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 06
Week 06

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; conditional statements [B] creating lists and initialising these with arbitrary values;[C] function call.

Write a programme that declares and initialises two lists of friends.

These lists should contain no more than 3 names of individuals that two users may think they have as friends in common.
For example, the first list should contain the names of friends (e.g. Danny, Elaine and Jane) that `user1` thinks he or she shares with `user2`. The second list should contain the names of friends (e.g. Jane, Elaine and Danny) that `user2` thinks he or she shares with `user1`.

The programme should compare the initialised lists to check whether the two lists share all the names in common, but not necessarily in the same order. If the users have same friends in common the program should display "You have friends in common!", otherwise display "Your friends are different!"

Tips: *Write this program with a function (with a function name- **compare**). Use for-loops to iterate over the names for both users and use conditional statements to check whether both users have same friends in common. Sort the lists using the sort() function before comparing them.*
***Note**: Test your program by adding a new name to any of the lists to display the message 'Your friends are different!'.*

The output of the programme may look as follows.

```
danny$ python3 list_comparison.py
Friends of user1 are:
Danny
Elaine
Jane
Friends of user2 are:
Danny
Elaine
Jane
You have friends in common!


danny$ python3 list_comparison.py
Friends of user1 are:
Danny
Elaine
Jane
```

```
Friends of user 2 are:
Daniel
Danny
Elaine
Jane
Your friends are different!
```

Exercise 2[*]

Learning Objectives: [A] loops; [B] creating lists and initialising these with arbitrary values;[C] passing lists to functions;

Write a programme that will use a function to calculate a Fibonacci number for a given index. Your new programme should take Fibonacci index via user input (e.g., 13) and create a list to accommodate *all* Fibonacci numbers up to and including the provided index (i.e., 13) as `long` numbers.

Your programme should then declare and implement a function that takes a `list` as a parameter, and initialise the elements of the list by assigning the relevant Fibonacci numbers. After returning the control, you should traverse the list and print all the Fibonacci number series or elements of the list.

Tips: *Think whether the list should be passed to the function by reference variable or by value. Therefore, do you need to return the initialised list from the function?*

The output of the programme may look as follows:

```
danny$ python3 fibonacci_numbers.py
How long the Fibonacci sequence should be? : 13
0 1 1 2 3 5 8 13 21 34 55 89 144
```

Section L

Exercise 3[*]

Learning Objectives: [A] loops; [B] creating lists and initialising these with arbitrary values; [C] loops & conditional statements; [D] passing list to functions; [E] splitting list.

Write a programme that would prompt user for input assuming that the user will provide a one-sentence summary of what was learnt while completing the current Tutorial Sheet. The sentence should be stored in an empty list called *sentence*.

The programme should print each of the words in the **sentence list** on a new line, and then print "*You learnt something about lists. Well done!*" if the list contains either the word "list" or "lists". Alternatively, it should print: "*Practice and study more about using lists in Python*".

Tips: *To complete this exercise, you will need to know how to split a list (see Lecture 6 for an insight). You should also know how to check if a list contains an element using the* ***in*** *keyword.*

The output of the programme may look as follows:

```
danny$ python3 Searching_list.py
Please enter your statement: I learnt how to check whether two lists
are equal.
I
learnt
how
to
check
whether
two
lists
are
equal.

You have learnt something about lists. Well done!

danny$ python3 Searching_list.py
Please enter your statement: Python programming is very interesting!.
Python
programming
is
very
interesting!.

Practice and study more about using lists in Python!
```

Section M

Additional (Optional task): *Write your program to remove all punctuation marks such as full-stop, commas, exclamation and so on. So if the user types a sentence that end with a word* ***list*** *or* ***lists*** *and a full-stop such as 'In Python programming, the most interesting concept is lists.' You program should first remove all the punctuations in the sentence before checking whether the word 'list' or 'lists' is in the sentence list.*

Exercise 4[*]

Learning Objectives: [A] loops & conditional statements; [B] creating lists and initialising these with arbitrary values;[C] functions

Write a programme that would ask the user to enter the length of the list to be created (e.g., 5). The list should hold elements of type `int`. The list should be an empty list.

Once a list of appropriate type and length is created, the programme should initialise or update the empty list using the user's input. The programme should then calculate the average value of all the numbers in the list and display the result.
The program should make use of two functions, one as the ***main()*** function and the other ***average()*** function. You can use either a *for-loop* or a *while-loop* for the iteration. Your program should take into consideration the length of the list within the loop construct. If the user type in the string 'done' the program should display the average using the input provide even though the initial length decided has not been reached. Finally, display the elements in the list. The program should be able to handle the ***error*** if the first input entered in index 0 is 'done' (see the expected outcome section).

Note: *The string 'done' should not be the first input (Error: we cannot convert string to floating point) . Consider how to track and handle the error if the first input is a string or empty space. Finally, handle ZeroDivisionError if the length of the list is zero.* ***ZeroDivisionError*** *simply means we are trying to divide by zero.*

The output of the programme may look as follows:

```
danny$ python3 list_average.py
Enter the length of the list: 7
Enter number for index 0: 3
Enter number for index 1: 7
Enter number for index 2: 9
Enter number for index 3: 2
Enter number for index 4: 5
Enter number for index 5: 8
Enter number for index 6: 4
The average is: 5.43
['3', '7', '9', '2', '5', '8', '4']
```

Section N

```
danny$ python3 list_average.py
Enter the length of the list: 4
Enter a number for index 0: 7
Enter a number for index 1: 9
Enter a number for index 2: done
Average is:  8.0
['7', '9']

danny$ python3 list_average.py
Enter the length of the list: done
Invalid entry! Please enter integer ONLY.
Enter the length of the list:

danny$ python3 list_average.py
Enter the length of the list: 4
Enter a number for index 1: 9

danny$ python3 list_average.py
Enter the length of the list: (empty space here)
Invalid entry! Please enter integer ONLY.
Enter the length of the list:

danny$ python3 list_average.py
Enter the length of the list: 4
Enter a number for index 0: done
Invalid entry! Please enter integer ONLY.
Enter the length of the list:

danny$ python3 list_average.py
Enter the length of the list: 9
Enter a number for index 0: k
Invalid entry! Please enter integer ONLY.
Enter the length of the list:

danny$ python3 list_average.py
Enter the length of the list: 0
Please enter positive integer ONLY!
Enter the length of the list:

danny$ python3 list_average.py
Enter the length of the list: 6
Enter a number for index 0: 0
Enter a number for index 1: 0
```

```
Enter a number for index 2: 0
Enter a number for index 3: done
Average is:  0.0
['0', '0', '0']
```

Exercise 5[*]

Learning Objectives: [A] loops & conditional statements; [B] creating lists and initialising these with arbitrary values;[C] updating and searching list items [D] functions

Given the list of names below, write a program that would continuously prompt and request for user to enter a name. The program would apply the concept of a search mechanism, the user should be able to search for name on the list. If the name is found, the program should display both the name and the index position of the name.

```
names = ["Danny", "Elaine", "Jane"]
```

If the name search for is not in the list, the program should add the name and display the name and the index position for that name. You should use a loop construct of your choice to perform the looping. The program should be able to stop looping once the user entered the word 'done' or 'finish' as input. Finally, the program should then display all the names in the list and their index position.

Program Requirement: This program should be written with two functions. One of the functions should be the *main()* function and the other should be the *search()* function.

Note: *Within the source code file, you will find more guidance on how to structure your program. Remember the search name mechanism adhere to Python case sensitive structure (for example if the name entered and stored in the list is "Danny", then this is different from "danny").*

The output of the programme may look as follows:

```
danny$ python3 adding_names_list.py

enter name to search for: Danny
The name: Danny is in: index[0]
enter name to search for: Elaine
The name: Elaine is in: index[1]
enter name to search for: Jane
```

Section P

```
The name: Jane is in: index[2]
enter name to search for: Mike
The name does not exist! Do you want to add this name (yes or no):y
adding name Mike to position:  4
enter name to search for: Joe
The name does not exist! Do you want to add this name (yes or no):n
name Joe NOT added to the list ...
enter name to search for: Lucy
The name does not exist! Do you want to add this name (yes or no):y
adding name Lucy to position:  5
enter name to search for: Mike
The name: Mike is in: index[3]
enter name to search for: Lucy
The name: Lucy is in: index[4]
enter name to search for: finish
The names are:
  The name Danny is in index: 0
  The name Elaine is in index: 1
  The name Jane is in index: 2
  The name Mike is in index: 3
  The name Lucy is in index: 4
```

Section Q

Exercise 6[*]

Learning Objectives: [A] loops; [B] creating function with arbitrary values;[C] declaring functions which allow unspecified number of arguments (variable-length arguments);[D] conditional statements;

Expand the given arbitrary arguments `by` adding two functions that takes an *`int`* parameter year and an *arbitrary* number of month parameters of data type `String`. The function should not return anything.

The program should use two functions; **main()** function and the **monthsOnLeave()** function. The required ***monthsOnLeave()*** function calls already exist in the ***main()*** function (e.g. `monthsOnLeave(year1, month01, month02, month12)` etc, together with the values for the months and years have been initialised within the **main()** function (you don't need to edit them … see the code snippet below). The result of these function calls should display the year, followed by the list of months you are planning to be on leave. The sample output is shown after the listing of the provide code snippets below.

Tips: *Please note, that you are not supposed to overload the* `monthsOnLeave(…)` *function, but instead declare it to allow arbitrary number (meaning to allow so many parameter lists for the function) of* `String` *data type arguments. Revise or look at Lecture 4 Functions (Arbitrary Arguments), to help you understand how this is done.*

```
#!/usr/bin/env python
# Author: Danny Onah
# Date: 22 February 2023
# Time: 01:46AM GMT
"""
A program to compute an arbitrary list of months.
"""
def main():
    month01 = "January"
    month02 = "February"
    month05 = "May"
    month07 = "July"
    month12 = "December"
    year1 = 2020
    year2 = 2021
    year3 = 2022
    year4 = 2023
    monthsOnLeave(year1, month01, month02, month12)
    monthsOnLeave(year2, month01, month07)
    monthsOnLeave(year3, month01, month05, month07, month12)
    monthsOnLeave(year4, month01, month02, month07, month12)

# *** add the monthsOnLeave() function here ***
```

Section R

The output of the programme may look as follows.

```
danny$ python3 arbitrary_arguments.py

In 2020, I am on leave in:
January,
February,
and December.

In 2021, I am on leave in:
January,
and July.

In 2022, I am on leave in:
January,
May,
July,
and December.

In 2023, I am on leave in:
January,
February,
July,
and December.
```

Section S

Exercise 7[*]

Learning Objectives: [A] loops; [B] List;[C] conditional statements [D] function

Write a program that declares and initialises a `List` of `String` elements with a number of entries: ***Solar, Wind, Hydrogen* and *Tidal,*** stored in a variable called ***clean_energy***. The programme should ask the user to provide a String input for an energy source, for example ***Coal***. If the declared List contains the user input (e.g., Wind) as one of the elements, the programme should print: *"Yes, Wind is a clean source of energy"*. If the declared List does not contain a user input (e.g., Coal), the programme should print: *"No, Coal is NOT a clean source of energy"*. Let's assume the list is case sensitive, if the user input appears to be all caps (upper case) or lower case, you program should be able to convert this to ***Title case*** (i.e., the first letter of the energy input should be capital letter e.g., solar = Solar).

Program Requirement: This program should be written using a single function know as **find()**. This **find()** function should take no parameter list and should return nothing. The program should prompt user continuously for input and display the outcome. The program should stop executing the loop and terminate when the user enters the word **'done'** as input.

The output of the programme may look as follows.

```
danny$ python3 list_find.py
Enter energy source: WIND
Yes, Wind is a clean source of energy

Enter energy source: solar
Yes, Solar is a clean source of energy

Enter energy source: TiDaL
Yes, Tidal is a clean source of energy

Enter energy source: HydrogeN
Yes, Hydrogen is a clean source of energy

Enter energy source: coal
No, Coal is NOT a clean source of energy

Enter energy source: done
Goodbye!
```

Section T

Exercise 8[*]

Learning Objectives: [A] loops; [B] creating lists and assigning arbitrary values; [C] handling exceptions;[D] return statements.

Write a programme that will create a list of elements to be added to an empty list. The length of the list should be determined by the user input. The program should accept only integer whole number for the length of the list. If the user entered any other data types (e.g., strings, float values etc), the program should throw an exception error (use ***try*** and ***except*** to handle this error). Once all elements are entered your program should print *"List is now full."*. This should print all the elements in the list (*"List element are:"*). The program should continue looping. To terminate the loop, the user will need to enter **-1**(*revise more on the concept of sentinel*) to stop the execution of the program.

Program Requirements:

main() Function:

You should write this program using three functions. The first function should be the **main()** function. The user input should be entered from the ***main()*** function and the value passed in as argument to the ***listInitialisation()*** function. The **main()** function should be able to track if non-integer value is entered using the *try/except* statement. The result of the program should also be displayed from this *main()* function.

listInitialisation() Function:

The ***listInitialisation()*** function should take one parameter called *length.* The function should declare an empty list and the length should be passed as argument to the for-loop range that would indicate the number of items to be added to the empty list. This ***listInitialisation()*** function should return the empty list at the end of the operation (using the return statement).

printList() Function:

The ***printList()*** function should take no parameter list. The function should be used to print the items in the list. In order to successfully do this, you would need to call this *printList()* function in the *main()* function as well.

Tips: *The program should continue prompting user for input. Once the length of the list is reached the program should display the list of items. If the user entered -1 in the length, the program should end and display "Goodbye!".*

The output of the programme may look as follows.

```
danny$ python list_initialisation_exception.py
Enter length of list: 10
Enter element in index[0]: 34
Enter element in index[1]: 67
Enter element in index[2]: Danny
```

Section U

```
Enter element in index[3]: INST0002 Programming 1
Enter element in index[4]: 4.6
Enter element in index[5]: 90.23
Enter element in index[6]: £65.90
Enter element in index[7]: $29.78
Enter element in index[8]: 1234534
Enter element in index[9]: [3,4]

List is now full!
List elements are:
['34', '67', 'Danny', 'INST0002 Programming 1', '4.6', '90.23',
'£65.90', '$29.78', '1234534', '[3,4]']

Displaying elements:
34
67
Danny
INST0002 Programming 1
4.6
90.23
£65.90
$29.78
1234534
[3,4]

Enter length of list: -1
Goodbye!
```