The Australian National University **Research School of Computer Science, CECS**

COMP1730/COMP6730 – Programming for Scientists
Mid-Semester Examination

Overview

This is the mid-semester examination for COMP1730/COMP6730, Programming for Scientists.

The exam consists of five short answer questions and two programming problems. Some questions have multiple parts. Questions are not all worth the same number of marks. **Question 7 is for COMP6730 students only**. If you are a COMP1730 student you will not receive any marks for attempting Question 7.

For COMP1730 students the exam is out of 20. For COMP6730 students, the exam is out of 24 which will be scaled to a mark out of 20.

You do not have to answer all questions, but you will, of course, only receive marks for the questions you answer. You can attempt the questions in any order you like.

On your desktop you will find a folder titled `MidSemesterExamination` which contains a file `answers.txt` in which you should write your answers to the written questions. It also contains files `Question_5.py` and `Question_6.py`, in which you should write your code for the two practical problems. **Do not rename or move the solution files!** Each practical problem has a corresponding test file `Test_Question_5.py` and `Test_Question_6.py` which works in an identical fashion to the tests we provided for the second and third homework exercises.

The Exam Environment

The lab exam environment is exactly the same as the regular CSIT lab computer environment except that there is no internet access and you will not be able to see your normal home directory. In particular, you should find the same python IDEs (Spyder, PyCharm, IDLE, etc.) in the exam environment as you have in the labs. To assist you, the following material is available:

You will find the lecture slides, lab exercises and a copy of Downey's text book in the `Resources` folder on the desktop. The python interpreter's built-in help function should work as normal.

If the environment is unfamiliar, you may want to quickly review the first part of Lab 1.

Short Answer Questions

For Questions 1 - 4 (and Q7 if you are COMP6730 student), you should write your answers into a template file called `answers.txt`. You will find this file in the the folder `MidSemesterExamination` on the desktop.

You can open the file in any text editor (e.g., double-clicking it will open it in gedit; you can also edit it in any of the python IDEs). Do not use a word processor (like OpenOffice) to edit the file, since these will save it in a different format.

Important: You must remember to save your answers file before closing and logging out. It will not happen automatically.

The template file contains some formatting lines (lines beginning with ////). Do not remove or change the formatting lines; if you do, we will not be able to find your answers and you may not receive any marks for them. The template file clearly shows where you should write your answers to each question.

There is no limit to the length of answers, but you should try to keep your answers short and clear. Answers that contain correct statements will gain less than full marks if they also contain irrelevant or incorrect statements.

Programming Problems

Solutions to the programming problems (Q5 and Q6) will be marked on the basis of **functionality only**. To be fully functional means **solving the problem that is described in the specification**. Passing the tests is an indication that the solution may be correct, but it is not the criterion. **Failing the tests proves that your solution is wrong**. A partially functional solution may receive some marks. However, marks are not proportional to the number of tests passed. A submission that does not run (for example, due to syntax errors) or that fails every test case will receive zero marks. We will **not** mark you on code quality for the **programming problems only**.

The Test Framework

Each programming question has a corresponding testing program, similar to those you saw for the later homework assignments. They are named accordingly, so the testing program for `Question_5.py` is called `Test_Question_5.py`. Running the test program will run a series of tests on your solution and provide you with some output including the number of tests passed.

The testing program will reject your file if it breaks any of the following requirements:

- Your solution file must be syntactically correct python code.
- You cannot rename the function you are asked to implement, nor change the number of parameters it takes, though you are free to write additional functions if it helps you to break down the problem.
- You cannot move the answer file to another location or rename it.

Remember that the testing program can only prove that your solution is wrong. **Passing the tests does not prove that your solution is correct**.

Examination Questions Start on the Next Page

Question 1: 3 Marks

Below are three suggested function names. For each name, state whether the name is a valid function name in Python, and if it is valid, in what situations, if any, the name would be an appropriate function name:

1(a): `lambda`

1(b): `list`

1(c): `2_times_the_number`

Question 2: 3 Marks

In the Latin alphabet (used by languages such as English), the letters *a*, *e*, *i*, *o* and *u* are known as *vowels*. (Note: the uppercase versions of these letters are also vowels). Below, are three attempts to define a function `count_vowels(input_string)` which counts the number of vowels in the given string.

For each of the functions below, answer whether it is correct or not. Correct means that the function runs without error and returns the right answer for all valid arguments (that is `input_string` is a string). If a function is not correct, explain precisely what is wrong with it. For example, if it has a syntax error, you should describe which part of which line is wrong; if the function runs but returns the wrong answer, give a concrete example of arguments that cause this to happen **and explain why it does**. Writing a new function without explaining what is wrong with the given function is not an acceptable answer to the question (regardless of whether that function is correct or not).

2(a):
```
def count_vowels(input_string):
    input_string = input_string.lower()
    number = 0
    for character in input_string:
        if character == 'a' or 'e' or 'i' or 'o' or 'u':
            number = number + 1
    return number
```

2(b):
```
def count_vowels(input_string):
    number = 0
    for character in input_string:
        if character in 'aeiouAEIOU':
            number = number + 1
    print(number)
```

2(c):
```
def count_vowels(input_string):
    input_string = input_string.lower()
    vowel_string = 'aeiou'
    number = 0
    index = 0
    while index < len(input_string):
        if input_string[index] in vowel_string:
            number = number + 1
    return number
```

Question 3: 4 Marks

The following function calculates the number of duplicate elements in a sequence.

```
def count(l):
    x = 0 # X is the average element
    p = 0 # P is for duplicates
    while x < len(l):
        y = x + 1 # Y is the index we are comparing with
        id = False # Keep track of whether the current number has a duplicate or not
        while y <= len(l) - 1:  # Put -1 so we don't get an index error
            if l[x] == l[y]:  # Yay, we found a duplicate
                id = True
                y = y + 1 # Check the next element
            else:
                y = y + 1 # Check the next element
        if id == True:
            p = p + 1 # Add 1 to P
        else:
            a = "Do Nothing" # We don't want to do anything in this case
        x = x + 1 # Make sure teh loop finishes
    # Return the number
    return p
```

Describe all the ways the code quality of this function could be improved. To gain marks, your suggested improvements must be specific. (For example, if you think the function should have additional comments, describe what comments should be added and where. Just answering "more comments" is not enough.)

Question 4: 2 Marks

Consider the following function:

```
def fun_A(x, y):
    '''
       Does fun stuff!
       Assumes that x and y are integers between
       0 and 100 (inclusive).
       Returns a float.
    '''

    import math

    if x == y:
       return 0

    first = math.cos((y % 75) * (math.pi / 180))
    second = math.sin((x % 30) * (math.pi / 180))
    return (first + second) / (abs(x - y))
```

The function docstring specifies that `fun_A` assumes `x` and `y` are integers between 0 and 100 (inclusive). Given this assumption, what is maximum value `fun_A` can return and what are the corresponding values of `x` and `y`.

Hint: Do not try and solve this analytically. Instead, think of a way you can quickly test a whole range of possible inputs.

Question 5: 4 Marks

Most of us know that in the Gregorian calendar system, we have a *leap* year roughly once every four years. However, the precise definition of a leap year is a bit more complex.

A leap year is a year that is divisible by four, but is not divisible by 100, unless it is also divisible by 400. So 1924, -2020 and 2000 are leap years, but 1975, -2022 and 2100 are not.

Write a function `is_leap_year(year)` that takes as input a year, and returns `True` if the year is a leap year, and `False` otherwise.

A skeleton file `Question_5.py` is provided and you should write you answer in this file. A test file `Test_Question_5.py` is also provided. Running `Test_Question_5.py` will call the function `is_leap_year` in `Question_5.py`.

- You can assume that the argument is an integer.
- The function must return `True` or `False`.
- You may receive part marks for a function that passes *some* of the test cases.

Remember that your solution must contain only function definitions and comments (optionally, import statements). If it contains anything else, it is invalid and you will receive zero marks.

Question 6: 4 Marks

Write a function `smallest_greater(sequence, target)` that takes a sequence, and returns the smallest element in the sequence greater than or equal to the target value.

For example, if the sequence is `(1, 11, 2, 21, 5, 9)` and the target is `4`, your function should return `5`. If the sequence is `'COMP1730 is hard'` and the target is `'Z'`, your function should return `'a'`.

- You can assume that at least one element in the sequence is greater than or equal to the target value.
- You can assume that all elements in the sequence are of the same type as the target value (that is, if the sequence is a list of numbers, then the target value is a number).
- You should not assume that the elements of the sequence are in any particular order.
- You should only use operations on the sequence that are valid for all sequence types.

Remember that your solution must contain only function definitions and comments (optionally, import statements). If it contains anything else, it is invalid and you will receive zero marks.

Question 7 for COMP6730 students only is on the next page

Question 7: 4 Marks COMP6730 ONLY

7(a): 2 marks

Let:

`s = 'Angry Public Swamp Methods'`

For each of the following substrings, find values of `a`, `b` and `c` such that `s[a:b:c]` produces the given substring. (You may write "not specified" if you do not wish to specify the value of a particular input so it uses the default).

(i): `'Angr'`

(ii): `'Swamp'`

(ii): `'sdohteM pmawS cilbuP yrgnA'`

(iv): `'dem b'`

7(b): 2 marks

Consider the following function:

```
def fun_B(x):
    total = 0
    while 1 + x < 1:
        total = total + x
        x = x / 2
    return total
```

Explain precisely why the call `fun_B(-1)` does **not** get stuck in an infinite loop.

End of Exam Questions