As always, this lab requires you and your partner to take on distinct roles: driver (the person typing at the keyboard) and navigator (the person watching for mistakes, and thinking ahead).
The rest of these instructions call you s1 and s2. Pick which one is which. s1 should log in, start up Wing, and be the first driver.

1 Objectives

- Practice with nested loops and nested lists.
- Practice using the input function.
- Practice using while loops and lists.

Some of the sections of this lab handout are for your own practice and will not require you to submit any code on MarkUs. However, you should NOT skip them. You would be doing yourself a disservice if you do, as these sections are meant to ensure you have a good grasp on important course topics.

2 Nested Lists and Nested Loops

First, complete the tasks below in the shell. Do not submit any code you write in the Shell on MarkUs.

1. Assume list artists = [['Claude', 'Monet'], ['Lawren', 'Harris'], ['Vincent', 'van', 'Gogh']] 
where every element is a list of an artist’s names starting with their first name.
   • Write code that prints every string in artists in a separate line. Do not make any assumptions
     about the number of elements in each inner list. For example, this should display:
     Claude
     Monet
     Lawren
     Harris
     Vincent
     van
     Gogh
   • Write a statement that modifies the artists list by adding an artist with the first name 'Daphne'
     and last name 'Odjig' to it. Note that artists should be a list of list of str. (HINT: Use
     one list method.)

2. Given a list my_list = [['a', 'c', 'b'], ['f', 'e', 'd']], write code that sorts, in place, in
   ascending alphabetic order the elements of each sublist. (HINT: Use a list method.)
Then, write the following function in a file named lab6.py. You will need to submit this function.

```python
def one_long_string(my_list):
    """ (list of list of str) -> str
    Return a string that contains all the strings from my_list, in the order they appear there.
    """
    return ''.join(''.join(s) for s in my_list)

>>> one_long_string([['hi', 'there'], ['Lab6'], ['one', 'two', 'three']])
'hithereLab6onetwothree'
>>> one_long_string([['A', 'B', 'C'], ['D', 'E']])
'ABCDE'
```

## 3 Input and Output

In this section, you will practice getting input from the shell.

1. Write a statement that asks the user for their favourite food and stores it in a variable named `favourite`.

2. Write a statement that asks the user for their name. Then print a greeting using their name (example: 'Hello, Myrto!).

3. Switch so that s2 is the driver. Write a function called `convert_temp` that asks the user to input a temperature in Celsius, then prints the corresponding temperature in Fahrenheit. (The conversion is: temp_F = temp_C × 9/5 + 32.) The function does not take any arguments or return anything. You do not need to check to make sure the input is a valid float, you can assume it will be. You may use different wording for the prompt and for the output string.

   Example:

   ```
   >>> convert_temp()
   Please enter the temperature in Celsius: 23.5
   That is equivalent to 74.3 Fahrenheit.
   ```

## 4 While loops

All code that you write in this section, should use `while` loops. Of course, you could use a `for` loop but we are practicing `while` loops so you can get more comfortable with them. Save all your functions from this section in the `lab6.py` you created earlier. Using any `for-loops` for the functions in this section will earn you a grade of zero.

s1 and s2 should swap roles as driver and navigator in between every question.

1. Write a function `every_third` that takes a list as a parameter and returns a new list that contains every third element of the original list. Don’t use slice notation.

   Example:

   ```
   >>> every_third([10, 5, 8, 12, 4, 16])
   [8, 16]
   >>> every_third([1, 2])
   []
   ```
2. `sum_list(lst)` returns the sum of the numbers in the list of int it takes as a parameter. Use a while loop, not the `sum` function!

Example:

```python
>>> sum_list([10, 5, 8, 12, 4, 16])
55
```

5  Declaring Lab Partnership on MarkUs and Submitting

As in the previous labs, you will need to declare, on MarkUs, whether you are working alone or with a partner before you are allowed to submit. Note that you will need to repeat this process for every lab; this is to allow you to switch lab partners or work alone if you choose to.

Once your partnership has been declared, you are ready to submit your work. Although only one partner should submit the assignment, because you declared your partnership, we will know that both of you should get credit for the work – no matter who submits it.

For this lab, you need to submit the code you wrote for functions `one_long_string`, `every_third`, and `sum_list` in a file named `lab6.py` to MarkUs by Friday October 20th at 9:59pm. Your function names must be exactly as written above. Recall that Python is a case sensitive language. Your functions should not contain any print statements. You should have a single main clause with a call to doctest. All your docstring examples should pass doctest.

Please do NOT add any code/message etc. outside your function definitions, the `import doctest`, and the main block. Also, you need to ensure that you can run your code before you submit it on MarkUs.

6  Advice

If your code does not work as expected, then trace through its execution by hand or use the Memory Visualizer (see course website for a direct link). Then, identify where your program’s behaviour deviates from the one expected. Fix your code as needed, and repeat this process until you are confident that your code works as it should.

Also, for every single line of code that you write, you need to be clear on what purpose it serves.