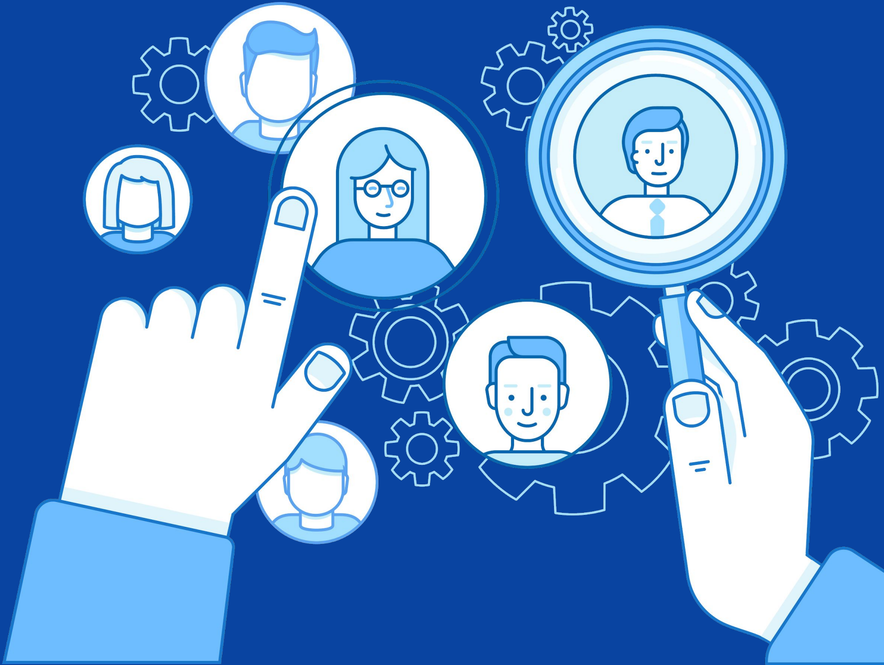
Digital Career Institute

Python - Testing



Digital Career Institute

Goal of the Submodule

The goal of this submodule is to familiarize students with why we are using testing in software and how to do basic testing in Python.

Digital Career Institute

Topics

- Errors in Software
- Historic failures of software
- Types of automated test
- How to write tests with unittest
- Overview of assert methods
- Practice TDD
- Why we need stubbing & mocking
- Introduce unittest.mock
- Practice with TDD and Pytest

Errors in Software

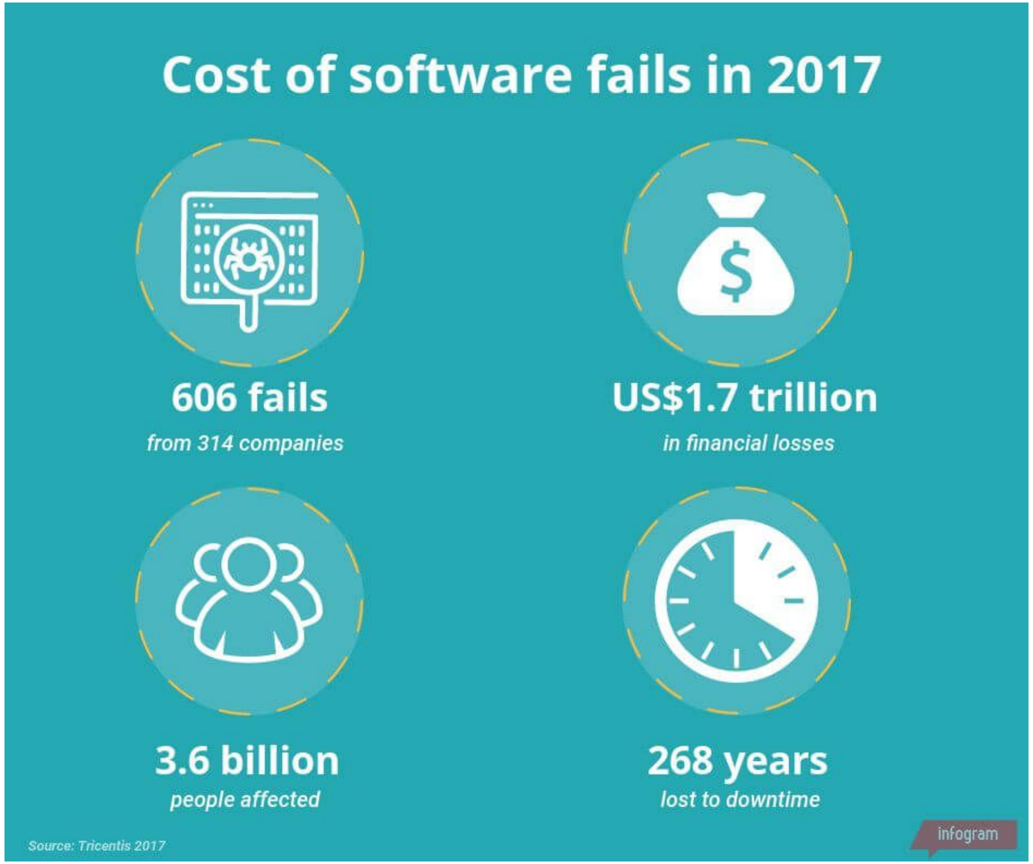
Digital Career Institute

Historic Failures of Software

Digital Career Institute

DCI

Software errors / Bug costs



Source: https://raygun.com/blog/cost-of-software-errors/

Failures of Software

Is it possible to build software without errors?

- No software is ever 100% bug-free.
- Building software is an ongoing process.
- Software is affected by external factors:
  - Integration with existing systems
  - Server and hosting
  - Integration with hardware
  - Integration with third-party providers
  - Legacy data formats
  - Scalability
- Normal day for any developer is:
  - Write code for one hour.
  - Spend the rest of the day to fix it.

Some of the largest software bugs

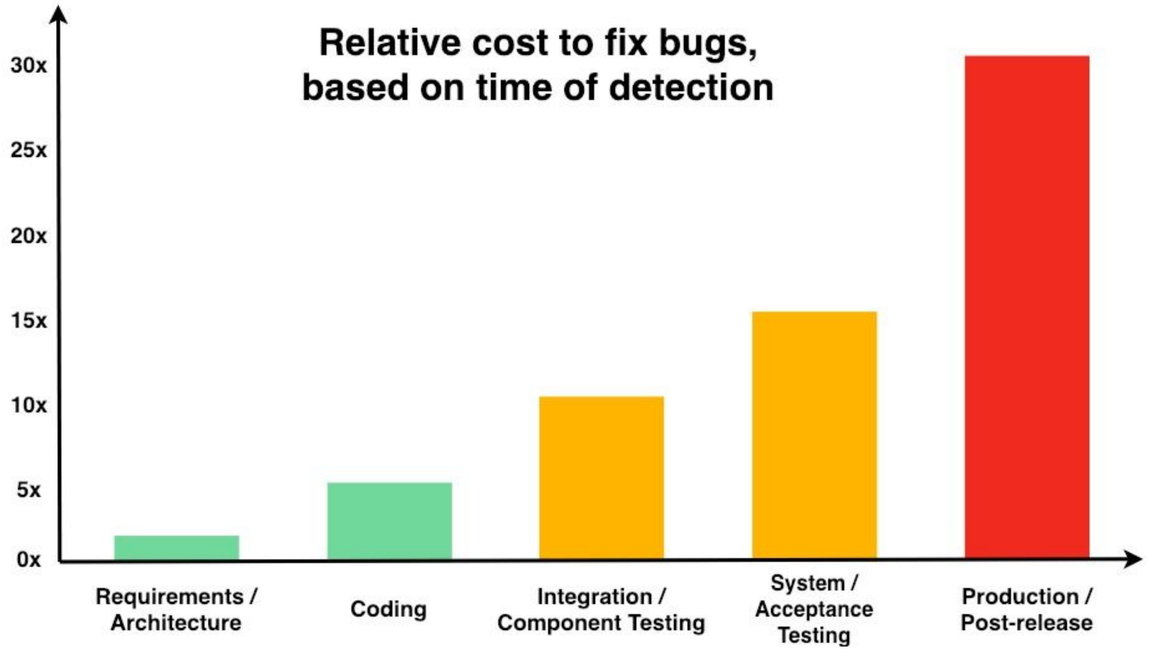
Some software errors have caused considerable material and human damage.

Watch this presentation of some of the largest bugs in history and try answering this question for each of them:

- Could the issue have been prevented?
- If yes, what should have been done to prevent the error from having the consequence it had?
- Could the issue have been prevented by letting a more senior software developer write the code?

Video: https://www.youtube.com/watch?v=AGI371htIN8

Costs of fixing bugs at various stages



Source: NIST via Deepsource

Errors in Software

Error Types in Software:

- **Syntax errors:**

  Can be detected easily because the program will not be able to run or compiled.

- **Run Time errors (bugs):**

  These kind of errors will only show after running the program.

Digital Career Institute

Run Time Errors

1. **Functional errors:** This is a broad type of error that happens whenever software doesn’t behave as intended (ex: the program doesn't show outputs).
2. **Logic errors:** represent a mistake in the software flow and causes the software to behave incorrectly (ex: incorrect output or crash).
3. **Calculation errors:** software returns an incorrect value for several reasons (wrong algorithm, data type mismatch, wrong encoding).
4. **Unit-level bugs:** most common and typically the easiest to fix.
5. **System-level integration bugs:** two or more pieces of software from separate subsystems interact erroneously.
6. **Out of bounds bugs:** the user sets a parameter outside the limits of intended use.

Testing Software before Release

Pros	Cons
Bugs will often be found before the software is released.	It takes extra time to test the software.

Software testing: Running the software and checking that it is producing the results that are expected.

Manual testing: Someone other than the developers is running the software prior to launch to see if they can find bugs in them.

Automatic testing: A script is written by a developer and this script runs the software and ensures that it produces the desired results

Testing for the largest software bugs

Thinking of the ten software issues mentioned in the video:

- Which type of error were each of them (functional, logic, calculation, unit-level, system level-integration, out of bounds)?
- Which ones could have been found by testing?
- Which ones would likely have been found by manual testing and which ones by automatic testing? Why?

Pros and Cons of Automated Tests

Digital Career Institute

Pros	Cons
Run quickly by a computer.	The testing scripts have to be written.
Fantastic return on investment. Write them once, then run them many, many times.	The test code has to be maintained.
Help you to maintain your application because the checks become part of the code.	Computers can't creatively find bugs that aren't checked for by the test.
Test can help other developers to understand the program, thus can act as a kind of documentation.	You have to be precise when you write the test scripts.

Pros and Cons of Human / Manual Tests

Digital Career Institute

Pros	Cons
Uses human creativity.	Humans are slow.
Does not require an exact, painfully precise script to be written.	Humans can only do one thing at a time.
Humans can find bugs creatively – exploratory testing.	Humans make mistakes.
	Human time is valuable and you have to worry about whether this is the best use of their time.
	Every time you run the test a human has to be there to run it.

Pro & Cons, Challenges and Costs of Automated Tests

Pros	Cons
Run quickly by a computer.	Tools still take time.
Reusable as automation process can be recorded.	It is expensive in the initial stage.
More interesting and everyone can see results.	It requires experienced developers to create good tests.
Test scripts can be run unattended.	Not every tool supports all types of testing.

At the Core of the Lesson

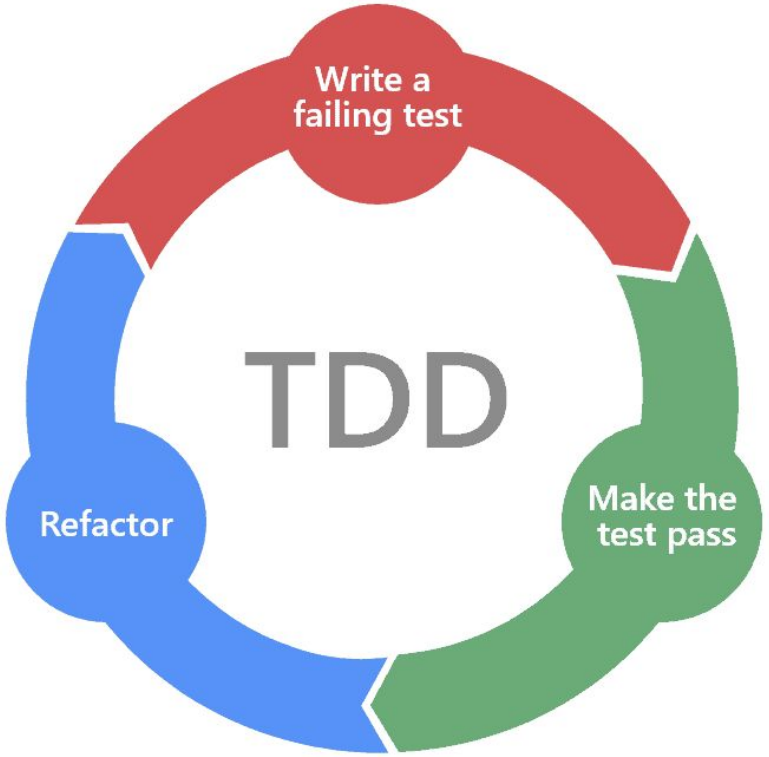
Lessons Learned:

- Error costs
- Historic failures of software
- Errors Types in Software
    - Syntax errors
    - Run Time errors
        - Functional errors
        - Logic errors
        - Calculation errors
        - Unit-level bugs
        - System-level integration bugs
        - Out of bounds bugs
- Automatic and manual testing

Test Driven Development and Types of Automated Tests

Digital Career Institute

Test Driven Development (TDD)



Test Driven Development (TDD)

1. Write tests before you code.
2. Then adjust the code until the tests pass.
3. Refactor code so it's easier to read and doesn't contain unnecessary duplications

Source:
https://medium.com/swlh/should-you-use-tdd-or-not-why-not-both-2e6f58767065

Pro & Cons of TDD

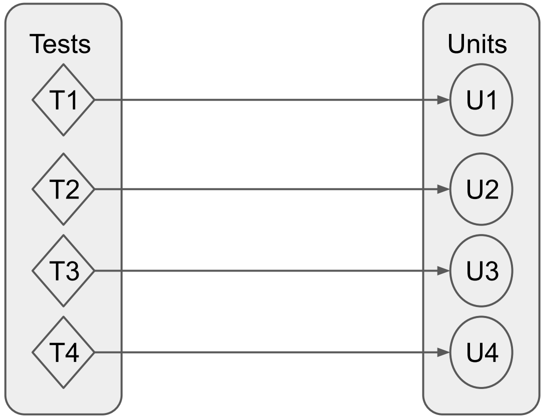
Pros	Cons
Separation of test and code writing means you will not forget testing parts that were hard to code.	Requirements and outside APIs may not be known at the time when code is initially written.
High test coverage: Creates tests for every feature and thereby makes code stable long-term.	It will slow-down the coding process.
	It is effective mostly only if all team members use it.

Types of Automated Tests:

- Unit Test
- Integration Test
    - System Test
    - Functional Test
        - End to End - Selenium
        - Acceptance
    - Black Box Test

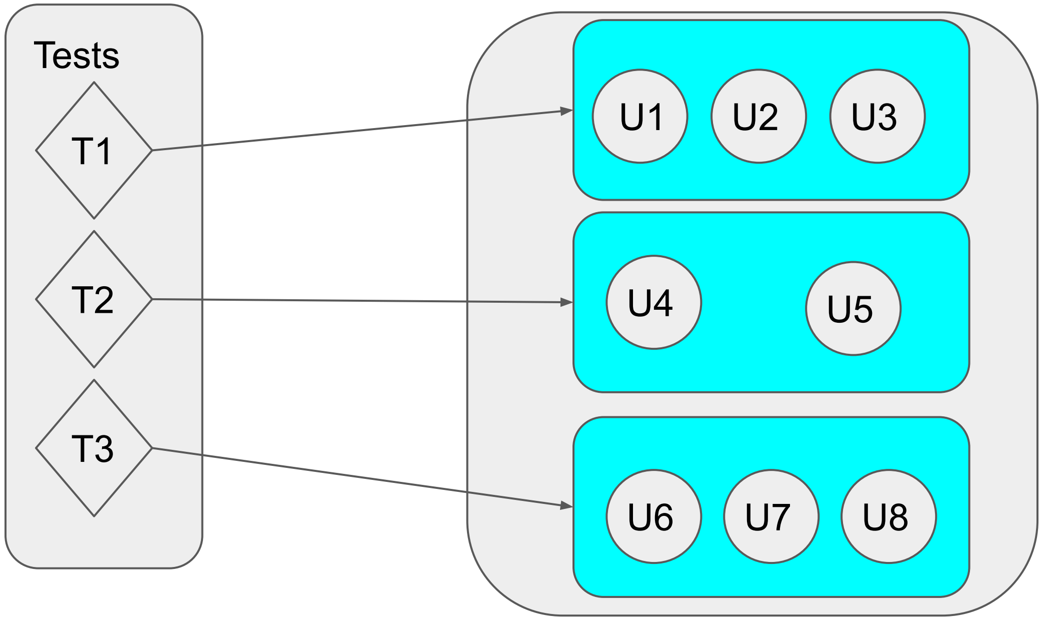
Unit Test

- Test for single unit (function, method)
- Describe what will happen with wrong input
- Describe what the code should or should not do
- Describe what the output should be
- Describe how much time the code should take to run



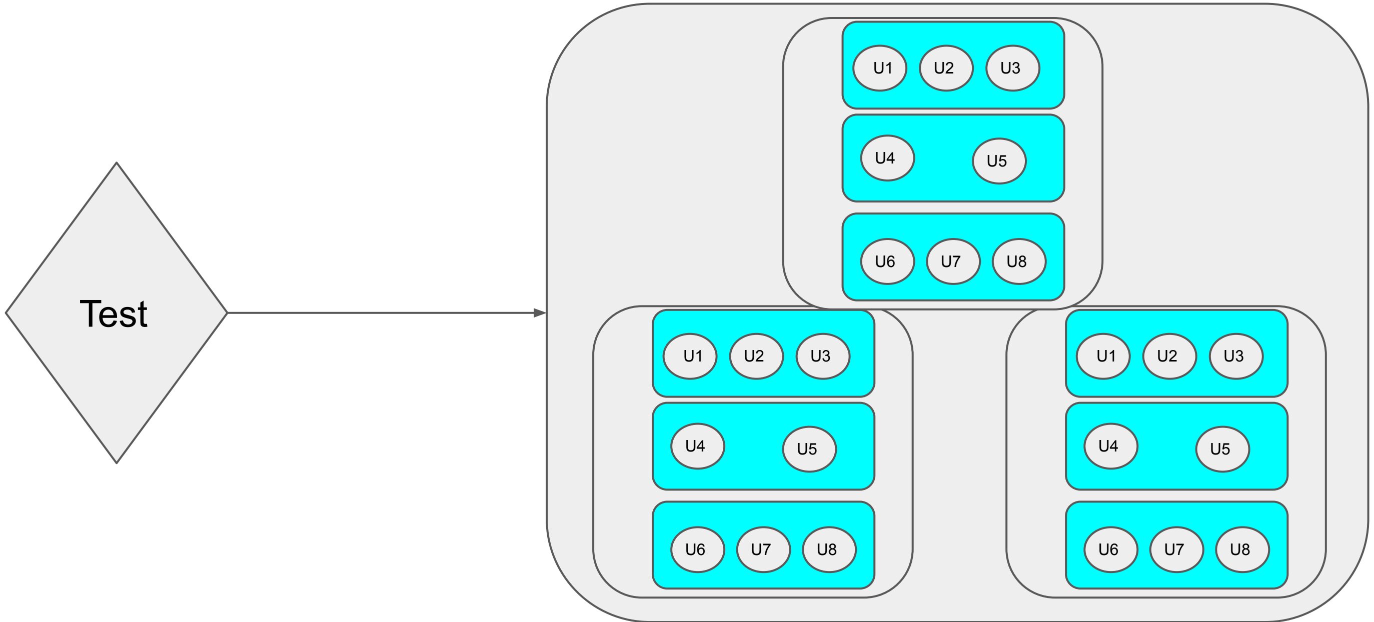
Integration Test

- After unit test
- Determine if independently developed units of software work correctly when they are connected to each other



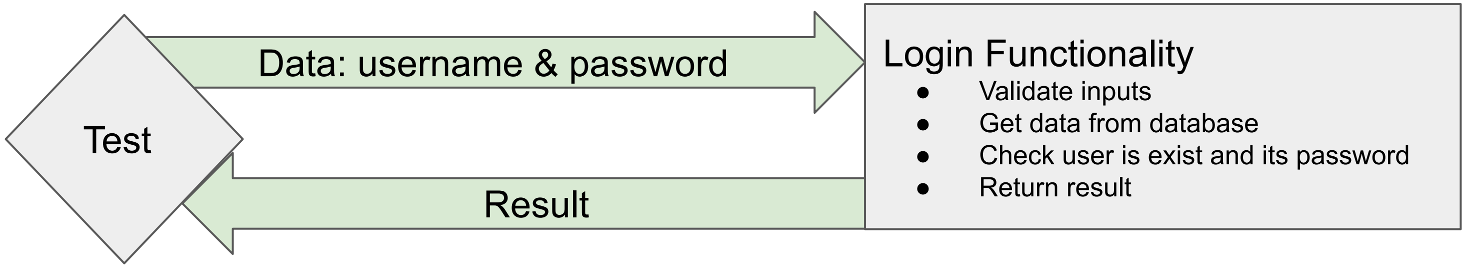
System Test

- After integration test
- Determine if all the component of software work correctly when they are connected to each other and it's called **end to end** test



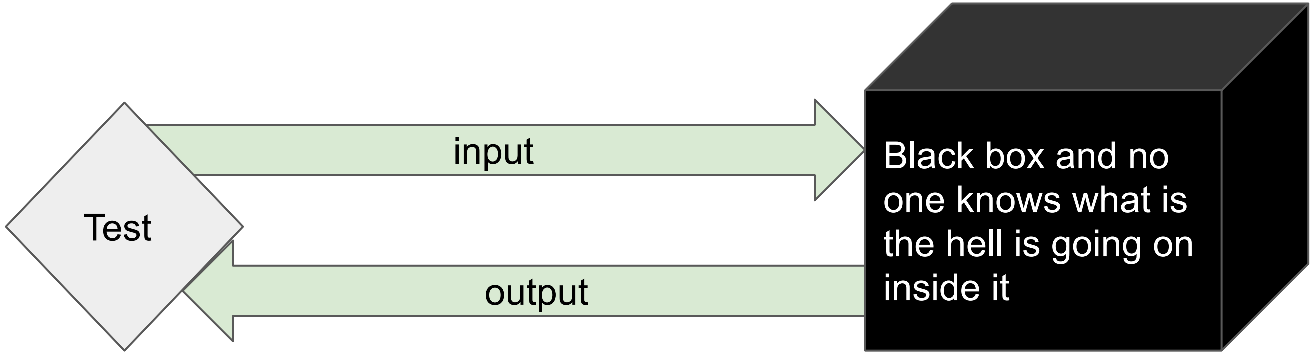
Functional Test

- Validates the software system against the functional requirements/specifications.
- Test each function of the software application, by providing appropriate input, verifying the output against the Functional requirements.



Black Box Test

- Test software without having knowledge of internal code structure, implementation details and internal paths.
- Focus on input and output of software applications and it is entirely based on software requirements and specifications.
- It can be an external library or a binary executable file.


Test
input
output
Black box and no one knows what is the hell is going on inside it

Test Coverage

- Measure of the percentage of the source code are covered by automatic tests.
- On python: **coverage** module

```
$ coverage report -m
Name                          Stmts   Miss  Cover   Missing
-------------------------------------------------------
my_program.py                    20      4    80%   33-35, 39
my_other_module.py               56      6    89%   17-23
-------------------------------------------------------
 TOTAL                            76     10    87%
```

At the Core of the Lesson

Digital Career Institute
DCI

Lessons Learned:

- Introduction of Test Driven Development (TDD)
- Types of Automated Tests
  - Unit Test
  - Integration Test
  - System Test
  - Functional Test
  - Black Box Test
- Coverage usage

Write Tests with **UnitTest**

Digital Career Institute

UnitTest

Python unit testing framework, based on Erich Gamma's JUnit and Kent Beck's Smalltalk testing framework.

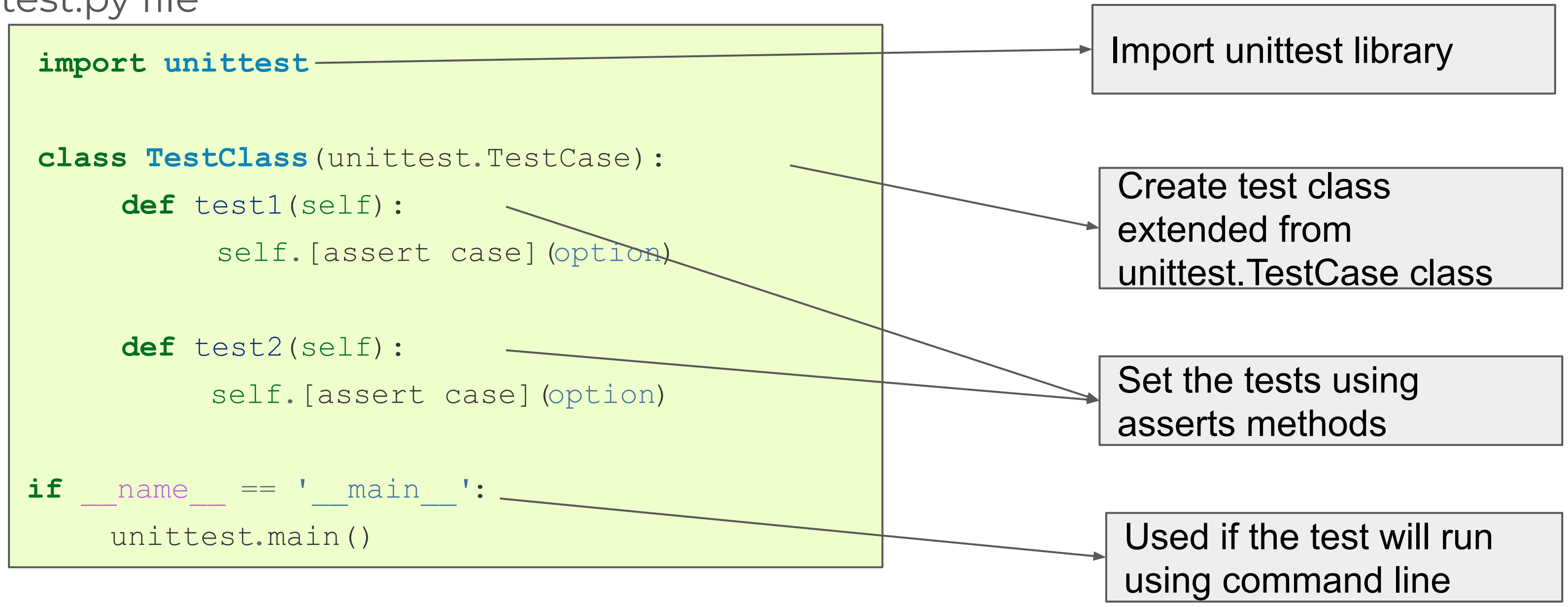
Concepts

- **Test fixture:** represents the preparation needed to perform one or more tests, and any associated cleanup actions
- **Test unit:** is the individual unit of testing. It checks for a specific response to a particular set of inputs.
- **Test suite:** is a collection of test cases, test suites, or both. It is used to aggregate tests that should be executed together.
- **Test runner:** is a component which orchestrates the execution of tests and provides the outcome to the user. The runner may use a graphical interface, a textual interface, or return a special value to indicate the results of executing the tests.

UnitTest

How to write a test:

test.py file



```
import unittest

class TestClass(unittest.TestCase):
    def test1(self):
        self.[assert case](option)

    def test2(self):
        self.[assert case](option)

if __name__ == '__main__':
    unittest.main()
```

Import unittest library

Create test class extended from unittest.TestCase class

Set the tests using asserts methods

Used if the test will run using command line

How to Run the Test?

- **Run tests from modules, classes or even individual test methods:**

```
python -m unittest test_module1 test_module2
python -m unittest test_module.TestClass
python -m unittest test_module.TestClass.test_method
```

- **Test modules can be specified by file path as well:**

```
python -m unittest tests/test_something.py
```

- **You can run tests with more detail (higher verbosity) by passing in the -v flag:**

```
python -m unittest -v test_module
```

- **When executed without arguments Test Discovery is started:**

```
python -m unittest
```

- **When using pytest:**

```
python -m pytest
```

- **When using coverage:**

```
coverage run -m unittest
coverage report -m
```

At the Core of the Lesson

Lessons Learned:

- Write tests with unittest:
  - Create simple test.
  - Different ways to run the tests:
    - from modules, classes or even individual test methods
    - by file path
    - with more detail
    - without arguments (Test Discovery)
    - with pytest
    - with coverage

Overview of Assert Methods

Overview of Assert Methods

Digital Career Institute

assertEqual	(a, b, optional message)	Pass if a = b
assertNotEqual	(a, b, optional message)	Pass if a != b
assertTrue	(a, optional message)	Pass if a = True
assertFalse	(a, optional message)	Pass if a = False
with assertRaises	(a, optional message)	Pass if sub scope return error with type 'a' (TypeError, OverflowError, RecursionError, ...)
assertIs	(a, b, optional message)	Pass if a is b
assertIsNot	(a, b, optional message)	Pass if a is not b

assertIsNone	(a, optional message)	Pass if x is None
assertIsNotNone	(a, optional message)	Pass if x is None
assertIn	(a, b, optional message)	Pass if a in b
assertNotIn	(a, b, optional message)	Pass if a not in b
assertIsInstance	(a, b, optional message)	Pass if a is an instant of b
assertNotIsInstance	(a, b, optional message)	Pass if a is not an instant of b
pass		Force test to pass
fail	(error message)	Force test to fail

At the Core of the Lesson

Lessons Learned:

- Overview of assert methods

Digital Career Institute

Exceptions and Testing
Exceptions

Digital Career Institute

Exceptions

- Exceptions are events that occur during the execution of programs that disrupt the normal flow of execution

- It is an **object** (derived from `BaseException` class) that represents an error.

Exception objects contain:

- Error type (exception name)
- The state of the program when the error occurred
- An error message describing the error event

Exceptions exist to help developers **debug** their programs for errors.

`raise`

Sometimes, an exception has to be thrown in code when a developer wants to write a program that would throw errors if it does not fulfill certain logic.

Python provides **`raise`** statement to **throw** exceptions in code.

The single arguments in the **`raise`** statement shows the exception that has to be raised.

We can raise exceptions in cases such as receiving wrong data or a validation failure.

Testing Exceptions

Digital Career Institute

Suppose you want to test a function, which should raise an exception under specific conditions:

```
def str_lower(word: str) -> str:
    if type(word) == str:
        return word.lower()
    raise TypeError("Expected str, got %s" %
type(word))
```

```
import unittest

class TestClass(unittest.TestCase):
    def test_lower(self):
        with self.assertRaises(TypeError):
            str_lower(1)

if __name__ == '__main__':
    unittest.main()
```

- `str_lower` raises an `TypeError` because of the wrong Type of the passed argument.
- The `TypeError` is asserted and `test_lower` passes.

At the Core of the Lesson

Lessons Learned:

- Introduction of Exceptions
- Throwing Exceptions
- Testing Exceptions

Digital Career Institute

Stubbing and Mocking

What / Why Stubbing & Mocking?

They are mechanisms to prevent unwanted side effects (such as deleted files, bank charges, user profile modifications, etc.) in the automated tests or to improve their performance.

Mocking:
A mock is a simulation of the object in the code that is being tested. It is simulated to allow running tests against the object without incurring in unwanted side effects.

Stubbing:
A stub is a set of fake data that will be used in the testing of the mock or another class. The stub is not the object being tested, but an object being used by the tested object.

- Imagine that we are writing a function that will delete some system files and we want to test it but we don't really want to delete any files!
- As a developer, you care that your code successfully called the system function for deleting a file more than successfully deleting a file.

Mocking

A mocking example

The file **my_module.py** contains a function to delete a file:

```python
import os

def rm(filename):
    os.remove(filename)
```

What if we don't want to delete anything but just to make sure that os remove function will be called correctly? You could mock it as part of the test:

```python
import unittest
from unittest.mock import patch

from my_module import rm

class TestClass(unittest.TestCase):

    def test_patch_remove(self):
        with patch("os.remove"):
            rm("test.txt")

if __name__ == '__main__':
    unittest.main()
```

No mocking test

File test.py to test my_module **rm** function

```
import unittest
from my_module import rm
import os

class RmTestCase(unittest.TestCase):
    def test_rm(self):
        # create a file
        open('somefile.txt', 'a')
        # try to delete it
        rm('somefile.txt')
        # check file if still exist
        self.assertFalse(os.path.isfile('/somefile.txt'), 'failed to remove the file')

if __name__ == '__main__':
    unittest.main()
```

The problem is our test will create a file **somefile.txt** and try to delete using **rm** function so maybe the file is **already exist** and it will be **deleted**

mocking test

File test.py to test myModule **rm** function

```
from my_module import rm
import mock
import unittest

class RmTestCase(unittest.TestCase):
    # in the following like mock will simulate os library in myModule
    @mock.patch('my_module.os')
    def test_rm(self, mock_os):
        rm("somefile1.txt")
        # test that rm called os.remove with the right parameters
        mock_os.remove.assert_called_with("somefile1.txt")

if __name__ == '__main__':
    unittest.main()
```

Test will run **without really deleting somefile1.txt**

At the Core of the Lesson

Lessons Learned:

- Stubbing and mocking:
  - Differences and similarities
  - When do we need them?
  - How do we use them?

Digital Career Institute

Let's Practice Life Code

Contact Details
DCI Digital Career Institute gGmbH

Digital Career Institute