CSCI2040 Tutorial 0: Introduction to Python Programming

```
print("Hello, Python!")
```

Jan. 15, 2020

Outline

Background and Installation

Python Syntax Basics

Data Structures and Sequences

Functions

Python Classes

Files and File Methods

Importing Modules

Useful Python Packages

Try It Now

For Further Learning

Background and Installation

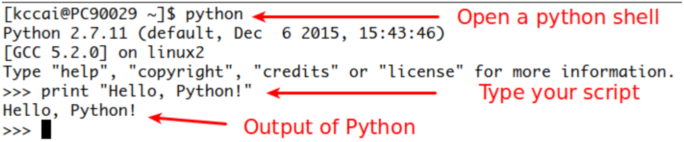
Python Overview

Python is an easy-to-learn and easy-to-read language. If you can learn and read English, you can also learn and read Python. Here are the two typical features of Python:

- Interpreted: You need no compiling before executing your code. If you have a python script file `hello.py`, you can run this file in a command shell/window with

  ```
  python hello.py
  ```

- Interactive: You can sit at a Python prompt and play with the interpreter. For example:



```
[kccai@PC90029 ~]$ python
Python 2.7.11 (default, Dec  6 2015, 15:43:46)
[GCC 5.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print "Hello, Python!"
Hello, Python!
>>> █
```

Figure: Interaction with Python Shell

Install Python

General recommendations:

- Use Python 3: we highly recommend you to use Python 3 instead of Python 2.
- Install Anaconda: Anaconda is a free Python distribution with over 150 scientific Python packages. There are Windows, Mac OSX, Linux versions available. Please follow the instructions at the link below to install
  Anaconda. (Run the installer and just follow the installer's guidance.)

  http://docs.continuum.io/anaconda/install
- IDE (Integrated Development Environment): You are encouraged to use Pycharm Community Edition to write your Python programs. You can also use other tools that you are comfortable with. Download it at the link (Community Edition is free!)

  https://www.jetbrains.com/pycharm/download/

Packages Management Using conda

- Install packages. (Please use conda to install packages and pip is not recommended!).

```
conda install seaborn
```

- Update packages. (Sometimes you need update packages to get new functions and features)

```
conda update scikit-learn
conda update ipython ipython-notebook
```

- Remove packages. (Save space by removing unwanted packages)

```
conda remove tornado
```

Setup Python Environment

Normally, the installer can set and find the Python installation path automatically. If not, you can setup Python environment manually.

On Windows:

- Open the Command Prompt, type the command `path`, you should see the output including the following paths:

```
C:\Anaconda3;C:\Anaconda3\Scripts;C:\Anaconda3\Library\bin
```

- If not, you can add the paths above to the system environment variable `PATH` following the instructions.
- Type the command `python` in the Command Prompt window. If you can see the python shell prompt (starting with `>>>`), then the python environment is set.

On Mac and Linux:

- Please follow the installation guide carefully!

Setup Python Environment (Windows)

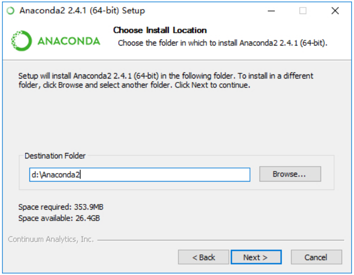
For students which have already installed Anaconda2 before installing Anaconda3, python3 is inactive and python2 is set as the default python selection.

- If you install Anaconda2 in the folder:

  `C:\Anaconda2`

  then you need to install Anaconda3 in the subdirectory of

  `C:\Anaconda2\envs`



Anaconda2 2.4.1 (64-bit) Setup

Choose Install Location

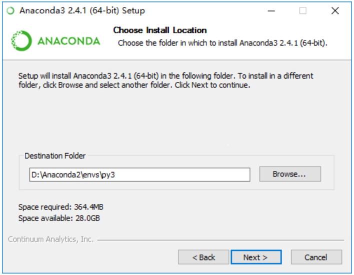
Choose the folder in which to install Anaconda2 2.4.1 (64-bit).

Setup will install Anaconda2 2.4.1 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: d:\Anaconda2

Space required: 353.9MB
Space available: 26.4GB

Continuum Analytics, Inc.



Anaconda3 2.4.1 (64-bit) Setup

Choose Install Location

Choose the folder in which to install Anaconda3 2.4.1 (64-bit).

Setup will install Anaconda3 2.4.1 (64-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: D:\Anaconda2\envs\py3

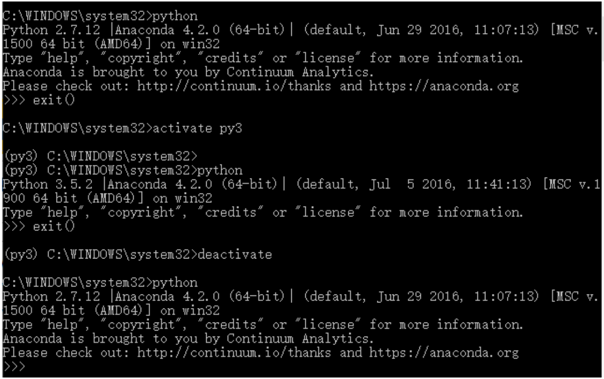
Space required: 364.4MB
Space available: 28.0GB

Continuum Analytics, Inc.

Setup Python Environment (Windows)

Switch between python2 and python3 in the command prompt window:

- Check the active python version: `python`
- Switch from python2 to python3: `activate py3`
- Switch from python3 to python2: `deactivate`

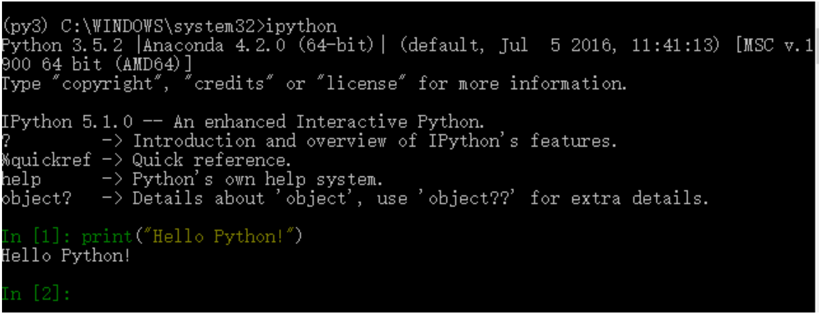


```
C:\WINDOWS\system32>python
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.
1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>> exit()

C:\WINDOWS\system32>activate py3

(py3) C:\WINDOWS\system32>
(py3) C:\WINDOWS\system32>python
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1
900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

(py3) C:\WINDOWS\system32>deactivate

C:\WINDOWS\system32>python
Python 2.7.12 |Anaconda 4.2.0 (64-bit)| (default, Jun 29 2016, 11:07:13) [MSC v.
1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
Anaconda is brought to you by Continuum Analytics.
Please check out: http://continuum.io/thanks and https://anaconda.org
>>>
```

Get Started with IPython (Command line)

IPython is an enchanced interactive Python interpreter.
IPython in the command line is much more powerful than the python built-in interpreter.

- Start IPython by typing the command ipython in the Windows Command Prompt or in your Mac OS Terminal.

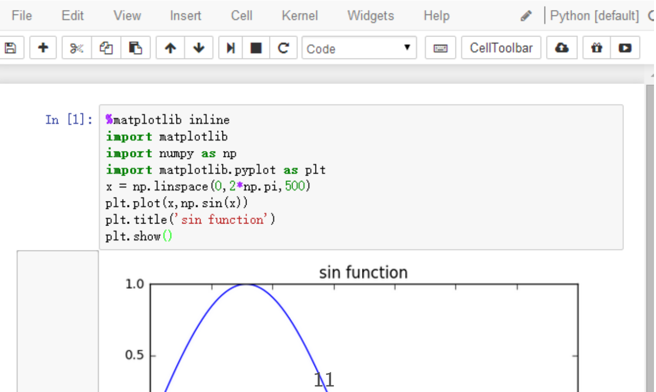


```
(py3) C:\WINDOWS\system32>ipython
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1
900 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

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 Python!")
Hello Python!

In [2]:
```

Get Started with IPython (Browser)

IPython Notebook is IPython in your browser, in which you can combine code execution, plots, images and so on.

- Start it by typing the command `ipython notebook` in the Command Prompt window or Terminal. (Don't close the window.) Then click New -> Python [default], you should see



```
%matplotlib inline
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,2*np.pi,500)
plt.plot(x,np.sin(x))
plt.title('sin function')
plt.show()
```

Python Syntax Basics

Data types and Variables

- Data Types: built-in types for handling numerical data, strings, Boolean values.

Type	Description	Example
`None`	The Python ”null” value	`score = None`
`str`	String type	`s = "first tutorial"`
`float`	double precision floating point number	`f = 1.23456`
`bool`	A `True` or `False` value	`game_over = False`
`int`	Signed integer	`age = 18`
`long`	Arbitrary precision signed integer	`c = 299792458000`

- Variables: a piece of storage which you can change its content during program execution. identifier is the name of the variable:
  - the first character must be a letter (a-z or A-Z) of an underscore (_), the rest can consist of *letters*, *underscore*, or *digits* (0-9)
  - identifiers are case-sensitive, `sid` and `SID` are not the same.

Operators

- Common Operators:

Operator	Name	Example
+	Plus	1+2 gives 3, 'ab'+'cd' gives 'abcd'
-	Minus	1-2 gives -1
*	Multiply	2*3 gives 6, 'ha'*3 gives 'hahaha'
**	Power	3**4 gives 81 (i.e., 3*3*3*3)
/	Divide	4/3 gives 1 (Python 2.x) or 1.3333333 (Python 3.x)
//	Floor Division	4//3.0 gives 1.0
%	Modulo	8%3 gives 2

- Logic Operators

Operator	Name	Example
<	less than	5<3 gives False
>	greater than	5>3 gives True
<=	less than or equal to	3<=6 gives True
>=	greater than or equal to	3>=6 gives False
==	equal to	5==5 gives True
!=	not equal to	5!=5 gives False
not	Boolean Not	x=True; not x returns False
and	Boolean And	x,y=True,False; x and y returns False
or	Boolean Or	x,y=True,False; x or y returns True

Control Flow (1)

`if`, `elif`, and `else`

```
'''
test if x is negative or not.
'''
x = 3.14
if x < 0:                      # Don't miss the COLONS!!!
    print('x is negative')
elif x == 0:
    print('x is 0')
else:
    print('x is nonegative')
```

REMARK:

- Python uses Indentation instead of Braces.
- Python Comment uses `# bla bla` or `'''bla bla'''`

Control Flow (2)

for loops

```
sequence = [1,3,5,7,9]
total = 0
for i in sequence: # add a list of numbers using for loop.
    total += i
print("Sum of sequence is", total)
```

break and continue in for loop

```
sequence_1 = [1,None,5,None,9,11]
total_1 = 0
for i in sequence_1: # add a list of non-None numbers that
    if i is None:    # less than 10 using for loop
        continue
    if i >= 10:
        break
    total_1 += i
print("Sum of sequence_1 is", total_1)
```

Control Flow (3)

while loop

```
# find numbers in Fibnacci Sequence that are less than 10
a,b = 0,1      # multiple assignments in Python
while True:
    print(b)
    a,b = b,a+b
    if b > 10:
        break
print('end')
```

pass means no action is to be taken

```
x = -1
if x < 0:
    pass  # No action here
else:
    print("x is nonegative!")
```

Data Structures and Sequences

String

String is a powerful and flexible built-in data type that comes along with many powerful functions.

- use single quotes ' or double quotes " to create strings

```
In [1]: a = 'this is a string'
In [2]: b = "I'm a string too"
```

- many Python objects can be converted to string using the str function

```
In [3]: c = 1.2345

In [4]: str(c)
Out[5]: '1.2345'
```

- concatenate strings using +

```
In [5]: a + b
Out[5]: "this is a stringI'm a string too"
```

- Index starts from 0 in Python

```
In [6]: b[0]
Out[6]: 'I'

In [7]: b[1]
Out[7]: "'"
```

- string has many powerful methods
  Please read through the methods of string in the official documentation.

Tuple

Tuple is one dimensional, fixed-length, *immutable* sequence of Python objects.
In an IPython command window,

- create a tuple with a *comma-separated* sequence of values:

```
In [1]: tup = 1,2,3

In [2]: tup
Out[2]: (1, 2, 3)
```

- a list can be converted to tuple:

```
In [3]: tuple([4,0,2])
Out[3]: (4, 0, 2)
```

- tuples can be concatenated using the + operator:

```
In [4]: (1,3) + (2,4) + (0,1)
Out[4]: (1, 3, 2, 4, 0, 1)
```

- concatenating copies of a tuple using *:

```
In [5]: ('you','lol') * 3
Out[5]: ('you', 'lol', 'you', 'lol', 'you', 'lol')
```

- methods of tuple:

```
In [7]: b_tup.
b_tup.count  b_tup.index
```

List

`list` is variable-length sequence and its content can be modified.

- define a list using square brackets `[]`:

```
In [1]: a_list = [2,3,7,None]

In [2]: a_list
Out[2]: [2, 3, 7, None]
```

- define a list with `list` type function:

```
In [3]: b_tup = ('Python', 'is', 'easy!')

In [4]: b_list = list(b_tup)

In [5]: b_list
Out[5]: ['Python', 'is', 'easy!']
```

- change the content of `list`

```
In [6]: a_list[3] = 9

In [7]: a_list
Out[7]: [2, 3, 7, 9]
```

- methods of list:

```
In [8]: a_list.
a_list.append    a_list.index    a_list.remove    ...
```

Dict

dict is the most important built-in Python data-structure. It is a flexibly-sized collection of *key-value* pairs.

- create dict using {} and using colons to separate keys and values

```
In [1]: d1 = {'a':'hello','b':[1,2,3,4]}

In [2]: d1
Out[2]: {'a': 'hello', 'b': [1, 2, 3, 4]}
```

- insert an element to dict

```
In [3]: d1[8] = 'eight'

In [4]: d1
Out[4]: {8: 'eight', 'a': 'hello', 'b': [1, 2, 3, 4]}
```

- delete an element from dict

```
In [5]: del d1['b']

In [6]: d1
Out[6]: {8: 'eight', 'a': 'hello'}
```

- dict has many powerful built-in methods, you should read the document of dict carefully.

Set

A set is an ordered collection unique elements. Sets are like dicts, but keys only, no values.

- create a set using the `set` function

  ```
  In [1]: a = set([2,3,3,2,1,1])

  In [2]: a
  Out[2]: {1, 2, 3}
  ```

- create a set using curly braces

  ```
  In [3]: b = {2,3,3,2,1,1}

  In [4]: b
  Out[5]: {1, 2, 3}
  ```

- Sets supports mathematical set operations like union (`a | b`), intersection (`a & b`), difference (`a - b`) and symmetric difference (`a ^ b`).
- (Unordered) Two sets are equal if their contents are equal

  ```
  In [5]: {1,2,3} == {3,1,2}
  Out[5]: True
  ```

- `set` also supports many methods. You can check them at the official documentation.

Built-in Sequence Functions

- enumerate returns a sequence of (i, value) tuples

```
In [1]: la = [4,5,3,2,1]

In [2]: for i, val in enumerate(la):
   ...:         print(i, val)
```

- sorted function returns a new sorted list from the elements of any sequence (string, tuple, list, dict, set).

```
In [1]: sorted('nba')
Out[1]: ['a', 'b', 'n']
# try sorted on other data structures by yourself
```

- zip pairs up the elements of numbers of lists, tuples, or other sequences to create a list of tuples.

```
In [1]: seq1 = ['one','two','three']
In [2]: seq2 = [1,2,3]

In [3]: zip(seq1, seq2)
Out[3]: [('one', 1), ('two', 2), ('three', 3)]

In [4]: seq3 = ['I','II','III']
In [5]: zip(seq1, seq2, seq3)
Out[5]: [('one', 1, 'I'), ('two', 2, 'II'), ('three', 3, 'III')]
```

List, Set and Dict Comprehensions

List comprehensions are of the most-lived Python language features! They allow you create a list by filtering the elements of a collection using a concise expression. They take the basic form:

```
[expr for val in collection if condition]
```

Several examples:

- Find even numbers in $[0, 100)$

  ```
  even_number = [n for n in range(0,100) if n%2 == 0]
  ```

- List Comprehension. Get upper case of a list of strings

  ```
  lower_str = ['i', 'am', 'string']
  upper_str = [s.upper() for s in lower_str]
  ```

- Set Comprehension. Get the unique length of strings

  ```
  s1 = ['a','cat','beats','a','dog']
  set_len = {len(x) for x in s1}
  ```

- Dict Comprehension. Get a mapping of integers to strings

  ```
  s1 = ['a','cat','beats','a','dog']
  idx_map = {idx:s for idx,s in enumerate(s1)}
  ```

Functions

define and call functions

Functions are the primary and most important method of code organization and reuse in Python.

- Functions are declared using the `def` keyword and returned from using the `return` keyword:

```
def myfunc(x, y, z=1.2):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

- In above `x` and `y` are *positional* arguments, `z` is a *keyword* argument. *keyword* arguments must follow *positional* arguments!
- `myfunc` can be called in the following ways:

```
myfunc(1,3,z=3.4)
myfunc(5,6,2.7)
```

Return Multiple Values

Unlike JAVA and C++, Python has the ability to return multiple values from a function. Here is a simple example:

```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c
```

A more interesting alternative is to return a `dict`

```
def f():
    a = 5
    b = 6
    c = 7
    return {'a' : a, 'b' : b, 'c' : c}
```

Do try writing this kind of functions, you will likely find yourself doing this often as many functions may have multiple outputs.

Name scope

- Names defined outside functions have global scope
- Any local names will shadow the same global name
- All values and names are destroyed after `return`
- Python uses the LEGB (Local -> Enclosed -> Global -> Built-in) rule to resolute the namescope.
- Python uses the keyword `global` to declare global variables. (Not recommended!)

```
In [1]: x = 4

In [2]: def scopetest(a):
   ...:     return x + a
   ...: print(scopetest(3))
   ...:
7
```

```
In [1]: x = 4

In [2]: def scopetest_1(a):
   ...:     x = 7
   ...:     return x + a
   ...: print(scopetest_1(3))
   ...:
10
```

Python Classes

Classes

- Encapsulate several related functions/data into a single unit
- Functions are thus called methods

An example of BankAccount class

```
class BankAccount(object):
    def __init__(self):              # default constructor method
        self.balance = 0
    def __init__(self, initBalance): # constructor method with arguments
        self.balance = initBalance
    def deposit (self, amount):      # deposit method
        self.balance = self.balance + amount
    def withdraw (self, amount):     # withdraw method
        self.balance = self.balance - amount
    def getBalance(self):            # getBalance method
        return self.balance
```

The methods can be called as following:

```
myAccount = BankAccount()
print(myAccount.getBalance())
# 0
myAccount.deposit(100)
print(myAccount.getBalance())
# 100
```

inheritance and overriding in Class

inheritance: new class has all the functionalities of parent class

```
class CheckAccount(BankAccount):# inherit from BankAccount class
    def __init__(self, initBal):
        BankAccount.__init__(self, initBal)
        self.checkRecord = {}
    def processCheck(self, number, toWho, amount):
        self.withdraw(amount)   # inherit withdraw() from BankAccount
        self.checkRecord[number] = (toWho, amount)
    def checkInfo(self, number):
        if self.checkRecord.has_key(number):
            return self.checkRecord[number]

ca = CheckAccount( 1000 )
ca.processCheck(100, 'town Gas', 328.)
ca.processCheck(101, 'HK Electric', 452.)
print(ca.checkInfo(101))
# ('HK Electric', 452.0)
print(ca.getBalance())       # inherit method getBalance() from BankAccount
ca.deposit(100)            # inherit method deposit() from BankAccount
print(ca.getBalance())
```

overriding: child class redefines the function using the same name and arguments (link)

Files and File Methods

Files

Files are persistent storage after program ends. In Python, you can simply open a file with

```
f = open('myfile.txt','r')
```

By default, the file is opened in read-only mode `'r'`. The following is a list for all valid file read/write modes.

Mode	Description
r	read-only mode (default mode).
w	write-only mode. Create a new file (delete file with the same name).
a	append to existing file.
r+	read and write.
b	Add to mode for binary file: `'rb'` or `'wb'`.

print each line of a file

```
f = open('myfile.txt','r')
for line in f:
    print(line)
```

File methods

The most commonly-used file methods:

Method	Description
`f = open("filename")`	open a file, return file value
`f = open("filename", 'w')`	open a file for writing
`f.read()`	return a single character value
`f.read(n)`	return no more than n character values
`f.readline()`	return the next line of input
`f.readlines()`	return all the file content as a list
`f.write(s)`	write string s to file
`f.writelines(lst)`	write list lst to file
`f.close()`	close file
`f.flush()`	Flush the internal I/O buffer to disk
`f.seek(pos)`	Move to indicated file position
`f.tell()`	Return current file position as integer
`f.closed`	`True` if the file is closed

Importing Modules

Modules

- A module is simply a Python file.
- The `import` statement scans a module and execute each statement in the module.
- Using `help(module_name)` to find out the document of the module

```
>>> import string
>>> print(type(string))
<type 'module'>
>>> help(string)
Help on module string:

NAME
    string - A collection of string operations ...
FILE
    C:\anaconda2\envs\py3\lib\string.py
...
```

Modules

Modules are just libraries in other languages, which have bundled a lot of useful functions.

- Here are several ways of importing module functions:

```
import math           # (1) import the math module
import numpy as np    # (1) import numpy and name it as np
from math import sin # (2) only import sin function from math
from math import *    # (3) import all the functions in math
```

- Difference between (1) and (3) when you call the sin function

```
# using (1)
math.sin(90) # specify the module name
# using (3)
sin(90)      # the module name can be ignored
```

- Method (1) is recommended! Why? To avoid name space collision!
- For more built-in modules, please check Python Standard Library.

Useful Python Packages

NumPy

NumPy (i.e., Numerical Python) is the foundational package for scientific computing in Python. *The majority programming assignments of CSCI3320 will be based on NumPy and libraries built on top of NumPy.*

- the NumPy package is usually imported as follows:

```
import numpy as np
```

- A simple example of using NumPy: 2D array (matrix)

```
import numpy as np
# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print(a)
# prints "array([[ 1,  2,  3],
#                [ 4,  5,  6],
#                [ 7,  8,  9],
#                [10, 11, 12]])"
```

SciPy

SciPy is a collection of packages addressing a number of different standard problem domains in scientific computing.

- Most useful (sub)packages of SciPy
  - scipy.constants: many mathematical and physical constants.
  - scipy.linalg: linear algebra routines and matrix decompositions.
  - scipy.sparse: sparse matrices and sparse linear system solvers.
- A simple example of using SciPy: calculate the determinant

```
import scipy.linalg
import numpy as np
arr = np.array([[1, 2],
                [3, 4]])
scipy.linalg.det(arr)
# the determinant is 2
```

pandas

pandas provides rich data structures and functions designed to make working with structured data fast, easy, and expressive. The commonly used object in pandas is the DataFrame, a two dimensional tabular column-oriented data structure with both row and column labels.

- A simple example of using pandas to read .csv file

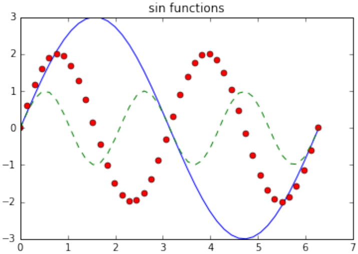
```
import pandas as pd
df = pd.read_csv("https://raw.githubusercontent.com/\
                  cs109/2014_data/master/countries.csv")
df.head(5)

#     Country   Region
# 0   Algeria   AFRICA
# 1   Angola    AFRICA
# 2   Benin     AFRICA
# 3   Botswana  AFRICA
# 4   Burkina   AFRICA
```

matplotlib

matplotlib is the most popular Python library for producing plots and other 2D data visualizations.

- matplotlib integrates well with IPython. You can see the plotting result in IPython by putting %matplotlib inline at the beginning of your script.
- A simple example of plotting the sin functions:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0, 2*np.pi, 42)
f1 = 3 * np.sin(x)
f2 = 2 * np.sin(2*x)
f3 = 1 * np.sin(3*x)
plt.plot(x, f1)
plt.plot(x, f2, 'ro')
plt.plot(x, f3, 'g--')
plt.title('sin functions')
plt.show()
```

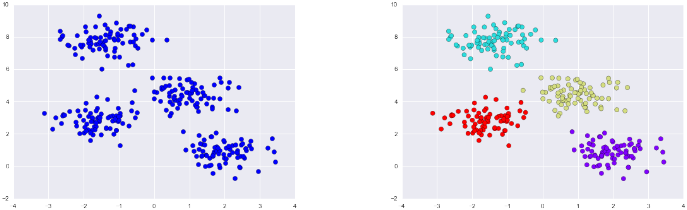


scikit-learn

scikit-learn is a Python package designed to give access to well-known machine learning algorithms within Python code, through a clean, well-thought-out API. It is built upon NumPy and SciPy.

- An unsupervised learning Example: KMean in scikit-learn

```
# Generate the data X
from sklearn.cluster import KMeans # import the function
est = KMeans(4)                    # init with 4 clusters
est.fit(X)                     # train the kmean model
y_kmeans = est.predict(X)      # cluster X into 4 clusters
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans,
            s=50, cmap='rainbow'); # plot clustering result
```



Try It Now!

Get Your Hands Dirty

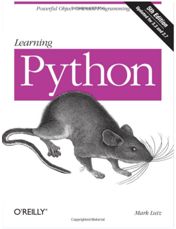
- Go over the lecture slides
- Familiar yourself with Python basics
- Implement any algorithms that you find interesting in Pycharm or IPython notebook
- Try out the examples on the scikit-learn website
- Face completion with a multi-output estimators
- Recognizing hand-written digits
- Comparing different clustering algorithms on toy datasets
- Read through the source code and try to understand the underlying theoretical principles.

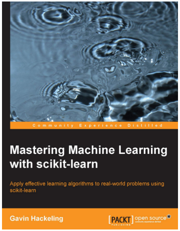
For Further Learning ...

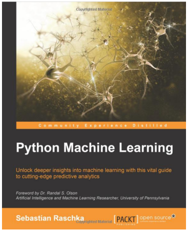
Books & Websites

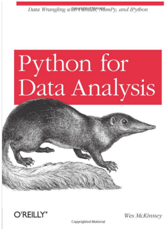
Recommended Books

- Learning Python, 5th Edition
- Mastering Machine Learning With scikit-learn
- Python Machine Learning
- Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython




Community Experience Distilled
Mastering Machine Learning with scikit-learn
Apply effective learning algorithms to real-world problems using scikit-learn
Gavin Hackeling
PACKT open source


Copyrighted Material
Community Experience Distilled
Python Machine Learning
Unlock deeper insights into machine learning with this vital guide to cutting-edge predictive analytics
Foreword by Dr. Randal S. Olson
Artificial Intelligence and Machine Learning Researcher, University of Pennsylvania
Sebastian Raschka
PACKT open source



Recommend Websites

- Python 2.7 Documentation https://docs.python.org/2/
- Python 3.5 Documentation https://docs.python.org/3.5/
- IPython Documentation http://ipython.org/
- NumPy Documentation http://www.numpy.org/
- SciPy Documentation http://www.scipy.org/
- matplotlib Documentation http://matplotlib.org/
- scikit-learn website http://scikit-learn.org/stable/