

# CDS 230: Modeling and Simulation I

Introduction to computational science with Python

Carlos Cruz

George Mason University

Fall 2019

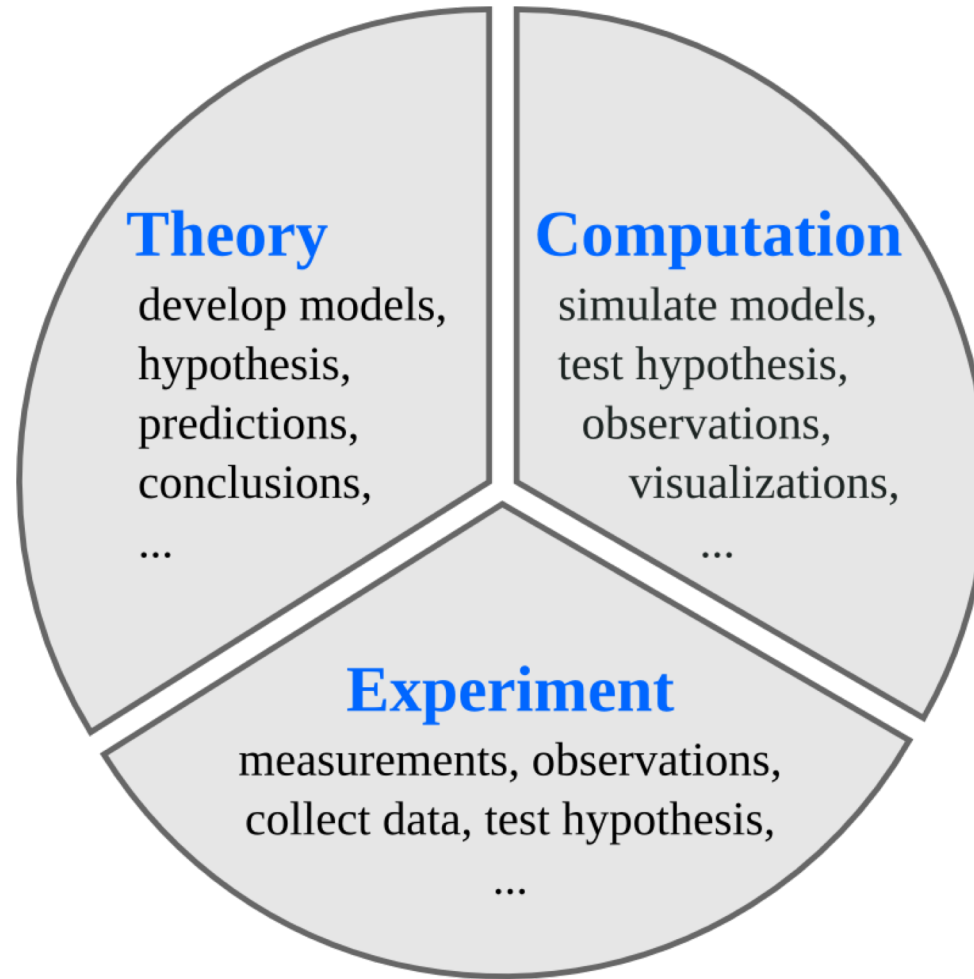
# Introduction

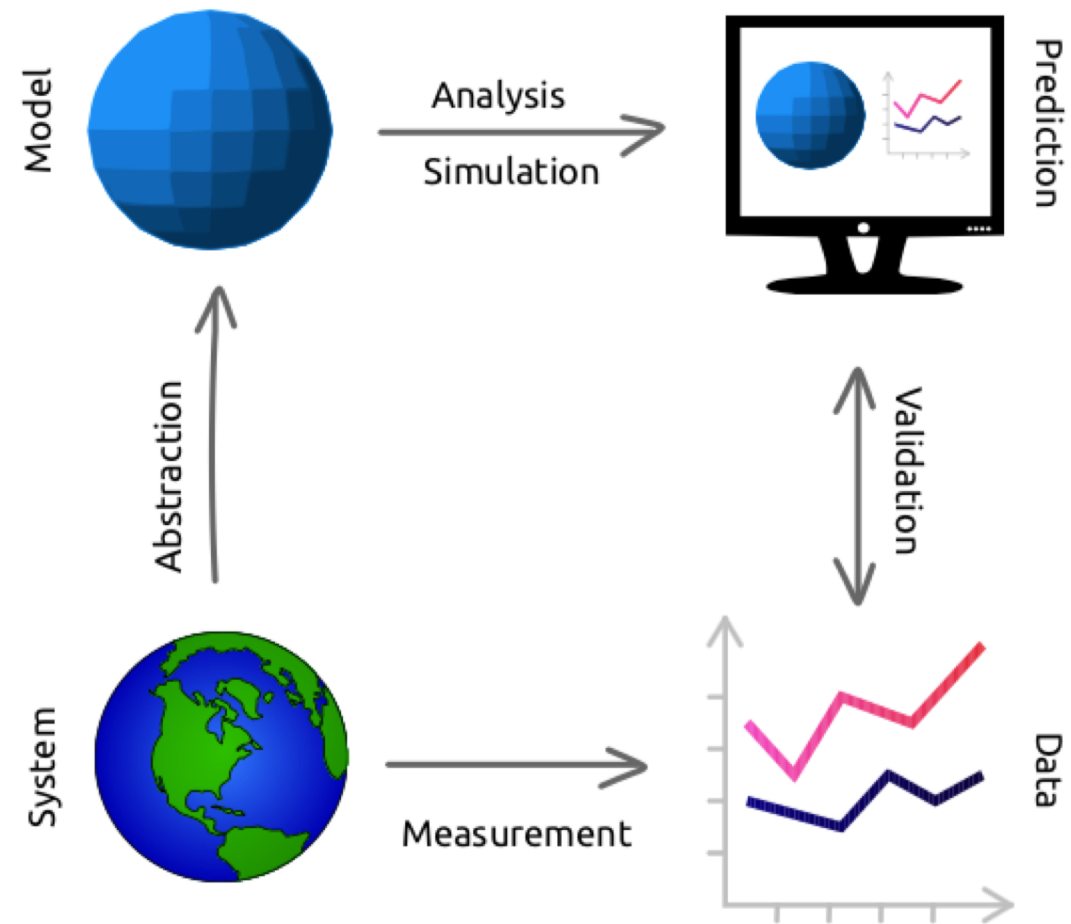
CDS 230 teaches modeling and simulation concepts using the Python programming language. Emphasis will be two-fold:

- ▶ introduction to the Python programming language
- ▶ conversion of problem descriptions into algorithms

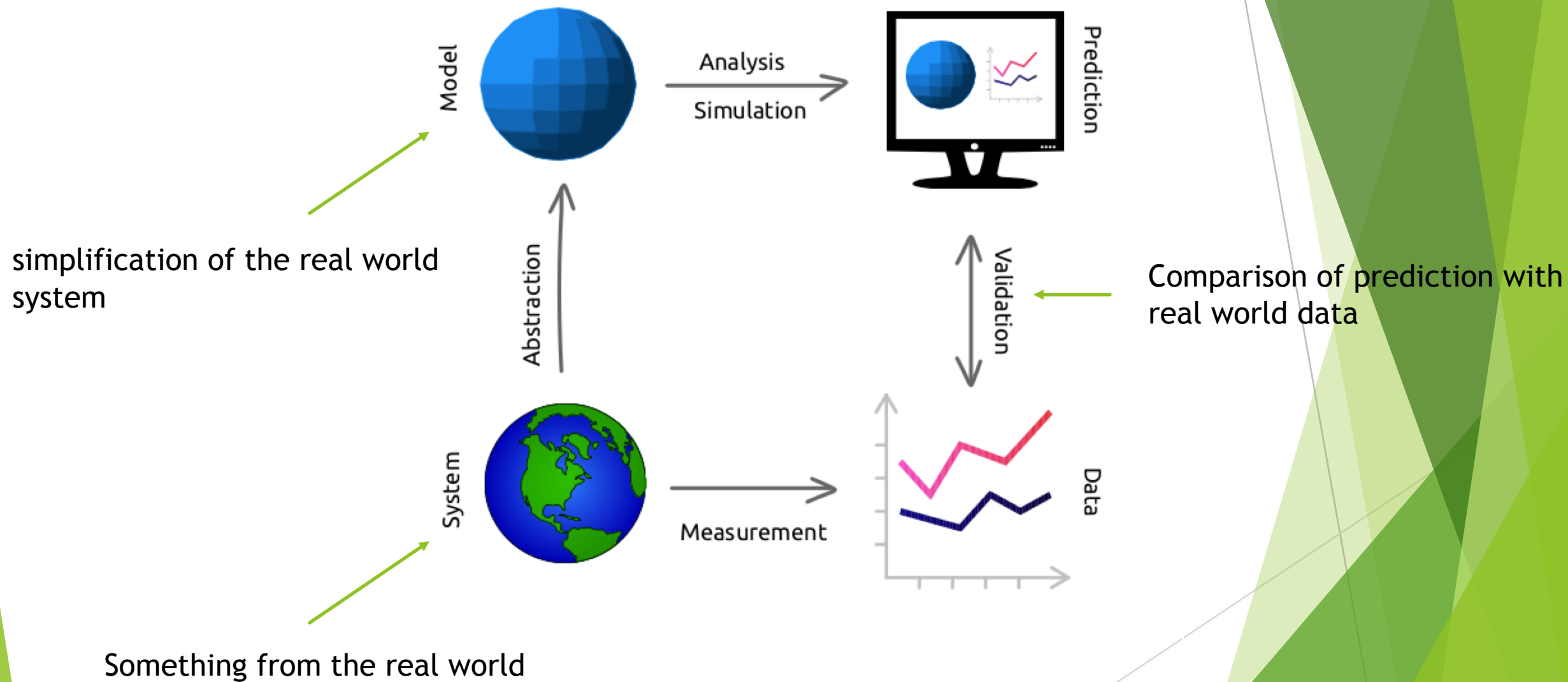
# Goal by the end of the semester

Given a scientific problem(\*) with a problem description, you will be able to write a complete, useful program to solve the problem





From Allen Downey, "Modeling and Simulation in Python".



# Requirements for computational science

- ▶ **Replication:** Given a simulation other scientists should also be able to perform the same calculations and obtain the same results.
- ▶ **Reproducibility:** The results obtained from numerical simulations should be reproducible (\*) with an independent implementation of the method, or using a different method altogether.

# To achieve these goals, we need to:

- ▶ Keep and take note of *\*exactly\** which source code and version that was used to produce data and figures in published papers.
- ▶ Record information of which version of external software that was used. Keep access to the environment that was used.
- ▶ Make sure that old codes and notes are backed up and kept for future reference.
- ▶ Be ready to give additional information about the methods used, and perhaps also the simulation codes, to an interested reader who requests it (even years after the paper was published!).
- ▶ Ideally codes should be published online, to make it easier for other scientists interested in the codes to access it.



# Tools for managing source code

Ensuring replicability and reproducibility of scientific simulations is a \*complicated problem\*, but there are good tools to help with this:

- ▶ Revision Control System (RCS) software.
  - ▶ We will use git - <http://git-scm.com>
- ▶ Online repositories for source code. Available as both private and public repositories.
  - ▶ Github - <http://www.github.com>
  - ▶ Gitlab - <http://www.gitlab.com>

Repositories are also excellent for version controlling manuscripts, figures, thesis files, data files, lab logs, etc. Basically for any digital content that must be preserved and is frequently updated. Again, both public and private repositories are readily available. They are also excellent collaboration tools!

# What is python<sup>™</sup> ?

- ▶ Python is a modern, open-source, general-purpose, object-oriented, high-level programming language.

# General characteristics of Python:

- ▶ **clean and simple language:**
  - ▶ easy-to-read and intuitive code
  - ▶ easy-to-learn minimalistic syntax
  - ▶ maintainability scales well with size of projects.
- ▶ **expressive language:**
  - ▶ fewer lines of code
  - ▶ fewer bugs
  - ▶ easier to maintain.

# Technical details:

- ▶ dynamically typed: No need to define the type of variables, function arguments or return types.
- ▶ automatic memory management: No need to explicitly allocate and deallocate memory for variables and data arrays. No memory leak bugs.
- ▶ interpreted: No need to compile the code. The Python interpreter reads and executes the python code directly.

# Advantages

- ▶ The main advantage is ease of programming, minimizing the time required to develop, debug and maintain the code.
- ▶ Well designed language that encourage many good programming practices:
- ▶ Modular and object-oriented programming, good system for packaging and re-use of code. This often results in more transparent, maintainable and bug-free code.
- ▶ Documentation tightly integrated with the code.
- ▶ A large standard library, and a large collection of add-on packages.

# Disadvantages:

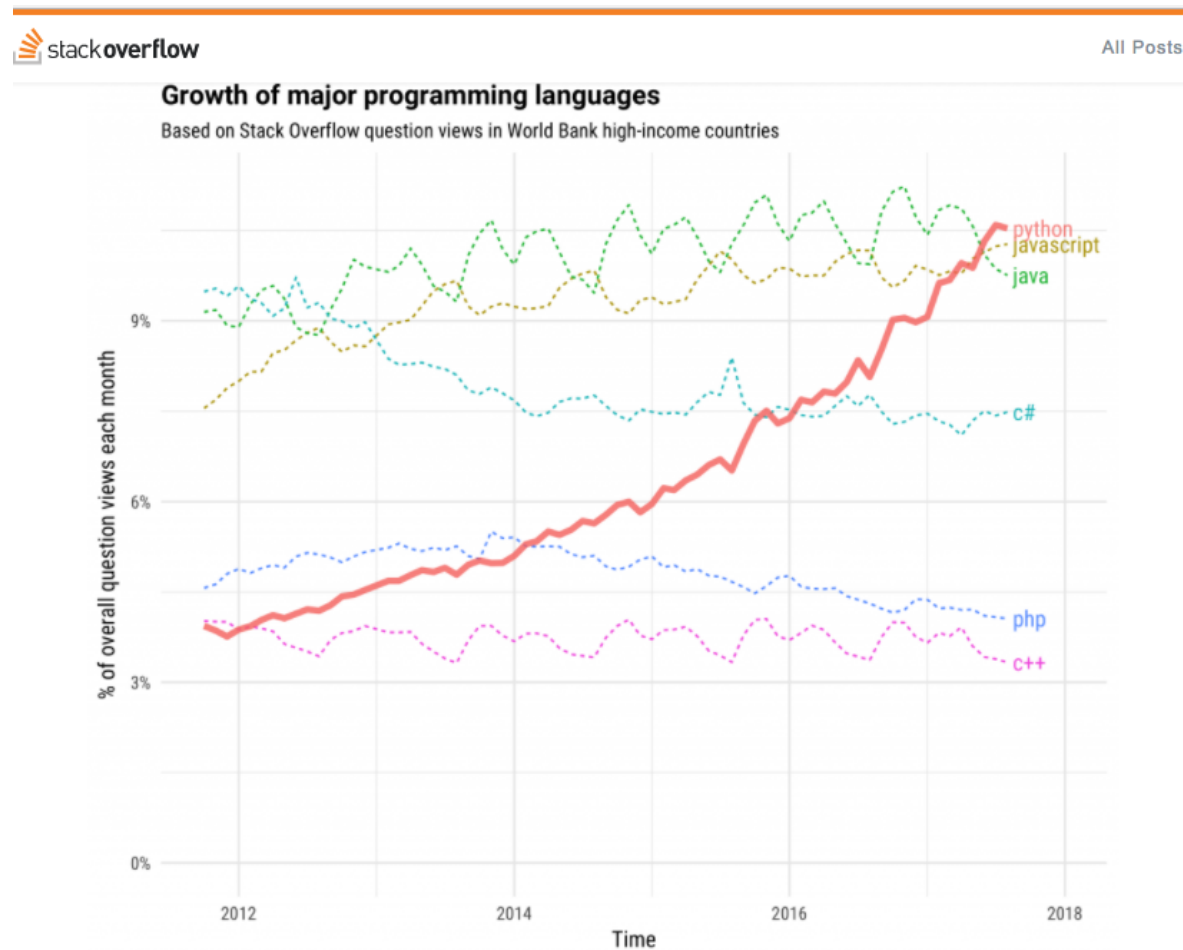
- ▶ Since Python is an interpreted and dynamically typed programming language, the execution of python code can be slow compared to compiled statically typed programming languages, such as C and Fortran.

To see how it compares to other languages:

<https://modelingguru.nasa.gov/docs/DOC-2783>

- ▶ Somewhat decentralized, with different environment, packages and documentation spread out at different places. Can make it harder to get started.

# Why Python?

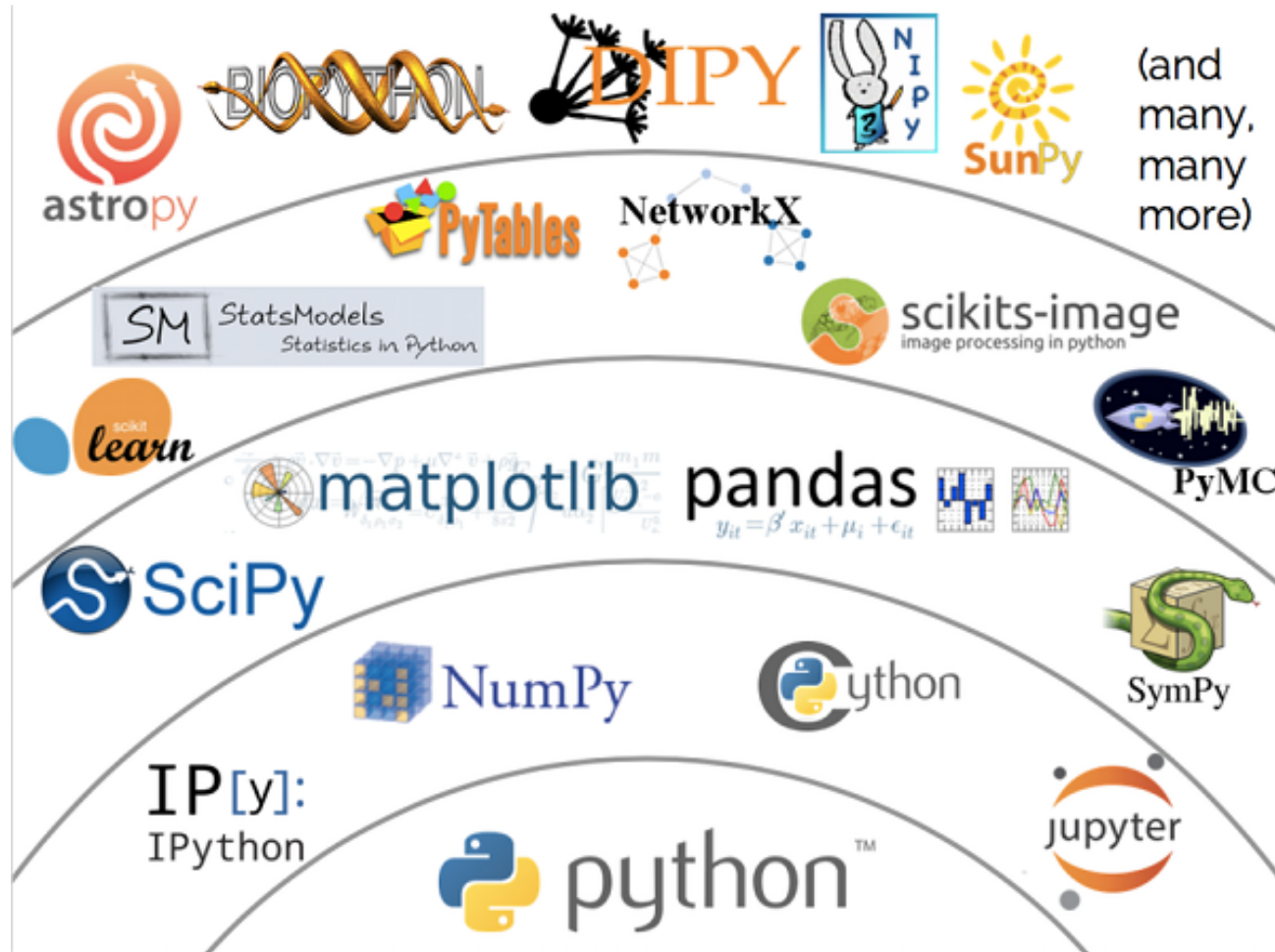


# Why Python?

- ▶ Python has a strong position in scientific computing:
  - ▶ Large community of users, easy to find help and documentation.
- ▶ Extensive ecosystem of scientific libraries and environments
  - ▶ numpy: <http://numpy.scipy.org> - Numerical Python
  - ▶ scipy: <http://www.scipy.org> - Scientific Python
  - ▶ matplotlib: <http://www.matplotlib.org> - graphics library
- ▶ Great performance due to close integration with time-tested and highly optimized codes written in C and Fortran:
  - ▶ blas, atlas blas, lapack, arpack, Intel MKL, ...
- ▶ Good support for
  - ▶ Parallel processing with processes and threads
  - ▶ Interprocess communication (MPI)
  - ▶ GPU computing (OpenCL and CUDA)
- ▶ Readily available and suitable for use on high-performance computing clusters.
- ▶ No license costs, no unnecessary use of research budget.



# The Python Ecosystem



# Python environments

Python is not only a programming language, but often also refers to the standard implementation of the interpreter (technically referred to as CPython) that actually runs the python code on a computer.

There are also many different environments through which the python interpreter can be used. Each environment has different advantages and is suitable for different workflows. One strength of python is that it is versatile and can be used in complementary ways, but it can be confusing for beginners so we will start with a brief survey of python environments that are useful for scientific computing.

# Python interpreter

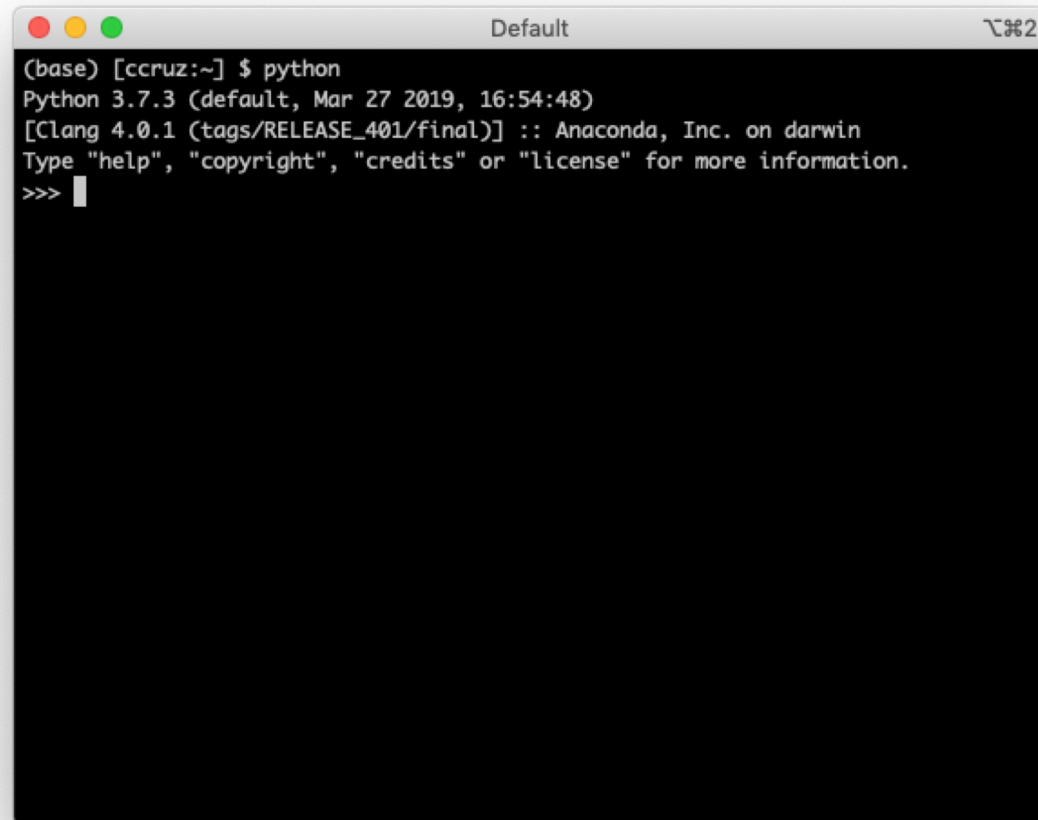
The standard way to use the Python programming language is to use the **Python interpreter** to run python code. The python interpreter is a program that reads and execute the python code in files passed to it as arguments. At the command prompt, the command *python* is used to invoke the Python interpreter.

For example, to run a file *my-program.py* that contains python code from the command prompt, use:

```
$ python my-program.py
```

# Python interpreter

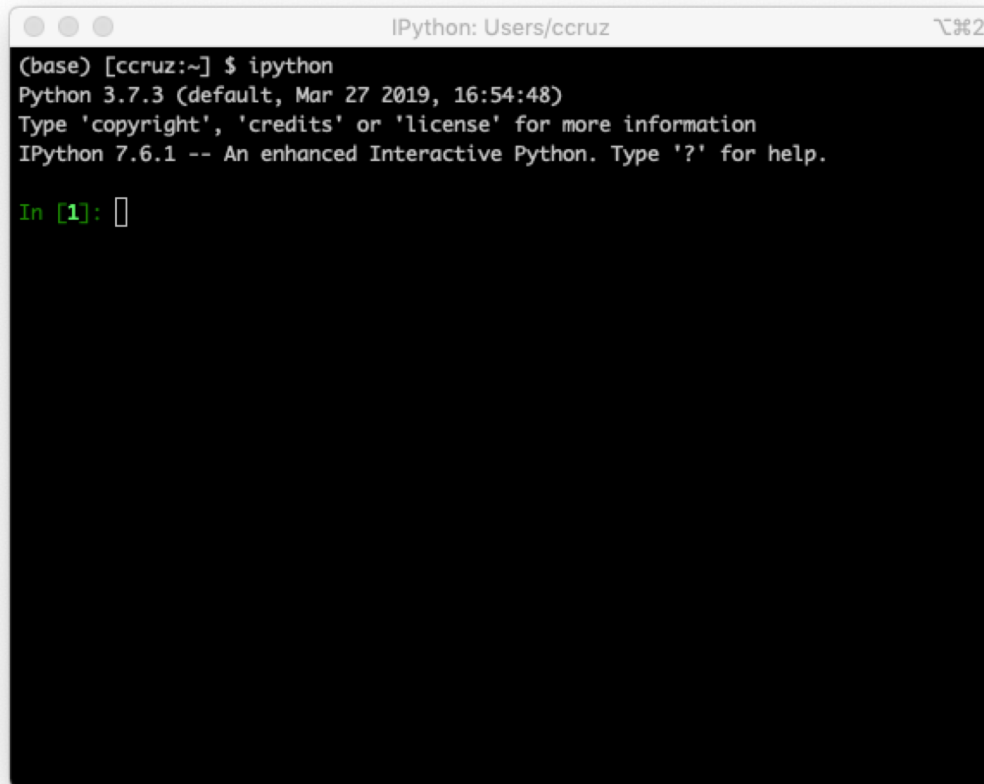
We can also start the interpreter by simply typing `python` at the command line, and interactively type python code into the interpreter.

A screenshot of a macOS terminal window titled "Default". The window shows the command `python` being executed. The output includes the Python version (3.7.3), the date and time (Mar 27 2019, 16:54:48), and the compiler information ([Clang 4.0.1]). It also displays the Anaconda logo and the text "Anaconda, Inc. on darwin". The prompt `>>>` is visible, indicating the interpreter is ready for input.

```
(base) [ccruz:~] $ python
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

# IPython

**IPython** is an interactive shell that addresses the limitation of the standard python interpreter, and it is a work-horse for scientific use of python. It provides an interactive prompt to the python interpreter with a greatly improved user-friendliness. To start, type *ipython* at the command prompt:



```
IPython: Users/ccruz  2
(base) [ccruz:~] $ ipython
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

# Jupyter Notebook

The **Jupyter Notebook** is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

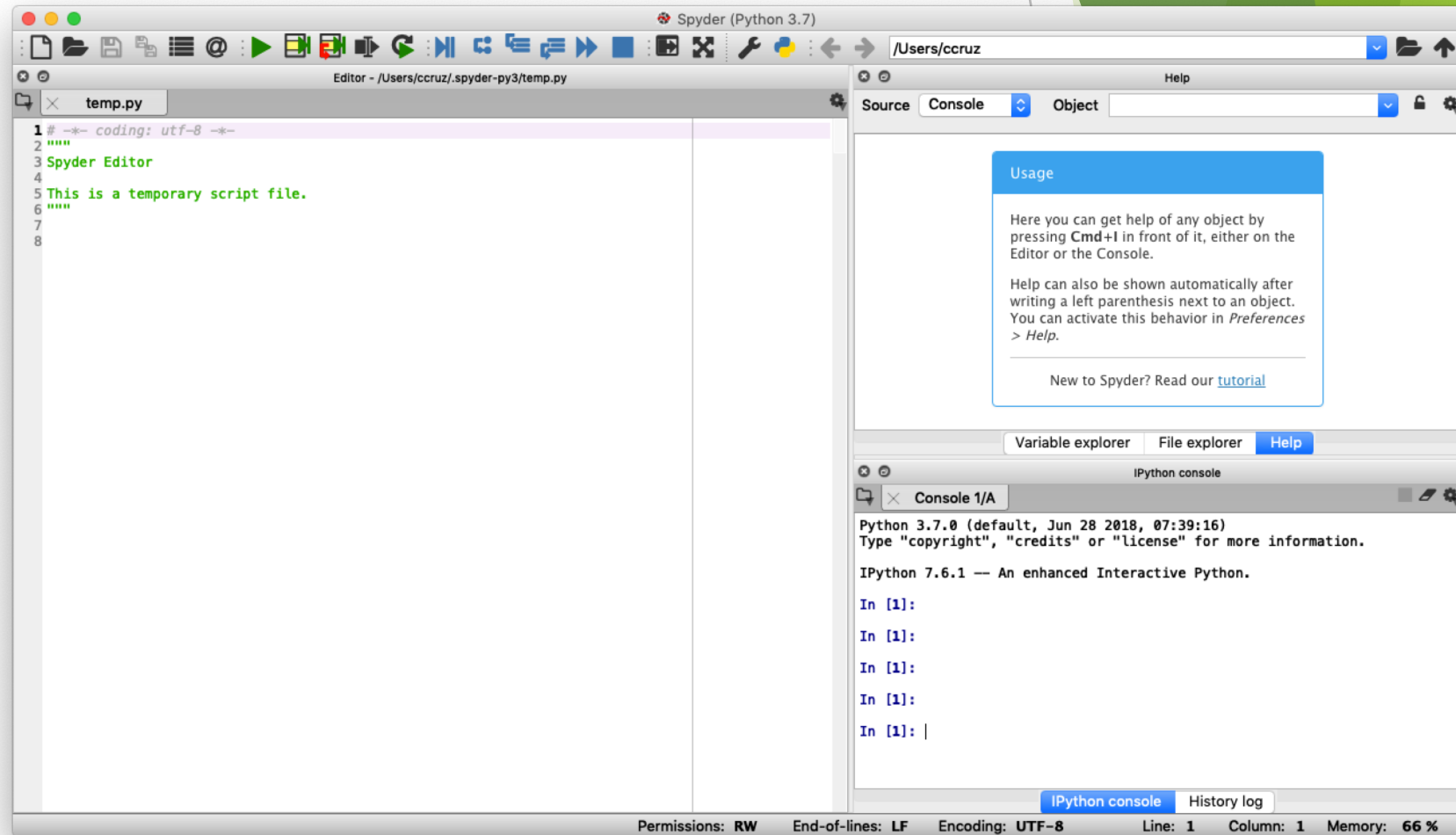
Although using a web browser as graphical interface, Jupyter notebooks run locally, from the same computer that run the browser. To start a new Jupyter notebook session, run the following command:

```
$ jupyter notebook
```

from a directory where you want the notebooks to be stored. This will open a new browser window (or a new tab in an existing window) with an index page where existing notebooks are shown and from which new notebooks can be created.

# Spyder

- is an **IDE** for scientific computing with python. It has the many advantages of a traditional IDE environment, for example that everything from code editing, execution and debugging is carried out in a single environment, and work on different calculations can be organized as projects in the IDE environment.



**IDE:** Integrated Developing Environment

# Summary of course objectives

- ▶ Computational **problem solving**: writing a "program" will become your "solution" for scientific problems.
- ▶ Basic Python proficiency: including experience with relevant libraries for scientific computing and visualization.
- ▶ Introduction to modeling and simulation concepts
- ▶ Experience working with some data sets



# Goal 0

- ▶ is to ensure that the computer you will be using for the python classes is fully set up and configured so you can get started. This involves getting packages installed, computers configured, everything updated to the correct versions, etc.

# Versions of Python

There are currently two versions of python: Python 2 and Python 3. Keep in mind that:

- ▶ Python 3 will very soon supersede Python 2
- ▶ Python 3 is not backward-compatible with Python 2.
- ▶ A lot of existing python code and packages has been written for Python 2, and it is still quite wide-spread.
- ▶ We will use Python 3 in this class.

# Installation of Python

The best way set-up an scientific Python environment is to use the cross-platform package manager `conda` from Continuum Analytics. To do so download and install Anaconda.

- ▶ Anaconda offers a bundled distribution of Python along with many utilities for use in the scientific/engineering disciplines. This software is available at <https://www.anaconda.com/download/>
- ▶ You will want Anaconda 3.7 (not 2.7). Windows users: How to determine if you need the 32- or 64- bit version:  
<https://support.microsoft.com/en-us/help/15056/windows-32-64-bit-faq>
- ▶ You may need to reboot after installation!

Note: If possible, please download and use the command-line installer designated for your operation system rather than any graphical installer

# Installation of Python

Once Anaconda is installed run "conda list" to see a list of the installed libraries:

```
$ conda list
```

Furthermore you can install additional libraries, e.g.:

```
$ conda install cython
```

# Git

- ▶ All lecture materials will be kept in a Git repository. To access this repository, you need the Git software available at <https://git-scm.com/download/>
- ▶ We will not be using a Git GUI Client, but rather using Git from within the Jupyter notebook or command line (Git Bash for Windows Users).

The lectures can be downloaded by issuing the following command (from your terminal or Git Bash - Windows Users):

- ▶ `git clone https://github.com/cds-230/fall-2019 cds-230`

# Git

You do not need to have a GitHub account to perform this and obtain the material online. If you would rather download a single snapshot .zip file of the lectures, you can do so via the web browser.

