Coursework Assessment Brief

Academic Year 2017-18

IMPORTANT STATEMENTS

Standard Undergraduate Assessment Regulations

Your studies will be governed by version 5 of the Standard Undergraduate Assessment Regulations (SUAR 5).

Under these regulations you are permitted two attempts at assessment for each module: a first sit and re-assessment attempt.

This means that you will be required to withdraw from the course if, following the reassessment attempt, you have not passed.

Cheating and Plagiarism

Both cheating and plagiarism are totally unacceptable and the University maintains a strict policy against them. It is YOUR responsibility to be aware of this policy and to act accordingly. Please refer to the Academic Registry Guidance at https://icity.bcu.ac.uk/Academic-Registry/Information-for-Students/Assessment/Avoiding-Allegations-of-Cheating

The basic principles are:

- Don't pass off anyone else's work as your own, including work from "essay banks". This is plagiarism and is viewed extremely seriously by the University.
- Don't submit a piece of work in whole or in part that has already been submitted for assessment elsewhere. This is called duplication and, like plagiarism, is viewed extremely seriously by the University.
- Always acknowledge all of the sources that you have used in your coursework assignment or project.
- If you are using the exact words of another person, always put them in quotation marks.
- Check that you know whether the coursework is to be produced individually or whether you can work with others.
- If you are doing group work, be sure about what you are supposed to do on your own.
- Never make up or falsify data to prove your point.
- Never allow others to copy your work.
- Never lend disks, memory sticks or copies of your coursework to any other student in the University; this may lead you being accused of collusion.

By submitting coursework, either physically or electronically, you are confirming that it is your own work (or, in the case of a group submission, that it is the result of joint work undertaken by members of the group that you represent) and that you have read and understand the University's guidance on plagiarism and cheating.

Students should be aware that, at the discretion of the module co-ordinator, coursework may be submitted to an electronic detection system in order to help ascertain if any plagiarised material is present.

Electronic Submission of Work

Students should also be aware that it is their responsibility to ensure that work submitted in electronic format can be opened on a faculty computer and to check that any electronic submissions have been successfully uploaded. If it cannot be opened it will not be marked. Any required file formats will be specified in the assignment brief and failure to comply with these submission requirements will result in work not being marked.

Students must retain a copy of all electronic work they have submitted and resubmit if requested.

Deliverable:	Design documentation, software development and testing and evaluation document (Weighting: 100%)
Learning Outcomes to be Assessed: 1. Design solutions to programming problems 2. Implement those solutions in an imperative programming language by using common programming tools (such as editors and interpreters). 3. Use common programming tools and techniques (e.g. IDE, testing APIs and theories) to test and evaluate programs	
Assessment Details: See below	
Assessment Criteria: See table below	

Submission Details:

All work should be submitted on Moodle as a single zip file. The zip file should contain the following:

- Software implementation checklist (Compulsory).
- Design documentation: 500 words paper inclusive of diagrams submitted online
- All of your source code and associated project files.
- 1000-word testing and evaluation documentation.

The zip file should be structured so that the location of its contents is clear and unambiguous. Please submit only one version of your work. Also, the zip file should be named using the following format

StudetName_studentID.zip

For example: **ShadiBasurra_ID123456789.zip**

PLEASE MAKE SURE THAT YOUR APPLICATION WILL WORK ON THE UNIVERSITY'S MACHINES. DO NOT RELY ON THE FACT THAT IT MAY WORK ON YOUR OWN LAPTOP/PC AND THEREFORE IT SHOULD WORK ELSEWHERE.

If you submit an assessment late at the first attempt then you will be subject to one of the following penalties:

- *If the submission is made up to two hours following the deadline, the mark that you achieve will be deducted by 10%*. If this deduction takes you below the pass threshold then you will be capped at that threshold, i.e. 40%;*
  (*For example: work awarded a mark of 60% would become 54%)
- *If the submission is made between two hours and up to 5 days following the deadline, your mark will be capped at the pass threshold;*
- *If the submission is made after 5 days following the deadline, your work will be deemed as a fail and returned to you unmarked.*

If you submit a reassessment late then it will be deemed as a fail and returned to you unmarked.

Workload:

It is expected that the students should take around 60/75 hours (in addition to the regular weekly teaching hours) of programming effort to design and develop an application as specified in this assessment brief.

Feedback:

Feedback on your submissions should be given electronically via Moodle.
Marks and Feedback on your work will normally be provided within 20 working days of the submission deadline.

Assessment Summary

This assessment comprises a single deliverable comprising three items as shown below. All the items are due in Week 11.

a) Design document: 500 words paper inclusive of diagrams submitted online (weight 20%).

b) Computer programme solution programme code submitted online (weight 50%).

c) 1000-word testing and evaluation article inclusive of test case design and diagrams submitted online. Further details for each deliverable are below (weight 30%).

Rationale

This programming assignment is to apply the programming principles covered in tutorials and lectures to develop a python software that implements core banking solution that is used in banks by its customers and staff members. The software can be fully implemented using text based interface, however, to achieve higher marks, a GUI based software can be developed using the Python GUI Programming module namely Tkinter. GUI design and Tkinter programming will be covered during weeks 7 and 8. The aim of the exercises is to enhance a student's experience of programming by applying programming principles to a larger problem of developing a complete application.

There are mainly two reasons behind the selection of the banking system as the topic of this coursework. Firstly the students are familiar with the banking systems, hence, students will spend less time and effort to understand the functions specification of the software they will be developing for this coursework. Hence, most of their time will be devoted to the design, development and testing of the banking system by applying the programming knowledge and skills they learnt throughout this module. Secondly, GUI based programs make it easier to interact with the developed system and demonstrate a direct relationship between the user interaction and the system functions and data. Also, students will learn about how a product works entirely from the user's (or customer's) perspective and not from just a developer prospective. Hence, they will need to develop a user friendly GUI.

Overview of banking systems

A banking system is a group or a network of institutions that provide financial services for citizens. These institutions are responsible for operating a payment system, providing loans, taking deposits, performing money transfer and helping with investments. In this coursework, students will create a 'prototype' Banking system to focus on basic operations such as deposit money, withdraw money, transfer money and agree to make loans to their customers. For staff members i.e. admins, they can create new bank accounts to customers, search for customers' accounts, update/view admin/customer personal details and close customers bank account.

Objective - Sample Prototype Application

The objective of this assessment is to develop a python software that implements a core banking solution that is used in banks by its customers and bank admins. A partial implementation of a prototype banking system will be developed as part of the practical sessions in the PC labs. This prototype will be developed to include basic banking system functionalities such as customer/admin login, deposit money and display balance. The main class will allow to add a number of customers/admins at the start of the software. At the initial stage, all customer/admin data will be hard coded. However, for students to achieve higher marks, they should enhance the system functionalities. For example, to use text files as its backend storage to store customer/admin related information.

How you will be marked:

To achieve a mark of 40% to maximum of 50%

The application **must** implement **all** the following:

- Create the necessary classes and functions which allow Customers to perform the following tasks
  - Login
  - Deposit money
  - Withdraw money
  - Check bank balance
  - View customer details e.g. name and address
  - Update customer information e.g. name and address

- Create the necessary classes and functions which allow admins to perform the following tasks:
  - Login
  - Search for a particular customer to perform various banking operations on his/her account i.e. check balance, deposit/withdraw money etc.
  - Close account i.e. remove customer from the system
  - Update customer information – name and address
  - Update admin own information i.e. name and address
  - Print a customer details
  - Print all customers details

To achieve a mark of 51% to maximum of 60%

The application **must** implement **all** the above and the following:

- Development of a suitable GUI to perform all the above banking functions

To achieve a mark of 61% to maximum of 70%

The application **must** implement **all** the above and the following:

- Customers can have different types of bank accounts. Accounts will differ in their name, interest rate and overdraft limit etc.
- The bank system should be able to store and load customer and admin data from and into a file.
- Save the status of the system to the backend storage (e.g. file storage) at any time. Also when the system is closed it should update its backend storage, such that the system can load the up to date status at the next start.
- Customers should be able to request a loan, if the loan is within a the bank loan limit e.g. £10,000, then the system will grant the loan with 30% probability. If the bank refuses to grant the load, a message should be given to the customer to notify them about the decision. However, when the loan is accepted, the requested amount will be deposited to the customer's account. Also, a loan payment due data and time should be generated e.g. 21 days from now. Customers should be able to clear their loan at any time, however if the repayment due date has passed, a daily fee should be added to the account. Hint, Random and time functions can be used to implement this feature.
- Development of a suitable GUI to perform all the above banking functions.

To achieve a mark of 71% and over

The application **must** implement **all** the above and the following:

- Admin should be able to view all customers' loan report. A loan report should include a list with all customer names, loan amount and loan due data. Also, Admins should be able to view the report based various orders e.g. loan amount and repayment due date and time.
- Transferring funds/money from one account to another. For example, one customer wants to send money to another customer.
- 30% of the loan requests may require further approval by admins. Hence, 60% from all loan request refusals should be further processed by admins for approval. Admins at this stage have all the right to accept / reject a loan request.
- Development of a suitable GUI to perform all the above banking functions.