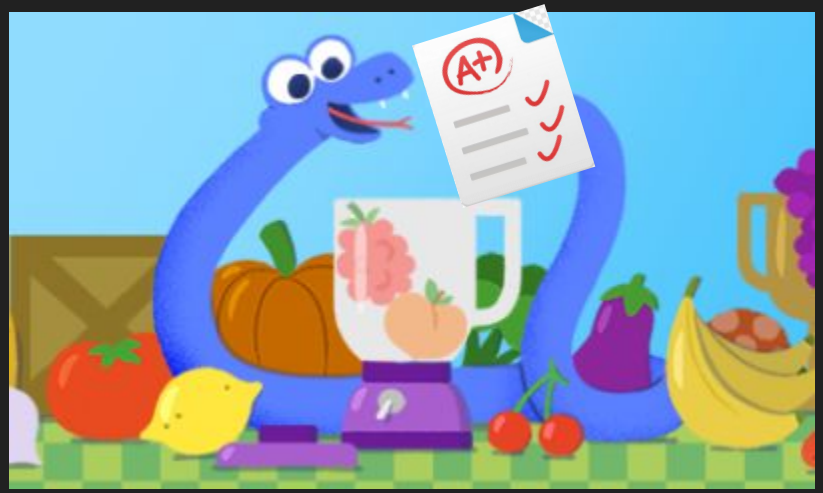
pytest



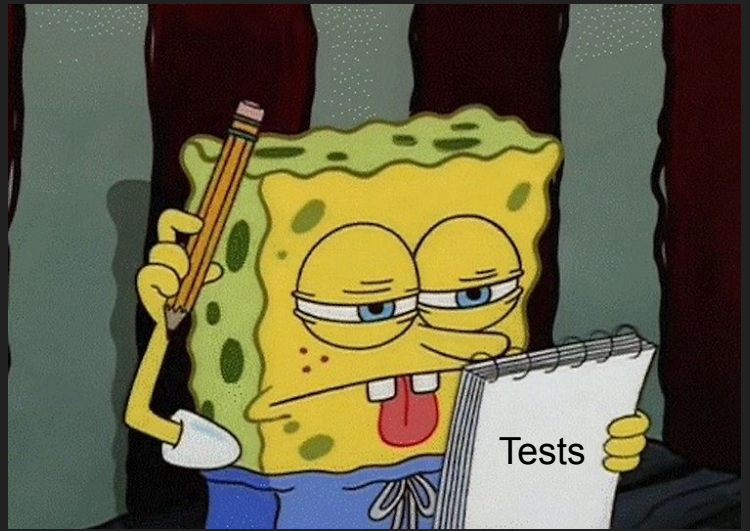
Andrew Goetz, Jin Xian Li, Yong Liu

Overview

- Importance of testing
- Pytest functionality
- How we can incorporate Pytest
- Pytest vs UnitTest

Why is Testing so Important?

- Testing your product before it’s release is very important
- When your product is released to large/small consumer base, your product should be used as intended
- Assure the quality of your software with extensive testing
- Less headaches in the long run
    - Fixing them early cost less compared to fixing them late



Proper Testing could have avoided Disaster

- Be smart when testing
  - Common cases, Edge cases
- When healthcare.gov was released, the website crashed due to heavy load
- Therac-25, arithmetic overflow occurred thus overdosing patients with radiation
- Consequences depends on the use cases of the product, generally just a good idea to extensively test your program
  - Using pyTest

PyTest Useful Functionality

- Assert
  - Checks if return is either True or False on the test case
- Parameterization
  - Can be used to pass parameters to test functions
  - Reduces duplicate code between tests
- Fixtures
  - Useful to run code before certain test methods
  - Handles test dependencies, state, and reusability functionality

PyTest Usage in our Project (MyMDB)

Example usage of assert:

- API gives data in JSON
- We will format it after for display on our site
- Use PyTest to test if our formatting works

```
def format_data(medialist):
    #formats data taken from API

def test_format():
    medialist = [
        {
            "title": "Binghamton University Movie",
            "genre": "Horror",
            "rating": "3"
            "media_type": "Movie"
        },
    ]
    assert format_data(medialist) ==
        ["Binghamton University, Horror Movie (3)"
    ]
```

PyTest Usage in our Project (MyMDB) cont.

Adding on Parameterization:

```
def test_format():
    medialist = [
        {
            "title": "Binghamton University Movie",
            "genre": "Horror",
            "rating": "3"
            "media_type": "Movie"
        },
        {
            "title": "Generic TV Show",
            "genre": "Action",
            "rating": "10"
            "media_type": "Series"
        },
    ]

    # test parameterization
    @pytest.mark.parametrize("medialist, expected", [(medialist[0], "Binghamton University, Horror Movie (3)"), (medialist[1], "Generic TV Show, Action Series (10)")])
    assert format_data(medialist) == expected
```

PyTest Usage in our Project (MyMDB) cont.

Finally, adding fixtures to our project:

```
def format_data(medialist):
    #formats data taken from API

@pytest.fixture
def media_data():
    return [
        {
            "title": "Binghamton University Movie",
            "genre": "Horror",
            "rating": "3"
            "media_type": "Movie"
        },
        {
            "title": "Generic TV Show",
            "genre": "Action",
            "rating": "10"
            "media_type": "Series"
        },
    ]

def test_format(media_data):
    # test parameterization
    @pytest.mark.parametrize("media_data, expected", [(media_data[0], "Binghamton University, Horror Movie (3)"), (media_data[1], "Generic TV Show, Action Series (10)")])
    assert format_data(media_data) == expected
```

Running your Tests

- Run tests with pytest [file_containing_tests]
- Run pytest on a directory
  - pytest [directory_containing_tests]
- Run certain tests with same marker
  - pytest -m slow, runs tests with @pytest.mark.slow decoration
- Run a particular test
  - pytest test1.py::test_function

```python
import pytest

@pytest.fixture
def function():
    return 0

def test_function(function):
    assert function == 0
```

Pytest vs Unittest

```python
import pytest

@pytest.fixture
def function():
    return 0

def test_function(function):
    assert function == 0
```

```python
import unittest

def function():
    return 0

class Testing(unittest.TestCase):
    def test_function(self):
        self.assertEqual(function(), 0)
```

- Unittest is more verbose, requiring testing classes to use (vs pytest's more lightweight fixtures)
- Unittest has multiple different assert functions (vs pytest's single assert)
- Unittest ships with python (vs pytest needing to be installed with pip)
- Pytest gives better error messages when a test fails

Verdict

- We recommend pytest
  - Easy to use
  - Versatile (use in fixtures or standalone)
  - Intuitive (resembles C library assert.h)