TDD is not JUST about tests

Fabrizio Romano

@gianchub

EuroPython 2015 - Bilbao

July 21th

Hello Bilbao!

Thank you for being here!

If you want to know more about me, here's a LITTLE bit...

$01100101011_{0}0110$

Got it?

The plan:

Hello! (Amazing joke about the bit...) ✓

The plan:

Hello! (Amazing joke about the bit...)

The plan (we're here)

Disclaimer

What drew me to TDD?

Why do we need it?

A story about TDD

The plan (we're here) ✓

What drew me to TDD?

Why do we need it?

A story about TDD

A little disclaimer

- This is my own view
- Simple examples
- No definitions

What drew me to TDD?

It all happened in London…

Mark Henwood

*“ Don't worry,
TDD will take us there*

Ondrej Kohout

“ Too much logic!

If someone is
achieving great results:

Observe and learn

Why do we need TDD?

Example #1

```python
def is_positive(n):
    # We assume n is integer.
    return n > 0
```

How do we test this function?

B*oundaries*

```
def is_positive(n):
    # We assume n is integer.
    return n > 0


eq(False, is_positive(0))
eq(False, is_positive(-3))
eq(True, is_positive(3))
```

Is this a good test?

```
def is_positive(n):
    # We assume n is integer.
    return n > 1


eq(False, is_positive(0))   # still passing
eq(False, is_positive(-3))  # still passing
eq(True, is_positive(3))    # still passing
```

Granularity

```python
def is_positive(n):
    # We assume n is integer.
    return n > 1


eq(False, is_positive(0))   # still passing
eq(False, is_positive(-1))  # still passing
eq(True, is_positive(1))    # NOW FAILS!
```

Much better!

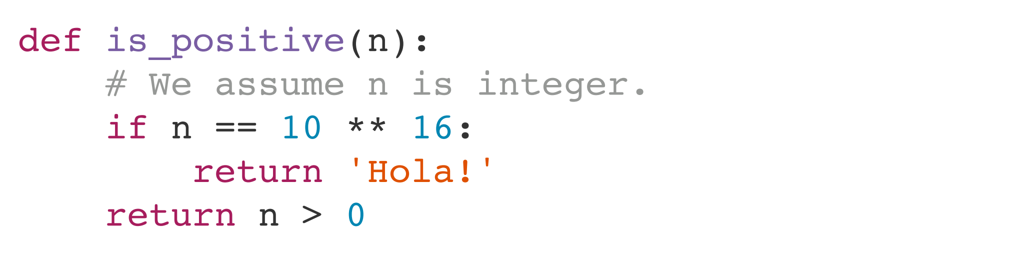
The boundary cannot jiggle any more!

```python
def is_positive(n):
    # We assume n is integer.
    return n > 0


eq(False, is_positive(0))

for n in range(10 ** 4):
    eq(False, is_positive(-n))
    eq(True, is_positive(n))
```

Even better!

“ *But unit tests need to be FAST…*

So, can we test everything?



```python
def is_positive(n):
    # We assume n is integer.
    if n == 10 ** 16:
        return 'Hola!'
    return n > 0
```

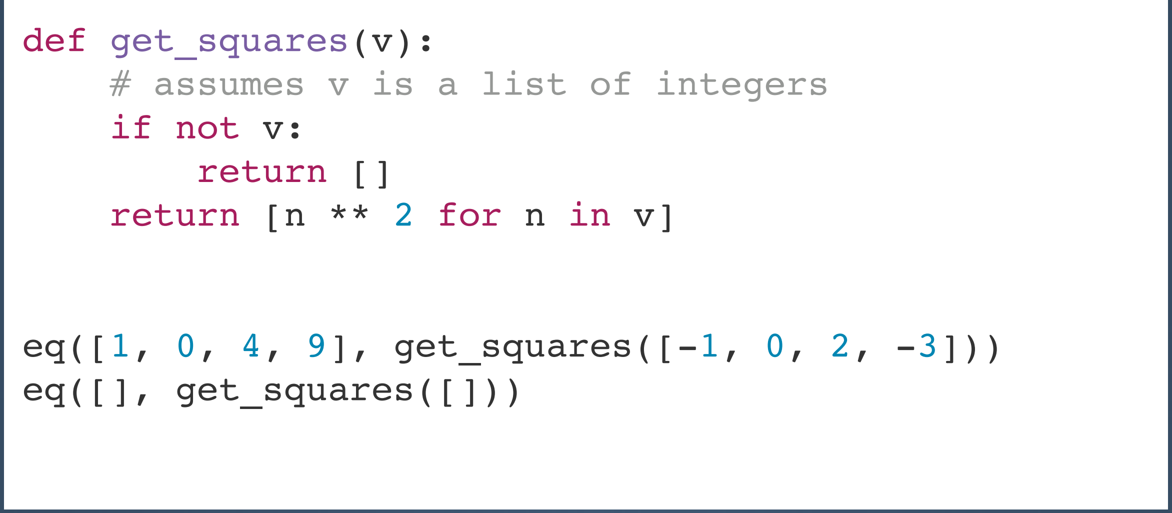
If you could test 1'000'000'000 numbers a second, it would take about **4 months** to spot this.

We **cannot** test everything.

Example #2

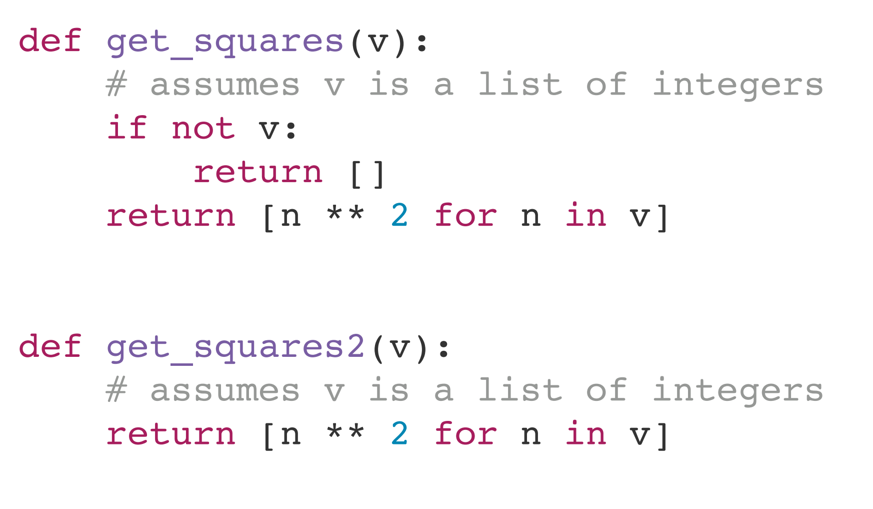
```python
def get_squares(v):
    # assumes v is a list of integers
    if not v:
        return []
    return [n ** 2 for n in v]
```

How do we test this function?

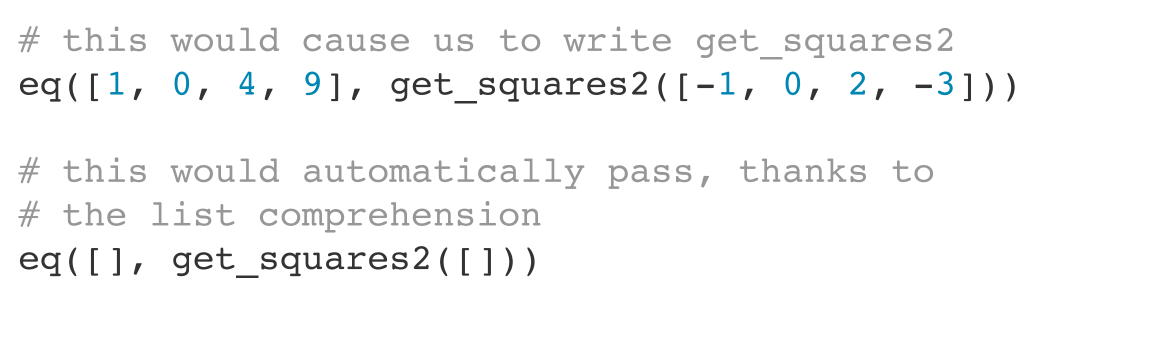


```
def get_squares(v):
    # assumes v is a list of integers
    if not v:
        return []
    return [n ** 2 for n in v]


eq([1, 0, 4, 9], get_squares([-1, 0, 2, -3]))
eq([], get_squares([]))
```

But what about that redundancy?
We may not notice it, if we tested AFTERWARDS



```python
def get_squares(v):
    # assumes v is a list of integers
    if not v:
        return []
    return [n ** 2 for n in v]


def get_squares2(v):
    # assumes v is a list of integers
    return [n ** 2 for n in v]
```



```python
# this would cause us to write get_squares2
eq([1, 0, 4, 9], get_squares2([-1, 0, 2, -3]))

# this would automatically pass, thanks to
# the list comprehension
eq([], get_squares2([]))
```

Had we written the tests first, there would be **no redundancy.**

Better than solving these problems, is to ***avoid introducing them*** in the first place.

And ***that's where TDD*** comes into the game

So let's hear a story about TDD

> *Psst: It's extremely technical, beware!*

Once Upon a Time…

there was a frog

He was in love with a beautiful princess. One day, at the pond, the frog took courage, jumped out of the water and told her:

“ I am a prince under a spell, kiss me, break the spell and marry me!

Prince? Intriguing!

But she was a Geek...

Let me get back to you on this...

And he ran off to learn TDD from the masters...

Kent Becko

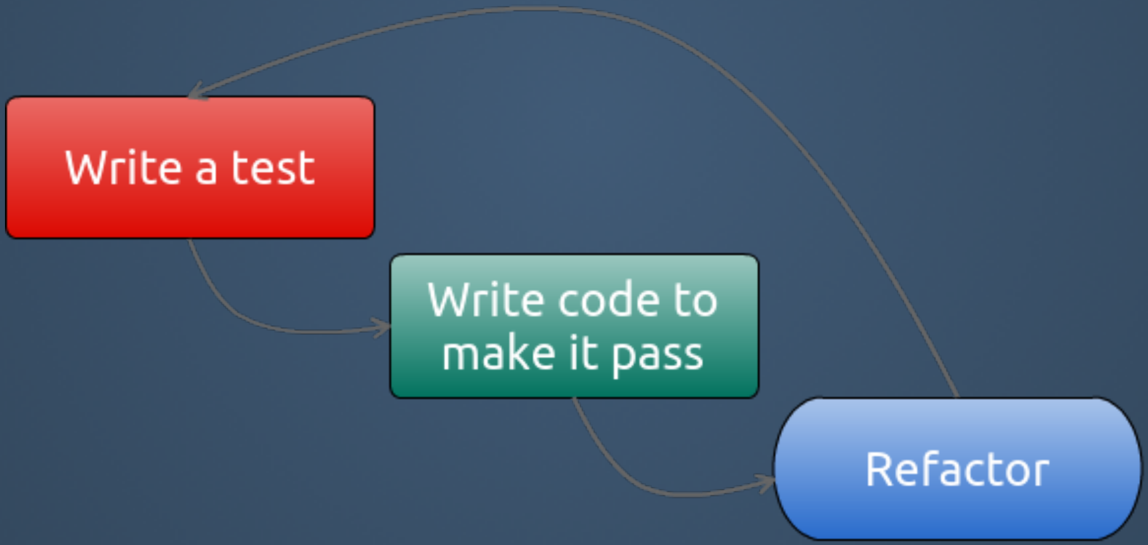
Robert Martiño

TDD was strooong in them...

TDD

Test Driven Development

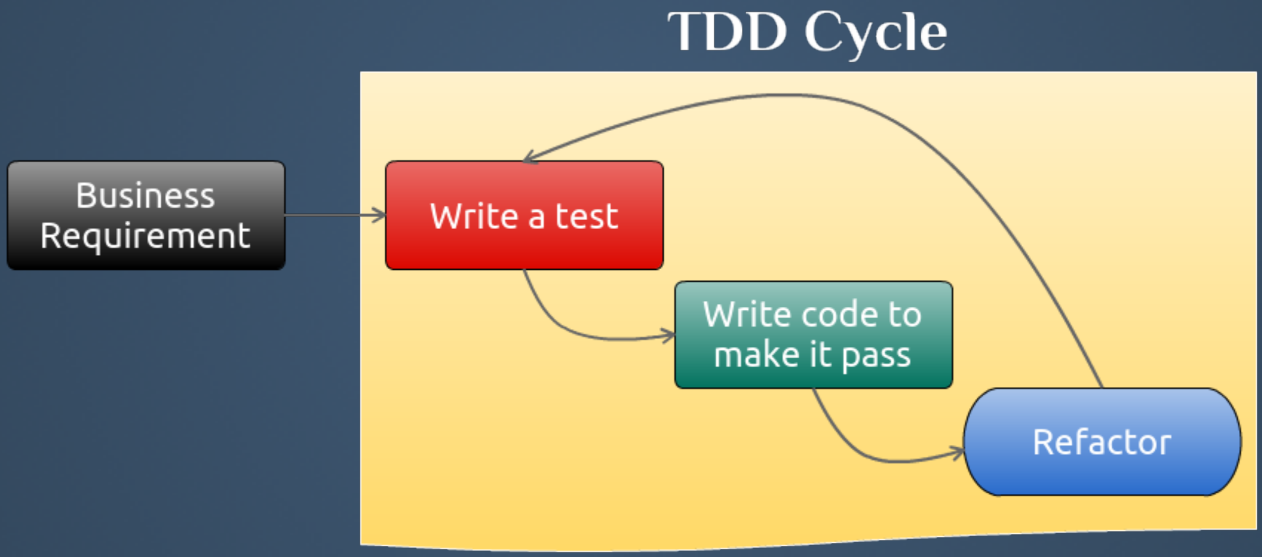
Def: TDD is a Software Development Process based on the repetition of a very short development cycle.



Steps

- Short at first
- With experience: longer
- Trouble? Go back to short

Where does it start?



TDD Cycle

Business Requirement

Write a test

Write code to make it pass

Refactor

The frog thought the training was completed

But the masters disagreed,
and they kept giving examples...

What changes?

Without TDD

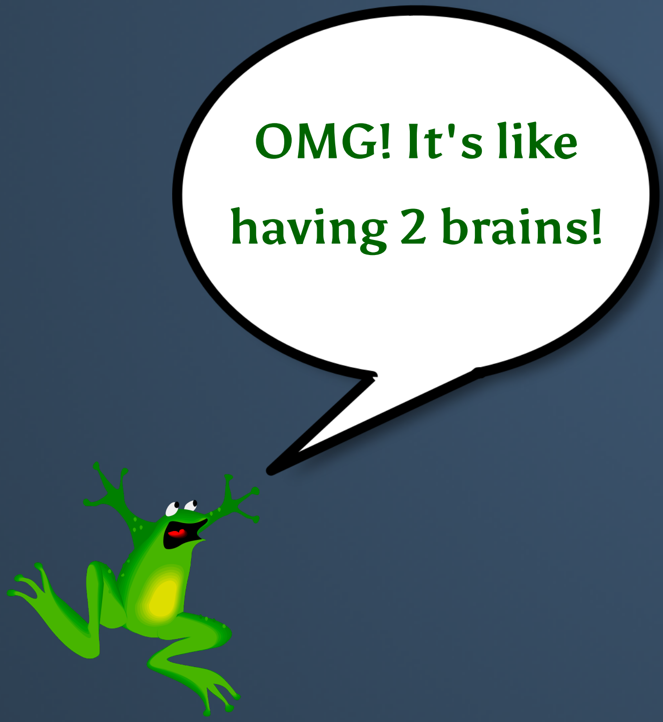
☹

What & How

With TDD

☺

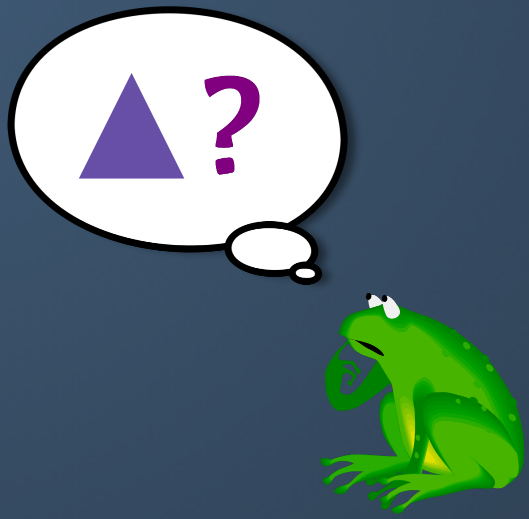
What How



TDD common aspects

- KISS
- YAGNI
- Three strikes and refactor

  (Test-Driven Development with Python - H. Percival)
- Architecture design during refactoring
- Triangulation



Triangulation

First step

```
eq(4, square(-2))


def square(n):
    # we can cheat, it is
    # the only requirement
    return 4
```

"Fake it 'till you make it"

With Triangulation

```
eq(4, square(-2))
eq(9, square(3))


def square(n):
    return n ** 2
```

You write

the actual logic.

Main Benefits

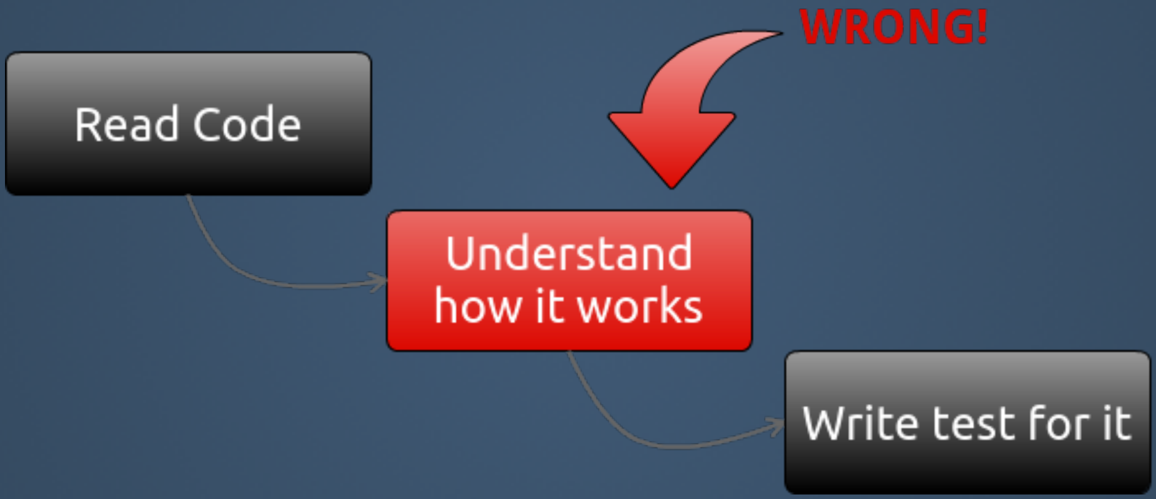
- Refactor with Confidence
- Readability
- Loose Coupling
- Easier to test and maintain
- Test first => Better understanding of requirements
- Small units => easier debugging and tests as docs
- Higher speed:
  It takes less to write tests and code
  than to write code and debug

Main Shortcomings

- Whole company needs to believe
- Blind spots
- Badly written tests are hard to maintain

Real life examples

How do you test legacy code?



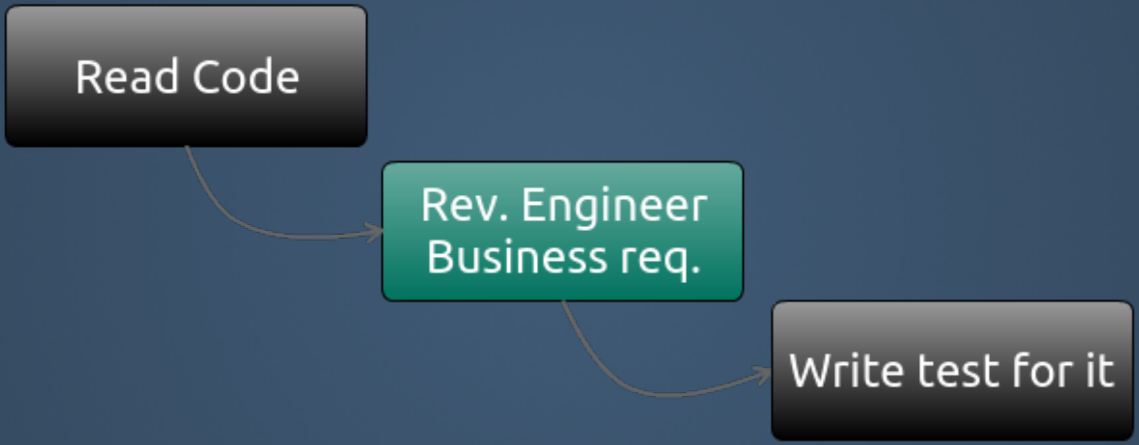
Read Code

Understand how it works

Write test for it

WRONG!

Much better way…



Read Code

Rev. Engineer Business req.

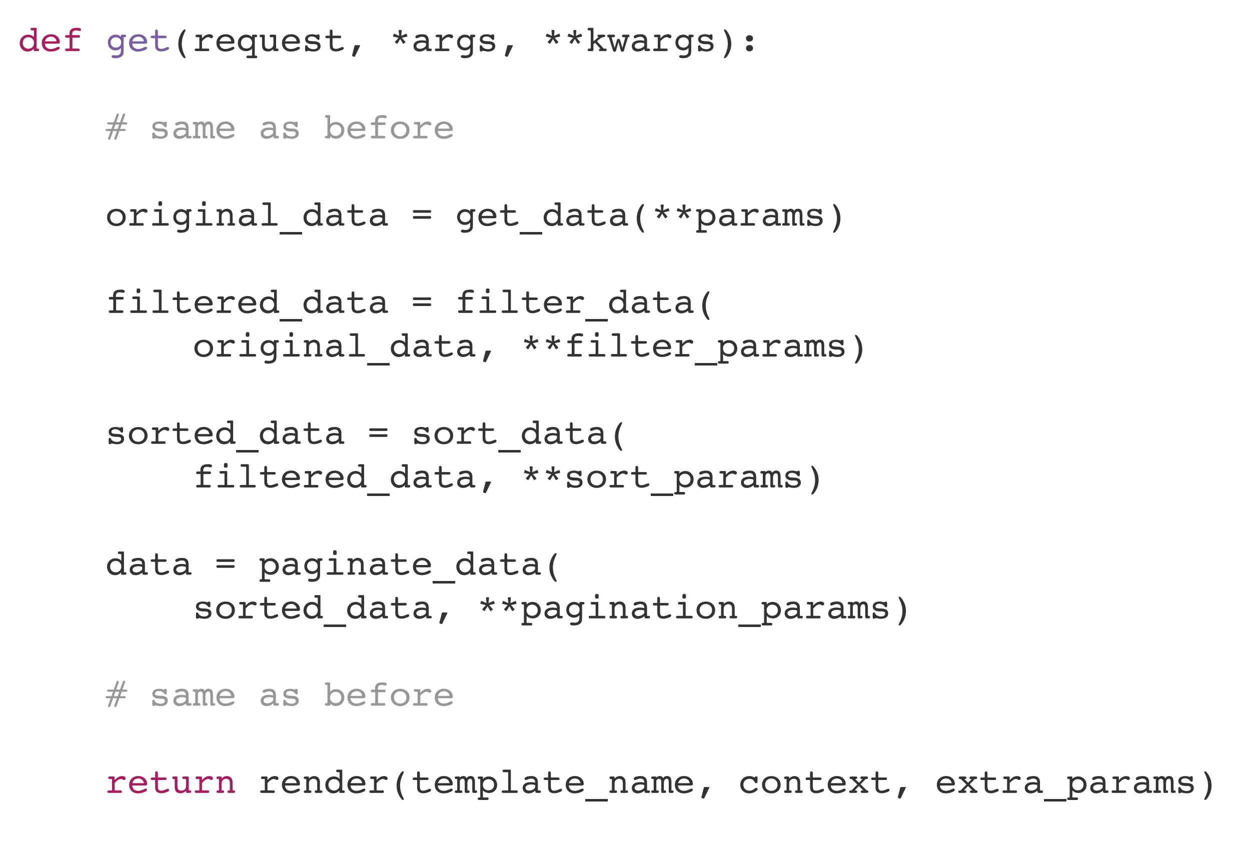
Write test for it

Changing a horribly long view

We need to insert pagination, filtering, sorting

```
def get(request, *args, **kwargs):

    # ...
    # imagine many lines of code here...
    # ...

    data = get_data(**params)

    # ...
    # data is prepared, worked on
    # put in the context dictionary
    # and the view finally renders
    # a template
    return render(template_name, context, extra_params)
```

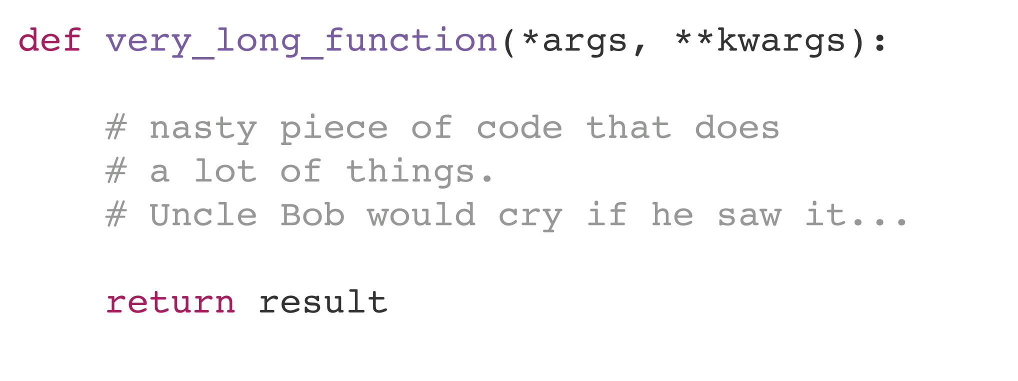
We code pagination, filtering and sorting with TDD



```python
def get(request, *args, **kwargs):

    # same as before

    original_data = get_data(**params)

    filtered_data = filter_data(
        original_data, **filter_params)

    sorted_data = sort_data(
        filtered_data, **sort_params)

    data = paginate_data(
        sorted_data, **pagination_params)

    # same as before

    return render(template_name, context, extra_params)
```

Introducing a new functionality in existing code

We need to add a feature to a long piece of untested code.



```python
def very_long_function(*args, **kwargs):

    # nasty piece of code that does
    # a lot of things.
    # Uncle Bob would cry if he saw it...

    return result
```

We cannot test it (no time, badly written, etc.)

One possible solution:

```python
def test_new_functionality():
    # preparation stage
    # ...

    result = very_long_function(*args, **kwargs)

    assert_equal(expected_result, result)


def very_long_function(*args, **kwargs):

    # same nasty piece of code
    # with NEW FUNCTIONALITY IN
    # Uncle Bob still unhappy

    return result
```

After all these examples,
the frog was in ZEN-Mode

He went back to the princess and passed the exam.

So they married, and when the minister said:

"You can kiss the bride"...

Nothing changed!

He was just a talking frog after all!

What's the moral of the story?

The princess should have *tested* ***FIRST!***

And so should you

Thank you! Come say hi!

f gianchub

🐦 gianchub

✉ gianchub [at] gmail [dot] com

http://slides.com/gianchub/ep2015-tdd