



Introduction to Python for Data Science

Veerasak Kritsanaphan

About myself?

- Graduated from San Francisco State University in Master of Science in Computer Information System, 1997
- PhD. Candidate at Chulalongkorn University, research focus on Data Science, Big Data, Mobile Computing and Internet of Thing (IOT)
- Assistant Chief Information Officer at Bangkok Hospital Group
- Instructor for Software Park in Data Science, Requirement Discovery and Practical Enterprise Integration
- Instructor for Chulalongkorn University in R and Data Mining

Introduce yourself?

- What is your name/position/role in your organization?
- Tell you a bit about yourself?
- What is your passion?
- What is your expectation for this class?



Slide and Solutions

- ▷ <https://github.com/vkrit/python-for-data-science>

Agenda

Agenda

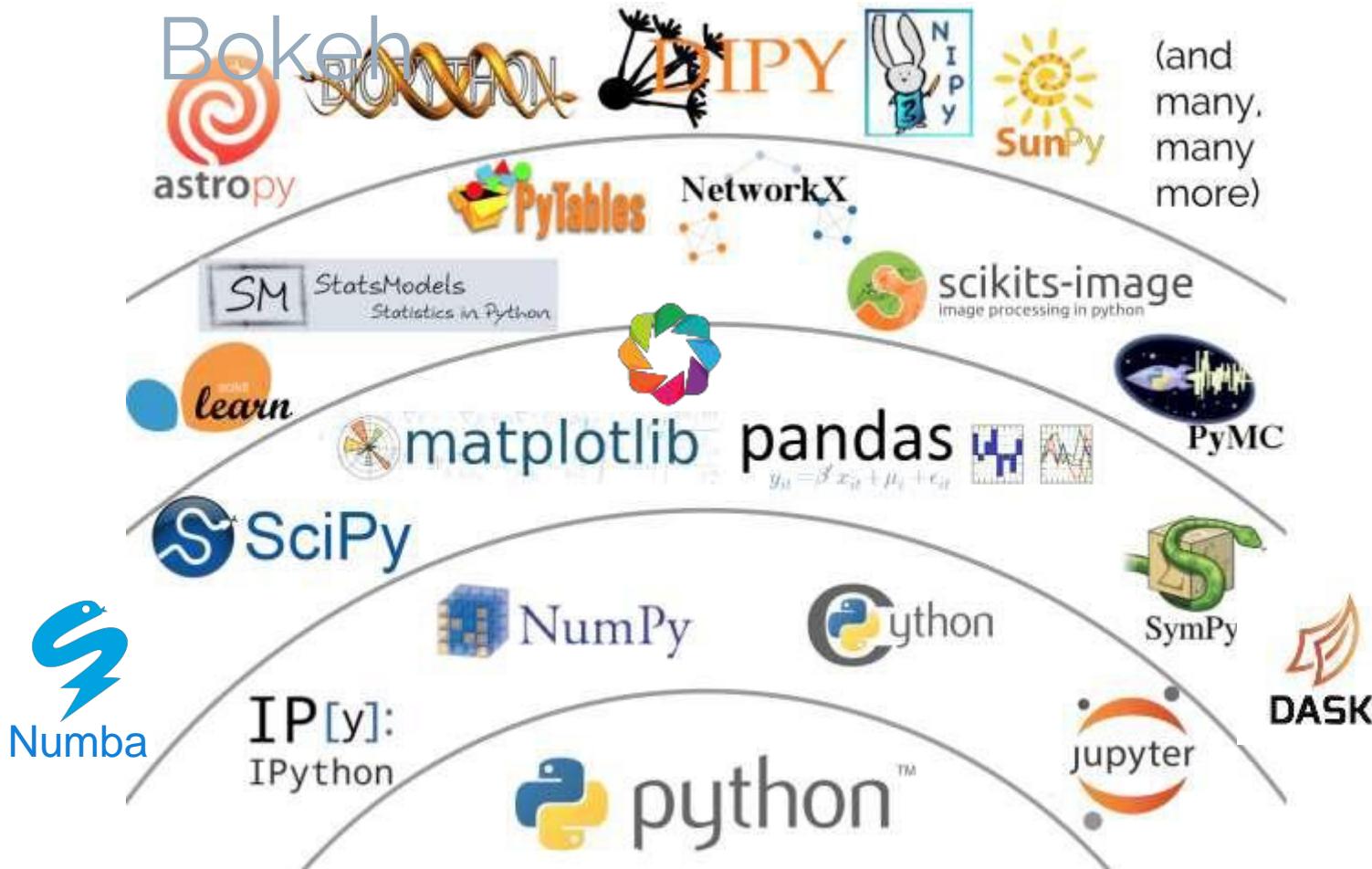
- ▶ Python Eco-system
- ▶ Anaconda
- ▶ Python Scripting
- ▶ Jupyter Notebook
- ▶ Basic Python
- ▶ Data Structure
- ▶ String
- ▶ Iteration
- ▶ Conditional Construct
- ▶ Python Libraries
- ▶ Python Graphics
- ▶ Exploratory Data Analysis using Panda

Agenda

- ▶ Data Wrangling
- ▶ Hypothesis Testing
- ▶ Linear Regression

Python Ecosystem

Bokeh



Anaconda

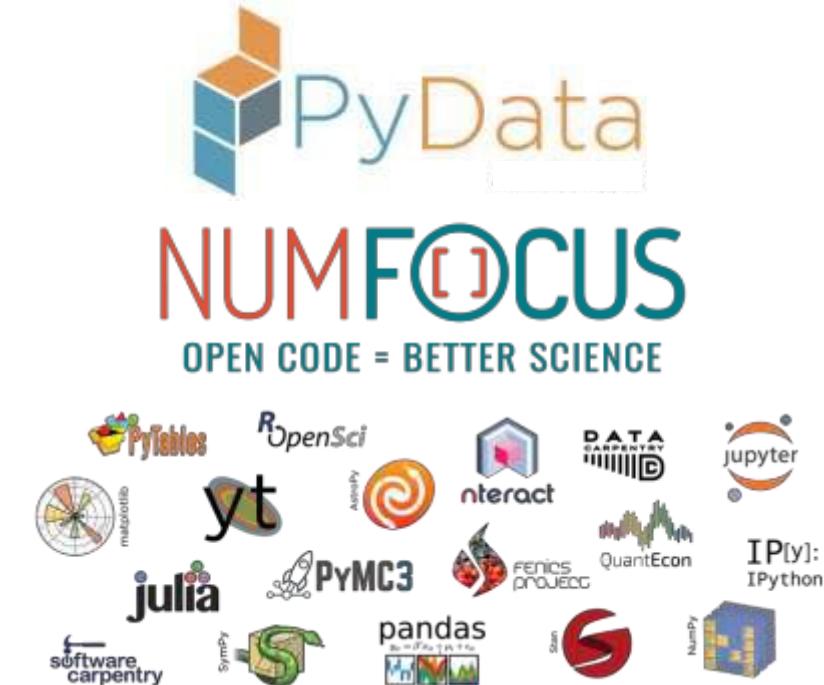
2012 – Anaconda Open Source History

Company

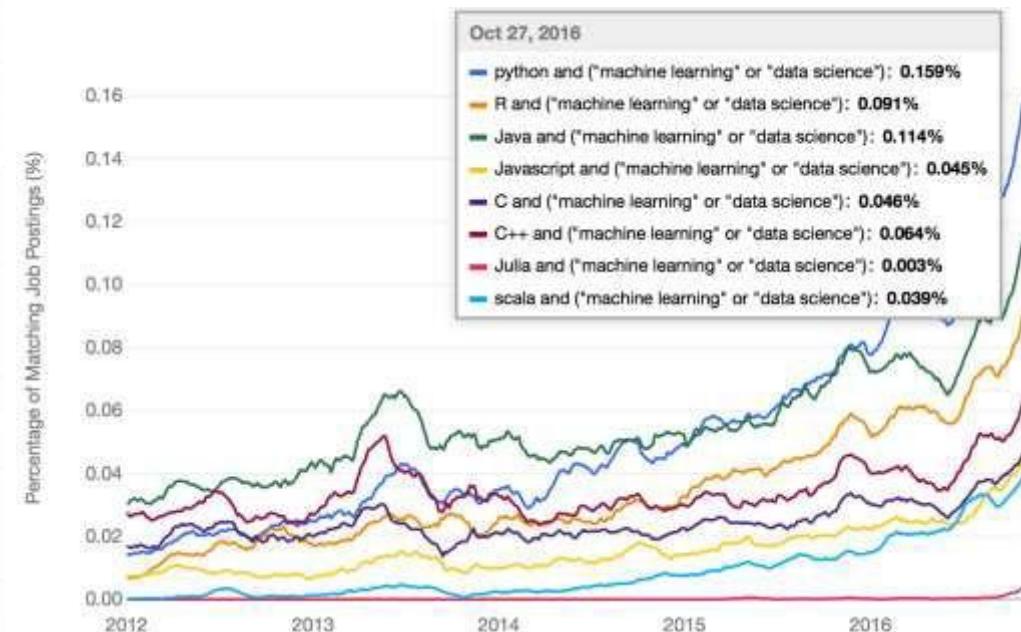
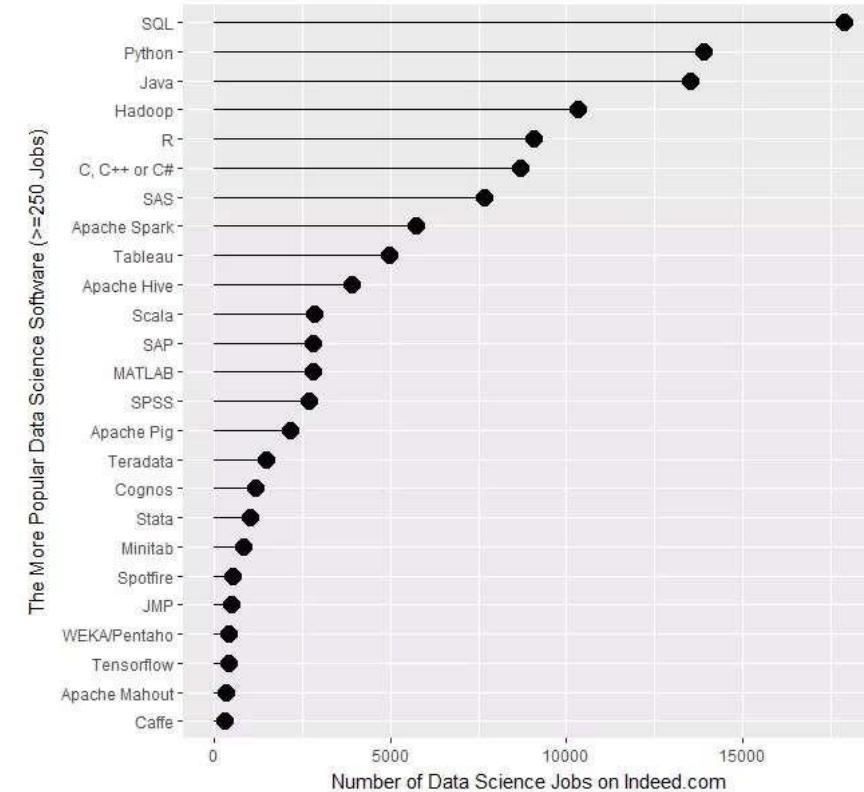


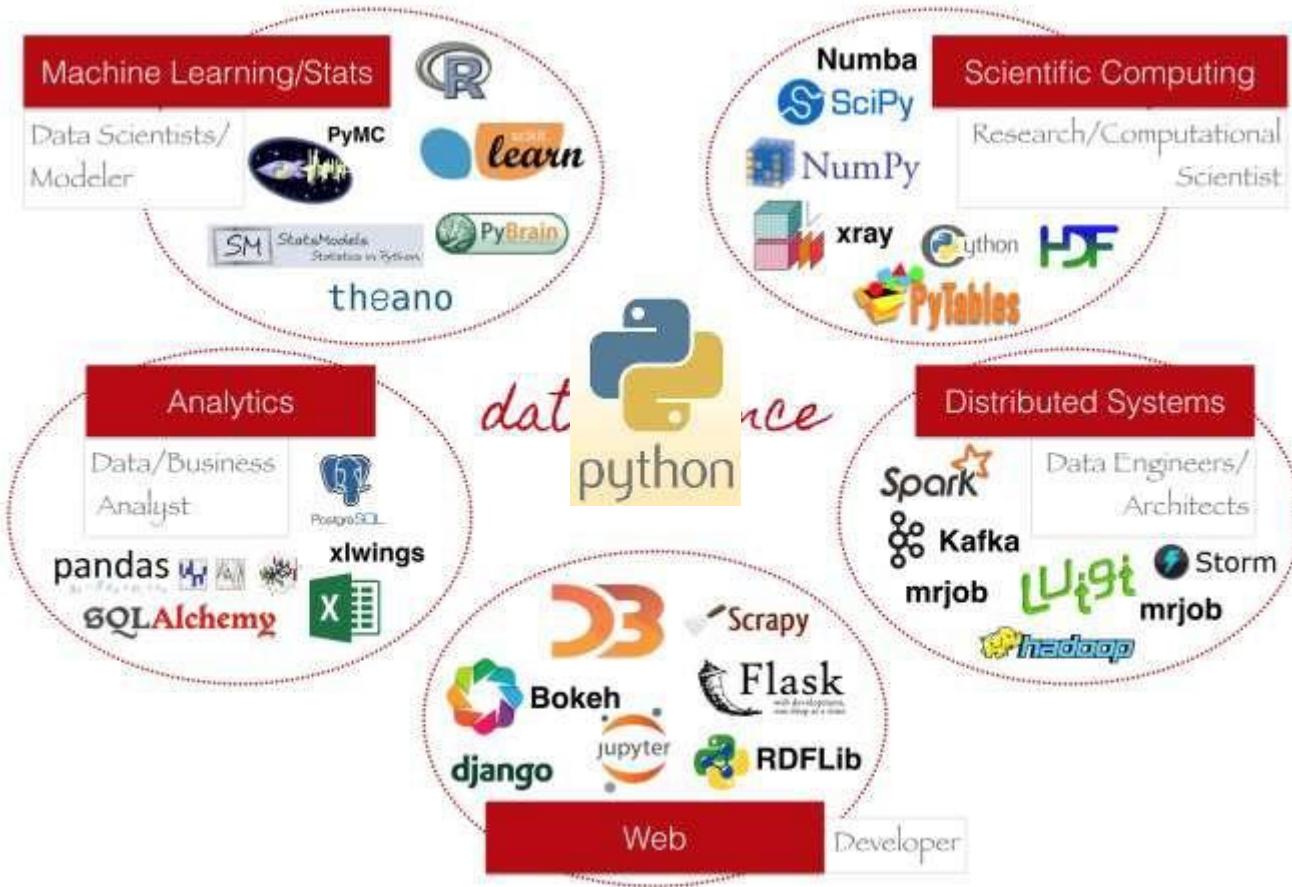
Enterprise **software, support and services** to empower people who change the world to get rapid insight from **all** of their data — built on **open-source** that we contribute to and sustain.

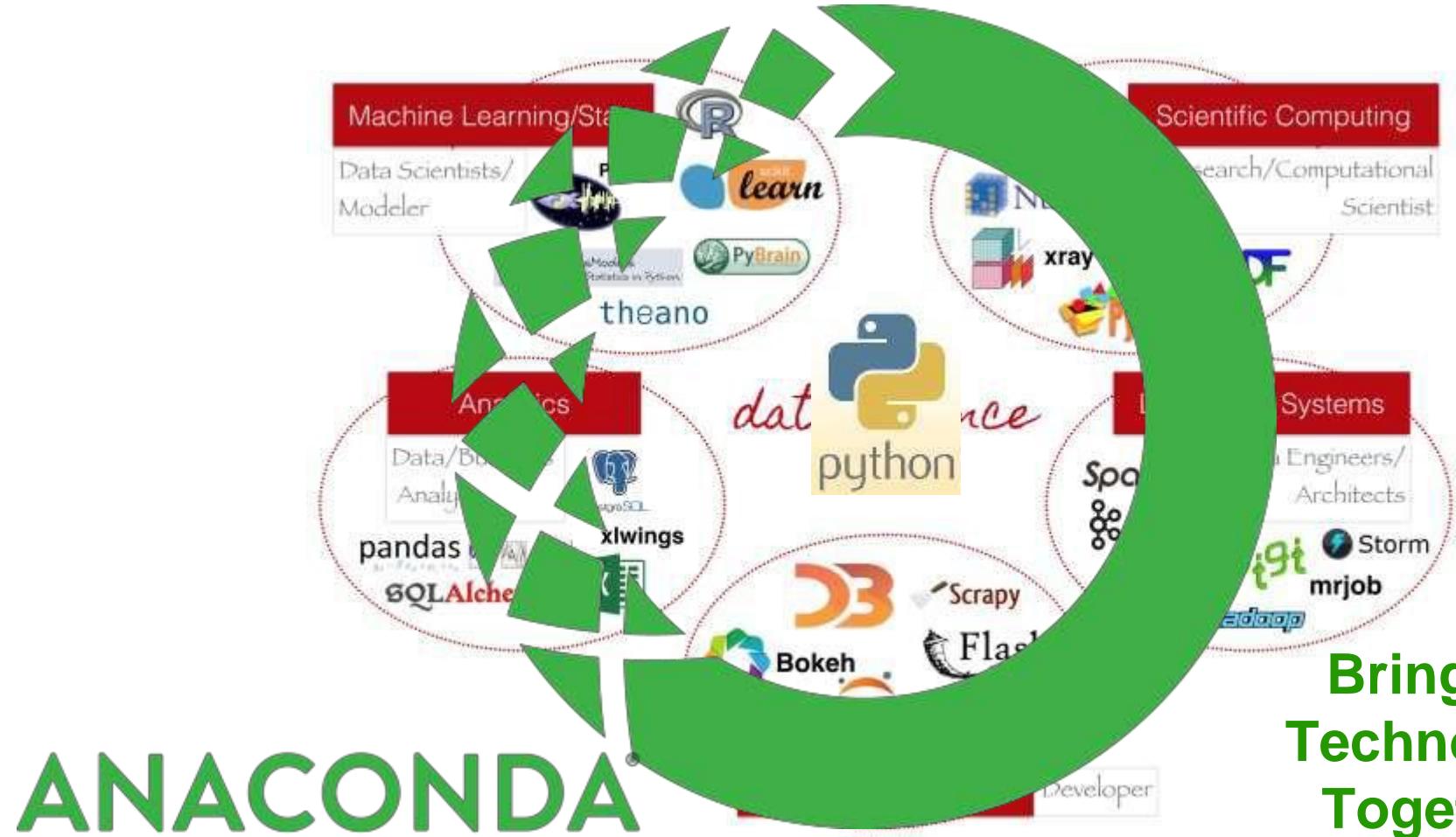
Community



Data Science is growing with Python leading the way







**Bringing
Technology
Together**

ANACONDA

Accelerate adoption of Data Science

<http://continuum.io/downloads>



PYTHON & R OPEN SOURCE ANALYTICS

NumPy

SciPy

Pandas

Scikit-learn

Jupyter/IPython

Numba

Matplotlib

Spyder

TensorFlow

Cython

Theano

Scikit-image

NLTK

Dask

Caffe

dplyr

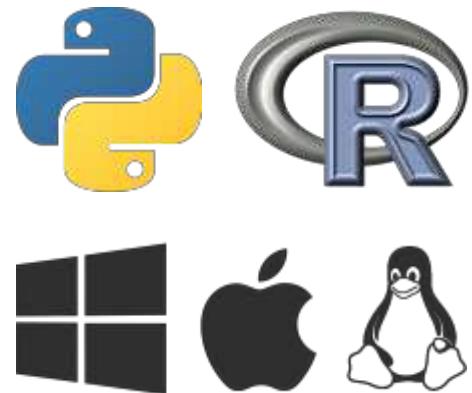
shiny

ggplot2

tidyr

& 1000+ packages

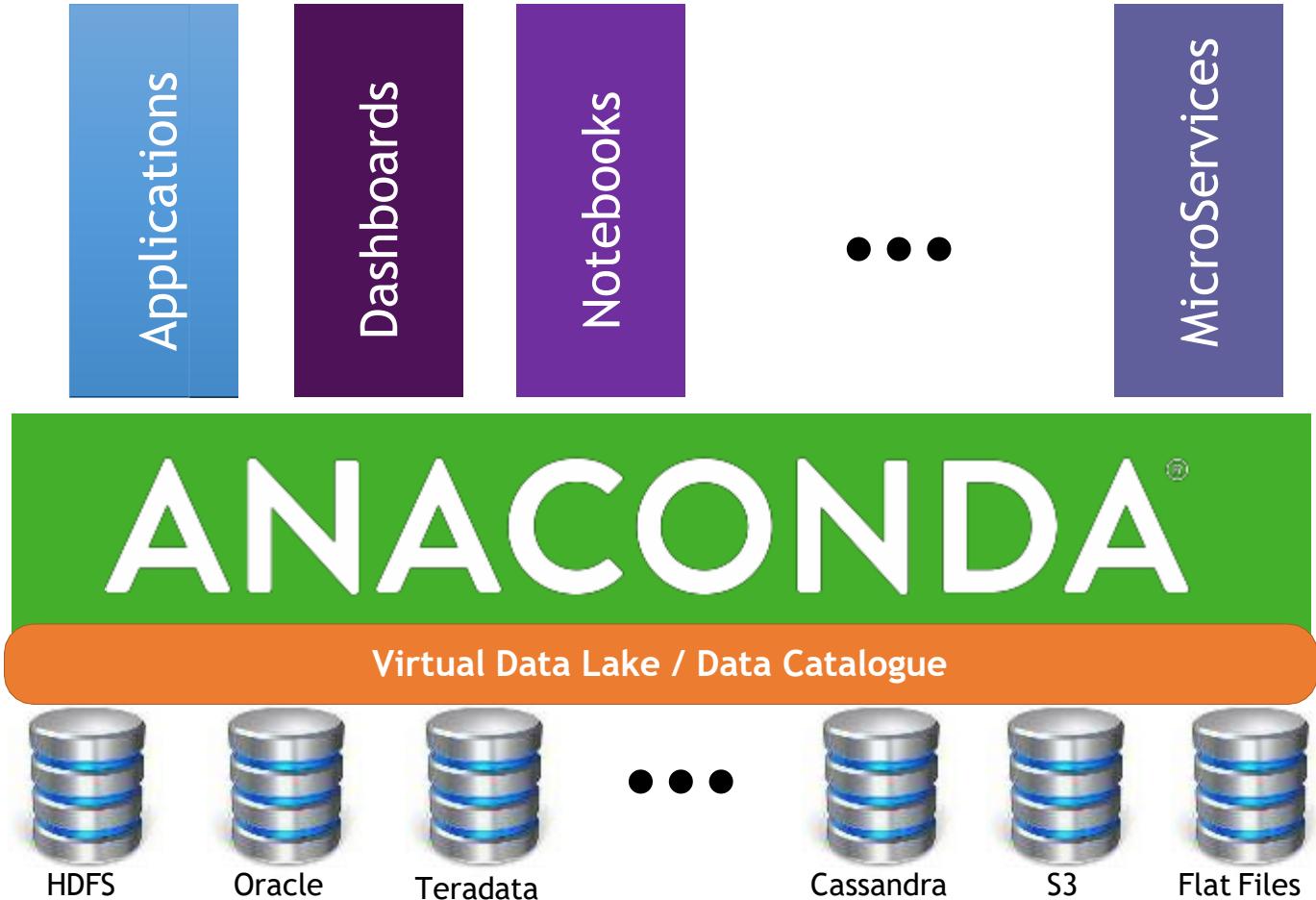
CONDA®



**Insight
Decisions
Actions
Results
Creations**

**Open Data
Science**

DATA



Anaconda Navigator

Anaconda Navigator

File Help

Sign in to Anaconda Cloud

ANACONDA NAVIGATOR

Home Environments Projects (beta) Learning Community Documentation Developer Blog Feedback

Refresh

Applications on root Channels

Applications:

- jupyterlab 0.27.0 An extensible environment for interactive and reproducible computing, based on the Jupyter Notebook and Architecture. [Launch](#)
- jupyter 5.0.0 Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis. [Launch](#)
- qtconsole 4.3.1 PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more. [Launch](#)
- spyder 3.2.3 Scientific PYthon Development Environment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features. [Launch](#)
- glueviz 0.10.4 Multidimensional data visualization across files. Explore relationships within and among related datasets. [Install](#)
- orange3 3.4.1 Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox. [Install](#)
- rstudio 1.0.153 A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks. [Install](#)

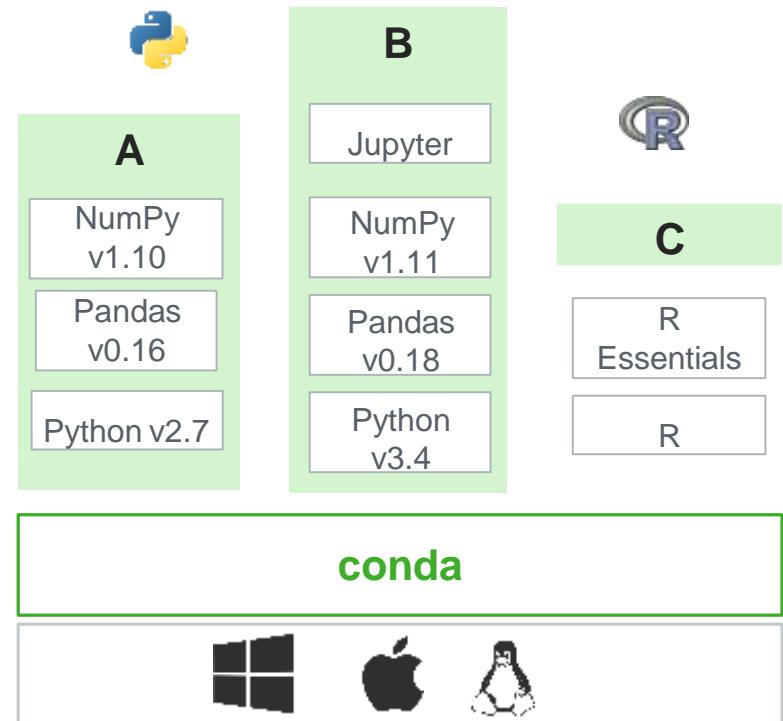
Documentation Developer Blog Feedback

Twitter YouTube GitHub

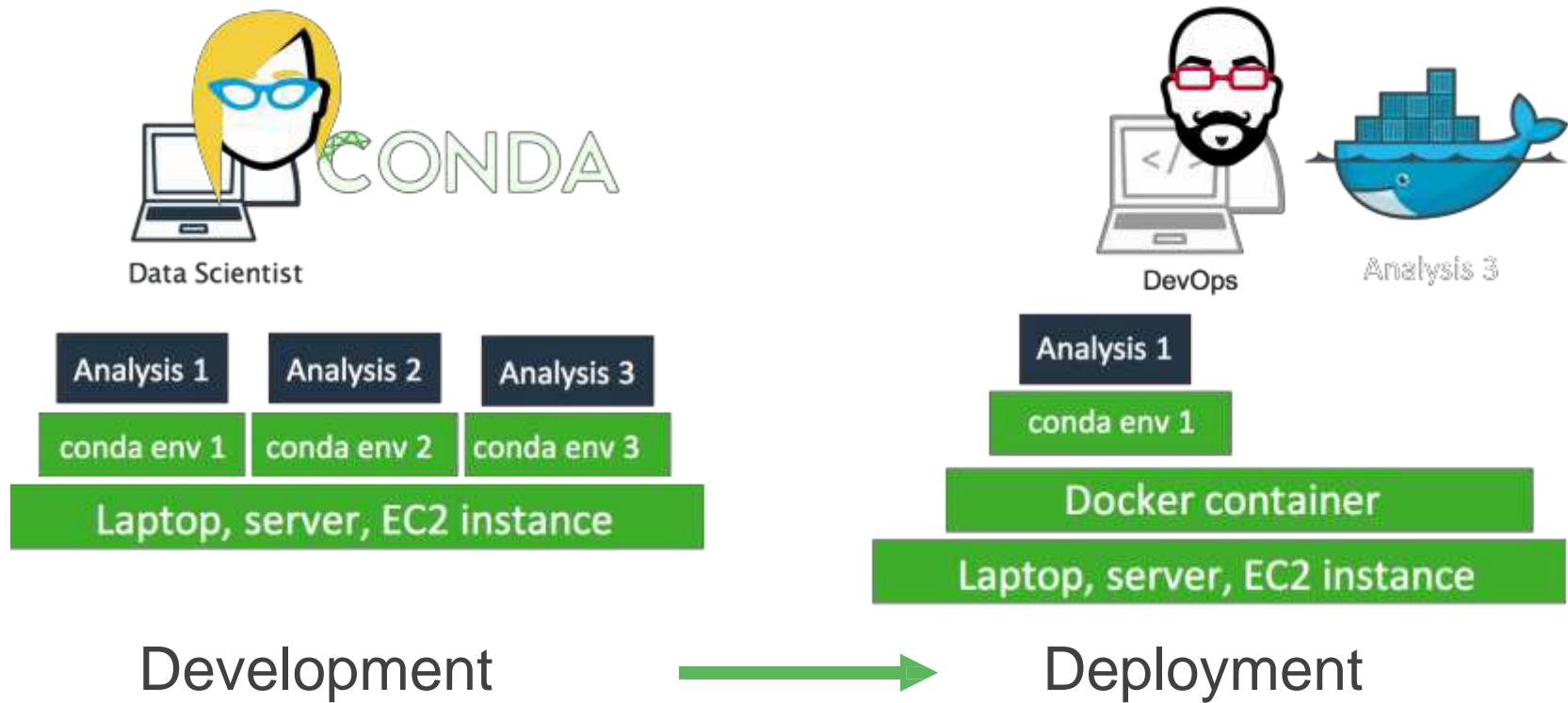
Conda Sandboxing Technology

- **Language** independent
- **Platform** independent
- No special privileges required
- No VMs or containers
- Enables:
 - **Reproducibility**
 - **Collaboration**
 - **Scaling**

“conda – package everything”



Conda for Deployment





```
$ conda install python=2.7  
$ conda install pandas  
$ conda install -c r r  
$ conda install -c conda-forge tensorflow
```

Install dependencies

```
name: myenv  
channels:  
  - chdoig  
  - r  
  - foo  
dependencies:  
  - python=2.7  
  - r  
  - r-ldavis  
  - pandas  
  - mongodb  
  - spark=1.5  
  - pip  
  - pip:  
    - flask-migrate  
    - bar=1.4
```

```
$ conda env create  
$ source activate myenv
```

Manage multiple environments

```
$ anaconda-project run plot --show
```

Deploy an interactive visualization

Python Scripting

About Python



- Development started in the 1980's by Guido van Rossum.
 - Only became popular in the last decade or so.
- Python 2.x currently dominates, but Python 3.x is the future of Python.
- Interpreted, very-high-level programming language.
- Supports a multitude of programming paradigms.
 - OOP, functional, procedural, logic, structured, etc.
- General purpose.
 - Very comprehensive standard library includes numeric modules, crypto services, OS interfaces, networking modules, GUI support, development tools, etc.

Philosophy

- ▶ From *The Zen of Python*
[\(https://www.python.org/dev/peps/pep-0020/\)](https://www.python.org/dev/peps/pep-0020/)

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!

Notable Features

- Easy to learn.
- Supports quick development.
- Cross-platform.
- Open Source.
- Extensible.
- Embeddable.
- Large standard library and active community.
- Useful for a wide variety of applications.

What sort of language is Python?

Compiled

Explicitly
compiled
to machine
code



C, C++,
Fortran

Explicitly
compiled
to byte
code

Java, C#

Implicitly
compiled
to byte
code

Python

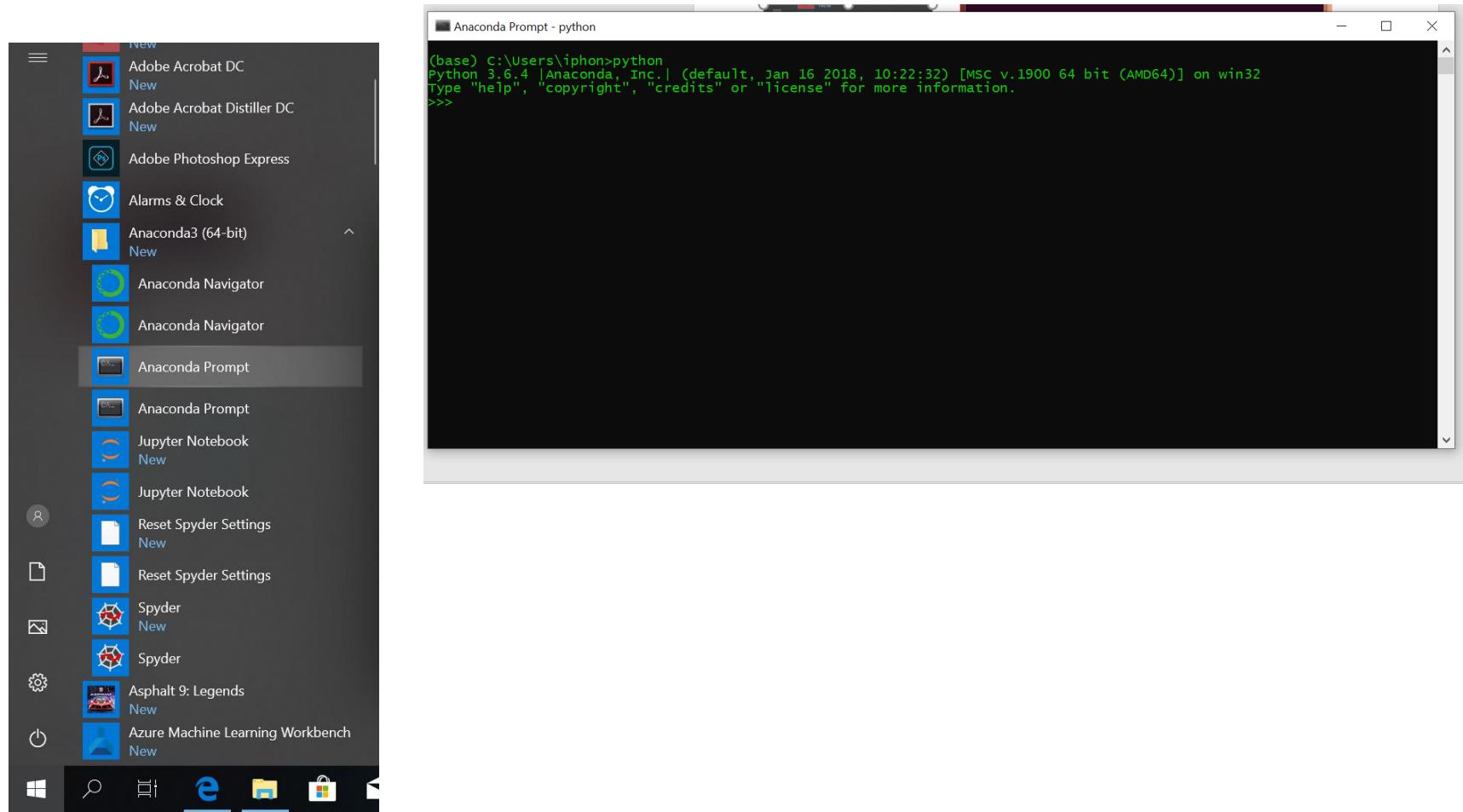
Interpreted

Purely
interpreted



Shell,
Perl

Running Python — 1



Running Python — 2

```
$ python3
Python 3.2.3 (default, May 3 2012, 15:54:42)
[GCC 4.6.3] on linux2
>>>
```

Annotations:

- Unix prompt
- Unix command
- Introductory blurb
- Python version
- Python prompt

Quitting Python

```
>>> exit()
```

```
>>> quit()
```

```
>>> Ctrl + D
```

Any one
of these

A first Python command

```
>>> print('Hello, world!')
```

Hello, world!

```
>>>
```

The diagram illustrates the interaction between a Python prompt and a command. It features four main components: a 'Python prompt' box at the top right, a 'Python command' box below it, an 'Output' box further down, and another 'Python prompt' box at the bottom right. A blue bracket on the left encloses the first two lines of text. Two blue arrows point from these lines to their respective boxes: one arrow from the first line to the 'Python prompt' box, and another from the second line to the 'Python command' box. A third blue arrow points from the 'Output' box back up to the third line of text ('Hello, world!').

Python commands

```
print('Hello, world!')
```

Python “function”

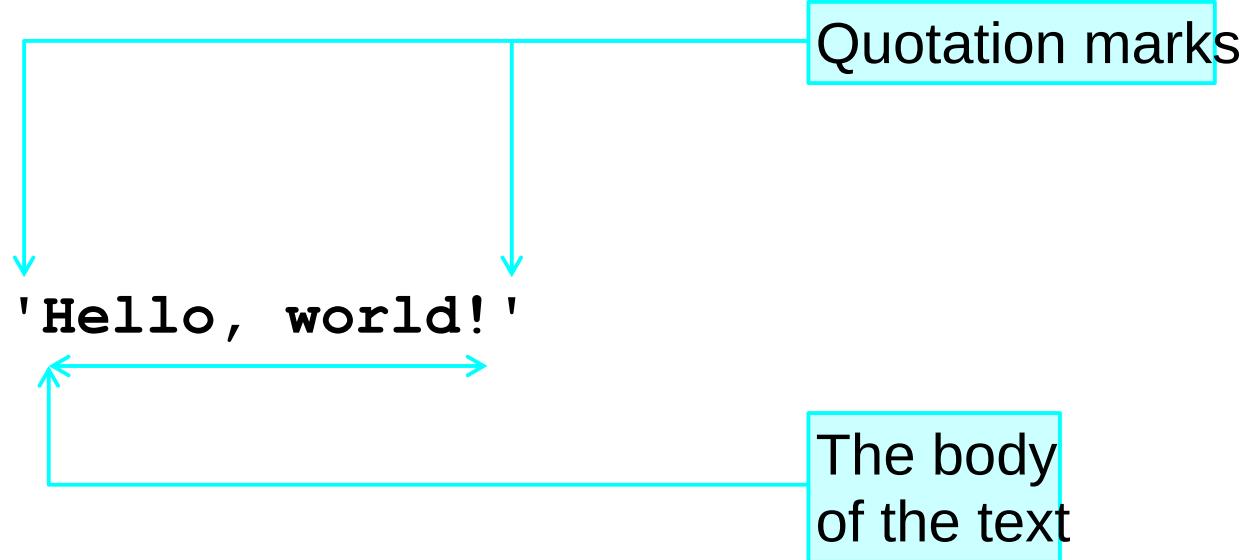
Round brackets
— “parentheses”

Function’s “argument”

```
print ≠ PRINT
```

“Case sensitive”

Python text



The quotes are not part of the text itself.

Quotes?

print → Command

'print' → Text

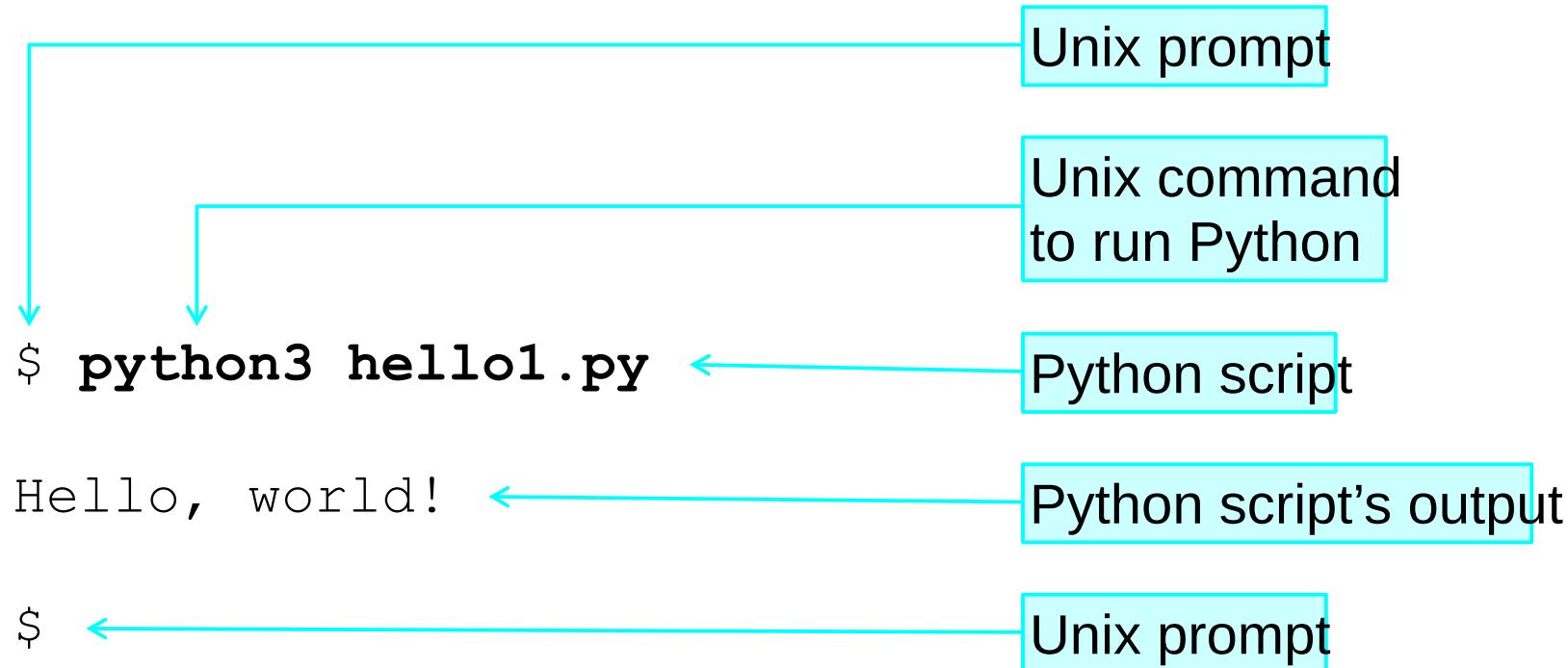
Python scripts

File in home directory

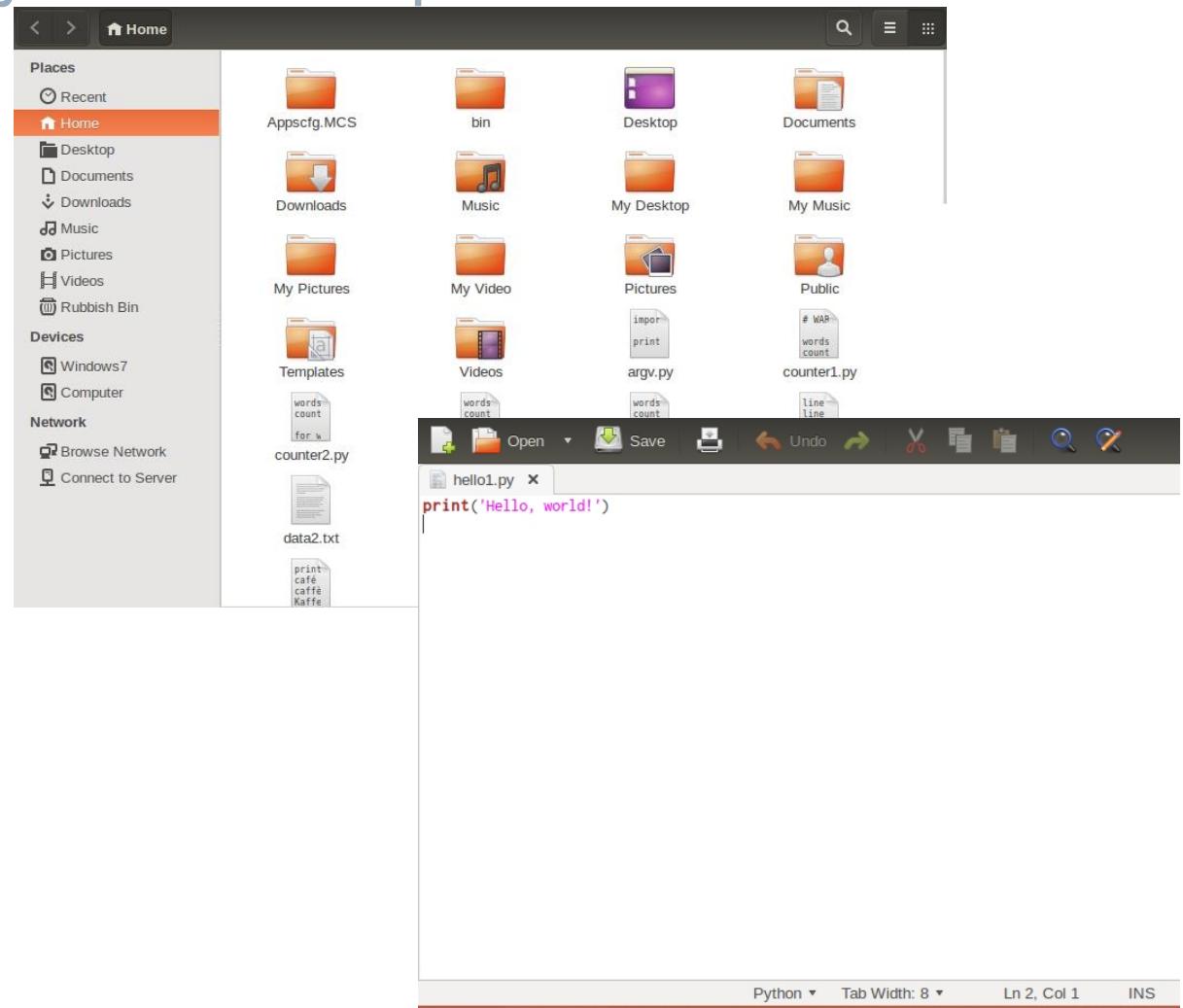
```
print('Hello, world!')
```

Run from *Unix* prompt

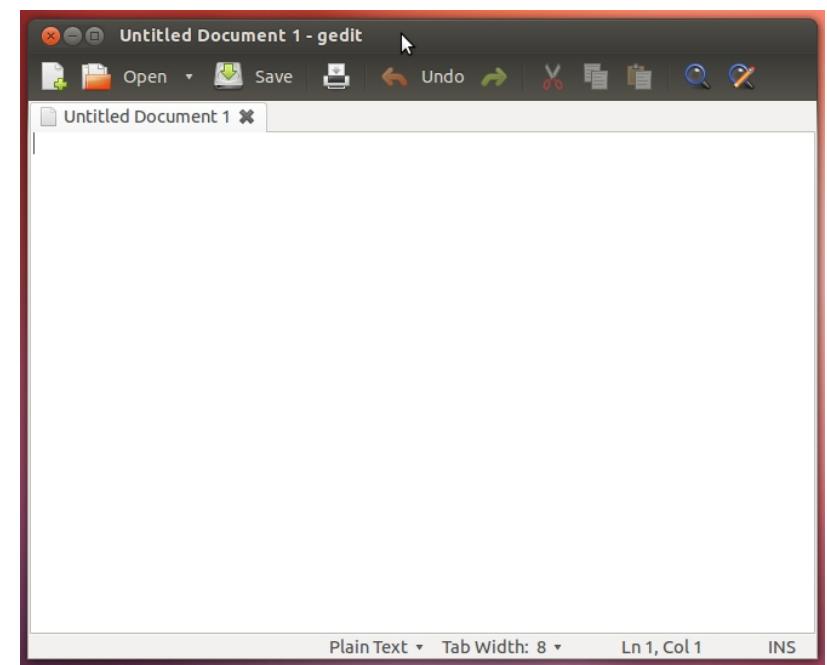
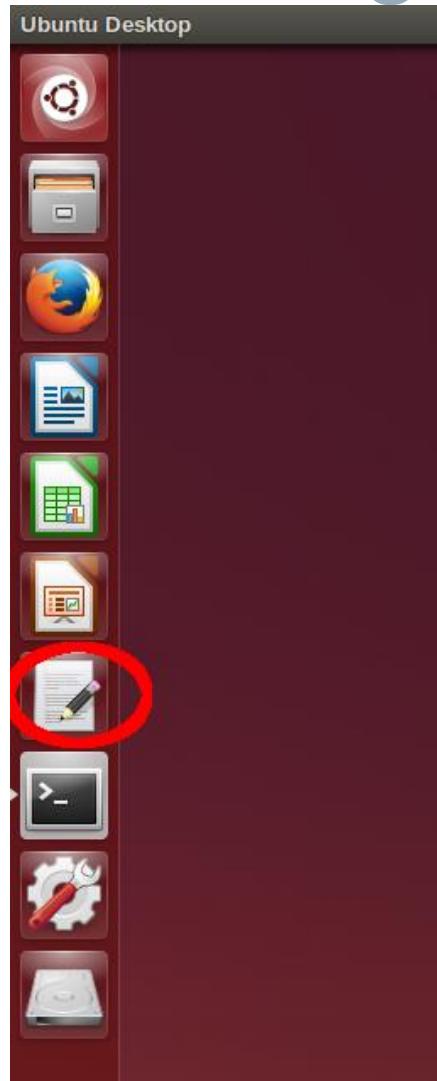
hello1.py



Editing Python scripts — 1



Editing Python scripts — 2



Exercise 1

1. Print “Goodbye, cruel world!” from interactive Python.
 2. Edit `exercise1.py` to print the same text.
 3. Run the modified `exercise1.py` script.
- ! Please ask if you have questions.

Jupyter

A bit of History



- Fernando Perez, 2001, CU Boulder (instead of writing a physics dissertation):
- Python can replace the collection of bash, perl, C/C++ Script. But the Python REPL can be better.

A screenshot of a terminal window titled "/bin/bash". The window shows a Python REPL session. The user runs the command "run ~/scratch/error" and receives a ValueError. The traceback shows the code in error.py, which contains a main() function that calls RampNum(). The error occurs at line 45 where result[:] = arange(size)*step + start. The message "ValueError: shape mismatch: objects cannot be broadcast to a single shape" is displayed. The window has a yellow background and a blue header bar.

```
In [13]: run ~/scratch/error
reps: 5
-----
ValueError                                Traceback (most recent call last)
/home/fperez/scratch/error.py in <module>()
    70     if __name__ == '__main__':
    71         #explode()
---> 72     main()
    73     g2='another global'

/home/fperez/scratch/error.py in main()
    60     array_num = zeros(size,'d')
    61     for i in xrange(reps):
---> 62         RampNum(array_num, size, 0.0, 1.0)
    63         RNtime = time.clock()-t0
    64         print 'RampNum time:', RNtime

/home/fperez/scratch/error.py in RampNum(result, size, start, end)
    43     tmp = zeros(size+1)
    44     step = (end-start)/(size-1-tmp)
---> 45     result[:] = arange(size)*step + start
    46
    47 def main():

ValueError: shape mismatch: objects cannot be broadcast to a single shape
In [14]:
```

NOVEMBER 2001: "JUST AN AFTERNOON HACK"

- 259 Line Python script. (<https://gist.github.com/fperez/1579699>)
- sys.ps1 -> In[N].
- sys.displayhook -> Out[N], caches results.
- Plotting, Numeric, etc.

2014 (OPENHUB STATS)

- 19,279 commits
- 442 contributors
- Total Lines: 187,326
- Number of Languages : 7 (JS, CSS, HTML, ...)

Improve over the terminal

- ❖ The REPL as a **network protocol**

- ❖ **Kernels**

- ❖ execute code

- ❖ **Clients**

- ❖ Read input

- ❖ Present output

**Simple abstractions enable rich,
sophisticated clients**

```
Python 2.7.2+ (default, Oct  4 2011, 20:06:09)
Type "copyright", "credits" or "license" for more information.

IPython 0.13.dev -- An enhanced Interactive Python.
?           -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
%uiref    -> A brief reference about the graphical user interface.

Welcome to pylab, a matplotlib-based Python environment [backend:
module://IPython.zmq.pylab.backend_inline].
For more information, type 'help(pylab)'.

In [1]: import scipy.linalg as la
...: mineigs = []
...: n = 256
...: for i in range(10):
...:     a = rand(n, n)
...:     mineigs.append(la.eigvals(a).min().real)
...:
...: mean(mineigs)
Out[1]: -4.569467643237938

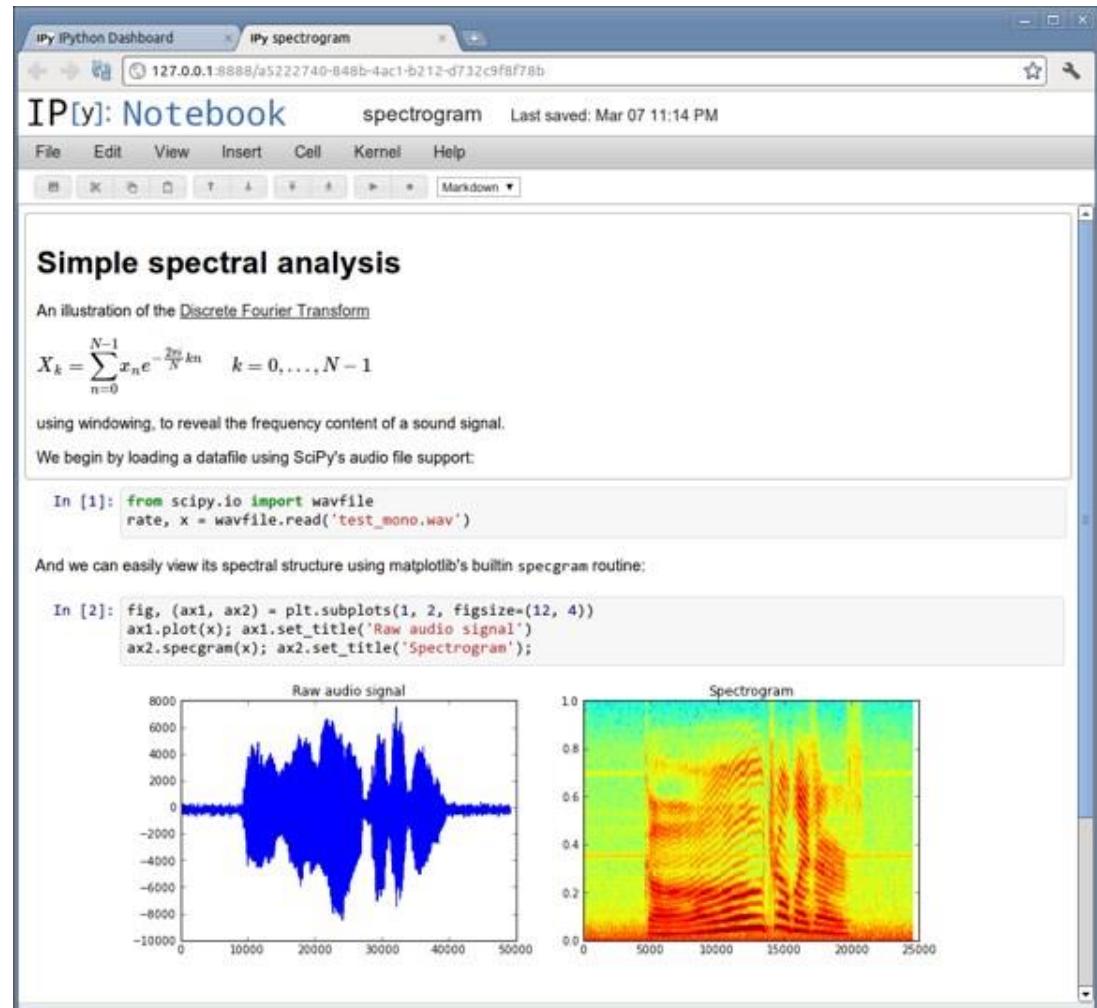
In [2]: %run mapping_seismic_stations.py

Seismic stations in the Himalaya
29.79°N
28.93°N
28.07°N
27.21°N
26.35°N
84.81°E 85.83°E 86.85°E 87.88°E 88.90°E
SAGA RC14 SSAN ONRN LAZE SAJA
XXI MNBUL NAIL DINX MAZA
YALA NLMO RSH
BUNG SUKT JRI NAMO
THAK SIND PHAP RUMJ TUML HILE PHID KAM
JANA BRA
```

In [3]: |

2011: The IPython Notebook

- ❖ Rich web client
- ❖ Text & math
- ❖ Code
- ❖ Results
- ❖ Share, reproduce.



Funding



**ALFRED P. SLOAN
FOUNDATION**

GORDON AND BETTY
MOORE
FOUNDATION



CONTINUUM
ANALYTICS

ENTHOUGHT
SCIENTIFIC COMPUTING SOLUTIONS

THE LEONA M. AND HARRY B.
HELMSLEY
CHARITABLE TRUST

SIMONS FOUNDATION

POWERED BY
rackspace[®]
the *open cloud* company

Microsoft

Google

Bloomberg

Why ?

- ▷ Don't reinvent the wheel: reimplement 1 piece, get the rest for free.
- ▷ You don't like the frontend, write a new one for Python
get 50+ languages that work out of the box with it.
(<https://github.com/ipython/ipython/wiki/IPython-kernels-for-other-languages>)
- ▷ You don't like a language, write your own kernel, get all the IDEs, conversion tools.
- ▷ Etc..

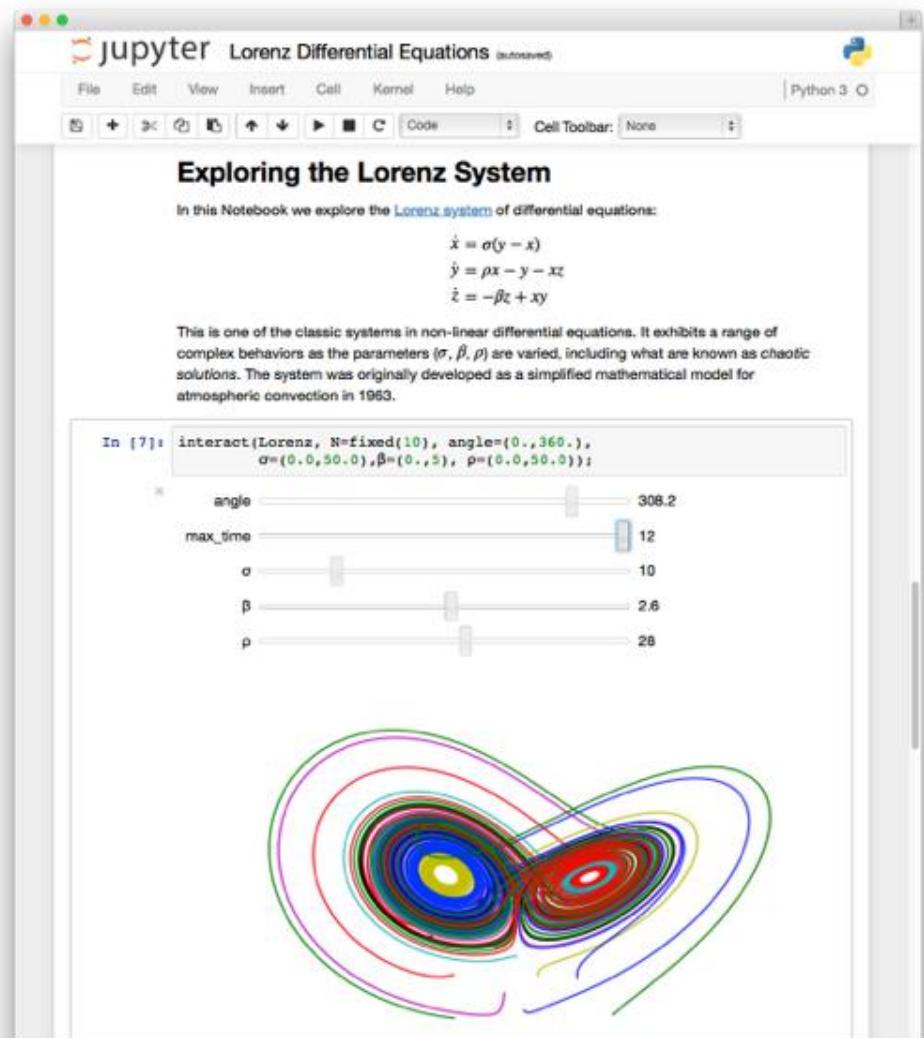
The notebook

Web Application, that allow code to produce web-rich representation (images, sound, video, math, ...)

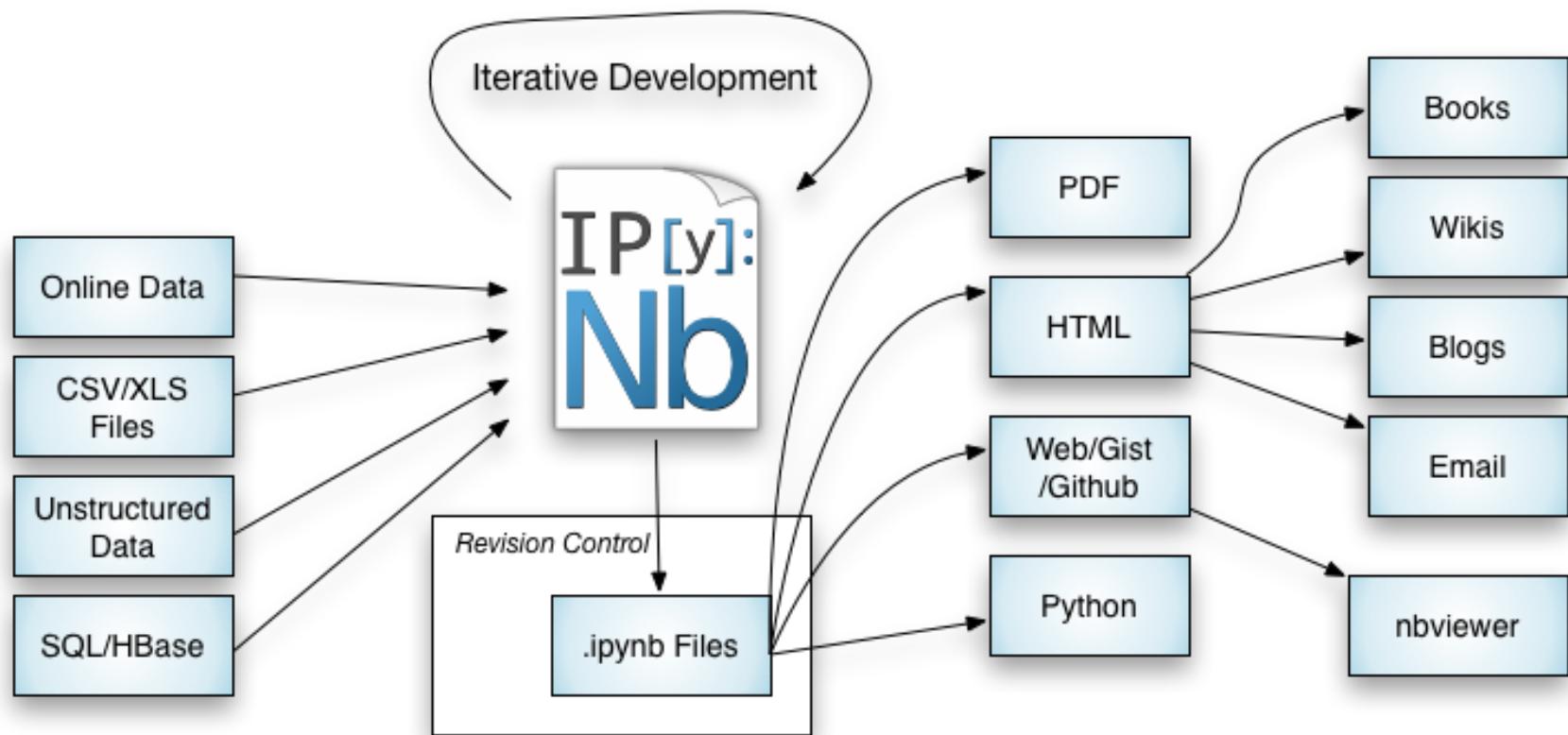
The Browser, Server, and kernel(s) can be on separate machines.

The default application to edit `ipynb` files.

`ipynb` file are JSON based files embedding input and output, so which can be read & converted without a running kernel.



The Notebook Fileformat (`.ipynb`)



NbViewer

- Zero-install reading of notebooks
- Just share a URL
- nbviewer.org

Under the hood: get raw URL and convert to HTML on the fly.

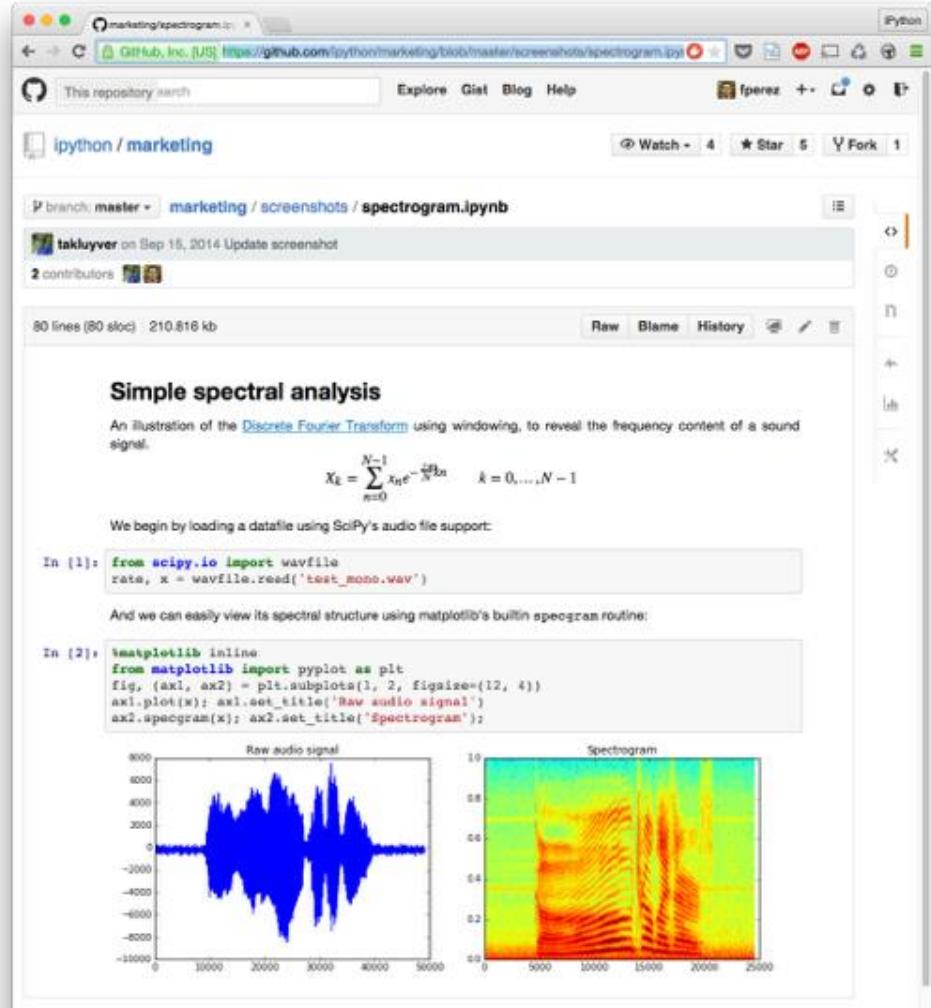
Sharing:

git push, or dropbox sync.

The screenshot shows the nbviewer homepage. At the top, there is a navigation bar with links to nbviewer, FAQ, IPython, and Jupyter. Below the navigation bar, the word "nbviewer" is prominently displayed in a large, bold, sans-serif font. Underneath it, the tagline "A simple way to share Jupyter Notebooks" is written in a smaller, regular font. A search bar at the top right contains the placeholder text "URL | GitHub username | GitHub username/repo | Gist ID" and a "Go!" button. The main content area is organized into several sections: "Programming Languages" featuring IPython, IRuby, and Julia; "Books" featuring "Python for Signal Processing" by O'Reilly and "Mining the Social Web"; "Probabilistic Programming" featuring "PROBABILISTIC PROGRAMMING & BAYESIAN METHODS FOR HACKERS"; and "Misc" featuring "Data Visualization with Lightning", "Interactive data visualization with Bokeh", and "Interactive plots with Plotly". Each section contains a thumbnail image of the respective notebook or book cover.

Nbviewer on GitHub

- Since May GitHub renders Notebooks
- Powered by `nbconvert`, the library that deals with `*.ipynb` -> *
- Over 200,000 notebooks on GitHub



The screenshot shows a GitHub repository page for 'ipython / marketing' with a file named 'spectrogram.ipynb'. The page includes a 'Raw' code view and two plots: a waveform titled 'Raw audio signal' and a spectrogram titled 'Spectrogram'.

Simple spectral analysis

An illustration of the [Discrete Fourier Transform](#) using windowing, to reveal the frequency content of a sound signal.

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \quad k = 0, \dots, N-1$$

We begin by loading a datafile using SciPy's audio file support:

```
In [1]: from scipy.io import wavfile  
rate, x = wavfile.read('test_mono.wav')
```

And we can easily view its spectral structure using matplotlib's builtin specgram routine:

```
In [2]: %matplotlib inline  
from matplotlib import pyplot as plt  
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))  
ax1.plot(x); ax1.set_title('Raw audio signal')  
ax2.specgram(x); ax2.set_title('Spectrogram')
```

Raw audio signal

Spectrogram

Exercise 2

- ▷ Create “Hello World” using Jupyter Notebook
- ▷ Issue command “print(“Hello World”)

Data Structure

A little more text

Full “Unicode” support

www.unicode.org/charts/

```
print('həllo, ω☺λpə!!')
```

hello2.py

Getting characters



AltGr + Shift + #

g

Linux

gó

"LATIN SMALL
LETTER G
WITH BREVE"

Character Selector

\u011f

Text: a “string” of characters

```
>>> type('Hello, world!')
```

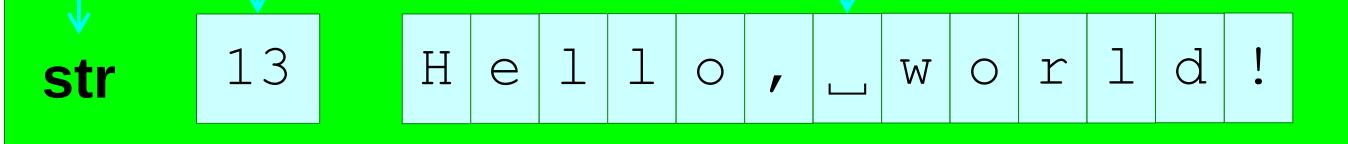
```
<class 'str'>
```

A string of characters

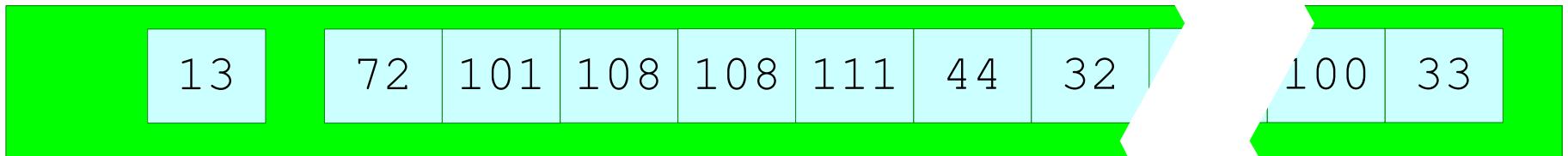
Class: string

Length: 13

Letters



Text: “behind the scenes”



```
>>> '\u011f'
```

011f₁₆

'g'

```
>>> ord('g')
```

287₁₀

287

```
>>> chr(287)
```

'g'

g

Adding strings together: +

“Concatenation”

```
print('Hello, ' + 'world!')
```

hello3.py

```
>>> 'Hello, ' + 'world!'
```

```
'Hello, world!'
```

```
>>>
```

Pure concatenation

```
>>> 'Hello, ' + 'world!'
```

```
'Hello, world!'
```

```
>>> 'Hello, ' + '_world!'
```

```
'Hello, world!'
```

```
>>> 'Hello, ' + 'world!'
```

```
'Hello,world!'
```

Only simple concatenation

No spaces added automatically.

Single & double quotes

```
>>> 'Hello, world! '
```

Single quotes

```
'Hello, world! '
```

Single quotes

```
>>> "Hello, world!"
```

Double quotes

```
'Hello, world! '
```

Single quotes

Python strings: input & output

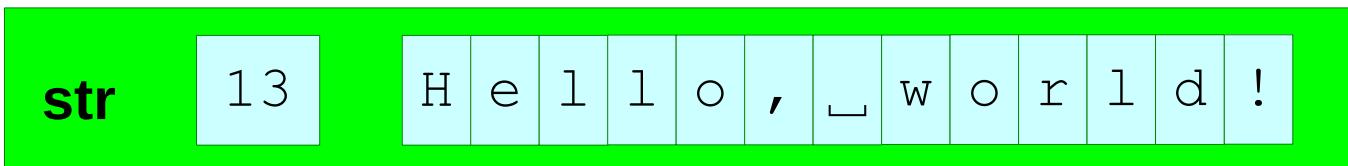
'Hello, world! '

"Hello, world! "

Single or double quotes on input.

Create same string object.

Single quotes on output.



Uses of single & double quotes

```
>>> print('He said "hello" to her.')
```

He said "hello" to her.

```
>>> print("He said 'hello' to her.")
```

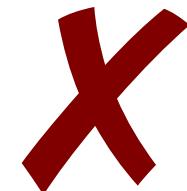
He said 'hello' to her.

Why we need different quotes

```
>>> print('He said 'hello' to her.')
```

```
File "<stdin>", line 1  
    print('He said 'hello' to her.'  
          ^
```

```
SyntaxError: invalid syntax
```



Adding arbitrary quotes

```
>>> print('He said \'hello\' to her.')
```

He said 'hello' to her.

\ ' → '

Just an ordinary
character.

\ " → "

“Escaping”

str [23] H e _ s a i d _ ' h e l l o ' _ t o _ h e r .

Putting line breaks in text

Hello,
world!

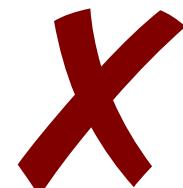
What we want

```
>>> print('Hello,  
world')
```

Try this

```
>>> print('Hello,  
File "<stdin>", line 1  
    print('Hello,  
        ^
```

SyntaxError: EOL while
scanning string literal



"EOL": End Of Line

Inserting “special” characters

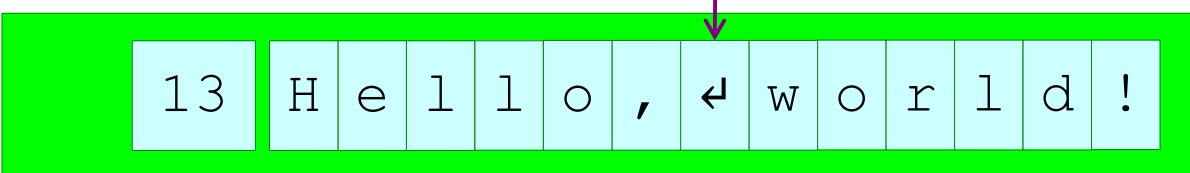
```
>>> print('Hello,\nworld!')
```

```
Hello,  
world!
```

Treated as
a new line.

\n

Converted into a
single character.



```
>>> len('Hello,\nworld!')
```

```
13
```

len() function: gives
the length of the object

The backslash

Special → Ordinary

\ ' → '

\ " → "

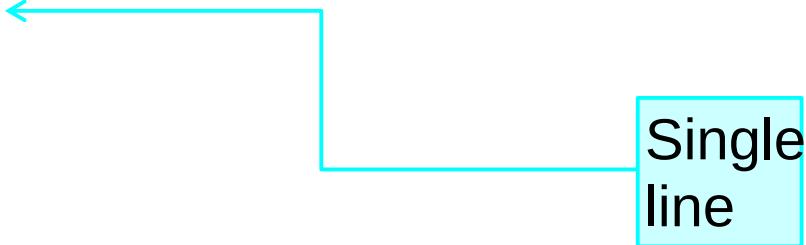
Ordinary → Special

\ n → ↵

\ t → →|

\n: unwieldy for long text

'SQUIRE TRELAWNEY, Dr. Livesey, and the\nrest of these gentlemen having asked me\nto write down the whole particulars\nabout Treasure Island, from the\nbeginning to the end, keeping nothing\nback but the bearings of the island,\nand that only because there is still\ntreasure not yet lifted, I take up my\npen in the year of grace 17\u00b7 and go\nback to the time when my father kept\nthe Admiral Benbow inn and the brown\nold seaman with the sabre cut first\ntook up his lodging under our roof.'



Single
line

Special input method for long text

''' SQUIRE TRELAWNEY, Dr. Livesey, and the rest of these gentlemen having asked me to write down the whole particulars about Treasure Island, from the beginning to the end, keeping nothing back but the bearings of the island, and that only because there is still treasure not yet lifted, I take up my pen in the year of grace 17__ and go back to the time when my father kept the Admiral Benbow inn and the brown old seaman with the sabre cut first took up his lodging under our roof. '''

Triple quotes

Multiple lines

Python's “secondary” prompt

```
>>> '''Hello,  
... world'''
```

Python asking for more
of the same command.

It's still just text!

```
>>> 'Hello, \nworld! '
```

```
'Hello\nworld'
```

Python uses `\n` to represent line breaks in strings.

```
>>> '''Hello,  
... world!'''
```

```
'Hello\nworld'
```

Exactly the same!

Your choice of input quotes:

Four inputs:

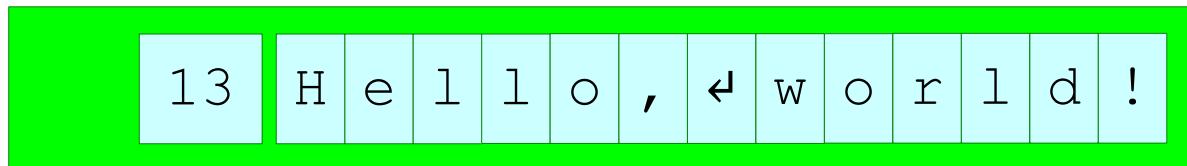
'Hello, \nworld! '

"Hello, \nworld!"

'''Hello,
world!'''

"""Hello,
world!"""

Same result:



Attaching names to values

“variables”

```
>>> message='Hello, world!'
```

```
message = 'Hello, world!'
print(message)
```

```
>>> message
```

hello3.py

```
'Hello, world!'
```

```
>>> type(message)
```

```
<class 'str'>
```

message

The diagram illustrates the memory representation of the string 'Hello, world!'. On the left, a green box contains the variable name 'message'. An arrow points from this box to a larger green rectangular area representing memory. This memory block is divided into 14 smaller cells, each containing a character or a space. The characters are: ' ', 'H', 'e', 'l', 'l', 'o', ',', ' ', 'w', 'o', 'r', 'l', 'd', '!'. The first cell contains a space character, and the last cell contains a punctuation mark. The entire sequence of characters is enclosed in a green border.

13 H e l l o , w o r l d !

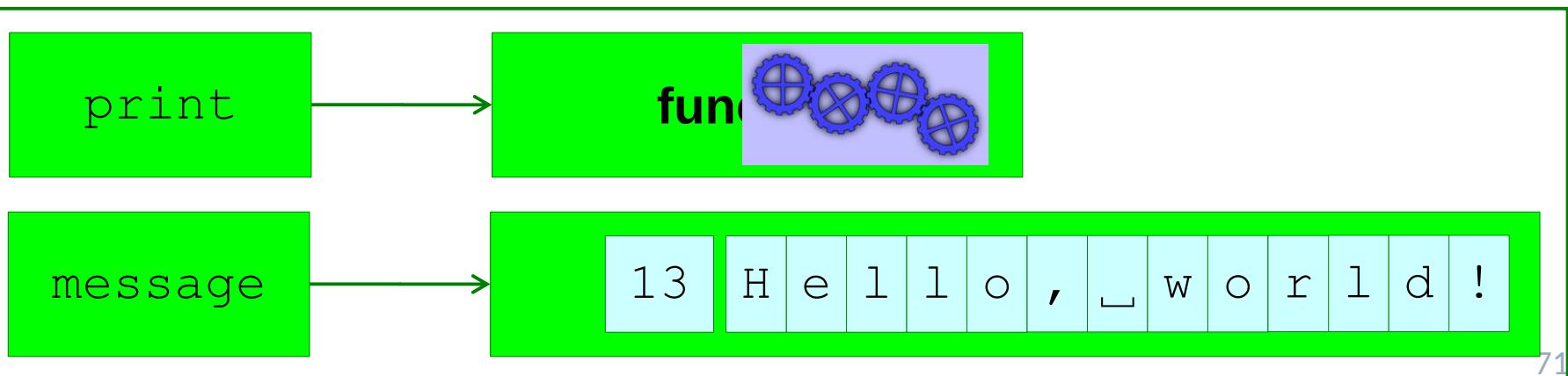
Attaching names to values

```
>>> type(print)
```

```
message = 'Hello, world!'
print(message)
```

```
<class 'builtin_function_or_method'>
```

hello4.py



Reading some text into a script

```
message = input('Yes? ')
print(message)
```

```
$ python3 input1.py
```

```
input ('Yes? ')
Yes? Boo!
```

```
print (message)
Boo!
```

Can't read numbers directly!

```
$ python3 input2.py
```

```
N? 10
```

```
number = input('N? ')  
print(number + 1)
```



input2.py

```
Traceback (most recent call last):  
  File "input2.py", line 2, in <module>  
    print(number + 1)  
TypeError:  
  Can't convert 'int' object  
  to str implicitly
```

string

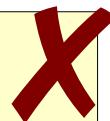
integer

input(): strings only

```
$ python3 input2.py
```

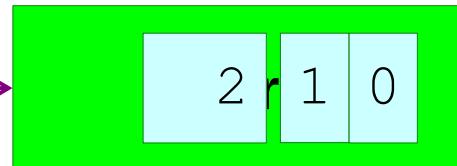
```
N? 10
```

```
number = input('N? ')  
print(number + 1)
```



input2.py

```
input('N? ')
```



≠



Some more types

```
>>> type('Hello, world!')
```

```
<class 'str'>←
```

string of characters

```
>>> type(42)
```

```
<class 'int'>←
```

integer

```
>>> type(3.14159)
```

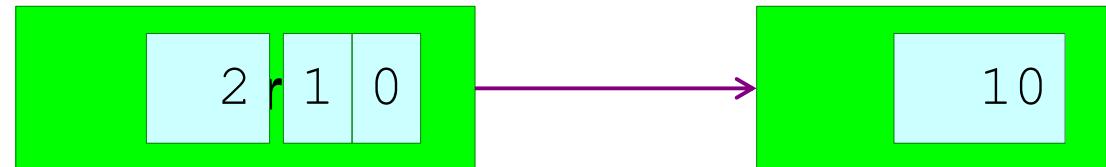
```
<class 'float'>←
```

floating point number

Converting text to integers

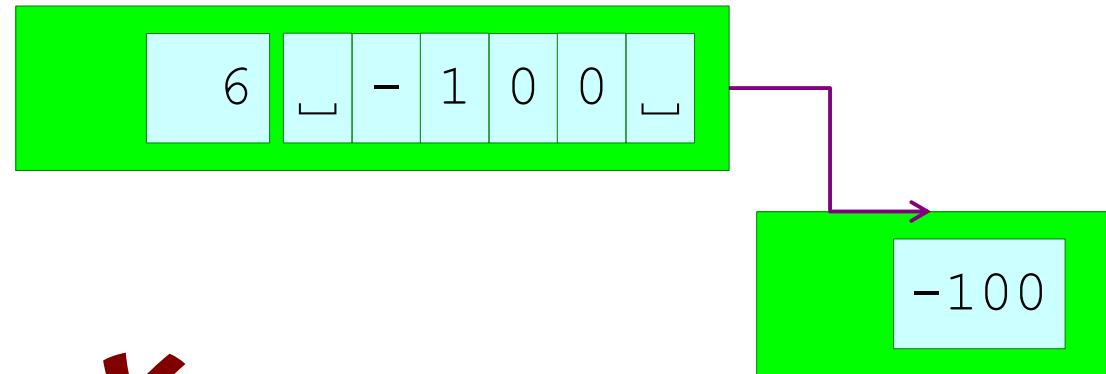
```
>>> int('10')
```

10



```
>>> int(' -100 ')
```

-100



```
>>> int('100-10')
```



ValueError:

invalid literal for int() with base 10: '100-10'

Converting text to floats

```
>>> float('10.0')
```

'10.0' is a string

```
10.0
```

10.0 is a floating
point number

```
>>> float('10.')  
10.0
```

Converting between ints and floats

```
>>> float(10)
```

```
10.0
```

```
>>> int(10.9)
```

```
10
```

Truncates
fractional part

```
>>> int(-10.9)
```

```
-10
```

Converting into text

```
>>> str(10)           integer → string
```

```
'10'
```

```
>>> str(10.000)      float → string
```

```
'10.0'
```

Converting between types

int()	anything → integer
float()	anything → float
str()	anything → string

Functions named after the type they convert *into*.

Integers

$$\mathbb{Z} \quad \{ \dots -2, -1, 0, \\ 1, 2, 3, 4 \dots \}$$

Integer addition & subtraction

```
>>> 20+5
```

```
25
```

```
>>> 20 _ - _ 5
```

```
15
```

Spaces around the operator don't matter.

“No surprises”

Integer multiplication

There is no “ \times ” on the keyboard.

Use “ $*$ ” instead

Linux:
AltGr + Shift + ,

```
>>> 20 _ * _ 5
```

```
100
```

Still no surprises

Integer division

There is no “ \div ” on the keyboard.

Use “ $/$ ” instead

Linux:

AltGr + **Shift** + **.**

```
>>> 20 _ / _ 5
```

4 . 0

This is a floating point number!

Surprise!

Integer division gives floats !



Fractions → Floats sometimes

Consistency → Floats **always**

```
>>> 20 / 40
```

```
0.5
```

```
>>> 20 / 30
```

```
0.6666666666666666
```

Integer powers

There is no “ 4^2 ” on the keyboard.

Use “`**`” instead

```
>>> 4 _ ** _ 2  
16
```

Spaces around the operator don't matter.

```
>>> 4* _*2  
SyntaxError: invalid syntax
```

Spaces in the operator do!

Integer remainders

e.g. Is a number even or odd?

Use “%”

```
>>> 4 % 2
```

```
0
```

```
>>> 5 % 2
```

```
1
```

```
>>> -5 % 2
```

```
1
```

Remainder is always non-negative

How big can a Python integer be?

```
>>> 2**2
```

```
4
```

```
>>> 4**2
```

```
16
```

```
>>> 16**2
```

```
256
```

```
>>> 256**2
```

```
65536
```

```
>>> 65536**2
```

```
4294967296
```

How big can a Python integer be?

```
>>> 4294967296**2
```

```
18446744073709551616
```

```
>>> 18446744073709551616**2
```

```
340282366920938463463374607431768211456
```

```
>>> 340282366920938463463374607431768211456**2
```

```
1157920892373161954235709850086879078532699846  
65640564039457584007913129639936
```

```
>>> 115792089237316195423570985008687907853269  
984665640564039457584007913129639936**2
```

```
1340780792994259709957402499820584612747936582  
059239337723561443721764030073546976801874298  
1669034276900318581864860508537538828119465699  
46433649006084096
```

How big can a Python integer be?

10443888814131525066917527107166243825799642490473837803842334832839
53907971557456848826811934997558340890106714439262837987573438185793
60726323608785136527794595697654370999834036159013438371831442807001
18559462263763188393977127456723346843445866174968079087058037040712
8404874011860911446797783598029006686938976881787785946905630190260
94059957945343282 ----- 421554169383555
98852914863182379 ----- 413490084170616
75093668333850551 ----- 213796825837188
09183365675122131 ----- 259567449219461
70238065059132456 ----- 0982023131690176
78006675195485079921636419370285375124784014907159135459982790513399
6115517942711068311340905842728842797915548497829543 ----- 26
9061394905987693002122963395687782878948440616007412 ----- Except for 505
7164237715481632138063104590291613692670834285644073 ----- machine 81
4657634732238502672530598997959960907994692017746248 ----- memory 65
9250178329070473119433165550807568221846571746373296 ----- .74
57002440926616910874148385078411929804522981857338977648103126085903
00130241346718972667321649151113160292078173803343609024380470834040
3154190336

There is no limit!

Big integers

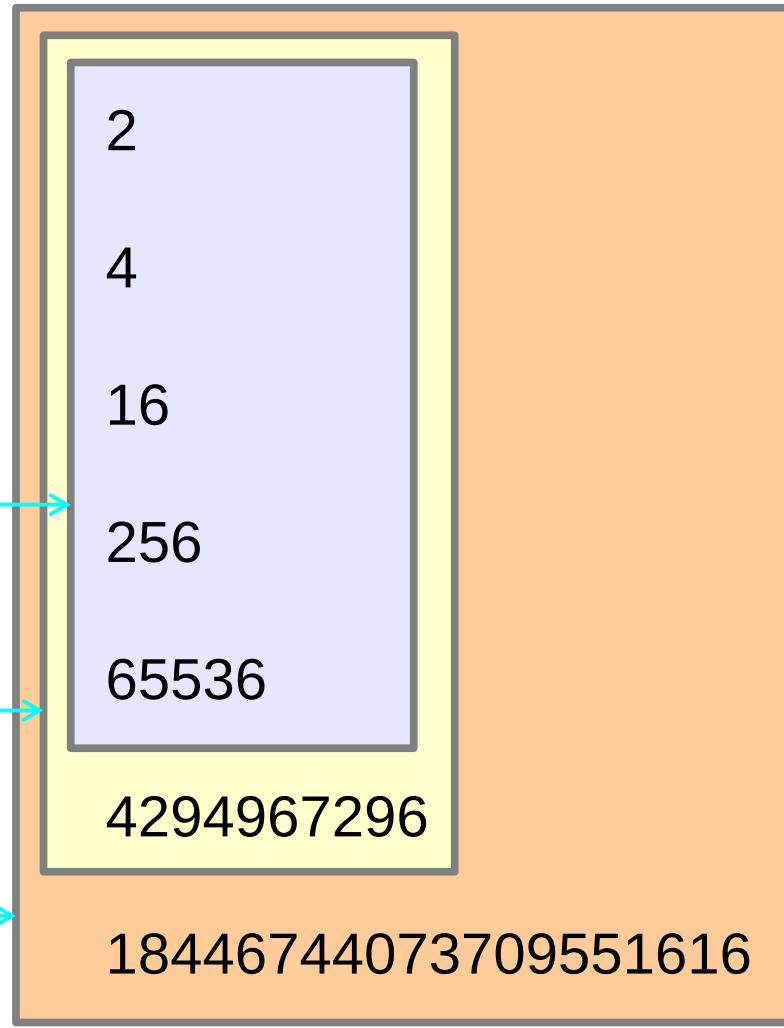
C / C++
Fortran

`int
INTEGER*4`

`long
INTEGER*8`

`long long
INTEGER*16`

Out of the reach
of C or Fortran!



3402823669209384634...
63374607431768211456

Floating point numbers

~~R~~

1.0

0.333333333

3.14159265

2.71828182

Basic operations

```
>>> 20.0 + 5.0
```

```
25.0
```

```
>>> 20.0 - 5.0
```

```
15.0
```

```
>>> 20.0 * 5.0
```

```
100.0
```

```
>>> 20.0 / 5.0
```

```
4.0
```

```
>>> 20.0 ** 5.0
```

```
3200000.0
```

Equivalent to integer arithmetic

Floating point imprecision

```
>>> 1.0 / 3.0
```

```
0.3333333333333333
```

```
>>> 10.0 / 3.0
```

```
3.3333333333333335
```

If you are relying on
this last decimal place,
you are doing it wrong!

≈ 17 significant figures

Hidden imprecision



```
>>> 0.1
```

```
0.1
```

```
>>> 0.1 + 0.1
```

```
0.2
```

```
>>> 0.1 + 0.1 + 0.1
```

```
0.30000000000000004
```

Really: if you are relying on
this last decimal place,
you are doing it wrong!



How big can a Python float be? — 1

```
>>> 65536.0**2
```

```
4294967296.0
```

So far, so good.

```
>>> 4294967296.0**2
```

```
1.8446744073709552e+19
```

Switch to
“scientific notation”

1.8446744073709552 e+19

1.8446744073709552 ×10¹⁹

Floats are not exact

```
>>> 4294967296.0**2  
1.8446744073709552e+19
```

Floating point

```
>>> 4294967296**2  
18446744073709551616
```

Integer

1.8446744073709552 $\times 10^{19}$ → 18446744073709552000

- 18446744073709551616

384

How big can a Python float be? — 2

```
>>> 1.8446744073709552e+19**2
```

```
3.402823669209385e+38
```

```
>>> 3.402823669209385e+38**2
```

```
1.157920892373162e+77
```

```
>>> 1.157920892373162e+77**2
```

```
1.3407807929942597e+154
```

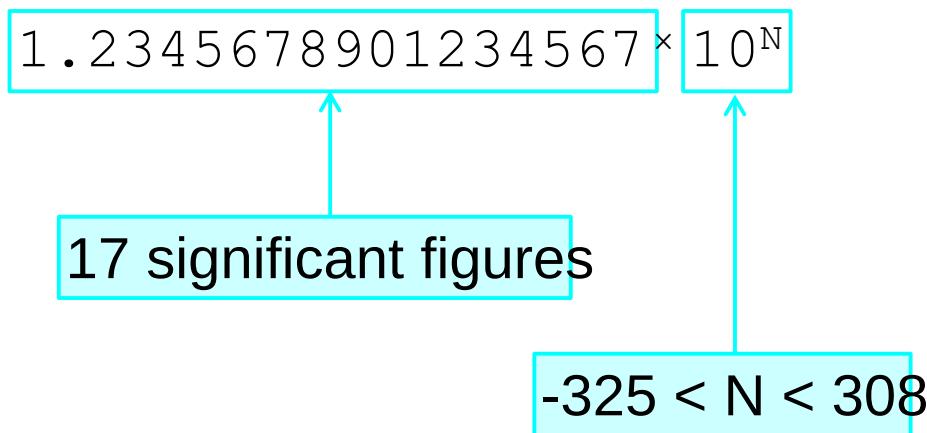
So far, so good.

```
>>> 1.3407807929942597e+154**2
```

```
OverflowError: (34,  
'Numerical result out of range')
```

Too big!

Floating point limits

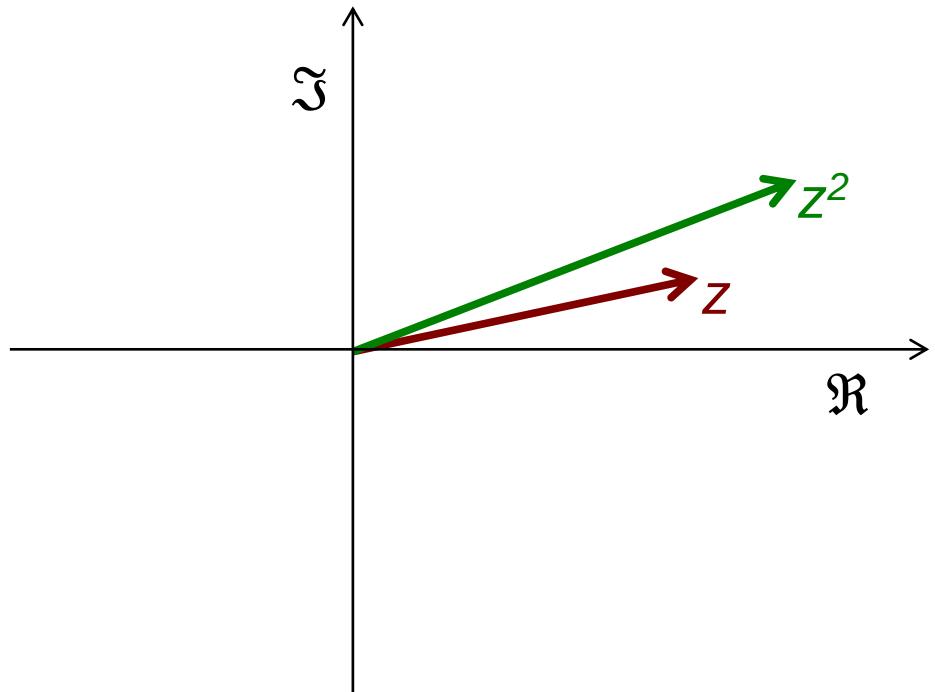


Positive values:

$$4.94065645841 \times 10^{-324} < N < 8.98846567431 \times 10^{307}$$

Complex numbers

C



```
>>> (1.25+0.5j)**2
```

```
(1.3125+1.25j)
```

Comparisons

$5 < 10$



$5 > 10$



Comparisons

```
>>> 5 < 10
```

Asking the question

True



```
>>> 5 > 10
```

Asking the question

False



True & False

```
>>> type(True)
```

```
<class 'bool'>
```

“Booleans”

5 + 10

15

int

int

int

5 < 10

True

bool

True & False

boo ✓

boo ✗

True

False

Only two values

Six comparisons

Maths

=

≠

<

>

≤

≥

Python

==

!=

<

>

<=

>=

Double equals sign

Equality comparison & assignment

=

`name = value`

Attach a name to a value.

==

`value1 == value2`

Compare two values

Textual comparisons

```
>>> 'cat' < 'dog'
```

Alphabetic ordering

True

```
>>> 'Cat' < 'cat'
```

Uppercase before lowercase

True

```
>>> 'Dog' < 'cat'
```

All uppercase before lowercase

True

Ordering text is *complicated*

Python inequalities use Unicode character numbers.

This is over-simplistic for “real” use.

“Collation” is a whole field of computing in itself

Alphabetical order?

German: z < ö

Swedish: ö < z

“Syntactic sugar”

`0 < number < 10`  `0 < number` and `number < 10`

```
>>> number = 5
```

```
>>> 0 < number < 10
```

True

Converting to booleans

`float()`

Converts to floating point numbers

`<class 'float'>`

`int()`

Converts to integers

`<class 'int'>`

`str()`

Converts to strings

`<class 'str'>`

`bool()`

Converts to booleans

`<class 'bool'>`

Useful conversions

'' → False

Empty string

'Fred' → True

Non-empty string

0 → False

Zero

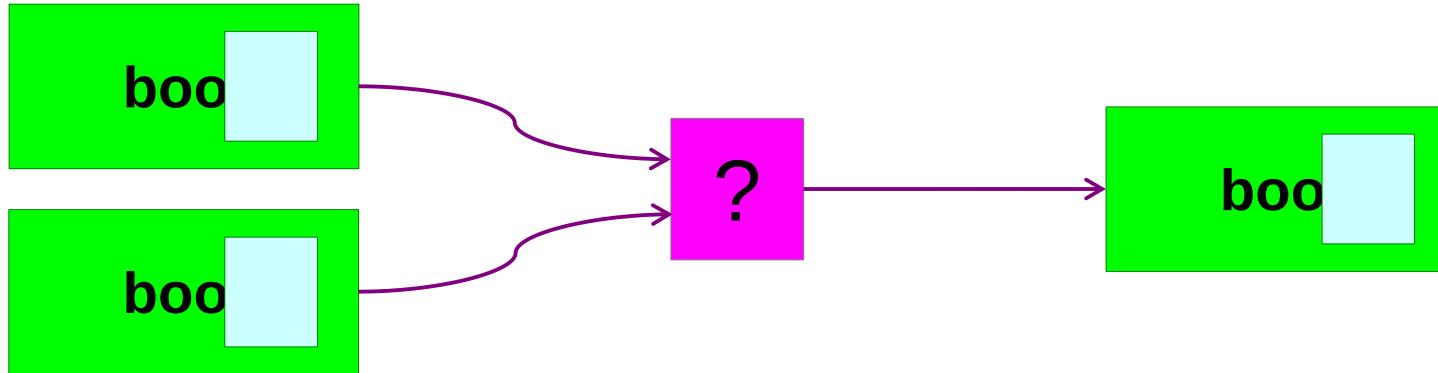
1 → True

Non-zero

12 → True

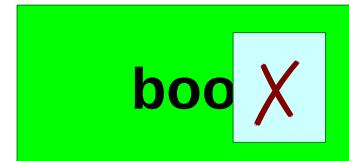
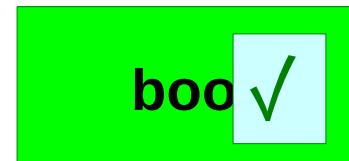
-1 → True

Boolean operations

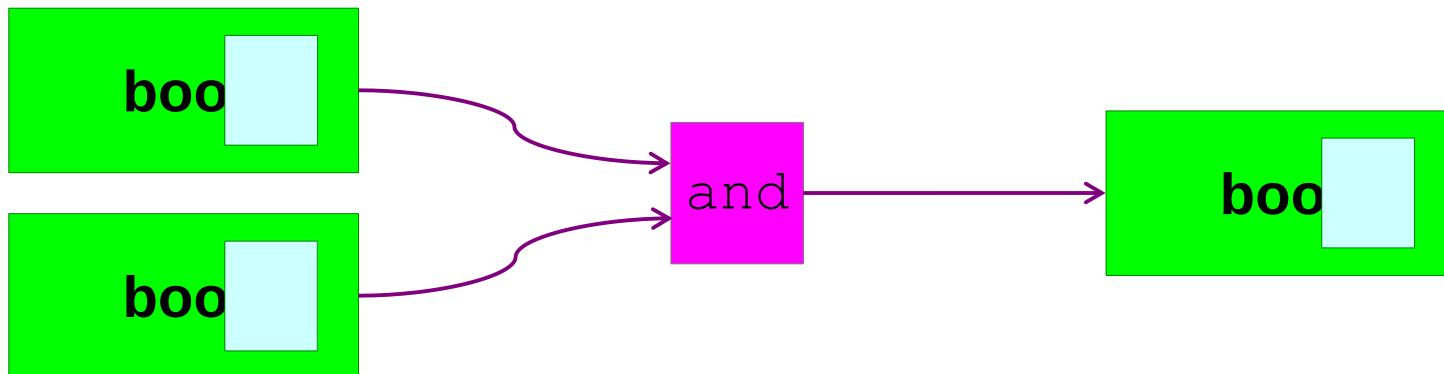


Numbers have $+$, $-$, $*$...

What do booleans have?



Boolean operations — “and”



True and True → True

True and False → False

False and True → False

False and False → False

Both have
to be True

Boolean operations — “and”

```
>>> 4 < 5 and 6 < 7
```

True

$4 < 5 \rightarrow \text{True}$
 $6 < 7 \rightarrow \text{True}$

} and $\rightarrow \text{True}$

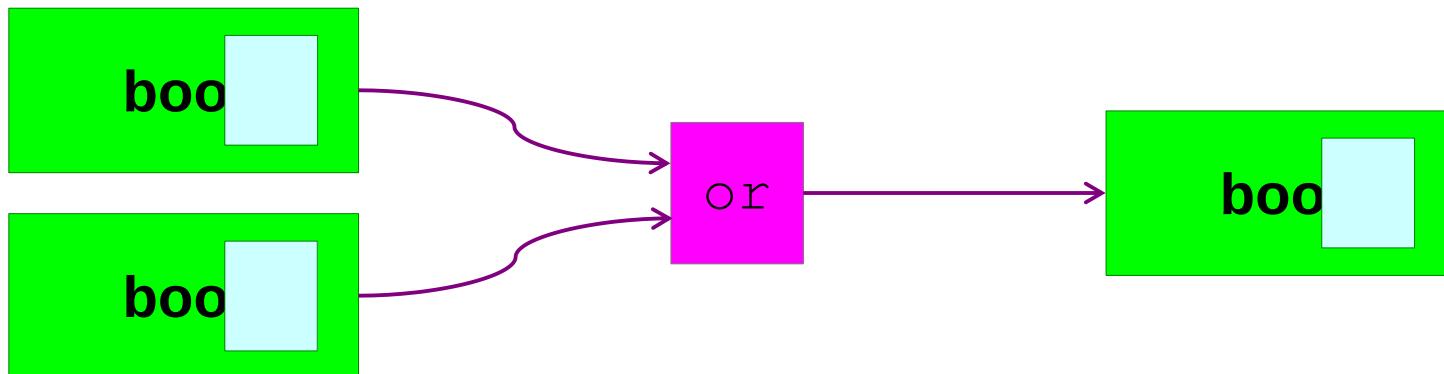
```
>>> 4 < 5 and 6 > 7
```

False

$4 < 5 \rightarrow \text{True}$
 $6 > 7 \rightarrow \text{False}$

} and $\rightarrow \text{False}$

Boolean operations — “or”



True or True → True

True or False → True

False or True → True

False or False → False

At least one has to be True

Boolean operations — “or”

```
>>> 4 < 5 or 6 < 7
```

```
True
```

$4 < 5 \rightarrow \text{True}$

$6 < 7 \rightarrow \text{True}$

} or $\rightarrow \text{True}$

```
>>> 4 < 5 or 6 > 7
```

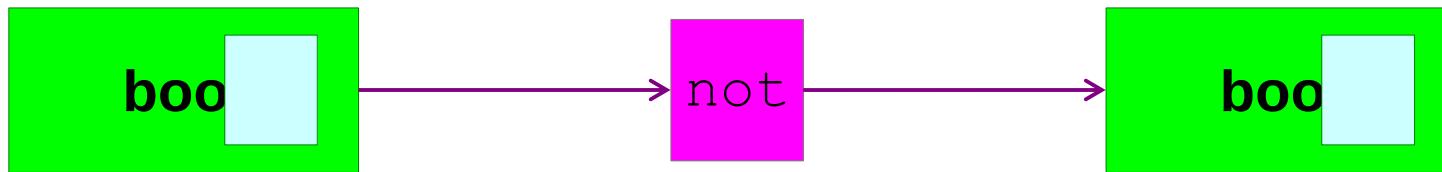
```
True
```

$4 < 5 \rightarrow \text{True}$

$6 > 7 \rightarrow \text{False}$

} or $\rightarrow \text{True}$

Boolean operations — “not”



not True → False

not False → True

Boolean operations — “not”

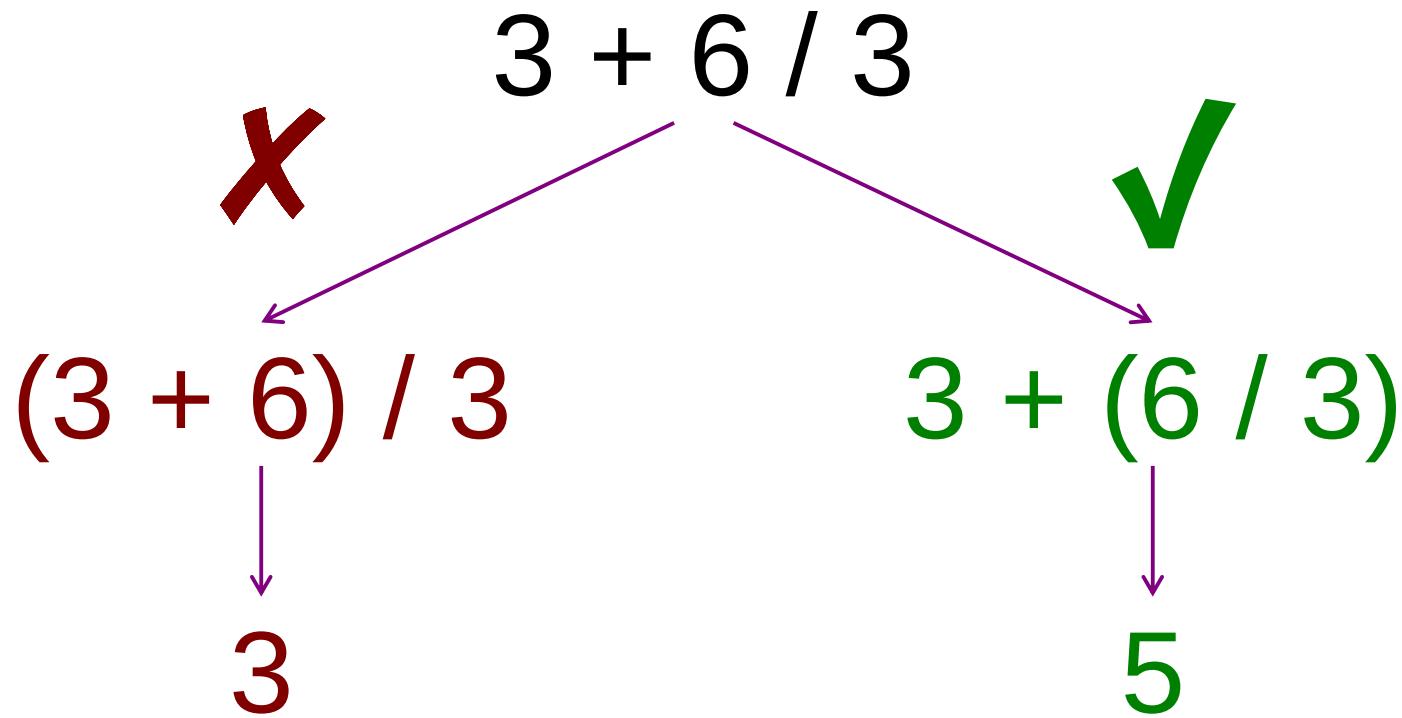
```
>>> not 6 < 7           6 < 7 → True — not → False
```

False

```
>>> not 6 > 7           6 > 7 → False — not → True
```

True

Ambiguity in expressions



Division before addition

Division first

$$3 + 6 / 3$$



$$3 + \quad 2$$



$$5$$

Addition second

“Order of precedence”

First

$x^{**}y$ $-x$ $+x$ $x \% y$ x / y $x * y$ $x - y$ $x + y$

$x == y$ $x != y$ $x >= y$ $x > y$ $x <= y$ $x < y$

not x x **and** y x **or** y

Last

Exercise 3

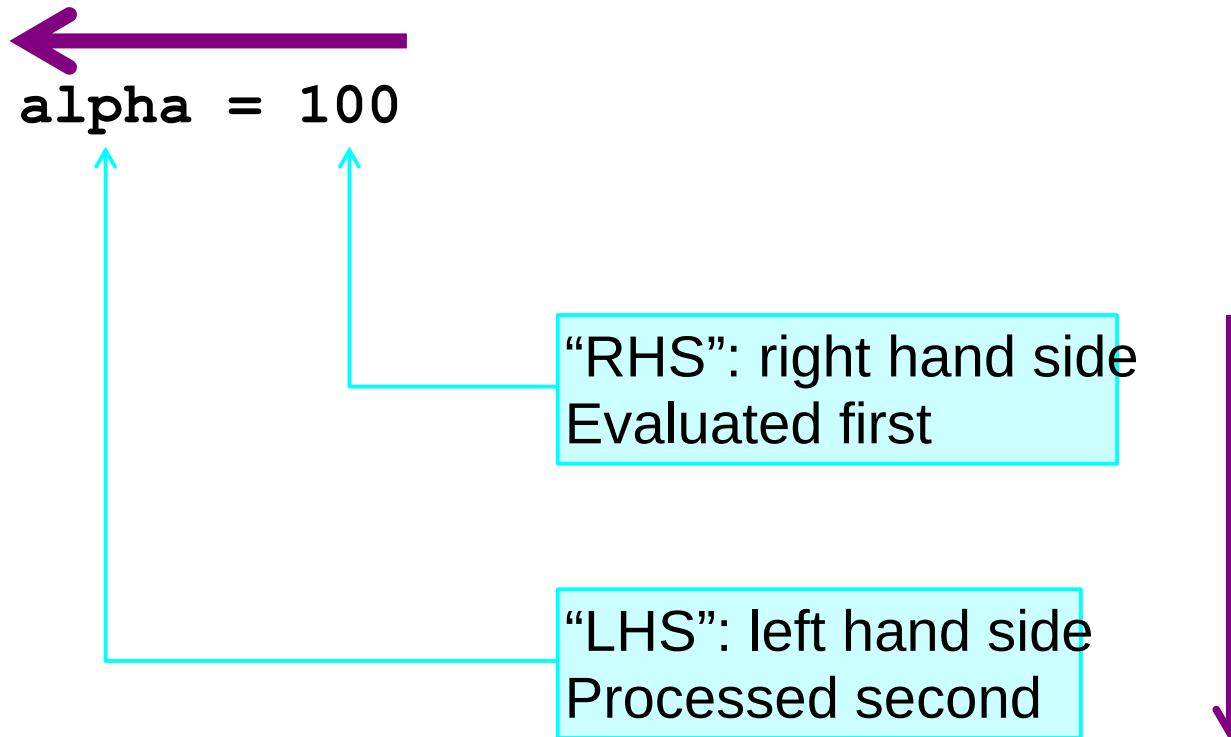
Predict whether these expressions will evaluate to True or False.
Then try them.

1. 'sparrow' > 'eagle'

2. 'dog' > 'Cat' or 45 % 3 == 0

3. 60 - 45 / 5 + 10 == 1

Assignment: right to left



“Syntactic sugar”

thing `+=` 10

thing = thing `+` 10

thing `-=` 10

thing = thing `-` 10

thing `*=` 10

thing = thing `*` 10

thing `/=` 10

is equivalent to

thing = thing `/` 10

thing `**=` 10

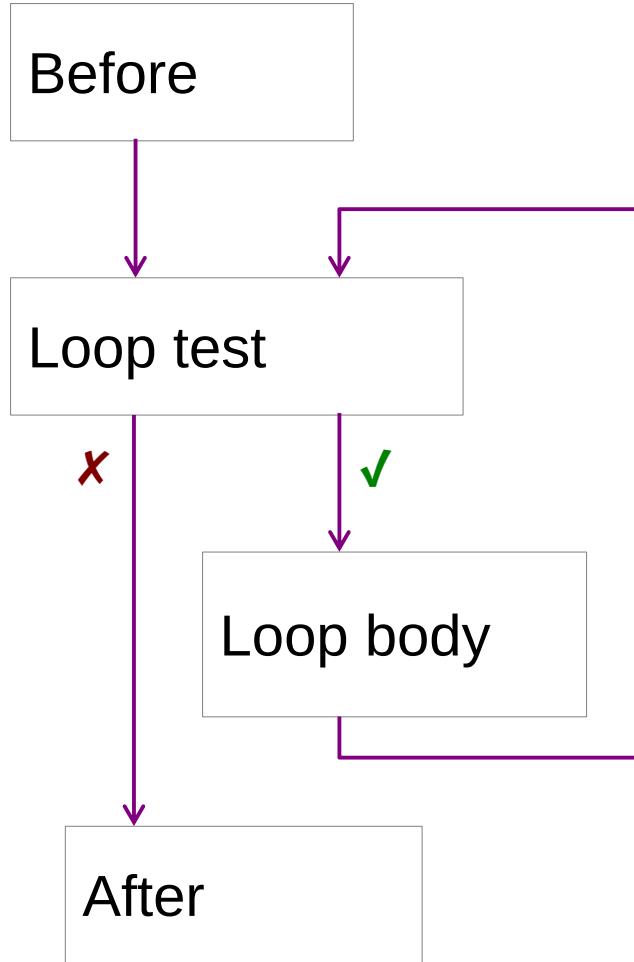
thing = thing `**` 10

thing `%=` 10

thing = thing `%` 10

Looping

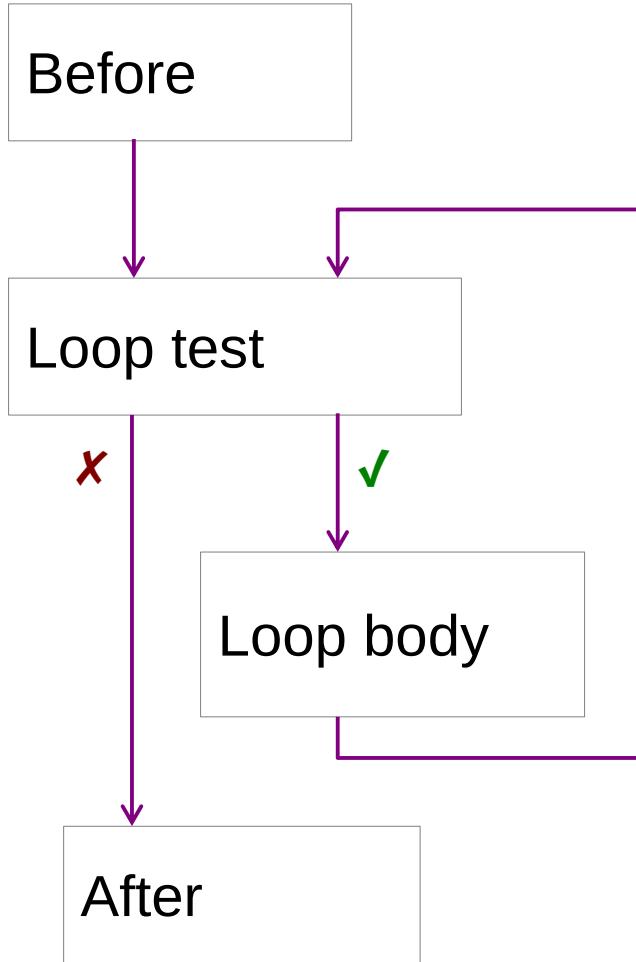
Looping



Should the
loop run
(again)?

What is
run each
loop?

Loop example: Count from 1 to 10



number = 1

number <= 10

print(number)
number += 1

print('Done!')

Loop example: Count from 1 to 10

```
number = 1
```

```
while number <= 10 :  
    print(number)  
    number += 1  
  
print('Done!')
```

```
number = 1
```

```
number <= 10  
X  
print(number)  
number += 1  
  
print('Done!')
```

Loop test: Count from 1 to 10

```
number = 1
```

```
while number <= 10 :
```

The code shows a while loop. The word "while" is followed by a condition "number <= 10" in a box. A colon ":" is preceded by an arrow pointing from the word "loop test" in a box.

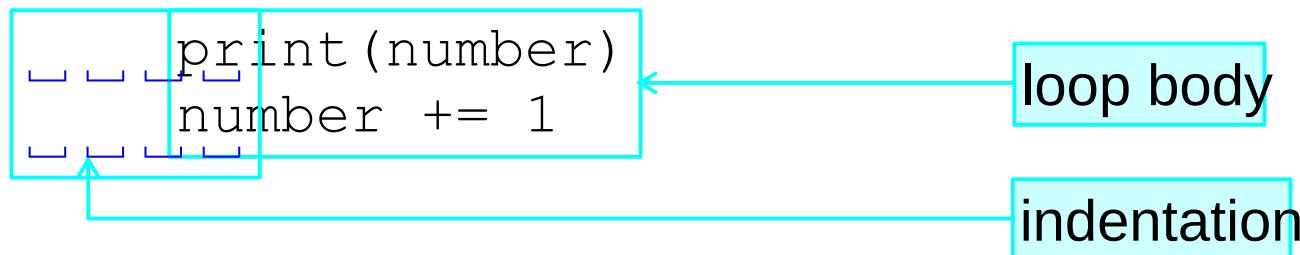
```
    print(number)  
    number += 1
```

```
print('Done!')
```

Loop body: Count from 1 to 10

```
number = 1
```

```
while number <= 10 :
```



```
print('Done!')
```

Loop example: Count from 1 to 10

```
number = 1  
  
while number <= 10 :  
  
    print(number)  
    number += 1  
  
print('Done!')
```

while1.py

```
$ python3 while1.py  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
Done!  
$
```

Python's use of indentation

```
number = 1

while number <= 10 :
    print(number)
    number += 1

print('Done!')
```

Four spaces' indentation indicate a “block” of code.

The block forms the repeated lines.

The first unindented line marks the end of the block.

Other languages

Shell

```
while ...
do
  ...
done
```

```
do ... done
```

Syntax

```
  ...
```

Clarity

C

```
while ...
{
  ...
}
```

```
{ ... }
```

Syntax

```
  ...
```

Clarity

Keep looping while ... ?

uncertainty > tolerance

```
while uncertainty > tolerance :
```

- Do stuff.

-

-

-

Square root: the loop

```
tolerance = 1.0e-15
```

Set up

```
lower = 0.0
```

```
upper = 2.0
```

```
uncertainty = upper - lower
```

```
while uncertainty > tolerance :
```

```
    middle = (lower + upper) / 2
```

Loop

?

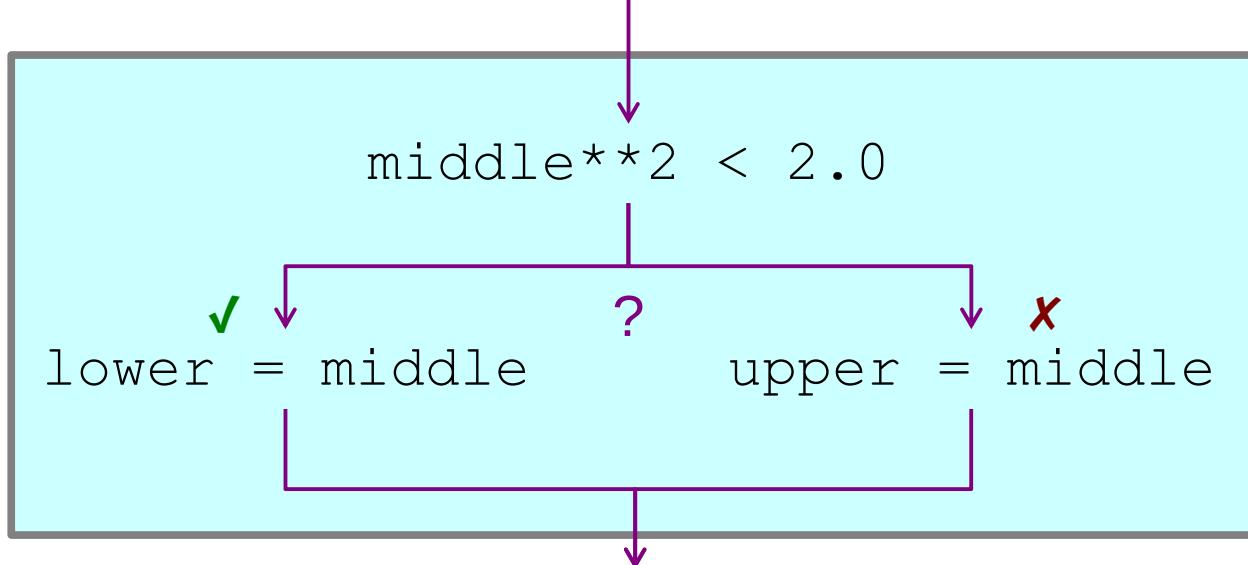
```
    print(lower, upper)
```

Choice

```
    uncertainty = upper - lower
```

Choosing

Choice



`middle ** 2 < 2.0` → True or False

True → `lower = middle`

False → `upper = middle`

Simple example

```
text = input('Number? ')
number = int(text)

if number % 2 == 0:
    print('Even number')
else:
    print('Odd number')

print('That was fun!')
```

ifthenelse1.py

```
$ python3 ifthenelse1.py
```

Number? **8**

Even number

That was fun

```
$ python3 ifthenelse1.py
```

Number? **7**

Odd number

That was fun

Lists

```
[ 'hydrogen', 'helium', 'lithium', 'beryllium', 'boron' ]  
  
[ -3.141592653589793, -1.570796326794896, 0.0, 1.5708  
  
[ 2, 3, 5, 7, 11, 13, 17, 19 ]
```

What is a list?

hydrogen, helium, lithium, beryllium, ..., protactinium, uranium

A *sequence* of values

The names of the elements

Values stored in order

Atomic number order

Individual value identified
by position in the sequence

“helium” is the name of the
second element

What is a list?

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59

A sequence of values

The prime numbers
less than sixty

Values stored in order

Numerical order

Individual value identified
by position in the sequence

7 is the fourth prime

Creating a list in Python

```
>>> primes = [ 2, 3, 5, 7, 11, 13, 17, 19]
```

A literal list

```
>>> primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

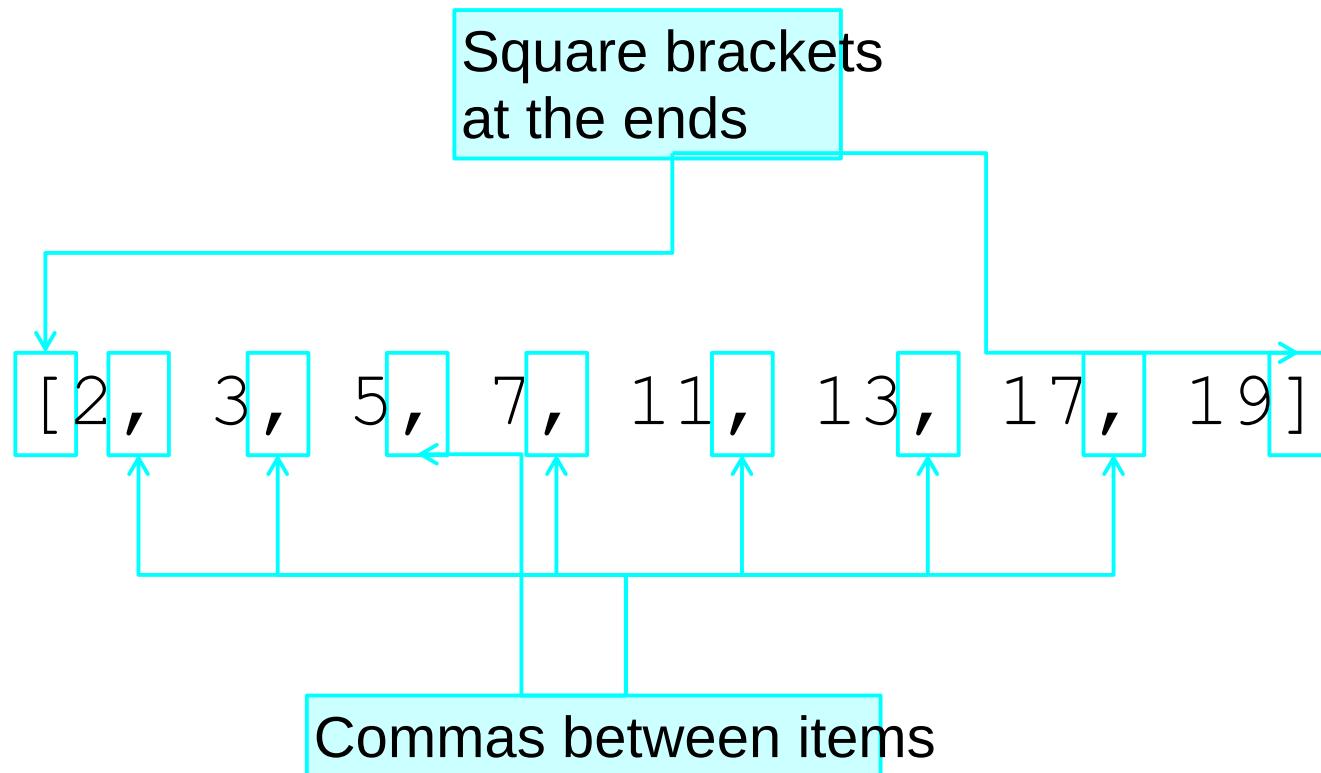
The whole list

```
>>> type(primes)
```

```
<class 'list'>
```

A Python type

How Python presents lists

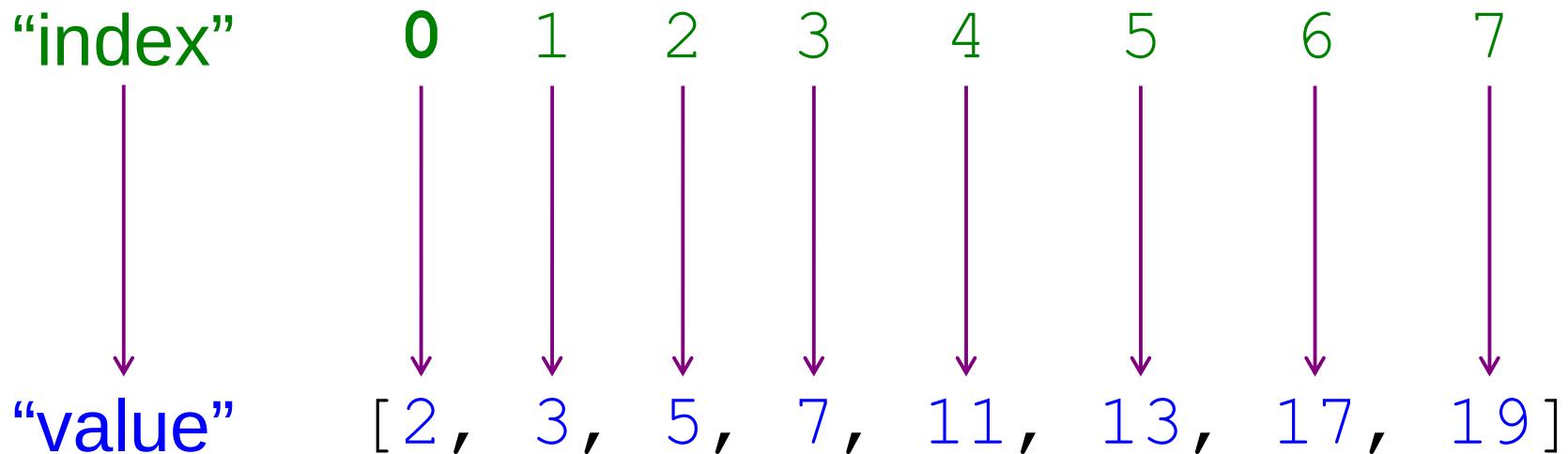


Square brackets

```
primes = [2, 3, 5, 7, 11]
```

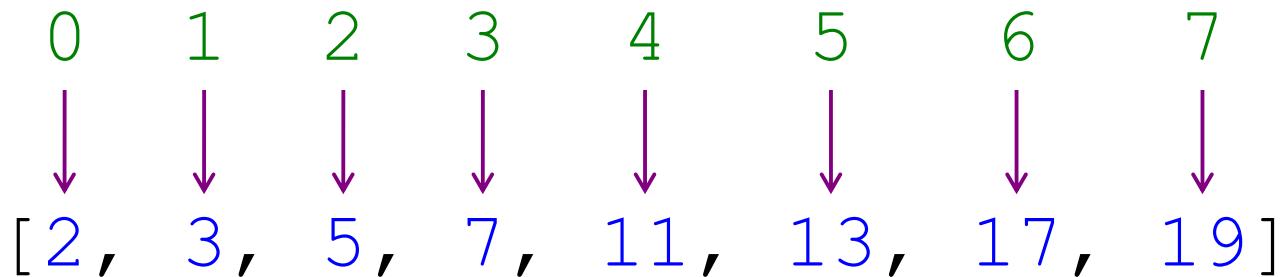
Literal list

Python counts from **zero**



Looking things up in a list

```
>>> primes = [ 2, 3, 5, 7, 11, 13, 17, 19]
```



index

```
>>> primes[0]
```

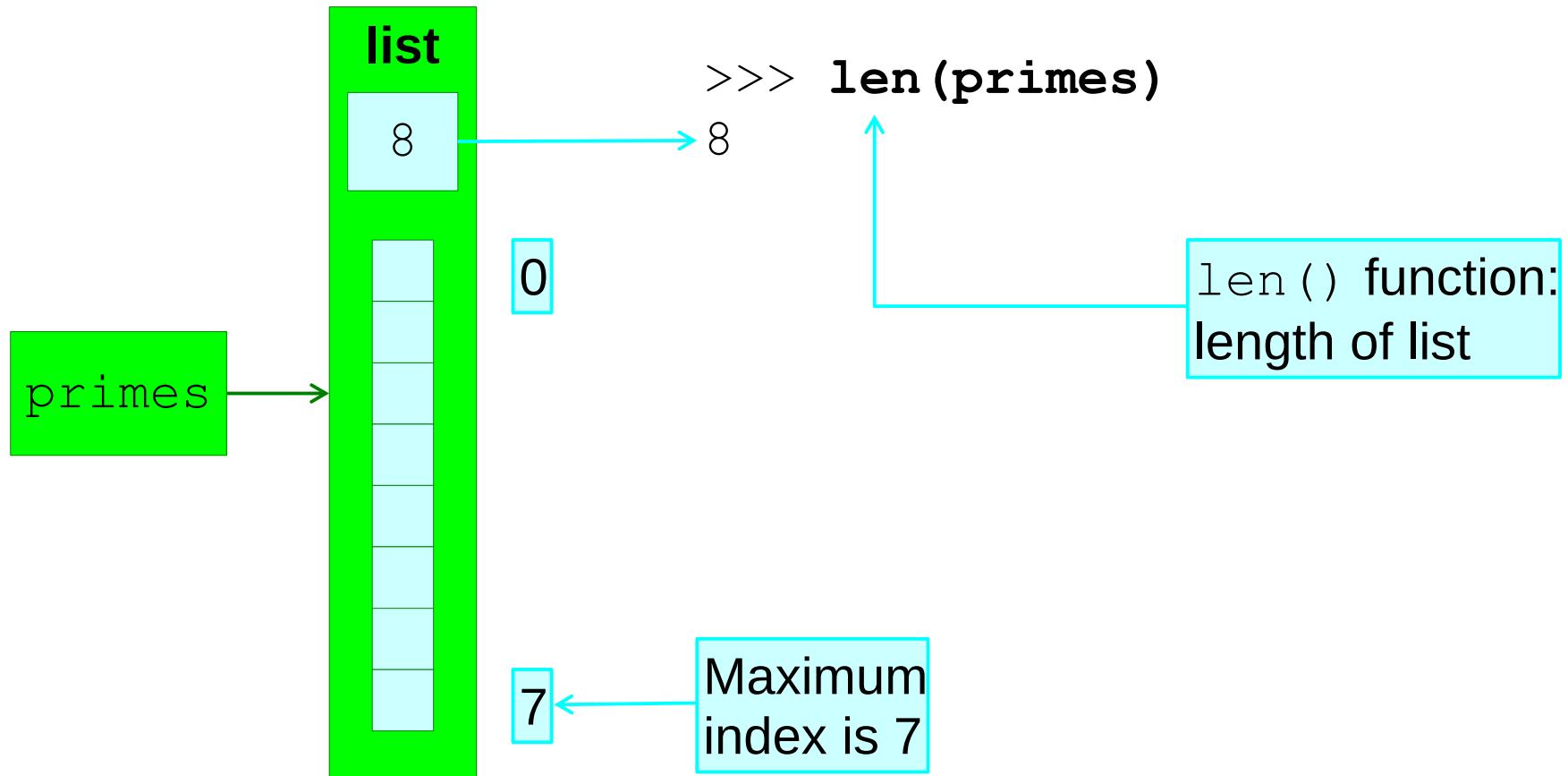
2

square brackets

```
>>> primes[6]
```

17

Length of a list



Changing a value in a list

```
>>> data = ['alpha', 'beta', 'gamma']
```

The list

```
>>> data[2]
```

Initial value

```
'gamma'
```

```
>>> data[2] = 'G'
```

Change value

```
>>> data[2]
```

Check change

```
'G'
```

```
>>> data
```

```
['alpha', 'beta', 'G']
```

Changed list

Exercise 4

Track what the value of `numbers` is at each stage of this sequence of instructions.

- 1 `>>> numbers = [5, 7, 11, 13, 17, 19, 29, 31]`
- 2 `>>> numbers[1] = 3`
- 3 `>>> del numbers[3]`
- 4 `>>> numbers[3] = 37`
- 5 `>>> numbers[4] = numbers[5]`

Appending to a list

```
>>> primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

A function built into a list

```
>>> primes.append(23)
```

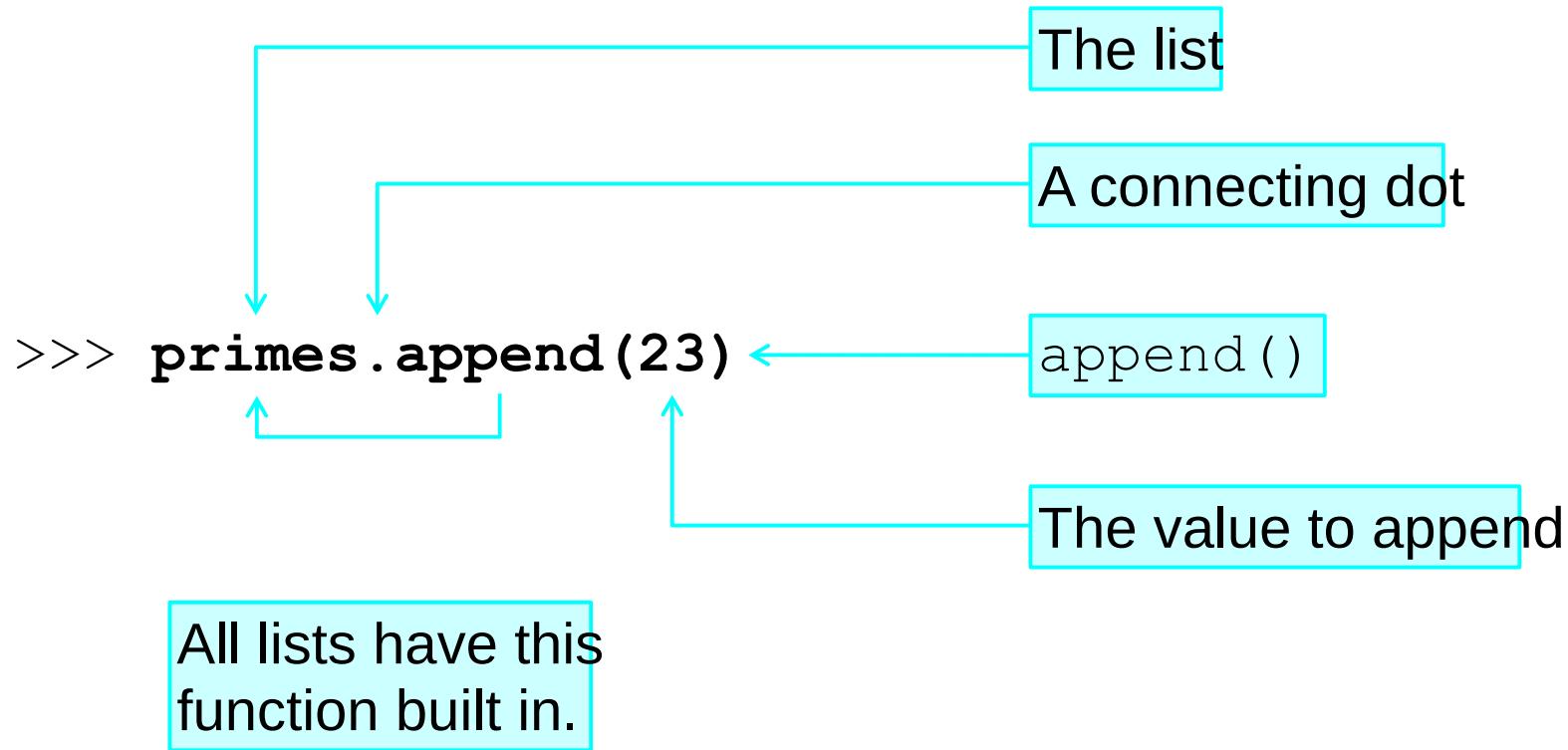
The list is now updated

```
>>> primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19,
```

```
23]
```

primes.append() ?



“Methods”

Behaves just
like a function

object . *function* (*arguments*)

a function that has
special access to
the object's data.

Using the append() method

```
>>> print(primes)
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

```
>>> primes.append(23)
```

```
<-->>> primes.append(29)
```

The function doesn't return any value.

```
>>> primes.append(31)
```

```
>>> primes.append(37)
```

It modifies the list itself.

```
>>> print(primes)
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37]
```

Other methods on lists: reverse ()

```
>>> numbers = [4, 7, 5, 1]
```

```
>>> numbers.reverse()
```

The function doesn't return any value.

```
>>> print(numbers)
```

It modifies the list itself.

```
[1, 5, 7, 4]
```

Other methods on lists: sort ()

```
>>> numbers = [4, 7, 5, 1]
```

```
>>> numbers.sort()
```

The function does not return the sorted list.

```
>>> print(numbers)
```

It sorts the list itself.

Numerical order.

Other methods on lists: sort ()

```
>>> greek = ['alpha', 'beta', 'gamma', 'delta']  
  
>>> greek.sort()  
  
>>> print(greek)  
['alpha', 'beta', 'delta', 'gamma']
```

Alphabetical order
of the words.

Other methods on lists: insert ()

0

1

2

```
>>> greek = ['alpha', 'gamma', 'delta']
```

```
>>> greek.insert(1, 'beta')
```

Where to insert

What to insert

```
>>> greek
```

```
['alpha', 'beta', 'gamma', 'delta']
```

0

1

Displaced

Other methods on lists: remove ()

```
>>> numbers = [7, 4, 8, 7, 2, 5, 4]  
>>> numbers.remove(8)           Value to remove
```

```
>>> print(numbers)
```

```
[7, 4, 7, 2, 5, 4]
```

c.f. `del numbers[2]` Index to remove

Other methods on lists: remove ()

```
>>> print(numbers)
```

```
[7, 4, 7, 2, 5, 4]
```

There are two instances of 4.

```
>>> numbers.remove(4)
```

```
>>> print(numbers)
```

```
[7, 7, 2, 5, 4]
```

Only the first instance is removed

What methods are there?

```
>>> help(numbers)
```

Help on list object:

```
class list(object)
...
|   append(...)
|       L.append(object) -- append object to end
...
...
```

Pagination:  next page

 back one page

 quit

Help on a single method

```
>>> help(numbers.append)
```

Help on built-in function append:

```
append(....)
L.append(object) -- append object to end
```

Adding to a list *redux*: “+”

```
>>> primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19]
```

Concatenation
operator

```
>>> primes + [23, 29, 31]
```

List to add

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

Concatenation

Create a new list

```
>>> newlist = primes + [23, 29, 31]
```

Update the list

```
>>> primes = primes + [23, 29, 31]
```

Augmented assignment

```
>>> primes += [23, 29, 31]
```

Print a list

Printing each element on a line

```
[ 'The', 'cat', 'sat', 'on', 'the', 'mat.' ]
```



```
The  
cat  
sat  
on  
the  
mat.
```

The “for loop”—1

```
name of list
list
words = ['The', 'cat', 'sat', 'on', 'the', 'mat.']
for word in words :
    print(word)
```

A new Python looping construct

print: What we want to do with the list items.

```
graph TD; A["name of list"] --> B[words]; C["list"] --> D[words]; E["A new Python looping construct"] --> F[for]; G["print: What we want to do with the list items."] --> H[print]
```

The “for loop”—2

```
words = ['The', 'cat', 'sat', 'on', 'the', 'mat. ']
```

A diagram illustrating the structure of a Python for loop. The code is:

```
for word in words:  
    print(word)
```

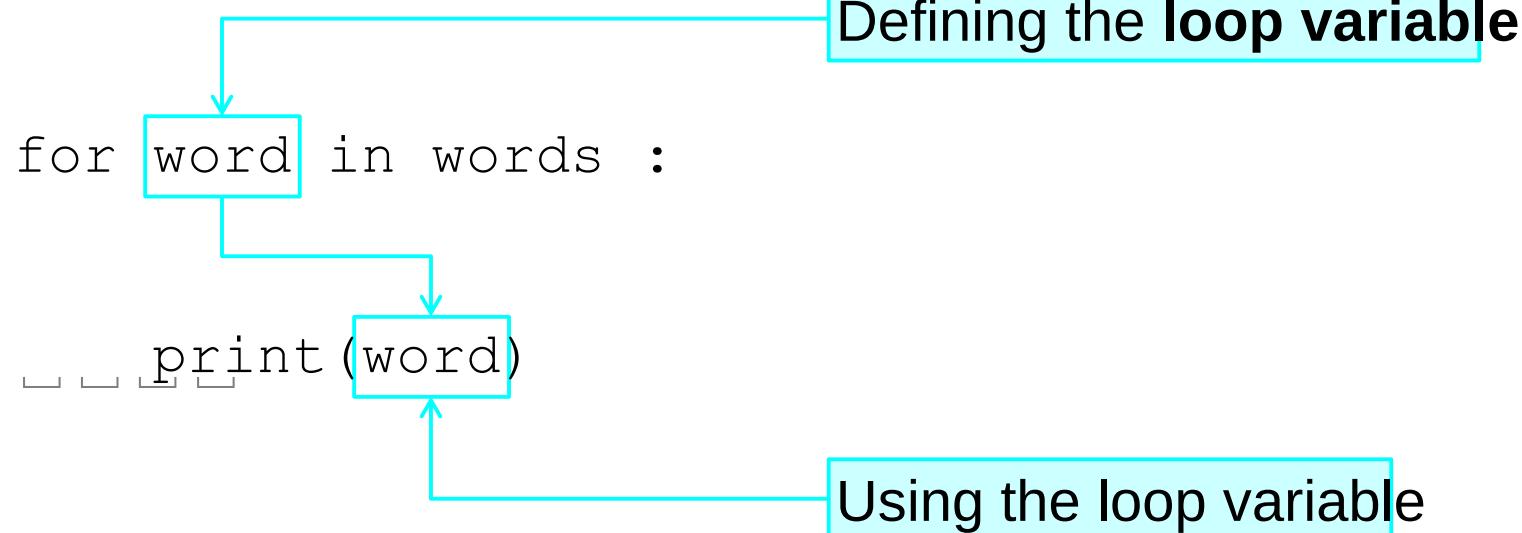
Annotations explain the structure:

- keywords**: A box containing "for", "in", and ":".
- colon followed by an indented block**: A box containing the line "print(word)".

Arrows point from the annotations to their respective parts in the code.

The “for loop”— 3

```
words = ['The', 'cat', 'sat', 'on', 'the', 'mat. ']
```



Exercise 5

What does this print?

```
numbers = [0, 1, 2, 3, 4, 5]

total = 0
total_so_far = []

for number in numbers:
    total += number
    total_so_far.append(total)

print(total_so_far)
```

List “slices”

```
>>> primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

```
>>> primes
```

```
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29] ← The list
```

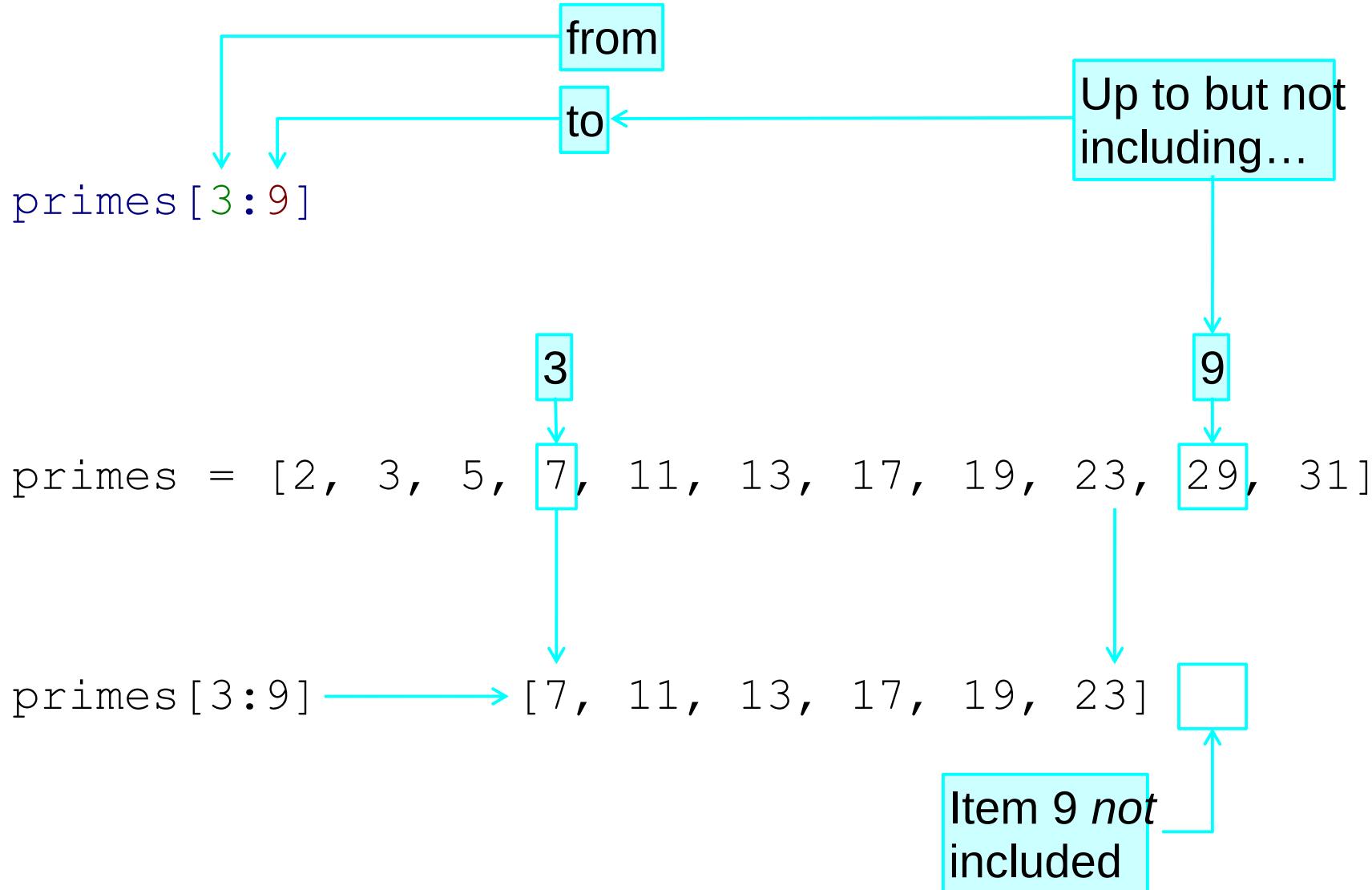
```
>>> primes[3]
```

```
7 ← An item
```

```
>>> primes[3:9]
```

```
[7, 11, 13, 17, 19, 23] ← Part of the list
```

Slices — 1



Slices — 2

primes	[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
primes[3:9]	[7, 11, 13, 17, 19, 23]
primes[:9]	[2, 3, 5, 7, 11, 13, 17, 19, 23]
primes[3:]	[7, 11, 13, 17, 19, 23, 29, 31]
primes[:]	[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]

Slices — 3

```
primes [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31]
```

```
primes[3:9] [7, 11, 13, 17, 19, 23]
```

```
primes[3:9:2] [7, 13, 19 ]
```

```
primes[3:9:3] [7, 17 ]
```



“

Function

Functions

$$y = f(x)$$



Functions we have met

`input(prompt)`

`bool(thing)`

`len(thing)`

`float(thing)`

`open(filename, mode)`

`int(thing)`

`print(line)`

`iter(list)`

`type(thing)`

`list(thing)`

`ord(char)`

`range(from, to, stride)`

`chr(number)`

`str(thing)`

Not that many!

“The Python Way”:
If it is appropriate to an object,
make it a method of that object.

Why write our own functions?

Easier to ...

... read

... write

... test

... fix

... improve

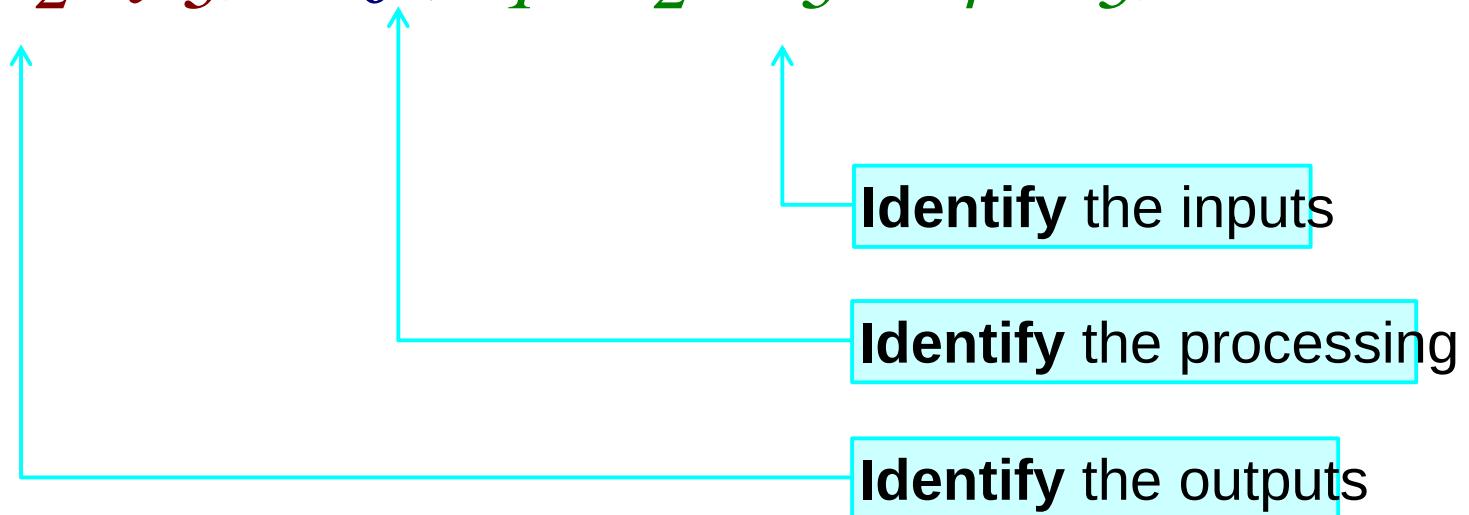
... add to

... develop

**“Structured
programming”**

Defining a function

$$(y_1, y_2, y_3) = f(x_1, x_2, x_3, x_4, x_5)$$



A function to define:
total ()

Sum a list

$$[1, 2, 3] \longrightarrow 6$$

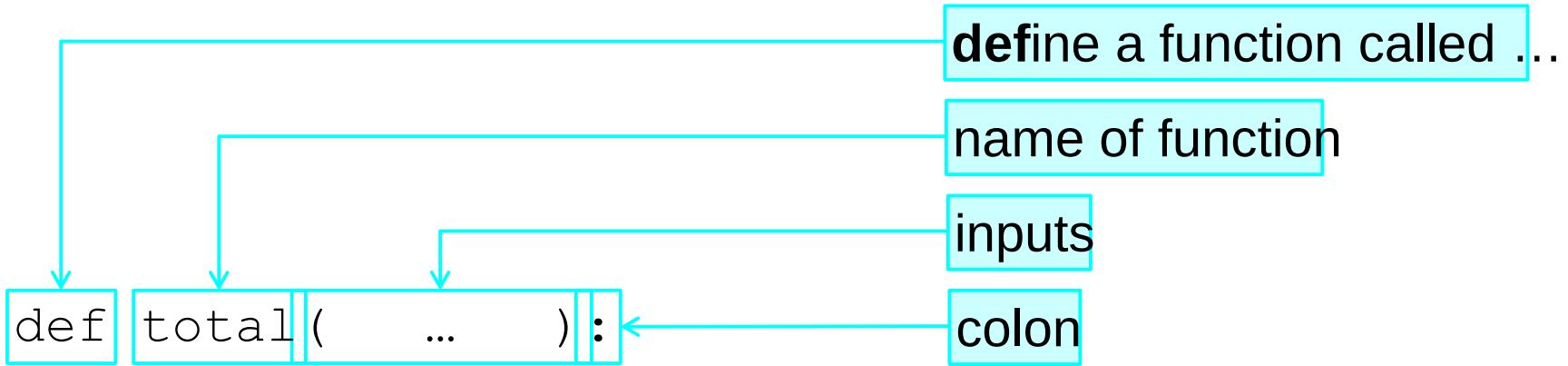
$$[7, -4, 1, 6, 0] \longrightarrow 10$$

$$[] \longrightarrow 0$$

“Edge case”

Defining a Python function —

1



Defining a Python function —

2

```
def total(numbers):
```

name for the input

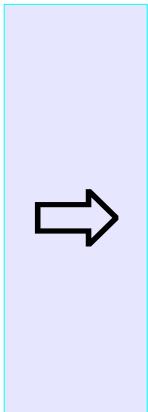
This name is
internal to
the function.

Defining a Python function —

3

```
def total(numbers)::
```

Colon followed by indentation



Defining a Python function —

4

```
def total(numbers):
```

```
    sum_so_far = 0

    for number in numbers:
        sum_so_far += number
```

“Body” of function

Defining a Python function —

4

```
def total(numbers):
    sum_so_far = 0
    for number in numbers:
        sum_so_far += number
```

These variables exist *only* within the function's body.

Defining a Python function —

5

```
def total(numbers):  
  
    sum_so_far = 0  
  
    for number in numbers:  
        sum_so_far += number  
  
    return sum_so_far
```

The diagram illustrates the flow of control in the `total` function. A blue arrow points from the `return` keyword up to the variable `sum_so_far`. Another blue arrow points from the text "This value is returned" down to the `sum_so_far` variable. A third blue arrow points from the text "return this value" down to the `return` keyword.

This value is returned

return this value

Defining a Python function —

6

And that's it!

```
def total(numbers):  
  
    sum_so_far = 0  
  
    for number in numbers:  
        sum_so_far += number  
  
    return sum_so_far
```

Unindented
after this



Defining a Python function —

7

And that's it!

All internal names *internal* → No need to avoid reusing names

All internal names cleaned up → No need for `del`

Using a Python function — 1

```
def total(numbers):  
  
    sum_so_far = 0  
  
    for number in numbers:  
        sum_so_far += number  
  
    return sum_so_far
```

```
print(total([1, 2, 3]))
```



The list we
want to add up

Using a Python function — 2

```
def total(numbers):  
  
    sum_so_far = 0  
  
    for number in numbers:  
        sum_so_far += number  
  
    return sum_so_far
```

```
print(total([1, 2, 3]))
```

The function we
have just written

Using a Python function — 3

```
def total(numbers):  
  
    sum_so_far = 0  
  
    for number in numbers:  
        sum_so_far += number  
  
    return sum_so_far
```

```
print(total([1, 2, 3]))
```

Printing out
the answer

Exercise 6

It currently defines and uses a function `total()` which adds the elements of a list.

Change it to create a function `threetimestotal()` which multiplies them with 3 for each elements before total them.

Functions with multiple inputs

```
def add_lists(a_list, b_list):  
    sum_list = []  
  
    for index in range(len(a_list)):  
        sum = a_list[index] + b_list[index]  
        sum_list.append(sum)  
  
    return sum_list
```

Multiple inputs are separated by commas

Functions with multiple inputs

```
def add_lists(a_list, b_list):  
    sum_list = []  
    for index in range(len(a_list)):  
        sum = a_list[index] + b_list[index]  
        sum_list.append(sum)  
    return sum_list
```

We have two lists...

...so we have to use indexing

Multiple outputs

Write a function to find minimum *and* maximum value in a list

[1, 2, 3] → 1 & 3

[10, -5] → -5 & 10

[3, 7, 4, 1, 7] → 1 & 7

Two outputs

Finding both

```
def minmax_list(a_list):  
  
    min_so_far = a_list[0]  
    max_so_far = a_list[0]  
  
    for a in a_list:  
  
        if a < min_so_far:  
            min_so_far = a  
  
        if a > max_so_far:  
            max_so_far = a
```

return *what?* ←

This is the real question 193

Returning both

```
def minmax_list(a_list):  
  
    min_so_far = a_list[0]  
    max_so_far = a_list[0]  
  
    for a in a_list:  
  
        if a < min_so_far:  
            min_so_far = a  
  
        if a > max_so_far:  
            max_so_far = a  
  
    return min_so_far, max_so_far
```

A pair of values

“Tuples”

e.g.

min_value, max_value

Pair

min_value, avg_value, max_value

Triple

Commas

Often written with parentheses:

(min_value, max_value)

(min_value, avg_value, max_value)

Using tuples to return values

```
def ...
```

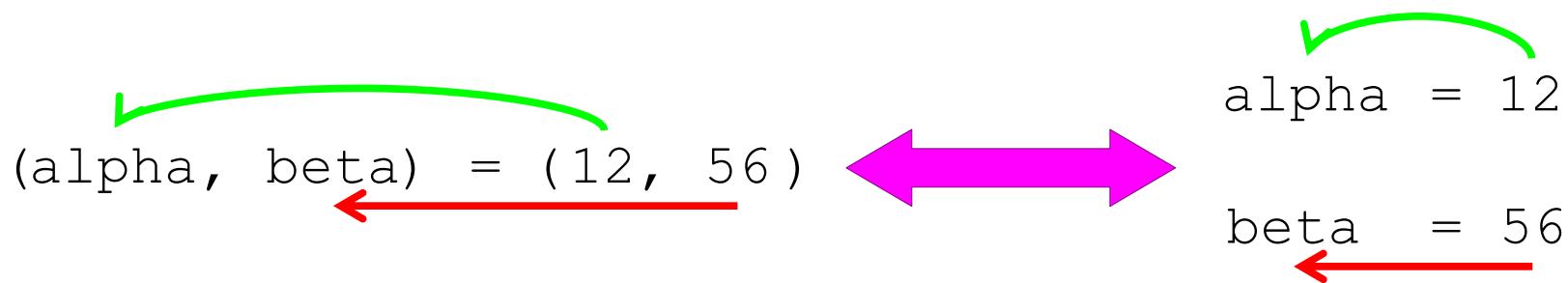
In the function definition

```
    return min_value , max_value
```

```
minimum , maximum = minmax_list(values)
```

Using the function

Using tuples to attach names



Swapping values

```
>>> alpha = 12
```

```
>>> beta = 56
```

```
>>> (alpha, beta) = (beta, alpha)
```

Swapping values

```
>>> print(alpha)
```

56

```
>>> print(beta)
```

12

Tuples and lists: similarities

```
>>> x = ['alpha', 'beta']
```

```
>>> x[1]  
'beta'
```

Indexing

```
>>> len(x)  
2
```

Length

```
>>> (a, b) = (1, 2)
```

```
>>> a
```

1

Simultaneous
assignment

```
>>> y = ('alpha', 'beta')
```

```
>>> y[1]  
'beta'
```

```
>>> len(y)
```

2

```
>>> [c, d] = [1, 2]
```

```
>>> c
```

1

Tuples and lists: differences

```
>>> x = ['alpha', 'beta']
```

```
>>> x[1] = 'B'
```

```
>>> x
```

```
['alpha', 'B']
```

Lists are
mutable

```
>>> y = ('alpha', 'beta')
```

```
>>> y[1] = 'B'
```

TypeError:
'tuple' object does not
support item assignment

Tuples are
immutable

Tuples and lists: philosophy

Lists

Sequential:
Concept of “next item”

Best with all items
of the same type

Serial

```
[2, 3, 5, 7, 11, 13,  
 17, 19, 23, 29, 31]
```

Sequence of prime numbers

Tuples

Simultaneous:
All items “at once”

Safe to have
multiple types

Parallel

```
('Dowling', 'Bob',  
 50, 105.0, 'rjd4')
```

Surname, forename,
age, weight, user id

Reusing functions within a script

```
def square(limit):  
    ...  
  
...  
  
squares_a = square(34)  
  
...  
  
five_squares = squares(5)  
  
...  
  
squares_b = squares(56)  
  
...
```

One definition

Multiple uses in
the same file

Easy!

Reusing functions between scripts?

```
def squares(limit):  
    ...
```

One definition

```
...  
squares_a = squares(34)  
...
```

```
...  
squares_b = squares(56)  
...
```

Multiple uses in
multiple files

```
five_squares = squares(5)
```

```
...
```

How?

“Modules”

```
def square(limit):
```

```
    ...
```

```
def cubes(limit):
```

```
    ...
```

```
...
```

```
five = squares(5)
```

```
...
```

Definition → Use

Module: a container of functions

Modules: a worked example

— 1a

utils.py

Starts empty

```
def squares(limit):  
    answer = []  
    for n in range(0,limit):  
        answer.append(n**2)  
    return answer
```

```
def total(numbers):  
    sum_so_far = 0  
    for number in numbers:  
        sum_so_far += number  
    return sum_so_far
```

```
text = input('Number? ')  
number = int(text)  
squares_n = squares(number)  
total_n = total(squares_n)  
print(total_n)
```

sum_squares.py

Modules: a worked example

— 1b

```
$ python3 sum_squares.py
```

Number? 5

30

$$= 0 + 1 + 4 + 9 + 16$$

```
$ python3 sum_squares.py
```

Number? 7

91

$$= 0 + 1 + 4 + 9 + 16 + 25 + 36$$

Modules: a worked example

— 2a

```
def squares(limit):  
    answer = []  
    for n in range(0,limit):  
        answer.append(n**2)  
    return answer  
  
def total(numbers):  
    sum_so_far = 0  
    for number in numbers:  
        sum_so_far += number  
    return sum_so_far
```

utils.py

```
text = input('Number? ')  
number = int(text)  
squares_n = squares(number)  
total_n = total(squares_n)  
print(total_n)
```

sum_squares.py

Move the definitions
into the other file.

Modules: a worked example

— 2b

```
$ python3 sum_squares.py
```

Number? **5**

```
Traceback (most recent call last):  
  File "sum_squares.py", line 4, in <module>  
    squares_n = squares(number)  
NameError: name 'squares' is not defined
```

Because we have (re)moved its definition.

Modules: a worked example

— 3a

```
def squares(limit):  
    answer = []  
    for n in range(0,limit):  
        answer.append(n**2)  
    return answer
```

```
def total(numbers):  
    sum_so_far = 0  
    for number in numbers:  
        sum_so_far += number  
    return sum_so_far
```

```
import utils  
  
text = input('Number? ')  
number = int(text)  
squares_n = squares(number)  
total_n = total(squares_n)  
print(total_n)
```

sum_squares.py

utils.py

import utils
and *not*
import utils.py

import: Make
a reference to
the other file.

Modules: a worked example

— 3b

```
$ python3 sum_squares.py
```

Number? **5**

```
Traceback (most recent call last):  
  File "sum_squares.py", line 4, in <module>  
    squares_n = squares(number)  
NameError: name 'squares' is not defined
```

Still can't find the function(s).

Modules: a worked example

— 4a

squares() →
utils.squares()

total() →
utils.total()

```
import utils

text = input('Number? ')
number = int(text)
squares_n = utils.squares(number)
total_n = utils.total(squares_n)
print(total_n)
```

sum_squares.py

utils....: Identify
the functions
as coming from
the module.

Modules: a worked example

— 4b

```
$ python3 sum_squares.py
```

Number? 5

30

```
$ python3 sum_squares.py
```

Number? 7

91

Working again!



Python Library

The Python philosophy

A small core
language ...

... plus lots
of modules



**“Batteries
included”**

Example: the “math” module

```
>>> import math
```

Load in the “math” module.

```
>>> math.sqrt(2.0)
```

Run the `sqrt()` function...

... from the `math` module.

```
1.4142135623730951
```

```
import module as alias
```

```
>>> import math
```

```
>>> math.sqrt(2.0)
```

```
1.4142135623730951
```

Too long to keep typing?

```
>>> import math as m
```

“Alias”

```
>>> m.sqrt(2.0)
```

```
1.4142135623730951
```

Don't do these

```
>>> from math import sqrt
```

```
>>> sqrt(2.0)
```

```
1.4142135623730951
```

*Much better
to track the module.* !



```
>>> from math import *
```

```
>>> sqrt(2.0)
```

```
1.4142135623730951
```



What system modules are there?

Python 3.2.3 comes with over **250** modules.

glob

math

argparse

csv

io

cmath

datetime

html

os

random

getpass

json

signal

colorsys

logging

re

subprocess

email

pickle

string

sys

http

sqlite3

unicodedata

tempfile

webbrowser

unittest

xml

Additional downloadable modules

Numerical

numpy

scipy

panda

Graphics

matplotlib

Databases

pyodbc

psycopg2

MySQLdb

cx_oracle

ibm_db

pymssql

PostgreSQL

MySQL

Oracle

DB2

SQL Server

Help with modules

```
>>> import math
```

```
>>> help(math)
```

NAME

math

DESCRIPTION

This module is always available. It provides access to the mathematical functions defined by the C standard.

...

Help with module functions

...

FUNCTIONS

`acos(x)`

Return the arc cosine (measured in radians) of `x`.

...

```
>>> math.acos(1.0)
```

0.0

Help with module constants

...

DATA

```
e = 2.718281828459045  
pi = 3.141592653589793
```

...

```
>>> math.pi
```

```
3.141592653589793
```

Help for our own modules?

```
def squares(limit):  
    answer = []  
    for n in range(0,limit):  
        answer.append(n**2)  
    return answer  
  
def total(numbers):  
    sum_so_far = 0  
    for number in numbers:  
        sum_so_far += number  
    return sum_so_far
```

utils.py

Basic help already
provided by Python

```
>>> import utils
```

```
>>> help(utils)
```

NAME

utils

FUNCTIONS

squares(limit)

total(numbers)

FILE

/home/y550/utils.py

Adding extra help text

```
"""Some utility functions  
from the Python for  
Absolute Beginners course  
"""
```

```
def squares(limit):  
    answer = []  
    for n in range(0,limit):  
        answer.append(n**2)  
    return answer
```

```
def total(numbers):  
    sum_so_far = 0  
    for number in numbers:  
        sum_so_far += number  
    return sum_so_far
```

utils.py

```
>>> import utils Fresh start
```

```
>>> help(utils)
```

NAME
utils

DESCRIPTION
Some utility functions
from the Python for
Absolute Beginners course

FUNCTIONS
squares(limit)
total(numbers)

Adding extra help text to functions

```
"""Some utility functions  
from the Python for  
Absolute Beginners course  
"""\n\ndef squares(limit):\n    """Returns a list of  
    squares from zero to  
    limit**2.\n    """\n    answer = []\n    for n in range(0,limit):\n        answer.append(n**2)\n    return answer
```

utils.py

```
>>> import utils Fresh start
```

```
>>> help(utils)
```

NAME

utils

DESCRIPTION

...

FUNCTIONS

squares(limit)

Returns a list of
squares from zero to
limit**2.

Adding extra help text to functions

```
"""Some utility functions  
from the Python for  
Absolute Beginners course  
"""\n\ndef squares(limit):\n    """Returns a list of  
    squares from zero to  
    limit**2.\n    """\n    answer = []\n    for n in range(0,limit):\n        answer.append(n**2)\n    return answer
```

utils.py

```
>>> import utils Fresh start
```

```
>>> help(utils.squares)
```

squares(limit)

Returns a list of
squares from zero to
limit**2.

NumPy

Without NumPy

```
from math import sin, pi

def sinc(x):
    if x == 0:
        return 1.0
    else:
        pix = pi*x
        return sin(pix)/pix

def step(x):
    if x > 0:
        return 1.0
    elif x < 0:
        return 0.0
    else:
        return 0.5
```

functions.py

```
>>> import functions as f
>>> xval = [x/3.0 for x in
range(-10,10)]
>>> yval1 = [f.sinc(x) for x
in xval]
>>> yval2 = [f.step(x) for x
in xval]
```

Python is a great language but needed a way to operate quickly and cleanly over multi-dimensional arrays.

With NumPy

```
from numpy import sin, pi
from numpy import vectorize
import functions as f

vsinc = vectorize(f.sinc)
def sinc(x):
    pix = pi*x
    val = sin(pix)/pix
    val[x==0] = 1.0
    return val

vstep = vectorize(f.step)
def step(x):
    y = x*0.0
    y[x>0] = 1
    y[x==0] = 0.5
    return y
```

functions2.py

```
>>> import functions2 as f
>>> from numpy import *
>>> x = r_[-10:10]/3.0
>>> y1 = f.sinc(x)
>>> y2 = f.step(x)
```

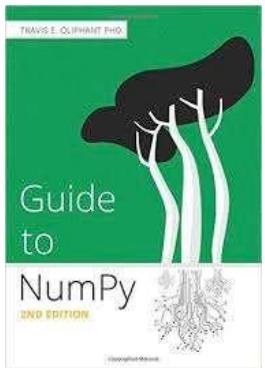
Offers N-D array, element-by-element functions, and basic random numbers, linear algebra, and FFT capability for Python

<http://numpy.org>

Fiscally sponsored by NumFOCUS

NumPy: an Array Extension of Python

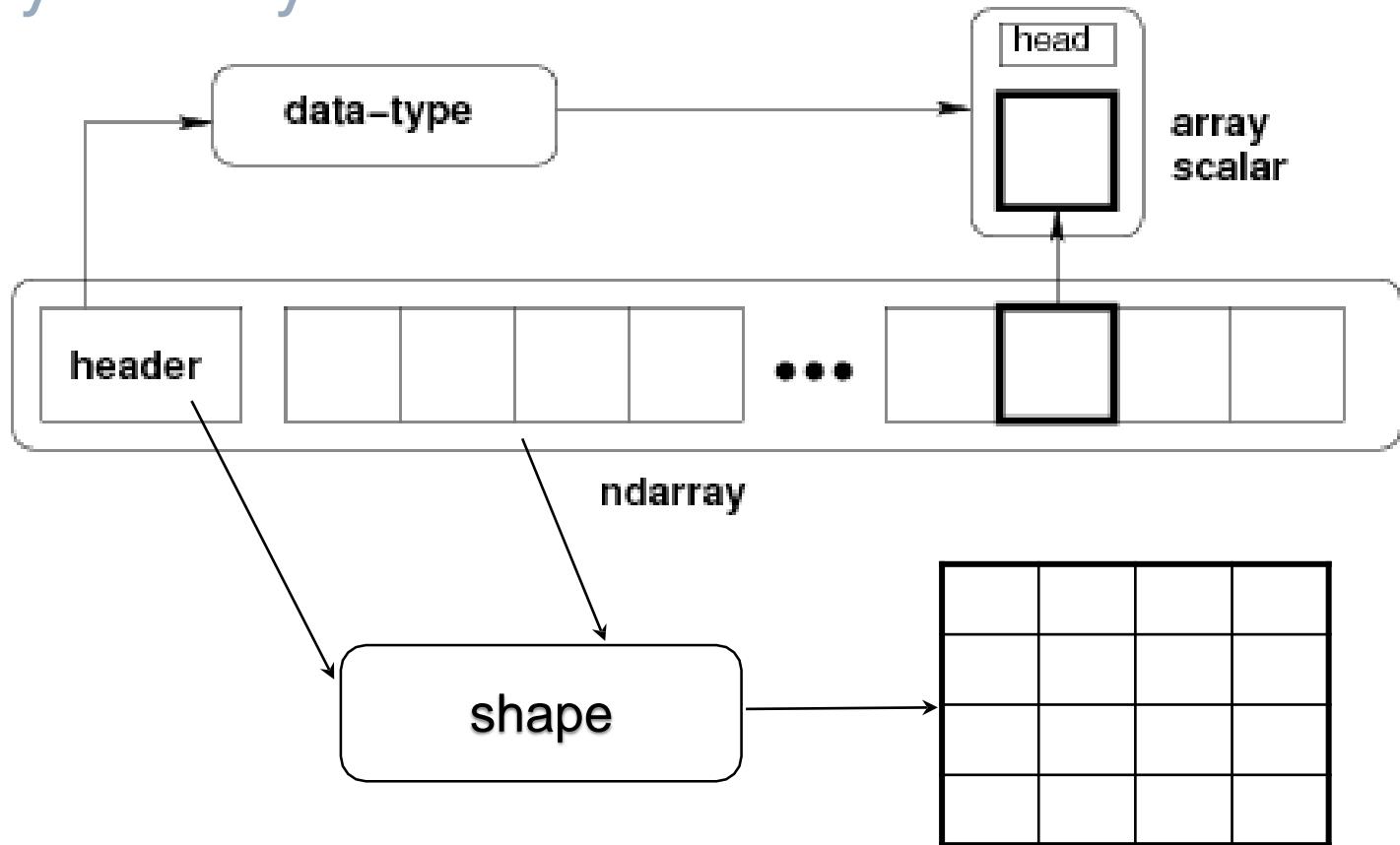
- Data: the array object
 - slicing and shaping
 - data-type map to Bytes
- Fast Math (ufuncs):
 - vectorization
 - broadcasting
 - aggregations



NumPy Array

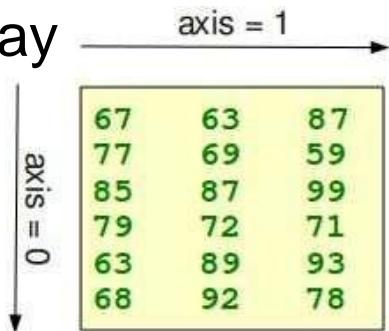
Key Attributes

- dtype
- shape
- ndim
- strides
- data



NumPy Examples

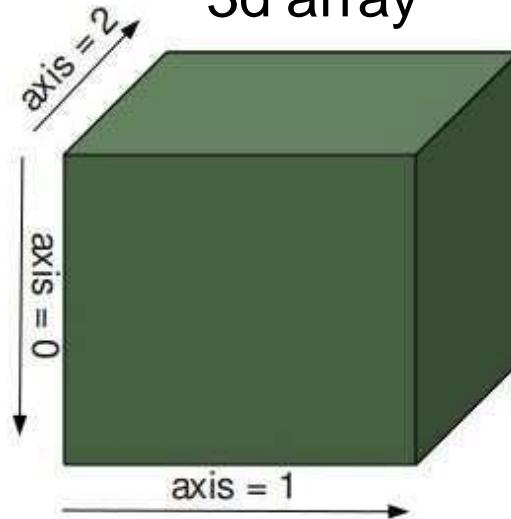
2d array



```
1 import numpy as np  
2  
3 x = np.array([[67, 63, 87],  
4 [77, 69, 59],  
5 [85, 87, 89],  
6 [79, 72, 71],  
7 [63, 89, 93],  
8 [68, 92, 78]])  
9 print x.sum(axis=0), x.sum(axis=1)
```

[439 472 477]
[217 205 261 222 245 238]

3d array



```
12 y = 3*np.random.randn(10,20,30)+10  
13 print y.mean(), y.std()
```

9.98330639789 2.96677717122

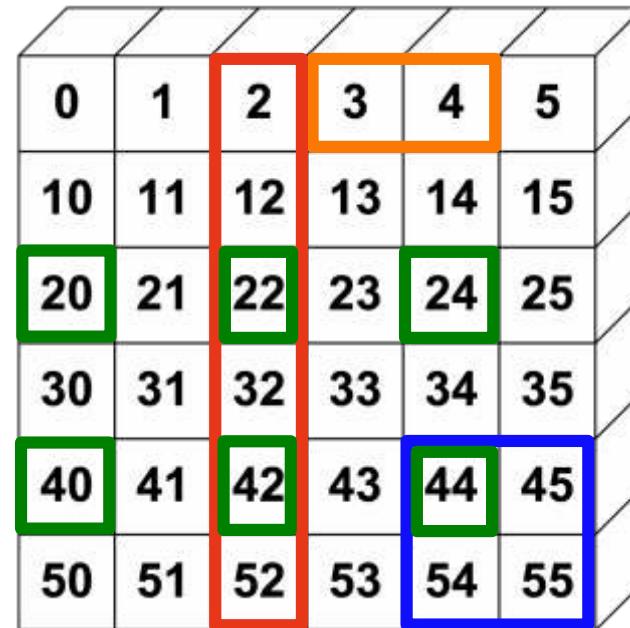
NumPy Slicing (Selection)

```
>>> a[0,3:5]  
array([3, 4])
```

```
>>> a[4:,4:]  
array([[44, 45],  
      [54, 55]])
```

```
>>> a[:,2]  
array([2,12,22,32,42,52])
```

```
>>> a[2::2,:2]  
array([[20, 22, 24],  
      [40, 42, 44]])
```



SciPy

SciPy

“Distribution of Python Numerical Tools masquerading as a Library”

Name	Description
cluster	KMeans and Vector Quantization
fftpack	Discrete Fourier Transform
integrate	Numerical Integration
interpolate	Interpolation routines
io	Data Input and Output
linalg	Fast Linear algebra
misc	Utilities
ndimage	N-dimensional Image processing

Name	Description
odr	Orthogonal Distance Regression
optimize	Constrained and Unconstrained Optimization
signal	Signal Processing Tools
sparse	Sparse Matrices and Algebra
spatial	Spatial Data Structures and Algorithms
special	Special functions (e.g. Bessel)
stats	Statistical Functions and Distributions

scipy.stats --- Continuous Distributions

94 continuous distributions!

Methods on all Random Variable Objects.

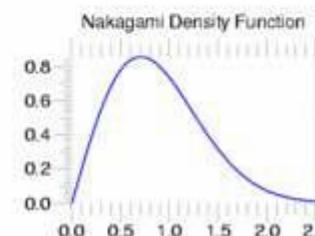
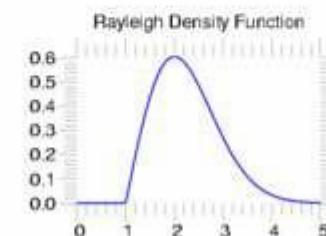
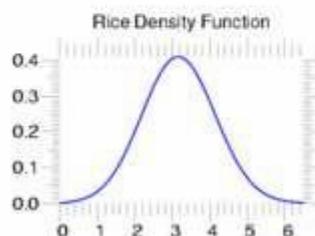
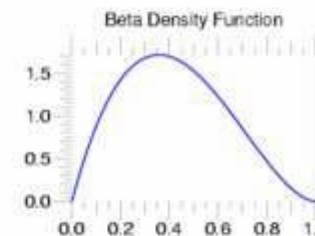
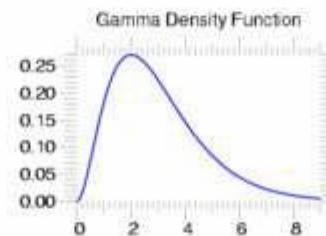
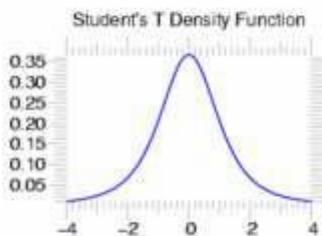
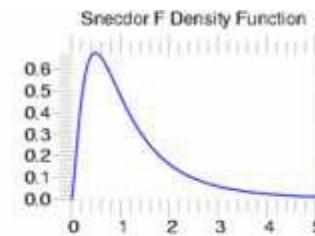
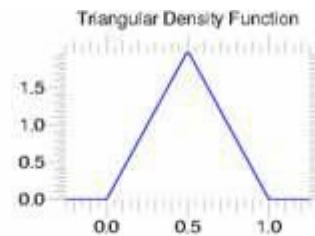
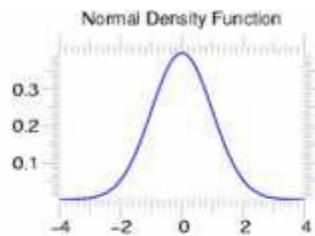
pdf

cdf

rvs

ppf

stats



scipy.stats --- Discrete Distributions

13 standard discrete distributions (plus any arbitrary finite RV)

Methods

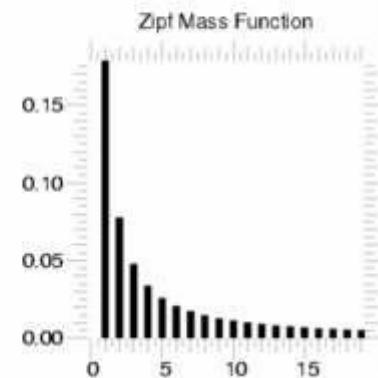
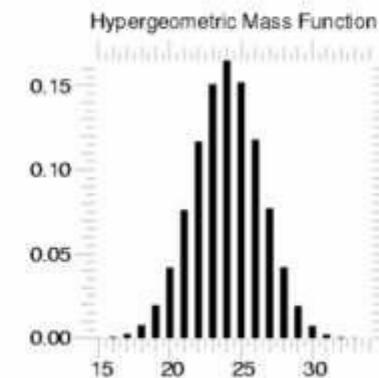
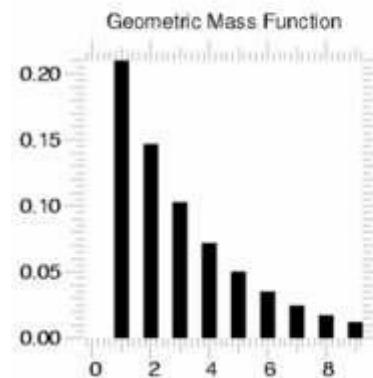
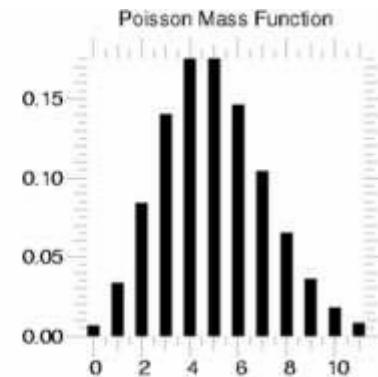
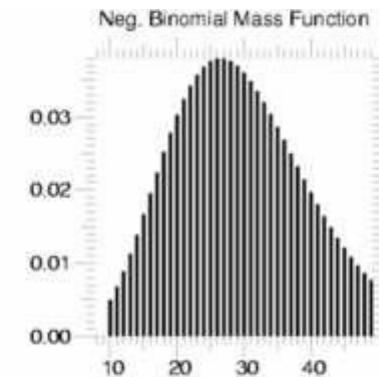
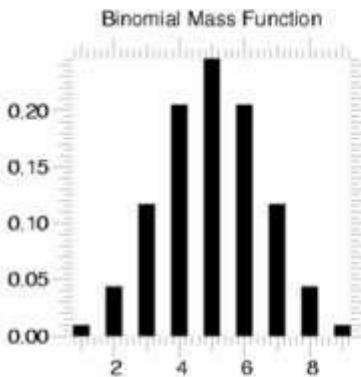
pdf

cdf

rvs

ppf

stats



Matplotlib

matplotlib

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

np.random.seed(0)

# example data
mu = 100 # mean of distribution
sigma = 15 # standard deviation of distribution
x = mu + sigma * np.random.randn(437)

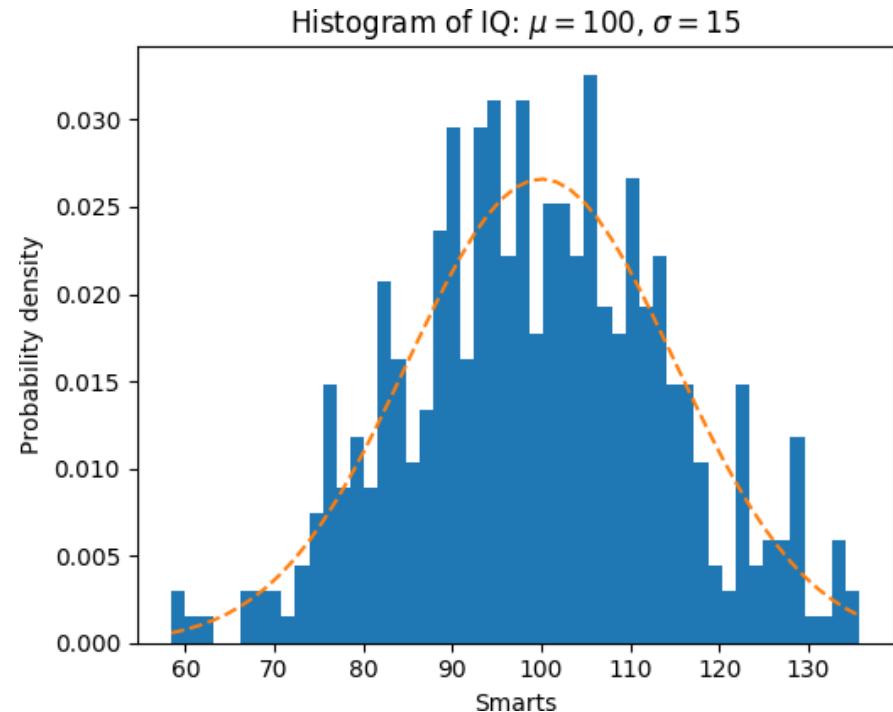
num_bins = 50

fig, ax = plt.subplots()

# the histogram of the data
n, bins, patches = ax.hist(x, num_bins, normed=1)

# add a 'best fit' line
y = mlab.normpdf(bins, mu, sigma)
ax.plot(bins, y, '--')
ax.set_xlabel('Smarts')
ax.set_ylabel('Probability density')
ax.set_title(r'Histogram of IQ: $\mu=100$', '$\sigma=15$')

# Tweak spacing to prevent clipping of ylabel
fig.tight_layout()
plt.show()
```



matplotlib

```
import matplotlib.pyplot as plt
import scipy.misc as misc

im = misc.face()
ax = plt.imshow(im)

plt.title('Racoon Face of size
%d x %d' % im.shape[:2])

plt.savefig('face.png')
```



matplotlib

```
import matplotlib.pyplot as plt
import numpy as np

with plt.xkcd():
    fig = plt.figure()
    ax = fig.add_axes((0.1, 0.2, 0.8, 0.7))
    ax.spines['right'].set_color('none')
    ax.spines['top'].set_color('none')
    plt.xticks([])
    plt.yticks([])
    ax.set_ylim([-30, 10])
    data = np.ones(100)
    data[70:] -= np.arange(30)

    plt.annotate('THE DAY I REALIZED\nI COULD COOK BACON\nWHENEVER I WANTED',
                 xy=(70, 1),
                 arrowprops=dict(arrowstyle='->'),
                 xytext=(15, -10))

    plt.plot(data)

    plt.xlabel('time')
    plt.ylabel('my overall health')
    fig.text(0.5, 0.05, '"Stove Ownership" from xkcd by Randall Monroe',
            ha='center')
```



"STOVE OWNERSHIP" FROM XKCD BY RANDALL MONROE

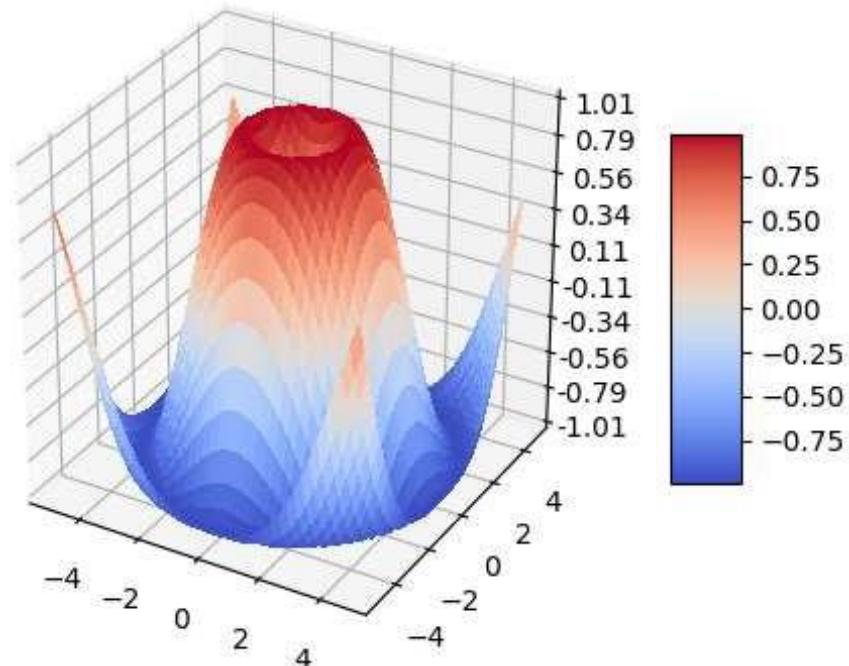
matplotlib

```
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator,
FormatStrFormatter
import numpy as np

fig = plt.figure()
ax = fig.gca(projection='3d')
# Make data.
X = np.arange(-5, 5, 0.25)
Y = np.arange(-5, 5, 0.25)
X, Y = np.meshgrid(X, Y)
R = np.sqrt(X**2 + Y**2)
Z = np.sin(R)
# Plot the surface.
surf = ax.plot_surface(X, Y, Z, cmap=cm.coolwarm,
linewidth=0, antialiased=False)

# Customize the z axis.
ax.set_zlim(-1.01, 1.01)
ax.zaxis.set_major_locator(LinearLocator(10))
ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))

# Add a color bar which maps values to colors.
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.show()
```



Pandas

Easy Data Wrangling

pandas

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



- Adds **indexes and labels** to 1-d and 2-d NumPy arrays (Series and DataFrame)
- Many convenience functions and methods to manipulate messy data-sets including **time-series**.
- Powerful indexing with automatic **data alignment**.
- Easy handling of **missing data**.
- Allows easy **joining and merging** Data Sets
- **Pivots** and reshaping (split-apply-combine)
- Powerful **group-by** operations with **summarization**
- Built-in visualization using labels and indexes

Easy Data Wrangling

pandas

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



- Series Data Structure
 - built for **1-dimensional** series data
 - homogeneous data
 - Two arrays. One of data and another which is the index that can be a homogeneous array of any type like integers, objects, or date-times.
- DataFrame
 - built for **2-dimensional** collections of tabular data (think Excel sheet)
 - heterogeneous data comprised of multiple Series
 - includes an index column allowing sophisticated selection and alignment

Easy Data Wrangling

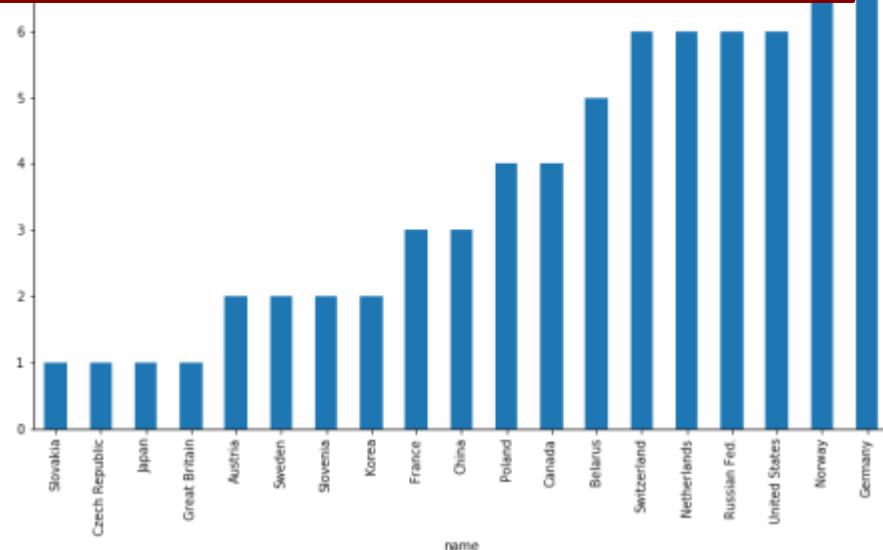
pandas

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



```
medals = pd.read_csv('data/medals.csv', index_col='name')
medals.head()
gold = medals['medal'] == 'gold'
won = medals['count'] > 0
medals.loc[gold & won, 'count'].sort_values().plot(kind='bar', figsize=(12,8))
```

	count	medal	country
name			
Australia	1	bronze	AUS
Australia	2	silver	AUS
Australia	0	gold	AUS
Austria	1	bronze	AUT
Austria	6	silver	AUT



Easy Data Wrangling

pandas

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



```
google = pd.read_csv('data/goog.csv', index_col='Date', parse_dates=True)
google.info()
google.head()
google.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1257 entries, 2010-01-04 to 2014-12-31
Data columns (total 5 columns):
Open    1257 non-null float64
High    1257 non-null float64
Low     1257 non-null float64
Close   1257 non-null float64
Volume  194 non-null int64
dtypes: float64(5)
memory usage: 58.9 KB
```

	Open	High	Low	Close	Volume
Date					
2010-01-04	313.16	314.44	311.81	313.06	NaN
2010-01-05	313.28	313.61	310.46	311.68	NaN
2010-01-06	312.62	312.62	302.88	303.83	NaN
2010-01-07	304.40	304.70	296.03	296.75	NaN
2010-01-08	295.70	301.32	294.26	300.71	NaN

	Open	High	Low	Close	Volume
count	1257.000000	1257.000000	1257.000000	1257.000000	1.940000e+02
mean	375.275593	378.450247	372.132474	375.327064	1.937264e+06
std	115.684354	116.288827	114.935742	115.664301	9.842775e+05
min	218.940000	220.920000	216.600000	217.820000	7.040350e+05
25%	285.790000	288.760000	283.060000	285.450000	1.338451e+06
50%	318.330000	320.800000	315.180000	317.260000	1.684634e+06
75%	452.540000	456.020000	449.740000	452.830000	2.164369e+06
max	612.790000	613.830000	608.690000	609.470000	6.795393e+06

Easy Data Wrangling

pandas

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



```
df = pd.read_excel("data/pbpython/salesfunnel.xlsx")
df.head()
table = pd.pivot_table(df,
                       index=["Manager", "Rep", "Product"],
                       values=["Price", "Quantity"],
                       aggfunc=[np.sum, np.mean])
```

	Account	Name	Rep	Manager	Product	Quantity	Price	Status
0	714466	Trantow-Barrows	Craig Booker	Debra Henley	CPU	1	30000	presented
1	714466	Trantow-Barrows	Craig Booker	Debra Henley	Software	1	10000	presented
2	714466	Trantow-Barrows	Craig Booker	Debra Henley	Maintenance	2	5000	pending
3	737550	Fritsch, Russel and Anderson	Craig Booker	Debra Henley	CPU	1	35000	declined
4	146832	Kiehn-Spinka	Daniel Hilton	Debra Henley	CPU	2	65000	won

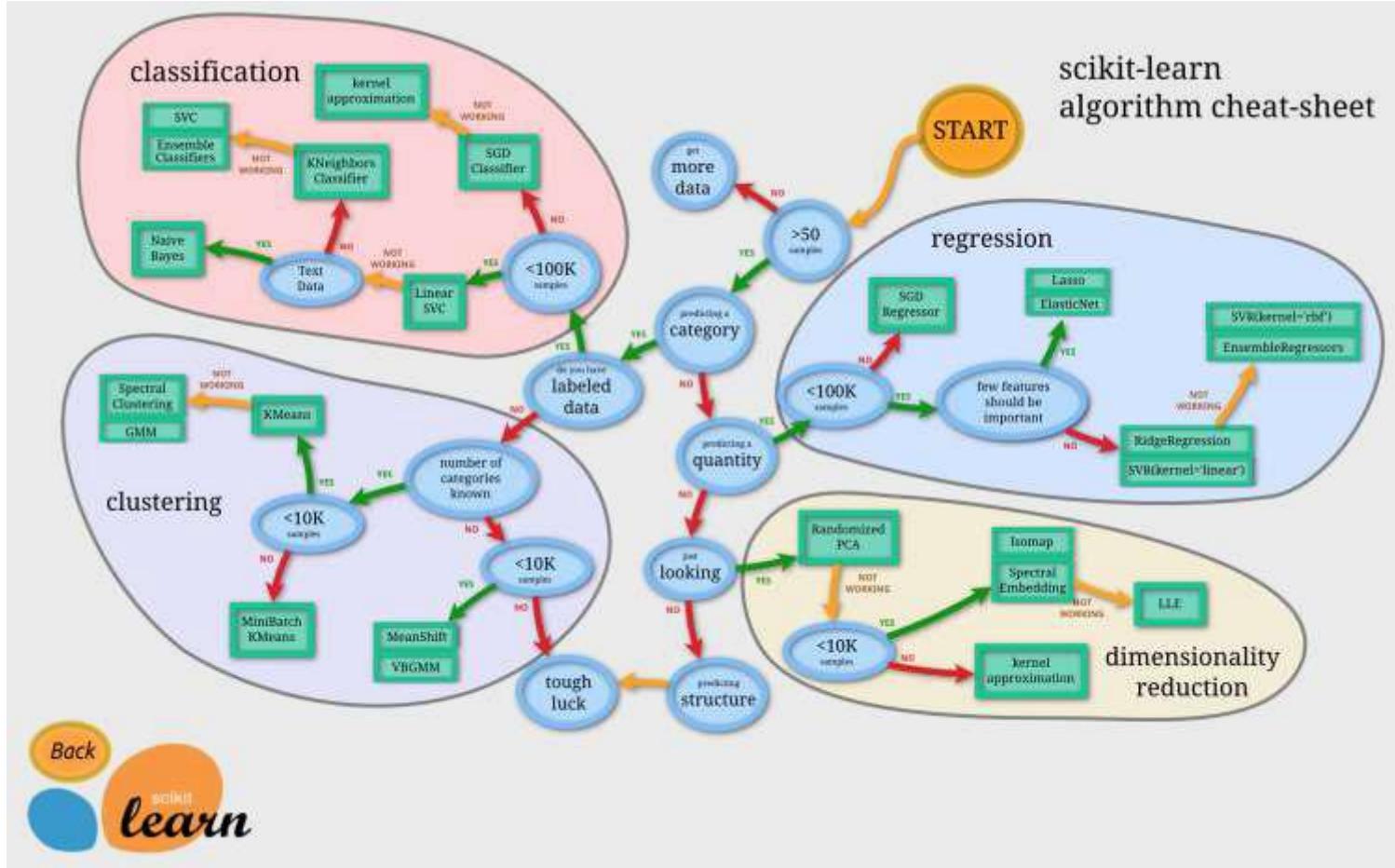
			sum		mean	
			Price	Quantity	Price	Quantity
Manager	Craig Booker	Product				
		CPU	65000	2	32500.0	1.0
		Maintenance	5000	2	5000.0	2.0
	Debra Henley	Software	10000	1	10000.0	1.0
		CPU	105000	4	52500.0	2.0
		Software	10000	1	10000.0	1.0
Rep	Daniel Hilton	CPU	35000	1	35000.0	1.0
		Maintenance	5000	2	5000.0	2.0
		Software	95000	3	47500.0	1.5
	John Smith	CPU	5000	1	5000.0	1.0
		Maintenance	10000	1	10000.0	1.0
		Software	165000	7	82500.0	3.5
Product	Cedric Moss	Maintenance	7000	3	7000.0	3.0
		Software	5000	2	5000.0	2.0
	Wendy Yule	CPU	70000	4	17500.0	3.5

Scikit- Learn



- **Supervised Learning** — uses “labeled” data to train a model
 - Regression — predicted variable is continuous
 - Classification — predicted variable is discrete
- **Unsupervised Learning**
 - Clustering — discover categories in the data
 - Density Estimation — determine representation of data
 - Dimensionality Reduction — represent data with fewer variables or feature vectors
- Reinforcement Learning — “goal-oriented” learning (e.g. drive a car)
- **Deep Learning** — neural networks with many layers
- Semi-supervised Learning (use some labeled data for training)

scikit-learn algorithm cheat-sheet



Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libraries

- matplotlib
- Seaborn

and many more ...

Python Libraries for Data Science



NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

Python Libraries for Data Science



SciPy:

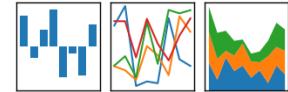
- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

Link: <https://www.scipy.org/scipylib/>

Python Libraries for Data Science

pandas

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



Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
- allows handling missing data

Link: <http://pandas.pydata.org/>

Python Libraries for Data Science



SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.
- built on NumPy, SciPy and matplotlib

Link: <http://scikit-learn.org/>

Python Libraries for Data Science



matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Link: <https://matplotlib.org/>

Python Libraries for Data Science

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

Link: <https://seaborn.pydata.org/>

Loading Python Libraries

```
In [ ]: #Import Python Libraries  
import numpy as np  
import scipy as sp  
import pandas as pd  
import matplotlib as mpl  
import seaborn as sns
```

Press Shift+Enter to execute the *jupyter* cell

Reading data using pandas

```
In [ ]: #Read csv file  
df = pd.read_csv("Salaries.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx', sheet_name='Sheet1', index_col=None,  
na_values=['NA'])  
pd.read_stata('myfile.dta')  
pd.read_sas('myfile.sas7bdat')  
pd.read_hdf('myfile.h5', 'df')
```

Exploring data frames

```
In [3]: #List first 5 records  
df.head()
```

Out[3] :

	rank	discipline	phd	service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

Data Frame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types

```
In [4]: #Check a particular column type  
df['salary'].dtype
```

```
Out[4]: dtype('int64')
```

```
In [5]: #Check types for all the columns  
df.dtypes
```

```
Out[4]: rank          object  
discipline    object  
phd            int64  
service         int64  
sex            object  
salary         int64  
dtype: object
```

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute description

dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Exercises 7

- ✓ Find how many records this data frame has;
- ✓ How many elements are there?
- ✓ What are the column names?
- ✓ What types of columns we have in this data frame?

Data Frames methods

Unlike attributes, python methods have *parenthesis*.

All attributes and methods can be listed with a *dir()* function:

dir(df)

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Data Frames Methods

```
In [10]: #List the column names  
df.columns
```

```
Out[10]: Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')
```

```
In [11]: #List the row Labels and the column names  
df.axes
```

```
Out[11]: [RangeIndex(start=0, stop=78, step=1),  
          Index(['rank', 'discipline', 'phd', 'service', 'sex', 'salary'], dtype='object')]
```

```
In [12]: #Number of dimensions  
df.ndim
```

```
Out[12]: 2
```

```
In [13]: #Total number of elements in the Data Frame  
df.size
```

```
Out[13]: 468
```

```
In [14]: #Number of rows and columns  
df.shape
```

```
Out[14]: (78, 6)
```

```
In [15]: #Output basic statistics for the numeric columns  
df.describe()
```

Out[15]:

	phd	service	salary
count	78.000000	78.000000	78.000000
mean	19.705128	15.051282	108023.782051
std	12.498425	12.139768	28293.661022
min	1.000000	0.000000	57800.000000
25%	10.250000	5.250000	88612.500000
50%	18.500000	14.500000	104671.000000
75%	27.750000	20.750000	126774.750000
max	56.000000	51.000000	186960.000000

```
In [16]: #Calculate mean for all numeric columns  
df.mean()
```

Out[16]: phd 19.705128
service 15.051282
salary 108023.782051
dtype: float64

Hands-on exercises



- ✓ Give the summary for the numeric columns in the dataset
- ✓ Calculate standard deviation for all numeric columns;
- ✓ What are the mean values of the first 50 records in the dataset?

Hint: use `head()` method to subset the first 50 records and then calculate the mean

```
In [17]: #Calculate the standard deviation (std() method) for all numeric columns  
# <your code goes here>  
df.std()
```

```
Out[17]: phd          12.498425  
service      12.139768  
salary     28293.661022  
dtype: float64
```

```
In [18]: #Calculate average of the columns in the first 50 rows  
# <your code goes here>  
df.head(50).mean()
```

```
Out[18]: phd          21.52  
service      17.60  
salary     113789.14  
dtype: float64
```

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

```
df['sex']
```

Method 2: Use the column name as an attribute:

```
df.sex
```

Note: there is an attribute *rank* for pandas data frames, so to select a column with a name "rank" we should use method 1.

Exercises 8

- ✓ Calculate the basic statistics for the *salary* column;
- ✓ Find how many values in the *salary* column (use *count* method);
- ✓ Calculate the average salary;

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group
- Similar to dplyr() function in R

```
In [ ]: #Group data using rank  
df_rank = df.groupby(['rank'])
```

```
In [ ]: #Calculate mean value for each numeric column per each group  
df_rank.mean()
```

	phd	service	salary
rank			
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Data Frames *groupby* method

Once groupby object is created we can calculate various statistics for each group:

```
In [ ]: #Calculate mean salary for each professor rank:  
df.groupby('rank')[['salary']].mean()
```

rank	salary
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping
- by default the group keys are sorted during the *groupby* operation.

You may want to pass `sort=False` for potential speedup:

```
In [ ]: #Calculate mean salary for each professor rank:  
df.groupby(['rank'], sort=False)[['salary']].mean()
```

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than \$120K:

```
In [ ]: #Calculate mean salary for each professor rank:  
df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

- > greater; >= greater or equal;
- < less; <= less or equal;
- == equal; != not equal;

```
In [ ]: #Select only those rows that contain female professors:  
df_f = df[ df['sex'] == 'Female' ]
```

Data Frames: Slicing

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

```
In [ ]: #Select column salary:  
df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

```
In [ ]: #Select column salary:  
df[['rank', 'salary']]
```

Data Frames: Selecting rows

If we need to select a range of rows, we can specify the range using ":"

```
In [ ]: #Select rows by their position:  
df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:

So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9

Data Frames: method loc

If we need to select a range of rows, using their labels we can use method loc:

```
In [ ]: #Select rows by their labels:  
df_sub.loc[10:20, ['rank', 'sex', 'salary']]
```

```
Out[ ]:
```

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frames: method iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc:

```
In [ ]: #Select rows by their labels:  
df_sub.iloc[10:20, [0, 3, 4, 5]]
```

```
Out[ ]:   rank  service     sex    salary  
26    Prof      19    Male  148750  
27    Prof      43    Male  155865  
29    Prof      20    Male  123683  
31    Prof      21    Male  155750  
35    Prof      23    Male  126933  
36    Prof      45    Male  146856  
39    Prof      18  Female  129000  
40    Prof      36  Female  137000  
44    Prof      19  Female  151768  
45    Prof      25  Female  140096
```

Data Frames: method iloc (summary)

```
df.iloc[0]    # First row of a data frame  
df.iloc[i]    #(i+1)th row  
df.iloc[-1]   # Last row
```

```
df.iloc[:, 0]  # First column  
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]      #First 7 rows  
df.iloc[:, 0:2]    #First 2 columns  
df.iloc[1:3, 0:2]  #Second through third rows and first 2  
columns  
df.iloc[[0,5], [1,3]] #1st and 6th rows and 2nd and 4th columns
```

Data Frames: Sorting

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is return.

```
In [ ]: # Create a new data frame from the original sorted by the column  
Salary  
df_sorted = df.sort_values( by ='service')  
df_sorted.head()
```

```
Out[ ]:
```

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

We can sort the data using 2 or more columns:

```
In [ ]: df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])  
df_sorted.head(10)
```

Out[]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500

Missing Values

Missing values are marked as NaN

```
In [ ]: # Read a dataset with missing values  
flights = pd.read_csv("flights.csv")
```

```
In [ ]: # Select the rows that have at least one missing value  
flights[flights.isnull().any(axis=1)].head()
```

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Missing Values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- `cumsum()` and `cumprod()` methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have `skipna` option to control if missing data should be excluded . This value is set to *True* by default (unlike R)

Handle Missing

```
In [52]: # Remove all the observations with missing values  
flights2 = flights.dropna()
```

```
In [53]: # Fill missing values with zeros  
nomiss = flights['dep_delay'].fillna(0)  
nomiss.isnull().any()
```

```
Out[53]: False
```

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max

count, sum, prod

mean, median, mode, mad

std, var

Aggregation Functions in Pandas

agg() method are useful when multiple statistics are computed per column:

```
In [ ]: flights[['dep_delay', 'arr_delay']].agg(['min', 'mean', 'max'])
```

```
Out[ ]:
```

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

Exploratory Data Analysis

Exploratory Data Analysis Concept

EDA is an iterative cycle. You:

- ▷ Generate questions about your data.
- ▷ Search for answers by visualizing, transforming, and modelling your data.
- ▷ Use what you learn to refine your questions and/or generate new questions.

EDA Question

1. What type of **variation** occurs within my variables?
2. What type of **covariation** occurs between my variables?

Variation

- ▷ **Variation** is the tendency of the values of a variable to change from measurement to measurement.
- ▷ Simple Rules

Data Type	EDA Graph
1. Categorical Data	Bar Plot
2. Numerical Data	Histogram

Covariation

- ▷ If variation describes the behavior *within* a variable, covariation describes the behavior *between* variables.
- ▷ **Covariation** is the tendency for the values of two or more variables to vary together in a related way.

Data Type Between	EDA Graph
1. A Categorical and Numerical Data	Frequency Plot, Box Plot
2. A Numerical and Numerical Data	Scatter Plot
3. A Categorical and Categorical Data	Heatmap

Graphics to explore the data

Seaborn package is built on matplotlib but provides high level interface for drawing attractive statistical graphics, similar to ggplot2 library in R. It specifically targets statistical data visualization

To show graphs within Python notebook include inline directive:

```
In [ ]: %matplotlib inline
```

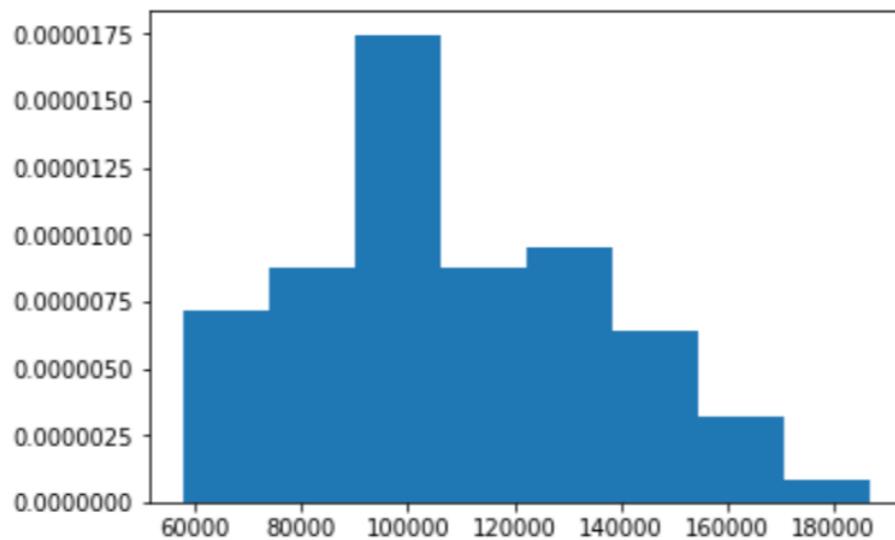
Graphics

description	
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

Histogram using Matplotlib

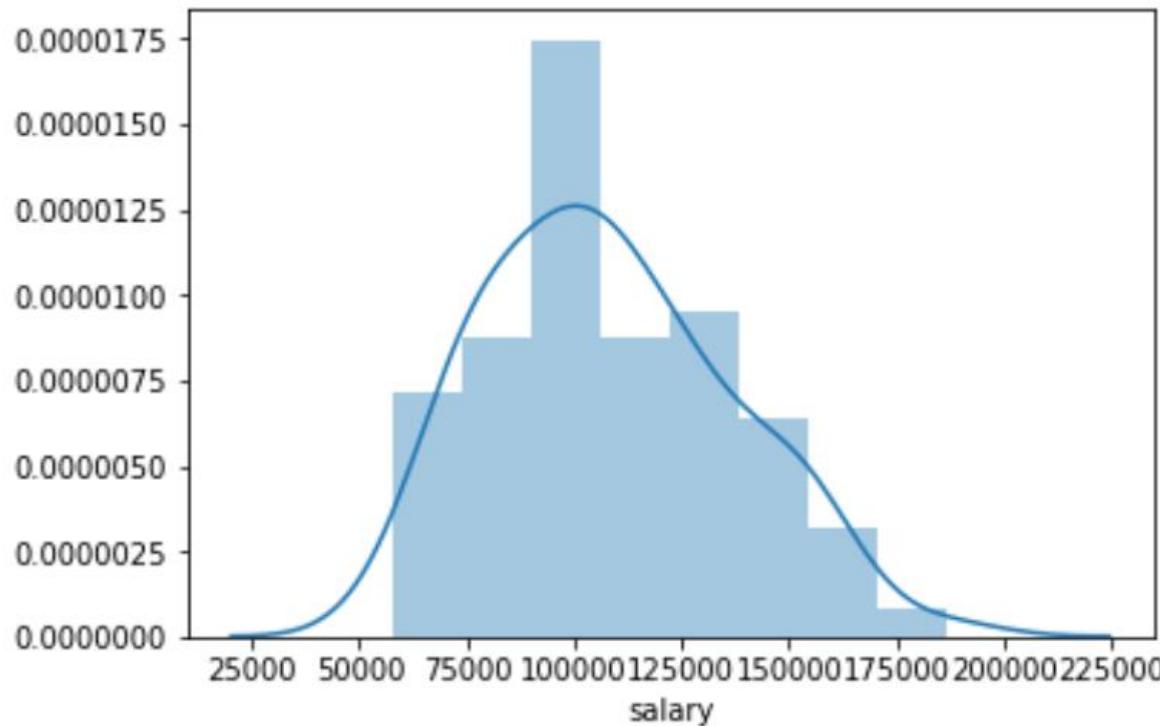
```
In [64]: #Use matplotlib to draw a histogram of a salary data  
plt.hist(df['salary'], bins=8, normed=1)
```

```
Out[64]: (array([ 7.14677085e-06,  8.73494215e-06,  1.74698843e-05,  
   8.73494215e-06,  9.52902780e-06,  6.35268520e-06,  
   3.17634260e-06,  7.94085650e-07]),  
 array([ 57800.,  73945.,  90090.,  106235.,  122380.,  138525.,  
  154670.,  170815.,  186960.]),  
<a list of 8 Patch objects>)
```



Histogram using Seabon

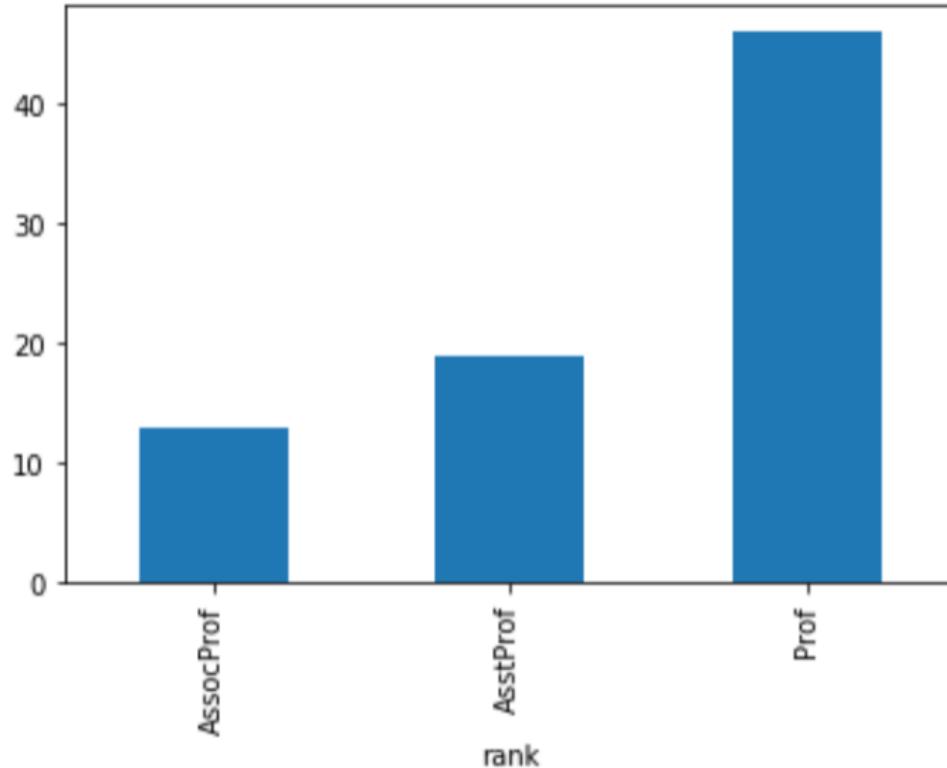
```
In [65]: #Use seaborn package to draw a histogram  
sns.distplot(df['salary']);
```



Barplot using Matplotlib

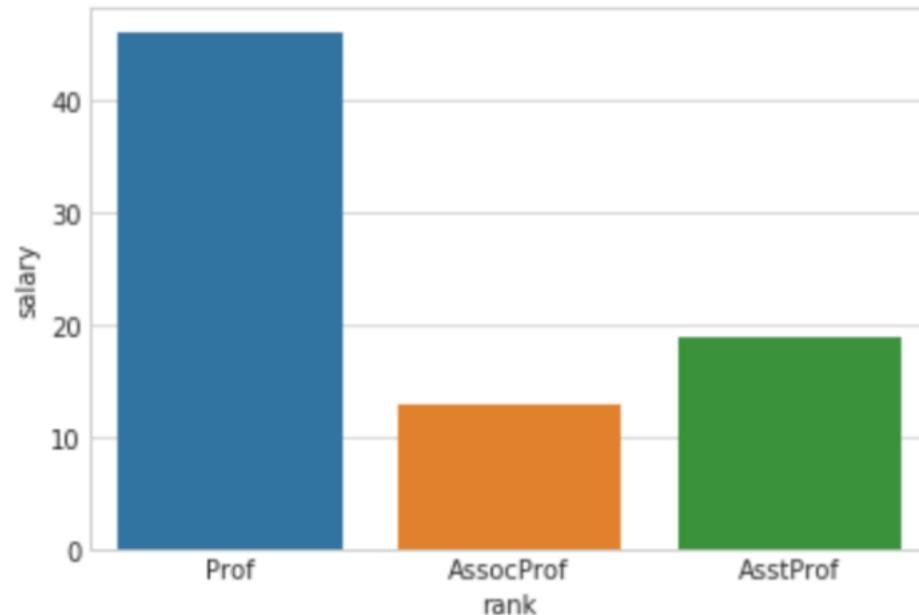
```
In [66]: # Use regular matplotlib function to display a barplot  
df.groupby(['rank'])['salary'].count().plot(kind='bar')
```

```
Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff58213f860>
```



Barplot using Seabon

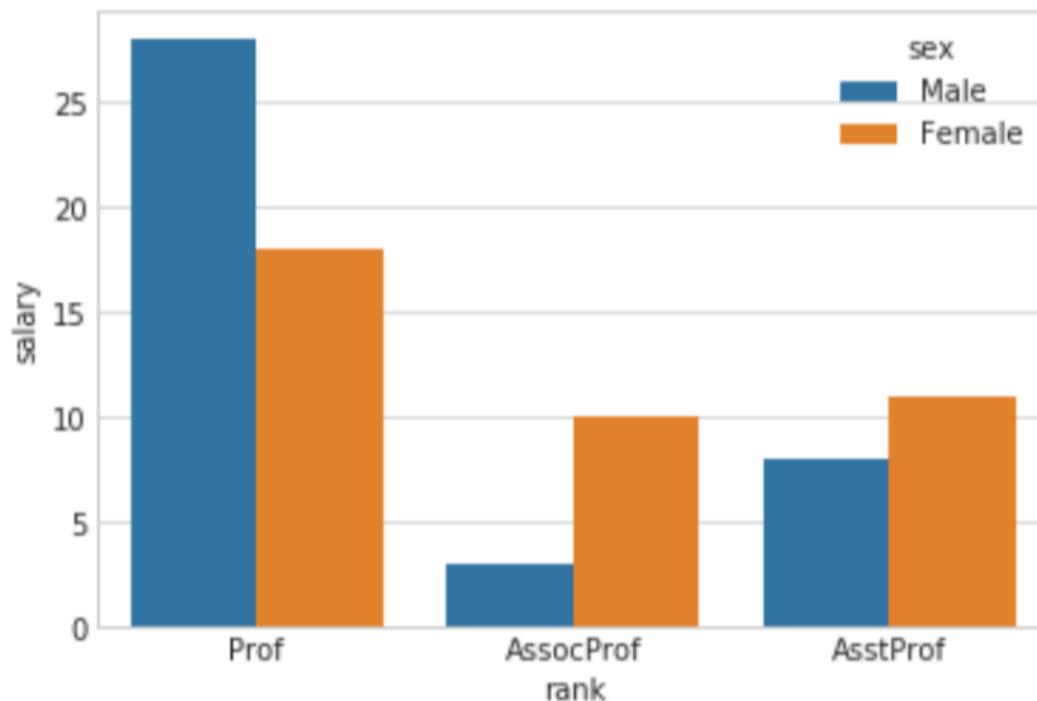
```
In [67]: # Use seaborn package to display a barplot  
sns.set_style("whitegrid")  
  
ax = sns.barplot(x='rank',y ='salary', data=df, estimator=len)
```



Multiple Group Barplot

In [68]: *# Split into 2 groups:*

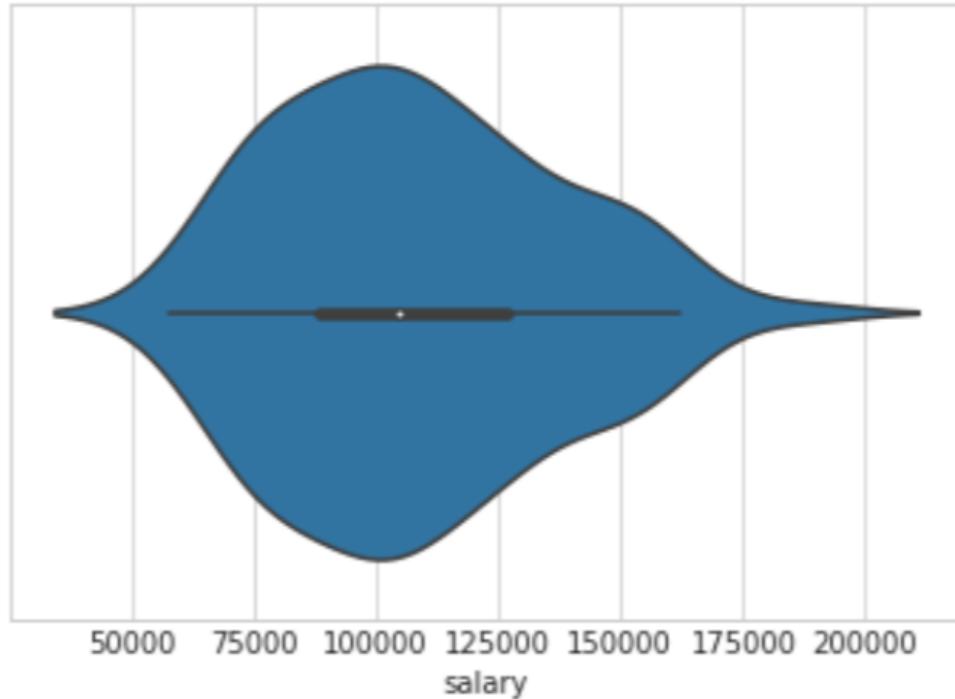
```
ax = sns.barplot(x='rank',y ='salary', hue='sex', data=df, estimator=len)
```



Violin Plot

```
In [69]: #Violinplot  
sns.violinplot(x = "salary", data=df)
```

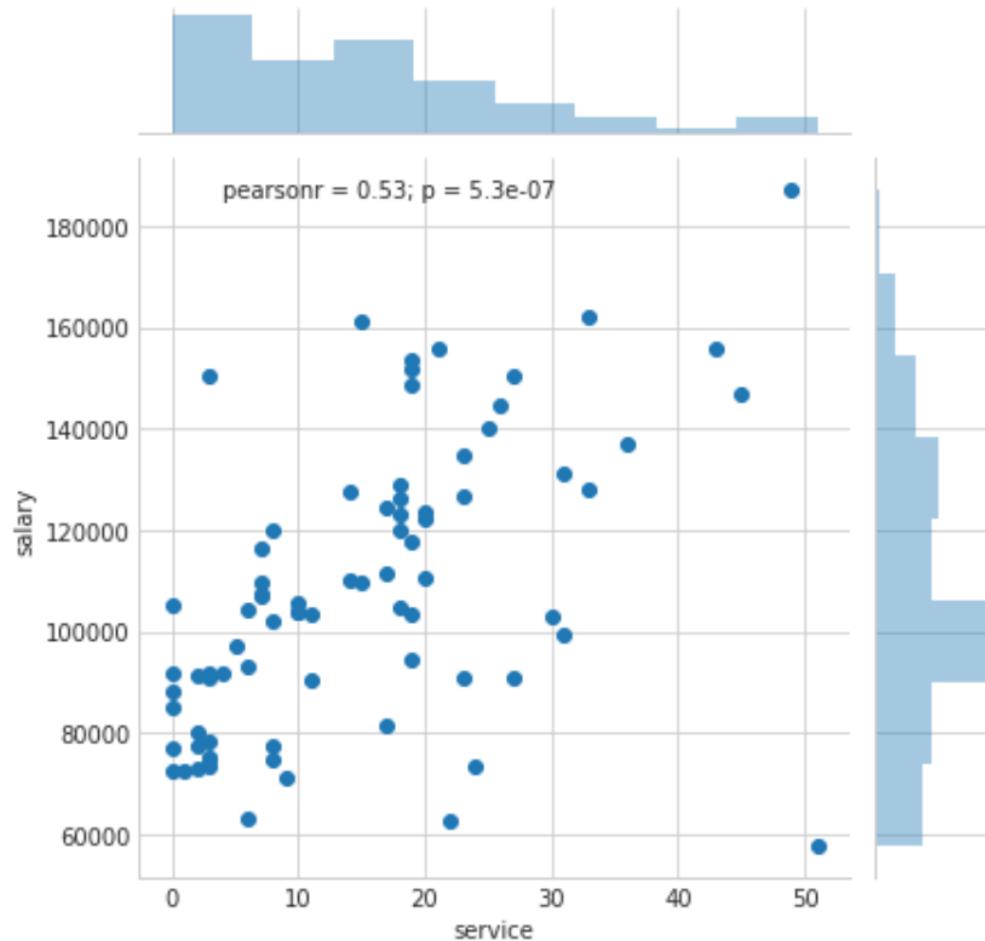
```
Out[69]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5819b79e8>
```



Scatter Plot

```
In [70]: #Scatterplot in seaborn  
sns.jointplot(x='service', y='salary', data=df)
```

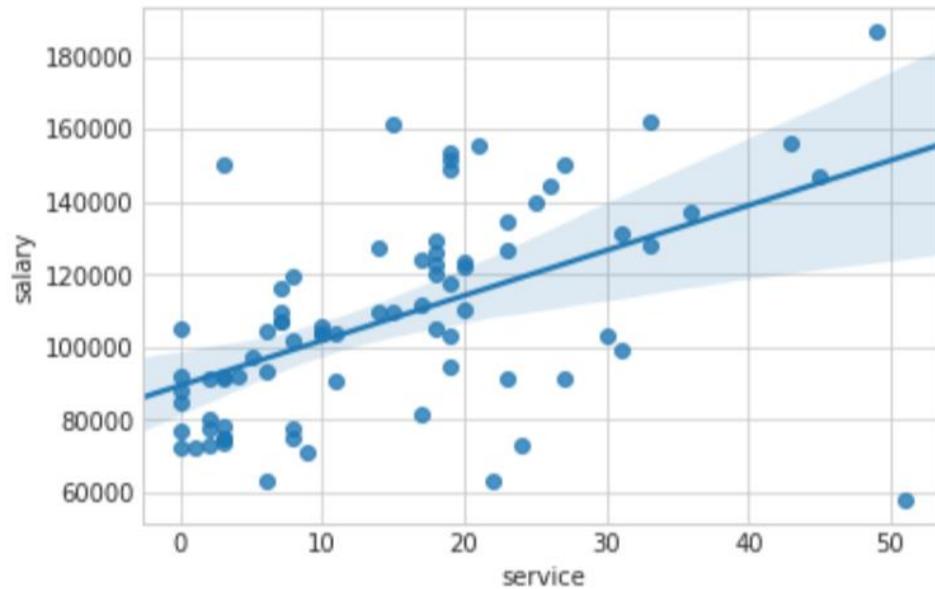
```
Out[70]: <seaborn.axisgrid.JointGrid at 0x7ff581984550>
```



Regression Plot

```
In [71]: #If we are interested in Linear regression plot for 2 numeric variables we can use regplot  
sns.regplot(x='service', y='salary', data=df)
```

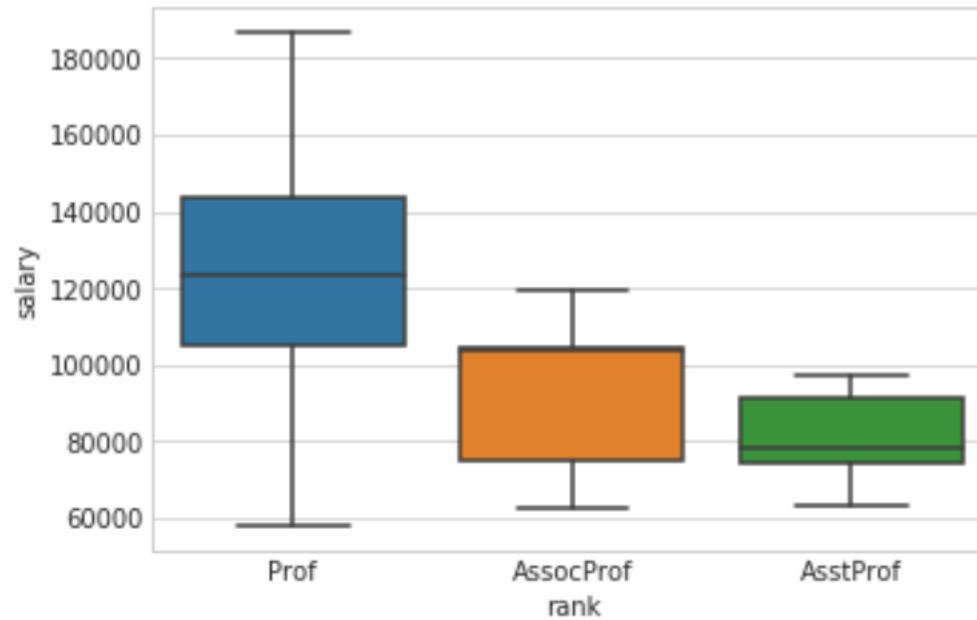
```
Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff58184c470>
```



Box Plot

```
In [72]: # box plot  
sns.boxplot(x='rank',y='salary', data=df)
```

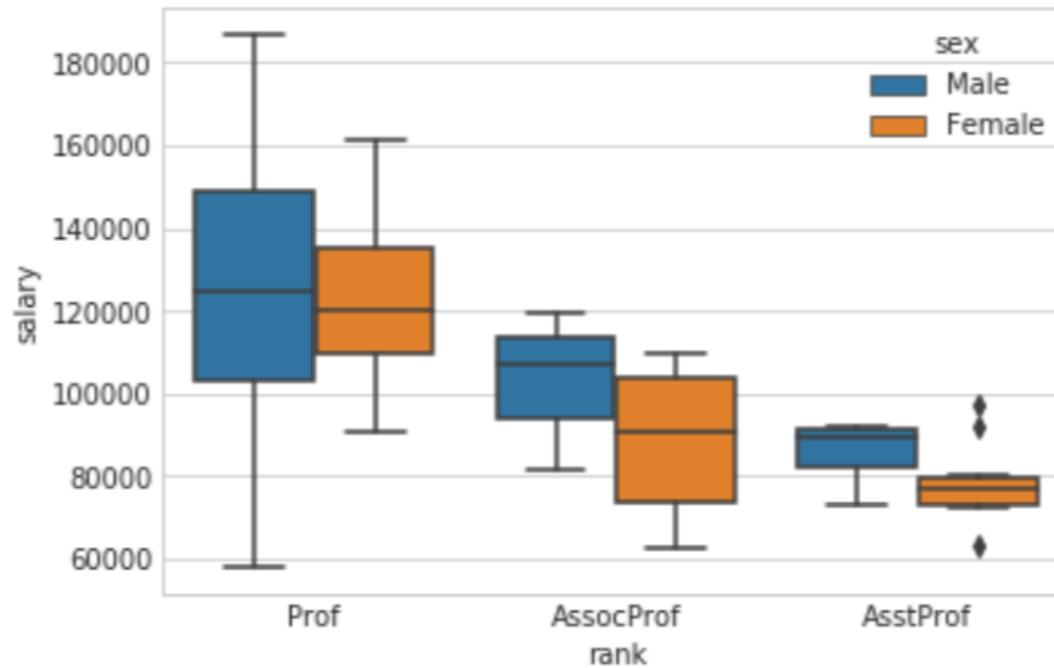
```
Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff58170de80>
```



Group Box Plot

```
In [73]: # side-by-side box plot  
sns.boxplot(x='rank',y='salary', data=df, hue='sex')
```

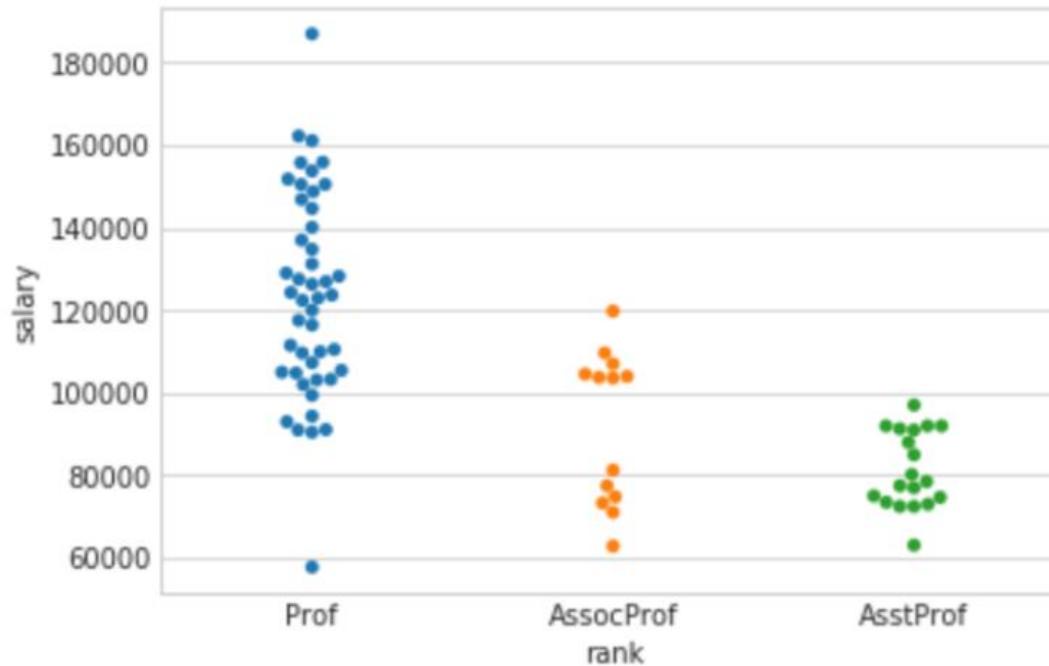
```
Out[73]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5818edc18>
```



Swam Plot

```
In [74]: # swarm plot  
sns.swarmplot(x='rank',y='salary', data=df)
```

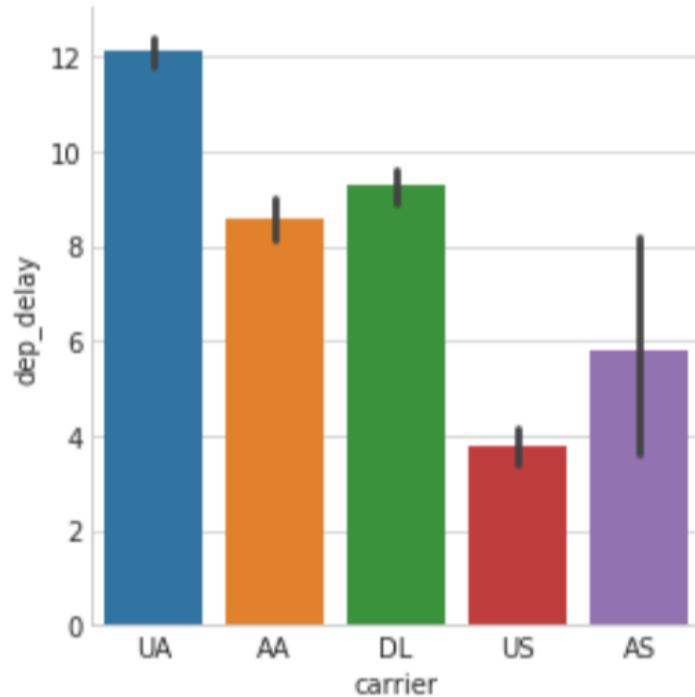
```
Out[74]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff5814f75c0>
```



Factor Plot

```
In [75]: #factorplot  
sns.factorplot(x='carrier',y='dep_delay', data=flights, kind='bar')
```

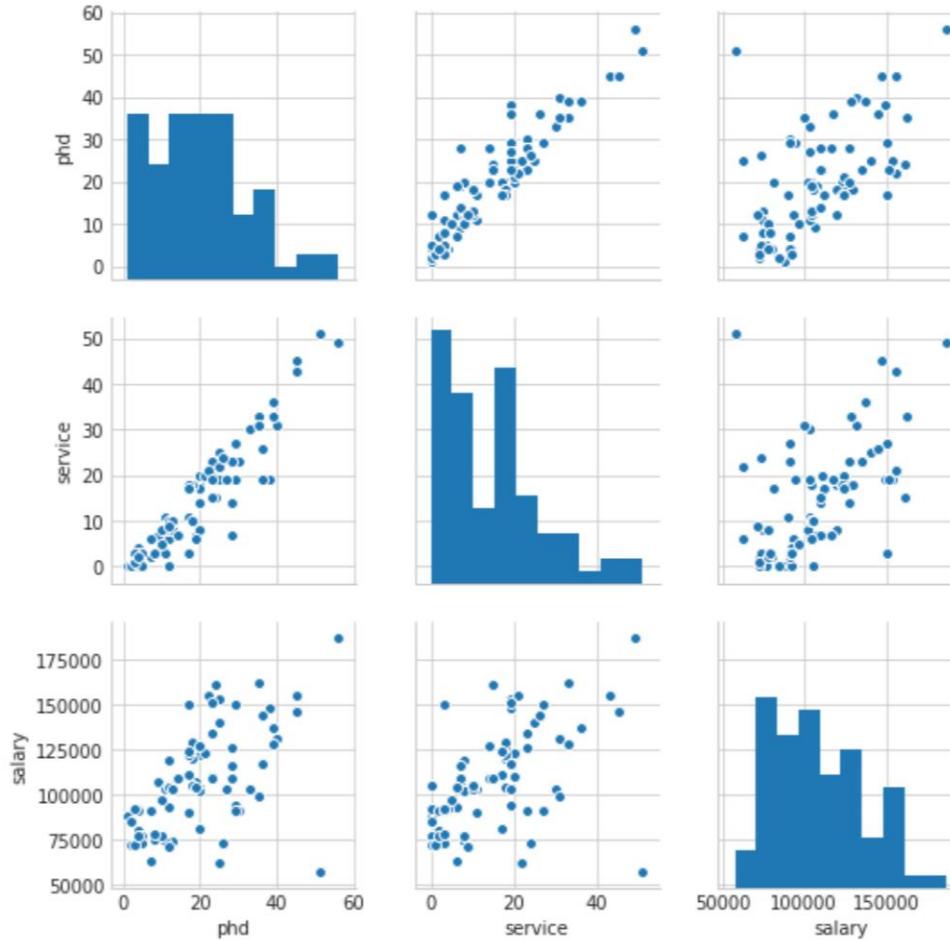
```
Out[75]: <seaborn.axisgrid.FacetGrid at 0x7ff58178a198>
```



Pair Plot

```
In [76]: # Pairplot  
sns.pairplot(df)
```

```
Out[76]: <seaborn.axisgrid.PairGrid at 0x7ff5822296a0>
```



Exercise 9

- ▷ Use flight dataset to practice generating seaborn graph for all variables.
- ▷ After than, explore the dependency of arrival delay(`arr_delay`) on departure delay (`dep_delay`) using scatterplot or regplot.
- ▷ Create Heatmap chart between `carrier` and `origin`.

Inferential Statistics

Descriptive vs. Inferential

- ▷ **Descriptive:** e.g., Mean; describes data you have but can't be generalized beyond that
 - We'll talk about Exploratory Data Analysis
- ▷ **Inferential:** e.g., t-test, that enable inferences about the population beyond our data
 - These are the techniques we'll leverage for Machine Learning and Prediction

Examples of Business Questions

- ▷ Simple (descriptive) Stats
 - “Who are the most profitable customers?”
- ▷ Hypothesis Testing
 - “Is there a difference in value to the company of these customers?”
- ▷ Segmentation/Classification
 - What are the common characteristics of these customers?
- ▷ Prediction
 - Will this new customer become a profitable customer? If so, how profitable?

Basic statistical Analysis

statsmodel and **scikit-learn** - both have a number of function for statistical analysis

The first one is mostly used for regular analysis using R style formulas, while scikit-learn is more tailored for Machine Learning.

statsmodels:

- linear regressions
- ANOVA tests
- hypothesis testings
- many more ...

scikit-learn:

- kmeans
- support vector machines
- random forests
- many more ...

Hypothesis Testing

- ▷ Compare an experimental group and a control group.
- ▷ **H_0 : Null Hypothesis**
 - No difference between the groups
- ▷ **H_A : Alternative Hypothesis**
 - Statistically significant difference between the groups
- ▷ “difference” defined in the terms of some **test statistic**
 - **Different means** (e.g. t-test), **Different variances** (e.g. F-test)
- ▷ Group defined through careful experimental design
 - Randomized, blinded, double-blinded
- ▷ Examples
 - “The new ad placement produces more click-throughs”
 - “This treatment produces better outcomes”

More on Hypothesis Testing

- ▷ **Null Hypothesis** is given the benefit of the doubt (e.g., innocent until proven guilty).
- ▷ **Alternative Hypothesis** directly contradicts the Null Hypothesis
- ▷ "Step 1: State the hypotheses."
- ▷ "Step 2: Set the criteria for a decision."
- ▷ "Step 3: Compute the test statistic."
- ▷ "Step 4: Make a decision."

p Value

- ▷ A **p value** is the probability of obtaining a sample outcome, given that the value stated in the null hypothesis is true.
- ▷ In many cases: when the p value is less than 5% ($p < .05$), we reject the null hypothesis
 - Note this means that 1 out of 20 times we incorrectly reject the null hypothesis

Reason to Reject H_0

Illustration: Let say, we **assume** that average age in the US is 50 years ($H_0=50$). **If in fact this is the true (unknown) population mean, it is unlikely that we get a sample mean of 20.** So, if we have a sample that produces an average of 20, then we **reject** that the null hypothesis that average age is 50. (note that we are rejecting our assumption or claim). (would we get 20 if the true population mean was 50? NO. That is why we reject 50)

How Is the Test done?

We use the distribution of a Test Statistic, such as Z or t as the criteria.

A. Rejection Region Method:

- Divide the distribution into rejection and non-rejection regions
- Defines the unlikely values of the sample statistic if the null hypothesis is true, the critical value(s)
- Defines **rejection region** of the sampling distribution
Rejection region(s) is designated by α , (level of significance)

Typical values are .01, .05, or .10

- is selected by the researcher at the beginning
- provides the critical value(s) of the test

Non-rejection region

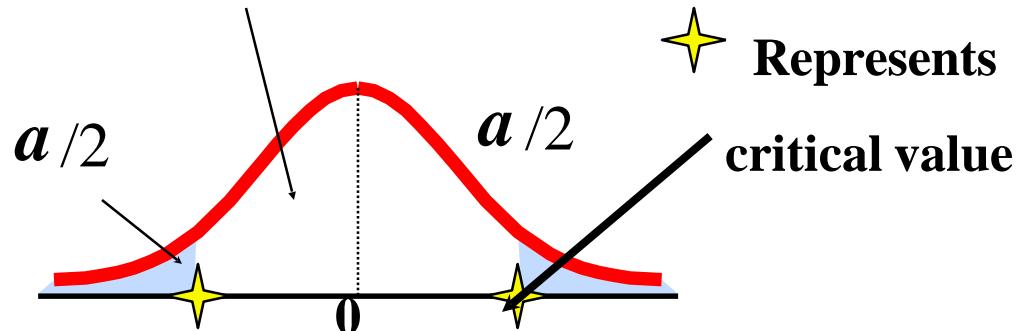
Rejection Region or Critical Value Approach:

Level of significance = α

$$H_0: \mu = 12$$

$$H_1: \mu \neq 12$$

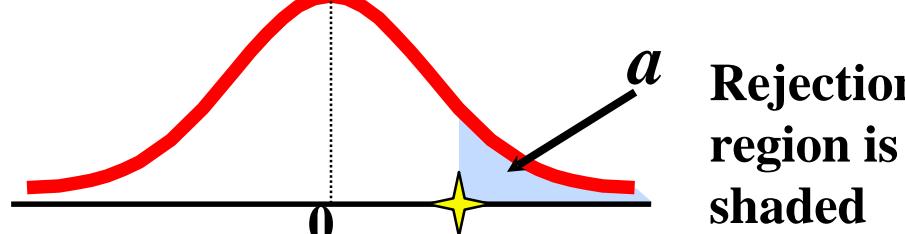
Two-tail test



$$H_0: \mu \leq 12 \quad H_1:$$

$$\mu > 12$$

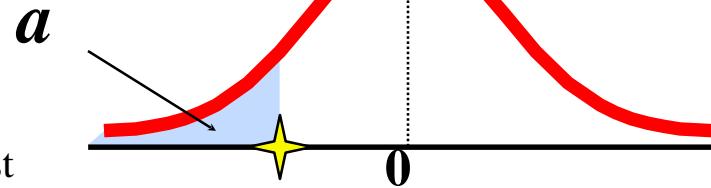
Upper-tail test



$$H_0: \mu \geq 12$$

$$H_1: \mu < 12$$

Lower-tail test



P-Value Approach –

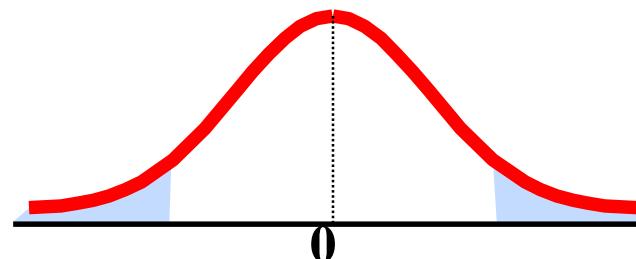
P-value=Max. Probability of (Type I Error), calculated from the sample.

Given the sample information what is the size of blue area?

$$H_0: \mu = 12$$

$$H_1: \mu \neq 12$$

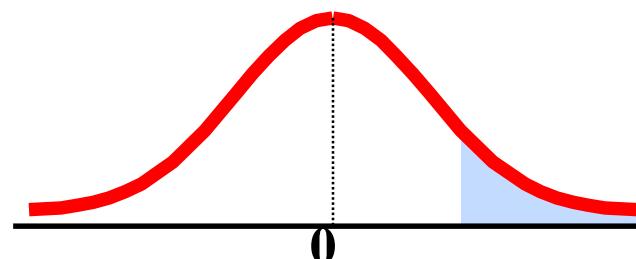
Two-tail test



$$H_0: \mu \leq 12 \quad H_1:$$

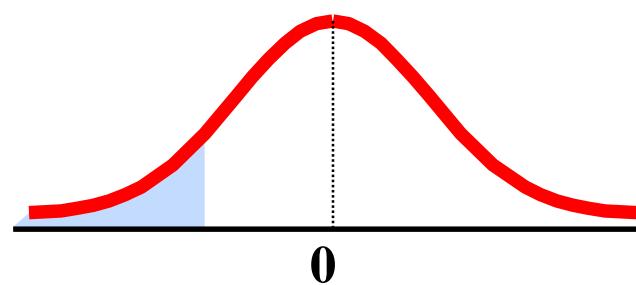
$$\mu > 12$$

Upper-tail test



$$H_0: \mu \geq 12$$

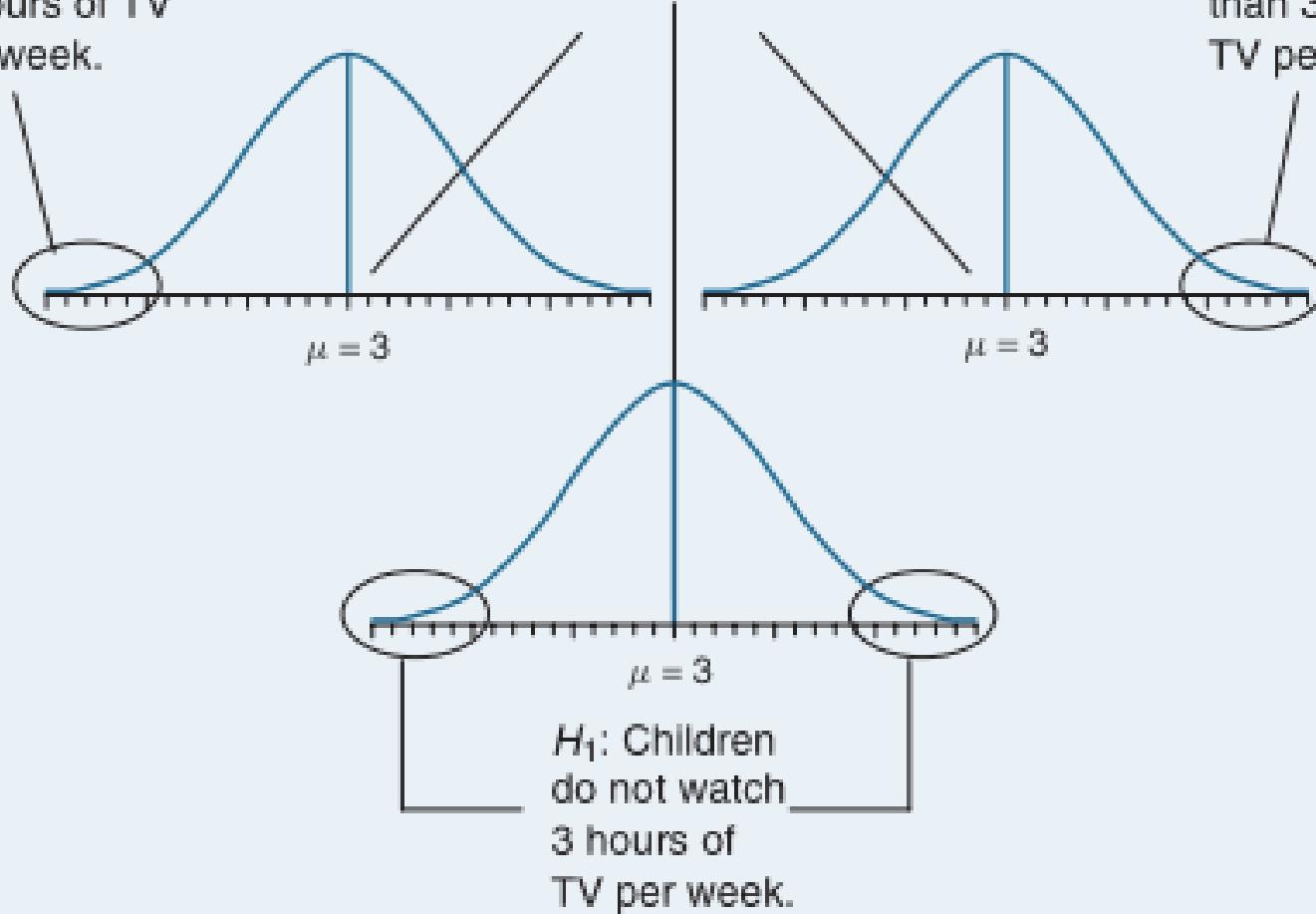
$$H_1: \mu < 12$$



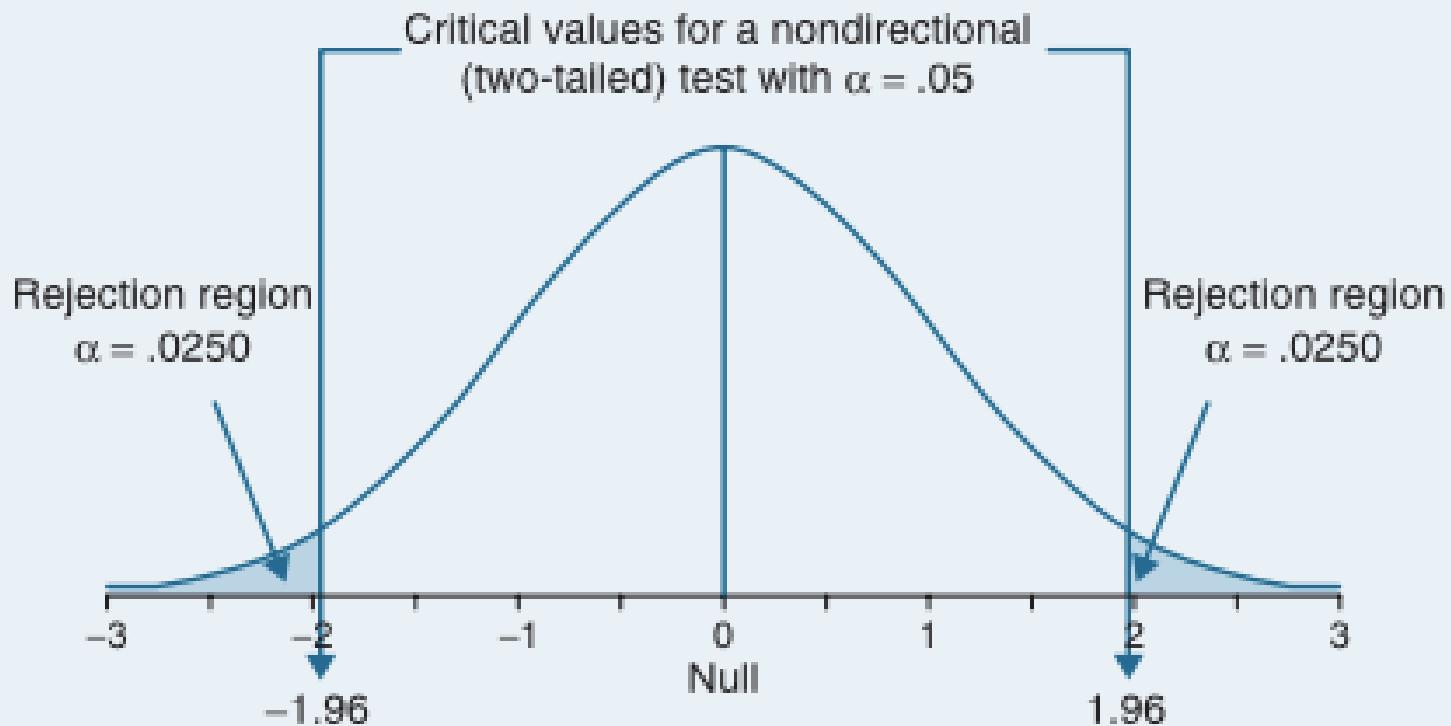
H_1 : Children
watch less than
3 hours of TV
per week.

We expect the
sample mean to be
equal to the
population mean.

H_1 : Children
watch more
than 3 hours of
TV per week.



Two-tailed Significance



When the p value is less than 5% ($p < .05$), we reject the null hypothesis

Hypothesis Testing

		Decision	
		Retain the null	Reject the null
Truth in the population	True	CORRECT $1 - \alpha$	TYPE I ERROR α
	False	TYPE II ERROR β	CORRECT $1 - \beta$ POWER

From G.J. Primavera, "Statistics for the Behavioral Sciences"

Hypothesis Testing in Python

Student t-test

For comparing means between 2 group, we can use ttest_ind from Scipy's stats package.

```
In [83]: # Using scipy package:  
from scipy import stats  
df_w = df[ df['sex'] == 'Female']['salary']  
df_m = df[ df['sex'] == 'Male']['salary']  
stats.ttest_ind(df_w, df_m)  
  
Out[83]: Ttest_indResult(statistic=-2.2486865976699053, pvalue=0.027429778657910103)
```

ANOVA (Analysis of Variance)

For comparing Means more than 2 group, we can use stats package from Scipy

```
stats.f_oneway(data_group1, data_group2, data_group3, data_groupN)
```

```
In [5]: from scipy import stats
stats.f_oneway(df['salary'][df['rank'] == 'Prof'],
                df['salary'][df['rank'] == 'AsstProf'],
                df['salary'][df['rank'] == 'AssocProf'])
```

```
Out[5]: F_onewayResult(statistic=31.47638426157928, pvalue=1.1882466669224665e-10)
```

Assumption

- ▷ There are 3 assumptions that need to be met for the results of an ANOVA test to be considered accurate. They are:
- ▷ Normality
 - Caveat to this is, if group sizes are equal, the F-statistic is robust to violations of normality
- ▷ Homogeneity of variance
 - Same caveat as above, if group sizes are equal, the F-statistic is robust to this violation
- ▷ Independent observations

Assumption of Normality

▷ Using Shapiro-Wilk test

```
In [6]: stats.shapiro(df['salary'][df['rank'] == 'Prof'])
```

```
Out[6]: (0.9807326793670654, 0.636059582233429)
```

```
In [7]: stats.shapiro(df['salary'][df['rank'] == 'AssocProf'])
```

```
Out[7]: (0.8890405893325806, 0.09464861452579498)
```

```
In [8]: stats.shapiro(df['salary'][df['rank'] == 'AsstProf'])
```

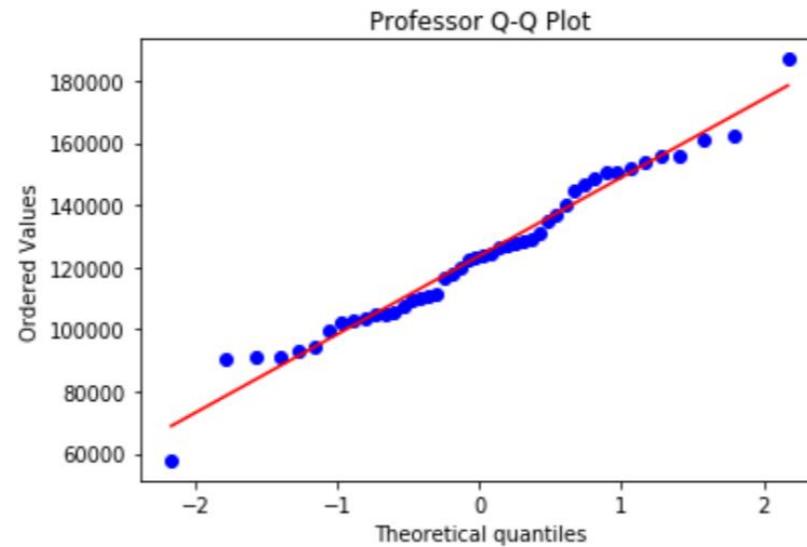
```
Out[8]: (0.9256936311721802, 0.14421962201595306)
```

Assumption of Normality

▷ Using Normal QQ-Plot

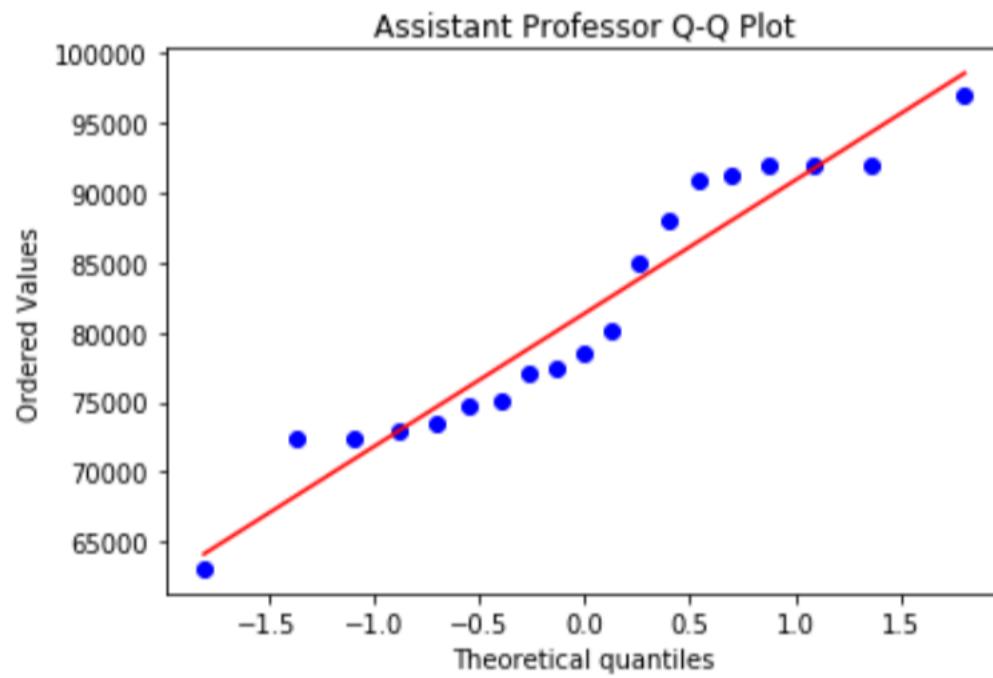
```
In [9]: stats.probplot(df['salary'][df['rank'] == 'Prof'], plot= plt)  
plt.title("Professor Q-Q Plot")
```

```
Out[9]: Text(0.5,1,'Professor Q-Q Plot')
```



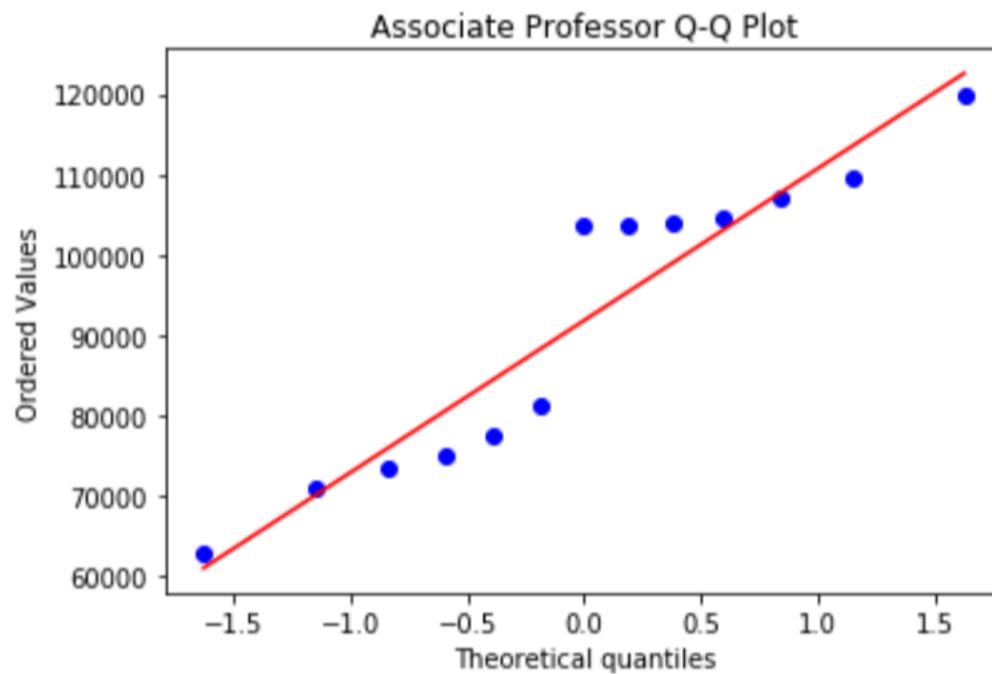
```
In [10]: stats.probplot(df['salary'][df['rank'] == 'AsstProf'], plot= plt)  
plt.title("Assistant Professor Q-Q Plot")
```

```
Out[10]: Text(0.5,1,'Assistant Professor Q-Q Plot')
```



```
In [11]: stats.probplot(df['salary'][df['rank'] == 'AssocProf'], plot= plt)
plt.title("Associate Professor Q-Q Plot")
```

```
Out[11]: Text(0.5,1,'Associate Professor Q-Q Plot')
```



Assumption of Homogeneity of Variance

- ▷ We can use Levene's Test for this

```
In [12]: stats.levene(df['salary'][df['rank'] == 'Prof'],
                     df['salary'][df['rank'] == 'AssocProf'],
                     df['salary'][df['rank'] == 'AsstProf'])
```

```
Out[12]: LeveneResult(statistic=5.455185896137257, pvalue=0.006138855806188982)
```

Exercise 10

Use flights dataset, find out whether
they are differences between mean
of departure delay between carriers?

Linear Regression

Regression Models

- ▷ Relationship between one **dependent variable** and **explanatory variable(s)**
- ▷ Use equation to set up relationship
 - Numerical Dependent (Response) Variable
 - 1 or More Numerical or Categorical Independent (Explanatory) Variables
- ▷ Used Mainly for Prediction & Estimation

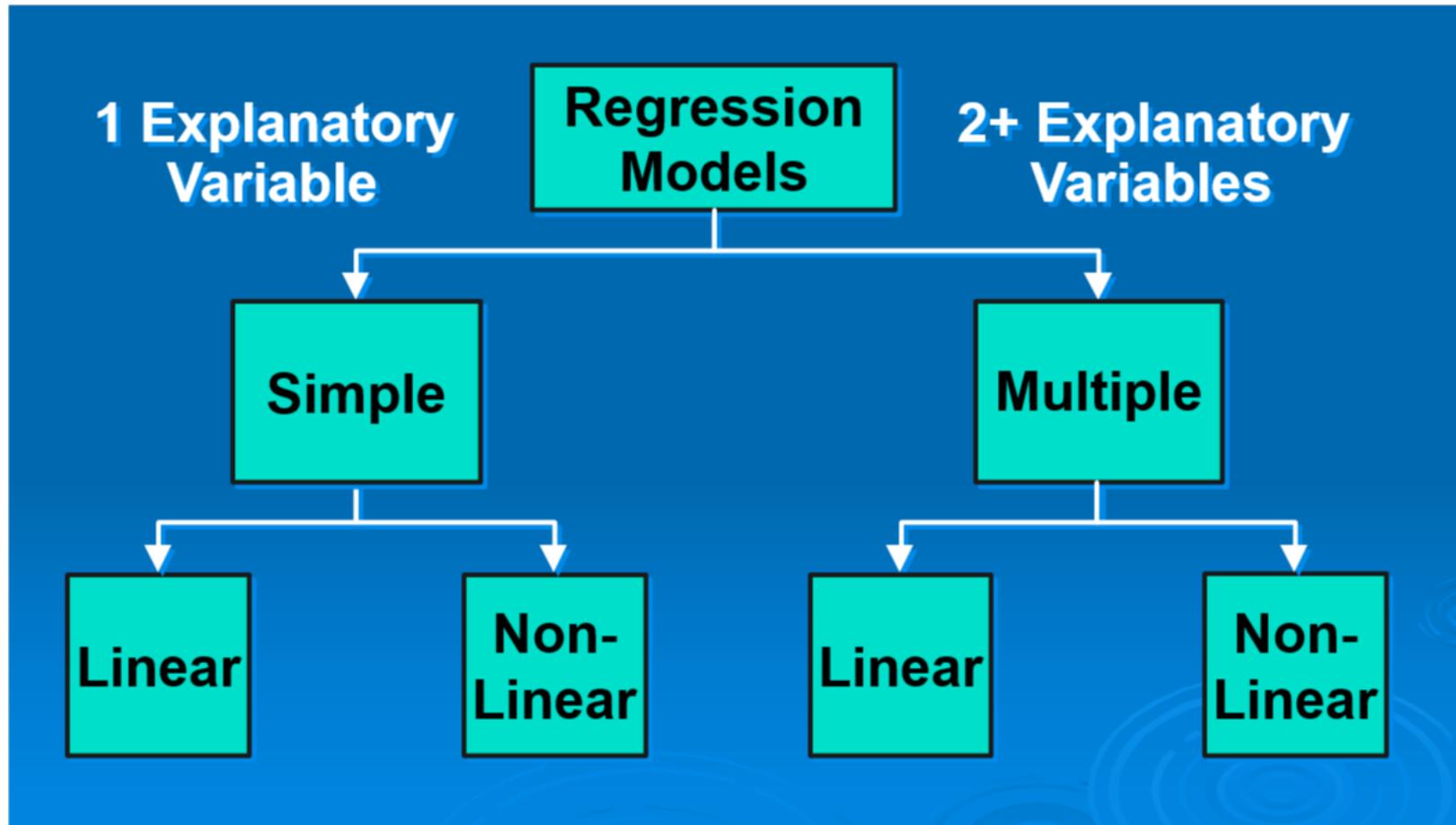
Regression Modeling Steps

1. Hypothesize Deterministic Component
 - Estimate Unknown Parameters
2. Specify Probability Distribution of Random Error Term
 - Estimate Standard Deviation of Error
3. Evaluate the fitted Model
4. Use Model for Prediction & Estimation

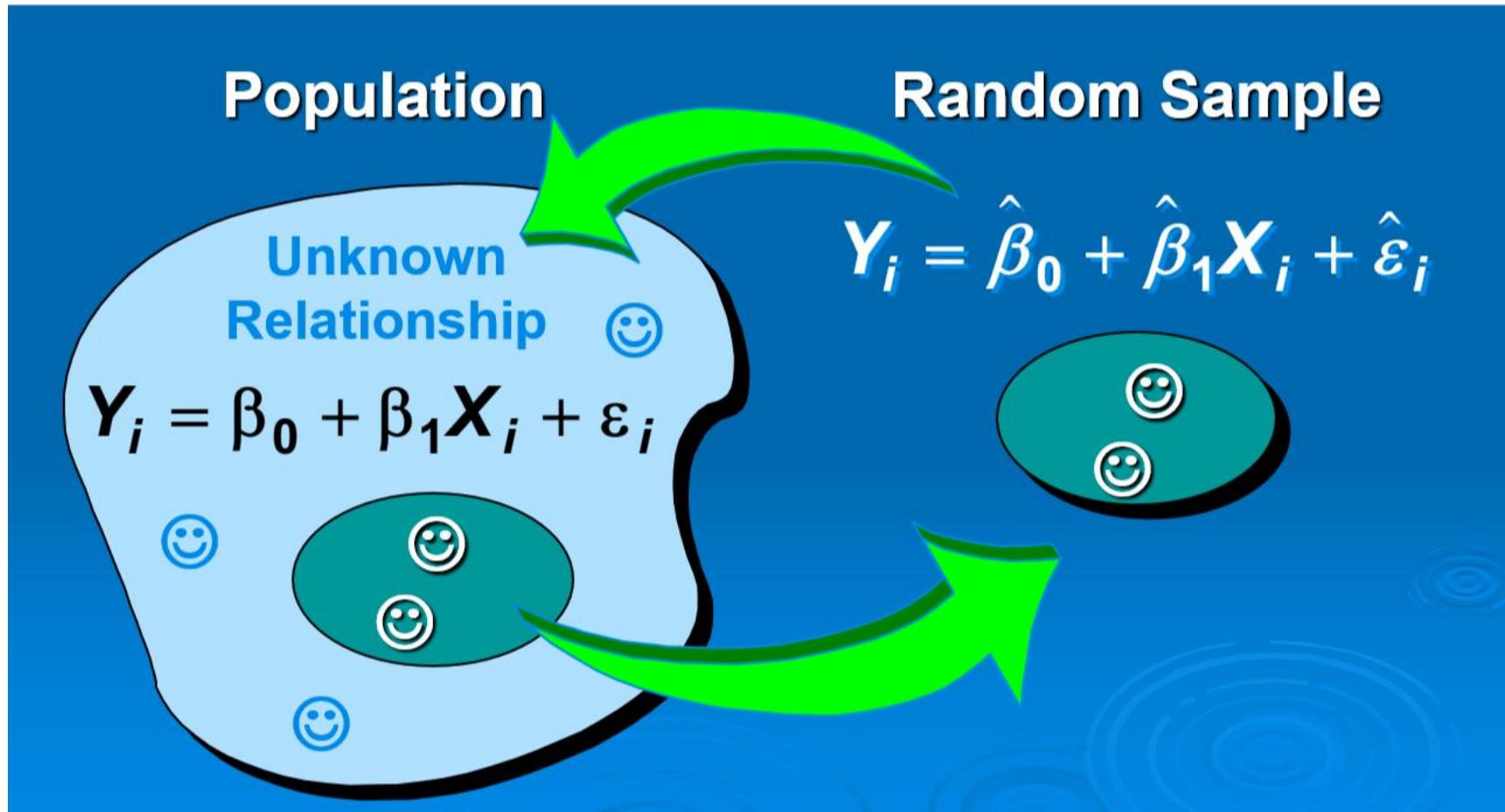
Specifying the deterministic component

- ▷ 1. Define the dependent variable and independent variable
- ▷ 2. Hypothesize Nature of Relationship
 - Expected Effects (i.e., Coefficients' Signs)
 - Functional Form (Linear or Non-Linear)
 - Interactions

Types of Regression Models



Population & Sample Regression Models



Sample Linear Regression Model

- The model to be estimated from sample data is:

$$Y_i = b_0 + b_1 X_i + e_i$$

Residual
(random
error from
the sample)

- The actual estimated from the sample

Estimated (or
predicted) Y
value for
observation i

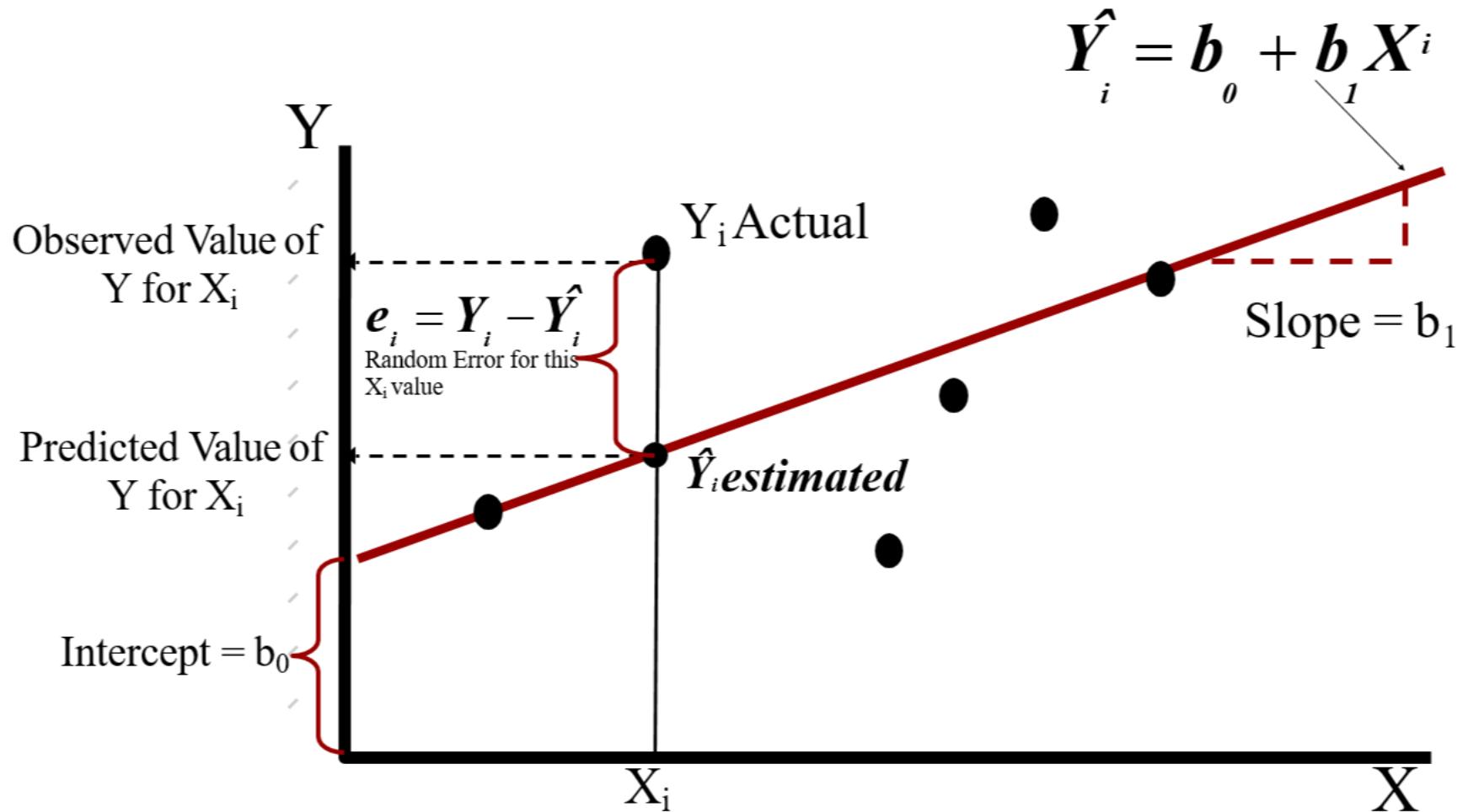
Estimate of the
regression
intercept

Estimate of the
regression slope

$$\hat{Y}_i = b_0 + b_1 X_i$$

– Where $e_i = Y_i - \hat{Y}_i$

Value of X for
observation i



Linear Regression using Python

Question

Using linear regression to create
model that have relationship
between services year and salary.

```
In [78]: # Import Statsmodel functions:  
import statsmodels.formula.api as smf
```

```
In [79]: # create a fitted model  
lm = smf.ols(formula='salary ~ service', data=df).fit()  
  
#print model summary  
print(lm.summary())
```

```
OLS Regression Results  
=====  
Dep. Variable: salary R-squared: 0.283  
Model: OLS Adj. R-squared: 0.274  
Method: Least Squares F-statistic: 30.03  
Date: Fri, 15 Sep 2017 Prob (F-statistic): 5.31e-07  
Time: 14:11:46 Log-Likelihood: -896.72  
No. Observations: 78 AIC: 1797.  
Df Residuals: 76 BIC: 1802.  
Df Model: 1  
Covariance Type: nonrobust  
=====  
coef std err t P>|t| [0.025 0.975]  
-----  
Intercept 8.935e+04 4365.651 20.468 0.000 8.07e+04 9.8e+04  
service 1240.3567 226.341 5.480 0.000 789.560 1691.153  
=====  
Omnibus: 12.741 Durbin-Watson: 1.630  
Prob(Omnibus): 0.002 Jarque-Bera (JB): 21.944  
Skew: -0.576 Prob(JB): 1.72e-05  
Kurtosis: 5.329 Cond. No. 30.9  
=====
```

```
In [80]: # print the coefficients  
lm.params
```

```
Out[80]: Intercept    89354.824215  
         service      1240.356654  
         dtype: float64
```

```
In [81]: #using scikit-learn:  
from sklearn import linear_model  
est = linear_model.LinearRegression(fit_intercept = True)    # create estimator object  
est.fit(df[['service']], df[['salary']])  
  
#print result  
print("Coef:", est.coef_, "\nIntercept:", est.intercept_)
```

```
Coef: [[ 1240.3566535]]  
Intercept: [ 89354.82421525]
```

Workshop 11

- ▷ Use flight dataset, find linear regression model for
 - Independent variable: Departure Delay (dep_delay)
 - Dependent variable : Arrival Delay (arr_delay)
- ▷ Explain the model

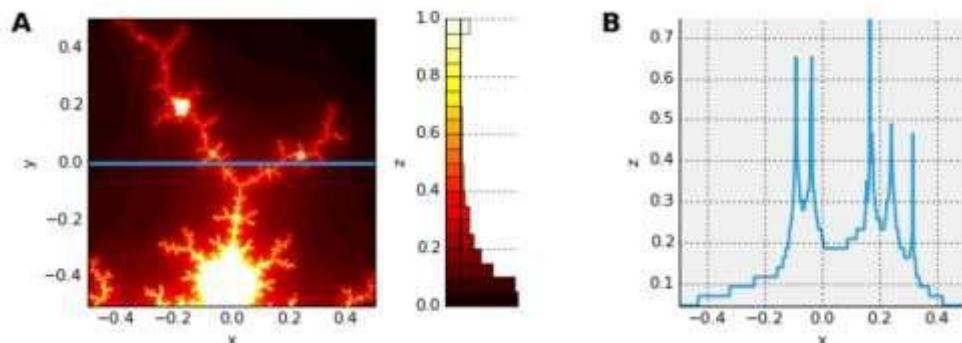
Data Visualization and Applications made easy with Holoviews

HoloViews: Stop plotting your data

HoloViews makes it simple to create beautiful interactive Bokeh or Matplotlib visualizations of complex data.

```
import numpy as np
import holoviews as hv
hv.notebook_extension('matplotlib')
fractal = hv.Image(np.load('mandelbrot.npy'))

((fractal * hv.HLine(y=0)).hist() + fractal.sample(y=0))
```



- Exploring data can be tedious if you use a plotting library directly, because you will need to specify details about your data (units, dimensions, names, etc.) every time you construct a new type of plot. With HoloViews, you instead annotate your data once, and then flexible plotting comes for free —
- HoloViews objects just display themselves, alone or in any combination.

It's now easy to lay out subfigures, overlay traces, facet or animate a multidimensional dataset, sample or aggregate to reduce dimensionality, preserving the metadata each time so the results visualize themselves.

NYC Taxi trips, with [Datashader](#), [HoloViews](#), [GeoViews](#) and [ParamNB](#)

https://anaconda.org/jbednar/nyc_taxi-paramnb/notebook

Data Widgets and Applications from Jupyter Notebooks!

```
tiles = gv.WMTSTileSource(url='https://server.arcgisonline.com/ArcGIS/rest/services/World_Imagery/MapServer/tile/{z}/{y}/{x}')
tile_options = dict(width=800, height=475, xaxis=None, yaxis=None, bgcolor='black')
passenger_counts = sorted(df.passenger_count.unique().tolist())

class Options(hv.streams.Stream):
    alpha      = param.Magnitude(default=0.75, doc="Alpha value for transparency")
    colormap   = param.ObjectSelector(default=cm["fire"], objects=cm.all)
    plot       = param.ObjectSelector(default="pickup", objects=["pickup", "dropoff"])
    passengers = param.ObjectSelector(default=1, objects=passenger_counts)

    def make_plot(self, x_range=None, y_range=None, **kwargs):
        map_tiles = tiles(style=dict(alpha=self.alpha), plot=tile_options)
        df_filt = df[df.passenger_count==self.passengers]
        points = hv.Points(gv.Dataset(df_filt, kdims=[self.plot+'_'+x,
                                                       self.plot+'_'+y]), vdims='count',
                            cmap=self.colormap, element_type=hv.Image)
        map_tiles * points

    return map_tiles * taxi_trips

selector = Options(name="")
paramnb.Widgets(selector, callback=selector.update)
hv.DynamicMap(selector.make_plot, kdims=[], streams=[selector, RangeXY])
```

alpha	<input type="range" value="0.30"/>
colormap	linear_kryw_0_100_c71
passengers	1
plot	dropoff



Anaconda and Hadoop

High Performance Hadoop

Bottom Line

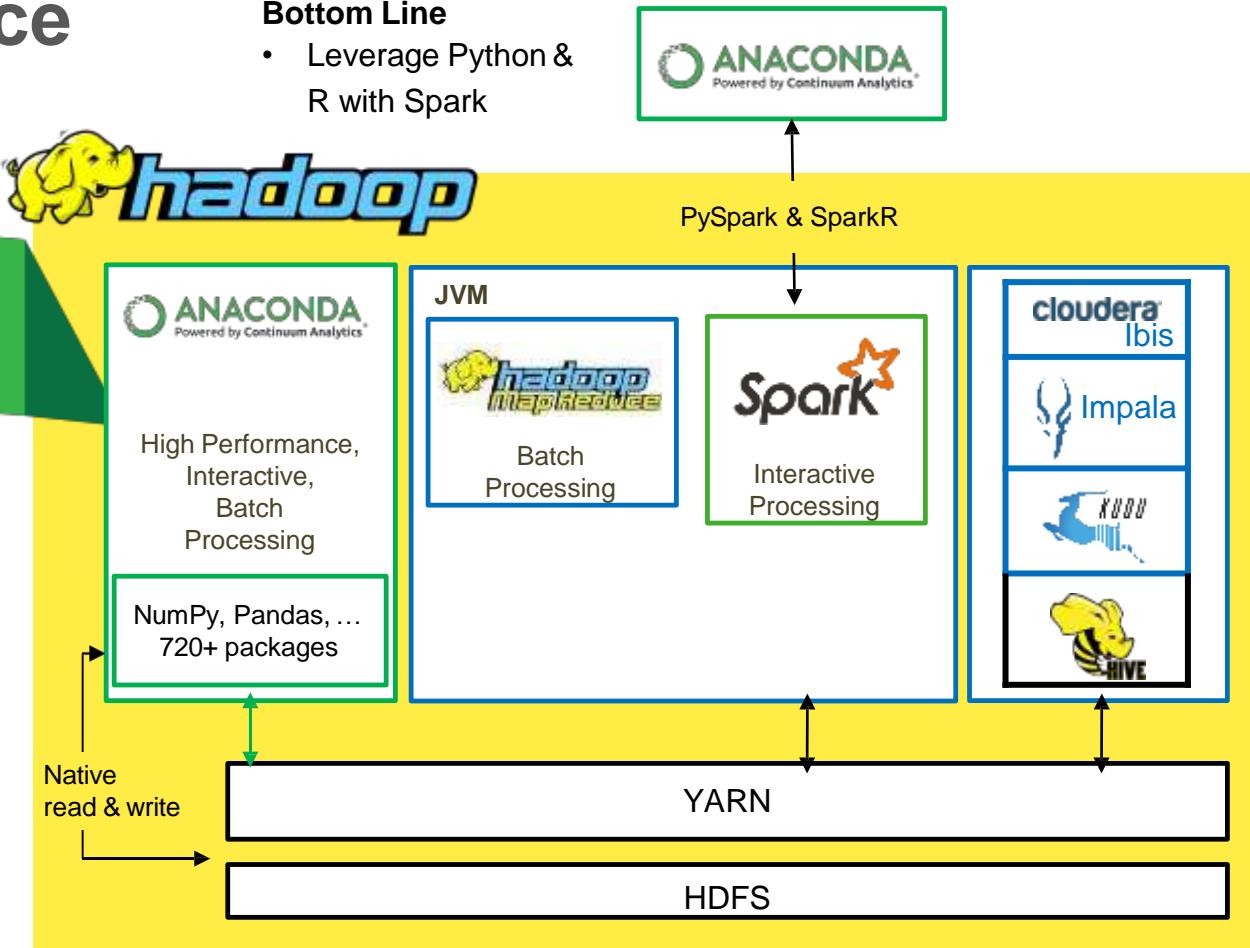
2X-100X faster overall performance

- Interact with data in HDFS and Amazon S3 natively from Python
- Distributed computations without the JVM & Python/Java serialization
- Framework for easy, flexible parallelism using directed acyclic graphs (DAGs)
- Interactive, distributed computing with in-memory persistence/caching

Python & R ecosystem
MPI

Bottom Line

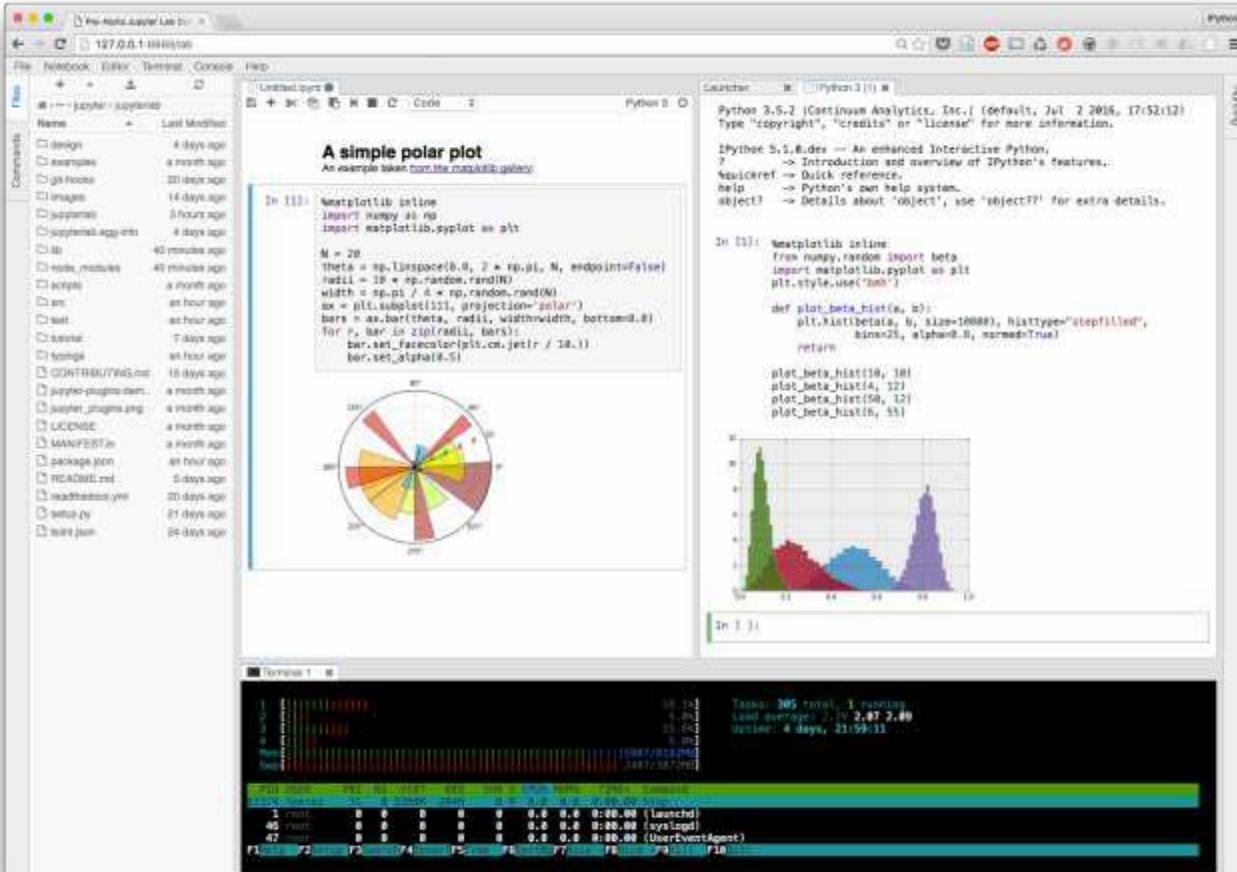
- Leverage Python & R with Spark



Jupyterlab (Where Jupyter is heading)

JupyterLab

- IDE
 - Extensible
 - Notebook -> Applications



Building Blocks

File Browser

Notebooks

Text Editor

Widgets

Output

Terminal

JupyterLab

- JupyterLab is the natural evolution of the Jupyter Notebook user interface
- JupyterLab is an extensible IDE: *Interactive Data Environment*
- Flexible user interface for assembling the fundamental building blocks of interactive computing
- Modernized JavaScript architecture based on npm/webpack, plugin system, model/view separation
- Built using PhosphorJS (<http://phosphorjs.github.io/>)
- Design-driven development <https://github.com/jupyterlab/jupyterlab>

Thanks!

Any questions?

You can find me at:

@veerasakk

Verasak.kritsanaphan@gmail.com