

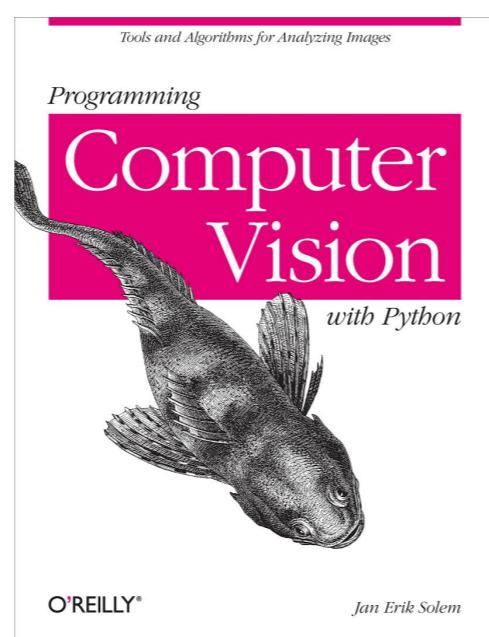
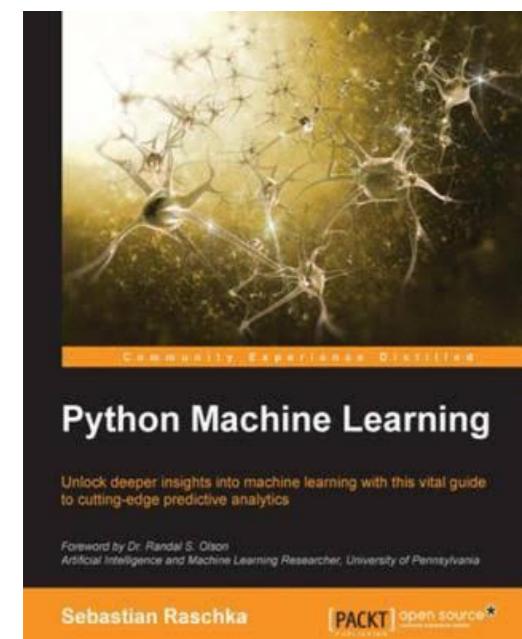


INTRODUCTION TO PYTHON

WHY PYTHON

- One of the most popular programming language for data science
 - Many useful add-ons and libraries
- Well supported
 - Large community. Many tutorials on the web
 - Current version is V-3.4 (we will use V-2.7 -more stable-)
 - Multi plateform
- Easy to use and understand the code (interpreted language)
 - Can do many things with few lines of code (pythonic!)
 - Extensions build upon lower layers (Fortran/C) allow fast and vectorized operation

O'REILLY



REFERENCES

.....

- Fluent Python
- Data Science from Scratch
- Python Machine learning
- Programming Computer Vision with Python
- <https://www.python.org>

INSTALLING PYTHON AND PACKAGES

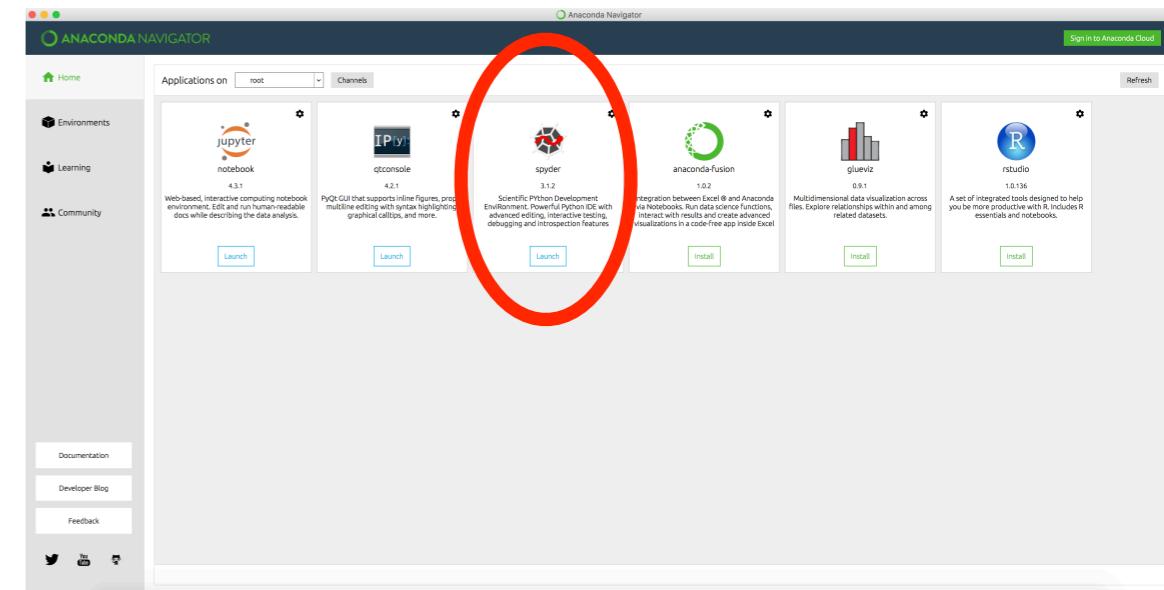
- Python is available for Windows / Mac OS X / Linux
- We will use Python 2.7.
 - If you want to use Python 3.4 -><https://wiki.python.org/moin/Python2orPython3>
 - Install Anaconda -> <https://docs.continuum.io/anaconda/install>
 - For Python 2.7 version
 - Or you can install Python only -> <https://www.python.org>
 - To install libraries use *pip* installer program
 - **pip install SomePackage (--upgrade)** ← need to install libraries

LIBRARIES OVERVIEW

- NumPy
 - Package used for scientific computing with Python.
- Matplotlib
 - Produces high quality figures (plot graph, points, lines, curves, ...).
- Panda
- PIL
 - Python Imaging Library: general image handling and lots of useful basic image operations.
- SciPy
 - Package for mathematics that provides many useful operations.
- OpenCv

CRASH COURSE IN PYTHON (THE BASICS)

- Python is an **interpreted language**
 - Easy readability
 - Fewer lines of codes
- Open Anaconda Navigator



Program run

File name

Help/documentation

Python script edition

Python console

A screenshot of the Spyder Python IDE. On the left, there's a code editor window titled 'Editor - /Users/diegothomas/Documents/Projects/Python/FFRec/Main.py' containing Python code. A green arrow points from the 'File name' callout to this window. In the center, there's a 'Help' panel with tabs for 'Source', 'Console', and 'Object', currently showing the 'Usage' tab. A green arrow points from the 'Help/documentation' callout to this panel. On the right, there's a 'Console' window titled 'Console 1/A' showing an IPython session. A green arrow points from the 'Python console' callout to this window. The top of the screen shows the Spyder interface with tabs for 'Application.py', 'Main.py', and 'Menu.py'.

HELLO WORLD

- In Python console type
print “Hello World”
- Press ‘return’ on keyboard

The screenshot shows the IPython console interface. At the top, it says "IPython console". Below that, it displays the Python version and license information. In the main area, there are two input lines: "In [1]: print "Hello World"" followed by the output "Hello World". A green oval highlights the command line, and a red oval highlights the title bar.

```
Python 2.7.13 |Anaconda custom (x86_64)| (default, Dec 20 2016, 23:05:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- 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.

In [1]: print "Hello World"
Hello World

In [2]:
```

- In the file manager type
print “Hello World”
- Click the ‘Run’ icon (save file)

This screenshot illustrates the workflow for running a script. On the left, a file manager window shows a file named "untitled0.py" with the code "print "Hello World"". A red oval highlights the line of code. On the right, the IPython console shows the command "runfile('.../untitled0.py', wdir='...')". A red oval highlights the command line, and another red oval highlights the title bar. The console also displays the output "Hello World".

Run icon

```
untitled0.py

1 #!/usr/bin/env python2
2 # -*- coding: utf-8 -*-
3 """
4 Created on Tue Feb 14 10:30:57 2017
5
6 @author: diegothomas
7 """

9 print "Hello World"
```

```
IPython console

Python 2.7.13 |Anaconda custom (x86_64)| (default, Dec 20 2016, 23:05:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.1.0 -- 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.

In [1]: print "Hello World"
Hello World

In [2]: runfile('/Users/diegothomas/untitled0.py', wdir='/Users/diegothomas')
Hello World

In [3]:
```

WHITESPACE FORMATTING

- Python uses indentation to delimit blocks of code:

```
for i in [1, 2, 3, 4, 5]:
```

```
    print i                      #first line in “for i” block
```

```
    for j in [1, 2, 3, 4, 5]:
```

```
        print j                  #first line in “for j” block
```

```
        print i+j                #last line in “for j” block
```

```
    print i                      #last line in “for i” block
```

```
print “done looping”
```

- Dangerous to copy paste code in the shell !!

```
for i in [1, 2, 3, 4, 5]:
```

Interpreter thinks that the blank line signals the end of the for loop's block

```
#notice the blank line
```

```
    print i
```

This would produce the error: **IndentationError: expected an indented block**

Whitespace is ignored inside parentheses and brackets

MODULES

- Some features of Python are not loaded by default

- Need to **import** the modules that contain them

```
import matplotlib.pyplot as plt
```

```
plt.plot([1,2,3,4])
```

```
plt.ylabel('some numbers')
```

```
plt.show()
```

- After the **import**, access the functions from the module by prefixing them with **plt..**

ARITHMETIC

- Addition

a = 1+2

- Subtraction

a = 2-1

- Multiplication

a = 2*3

- Division

- Python uses integer division by default:

- $5/2$ equals 2

from __future__ import division

- Now $5/2$ equals 2.5. For integer division, use $5 // 2$ (equals 2)

FUNCTIONS

- A function is a rule for taking zero or more **inputs** and returning a corresponding **output**

- Definition:

Name of the function

Magic word that says we are going to create a function

```
def double(x):
```

Input

```
# Put your comments here (what the function does etc...)
```

```
# For example: this function multiplies its input by 2
```

```
return x*2
```

Magic word that says this is the output

STRINGS

- Array of characters (represent common words)
- Delimited by quotes (have to match)

```
single_quoted_string = 'data science'  
double_quoted_string = "data science"
```

- Backslash for special characters

```
tab_string = "\t" # the tab character  
len(tab_string) # is 1
```

- Use r"" for raw strings (useful for directories)

```
raw_string = r"\t" # represents '\' and 't'  
len(raw_string) # is 2
```

- For multiline strings, use double or triple quotes

```
multi_line_string = """ This is the first line.  
This is the second line  
This is the third line """
```

LISTS

- Ordered collection. **Important data structure!!**

```
integer_list = [1, 2, 3]  
  
heterogeneous_list = ["string", 0.1, true]  
  
list_of_lists = [integer_list, heterogeneous_list, []]  
  
list_length = len(integer_list) # equals 3  
  
list_sum = sum(integer_list) # equals 6
```

- Access the nth element with brackets

```
X = range(10) # is the list [0, 1, ..., 9]  
  
Zero = X[0] # equals 0  
  
Nine = X[-1] # the last element  
  
X[0] = -1 # now X equals [-1, 1, 2, ..., 9]  
  
one_to_four = X[1:5] # [1, 2, 3, 4]
```

- Check for membership with in

```
1 in [1, 2, 3] # is true  
0 in [1, 2, 3] # is false
```

- Other common operations

```
x = [1, 2, 3]  
x.extend([4, 5, 6]) # x is [1,2,3,4,5,6]  
  
y = x + [7, 8, 9] # y is [1,2,3,4,5,6,7,8,9]  
x.append(0) # x is [1,2,3,4,5,6,0]  
x, y = [1, 2] # x is 1 and y is 2  
_, y = [1, 2] # y is 2, don't care  
# about the first element
```

TUPLES

- Uses parentheses (or nothing) instead of brackets

```
my_tuple = (1, 2)
```

```
other_tuple = 1, 2
```

- Cannot modify a tuple

```
my_tuple[0] = 0 # ERROR!!
```

- Convenient way to return multiple values from functions

```
def sum_and_product(x, y)
```

```
    return (x+y), (x*y)
```

DICTIONARIES

- Associate *values* with *keys*. Quickly retrieves *value* corresponding to a given *key*

```
Empty_dict = {}
```

```
empty_dic2 = dict()
```

```
Grades = {"Joel": 80, "Tim": 95}
```

- Lookup for the value for a key using brackets
(safer version with **get**)

```
joel_grade = Grades["Joel"] #equal 80
```

- check for existence using in

```
Joel_has_grade = "Joel" in Grades # True
```

```
Kate_has_grade = "Kate" in Grade # False
```

- Assign value with brackets

```
Grades["Joel"] = 70 # Joel now has a grade of 70
```

```
Grades["Kate"] = 85 # adds a new entry for Kate
```

- Simple way to represent structured data

```
Tweet = {"user": "Kentaro",  
         "text": "Data science is Awesome",  
         "retweet count": 100,  
         "hashtag": ['#data', '#science', '#datascience',  
                    '#Awesome', '#blabla']  
       }
```

- Other functions

```
tweet_keys = Tweet.keys() # list of keys
```

```
tweet_values = Tweet.values() # list of values
```

```
tweet_items = Tweet.items() # list of (key, value)  
tuples
```

SETS

- Collection of *distinct* elements

```
s = set()  
s.add(1) # s is now {1}  
s.add(2) # s is now {1, 2}  
s.add(2) # s is still {1, 2}  
x = len(s) # equals 2
```

- `in` is very fast on sets
- Easy to find *distinct* elements in a collection

```
item_list = [1, 2, 3, 1, 2, 3]  
num_items = len(item_list) # equals 6  
item_set = set(item_list) # equals {1, 2, 3}  
num_distinct_items = len(item_set) # equals 3  
distinct_item_list = list(item_set) # equals [1, 2, 3]
```

CONTROL FLOW

► Conditional action using **if**

```
if 1 > 2
```

Message = “if only 1 were greater than 2 ...”

```
elif 1 > 3
```

Message = “**elif** stands for **else if**”

```
else
```

Message = “when all **else** fails use **else**”

```
parity = “even” if x % 2 == 0 else “odd”
```

```
# write on 1 line
```

► The **while** loop

```
x = 0
```

```
while x < 10:
```

```
    print x, “is less than 10”
```

```
    x += 1
```

► The **for** and **in** loop

```
for x in range(10):
```

```
    print x, “is less than 10”
```

► The **continue** and **break** command

```
for x in range(10):
```

```
    if x == 3
```

```
        continue # go immediately to the next iteration
```

```
    if x == 5
```

```
        break # quit the loop entirely
```

```
    print x
```

```
# this will print 0, 1, 2, 4
```

TRUTHINESS

- **Booleans** are used to express truthiness

```
one_is_less_than_true = 1 < 2      # equals True
```

```
true_equals_false = True == False  # equals False
```

- A non-existent value is identified with None

EXERCISES

- 1. Implement a function that computes the sum of :

$$(1 + 2 + 3 + \dots + n)$$

- Verify for some numbers that:

Result equals $n(n-1) / 2$

Hint:

- (1) Use the range() function
- (2) Use the sum() function

- 2. Implement a function that computes the sum of the n first numbers of the Fibonacci sequence:

- $f(0) = f(1) = 1;$

- $f(t+2) = f(t) + f(t+1)$