Intro to Test Driven Development

Hello!

My name is Sana Javed :)
@sanacodes

I’m a software developer at National Journal

Transitioned to the tech field from digital media

I also co-organize Prince George’s Tech Meetup so join us!

What is Testing and Why Does It Matter?

As applications grow and evolve over time, tests are pieces of code that we write to ensure that:

1. Current features work the way we expect them to
2. New features don’t break existing features
3. Software can easily adapt.

Why Should I Write Tests?

Whether it's the users or the developer, one way or another, someone is testing your code.

Users are bound to encounter bugs but as developers, we can minimize *how many* they encounter.

Tests provide a safety net so that we can:

- ❑ Refactor code
- ❑ Add new features
- ❑ Avoid reintroducing bugs we already fixed

So What's Test Driven Development?

Breaking requirements into smaller pieces is a regular thing when writing code.

TDD reinforces that concept and applies it to testing.

When you write the test first, you have to figure out what the requirements for this bit of code are and what it would mean for the test to "pass" or to "fail".

When paired with tests like this, you naturally keep your functions and methods quite succinct and DRY.

An Example: Letter Count (Without TDD)

```
def count_letter(text, letter_to_find):
    """
    Counts the number of times the given
    letter appears in the text
    """
    count = 0
    for character in text:
        if character == letter_to_find:
            count += 1
    return count

print(count_letter('My name is Sana and I have a sister named Ana'
    , 'N'))

# Returns 0, when there are actually 5 because 'n'
# is different from 'N' in programming

```

Let’s fix things!

```python
def count_letter(text, letter_to_find):
    """
    Counts the number of times the given
    letter appears in the text
    """
    count = 0
    for character in text:
        if character == letter_to_find.lower():
            count += 1
    return count

print(count_letter('My name is Sana and I have a sister named Ana'
    , 'N'))

# Returns 5, hooray!

```

But wait...

```python
def count_letter(text, letter_to_find):
    """
    Counts the number of times the given
    letter appears in the text
    """
    count = 0
    for character in text:
        if character == letter_to_find.lower():
            count += 1
    return count

print(count_letter('My name is Sana and I have a sister named Ana'
    , 'a'))

# Returns 8....but there are 9 :(
# What happened?! 🤔
```

DAMN!
WHAT DO I DO NOW??

100%

How can I be confident that I’ve fixed the bug but not created new ones in the process?

TDD Time!

Yikes a Bug!
How Would TDD Apply Here?

1. Write your test first.
2. Add the docstring with your requirements for the test.
3. Run your test (yes, even before you've written any code). Ensure it fails.
4. Fix the code.
5. Run the test again and ensure it passes.

```python
from counter import count_letter

def test_lowercase_letter_count():
    """
    Pass in a lower case letter

    Return the correct count of all
    upper and lower case instances
    of that letter

    """
    expected_result = 3
    result = count_letter('Hi there, Eduardo!', 'e')
    assert result == expected_result

```

Step 1: Let's write a test and our requirements!

```
counter_test.py F

========================================= FAILURES =========================================
_____________________________ test_lowercase_letter_count _____________________________

    def test_lowercase_letter_count():
        """
        Pass in a lower case letter

        Return the correct count of all
        upper and lower case instances
        of that letter

        """
        expected_result = 3
        result = count_letter('Hi there, Eduardo!', 'e')
>       assert result == expected_result
E       assert 2 == 3

counter_test.py:14: AssertionError
================================== 1 failed in 0.03 seconds ==================================
```

Step 2: Let's run our test and make sure it fails

```
1  def count_letter(text, letter_to_find):
2      """
3      Counts the number of times the given
4      letter appears in the text
5      """
6      count = 0
7      for character in text.lower():
8          if character == letter_to_find:
9              count += 1
10     return count
11
```

Step 3: Let's add our solution

```
========================================= test session starts ==========================================
platform darwin -- Python 2.7.10, pytest-3.0.3, py-1.4.31, pluggy-0.4.0
rootdir: /Users/sanajaved/Desktop/presentation, inifile:
collected 1 items

counter_test.py .

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

Step 4: Run our test and see if they pass

```python
from counter import count_letter

def test_lowercase_letter_count():
    """
    Pass in a lower case letter

    Return the correct count of all
    upper and lower case instances
    of that letter

    """
    expected_result = 3
    result = count_letter('Hi there, Eduardo!', 'e')
    assert result == expected_result


def test_uppercase_letter_count():
    """
    Pass in an upper case letter

    Return the correct count of all
    upper and lower case instances
    of that letter

    """
    expected_result = 3
    result = count_letter('Hi there, Eduardo!', 'E')
    assert result == expected_result

```

We know we ran into issues with lowercase vs. uppercase letters.

Let's add a test to make sure we're covered.

Step 1: Test and documentation!

```
collected 2 items

counter_test.py .F

================================================ FAILURES =================================================
_____________________________________ test_uppercase_letter_count _____________________________________

    def test_uppercase_letter_count():
        """
        Pass in an upper case letter

        Return the correct count of all
        upper and lower case instances
        of that letter

        """
        expected_result = 3
        result = count_letter('Hi there, Eduardo!', 'E')
>       assert result == expected_result
E       assert 0 == 3

counter_test.py:27: AssertionError
================================== 1 failed, 1 passed in 0.03 seconds ==================================
```

Step 2: Ensure test fails

```
1  def count_letter(text, letter_to_find):
2      """
3      Counts the number of times the given
4      letter appears in the text
5      """
6      count = 0
7      for character in text.lower():
8          if character == letter_to_find.lower():
9              count += 1
10     return count
11
```

Step 3: Let's add code that we think will fix things

```
collected 2 items

counter_test.py ..

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

Step 4: Run tests and tests pass! Hooray!

But wait…
How do I know my tests aren’t buggy?

This is why we run tests before we write code!

A critical part of TDD is running the test before you've implemented the solution.

This helps us ensure that our tests aren't giving us a false positive or negative.

If we get a positive test when we don't have a solution yet, we know the test is wrong.

But doesn’t this take a lot longer?

Yes, but...it all depends.

- Writing tests takes longer in the short term. But the return on the investment with tests happens when:
  - You can onboard new developers without fear that they'll break anything
  - You'll be able to add new features faster without fear of regressions
  - You'll be able to refactor/rewrite code without fear of breaking a working system
- Without test developer time is lost when:
  - Even the smallest of changes break features across the code base
  - Hunting down bugs
  - Trashing a code base because it can't adapt

Testing & Refactoring

Testing isn’t only useful for bug fixing.

It can be difficult to remember why someone else or even yourself wrote something a certain way several months later.

Tests gives developers some trust that they can change an application and know they won’t break things in the process.

Let’s refactor :)

```python
# def count_letter(text, letter_to_find):
#     """
#     Counts the number of times the given
#     letter appears in the text
#     """
#     count = 0
#     for character in text.lower():
#         if character == letter_to_find.lower():
#             count += 1
#     return count

def count_letter(text, letter_to_find):
    """
    Counts the number of times the given
    letter appears in the text
    """
    lower_text = text.lower()
    count = lower_text.count(letter_to_find.lower())
    return count

```

```
collected 2 items

counter_test.py ..

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

Tests pass! Hooray!

Summary of TDD

1. Write your test first.
2. Add the docstring with your requirements for the test.
3. Run your test (yes, even before you've written any code). Ensure it fails.
4. Fix the code.
5. Run the test again and ensure it passes.

Questions?