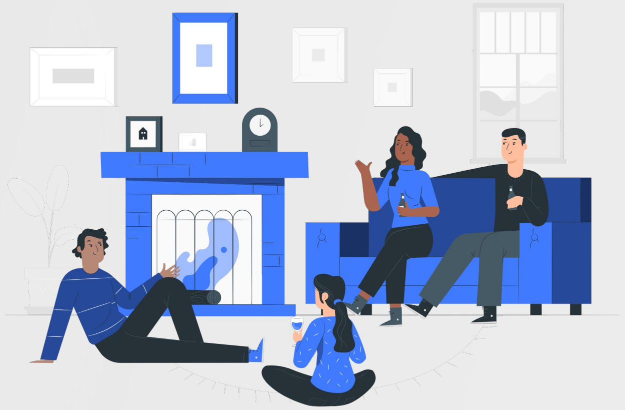
Session 1: Adding Tests to Projects

Nicholas Del Grosso
iBOTS Code
Jan 22, 2024

https://ibehave.nrw/ibots-platform/about-ibots/

What’s the Plan?



Motivations Round Table: Why Are We Interested in Writing Automated Tests Together?

1

“Hello, I’m…”
(Introduce Yourself): Name, Role, Project You’ll Be On

2

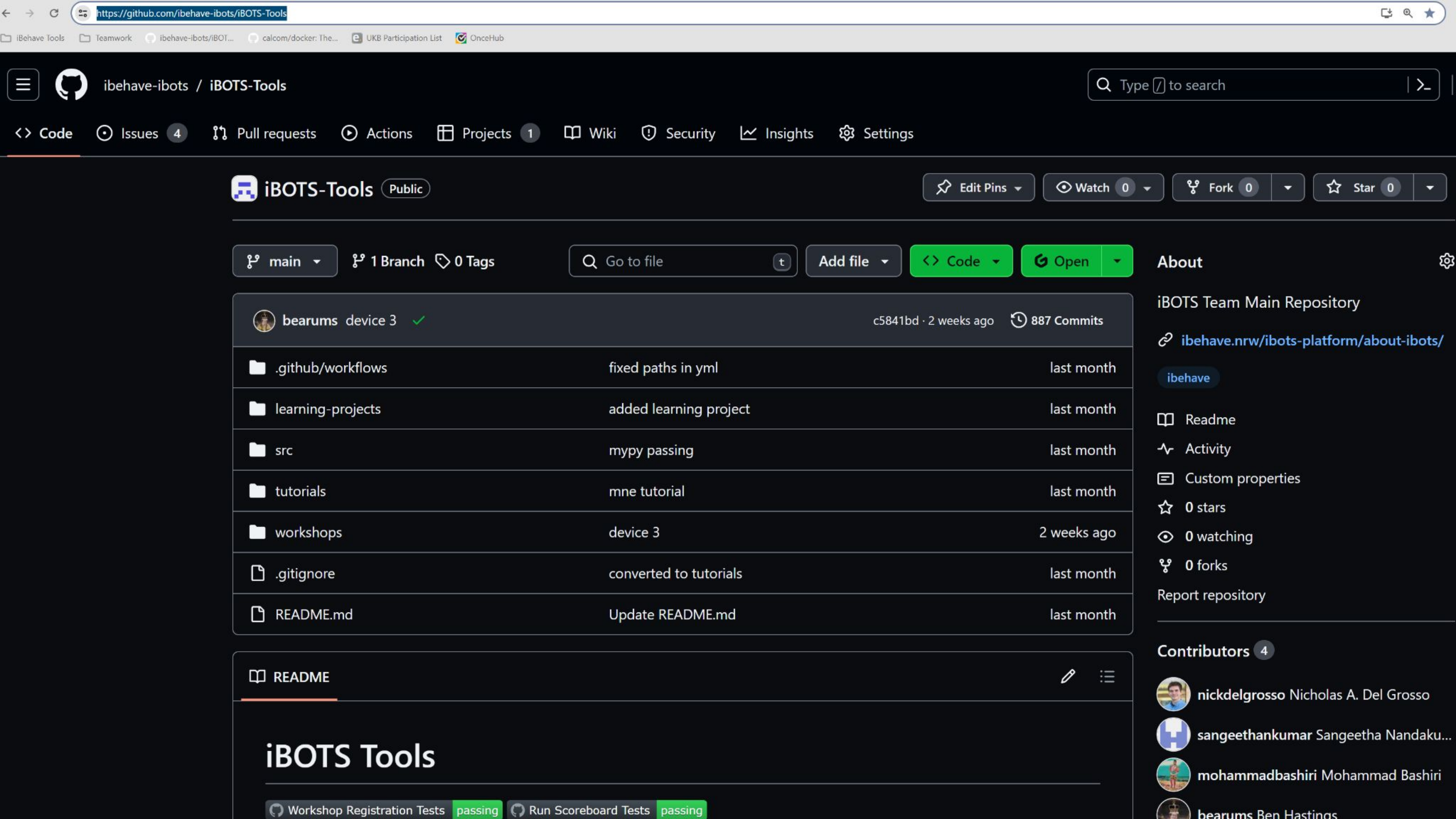
“I’m here because…”
(General Goals and Motivations)

3

“I hope in these series we get to…”
(Specific Learning Goals and Tasks)

Developing on Other's Source Code

Working with Git and GitHub/GitLab



https://github.com/ibehave-ibots/iBOTS-Tools

iBehave Tools | Teamwork | ibehave-ibots/iBOT... | calcom/docker: The... | UKB Participation List | OnceHub

ibehave-ibots / iBOTS-Tools

Code | Issues 4 | Pull requests | Actions | Projects 1 | Wiki | Security | Insights | Settings

iBOTS-Tools Public

Edit Pins | Watch 0 | Fork 0 | Star 0

main | 1 Branch | 0 Tags | Go to file | Add file | Code | Open

bearums device 3 ✓ c5841bd · 2 weeks ago 887 Commits

.github/workflows	fixed paths in yml	last month
learning-projects	added learning project	last month
src	mypy passing	last month
tutorials	mne tutorial	last month
workshops	device 3	2 weeks ago
.gitignore	converted to tutorials	last month
README.md	Update README.md	last month

About

iBOTS Team Main Repository

ibehave.nrw/ibots-platform/about-ibots/

ibehave

Readme
Activity
Custom properties
0 stars
0 watching
0 forks
Report repository

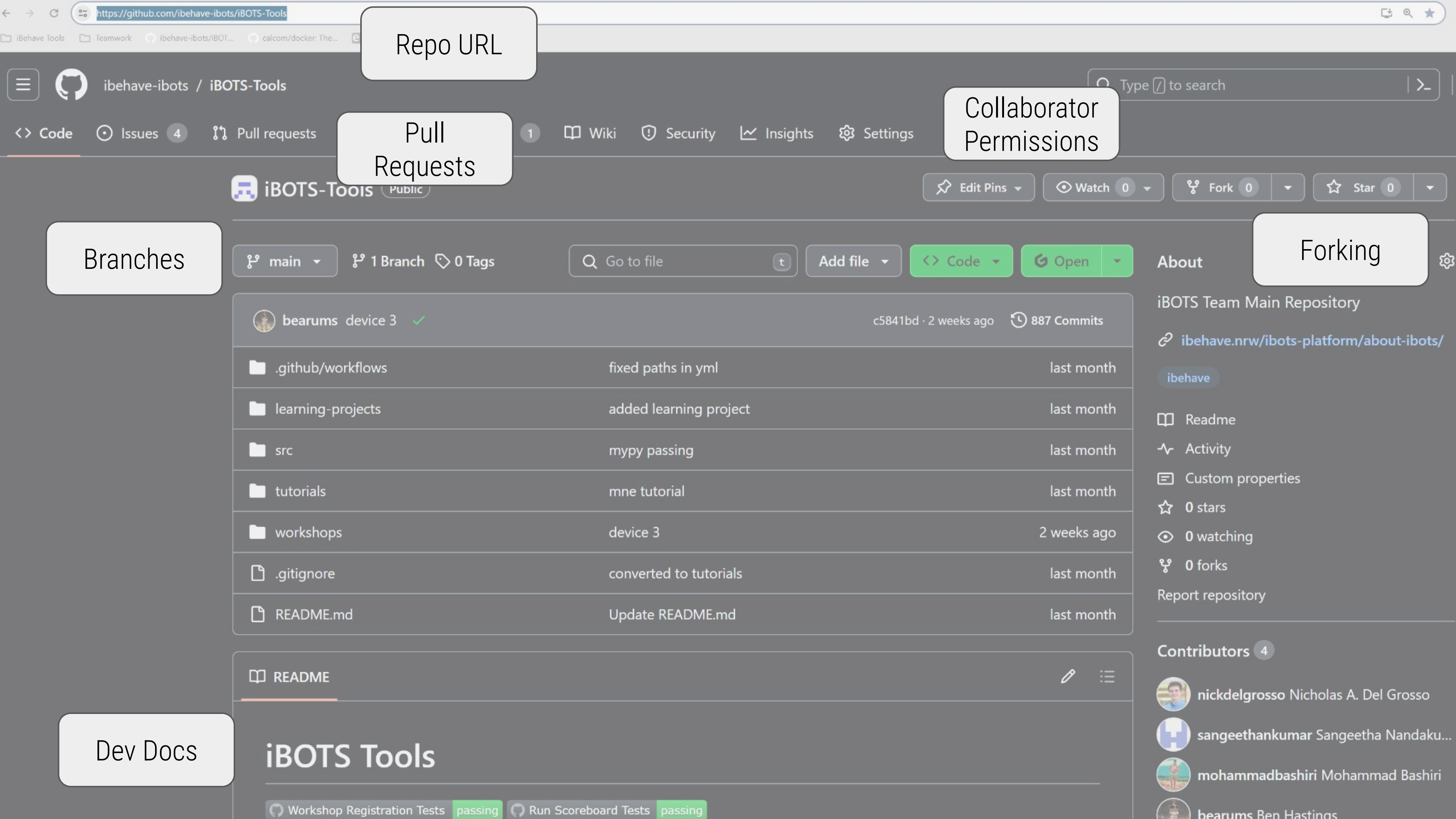
Contributors 4

nickdelgrosso Nicholas A. Del Grosso
sangeethankumar Sangeetha Nandaku...
mohammadbashiri Mohammad Bashiri
bearums Ben Hastings

README

iBOTS Tools

Workshop Registration Tests passing | Run Scoreboard Tests passing


Repo URL

Pull Requests

Collaborator Permissions

Branches

Forking

Dev Docs

https://github.com/ibehave-ibots/iBOTS-Tools

iBehave Tools · Teamwork · ibehave-ibots/iBOT... · calcom/docker: The...

ibehave-ibots / iBOTS-Tools

Type / to search

Code · Issues 4 · Pull requests 1 · Wiki · Security · Insights · Settings

iBOTS-Tools Public

Edit Pins · Watch 0 · Fork 0 · Star 0

main · 1 Branch · 0 Tags · Go to file · Add file · Code · Open

bearums device 3 ✓ c5841bd · 2 weeks ago · 887 Commits

.github/workflows	fixed paths in yml	last month
learning-projects	added learning project	last month
src	mypy passing	last month
tutorials	mne tutorial	last month
workshops	device 3	2 weeks ago
.gitignore	converted to tutorials	last month
README.md	Update README.md	last month

About

iBOTS Team Main Repository

ibehave.nrw/ibots-platform/about-ibots/

ibehave

Readme · Activity · Custom properties · 0 stars · 0 watching · 0 forks

Report repository

Contributors 4

- nickdelgrosso Nicholas A. Del Grosso
- sangeethankumar Sangeetha Nandaku...
- mohammadbashiri Mohammad Bashiri
- bearums Ben Hastings

README

iBOTS Tools

Workshop Registration Tests passing · Run Scoreboard Tests passing

1

“Git”ting the Code

Building a Software Project onto your computer from a remote source and co-developing it asynchronously with others.

1

“Git”ting the Code

Building a Software Project onto your computer from a remote source and co-developing it asynchronously with others.

a. Clone the Repository

- (if Private or in your team) Be added as a collaborator on GitHub
  - (if can’t be a GitHub collaborator): Fork the project to your own GitHub username
- Clone the Repo to a local folder using Git

b. Checkout or Create the session’s Branch

- Create a new branch for development, or checkou the existing development branch.
- Make a whitespace commit and test you can push the branch to GitHub.

c. Run the Tests

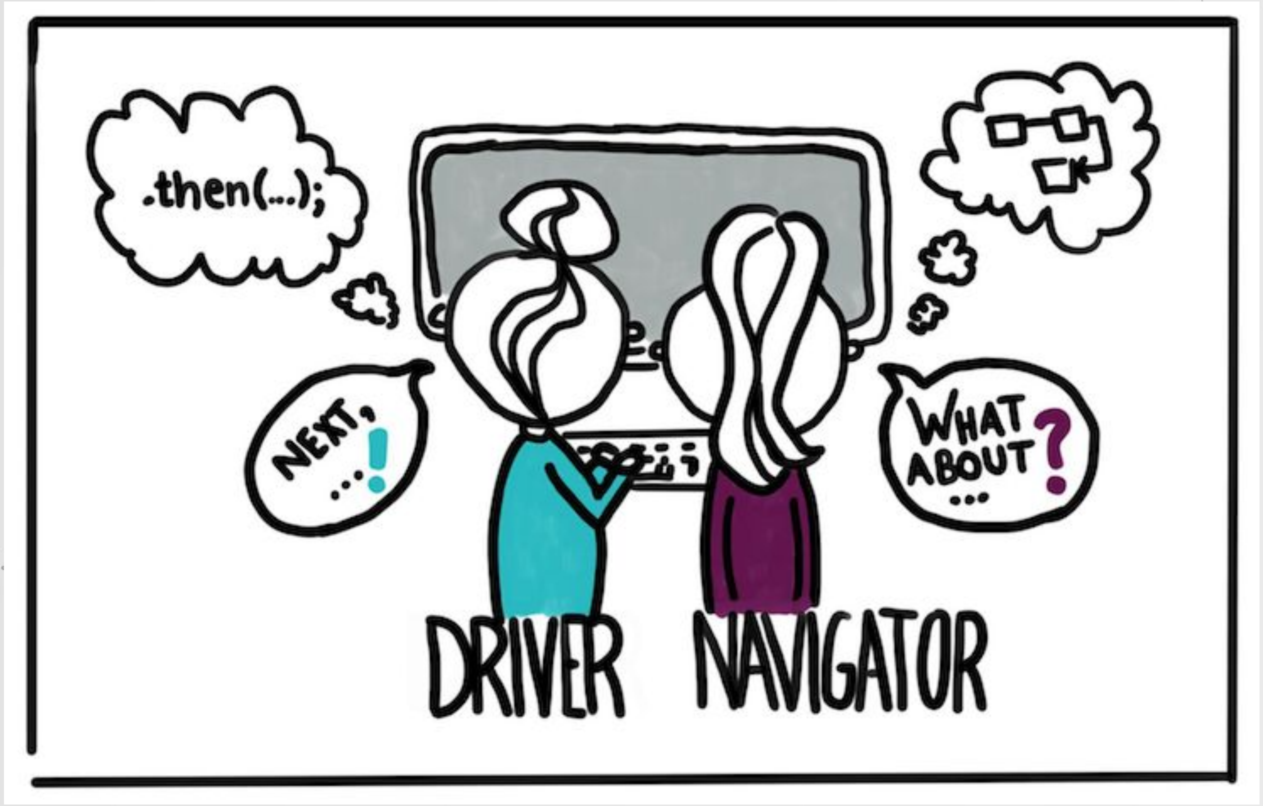
- Open the Repo as a Project in your IDE
- Install all the User Dependencies
- Install all the Developer Dependencies
- Run the Test Suite and Confirm Everything Passes
- On problems, improve the README docs.
- (if no test suite) run some of the code.

(d. Repeat A-C for each team member)

Coding in a Group

Remote “Ensemble” Programming

Core Concept: The “2 Minds” Rule



https://martinfowler.com/articles/on-pair-programming/driver_navigator.png

Ensemble Programming: The Three Roles

The Navigator
Leadership Role

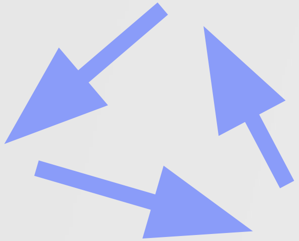
The Driver
Typer Role

The Ensemble
Support Role

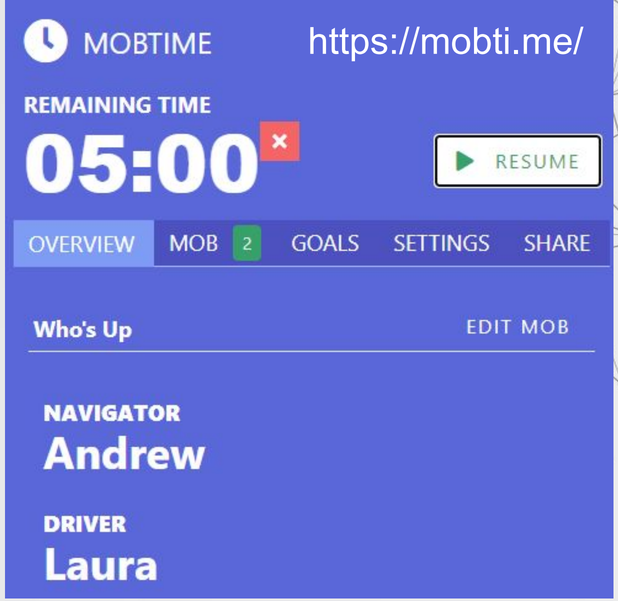
Mob Programming

The Facilitator
Leadership Role

The Team
Support Role



The Driver
Worker Role



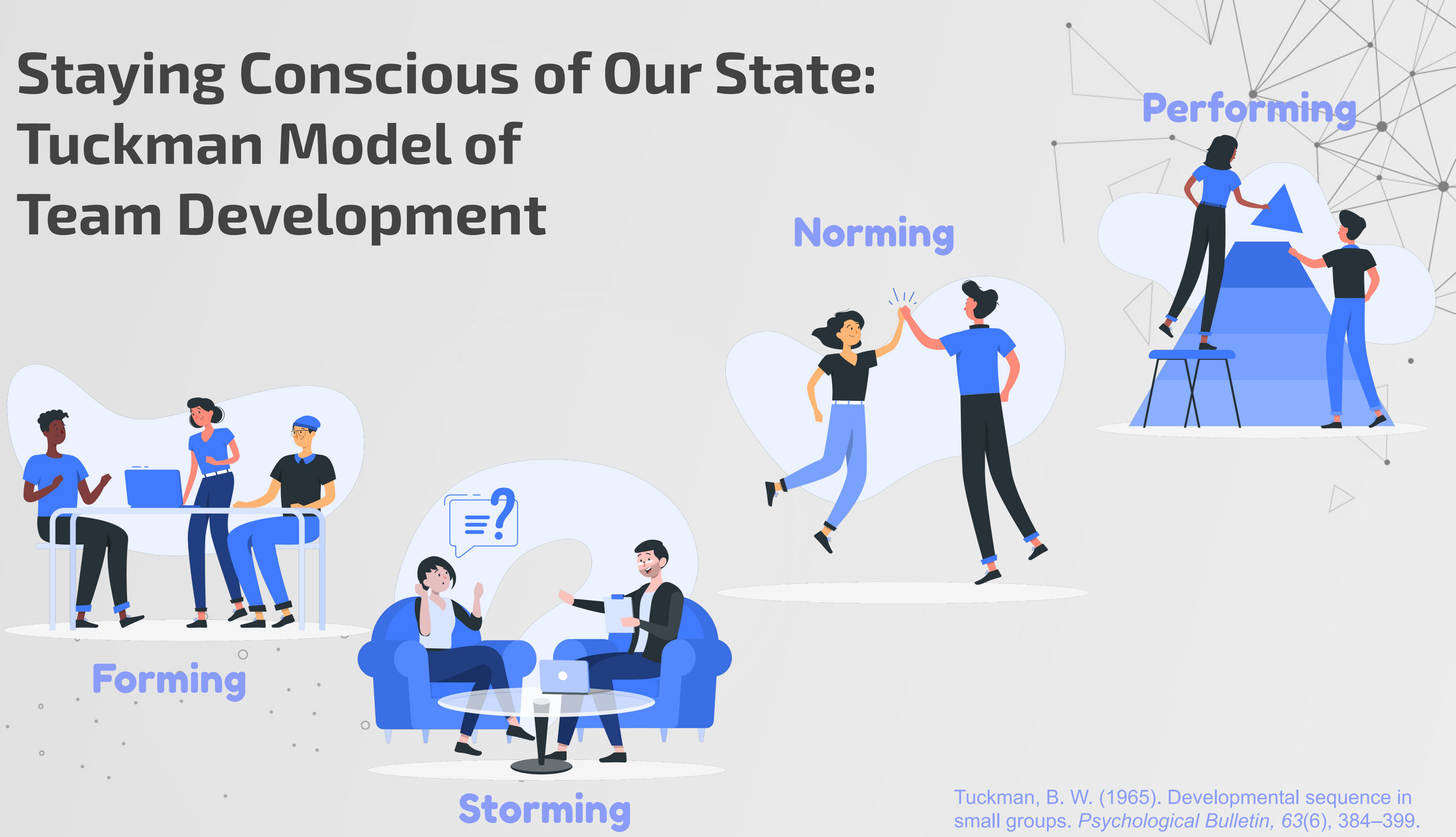
REMAINING TIME
05:00

Who's Up

NAVIGATOR
Andrew

DRIVER
Laura

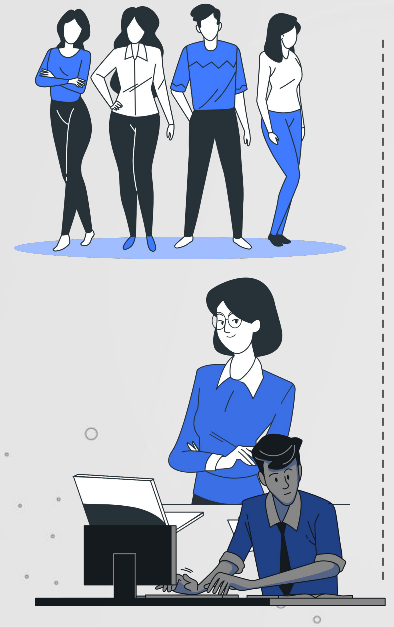
Staying Conscious of Our State: Tuckman Model of Team Development

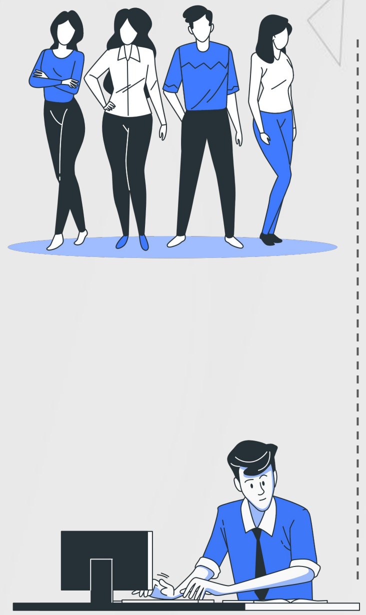
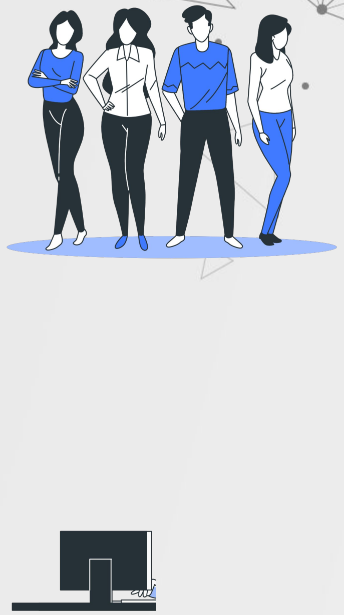


Tuckman, B. W. (1965). Developmental sequence in small groups. *Psychological Bulletin, 63*(6), 384–399.

Ensemble Programming also has different styles.

“The Micromanager” → “The Manager” → “The Facilitator” → “The Ensemble” → “”



2

Refactoring the Code

Improving the Code’s structure, without changing its behavior.

2

Refactoring the Code

Improving the Code's structure, without changing its behavior.

a. "Extract to Method"

- Select a section of code that you think would be helpful to split out.
- Run your test suite, confirm all is passing.
- Make it into a function with all variables as parameters or return values, and call the function where it was originally used.
- Re-run your tests, confirm all is still passing.

b. "Extract to Parameter"

- Run your test suite, confirm all is passing.
- Find hard-coded variables and move them to be parameters in the function definition.
- Re-run your tests, confirm all is still passing.

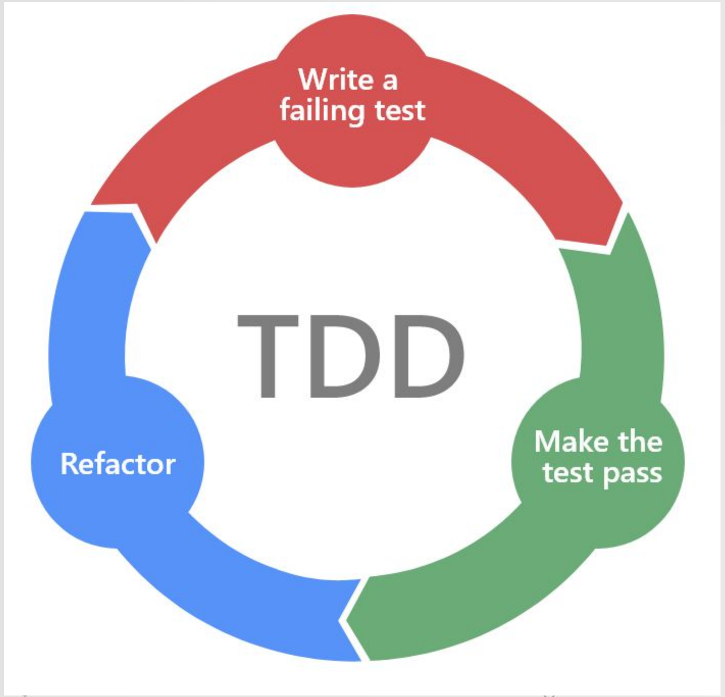
c. "Merge to Data Structure"

- Run your test suite, confirm all is passing
- Identify large groups of variables being passed around together
- Put variables together into a data structure, updating any code dependencies to the new change.
- Update tests.
- Re-run your tests, confirm all is still passing.

Designing in a Group

Test-Driven Development

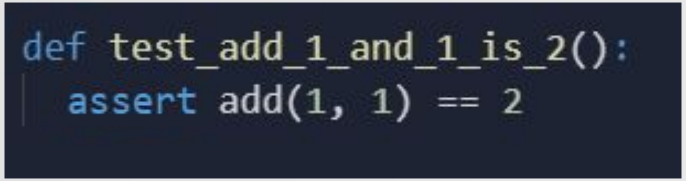
Test-Driven Development



https://marsner.com/wp-content/uploads/test-driven-development-TDD.png

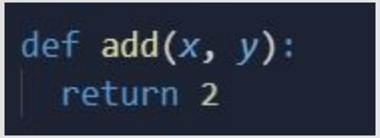
Test-Driven Development

Test-Driven Development

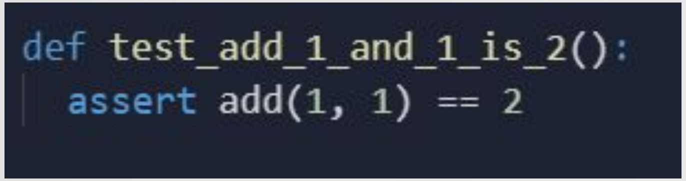


```python
def test_add_1_and_1_is_2():
  assert add(1, 1) == 2
```

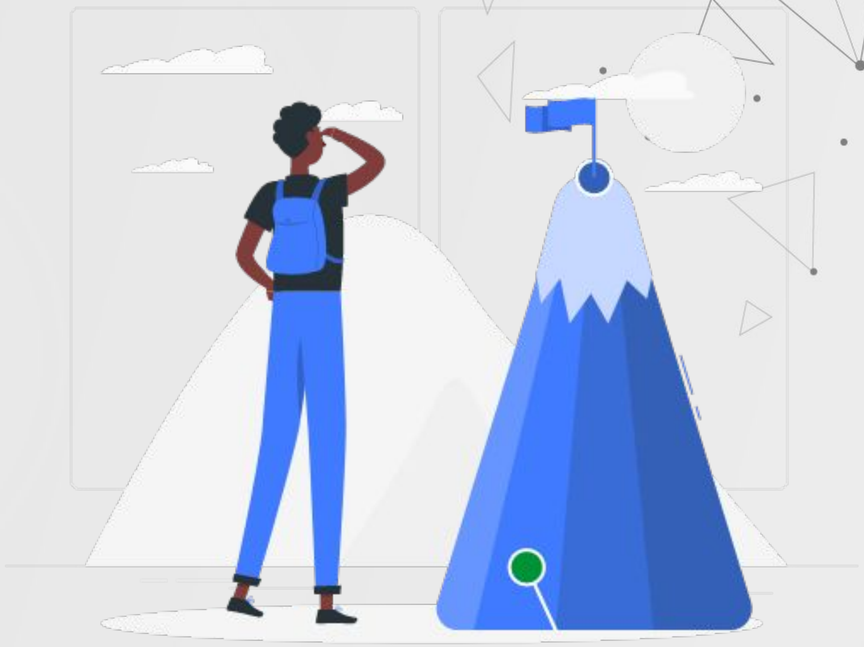
Test-Driven Development



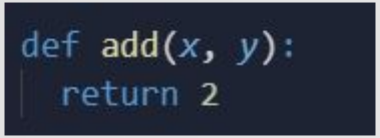
```
def add(x, y):
  return 2
```



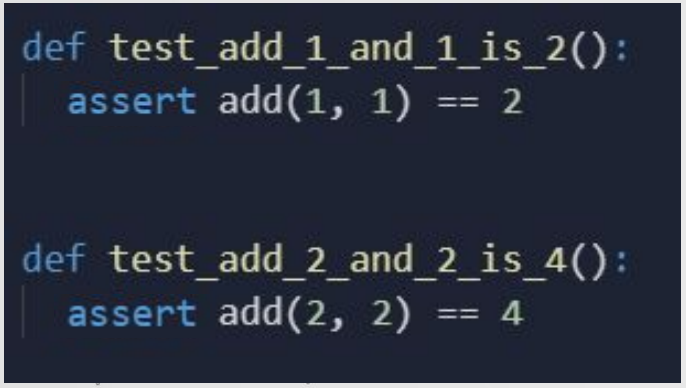
```
def test_add_1_and_1_is_2():
  assert add(1, 1) == 2
```



Test-Driven Development

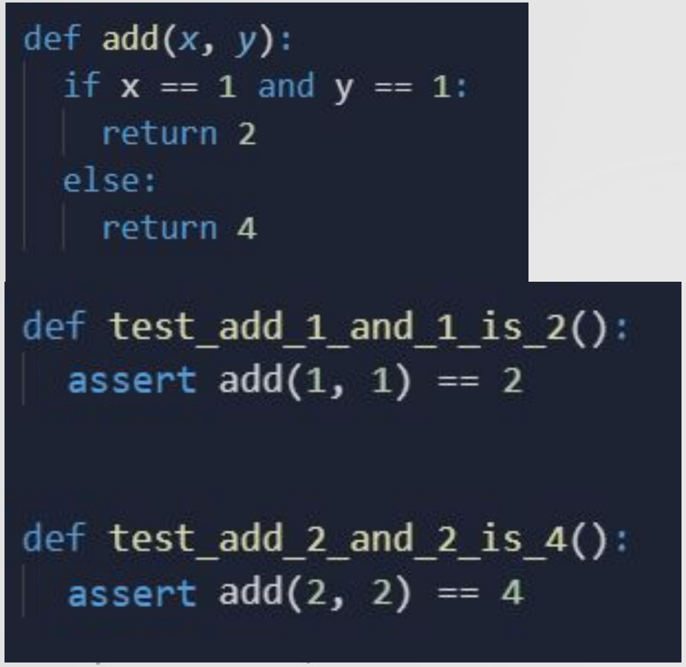


```python
def add(x, y):
  return 2
```



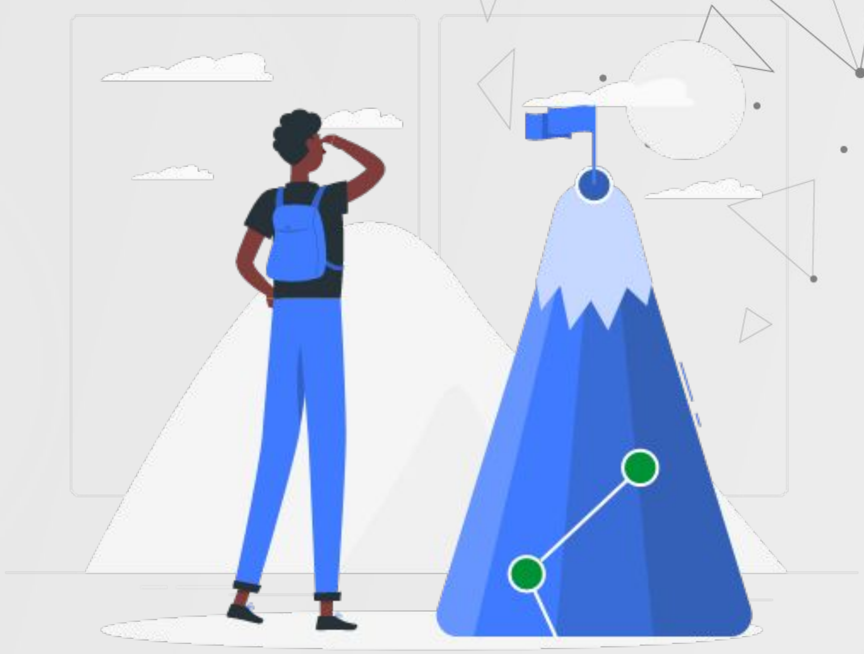
```python
def test_add_1_and_1_is_2():
  assert add(1, 1) == 2


def test_add_2_and_2_is_4():
  assert add(2, 2) == 4
```

Test-Driven Development

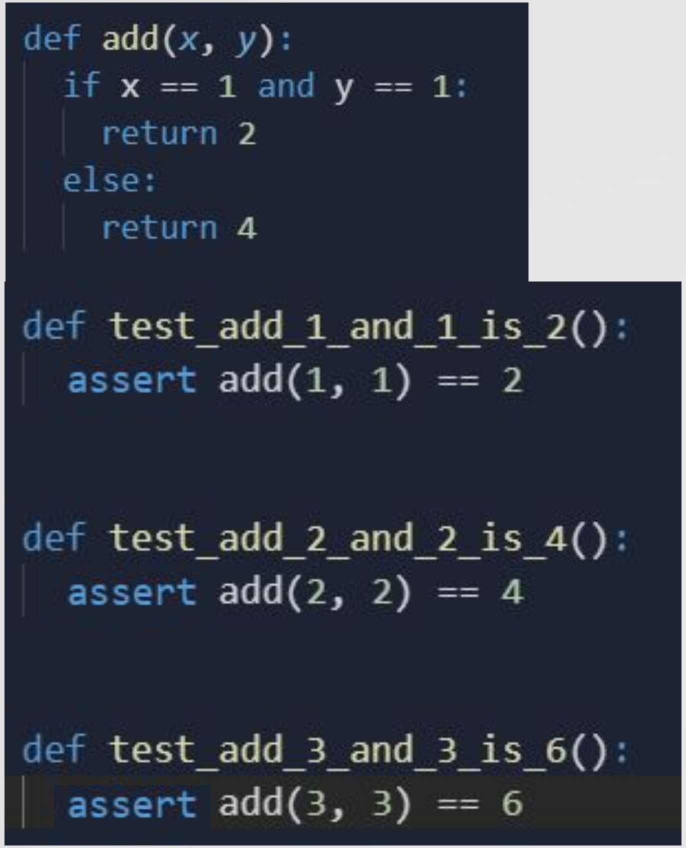


```python
def add(x, y):
  if x == 1 and y == 1:
    return 2
  else:
    return 4
```

```python
def test_add_1_and_1_is_2():
  assert add(1, 1) == 2


def test_add_2_and_2_is_4():
  assert add(2, 2) == 4
```

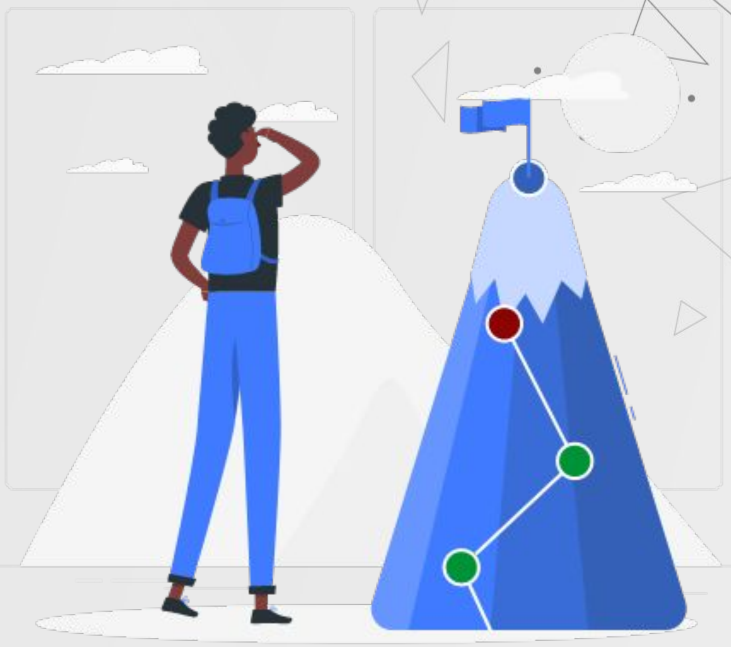


Test-Driven Development

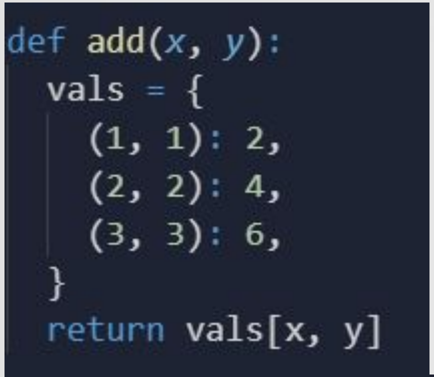


```python
def add(x, y):
    if x == 1 and y == 1:
        return 2
    else:
        return 4
```

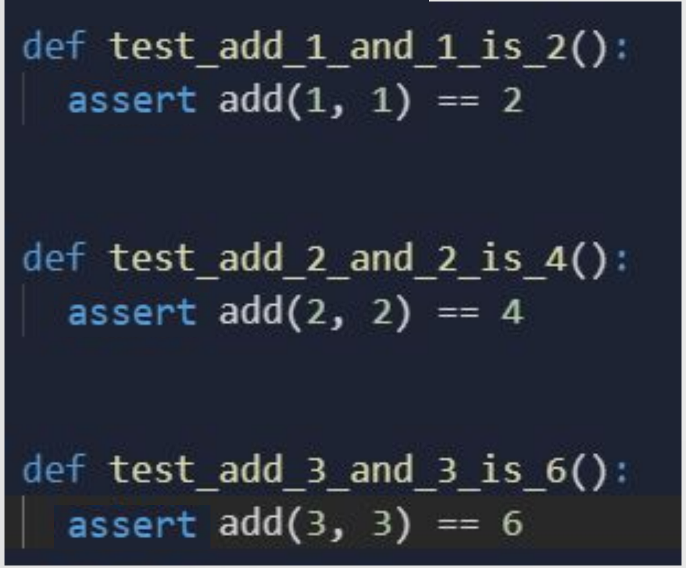
```python
def test_add_1_and_1_is_2():
    assert add(1, 1) == 2


def test_add_2_and_2_is_4():
    assert add(2, 2) == 4


def test_add_3_and_3_is_6():
    assert add(3, 3) == 6
```



Test-Driven Development

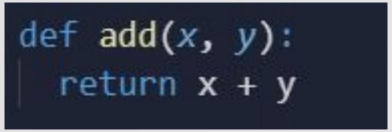


```python
def add(x, y):
  vals = {
    (1, 1): 2,
    (2, 2): 4,
    (3, 3): 6,
  }
  return vals[x, y]
```

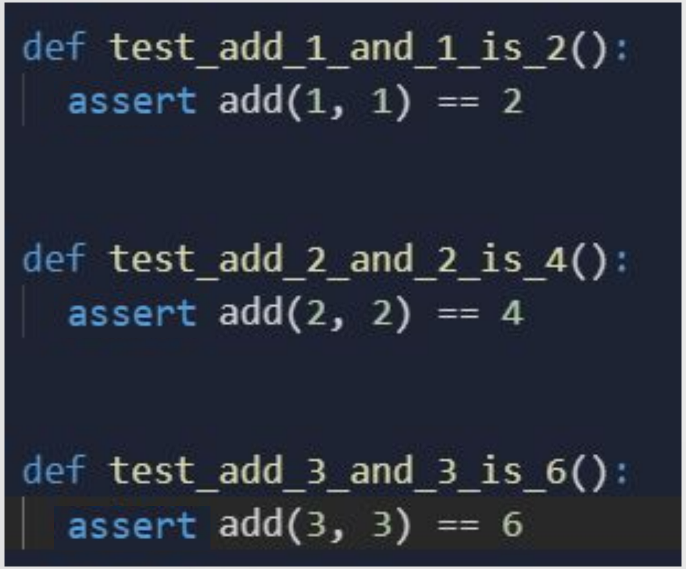


```python
def test_add_1_and_1_is_2():
  assert add(1, 1) == 2


def test_add_2_and_2_is_4():
  assert add(2, 2) == 4


def test_add_3_and_3_is_6():
  assert add(3, 3) == 6
```

Test-Driven Development

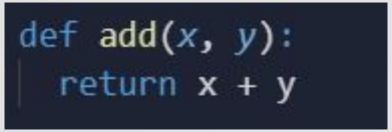


```python
def add(x, y):
  return x + y
```

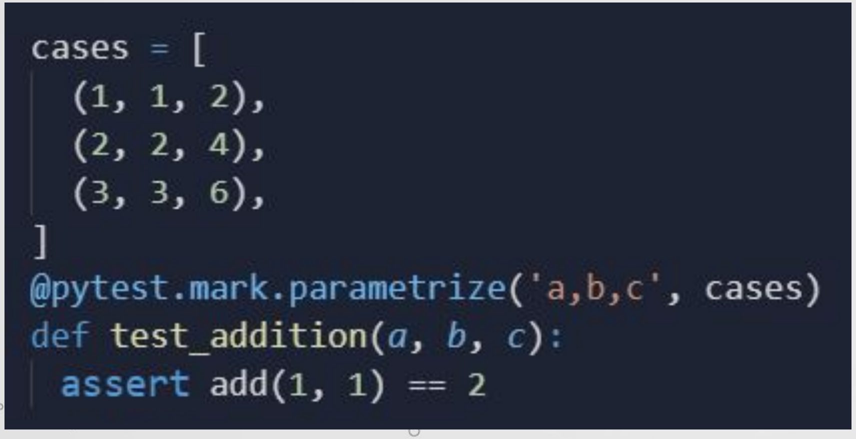


```python
def test_add_1_and_1_is_2():
  assert add(1, 1) == 2


def test_add_2_and_2_is_4():
  assert add(2, 2) == 4


def test_add_3_and_3_is_6():
  assert add(3, 3) == 6
```

Test-Driven Development

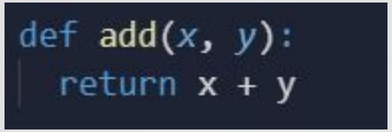


```python
def add(x, y):
  return x + y
```



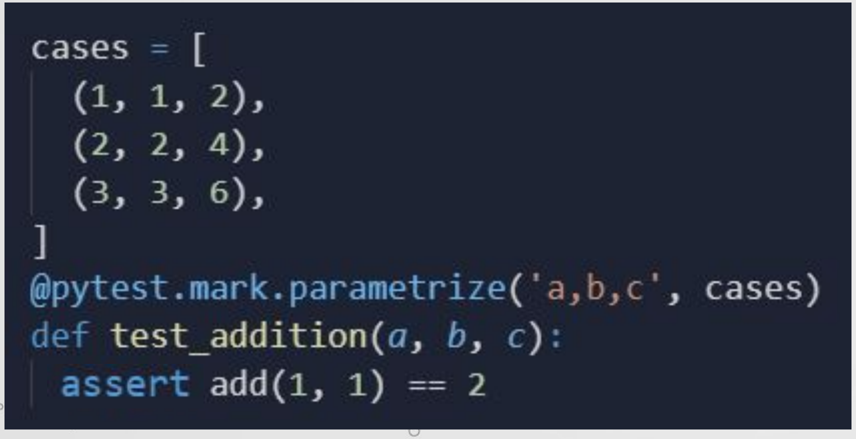
```python
cases = [
  (1, 1, 2),
  (2, 2, 4),
  (3, 3, 6),
]
@pytest.mark.parametrize('a,b,c', cases)
def test_addition(a, b, c):
 assert add(1, 1) == 2
```

Test-Driven Development



```python
def add(x, y):
  return x + y
```

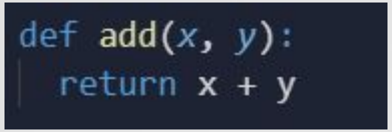
What Does the Code Do?



```python
cases = [
  (1, 1, 2),
  (2, 2, 4),
  (3, 3, 6),
]
@pytest.mark.parametrize('a,b,c', cases)
def test_addition(a, b, c):
 assert add(1, 1) == 2
```

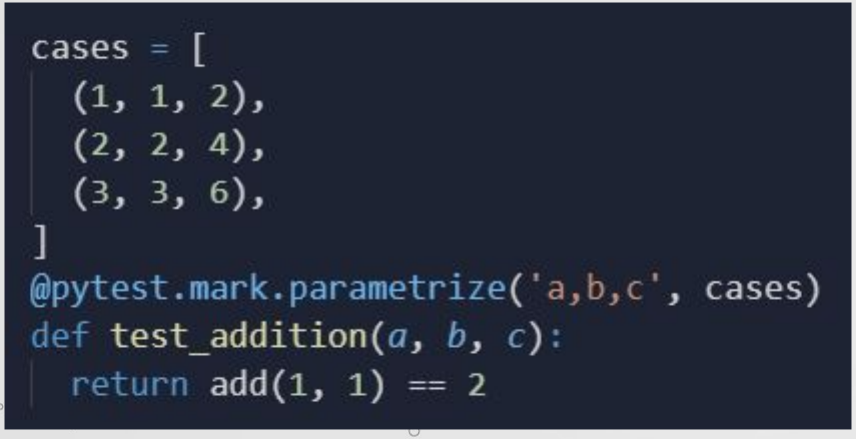
Test-Driven Development

How Does the Code Work?



```
def add(x, y):
  return x + y
```

What Does the Code Do?



```
cases = [
  (1, 1, 2),
  (2, 2, 4),
  (3, 3, 6),
]
@pytest.mark.parametrize('a,b,c', cases)
def test_addition(a, b, c):
  return add(1, 1) == 2
```

3

“Proving” the Software

Using Tests to Document Features: What do we know the software can do?

3

“Proving” the Software

Using Tests to Document Features: What do we know the software can do?

a. Find Code That Should Be Robust

- Identify areas of the code that are, as yet, untested and aren’t totally obvious that it works properly in all relevant situations.
- Extract out the portion that you’d like to test into a new function.
- Write test cases on that function until the code’s robustness is clear. Improve the source code where bugs are found.
- Re-run your tests, confirm all is passing

b. Find Features that Should Be Advertised

- Identify a feature that you are proud of for the software.
- Demonstrate that it works in a close-to-real world setting by writing a test case with real-world data, including the data in your repo.
- Re-run your tests, confirm all is passing

c. “Inject” Dependencies that Shouldn’t be Tested

- Identify hard-to-test code (because they depend on some complex system)
- Move the functions called to parameters in the function, to enable dependency mocking in tests.
- Add a test, replacing the hard-to-test code with easy-to-test code.
- Re-run your tests, confirm all is still passing.