Testing and Debugging

Guttag Chapter 8

Goal

Learn terminology associated with testing and debugging from Chapter 8

Discuss testing and debugging strategies

Examine structure of test code

Terminology for Testing and Debugging

Terms: Testing and Debugging

"**Testing** is the process of running a program to try and ascertain whether it works as intended."

"**Debugging** is the process of trying to fix a program that you already know does not work as intended."

~Guttag Chapter 8

Terms: Black-Box Testing

Definition

- Testing a function based on documentation only
- Aim is to determine if it works as intended
- The inside of the function is an unknown (black-box)

Example

```
def is_prime(x: int) -> bool:
    """Return True if x is prime; False otherwise"""
```

Q: What would you check?

A: Something prime and not prime

Terms: Glass-Box Testing

Definition

- Testing a function based on implementation
- Aim is to determine if it works as intended
- The inside of the function is known and clear (glass-box)

Example

```
def is_prime(x: int) -> bool:
    """Return True if x is prime; False otherwise"""
    if x <= 2:
        return False
    for i in range(2, x):
        if x%i == 0:
            return False
    return True
```

Q: What would you check?

A: input 2 and less, input larger than 2, because 2 is a boundary!

Terms: Unit Testing

Definition

- Tests for **individual functions**
- Aim is to determine if each individual function works as intended

Integration Testing

Definition

- Tests for **groups of functions**
- Aim is to determine if a group is working together as intended

Functional Testing

Definition

- Tests for **entire program**
- Aim is to determine if the entire program works as intended

Terms: Test Stubs

Definition

- code used to simulate part of a program for (unit) testing
- in the code, input must be specified
- in the code, expected output must be specified
- in the code, function call must be made to get the actual output
- in the code, the expected and actual output must be compared

Example for testing `is_prime`

- input_value = 2
- expected_output_value = True
- actual_output_value = is_prime(input_value)
- assert expected_output_value == actual_output_value

Terms: Pytest

Definition

- Testing framework
- Runs test functions implemented within test files in a `tests` directory!

Example from Engineering Labs

- poetry run task test

Terms: Pytest

```
egraber ~/Documents/Teaching/S2024-CMPSC101/fibonacci-algorithms-starter/fibonaccicreator % 🦚🦚🦚 poetry run task test
============================== test session starts ==============================
platform darwin -- Python 3.12.2, pytest-7.4.4, pluggy-1.4.0
rootdir: /Users/egraber/Documents/Teaching/S2024-CMPSC101/fibonacci-algorithms-starter/fibonaccicreator
collected 0 items / 1 error

==================================== ERRORS =====================================
_________________________ ERROR collecting tests/test_fibonacci.py _________________________
tests/test_fibonacci.py:13: in <module>
    from fibonaccicreator import fibonacci
fibonaccicreator/fibonacci.py:9: in <module>
    def fibonacci_recursivelist(number: int) -> List[int]:
E   NameError: name 'List' is not defined
============================ short test summary info ============================
ERROR tests/test_fibonacci.py - NameError: name 'List' is not defined
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! stopping after 1 failures !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!! Interrupted: 1 error during collection !!!!!!!!!!!!!!!!!!!!!!!!!!
================================ 1 error in 1.36s ================================
egraber ~/Documents/Teaching/S2024-CMPSC101/fibonacci-algorithms-starter/fibonaccicreator % 🦚🦚🦚
```

Can you define?

- black-box testing
- glass-box testing
- unit testing
- integration testing
- functional testing
- test stubs
- Pytest

Testing and Debugging Strategies

Testing Strategies

- Categorize possible inputs and test one input from each category
- Test boundaries or edge cases that are evident in the code
- Be cautious with loops
  - test an input that would not loop
  - test an input that would loop once
  - test an input that would loop more than once
- Be caution with conditional statements
  - test an input that can get into each branch
- Test things a user will be likely to try

Debugging Strategies

- Read errors slowly
- Describe what you did
- Describe the outcome and why it was unexpected
- Hypothesize about potential solutions
- Check your hypothesis
- Use printing or debuggers to see what the code is doing
- Isolate the problem area
- Use commit messages to appropriately tag changes if reversion is needed

- Try experimenting in Colab to reproduce and study the bug

Guttag

8.2.3

- *Look for the usual suspects*. Have you
  - Passed arguments to a function in the wrong order?
  - Misspelled a name, e.g., typed a lowercase letter when you should have typed an uppercase one?
  - Failed to reinitialize a variable?
  - Tested that two-floating point values are equal (==) instead of nearly equal (remember that floating-point arithmetic is not the same as the arithmetic you learned in school)?
  - Tested for value equality (e.g., compared two lists by writing the expression `L1 == L2`) when you meant to test for object equality (e.g., `id(L1) == id(L2)`)?
  - Forgotten that some built-in function has a side effect?
  - Forgotten the `()` that turns a reference to an object of type `function` into a function invocation?
  - Created an unintentional alias?
  - Made any other mistake that is typical for you?

Structure of Test Code

Test Code

- function name starts with test_
- hard coding is used!
- assert statement is used!
- the test function does not take parameters (for now)

```
def test_short_not_palindrome_word_reverse():
    """Ensure that a short word of "taylor" does not work correctly with reverse."""
    # TODO: hard code the input
    # TODO: hard code the expected answer
    # TODO: call the function under test with the hard coded input
    # TODO: make sure the return value is saved!
    # TODO: use the assert syntax compare the hard coded expected answer with
    # the returned answer
```

Example

```python
def pal_rev(word: str) -> bool:
  """Determine if a given word is a palindrome."""
  return word == word[::-1]
```

```python
def test_short_not_palindrome_word_reverse():
    """Ensure that a short word of "taylor" does not work correctly with reverse."""
    test_word = 'taylor' # hard coded
    expected_result = False # hard coded
    result = pal_rev(test_word)
    assert result == expected_result
```

Explore Code

Summary

Tests help to reveal bugs

Debugging is the process of fixing bugs

Test code sets up and runs known inputs under controlled conditions