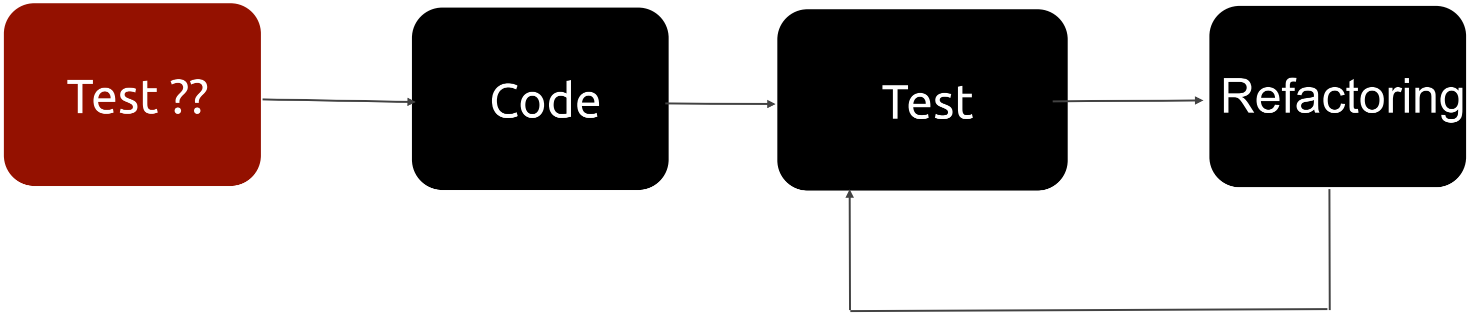
Test Driven Development

Course of Software Engineering II
A.A. 2011/2012

Valerio Maggio, PhD Student
Prof. Marco Faella

Development process

- Let's think about the development process of this example:



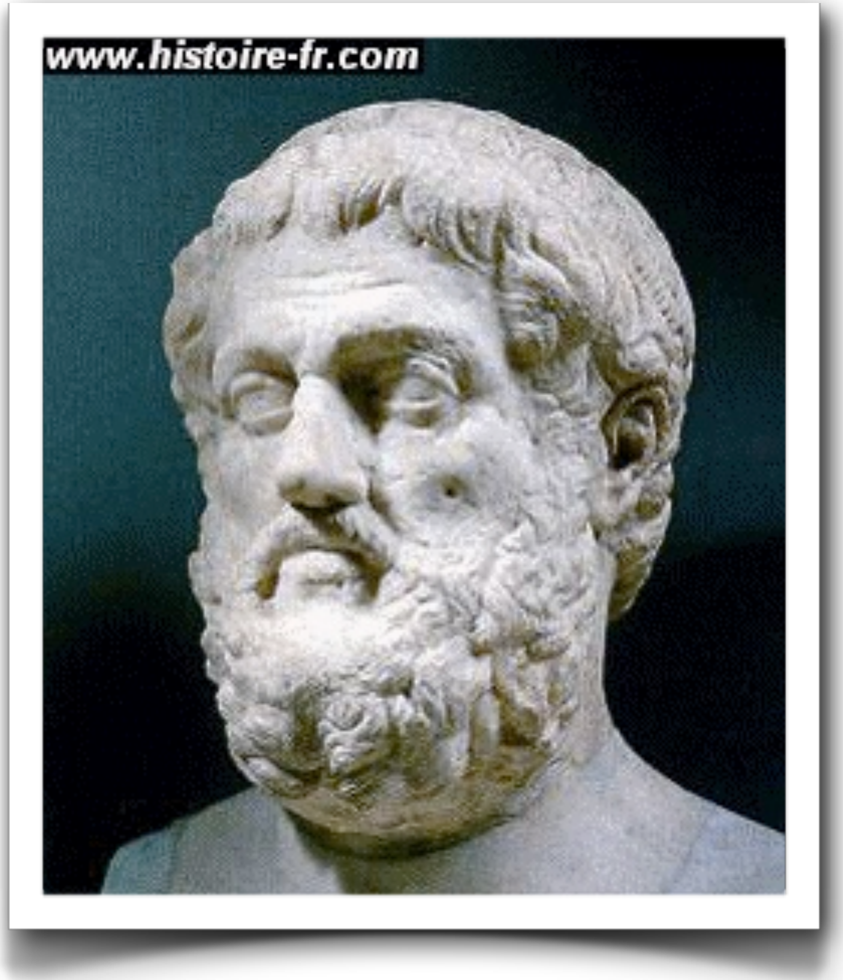
- **Q: Does make sense to write tests before writing production code?**
- **A: Two Keywords**
  - **TDD:** Test Driven Development
  - **Test-first Programming**

Outline

- What is TDD?
- TDD and eXtreme Programming
- TDD Mantra
- TDD Principles and Practices

1. Motivations

Software Development as a Learning Process



One must learn by doing the thing; for though you think you know it, you have no certainty until you try

Sofocle *(496 a.c. 406 a.C)*

Software Development as a Learning Process

- Almost all projects attempts something new
- Something refers to
  - People involved
  - Technology involved
  - Application Domain
  - … (most likely) a combination of these

Software Development as a Learning Process

- Every one involved has to learn as the projects progresses
  - Resolve misunderstanding along the way
- There will be changes!!
- Anticipate Changes
  - How ?

Feedback is a fundamental tool

- Team needs cycle of activities
  - Add new feature
  - Gets feedback about what already done!
- Time Boxes
- Incremental and Iterative Development
  - Incremental : Dev. feature by feature
  - Iterative: improvement of features in response to feedback

Practices that support changes

- Constant testing to catch regression errors
  - Add new feature without fear
  - Frequent manual testing infeasible
- Keep the code as simple as possible
  - More time spent reading code that writing it
- Simplicity takes effort, so Refactor

2. Test Driven Development

What is TDD ?

- TDD: Test Driven Development
  - Test Driven Design
  - Test-first Programming
  - Test Driven Programming
- Iterative and incremental software development
- TDD objective is to DESIGN CODE and not to VALIDATE Code
  - Design to fail principle

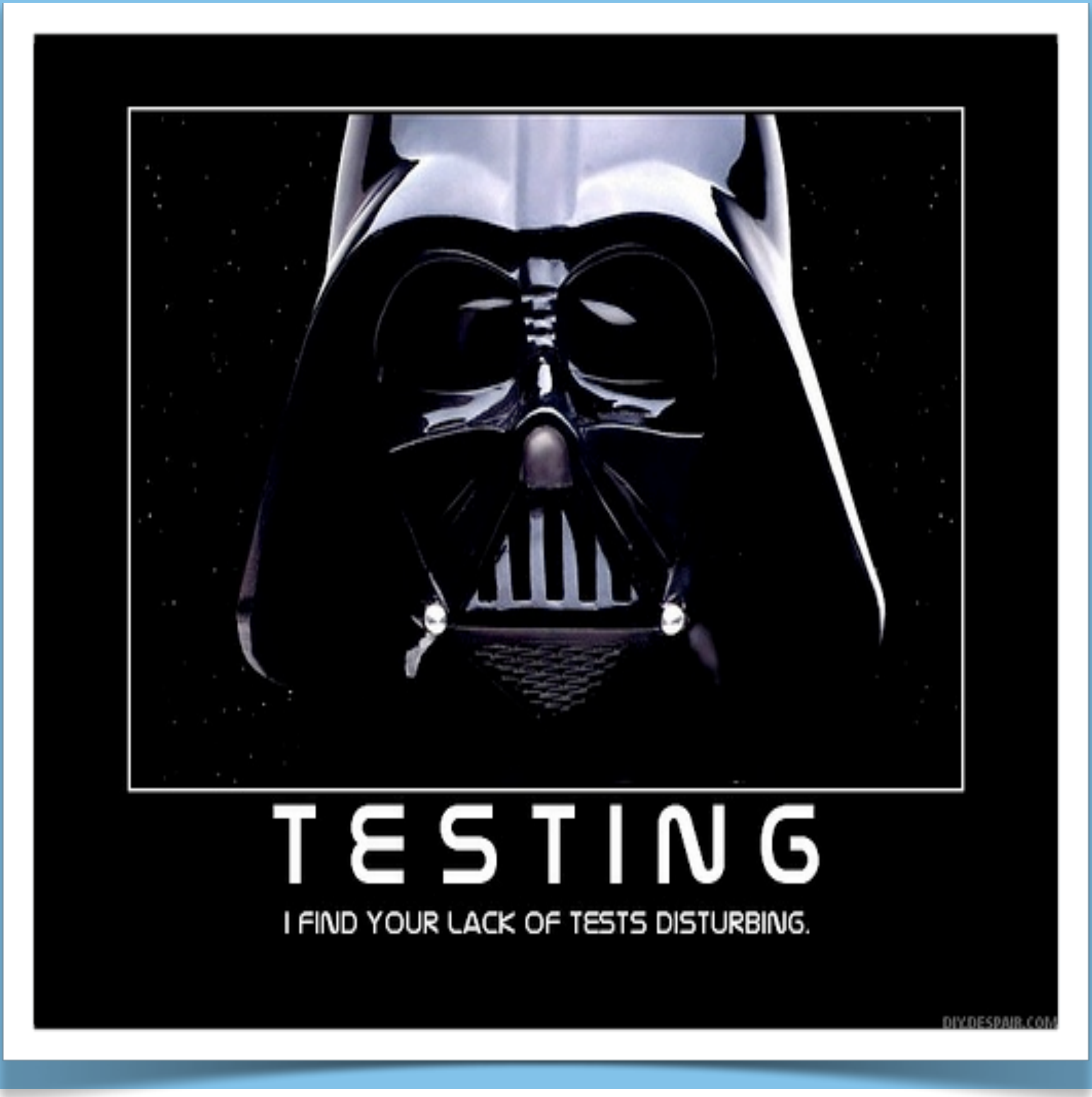
Test Driven Development

- We write tests before we write the code
- Testing as a way to clarify ideas about what we want the code has to do
- Testing as a Design Activity
  - Think about the feature
  - Write a test for that feature (Fail)
  - Write the code to pass the test
  - Run same previous test (Success)
  - Refactor the code

TDD and XP

- TDD vs XP
  - TDD is an agile practice
  - XP is an agile methodology
- Core of XP
  - No needs of others XP practices
- Avoid software regression
  - Anticipate changes
- Product code smarter that works better
- Reduce the presence of bugs and errors
  - “You have nothing to lose but your bugs”

3. TDD and Unit Testing



Unit test

- “ Unit tests run fast. If they don't run fast they're not unit tests. ”

- A test is not a unit test if:
  - communicate with DB
  - communicate with networking services
  - cannot be executed in parallel with other unit tests

- Unit tests overcome dependencies
  - How?
  - Why is it so important?

Unit Test and TDD

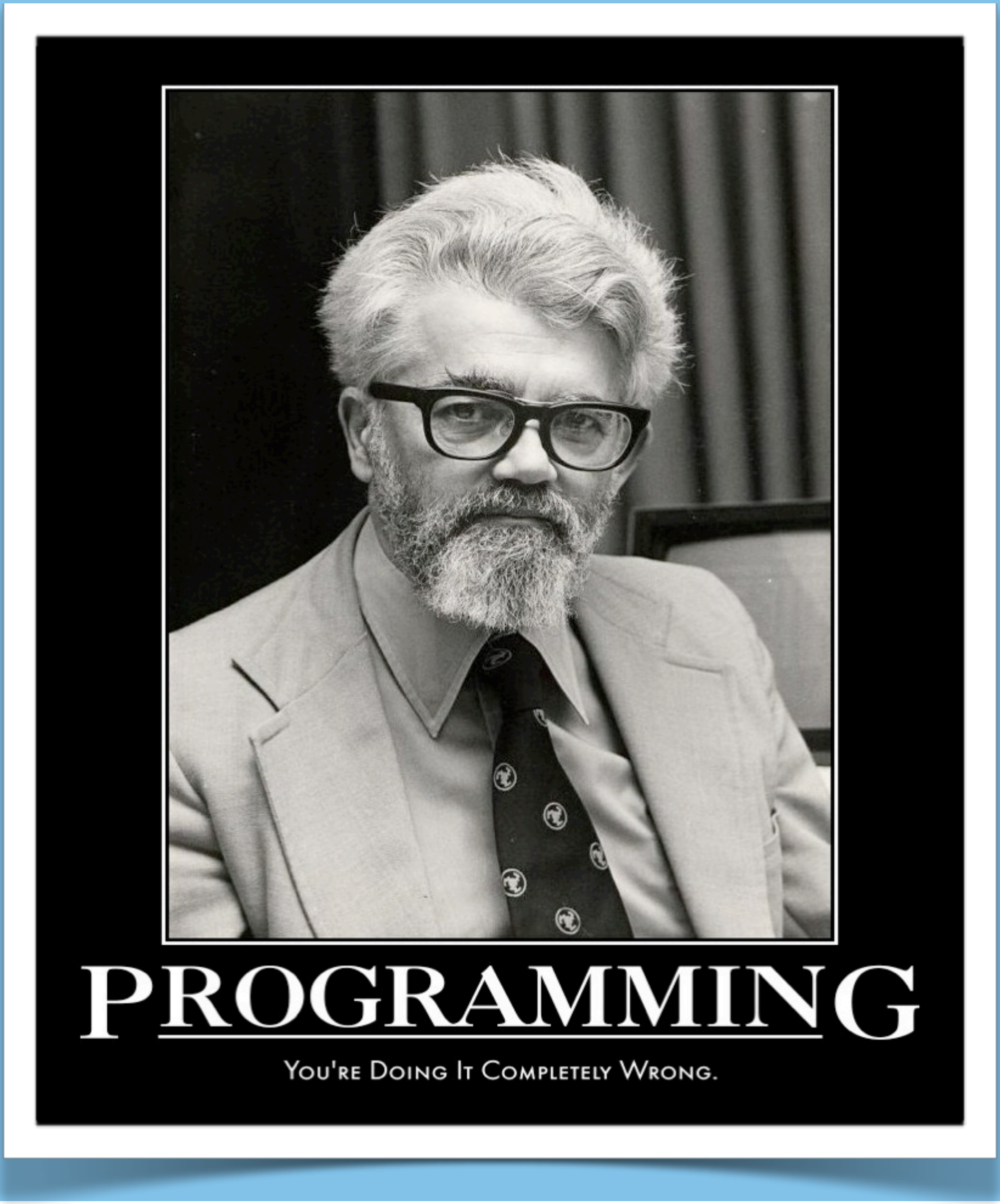
- Testing code is released together with production code

- A feature is released only if
  - Has at least a Unit test
  - All of its unit tests pass

- Do changes without fear
  - Refactoring

- Reduce debugging

4. TDD Mantra



TDD Mantra

Think : step by step

Think

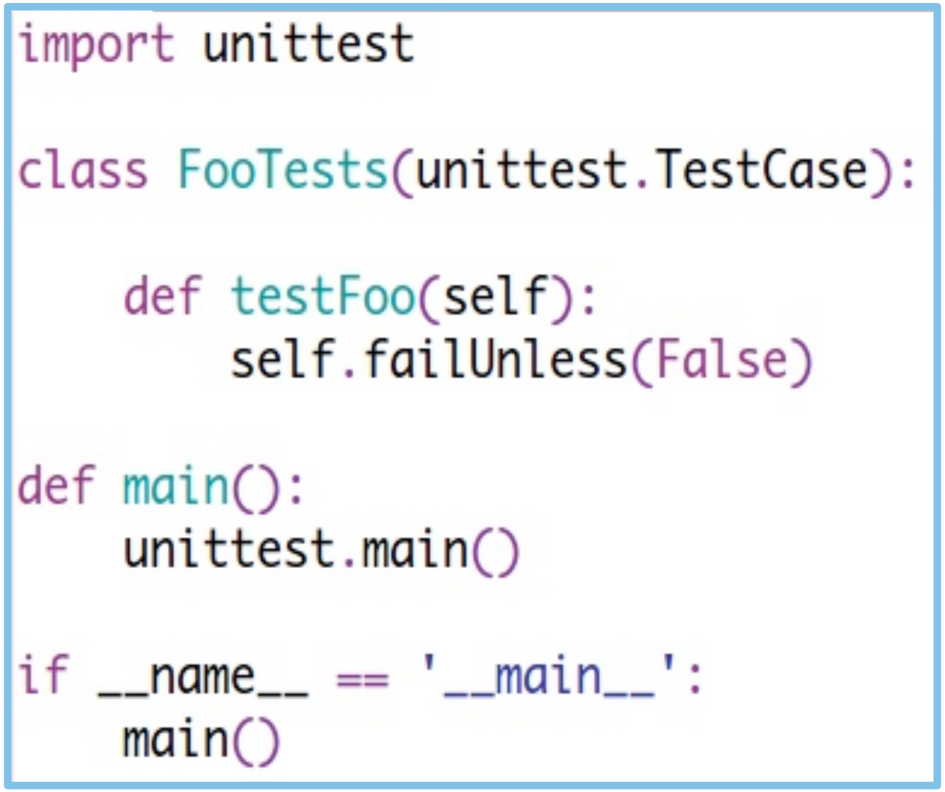
Think about what we want the code to do

TDD Mantra

Think : step by step

Think

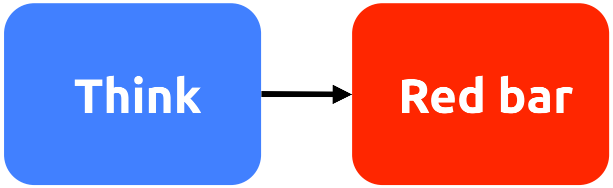
“Set up a *Walking Skeleton*”

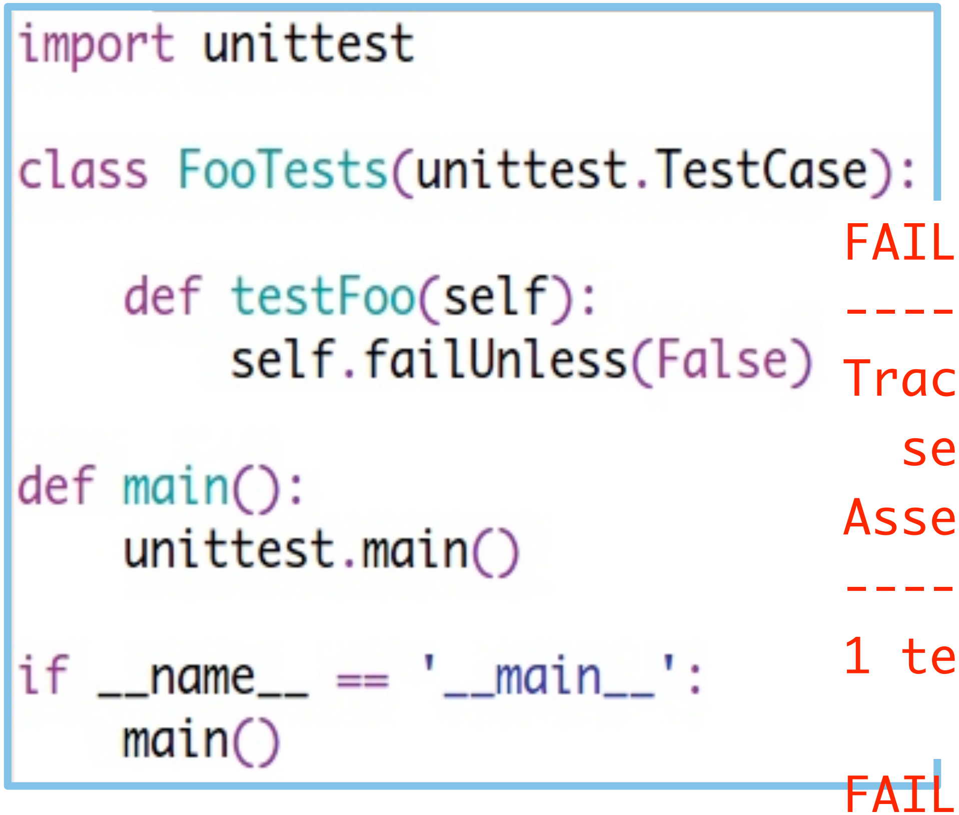


```
import unittest

class FooTests(unittest.TestCase):

    def testFoo(self):
        self.failUnless(False)

def main():
    unittest.main()

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

TDD Mantra

Red Bar : Writing tests that fails





```
import unittest

class FooTests(unittest.TestCase):

    def testFoo(self):
        self.failUnless(False)

def main():
    unittest.main()

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

DENIED

```
FAIL: testFoo (__main__.FooTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  self.failUnless(False)
AssertionError
----------------------------------------------------------------------
1 test in 0.003s

FAILED (failures=1)
```

TDD Mantra

Think : step by step

Think

“We want to create objects that can say whether two given dates "match".
These objects will act as a "pattern" for dates. ”

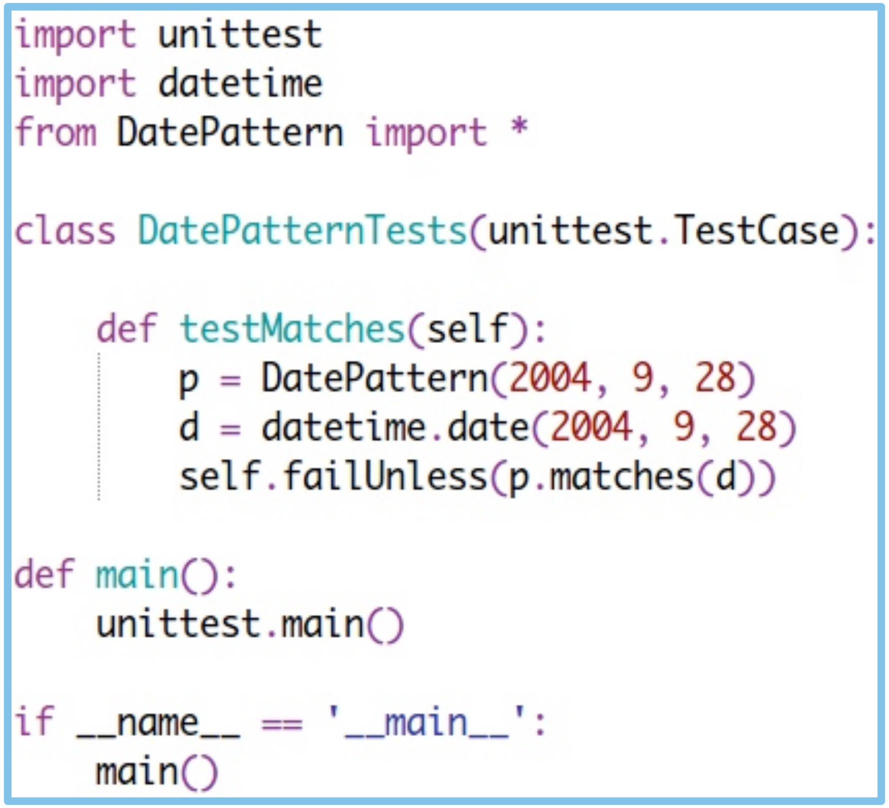
- So, Pattern....What is the pattern did you think about?
  - Design Pattern such as **Template Method**
  - Implementation Pattern such as **Regular Expressions**
- **Anyway, It doesn't matter now!**

TDD Mantra

Think : step by step

Think

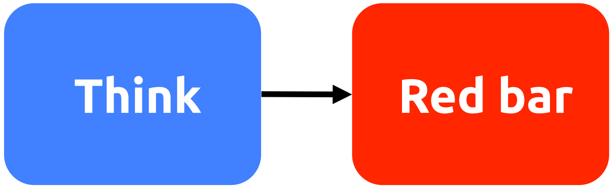
Feature 1: Date Matching



```
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestCase):

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

def main():
    unittest.main()

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

TDD Mantra

Red Bar : Writing tests that fails



Think

Red bar

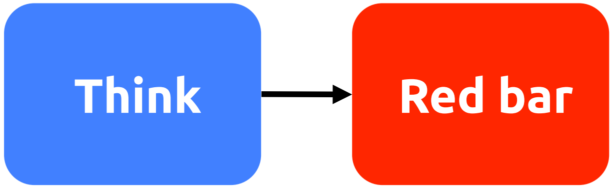
Think about the **behavior of the class** and its **public interface**

-What will you expect that happens?

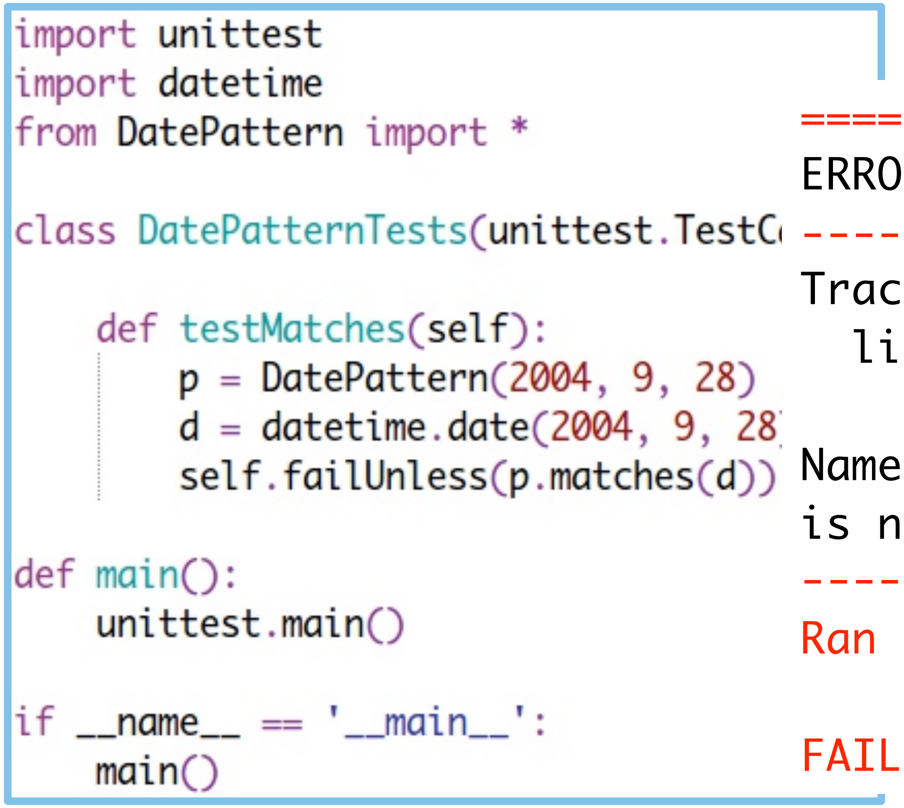
- **Why?**

TDD Mantra

Red Bar : Writing tests that fails



DENIED

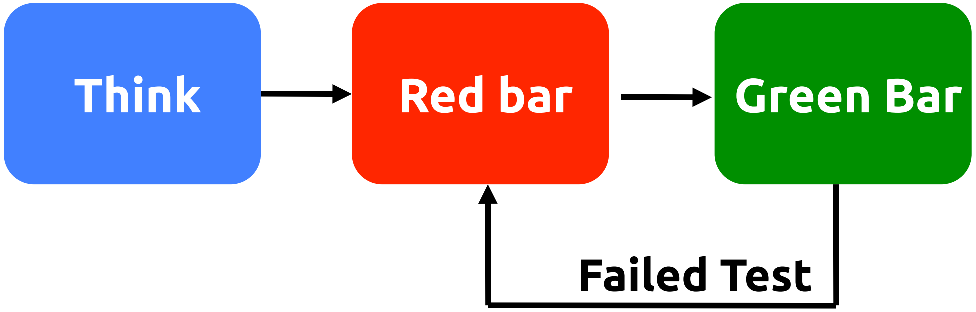


```
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestC

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28
        self.failUnless(p.matches(d))

def main():
    unittest.main()

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

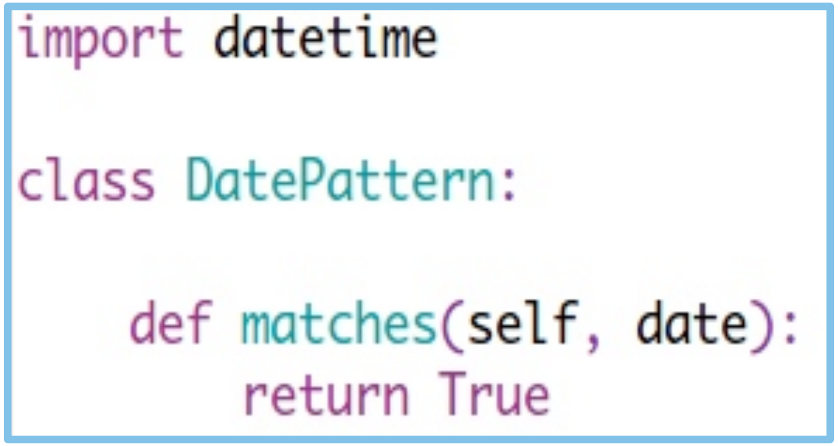
```
====================================
ERROR: testMatches
------------------------------------
Traceback (most recent call last):
  line 8, in testMatches
    p = DatePattern(2004, 9, 28)
NameError: global name 'DatePattern'
is not defined
------------------------------------
Ran 1 test in 0.000s

FAILED (errors=1)
```

TDD Mantra

Green Bar : Writing production code



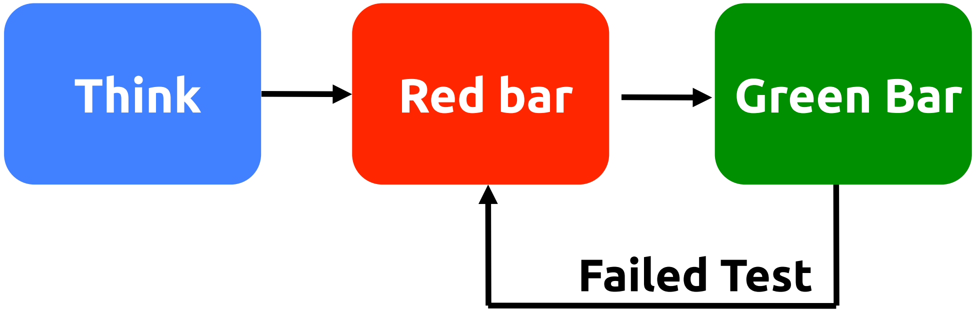
Write production code **ONLY** to pass previous failing test

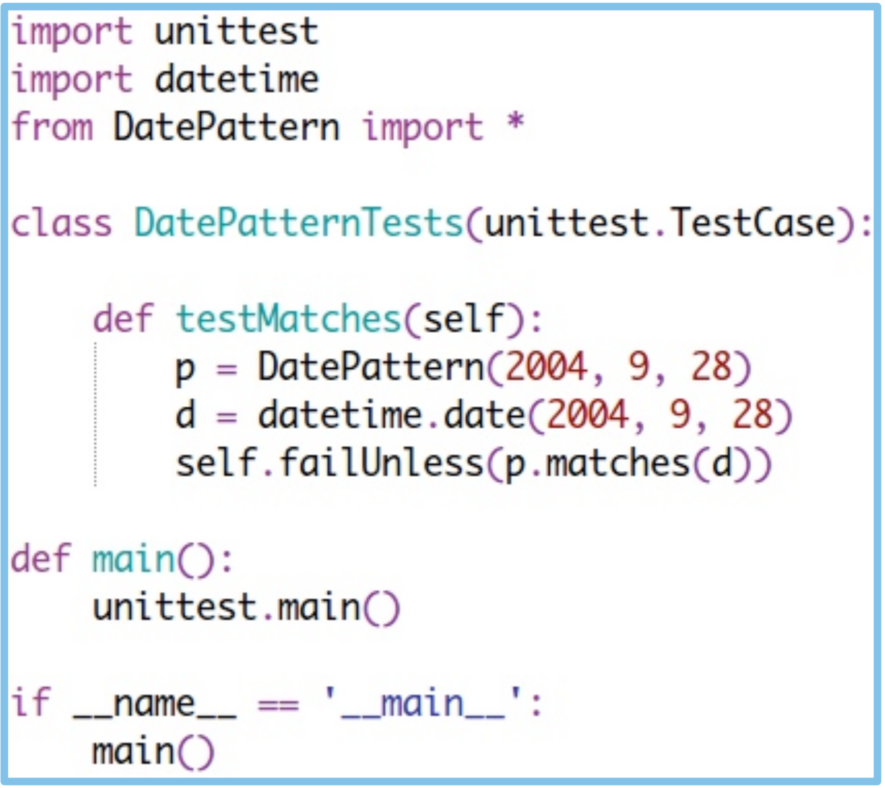


```
import datetime

class DatePattern:

    def matches(self, date):
        return True
```

TDD Mantra

Green Bar : Writing production code





```
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestCase):

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

def main():
    unittest.main()

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

APPROVED

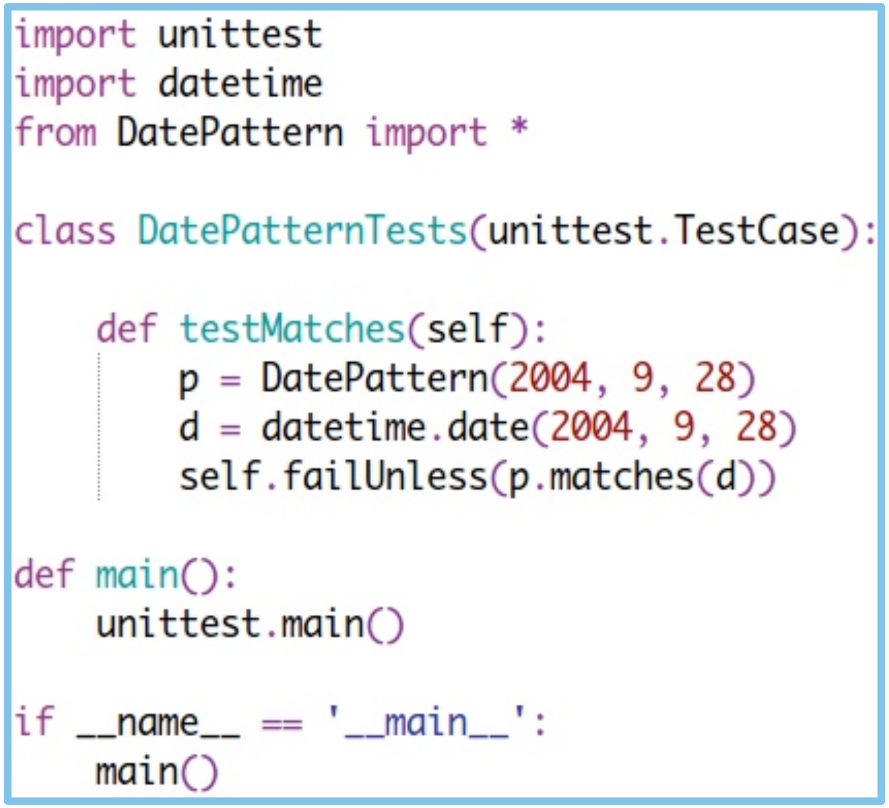
```
==========================
--------------------------
Ran 1 test in 0.000s

OK
```

TDD Mantra

Think : step by step

Think

Feature 1: Date Matching



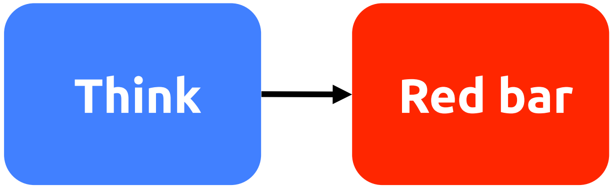
```python
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestCase):

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

def main():
    unittest.main()

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

Now that first test passes,
It's time to move to the second test!

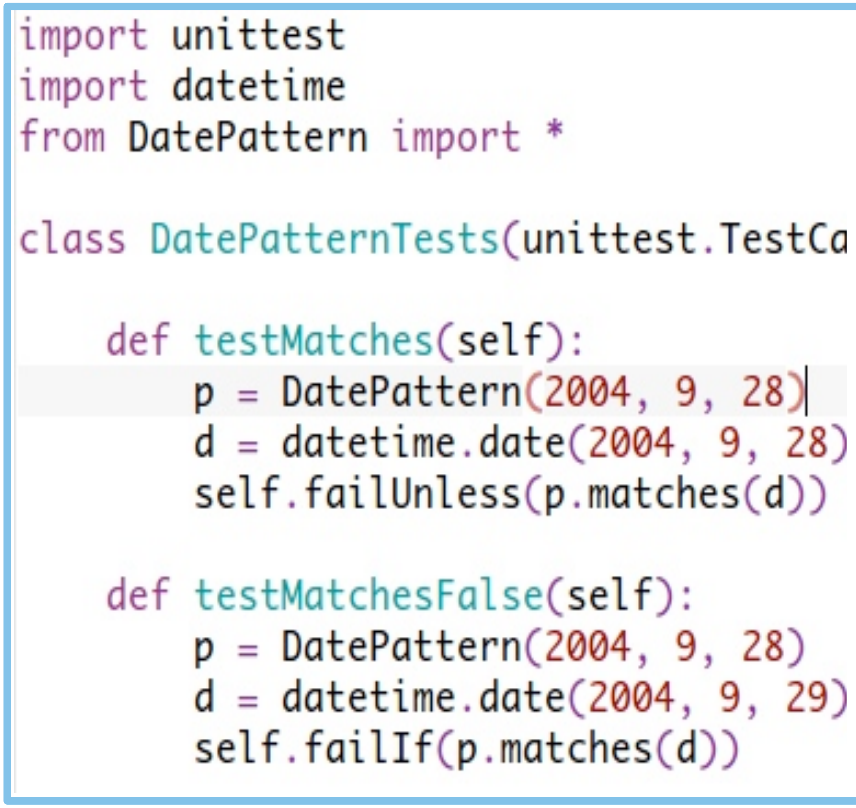
Any Guess?

TDD Mantra

Red Bar : Writing tests that fails



DENIED

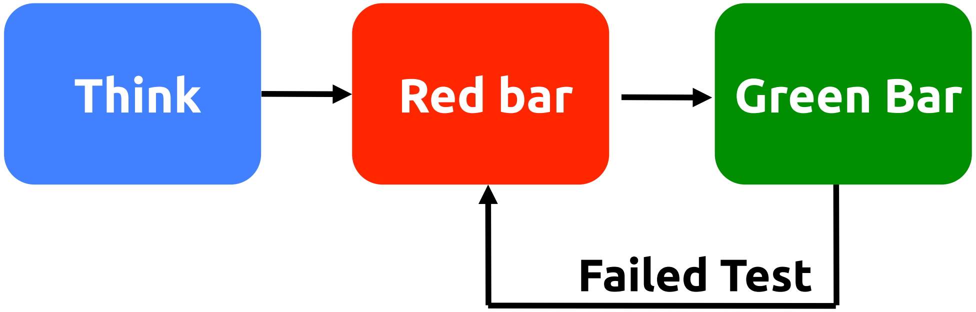


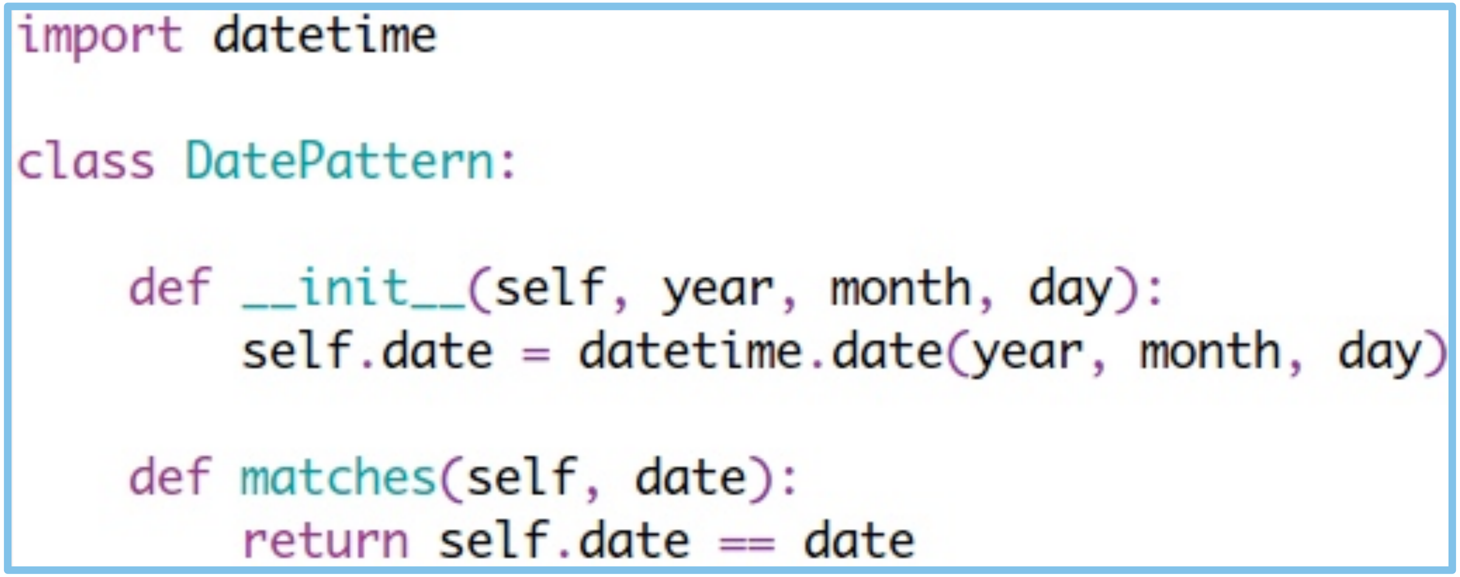
```
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestCa

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

    def testMatchesFalse(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 29)
        self.failIf(p.matches(d))
```

```
==================================
ERROR: testMatches
----------------------------------
Traceback (most recent call last):
  line 15, in testMatchesFalse
    self.failIf(p.matches(d))
AssertionError

-----------------------------------
Ran 2 tests in 0.001s

FAILED (failures=1)
```

TDD Mantra

Green Bar : Writing production code

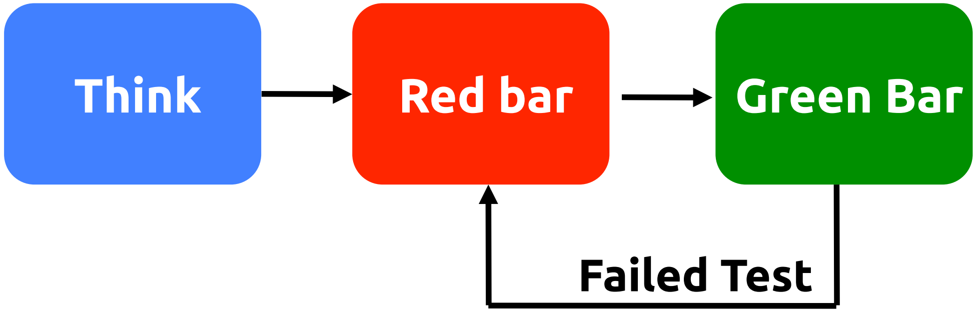


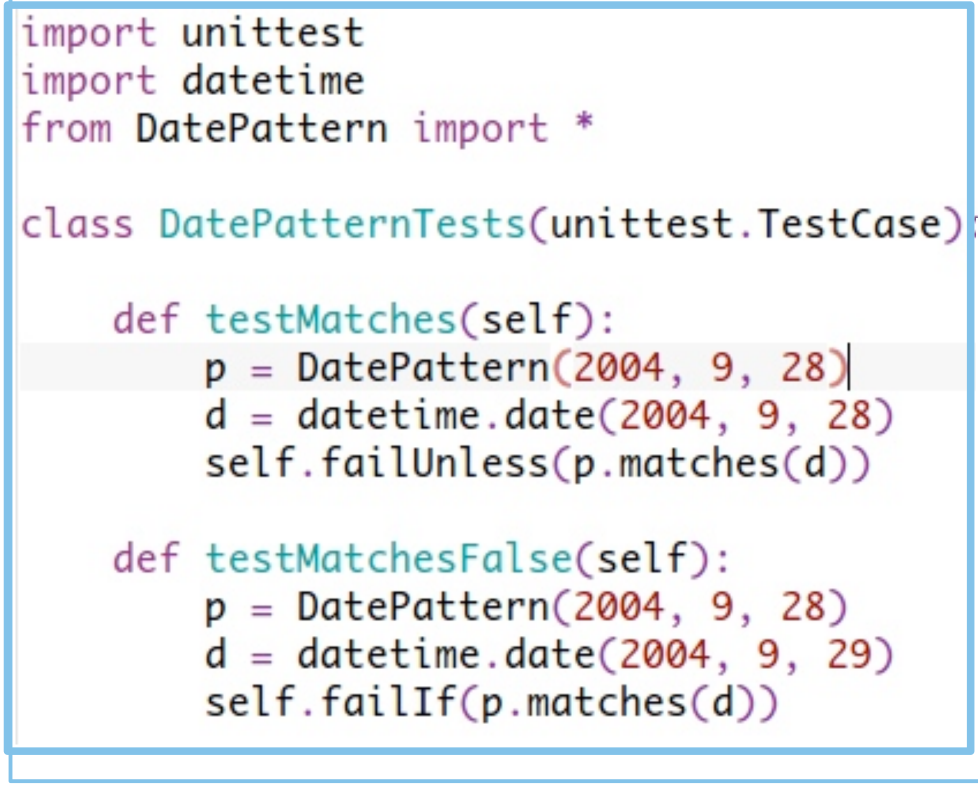


```
import datetime

class DatePattern:

    def __init__(self, year, month, day):
        self.date = datetime.date(year, month, day)

    def matches(self, date):
        return self.date == date
```

TDD Mantra

Green Bar : Writing production code





```
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestCase):

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

    def testMatchesFalse(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 29)
        self.failIf(p.matches(d))
```

APPROVED

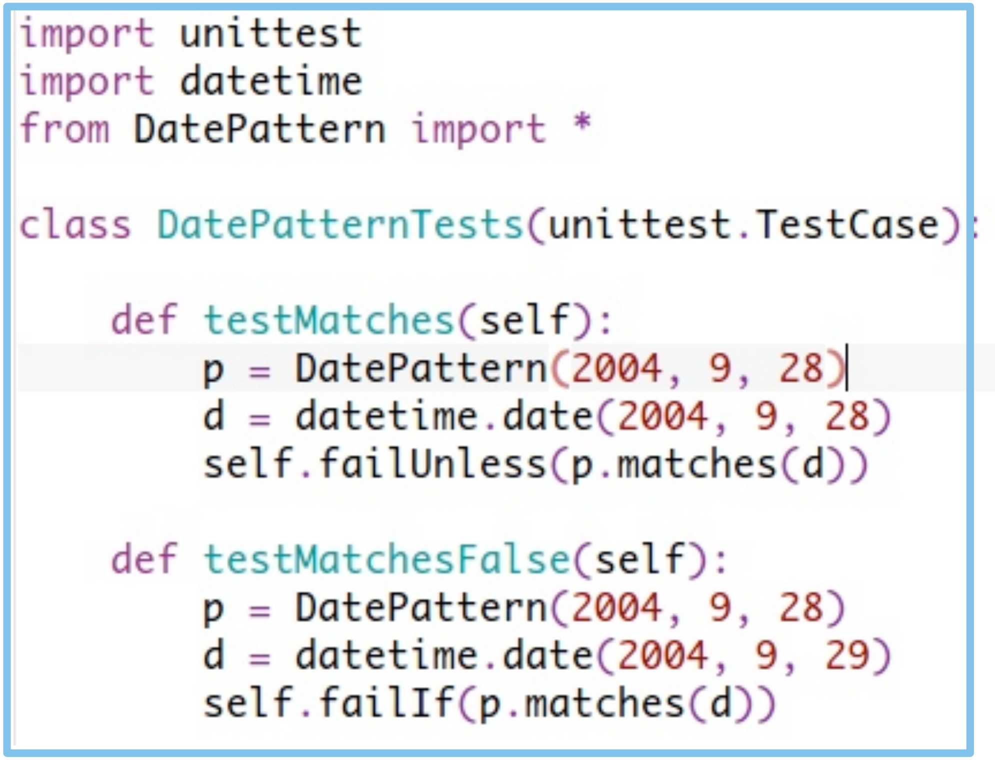
```
===========================
---------------------------
Ran 2 test in 0.000s

OK
```

TDD Mantra

Think : step by step

Think

Feature 2: Date Matching as a WildCard

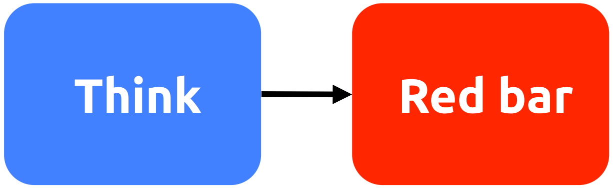


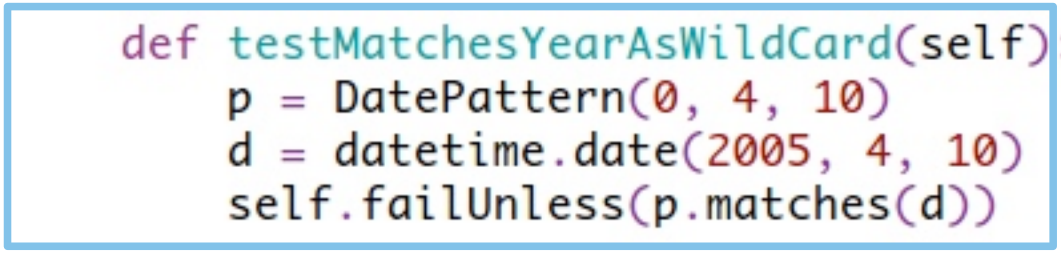
```
import unittest
import datetime
from DatePattern import *

class DatePatternTests(unittest.TestCase):

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

    def testMatchesFalse(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 29)
        self.failIf(p.matches(d))
```

What happens if I pass a zero as for the year parameter?

TDD Mantra

Red Bar : Writing tests that fails





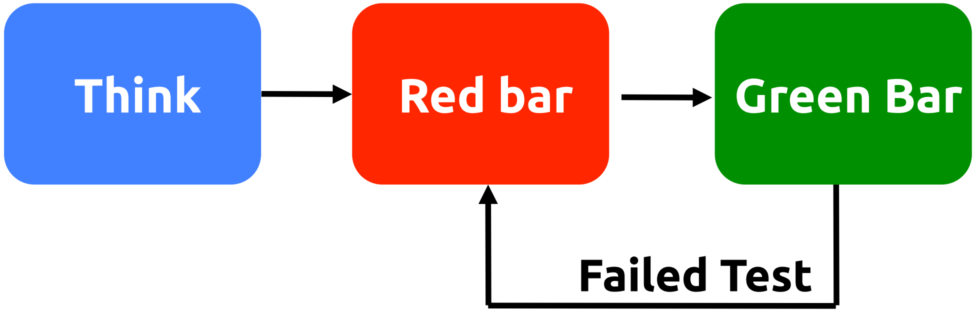
```
def testMatchesYearAsWildCard(self):
    p = DatePattern(0, 4, 10)
    d = datetime.date(2005, 4, 10)
    self.failUnless(p.matches(d))
```

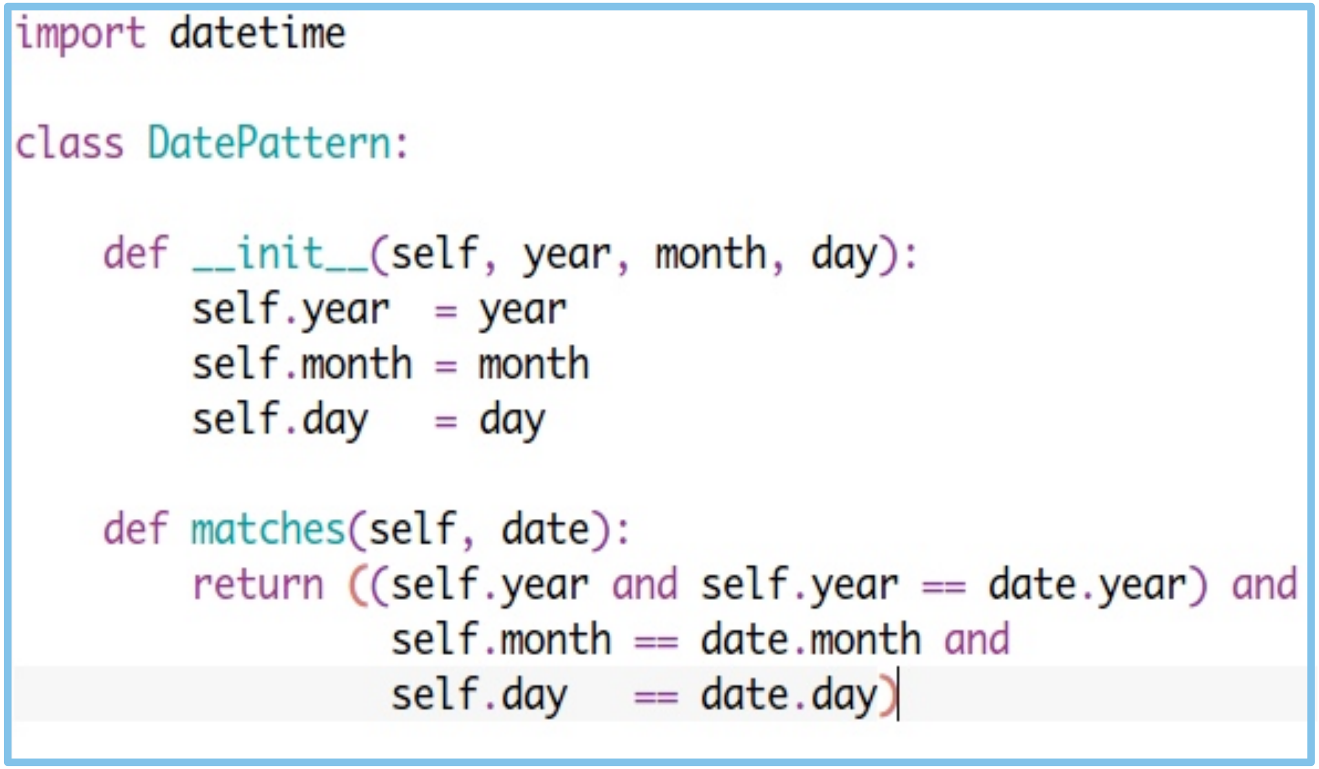
DENIED

```
===================================
ERROR testMatchesYearAsWildCard
---------------------------------------------
 [..]
ValueError: year is out of range
---------------------------------------------
Ran 3 tests in 0.000s
FAILED (errors=1)
```

TDD Mantra

Green Bar : Writing production code

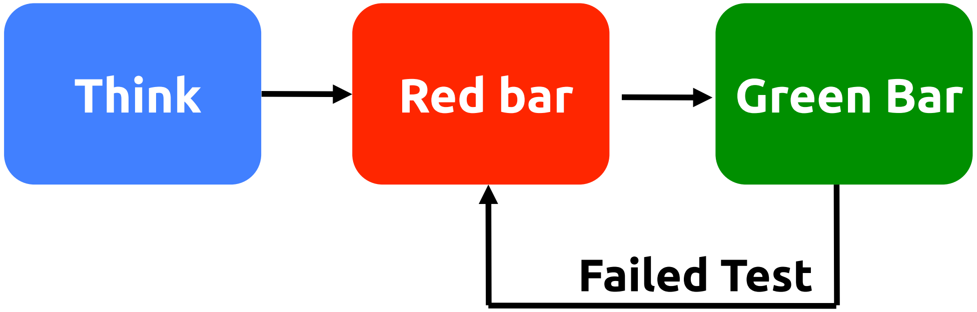


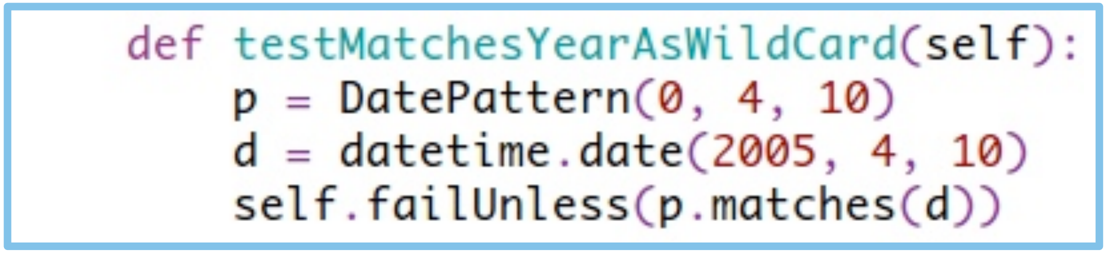


```
import datetime

class DatePattern:

    def __init__(self, year, month, day):
        self.year  = year
        self.month = month
        self.day   = day

    def matches(self, date):
        return ((self.year and self.year == date.year) and
                 self.month == date.month and
                 self.day   == date.day)
```

TDD Mantra

Green Bar : Writing production code





```
    def testMatchesYearAsWildCard(self):
        p = DatePattern(0, 4, 10)
        d = datetime.date(2005, 4, 10)
        self.failUnless(p.matches(d))
```

APPROVED

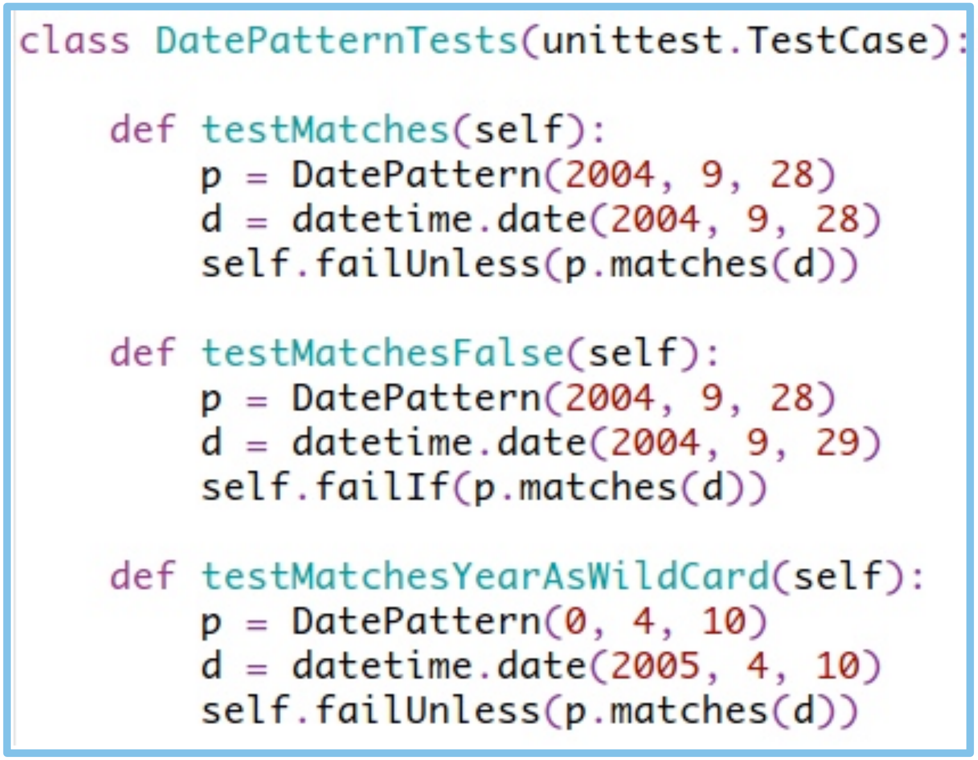
```
==========================
--------------------------
Ran 3 test in 0.000s

OK
```

TDD Mantra

Think : step by step

Think

Feature 3: Date Matching as a WildCard

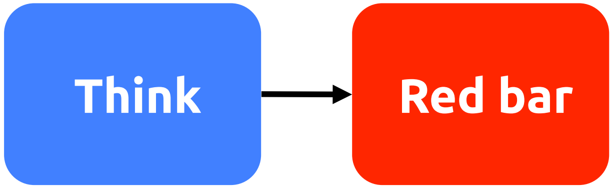


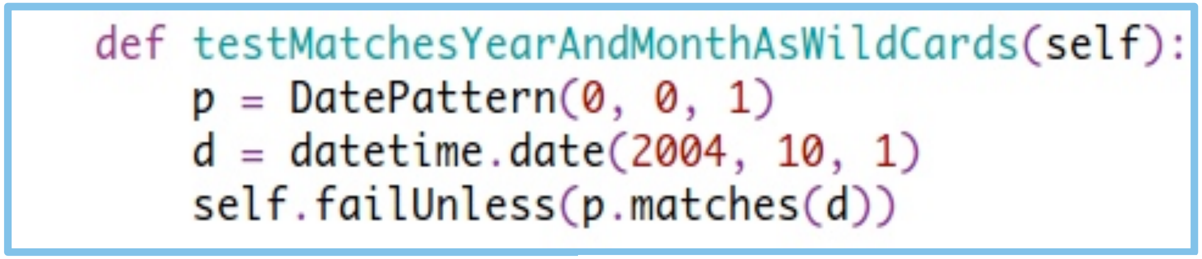
```
class DatePatternTests(unittest.TestCase):

    def testMatches(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 28)
        self.failUnless(p.matches(d))

    def testMatchesFalse(self):
        p = DatePattern(2004, 9, 28)
        d = datetime.date(2004, 9, 29)
        self.failIf(p.matches(d))

    def testMatchesYearAsWildCard(self):
        p = DatePattern(0, 4, 10)
        d = datetime.date(2005, 4, 10)
        self.failUnless(p.matches(d))
```

What happens if I pass a zero as for the month parameter?

TDD Mantra

Red Bar : Writing tests that fails





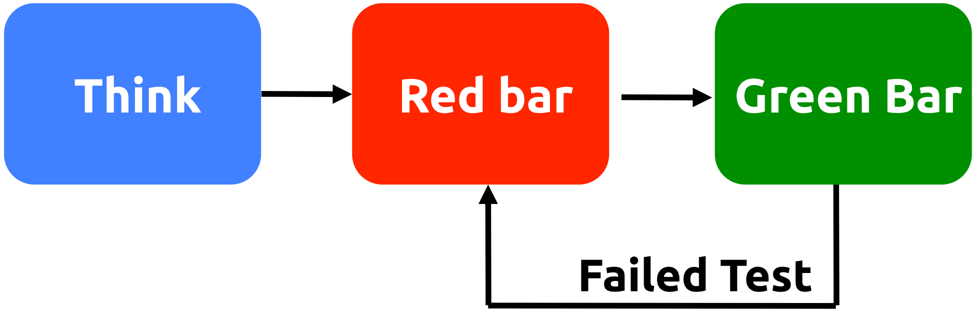
```
def testMatchesYearAndMonthAsWildCards(self):
    p = DatePattern(0, 0, 1)
    d = datetime.date(2004, 10, 1)
    self.failUnless(p.matches(d))
```

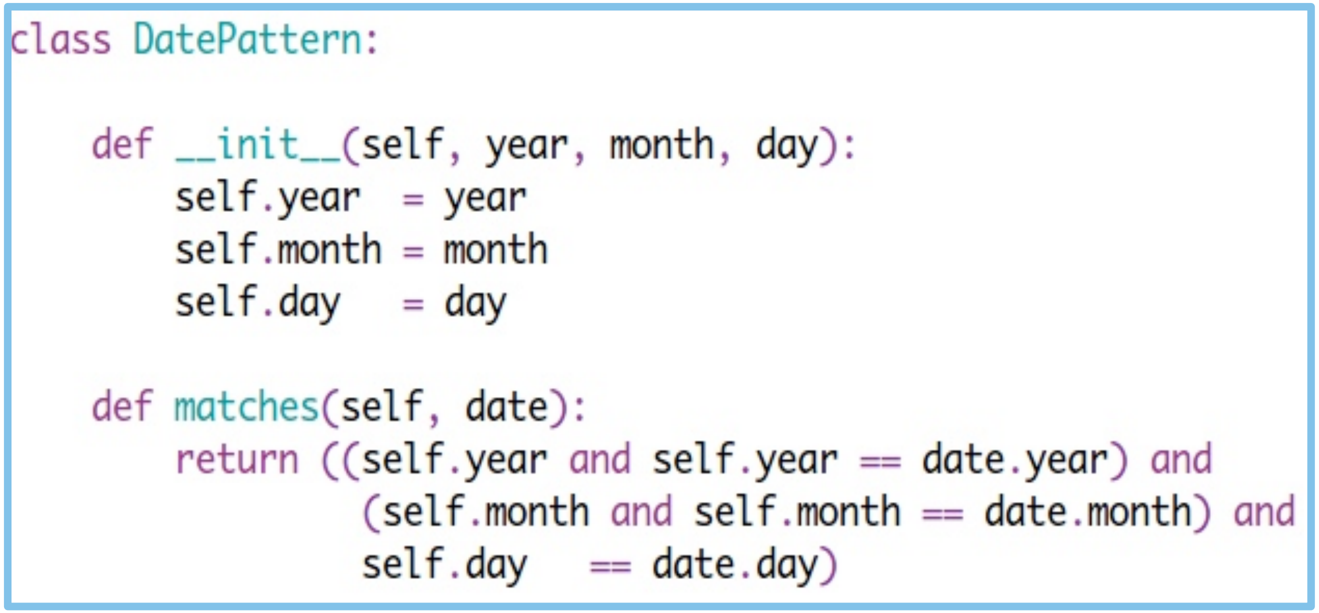
DENIED

```
===================================
ERROR testMatchesYearAsWildCard
--------------------------------------------
 [..]
ValueError: month is out of range
--------------------------------------------
Ran 4 tests in 0.000s
FAILED (errors=1)
```

TDD Mantra

Green Bar : Writing production code

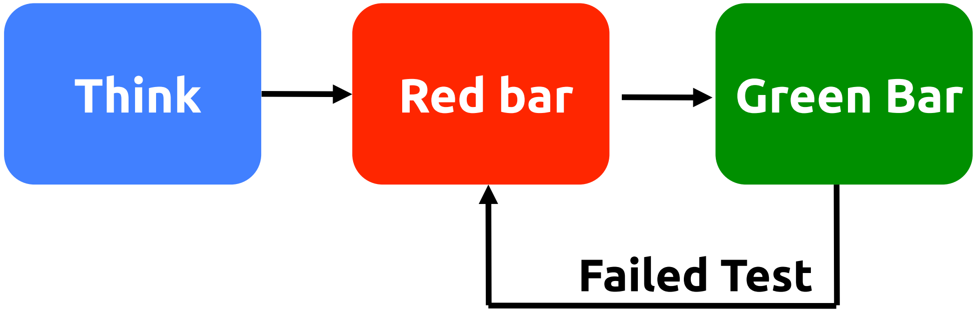


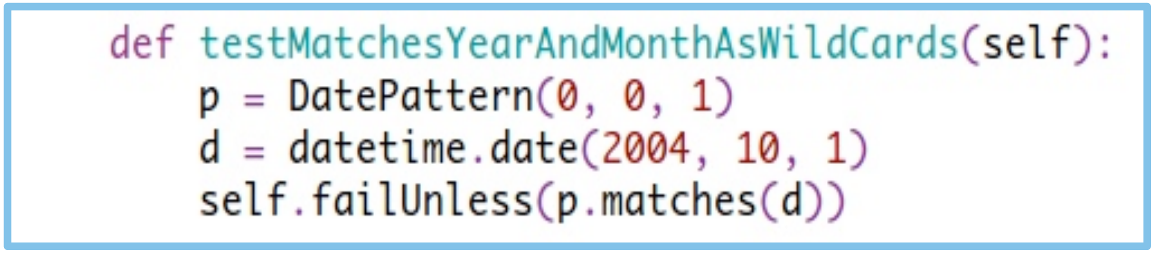


```
class DatePattern:

    def __init__(self, year, month, day):
        self.year  = year
        self.month = month
        self.day   = day

    def matches(self, date):
        return ((self.year and self.year == date.year) and
                (self.month and self.month == date.month) and
                self.day    == date.day)
```

TDD Mantra

Green Bar : Writing production code





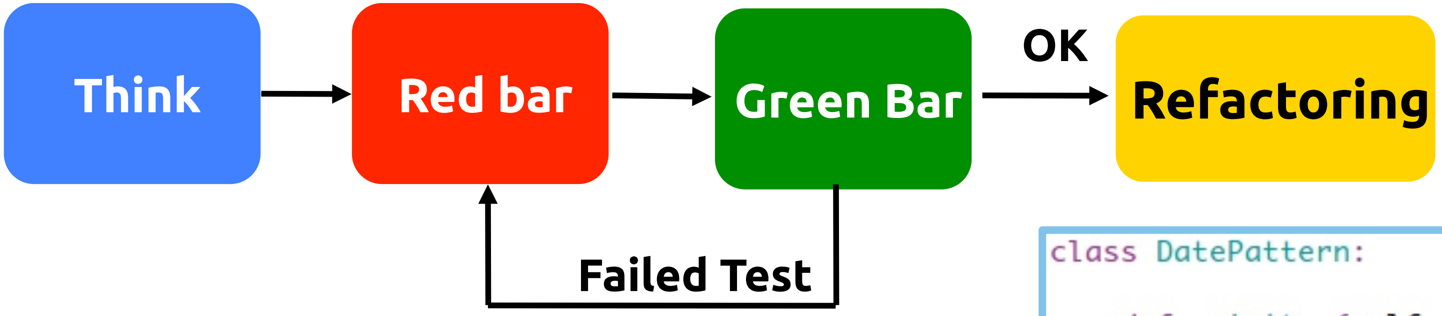
```
def testMatchesYearAndMonthAsWildCards(self):
    p = DatePattern(0, 0, 1)
    d = datetime.date(2004, 10, 1)
    self.failUnless(p.matches(d))
```

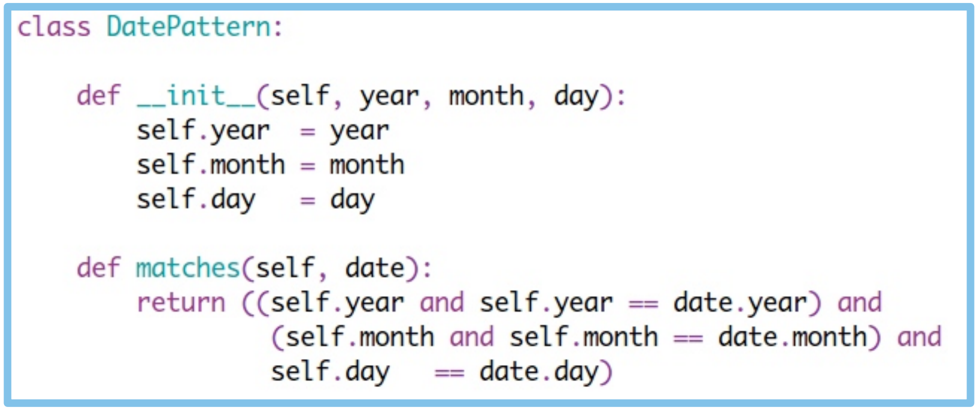
APPROVED

```
==========================
--------------------------
Ran 4 test in 0.000s

OK
```

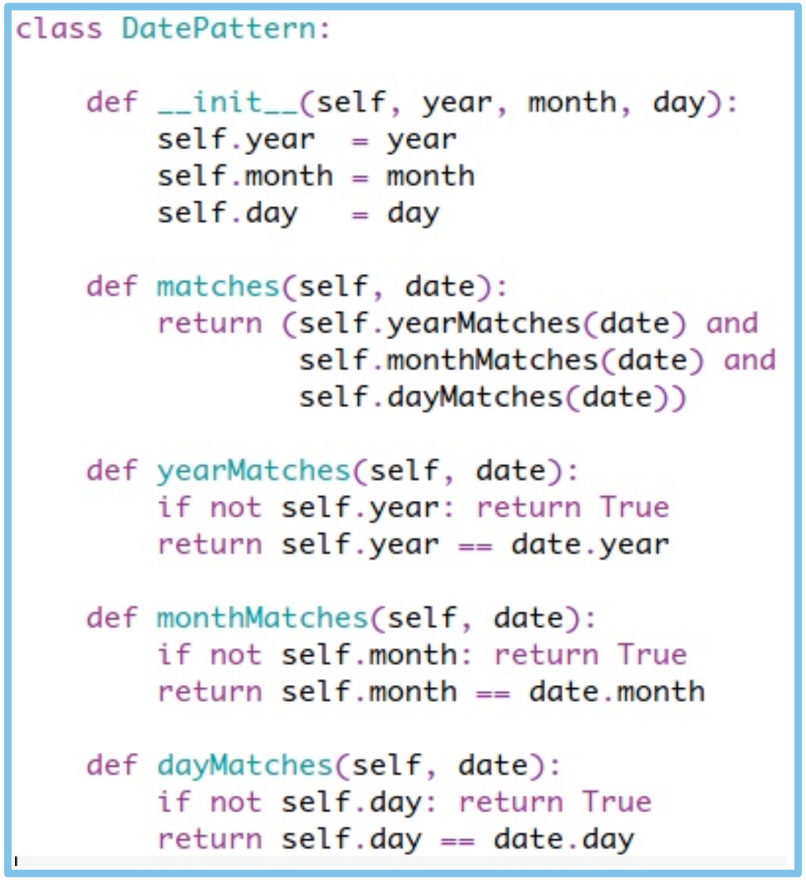
TDD Mantra

Refactoring: Simply and refactor production code





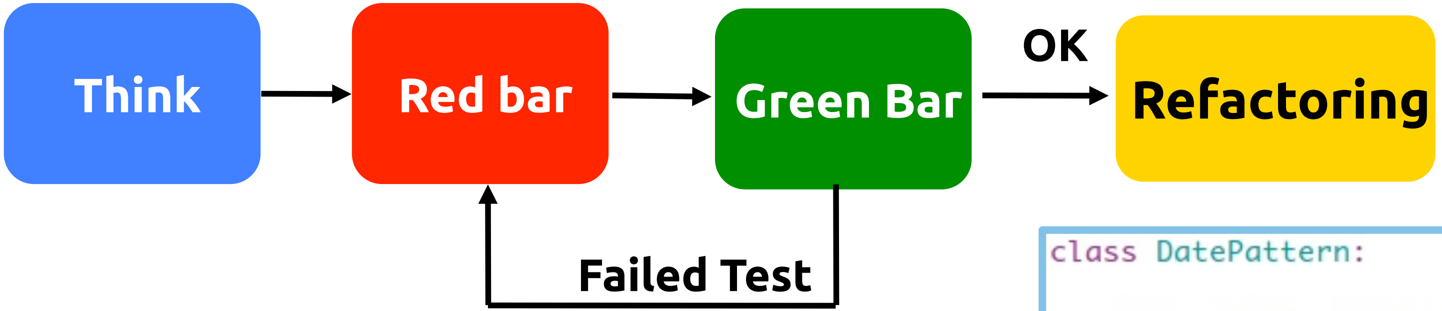
```
class DatePattern:

    def __init__(self, year, month, day):
        self.year  = year
        self.month = month
        self.day   = day

    def matches(self, date):
        return ((self.year and self.year == date.year) and
                (self.month and self.month == date.month) and
                self.day   == date.day)
```



```
class DatePattern:

    def __init__(self, year, month, day):
        self.year  = year
        self.month = month
        self.day   = day

    def matches(self, date):
        return (self.yearMatches(date) and
                self.monthMatches(date) and
                self.dayMatches(date))

    def yearMatches(self, date):
        if not self.year: return True
        return self.year == date.year

    def monthMatches(self, date):
        if not self.month: return True
        return self.month == date.month

    def dayMatches(self, date):
        if not self.day: return True
        return self.day == date.day
```

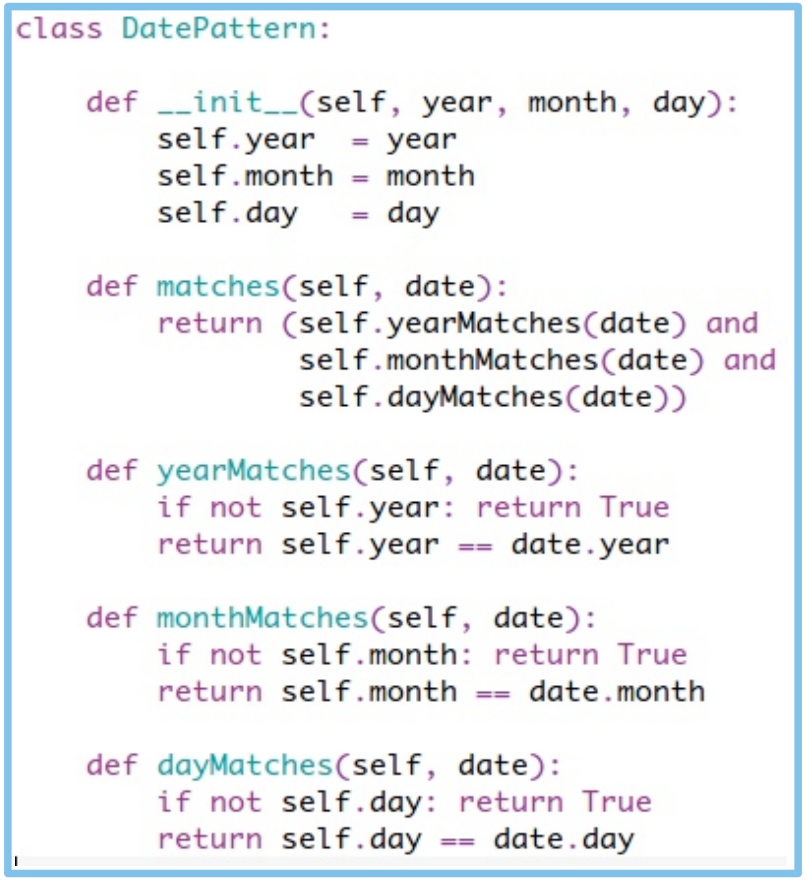
TDD Mantra

Refactoring: Simply and refactor production code



APPROVED

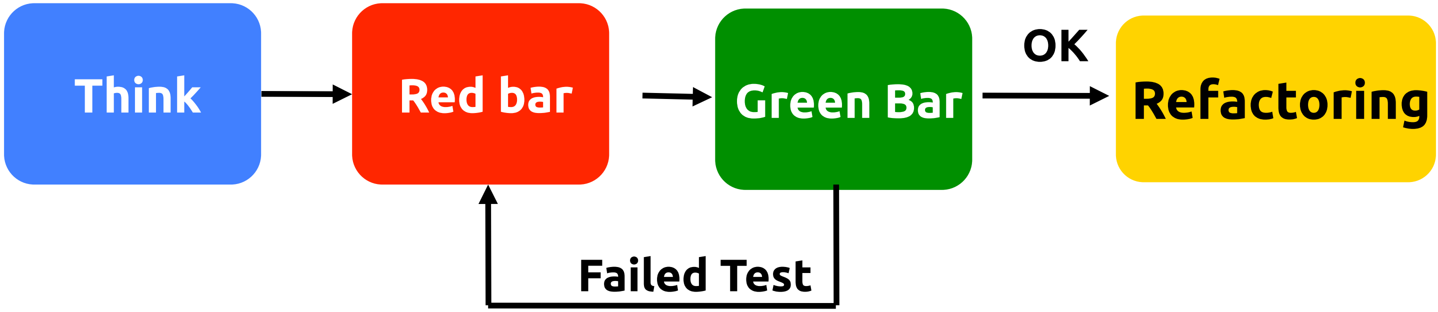
```
==========================
--------------------------
Ran 4 test in 0.000s

OK
```



```
class DatePattern:

    def __init__(self, year, month, day):
        self.year  = year
        self.month = month
        self.day   = day

    def matches(self, date):
        return (self.yearMatches(date) and
                self.monthMatches(date) and
                self.dayMatches(date))

    def yearMatches(self, date):
        if not self.year: return True
        return self.year == date.year

    def monthMatches(self, date):
        if not self.month: return True
        return self.month == date.month

    def dayMatches(self, date):
        if not self.day: return True
        return self.day == date.day
```

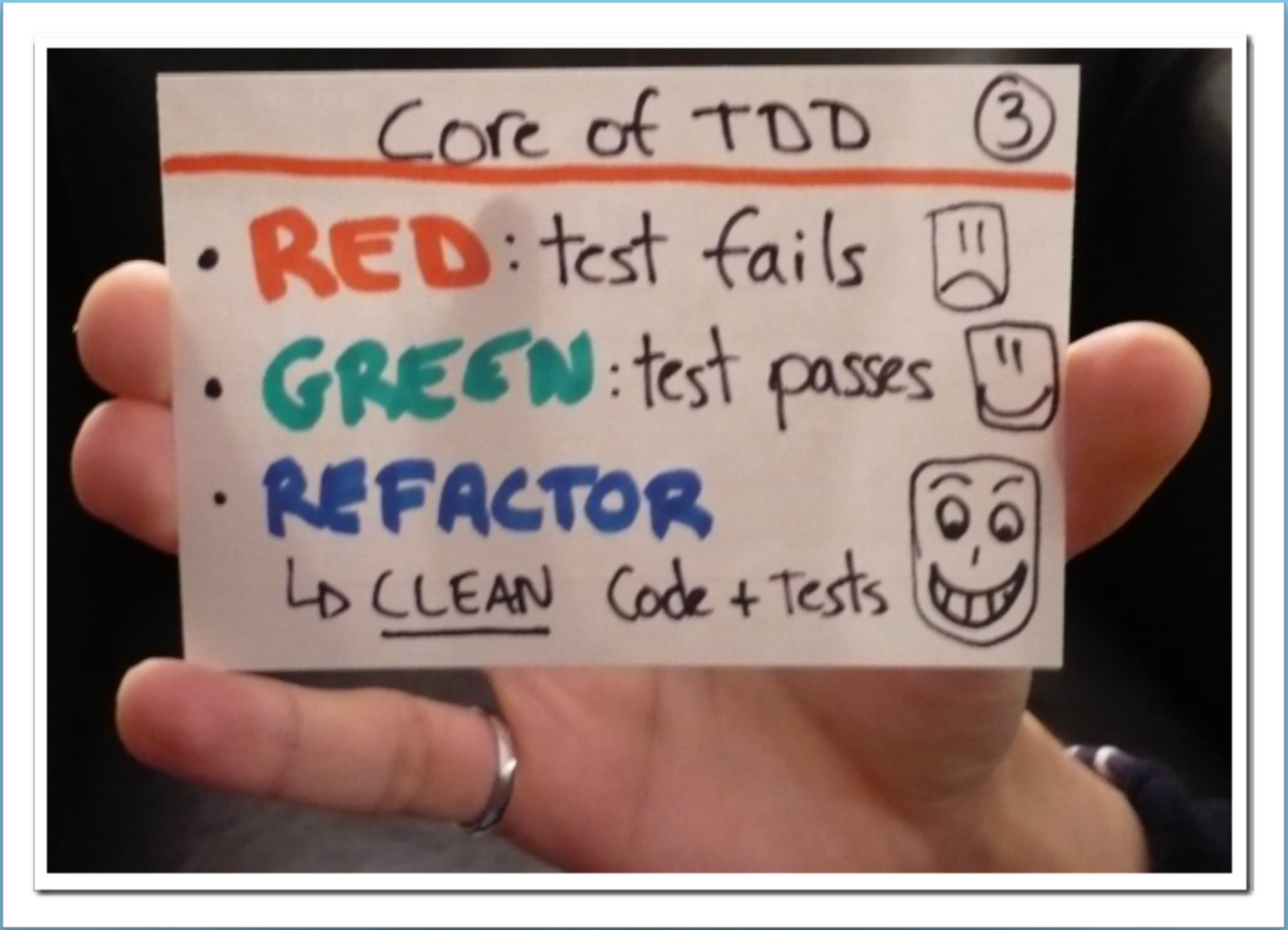
TDD Mantra

Principles



- Code once, test twice
- Clean code that works
- KISS: Keep It Short & Simple
- YAGNI: You Ain’t Gonna Need It
- DRY: Don't repeat yourself

5. TDD Patterns



Core of TDD ③

- RED: test fails
- GREEN: test passes
- REFACTOR
  ↳ CLEAN Code + Tests

TDD Patterns

Red Bar patterns:

- Begin with a simple test.
- If you have a new idea
  - add it to the test list
  - stay on what you're doing.
- Add a test for any faults found.
- If you can not go on **throw it all away and change it.**

TDD Patterns

Green Bar patterns:

- Writing the easier code to pass the test.
- Write the simpler implementation to pass current test
- If an operation has to work on collections
  - write the first implementation on a single object
  - then generalizes.

Tests for Documentation

- Test names describe features

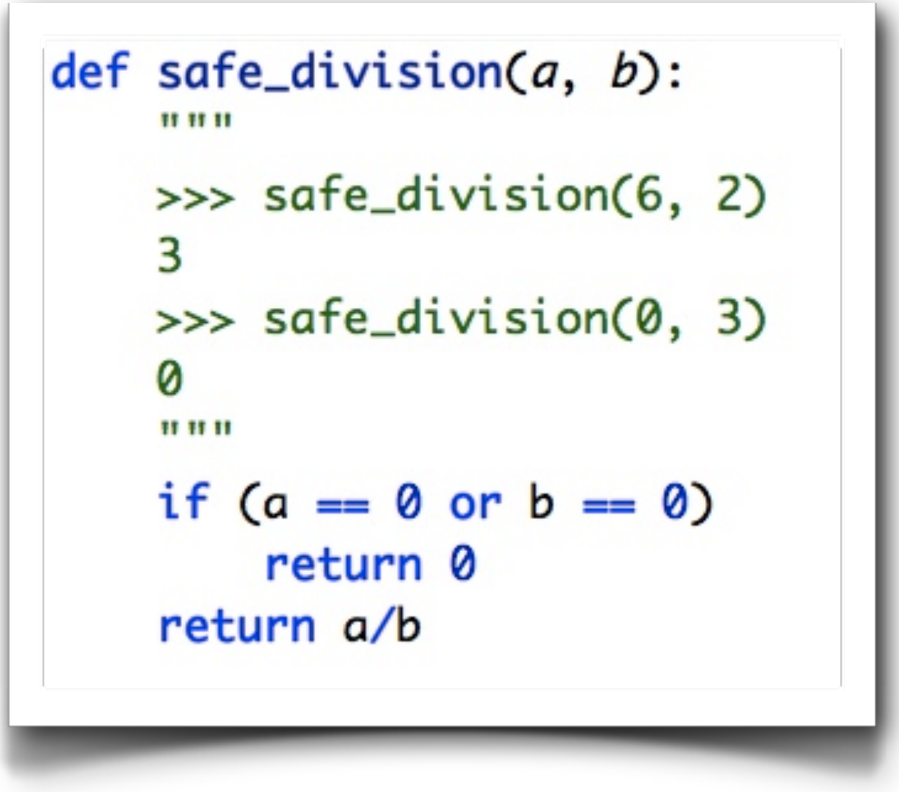
```
public class TargetObjectTest
{
    @Test public void test1() { [...]
    @Test public void test2() { [...]
    @Test public void test3() { [...]
}
```

```
public class TargetObjectTest
{
    @Test public boolean isReady() { [...]
    @Test public void choose(Picker picker) { [...]

}
```

doctest: Test through Documentation

- Lets you test your code by running examples embedded in the documentation and verifying that they produce the expected results.

- It works by parsing the help text to find examples, running them, then comparing the output text against the expected value.



```
def safe_division(a, b):
    """
    >>> safe_division(6, 2)
    3
    >>> safe_division(0, 3)
    0
    """
    if (a == 0 or b == 0)
        return 0
    return a/b
```

```
$ python -m doctest -v sample.py

Trying:
    my_function(6, 2)
Expecting:
    3
ok
Trying:
    my_function(0, 3)
Expecting:
    0
ok
1 items passed all tests:
   2 tests in
sample.safe_division
2 tests in 1 items.
2 passed and 0 failed.
Test passed.
```

8. Conclusions

AS YOU CAN CLEARLY SEE IN SLIDE 397...
scottadams@aol.com
www.dilbert.com
GAAAAH!
Z
Z
© 2000 United Feature Syndicate, Inc.
"POWERPOINT" POISONING.

Social Implications

- TDD handles "the *fears*" during software development
  - Allows programmers to perfectly know the code
  - New feature only if there are 100% of passed tests

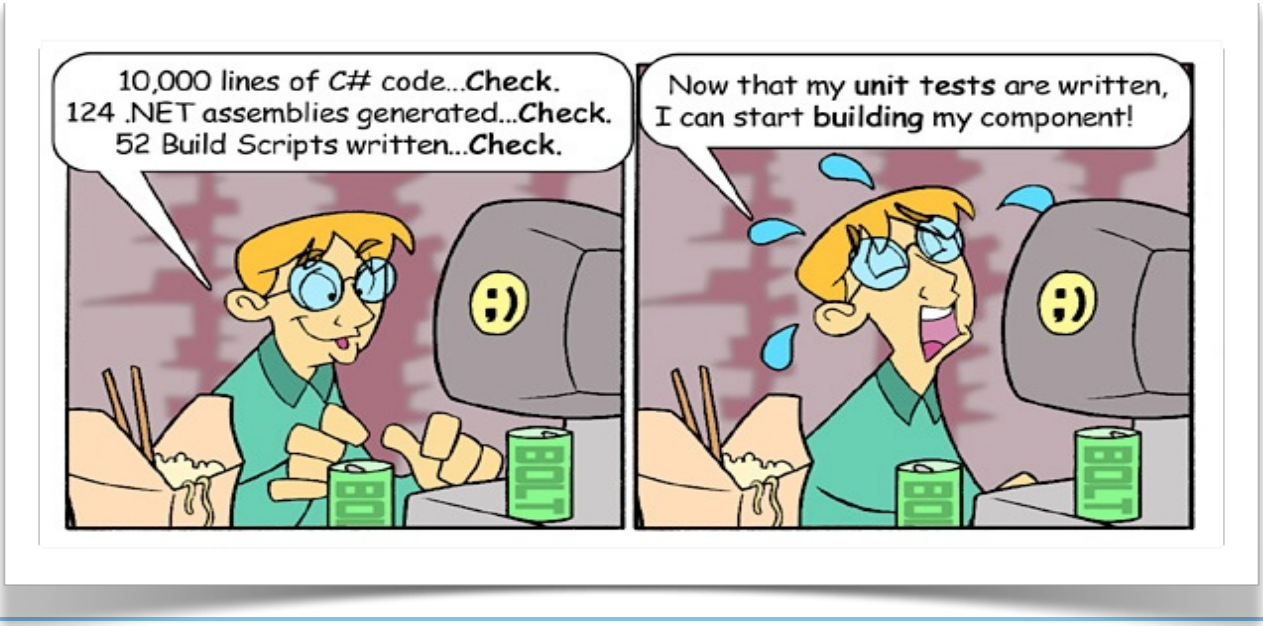
- Fears has a lot of negative aspects:
  - makes it uncertain
  - removes the desire to communicate
  - makes it wary of the feedback
  - makes nervous

TDD Benefits

- It keeps the code simple
  - Rapid development
- The tests are both design and documentation
  - Easy to understand code
- Bugs found early in development
  - Less debugging
- Low cost of change

TDD Limits

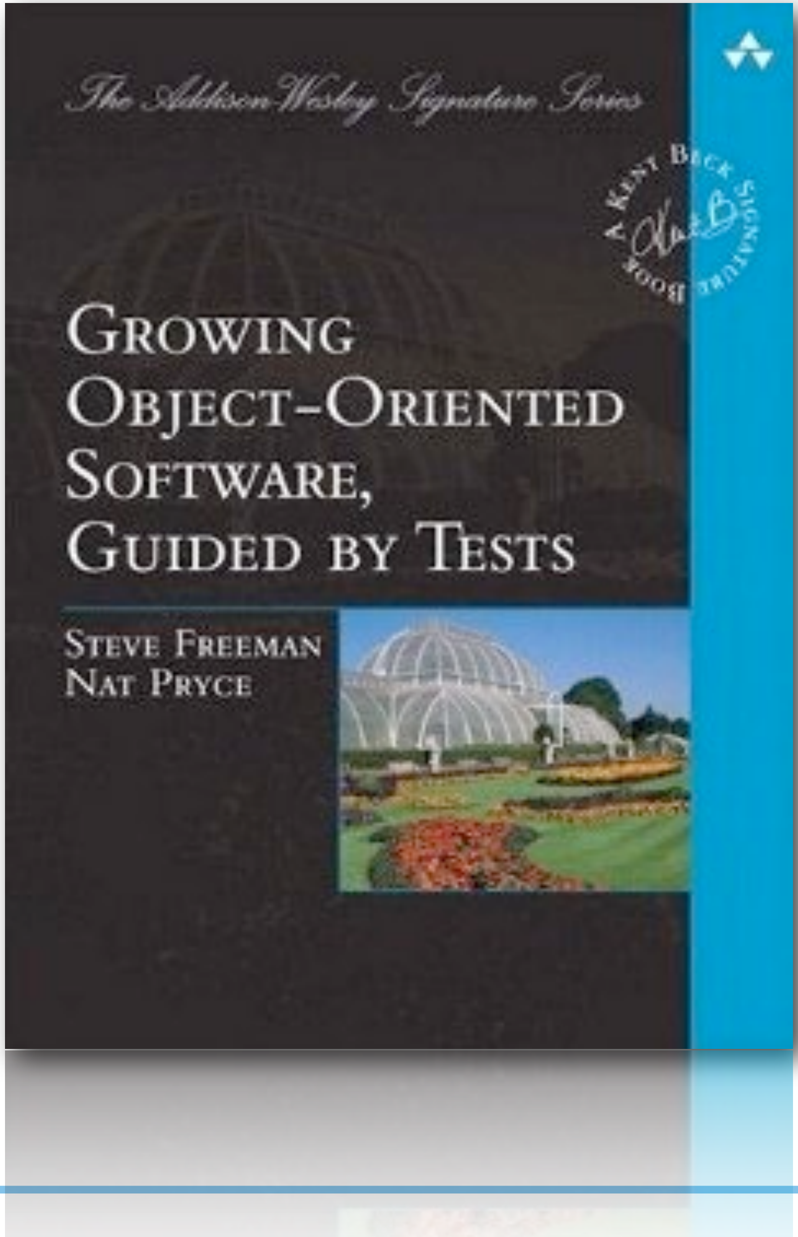
- High learning curve
- Managers are reluctant to apply
- Requires great discipline
- Difficult to implement the GUI
- Difficult to apply to Legacy Code


10,000 lines of C# code...Check.
124 .NET assemblies generated...Check.
52 Build Scripts written...Check.
Now that my unit tests are written, I can start building my component!
;)
BOLT

References

Growing Object-Oriented Software, Guided By Tests

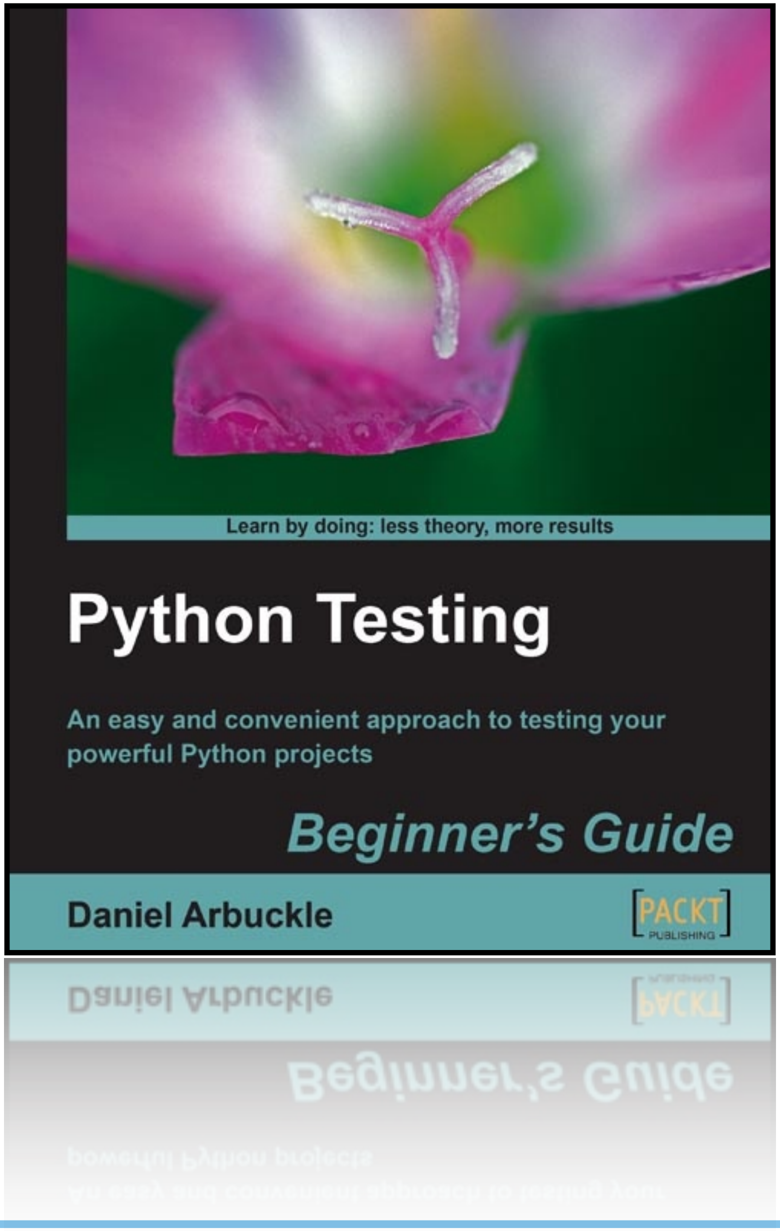
Freeman and Pryce, Addison Wesley 2010


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Growing Object-Oriented Software, Guided by Tests
Steve Freeman
Nat Pryce

References

Python Testing

Daniel Arbuckle, PACKT Publishing 2011


Learn by doing: less theory, more results
Python Testing
An easy and convenient approach to testing your powerful Python projects
Beginner's Guide
Daniel Arbuckle
PACKT PUBLISHING