehensive testing plan and copies of the inputs and outputs (screenshots) of tests at each major step in the development
- A ***critical appraisal*** of your work on the project –each team member must provide his or her own appraisal. This should be an explanation of the success or otherwise of the project, and so should include things like:
    - Whether the program you built meets the project specification, including
        - Parts of the project that were adequately completed
        - Parts of the project that were not done well
        - Which parts of the code you wrote/contributed towards
    - How each member of the team performed in the project (Banner IDs, no names please)

The critical appraisal allows you to reflect on the work you have done and what you have learned from it and can be a useful input into your Personal Development Plan. The critical appraisal is as important as getting the project to work, since it gives you an opportunity to explain that you understand the strengths and weaknesses of the work you have completed. Bear in mind that markers will almost certainly notice any flaws in your project work, so it is better to indicate that you are aware of them.