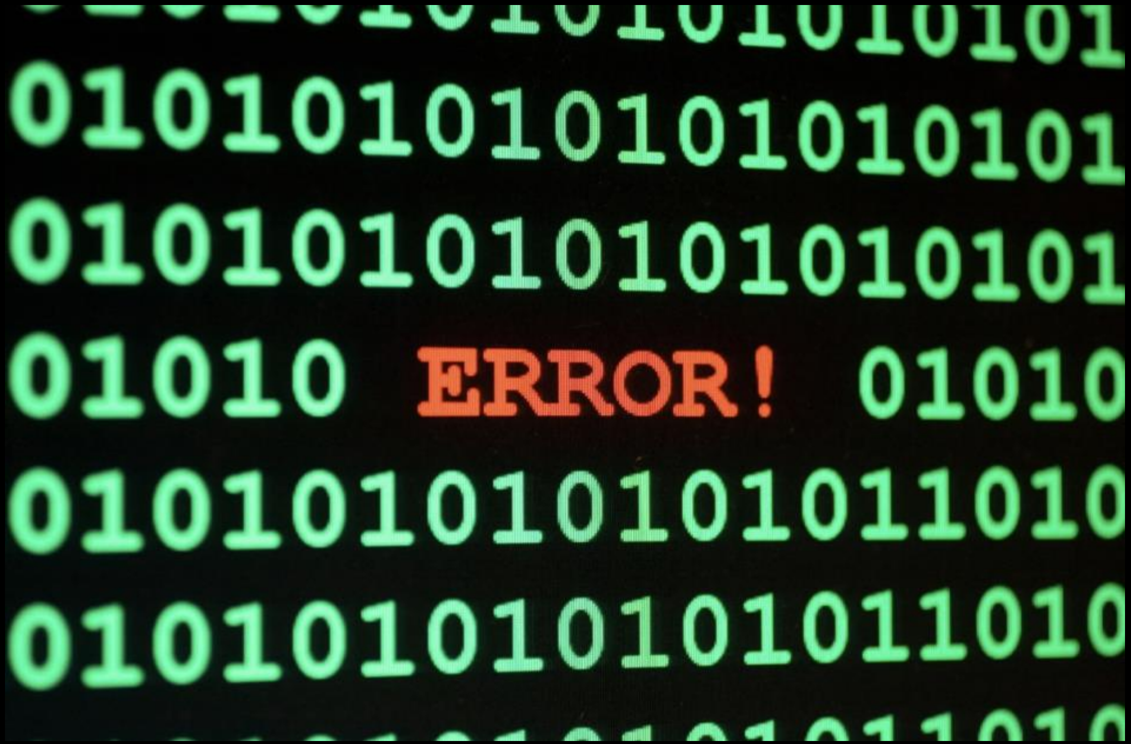
TESTING TOOLS

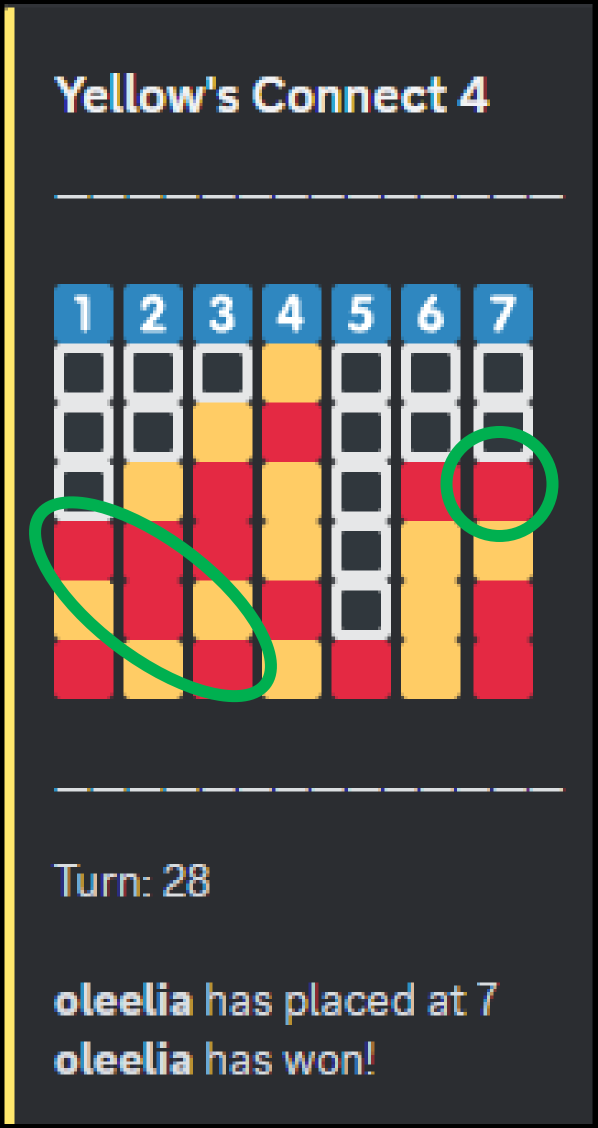
By Justin N, Muhammad S, Zoila V



WHY TEST YOUR CODE?

- Unit testing can help ensure every part of your code works as intended
- Robust testing can help prevent unexpected errors
- Test driven development can help create guidelines or goals for code to meet
- Red-Green-Refactor can optimize performance

STORY TIME

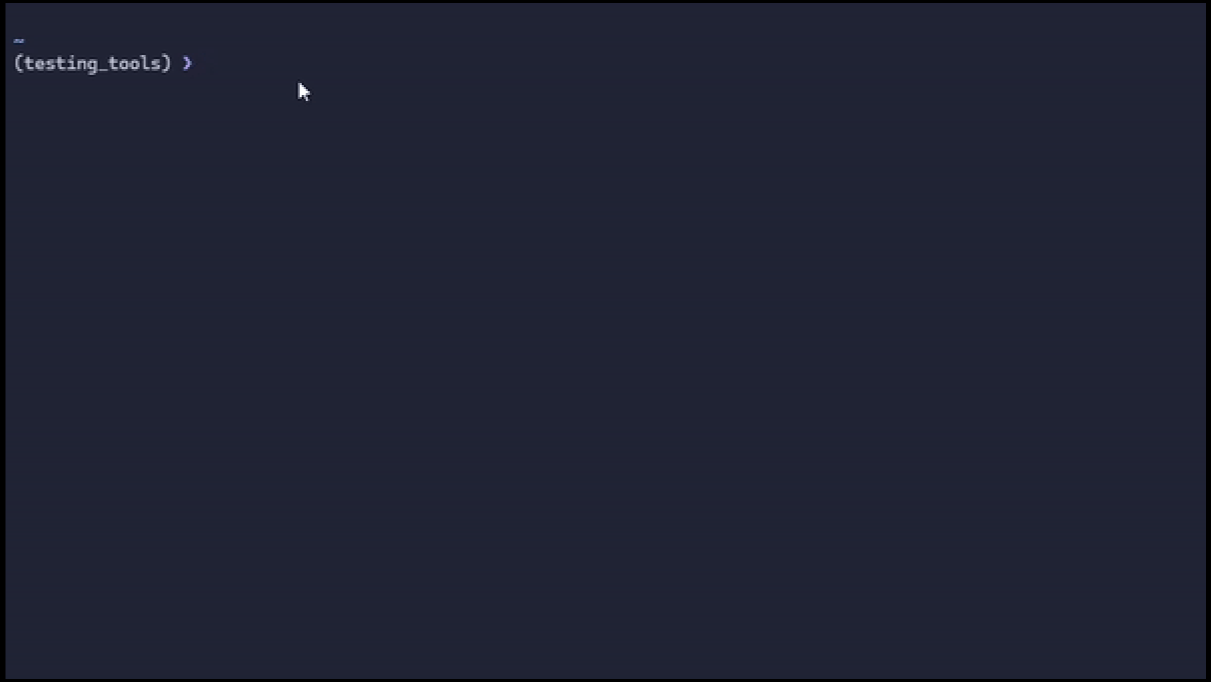


Yellow's Connect 4

Turn: 28

oleelia has placed at 7
oleelia has won!

TOOLS OVERVIEW



```
~
(testing_tools) ❯
```

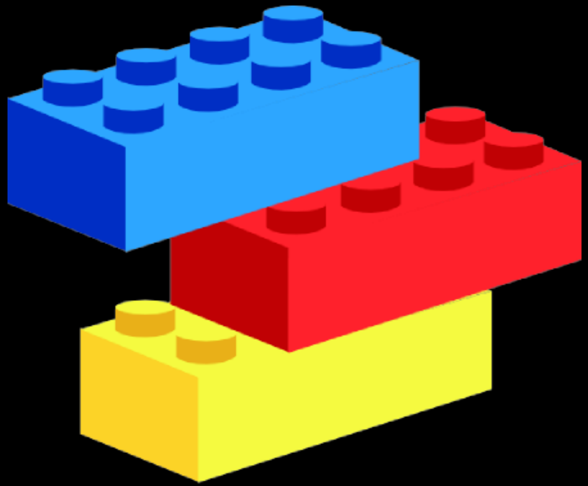
- pip (python package manager)
  - [python/py] –m ensurepip –upgrade
  - pip install [package_name]
- pytest
- pytest-mock
- faker
- pytest-BDD

UNIT TESTING

- Purpose: Test the smallest "units" of code in isolation, independent from the code base
- Examples: Testing functions, methods, classes separately
- Analogy: Doing quality control on an individual Lego piece

INTEGRATION TESTING

- Purpose: Testing how different parts interact with each other
- Example: Function to Function, Class to Class, Function to Class
- Analogy: Checking if Lego bricks fit to each other by connecting them

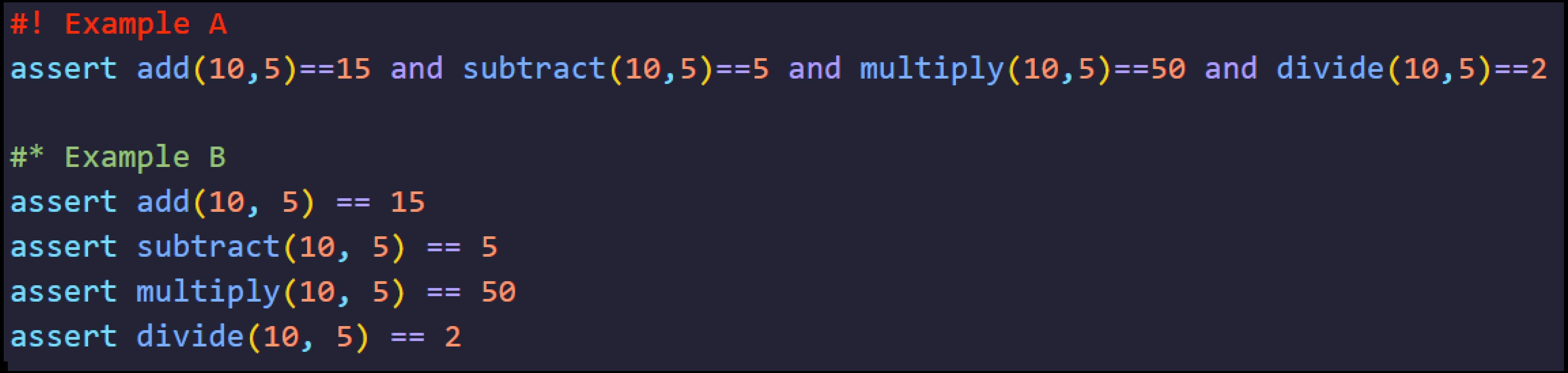


FUNCTIONAL TESTING

- Purpose: Test to verify if a features behaves according requirements usually regarding the end user's perspective
- Example: A certain hand gesture should produce the correct response
- Analogy: Check to see if an assembled Lego car rolls

WRITING TEST CASES WITH PYTEST

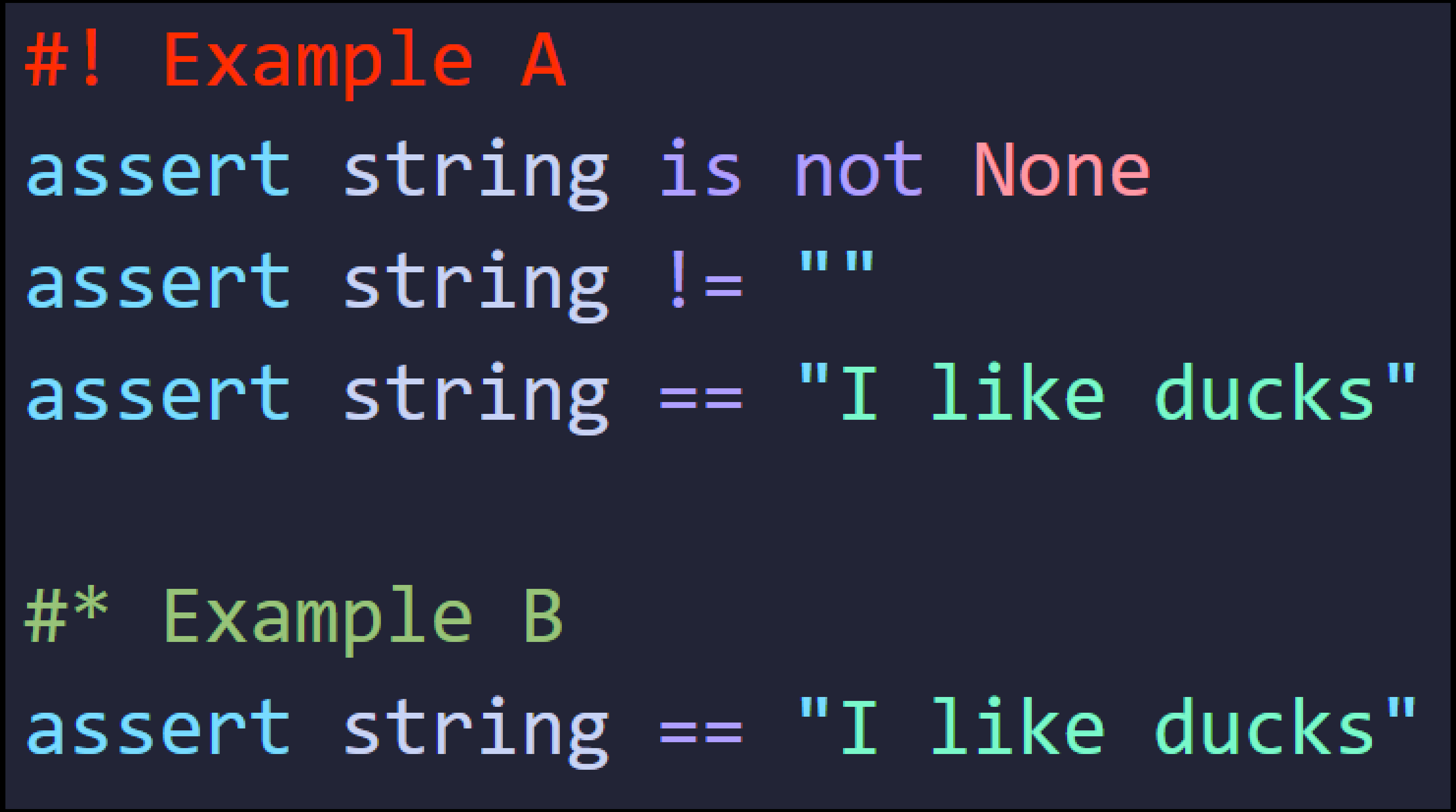
GOOD VS BAD



```
#! Example A
assert add(10,5)==15 and subtract(10,5)==5 and multiply(10,5)==50 and divide(10,5)==2

#* Example B
assert add(10, 5) == 15
assert subtract(10, 5) == 5
assert multiply(10, 5) == 50
assert divide(10, 5) == 2
```

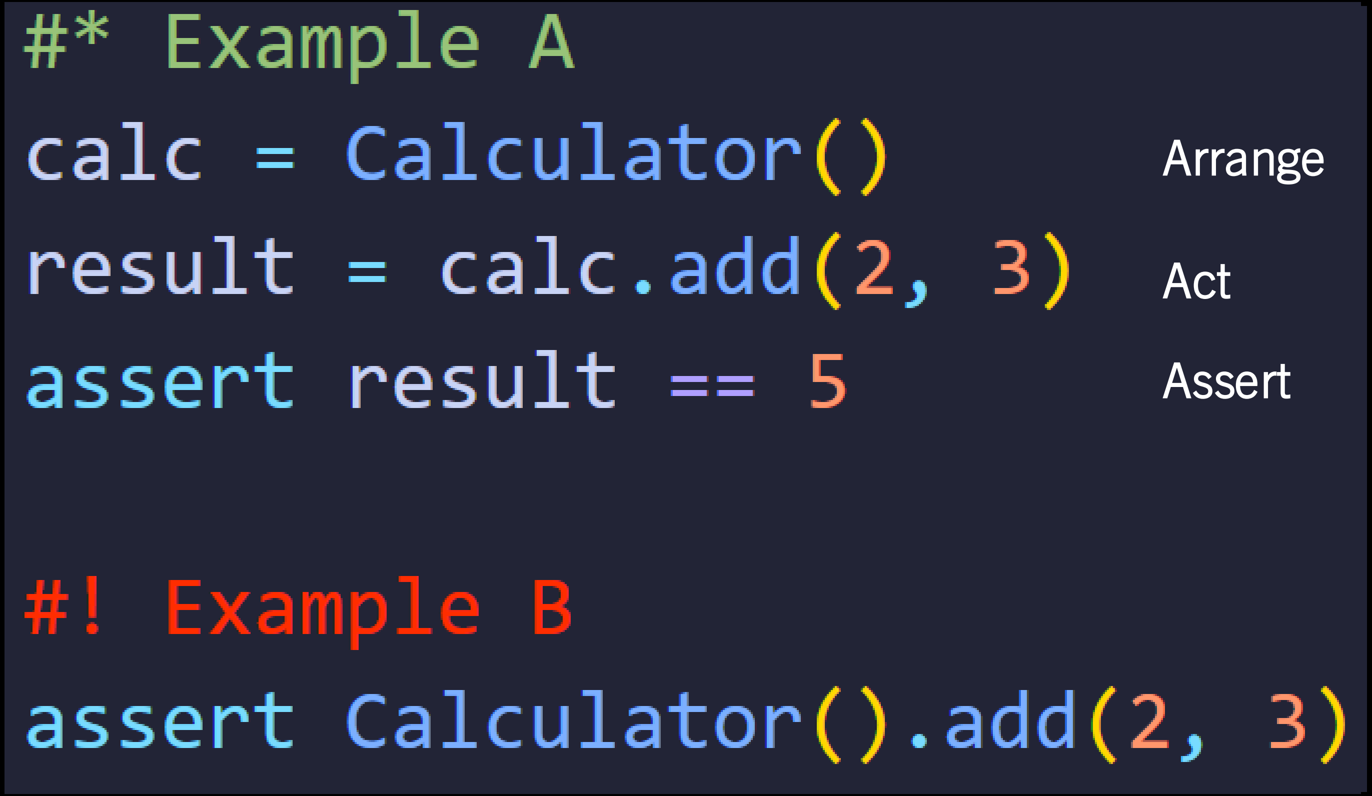
GOOD VS BAD



```
#! Example A
assert string is not None
assert string != ""
assert string == "I like ducks"

#* Example B
assert string == "I like ducks"
```

GOOD VS BAD



```
#* Example A
calc = Calculator()
result = calc.add(2, 3)
assert result == 5


#! Example B
assert Calculator().add(2, 3)
```

Arrange

Act

Assert

TEST DOUBLES

- Temporary stand-ins used to simulate external systems and dependencies.
- Used so programmers do not have to implement the whole system themselves
- Some example of components that test doubles can mimic are:
  - API's
  - Databases
  - Functions
- Usually used in unit testing due to the isolation from other systems test doubles provides
- Most common types of Test Doubles are:

MOCKS

- Purpose: Mocks verify behavior by imitating a component, so the actual component does not need to be used
- Examples: Instead of calling send_email(), create a mock object with the same parameters and it'll verify the behavior.
- Usage: Focuses on asserting calls/arguments

STUBS

- Purpose: Stubs simulate behavior by returning predetermined outputs or responses.
- Examples: Instead of implementing get_user() fully, make it return "John Doe"
- Usage: Focuses on controlling return values

FAKES

- Purpose: Fakes are a simplified version of the real thing using a lightweight implementation
- Example: A fake database stored in memory used solely for testing purposes
- Usage: Used when as a simplified replacement

TEST DATA MANAGEMENT

- Purpose: Ensuring the right data is being used for test by mimicking real world data
- Acquiring organic data can be time consuming, taking weeks, months or years
- Messy data require a data processing stage
- Tools like Faker can generate various types of fake data in seconds, speeding up the data acquiring and organization stage

TEST DRIVEN DEVELOPMENT (TDD)

- Purpose: Drives design and code quality by ensuring tests are written before the code
- Principle: RED-GREEN-REFACTOR
  - Red: Write a test
  - Green: Write just enough code to pass the test
  - Refactor: Just as the name suggest, Refactor

BEHAVIOR DRIVEN DEVELOPMENT (BDD)

- Purpose: A software development methodology that encourages collaboration. Includes:
  - Engineers
  - QA
  - Stakeholders
- Defines the behavior of an application in a language that anyone can understand
- Gherkin Language: A structed language used in BDD that has a syntax that is simple and is written in a natural language style. Supports many languages beyond English

AFTER PRESENTATION ASSIGNMENT

- Choose one of the testing styles:
  - Write two unit tests
  - Write two integration tests
  - Write two functional test