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	2024/25
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 05
Week 05

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] Loops; [B] declaring functions with parameters; [C] calling functions.

Same task as the optional task 8 in tutorial sheet 4.

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

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

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

Section K

Exercise 2[*]

Learning Objectives: [A] Loops; [B] Strings; [C] Conditional statements.

Write a program that would take a name as input and print the name in asterisk (*) large/CAPITAL letters, such as in:

```
danny$ python3 name_asterisk.py
enter name: DANNY
****    ***   *    * *    * *    *
*   * *    * **   * **   *   * *
*    * ***** * * * * * *    *
*    * *    * *  ** *  **    *

****  *    * *    * *    *    *
```

Your program should be able to convert all lower-case letters to upper-case, such as in:

```
danny$ python3 name_asterisk.py
enter name: danny
****    ***   *    * *    * *    *
*   * *    * **   * **   *   * *
*    * ***** * * * * * *    *
*    * *    * *  ** *  **    *

****  *    * *    * *    *    *
```

We have provided you with letters representing alphabetical letters from A-Z in asterisk(*). You don't need to change this. Your task is to use the letters and a for loop to write the program. The program should be able to convert all lower case letters to upper case before printing the outcome(study Lecture 5 resources to understand how the conversion is done).

Section L

```
#!/usr/bin/env python
# write a program that will take a name as input and print the name using asterisk (*)
# You should consider using alphabetical letters from A - Z.
letters = {
    'A': [' *** ', '*   *', '*****', '*   *', '*   *'],
    'B': ['**** ', '*   *', '**** ', '*   *', '**** '],
    'C': [' ****', '*    ', '*    ', '*    ', ' ****'],
    'D': ['**** ', '*   *', '*   *', '*   *', '**** '],
    'E': ['*****', '*    ', '*****', '*    ', '*****'],
    'F': ['*****', '*    ', '***  ', '*    ', '*    '],
    'G': [' ****', '*    ', '*  **', '*   *', ' ****'],
    'H': ['*   *', '*   *', '*****', '*   *', '*   *'],
    'I': ['*****', '  *  ', '  *  ', '  *  ', '*****'],
    'J': ['*****', '    *', '    *', '*   *', ' *** '],
    'K': ['*   *', '*  * ', '**   ', '*  * ', '*   *'],
    'L': ['*    ', '*    ', '*    ', '*    ', '*****'],
    'M': ['*   *', '** **', '* * *', '*   *', '*   *'],
    'N': ['*   *', '**  *', '* * *', '*  **', '*   *'],
    'O': [' *** ', '*   *', '*   *', '*   *', ' *** '],
    'P': ['**** ', '*   *', '**** ', '*    ', '*    '],
    'Q': [' *** ', '*   *', '*   *', '*  **', ' ** *'],
    'R': ['**** ', '*   *', '**** ', '*  * ', '*   *'],
    'S': [' ****', '*    ', '**** ', '    *', '**** '],
    'T': ['*****', '  *  ', '  *  ', '  *  ', '  *  '],
    'U': ['*   *', '*   *', '*   *', '*   *', ' *** '],
    'V': ['*   *', '*   *', '*   *', ' * * ', '  *  '],
    'W': ['*   *', '*   *', '* * *', '** **', '*   *'],
    'X': ['*   *', ' * * ', '  *  ', ' * * ', '*   *'],
    'Y': ['*   *', ' * * ', '  *  ', '  *  ', '  *  '],
    'Z': ['*****', '   * ', '  *  ', ' *   ', '*****'],
      }  # add more letters

# create your for-loop and conditional construct here
```

Published September 3rd 2022

Section M

Exercise 3[*]

Learning Objectives: [A] Functions; [B] Strings; [C] Function calls.

In this exercise you will be printing a string using the given functions below. We have provided you with the functions, you don't need to edit them. Use the function to display the output from the sentence.

The sentence for this task should be ***" All good programming skills come to those who practice constantly."***

The task requirements are (steps):

1. Display the words in the sentence separately, this will appear in the form of single words in quote and comma (as seen in the expected output). Use the **break_words()** function for this. Think carefully about the argument to pass into the function.
2. Sort the words from the sentence and display. Use the **sort_words()** function, also think carefully about the argument to pass into the function.
3. Display the sorted words
4. Call the function to print the first word. Think about the arguments to pass into the function.
5. Call the function to print the last word. Taking into account the argument to pass into the function.
6. The next step is to display the sentence excluding the first word and the last word (from steps 3 & 4).
7. Using the function provided, call the function to display the first sorted word.
8. Using the function provided, call the function to display the last sorted word.
9. After the last two steps, now display the remaining sorted words. This will exclude the last two operations i.e. first sorted word and last sorted word displayed from the sorted words (in step 3).
10. The next task is to display the sorted words from the original sentence. Think carefully about the argument to pass into the function when you call the function.
11. Call the function to print the first and last words from the original sentence. Remember to us the appropriate function to perform this task. This function call will be different from the functions used in steps (4 & 5).
12. Finally, using the function provided here, to display the first and last sorted words in the sentence. Think carefully about the argument to pass into the function when calling the function.

Hint: Define a main() function to perform the operation.

Section N

Please use this codes for the functions (no edit required).

```
"""
A program to use function to compute a string sentence
"""
# Author: Danny Onah
# Date: 3 Feb 2023
# Time: 23:42PM
def break_words(sent):
    """ This function will break up the words in the sentence """
    words = sent.split(' ')
    return words

def sort_words(words):
    """ This function will sort the words in the string sentence. """
    return sorted(words)

def print_first_word(words):
    """ This function will print out the first word after popping it off. """
    word = words.pop(0)
    print(word)

def print_last_word(words):
    """ This function print the last word after popping it off. """
    word = words.pop(-1)
    print(word)

def sort_sentence(sentence):
    """ This function should take in the full sentence and return the sorted words. """
    words = break_words(sentence)
    return sort_words(words)

def print_first_and_last(sentence):
    """ This function should print the first and last words of the sentence. """
    words = break_words(sentence)
    print_first_word(words)
    print_last_word(words)
def print_first_and_last_sorted(sentence):
    """ This function should sort the words and print the first and last words in the sentence.
"""
    words = sort_sentence(sentence)
    print_first_word(words)
    print_last_word(words)
```

Section O

Expected Output:

```
danny$ function_string.py

*** Displaying all words in the sentence separately using single quote and comma ***
['All', 'good', 'programming', 'skills', 'come', 'to', 'those', 'who', 'practice', 'constantly.']


*** Displaying the sorted words ****
['All', 'come', 'constantly.', 'good', 'practice', 'programming', 'skills', 'those', 'to', 'who']


*** Displaying the first word ***
All


*** Display the last word ***
constantly.


*** Display all words excluding the first and last word
['good', 'programming', 'skills', 'come', 'to', 'those', 'who', 'practice']


*** Display the first sorted words ***
All


*** Displaying the last sorted words ***
who


*** Displaying remaining sorted words excluding the first and last sorted words ***
['come', 'constantly.', 'good', 'practice', 'programming', 'skills', 'those', 'to']


*** Displaying sorted words in the sentence ***
['All', 'come', 'constantly.', 'good', 'practice', 'programming', 'skills', 'those', 'to', 'who']
```

Section P

*** Printing first and last words in the sentence ***
All
constantly.

*** Print first and last sorted words in the sentence ***
All
who

Exercise 4[*]

Learning Objectives: [A] Functions; [B] Strings; [C] For loops and Conditional statements.

Write a program to print four different boxes using *symbols*, *width* and *height* of the box. You should define a function called *print_box().* This function should take three input parameters (symbol, width and height). The conditional requirements for this task are as follows:

- The symbols must be a single character string.
- The width of the box must be greater than 2.
- The height of the box must be greater than 2.

Consider using the *if* conditional construct to ensure that the ***symbol***, the ***width*** and the ***height*** met the requirements for the logical conditions set for the task above. Use a for loop to iterate over the given task constructs. The range of the *for-loop* argument should subtract integer number 2 from the height (i.e., …range(height – 2)). In the case of the symbol, ensure you use a conditional statement in your code to check the length of the symbol and ensure that it is *not equal to integer number 1* (i.e., len(symbol) !=1).

We have provided you with a few code to help with the exception part. What you are required to do is to raise Exception if any of the condition(s) is/are not met. For example :

If the symbol is 2 characters (i.e.s contains 2 characters) instead of a single character, the program should trigger the exception message that '*Symbol must be a single character string*'.

To raise an exception after the conditional statement, you will need to do the following:

```
If(condition):
        Raise Exception('Symbol must be a single character string')
```

Do this for all the conditions. The exception message for the ***width*** condition should be:

- '*Width must be greater than 2'*

The exception message for the ***height*** condition should be:

- *'Height must be greater than 2'*

Section Q

Tips: *Using len(symbol !=1) in your Python code simply means you want to ensure that the symbol must be only 1. Think of how to write a code construct to enforce this (len(symbol !=1), if the symbol is not equal to 1, then this should trigger the exception error.*

Code snippets to help you with the exception part.

```
# Author: Danny Onah
# Date: 4 Feb.2023
# Time: 03:39AM GMT


# define a function called box_print(), this function should take three input parameters.
    # check the condition for the symbol here (ensure the length is not equal to 1)

        # raise the exception message here
    # define the if condition for width
        # raise the exception message here
   # define the if condition for height
        # raise the exception message here
# create a for-loop for the first iteration


# create the for loop containing the symbols, width and height of the box as input arguments

    # use a try and except to handle the error message
    try:
        # call the function and pass in the arguments (i.e. symbol, width and height)

    except Exception as err:
       print('An exception happened: ' + str(err))
```

Test your program with some of the conditions set to be **False** and see the resulting outcome (see a sample at the expected outcome with errors).

Section R

Expected output with Errors (here some of the conditions set to be False).

```
danny$ print_box.py


****
*  *
*  *
****
OOOOOOOOOOOOOOOOOOO
O                 O
O                 O
O                 O
OOOOOOOOOOOOOOOOOOO
An exception happened: Width must be greater than 2.
An exception happened: Symbol must be a single character string.
```

Expected output without errors (True conditions).

```
danny$ print_box.py


****
*  *
*  *
****
OOOOOOOOOOOOOOOOOOO
O                 O
O                 O
O                 O
OOOOOOOOOOOOOOOOOOO
xxx
x x
xxx
ZZZ
Z Z
ZZZ
```

Section S

Exercise 5[*]

Learning Objectives: [A] Slice, substrings; [B] Strings concatenation; [C] substitution.

Suppose string1 and string2 are string variables. The string1 hold a value ("super") and string2 hold a value ("person")

Write a program to create and output the strings as follows:

- The concatenation of string1 and string2 with a space between;
- The 0th to 3rd letters (inclusive) of string2
- A string which is exactly the same as string1 except that its 0th character is a "Z".

Hint: Study the Lecture 5 resources and read the slides to understand how this is done.

Expected output.

```
danny$ python3 string_concatenation.py

Concatenation: super person
0th to 3rd letters (inclusive): pers
Replaced first character of string1: Zuper
```

Exercise 6[*]

Learning Objectives: [A] print formatter approach; [B] Strings concatenation; [C] conditional statements.

Write a program to compare two integer string values and check whether they are equal. Using a conditional statement construct to assign the first string1 to the second string2. If both strings are the same, display a comparison message : "*is the same as*" i.e., this means that the value of string1 is identical as the value of string2 (see expected output for an insight). Otherwise, display "*is not the same as*" , if the condition is *False*. Note, you will need to create a variable to store the messages (e.g., comparison = "**is the same as**" etc.). Use any of the print formatter methods or string formatting operator to print the outcome (study the lecture 5 for an insight).

Add another string3 variable to your program. Assign this variable to an integer "**1**" string and concatenate this to string2. Declare a condition to check whether string1 is not equal to string3. If the condition is True, display a comparison message "*not*" (as done above). Otherwise, display an empty string message (i.e., comparison = ""). Use the print formatter function to print the result of the new condition.

Section T

Expected output:

```
danny$ python3 compare_string.py

The strings '119' is not the same as the string '19'.
The strings '119' and '119' are identical.
```

Exercise 7[*]

Learning Objectives: [A] function call; [B] conditional statements; [C] return statement [D] comparison/relational operators [E] return statement

Write a Python program to prompt user to enter three integer numbers and find the maximum number of the three input numbers. Define a *user-defined* function called ***findMaximumNumber()***. This function is required to have only two parameters. The program function should be able to compute and return the maximum number of the three numbers. Ensure you define the function with the parameters before calling this in your program.

Note: Pre-defined built-in function such as ***max()*** is not allowed in this task. Instead use comparison operator and return statement to compare the input arguments.

Expected output:

```
danny$ python3  maximum_number.py

Enter first number: 23
Enter second number: 45
Enter third number: 67
The maximum number is:  67
```