Data science for everyone

Prof. Jones-Rooy & Prof. Policastro

Feb. 12, 2020

3.2: Data types & sequences

ANNOUNCEMENTS

1. Assignments
    1. Lab 0 due today, Feb. 12, 8p
    2. Lab 1 out today, Feb. 12, 8p
    3. Homework 1 due Tue., Feb. 18, 8p
    4. Lab 1 due Wed., Feb. 19, 8p
2. For coding-intensive lectures you can find accompanying Jupyter notebooks on JupyterHub & Classes
    1. You don't need to use these, just there to support!
    2. Your TAs have put together lots of great materials, so if that's sufficient (!) for you, that's all you need to use!

Outline

1. Data types & sequences
2. Zooming out on packages & modules
3. Why Python? (and programming tips)

Ch. 3 of *Inferential Thinking*
(and all over the Internet!)

LECTURE 3.1

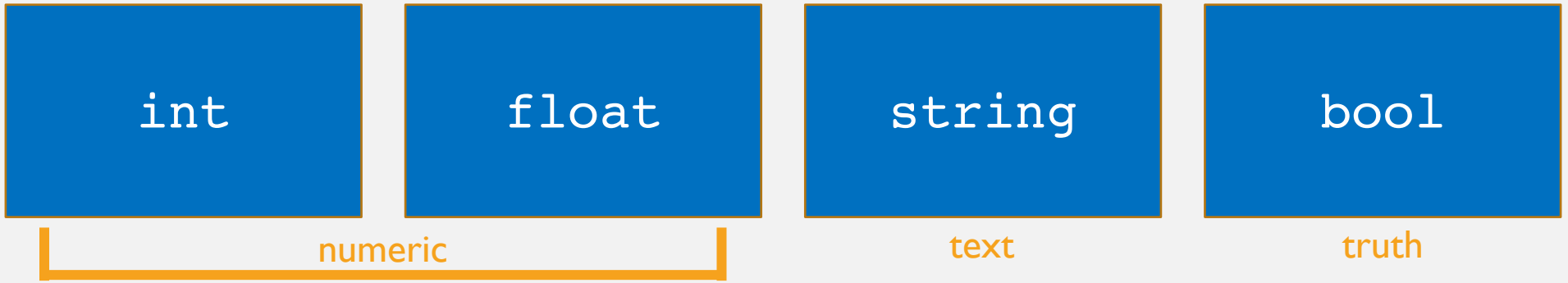
Arithmetic	Naming objects	Built-in functions	Modules & packages
`12+8*4`	`price = 2+3`	`abs(-6)`	`import math`

See also Lecture 3.1—Example Code on Classes & JupyterHub

Ch. 4 of *Inferential Thinking* (and all over the Internet!)

Read more here!

DATA TYPES



NUMERIC DATA

- When our data is numeric, we can do mathematical and statistical operations on it (like find the mean or the maximum)
  - Note that `int / int = float`; and `float + int = float`
- Why one or the other?
  - For our purposes, it's more about using data types that reflect the data we have and what it represents (e.g., if we are counting people, we shouldn't use floats)
    - Can also be clarifying if someone else is using our code
  - Speed and efficiency are the other big considerations – integer math is faster, integers typically consume less memory
- Integers can be of any length; floats are accurate up to 15 decimal places
  - Rounding errors – rarely substantively problematic
  - Not unique to Python – though Python `float` = other programs' `double`
- Two other numeric types (we won't use them, but just so you know!)
  - Long integers (`long`): Integers of infinite length (150L) ← this is gone in Python 3!
  - Complex numbers (`complex`): Imaginary numbers (a+bj where a and b are floats and j is the square root of 1)

STRING AND BOOLEAN DATA

- Strings
  - We'll encounter strings when we work with text data
  - And when we work with data that has symbols that were probably put in by someone who isn't a data scientist (e.g., $3.04 rather than 3.04)
  - We write them surrounded by quotes: `"mystring"` or `'mystring'`
  - We can do some operations on them; e.g., `"new" + "york" = "newyork"`
- Bools
  - Can only take on `True` and `False` (capitalization is important)
  - They are not strings (and there are no quotes)
  - A Boolean expression is an expression that evaluates to a Boolean value
    - `4==4` will give us `True`
    - `4==9` will give us `False`

More on bools here!

INSPECTING & CHANGING TYPES

- When working with data we will often want to know what type of data we are working with (preview of coming weeks: we will think of our data in terms of **variables**)
- A common version of this is we try to perform a mathematical manipulation on a column of data that we think is numeric and it turns out we can't because it's a string
- Inspecting & changing types is relatively straightforward:
  - `type(myvar)` ← gets you the type of the variable or named object you're interested in
  - `int(myvar)` ← turns it into an `int`
  - `str(myvar)` ← turns it into a `string`
  - `float(myvar)` ← you guessed it!

FYI: Other types of sequences & containers out there: tuples, ranges, sets, dictionaries

SEQUENCES

Lists

Arrays

ways to store data

any data type (int., string, etc.)

can be indexed & iterated through

More common

Built into Python

Have to be declared

Can perform arithmetic on them

Data stored more compactly, efficiently

LISTS

- Represented by square brackets with each item (or element) separated by commas
  - `mylist = [1, 5, 8, 3, 7]`
- Holds these items in that order
- We can add and remove elements from the sequence
  - `mylist.append(4)` will put 4 at the end of `mylist` – so it will be `[1, 5, 8, 3, 7, 4]`
  - **`del`** `mylist[1]` will remove the second element from the sequence – so we get `[1, 8, 3, 7, 4]`
- Can contain more than one type of data
  - `myotherlist = [1, 5, 'apple', 'steve']`
- We can inspect individual elements at a time
  - `mylist[0]` will show me the first element in `mylist` – so I will get `1`
  - Challenge: this is working from left to right. What would we need to write to label elements from right to left? (e.g., to tell me the LAST element in `mylist` without counting how many elements there are?)
- We can manipulate lists a bit, but we don't have as much functionality compared to arrays
  - `mylist*2` will give me mylistmylist – so it will be `[1, 5, 8, 3, 7, 1, 5, 8, 3, 7]`

ARRAYS

- Like lists, they are collections of elements
- Unlike lists, all elements must be of the same type
- Arrays are not built in, so we need to import a package or module for array functionality
  - There are lots of these, but we will use `numpy`
  - We write them inside a parentheses and square bracket, e.g., `([3, 7, 4])`
- Creating an array:
  - `import numpy as np`
  - `myarray = np.array([3, 7, 4])`
- We have more manipulation functionality with arrays
  - `myarray*2` will multiply every element in the array by two (gives us `([6, 14, 8)])`
  - Compare this to what `mylist*2` does!
- We can also check the type of sequences:
  - `type(myarray)` will give us `numpy.ndarray`
  - `list(myarray)` will turn our array into a list

Here's another package for arrays if you're curious

More examples of np.array in action here!

See also Lecture 3.2—Example Code on Classes & JupyterHub

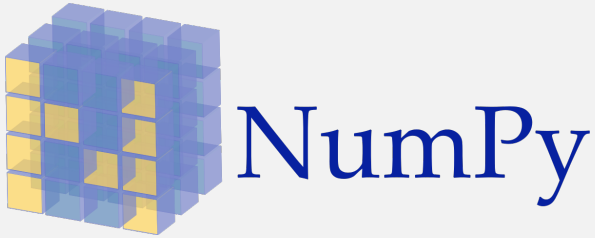
Outline

1. Data types & sequences
2. Zooming out on packages & modules
3. Why Python? (and programming tips)

ZOOMING OUT ON PACKAGES AND MODULES

- A piece of software that has a specific functionality
- We can import them using the `import` function
  - Mainly so we don't have to write a bunch of code ourselves
  - We can also write them ourselves and store a program as a module
  - When we import, if we are likely to use them often, we prefer to abbreviate the name
    - `import numpy as np` ← np is generally the agreed-upon standard
- Package: A collection of modules
- Why not just import tons of modules/the entire package always?
  - Simplicity
  - Reusability
  - Robustness

Some (very) optional resources if you want to read more are here and here.



Numerical Python:
Scientific computing, mathematical functions, multi-dimensional arrays

Scientific Python:
Scientific computing, mathematical functions, complex computing of numerical data, popular in ML

Mathematical plotting:
Plotting library for Python and NumPy

How do you pronounce "matplotlib"

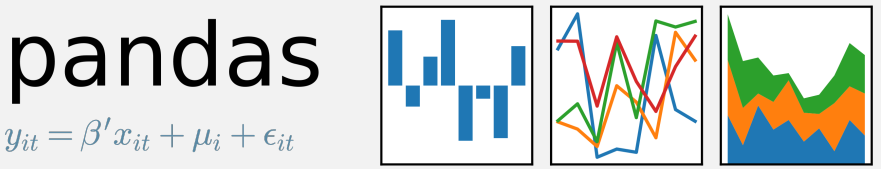
- matte plɒt ˈlɪb (as in "liberty")

or

- matte plɒt ˈlaɪb (as in "library")

My housemate pronounces it one way, and a guy I work with pronounces it the other.

SciPy Toolkit:
Machine learning; classification, regression, clustering algorithms; works with SciPy & NumPy; created as Google Summer of Code project in 2007



Panels Data:
Data structures, data manipulation, and data analysis

A few other fun ones…!

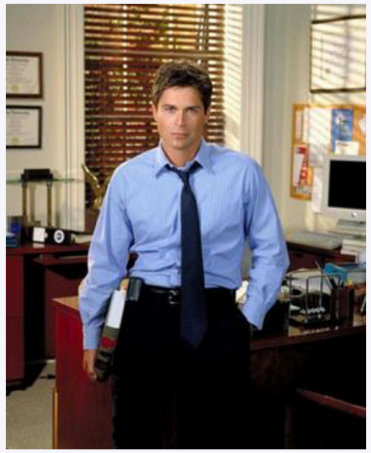
Interactive plots

Samuel Nelson Seaborn (?)
Statistical graphics; builds from matplotlib, integrates with pandas

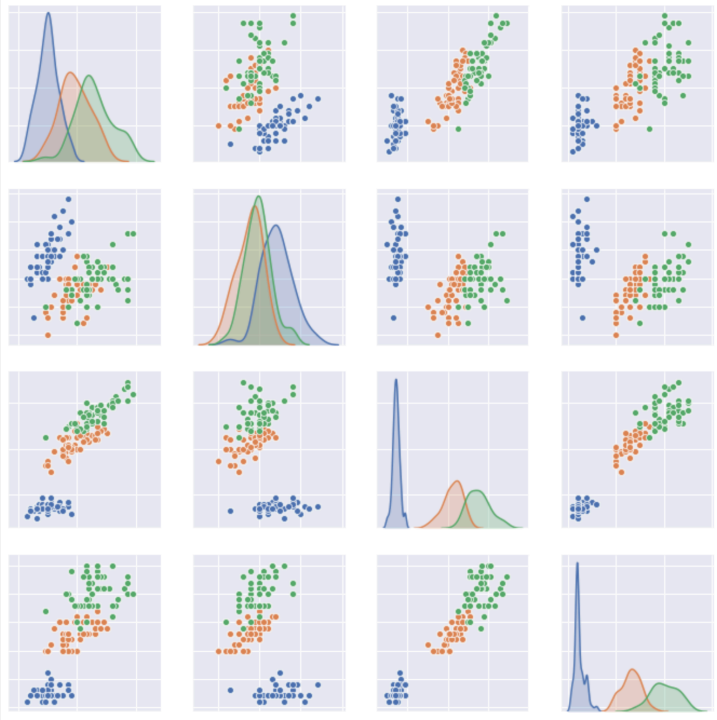
Seaborn

Samuel Seaborn

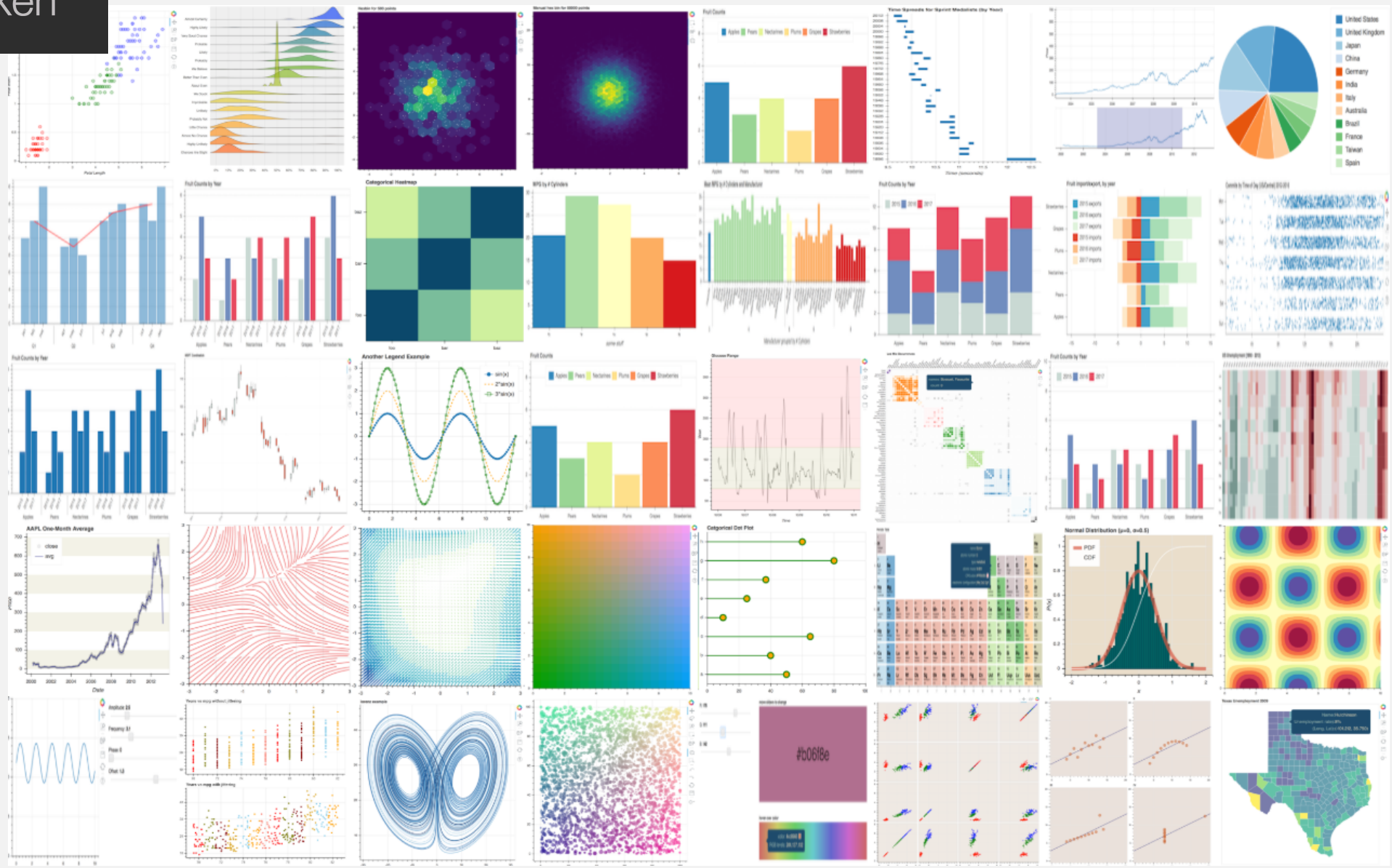
The West Wing character



Rob Lowe as Sam Seaborn



Bokeh


Categorical Heatmap
Another Legend Example
AAPL One-Month Average
Categorical Dot Plot
Normal Distribution (μ=0, σ=0.5)
PDF
CDF
#b0618e

IN THIS COURSE

- Pandas!
- NumPy!
- Datascience in textbook
- A few more here and there for **fun**
- You are encouraged to explore on your own!
- Each module has its own vocabulary and (sometimes) syntax, but it's all compatible with and follows the logic of the Python language
- As with computer languages, many of these modules/libraries have overlapping capabilities
- The "best" one may depend on your goals, the broader project, familiarity, other constraints

Learning by doing!

(Key in programming and most things!)

Outline

1. Data types & sequences
2. Zooming out on packages & modules
3. Why Python? (and programming tips)

PYTHON

Guido van Rossum (Benevolent Dictator For Life) (Netherlands)

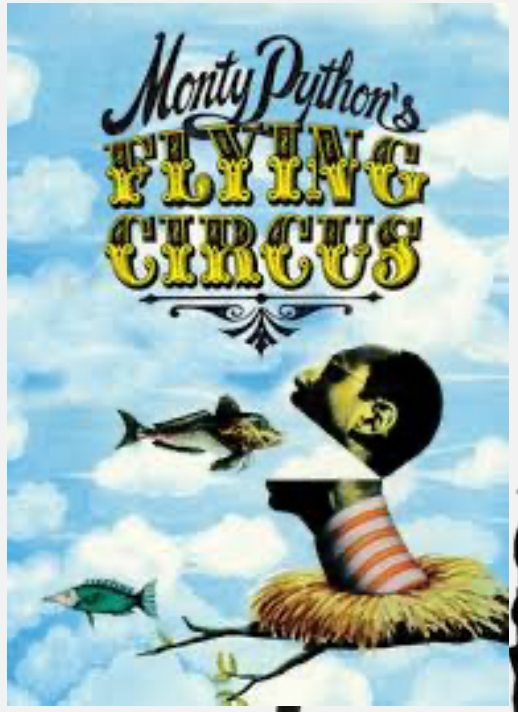
Developed in late 1980s
First implementation: 1989

Python 2.0: 2000

Python 3.0: 2008

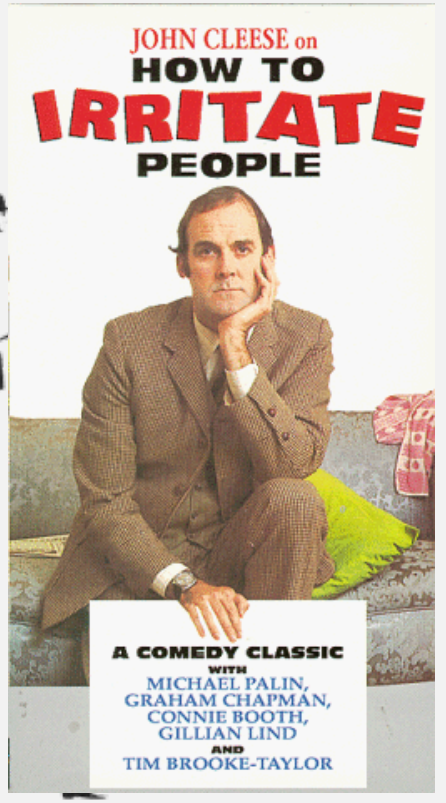
Philosophy:

- Simple
- Uncluttered
- Fun to use



Monty Python

Ministry of Silly Walks



The Zen of Python

Guiding principles for Python's design
20 aphorisms
Only 19 have been written down

```
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!
```

PYTHON

1. High level
2. Literate programming
3. General purpose
4. Interpreted
5. Free and open source
6. Libraries
7. Many users
8. Many packages for scientific computing

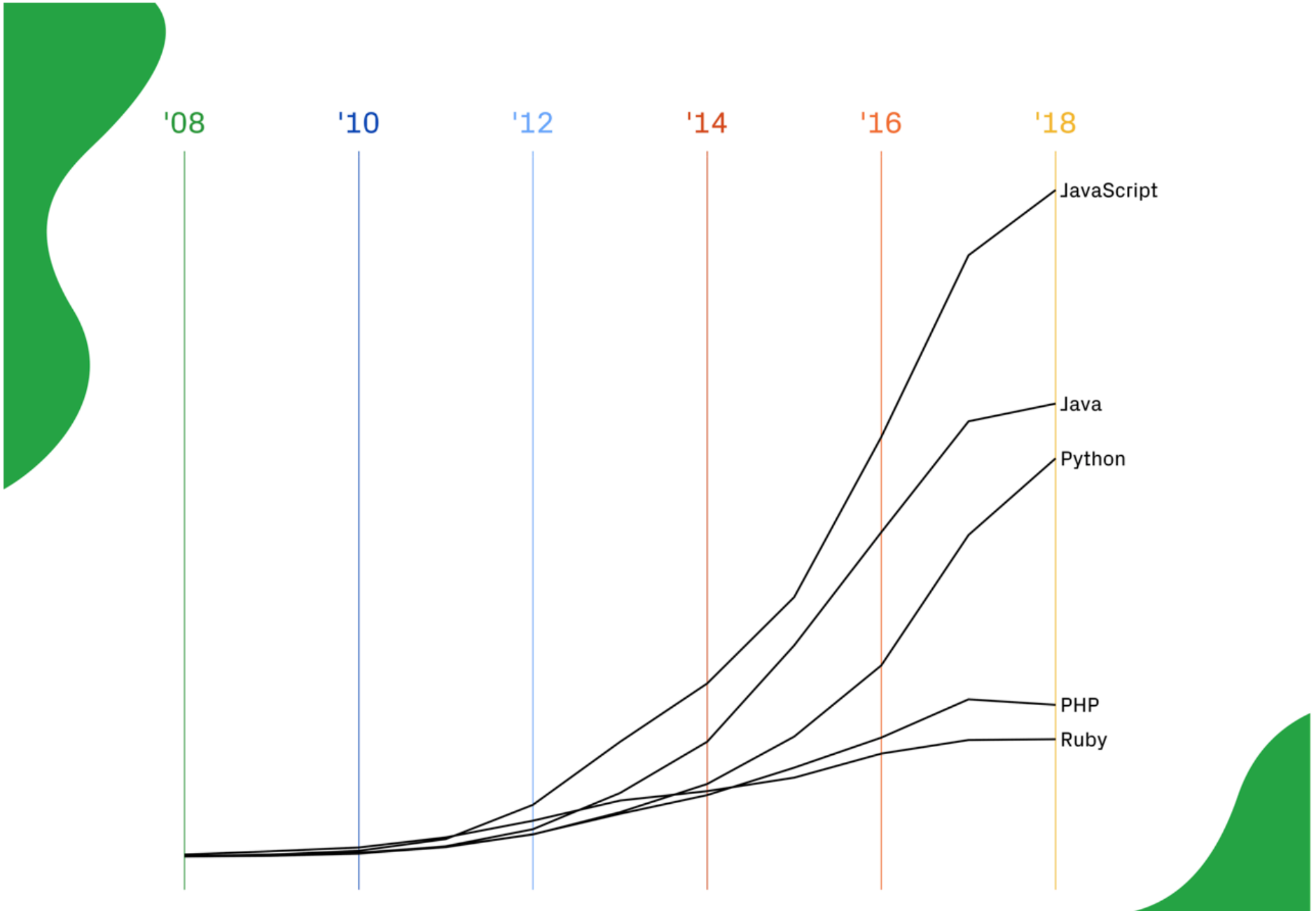
1. You don't have to think about underlying computational details
2. English prose and math-forward language; whitespace-based
3. All problems for computers: AI, stats, many tasks in one workflow
4. Immediate evaluation of code, no compiling; more experimentation
5. Always being improved, use anywhere, for anything
6. Standard & 3rd-party; great for solving complicated tasks, popular
7. Tons of online support
8. numpy, matplotlib, pandas, cartopy, more

Source: GitHub 2018

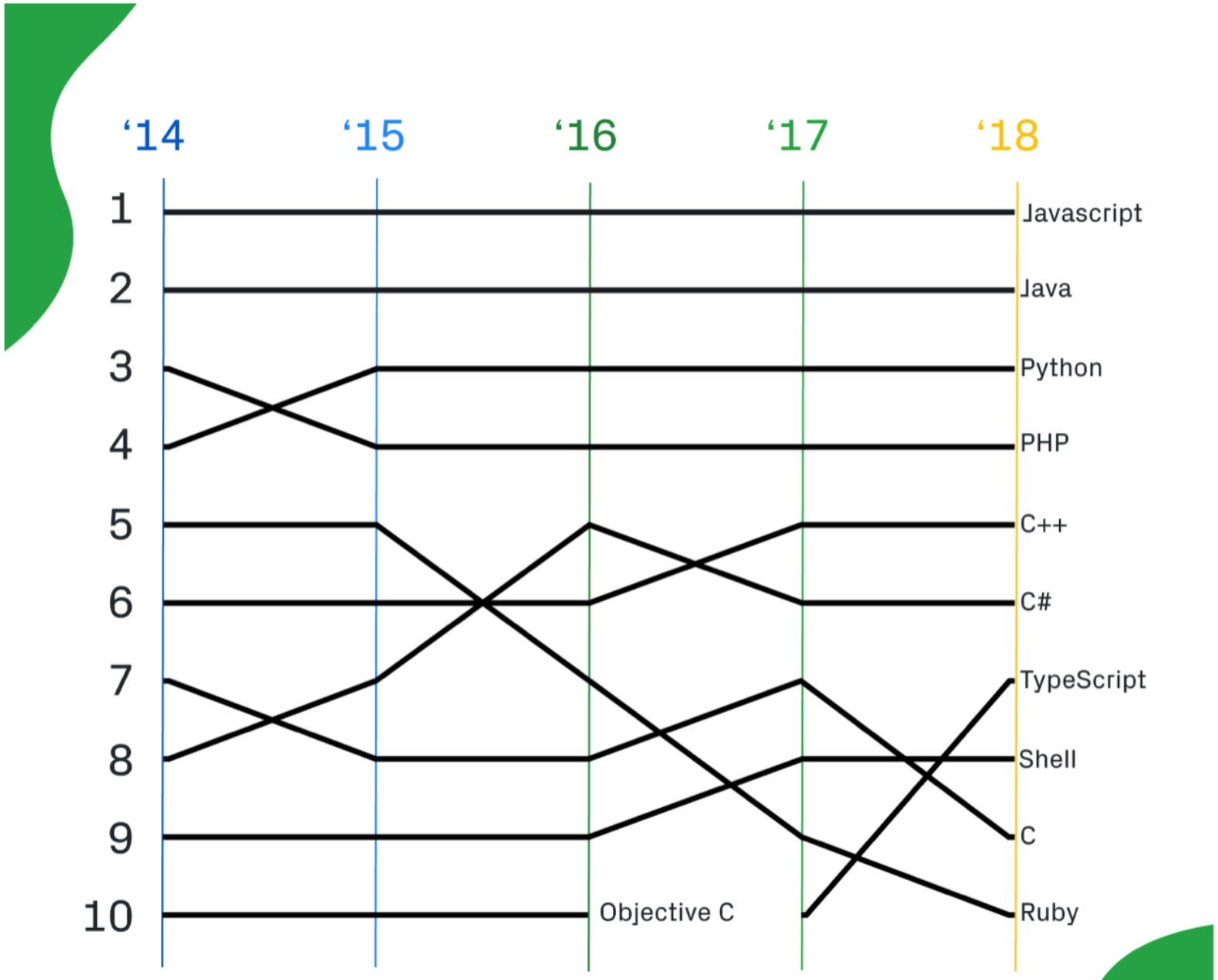
POPULAR COMPUTER LANGUAGES

1.	JavaScript	1. Websites, plugins
2.	Java	2. Large systems, databases, Android apps
3.	Python	3. One of fastest growing, easier to learn, ML, data science
4.	PHP	4. Websites, not considered that good
5.	C++	5. Since 1970s! Core of many operating systems, browsers, games
6.	C#	6. ("C sharp") developed by Microsoft, similar to Java
7.	TypeScript	7. JavaScript but better, also created by Microsoft
8.	Shell	8. Shell scripts tell operating systems to run certain commands
9.	C	9. Since 1970s! Systems level programming
10.	Ruby	10. Ruby on Rails (web framework), simple, used for SoundCloud, Zendesk, Square, GitHub

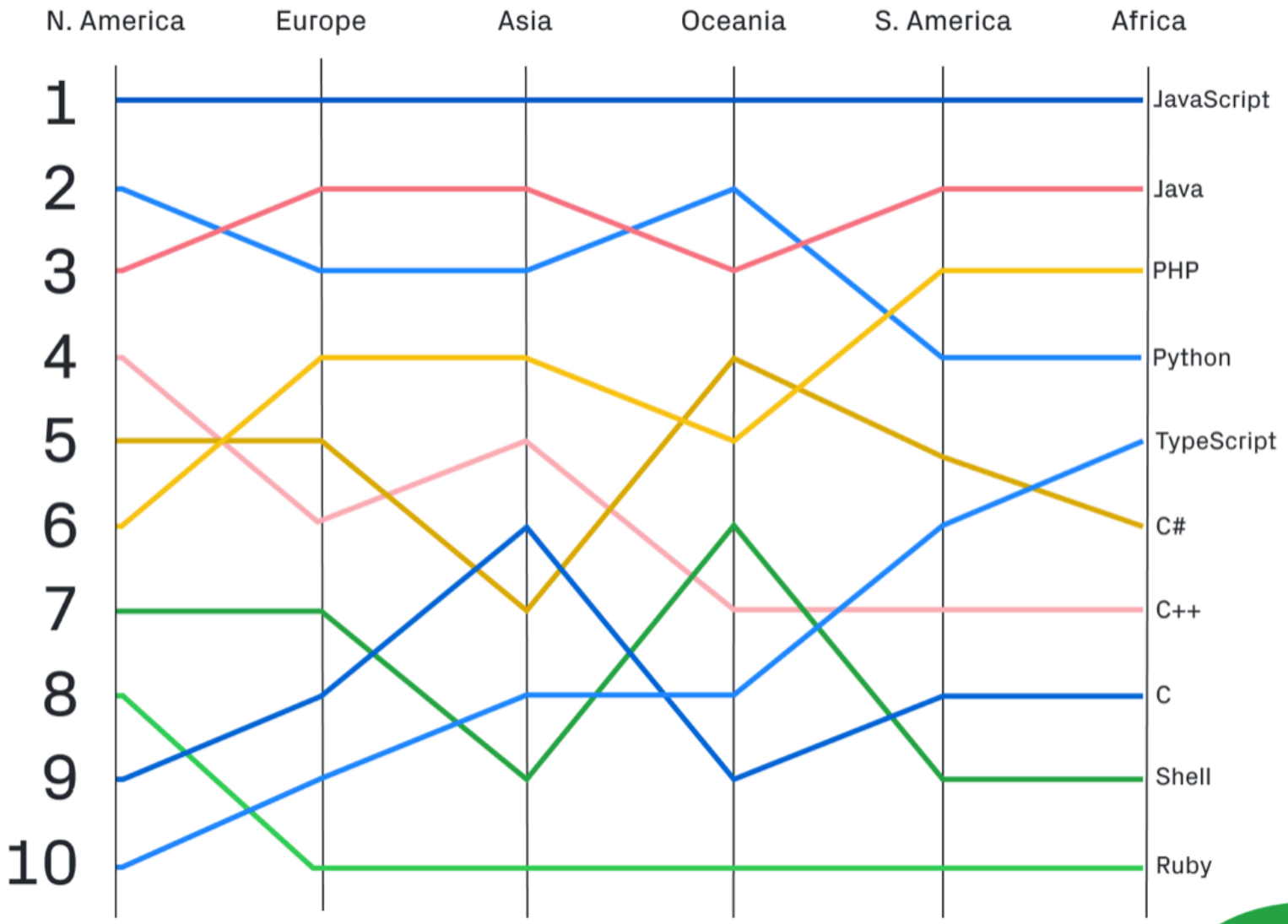
Top programming languages by repositories created, 2008-2018



Top programming languages by contributors as of September 30, 2018

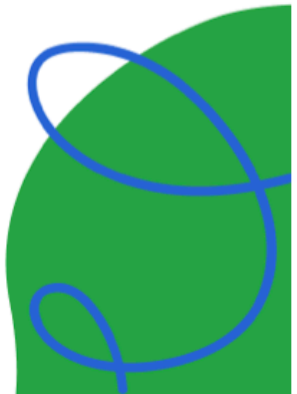


Geographic trends in languages by contributors as of September 30, 2018



Fastest growing languages by contributors as of September 30, 2018

		Growth in contributors
1	Kotlin	2.6x
2	HCL	2.2x
3	TypeScript	1.9x
4	PowerShell	1.7x
5	Rust	1.7x
6	CMake	1.6x
7	Go	1.5x
8	Python	1.5x
9	Groovy	1.4x
10	SQLPL	1.4x



CHOOSING BETWEEN PROGRAMMING LANGUAGES

- Whether it's designed for the project you're working on
  - E.g., Creating a computer game, analyzing data, building a website
- Popularity
  - More likely to have broader uses outside of what you learned it for
  - More likely to be well-documented
  - More likely to have more useful packages & add-ons
  - More likely to be supported in forums, e.g., Stack Overflow
- Why choose a less popular language?
  - If the project you're working on has very specific requirements and needs
  - You like learning programming languages
- Summary: We are usually trading off between usefulness for a specific task vs. general applicability

TIPS FOR LEARNING PYTHON

(and anything!)

- Code every day
  - Consistency, repetition, muscle memory
- Write – even by hand if you can!
  - Improves retention
  - Good practice for bigger projects on whiteboards
- Be interactive
  - Do the reading alongside a notebook and try things out
  - Do not be afraid to make mistakes; trial & error is your friend

Build something (anything)!

- Take breaks
  - Pomodoro method, esp. while debugging
- Practice fixing
  - Use the scientific method
- Be around others who are learning
  - Ask questions
  - Challenge each other
  - Teach each other
  - Pair programming
  - Ask good questions online (if you go that route)
  - Contribute to open source projects

! Most important, and often most difficult part: (1) Knowing what statistical analyses to conduct in the first place, (2) Knowing how to think about and evaluate the quality and limitations of your data, and how it affects your conclusions! !

EXAMPLE PROBLEM IN DATA SCIENCE

- Problem: Finding out the relationship between two variables
- Possible sub-problems:
  - Finding reliable data that captures the variables of interest in an acceptable way
  - If two datasets making sure they are compatible, or making them compatible, and joining them together
  - Cleaning the data to consistently account for errors, nuances of coding (e.g., how is missing data accounted for?)
  - Narrowing to the sample of interest (e.g., new variable, just certain years, only values over a certain level)
  - Generating a visualization and/or conducting a statistical analysis to better understand the relationship between these variables

What will likely a lot of time

Fair amount of coding here

What many think is the primary work

SUMMARY

1. **Programming**: A series of instructions to tell a computer how to solve a problem (divided into smaller problems)
2. **Python**: One of many computer languages, very versatile and user-friendly, quite popular, not necessarily the fastest for some projects
3. **Things to be comfortable with in Python so far**
    1. **Building blocks**: arithmetic, naming objects/assigning names, built-in functions, modules & packages
    2. **Data types**: int, float, string bool; inspecting types, changing types
    3. **Sequences**: lists and arrays; inspecting elements, changing elements, arithmetic manipulations and how they are different between the two

Reminder!

For extra support: There are two lab notebooks that accompany this lecture! Find them on Classes and JupyterHub

Outline

1. Data types & sequences
2. Zooming out on packages & modules
3. Why Python? (and programming tips)