Course Goals

• Simply: to learn how to use python to do
  – Numerical analysis
  – Data analysis
  – Plotting and visualizations
  – Symbol mathematics
  – Write applications
  – ...
Class Participation

- We'll learn by interactively trying out some ideas and looking at code
  - I want to learn from all of you—share your experiences and expertise
- **We'll use slack to communicate out of class**
  - Everyone should have received an invite to our slack: https://python-sbu.slack.com
- Try out ideas and report to the class what you've learned
  - I can set up an account for you on one of my Linux servers if you want a place to try things out (you would log in remotely)
• Log onto our slack as soon as possible
  – If you didn’t get an invite, e-mail me, and I’ll add you
• Slack is a web-based team chat tool
  – I’ve setup a number of channels for us to focus our discussions
  – Everyone is expected to participate
• Your course grade is based on your participation
  – I have a simple script(https://github.com/zingale/slack_grader) that I’ll use for grading
    • Good contributions will get a comment on the slack, counting as “+1”
    • The script will record these points over the semester
    • Help, by using slack reactions for useful comments from your classmates
• In “free” mode, slack only keeps 10k messages—I don’t think we’ll go over that during the semester
Why Slack?

- One of the goals of this class is to teach you tools that are used by computational science groups
- Slack has gained enormous adoption by research groups over the past few years
  - We’ll see how to integrate github repos to it, so everyone can keep on top of code developments
  - It’s easy to post code snippets in conversations
  - There are lots of integrations that are available to extend its usefulness
Why Python?

- Very high-level language
  - Provides many complex data-structures (lists, dictionaries, ...)
  - Your code is shorter than a comparable algorithm in a compiled language

- Many powerful libraries to perform complex tasks
  - Parse structured inputs files
  - send e-mail
  - interact with the operating system
  - make plots
  - make GUIs
  - do scientific computations
  - ...

- Easy to prototype new tools

- Cross-platform and Free
Why Python?

- Dynamically-typed
- Object-oriented foundation
- Extensible (easy to call Fortran, C/C++, ...)
- Automatic memory management (garbage collection)
- Ease of readability (whitespace matters)

... and for this course ...

- Very widely adopted in the scientific community
  - Mostly due to the very powerful array library NumPy
You're flying! How?

I learned it last night! Everything is so simple!
Hello world is just print "Hello, world!"

I dunno... dynamic typing? Whitespace?
Come join us! Programming is fun again!
It's a whole new world up here!
But how are you flying?

I just typed import antigravity
That's it?
... I also sampled everything in the medicine cabinet for comparison.
But I think this is the Python.
Installing Python

- **Linux**
  - Python is probably already installed
  - The dependencies we need for our class should be available through your package manager

- **OS X / Windows**
  - The easiest way to get everything we need for class is by installing Anaconda: [https://www.continuum.io/downloads](https://www.continuum.io/downloads)
    - You'll have a choice of python 2.x or 3.x—we'll try to write code compatible with both, but 3.x is the future

- If you run into problems ask on the discussion forum (I've already started a thread), or come by my office
Hello, World!

- If you have python installed properly, you can start python simply by typing `python` at your command line prompt
  - The python shell will come up
  - Our hello world program is simply:
    ```python
    print("hello, world")
    ```
  - Or, in python 2 (more on this later...):
    ```python
    print "hello, world"
    ```
Communities

- Many scientific disciplines have their own python communities that
  - Provide tutorials
  - Cookbooks
  - Libraries to do standard analysis in the field
- For the most part, these build on the Open nature of python
- I've put links on our course page to some information on the python communities for:
  - Astronomy
  - Atmospheric Science
  - Biology
  - Cognitive Science
  - Psychology
  - Let me know of any others!
Most scientific libraries in python build on the Scientific Python stack:

- **NumPy**: Base N-dimensional array package
- **SciPy**: Fundamental library for scientific computing
- **Matplotlib**: Comprehensive 2D Plotting
- **IPython**: Enhanced Interactive Console
- **Sympy**: Symbolic mathematics
- **pandas**: Data structures & analysis

(scipy.org)
Basics of Computation

- Computers store information and allow us to operate on it.
  - That's basically it.
  - Computers have finite memory, so it is not possible to store the infinite range of numbers that exist in the real world, so approximations are made.

- You should have some familiarity with how computers store numbers
  - Great floating point reference
    - *What Every Computer Scientist Should Know About Floating-Point Arithmetic* by D. Goldberg
You can write any algorithm in any programming language—they all provide the necessary logical constructs

- However, some languages make things much easier than others
  - **C**
    - Excellent low-level machine access (operating systems are written in C)
    - Multidimensional arrays are “kludgy”
  - **Fortran** (Formula Translate)
    - One of the earliest compiled languages
    - Large code base of legacy code
    - Modern Fortran offers many more conveniences than old Fortran
    - Great support for arrays
  - **Python**
    - Offers many high-level data-structures (lists, dictionaries, arrays)
    - Great for quick prototyping, analysis, experimentation
    - Increasingly popular in scientific computing
Computer Languages

- IDL
  - Proprietary
  - Great array language
  - Modern (like object-oriented programming) features break the “clean-ness” of the language
- C++
- Others
  - Ruby, Perl, shell scripts, ...
Computer Languages

- **Compiled languages** (Fortran, C, C++, ...)
  - Compiled into machine code—machine specific
  - Produce faster code (no interpretation is needed)
  - Can offer lower level system access (especially C)

- **Interpreted languages** (python, IDL, perl, Java (kind-of) ...)
  - Not converted into machine code, but instead interpreted by the interpreter
  - Great for prototyping
  - Can modify itself while running
  - Platform independent
  - Often has dynamic typing and scoping
  - Many offer garbage collection

Vector languages

Some languages are designed to operate on entire arrays at once (python + NumPy, many Fortran routines, IDL, ...)

- For interpreted languages, getting reasonable performance requires operating on arrays instead of explicitly writing loops
  - Low level routines are written in compiled language and do the loop behind the scenes
- We'll see this in some detail when we discuss python

Next-generation computing = GPUs?

- Hardware is designed to do the same operations on many pieces of data at the same time
Python is an object-oriented language

Think of an object as a container that holds both data and functions (methods) that know how to operate on that data.

- Objects are built from a datatype called a class—you can create as many instances (objects) from a class as memory allows
  - Each object will have its own memory allocated

Objects provide a convenient way to package up data

- In Fortran, think of a derived type that also has its own functions
- In C, think of a struct that, again, also has its own functions

Everything, even integers, etc. is an object

- $1 + 2$ will be interpreted as $(1).\text{__add__}(2)$ in python
## Programming Paradigms

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Description</th>
<th>Main characteristics</th>
<th>Related paradigm(s)</th>
<th>Critics</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Imperative</strong></td>
<td>Computation as statements that directly change a program state (datafields)</td>
<td>Direct assignments, common data structures, global variables</td>
<td></td>
<td>Edsger W. Dijkstra, Michael A. Jackson</td>
<td>C, C++, Java, PHP, Python</td>
</tr>
<tr>
<td><strong>Structured</strong></td>
<td>A style of imperative programming with more logical program structure</td>
<td>Structograms, indentation, either no, or limited use of goto statements</td>
<td>Imperative</td>
<td></td>
<td>C, C++, Java</td>
</tr>
<tr>
<td><strong>Procedural</strong></td>
<td>Derived from structured programming, based on the concept of modular programming or the procedure call</td>
<td>Local variables, sequence, selection, iteration, and modularization</td>
<td>Structured, imperative</td>
<td></td>
<td>C, C++, Lisp, PHP, Python</td>
</tr>
<tr>
<td><strong>Functional</strong></td>
<td>Treats computation as the evaluation of mathematical functions avoiding state and mutable data</td>
<td>Lambda calculus, compositionality, formula, recursion, referential transparency, no side effects</td>
<td></td>
<td>Erlang, Haskell, Lisp, Clojure, Scala, F#</td>
<td></td>
</tr>
<tr>
<td><strong>Event-driven</strong></td>
<td><strong>including time driven</strong> Program flow is determined mainly by events, such as mouse clicks or interrupts including timer</td>
<td>Main loop, event handlers, asynchronous processes</td>
<td>Procedural, dataflow</td>
<td>ActionScript</td>
<td></td>
</tr>
<tr>
<td><strong>Object-oriented</strong></td>
<td>Treats datafields as objects manipulated through pre-defined methods only</td>
<td>Objects, methods, message passing, information hiding, data abstraction, encapsulation, polymorphism, inheritance, serialization-marshaling</td>
<td></td>
<td>C++, C#, Java, PHP, Python, Ruby, Scala</td>
<td></td>
</tr>
<tr>
<td><strong>Declarative</strong></td>
<td>Defines computation logic without defining its detailed control flow</td>
<td>4GLs, spreadsheets, report program generators</td>
<td></td>
<td>SQL, regular expressions, CSS</td>
<td></td>
</tr>
<tr>
<td><strong>Automata-based</strong></td>
<td>programming Treats programs as a model of a finite state machine or any other formal automata</td>
<td>State enumeration, control variable, state changes, isomorphism, state transition table</td>
<td>Imperative, event-driven</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(Wikipedia)
## Potential Topics

- **General python**
- **Advanced python / special modules**
  - Regular expressions
  - Command line/input file parsing
  - Filesystem operations
- **NumPy**
- **SciPy** and numerical methods
  - Integration
  - ODEs
  - Curve fitting/optimization
  - Interpolation
  - Signal processing (FFT)
  - Linear algebra
- **Plotting / visualization**
  - `matplotlib`
  - MayaVi
- **Workflow management with IPython**
- **Extending python w/ Fortran and C/C++**
- **Symbolic math with SymPy**
- **Writing python applications**
- **GUI programming**
- **Unit testing**
- **Git/github**
- **Others?**
Python Shell

- This is the standard (and most basic) interactive way to use python
  - Runs in a terminal window
  - Provides basic interactivity with the python language
- Simply type `python` (or `python3`) at your command prompt
  - You can scroll back through the history using the arrows
IPython Shell

- Type ipython (or ipython3) at your prompt
- Like the standard shell, this runs in a terminal, but provides many conveniences (type %quickref to see an overview)
  - Scrollback history preserved between sessions
  - Built-in help with ?
    - function-name?
    - object?
  - Magics (%lsmagic lists all the magic functions)
    - %run script: runs a script
    - %timeit: times a command
    - Lots of magics to deal with command history (including %history)
  - Tab completion
  - Run system commands (prefix with a !)
  - Last 3 output objects are referred to via _, __, ___
Jupyter Notebooks

- A web-based environment that combines code and output, plots, plain text/stylized headings, LaTeX, ...
  - Notebooks can be saved and shared
  - Viewable on the web via: http://nbviewer.ipython.org/
  - Provides a complete view of your entire workflow

- Start with `ipython notebook`

- I'll provide notebooks for a lot of the lectures to follow so you can play along on your own
  - The best way to learn is to experiment—download the notebooks and play around
  - Discuss anything you don't understand in the discussion forum
Python Scripts

- Scripts are a non-interactive means of writing python code
  - filename.py
  - Can be executable by adding:
    `#!/usr/bin/env python`
    as the first line and setting the executable bit for the file
- This is also the way that you write python modules that can be imported into other python code (more on that later...)
Python 2.x vs. Python 3

- See [https://wiki.python.org/moin/Python2orPython3](https://wiki.python.org/moin/Python2orPython3)
- Mostly about cleaning up the language and making things consistent
  - e.g. `print` is a statement in python 2.x but a function in 3.x
- Some trivial differences
- `.pyc` are now stored in a `__pycache__` directory
- Some gotyas:
  - `1 / 2` will give different results between python 2 and 3
- It's possible to write code that works with both python 2 and 3—often we will do so by importing from `__future__`
Python 2.x vs. Python 3

- Write for both
  - In python 2.6+, do
    ```python
    from __future__ import print_function
    and then use the new print() style
    - exec cmd becomes exec(cmd)
    - Some small changes to how __init__.py are done (more on this later)
  ```
- One some systems, you can have python 2.x and 3.x installed side by side
  - May need to install packages twice, e.g. python-numpy and python3-numpy
Class Organization

- We’ll work mostly with Jupyter notebooks from now one (with a few exceptions)
- Each week, I’ll ask you to work through some notebooks on your own, outside of class
  - These will always be posted on the class website
  - Great opportunity to ask questions on slack
- The hope is that we’ll get a lot of the basic concepts for the week covered by working through the notebooks
- In class, we’ll work on some exercises together
- We’ll fine-tune some of this as we go through the semester
Let’s Play

- There are a number of notebooks on our website to demonstrate some core ideas
  - Data types
  - Advanced data types
  - Control flow
  - Functions
  - Classes
  - Modules