Programming Principles in Python (CSCI 503/490)

Principles & Notebooks

Dr. David Koop

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

Administrivia

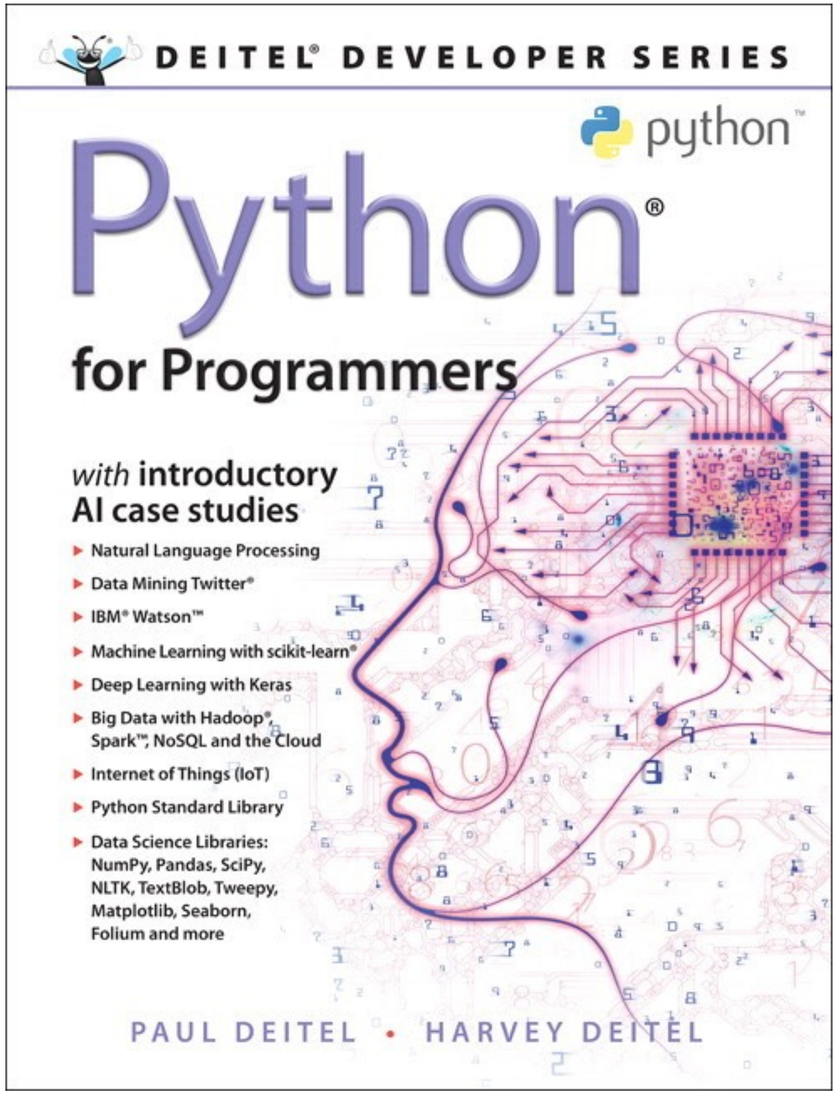
- Course Web Site
- TAs: Naga Jyothi Kota & Angel Prathyusha Koyi
- Syllabus
  - Plagiarism
  - Accommodations
- Assignments
- Tests: 2 (Feb. 21, Apr. 3) and Final (May 6)
- Course is offered to both undergraduates (CS 490) and graduates (CS 503)
  - Grad students have extra topics, exam questions, assignment tasks

Office Hours & Email

- TA office hours will be held in person in PM 356
  - M 11am-12pm, 3–5pm, Tu 9:30am-12:30pm, W 1-4pm, Th 9:30am-12:30pm
- Prof. Koop's office hours will be held in person in PM 461
  - M: 1:45-3:00pm, W: 10:45am-12:00pm, or by appointment
  - You do not need an appointment to stop by during scheduled office hours,
  - 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

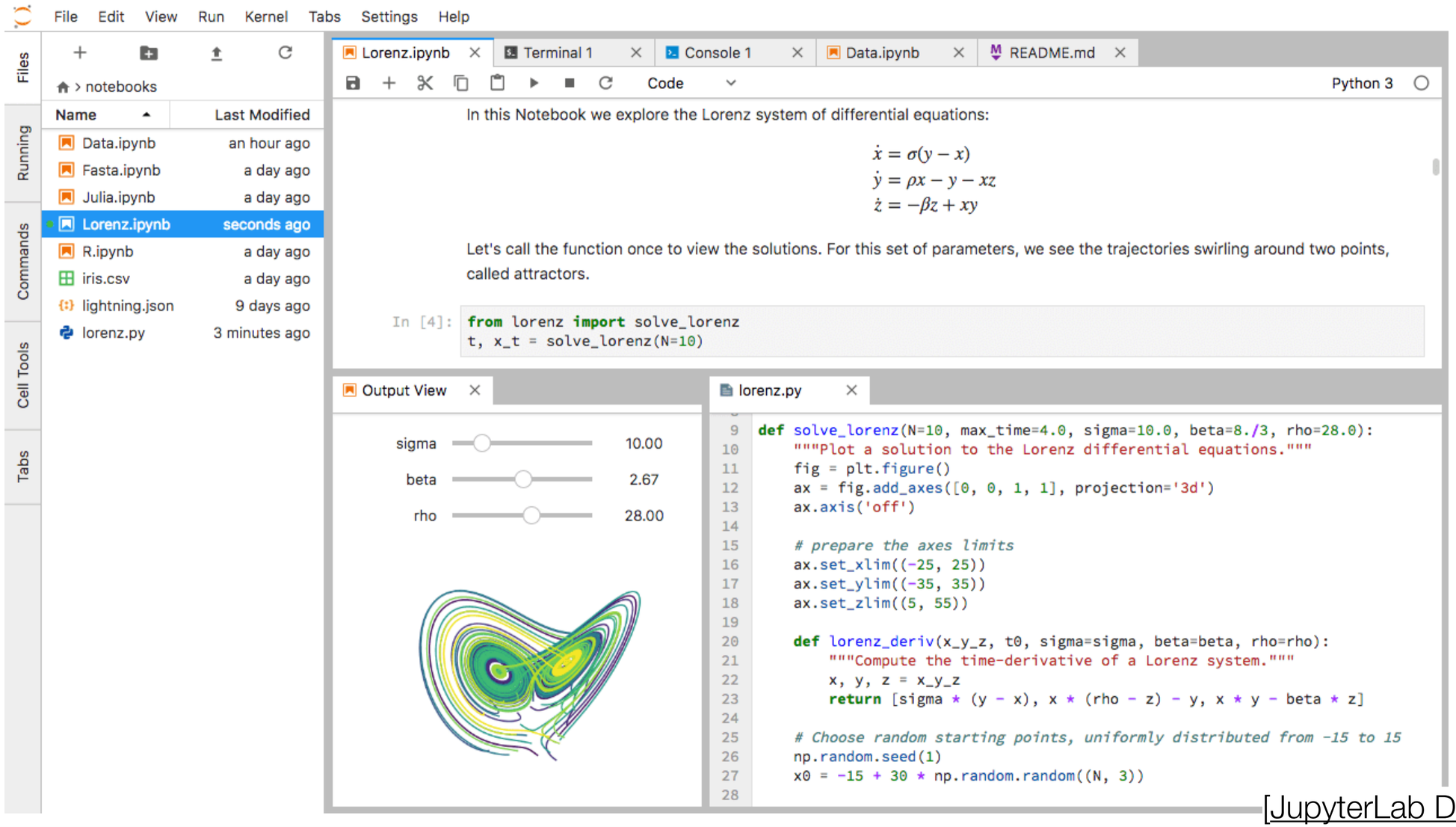
Syllabus Questions?

Assignment 1

- Released today, due next Monday
- Goal: Become 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

Class Roster

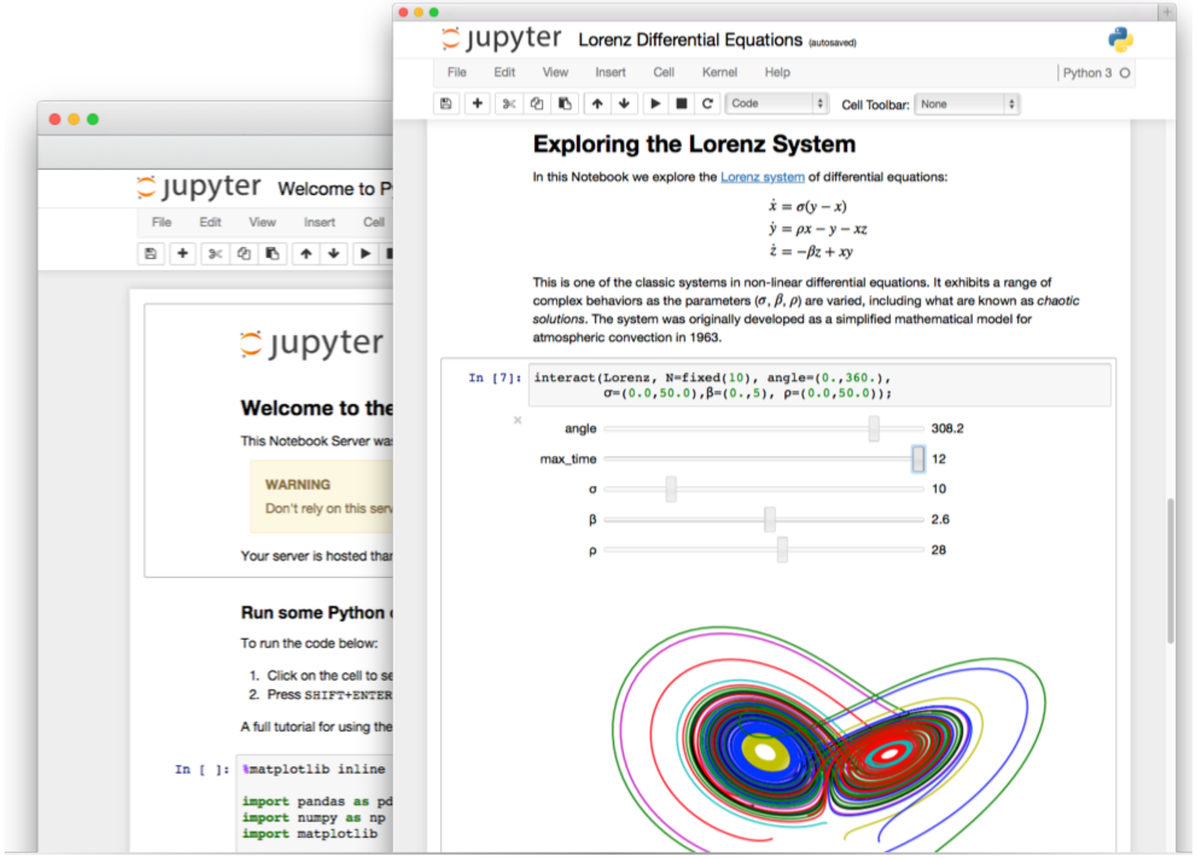
JupyterLab and Jupyter Notebooks



[JupyterLab Documentation]

Jupyter Notebooks

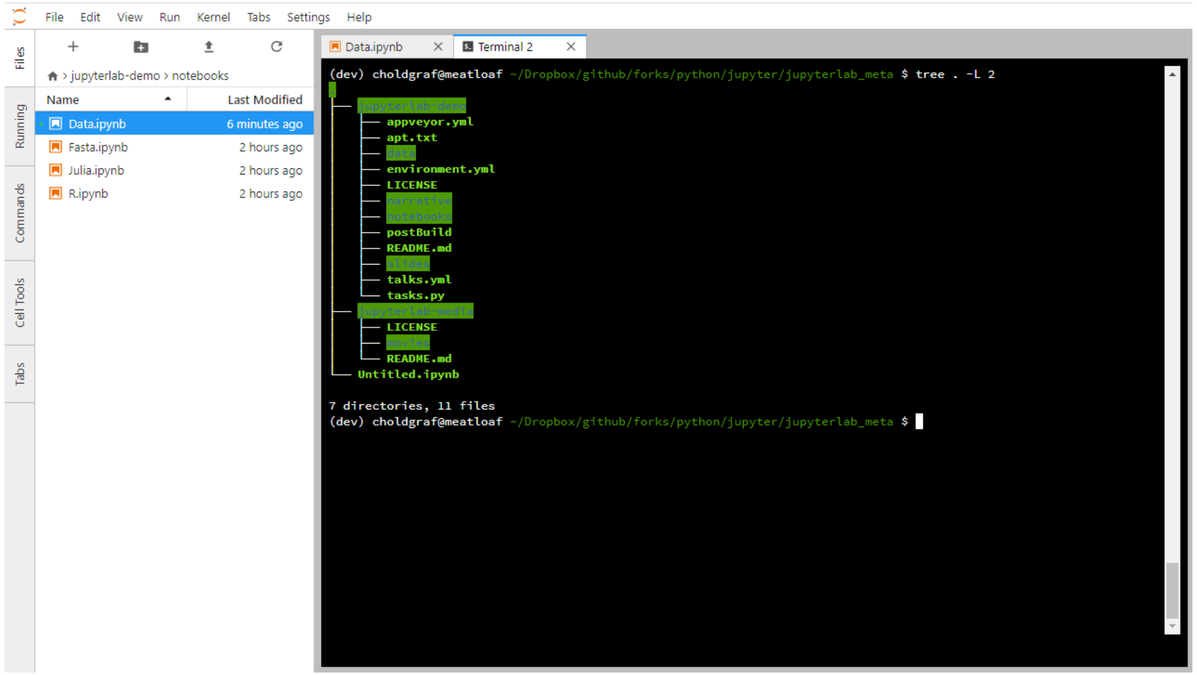
- Display rich representations and text
- Uses Web technology
- Cell-based
- Built-in editor
- GitHub displays notebooks



[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/
- Consider mamba (faster) and conda-forge
- Anaconda includes JupyterLab
- Use Python 3.12
- Anaconda Navigator
  - GUI application for managing Python environment
  - Can install packages & 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

JupyterLab Notebook Tips

- Starts with a directory view
- Create new notebooks using the Launcher (+ icon on the left)
  - New notebooks have the name "Untitled"
  - File → Rename Notebook… (or right-click) to change the name
- Save a notebook using the command under the File menu
- Shutting down the notebook requires quitting the kernel
  - Web browser is **interface** to display code and results
  - **Kernel** runs the code: may see messages in a console/terminal window
  - Closing the browser window does not stop Jupyter
  - Use File → Hub Control Panel → Stop My Server to reset on tiger

JupyterLab Notebooks

- Open a notebook using the left panel like you would in a desktop view
- Past results are displayed—does not mean they are loaded in memory
- Use "Run All" or "Run All Above" to re-execute past work
  - If you shut down the kernel, all of the data and variables you defined need to be redefined (so you need to re-run all)
  - **Watch Out—Order Matters**: If you went back and re-executed cells in a different order than they are shown, doing "Run All" may not produce the same results!
- Edit mode (green) versus Command mode (blue == **Be Careful**)

JupyterLab Notebooks

- Can write code or plain text (can be styled Markdown)
  - Choose the type of cell using the dropdown menu
- Cells break up your code, but all data is **global**
  - Defining a variable `a` in one cell means it is available in **any** other cell
  - This includes cells **above** the cell `a` was defined in!
- Remember **Shift+Enter** to execute
- Enter just adds a new line
- Use `?<function_name>` for help
- Use Tab for **auto-complete** or suggestions
- Tab also indents, and Shift+Tab unindents

JupyterLab Notebooks

- You can interrupt the kernel or restart if things seem stuck
- You can download your notebooks if working remotely
- Common Keyboard Shortcuts

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