Writing Test Cases in Python

Your Name

April 19, 2024

1 Introduction

This document demonstrates how to write and run test cases in Python using various techniques.

1.1 Naming Conventions

Test files should follow the naming convention `test_<module_name>.py`. For example, if your module is named `calculator.py`, your test file should be named `test_calculator.py`.

1.2 Importing Modules

In your test file (`test_<module_name>.py`), import the necessary modules and functions/classes from your main Python file (`<module_name>.py`) that you want to test. For example:

```
from calculator import add_numbers
```

1.3 Writing Test Cases

Use a testing framework like `unittest` or `pytest` to write your test cases. Here's an example using `unittest`:

```
import unittest
from calculator import add_numbers

class TestCalculator(unittest.TestCase):
    def test_add_numbers(self):
        result = add_numbers(3, 5)
        self.assertEqual(result, 8)  # Assert that the result is equal to the expected value
```

1.4 Running the Tests

After writing your test cases, run them using a test runner. For `pytest`, you would run the tests with:

```
pytest test_calculator.py
```

To run a specific function in the test file

```
pytest test_Math.py::test_Add
```

```
pytest -k "add" test_Math.py                #runs all functions that has 'Add' init
pytest -k "add or string" test_Math.py      # runs 'Add' or 'string' init
```

to run all the test cases and return a explanation if a case fails

```
pytest -v -x
```

2 Complex Test Example

```python
import subprocess

def test_user_input_output():
    user_inputs=["Abdulla",22]
    # Run the main program and capture the output
    process = subprocess.Popen(['python', '1.py'], stdin=subprocess.PIPE, stdout=subprocess.PIPE,
        stderr=subprocess.PIPE, text=True)
    out, err = process.communicate(input='Abdulla\n22\n')  # Simulate user input for name and age
    i=0
    for item in out:
        print(item,end="")
        if(item == ':'):
            print(user_inputs[i])
            i+=1
            print("\n")

if __name__ == "__main__":
    test_user_input_output()
```

1. **Importing subprocess:**
   - The subprocess module is imported to run external processes (in this case, running another Python script).
2. **Test Function `test_user_input_output`:**
   - This function simulates user input and captures the output of another Python script (1.py).
3. **User Inputs Simulation:**
   - User inputs are defined as "Abdulla" (name) and 22 (age) to simulate user interaction with the program.
4. **Running the Main Program:**
   - The subprocess.Popen function is used to run the main program (1.py) and capture its output.
5. **Communicating User Input:**
   - The communicate method is used to send user input ("Abdulla22") to the program being tested.
6. **Processing Output:**
   - The output received from the program is processed character by character in a loop.
   - If a colon (:) is encountered in the output, it is assumed to be a prompt for user input, and the corresponding user input is printed.
7. **Test Execution:**
   - The `test_user_input_output` function is executed if the script is run directly (`__name__ == "__main__"`).