Project: Parity

Project Description

Computers use what we call “parity” to check if data has become corrupt. Put simply, they check for odd or even numbers of bits, to determine if the data is still intact. Let’s represent this using a grid...

Create a 5x5 grid with different coloured cells or different characters:

X	O	O	X	X
X	X	O	X	O
O	O	X	X	O
O	O	O	O	X
X	X	X	O	X

Imagine each cell represents a bit, and is either on or off (aka zero or one).

Now add a new column and row to the grid, to make a 6x6 grid. This is simulating how computers add extra bits to the data.

Each new cell should ensure that there is an even number of purple squares or x’s and y’s in a row and column, like so:

X	O	O	X	X	X
X	X	O	X	O	X
O	O	X	X	O	O
O	O	O	O	X	X
X	X	X	O	X	O
X	O	O	X	X	X

Now ask someone to flip a cell without you watching:

X	O	O	X	X	X
X	X	O	X	O	X
O	O	X	X	O	O
O	O	X	O	X	X
X	X	X	O	X	O
X	O	O	X	X	X

To determine which cell was flipped, count the number of purple squares in each row and column:

X	O	O	X	X	X	4
X	X	O	X	X	O	4
O	O	X	X	X	O	3
O	O	X	O	O	X	2
X	X	X	O	O	X	4
X	O	O	X	X	X	4
4	2	3	4	4	4	

The flipped cell is the one at the cross section of the column and row with odd numbers of purple cells:

X	O	O	X	X	X	4
X	X	O	X	X	O	4
O	O	Me!	X	X	O	**3**
O	O	X	O	O	X	2
X	X	X	O	O	X	4
X	O	O	X	X	X	4
4	2	**3**	4	4	4	

For this project, you are going to simulate this process of adding new rows/columns and finding a flipped bit using Python.

Starter Code

Some starter code has been provided to help you get started. `parity.py` contains several functions, most which are to be completed (by you!). To help you understand what each function is meant to do, they each contain something called a "docstring".

Also provided is a `run_tests.py` file, and a `tests` directory. You won't actually be running the `parity.py` file, instead, the tests will run it for you. More on this later...

Docstrings

A docstring is a comment that describes exactly what the function does, including it's arguments and what it should return.

Let's look at this `read_csv_file` function as an example:

```
def read_csv_file(file_name):
      '''Reads a csv file and returns the data as a list.
```

```
    Args:
        file_name: a string representing the path and name of a csv file.

    Returns: a list.
    '''
    pass
```

The first line in the docstring gives an overview of the function. In this case, this function will read a csv file and return a list.

"Args" are the arguments/parameters that the function accepts. In this case, this function accepts only one argument called file_name, which is a string representing the path to a csv file, e.g. `'data/some_parity_data.csv'`.

"Returns" is what the function is meant to return when the function is complete. In this case, it should return a list.

So this docstring is telling us that this function should read a csv file, save the data to a list, and return that list.

Tests

More often than not, when working on a programming project you will be required to run your code against tests. Tests help you to ensure that your code has the expected behavior and is correct for a variety of different inputs.

As mentioned above, the starter code includes a `run_tests.py` file, and a `tests` directory. To test your code, run this `run_tests.py` file. This will give a lot of output, and that's not a bad thing and it's nothing to worry about! If you look in the `tests` directory, you'll see a bunch of different files. Each of these files have a bunch of tests for a single function in `parity.py`. Since you are running this file for the first time, all of those tests are going to fail, because you haven't written any code to make those functions work yet. As you complete more and more of the functions, less and less tests will fail, and that output will eventually be reduced to a few lines.

In the meantime, let's understand how to read that output. Here's a snippet of the output:

```
======================================================================
FAIL: test_flip_cell (tests.test_flip_cell.FlipCellTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/hayley/Projects/she-codes/Python-content/Python_P.D.1/starter/tests/test_flip_cell.py",
line 28, in test_flip_cell
       self.assertListEqual(result_grid, self.small_grid_flipped)
AssertionError: First sequence is not a list: None


----------------------------------------------------------------------
```

The white lines are the ones we care about:

- `FAIL: test_flip_cell (tests.test_flip_cell.FlipCellTests)`
    - This is the test that failed. If I wanted to see exactly what that test is doing, then I could look at the function called `test_flip_cell` in the `tests/test_flip_cell.py` file. From the file name and name of this test, I know that the function it tried to call in my `parity.py` file was the `flip_cell()` function.

- `AssertionError: First sequence is not a list: None`
    - This is what the error was. In this case, it is telling me that when the test called my `flip_cell()` function in the `parity.py` file, my function returned `None` but the test was expecting a list.

To make this a little less daunting, you might prefer to only run a few tests at a time. You can do this by looking for the following lines in `run_tests.py`:

```
runner.run(unittest.TestSuite((unittest.makeSuite(LoadGridTests))))
runner.run(unittest.TestSuite((unittest.makeSuite(AddColumnTests))))
runner.run(unittest.TestSuite((unittest.makeSuite(AddRowTests))))
runner.run(unittest.TestSuite((unittest.makeSuite(FindFlippedCellTests))))
runner.run(unittest.TestSuite((unittest.makeSuite(FlipCellTests))))
runner.run(unittest.TestSuite((unittest.makeSuite(IntegrationTests))))
```

Each line corresponds to a function in `parity.py`, so if you wanted to just start with the function for flipping a cell, you could comment out all the other tests to make the output shorter and easier to read:

```
# runner.run(unittest.TestSuite((unittest.makeSuite(LoadGridTests))))
# runner.run(unittest.TestSuite((unittest.makeSuite(AddColumnTests))))
# runner.run(unittest.TestSuite((unittest.makeSuite(AddRowTests))))
# runner.run(unittest.TestSuite((unittest.makeSuite(FindFlippedCellTests))))
runner.run(unittest.TestSuite((unittest.makeSuite(FlipCellTests))))
# runner.run(unittest.TestSuite((unittest.makeSuite(IntegrationTests))))
```

Project Requirements

Your job is to make all of the tests pass!

Do not make any changes to anything in the `tests` directory or `run_tests.py` file (aside from commenting some tests while you are working). The contents of your `parity.py` file will be copied into a new directory and run against the original tests, so if you have changed any tests in your own repo, your code will likely fail upon submission.

Submission

Please submit the following:

- A text file containing a link to your project repository.
- Include a screenshot of your code passing all of the tests in Terminal/Powershell.