TEST DRIVEN DEVELOPMENT with Pytest

TEST YOUR PYTHON FUNCTIONS LIKE PRO

PROBLEM SOLVED BY TDD

1 UNDERSTANDING TDD

- Tests are written first based on the required specification
- Functions are written next and updated until the tests pass
- Unit Tests have to first pass before Integration tests can pass

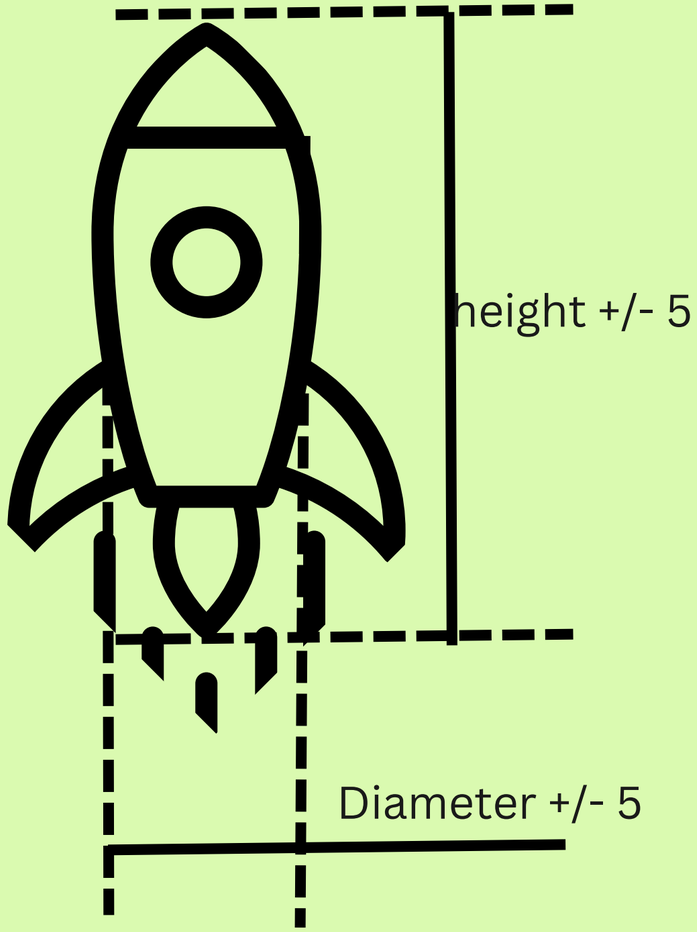
2 WHAT IS THE PROBLEM?

- Automate the code written by anyone without a human looking at the output.
- Ensure Code quality by testing whether function meets specs

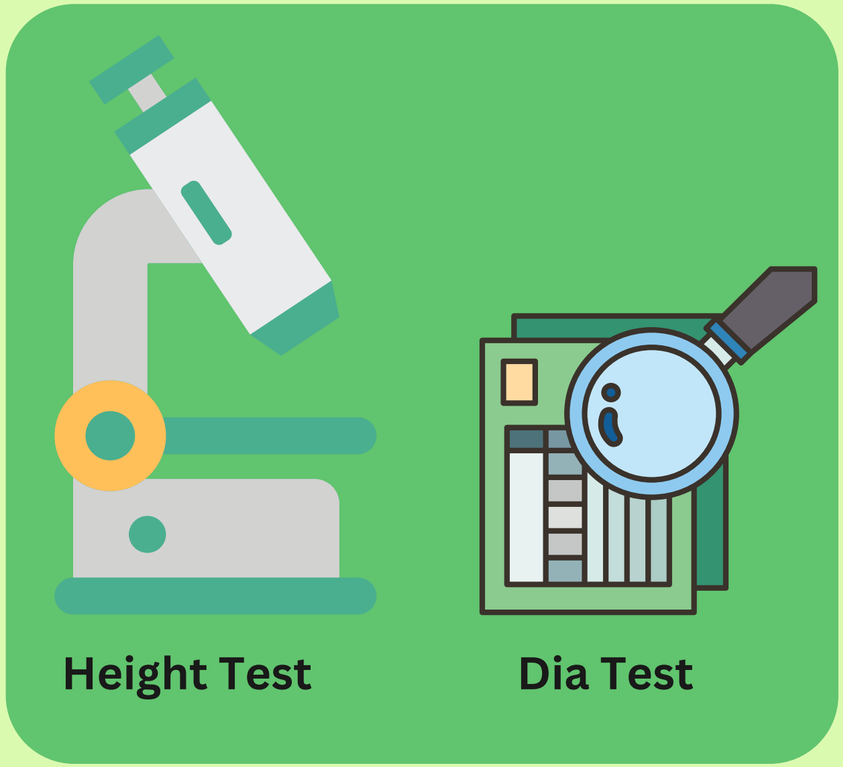
3 HOW PYTHON SOLVES IT

- Python ships with Unit Tests
- A better library will be Pytest
- Assert method, Pytest fixtures and prototype methods speed up variety of tests done

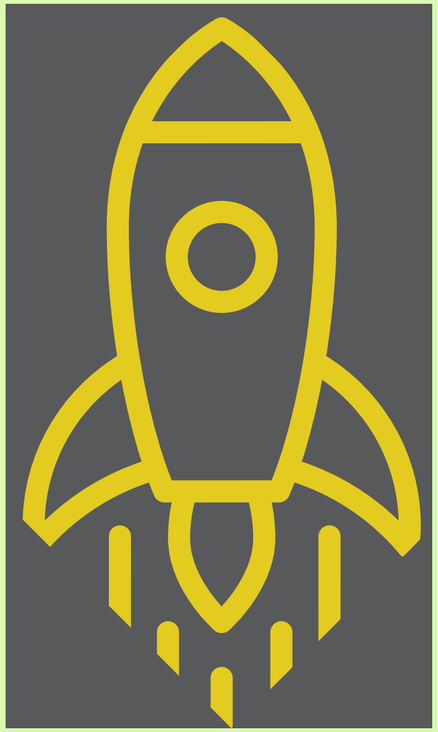
TDD IN ACTION



SPECIFICATION



TEST SUITE



ROCKET FUNCTION

PIP INSTALL PYTEST

TEST FUNCTIONS

Idea : Using Assert statement in Test functions

```
import pytest
from calculations import *

def test_add():
    assert add_num(5,7) == 12

def test_mul():
    assert mul_num(5,8) == 40

def test_mean():
    assert mean_num(5,2,7,10) == 6
```

CODE EXPLANATION

- Note test_ in front of the functions
- Note import, from and assert statements
- File name has to start with test

These functions are written with the desired answers based on the specification.

This is called the test suite. Pytest has additional features that will be explained next slide

CALCULATIONS FUNCTIONS

Idea : Creating the functions so the tests can pass

```
def add_num(a, b):
    #pass
    return a + b

def mul_num(a, b):
     #pass
     return a * b

def mean_num(*a):
    #pass
    length = len(a)
    return sum(a)/length
```

CODE EXPLANATION

- Note the pass
- Functions have to return value
- Functions names can be created after the test_suite is provided by the testers

Challenge in using TDD during learning python is, to be able to write tests.

@work devs and testers will be different teams. When learning to code, need to practice writing tests and writing functions passing tests

ADVANCED AUTOMATION

Idea : Using Pytest Fixtures and Prototypes

```
@pytest.mark.parametrize("x, y, result",[
    (50, 30, 1500),
    (15,7,105),
    (7,6,42)
    ])
def test_mul(x,y,result):
    print('testing mul')
    assert mul_num(x,y) == result
```

CODE EXPLANATION

- Pytest provides decorators which can reduce the need for writing multiple test functions
- When executing the tests, the command is

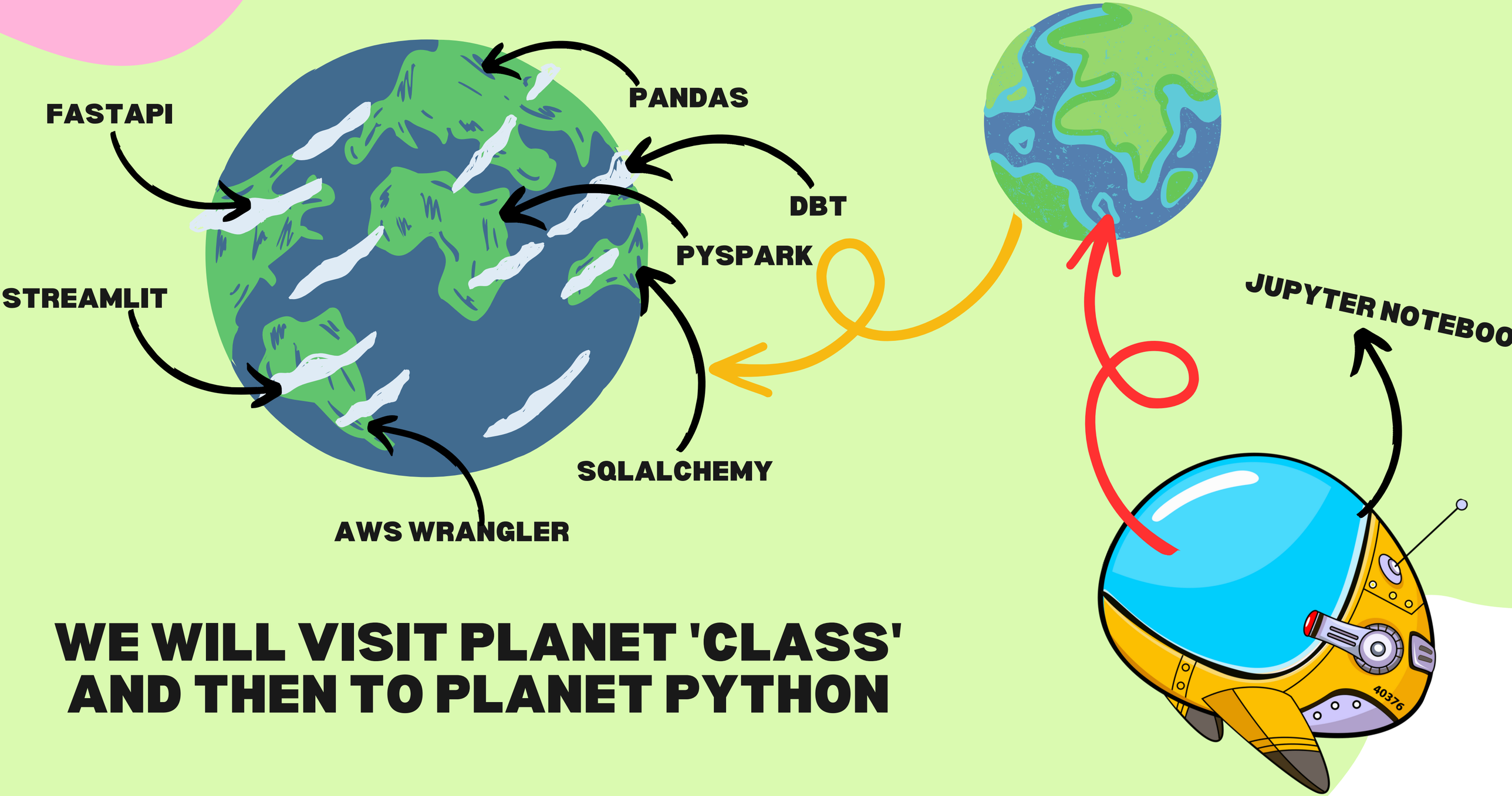
pytest -s -v .

note the '.' that means the current directory. It will collect all the tests and execute

Write test first & then code!!!!

TDD will look like additional work. It will save a tonne later

- Ci/Cd pipeline will require TDD
- Working with Database require TDD
- API creation will require TDD
- Creating dashboard require TDD
- Working on Big Data require TDD



WE WILL VISIT PLANET 'CLASS' AND THEN TO PLANET PYTHON