Assessment (non-exam) Brief

UCL
SCHOOL OF
MANAGEMENT

Module code/name	INST0004 Programming 2
Module leader name	Dr Daniel Onah
Academic year	2023/24
Term	1
Assessment title	Tutorial Sheets 1, 2,3,4,6,8
Individual/group assessment	Individual

Submission deadlines: Students should submit all work by the published deadline date and time. Students experiencing sudden or unexpected events beyond your control which impact your ability to complete assessed work by the set deadlines may request mitigation via the extenuating circumstances procedure. Students with disabilities or ongoing, long-term conditions should explore a Summary of Reasonable Adjustments.

Return and status of marked assessments: Students should expect to receive feedback within one calendar month of the submission deadline, as per UCL guidelines. The module team will update you if there are delays through unforeseen circumstances (e.g. ill health). All results when first published are provisional until confirmed by the Examination Board.

Academic Misconduct: Academic Misconduct is defined as any action or attempted action that may result in a student obtaining an unfair academic advantage. **Academic misconduct includes plagiarism, obtaining help from/sharing work with others be they individuals and/or organisations or any other form of cheating.** Refer to Academic Manual Chapter 6, Section 9: Student Academic Misconduct Procedure - 9.2 Definitions.

Referencing: You must reference and provide full citation for ALL sources used, including articles, textbooks, lecture slides and module materials. This includes any direct quotes and paraphrased text. If in doubt, reference it. If you need further guidance on referencing please see UCL's referencing tutorial for students. Failure to cite references correctly may result in your work being referred to the Academic Misconduct Panel.

Use of Artificial Intelligence (AI) Tools in your Assessment: Your module leader will explain to you if and how AI tools can be used to support your assessment. In some assessments, the use of generative AI is **not permitted** at all. In others, AI may be used in an **assistive** role which means students are permitted to use AI tools to support the development of specific skills required for the assessment as specified by the module leader. In others, the use of AI tools may be an **integral** component of the assessment; in these cases the assessment will provide an opportunity to demonstrate effective and responsible use of AI. See page 3 of this brief to check which category use of AI falls into for this assessment. Students should refer to the UCL guidance on acknowledging use of AI and referencing AI. Failure to correctly reference use of AI in assessments may result in students being reported via the Academic Misconduct procedure. Refer to the section of the UCL Assessment success guide on Engaging with AI in your education and assessment.

:

Content of this assessment brief

Section	Content
A	**Core information**
B	**Coursework brief and requirements**
C	**Module learning outcomes covered in this assessment**
D	**Groupwork instructions (if applicable)**
E	**How your work is assessed**
F	**Additional information**

Section A Core information

Submission date	On the module page
Submission time	On the module page
Assessment is marked out of:	100
% weighting of this assessment within total module mark	100%
Maximum word count/page length/duration	NA
Footnotes, appendices, tables, figures, diagrams, charts included in/excluded from word count/page length?	NA
Bibliographies, reference lists included in/excluded from word count/page length?	NA
Penalty for exceeding word count/page length	No specified maximum (and thus no penalty for exceeding)
Penalty for late submission	Standard UCL penalties apply. Students should refer to Refer to https://www.ucl.ac.uk/academic-manual/chapters/chapter-4assessment-framework-taught-programmes/section-3-moduleassessment#3.12
Submitting your assessment	Weekly at the scheduled deadline/time

:

Anonymity of identity. Normally, <u>all</u> submissions are anonymous unless the nature of the submission is such that anonymity is not appropriate, illustratively as in presentations or where minutes of group meetings are required as part of a group work submission	The nature of this assessment is such that anonymity is not required.

Section B Assessment Brief and Requirements

On the Tutorial sheets.

Section C: Module Learning Outcomes covered in this Assessment

This assessment contributes towards the achievement of the following stated module Learning Outcomes as highlighted below:
Weekly lectures for INST0004 Programming 2

Section D : Groupwork Instructions (where relevant/appropriate)

NA

Section E

: How your work is assessed

Within each section of this assessment you may be assessed on the following aspects, as applicable and appropriate to this assessment, and should thus consider these aspects when fulfilling the requirements of each section:

- The accuracy of any calculations required.
- The strengths and quality of your overall analysis and evaluation;
- Appropriate use of relevant theoretical models, concepts and frameworks;
- The rationale and evidence that you provide in support of your arguments;
- The credibility and viability of the evidenced conclusions/recommendations/plans of action you put forward;
- Structure and coherence of your considerations and reports;
- Appropriate and relevant use of, as and where relevant and appropriate, real world examples, academic materials and referenced sources. Any references should use either the Harvard OR Vancouver referencing system (see References, Citations and Avoiding Plagiarism)
- Academic judgement regarding the blend of scope, thrust and communication of ideas, contentions, evidence, knowledge, arguments, conclusions.
- Each assessment requirement(s) has allocated marks/weightings.

Student submissions are reviewed/scrutinised by an internal assessor and are available to an External Examiner for further review/scrutiny before consideration by the relevant Examination Board.
It is not uncommon for some students to feel that their submissions deserve higher marks (irrespective of whether they actually deserve higher marks). To help you assess the relative strengths and weaknesses of your submission please refer to UCL Assessment Criteria Guidelines, located at https://www.ucl.ac.uk/teaching-learning/sites/teaching-learning/files/migratedfiles/UCL_Assessment_Criteria_Guide.pdf

The above is an important link as it specifies the criteria for attaining 85% +, 70% to 84%, 60% to 69%, 50% to 59%, 40% to 49%, below 40%.

You are strongly advised to **not** compare your mark with marks of other submissions from your student colleagues. Each submission has its own range of characteristics which differ from others in terms of breadth, scope, depth, insights, and subtleties and nuances. On the surface one submission may appear to be similar to another but invariably, digging beneath the surface reveals a range of differing characteristics.

Students who wish to request a review of a decision made by the Board of Examiners should refer to the UCL Academic Appeals Procedure, taking note of the acceptable grounds for such appeals.

Section F

Note that the purpose of this procedure is not to dispute academic judgement – it is to ensure correct application of UCL's regulations and procedures. The appeals process is evidence-based and circumstances must be supported by independent evidence.

Section G

: Additional information from module leader (as appropriate)

Work in accordance with the weekly assessment brief and the tutorial sheet questions herein.

Section H

INST0004: Programming 2
Tutorial Sheet 01
Week 01

Academic Year: 2023/24

Set by: Daniel Onah

The aim of the practical sheet is to provide you with a set of exercises to practice developing your programming skills. Writing programmes will help you develop programming skills and prepare for the Programming Test and the Moodle Quiz.

Assessed Tutorial Questions:
Some questions in the labs are marked with a [*] symbol. These questions are compulsory and you will be assessed on one or more of these in the following week's lab. For most of you, this means you will be assessed on:

- Tutorial sheet 1 in Lab 2
- Tutorial sheet 2 in Lab 3
- Tutorial sheet 3 in Lab 4
- Tutorial sheet 4 in Lab 5
- Tutorial sheet 6 in Lab 7, and
- Tutorial sheet 8 in Lab 9

Please look at all the questions spend some time thinking carefully about them, before asking for help. If you are still stuck:

- attend the live practical & booster sessions and ask me, or the TAs, for help
- or post a question on the Moodle discussion board.

Once you have completed the assessed questions you should upload the specified files to the Assessment tab on Moodle under the appropriate submission link. You can submit at the specified submission time on Moodle. In preparation for your 1-to-1 viva, you should be able to clearly indicate which constructs such as classes, functions, variables used in your program is of your own work and which classes or features where from other sources (e.g., from textbooks, online sources etc.).

Please refer to Read about scheduling and live sessions and Read about tutorial based assessment, submissions and vivas for additional information on the procedures of being assessed.

A reminder on how to compile a programme

This is just a quick reminder of how to compile a simple Python program.

1) Make sure you organise your files into folders for each of the weeks:

Section I

   a. You should keep your files on the `N:` drive. I suggest creating a folder called `INST0004` with subfolders `week01, week02` etc. for each of the tutorial files.

2) You should open the Command Prompt Shell and navigate to the current directory where your Python files are located, for example, `N:/INST0004/week01`. To do this, use the cd command. For example:

```
cd  N:/INST0004/week01
```

   To change the drive from C: to N: simply type in `N:` in the command prompt shell.
   Speak to a TA or a peer student if you need help with using the N drive or the Command Prompt shell.

3) Once in the working directory, you can execute/run the files without providing the path to the file:

```
python hello_world.py
```

   (where python is the name of the interpreter that will trigger the Runtime Environment and execute specific file)

Section J

Exercise 1 – **Maximum Number Program** [*]

Learning Objectives: [A] return statement; [B] function; [C] Primitive types and relational operations [D] conditional statements – if, elif, else.

Write a program to request a user to enter three numbers and display the maximum number. The program should be designed using two functions: the main() function and the findMaximum() function. The number input should be taken from the *main()* function and the *findMaximum()* should be invoked within the main() function with the appropriate arguments. The *findMaximum()* function should take two parameter lists. Within the body of the *findMaximum()* function, the program should compute whether the first number is greater than the second number, if this is the case, return the first number otherwise return the second number.

Note: Remember we are comparing three random input numbers to identify the maximum number. *Use Lecture 1 resources as a guide to complete the task.*

The output of the program may look as follows:

```
danny$ python3 max_number.py
Enter first number: 36
Enter second number: 98
Enter third number: 59
The maximum number is:  98
```

Exercise 2 – **Number Guesser Program** [*]

Learning Objectives: [A] use a for & while loop; [B] use of Python random; [C] Primitive types and relational operations [D] conditional statements – if, elif, else, break or quit() function.

Write a program which randomly picks a number between 1 and 100 (both inclusive) and repeatedly asks the user to enter a guess ("Guess:"). You should try to guess that number in as few guesses as possible. Depending on the number entered, do one of the following things:

The program should represent a simple game logic where users are allowed 5 guesses to start with and asked to guess a number that was randomly generated using a `random library (import random)`.

Description
Write a program to use a loop that would allow a player only **5 guesses** of a randomly generated machine number. Note the loop that you are asked to create should allow the player only 5 guesses.

Section K

Begin the program to start the game by offering the user an initial *5 guesses*. The allocated **guesses** should be reduced by 1 after each unsuccessful attempt. This means each wrong guess will reduce the allowed guess by 1. If the player correctly guesses the number on or before the 5 attempts, the program should display *'Correct! Congratulations. You use _ guess(es). Well done!* (replace the **'_'** with the number of guesses entered in order to arrive at the correct guess). Otherwise, if the player's allowed attempt reaches 5, the program should terminate the loop and display *'You have exceeded your allowed number of guesses!. The correct number was _'* (replace the **'_'** with the random number).

<u>Program display:</u>

- If the player's guess is higher than the random number generated, the program should display '*Too Hight. Try Again!.'* and reduce the allowed guesses by 1. This should display the current available guesses left while the program is still executing.

- If the player's guess is lower than the random number generated, the program should display *'Too Low. Try Again!.'* and reduce the points by 1. This should also display the current guesses left while the program is still executing.

You are advised to study the given code and understand it fully before attempting the exercise. The sample output provided would guide you to complete the task.

<u>Hint:</u> *You should use a* ***main()*** *function to write the program. Use Lecture 1 resources as a guide to complete the task.*

The output of the program may look as follows:

```
danny$ python3 number_guesser.py
The number is between 1 and 100.
You are allowed 5 guesses.
Enter your guess: 12
>> Too Low. Try Again! <<
You have 4 guess(es) remaining!
Enter your guess: 17
>> Too High. Try Again! <<
You have 3 guess(es) remaining!
Enter your guess: 15
Correct! Congratulations. You use 3 guess(es). Well done!

danny$ python3 number_guesser.py
The number is between 1 and 100.
You are allowed 5 guesses.
```

```
Enter your guess: 45
>> Too Low. Try Again! <<
You have 4 guess(es) remaining!
Enter your guess: 89
>> Too High. Try Again! <<
You have 3 guess(es) remaining!
Enter your guess: 74
>> Too High. Try Again! <<
You have 2 guess(es) remaining!
Enter your guess: 55
>> Too Low. Try Again! <<
You have 1 guess(es) remaining!
Enter your guess: 71
>> Too Low. Try Again! <<
You have 0 guess(es) remaining!
You have exceeded your allowed number of guesses! The correct number was:  73
```

Exercise 3 - Computing Balance of an Account [*]

Learning Objectives: [A] print function; [B] main function; [C] Primitive types and relational operations [D] while loop [E] arithmetic operations.

A customer account balance is $20,000, and this earns 2% interest per year. Compute how many years would this take for the account balance to be double the original amount. First try to solve this by hand. The program should initialise the ***current balance***, ***interest rate*** and the ***expected balance*** at the end of the execution. You should use a while loop constructs to continuously compute the balance with the interest until the balance reaches the expected balance. Note that the balance could be exactly the same as expected when the bank account is doubled or slightly over the expected new balance.

Ideally, you program should keep track of all the number of years it takes to achieve the target balance within the while loop construct block. This is done be incrementing the initial year by 1. When the loop ends, the program should print the outcome as shown in the expected output result section.

Hint: When you execute your program, this should display the number of years it takes to double the initial account balance of $20,000 at an annual interest rate of 2%.

Note: The expected target balance figure/value might be determined by the interest rate and yearly current balance.

The output of the program may look as follows:

Section M

```
danny$ python3  account_double_ex3.py

Current Account Balance: $20000
It would take about 36 years for the current account balance to double to the sum of: $40000 or
slightly over.
The expected balance after 36 years would be: $40797.75
```

Exercise 4 - Number of Days in a Month [*]
Learning Objectives: [A] print function; [B] main function; [C] Primitive types and equality operators [D] while loop [E] arithmetic operations.

Write a simple Java program to determine how many days there are in a given month.

Remember:

- Thirty days hath September,
- April, June and November,
- All the rest have thirty-one,
- Except February, with twenty-eight days clear and 29 days in a leap year
    - Create a new Python program with an appropriate variable name and main() function
    - Declare variable called ***month*** in your main function to represent the month (e.g. 1 -> Jan, 2 -> Feb, etc).
    - Write your Python code (as concisely as you can) that prints out the number of days in that month. Use a few lines of code as you can if possible (Hint: don't forget to use the ***or*** operator in your selection of the months!).
    - Test your program with all the different possible values of month.

The program should request user's input for the month and year. Remember to take in consideration the month of February which at some years might have 28 or 29 days.

Hint: The program should continuously request for user input and should stop when the user enters 0 as input value in either the month or year.

The output of the program may look as follows:

Section N

```
danny$ python3  days_in_month_ex4.py

Enter month and year (0 to exit)
Please enter month: 3
Please enter the year: 2000
The number of days in the 3 month of the year 2000  is:  31 days

Enter month and year (0 to exit)
Please enter month: 2
Please enter the year: 2023
The number of days in the 2 month of the year 2023  is:  28 days

Enter month and year (0 to exit)
Please enter month: 2
Please enter the year: 2004
The number of days in the 2 month of the year 2004  is:  29 days

Enter month and year (0 to exit)
Please enter month: 0
Please enter the year: 2023
You've exit the program.
Bye and see you again soon!
```

Exercise 5 – **Number of Years in Months** [*]

Learning Objectives: [A] print(); [B] main() function; [C] Primitive types and relational operations [D] conditional statements – if, elif, else.

Write a program to calculate the number of years in months. The program should allow users to enter the number of months, the input entry should be more than or equivalent to 12 months. The program should calculate the number of years in the months entered by the user. If the user entered months less than 12, the program should alert the user of the accepted number of months to enter (i.e. 12 months and above). At the end of the program, you should print the number of months entered, the equivalent year(s) and month(s).

Pseudocode:

```
1. Declare a variable for months
2. Initialization of the months (i.e. 12)
3. Prompt user to enter number of months
```

4. Store the entry in a variable called **numberOfMonths**. You should typecast the input to integer.
5. Use any conditional statement to prevent entry less than 12 months
6. Use another condition to calculate the number of years and remaining months
7. Display the number of months entered, the equivalent year(s) and month(s)

Hint: Use the ***main()*** function to write the program.

The output of the program may look as follows:

```
danny$ python3  years_in_months_ex5.py

Enter the number of Months: 45
45  months is equivalent to  3 year(s) and 9 month(s)


danny$ python3  years_in_months_ex5.py

Enter the number of Months: 9
Please enter months more than or equal to 12
```