

Python Introduction

Objectives

- Python - brief history
- Python popularity, and use in Data Science
- Python 2 vs. 3
- Work environment
- Data types and data structures
- Coding guidelines
- Python good practice

Python - a brief history

- In the late 1980's Guido van Rossum conceived and started to implement Python as a successor to the [ABC programming language](#). Guido said he needed “a decent scripting language.” Python itself named from *Monty Python's Flying Circus*.
- In 1994 Python 1.0 was released. Some functional programming tools - lambda, reduce(), filter(), map() - were added courtesy of a Lisp hacker.
- Python 2 was released in 2000 with the help of a more transparent, community-based development process ([Python Software Foundation](#)). Introduced list comprehensions and generators.
- Python 3 was released in 2008. Had an emphasis on removing duplicative constructs and modules. It's a backward incompatible release, though many of its major features have been back-ported to Python 2.
- EOL date for Python 2 was originally set for 2015, been extended to 2020.



Guido,
named BDFL
by Python
community

Python popularity

- General purpose programming language that supports many paradigms
 - imperative, object-oriented, procedural, functional
- Interpreted, instead of compiled
 - has rapid REPL (Read-Evaluate-Print-Loop)
- Design philosophy emphasizes code readability
 - white space rather than brackets/braces determine code blocks
 - [Zen of Python](#)
- Efficient syntax
 - fewer lines of code needed to do the same thing relative to C++, Java
- Large development community
 - large and comprehensive standard library (NumPy, SciPy, Matplotlib, Pandas, Scikit-Learn)
 - open-source development ([Python on Github](#))

Python for Data Science

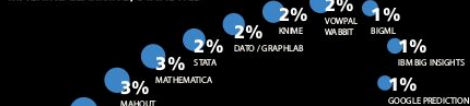
O'REILLY™

2016 Data Science Salary Survey

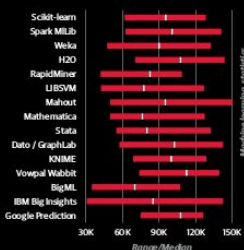
Tools, Trends, What Pays (and What Doesn't) for Data Professionals



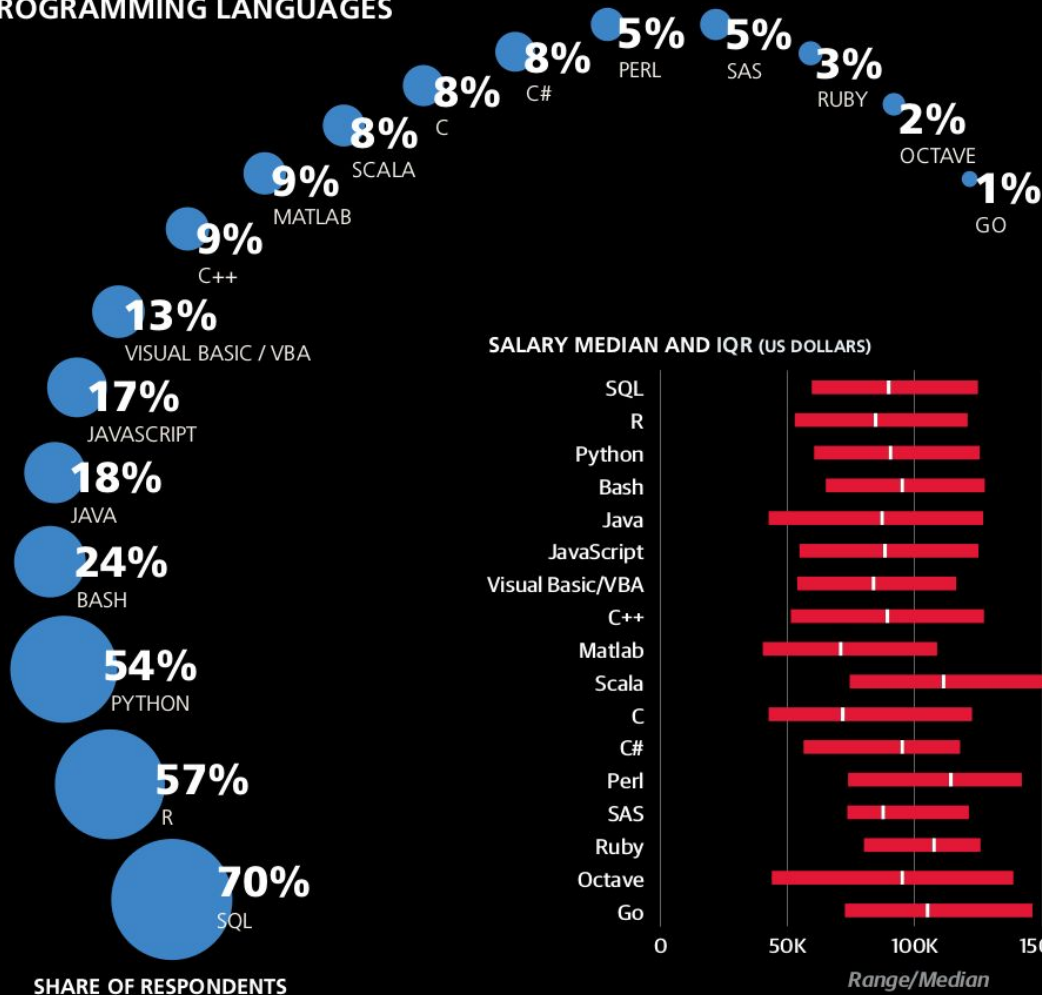
MACHINE LEARNING, STATISTICS



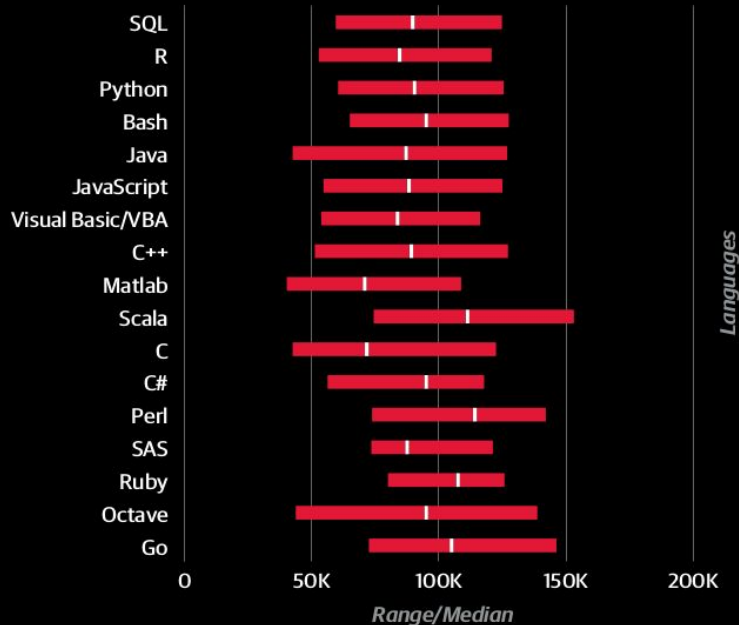
SALARY MEDIAN AND IQR (US DOLLARS)



PROGRAMMING LANGUAGES



SALARY MEDIAN AND IQR (US DOLLARS)



Python 2 vs. 3

Principle of Python 3: "reduce feature duplication by removing old ways of doing things"

Python 2	Python 3
<code>print 'Hello World'</code>	<code>print('Hello World')</code>
<code>3 / 2 = 1, (3 / 2.) = 1.5</code>	<code>3 / 2 = 1.5, 3 // 2 = 1</code>
types: <code>str()</code> , <code>unicode()</code>	type: <code>str()</code>
<code>range(n)</code> - makes a list <code>xrange(n)</code> - iterator	<code>range(n)</code> - iterator <code>list(range(n))</code> - makes a list
<code>.items()</code> - makes a list <code>.iteritems()</code> - iterator	<code>.items()</code> - iterator
<code>map()</code> - makes a list	<code>map()</code> - map object <code>list(map())</code> - makes a list
<code>my_generator.next()</code>	<code>next(my_generator)</code>

Python 2 vs. 3

- You need to be able to work in both
 - use [conda](#) to create an environment in your Anaconda distribution
 - `$ conda create -n py2 python=2 anaconda` (if you have Python 3 installed)
 - `$ conda create -n py3 python=3 anaconda` (if you have Python 2 installed)
 - `$ source activate py2` (or `py3`)
 - `$ source deactivate`

Work environment

- Opinions differ. I recommend a Text editor - IPython - Terminal combination
 - Demonstrate it
- Jupyter Notebooks are great for classroom breakouts, presenting results, and EDA but are a poor development environment. They can be addictive! Don't let it happen to you.
- IDEs such as PyCharm are nice, but you need to be comfortable working in Terminal first. On AWS all you have is Terminal. Transition to an IDE later.
- In the end, you need to be able to write code that executes from the command line:
 - `$ python script.py datafile.csv`

Python data types and data structures

Data types

Data structures

TYPE	DESCRIPTION	EXAMPLE VALUE(S)
int	integers	1, 2, -3 ...
float	real numbers, floating values	1.0, 2.5, 102342.32423 ...
str	strings	'abc'
tuple	an immutable tuple of values, each has its own type	(1, 'a', 5.0)
list	a list defined as an indexed sequence of elements	[1, 3, 5, 7]
dict	a dictionary that maps keys to values	{ 'a' : 1, 'b' : 2 }
set	a set of distinct values	{1, 2, 3}

More on data structures

- **Lists:** ordered, dynamic collections that are meant for storing collections of data about disparate objects (e.g. different types). Many list methods. (type list)
- **Tuples:** ordered, static collections that are meant for storing unchanging pieces of data. Just a few methods. (type tuple)
- **Dictionaries:** unordered collections of key-value pairs, where each key has to be unique and immutable (type dict) Hash map associates key with the memory location of the value so lookup is fast.
- **Sets:** unordered collections of unique keys, where each key is immutable (type set). Hash map associates key with membership in the set, so checking membership in a set is fast (much faster than a list).

Breakout

```
$ jupyter notebook data_structures.ipynb
```

Coding guidelines

- In Python whitespace indentation sets scope.
 - In your text editor set a tab to 4 spaces
- Use `if __name__ == '__main__':` blocks to separate classes/functions from code you wish to execute. Imagine a file named `functions.py` and we import the file into another script. Upon import

`functions.py`

```
def my_func(number):  
    return number + 2  
  
print("I'm going to print when you import this file...")
```

`functions.py`

```
def my_func(number):  
    return number + 2  
  
if __name__ == '__main__':  
    print("I'm only going to print when you run the file!")
```

I'm going to print when you import this file...

Coding guidelines - checking speed of execution

```
$ jupyter notebook timeit_examples.ipynb
```

Python good practice

- **PEP8:** [Style guide for Python](#). Addresses spacing, variable names, function names, line lengths. Highlights:
 - variable and function names are snake_case, classes CamelCase
 - avoid extraneous white space
 - lines not longer than 79 characters
 - documentation!
- **Pythonic code:** [A guideline](#)
 - use `for` loops instead of indexing into arrays
 - use `enumerate` if you need the index
 - use `with` statements when working with files
 - use list comprehensions
 - `(if x:)` instead of `(if x == True:)`
 - and many others (see guide)