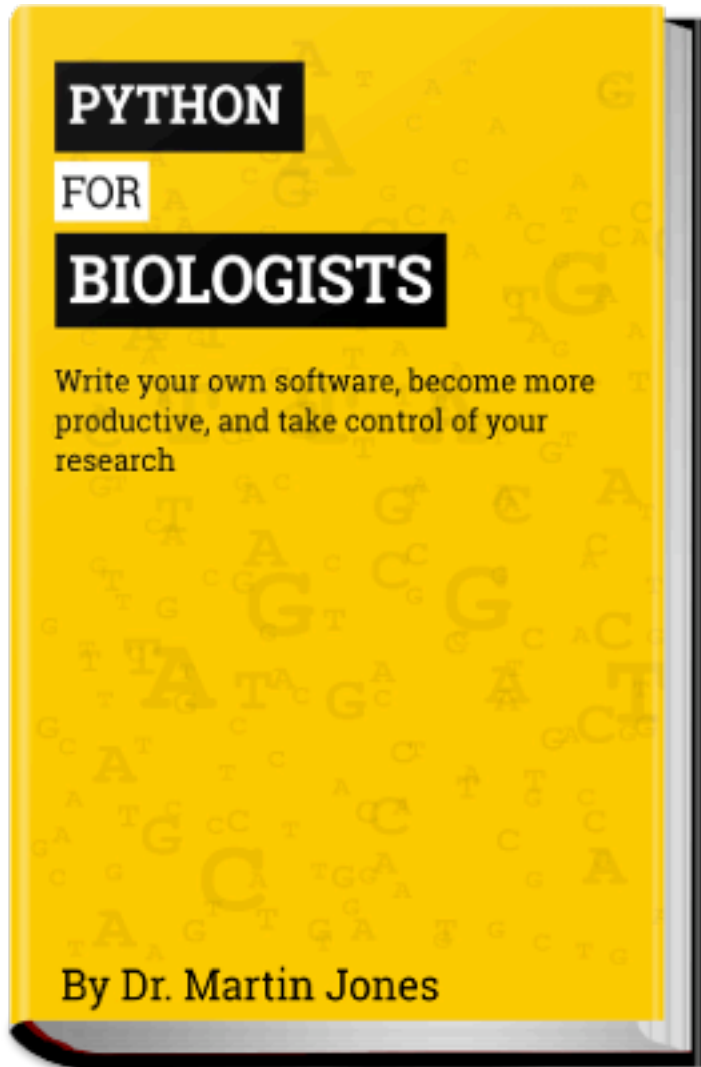




Book



The first part of the material is available for free on the website:

<http://pythonforbiologists.com/>

But you should consider buying!
\$39 on website or on amazon

All examples are pertinent to biologists, most are directly based on sequence data.

Book Bundle



\$99 for bundle

Good investment for a lab group.

Which programming language should I learn?

This is surprisingly unimportant for 3 reasons:

1. Nearly everyone who programs professionally uses more than one language
2. Learning one programming language gets you 90% of the way to learning another one. Programming is more a way of thinking, syntax is just details.
3. Most problems can be solved in any language.

Which programming language should I learn?

What is important?

- Putting in the time
- Motivation
- Colleagues that can help
- Trying to solve real problems

Learning a language

This class provides a teaser. (only 4 lessons!)

Our current goals:

1. Understand what it means to write code and execute scripts
2. Understand how programmers break problems into instructions
3. Be able to communicate with your bioinformatics colleagues
4. Be able to use scripts and maybe even debug/alter them

Learning a language

To become proficient, you'll need a lot more. Classes to try:

- LFSC 507 - Programming for Statistical and Graphical Analysis of Biological Data (Python)
- GEOL 590 - Introduction to Reproducible Data Analysis (R)
- COSC 505 - Introduction to Programming for Scientists and Engineers (Python and C++ and ???)

Next level after 1 full semester of programming:

- COSC 594 - Special Topics: Bioinformatics Computing

Bioinformatic course list:

<https://ceb.utk.edu/bioinformatics-courses-at-ut/>

Why we will use Python

- It has a consistent syntax, so you can generally learn one way of doing things and then apply it in multiple places
- It has a sensible set of built in libraries for doing lots of common tasks
- It's one of the most widely used languages in the world, and there's a lot of advice, documentation and tutorials available on the web
- Easy to start learning - It's designed in a way that lets you start to write useful programs as soon as possible
- Its use of indentation, while annoying to people who aren't used to it, is great for beginners as it enforces a certain amount of readability

Why python is great for biology

- Widely used in the scientific community
- Well designed libraries for doing complex scientific computing (although we won't encounter them in this course or in the book)
- It has features which make it easy to manipulate strings of characters (for example, strings of DNA bases and protein amino acid residues, which we as biologists are particularly fond of)

The bioinformatics field is moving from perl to python

The switch happened between 2008 and 2010

What do biologists need?

- A tool to get science done
- Low barrier to entry and efficiency in writing code

How does python meet those needs?

- Emphasis on readability and simplicity
- Possibly easier to learn than perl*
- Excellent scientific libraries of code to leverage
- “Trendy” factor

Some good news:
If you learn python the leap to perl is very short.
You'll probably already be able to read perl.

An Empirical Investigation into Programming Language Syntax, 2013, Stefik and Seibert, ACM Transactions on Computing Education



- The debate is now python vs R
- Fortunately learning one language makes it much easier to learn a second language
- You start by learning to think like programmer



R:

- Focused on statistics and graphical models
- Preferred by data scientists, statisticians, quants
- Has always been centered in academics and research, less in industry



#4

And The Winner is...

It's a tie!
It's up to you, the data scientist,
to pick the language that best fits your needs.
The following questions can guide you in your decision.

1

What problems do you want to solve?

2

What are the net costs for learning a language?*

* it will cost time to learn a new system that is better aligned for the problem you want to solve, but staying with the system you know may not be made for that kind of problem.

3

What are the commonly used tool(s) in your field?

4

What are the other available tools in your field and how do these relate to the commonly used tool(s)?

TIOBE for Oct 2018

Oct 2018	Oct 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.801%	+5.37%
2	2		C	15.376%	+7.00%
3	3		C++	7.593%	+2.59%
4	5	^	Python	7.156%	+3.35%
5	8	^	Visual Basic .NET	5.884%	+3.15%
6	4	v	C#	3.485%	-0.37%
7	7		PHP	2.794%	+0.00%
8	6	v	JavaScript	2.280%	-0.73%
9	-	^^	SQL	2.038%	+2.04%
10	16	^^	Swift	1.500%	-0.17%
11	13	^	MATLAB	1.317%	-0.56%
12	20	^^	Go	1.253%	-0.10%
13	9	v	Assembly language	1.245%	-1.13%
14	15	^	R	1.214%	-0.47%
15	17	^	Objective-C	1.202%	-0.31%
16	12	v	Perl	1.168%	-0.80%

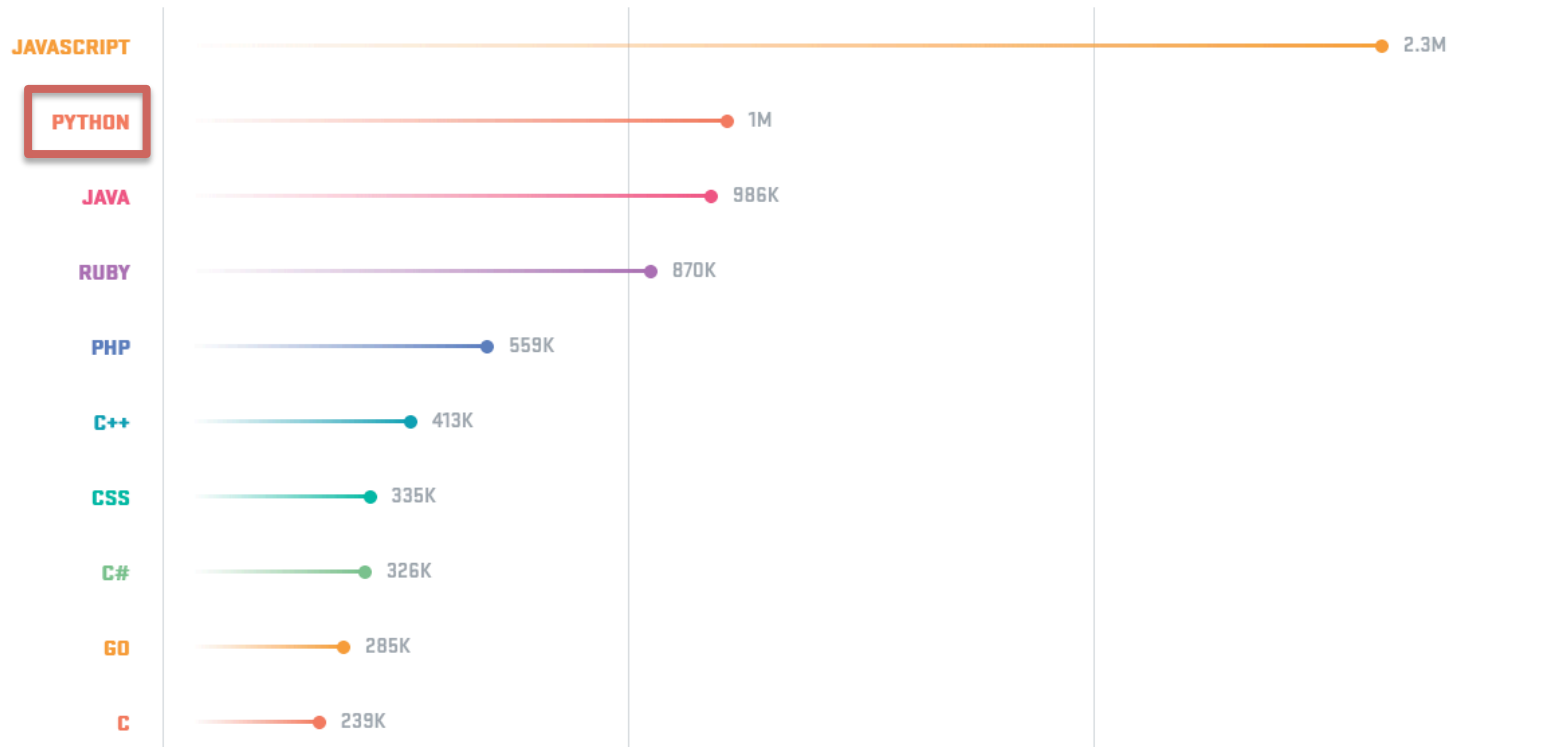
GitHub (Open Source), Year in Review 2017

The fifteen most popular languages on GitHub

by opened pull request

GitHub is home to open source projects written in 337 unique programming languages—but especially JavaScript.

R and perl didn't make the top 15.



Python - History

- Created by Guido van Rossum, first released in 1991
- Currently developed by the Python Software Foundation
- Named after Monty Python's Flying Circus
- Design philosophy
 - Code readability
 - Consistency
 - Compact, uncluttered layout
 - English keywords instead of punctuation
 - Whitespace indentation is part of the syntax



Benevolent Dictator For Life



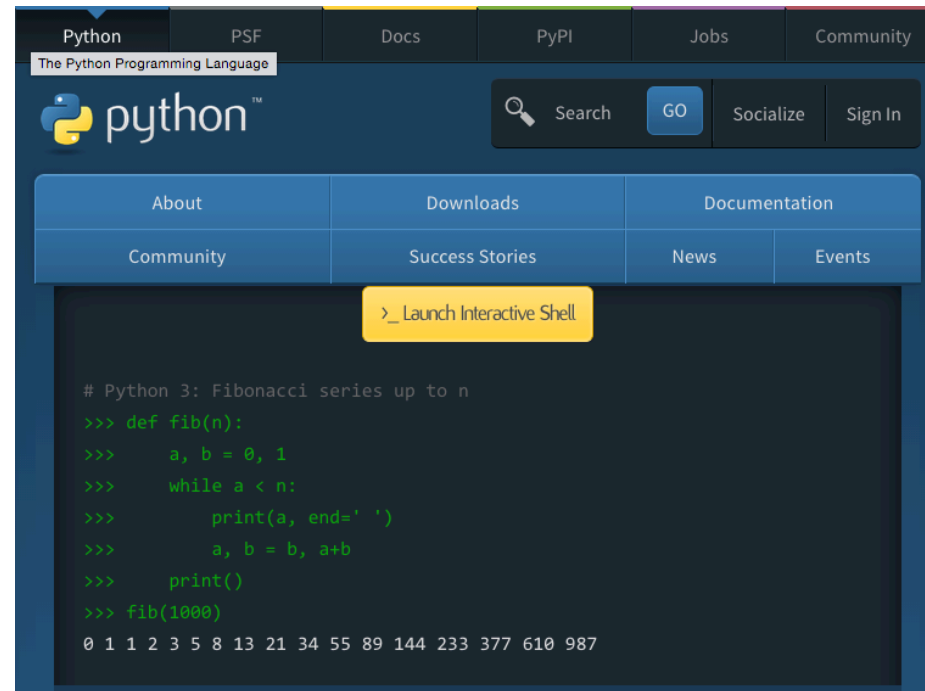
British sketch comedy series created by the comedy group Monty Python and broadcast by the BBC from 1969 to 1974

Python - Philosophy

- document "PEP 20 (The Zen of Python)" has 20 aphorisms, of which 19 are written down.
- Including:
 - Beautiful is better than ugly
 - Simple is better than complex
 - Complex is better than complicated
 - Readability counts
 - Special cases aren't special enough to break the rules.
 - If the implementation is hard to explain, it's a bad idea.

Python – Resources

- Python Software Foundation
 - <http://python.org>
 - Downloads and installation instructions
 - Documentation (in > 70 languages)
 - Python for beginners
- Python Package Index
 - <http://pypi.python.org>
 - official repository for python software packages
 - more than 49,000 packages
 - Codename PyPI



Python Versions

The python world is in the midst of a rather messy transition.

- 3.7 current stable release
 - 3.0 released in 2008
 - Shiny and new
 - Not much backward compatibility with Version 2
- 2.7
 - The final 2.x version, released 2010 – no new major releases
 - Statement of extended support
 - Still being supported because a lot of existing code has never been upgraded

Backwards Compatible

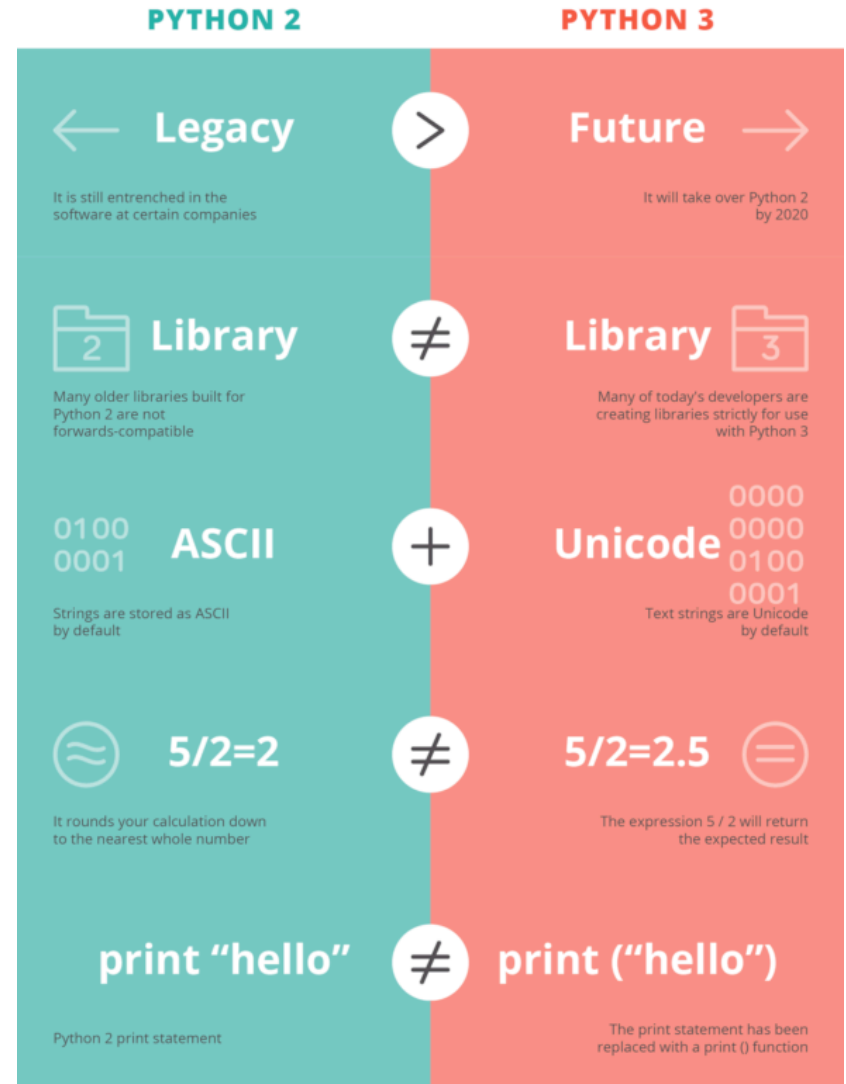
- software or programming language that can successfully use interfaces and data from earlier versions of the system
- “interoperable” with an older system
- Has benefits
 - Users can continue to use old code/data
 - Easy upgrades (no need to rewrite everything!)
- And disadvantages
 - Have to maintain legacy features
 - Increased complexity (can’t streamline by tossing old stuff)
 - Slowing innovation



Python Versions

We are going to stick with version 3

- Beware that old code from python version 2 may not work
- Beware that old documentation from python version 2 may steer you in the wrong direction



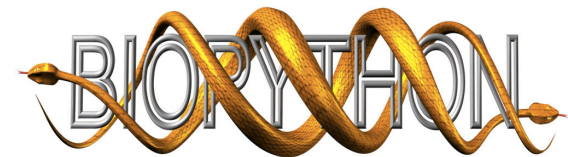
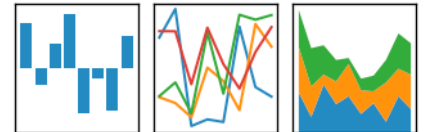
Packages You May Want to Learn

- NumPy
 - Scientific computing package
 - Large multidimensional arrays
 - linear algebra, Fourier transform, and random number capabilities
- Pandas
 - data structures and operations for manipulating numerical tables and time series
- Biopython
 - www.biopython.com
 - Bioinformatics support
 - Support for 2.7 and 3.3



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



Python on ACF

- Use the module system
- Available versions:

```
[mstaton1@acf-login2 ~]$ module avail python
[
----- /sw/acf/modulefiles -----
python/2.7.13(default)      python3/3.6.1
python/3.6.1               python3/3.6.5
python2/conda2-4.4.0(default) python3/conda3-4.4.0(default)
```



A distribution of python that has tools to manage packages, dependencies, and environments.

- Simplifies installing packages like numpy and biopython
- Especially useful on the ACF where we don't have admin access
- Can install packages in your own user space

Many other features, check out website.

Data types

Python has types of data. Here are a few.

Type	Abbr.	Example 1	Example 2
Integer	int	4	12
Floating Point Number	float	4.2	1.00e-04
Boolean	bool	True	False
String	str	'I am a string'	"so am I"

Strings

- A sequence of characters
- A string literal is a representation of a sequence of characters in source code
- In python, delimited by single or double quotes

`'This is a string'`

`"This is also a string"`

(or triple single quotes but we aren't going to worry about that)

```
# This is a comment
print("This is a string")
x = y + z
```

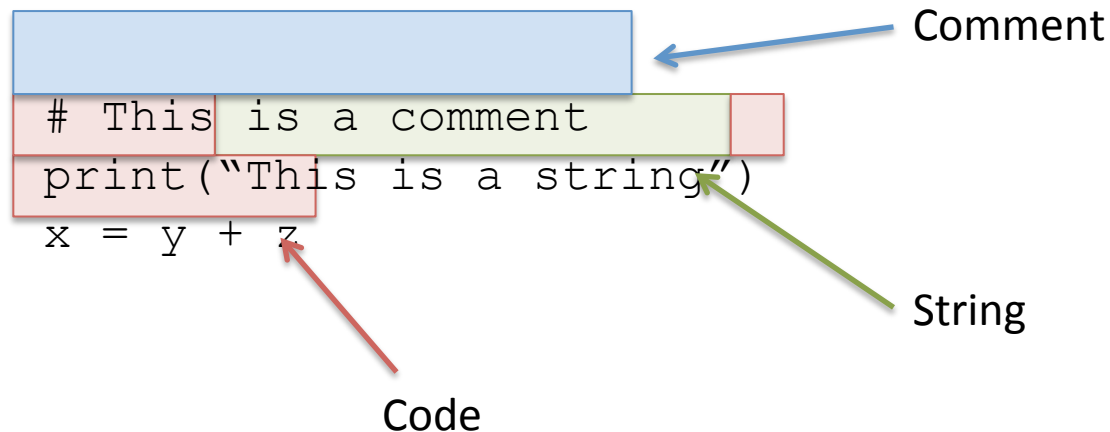
Strings

- A sequence of characters
- A **string** of characters
- A string literal is a representation of a sequence of characters in source code
- In python, delimited by single or double quotes

'This is a string'

"This is also a string"

(or triple single quotes but we aren't going to worry about that)



The diagram shows a snippet of Python code with three lines. The first line is a comment, highlighted with a light blue box, and an arrow points from the label 'Comment' to it. The second line is `print("This is a string")`, where the string literal `"This is a string"` is highlighted with a light green box and an arrow points from the label 'String' to it. The third line is `x = y + z`, which is highlighted with a light red box and an arrow points from the label 'Code' to it. The comment line itself is `# This is a comment`, with the hash and the text highlighted by the blue box.

```
# This is a comment
print("This is a string")
x = y + z
```

Comment

String

Code

Different types of brackets and quotes

() Parentheses

[] Square brackets

{ } Curly brackets

' ' Single quote

" " Double quote

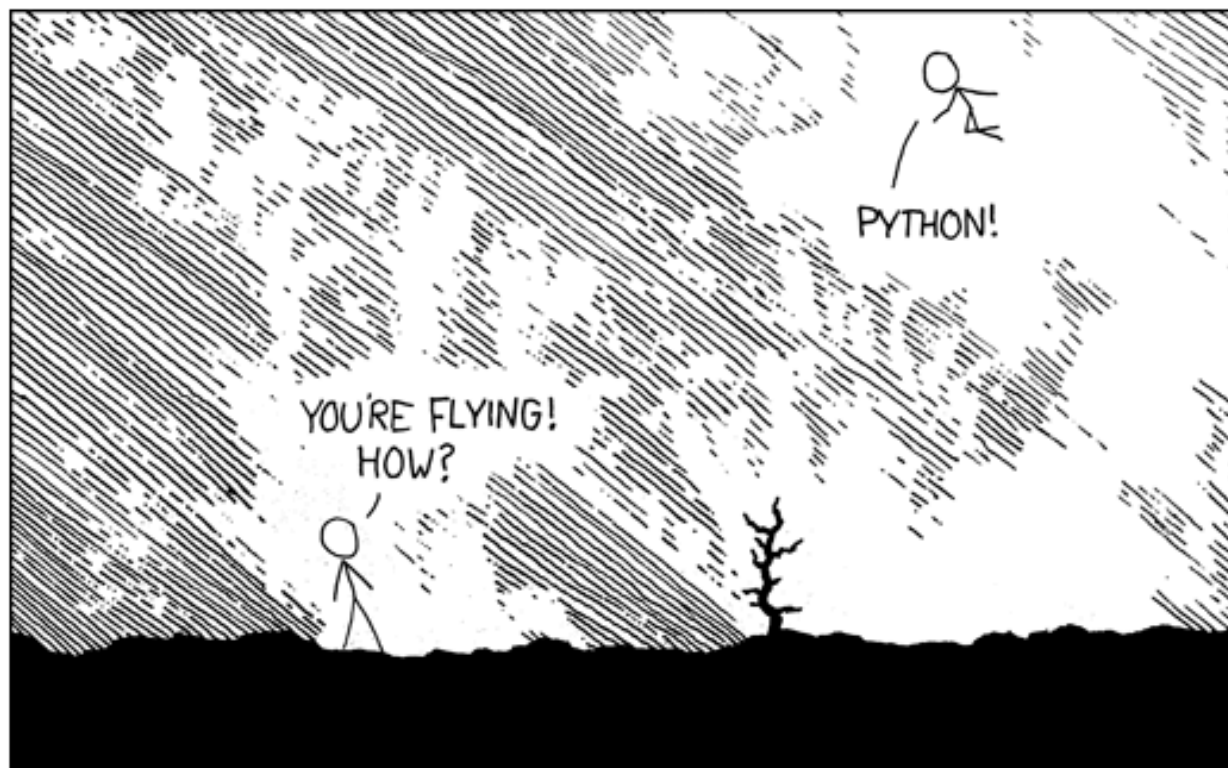
` ` Back quotes (are not the same as single quotes!)

What about shell scripting?

We started learning about variables and for loops with bash shell scripts. Why can't you use those for everything?

- Shell scripting tends to be used for basic tasks
- Good for automating command line actions
- Good for file manipulations
- Not for building software
- Few sophisticated data structures
- Few libraries of functions





xkcd

