OLLSCOIL NA hÉIREANN, CORCAIGH
THE NATIONAL UNIVERSITY OF IRELAND, CORK

COLÁISTE NA hOLLSCOILE, CORCAIGH
UNIVERSITY COLLEGE, CORK

Examination Session and Year	Summer 2020 - Tuesday 28th April (Irish Summer Time)
Module Code	CS1117
Module Title	Introduction to Programming
Paper Number	1
External Examiner	Prof. Omer Rana
Head of School/ Department	Prof. Cormac J. Sreenan
Internal Examiners	Dr. Jason Quinlan
Instructions to Candidates	Answer all five questions. **Your answers need to be commented**. This is an open-book exam. You may consult any publicly accessible written sources, either printed or online, and you may use any publicly available software on your laptop or desktop to compose, execute and test your work. You must list any resources you use apart from CS1117 written or software resources (materials from CS1117 Canvas page and basic Python resources used in the CS1117 labs/assignments). You must submit your work using the "answer-book" template provided, *CS1117_Sum20_solutions.py*, within the specified time. You may not communicate with any individual (other than the module instructor) by any means and for any purpose for the duration of this exam. Write your code in Python3 but do not use Python3 functions or libraries, we did not cover in class. I may not have those specific libraries installed on my test machine, and you may lose marks. This is to certify that the work I am submitting is my own and has been done by me solely and not in consultation with anyone else. Neither I nor anyone else have submitted this work for assessment, either at University College Cork or elsewhere. I have read and understood the regulations of University College Cork concerning plagiarism. Where breaches of the declaration are detected, these will be reviewed under UCC student conduct and plagiarism policies. Any breach of the examination rules is a serious issue and can incur penalties. **By proceeding you signify your agreement to ALL the above rules and declaration**.
Duration of Paper	3 hours + 20 minutes read/upload + personal DSS examination allowances (if applicable).
Special Requirements	**This paper equates to 225 marks of your overall module mark (/300).** **The paper is marked out of 100%.**

Q1 Multiple Choice - please circle your choice of correct answer - ONE answer to be selected only. [20%]

(i) What is the assigned type of the variable "*var*" when this code is run, if the key 7 is pressed on the keyboard, followed by pressing the return key? **2%**

```
var = input("What is your favourite number? ")
```

A. integer B. string C. tuple D. error in code

(ii) Which of these code snippets will ***NOT*** reverse the list named *alist*? **2%**

A. alist.reverse() B. alist[::-1] C. alist[:-1] D. alist[len(my_list)::-1]

(iii) Which of these assignment values is ***NOT*** mutable, i.e., can not be modified? **2%**

A. var = 2 B. var = 2.1 C. var = "2" D. var = (2, 2)

(iv) Which of these code snippets will cause an exception? **2%**

A. 1 + 2 B. 1 + "2" C. "1" + "2" D. 1.2 + 1

(v) Which of these is ***NOT*** a conditional statement, assuming a1 and a2 are strings? **2%**

A. a1 != a2 B. not a1 = a2 C. not a1 == a2 D. a1 < a2

(vi) Which of these variable names will ***NOT*** cause shadowing? **2%**

A. string B. int C. list D. str

(vii) Which of these code snippets will cause an exception/error, assuming a1 and a2 are integers? **2%**

A.
```
if (a1 < a2):
    print(a1)
```

B.
```
if (a1 < a2):
    print(a1)
else:
    print(a1+a2)
```

C.
```
if (a1 < a2):
    print(a1)
elif (a2 < a1):
    print(a2)
```

D.
```
elif (a2 < a1):
    print(a2)
else:
    print(a1+a2)
```

(viii) Which of these code snippets does not determine object equality? **2%**

A. a1 is a2 B. a1 in a2 C. a1 == a2 D. id(a1) == id(a2)

(ix) What is the assigned value of "*var*" after this code is run? **2%**

```
var = "hello you"
var.capitalize()
```

A. hello you B. Hello you C. HELLO YOU D. Hello You

(x) Which of these is the correct structure of a list comprehension? *i*i* is the expression to be calculated... **2%**

A. [for.. i*i if..] B. [if.. for.. i*i] C. [i*i if.. for..] D. [i*i for.. if..]

Q2 **[20%]**

```
def loop_the_loop(a1, a2):
    new_loop = []
    for e1 in a1:
        for e2 in a2:
            new_loop.append(e1+e2)
    return new_loop
```

(a) Rewrite the function *loop_the_loop* using *while* loops rather than *for* loops. **5%**

(b) Rewrite the function *loop_the_loop* using list comprehension. **5%**

(c) Rewrite the function *loop_the_loop* using map, lambda, and zip. **6%**

(d) Add error handling to the original *loop_the_loop* function, shown at the top of this page, and return an error message. **4%**

Q3 **[20%]**

(a) Write a function called *add_to_list* which has three parameters, *element, alist* and *index*

This function is called using: *add_to_list(element, alist, index)* and returns a modified list.

The objective of this function is to add an *element* to *alist* at a specific *index* value.

```
add_to_list("a", [1,2,3,4,5], 3) -> [1,2,3,"a",4,5]
```

If the *index* value is larger than the length of the *alist*, the *element* is added to the end of the *alist*.

```
add_to_list("a", [1,2,3,4,5], 6) -> [1,2,3,4,5,"a"]
```

If the *index* value is lower than the length of the *alist*, the *element* is placed at that *index*, and each of the other values in the *alist* are moved to their respective next index value in the *alist*:

```
add_to_list("a", [1,2,3,4,5], 0) -> ["a",1,2,3,4,5]
```

If the provided *index* value is negative, or if the *index* value is not passed to the function, the *element* should be placed at *index* zero

```
add_to_list("a", [1,2,3,4,5], -4) -> ["a",1,2,3,4,5]
add_to_list("a", [1,2,3,4,5]) -> ["a",1,2,3,4,5]
```

Finally, strengthen the function by adding exception handling, returning an error message if the wrong type of parameters is passed to the function.

```
add_to_list([1,2,3,4,5], 7, "a") -> "Oops, error..."
```

Q4 **[20%]**

(a) The document *details.txt* contains rows of information formatted as tuples, each tuple has 3 elements, which are *user_id, user_name*, and *user_address*. Note: multiple rows for the same *user_id, user_name* will contain different *user_address*. A sample of the formatting of the document text is shown. **14%**

```
(765, "Rick Sanchez", "space address 1")
(865, "Morty Smith", "home address 2")
(825, "Summer Smith", "home address 2")
(765, "Rick Sanchez", "space address 2")
```

Write a function *read_file*(*input_file*) that will read in the content of the input file, and create a dictionary with *user_id* as the key and add a list of *user_name*, and *user_address(s)* as the dictionary value.

```
{765 : ["Rick Sanchez", "space address 1", "space address 2"],
 865: ["Morty Smith", "home address 2"]
 825: ["Summer Smith", "home address 2"]}
```

Items to consider include:

a. formatting of the original text file. What needs to be removed during formatting.

b. the text file may contain duplicate information, how will this impact the creation of the dictionary keys.

c. adding the additional addresses for the same *user_id, user_name.*

(b) Write a function, *write_dict(d),* that will sort and write each item in the dictionary, *d*, on a separate line to the document *output.txt*, in this formatted output: **6%**

id <key> <spacing added here> <user_name> (also has/has) an address of <user_address>

```
id 765  Rick Sanchez has an address of space address 1
id 765  Rick Sanchez also has an address of space address 2
id 825  Summer Smith has an address of home address 2
id 865  Morty Smith has an address of home address 2
```

Strengthen all functions in this question by adding exception handling, return `"Oops, error..."` if an error occurs.

Q5 **[20%]**

(a) Write a function called *biggest_retail_chain*, which has one parameter, a dictionary *d.* **8%**

This function is called with: *biggest_retail_chain (d)* and returns a list of the retail chains which serve the greatest number of towns (this list may be zero if the dictionary is empty) or contain one or more items.

A sample dictionary, *d*, and the returned list from calling the *biggest_retail_chain (d)* function is shown:

d ={'tesco':["cork", "dublin"], 'dunnes':["kerry", "cobh", " dublin", "wilton"]}

```
biggest_retail_chain(d) -> ['dunnes']
```

(b) Write a function called *common_towns*, which has 3 parameters, a dictionary *d*, a retailer *rc1*, a retailer *rc2.* **8%**

This function is called with: *common_towns (d, rc1, rc2)* and returns a list of towns that the two retail chains *rc1* and *rc2* have in common.

Using the same dictionary, *d*, from Q5. part (a), the returned list from calling the *common_towns (d, rc1, rc2)* function is shown:

```
common_towns(d, "tesco", "dunnes") -> ["dublin"]
```

Return an empty list if the retailers are not defined in *d*, or if the retailers have no towns in common:

```
common_towns(d, "aldi", "lidl") -> []
```

(c) Write a function *sorted_print*, which has one parameter, a dictionary *d* - *sorted_print (d).* **4%**

This function prints out the names of the retailers sorted alphabetically, and also the locations of the retailers sorted alphabetically, in a columnised format.

Use tab to separate the string, consider how the length of the strings impact on formatting, and Capitalise each output value in the print function.

Using the following dictionary:

d ={'tesco':["cork", "dublin"], 'dunnes':["kerry", "cobh", " dublin", "wilton"]}

```
biggest_retail_chain(d) -> ['dunnes']
```

With the printed output is shown below:

```
sorted_print(d) ->
```

Dunnes	Cobh	Dublin	Kerry	Wilton
Tesco	Cork	Dublin		

Strengthen all functions in this question by adding exception handling, return `"Oops, error..."` if an error occurs.

THIS PAGE IS INTENTIONALLY BLANK