GENERAL ASSEMBLY

INTRODUCTION TO PYTHON PROGRAMMING


LET'S MEET
GA

ABOUT YOUR PRODUCER!

Chris Wright

chris@generalassemb.ly

Classes & Workshops Lead

ABOUT YOUR INSTRUCTOR!

Ruben Naeff

rubennaeff@gmail.com

Data Science Instructor

DATA SCIENTIST
AT KNEWTON

MUSIC COMPOSER
STRATEGY CONSULTANT
ECONOMIC RESEARCHER
MATH TEACHER

AMSTERDAM, NL
BROOKLYN, NY

ABOUT YOU!

Who
are you

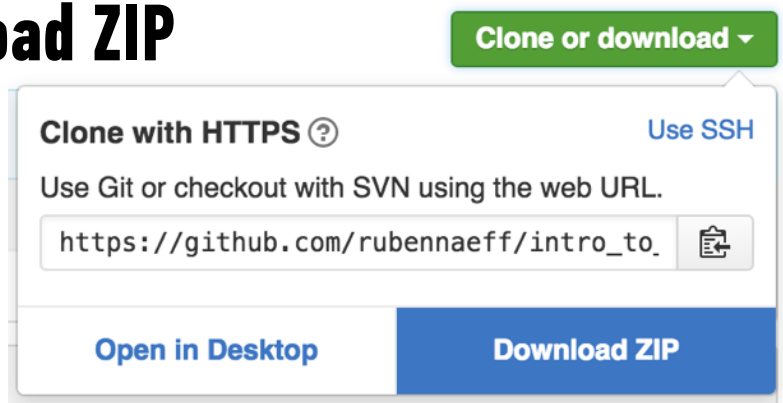
What
do you do

Why
are you here

LET'S GET SET UP

DOWNLOAD REPO TO MACHINE

- Go to **github.com/rubennaeff/intro_to_python**
- Click **Clone or download | Download ZIP**



OR

- git clone https://github.com/rubennaeff/intro_to_python

EVERYONE ALL SET WITH THE INSTALLATION?

```
python
>>> _
```

0. MEET, SETUP, TROUBLESHOOT – DONE!
I. WHY PYTHON?
II. PYTHON SYNTAX
III. WRITING A SCRIPT
IV. LIBRARIES
V. EXAMPLES

- PLAYING AROUND IN THE PYTHON SHELL
- WRITING, SAVING AND IMPORTING PYTHON SCRIPTS
- PLUGGING INTO THE WEALTH OF PYTHON LIBRARIES
- CREATING IPYTHON NOTEBOOKS
- DEVELOPING AN INTERACTIVE WEBSITE IN FLASK

THIS IS TOO MUCH FOR 3 HOURS, BUT ALL CODE IS INCLUDED.

- LEARN HOW TO GET SET UP, INSTALL PYTHON AND LIBRARIES
- FIND OUT HOW TO WRITE YOUR FIRST PYTHON SCRIPT
- FIGURE OUT WHAT TOOLS TO USE FOR YOUR PYTHON PROJECTS

- AS AN ABSOLUTE BEGINNER, GET UP TO SPEED
- EXPLAIN SUPERFICIALLY WHY PYTHON IS DIFFERENT
- BE COMFORTABLE WITH THE PYTHON SHELL
- BE ABLE TO WRITE, SAVE, AND RUN SCRIPTS
- BE ABLE TO INSTALL AND IMPORT PACKAGES
- KNOW WHERE TO LOOK FOR FURTHER STUDY

PYTHON

- Created by Guido van Rossum in 1991
- Benevolent Dictator for Life

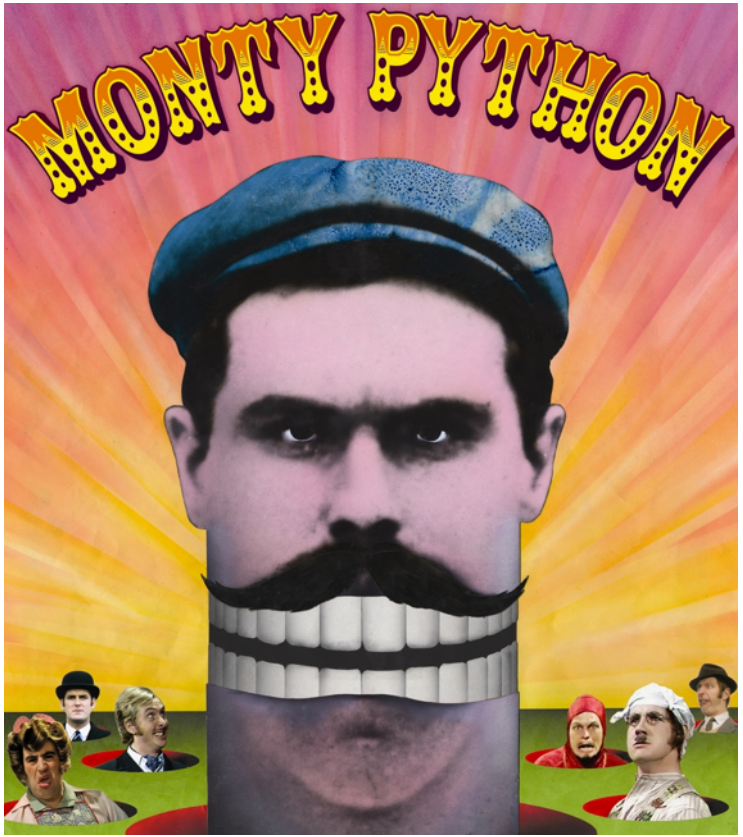
- Created by Guido van Rossum in 1991
- Benevolent Dictator for Life

- Currently on version 3 ...
  - but most still use 2.7+

- Created by Guido van Rossum in 1991
- Benevolent Dictator for Life

- Currently on version 3 ...
  - but most still use 2.7+

- Named after Monty Python
  - Still many references to TV show



WHY PYTHON?

PYTHON IS FREE

PYTHON IS FREE

- Open source

PYTHON IS FREE

- Open source
- Batteries included: lots of built-in functionality

PYTHON IS FREE

- Open source
- Batteries included: lots of built-in functionality
- Many (free or open-source) third-party libraries

PYTHON IS EASY

Java

```
public class HelloJava {
    public static void main( String[] args ) {
        System.out.println("Hello, Java!");
    }
}
```

PYTHON IS EASY

Python

```
print "Hello, world!"
```

PYTHON IS EASY: INSTALLING PACKAGES

On command line, type

```
> pip install flask
```

In python, write

```
>>> import flask
```

PYTHON IS DYNAMIC

PYTHON IS DYNAMIC

- No compiling

PYTHON IS DYNAMIC

- No compiling

PYTHON IS DYNAMIC

- No compiling
- Dynamic typing

PYTHON IS DYNAMIC

- No compiling
- Dynamic typing

Usually, you need to define the type of your variable:

– is it text?

– is it an integer?

– is it a decimal?

```
double pi = 3.14;
```

PYTHON IS DYNAMIC

- No compiling
- Dynamic typing

In python, your types a *dynamic*.

```
>>> x = 1
>>> x
1
>>> x = 'horseshoe'
>>> x
'horseshoe'
>>> _
```

WHAT ARE THE ADVANTAGES TO PYTHON?

- Easy to learn, easy to use
- Batteries Included: large collection of built-in libraries
- Simple and clean syntax

WHAT ARE THE ADVANTAGES TO PYTHON?

- Easy to learn, easy to use
- Batteries Included: large collection of built-in libraries
- Simple and clean syntax – very strict indent rules

Python sounds amazing! What is it bad at?

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)
- No compilation means discovery of errors at runtime

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language (but keep in mind that this is a conscious tradeoff)
- No compilation means discovery of errors at runtime

```
if year == “2016”:
    print “Election year!” + 2016
```

TypeError will only raise if the condition is true.

Python sounds amazing! What is it bad at?

- Python is slower than a lower-level language
  (but keep in mind that this is a conscious tradeoff)
- No compilation means discovery of errors at runtime
- Dynamic typing allows for bad practice

PYTHON SYNTAX

DATA TYPES

```
x = 36  # this is an integer
x = 3.14  # a decimal number
x = True  # either True or False
x = "This is a string"
```

DATA TYPES

```
x = [1, 2, 3, 4]  # a list

# lists can contain elements of any type
x = [36, 3.14, True, "This is a string"]
x = [36, 3.14, True, "This is a string", [1, 2, 3, 4]]

# elements are numbered, starting with 0 (!)
print x[0]  # will print first element
```

DATA TYPES

```
# dictionaries (maps)
x = {'name': 'Joe', 'age': 75}  # this is a dictionary
x = dict(name='Joe', age=75)  # same as above (old syntax)

print x['name']  # will print 'Joe'
```

IF/ELSE STATEMENTS

- Allow us to take different paths through depending on some condition

```
x = 5
if x > 4:
    print "This number is greater than 4"
```

IF/ELSE STATEMENTS

- Allow us to take different paths through depending on some condition

```
x = 5
if x > 4:
    print "This number is greater than 4"
else:
    print "This number is not greater than 4"
```

IF/ELSE STATEMENTS

- Allow us to take different paths through depending on some condition

```
x = 5
if x > 4:
    print "This number is greater than 4"
elif x == 4:
    print "This number is equal to 4"
else:
    print "This number is smaller than 4"
```

LOOPING – FOR

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
for state in emotions:
    print "I feel", state
    if state == "happy":
        print "Happy is good, hooray!"
```

LOOPING – FOR

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
for state in emotions:
    print "I feel", state
    if state == "happy":
        print "Happy is good, hooray!"
```

```
I feel happy
Happy is good, hooray!
I feel sad
I feel ¯\_(ツ)_/¯
```

LOOPING – WHILE

- Allows us to perform the same operation on each element, one by one

```
emotions = ["happy", "sad", "¯\_(ツ)_/¯"]
while len(emotions) > 0:
    state = emotions.pop()
    print "I feel", state
    if state == "happy":
        print "Happy is good, hooray!"
```

FUNCTIONS

- Allow us to save some piece of code to reuse later

```
def avg(lst):
    """Compute the average of a list."""
    return sum(lst) / float(len(lst))
```

FUNCTIONS

- Allow us to save some piece of code to reuse later

```
def avg(lst):
    """Compute the average of a list."""
    return sum(lst) / float(len(lst))
```

```
>>> avg([1, 2, 3])
2.0
>>>
```

EXCEPTIONS

- If code doesn't compute, an *Exception* is raised, and the script crashes

```
def avg(lst):
    """Compute the average of a list."""
    return sum(lst) / float(len(lst))
```

```
>>> avg([])
Traceback (most recent call last):
  File "<stdin>", line 3, in avg
ZeroDivisionError: float division by zero
```

EXCEPTIONS – TRY / EXCEPT

- With try/except, you can catch exceptions and save the day

```
def avg(lst):
    """Compute the average of a list."""
    try:
        return sum(lst) / float(len(lst))
    except ZeroDivisionError:
        return None
```

EXCEPTIONS – TRY / EXCEPT

- With try/except, you can catch exceptions and save the day

```
def avg(lst):
    """Compute the average of a list."""
    try:
        return sum(lst) / float(len(lst))
    except ZeroDivisionError:
        return None
```

```
>>> avg([])
>>>
```

EXCEPTIONS – TRY / EXCEPT

- With try/except, you can catch exceptions and save the day

```
def avg(lst):
    """Compute the average of a list."""
    try:
        return sum(lst) / float(len(lst))
    except ZeroDivisionError:
        return None
```

```
>>> avg(3)
TypeError: 'int' object is not iterable
```

OPERATIONS

- Python shell is just a complex calculator:

```
>>> 3 + 4
7
>>> 1 / 2
0
>>> 1 / 2.
0.5
>>> 3 ** 2
9
```

OPERATIONS

- Python shell is just a complex calculator:

```
>>> ['A', 'B'] + ['A', 'C']
['A', 'B', 'A', 'C']
>>> ['A'] * 5
['A', 'A', 'A', 'A', 'A']
>>> 'A' * 5
'AAAAA'
>>> list('ABCDEF')
['A', 'B', 'C', 'D', 'E', 'F']
```

- Let's practice!

WRITING A SCRIPT

So far, we have coded directly in the interpreter.

You can imagine that for larger-scale projects, you'd like to save your work in a file, or even build a big application containing multiple files and packages.

```
print "Hello, world!"
```

To run, type:

python hello.py

```
print "Hello, world!"
```

```python
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def main():
    """Print 'Hello, world!'."""
    print "Hello, world!"


if __name__ == "__main__":
    main()
```

```
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def main():
    """Print 'Hello, world!'."""
    print "Hello, world!"


if __name__ == "__main__":
    main()
```

The # indicated a comment, which will be ignored by python.

```
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def main():
    """Print 'Hello, world!'."""
    print "Hello, world!"


if __name__ == "__main__":
    main()
```

The # indicated a comment, which will be ignored by python.

In this case, however, it *is* actually some code to tell how to run this program independently.

To run, type:

./hello_world.py

```
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def main():
    """Print 'Hello, world!'."""
    print "Hello, world!"


if __name__ == "__main__":
    main()
```

On top of your file, it is good practice to include a doc string, which is a paragraph of text between triple double-quotes “”“[text]”“”, explaining the program to developers and users.

```
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def main():
    """Print 'Hello, world!'."""
    print "Hello, world!"


if __name__ == "__main__":
    main()
```

On top of your file, it is good practice to include a doc string, which is a paragraph of text between triple double-quotes “”“[text]”“”, explaining the program to developers and users.

Each function is supposed to have a doc string, too.

```
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def main():
    """Print 'Hello, world!'."""
    print "Hello, world!"


if __name__ == "__main__":
    main()
```

Python has a number of system variables, which contain information about the current application.

If the file is run as main program,

__name__ = “main”,

but if the file was imported in another program, then we’d have

__name__ = “hello_world”.

```python
#!/usr/bin/env python
"""
Simple program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""


def hello():
    """Print 'Hello, world!'."""
    print "Hello, world!"


def bye():
    """Print 'Bye, world!'."""
    print "Bye, world!"


def main():
    """Say hello and bye."""
    hello()
    bye()


if __name__ == "__main__":
    main()
```

```python
#!/usr/bin/env python
"""
Unnecessarily complicated program that prints 'Hello, world!'.

Ruben Naeff
May 2016
"""
from random import random

from meet_world import hello, bye


def main():
    """Randomly says hello or bye."""
    if random() > .5:
        hello()
    else:
        bye()


if __name__ == "__main__":
    main()
```

- Let's practice!

LIBRARIES

On command line, type

```
> pip install numpy
```

On command line, type

```
> pip install numpy
```

You can view your installed packages with

```
> pip freeze
```

On command line, type

```
> pip install numpy
```

You can view your installed packages with

```
> pip freeze | grep numpy
```

On command line, type

```
> pip install numpy
```

On command line, type

```
> pip install numpy
```

In python, write

```
>>> import numpy
```

On command line, type

```
> pip install numpy
```

In python, write

```
>>> import numpy as np
```

On command line, type

```
> pip install numpy
```

In python, write

```
>>> import numpy as np
>>> np.random.random(3)
```

On command line, type

```
> pip install numpy
```

In python, write

```
>>> import numpy as np
>>> np.random.random(3)
array([ 0.30807441,  0.58508608,  0.78048742])
```

On command line, type

```
> pip install numpy
```

In python, write

```
>>> from numpy import random
>>> random.random(3)
array([ 0.30807441,  0.58508608,  0.78048742])
```

On command line, type

```
> pip install numpy
```

In python, write

```
>>> from numpy.random import random
>>> random(3)
array([ 0.30807441,  0.58508608,  0.78048742])
```

Subject	Package
Web development	flask, django, mezzanine, Jinja2
Math & Science	numpy, scipy
Data Analysis	pandas, statsmodels
Machine Learning	scikit-learn
Web scraping	beautifulsoup
Data visualizations	matplotlib, seaborn, vincent

IPYTHON AND JUPYTER NOTEBOOK

IPython is an advanced shell that makes programming in python much easier.

IPython is an advanced shell that makes programming in python much easier.

```
In [6]: max?
Docstring:
max(iterable[, key=func]) -> value
max(a, b, c, ...[, key=func]) -> value

With a single iterable argument, return its largest item.
With two or more arguments, return the largest argument.
Type:      builtin_function_or_method
```

IPython is an advanced shell that makes programming in python much easier.

```
>>>
>>> print max.__doc__
max(iterable[, key=func]) -> value
max(a, b, c, ...[, key=func]) -> value

With a single iterable argument, return its largest item.
With two or more arguments, return the largest argument.
>>> _
```

IPython is an advanced shell that makes programming in python much easier.

Tab completion

```
In [8]: ma
%macro      %magic      %man        %matplotlib  map         max
```

IPython is an advanced shell that makes programming in python much easier.

Cell magic

```
In [3]: %time 1 + 1
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 9.06 µs
Out[3]: 2
```

IPython is an advanced shell that makes programming in python much easier.

Cell magic

```
In [2]: %timeit 1.23 ** 1.23
10000000 loops, best of 3: 25 ns per loop
```

IPython is an advanced shell that makes programming in python much easier.

But it even gets better...

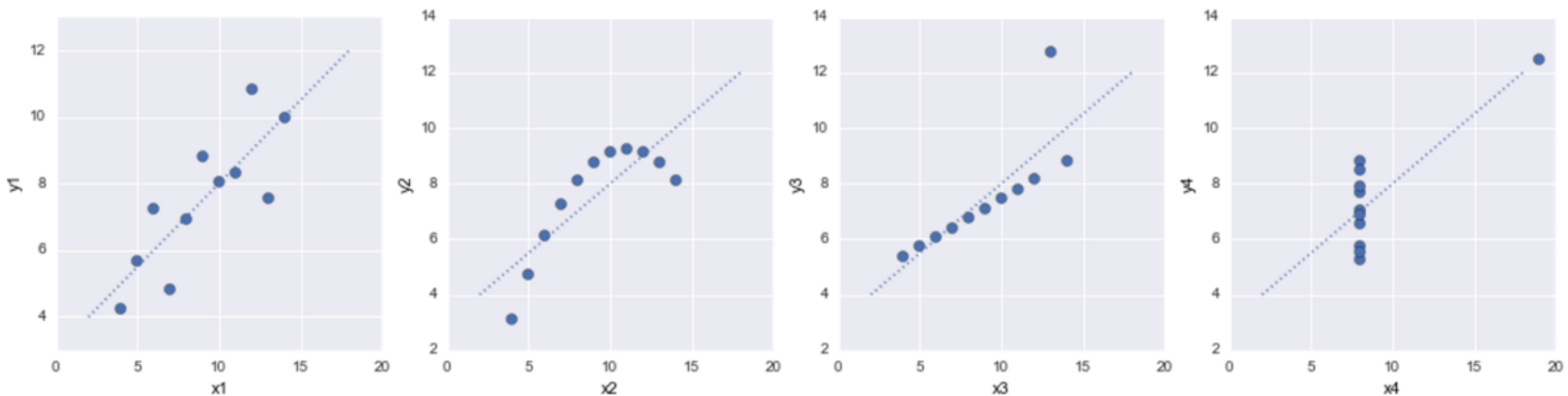
Jupyter notebook, formerly known as **IPython notebook**, lets you write python in a web browser.

- Copy and paste code like in a text editor
- Integrate graphs in the same document
- Rich markup and HTML support for beautiful layout

Basic visualizations

Plot each dataset.

In [4]:
```
fig, axes = plt.subplots(nrows=1, ncols=4, figsize=(18,4))
for i in xrange(1, 5):
    x, y = "x" + str(i), "y" + str(i)
    data.plot(kind='scatter', x=x, y=y, ax=axes[i-1], s=50)
    slope, intercept = trendlines[i - 1]
    axes[i - 1].plot([2, 18], [intercept + 2 * slope, intercept + 18 * slope], ':')
```



To install, just type on the command line

```
pip install jupyter
pip install seaborn   # this is another visual package
```

To launch, just type

```
jupyter notebook
```

- Let’s do it!

FLASK

THANK YOU.