Test Plan and Scripts - Fourward Programming Language

1. Testing Strategy

1.1 Testing Approach

- Manual testing using `.fwd` example files
- Functional validation of lexer, parser, and interpreter
- Focused testing on language features and control flow

1.2 Planned Testing Types

- Functional Testing (core language features)
- Error Handling Testing (syntax and runtime)
- Usability Testing (readability and documentation clarity)

---

2. Test Environment

2.1 Setup Requirements

- Python 3.8+ installed
- No additional dependencies required
- Terminal or command-line access

2.2 Test Configuration

- Place test files in the root directory
- Execute using:

```
python3 fourward_interpreter.py examples_compatible.fwd
```

---

3. Test Cases

3.1 Language Features Tested

- Variable declaration and assignment
- Arithmetic and string operations
- `if` / `else` conditional execution
- `while` loop behavior
- Built-in `print()` and `input()` functions
- Comment handling

3.2 Error Handling Tests

- Missing semicolons
- Undefined variables
- Division by zero
- Incorrect token sequences
- Unsupported keywords (e.g., `function`)

---

4. Test Scripts

4.1 Manual Test Scripts (Example-Based)

Tests are written as `.fwd` programs. Example:

```
# Variable and arithmetic test
let x = 10;
let y = 5;
let z = x + y * 2;
print("Result is " + z);
```

4.2 Future Automated Tests (Planned)

```
def test_variable_declaration():
    code = "let x = 5;"
    # Tokenize, parse, evaluate, and assert variable in environment
    pass

def test_if_else():
    code = "let x = 10; if (x > 5) { print('yes'); } else { print('no');
}"
    pass
```

---

5. Test Execution

5.1 Execution Methods

- Run `.fwd` test files manually via CLI
- Observe interpreter output
- Check for correct result or error handling

5.2 Result Evaluation

- Visual confirmation of output
- Error trace validation for invalid programs

---

6. Test Coverage

6.1 Current Goals

- Validate each feature via examples
- Cover core grammar rules
- Ensure all branches in control flow are exercised

6.2 Future Tools (Optional)

- Use `coverage.py` if Python unit tests are added
- Extend with `pytest` for function-level testing

---

7. Bug Tracking

7.1 Reporting and Tracking

- Use GitHub Issues for tracking bugs
- Manual labeling of errors (runtime, syntax, etc.)

7.2 Regression Testing

- Re-run fixed `.fwd` test cases
- Maintain a test suite of previously broken features

---

8. Test Documentation

8.1 Reporting

- Record test results using comments or README notes
- Log interpreter crashes or unexpected behavior

8.2 Maintenance

- Keep `.fwd` test files up-to-date
- Add new tests as features are added
- Maintain alignment with evolving syntax and grammar
