Python Introduction

Objectives

- Python - brief history
- Python popularity, and use in Data Science
- Python 2 vs. 3
- Work environment
- Work flow
- Data types and data structures
- Python good practice
- Zen of Python examples

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

https://en.wikipedia.org/wiki/History_of_Python

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)

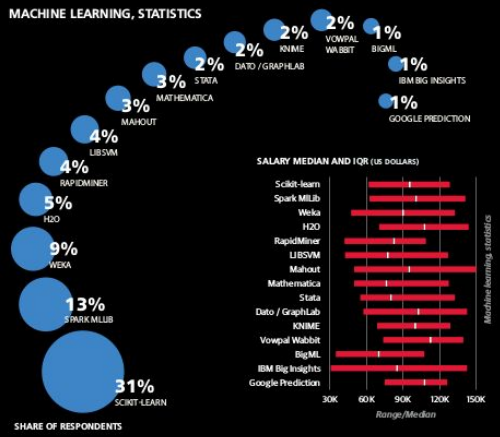
https://en.wikipedia.org/wiki/Python_(programming_language)

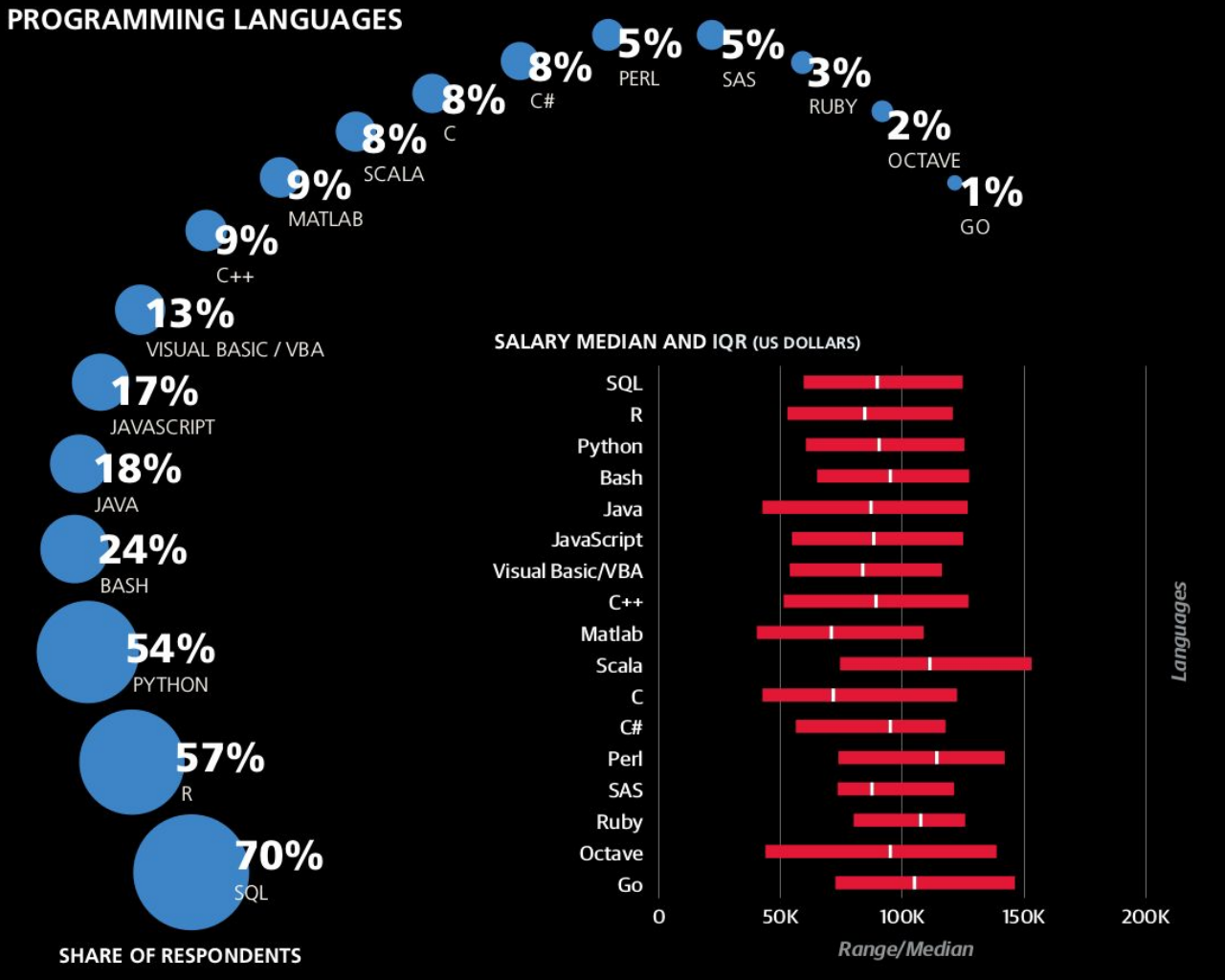
Python for Data Science

O'REILLY

2016 Data Science Salary Survey

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





Python 2 vs. 3

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

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

http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html

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 addicting! 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. Software developer at Pivotal quote: "IDEs are earned."
- In the end, you need to be able to write code that executes from the command line:
  - `$ python script.py datafile.csv`

Suggested work flow

- In a script, start with an `if __name__ == '__main__':` block.
  - for now, just trust me
- `import` whatever you need to above the `if __name__` block, start writing code below.
- In the Ipython console, `run` your code and then check to see if you are getting values you expect.
- If you are getting values you expect, start encapsulating your code into functions (and later classes) above the `if __name__` block.
- `import` these functions (and/or classes) into ipython to make sure they work.
  - (show why the `if __name__` block is useful now)
- Demo - make a deck of cards

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
```

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)

Zen of Python

From within ipython:

In [1]: `import this`

The Zen of Python, by Tim Peters

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.
...

Zen of Python examples

Explicit is better than implicit

```
def make_dict(*args):
    x, y = args
    return dict(**locals())
```

```
def make_dict(x, y):
    return {'x': x, 'y': y}
```

Which one is more explicit?

Zen of Python examples

Explicit is better than implicit

```
def make_dict(*args):
    x, y = args
    return dict(**locals())
```

```
def make_dict(x, y):
    return {'x': x, 'y': y}
```

Which one is more explicit?

Zen of Python examples

Sparse is better than dense

```
if x == 1: print('one')
```

```
if (<complex comparison 1> and
    <complex comparison 2>):
    # do something
```

```
if x == 1:
    print('one')
```

```
cond_1 = <complex comparison 1>
cond_2 = <complex comparison 2>
if (cond_1 and cond_2):
    # do something
```

Which ones are more sparse?

Zen of Python examples

Sparse is better than dense

```
if x == 1: print('one')
```

```
if (<complex comparison 1> and
    <complex comparison 2>):
    # do something
```

```
if x == 1:
    print('one')
```

```
cond_1 = <complex comparison 1>
cond_2 = <complex comparison 2>
if (cond_1 and cond_2):
    # do something
```

Which ones are more sparse?

Zen of Python examples

Sparse is better than dense

```
if x == 1: print('one')

if (<complex comparison 1> and
    <complex comparison 2>):
    # do something
```

```
if x == 1:
    print('one')

cond_1 = <complex comparison 1>
cond_2 = <complex comparison 2>
if (cond_1 and cond_2):
    # do something
```

Which ones are more sparse?

Line continuation

```
french_insult = \
“Your mother was a hamster, and \
your father smelt of elderberries!”
```

```
french_insult = (
“Your mother was a hamster, and “
“your father smelt of
elderberries!”
)
```

Preferred?

Line continuation

```
french_insult = \
“Your mother was a hamster, and \
your father smelt of elderberries!”
```

```
french_insult = (
“Your mother was a hamster, and “
“your father smelt of
elderberries!”
)
```

Preferred?