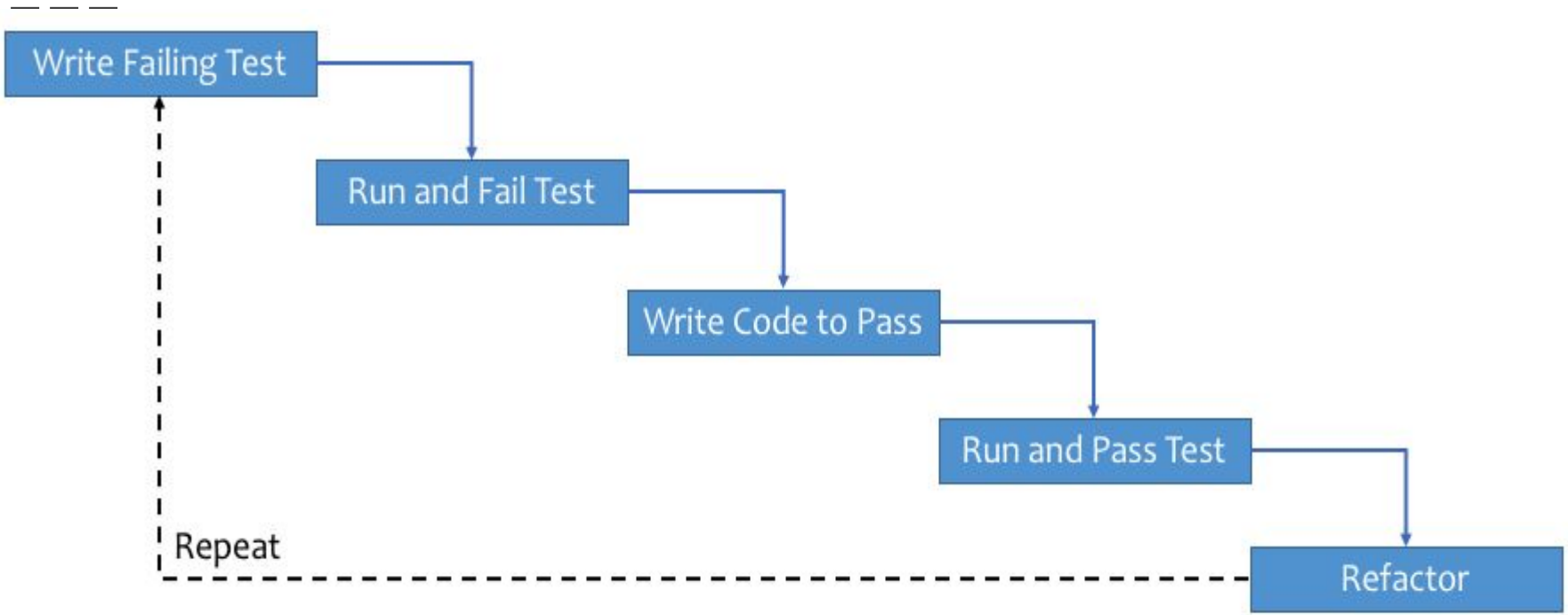
Testing your code

Code without tests???

TDD - Test Driven Development - how?



pytest

```
From virtualenv: pip install pytest

Globally:        pip3 install pytest
```

TDD - project structure

- Create TDD folder inside Documents

  ```
  mkdir -p ~/Documents/TDD
  ```

- Create mycode and tests folder inside TDD

  ```
  mkdir -p ~/Documents/TDD/mycodes
  ```

  ```
  mkdir -p ~/Documents/TDD/test
  ```

- touch ~/Documents/TDD/mycodes/mycode.py
- touch ~/Documents/TDD/test/test_mycode.py

Creating failing test first - according to TDD

```
import pytest
import sys
sys.path.append('..')

from mycodes import mycode

class TestMycode:

    def test_hello_world(self):
        assert mycode.hello_world() == "Hello, World!"
~
```

Create hello_world() inside mycode.py

Open ~/Documents/TDD/mycodes/mycode.py

Run pytest

```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest test_mycode.py
============================= test session starts ==============
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 1 item

test_mycode.py F

================================== FAILURES ====================
_____ TestMycode.test_hello_world _____

self = <TDD.test.test_mycode.TestMycode object at 0x7f29ee3d4dd8>

    def test_hello_world(self):
>       assert mycode.hello_world() == "Hello, World!"
E       AssertionError: assert None == 'Hello, World!'
E        +  where None = <function hello_world at 0x7f29ee37f6a8>()
E        +    where <function hello_world at 0x7f29ee37f6a8> = mycode.hello_world

test_mycode.py:10: AssertionError
========================== 1 failed in 0.03 seconds ============
```

Fixing code to pass test

Edit ~/Documents/TDD/mycodes/mycode.py

```
def hello_world():
    return "Hello, World!"
```

Running again pytest after fixing original code

```
cd ~/Documents/TDD/test
```

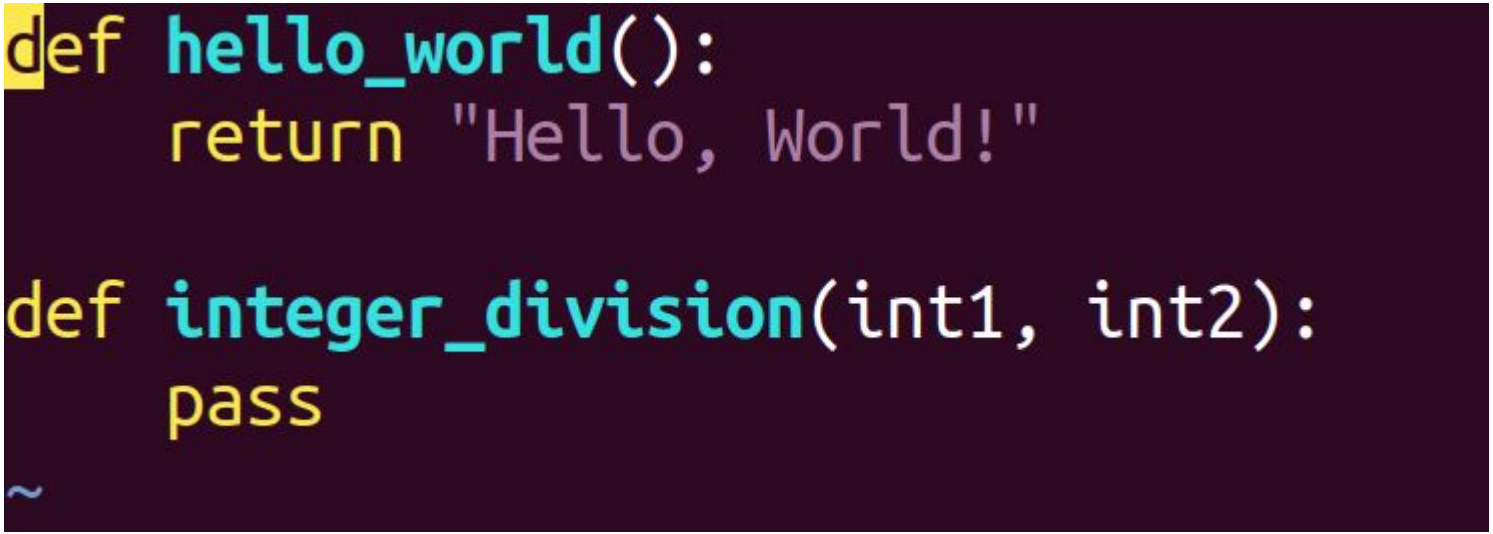
```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest test_mycode.py
============================================== test session starts ===
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 1 item

test_mycode.py .

============================================== 1 passed in 0.01 seconds
```

Let's test another example

```
import pytest
import sys
sys.path.append('..')

from mycodes import mycode

class TestMycode:

    def test_hello_world(self):
        assert mycode.hello_world() == "Hello, World!"

    def test_integer_division(self):
        assert mycode.integer_division(6, 2) == 3
~
```

Add integer_division() into mycode.py

Edit ~/Documents/TDD/mycodes/mycode.py



```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest -v test_mycode.py
============================= test session starts ==============
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0 -- /home/shako/virtualenvs/py_
cachedir: .pytest_cache
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 2 items

test_mycode.py::TestMycode::test_hello_world PASSED
test_mycode.py::TestMycode::test_integer_division FAILED

=================================== FAILURES ===================
_____________________ TestMycode.test_integer_division _________

self = <TDD.test.test_mycode.TestMycode object at 0x7f477f4d2780>

    def test_integer_division(self):
>       assert mycode.integer_division(6, 2) == 3
E       assert None == 3
E        +  where None = <function integer_division at 0x7f477f4f4840>(6, 2)
E        +    where <function integer_division at 0x7f477f4f4840> = mycode.integer_division

test_mycode.py:13: AssertionError
========================== 1 failed, 1 passed in 0.03 seconds ======
```

Fixing failed test

Edit ~/Documents/TDD/mycodes/mycode.py

```
def hello_world():
    return "Hello, World!"

def integer_division(int1, int2):
    return int1 / int2
~
```

Running pytest

```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest -v test_mycode.py
============================================================ test session starts ==
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0 -- /home/shako/virt
cachedir: .pytest_cache
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 2 items

test_mycode.py::TestMycode::test_hello_world PASSED
test_mycode.py::TestMycode::test_integer_division PASSED

========================================================= 2 passed in 0.01 seconds
```

Hacking our integer_division() function

Edit ~/Documents/TDD/test/test_mycode.py

```
def test_integer_division(self):
    assert mycode.integer_division(6, 2) == 3
    mycode.integer_division(6, 0)
    mycode.integer_division('Tech', 9)
    mycode.integer_division(8.4, 2)
```

Run pytest

```
=================================================================== FAILURES ===================
_____________________________________________ TestMycode.test_integer_division _

self = <TDD.test.test_mycode.TestMycode object at 0x7fbbddc0e780>

    def test_integer_division(self):
        assert mycode.integer_division(6, 2) == 3
>       mycode.integer_division(6, 0)

test_mycode.py:14:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

int1 = 6, int2 = 0

    def integer_division(int1, int2):
>       return int1 / int2
E       ZeroDivisionError: division by zero

../mycodes/mycode.py:5: ZeroDivisionError
============================================================= 1 failed, 1 passed in 0.03 seconds
```

How we can fix this? Catching exception on mycode.py

Edit ~/Documents/TDD/mycodes/mycode.py

```
def integer_division(int1, int2):
    try:
        return int1 / int2
    except ZeroDivisionError:
        return "Division by Zero detected!"
```

Adding assert to ZeroDivisionError test

Edit ~/Documents/TDD/test/test_mycode.py

```
def test_integer_division(self):
    assert mycode.integer_division(6, 2) == 3
    assert mycode.integer_division(6, 0) == "Division by Zero detected!"
    mycode.integer_division('Tech', 9)
    mycode.integer_division(8.4, 2)
```

```
========================================================================== FAILURES ==========
_____________________________________________________ TestMycode.test_integer_division _

self = <TDD.test.test_mycode.TestMycode object at 0x7f88678d74a8>

    def test_integer_division(self):
        assert mycode.integer_division(6, 2) == 3
        assert mycode.integer_division(6, 0) == "Division by Zero detected!"
>       mycode.integer_division('Tech', 9)

test_mycode.py:15:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

int1 = 'Tech', int2 = 9

    def integer_division(int1, int2):
        try:
>           return int1 / int2
E           TypeError: unsupported operand type(s) for /: 'str' and 'int'

../mycodes/mycode.py:6: TypeError
================================================== 1 failed, 1 passed in 0.03 seconds ========
```

Expecting ZeroDivisionError on pytest side?

Edit ~/Documents/TDD/mycodes/mycode.py

```
def integer_division(int1, int2):
    try:
        return int1 / int2
    except ZeroDivisionError:
        raise
```

Change test on pytest side

Edit ~/Documents/TDD/test/test_mycode.py

```
def test_integer_division(self):
    assert mycode.integer_division(6, 2) == 3
    with pytest.raises(ZeroDivisionError):
        mycode.integer_division(6, 0)
    mycode.integer_division('Tech', 9)
    mycode.integer_division(8.4, 2)
```

```
========================================================================== FAILURES ===========
_____________________________________________________ TestMycode.test_integer_division

self = <TDD.test.test_mycode.TestMycode object at 0x7f129e1a4860>

    def test_integer_division(self):
        assert mycode.integer_division(6, 2) == 3
        with pytest.raises(ZeroDivisionError):
            mycode.integer_division(6, 0)
>       mycode.integer_division('Tech', 9)

test_mycode.py:16:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

int1 = 'Tech', int2 = 9

    def integer_division(int1, int2):
        try:
>           return int1 / int2
E           TypeError: unsupported operand type(s) for /: 'str' and 'int'

../mycodes/mycode.py:6: TypeError
========================================================== 1 failed, 1 passed in 0.03 seconds
```

Fixing TypeError

Edit ~/Documents/TDD/mycodes/mycode.py

```
def integer_division(int1, int2):
    try:
        return int1 / int2
    except ZeroDivisionError:
        raise
    except TypeError:
        return "Expecting integers as parameters!"
~
```

Fixing test code

Edit ~/Documents/TDD/test/test_mycode.py

```
def test_integer_division(self):
    assert mycode.integer_division(6, 2) == 3
    with pytest.raises(ZeroDivisionError):
        mycode.integer_division(6, 0)
    assert mycode.integer_division('Tech', 9) == "Expecting integers as parameters!"
    mycode.integer_division(8.4, 2)
```

Running again tests

```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest -v test_mycode.py
============================================================ test session starts ==
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0 -- /home/shako/virt
cachedir: .pytest_cache
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 2 items

test_mycode.py::TestMycode::test_hello_world PASSED
test_mycode.py::TestMycode::test_integer_division PASSED

========================================================= 2 passed in 0.01 seconds
```

But we have still problems!

The last test is not properly configured.
Our integer division function should return Integer not Float. Let's add this check.

```
def test_integer_division(self):
    assert mycode.integer_division(6, 2) == 3
    with pytest.raises(ZeroDivisionError):
        mycode.integer_division(6, 0)
    assert mycode.integer_division('Tech', 9) == "Expecting integers as parameters!"
    assert isinstance(mycode.integer_division(8.4, 2), int)
```

Type check will fail

```
================================================================= FAILURES ==========
_____________________________________________ TestMycode.test_integer_division _

self = <TDD.test.test_mycode.TestMycode object at 0x7fac9c7c9860>

    def test_integer_division(self):
        assert mycode.integer_division(6, 2) == 3
        with pytest.raises(ZeroDivisionError):
            mycode.integer_division(6, 0)
        assert mycode.integer_division('Tech', 9) == "Expecting integers as parameters!"
>       assert isinstance(mycode.integer_division(8.4, 2), int)
E       assert False
E        +  where False = isinstance(4.2, int)
E        +    where 4.2 = <function integer_division at 0x7fac9c7eb7b8>(8.4, 2)
E        +      where <function integer_division at 0x7fac9c7eb7b8> = mycode.integer_divisio

test_mycode.py:17: AssertionError
========================================================== 1 failed, 1 passed in 0.03 seconds
```

Fixing the return type issue

```
def integer_division(int1, int2):
    try:
        return int(int1 / int2)
    except ZeroDivisionError:
        raise
    except TypeError:
        return "Expecting integers as parameters!"
~
```

Passing all tests

```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest -v test_mycode.py
============================================================ test session starts =
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0 -- /home/shako/vir
cachedir: .pytest_cache
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 2 items

test_mycode.py::TestMycode::test_hello_world PASSED
test_mycode.py::TestMycode::test_integer_division PASSED

========================================================= 2 passed in 0.01 seconds
```

PyTest fixtures -> Testing Classes

Useful DOCs:
https://docs.pytest.org/en/latest/fixture.html#fixtures

https://docs.pytest.org/en/latest/reference.html#pytest-fixture

Convert our working code to class -> edit mycode.py

```
class WantTest:

    def hello_world(self):
        return "Hello, World!"

    def integer_division(self, int1, int2):
        try:
            return int(int1 / int2)
        except ZeroDivisionError:
            raise
        except TypeError:
            return "Expecting integers as parameters!"
```

Creating conftest.py file inside test folder

`touch ~/Documents/TDD/test/conftest.py`

```
import sys
sys.path.append('..')
from mycodes.mycode import WantTest
import pytest

obj = WantTest()

@pytest.fixture()
def return_obj():
    return obj
```

Changing test_mycode.py file to use class scope fixtures

```python
import pytest

@pytest.mark.usefixtures("return_obj")
class TestMycode:

    def test_hello_world(self, return_obj):
        assert return_obj.hello_world() == "Hello, World!"

    def test_integer_division(self, return_obj):
        assert return_obj.integer_division(6, 2) == 3
        with pytest.raises(ZeroDivisionError):
            return_obj.integer_division(6, 0)
        assert return_obj.integer_division('Tech', 9) == "Expecting integers as parameters!"
        assert isinstance(return_obj.integer_division(8.4, 2), int)
```

Running tests

```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest -v test_mycode.py
============================================================ test session starts ============================
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0 -- /home/shako/virtualenvs/py_3.6/bin/python3
cachedir: .pytest_cache
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 2 items

test_mycode.py::TestMycode::test_hello_world PASSED
test_mycode.py::TestMycode::test_integer_division PASSED
```

Create a new Class and counter method in mycode.py

```
class AdditionalClass:
    def __init__(self, mylist):
        self.mylist = mylist

    def counter(self):
        count = 0
        for _ in self.mylist:
            count = count + 1

        return count
```

Add changes to conftest.py file

```
import sys
sys.path.append('..')
from mycodes.mycode import WantTest, AdditionalClass
import pytest

obj = WantTest()

@pytest.fixture()
def return_obj():
    return obj

obj_additional1 = AdditionalClass([1, 2, 3])

@pytest.fixture()
def return_obj_additional1():
    return obj_additional1
```

Use method scope fixture test_mycode.py

```
@pytest.mark.usefixtures("return_obj_additional1")
def test_additional_class_counter(self, return_obj_additional1):
    assert return_obj_additional1.counter() == 3
```

How about using 2 objects from fixture? conftest.py

```
obj_additional2 = AdditionalClass([])

@pytest.fixture()
def return_obj_additional2():
        return obj_additional2
~
```

Passing 2 fixture object to test method

Edit ~/Documents/TDD/test/test_mycode.py

```
@pytest.mark.usefixtures("return_obj_additional1", "return_obj_additional2")
def test_additional_class_counter(self, return_obj_additional1, return_obj_additional2):
    assert return_obj_additional1.counter() == 3
    assert return_obj_additional2.counter() == 0
```

Final run of tests

```
(py_3.6) shako@shako-localhost:~/Documents/TDD/test$ pytest -v test_mycode.py
============================================================ test session starts ==
platform linux -- Python 3.6.3, pytest-3.5.1, py-1.5.3, pluggy-0.6.0 -- /home/shako/virt
cachedir: .pytest_cache
rootdir: /home/shako/Documents/TDD/test, inifile:
collected 3 items

test_mycode.py::TestMycode::test_hello_world PASSED
test_mycode.py::TestMycode::test_integer_division PASSED
test_mycode.py::TestMycode::test_additional_class_counter PASSED

========================================================= 3 passed in 0.01 seconds
```