Chapter 20 Unit Testing

20.1 Introduction: Unit Testing

- **Unit test**: Code that tests a single function to ensure it works properly.
- Purpose:
    - Catch errors early.
    - Verify correctness of small parts before combining them into larger systems.
    - Improves reliability and confidence in code.

20.2 Checking Assumptions with assert

- **assert statement**: Verifies that a boolean condition is true.
    - If condition is false, program stops with an error message.
- Helps detect errors quickly during development.

20.2.1 Designing Defensive Functions

- Functions should be written to prevent misuse.
- Example: Checking inputs before performing calculations.

20.2.2 The assert Statement

Syntax:

```
assert condition, "Error message"
```

-

Example:

```
assert n >= 0, "n must be non-negative"
```

-

20.2.3 More on assert and Preconditions

- **Precondition**:
    - A boolean condition the caller must satisfy before calling the function.
    - Example: sqrt(x) requires x >= 0.
- **Postcondition**:
    - A guarantee about what the function will return/output if preconditions are met.
    - Example: sqrt(x) guarantees the result squared will equal x.

20.3 Testing Functions

- Purpose: Verify functions work under all expected conditions.
- Benefits:
    - Identifies incorrect logic.
    - Encourages modular design.

20.3.1 Automated Unit Tests

- Repeatedly run tests automatically.
- Saves time compared to manual testing.

20.3.2 Automated Unit Tests with assert

- Tests can be written as small scripts using assert to confirm expected behavior.

20.3.3 Unit Tests can have Bugs

- Tests themselves may be incorrect.
- Importance of reviewing both the code and the test.

20.4 Designing Testable Functions

- Write functions that are simple, predictable, and modular.
- Functions should clearly state expectations and guarantees.

20.4.1 Design by Contract

- Formal approach to writing functions with:
    - **Preconditions** (caller's responsibility).
    - **Postconditions** (function's responsibility).
- Benefits:
    - Reduces misunderstandings between programmer and user.
    - Encourages precise function design.

20.5 Writing Unit Tests

- Good tests anticipate possible errors and edge cases.
- Test normal values, extreme values, and invalid inputs.

20.6 Test-First Development

- Tests are written *before* writing the function.
- Encourages clear planning and avoids vague function definitions.

20.6.1 Benefits of Test-First Development

- Ensures clarity of function purpose before coding.
- Reduces debugging time.

20.7 Testing with pytest

- **pytest**: A Python library for automated testing.
- Allows functions to be tested systematically with detailed feedback.

20.7.1 Organizing pytest Functions

- Test functions usually start with test_.

Example:

```
def test_is_even():
  assert is_even(4) == True
```

- **Using pytest**
- Run tests with pytest command.
- Finds and runs all test_ functions automatically.

20.7.3 Understanding pytest Failure Reports

- Provides detailed output when a test fails.
- Shows:
    - Which test failed.
    - What was expected vs. what was returned.