Introduction to Computational Physics

PHYS 250 (Autumn 2025) – Lecture 1

David Miller

Department of Physics, Enrico Fermi Institute
Kavli Institute for Cosmological Physics, University of Chicago

September 30, 2025

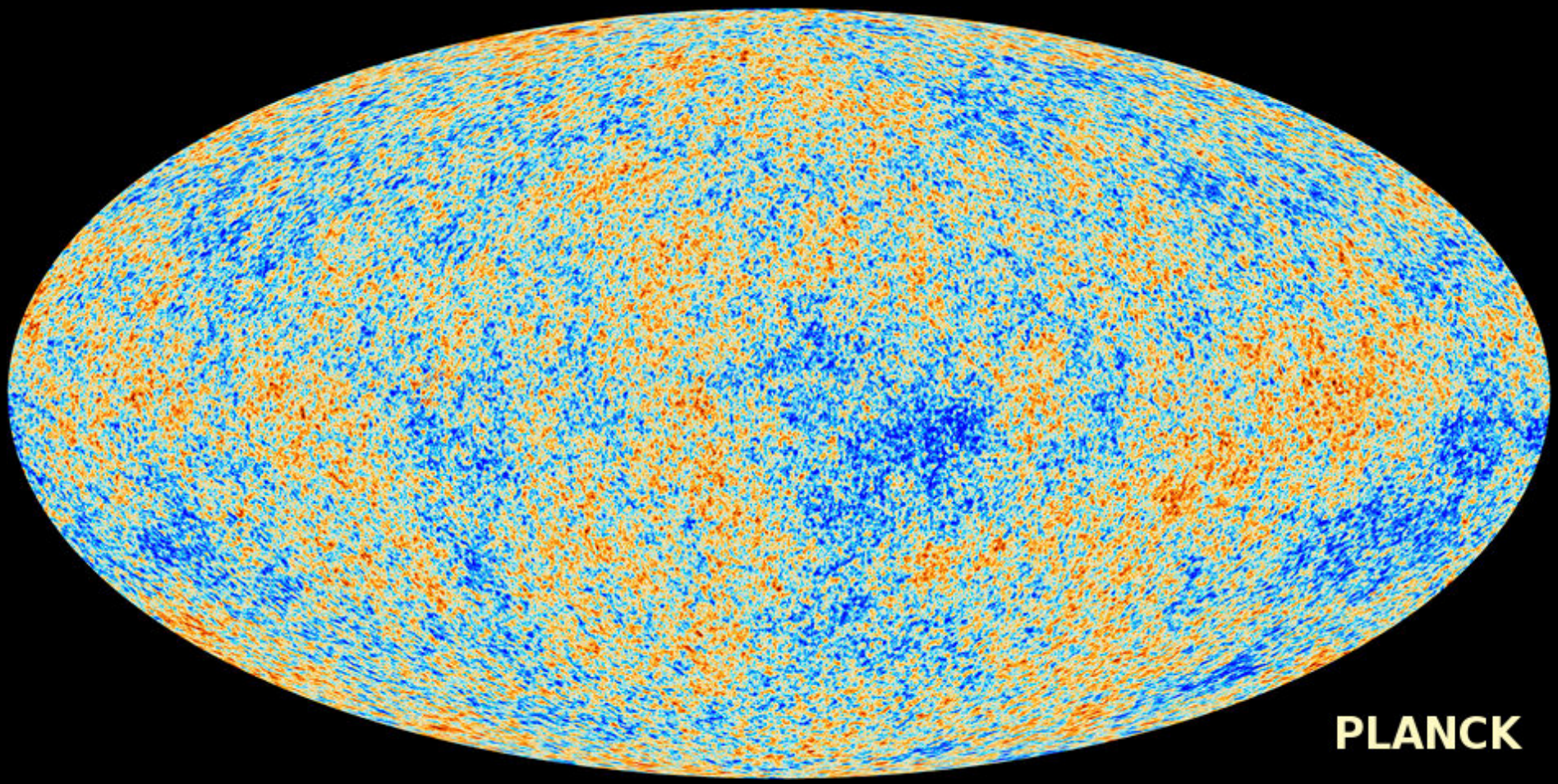
Outline

Computational Physics (PHYS 250)

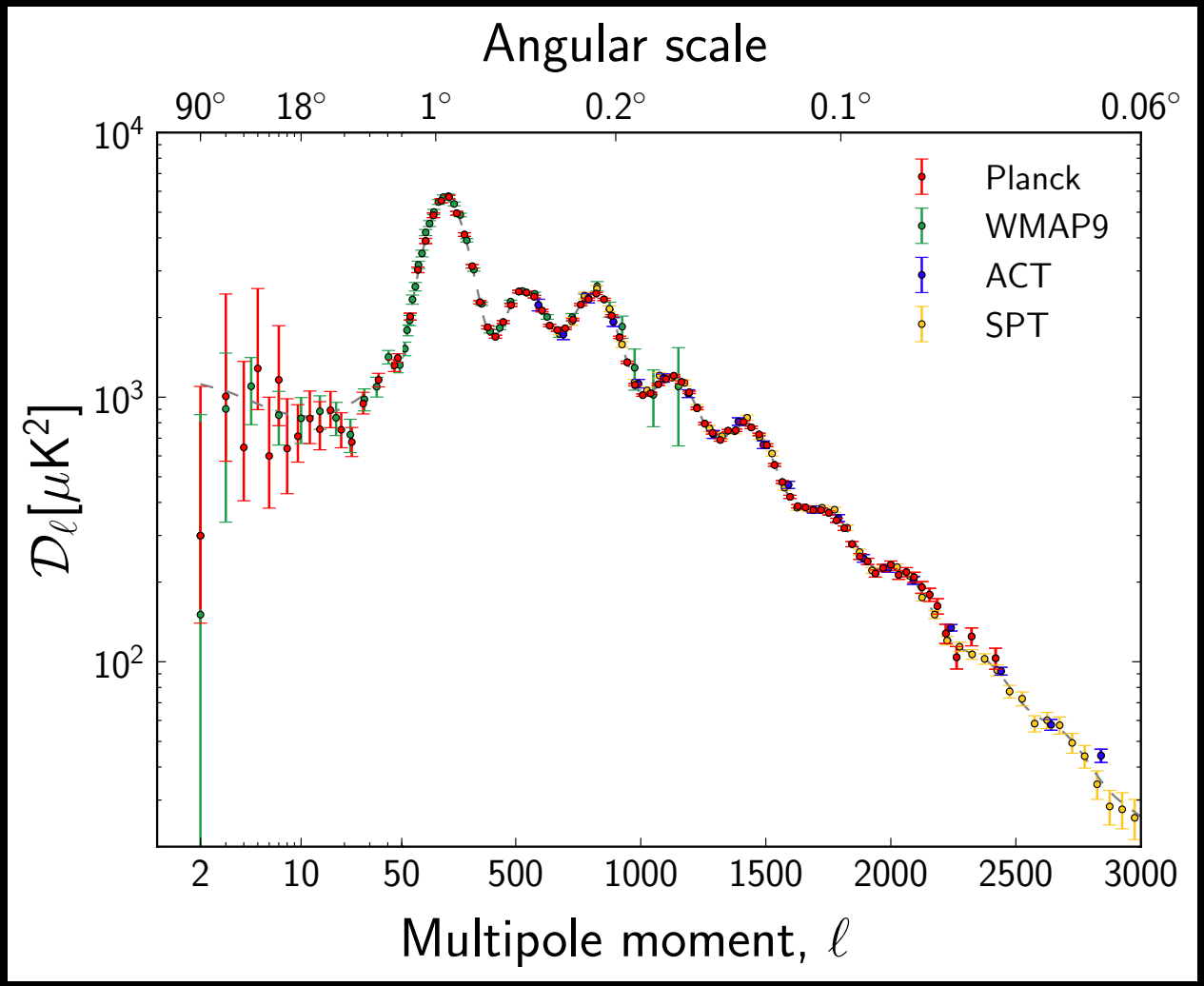
Course Description PHYS 250 (link to Course Catalog)

This course introduces the use of computers in the physical sciences. After an introduction to programming basics, we cover numerical solutions to fundamental types of problems, including cellular automatons, artificial neural networks, computer simulations of complex systems, and finite element analysis. Additional topics may include an introduction to graphical programming, with applications to data acquisition and device control.

There are an infinite number of paths that we might follow and still not deviate from this description. I therefore would like to lay out some of the principles that will guide me, and us, in how we navigate through those many possibilities.



The Nature of Science: Science is all about models. We look at something in real life and try to make a model of it. We can use this model to predict future (or new) events in real life. If the model doesn't agree with real data, we change the model. Repeat forever. [From Wired]



The Nature of Science: Science is all about models. We look at something in real life and try to make a model of it. We can use this model to predict future (or new) events in real life. If the model doesn't agree with real data, we change the model. Repeat forever. [From Wired]

Theory and Practice (or...Theory and Experiment)

We often divide the world of science (and perhaps even moreso, Physics) into two deeply related and intertwined but often distinct categories of activities and research:

- **Theoretical:** building models
- **Experimental:** testing models

We use computers to do *both* of these crucial activities!

In doing so, we often blur the distinction between the two, as well.

Overarching Learning Goals for this Quarter

- **Identify models** that benefit from or require **computational/numerical approaches** and tools to either develop and understand or to evalaute.
- Develop an **algorithmic approach** to addressing those problems computationally.
- Familiarity with **common computational models and algorithms** and numerical approaches to typical problems.
- Use **modern, high-level programming languages** to implement the computational algorithms.
- Use modern software tools for **developing, preserving, disseminating, and expanding** on those solutions.
- **Apply computational tools** to contemporary physics questions.

The role of the physicist

"The computer is incredibly fast, accurate, and stupid. Man is incredibly slow, inaccurate, and brilliant. The marriage of the two is a force beyond calculation."

→ Maybe this is a quote from Albert Einstein, maybe it is from Leo Cherne...I'm not sure, but it's certainly true.

Some of the specific problems that we will tackle

There are literally hundreds of texts, and even more webpages and software repos, for computational physics, methods, numerical recipes.
$\rightarrow$ **I have chosen a subset of ubiquitous problems and which are the "bread and butter" of nearly all domains in physics and beyond**.

- Programming basics and Software design concepts
- Basic visualization
- Random numbers, errors
- Ising model, Metropolis algo
- Minimization
- Monte Carlo method
- Ordinary differential equations
- Partial differential equations
- Fourier transforms
- Data analysis techniques
- Neural networks

I hope you walk out of this class able to effectively engage with most of the computational physics issues and tools in a modern research group.

Outline

Algorithmic and process-based thinking

You probably already learned in your first year courses that certain approaches to problem solving in physics are important to efficiently identifying solutions and paths through a particular problem.

Before we get into any code, let's discuss the **thought process behind developing a computational model or approach to problem solving**.

Thinking like a machine

From the Kinder & Nelson text (see list in a few slides)

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Thinking like a machine

From the Kinder & Nelson text (see list in a few slides)

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Thinking like a machine

From the Kinder & Nelson text (see list in a few slides)

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Thinking like a machine

From the Kinder & Nelson text (see list in a few slides)

Human-readable (high level) instructions to start a car

```
Put the key in the ignition
Turn the key until the engine starts,
    then let go
Push the button on the shifter and move it
    to REVERSE
...
```

Bug!!!

```
Press down the left pedal # Needed for some cars
```

Machine-readable (low level) instructions

```
Grab the wide end of the key
Insert pointed end of key into slot on lower
    right side of steering column
Rotate key clockwise about long axis when viewed
    toward the pointed end
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                    # Input
calculate area of circle       # Numeric
print area                     # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")  # Input
pi     = 3.141593                 # Set constant
area   = pi  * radius^2           # Algorithm
print("Area = " + area)           # Output
```

Logical vs. algorithmic instructions

Let's say you want to write an algorithm to calculate the area of a circle.

1. Write "pseudocode" or the **logical instructions** for the algorithm
2. Extend pseudocode with **parameters and input/output**
3. Specify algorithm **explicitly**

Pseudocode algorithm for area calculation

```
calculate area of circle  # Do this computer!
```

Better algorithm for area calculation

```
read radius                  # Input
calculate area of circle     # Numeric
print area                   # Output
```

Actual algorithm for area calculation

```
radius = input("Specify radius")   # Input
pi     = 3.141593                  # Set constant
area   = pi  * radius^2            # Algorithm
print("Area = " + area)            # Output
```

Translate algorithmic thinking into a development process

We can adapt to the need for algorithmic thinking by adopting a process for developing algorithms, computational approaches, and software generally.

Step 1: write the algorithm down on paper
Step 2: think
- *If:* you don't understand everything
- *Then:* `goto` Step 1
- *Else:* `continue`

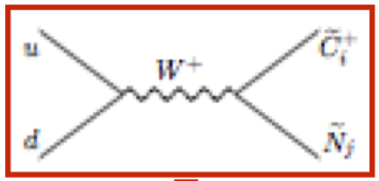
Step 3: write pseudocode
Step 4: think
- *If:* you don't understand everything
- *Then:* `goto` Step 3
- *Else:* `continue`

Step 5: write actual code
Step 6: test code with unit tests
- *If:* code does not unit checks
- *Then:* `goto` Step 5
- *Else:* `continue`

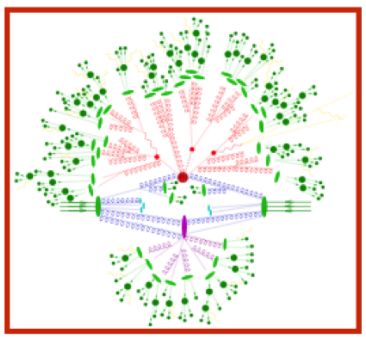
Step 7: publish!

Alas, sometimes, it's a bit more complicated than that...

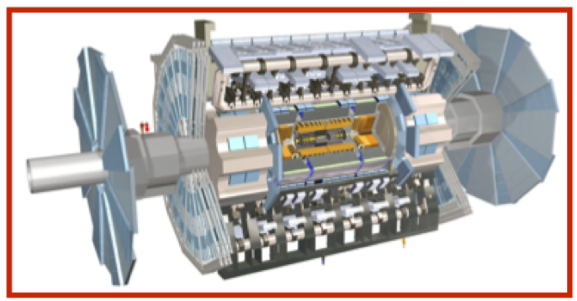
Simulation of hard-scatter for signal + SM background



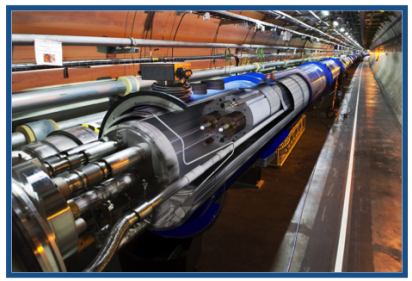
Simulation of "soft physics" (shower + hadronization)



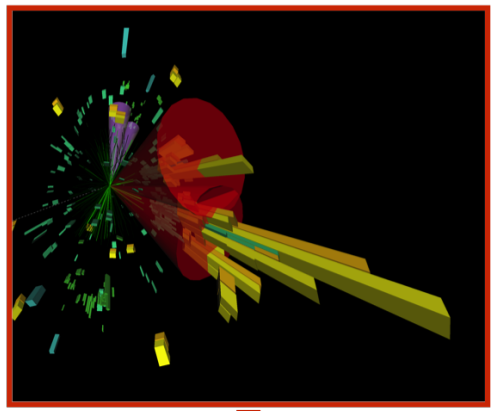
Detector simulation



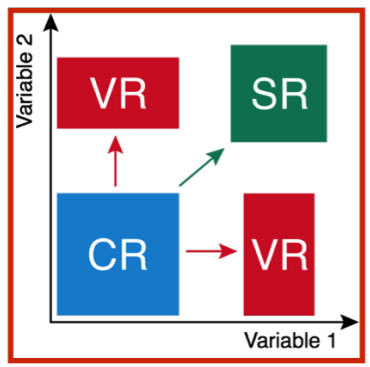
LHC real data



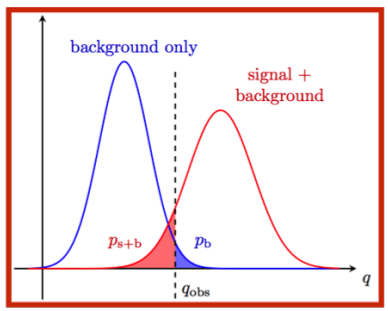
Reconstruction of physical observables



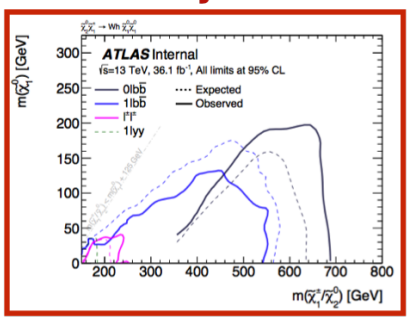
Event selection and background estimation



Hypothesis testing



Discovery or exclusion



Outline

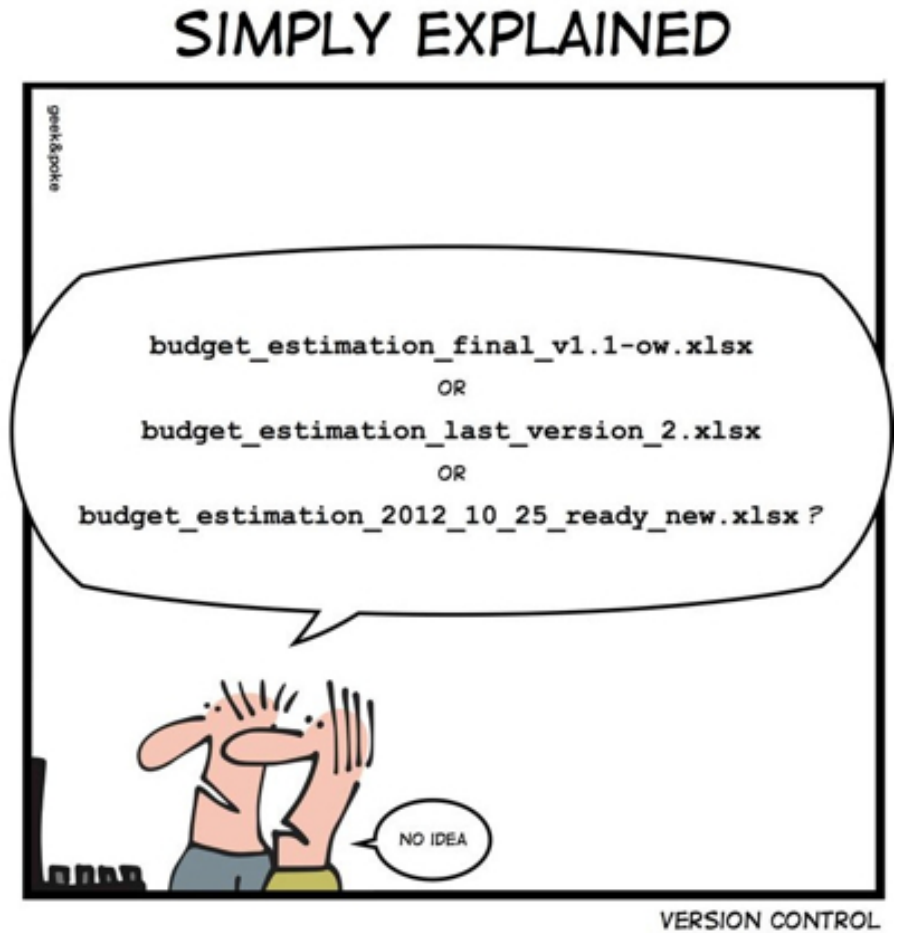
Version control

- The most important message of this slide is simple...**Use a software version control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

Version control

- The most important message of this slide is simple...**Use a software version control system for all of your code**
  - And that means now...not tomorrow or next week
  - Because if you wait until you need it, it will be too late

GitHub



A brief history of version control

- **The first version control systems were designed to be used on large systems where everyone logged into the same machine**
  - They tracked code on the same filesystem where it lived (e.g., in a subdirectory)
  - SCCS and RCS are examples
- **Then client-server systems were developed, so that developers could work on their own machines**
  - Checking code into a central server to share and collaborate
  - CVS and SVN are examples
- **More recently distributed version control systems have arisen**
  - These are decentralised, so everyone has a complete copy of the repository
  - Gives a lot of freedom to developers to share and merge as they like, so liked very much by the open source community
  - git, mercurial and bit keeper are examples

`git`, `GitHub`, & `GitLab`

https://git-scm.com, *https://github.com*, *https://about.gitlab.com*

- **`git` is the most popular open source version control system**
  - can host huge projects (Linux Kernel, LHC experiment software, etc)
  - scales very well and it's extremely fast and powerful
  - very flexible (= complex)
- **Distributed version control systems (`git`) are great, but they're made even better by using a social coding site (`GitHub` or `GitLab`)**
- **These sites allow developers:**
  - browse code easily
  - compare different versions
  - take copies (a.k.a. fork)
  - offer patches back to upstream repositories
    - And discuss and review these patches before acceptance
  - even build websites
- **The best known social coding site is GitHub, but there are others, e.g. BitBucket and GitLab**
  - Familiarity with **`git`/`GitHub`/`GitLab`** will serve you well, trust me

ATLAS Experiment analysis software package on **GitHub**

ATLAS Run II analysis framework for AnalysisTop and AnalysisBase for proton-smashing physics https://ucatlas.github.io/xAODAnaHelp...

physics-analysis | high-energy-physics | analysis-framework

1,911 commits | **16** branches | **130** releases | **40** contributors

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

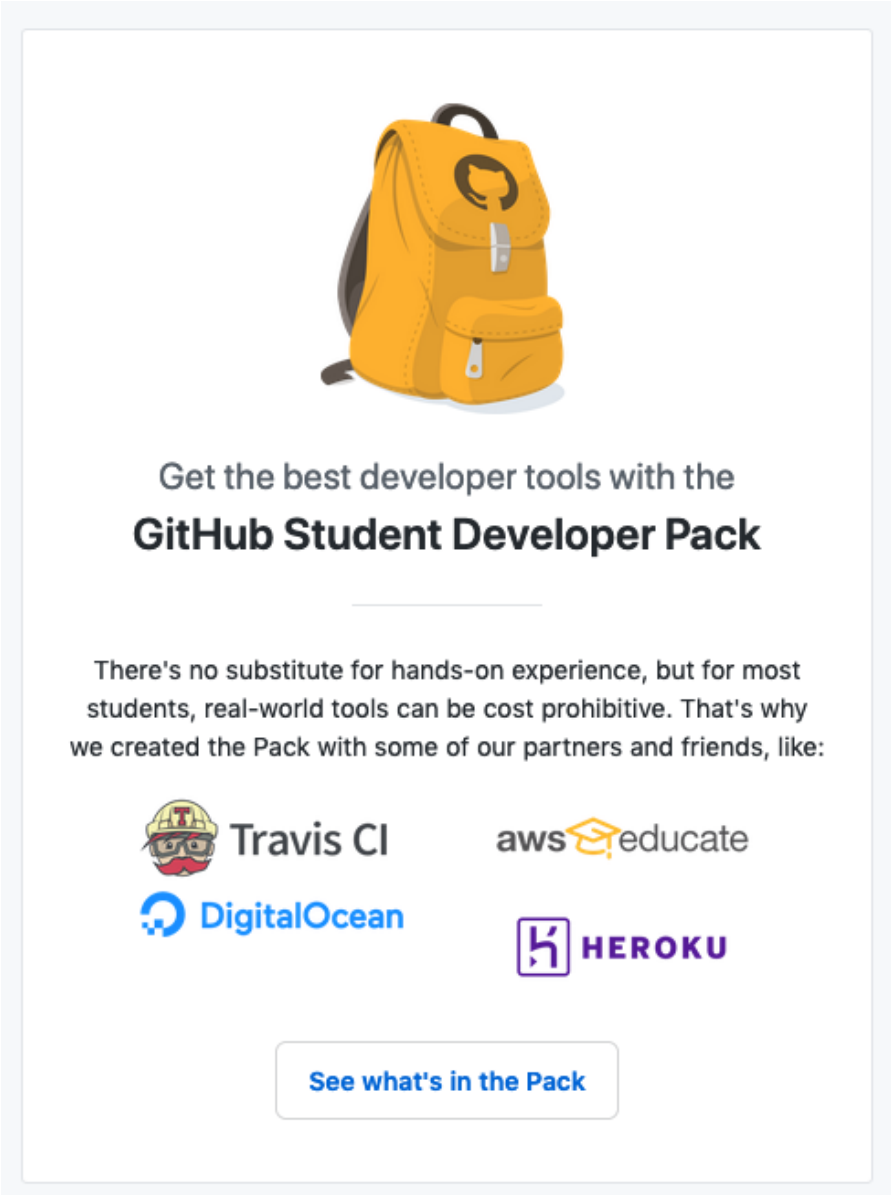
kratsg Custom jvt cuts need to use detector-eta (#1250) — Latest commit ec9c86b 17 hours ago

Root	Custom jvt cuts need to use detector-eta (#1250)	17 hours ago
ci	deploy to multiple repos instead (#1126)	9 months ago
cmake	Add bash command line completion for xAH_run.py in CMake (#1038)	a year ago
data	A few more changes to autoconfigure PRW (#1223)	3 months ago
docs	Add TauJet track matching (#1242)	18 days ago
python	remove is_release20. resolves #1243 (#1246)	4 days ago
scripts	remove is_release20. resolves #1243 (#1246)	4 days ago
xAODAnaHelpers	Keep OR of triggers together for extra chains (#1186)	17 days ago
.gitignore	large-R area info (#736)	2 years ago
.travis.yml	Fixing compilation warning for deprecated jet jvt efficiency tool hea... (	2 months ago
CMakeLists.txt	Fixing compilation warning for deprecated jet jvt efficiency tool hea... (	2 months ago
CONTRIBUTING.md	Add analysis top (#1125)	9 months ago
README.md	Remove RootCore functionality (#1124)	9 months ago

GitHub Student Developer Pack

https://education.github.com/pack

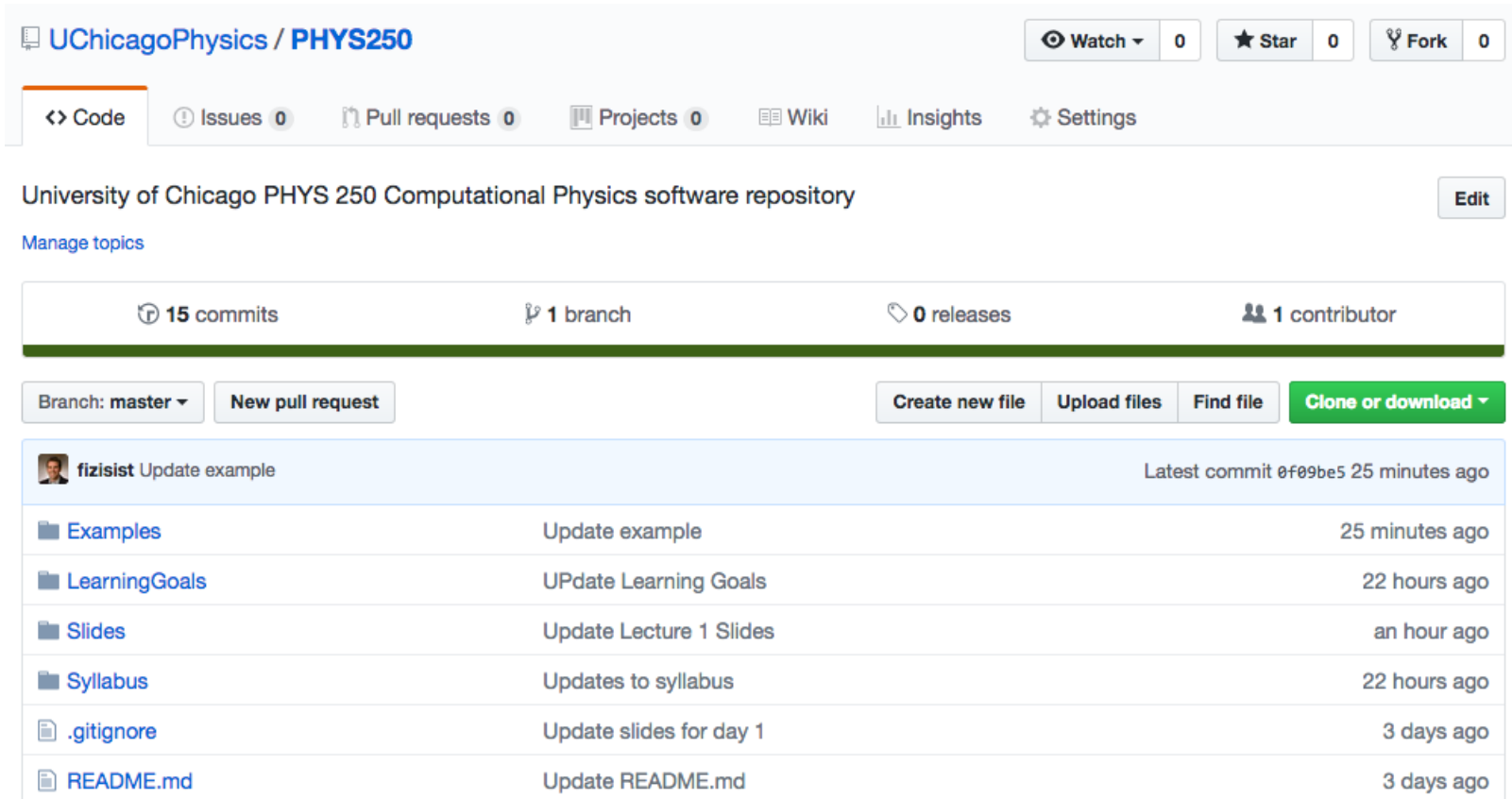
You get free unlimited private repositories as a student!



PHYS 250 **GitHub**

https://github.com/UChicagoPhysics/PHYS250

Course materials are hosted in the **GitHub** `UChicagoPhysics` repository



- Slides (e.g. *these!*), syllabi, learning goals, and code examples
- Stable versions will be cross-posted to Canvas as well.
- Homework submission will be done via **GitHub** (*instructions to come*)

GitHub *Classroom*

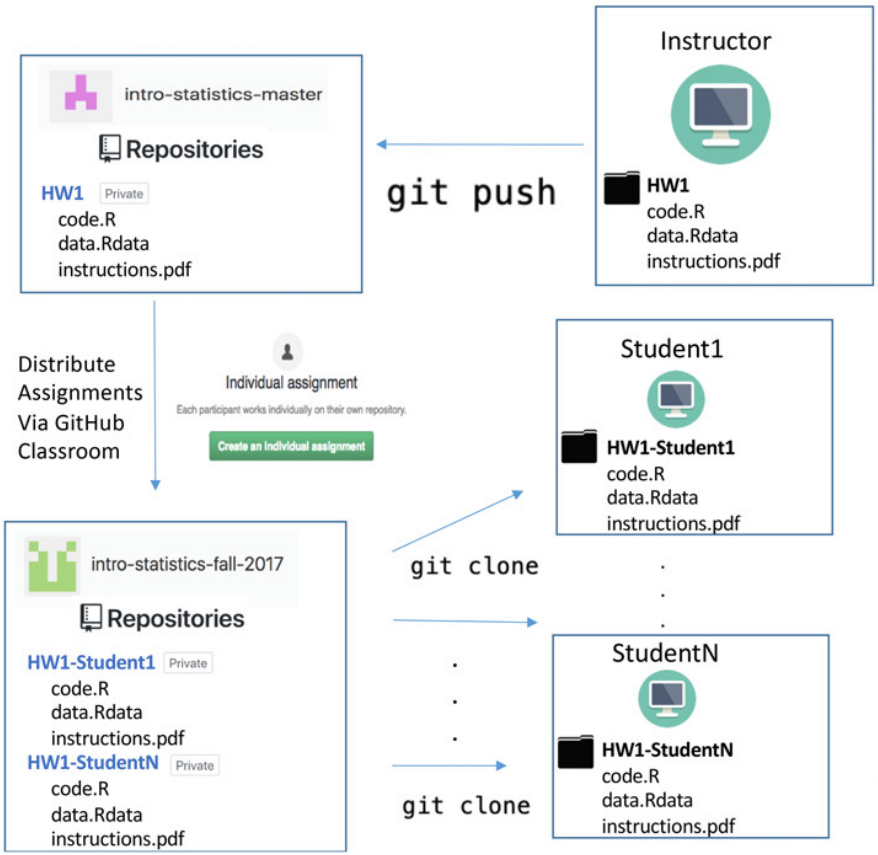
https://classroom.github.com/

Course materials are hosted in the **GitHub** `UChicagoPhysics` repository, but the assignments will be distributed via **GitHub** Classroom

- I create a repository with the assignment
- With the click of a button, that repository gets distributed to all of you as your own private repository
- The deadline is set, and the lass commit at that point is "submitted" to me as your homework

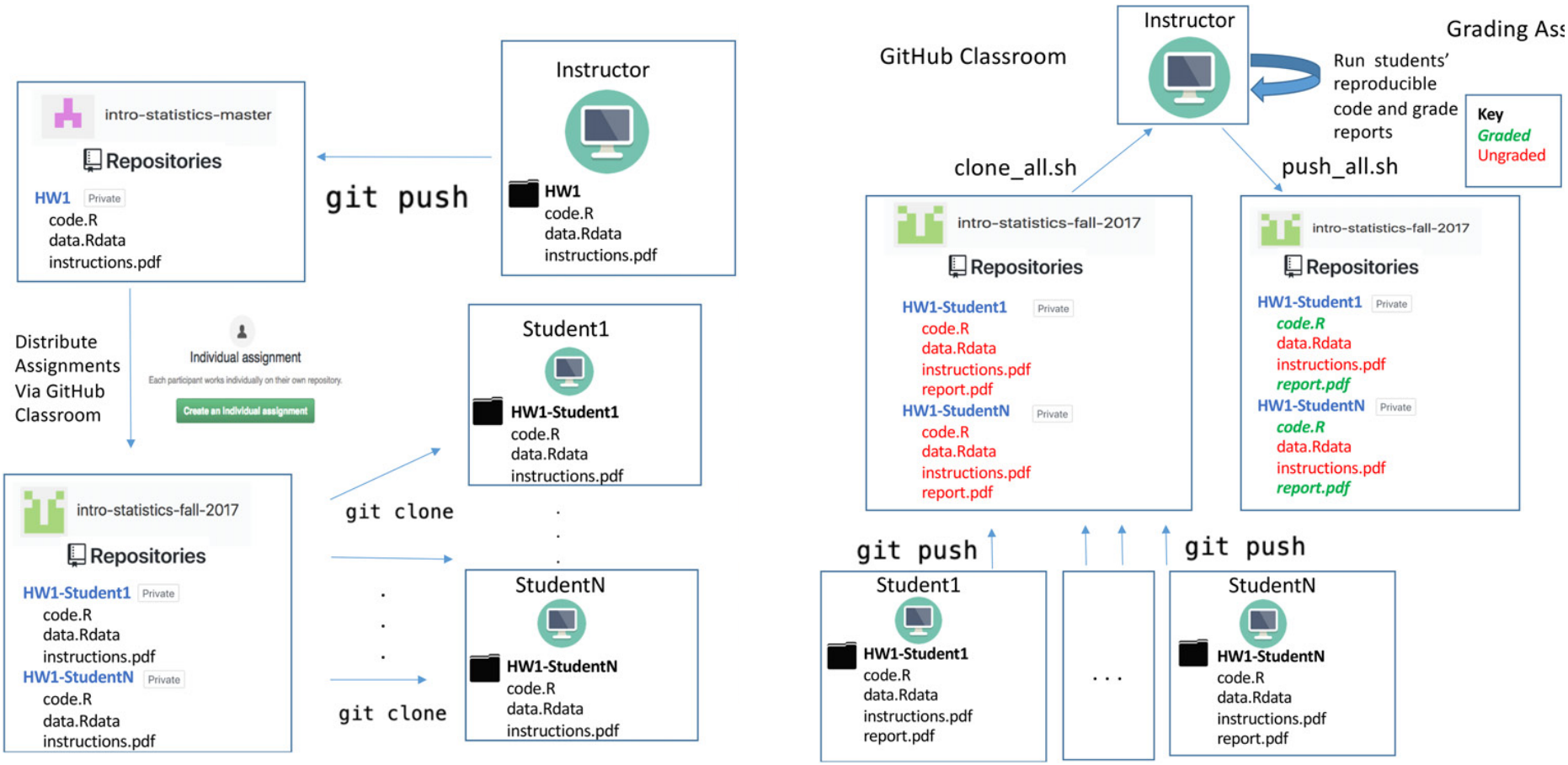
GitHub Classroom

https://classroom.github.com/



GitHub Classroom

https://classroom.github.com/



Source: arXiv:1811.02021

You need a GitHub account, so please sign up ASAP!

Operating systems and platforms

- Operating systems (like Windows or Mac OS)
- We will generally be using UNIX/LINUX (an offshoot of UNIX)
  - Invented by Linus Torvalds in 1991
  - Mac OS X is built upon LINUX
- I have arranged for a "minicourse" from CSIL on Friday (info in later slides)
  - Also have an EdX course (free) from the creator here:
  - `https://www.edx.org/course/introduction-linux-linuxfoundationx-lfs101x-0#!`
- Linux is completely open source
  - you can modify it at your will and debuggers are also free
- Predominantly command line tools

Linux "shell"

- We will be using an interface to Linux called a "shell"
- It is a command-line interpreter : you type, it executes
- Two major options are `bash` (as in, smash) and `csh` (like "sea shell", modern version is "tcsh", "tea sea shell")
  - Only real difference: environment variables syntax
  - `bash: export X=value`
  - `csh: setenv X value`

Shell basics

Listing directory contents : `ls`, like "list"

```
> ls
Examples/ LearningGoals/ README.md Slides/
Syllabus/ global.sty
```

Copy: `cp`

```
> cp stuff.txt stuff1.txt
```

Where am I?: `pwd`, `cd`

```
> pwd
/ComputationalPhysics/PHYS250/PHYS250-Fall2018
> ls
Examples/ LearningGoals/ README.md Slides/ Syllabus/
> cd Examples/
> ls
HelloGaussian.ipynb
HelloGaussian.py
Introduction_to_Jupyter_Notebooks_and_Python.ipynb
```

Python as our programming language

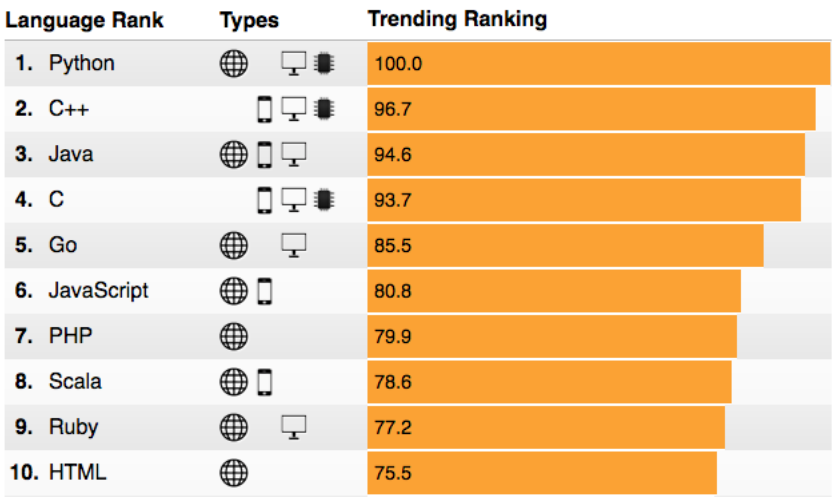
From www.python.org

Python is a dynamic object-oriented programming language that can be used for many kinds of software development. It offers strong support for integration with other languages and tools, comes with extensive standard libraries, and can be learned in a few days.

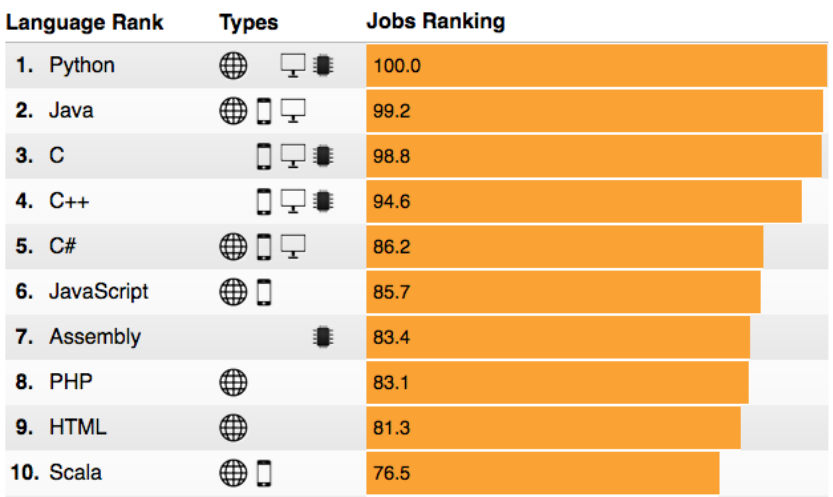
And in my own words:

- It is **ubiquitous, flexible, useful, and *relatively* easy**

From IEEE Spectrum rankings (July 2018):



Language Rank	Types	Trending Ranking
1. Python		100.0
2. C++		96.7
3. Java		94.6
4. C		93.7
5. Go		85.5
6. JavaScript		80.8
7. PHP		79.9
8. Scala		78.6
9. Ruby		77.2
10. HTML		75.5



Language Rank	Types	Jobs Ranking
1. Python		100.0
2. Java		99.2
3. C		98.8
4. C++		94.6
5. C#		86.2
6. JavaScript		85.7
7. Assembly		83.4
8. PHP		83.1
9. HTML		81.3
10. Scala		76.5

Hello world!

Interactive in the python interpreter

```
python
>>> print "hello world"
hello
>>> CTRL-d  # to exit python
```

From a script (containing the above print line):

```
python helloworld.py
```

Self-running script:

```
#!/usr/bin/env python
# This script prints hello to the screen
print "hello world"
```

```
chmod +x helloworld.py
./helloworld.py
hello world
```

Python syntax

- Everything after a hash (`#`) is a comment (until the end of the line)
- A statement ends at the end of the line
- Multiple statements on the same line are separated by a semicolon (`;`)
- A block of code is defined by its equal indentation
  - Don't use tabs, always use spaces! (tab = 8 spaces)
- A backslash at the end of a line joins it with the next line - like in shell scripts
- An unclosed `()`, `[]` or `{}` pair also continues to the next line(s) until the closing `)`, `]` or `}`

Numerical types

- `int`: integer (at least 32bit)
- `float`: floating point, like C++ double
- complex
- `str`: string (constant), like C++ `const char*`
- `bool`: True or False
- `list`: a vector, like C++ `std::vector`
- `tuple`: constant list
- `dict`: a map, like C++ `std::map`

You can ask an object for it's type with the '`type`' function:

```
>>> type("hello")
<type 'str'>
>>> type("hello").__name__
'str'
```

Operators

- `+, -, *, /, **`: addition, subtraction, multiplication, division, power
- `+=, -=, *=, /=`: operator and assignment in one go (as C++)
- `%`: modulus (int), format (str)
- `or, and`: logical `OR` and `AND`
- `not`: logical negation
- `<, <=, >, >=`: comparison
- `==, !=`: equality, not equality
- `is, is not`: object identity (pointer comparison)
- `in, not in`: membership test
- `|, ^, &, ~, «, »`: bitwise operators, as in C++
- `X < Y < Z`: True if `Y` is in between `X` and `Z`

Lists (I)

In my opinion, python's great advantage is **list comprehension**.

List basics

```
v = []    # empty list
v = list()  # empty list
v = [ 1, 2, 4, 5 ]  ; v = [ 'a', 'b', 'c' ]
v = range(4,10,2) # results in [ 4,6,8 ]
v = [ 4, 2.5, 'Hi', [ 1,3,5 ] ]  # can mix types
```

Append elements

```
>>> v.append( 70 )
>>> print v
```

Concatenation

```
>>> v += [ 'some', 'more', 'elements' ]
>>> v  # shows the object
```

Removal of elements

```
>>> v.remove(2.5)
>>> del v[0]
```

Lists (II)

Element acces read/write

```
>>> v[0]
'hi'
>>> v[0] = 'hey'
>>> v[-1] # last element. Negative = count from the en
>>> v[1:3] # subrange by index (start index, one-beyon
```

Test if an element is in a list (or not)

```
>>> if 4 in v:
...     print "Found it"
Found it
>>> if 200 not in v:
...     print "Not found"
Not found
```

`for` and `while` loops

The `for` statement iterates through a collection, iterable object or generator function.

The `while` statement merely loops until a condition is `False`.

Iterate over list

```
fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
```

Iterate using built-in `range` function

```
for x in range(0, 3):
  print "We're on time %d" % (x)
```

Iterate until a condition is met

```
count = 0
while count < 5:
  print(count)
  count += 1   # Same as: count = count + 1
```

Putting lists and loops together is amazing (and complex)

Filter one list into another (the "old" way)

```
newlist = []
for i in oldlist:
    if filter(i):
        newlist.append(function(i))
```

List comprehension (the "pythonic" way)

```
newlist = [function(i) for i in oldlist if filter(i)]
```

where `filter` and `function` just perform "some" operation on the list elements. Basically, the syntax is:

```
[ expression for item in list if conditional ]
```

and this replaces:

```
for item in list:
  if conditional:
    expression
```

Useful list comprehension

Filter one list into another (the "old" way)

```
>>> v = [ x**2 for x in range(10) if x % 3 == 0 ]
>>> v
[0, 9, 36, 81]
```

List comprehension (the "pythonic" way)

```
newlist = [function(i) for i in oldlist if filter(i)]
```

where `filter` and `function` just perform "some" operation on the list elements. Basically, the syntax is:

```
[ expression for item in list if conditional ]
```

and this replaces:

```
for item in list:
  if conditional:
    expression
```

Hello Gaussian!

Basic but useful code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()
```

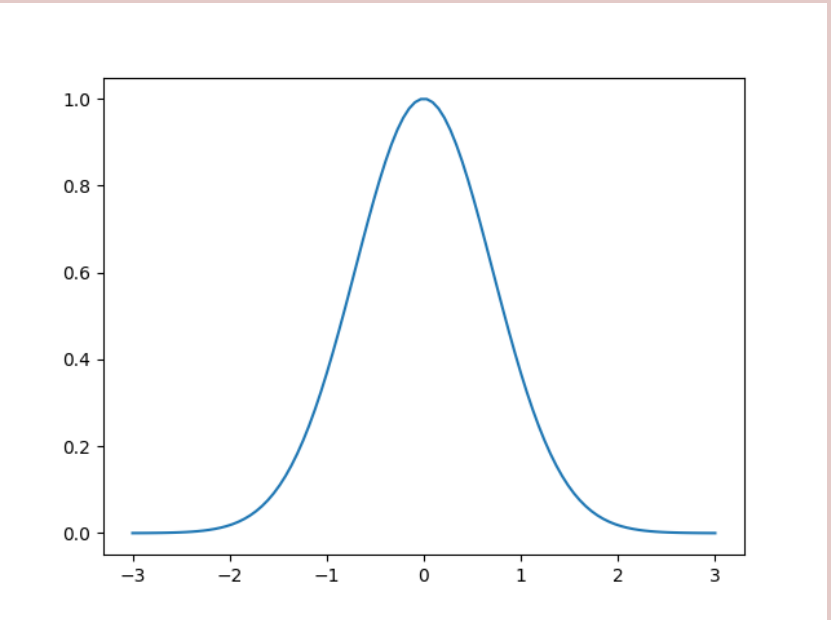
Hello Gaussian!

Basic but useful code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()
```



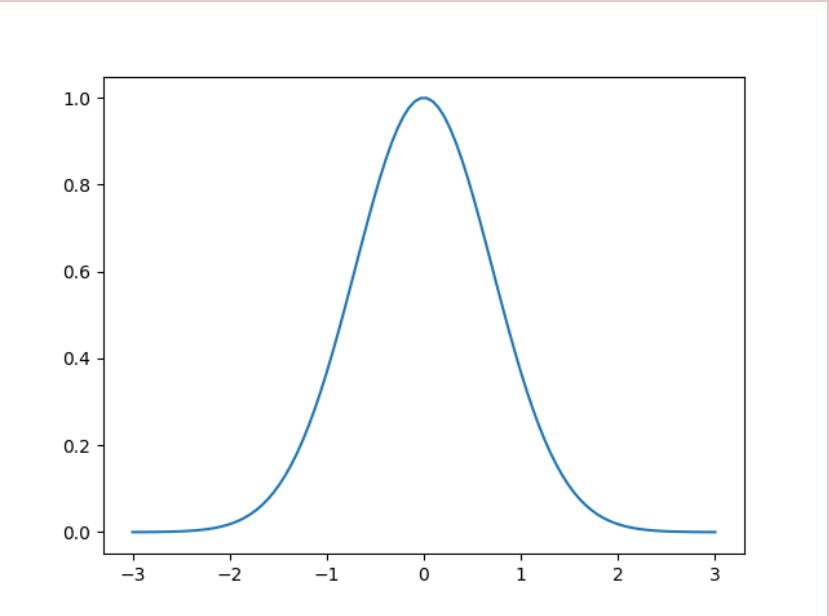
Hello Gaussian!

Basic but useful code example

```
import numpy as np
import matplotlib.pyplot as plt

def p(x):
    return np.exp(-x**2)

#let's plot it
x = np.linspace(-3,3,100)
y = p(x)
plt.plot(x,y)
plt.show()
```



But what about that `linspace` thingy? Google it! (numpy docs)

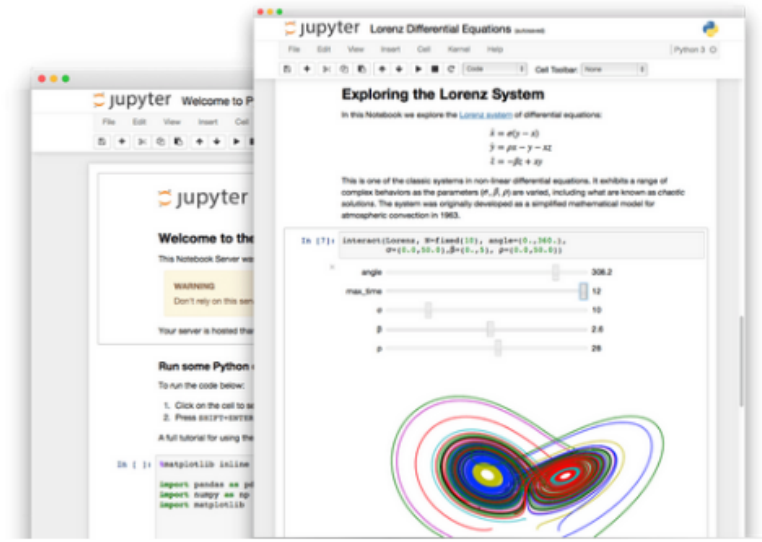
`numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None)`

"Returns num evenly spaced samples, calculated over the interval [start, stop]."

Jupyter notebooks

Interactive, web-based, integrated code and documentation environment

We will be following-up with more technical practice with python, but I want to introduce you to the resources that we'll be using this quarter for many of our examples and projects: Jupyter notebooks.



The 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. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Try it in your browser | Install the Notebook

</> Language of choice

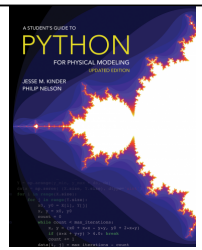
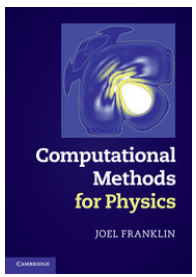
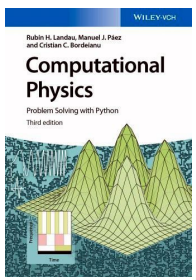
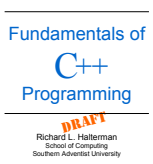
Share notebooks

Interactive output

Spark Big data integration

Outline

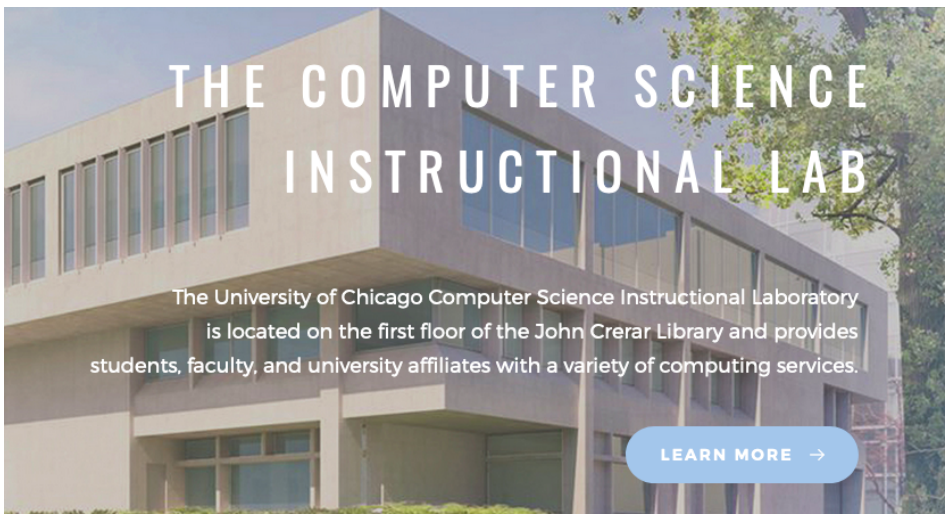
Textbooks that I suggest

	Author	Textbook	Comments
	Kinder & Nelson (link)	A Student's Guide to Python for Physical Modeling	Very good for python examples, not great for physics principles
	Franklin (link)	Computational Methods for Physics	Amazing for physical principles, no python
	Landau, Paez, Bordeianu (link)	Computational Physics, Problem Solving with Python	Great for both physics and python
	Halterman (link)	Fundamentals of C++ Programming	Great for physical principles, no python

Computer Science Instructional Lab (CSIL)

https://csil.cs.uchicago.edu/index.html

- **Location:** first floor of Crerar (back right by stairs)
- **PHYS 250 Lab Hours:**
  - Will we need this?
- **CSIL Minicourses**
  - Linux and **`git`** minicourses available through CSIL
  - additional tutorials are possible
  - https://csil.cs.uchicago.edu/minicourses



Research Computing Center (RCC)

https://rcc.uchicago.edu/

In addition to the short "minicourse" from CSIL, There is the possibility of holding longer tutorials (2-3 hrs) conducted by the RCC or CSIL for the following topics:

- Introduction to Linux and the RCC
- Introduction to Python

These are much more in-depth and would be a jumpstart to the quarter if you are interested. Please make sure to fill out the survey if you are interested:

- https://forms.gle/442M6ExcCvHmysr97