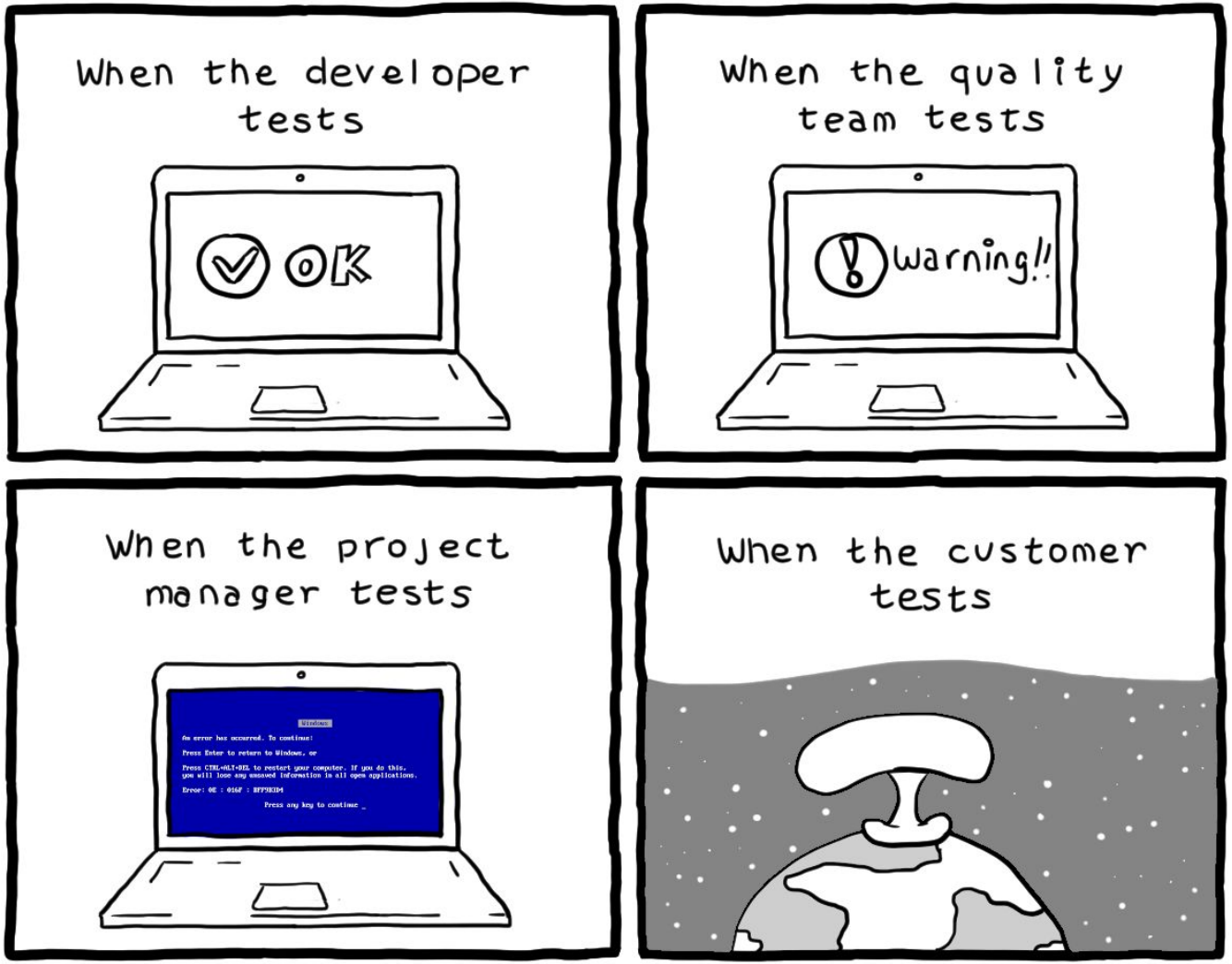

When the developer tests
OK
When the quality team tests
Warning!!
When the project manager tests
Windows
An error has occurred. To continue:
Press Enter to return to Windows, or
Press CTRL+ALT+DEL to restart your computer. If you do this, you will lose any unsaved information in all open applications.
Error: 0E : 016F : BFF9B3D4
Press any key to continue _
When the customer tests

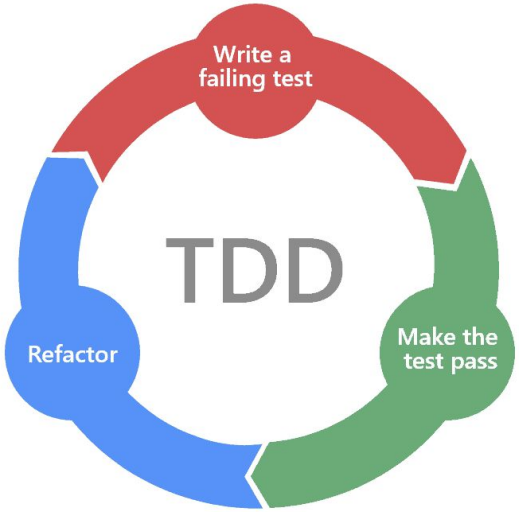
Motivation

- Old: Write code → test if it works
- Problem?
  - Follows programmer's mental model
  - Alternative: Employ test teams
  - Another alternative: Use **test driven development (TDD)**
    - Automated testing
    - Follows a Red-Green-Refactor cycle
- Why test first?
  - Confidence
  - Design focus
  - Living documentation

Recall: Refactoring

- Refactoring is the process of ***improving*** the internal structure of code **without changing its external behavior**
- Makes code cleaner, more readable, easier to maintain, or more efficient
  - External behavior = what tests check (inputs/outputs, observable effects)
  - Internal structure = naming, organization, duplication, complexity, performance
- Examples
  - Readability (e.g., naming, single responsibility)
  - Remove duplication
  - Reorganize code / structure
  - Improve control flow (e.g., guard exceptions, minimize loops)
  - Improve data and object handling (e.g., constants, globals, encapsulation)
  - Improve performance (e.g., caching, changing data structures)

Test Driven Development

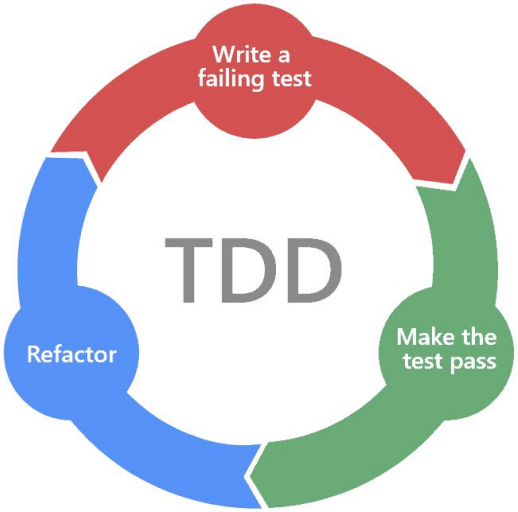


- Given a feature:

test before code

- Easy to do for unit testing or component testing

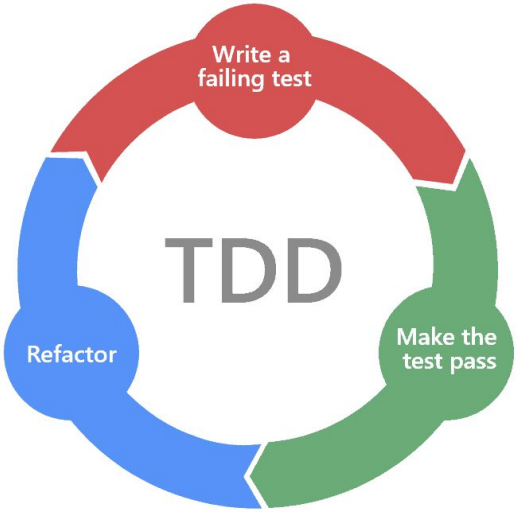
Test Driven Development



- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?

- Easy to do for unit testing or component testing

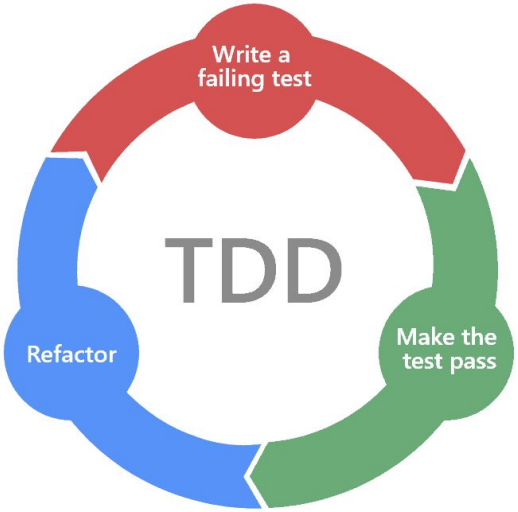
Test Driven Development



- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases

- Easy to do for unit testing or component testing

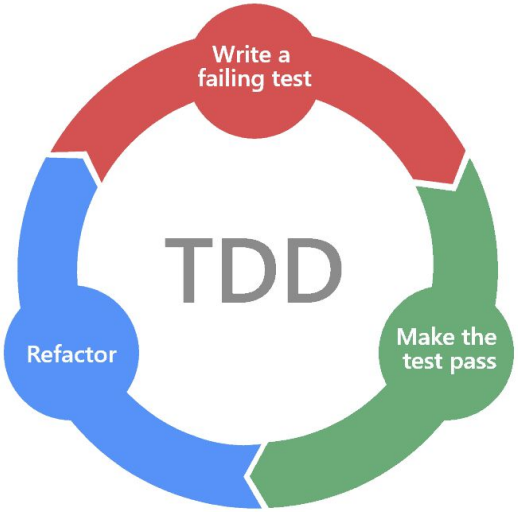
Test Driven Development



- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases
  - Let the tests fail

- Easy to do for unit testing or component testing

Test Driven Development

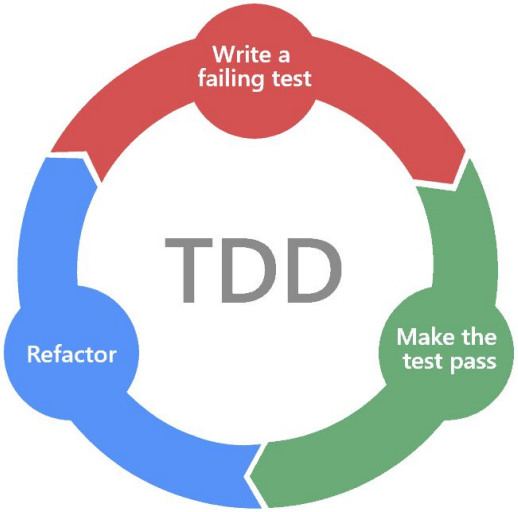


- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases
  - Let the tests fail
  - Write the code to pass each test

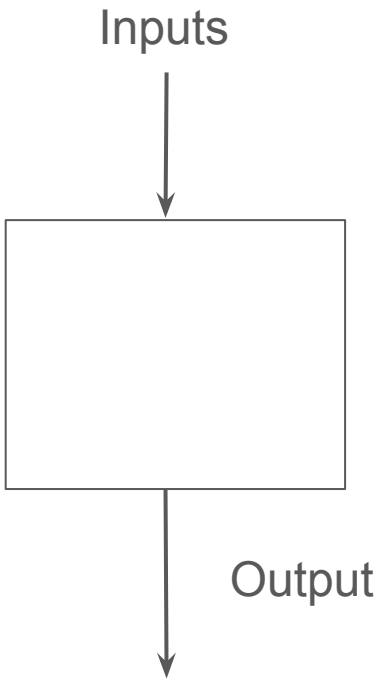
- Easy to do for unit testing or component testing

Test Driven Development

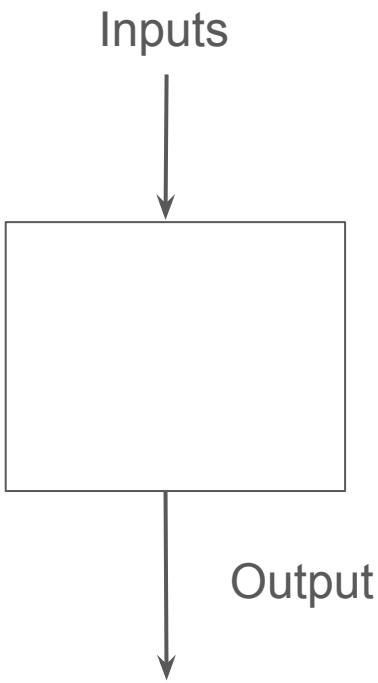
- Given a feature:
  - What does it need to do to meet the requirement?
  - Translate this to:
    Which tests does this feature need to pass?
  - Write the tests:
    - Include positive and negative cases
    - Include boundary cases
  - Let the tests fail
  - Write the code to pass each test
  - Refactor the code
- Easy to do for unit testing or component testing



Where to Start?



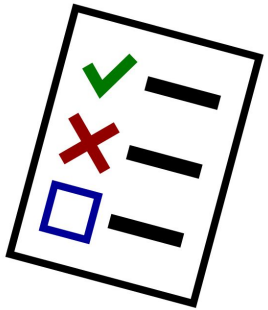
Where to Start?



- Expectations about input?
  - How many?
  - What format?
- Expectations about output?
  - How many?
  - What format?
- What about the code that massaged the output?
  - Not yet!

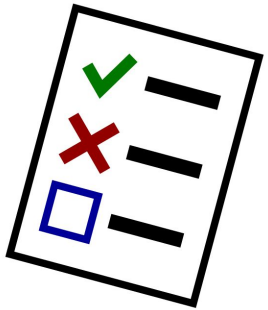
Example 1

- Implement an add(a, b) function



What's an example test case?

Example 1

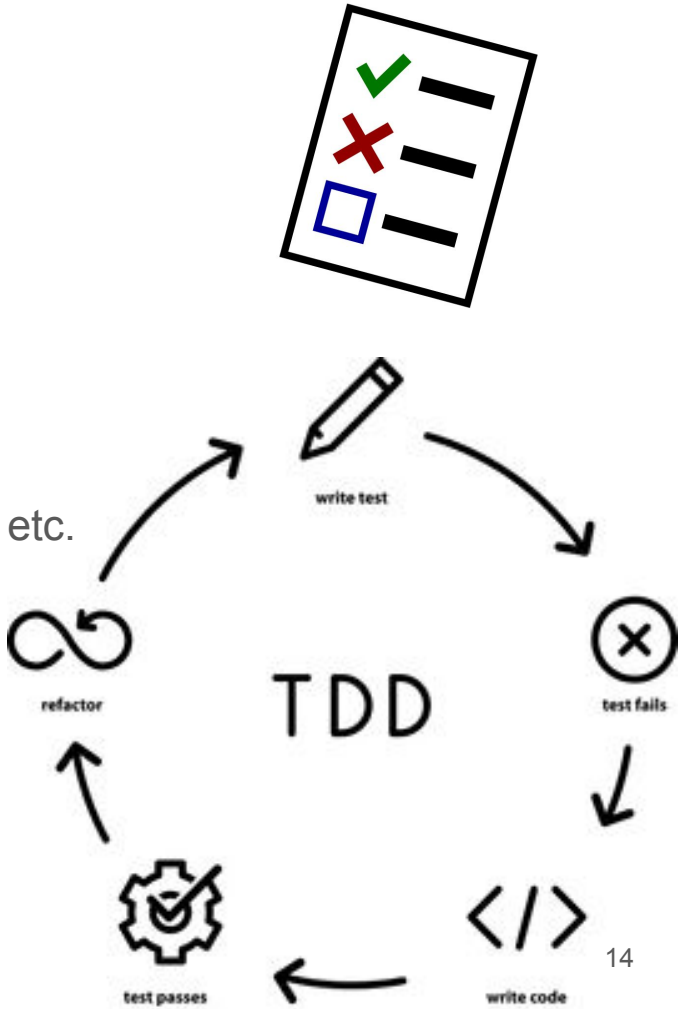


- Implement an add(a, b) function
  - Positive test case?
  - Negative test case?
  - Argument ordering?
  - Other variations?
    Integers, floats, negatives, more than 2 arguments, etc.

Example 1

- Implement an add(a, b) function
  - Positive test case?
  - Negative test case?
  - Argument ordering?
  - Other variations?

    Integers, floats, negatives, more than 2 arguments, etc.
- Write code with minimal interpretation of function
  - Some tests pass
  - Improve code to let additional test cases pass
  - Refactor
  - Repeat



Example 2

- Implement a sort_list(alist) function

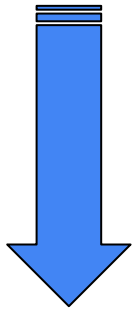
What's an example test case?

Example 2

- Implement a sort_list(alist) function
  - Numbers?
  - Characters?
  - Empty list?
  - Duplicates?
  - Optional ordering parameter to pass in as argument?
  - For a unique input, can output have more than one correct answer?
- Adopt same iterative process as previous example

Example 2

- Implement a sort_list(alist) function
  - Numbers?
  - Characters?
  - Empty list?
  - Duplicates?
  - Optional ordering parameter to pass in as argument?
  - For a unique input, can output have more than one correct answer?



- Adopt same iterative process as previous example
- Notice design unfolding: tests serve as requirements specifications!

Try TDD Coding

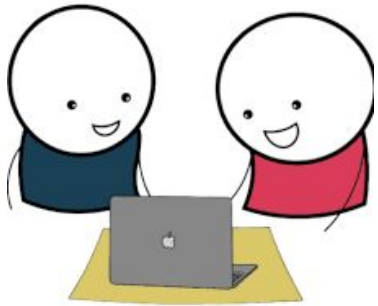
- Download exercise (.zip) from course website
- unzip and cd into "tdd-mini-exercise-python/"
- Run:

```
python -m venv .venv
source .venv/bin/activate                    # Windows: .venv\Scripts\activate
pip install -U pip pytest
pytest                                       # see failing tests (RED)
```

- Fix 3 functions in .py files in src/ to pass each test (GREEN)
- Clean up once they pass (REFACTOR)

Task (~30 min)

- Complete 3 functions so that all tests pass:
- src/tdd_katas/math_utils.py → is_prime(n)
- src/tdd_katas/text_utils.py → slugify(text)
- src/tdd_katas/cart.py → total(items, discounts=None)
- Add a test for each function
- Make them pass
- Refactor your code as needed
- You may work individually or in pairs

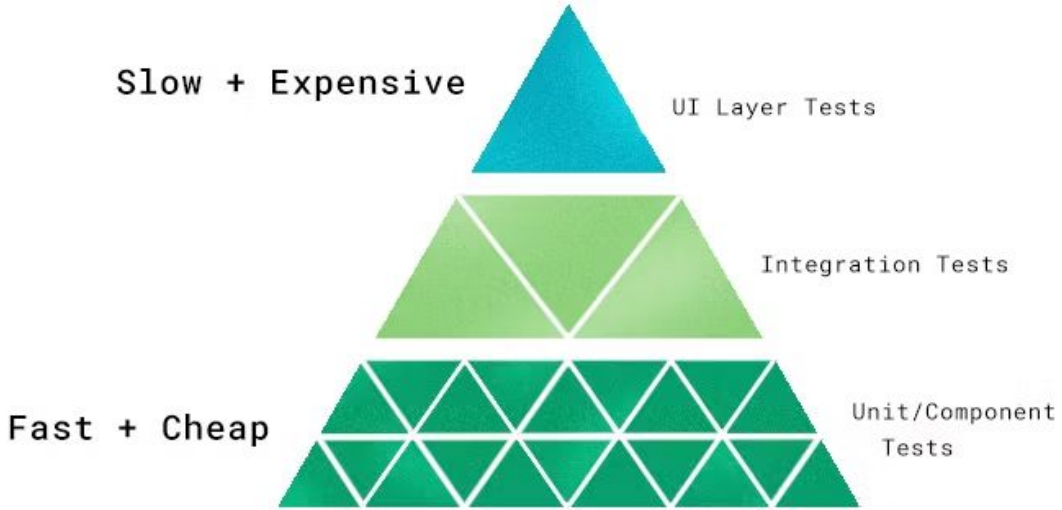


Reflection (Complete on Canvas, ~5 min)

- How did the Red-Green-Refactor cycle feel compared to your usual way of coding?
- Did you ever feel the urge to “skip ahead” and write more code than the test required? What happened?
- How often did you run tests? Was it more or less than you expected?
- How did adding tests improve your understanding of the functions?
- What kind of refactoring did you do? Did the tests help you simplify or restructure your implementation?

Types of Testing

- Integration and unit/component testing can be mocked and stubbed
- UI testing is often manual



Types of Testing

- Integration and unit/component testing can be mocked and stubbed
- UI testing is often manual
- Alternative is to use behavior-driven development
  - Automated testing while anticipating user behavior
  - Check: Selenium, Cypress