Coding Dojo: Test Driven Development

道場

A practical guide to creating a space where good programmers can become great programmers.

Source for this presentation

This presentation are de-facto notes from the Pluralsight.com course:

Coding Dojo: Test Driven Development

Emily Bache

https://www.pluralsight.com/courses/the-coding-dojo

Overview

What is a Coding Dojo?

Learning Test Driven Development

Collaborative Games for Programmers

A Sample Series of Dojo meetings

Organizing and Facilitating

Tool for the Coding Dojo

What is Coding Dojo

A Dojo is a hall or space for immersive learning or meditation. This is traditionally in the field of martial arts, but has been seen increasingly in other fields, such software development.
The term literally means "place of the Way" in Japanese.

(wikipedia)

Dojo is a japan word.

Dojo is place where people are practicing martial arts or other craftsmanship.

It’s place where we are meeting.

How do you learn in a Coding Dojo

Dojo Principles

Principles are like values, it‘s what informs the way you behave.

- **We value code with tests**
- **Collaborative learning environment**
- **Interactive, collaborative and fun learning experience**

- **http://bossavit.com/dojo/archives/2005_02.html**

Dojo Principles

The first rule of the dojo:

- **You can‘t discuss a technique without code**
- **You can‘t show code without tests**
- **Code without tests simply doesn‘t exist!**

Dojo Principles

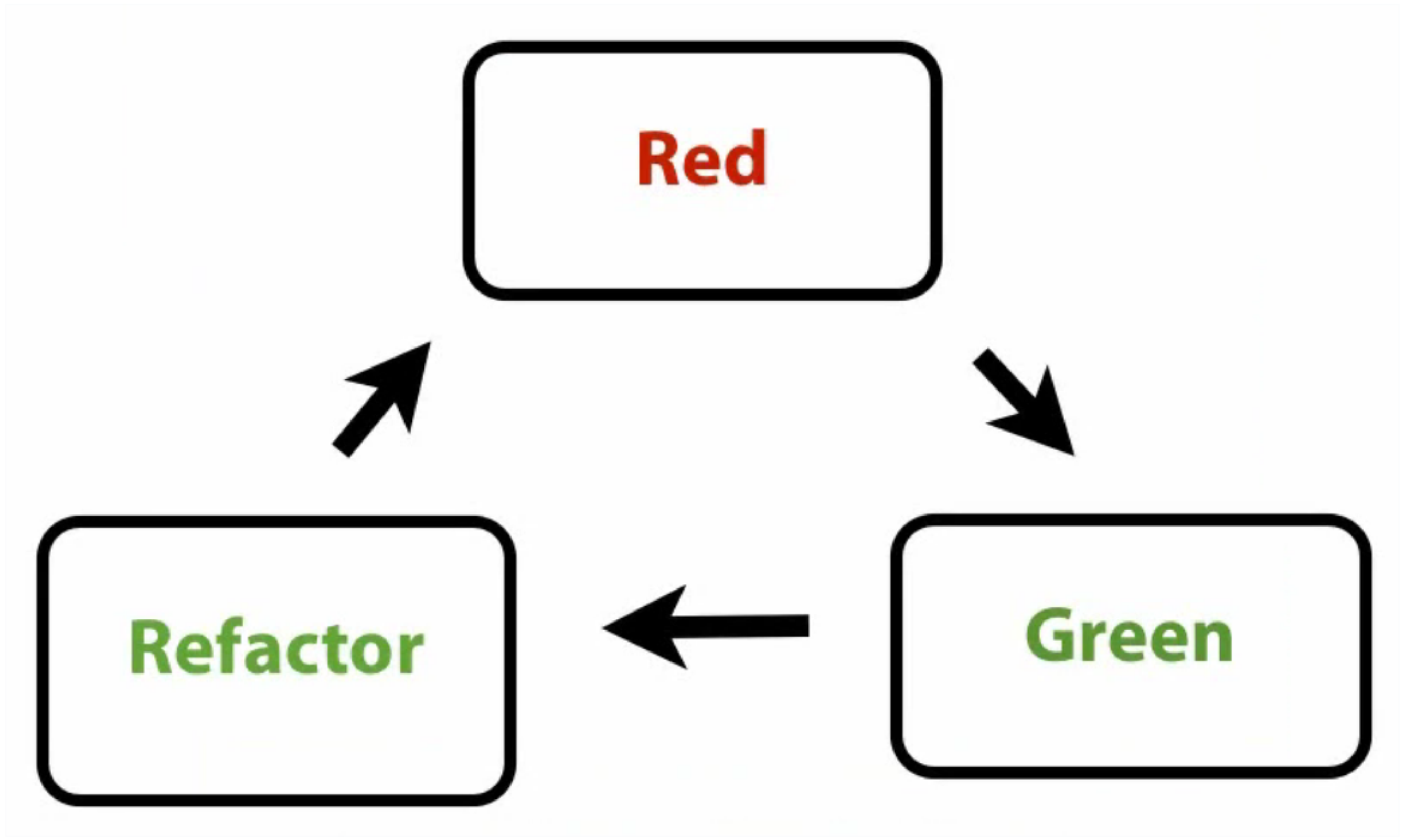
- If it seems hard, find someone who can explain it
- If it seems easy, explain to those who find it hard
- Everyone will both teach & learn at different times

Practical Coding Skills

- **Using IDE and keyboard shortcuts**
- **Pair Programming**
- **TDD**
- **Refactoring**
- **Designing good Test Cases**
- **Working incrementally**
- **Design using SOLID principles**
- **Object Oriented Paradigm**
- **Functional Programming Paradigm**

Is there anything on this list that you‘d like to be better at?

Test Driven Development



How people become experts

Incidental Practice:

- Repeatedly doing something you can already do
- It becomes kind of habit

Deliberate Practice

- Trying to do something you can‘t comfortably do
- Breaking down a skill into components you practice separately

Deliberate and incidental practice
Dr. Erikson

Deliberate Practice

Need to feel safe

If the cost of failure is too high, like in your production code, you won‘t take the risk

Need to feel motivated

Good Habits

„I‘m not a great programmer; I‘m just a good programmer with great habits.“

- Kent Beck

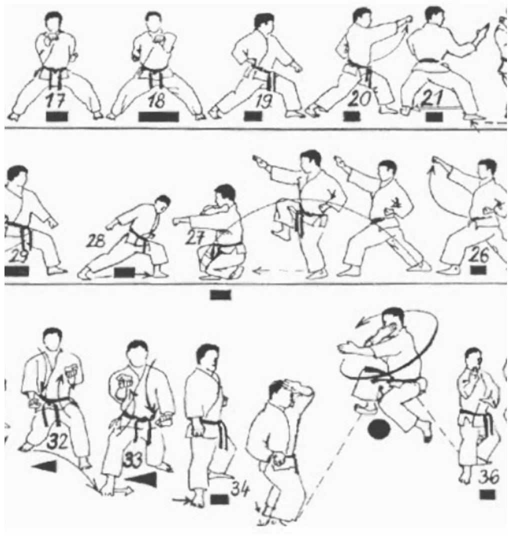
Your habits are what you do when you‘re not really thinking. They‘re what you continue to do when you‘re felling stressed,when there‘s a deadline, when you‘re tired. If you have good habits, you‘ll continue to write tests, make design improvements, continue to write great code.

Code Kata

Kata is a Japan word and it means a form. For example sequence of moves.

Dave Thomas proposed the idea of the „Code Kata“

http://codekata.pragprog.com/



Kata is a symbol of principles.

Code Kata – Leap Years

Write a function that returns true or false depending on whether it input integer is leap your or not.

A leap year is divisible by 4, but is not otherwise divisible by 100, unless it is also divisible by 400.

Examples:

1996 -> true

2001 -> false

2000 -> true

1900 -> false

The point is that you can
practice the way you solve it and
you should use test your own development.

Teaching and Learning TDD

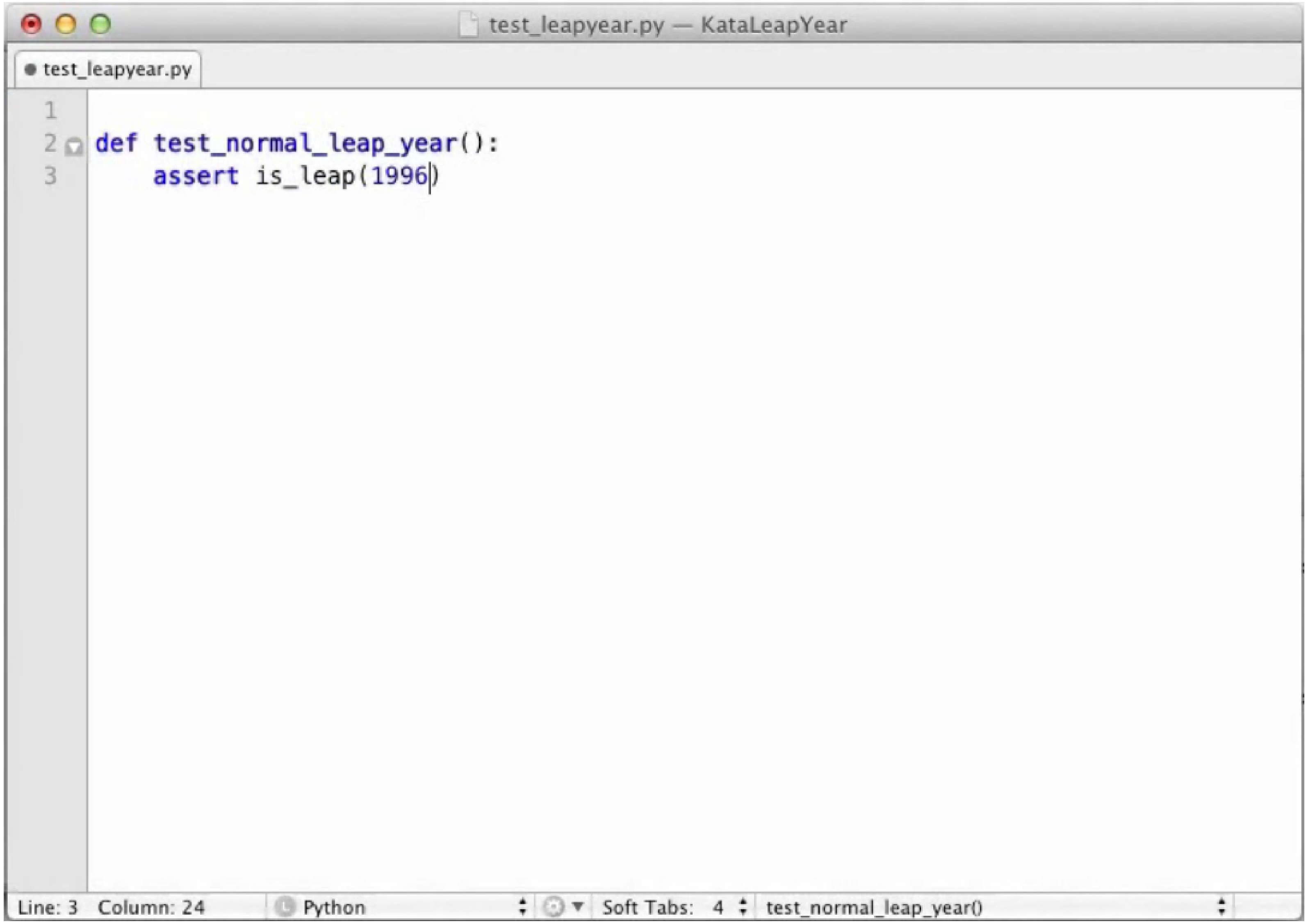
Overview – Analyze Problems, Test List, Guiding Test

Red – Declare & Name Arrange-Act-Assert Satisfy compiler

Green – Implement solution Fake it Start over

Refactor – Remove Fake, Remove Code Smell (No new functionality), Note new test cases

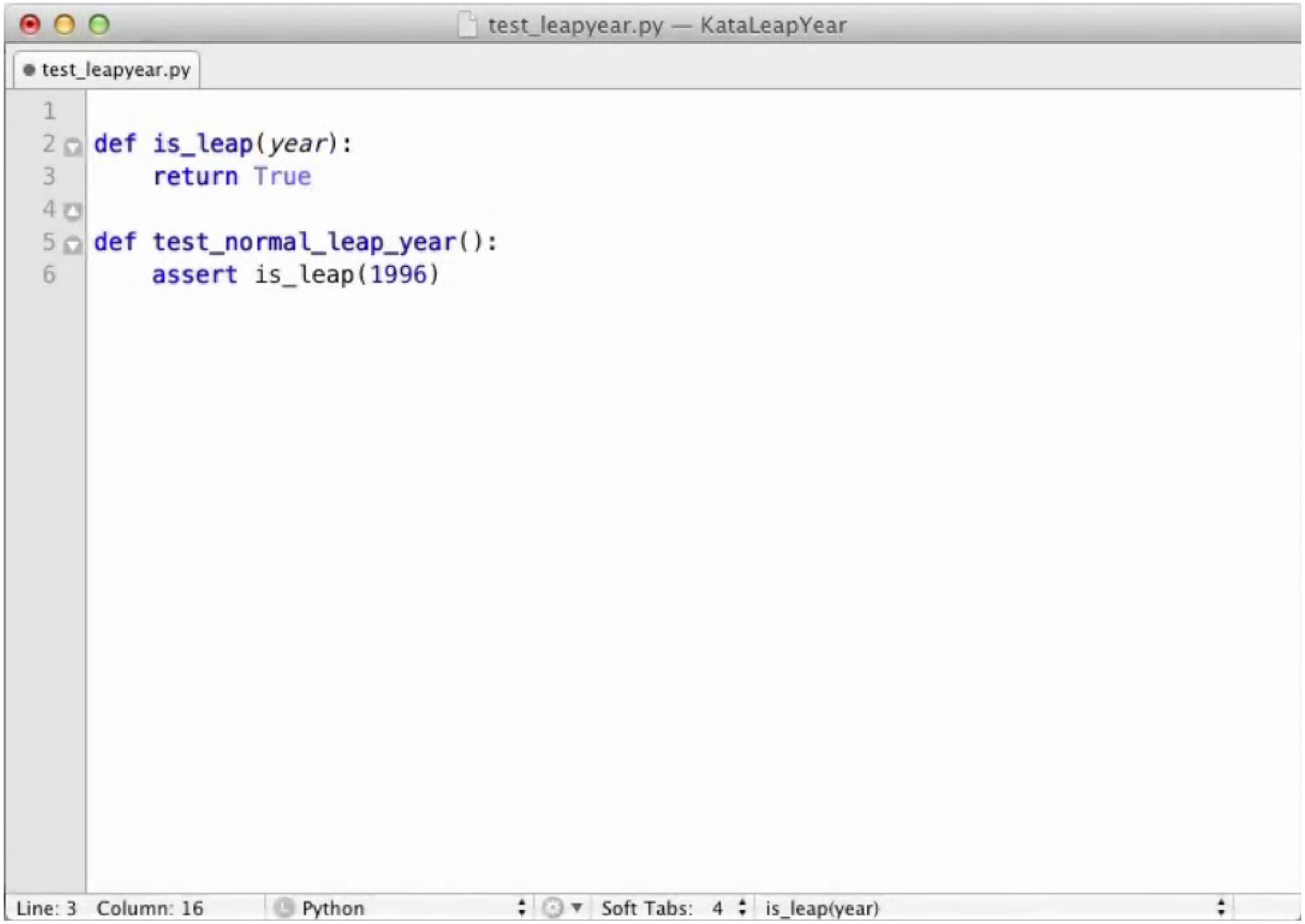
Fake it strategy


test_leapyear.py — KataLeapYear

```python

def test_normal_leap_year():
    assert is_leap(1996)
```

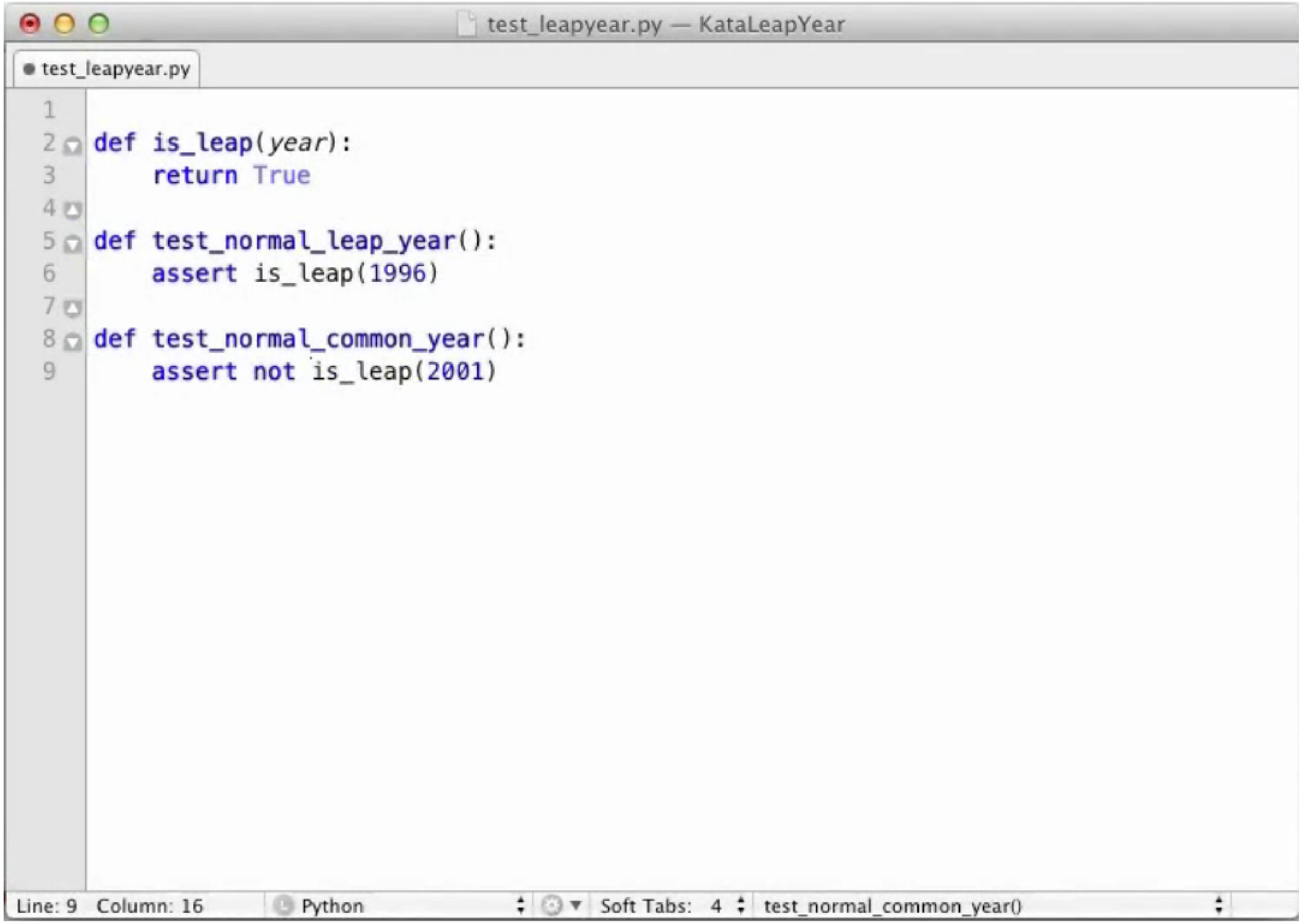
Line: 3 Column: 24 Python Soft Tabs: 4 test_normal_leap_year()

Fake it strategy



```python

def is_leap(year):
    return True

def test_normal_leap_year():
    assert is_leap(1996)
```

Fake it strategy



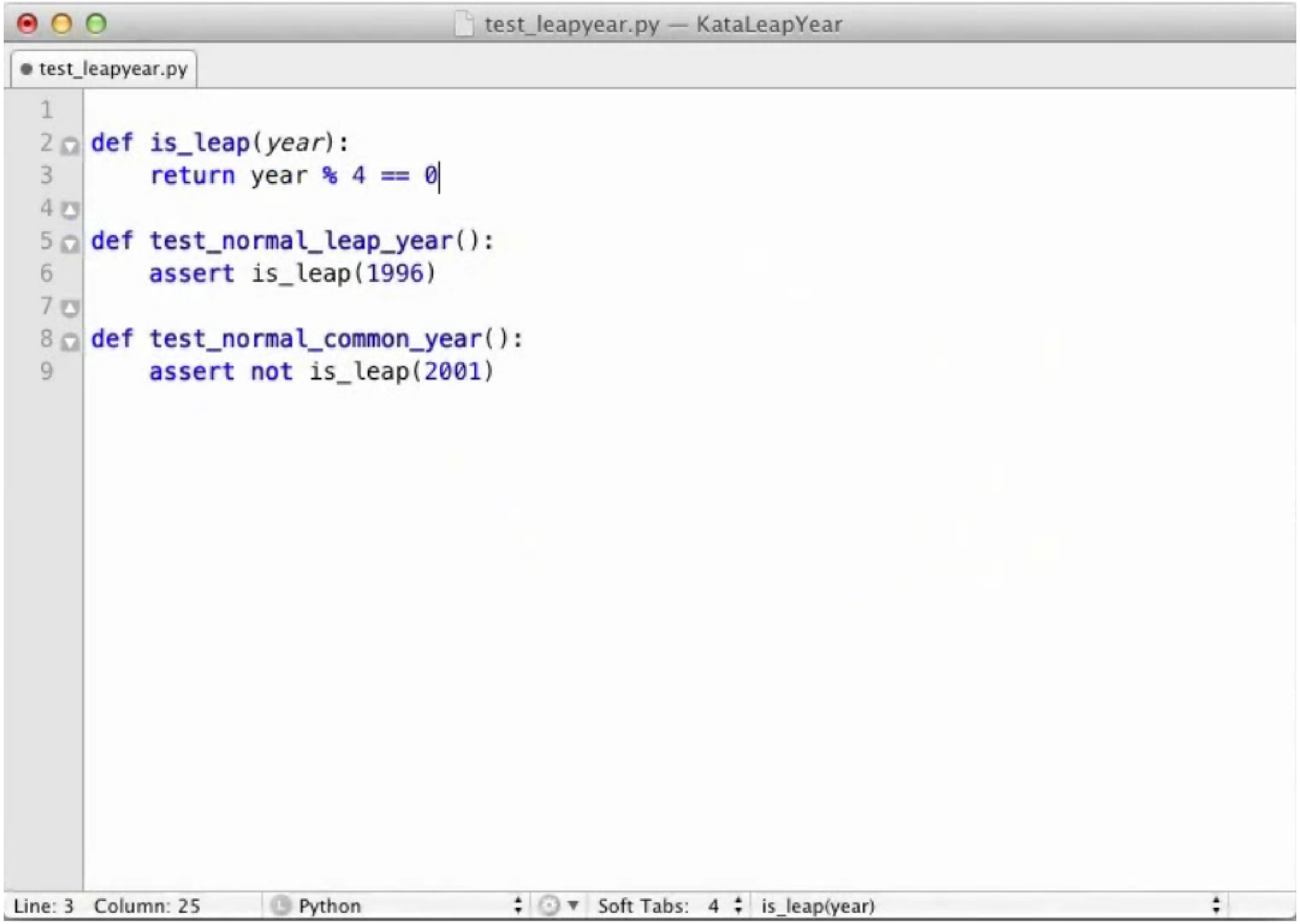
test_leapyear.py — KataLeapYear

test_leapyear.py

```

def is_leap(year):
    return True

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)
```

Line: 9 Column: 16 Python Soft Tabs: 4 test_normal_common_year()

Fake it strategy



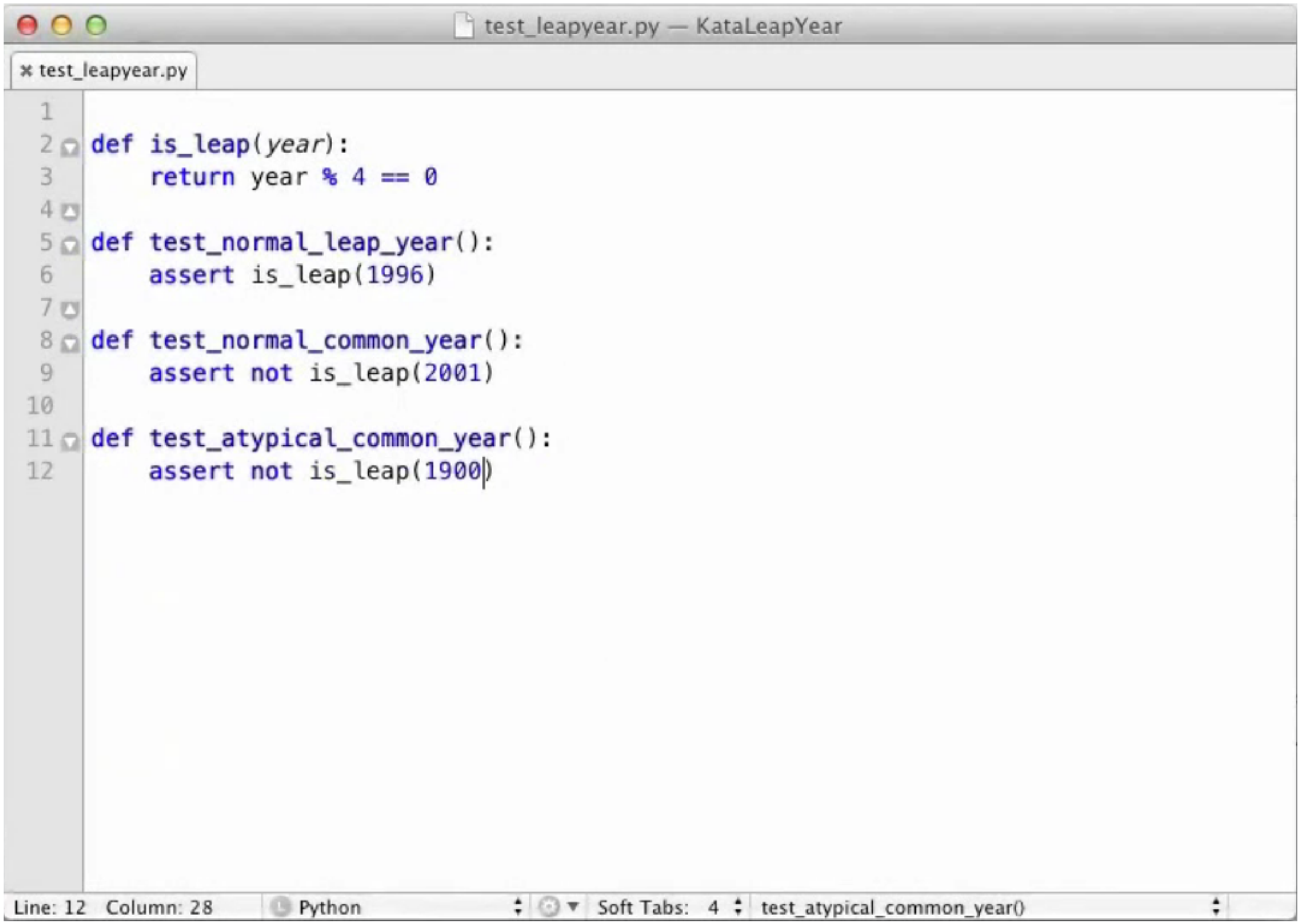
test_leapyear.py — KataLeapYear

● test_leapyear.py

```

def is_leap(year):
    return year % 4 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)
```

Line: 3 Column: 25 Python Soft Tabs: 4 is_leap(year)

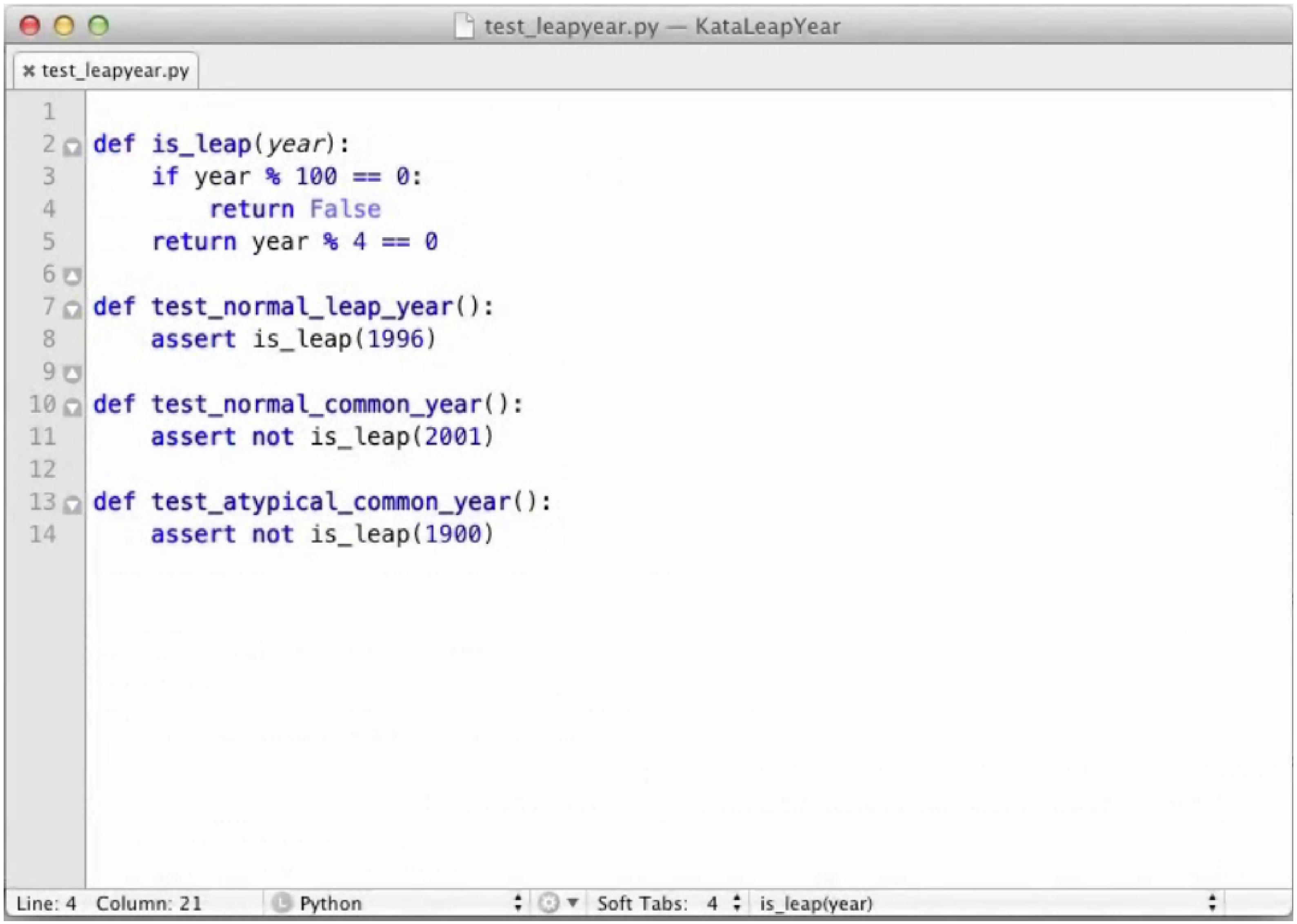
Fake it strategy


test_leapyear.py — KataLeapYear

```python

def is_leap(year):
    return year % 4 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)
```

Line: 12 Column: 28 Python Soft Tabs: 4 test_atypical_common_year()

Fake it strategy

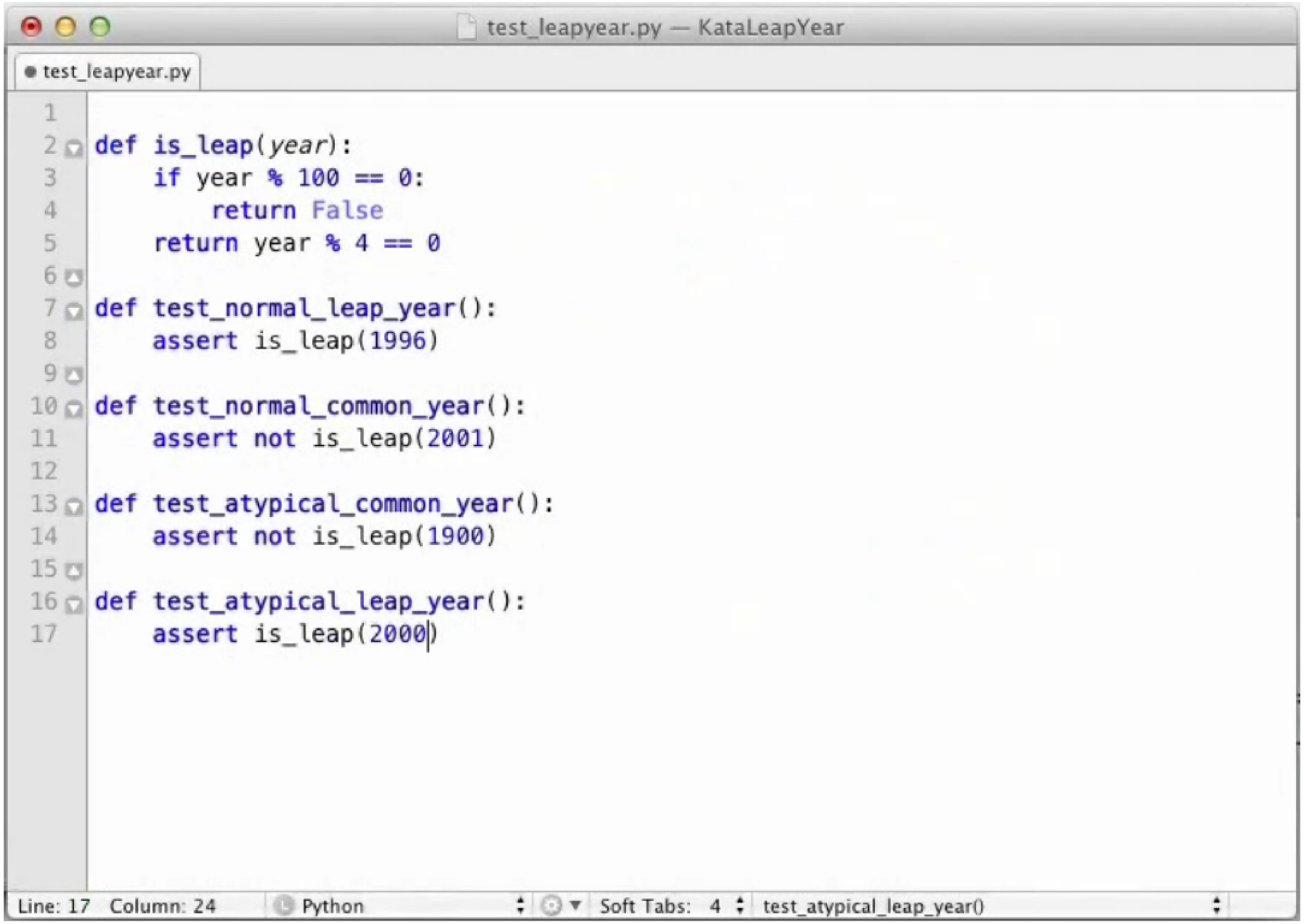


test_leapyear.py — KataLeapYear

```python

def is_leap(year):
    if year % 100 == 0:
        return False
    return year % 4 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)
```

Line: 4 Column: 21 Python Soft Tabs: 4 is_leap(year)

Fake it strategy

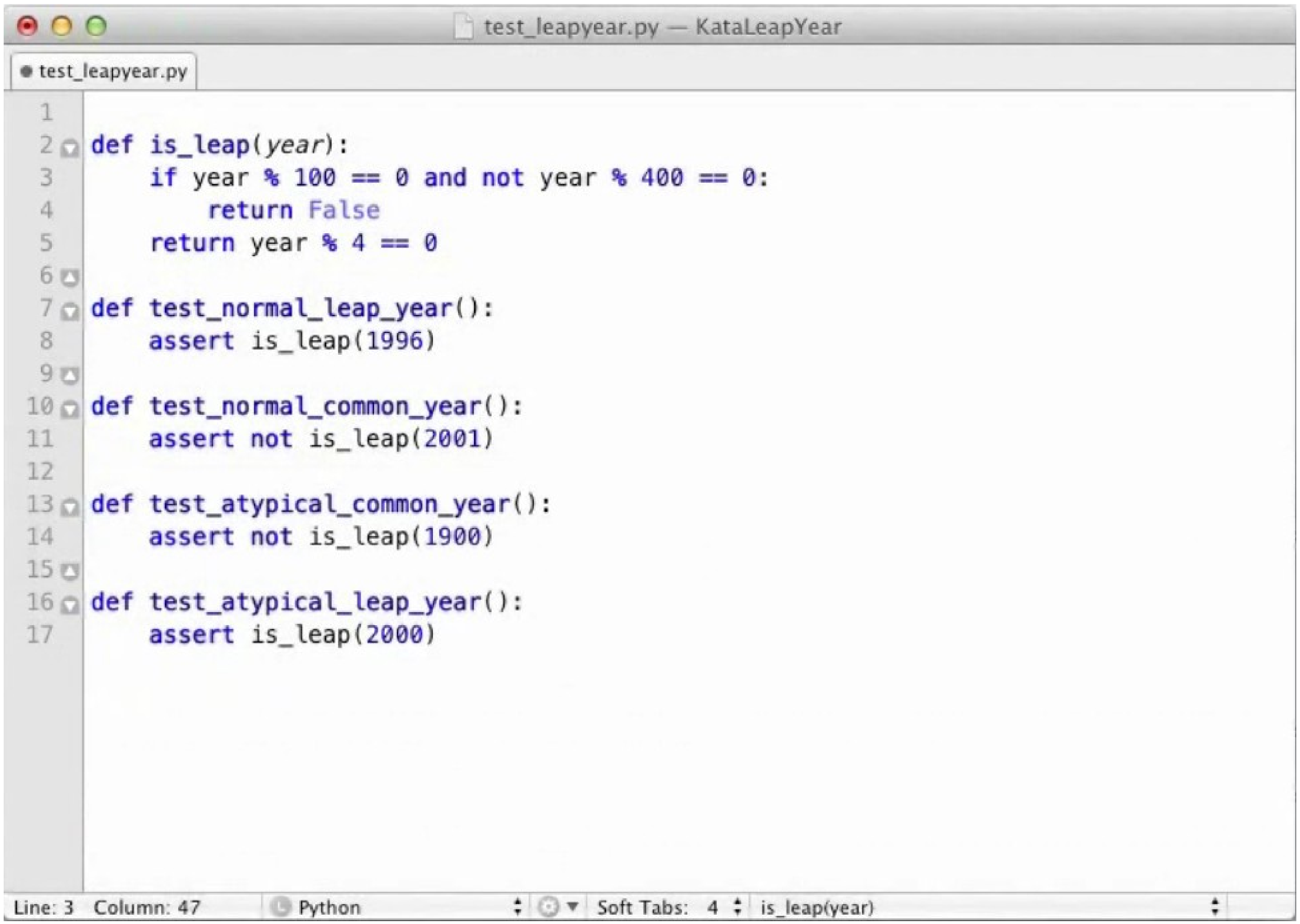

test_leapyear.py — KataLeapYear

test_leapyear.py

```

def is_leap(year):
    if year % 100 == 0:
        return False
    return year % 4 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)

def test_atypical_leap_year():
    assert is_leap(2000)
```

Line: 17 Column: 24 | Python | Soft Tabs: 4 | test_atypical_leap_year()

Fake it strategy

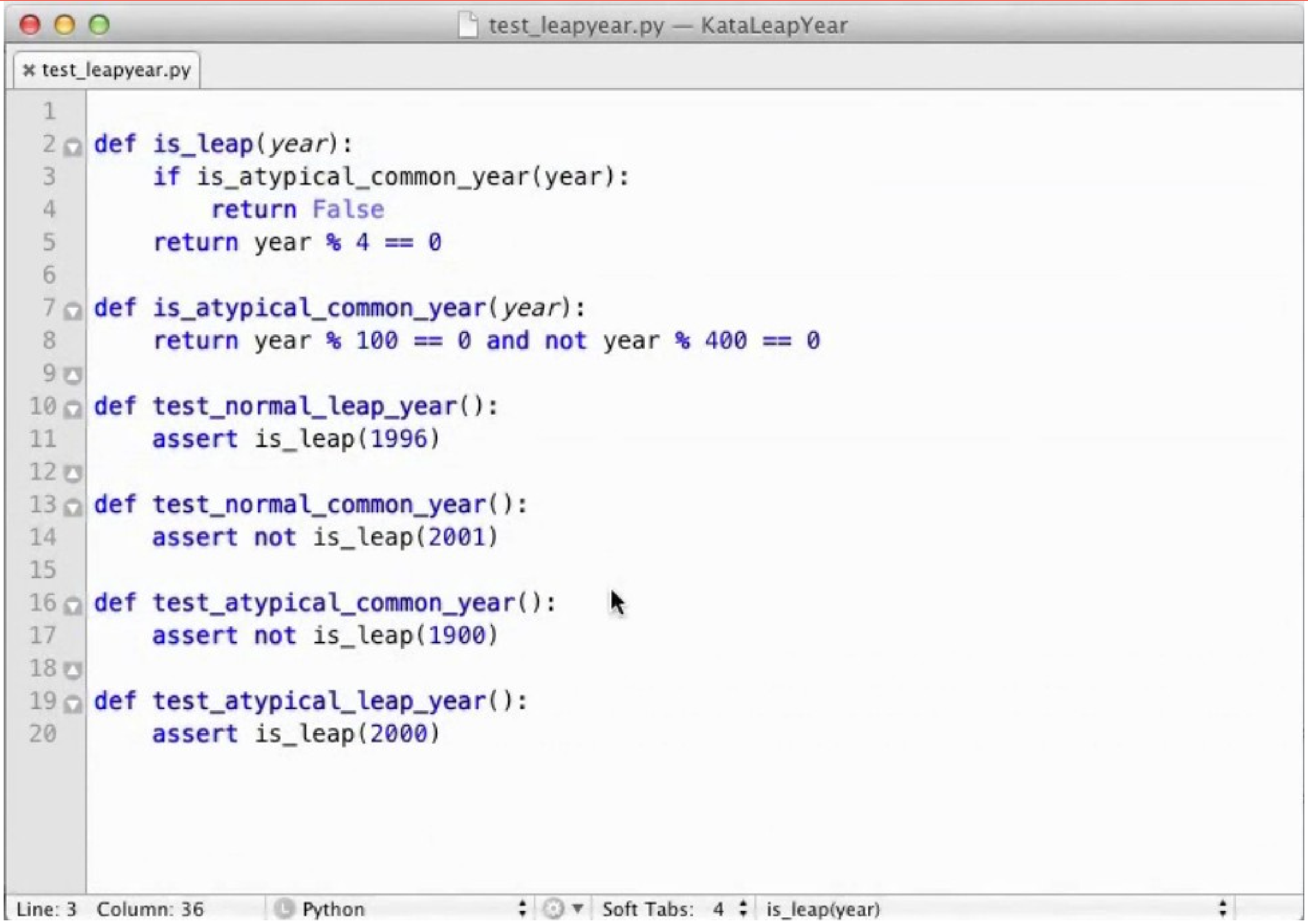

test_leapyear.py — KataLeapYear

test_leapyear.py

```

def is_leap(year):
    if year % 100 == 0 and not year % 400 == 0:
        return False
    return year % 4 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)

def test_atypical_leap_year():
    assert is_leap(2000)
```

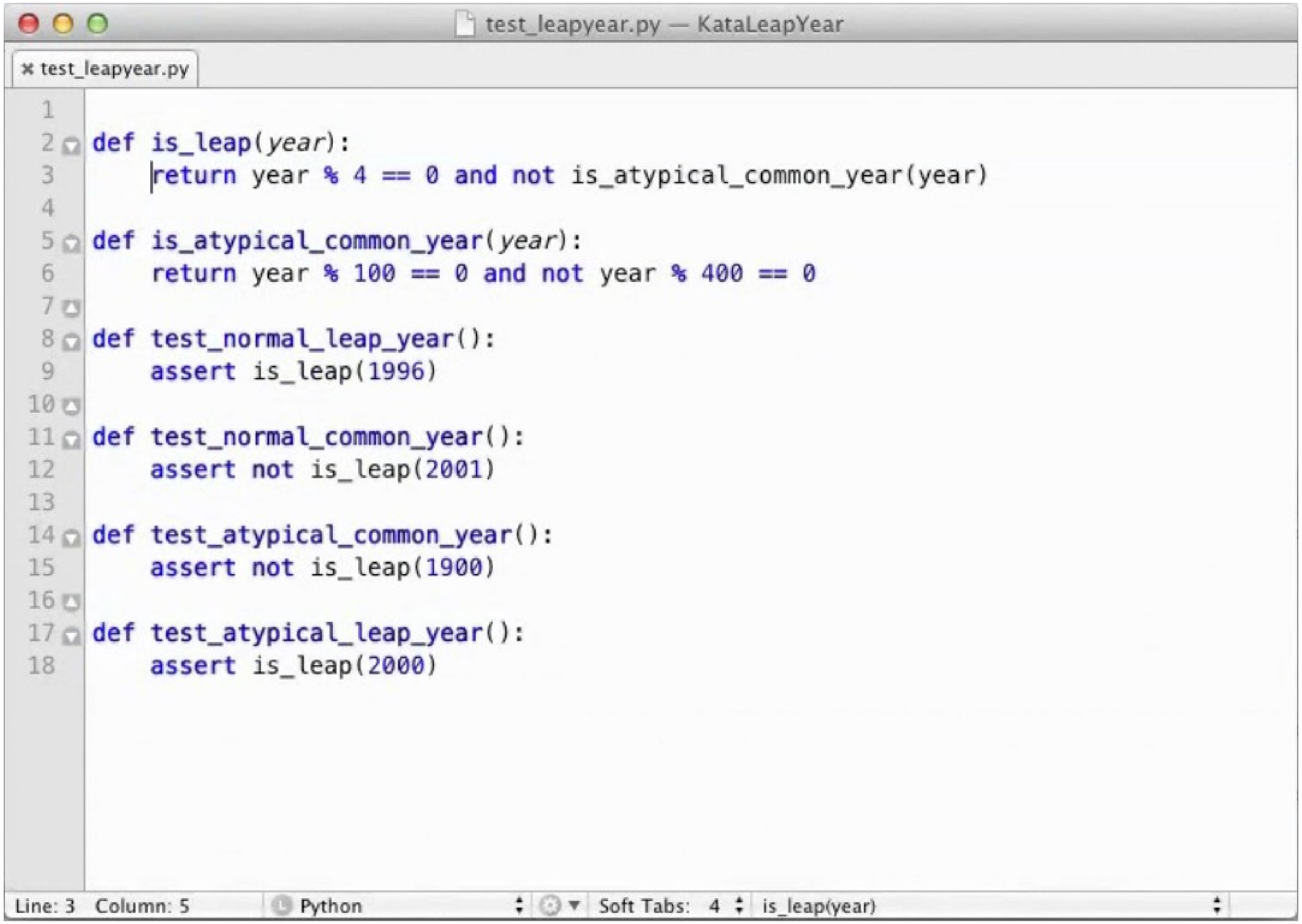
Line: 3 Column: 47 Python Soft Tabs: 4 is_leap(year)

Fake it strategy



```
def is_leap(year):
    if is_atypical_common_year(year):
        return False
    return year % 4 == 0

def is_atypical_common_year(year):
    return year % 100 == 0 and not year % 400 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)

def test_atypical_leap_year():
    assert is_leap(2000)
```

Fake it strategy

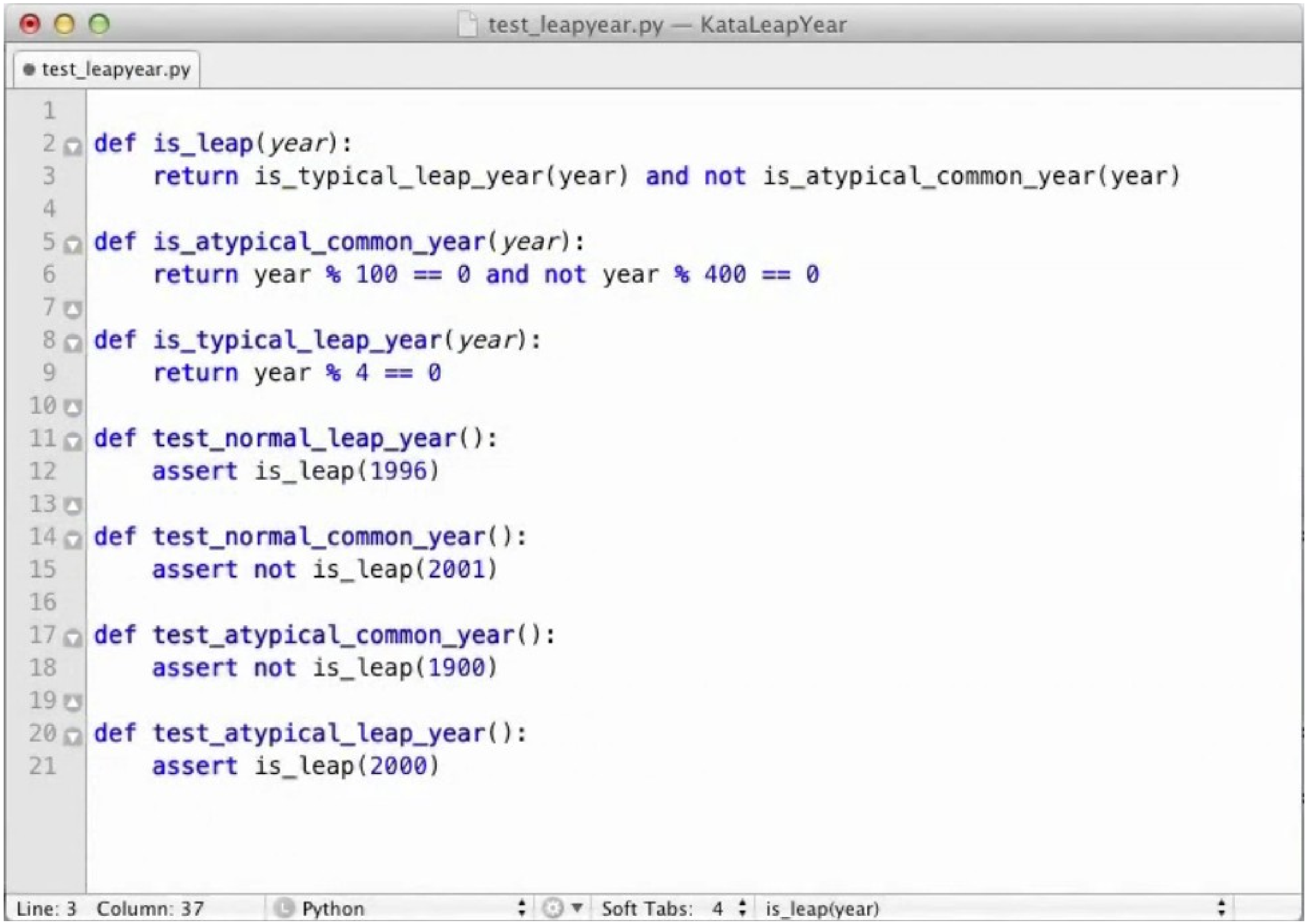

test_leapyear.py — KataLeapYear

test_leapyear.py

```
def is_leap(year):
    return year % 4 == 0 and not is_atypical_common_year(year)

def is_atypical_common_year(year):
    return year % 100 == 0 and not year % 400 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)

def test_atypical_leap_year():
    assert is_leap(2000)
```

Line: 3 Column: 5 Python Soft Tabs: 4 is_leap(year)

Fake it strategy



test_leapyear.py — KataLeapYear

test_leapyear.py

```python

def is_leap(year):
    return is_typical_leap_year(year) and not is_atypical_common_year(year)

def is_atypical_common_year(year):
    return year % 100 == 0 and not year % 400 == 0

def is_typical_leap_year(year):
    return year % 4 == 0

def test_normal_leap_year():
    assert is_leap(1996)

def test_normal_common_year():
    assert not is_leap(2001)

def test_atypical_common_year():
    assert not is_leap(1900)

def test_atypical_leap_year():
    assert is_leap(2000)
```

Line: 3 Column: 37 Python Soft Tabs: 4 is_leap(year)

Component Skill for TDD

Designing Test Cases

Designing Clean Code

Driving Development with Tests

Refactoring Safely

Summary

TDD is like a series of moves from one state to another.

Red, green, refactor and over again..

You can demonstrate TDD on a simple code kata

TDD has component skill you can practice separately

Collaborative Games for Programmers

Prepared Kata

Randori

Randori in Pairs

Constraint Games

You aim to beat the game itself, not the other players. Players help each other – they collaborate. All the players in the game are helping one another, collaborating to beat the game.

There are rules, activities, and goals which you‘re trying to achieve.

Why not Competitive Games?

Learning happens more easily when you feel safe and relaxed.

Prepared Kata

King of collaborative game

Show you best solution to a Code Kata

You show all the steps that have to solve the Kata right foot empty editor until you have a working solution.

While you are coding you explain what you are doing and why, and people in the audience can ask questions and give you feedback on what they think about the code you‘ve written.

Everyone is learning – both presenter & audience. Presenter is getting feedback and audience seeing practice solution.

After the presentation, the hope is that everyone in the room should be able to go away and do the kata again by themselves. And preferably the do it better than you did during the meeting.

Prepared Kata – Tips

Practice many times before performance

Keep it short! (15 minutes)

Find a pair

Explain each coding decision

Expect that other and better solutions exist

No need to follow advice at the time, note it for the retrospective.

Randori

Another term from Karate – a free form interaction

Everyone in the dojo contributes some code

Show the code on a projector

60 – 90 minutes.

Take turns with the keyboard

- Time Limit (5 or 7 minutes)
- Ping Pong (2 or 3 people, up to 10)

Randori Rules

If you have the keyboard, you decide what to type. It is your decision. Everyone else might have opinions, but it‘s your opinion that counts.

If you are asked for help, kindly respond. But don‘t swamp them with conflicting advice.

If you are not asked, but see an opportunity for improvement, choose a wise moment to speak, not just blurt it out straight away. It‘s not usually a wise moment to speak, when somebody‘s in the middle of trying to make some tests pass. The best opportunity for re-factoring and improving the design is when all the tests are passing. So try and save your comments until then.

Randori in Pairs

Split the group into pairs (or trios)

Each pair works on the same Kata

A facilitator goes between pairs, helping them

In the retrospective, share code and discuss how you wrote it

Swap pairs and repeat the exercise from the beginning

Whole-day event „Code Retreat“ uses this format

One of the dojo principles says,
that you should show your working, and not just the final code.

Code Retreat

Whole Day event, 5 or 6 coding sessions

All the elements of a Coding Dojo are there

Repeat the same Code Kata in every session

The Game of Life Kata is a good one for practicing test-driven development. As you repeat it over the day, the actual problem just starts to fade into the background and you can concentrate your brain on how you‘re coding. What tests you‘re writing, if the code smells, the steps of test-driven development?

Constraint Games

The idea is that once you‘ve got to know a Code Kata, your brain is no longer occupied with just solving the problem, and you can start to concentrate more on how you‘re coding.

Force yourself to code differently and test your limits by: Tool Constraints, Design Constraints and Social Constraints.

The hope is, that this practice will help you when you face tricky situations in your production code.

Tool Constraint Games

With a tool constraint, you deliberately restrict the way you use your tools.

Keyboard only – no mouse

Use a plain text editor (no IDE)

It really forces you to remember how your programming language actually works.

By forcing everyone to use a plain text editor, you bring everyone down to the lowest common denominator and that can really help with the pair programming.

Design Constraint Games

You constrain the design of your code

Really small methods

We all know, that long methods, is a code smell. So what happens, if we artificially restrict ourselves, to really, really small methods, max 2 lines in method body.

Ban conditionals

If, else --> polymorphism

Take away all if and else statements. And to do that, you have to find other ways to control the flow of execution in your program. For example, with polymorphism where you use subclasses and overriding methods.

Design Constraint Games

No loops

For, while, foreach --> map, filter, recursion

Without constructs like for, and while, and for each, you have to turn to maybe more functional style constructs, like map, reduce, filter, and use recursion.

Social Constraint Games

Ho you work together in your group or your pair

If you have a whole group working on a Randorian paths to facilitate can announce in the middle:

Collective green deadline! I want to see all the tests passing in all the paths simultaneously. Your collective green deadline is in two minutes , and I set the timer.

At this point, anyone who's tests are currently failing, will find out if they've made a lot of changes since their last green line and how hard it is to get back to everything working and the tests being green. They may need to revert the code.

Even people who currently have all the tests passing can be caught out, thought. It can be very tempting to think, well, I could just make this small re-factoring. And still be in time for the deadline. And then be really embarrassed when the timer goes off, the two minutes is up and you're still in the middle of the re-factoring and you've made a mistake and your test are failing.

Discussion about the size of steps you should take when doing test-drive development.

Social Constraint Games

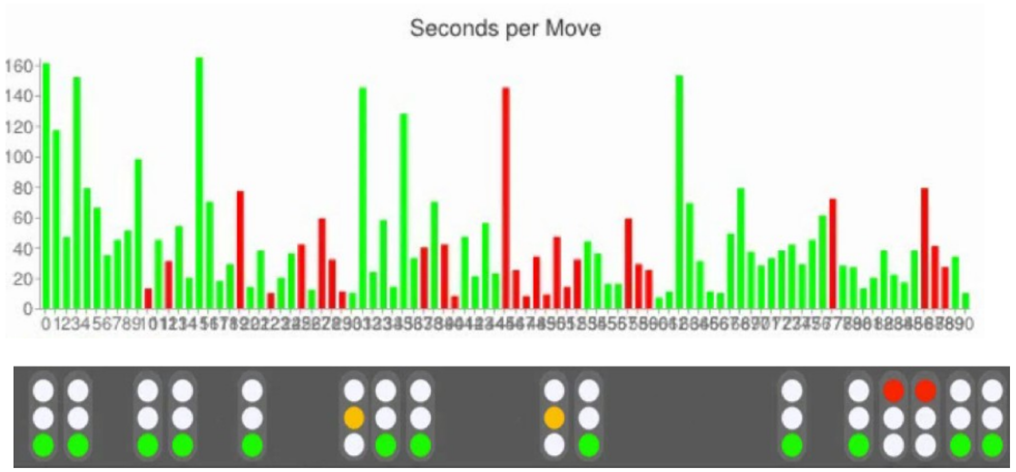
Ping-Pong Pair Programming

No talking, only allow yourselves to communicate via typing source code, and tests into editor.

Do the absolute minimum to get the test passing, so if your pair writes the failing test, you write as little code as possible. Perhaps even deliberately misinterpret what they mean. And have code something just to get the test to pass.

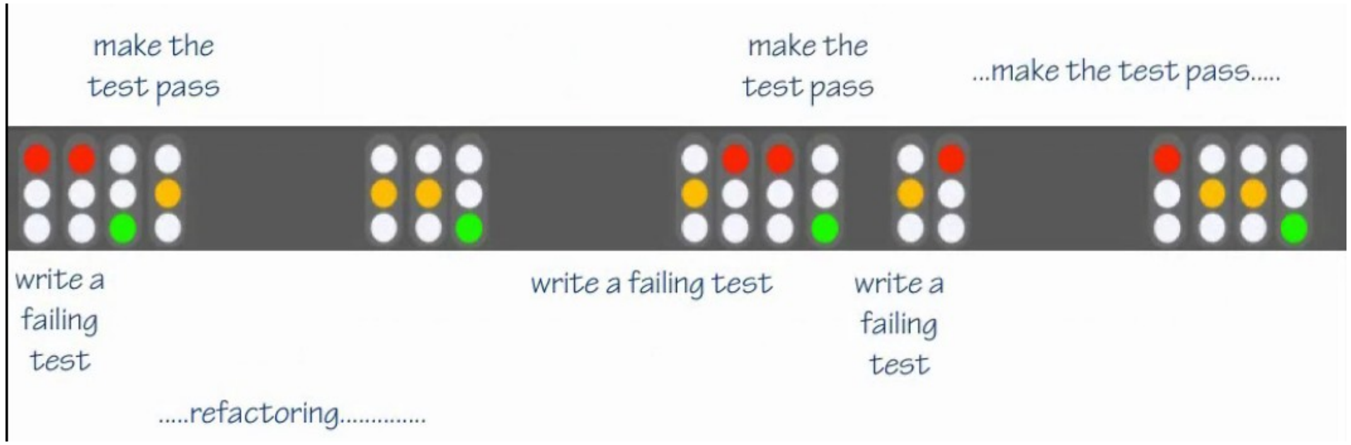
Tools

http://codersdojo.org

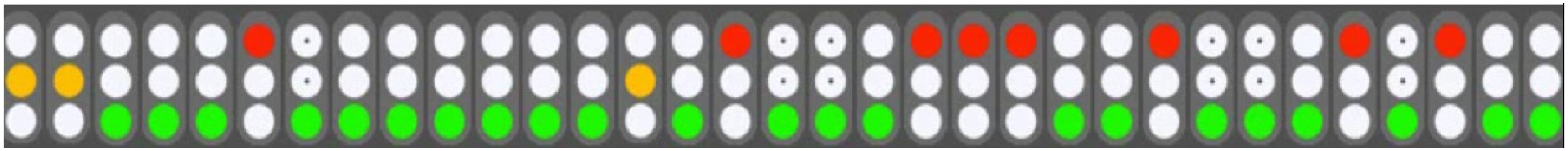


http://cyber-dojo.com

(John Jagger)



Revert to Last Green



Links

- https://www.pluralsight.com/courses/the-coding-dojo
- http://codersdojo.org
- http://cyber-dojo.com
- http://codekata.pragprog.com/
- http://bossavit.com/dojo/archives/2005_02.html

- http://codingdojo.org/
- https://github.com/dojo-brno/dojo-brno
- http://www.juanlopes.net/dojotimer/