Programming Principles in Python (CSCI 503/490)

Principles & Notebooks

Dr. David Koop

Administrivia

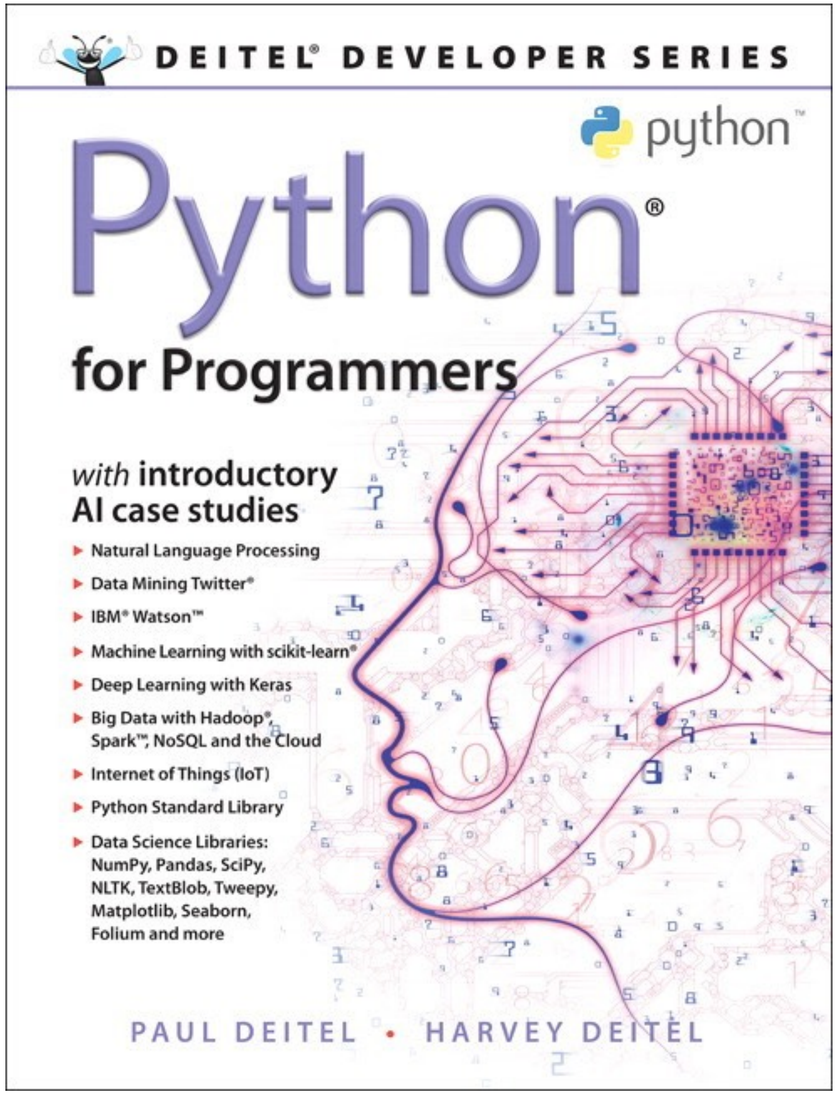
- Course Web Site
- TA: Mohammed Abdul Moyeed (Blackboard Collaborate)
- Syllabus
  - Plagiarism
  - Accommodations
- Assignments
- Tests: 2 (Sept. 28, Nov. 4) and Final (Dec. 7)
- Course is offered to both undergraduates (CS 490) and graduates (CS 503)
  - Grad students have extra topics, exam questions, assignment tasks
- Make sure you are registered for the course!

Office Hours & Email

- Moyeed's office hours will be held via Blackboard Collaborate
  - MW: 12:00-3pm
- Prof. Koop's office hours will be held in person
  - Tu: 1:45-3pm, Th: 10:45am-12pm, or by appointment
- You do not need an appointment to stop by during scheduled office hours, but please adhere to university regulations (Protecting the Pack)
- If you wish to meet virtually, please schedule an appointment
- If you need an appointment, please email me with **details** about what you wish to discuss and times that would work for you
- Many questions can be answered via email. **Please consider writing an email before scheduling a meeting.**

Course Material

- Textbook:
  - Recommended: Python for Programmers
  - Good overview + data science examples
- Many other resources are available:
  - https://wiki.python.org/moin/BeginnersGuide
  - https://wiki.python.org/moin/IntroductoryBooks
  - http://www.pythontutor.com
  - https://www.python-course.eu
  - https://software-carpentry.org/lessons/



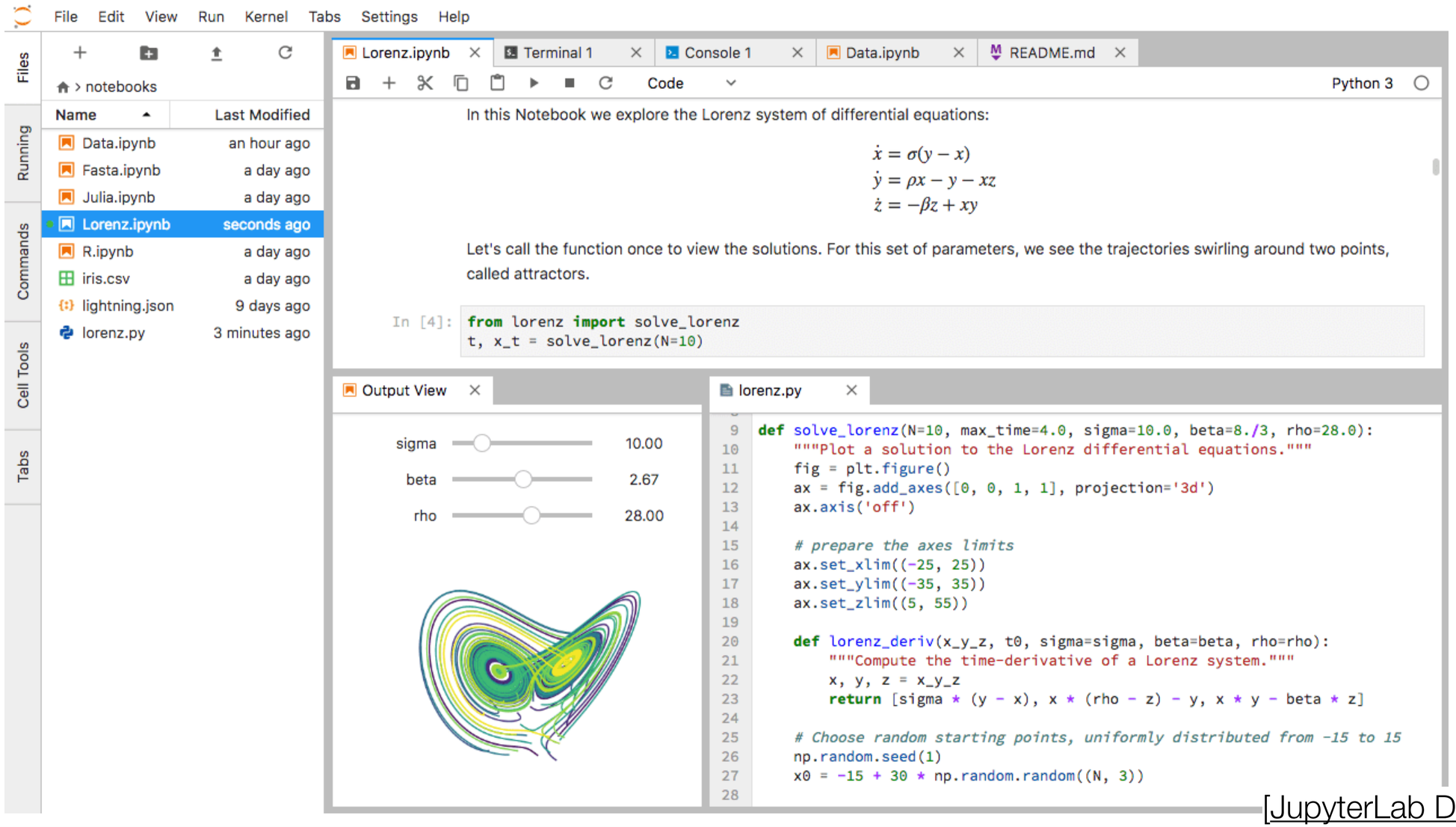
Course Material

- Software:
  - Anaconda Python Distribution (http://anaconda.com/download/): makes installing python packages easier
  - Jupyter Notebook: Web-based interface for interactively writing & executing Python code
  - JupyterLab: An updated web-based interface that includes the notebook and other cool features
  - JupyterHub: Access everything through a server

Why Python?

- High-level, readable
- Productivity
- Large standard library
- Libraries, Libraries, Libraries
- What about Speed?
  - What speed are we measuring?
  - Time to code vs. time to execute

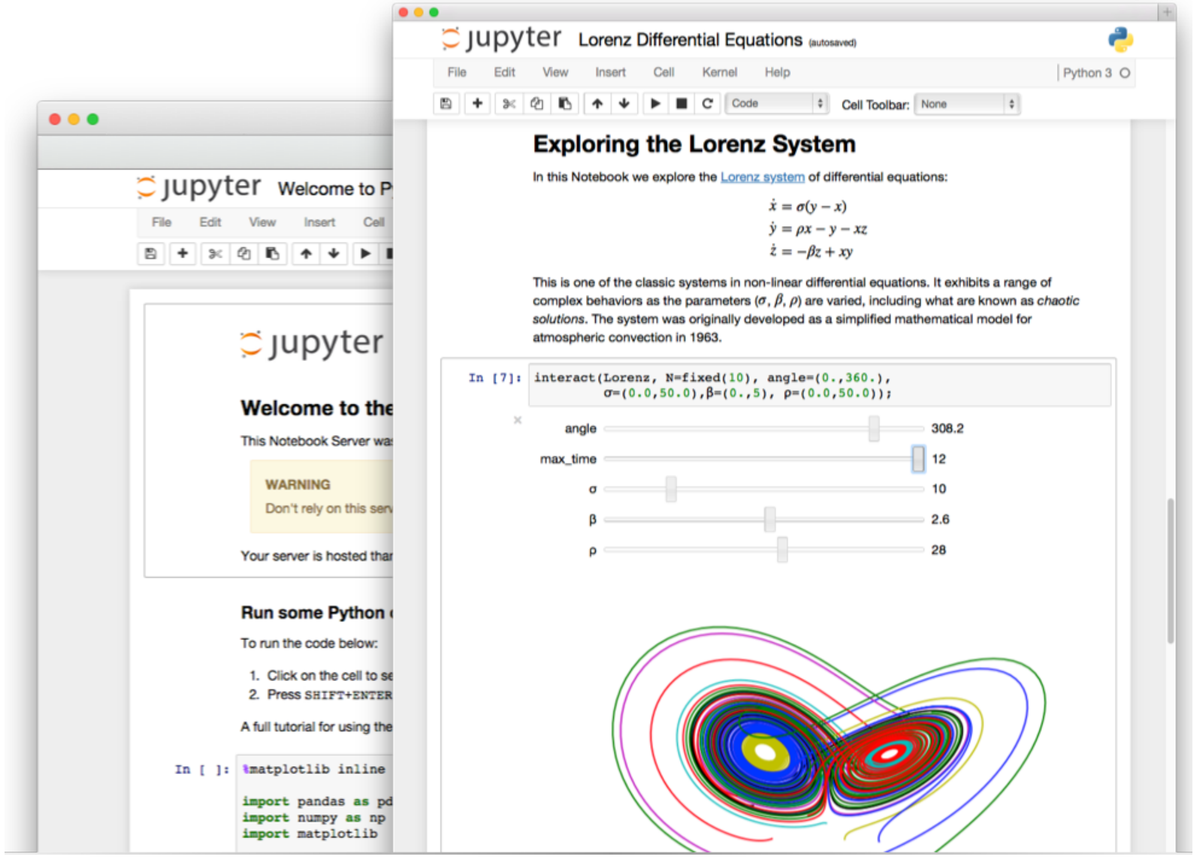
JupyterLab and Jupyter Notebooks



[JupyterLab Documentation]

Jupyter Notebooks

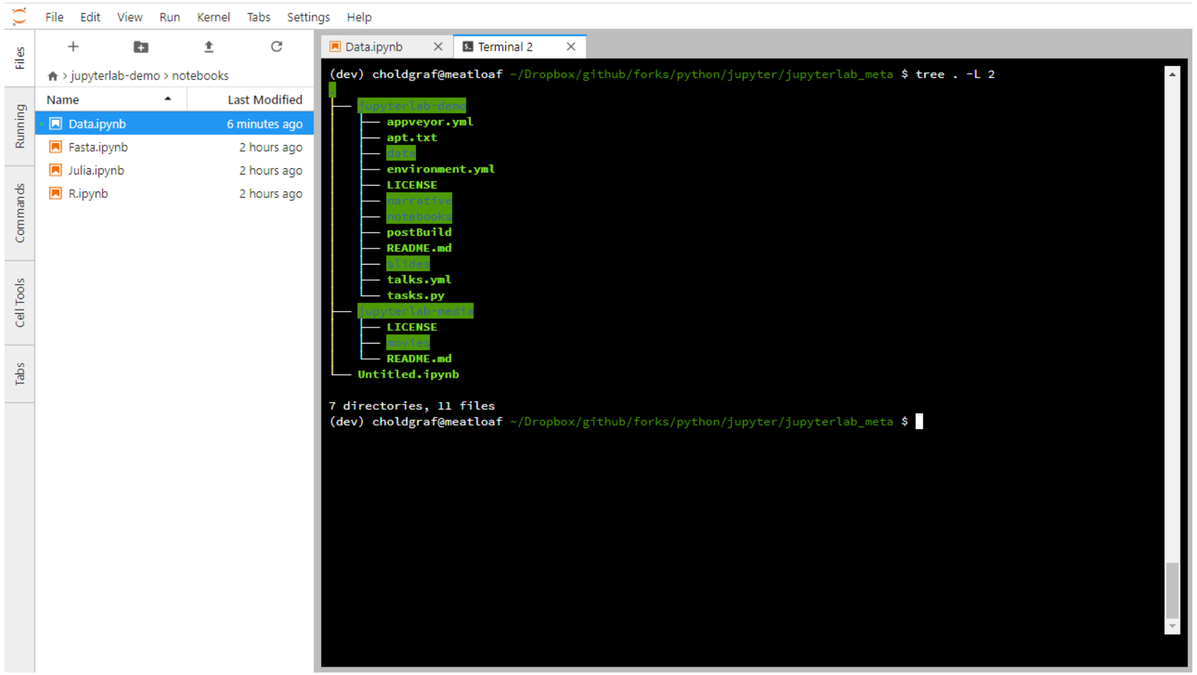
- Display rich representations and text
- Uses Web technology
- Cell-based
- Built-in editor
- GitHub displays notebooks



[Jupyter]

Other JupyterLab Features

- Terminal
  - Similar to what you see on turing/ hopper but for your local machine
- File Viewers
  - CSV
  - Plugins available
- Console
  - Can be linked to notebooks



Using Python & JupyterLab Locally

- www.anaconda.com/download/
- Anaconda has JupyterLab
- Use Python 3.8
- Anaconda Navigator
  - GUI application for managing Python environment
  - Can install packages
  - Can start JupyterLab
- Can also use the shell to do this:
  - `$ jupyter lab`
  - `$ conda install <pkg_name>`

Using Python & JupyterLab on Course Server

- https://tiger.cs.niu.edu/jupyter/
- Login with you Z-ID (lowercase z)
- You should have received an email with your password
- Advanced:
  - Can add your own conda environments in your user directory

Assignment 1

- Due next Thursday
- Get acquainted with Python using notebooks
- Make sure to follow instructions
  - Name the submitted file a1.ipynb
  - Put your name and z-id in the first cell
  - Label each part of the assignment using markdown
  - Make sure to produce output according to specifications

Programming Principles

Zen of Python

- Written in 1999 by T. Peters in a message to Python mailing list
- Attempt to channel Guido van Rossum's design principles
- 20 aphorisms, 19 written, 1 left for Guido to complete (never done)
- Archived as PEP 20
- Added as an easter egg to python (`import this`)
- Much to be deciphered, in no way a legal document
- Jokes embedded
- Commentary by A.-R. Janhangeer

Zen of Python

```
>>> import this
```

1. Beautiful is better than ugly.
2. Explicit is better than implicit.
3. Simple is better than complex.
4. Complex is better than complicated.
5. Flat is better than nested.
6. Sparse is better than dense.
7. Readability counts.
8. Special cases aren't special enough to break the rules.
9. Although practicality beats purity.

Zen of Python

10. Errors should never pass silently.
11. Unless explicitly silenced.
12. In the face of ambiguity, refuse the temptation to guess.
13. There should be one-- and preferably only one --obvious way to do it.
14. Although that way may not be obvious at first unless you're Dutch.
15. Now is better than never.
16. Although never is often better than *right* now.
17. If the implementation is hard to explain, it's a bad idea.
18. If the implementation is easy to explain, it may be a good idea.
19. Namespaces are one honking great idea—let's do more of those!

Explicit Code

- Goes along with complexity
- Bad:

```
def make_complex(*args):
    x, y = args
    return dict(**locals())
```

- Good

```
def make_complex(x, y):
    return {'x': x, 'y': y}
```

[The Hitchhiker's Guide to Python]

Avoid the Magical Wand

- You can change almost anything Python does
  - Modify almost any core function
  - Change how objects are created/instantiated
  - Change how modules are imported
- Good because no problem is impossible
- But know when **not** to use extraordinary measures

[The Hitchhiker's Guide to Python]

One Statement per Line

- Bad:
  - `print 'one'; print 'two'`
  - `if <complex comparison> and <other complex comparison>:`
    `    # do something`
- Good:
  - `print 'one'`
    `print 'two'`
  - `cond1 = <complex comparison>`
    `cond2 = <other complex comparison>`
    `if cond1 and cond2:`
    `    # do something`

[The Hitchhiker's Guide to Python]

Don't Repeat Yourself

- "Two or more, use a for" [Dijkstra]
- Rule of Three: [Roberts]
  - Don't copy-and-paste more than once
  - Refactor into methods
- Repeated code is harder to maintain
- Bad

```
f1 = load_file('f1.dat')
r1 = get_cost(f1)
f2 = load_file('f2.dat')
r2 = get_cost(f2)
f3 = load_file('f3.dat')
r3 = get_cost(f3)
```

- Good

```
for i in range(1,4):
    f = load_file(f'f{i}.dat')
    r = get_cost(f)
```

Defensive Programming

- Consider corner cases
- Make code auditable
- Process exceptions
- Bad
  -
```
def f(i):
    return 100 / i
```
- Good:
  -
```
def f(i):
    if i == 0:
        return 0
    return 100/i
```

Object-Oriented Programming

- ?

Object-Oriented Programming

- Encapsulation (Cohesion): Put things together than go together
- Abstraction: Hide implementation details (API)
- Inheritance: Reuse existing work
- Polymorphism: Method reuse and strategies for calling and overloading

Programming Requires Practice

Modes of Computation

- Python is **interpreted**: you can run one line at a line without compiling
- Interpreter in the Shell
  - Execute line by line
  - Hard to structure loops
  - Usually execute whole files (called scripts) and edit those files
- Notebook
  - Richer results (e.g. images, tables)
  - Can more easily edit past code
  - Re-execute any cell, whenever

Python Interpreter from the Shell

- On tiger, use `conda init` to make sure you are using the latest version of python (the same version used by the notebook environment)
- We will discuss this more later, but want to show how this works

Python in a Notebook

- Richer results (e.g. images, tables)
- Can more easily edit past code
- Re-execute any cell, whenever

Multiple Types of Output

- stdout: where print commands go
- stderr: where error messages go
- display: special output channel used to show rich outputs
- output: same as display but used to display the value of the last line of a cell

Print function

- `print(“Welcome, Jane")`
- Can also print variables:

```
name = "Jane"
print("Welcome,", name)
```

Python Variables and Types

- No type declaration necessary
- Variables are names, not memory locations

```
a = 0
a = "abc"
a = 3.14159
```

- Don't worry about types, but think about types
- Strings are a type
- Integers are as big as you want them
- Floats can hold large numbers, too (double-precision)

Python Math and String "Math"

- Standard Operators: +, -, *, /, %
- Division "does what you want" (new in v3)
  - `5 / 2 = 2.5`
  - `5 // 2 = 2 # use // for integer division`
- Shortcuts: `+=, -=, *=`
- No `++, --`
- Exponentiation (Power): `**`
- Order of operations and parentheses: (`4 - 3 - 1` vs. `4 - (3 - 1)`)
- `"abc" + "def"`
- `"abc" * 3`

Python Strings

- Strings can be delimited by single or double quotes
  - `"abc"` and `'abc'` are exactly the same thing
  - Easier use of quotes in strings: `"Joe's"` or `'He said "Stop!"'`
- Triple quotes allow content to go across lines and preserves linebreaks
  - ```
    """This is another
    string"""
    ```
- String concatenation: `"abc" + "def"`
- Repetition: `"abc" * 3`
- Special characters: `\n \t` like Java/C++
    ```