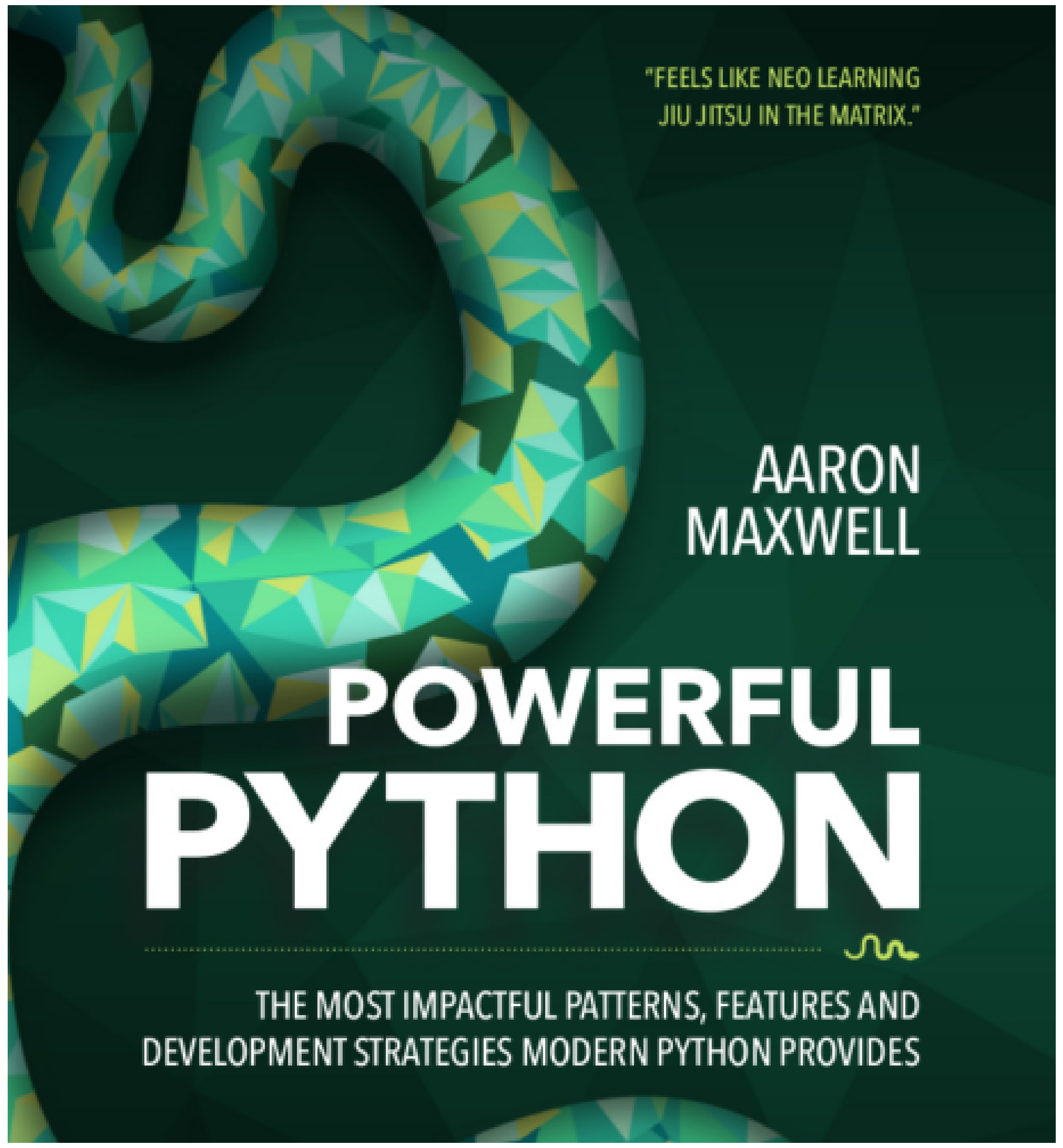
Pythonic Design Patterns

Welcome

I’m your host, Aaron Maxwell.

- Author of **Powerful Python**
- aaron@powerfulpython.com
- On twitter:
    - @powerfulpython (professional)
    - @redsymbol (personal)


"FEELS LIKE NEO LEARNING JIU JITSU IN THE MATRIX."
AARON MAXWELL
POWERFUL PYTHON
THE MOST IMPACTFUL PATTERNS, FEATURES AND DEVELOPMENT STRATEGIES MODERN PYTHON PROVIDES

Our focus in this class: **Design Patterns done in Python.** We’ll emphasize how they are done **differently** from languages like Java, C#, and C++.

How we will proceed

Download courseware ZIP:

- Click on the green "Resource List" icon
- Or fetch from: https://powerfulpython.com/courseware-pat.zip

What's included:

- PDF course book
- Slides
- README.txt with pointers
- Labs (i.e., programming exercises - more on that later)

Give you a break every hour (10 minutes).

Give me a thumbs up. (Let's try it now)

Ask questions anytime.

Python versions

Most code I show you will run in both Python 2 and 3.

Where it's different, I'll code in Python 3, and point out the differences. (There won't be many.)

You can do the programming exercises in either 3, or 2.7.

What makes perfect?

Practice, practice, practice.

- Practice syntax (typing things in)
- Practice programming (higher-level labs)

I expect you to do your part!

You **exponentially** get out of this what you put into it.

GO FOR IT.

Running the labs

Labs are the main programming exercises. You are given a failing automated test; your job is to write Python code to make it pass.

Simply run it as a Python program, any way you like. (For example, "python3 helloworld.py")

Run unmodified first, so you can see the failure report.

When done, click the thumb’s up, and find someone to high-five.

Then: Move on to the extra credit.

Lab: helloworld.py

Let's do our first lab now: 'helloworld.py`

- In `labs/py3` for Python 3.x, or `labs/py2` for 2.7

Instructions are in `LABS.txt` in the courseware.

When you finish:

- Give me a HIGH FIVE! in the chat room, and click Thumbs up, so I know you're done.
- Proceed to `helloworld_extra.py`

You'll know the tests pass when you see:

```
*** ALL TESTS PASS ***
Give someone a HIGH FIVE!
```

When that's done:
Open and skim through **PythonicDesignPatterns.pdf**.
Just notice what topics interest you.

Getting the most

We'll take some class time for each lab. You may not finish, but it's **critically important** that you at least start when I tell you to.

After we're done for the day, find time to finish all the main labs before tomorrow.

Solutions are provided. Use them wisely, not foolishly:

- After you get the lab passing, compare your code to the official solution.
- Other than that, don't look at them if you can avoid it.
- The more work you can do on your own, the more you will learn. Peek at the solution to get a hint when you really need it.

Optional (**only** for future master Pythonistas): Do all the extra labs as well, as soon as you can manage.

Python Trainings

- **Scaling Python with Generators**: Write robust, reliable code to gracefully handle increasing amounts of data
- **Pythonic Object-Oriented Programming**: Master the principles of OOP, in Python
- **Pythonic Design Patterns**: Understand higher-level design patterns. Pythonically
- **Test-Driven Development In Python**: Master automated testing. It’s a *superpower*.
- **Beyond Python Scripts: Logging, Modules, & Dependency Management**: For larger Python applications. Part 1
- **Beyond Python Scripts: Exceptions, Error Handling and Command-Line Interfaces**: Continued with Part 2
- **Python - The Next Level**: See into the Matrix of Python. Master facets of this language that 99% of Python developers will *never* learn

For detailed course descriptions and sign-up information:

https://powerfulpython.com/safari-trainings/

(Select "Python Trainings" under the green "Resource List" icon.)

2.x: Inherit from "object"

If you use Python 2, make sure all your classes inherit from `object`.

Failing to do this creates "old-style classes", and many features will not work. **There is no place in modern Python for old style classes.**

Once you're using Python 3, this is automatic, and you can relax.

```
>>> # Python 2
... class PetFromObject(object):
...     def __init__(self, name):
...         self.name = name
>>> class PetNotFromObject:
...     def __init__(self, name):
...         self.name = name
>>> issubclass(PetFromObject, object)
True
>>> issubclass(PetNotFromObject, object)
False
```

"Protected" Attributes

Python doesn’t have access control modifiers, like Java, C# or C++.

Convention: prefixing with a single underscore means "don’t rely on this being available."

```
class SignalParser:
    def __init__(self):
        self._state = 'waiting'
    def receive(self):
        self._state = 'receiving'
```

But the language doesn’t enforce it.

```
>>> sp = SignalParser()
>>> sp.receive()
>>> print("Mwahaha, I can see your hidden state is " + sp._state)
Mwahaha, I can see your hidden state is receiving
```

"Private" Attributes

Prefix with "__" and subclasses cannot access.

```
class SignalParser:
    def __init__(self):
        self.__state = 'waiting'
    def receive(self):
        self.__state = 'receiving'
```

```
>>> sp = SignalParser()
>>> sp.receive()
>>> print("I cannot read your hidden state! " + sp.__state)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'SignalParser' object has no attribute '__state'
```

But actually, it just mangles the name:

```
>>> print("Oh wait, I CAN! " + sp._SignalParser__state)
Oh wait, I CAN! receiving
```