# DSC-014: Scientific Programming with Python

Introduction

---

Arvind K. Saibaba

August 22, 2022

# Preliminary information

# Class details

- Instructor: Prof. Arvind K. Saibaba.
- Office: SAS Hall 3118.
- Class Times: F 1:55 PM - 2:45 PM.
  - August 22-December 5. (last week: Nov 29-Dec 5)
  - Exceptions: September 5 (Labor Day), October 10 (Fall Break)
- Office hours: Monday 2:45-3:30 PM (SAS office), Thursday 9-10 (via zoom).

**Funding**: The teaching material developed as a part of this course is funded by the National Science Foundation through the awards DMS-1720398 and DMS-1821149.

# Topics

- Shell Programming (1 lecture)
- Version Control using Git (1 lecture)
- Python (9-10)
  - Basics: Variables/Operators/Functions
  - Data structures (list, dictionaries, tuples)
  - Loops and conditionals
  - Object-oriented programming
  - Error handling
  - Plotting (matplotlib)
  - Scientific Computing packages (NumPy/SciPy)
  - Data Analysis (Pandas)
- In-class problem session (1-2)

# Bird's eye view of the course

Overall letter grade based on

- Weekly problem sets due Mondays 5 PM (ET).
- Short Project (due end of semester).

In-class problem sessions will be held to help with these aspects.

Prerequisites

- Basic programming in any high-level programming language (e.g., C/C++, MATLAB, R).
- Some basic knowledge of mathematics (Calculus, Linear Algebra, Probability) is desirable but not necessary.

Required books:

- None. A list of references is provided on the course webpage.

# Learning Objectives

- Exposure to modern software tools
  - Shell commands, Version Control (Git), Computer clusters
- Learn object-oriented programming

- Learn useful tools for scientific computing
  - Unit testing, Exception handling, Debugging
- Creating packages for sharing/reproducible research

# Software and Hardware

Hardware:

- I will assume that you have access to a laptop
- If you don't have one, I can arrange for one (during in-class problem sessions)
- The in-class demonstrations will use MacOSX

Software:

- Git
- Shell environment
  - If using Windows you may need a shell emulator
  - MacOSX and GNU/Linux typically come with a bash shell
- A Python Environment - I recommend Miniconda/Anaconda to manage packages

Details on this are posted on the website (this is part of Homework 0).

# Python

# Python (**)

Guido Van Rossum
"Benevolent dictator for life!**"

His vision for Python

- An easy and intuitive language just as powerful as major competitors
- Open source, so anyone can contribute to its development
- Code that is as understandable as plain English
- Suitability for everyday tasks, allowing for short development times

---

(*) Slide generously borrows material from Wikipedia.
(**) At Google until 2018, after which he moved to Dropbox, then retired, then Microsoft.

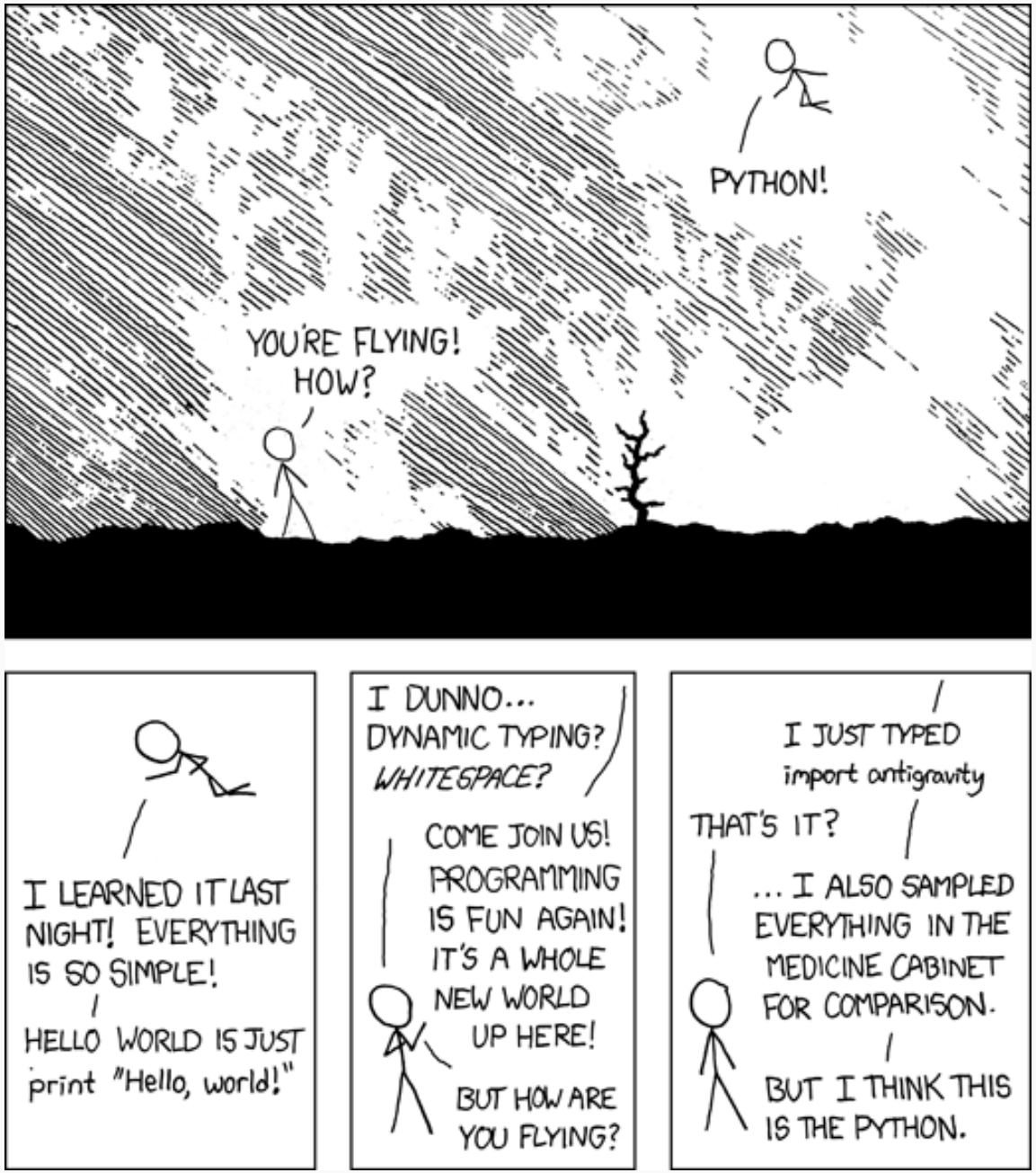
# Why the name Python?

# Why the name Python?

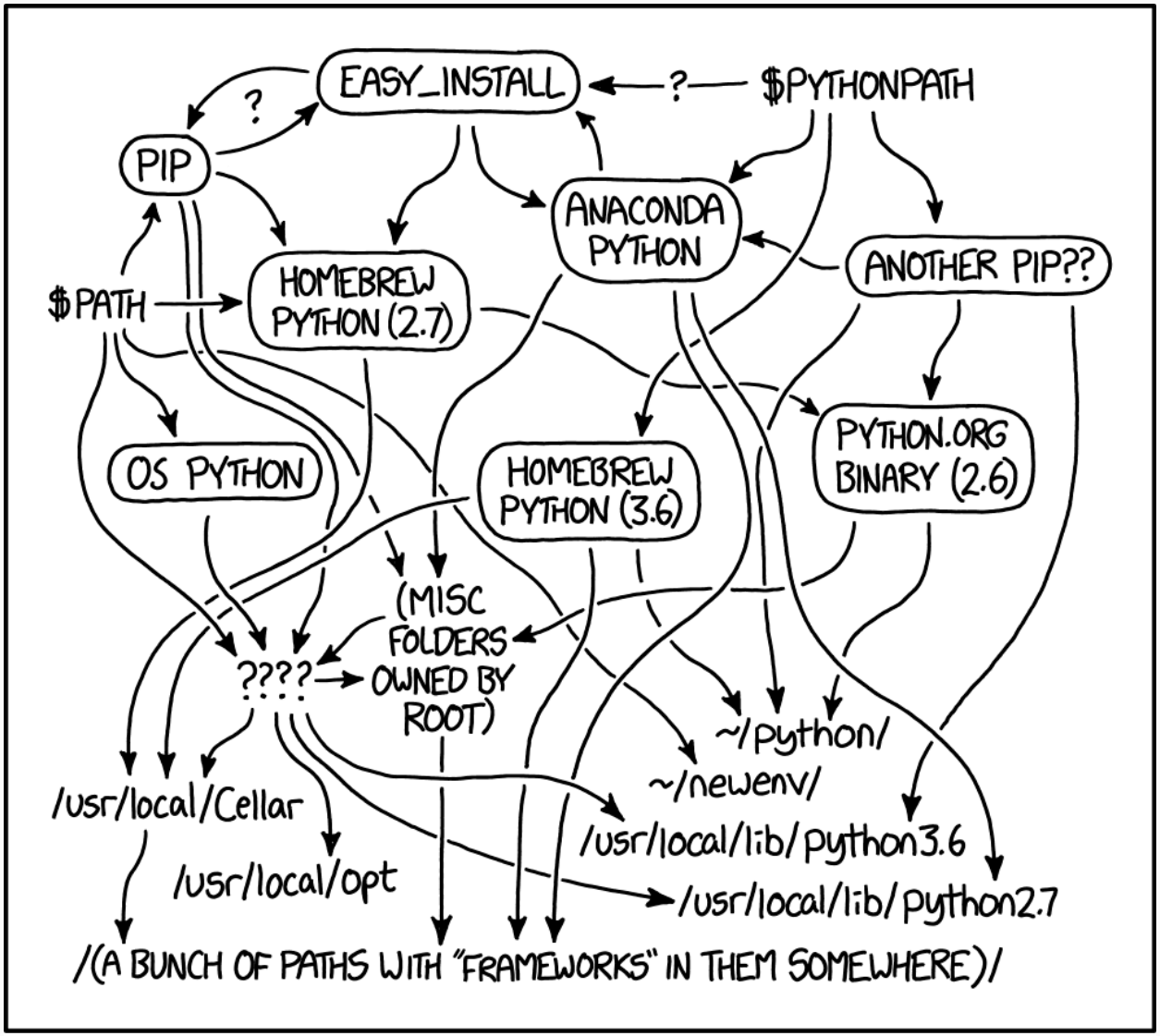
And now for something completely different ...


Monty Python references are always encouraged!

# XKCD + Python


PYTHON!
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.

# The reality of Python installation




MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

# Why Python?

From wikipedia: "Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including procedural, object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library. "

# Why Python?

Strengths:

1. Good for quick prototyping, i.e., programmer time is more valuable than computer time
2. Good set of packages for scientific computing and data science
3. Good for "gluing" pieces of code written in different languages
4. Healthy community and ecosystem

Weaknesses:

1. Interpreted language, so some loss of performance
2. No centralized development of language/packages
3. High performance computing?

If performance is critical either a lower level language (e.g., C/Fortran), or a combination of a lower-level language with Python may be more optimal.

# Python Packages


1. Scientific Computing
    - SymPy, scikits, PDE (FireDrake, FEniCS)
2. Data Science/Machine Learning
    - Pandas, AstroML, scikit-Learn, Keras, TensorFlow.

# PEP 10: The Zen of Python

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than right now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.

Namespaces are one honking great idea – let's do more of those!

Questions?