P40: INTRODUCTION

Eric Prebys

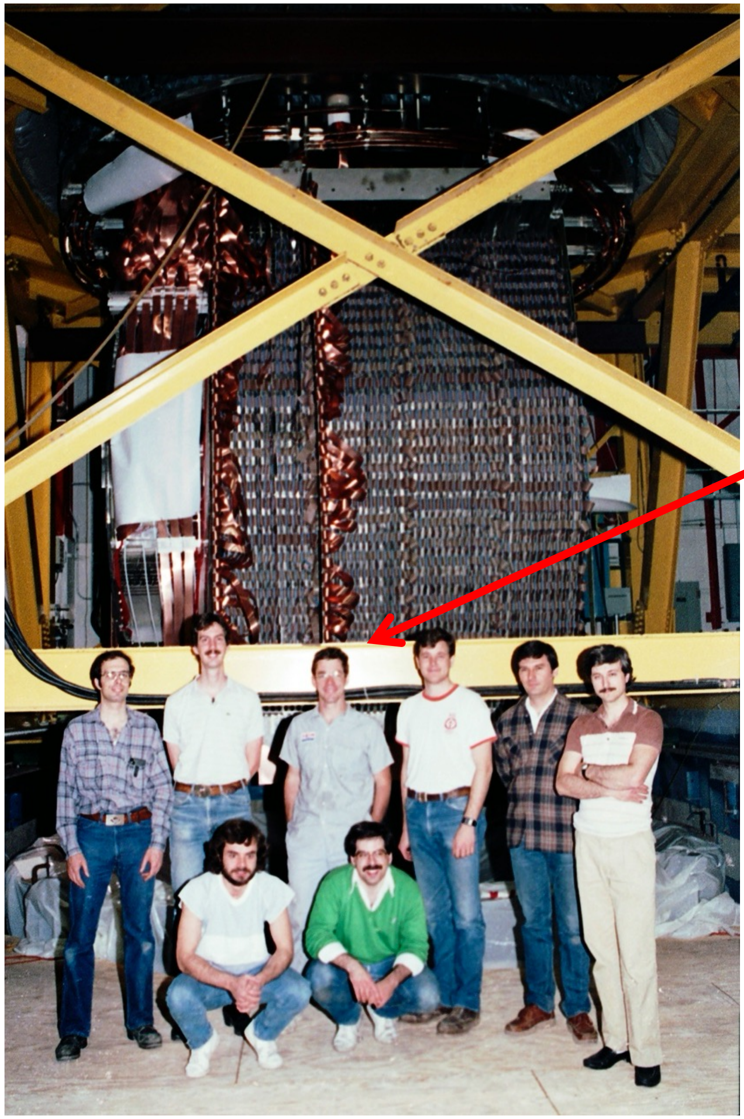
My Background

- 1984: BS in Engineering Physics, University of Arizona
  - Got a job in an HEP group after being fired from a gas station.
- 1984-1990: Grad Student, University of Rochester
  - PhD topic: Direct Photon Production in Hadronic Interactions
- 1990-1992: CERN Fellow, CERN
  - Studied e+e- reactions on the OPAL Experiment at LEP
- 1992-2001: Postdoc and Assistant Professor, Princeton U.
  - GEM Experiment at the Superconducting Super Collider ☹
  - Belle CP Violation Experiment at KEK, Japan
  - Nonlinear QED in E-144 Experiment at SLAC
- 2001-2017: Scientist, Fermilab
  - MiniBooNE short baseline neutrino oscillation experiment
  - Proton Source Department Head
  - Director of LHC Accelerator Research Program (LARP)
  - Mu2e rare muon conversion experiment
  - Created Lee Teng Internship and ran it for 10 years
- 2017-present, Professor, UC Davis
  - Continuing my work on Mu2e
  - Director, Crocker Nuclear Laboratory (cyclotron)

Experimental HEP

Accelerator Physics and HEP

Nuclear Physics
Medical Physics



Fermilab E-706 Rochester Group
~1987

“Buck’s River Road Exxon”

Me

Kwan Lai, the guy that hired me

Then

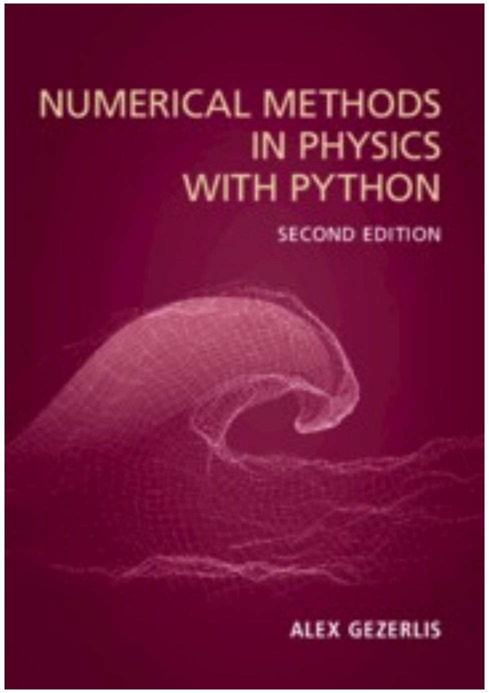
Recent

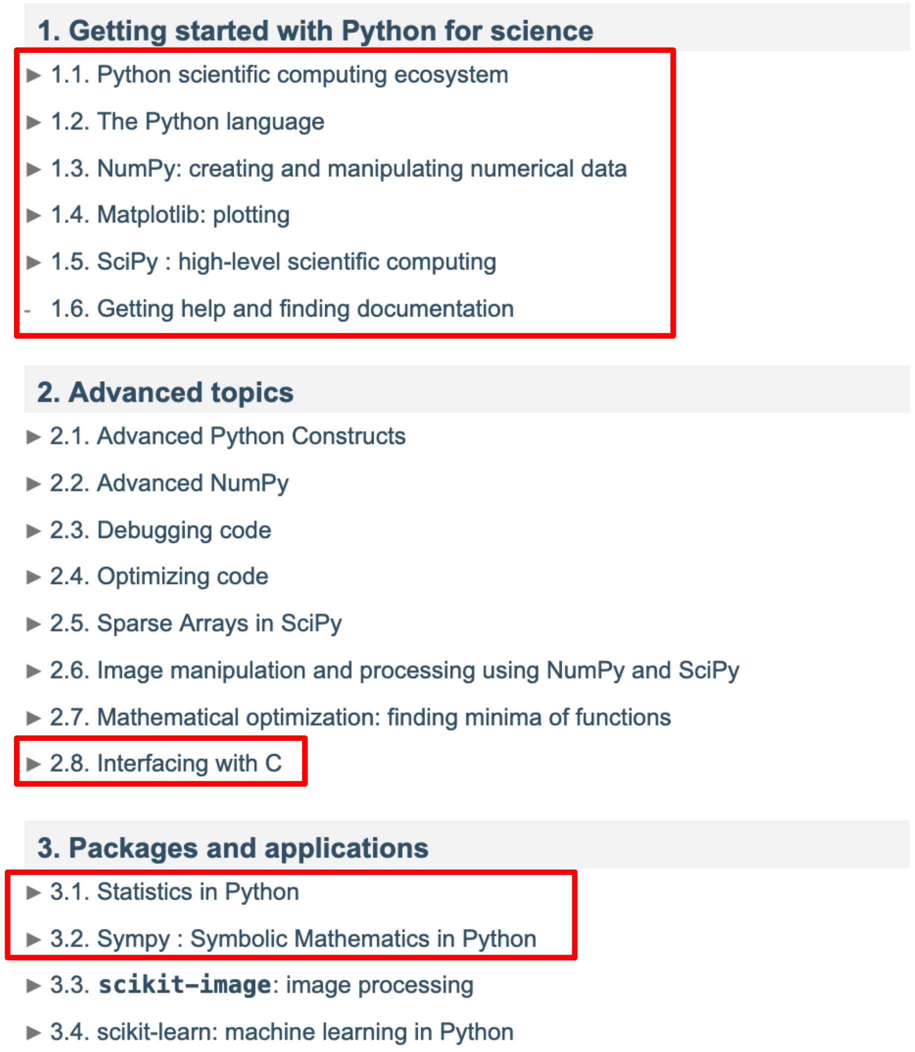
A Few Points About this Class…

- You'll find this class of great practical value in the future
  - The new curriculum assumes a knowledge of Python/Jupyter Notebooks in the upper division classes
  - You'll just find these tools very useful in your day-to-day science life.
- Not all universities have this class
  - Even graduate students often don't have these skills when they start out.
- This is a physics class, not a programming class.
- We're going to stress understanding concepts that are important to scientific computing:
  - Precision
  - Algorithms
  - Efficiency
- We're not going to go into some of the more arcane aspects of Python that you would in a Python course
  - In fact, you'll learn as little as possible, mostly by doing
- A lot of these skills will be transferrable to other languages.

Resources for the Course

- You can learn most of what you need to know about Python at https://lectures.scientific-python.org/
- More information about the applications are in Gezerlis, "Numerical Methods in Physics with Python" (available for free online)



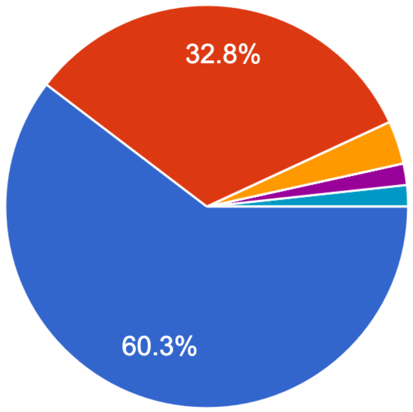


Schedule and Grading

- Lectures Monday and and Wednesday
- Labs Tuesday and Thursday
  - Attendance of the first lab is mandatory
- Short quiz in class every Wednesday
- Midterm: Wednesday, October 22, in class
- Final: Thursday, December 11, 8-10AM
- Grading:
  - 10% Lab attendance
  - 20% Midterm
  - 10% Quizzes
  - 40% Lab Reports
  - 20% Final

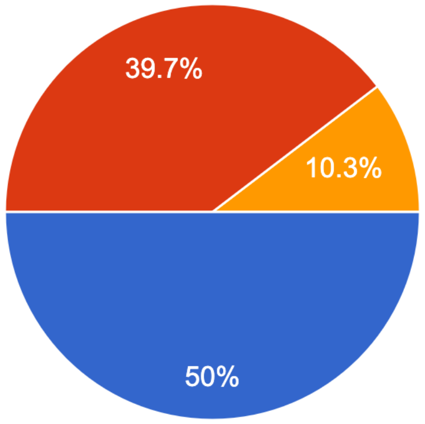
Course Demographics

Which operating system do you primarily plan to use for this course?
58 responses



- ● Windows 10 or 11
- ● MacOS
- ● Linux
- ● ChromeBook
- ● Dell
- ● Hp

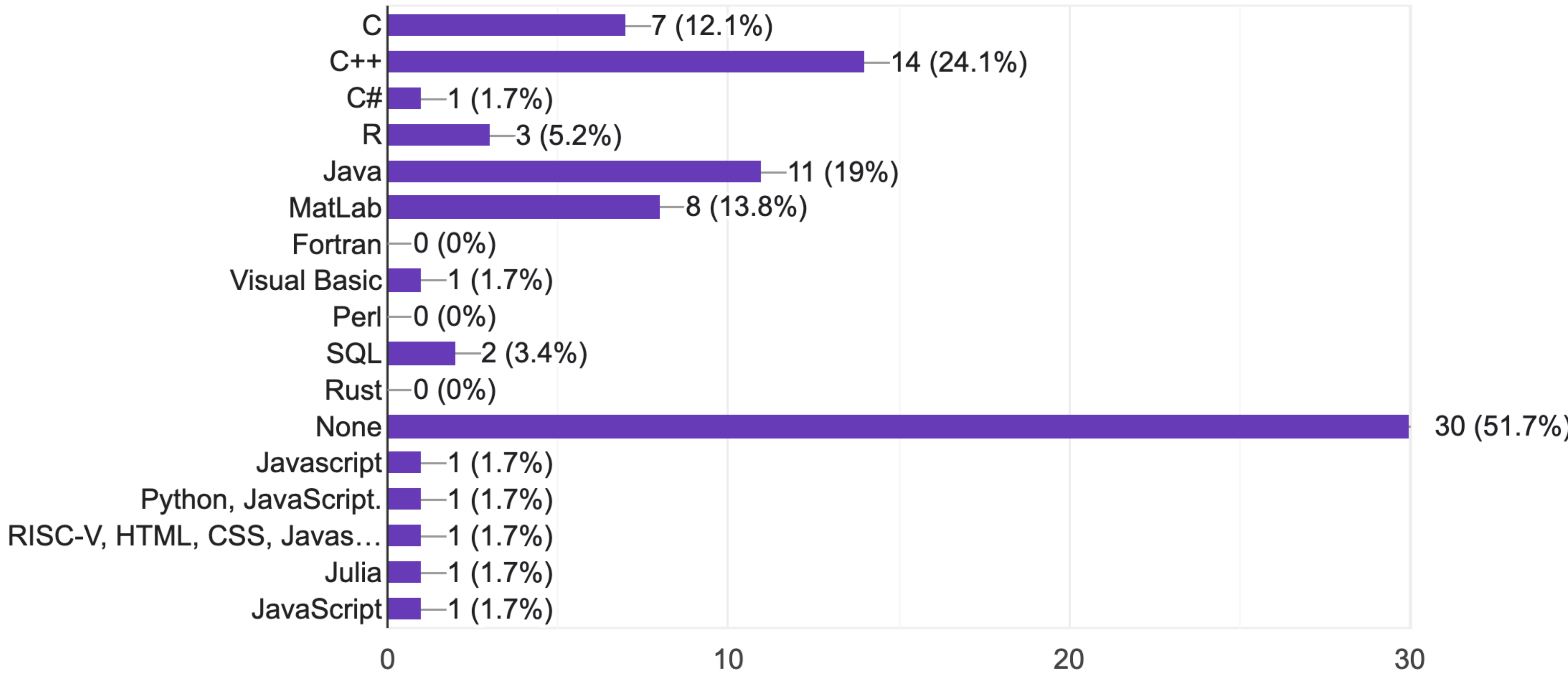
What is your experience with Python and Jupyter Notebooks? (Be honest!)
58 responses



- ● Complete newbie.
- ● I can do some basic operations, but nothing very complicated.
- ● Based on the syllabus, I think I aready have all the programming skills necessary for this course.

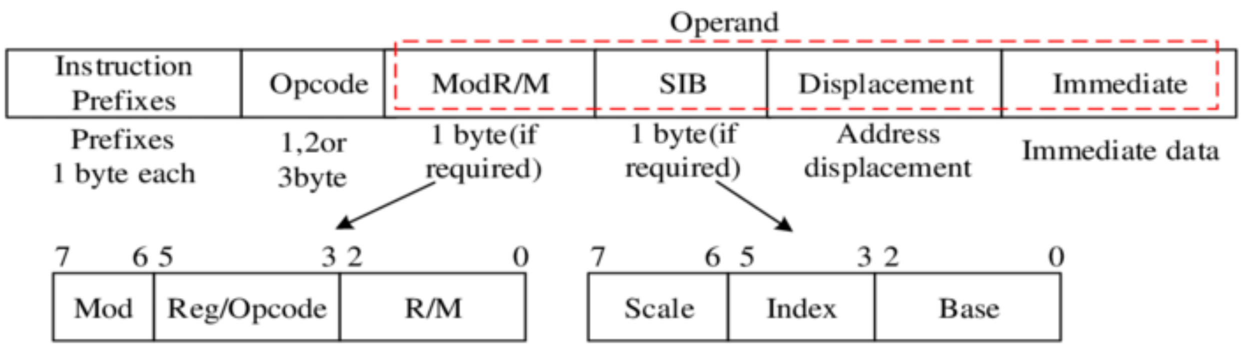
Other Experience

What other languages can you comfortably program in? Check any that apply.
58 responses



Evolution of Programming Languages

- At the most fundamental level, each type of processor is controlled by groups of bits, consisting of operational code (“op code”) words and operands.
  - This is the final output of any programming language! x86 example:



- The lowest level type of programming language is processor-dependent “assembly language”, in which human readable words are translated into op codes and mnemonics can be defined for memory locations, etc

```
push     ebp
mov      ebp,esp
sub      esp,0Ch
mov      dword ptr [ebp-4],0Ah
mov      dword ptr [ebp-8],14h
mov      dword ptr [ebp-0Ch],0
mov      eax,dword ptr [ebp-4]
add      eax,dword ptr [ebp-8]
mov      dword ptr [ebp-0Ch],eax
xor      eax,eax
mov      esp,ebp
pop      ebp
ret
```

Higher Level Languages

- Higher level languages add more complex instruction and data handling formats that don't directly map to processor op codes.
  - e.g. one line of a high-level code code can produce many individual op codes.
  - **Higher level languages are built on lower-level languages.**
- This increased abstraction allows high-level codes to be platform independent, at least at some level.
- Some important high-level languages over the years:
  - FORTRAN ("FORmula TRANSLATION", 1957)
    - dominated science from its inception in 1957 until ~early 1990s
    - Eventually developed an object-oriented version (Fortran 90), which is still used by some today.
  - BASIC ("Beginners' All-purpose Symbolic Instruction Code", 1964)
    - A lot of us learned programming with this language
    - Continues as "Visual Basic"
  - COBOL ("COmmon Business Oriented Language", 1960)
    - Heavily used in the banking industry to this day
    - Never used for science
  - Pascal (1970)
    - More efficient with better memory handling tools than FORTRAN
    - The native language of MacOS before OSX.

FORTRAN 77 Example

```fortran
C compare.f - Input two number, compare them, and print the sum
      PROGRAM COMPARE_INPUT
C     Reads two integers, compares them, and prints the results

      INTEGER A, B, SUM

C     Prompt and read A
      PRINT *, 'Enter value for a:'
      READ *, A

C     Prompt and read B
      PRINT *, 'Enter value for b:'
      READ *, B

      SUM = A + B

C     Conditional check
      IF (A .GT. B) THEN
         PRINT *, 'a is greater than b'
      ELSE
         PRINT *, 'a is less than or equal to b'
      ENDIF

C     Print formatted values
      PRINT *, 'a =', A, ', b =', B, ', a+b =', SUM

      END
```

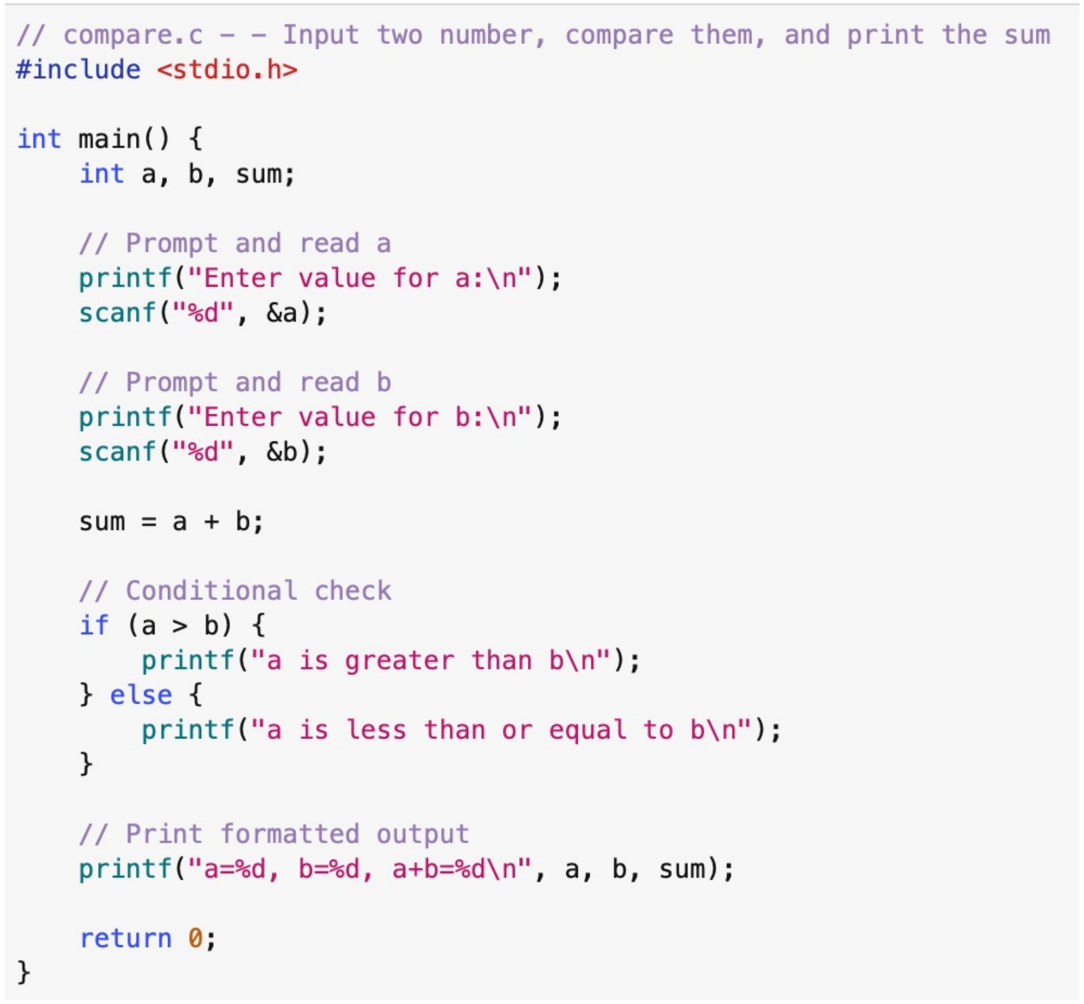
Note! Indentation is just to make it readable. It has no effect on the execution

```
(base) prebys@Erics-iMac phy40 % gfortran -o compare compare.f
(base) prebys@Erics-iMac phy40 % ./compare
 Enter value for a:
6
 Enter value for b:
5
 a is greater than b
 a =           6 , b =           5 , a+b =          11
```

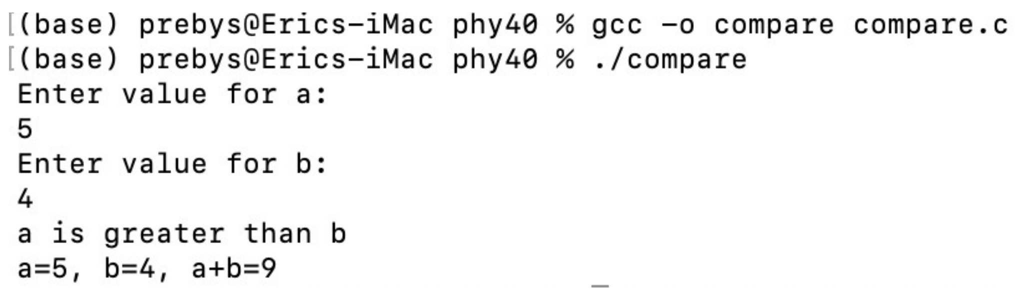
C

- The C programming language was released in 1972
- Compared to FORTRAN, it has
  - dynamic memory management
  - More elaborate data structures
  - multiple levels of indirection (value, pointer, pointer to pointer)
  - better and more efficient tools for low level operations
- Originally written to implement utilities in the Unix operating system, it eventually became the language of unix (and alter Linux) kernel.
- Heavily used in the computing industry by the late 1970s
- Became more and more common in physics starting around 1990 (although FORTRAN did not go quietly into that good night)

C Example



```c
// compare.c - - Input two number, compare them, and print the sum
#include <stdio.h>

int main() {
    int a, b, sum;

    // Prompt and read a
    printf("Enter value for a:\n");
    scanf("%d", &a);

    // Prompt and read b
    printf("Enter value for b:\n");
    scanf("%d", &b);

    sum = a + b;

    // Conditional check
    if (a > b) {
        printf("a is greater than b\n");
    } else {
        printf("a is less than or equal to b\n");
    }

    // Print formatted output
    printf("a=%d, b=%d, a+b=%d\n", a, b, sum);

    return 0;
}
```



```
[(base) prebys@Erics-iMac phy40 % gcc -o compare compare.c
[(base) prebys@Erics-iMac phy40 % ./compare
Enter value for a:
5
Enter value for b:
4
a is greater than b
a=5, b=4, a+b=9
```

Object Oriented Programming

- In traditional “structured” programming, data and executable code are maintained separately.
  - It is the programmer’s responsibility to ensure procedure gets the data it needs and that any output is stored appropriately.
- In “object-oriented programming”, “objects” contain related procedures (“methods”) AND the data associated with them.
- There have been many attempts at object-oriented languages, dating back to the early 1970s, but the most successful was C++, where objects (“classes”) evolved out of C data structures.
  - First released in 1985
  - Grew…”organically”
  - The inclusion of the “standard template library” (STL) in the mid 1990s made it *much* more usable.
  - Remains the go to language for high performance computing.

C vs. C++

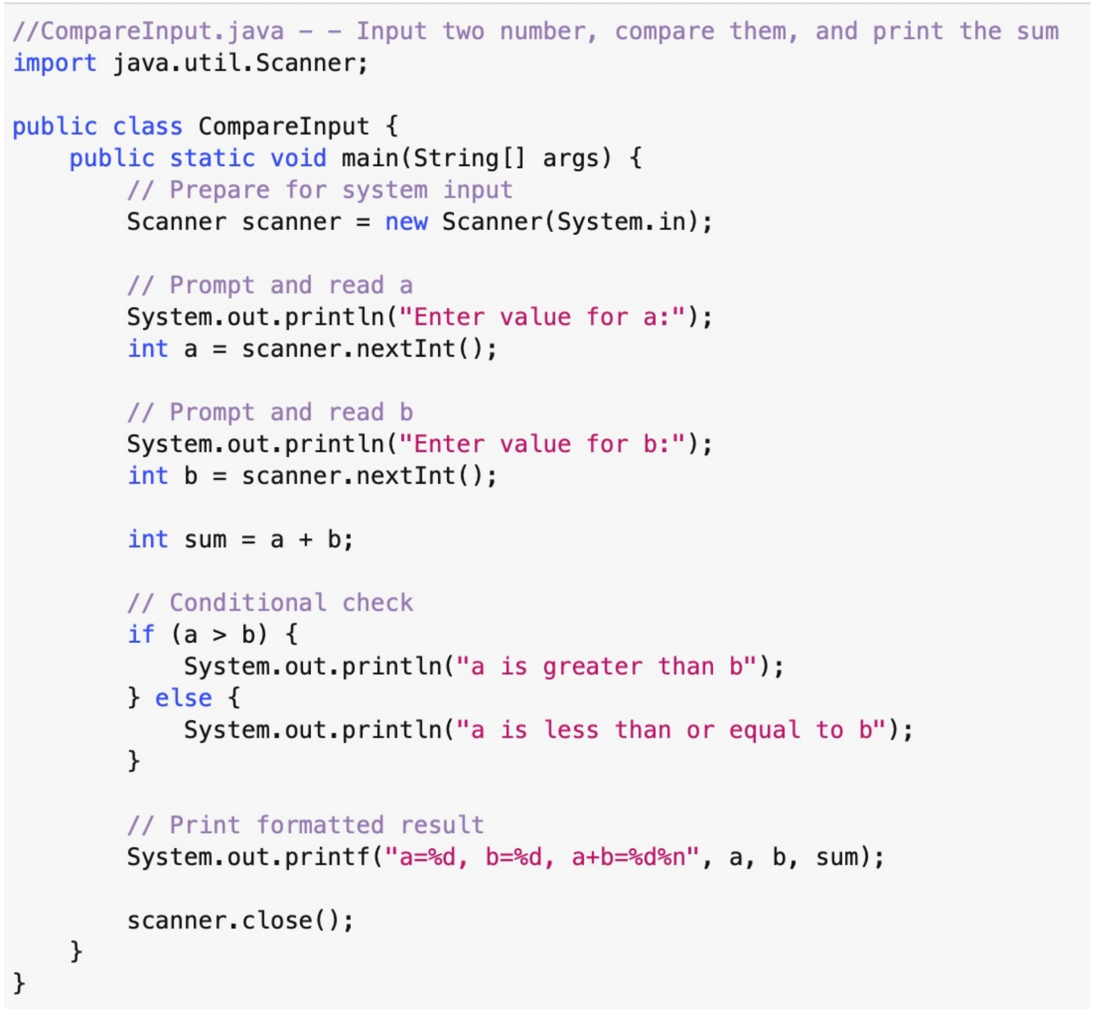
```c
// Structured version of code
#include <stdio.h>
#include <math.h>

// Define a "Point" data structure
struct Point {
    double x;
    double y;
};

// Define a routine to calculate the radius
double r(struct Point p) {
    return sqrt(p.x * p.x + p.y * p.y);
}

int main() {
  struct Point p[2]; // construct an array of two points
  // Initialize the data
  p[0].x = 3.;
  p[0].y = 4.;
  p[1].x = 5.;
  p[1].y = 12.;

  // Loop over the points
  for (int i=0;i<2;i++) {
    // Call the radius routine with each point
    double rad = r(p[i]);
    printf("Radius %d = %f\n",i,rad);   // Print it out
  }

  return(0);
}
```

```cpp
// Object Oriented version
#include <stdio.h>
#include <math.h>

// Define a "Point" class
class Point {
  public:
    double x;
    double y;
    // Define the radius routine *within* the object
    double r() {
       return sqrt(x * x + y * y);
    }
};


// Define a routine to calculate the radius

int main() {
  Point p[2]; // construct an array of two objects

  // Initialize the data
  p[0].x = 3.;
  p[0].y = 4.;
  p[1].x = 5.;
  p[1].y = 12.;

  // Loop over the points
  for (int i=0;i<2;i++) {
    // Implement the r() method of the object
    double rad = p[i].r();
    printf("Radius %d = %f\n",i,rad);   // Print it out
  }

  return(0);
}
```

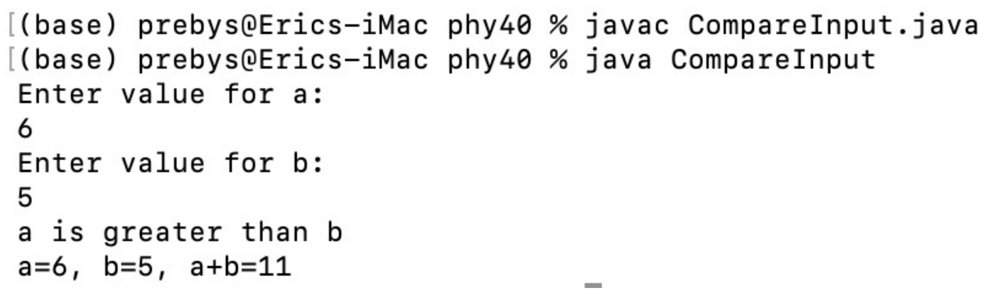
Java

- Java (not to be confused with javascript!) was released in 1995 by Sun Microsystems (now owned by Oracle)
- Syntax was based on C++, but
  - Development was centrally controlled, so its much more self-consistent
  - *Manifestly* object oriented, as opposed to C++, which can be a hybrid
  - Had much better built-in GUI tools than C++
- Based on a “virtual machine” model
  - Pros:
    - Made it the first credible language that was truly platform independent
    - The “pure” object-oriented nature make it a good learning platform
  - Cons:
    - Increased level of abstraction made direct interaction with hardware difficult
    - Less efficient

Java Example



```java
//CompareInput.java - - Input two number, compare them, and print the sum
import java.util.Scanner;

public class CompareInput {
    public static void main(String[] args) {
        // Prepare for system input
        Scanner scanner = new Scanner(System.in);

        // Prompt and read a
        System.out.println("Enter value for a:");
        int a = scanner.nextInt();

        // Prompt and read b
        System.out.println("Enter value for b:");
        int b = scanner.nextInt();

        int sum = a + b;

        // Conditional check
        if (a > b) {
            System.out.println("a is greater than b");
        } else {
            System.out.println("a is less than or equal to b");
        }

        // Print formatted result
        System.out.printf("a=%d, b=%d, a+b=%d%n", a, b, sum);

        scanner.close();
    }
}
```



```
[(base) prebys@Erics-iMac phy40 % javac CompareInput.java
[(base) prebys@Erics-iMac phy40 % java CompareInput
Enter value for a:
6
Enter value for b:
5
a is greater than b
a=6, b=5, a+b=11
```

Computing, Physics, and Physics Education

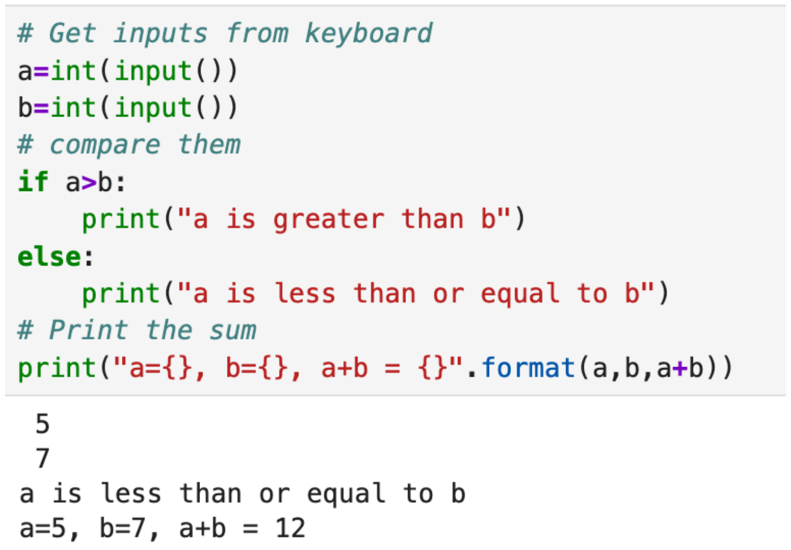
- Computing has been a central part of physics since the early 1960s.
  - Lots of algorithms developed back then are still in use today.
- Integrating computing into the physics curriculum has always been problematic.
  - Computing education was traditionally handled completely separately from physics education.
  - It was quite a while before people had their own computers.
  - There’s a bit of a learning curve with traditional languages before you get to anything useful.
  - If a class had a computational component, you’d typically learn that professor’s favorite language, which you’d likely use on central computing facilities.
  - Not everything is free!
- There’s never been a clear “right answer” about how to handle this
  - Until now…

What is Python and Why is it Cool?

- The development of Python began in 1989 by Guido Van Rossum, in the Netherlands
- Named after Monty Python's Flying Circus
- Originally written in C
- Defining features:
  - **Readability and simplicity**
  - Large included library
  - Truly cross-platform
  - Dynamically typed
  - Interpreted
  - Multi-paradigm
    - Can implement structured or object-oriented code
  - Easily extensible
- It's a resource hog, so it had to wait for computers to catch up to be truly useful
- Usage really took off with the release of Python 2.0 in 2000
  - Since then, it has developed an enormous ecosystem
- Python 3.0 was a backwards INcompatible release in 2008
  - Now the standard

Some Pros and Cons of Python

- Pros:
  - Basic operations are as simple as they can get
  - More complex things are also about as simple as they can get
  - Free and truly platform independent
  - By now there's a Python-based solution to just about any problem you can come up with.
- Cons:
  - The "loosey-goosey" interpreter and dynamic typing can be a blessing and a curse
    - Will often forgive mistakes and do what it *thinks* you want it to do.
  - As an interpreted language, it's quite slow, which brings us to…



```python
# Get inputs from keyboard
a=int(input())
b=int(input())
# compare them
if a>b:
    print("a is greater than b")
else:
    print("a is less than or equal to b")
# Print the sum
print("a={}, b={}, a+b = {}".format(a,b,a+b))
```

```
 5
 7
a is less than or equal to b
a=5, b=7, a+b = 12
```

Python and Speed

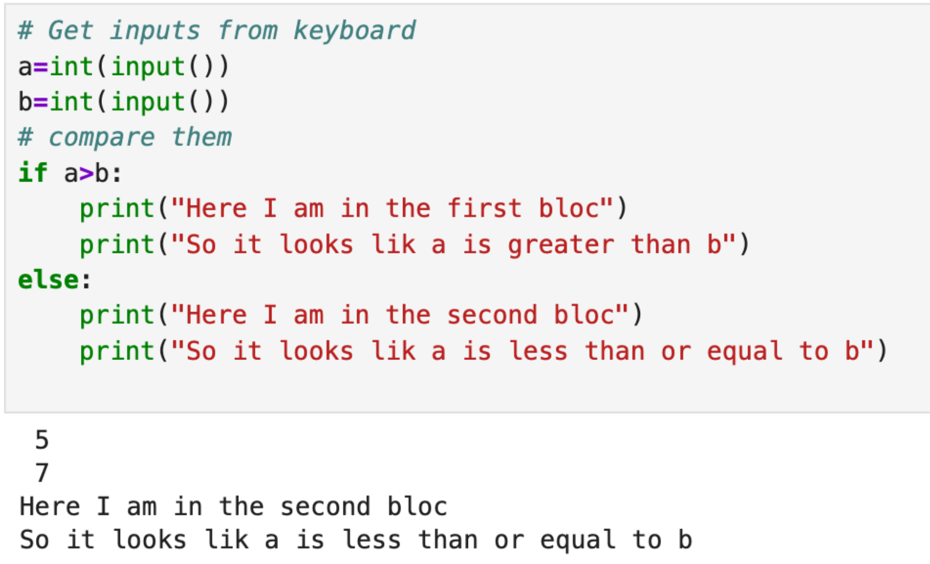
- Interpreted Python is extremely slow, BUT
- Python packages are written in other, faster languages
  - The core of Python is written in C
  - Advanced packages are mostly written in C++
  - Very compute intensive stuff like machine learning may actually be running on GPUs or even firmware (FPGAs)
- Whenever possible, use a utility rather than coding something yourself
  - We'll do some examples of this in lab
- This leads us to…

Python as "Glue"

- Because of its universality and ease of use, many *extremely* complex software packages have now adopted the philosophy of developing modules in compiled languages or even on dedicated hardware, adding Python "wrappers", and then using it to control the workflow at the top level.
- Examples:
    - Machine learning:
        - scikit-learn
        - PyTorch
        - TensorFlow
    - Accelerator modeling:
        - PyOrbit
        - Synergia
        - xSuite (included with the standard Python release!)
    - Physical Modeling
        - Numerous attempts to put Python wrappers around the user hostile horror show that is Geant4

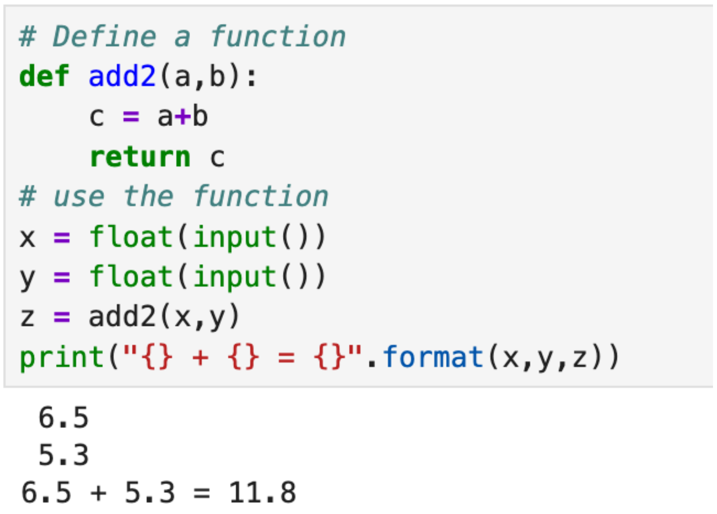
Key Features of Python

- Most of Python syntax is pretty obvious and very readable
- The key feature that sets it apart from all other major scientific languages is the fact that blocs of code are delimited by level of indentation:
  - This is known as “off-side rule”
  - It turns out a number of other languages use it, but I’ve never heard of ANY of them
- Because it’s an interpreted languages libraries, classes, routines, etc must be declared *before* you can use them.



```python
# Get inputs from keyboard
a=int(input())
b=int(input())
# compare them
if a>b:
    print("Here I am in the first bloc")
    print("So it looks lik a is greater than b")
else:
    print("Here I am in the second bloc")
    print("So it looks lik a is less than or equal to b")
```

```
 5
 7
Here I am in the second bloc
So it looks lik a is less than or equal to b
```



```python
# Define a function
def add2(a,b):
    c = a+b
    return c
# use the function
x = float(input())
y = float(input())
z = add2(x,y)
print("{} + {} = {}".format(x,y,z))
```

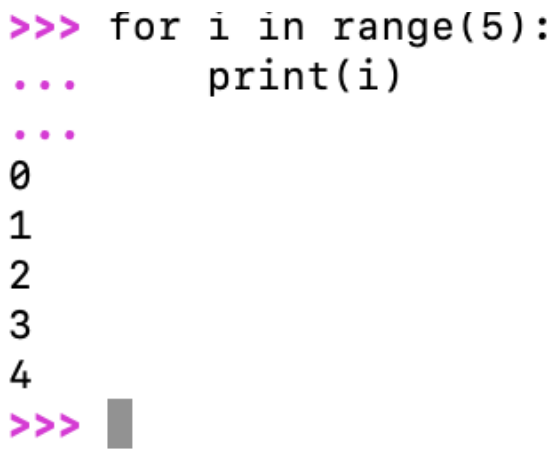
```
 6.5
 5.3
6.5 + 5.3 = 11.8
```

Some Major Python Libraries

- Core Libraries for this Course
  - **numpy:** very powerful array and numerical manipulation tools
  - **matplotlib:** extensive plotting tools
  - **scipy:** more advanced tools for statistics, fitting, etc
  - **pandas:** powerful data handling and manipulation tools
- Some useful utility libraries
  - **os:** interacting directly with the operating system
  - **sys:** system access complementary to os
  - **pickle:** serialization/deserialization tools. Useful for storing and restoring the state of code.
- More advanced libraries
  - **sympy:** symbolic manipulation tools (goodbye MatLab!)
  - **scikit-image:** image processing tools
  - **scikit-learn:** machine learning tools (we'll use this in 118)

The Development of Interactive Python

- Python is an interpreted language, and therefore manifestly interactive

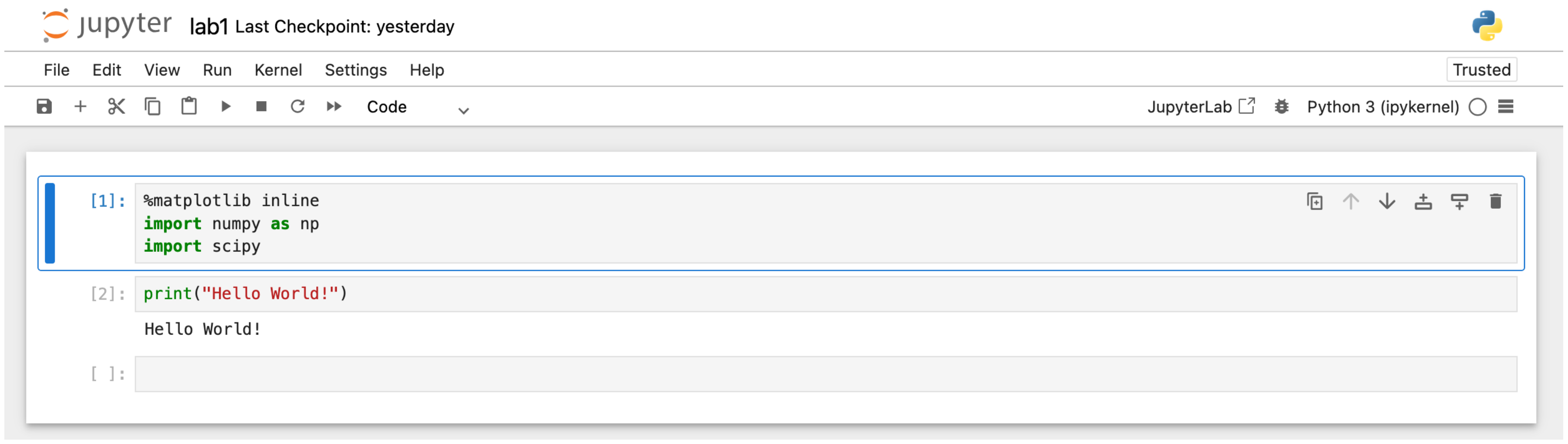


```
>>> for i in range(5):
...     print(i)
...
0
1
2
3
4
>>> 
```

- In 2001, iPython ("Interactive Python") was introduced.
  - Enhanced interactive capability, particularly for graphics
- In 2011, iPython introduced browser-based "notebooks"
  - Even more interactive capability
  - Powerful markdown language to add to executable content
  - "Language agnostic" model supported other languages
  - Became the primary kernel for Project Jupyter

Jupyter Notebooks

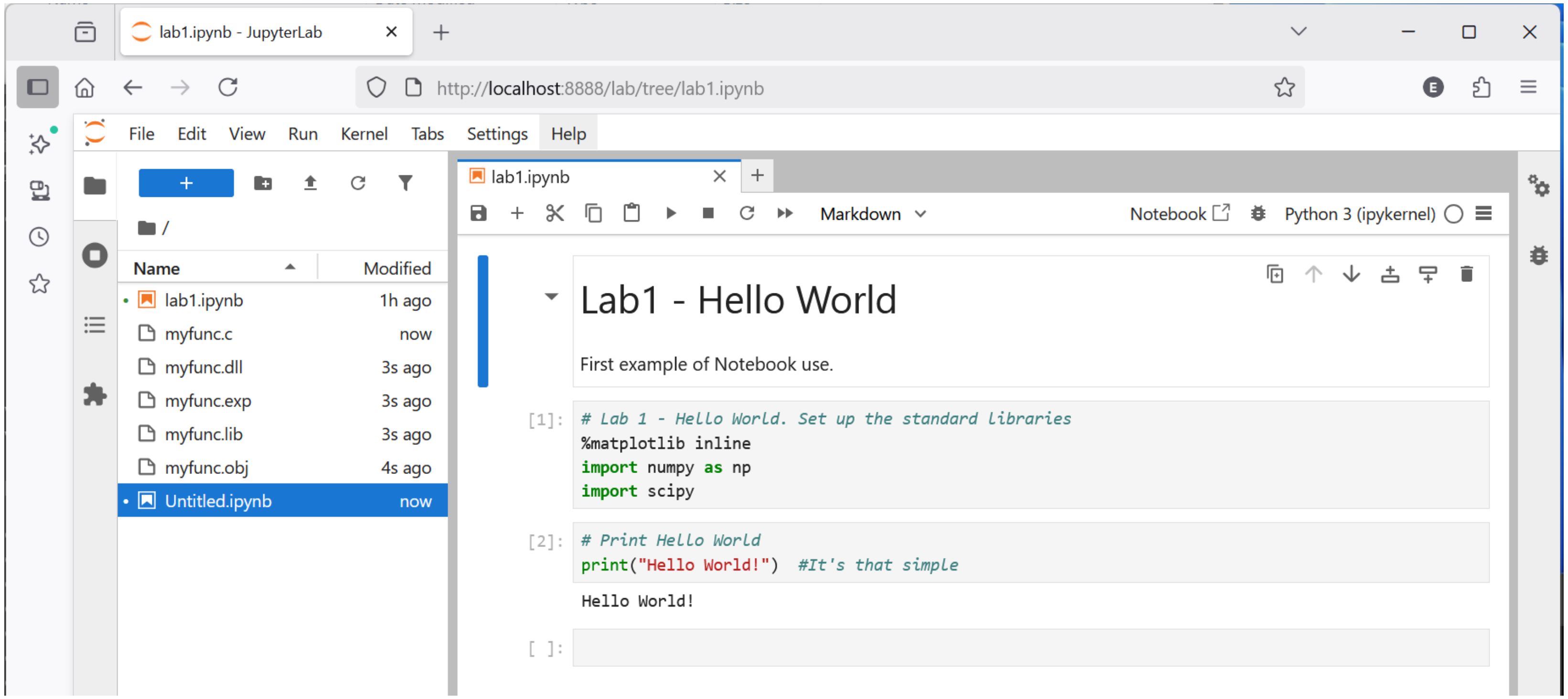
- In 2014, Project Jupyter spun off of iPython
- Enhanced the web interface and support for other kernels
  - R, Julia, etc, which we won't use
- The term Jupyter Notebook can refer to
  - The web interface itself
  - The .ipynb notebook files in which it saves content


jupyter lab1 Last Checkpoint: yesterday
File Edit View Run Kernel Settings Help
Trusted
Code
JupyterLab
Python 3 (ipykernel)

```
[1]: %matplotlib inline
     import numpy as np
     import scipy

[2]: print("Hello World!")

     Hello World!

[ ]:
```

Jupyter Lab

- In 2018, the JupyterLab interface was released
  - Same notebook format and interaction as the Jupyter Notebook interfact
  - Improved development environment. In particular, an integrated file browser system.


lab1.ipynb - JupyterLab
http://localhost:8888/lab/tree/lab1.ipynb
File Edit View Run Kernel Tabs Settings Help
Markdown
Notebook
Python 3 (ipykernel)

Name	Modified
lab1.ipynb	1h ago
myfunc.c	now
myfunc.dll	3s ago
myfunc.exp	3s ago
myfunc.lib	3s ago
myfunc.obj	4s ago
Untitled.ipynb	now

Lab1 - Hello World

First example of Notebook use.

```
[1]: # Lab 1 - Hello World. Set up the standard libraries
     %matplotlib inline
     import numpy as np
     import scipy

[2]: # Print Hello World
     print("Hello World!")  #It's that simple

     Hello World!

[ ]:
```