Lab Test 1

General Instructions:

- Time allowed: 120 mins
- Total: 120 marks
- There are 6 questions
- Each Q in this Lab Test 1 is worth 20 marks
- Marks may be deducted for not following Python coding conventions, writing code that is hard to understand (e.g. unclear variable names or poor layout) or writing very inefficient code (e.g. unnecessary lines of code that do not improve clarity)
- You may write on this question paper, and you may keep this question paper
- For all questions, you may assume that if a function is expecting input parameters of a certain data type, then the function will always receive input parameters of that data type. For example, if a function is expecting to receive an integer, then the function will only ever be given integers.
- If you finish early, you are welcome to leave early. However, please raise your hand first, so that the time can be recorded. You are not allowed to resubmit code after you leave. If you submit after you leave the room, you will be given a total score of 0 marks.

Submission Instructions:

- Write your answer for each Q in a separate cell, but all in ONE .ipynb (Jupyter Notebook) file
- Write a comment at the top of each cell, indicating which Q, e.g. "`### Q1`"
- When marking your work, we will assume that all the code for a Q is in only ONE cell, and that no other cell has previously been run
- Submit only ONE .ipynb file
- The name of the zip file should be the same as your ID. For example, if your email address is **jason.chan.2024@smu.edu.sg**, then your file should be called **jason.chan.2024**.ipynb

Q1: Hypotenuse of right-angled triangle

Write a function called `calculate_hypotenuse()` which returns the length of the hypotenuse, which is the longest edge of a right-angled triangle. The function takes in two numbers as parameter, which represent the length of the other two edges (that are adjacent to the right-angle).

The length of the hypotenuse c is given by this formula:

$$c = \sqrt{a^2 + b^2}$$

where:

- c is the length of the hypotenuse
- a is the length of one of the edges of the right-angle triangle
- b is the length of one of the remaining edge of the right-angle triangle

For example:

- `calculate_hypotenuse(3,4)` should return 5
- `calculate_hypotenuse(2,7)` should return 7.280109889280518

Remember to write this code before you call your function, so that you can perform square root:

```
# Import math package so that we can perform square root
import math
```

Q2: Check temperatures

Write a function called `check_temps()` which takes three integers as parameters (not a list, not a tuple), which represents the measurement of 3 temperatures. The function should not return any values. Instead, the function should print messages on the screen (at least 1 message, possibly more, each on separate lines) depending on the following.

- If at least one of the temperatures is at least 100, then the function prints “boiling”.
- If at least one of the temperatures is 0 or less, then the function prints “freezing”.
- If all 3 temperatures are the same, then the function prints “all equal”.
- If none of the above messages are printed out, then the function prints “normal”.

For example:

- `check_temps(4,102,76)` should print to the screen:

  ```
  boiling
  ```

- `check_temps(4,102,-76)` should print to the screen:

  ```
  boiling
  freezing
  ```

- `check_temps(-76,-76,-76)` should print to the screen:

  ```
  freezing
  all equal
  ```

- `check_temps(76,83,4)` should print to the screen:

  ```
  normal
  ```

Q3: Match positions of strings

Write a function called `match_positions()`. The function takes in two strings as parameters and returns no values. Instead, the function should print the first string on one line, and then should print on the second line the characters that are in both the first and second parameter string, by first searching for the characters in the second string, in the order that they appear. The characters are printed in the same positions as where they appear in the first string, with spaces printed elsewhere to achieve the correct alignment. The function is case-sensitive.

For example:

- `match_positions('Programming with data','gmwDa')` should print:

```
Programming with Data
   g  m     w    Da
```

- `match_positions('Programming with data','gmwda')` should print:

```
Programming with Data
   g  m     w
```

  The "d" was not printed because it is lowercase d, but the first string contains only an uppercase "D". Note that we "a" is NOT on the second line, because we did not get to search for "a", since "d" was not found in the first string.

- `match_positions('Programming with data','gmP')` should print:

```
Programming with Data
   g  m
```

  The "P" was not printed because although it is in the first string, we have already passed that character when we were looking for the "g" and "m", so we continue to search through the first string, and only look at each character in the first string once.

- `match_positions('Programming with data','Xabc')` should print:

```
Programming with Data
```

  The second line that is printed is blank, because "X" is not found in the first string, so we don't the next character "a" in the second string is not even searched for in the first string.

Q4: Swap halves

Write a function called `swap_halves()`. The function takes in one integer as a parameter. The function should return a new integer, where the digits in the first half have been swapped with the digits in the second half.

For example:

- `swap_halves(123456)` should return 456123 (the first half of digits (123) swapped with the second half of digits (456))
- `swap_halves(12345)` should return 45312 (the first half of digits (12) swapped with the second half of digits (45), and the 3 was in the middle so it remained between the halves)

You may assume that the function is only ever given non-negative integers, and that no integer begin with a 0.

Your function should only use the integer data type. If you use any other data type (e.g. strings), then a 10 mark penalty will be imposed.

Q5: Print pattern

Write a function called `print_pattern()` which determines a pattern on the screen, depending on the value of the integer parameter n.

Some examples are given below.

If n is 5, your function should print the following:

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

If n is 8, your function should print the following:

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

Your solution should only contain one pair of nested loops (a loop inside a loop), and only one if-statement inside the inner-most loop, as taught in lesson 4. If you do not follow this requirement, then a maximum of 5 marks (not 20 marks) can be awarded for this question.

Q6: Increase integers

Write a function called `increase_integers()` which is given two parameters: a string and an integer.

The function finds all integers in the string (we define an integer as a series of digits in the string, even if they are preceded or followed immediately by another character), increases their value by the given integer, then puts them into a list.

The function returns this list of integers (not a list of strings).

You may assume that the string does not contain any full stops nor forward slashes, therefore all numbers will be integers and not decimal numbers nor fractions.

However, integers may begin with a series of zeroes.

For example:

- `increase_integers("abc 123 defgh 456 ij", 91)` should return the list [214, 547]
- `increase_integers("abc 123 defgh 00456 ij", 91)` should return the list [214, 547]
- `increase_integers("abc 123de fgh456 ij", 91)` should return the list [214, 547]

Your function must NOT use the split() method on strings. If you use the split() method, then a 10 mark penalty will be imposed.

There are no more questions