1. Write unit tests to ensure correct functionality of Python code.

A unit test is a test that verifies the functionality of a single line of code, typically modularised as a function. Regression testing relies heavily on unit tests to guarantee that the code is stable and continues to operate as expected after modifications are made (Chng, 2022).

Pykes (2022) states that the general reason why unit testing is important is that before allowing code to enter a production environment, developers must confirm that it complies with quality requirements. Unit tests, however, are essential for several reasons, as unit testing during software development aids in the early detection of code defects, preventing the need for expensive corrections later. Additionally, it offers ongoing documentation for thorough comprehension and boosts developers' trust in the code's functionality.

Python's standard library includes a module called "unittest," which offers tools for testing code. Unit testing ensures that specific parts of a function's behaviour are correct, making integrating them with other parts much easier (Aviani, 2019).

According to Aviani (2019), a "test case" is a set of unit tests that prove a function works as intended in a full range of situations in which it may find itself. Test cases should consider all possible types of input a function could receive from users and include tests to represent each situation.

To write tests, create a test file, import the unittest module, and define the testing class that inherits from the unittest. TestCase. Then, write a series of methods to test all cases of your function's behaviour. A line-by-line explanation of the code can be found below, based on the ideas of Aviani (2019):

```
import unittest

# Here is the code that requires testing.
def add_numbers(a, b):
    return a + b

# A test case class that inherits from unittest.TestCase
class TestAddNumbers(unittest.TestCase):

    # A test method that starts with the word "test_"
    def test_add_numbers(self):
        # Test case inputs and expected output
        a = 2
        b = 3
        expected_result = 5

        # Call the function being tested
        result = add_numbers(a, b)

        # Assert that the result matches the expected output
        self.assertEqual(result, expected_result)

# Run the tests
if __name__ == '__main__':
    unittest.main()
```

I had to ensure the add_numbers() function worked correctly. To do so, I create a test case class named TestAddNumbers that inherits from unittest.TestCase. In this class, I defined a test method called test_add_numbers(), where we set up the add_numbers() function inputs, a and b, and the expected output, expected_result. Next, we call the add_numbers() function and store the result in the result variable. To verify whether the result matches the expected_result, we use the assertEqual() method provided by the unittest.TestCase. If the result and expected_result match, the test passes; otherwise, it fails. To execute the tests, we check if the module is running as the primary program using if __name__ == '__main__':. If it is, we call the unittest.main() to run the tests.

2. Run pylint against a Python script to demonstrate stylistic correctness.

Using Pylint as a Python developer can be quite beneficial. Powerful static code analyser Pylint extensively examines your code for mistakes, upholds coding standards, spots code smells, and makes refactoring recommendations. Additionally, Pylint can complete all of these tasks without executing your code! It's a crucial tool that functions flawlessly with Python 2 and 3, making it a priceless resource for developers of all skill levels (PyPI, N.D).

To run Pylint, we need to install it first using pip on the command line:

```
pip install pylint
```

Then, we can run it on our script using this command:

```
pylint my_script.py
```

And we should see an output like this:

```
(base) C:\Users\hcham\PycharmProjects\pythonTest>cd C:\Users\hcham\Desktop\Essex\2. Object
Oriented Programming May 2023\Unit 10

(base) C:\Users\hcham\Desktop\Essex\2. Object Oriented Programming May 2023\Unit 10>pylint
my_test.py
************* Module my_test
my_test.py:18:0: C0304: Final newline missing (missing-final-newline)
my_test.py:1:0: C0114: Missing module docstring (missing-module-docstring)
------------------------------------------------------------------
Your code has been rated at 7.78/10
```

In my example, pylint found two stylistic errors and scored 7.78/10, meaning my code does not comply with the PEP 8 style guide. As stated by NeuralNine (2023), to make it right, I will need to add the missing 'Final newline missing (missing-final-newline)' and Missing module docstring (missing-module-docstring) to my script.

3. Document code for release to stakeholders.

Creating comprehensive software documentation is crucial because it assists users in understanding how to use your software, provides developers and other technical stakeholders with information regarding the technical aspects of your software, and guarantees a seamless and consistent software development process. Moreover, well-crafted software documentation can enhance your software's overall quality and user experience (Oragui, 2023). Furthermore, according to Oragui (2023), producing software documentation benefits users, developers, and technical stakeholders by improving collaboration, efficiency, quality, and user experience by providing clear and consistent information about the software.

According to Brown (2022), every project is distinct, and varied documentation is required depending on the stage, the team's requirements, and other factors. Usually, a project's documentation is broken down into five phases:

1. At the start of a project, it's crucial to create documentation that outlines its purpose, scope, timelines, stakeholders, and roles. This helps assign the right team members and lays the groundwork for future decisions. Key documents include a project proposal, charter, and business case.
2. Create a high-level plan with milestones and timelines during project planning. This helps identify tasks, dependencies and achieve goals. Documentation includes a project plan, business requirements, product requirements, communication plan, risk management plan, quality management plan, procurement plan, and acceptance test plan.
3. Document progress and challenges during project execution, including deliverables and tracking documents. Hold project meetings to monitor progress and address any issues. Track issues and record information on their cause, status, resolution, and plan to manage them. Create risk control, change requests, quality assurance, and acceptance reports to summarise the project's progress and final deliverables.
4. Monitoring and controlling are critical aspects of project management. It ensures that the project is on track by reviewing its progress compared to the project plan. This phase co-occurs with the execution phase, where the team creates deliverables and updates tracking documents. Key documentation includes team member and contractor status reports and a product acceptance form to confirm that the deliverable meets all required standards and requirements.
5. During project closure, it's important to document and validate completion, reflect on successes and failures, and gather recommendations for future projects. Key documents include a final report, closure checklist, summary, and lessons learned document. These provide insights into best practices, factors for success or failure, and valuable information for future projects.

References:

Chng, Z.M. (2022). *A Gentle Introduction to Unit Testing in Python*. [online] Machine Learning Mastery. Available at: https://machinelearningmastery.com/a-gentle-introduction-to-unit-testing-in-python/.

Pykes, K. (2022). Pytest Tutorial: A hands-on guide to unit testing. [online] Available at: https://www.datacamp.com/tutorial/pytest-tutorial-a-hands-on-guide-to-unit-testing.

Aviani, G. (2019). *An Introduction to Unit Testing in Python*. [online] Available at: https://www.freecodecamp.org/news/an-introduction-to-testing-in-python/ [Accessed 9 Jul. 2023].

PyPI (N.D.). *pylint: python code static checker*. [online] PyPI. Available at: https://pypi.org/project/pylint/.

NeuralNine (2023). *Mastering Python Code Quality with Pylint*. [online] Available at: https://www.youtube.com/watch?v=RqdhVaX50mc.

Oragui, D. (2023). *Software Documentation Best Practices [With Examples]*. [online] Available at: https://helpjuice.com/blog/software-documentation.

Brown, J. (2022). *Project Documentation 101: A Powerful Way to Share Knowledge*. [online] Available at: https://helpjuice.com/blog/project-documentation.