Why unit test?
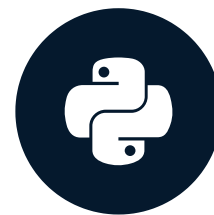
UNIT TESTING FOR DATA SCIENCE IN PYTHON

Dibya Chakravorty
Test Automation Engineer

How can we test an implementation?

```python
def my_function(argument):
    ...
```

✔️

😓

```
my_function(argument_1)
```

```
return_value_1
```

```
my_function(argument_2)
```
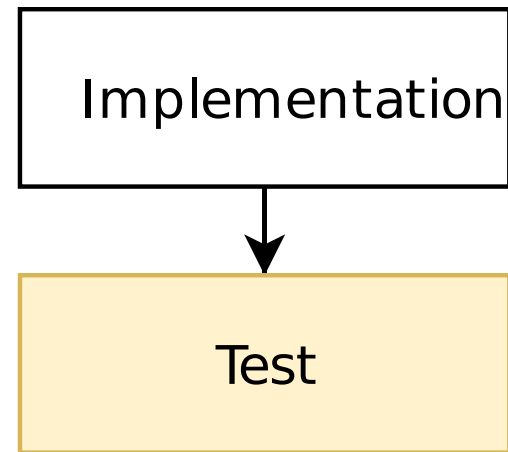
```
return_value_2
```

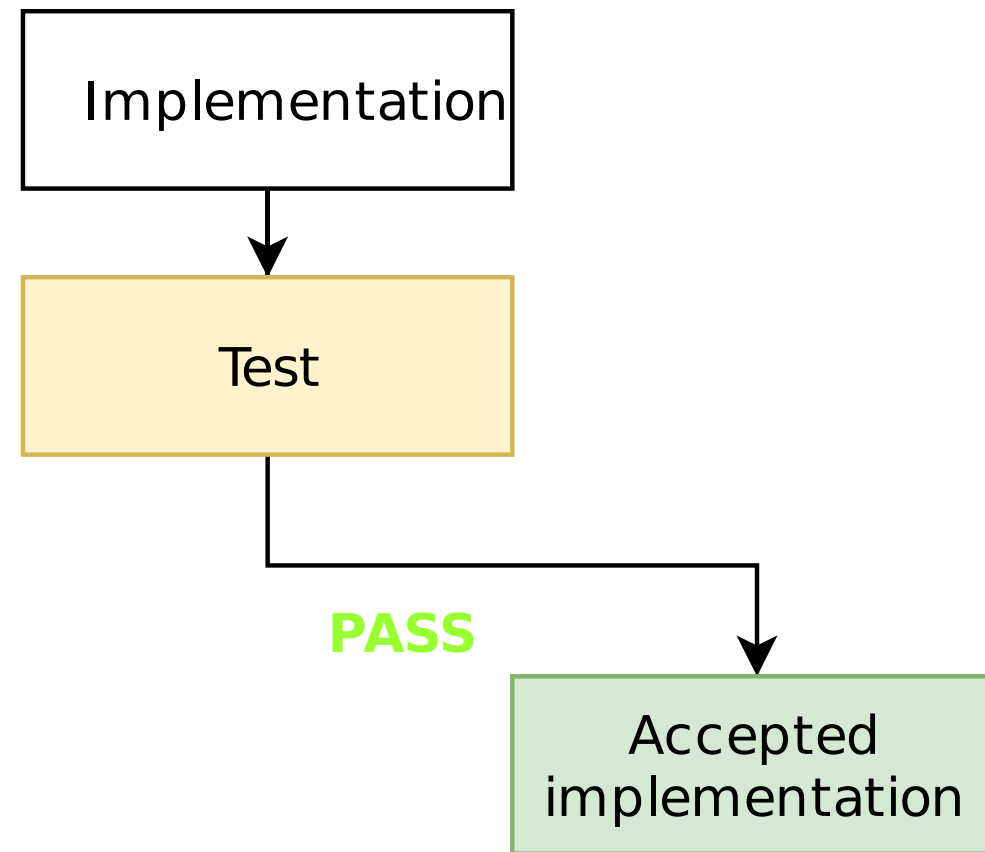```
my_function(argument_3)
```

```
return_value_3
```
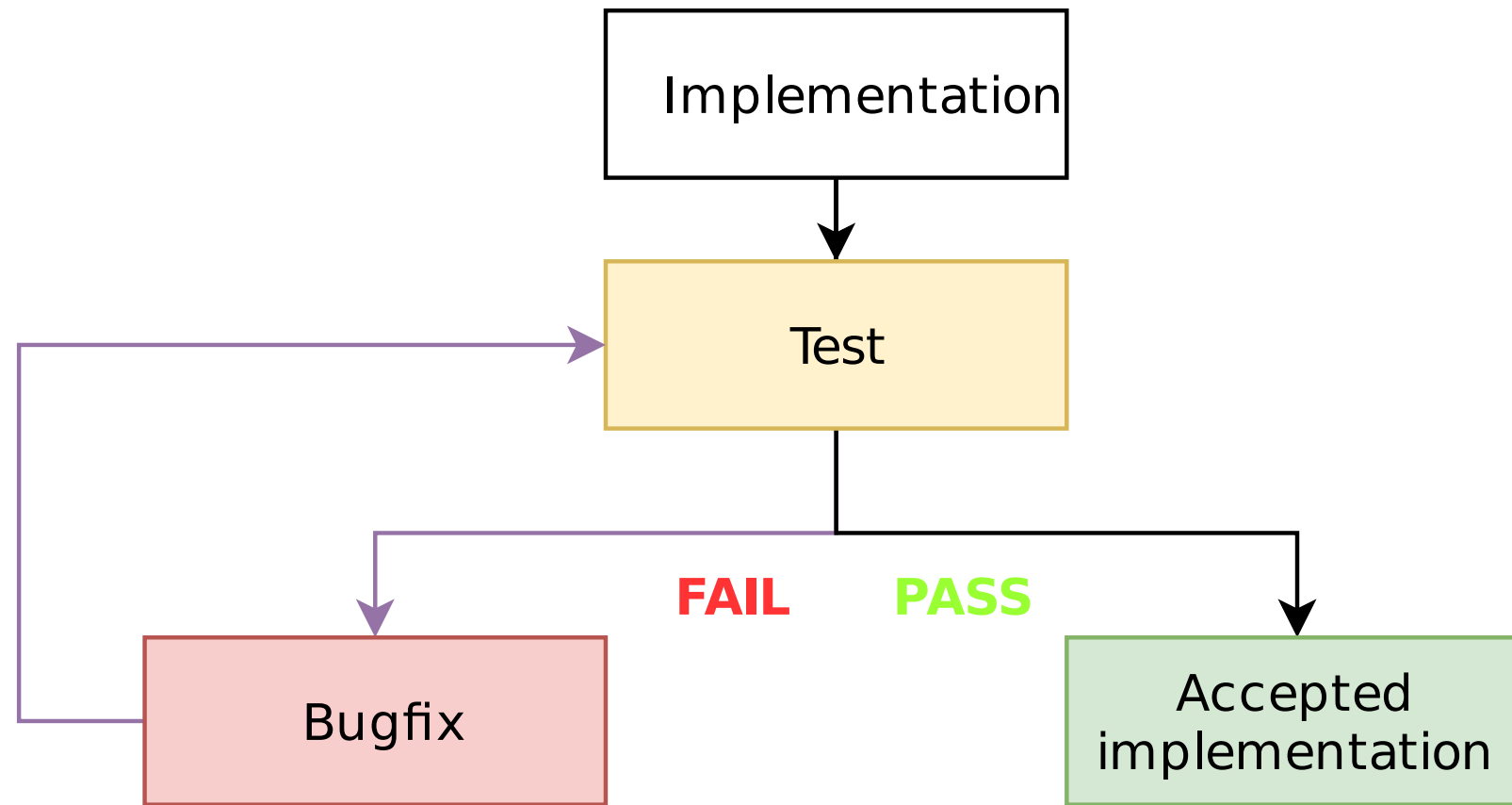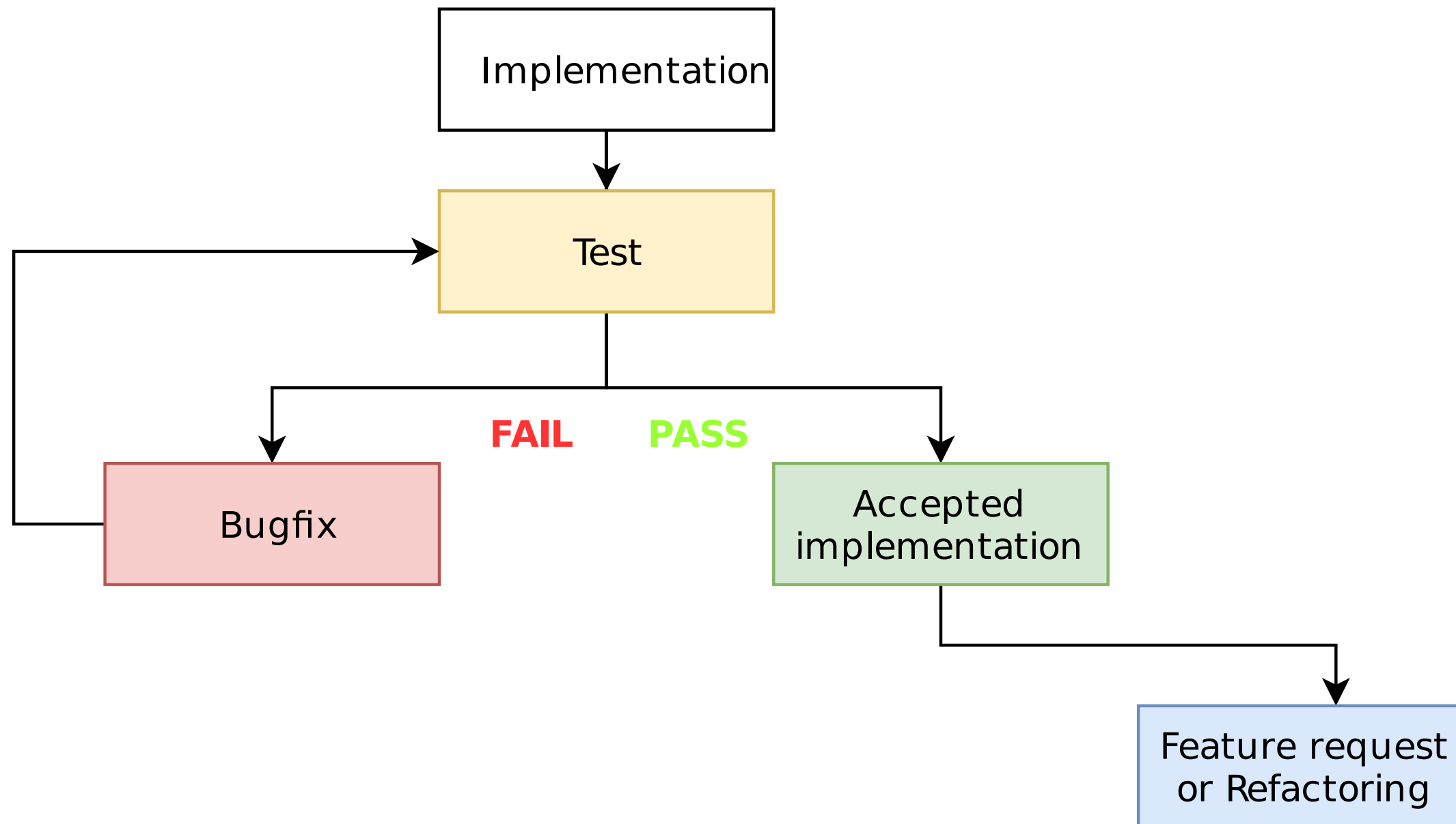
# Life cycle of a function

Implementation

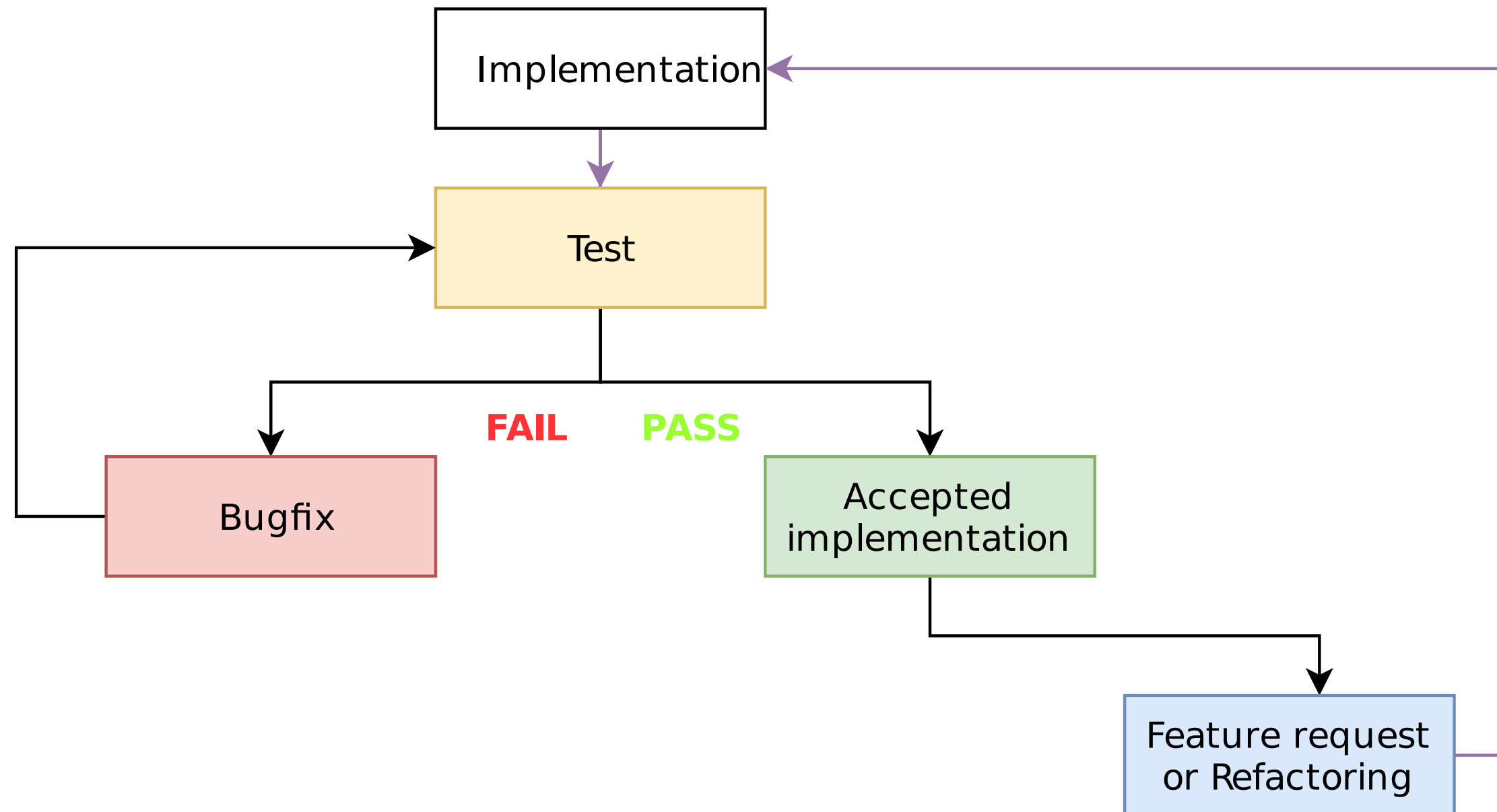# Life cycle of a function

# Life cycle of a function

# Life cycle of a function

# Life cycle of a function

# Life cycle of a function

# Life cycle of a function

# Life cycle of a function

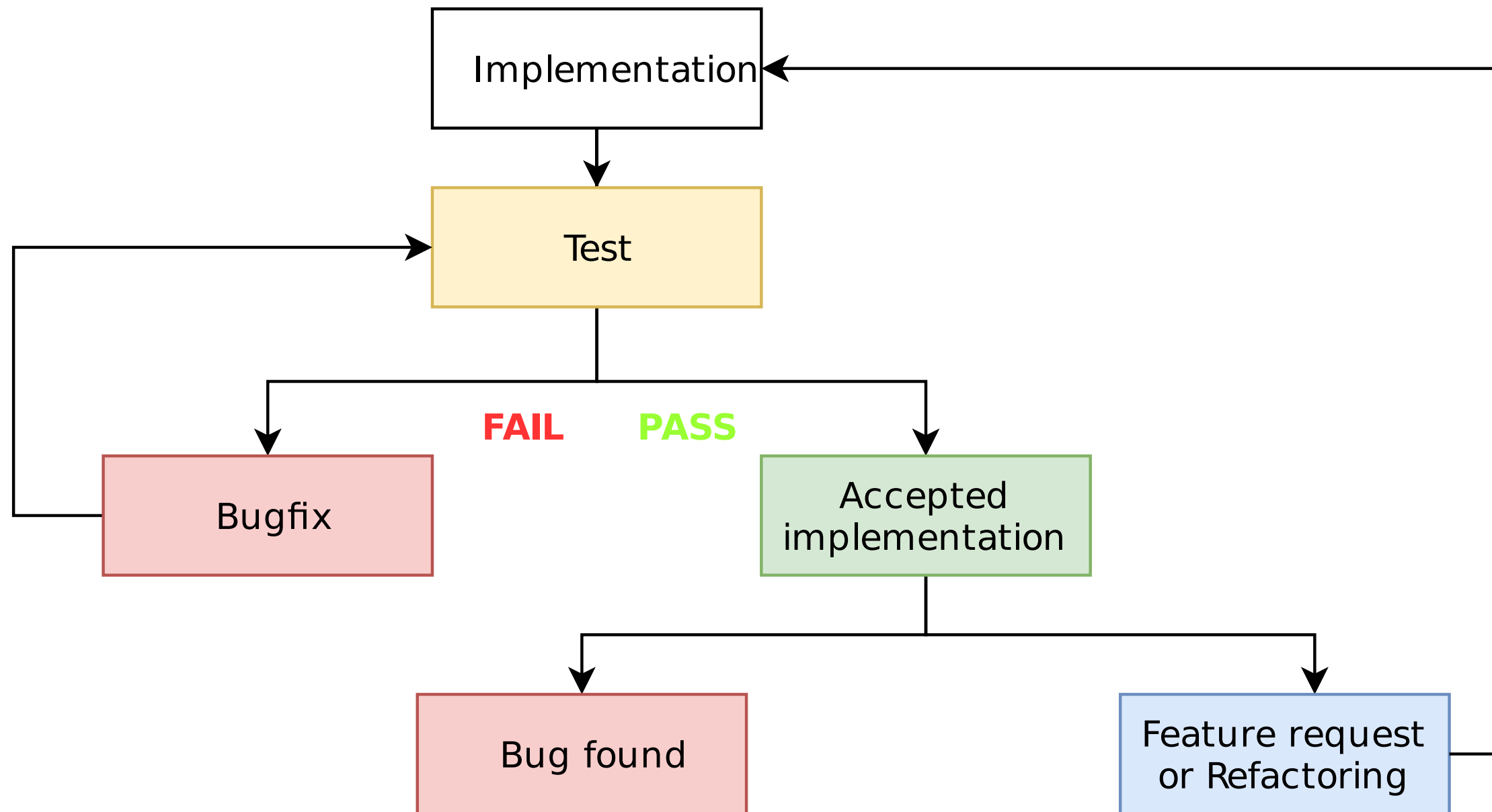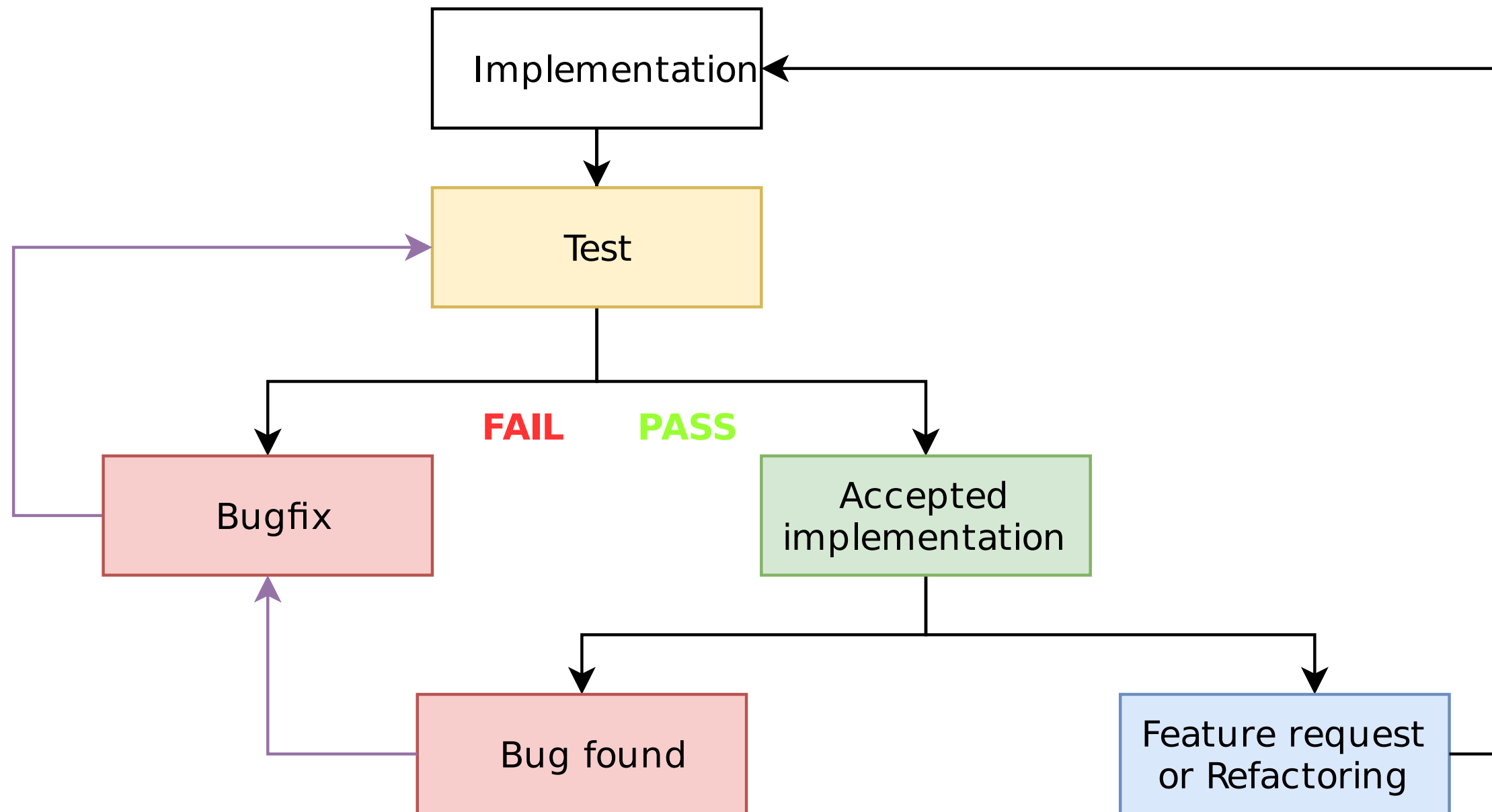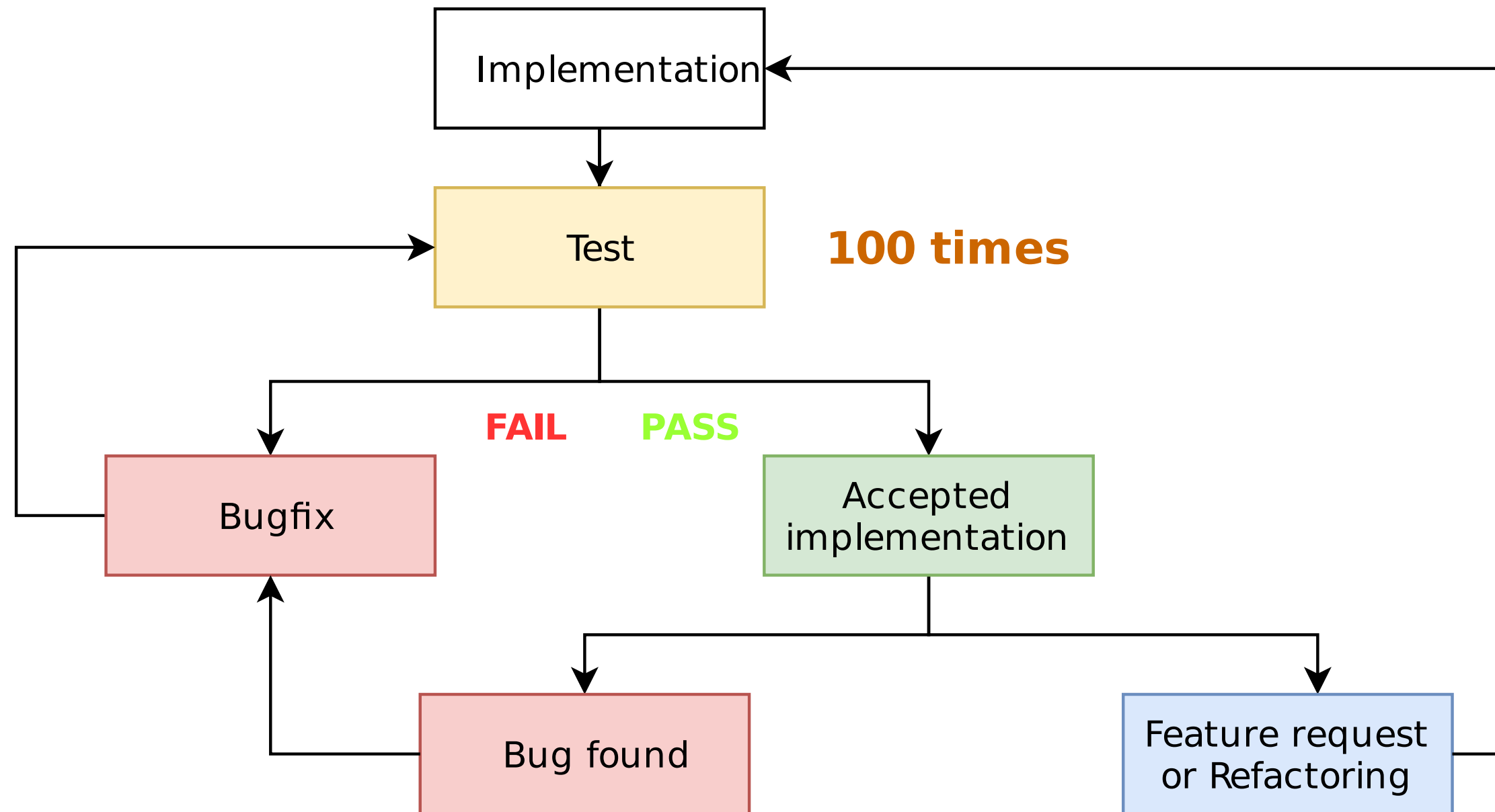# Life cycle of a function

# Example

```
def row_to_list(row):
    ...
```

```
area (sq. ft.)  price (dollars)
2,081     314,942
1,059     186,606
          293,410
1,148     206,186
1,506     248,419
1,210     214,114
1,697     277,794
1,268     194,345
2,318     372,162
1,463238,765
1,468     239,007
```

File: housing_data.txt

Data format

```python
def row_to_list(row):
    ...
```

Argument	Type	Return value
`"2,081\t314,942\n"`	Valid	`["2,081", "314,942"]`

```
area (sq. ft.)  price (dollars)
2,081      314,942
1,059      186,606
           293,410
1,148      206,186
1,506      248,419
1,210      214,114
1,697      277,794
1,268      194,345
2,318      372,162
1,463238,765
1,468      239,007
```

File: housing_data.txt

Data isn't clean

```python
def row_to_list(row):
    ...
```

Argument	Type	Return value
"2,081\t314,942\n"	Valid	["2,081", "314,942"]
"\t293,410\n"	Invalid	None

```
area (sq. ft.)  price (dollars)
2,081     314,942
1,059     186,606
          293,410      <-- row with missing area
1,148     206,186
1,506     248,419
1,210     214,114
1,697     277,794
1,268     194,345
2,318     372,162
1,463238,765
1,468     239,007
```

File: housing_data.txt

Data isn't clean

```python
def row_to_list(row):
    ...
```

Argument	Type	Return value
"2,081\t314,942\n"	Valid	["2,081", "314,942"]
"\t293,410\n"	Invalid	None
"1,463238,765\n"	Invalid	None

```
area (sq. ft.)  price (dollars)
2,081     314,942
1,059     186,606
          293,410    <-- row with missing area
1,148     206,186
1,506     248,419
1,210     214,114
1,697     277,794
1,268     194,345
2,318     372,162
1,463238,765    <-- row with missing tab
1,468     239,007
```

File: housing_data.txt

Time spent in testing this function

```python
def row_to_list(row):
    ...
```

Argument	Type	Return value
"2,081\t314,942\n"	Valid	["2,081", "314,942"]
"\t293,410\n"	Invalid	None
"1,463238,765\n"	Invalid	None

```python
row_to_list("2,081\t314,942\n")
```

```
["2,081", "314,942"]
```
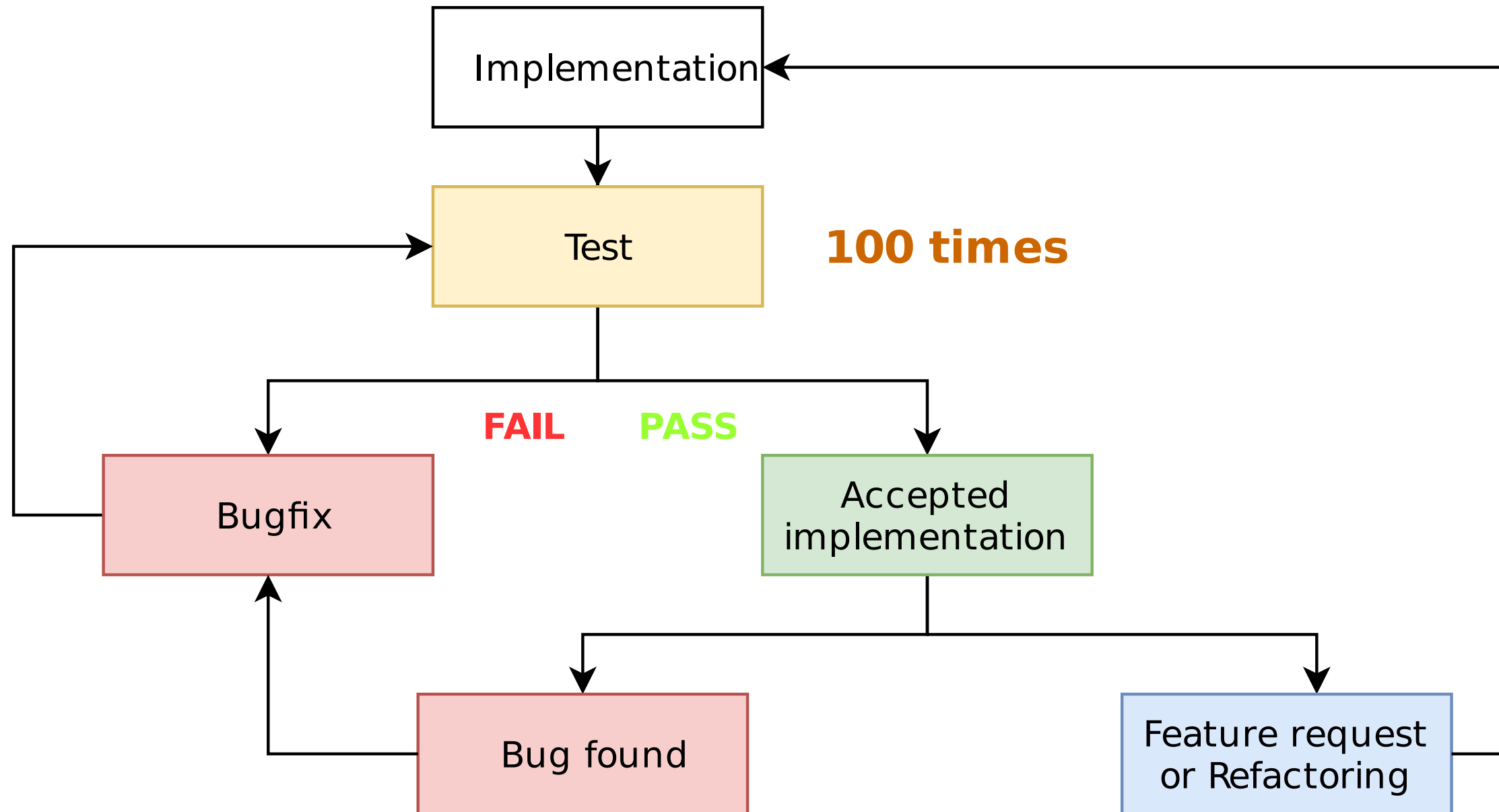
```python
row_to_list("\t293,410\n")
```

```
None
```

```python
row_to_list("1,463238,765\n")
```
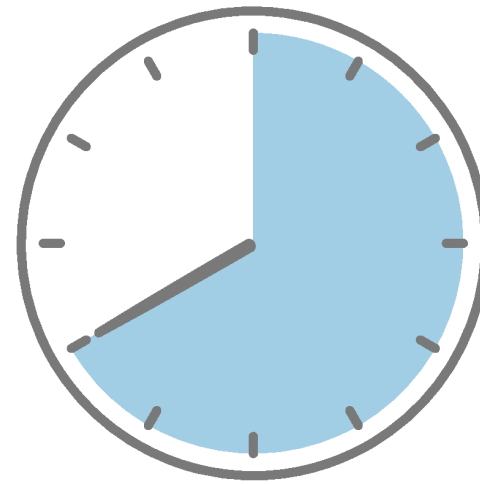
```
None
```

Time spent in testing this function
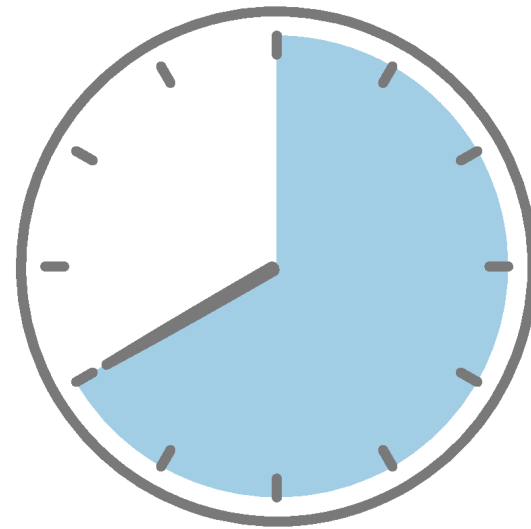
Time spent in testing this function

5 mins x 100 ~
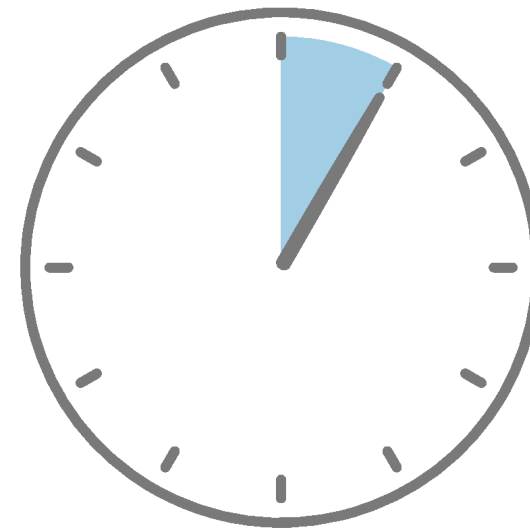

8 hours

Manual testing vs. unit tests

- Unit tests automate the repetitive testing process and saves time.

Manually testing on
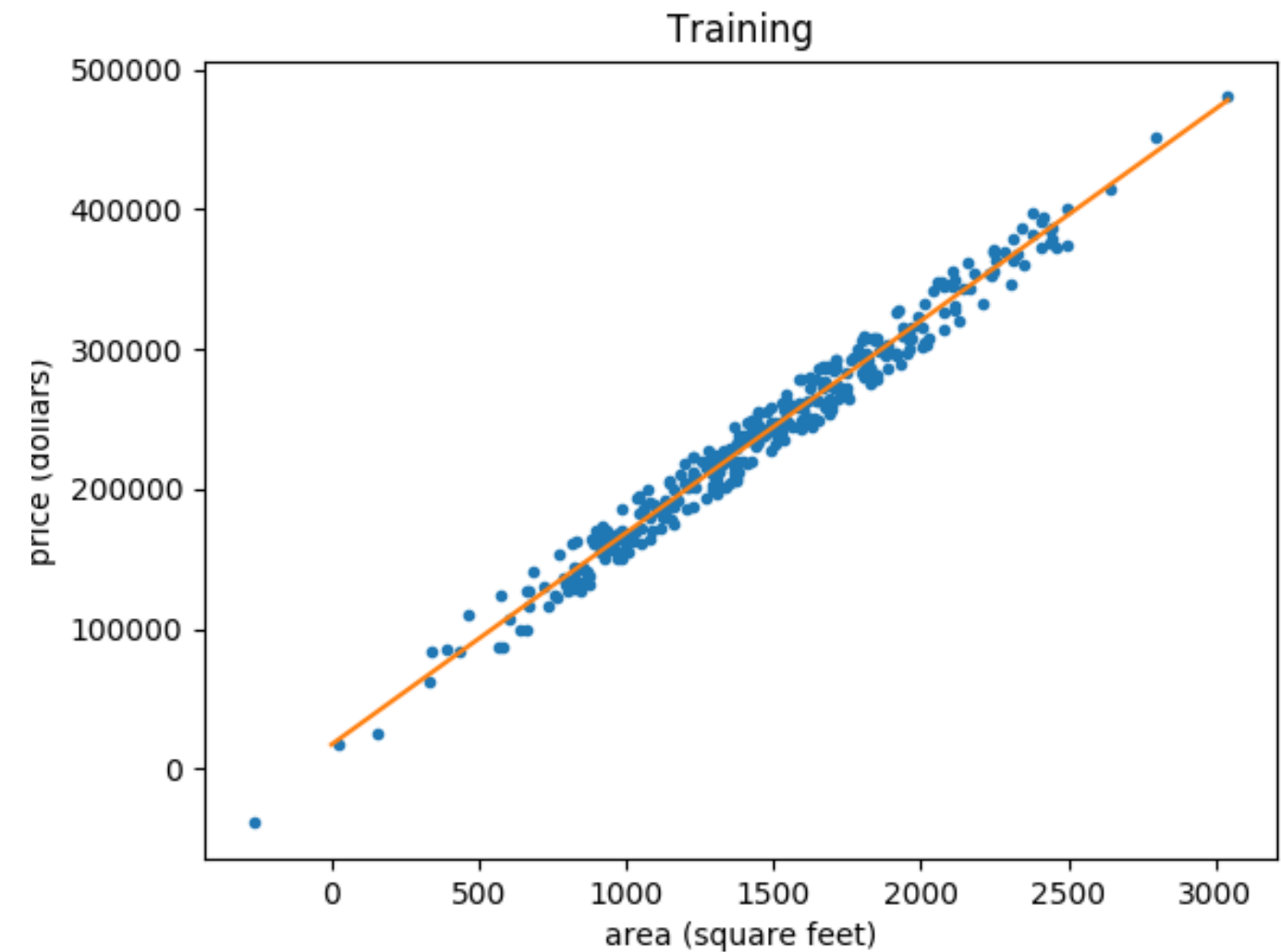the interpreter

Unit tests

8 hours

1 hour

Learn unit testing - with a data science spin
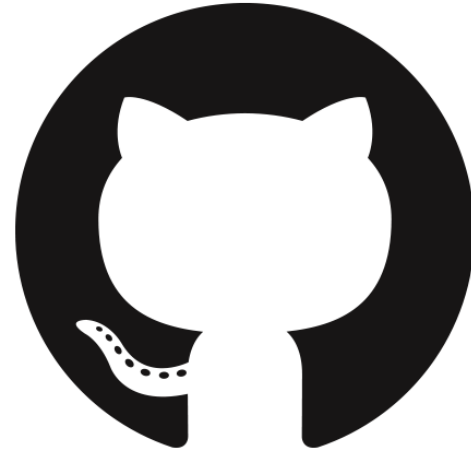
```
area (sq. ft.)    price (dollars)
2,081      314,942
1,059      186,606
           293,410
1,148      206,186
1,506      248,419
1,210      214,114
1,697      277,794
1,268      194,345
2,318      372,162
1,463238,765
1,468      239,007
```


Linear regression of housing price against area

GitHub repository of the course

- Implementation of functions like `row_to_list()`.

Develop a complete unit test suite

```
data/
src/
|-- data/
|-- features/
|-- models/
|-- visualization/
```

Develop a complete unit test suite

```
data/
src/
|-- data/
|-- features/
|-- models/
|-- visualization/
tests/                                          # Test suite
|-- data/
|-- features/
|-- models/
|-- visualization/
```

- Write unit tests for your own projects.

Let's practice these concepts!

UNIT TESTING FOR DATA SCIENCE IN PYTHON

Write a simple unit test using pytest

UNIT TESTING FOR DATA SCIENCE IN PYTHON

Dibya Chakravorty
Test Automation Engineer

Testing on the console

```
row_to_list("2,081\t314,942\n")
```

```
["2,081", "314,942"]
```

```
row_to_list("\t293,410\n")
```

```
None
```

```
row_to_list("1,463238,765\n")
```

```
None
```

- Unit tests improve this process.

Python unit testing libraries

- pytest

- unittest

- nosetests

- doctest

We will use pytest!

- Has all essential features.

- Easiest to use.

- **Most popular.**

Step 1: Create a file

- Create `test_row_to_list.py` .

- `test_` indicate unit tests inside (naming convention).

- Also called **test modules.**

Step 2: Imports

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list
```

Step 3: Unit tests are Python functions

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
```

Step 3: Unit tests are Python functions

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
```

Argument	Type	Return value
`"2,081\t314,942\n"`	Valid	`["2,081", "314,942"]`

Step 4: Assertion

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
    assert ...
```

Argument	Type	Return value
`"2,081\t314,942\n"`	Valid	`["2,081", "314,942"]`

Theoretical structure of an assertion

```
assert boolean_expression
```

```
assert True
```

```

```

```
assert False
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

Step 4: Assertion

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
    assert row_to_list("2,081\t314,942\n") == \
            ["2,081", "314,942"]
```

Argument	Type	Return value
`"2,081\t314,942\n"`	Valid	`["2,081", "314,942"]`

A second unit test

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
  assert row_to_list("2,081\t314,942\n") == \
          ["2,081", "314,942"]


def test_for_missing_area():
  assert row_to_list("\t293,410\n") is None
```

Argument	Type	Return value
`"2,081\t314,942\n"`	Valid	`["2,081", "314,942"]`
`"\t293,410\n"`	Invalid	`None`

Checking for None values

Do this for checking if `var` is `None`.

```
assert var is None
```

Do *not* do this.

```
assert var == None
```

A third unit test

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
    assert row_to_list("2,081\t314,942\n") == \
            ["2,081", "314,942"]


def test_for_missing_area():
    assert row_to_list("\t293,410\n") is None


def test_for_missing_tab():
    assert row_to_list("1,463238,765\n") is None
```

Argument	Type	Return value
"2,081\t314,942\n"	Valid	["2,081", "314,942"]
"\t293,410\n"	Invalid	None
"1,463238,765\n"	Invalid	None

Step 5: Running unit tests

- Do this in the command line.

```
pytest test_row_to_list.py
```

Running unit tests in DataCamp exercises

Running unit tests in DataCamp exercises

Running unit tests in DataCamp exercises

Running unit tests in DataCamp exercises

Next lesson: test result report

Let's write some unit tests!

UNIT TESTING FOR DATA SCIENCE IN PYTHON

Understanding test result report

UNIT TESTING FOR DATA SCIENCE IN PYTHON

Dibya Chakravorty
Test Automation Engineer

Unit tests for row_to_list()

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list

def test_for_clean_row():
    assert row_to_list("2,081\t314,942\n") == \
            ["2,081", "314,942"]


def test_for_missing_area():
    assert row_to_list("\t293,410\n") is None


def test_for_missing_tab():
    assert row_to_list("1,463238,765\n") is None
```
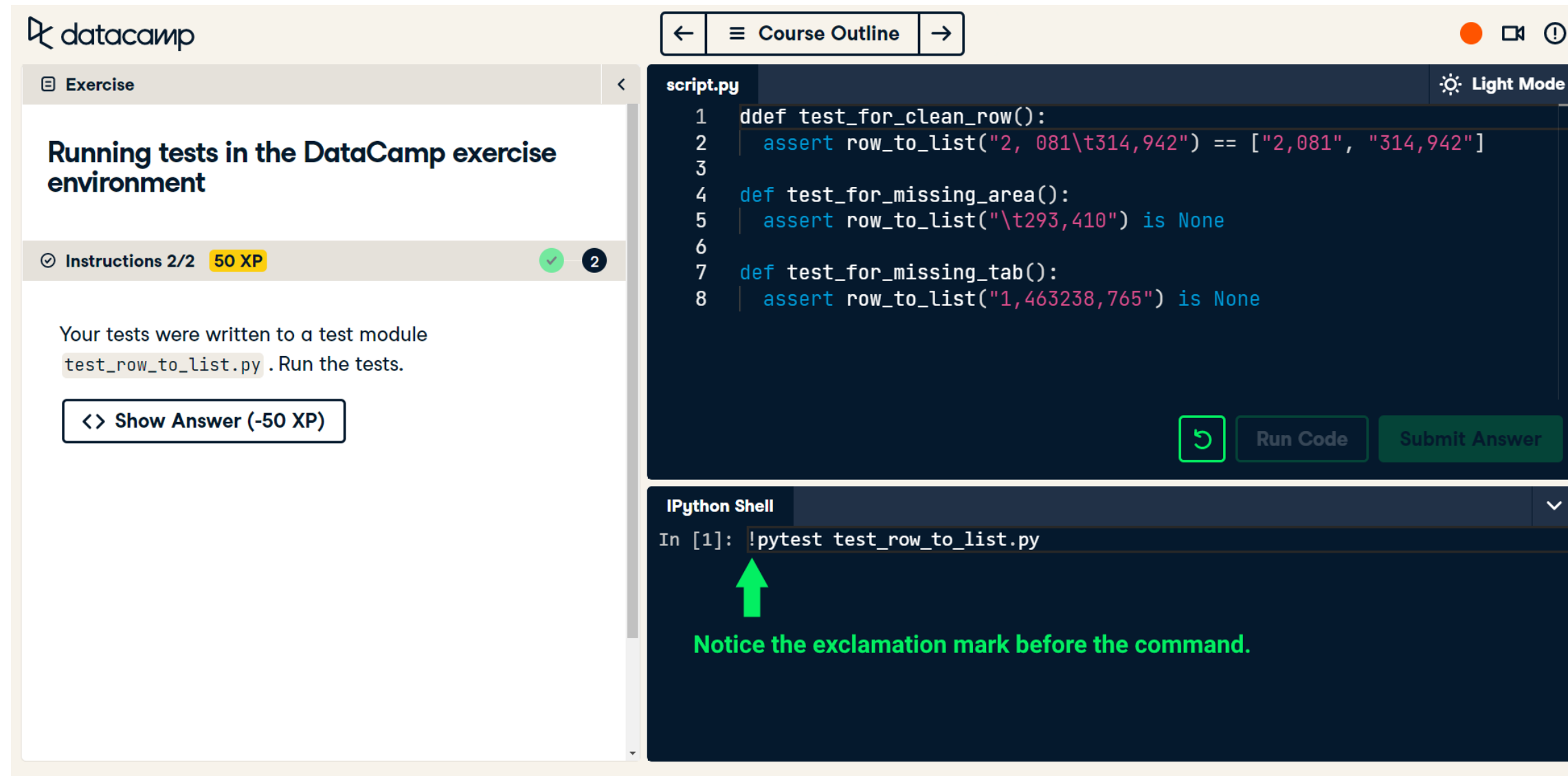
Argument	Type	Return value
"2,081\t314,942\n"	Valid	["2,081", "314,942"]
"\t293,410\n"	Invalid	None
"1,463238,765\n"	Invalid	None

Test result report

```
!pytest test_row_to_list.py
```

```
============================ test session starts ============================
platform linux -- Python 3.6.7, pytest-4.0.1, py-1.8.0, pluggy-0.9.0
rootdir: /tmp/tmpvdblq9g7, inifile:
plugins: mock-1.10.0
collecting ...
collected 3 items


test_row_to_list.py .F.                                               [100%]


================================== FAILURES =================================
_____ test_for_missing_area _____


    def test_for_missing_area():
>       assert row_to_list("\t293,410\n") is None
```

Section 1: general information

```
======================== test session starts ========================
platform linux -- Python 3.6.7, pytest-4.0.1, py-1.8.0, pluggy-0.9.0
rootdir: /tmp/tmpvdblq9g7, inifile:
plugins: mock-1.10.0
```

Section 2: Test result

```
collecting ...
collected 3 items


test_row_to_list.py .F.                                    [100%]
```

Section 2: Test result

```
collecting ...
collected 3 items


test_row_to_list.py .F.                              [100%]
```

Character	Meaning	When	Action
F	Failure	An exception is raised when running unit test.	Fix the function or unit test.

Section 2: Test result

```
collecting ...
collected 3 items


test_row_to_list.py .F.                              [100%]
```

Character	Meaning	When	Action
F	Failure	An exception is raised when running unit test.	Fix the function or unit test.

- assertion raises `AssertionError`

```python
def test_for_missing_area():
    assert row_to_list("\t293,410") is None    # AssertionError from this line
```

Section 2: Test result

```
collecting ...
collected 3 items



test_row_to_list.py .F.                                    [100%]
```

Character	Meaning	When	Action
F	Failure	An exception is raised when running unit test.	Fix the function or unit test.

- another exception

```python
def test_for_missing_area():
    assert row_to_list("\t293,410") is none    # NameError from this line
```

Section 2: Test result

```
collecting ...
collected 3 items


test_row_to_list.py .F.                         [100%]
```

Character	Meaning	When	Action
F	Failure	An exception is raised when running unit test.	Fix the function or unit test.
.	Passed	No exception raised when running unit test	Everything is fine. Be happy!

Section 3: Information on failed tests

```
========================== FAILURES ==========================
_____ test_for_missing_area _____


    def test_for_missing_area():
>       assert row_to_list("\t293,410\n") is None
E       AssertionError: assert ['', '293,410'] is None
E        +  where ['', '293,410'] = row_to_list('\t293,410\n')


test_row_to_list.py:7: AssertionError
```

- The line raising the exception is marked by `>` .

```
>       assert row_to_list("\t293,410\n") is None
```

Section 3: Information on failed tests

```
================================ FAILURES ================================
_____ test_for_missing_area _____


    def test_for_missing_area():
>       assert row_to_list("\t293,410\n") is None
E       AssertionError: assert ['', '293,410'] is None
E        +  where ['', '293,410'] = row_to_list('\t293,410\n')


test_row_to_list.py:7: AssertionError
```

- the exception is an `AssertionError`.

```
E       AssertionError: assert ['', '293,410'] is None
```

Section 3: Information about failed tests

```
================================= FAILURES =================================
_____ test_for_missing_area _____


    def test_for_missing_area():
>       assert row_to_list("\t293,410\n") is None
E       AssertionError: assert ['', '293,410'] is None
E        +  where ['', '293,410'] = row_to_list('\t293,410\n')


test_row_to_list.py:7: AssertionError
```

- the line containing `where` displays return values.

```
E        +  where ['', '293,410'] = row_to_list('\t293,410\n')
```

Section 4: Test result summary

```
==================== 1 failed, 2 passed in 0.03 seconds ====================
```

- Result summary from all unit tests that ran: **1 failed, 2 passed tests.**

- Total time for running tests: **0.03 seconds.**
  - Much faster than testing on the interpreter!

Let's practice reading test result reports

UNIT TESTING FOR DATA SCIENCE IN PYTHON

More benefits and test types

UNIT TESTING FOR DATA SCIENCE IN PYTHON

Dibya Chakravorty
Test Automation Engineer

datacamp

Unit tests serve as documentation

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list


def test_for_clean_row():
    assert row_to_list("2,081\t314,942\n") == \
            ["2,081", "314,942"]


def test_for_missing_area():
    assert row_to_list("\t293,410\n") is None


def test_for_missing_tab():
    assert row_to_list("1,463238,765\n") is None
```

Unit tests serve as documentation

Test module: `test_row_to_list.py`

```python
import pytest
import row_to_list

def test_for_clean_row():
  assert row_to_list("2,081\t314,942\n") == \
          ["2,081", "314,942"]


def test_for_missing_area():
  assert row_to_list("\t293,410\n") is None


def test_for_missing_tab():
  assert row_to_list("1,463238,765\n") is None
```

- Created from the test module

Argument	Return value
`"2,081\t314,942\n"`	`["2,081",` `"314,942"]`
`"\t293,410\n"`	`None`
`"1,463238,765\n"`	`None`

Guess function's purpose by reading unit tests

```
!cat test_row_to_list.py
```

Guess function's purpose by reading unit tests

```
!cat test_row_to_list.py
```

More trust

- Users can run tests and verify that the package works.

Travis CI `passing` AppVeyor `passing` 🚀 Azure Pipelines `succeeded` codecov `85%`
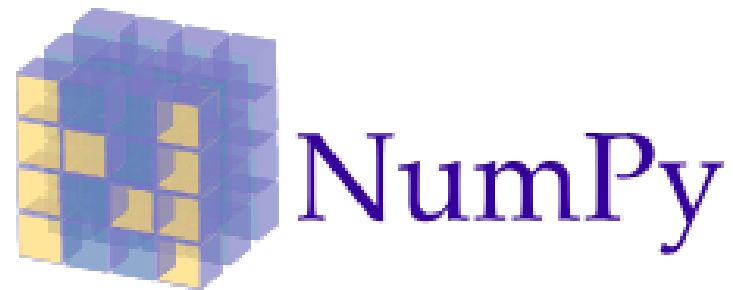
NumPy is the fundamental package needed for scientific computing with Python.

- **Website (including documentation):** https://www.numpy.org
- **Mailing list:** https://mail.python.org/mailman/listinfo/numpy-discussion

More trust

- Users can run tests and verify that the package works.

NumPy

| Travis CI | passing | | AppVeyor | passing | | 🚀 Azure Pipelines | succeeded | | codecov | 85% |

NumPy is the fundamental package needed for scientific computing with Python.

- **Website (including documentation):** https://www.numpy.org
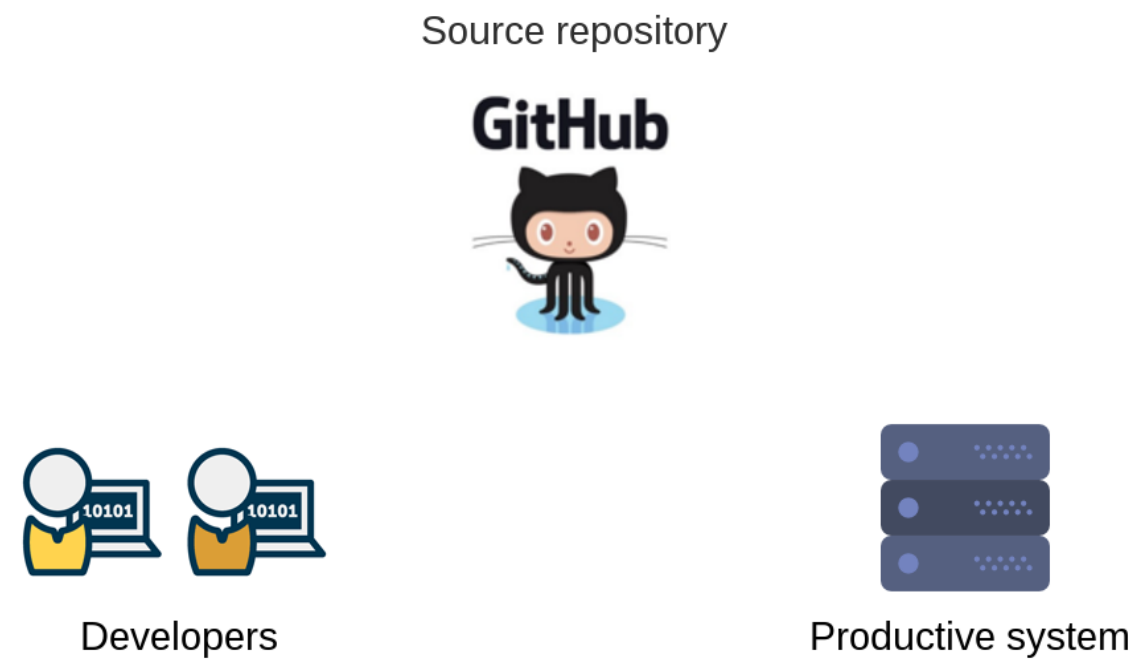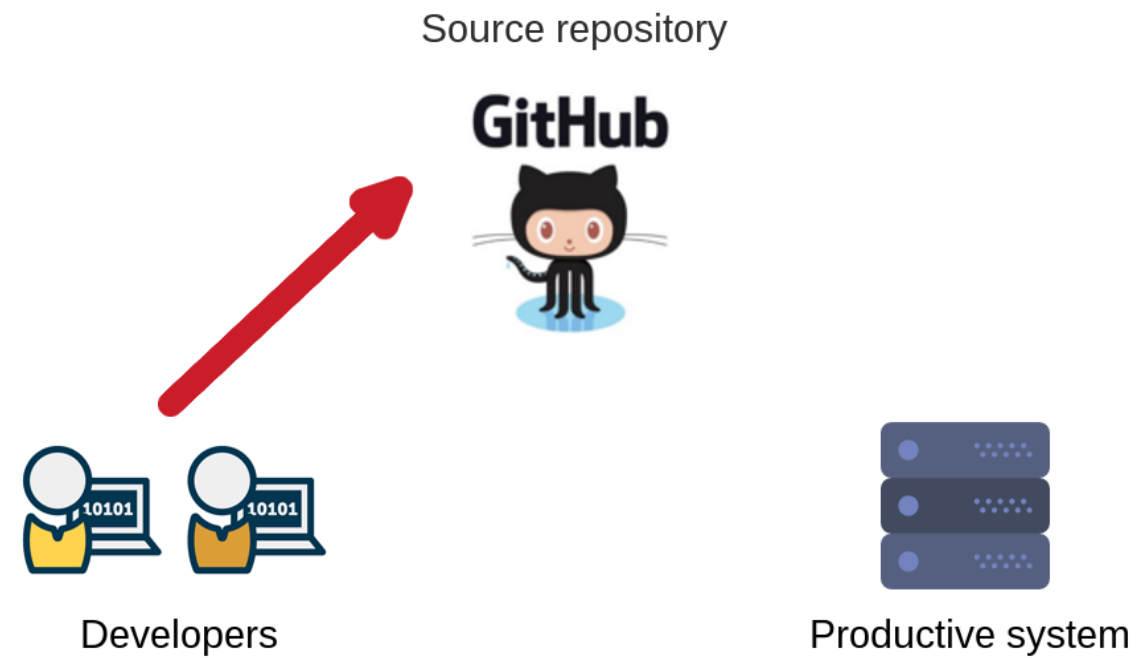- **Mailing list:** https://mail.python.org/mailman/listinfo/numpy-discussion

More trust

- Users can run tests and verify that the package works.

NumPy is the fundamental package needed for scientific computing with Python.
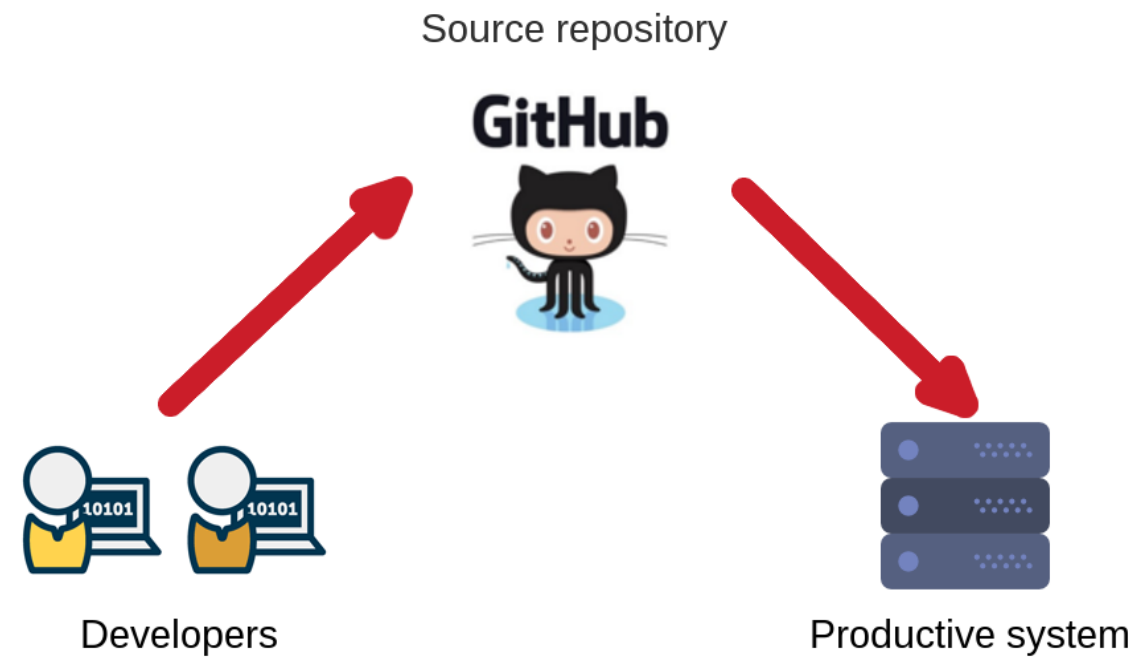
- **Website (including documentation):** https://www.numpy.org
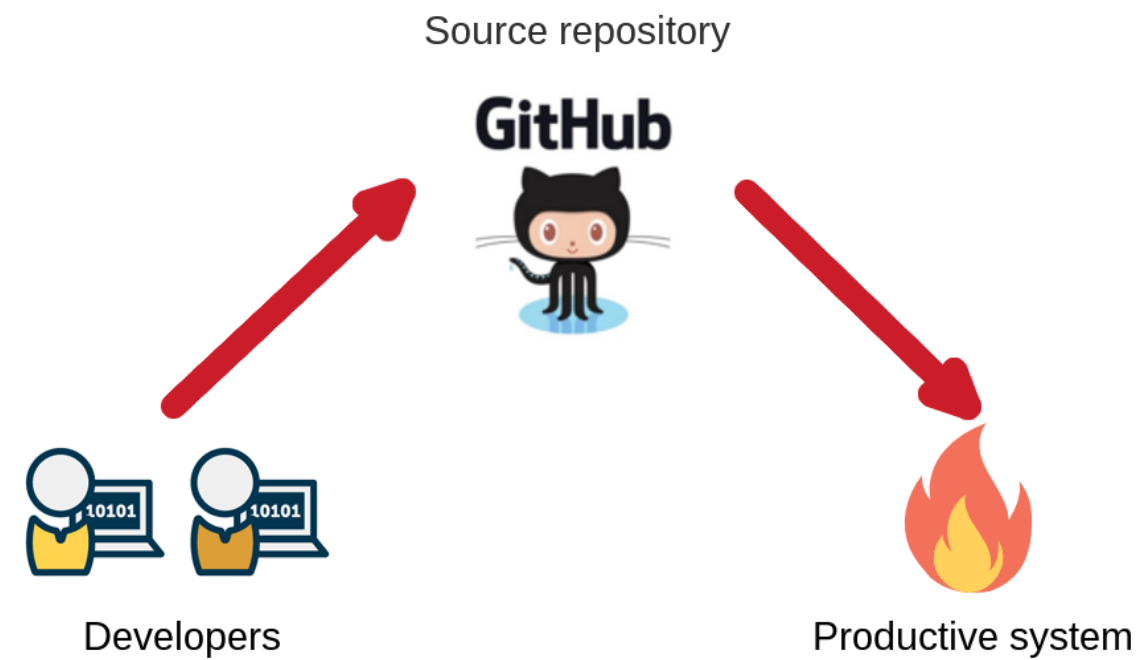- **Mailing list:** https://mail.python.org/mailman/listinfo/numpy-discussion

Reduced downtime

Source repository

Developers

Productive system

Reduced downtime

Source repository

Developers

Productive system

Reduced downtime

Source repository

GitHub

Developers

Productive system

Reduced downtime

Source repository

GitHub

Developers

Productive system

Reduced downtime

Source repository

Developers

CI Server runs unit tests

Productive Sytem

Reduced downtime

Source repository

Developers

CI Server runs unit tests

Productive System

Reduced downtime

Source repository

Developers

Notification

CI Server runs unit tests

Productive System
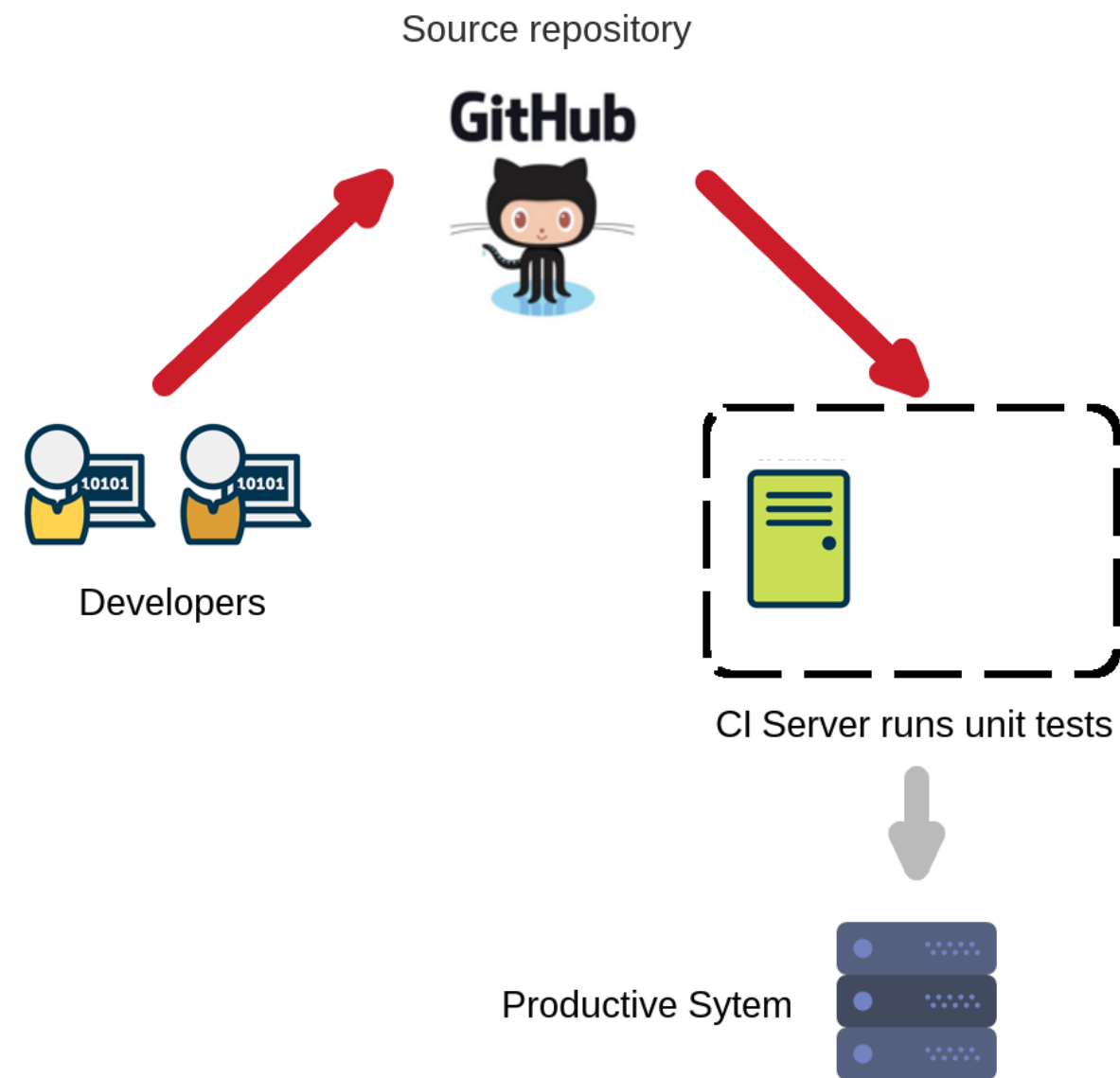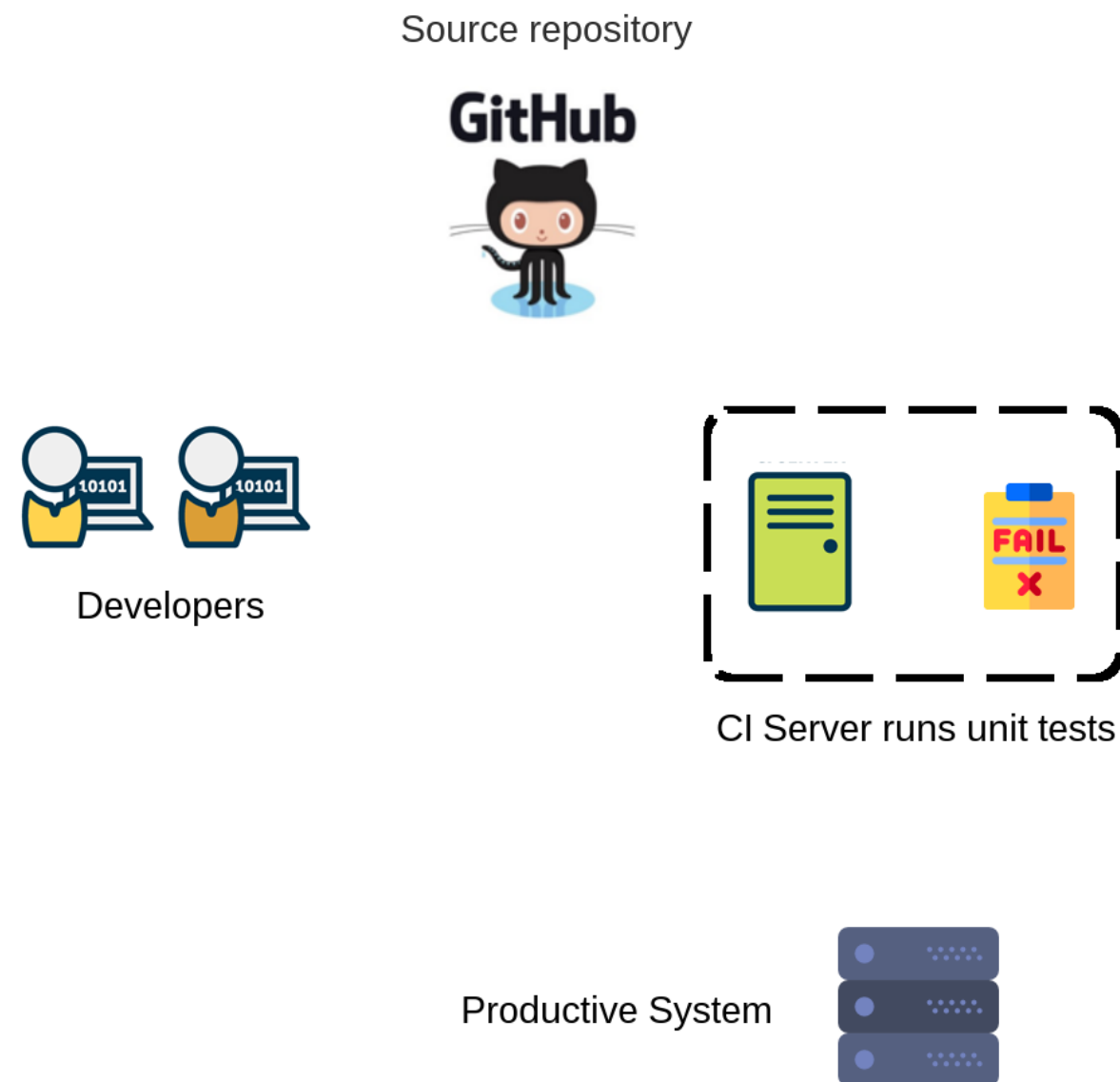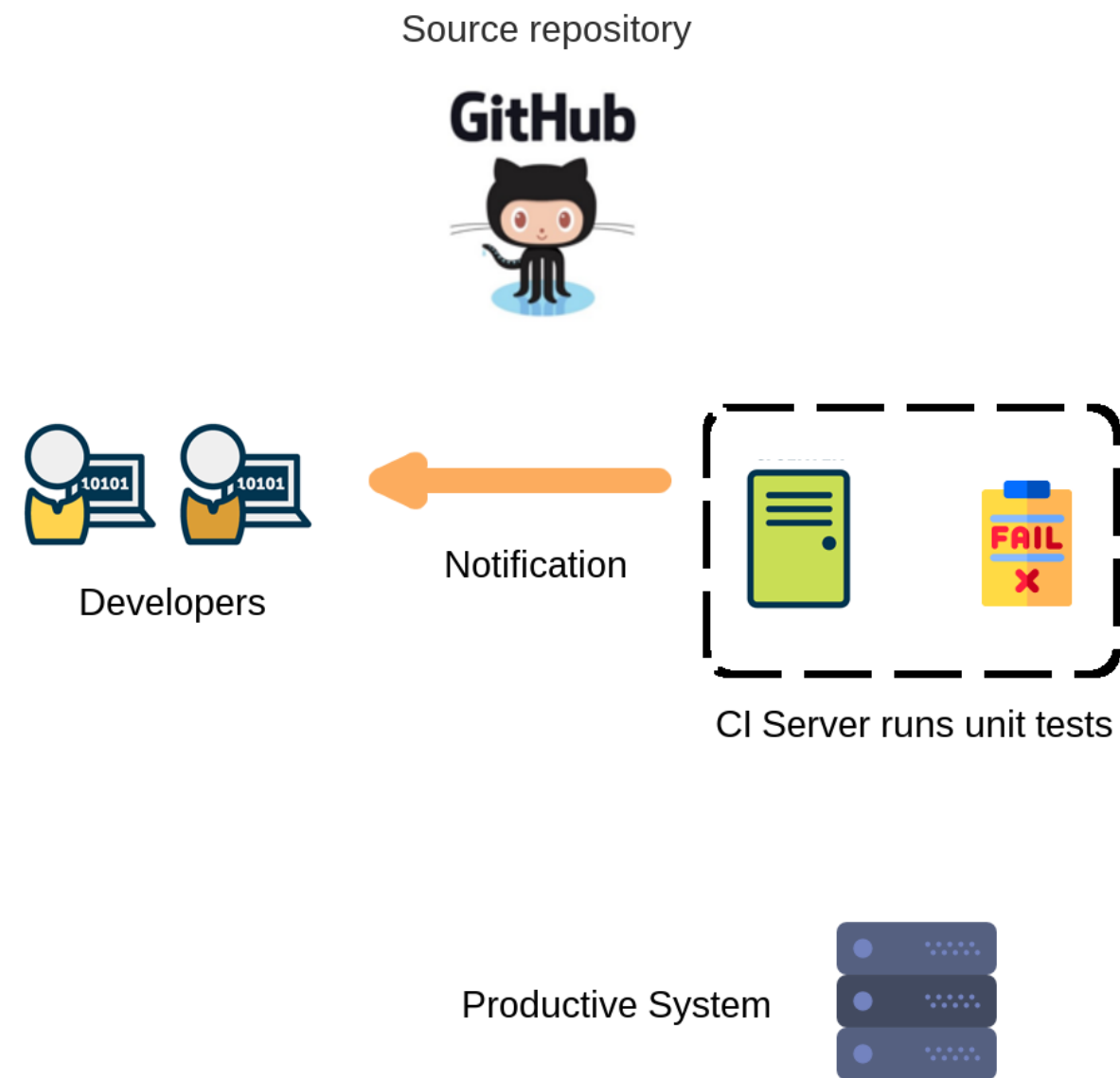
All benefits

- Time savings.

- Improved documentation.
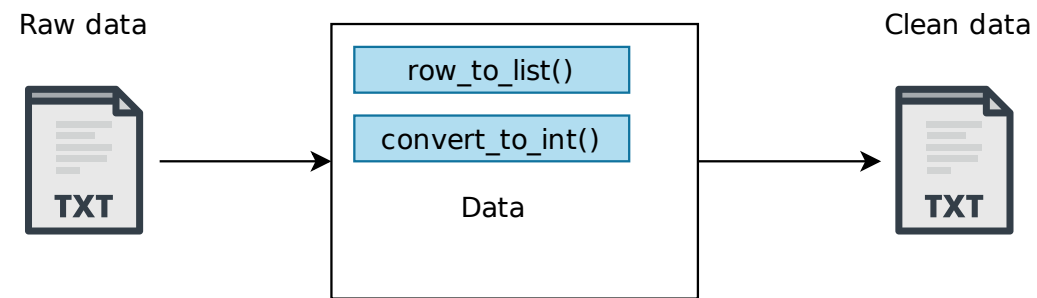
- More trust.

- Reduced downtime.

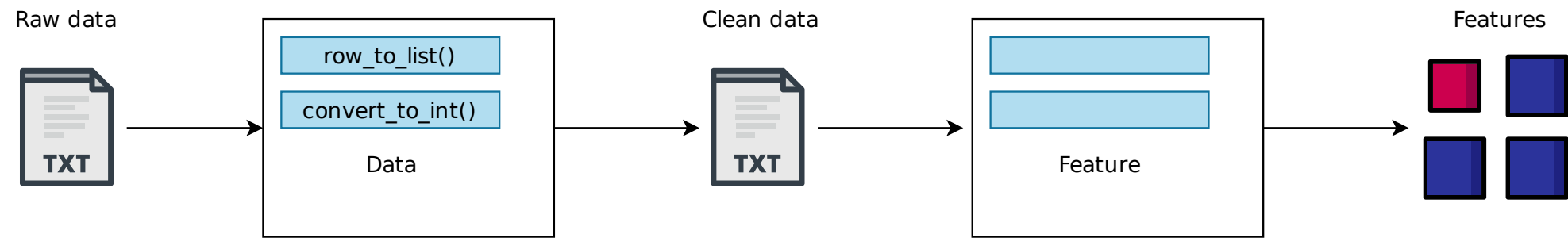# Tests we already wrote

row_to_list()

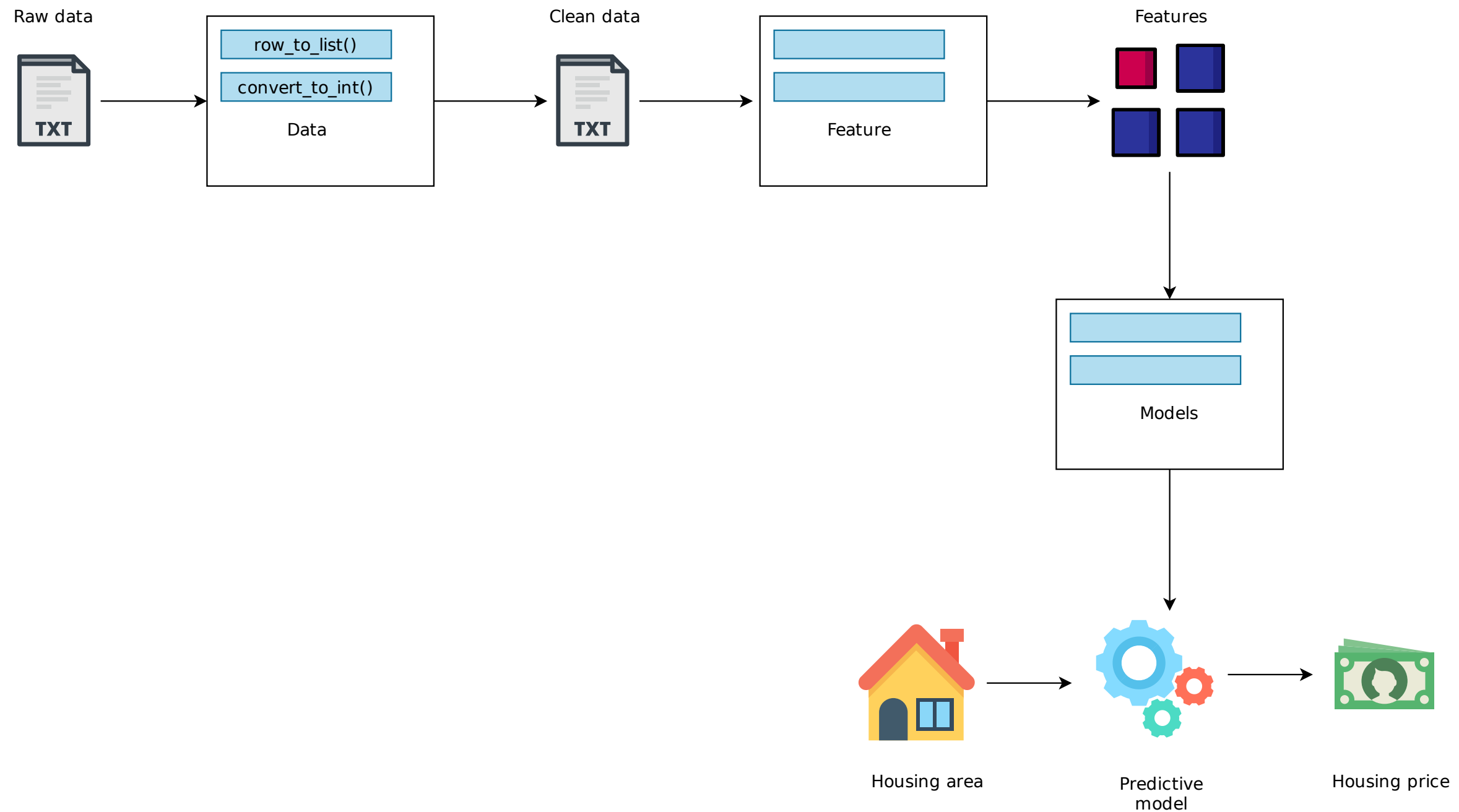# Tests we already wrote

row_to_list()

convert_to_int()

Data module

Feature module

Models module

Unit test

Raw data

Data
row_to_list()
convert_to_int()

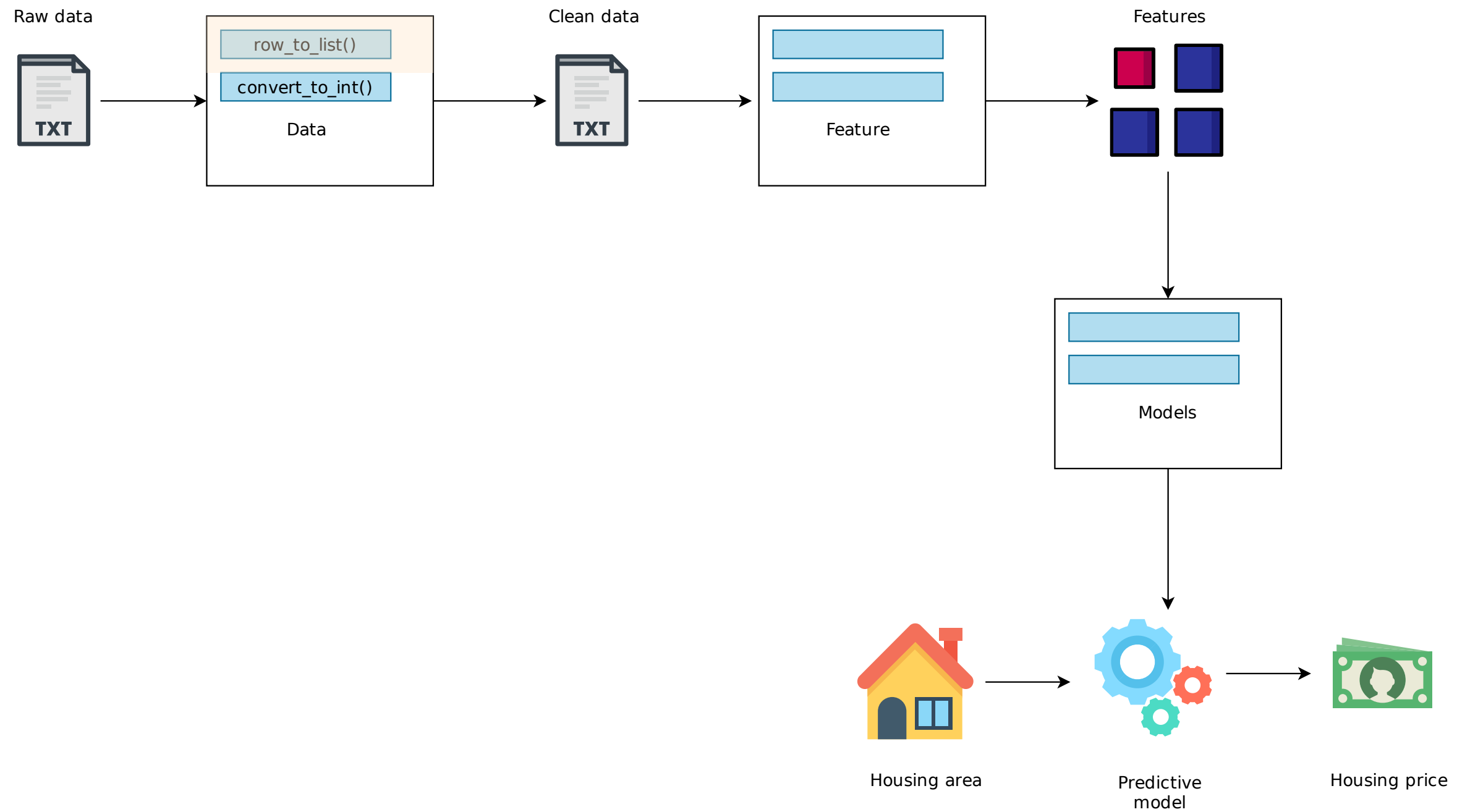Clean data

Feature

Features

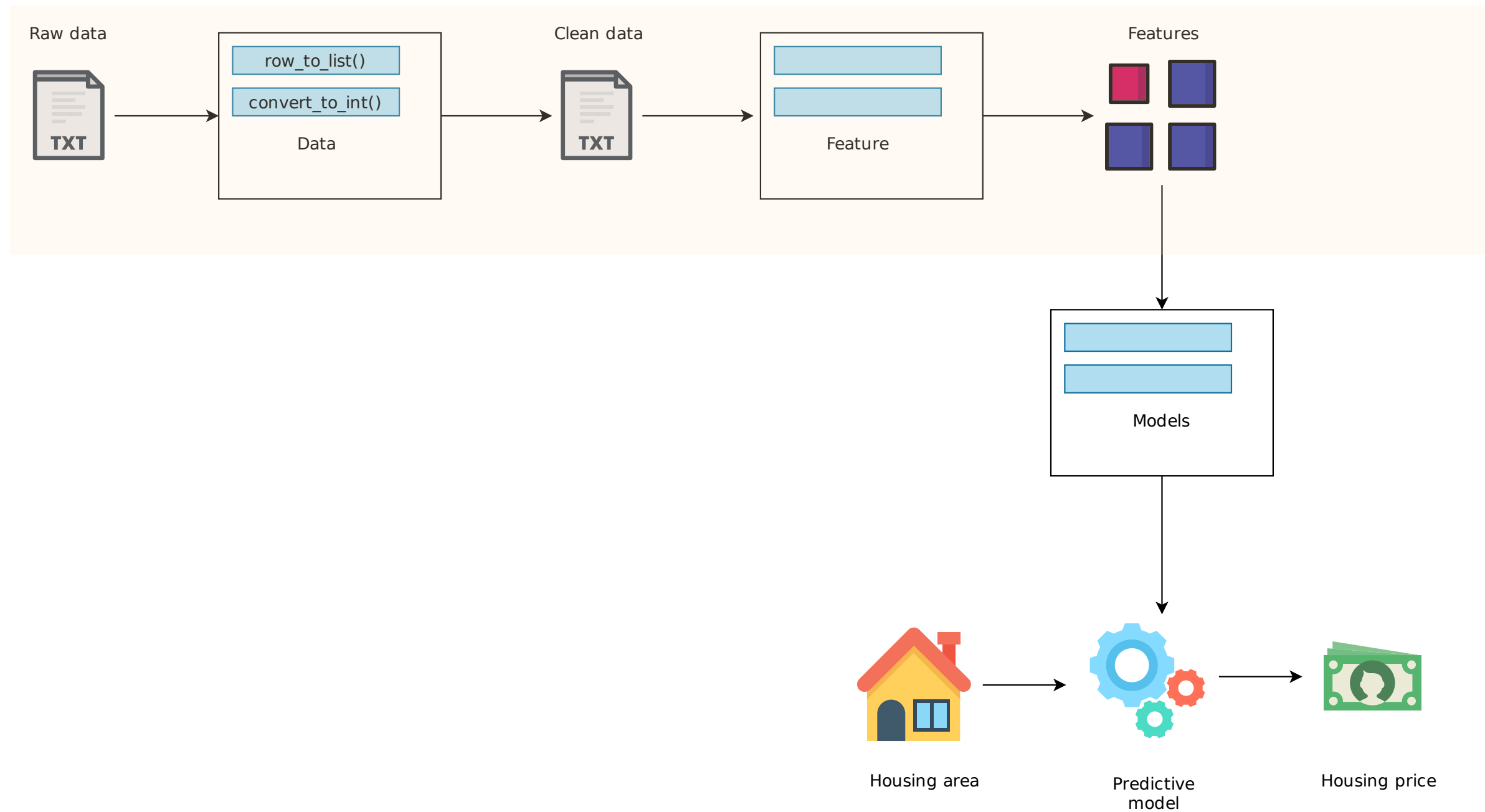Models

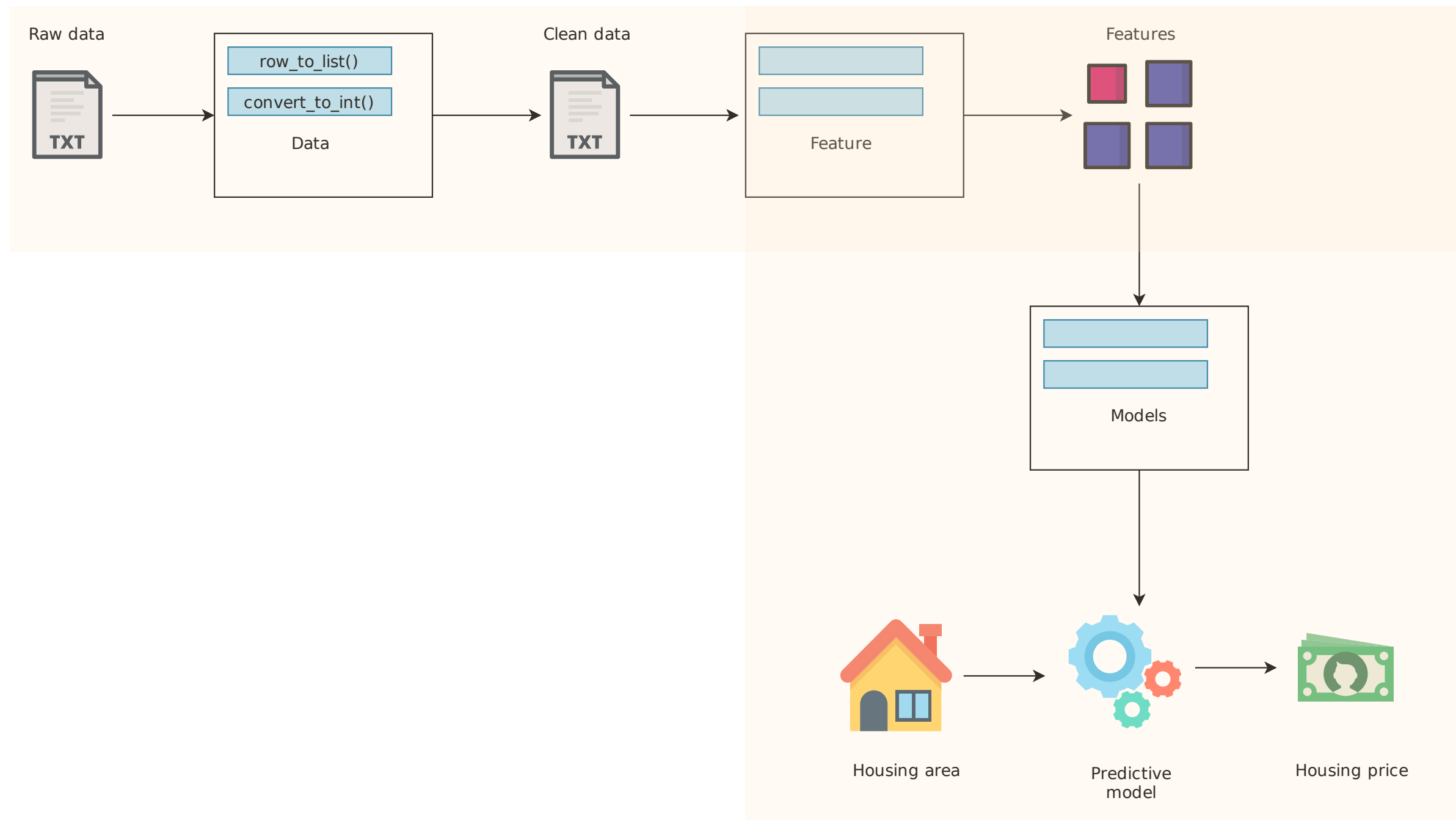Housing area → Predictive model → Housing price

What is a unit?

- Small, independent piece of code.

- Python function or class.

Integration test

End to end test

This course focuses on unit tests

- Writing unit tests is the best way to learn pytest.

In Chapter 2...

- Learn more pytest.

- Write more advanced unit tests.

- Work with functions in the `features` and `models` modules.

Let's practice these concepts!

UNIT TESTING FOR DATA SCIENCE IN PYTHON