COMP1730/COMP6730 – Programming for Scientists
Mid-Semester Exam Sample Questions

Note: These questions are taken from the Semester 2, 2018 COMP1730 Mid-Semester Examination.

Question 1 (3 Marks):

Below are three suggested function names. For each name, answer (a) whether this is a valid name in python, and (b) describe in what situation, if any, it would be appropriate to use as a function name.

1(a): `over-21`

1(b): `_2_steps_forward`

1(c): `Def`

Question 2 (1 Mark):

For each of the following statements, answer whether it is true or false. (There is no mark penalty for incorrect answers, but your answer must be correct for more than half the statements to receive any marks).

2(a): The arguments provided in a function call map to the function's parameters in the same order (i.e., first argument to first parameter, and so on).

2(b): A function must be defined before it can be called.

2(c): A function can only be called from one place within a program.

2(d): A function suite can contain only one return statement.

2(e): The name of a function can not be used as a variable name anywhere in the file where the function is defined.

Question 3 (3 Marks)

Below are three attempts to define a function `equal_ratio(a, b, c, d)`. The arguments should all be integers, and the function should return `True` if and only if the ratios `a/b` and `c/d` are equal.

For example, `equal_ratio(1,2,2,4)` should return `True`, because `1/2 == 2/4`. `b` and `d` must be different from zero, as otherwise the ratio is undefined.

For each of the functions below, answer whether it is correct or not. Correct mean that the function runs without error and returns the right answer for all valid arguments (that is, a , b , c and d are all integers and b and d are not zero). 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).

3(a):
```
def equal_ratio(a, b, c, d):
    return a // c == b // d
```

3(b):
```
def equal_ratio(a, b, c, d):
    a * d == c * b or a == c and b == d
```

3(c):
```
def equal_ratio(a, b, c, d):
    return print(a / b) == print(c / d)
```

Question 4 (1 Mark):

A function is defined as follows:

```
def funA(x):
    y = 2 * x - 1
    return y
    z = x ** 2
    return z
```

Give an example of an argument that will make the function return the value `16` of type `int`, or explain why this can not happen.

Question 5 (4 Marks):

Here is a definition of a somewhat complicated and poorly documented function:

```
def funB(x):
    '''x must be a sequence'''
    i = 0
    y = x[0]
    a = 0
    while i < len(x):
        j = 0
        z = 0
        while j < len(x):
            if x[i] == x[j]:
                z = z + 1
            j = j + 1
        if z > a:
            y = x[i]
        a = max(z, a)
        i = i + 1
    return y
```

5(a): (2 marks) Explain what funB does in general. Write your explanation so that it is like a good docstring for the function. In other words, a good explanation should describe the purpose of the function, not step-by-step how it works.

5(b): (2 marks) The author of the function above has made it clear that it is intended to work only on arguments that are sequences. But does it work on any sequence? Write down all requirements on the argument sequence that are necessary for the function to run without causing a runtime error.

Question 6 (4 Marks):

An integer n is called a square if $n = m * m$ for some integer m. As we all know, 1, 4, 9, 16, and 25 are squares. But what about bigger numbers, like 543168? (It's not a square. 543169 is.)

Write a function, `is_square` that takes as argument a positive integer and returns `True` if that integer is the square of another integer, and `False` if it is not. A skeleton file called `Question_6.py` is provided. A testing program called `Test_Question_6.py` is also provided. The testing program will test the function `is_square` in the file `Question_6.py`. The function that you write must be named `is_square` and it must have one parameter.

- You can assume that the argument is a positive integer.
- The function must return `True` or `False`.

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.

Hint: The square root function returns a floating point number. Even for integers of a relatively modest size, such as 10^{16}, the difference between a perfect square and the same number plus or minus one disappears when we take the square root. The square root function can be useful, but you need to think about how to work around the limited precision.

Question 7 (4 Marks):

A sequence of numbers is called super-increasing if and only if every number in the sequence is strictly greater than the sum of all numbers preceding it in the sequence. The first element in the sequence can be any number.

For example, the sequences $1, 3, 5, 11, 21$ and $-2, 1, 2$ are both super-increasing; the sequence $1, 3, 5, 7, 19$ is increasing, but not super-increasing.

Write a function `super_increasing(seq)` that takes as argument a sequence of numbers and returns `True` if the sequence is super-increasing and `False` if it is not. A skeleton file called `Question_7.py` is provided. A testing program called `Test_Question_7.py` is also provided. The testing program will test the function `super_increasing` in the file `Question_7.py`.

- The function that you write must be named `super_increasing` and it must have one parameter.
- You can assume that the argument is a non-empty sequence of numbers.
- You should not assume that values in the sequence are unique or increasing.
- You should not make any assumption about the type of the argument other than that it is a sequence type; likewise, you should not make any assumption about the type of the numbers in the sequence, other than that they are numbers.
- The function must not modify the argument sequence.
- The function must return a value of type `bool`.

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

Question 8 (4 Marks) (COMP6730 ONLY):

The following function computes the largest absolute difference between any pair of values in a list of numbers:

```
def mpd(x):
    '''Returns the largest pair-wise difference in x.'''
    i = 1 # i is the distance between elements in the pair
    a = -1 # a keeps track of the maximum
    while i < len(x):
        y = i + 1
        while y < len(x):
            # check the pair in both directions to find the largest absolute difference:
            if x[y] - x[y - i] > a:
                a = x[y] - x[y - i]
            if x[y - i] - x[y] > a:
                a = x[y - i] - x[y]
            y = y + 1 # increment loop variable to make sure the loop ends
        i = i + 1
    # return the largest difference found:
    return a
```

Describe all the ways in which the function above can 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.)