Assessment (non-exam) Brief

UCL
SCHOOL OF MANAGEMENT

Module code/name	INST0002 Programming 1/INST0091 Introduction to Programming
Module leader name	Dr Daniel Onah
Academic year	2023/24
Term	2
Assessment title	Tutorial Sheets 1, 2,3,4,5,6
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 INST0002 Programming 1 /INST0091 Introduction to Programming

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

INST0002: Programming 1
Tutorial Sheet 04
Week 04

Academic Year: 2024/25

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 5 in Lab 7, and
- Tutorial sheet 6 in Lab 8

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:

Published September 3rd 2022

Section I

a. You should keep your files on the `N:` drive. I suggest creating a folder called `INST0002` 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:/INST0002/week01`. To do this, use the cd command. For example:

```
cd  N:/INST0002/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[*]

Learning Objectives: [A] declaring functions without parameters; [B] calling functions;

Write a programme that will ask the user if they want to have a motivational quote displayed through a [*y/n*] prompt for *yes* and *no* input. You should declare a function called **`printQuote().`** The function should take no arguments and return nothing. If the user types in ***y*** *or* ***yes***, your programme should call the function **`printQuote()`** that will then print a quote of your choice.

In addition to **`printQuote()`** you should also declare another similar function called **`printComeBackAgain()`** that does not take any arguments and returns nothing. The function should simply print *"Come back again!"*. This function should be called when the user types in ***n*** *or* ***no*** *as input*.

Hint: *To take textual input from the user such as 'yes or y' use the* `input()` *function. Use the* `substring()` *function of the String class to get the first character of the string (e.g. if user entered 'yes'). Read about slice or substring in the* ***Snakify*** *here:* https://snakify.org/en/lessons/strings_str/ *and study the concept of slices in Python.*

The output of the programme may look as follows:

```
danny$ python3 quote_print.py
Would you like a quote? [y/n]: y
"Learning never exhausts the mind."
          - Leonardo da Vinci

danny$ python3 quote_print.py
Would you like a quote? [y/n]: yes
"Learning never exhausts the mind."
          - Leonardo da Vinci

danny$ python3 quote_print.py
Would you like a quote? [y/n]: n
Come back again!

danny$ python3 quote_print.py
Would you like a quote? [y/n]: no
Come back again!
```

Section K

Exercise 2 []

Learning Objectives: [A] declaring functions with parameters; [B] calling functions;

You are familiar with the Math module and the functions it offers, for example `add(…)`, `max(…)`, and `pow(…) etc`.

Imagine that you were required to implement function **`add(…)`** just as one of the programmers who developed it as part of the Math module, which is part of the Python package library.

This task required you to create a function that would take two input number and compute the sum. You should declare a function that has two parameters of type *int*, and returns the *sum* of the arguments passed to the function.

To test your programme, use an input() function to get two **int** values from the user and pass these to the *add()* function. Print the value returned by the function you declared.

Hint: You could define the ***input()*** function within a ***main()*** function and invoke the add() function within the ***main()*** function..

The output of the programme may look as follows:

```
danny$ python3 custom_sum.py
Please enter Number 1: 5
Please enter Number 2: 6
The sum is: 11
```

Exercise [3*]

Learning Objectives: [A] declaring functions with parameters; [B] calling functions;

We have now learnt that we can use the *cast (typecasting) operator* for converting from one type of data to another. Note, that every **`char`** variable can be converted into an `int` representation of the character and vice versa. For example, the following code will result in an output shown immediately after the code.

```
x = ord('a')
print(x)  # This will also work – print(str(x))
```

```
print(chr(x))
```

```
97
a
```

Read more about this function **ord()** Python built-in function here:
https://docs.python.org/3/library/functions.html#ord

Given the above, write a programme that will ask the user to enter three values of ***type char***. Your programme should implement a method that will take the three **char** values as parameters, cast and find the character with the smallest integer value, and return the value of the smallest one as a char after casting it from `integer` back to `char`. If the user input contains the same character there is a possibility of having integer values that are equal. You can refer to either of these chars when they are the smallest.

Read more about ASCII code and character representation here: https://www.ascii-code.com/

Your programme should print the output of the function call you implemented.

The output of the programme may look as follows:

```
danny$ python min_char_value.py
Please enter Character 1: a
Please enter Character 2: M
Please enter Character 3: ƕ
The integer value of 'a' is: 97
The integer value of 'M' is: 77
The integer value of 'ƕ' is: 1341
The character with the smallest integer value is: M
```

Exercise 4[*]

Learning Objectives: [A] declaring functions with parameters; [B] calling functions; [C] arithmetic operations.

Modify the programme you developed in Exercise 1 from Tutorial Sheet 2.

Your programme should now perform the operation as specified in exercise 1 of tutorial sheet2. The calculation for this task should be done using a specific function. Ensure that the function is

declared using all parameters or relevant types. As a reminder, the formula for calculating the number of calories burnt walking should be:

$$tcb = t * ((0.035 * bw) + (s^2 / h * 0.029 * bw))$$

Extend the programme by adding another function that will calculate the number of calories burnt through climbing stairs. The formula for calculating the number of calories burnt climbing stairs should be:

```
Calories burned = 6.2 x Weight (kg) x Time (hours)
```

The output of the programme may look as follows:

```
danny$ python3 calories_calculator.py
Enter bodyweight (kg): 65
Enter height (m): 1.69
Enter average walking speed (m/s): 1.5
Enter time walked (min): 20
Enter time climbing stairs (min): 10
The number of calories burnt walking is: 95.69
The number of calories burnt climbing is: 67.17
```

Note that the floating point displayed in the outcome of the programme are rounded to two decimal places. Find out how to do this and use `print() & format()` *for rounding the numbers as shown.*

Exercise 5[*]

Learning Objectives: [A] declaring functions with parameters; [B] calling functions; [C] loops; [D] Python random module.

Write a programme that will ask the user to input the number of "virtual" coin tosses. The programme should perform the coin flip with a randomly generated value from 0 to 1. The programme should have a function which takes an integer value to specify the number of tosses and return the number of heads. You should use `random module` from `Python` package for this. For each flip use the `random module` for generating a random integer between 0 and 1, where 0 can be considered **heads** and 1 **tails**.

Print the result of the retuned number of heads and tails. *Think carefully how you could return both of the numbers for:* ***heads*** *and* ***tails****.*

Section N

The output of the programme may look as follows:

```
danny$ python3 coin_toss.py
How many coin flips? : 100
Number of Heads is: 54
Number of Tails is: 46

danny$ python3  coin_toss.py
How many coin flips? : 100
Number of Heads is: 51
Number of Tails is: 49
```

Exercise 6[*]

Learning Objectives: [A] function overloading; [B] declaring functions with parameters; [C] calling functions; [E] Boolean operators

At the beginning of the program, you should request for user input to store the numbers in two variables (**number1**, **number2**) and the Boolean values of **True** and **False** as input (stored in **bool1**, **bool2**) within a **main()** function. You should also display the output of the program within the main() function body.

Task 1

Your task is to create a function overload called **`sum_number()`**, which should add two numbers: (a) the first should add integer values and (b) add floating-point numbers. Think carefully how to call the function and passed in the arguments or parameter lists (study the Lecture 4 resources to help you understand how to do this).

Task 2

You should create a set of functions with the following signatures:

```
int       sum_number(int …    , int … )
float     sum_number(float …  , float … )
```

Section O

`boolean  add(boolean …  , boolean … )`

Remember the parameter lists within the functions' parenthesis are of a particular data type.

The program should take 2 *`int`* numbers as user input which will be used as arguments/parameters for the first two functions. Your function should take two arguments using these data types (e.g. integers, floating-point and Boolean) when invoking or calling the functions.

Remember the program should also take two `boolean` values as user input (**True** & **False**), which should be used for testing the **`add()`** function that takes `boolean` arguments.

Note: `boolean add (boolean …, boolean …)` *should follow the logic of logical OR and AND operators and should be similar to arithmetic addition of 0s and 1s, where 0 stands for* `false` *and 1 stands for* `true`*. Namely:*

```
0 + 0 = 0
1 + 0 = 1
0 + 1 = 1
1 + 1 = 1
```

Think carefully which logical operator would be suitable for the program. Remember you would need to return the appropriate Boolean value when you add the **True** and **False** input.

Run the overloaded functions to test and display the result for each of the above using the **main()** function.

Important Point: Python does not fully support function overloading. This task is to allow you to understand how this is done in general.

The output of the programme may look as follows:

```
danny$ python function_overloaded.py
Please enter Number 1: 2
Please enter Number 2: 4
Please enter boolean 1: False
Please enter boolean 2: False
- - -> add(2, 4) - - - > returns 6

 - - -> add(2.0, 4.0 ); - - - > returns 6.0

 - - -> add(False, False); - - - > returns False
```

Section P

```
danny$ python function_overloaded.py

Please enter Number 1: 23
Please enter Number 2: 45
Please enter boolean 1: True
Please enter boolean 2: False


- - -> add(23, 45) - - - > returns 68

 - - -> add(23.0, 45.0 ); - - - > returns 68.0

 - - -> add(True, False); - - - > returns True

danny$ python function_overloaded.py

Please enter Number 1: 24
Please enter Number 2: 34
Please enter boolean 1: False
Please enter boolean 2: True


- - -> add(24, 34) - - - > returns 58

 - - -> add(24.0, 34.0 ); - - - > returns 58.0

 - - -> add(False, True); - - - > returns True


danny$ python function_overloaded.py

Please enter Number 1: 20
Please enter Number 2: 25
Please enter boolean 1: True
Please enter boolean 2: True
- - -> add(20, 25) - - - > returns 45

 - - -> add(20.0, 25.0 ); - - - > returns 45.0

 - - -> add(True, True); - - - > returns True
```

Section Q

Exercise 7[*] Game Puzzle

Learning Objectives: [A] Creating Function statement; [B] return statement.

Write a puzzle game program that would use some arithmetic operations to calculate the input from a function. The program should define four functions : 'add', 'subtract', 'multiply' and 'divide'. Each of this function should take in two parameter list 'a' , 'b'.

In the program you are required to add the age, subtract the hight, multiply the weight and divide the iq by 2. Note that the previous iq was divided by the initial input value.

The program should accept two input values for **age**, **height**, weight and **iq**.

The following formula should be used for the calculation of the final game puzzle.

$$k = a + (h - \left(w * \frac{iq}{2}\right))$$

Translate this equation to simple Python construct:

Hints: Apply the principle of operation/operator precedence. In this case you will need to consider in which order the calculation should take place:

- Bracket (compute everything in the bracket first)
- Division
- Multiplication
- Addition
- Subtraction

```
danny$ python game_puzzle.py
Let's do some math with just functions!
 Enter first age: 30
 Enter second age: 5
ADDING 30.0 + 5.0

 Enter first height: 78
 Enter second height: 4
SUBTRACTING 78.0 - 4.0

 Enter first weight: 90
 Enter second weight: 2
```

Section R

```
MULTIPLYING 90.0 * 2.0

 Enter first iq: 100
 Enter second iq: 2
DIVIDING 100.0 / 2.0

Age: 35.0, Height: 74.0, Weight: 180.0, IQ: 50.0

Here is the puzzle.
DIVIDING 50.0 / 2
MULTIPLYING 180.0 * 25.0
SUBTRACTING 74.0 - 4500.0
ADDING 35.0 + -4426.0
That becomes:  -4391.0 Can you do this by hand?
```

Exercise 8[!]

Learning Objectives: [A] Loops; [B] declaring functions with parameters; [C] calling functions.

Write a programme that will contain separate functions for printing each of the following shapes:

a. Line
b. Square
c. Triangle

The programme should ask the user which character to use for printing the shapes and the size (i.e. length/height) of the shape.

Your functions should be declared with two parameters: **`char`** character and **`int`** size. The functions should return the number of characters used for printing the shape as an **`int`** value.

```
danny$ python3 shape_printer.py
Enter character : @
Enter size : 6

@@@@@@
Number of chars used: 6

@@@@@@
@@@@@@
@@@@@@
@@@@@@
```

Section S

```
@@@@@@
@@@@@@
Number of chars used: 36

@
@@
@@@
@@@@
@@@@@
@@@@@@
Number of chars used: 21
```