🤔 Project Hints

These sections use the “Project Hints” zip file provided on Skill Social.

Testing

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 correct for a variety of different inputs.

Open the `testing` folder in a code editor and the terminal. Here you will find two files:

- `simple_functions.py` - This contains four small functions.
- `run_tests.py` - This contains several tests (one or two tests per function).

You’ll find that when you run `simple_functions.py`, nothing will happen. But when you run, `run_tests.py` you’ll get something that looks like this:

```
Running Tests...

.F.F.E
======================================================================
ERROR: test_function_4_correct_output_3 (__main__.Part1Tests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "run_tests.py", line 42, in test_function_4_correct_output_3
    result_output = simple_functions.function_4(9.00, "3.2")
                                                                  File
"/Users/hayleyavw/Projects/plus-resources/project_2/week_5/project_hints/mentors/testin
g/simple_functions.py", line 14, in function_4
    return x + y
TypeError: unsupported operand type(s) for +: 'float' and 'str'

======================================================================
FAIL: test_function_2_correct_output (__main__.Part1Tests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "run_tests.py", line 19, in test_function_2_correct_output
    self.assertEqual(result_output, expected_output)
AssertionError: 'Broken output for function 2.' != 'Correct output for function 2.'
- Broken output for function 2.
? ^ ^^ ^
+ Correct output for function 2.
? ^^ ^ ^^



======================================================================
FAIL: test_function_3_correct_output_2 (__main__.Part1Tests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "run_tests.py", line 29, in test_function_3_correct_output_2
    self.assertEqual(result_output, expected_output)
AssertionError: 'x = 3' != 'x = 9.00'
```

```
- x = 3
+ x = 9.00



======================================================================
FAIL: test_function_4_correct_output_2 (__main__.Part1Tests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "run_tests.py", line 39, in test_function_4_correct_output_2
      self.assertEqual(result_output, expected_output)
AssertionError: '-10' != -1

----------------------------------------------------------------------
Ran 7 tests in 0.002s

FAILED (failures=3, errors=1)
```

The final two lines in the output tell us that 7 tests were run, but there were 3 failures and 1 error.

Let's tackle the error first:

```
======================================================================
ERROR: test_function_4_correct_output_3 (__main__.Part1Tests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "run_tests.py", line 42, in test_function_4_correct_output_3
      result_output = simple_functions.function_4(9.00, "3.2")
                                                                  File
"/Users/hayleyavw/Projects/plus-resources/project_2/week_5/project_hints/mentors/testin
g/simple_functions.py", line 14, in function_4
      return x + y
TypeError: unsupported operand type(s) for +: 'float' and 'str'
```

The first line tells us that there was an error in `test_function_4_correct_output_3()`. The final line in the error output tells us that we had a type error, we tried to add a float and a string.

Sure enough, by looking at the `run_tests.py` file, we can see that `function_4()` was called with the parameters `9.00` and `3.2` - a float and a string!

To fix this error, you need to amend `function_4()` to handle adding floats and strings. (Hint: the test function also shows the expected output!).

Now let's take a look at a failed test:

```
======================================================================
FAIL: test_function_2_correct_output (__main__.Part1Tests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "run_tests.py", line 19, in test_function_2_correct_output
      self.assertEqual(result_output, expected_output)
AssertionError: 'Broken output for function 2.' != 'Correct output for function 2.'
- Broken output for function 2.
? ^ ^^ ^
```

```
+ Correct output for function 2.
? ^^ ^ ^^
```

The four lines tell us that `test_function_2_correct_output()` failed. The last four lines tell us why. The `^` characters highlight the difference between the output the test expected, and output the function actually provided. To fix this error, you will need to amend `function_2()` to return the correct string.

Reading JSON Files

Data comes in many different formats, so far we have looked at plain text files and CSVs. Now let's have a very quick look at the JSON format. I've provided a json file with quiz data:

```
{
    "quiz": {
      "One": {
            "question": "Our goal at She Codes is to get x women in tech by y.",
            "options": [
                  "x=50000, y=2030",
                  "x=50000, y=2025",
                  "x=100000, y=2025",
                  "x=100000, y=2030"
            ],
            "answer": "x=100000, y=2025"
      },
      "Two": {
            "question": "What percentage of tech roles in Australia are currently held
by women?",
            "options": [
                  "29%",
                  "23%",
                  "14%",
                  "32%"
            ],
            "answer": "29%"
      },
      "Three": {
            "question": "Which is the tutorial we DON'T have on our website?",
            "options": [
                  "HTML/CSS",
                  "Python",
                  "Django",
                  "Hardware"
            ],
            "answer": "Hardware"
      },
      "Four": {
            "question": "What kind of food is always served at She Codes events?",
            "options": [
                  "Brownies",
                  "Sandwiches",
                  "Cupcakes",
```

```
                "Chocolate"
            ],
            "answer": "Cupcakes"
        }
    }
}
```

You might notice that it looks pretty similar to a dictionary in Python. In fact, it is so similar, that there is even a way in Python to load the file directly into a dictionary:

```
import json
with open("path/to/json/file.json") as json_file:
      json_data = json.load(json_file)
```

Using the above code snippet as a starter, read the quiz data, and output the quiz questions in the following format:

```
Question One: Our goal at She Codes is to get x women in tech by y.
      x=50000, y=2030
      x=50000, y=2025
      x=100000, y=2025
      x=100000, y=2030

Question Two: What percentage of tech roles in Australia are currently held by women?
      29%
      23%
      14%
      32%

Question Three: Which is the tutorial we DON'T have on our website?
      HTML/CSS
      Python
      Django
      Hardware

Question Four: What kind of food is always served at She Codes events?
      Brownies
      Sandwiches
      Cupcakes
      Chocolate
```