CSC108H Week 4 Lab

To earn your lab marks, you must actively participate in the lab.

1 Objectives

1. Use type `bool`
2. Explore type `str`

2 Driver and Navigator

driver: The person typing at the keyboard.

navigator: The person watching for mistakes and thinking ahead.

The rest of these instructions call you `s1` and `s2`. Pick which of you is which. `s1` should log in, start up Wing, and be the first driver.

3 Type `bool` and the Poisonous Potions Problem

Boolean logic employs two values, `True` and `False`, and three basic Boolean operators, `and`, `or`, and `not`. If `a` and `b` are Boolean variables:

`a and b`	Evaluates to `True` if **both** `a` and `b` are `True`, and `False` otherwise.
`a or b`	Evaluates to `True` if **at least one of** `a` and `b` is `True`, and `False` otherwise.
`not a`	Evaluates to the negated value of `a`; `True` if `a` is `False`, and `False` if `a` is True.

Similarly, "`a and b and c`" will evaluate to `True` only if all three variables have the value `True` and "`a or b or c`" will be `True` if any one or more of the three variables is `True`.

This part of your lab is about a logic puzzle involving four potions. You don't know which, if any, of them are poisonous, but you are given five hints:

1. At least one of the four potions is poisonous.
2. `p2` is not poisonous.
3. At least one of `p1` and `p3` is poisonous.
4. At least one of `p3` and `p4` is not poisonous.
5. Exactly one of the potions `p1`, `p3`, and `p4` is poisonous.

The potions puzzle can be represented in Python. Download `puzzle.py` from the Labs page of the course website.

We use four boolean variables, `p1` to `p4` to represent the potions; a True value indicates that the potion is poisonous, and a False indicates that it is not. The first hint has been translated into a Python function, `hint1`. If you have a guess at the solution to the puzzle, represented as four boolean values, it will return a boolean indicating whether or not your guess satisfies hint 1. For example, if you call `hint1(True, True, False, True)`, you are testing the guess that `p1`, `p2`, `p4` are poisonous, and `p3` is not. This satisfies hint 1, so the function will return True. But to solve the puzzle, a potential solution must satisfy all five hints. This is what function `solution` is for: if you give it a potential solution, function `solution` will return a boolean indicating whether it satisfies all five of the hints. But the program doesn't work because only one hint function has been written. Write the other four according to the descriptions in the table below, using `hint1` as an example of what the code should look like. Add the functions to `puzzle.py`, **switching driver and navigator after each one**.

`hint1(p1, p2, p3, p4)`	Return `True` if one or more of the potions is poisonous and `False` otherwise.
`hint2(p2)`	Return `True` if potion `p2` is not poisonous, and `False` otherwise.
`hint3(p1, p3)`	Return `True` if at least one of potions `p1` and `p3` is poisonous, and `False` otherwise.
`hint4(p3, p4)`	Return `True` if at least one of potions `p3` and `p4` is not poisonous, and `False` otherwise.
`hint5(p1, p3, p4)`	Return `True` if exactly one of the given potions is poisonous, and `False` otherwise.

The code for `hint5` is tricky. One approach is to start with an expression that evaluates to `true` if **at least** one potion is poisonous, and then add further clauses to eliminate the cases where **more than one** potion is poisonous.

Once you have added the rest of the hint functions, you can call the function `solution` with your proposed guess about which potions are poisonous.

You can try to solve the puzzle by reading the hints. Verify your answer by calling `solution` with your proposed solution (e.g., `solution(False, True, True, False)`). If your guess is correct, `solution` will return `True`. If `solution` returns `False`, either the proposed solution or one of the `hint` functions is wrong! Call `solution` again, and use the debugger this time to verify that each `hint` function works as you expect. Hint: `solution` has been written so that the result of each `hint` is saved in a variable that the debugger will let you inspect.

Once you have found the solution, show your TA your working puzzle code and then proceed to the next section.

4 Functions involving strings

Complete the following functions in a file called `w4.py` and verify your solutions by calling the functions with appropriate arguments. Switch the driver and navigator roles for each function. For this section, do not use any of Python's `str` methods. However, you may use `__builtins__` functions.

`longer(str, str)`	Given two strings, return the length of the longer string.
`earlier(str, str)`	Given two strings made up of lowercase letters, return the string that would appear earlier in the dictionary.
`count_letter(str, str)`	Given a string and a single character string, count all occurrences of the second string in the first and return the count.
`display_character(str, str)`	Given a string and a single character string, return a string containing the single character string the same number of times it appears in the first string.
`where(s,c)`	Given a string and a single character string, return the *position* of the *first* occurrence of the second string in the first. If the second string is not in the first, return `-1`. For example, `where("abc","b")` should return `1`.

5 Choosing test cases

In part 4, when you verified your function `longer`, how many different test cases did you try? How many possible pairs of strings *could* you call it with? The second number is probably way, way, WAY bigger than the first. So big that even if you tested your function all afternoon, you couldn't try them all. So how do we ever have confidence that our functions work? By testing them on very carefully chosen test cases that we believe each represent a large category of possible parameter values.

With this in mind, make a table of representative test cases for function `longer`. In each row, give a specific test case (a specific value for the first string argument and the second string argument), and then describe the purpose or significance of the test case, that is, what situation it is capturing. As an example, suppose we were testing function `lower`. One test case might be the string `"e"`, and the significance of this test case might be that it's a string of length one.

If you have time, go back and test your function with all of these cases. If your function passes all of them, great! If your function doesn't, that's great too; if you have a bug, it's much better to know about it than to think, incorrectly, that your code works.