Python Review

Dr. Huiping Cao

Outline

- History of Python
- Installation of Python and related software
- Python commands
- Python packages
- Python program editors
- Python language

Python history

- A high-level interpreted programming language which began implementation in 1989 and first appeared in 1990.
- It was created by Guido van Rossum, a Dutch programmer, as a hobby project.
- A simple but powerful object-oriented scripting language.
- Current version: see https://www.python.org/
- This course uses Python 3.5.* and above.

Installation of Python

- Installation from official Python website (https://www.python.org)
- Installation using Anaconda Python distribution (https://www.anaconda.com)
  - Platforms: Linux, Windows, and Mac OS X
  - Quickly download 1,500+ Python/R data science packages
  - Manage libraries, dependencies, and environments with Conda
  - Develop and train machine learning and deep learning models with scikit-learn, TensorFlow, and Theano
  - Analyze data with scalability and performance with Dask, NumPy, pandas, and Numba
  - Visualize results with Matplotlib, Bokeh, Datashader, and Holoviews

Installation of Python - Anaconda

- The Anaconda **installer** can be downloaded at https://www.anaconda.com/distribution/#download-section.
  - Please remember to install Python 3.7 version.
  - The installation may take some time
- After successfully installing Anaconda, we can install new Python packages using different commands.
  - conda search <some package>
  - conda install <some package>
  - conda list
  - E.g., conda install numpy scipy scikit-learn –y
- An Anaconda **quick-start guide** is at https://conda.io/docs/test-drive.html

Use Conda to install Python with new version

Step 1: To list the versions of Python that are available to install, in your terminal window or an Anaconda Prompt, run:
$ conda search python

Step 2: To create the new environment for Python 3.9, in your terminal window or an Anaconda Prompt, run:
$ conda create -n py39 python=3.9 anaconda

Step 3. Activate the new environment
$ conda activate py39

Step 4: Verify that the new environment is your current environment.
$conda info –envs
In the environments list that displays, your current environment is highlighted with an asterisk (*).

Or To verify that the current environment uses the new Python version, in your terminal window or an Anaconda Prompt, run:
$python --version

Basic commands

- Check the version of default Python
  - python –V
  - python --version
  - conda --version

```
(base) hcao9c52:~ huipingcao$ python --version
Python 3.7.4
(base) hcao9c52:~ huipingcao$ conda --version
conda 4.7.12
```

- Run Python code
  - python <python file name>
  - E.g., python test.py
  - Run Python code with arguments (watch video)

Python packages & libraries (1)

- Libraries for scientific computing such as NumPy and SciPy
  - **NumPy** 1.12.1: operate multidimensional arrays
  - **SciPy** 0.19.0: a fundamental library for mathematics, science, and engineering.
- Performance of interpreted languages is inferior
- But NumPy and SciPy build upon lower-level C and Fortran subroutines

Python packages & libraries (2)

- **Pandas** 0.20.1: a library built on top of NumPy that provides additional higher-level data manipulation tools. Such tools make working with tabular data more convenient.
- **scikit-learn** 0.18.1: Simple and efficient python tools for data mining and data analysis.
- **Visualization**
  - **Matplotlib** 2.0.2: visualize quantitative data.
  - **Seaborn**: statistical data visualization. Seaborn library is a Python library for drawing statistical plots based on Matplotlib.
- **TensorFlow, PyTorch,** …

Python Editor/IDE

- Online
  - Google Colab: Watch video
  - repl.it
- Many other IDE
  - https://www.programiz.com/python-programming/ide
  - PyCharm (community edition)
  - Spider
  - Sublime text (https://www.sublimetext.com)
  - Etc
- Local Jupyter Notebook
  - Notebook is very good for **interactively** running your code. Debug! Need to understand the logic of Cell, Kernel. More information: http://jupyter.org/try)
- How to use Jupyter Notebook
  - Quick guide: see the note file for python background.
  - Or, directly use online sources

Python language

- Python
- NumPy
- Pandas
- Matplotlib

Python – Basic data types

- Python provides *type* function to see the type of data
- Numbers (e.g., integer, float
  - >>> type(-5)
    <type ’int’>
- Strings
  - >>> type("This is a string")
  - <type ’str’>
- Boolean values: *True* or *False*.
- Container types

Python containers - lists

- Python includes several built-in container types: lists, dictionaries, sets, and tuples.
- A list is the Python equivalent of an array.
- A list is **resizeable** and can contain elements of **different** types.

```
xs = [3, 1, 2]           # Create a list
print("xs[0]=", xs[0]) # prints “xs[0]= 3”
print(xs, xs[2])         # Prints "[3, 1, 2] 2"
print(xs[-1])            # Negative indices count from the end of the list; prints "2"
xs[2] = 'foo'            # Lists can contain elements of different types
print(xs)                # Prints "[3, 1, 'foo']"
xs.append('bar')  # Add a new element to the end of the list
print(xs)                # Prints "[3, 1, 'foo', 'bar']"
x = xs.pop()             # Remove and return the last element of the list
print(x, xs)             # Prints "bar [3, 1, 'foo']"
```

Python containers – list slicing

- **Slicing**: Python provides concise syntax to access sublists; this is known as slicing.

```
nums = list(range(5))   # range is a built-in function that creates a list of integers
print(nums)             # Prints "[0, 1, 2, 3, 4]"
print(nums[2:4])        # Get a slice from index 2 to 4 (exclusive); prints "[2, 3]"
print(nums[2:])         # Get a slice from index 2 to the end; prints "[2, 3, 4]"
print(nums[:2])         # Get a slice from the start to index 2 (exclusive); prints "[0, 1]"
print(nums[:])          # Get a slice of the whole list; prints "[0, 1, 2, 3, 4]"
print(nums[:-1])        # Slice indices can be negative; prints "[0, 1, 2, 3]"
nums[2:4] = [8, 9]      # Assign a new sublist to a slice
print(nums)             # Prints "[0, 1, 8, 9, 4]"
```

Python containers

- A **dictionary** stores (key, value) pairs, similar to a Map in Java or an object in Javascript.
- A **set** is an unordered collection of distinct elements.
- A **tuple** is an ordered list of values. A tuple is in many ways similar to a list. Differences are
  - Tuples can be used as keys in dictionaries and as elements of sets, while lists cannot.
  - Lists are enclosed in square brackets while tuples are enclosed in parentheses. E.g., x= (1,2,3)
- More explanations: see http://cs231n.github.io/python-numpy-tutorial/#python-containers

Basic operators

- **Arithmetic** operators: + for addition, − for subtraction, $*$ for multiplication, and / for division. Meanwhile, // for integer division, and $**$ for exponentiation
  - >>> 25/3
  - 8.333333333333334
  - >>> 25//3
  - 8
- **Comparison** operators: <, >, <=, >=, ==, !=, <>
  - Result is a Boolean value (True or False)

Basic syntax

- Comment
  - Single-line comments are created simply by beginning a line with the hash (#) character.
  - Comments that span multiple lines – used to explain things in more detail – are created by adding a delimiter (""") on each end of the comment.

```
""" This would be a multiline comment
in Python that spans several lines and
describes your code, your day, or anything you want it to
"""
```

- Variables
  - A variable is created at the moment you first assign a value to it.

Examples: Control statement, loop, functions

```
animals = ['cat', 'dog', 'monkey']
for animal in animals:
    print(animal)
```

```
def sign(x):
    if x > 0:
        return “positive”
    elif x < 0:
        return “negative”
    else:
        return “zero”

for x in [-1, 0, 1]:
    print(sign(x))
```

Basic syntax

- Control statements
- Loop: for, while
- Function definition

Example of control statement

if condition1**:**
    action1
elif condition2**:**
    action2
else:
    action3

For loop

for *item* **in** *list*:
    action

While loop

while(condition):
    action

Example of function definition

def f(x):
    return x*x

Examples: Class

```
def hello(name, loud=False):
    if loud:
        print('HELLO, %s!' % name.upper())
    else:
        print('Hello, %s' % name)

hello('Bob') # Prints "Hello, Bob"
hello('Fred', loud=True)
```

Python class

- Everything is an object. I.e., classes and types are also objects.
- Attribute**:** The data values associated with an object.
- Method**:** The functions that are associated with an object.

Define a class

class *class-name*(object):

#constructor function

def _init_(self, *parameters*):

constructor statements

def *user-function*(self, *parameters*):

user-defined statements

Use a class

class-name class-object

class-object.user-function(parameters)

File reading and writing

- Import *os* package to the environment.
- >>>import os

Reading

with open(file-name**) as** f:
    for line **in f:**
        print(line**)**
        process-statement

Writing

target **= open(**file-name**, 'w'**)
target**.write(**"test"**)**
target**.close()**

NumPy package

- **NumPy** represents Numerical Python.
- It is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
- It provides the operations to **homogeneous** multidimensional arrays.
- >>>import numpy as np

NumPy Array

- A numpy array is a grid of values, all of the **same type**, and is indexed by a tuple of **nonnegative** integers.
- The number of dimensions is the **rank** of the array.
- The **shape** of an array is a tuple of integers giving the size of the array along each dimension
  - **a.shape** shows the dimensions of array a
  - >>>a.shape
  - **a[i,j,k]** access array a's specific element
  - >>>a[1,1,1] = 4

Example of NumPy Array

```
1. import numpy as np

2. a = np.array([1, 2, 3])   # Create a rank 1 array
3. print(type(a))            # Prints "<class 'numpy.ndarray’>”
4. print(a.shape)            # Prints "(3,)"
5. print(a[0], a[1], a[2])   # Prints "1 2 3"
6. a[0] = 5                  # Change an element of the array
7. print(a)                  # Prints "[5, 2, 3]"

8. b = np.array([[1,2,3],[4,5,6]]) # Create a rank 2 array
9. print(b.shape)                   # Prints "(2, 3)"
10. print(b[0, 0], b[0, 1], b[1, 0])  # Prints "1 2 4"
```

Example: rank/ndim and shape

```
A3 = np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
print(np.ndim(A3))
print(A3.shape)

Output:
3
(2, 2, 3)

python3.7:
 VisibleDeprecationWarning: `rank` is deprecated; use
the `ndim` attribute or function instead. To find the rank
of a matrix see `numpy.linalg.matrix_rank`.
```

NumPy Array Indexing

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

numpy.arange

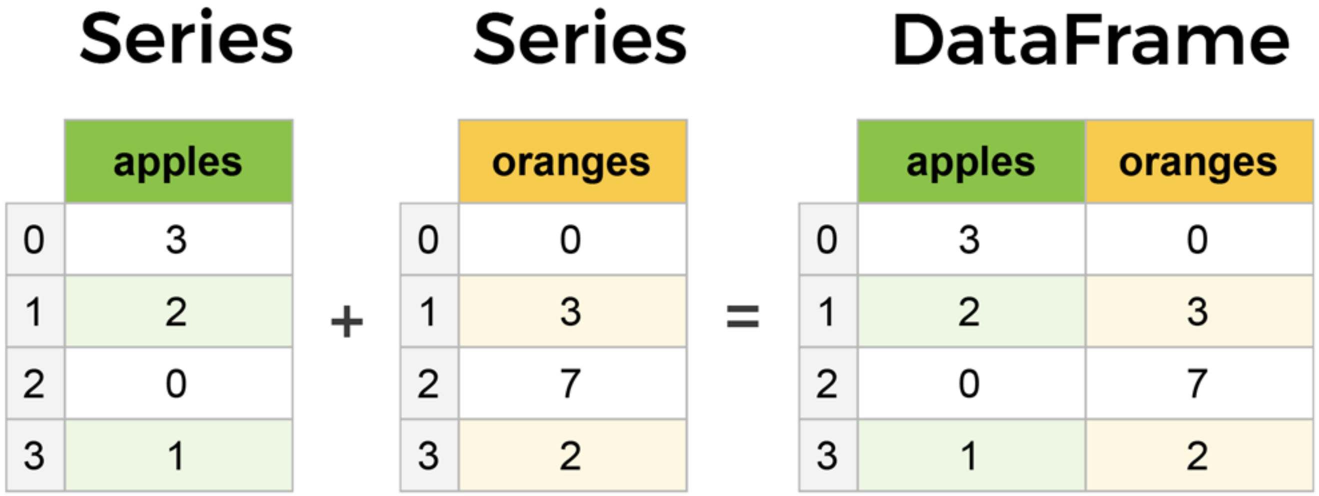
- **numpy.arange([start,]stop, [step,]dtype=None)**
- Return evenly spaced values within a given interval.
- Values are generated within the half-open interval [start, stop).
- For integer arguments the function is equivalent to the Python built-in **range** function, but returns an ndarray rather than a list.

```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([ 0.,  1.,  2.])
>>> np.arange(3,7)
array([3, 4, 5, 6])
>>> np.arange(3,7,2)
array([3, 5])
```

Pandas package

- Pandas is built on top of the **NumPy** package, meaning a lot of the structure of NumPy is used or replicated in Pandas.
- Data in pandas is often used to feed statistical analysis in **SciPy**, plotting functions from **Matplotlib**, and machine learning algorithms in **scikit-learn**.
- The primary two components of pandas are the **Series** and **DataFrame**.
  - A Series is essentially a column,
  - A DataFrame is a multi-dimensional table made up of a collection of Series.

Example


Series
apples
0 3
1 2
2 0
3 1
+
Series
oranges
0 0
1 3
2 7
3 2
=
DataFrame
apples oranges
0 3 0
1 2 3
2 0 7
3 1 2

Pandas read in data

- Reading data from CSVs
- import pandas as pd
- read_csv: fast and versatile, a recommended tool for working with tabular data stored in a plaintext format. get a data frame.
  df = pd.read_csv("../data/housing.data.txt", header=None)
- Data frame *df*
  - df.columns shows column names

Other useful information

- Getting info about your data df.info()
- Another fast and useful attribute is **df.shape**, which gets dimensions of a data frame and outputs just a tuple of (rows, columns).
  - E.g., for the housing dataset, df.shape outputs (506, 14)
- Viewing your data: df.head(), df.tail()

```
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10 entries, 0 to 9
Data columns (total 5 columns):
0    150 non-null float64
1    150 non-null float64
2    150 non-null float64
3    150 non-null float64
4    150 non-null object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
>>> df.shape
(10, 5)
```

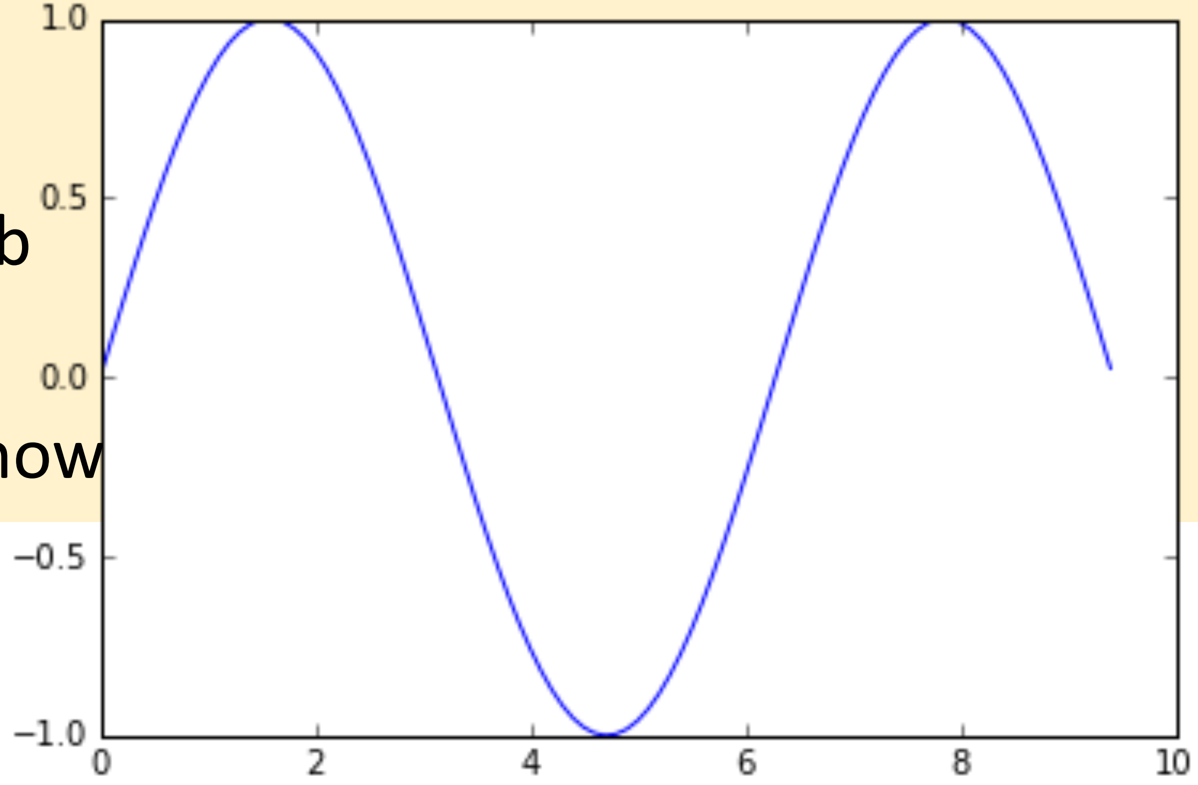
Matplotlib package

- Matplotlib is a plotting library. It has many different modules.
- Module matplotlib.pyplot.

Example - basic plotting

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

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show() to make graphics appear.
```

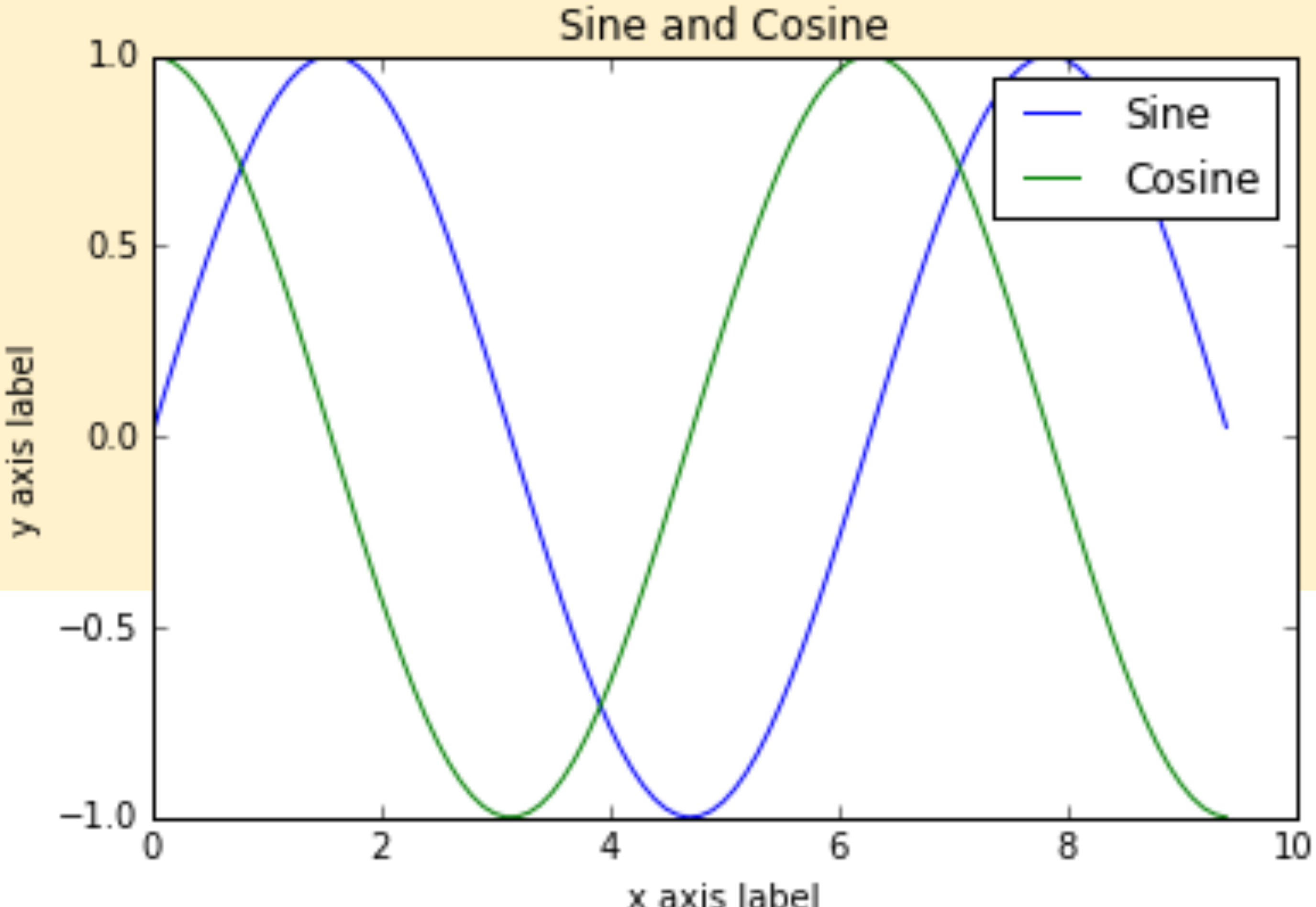
```python
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
plt.show()  # You must call plt.show
```



Example - plotting two lines

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```

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

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



More resources

- Textbook source code: https://github.com/rasbt/python-machine-learning-book-3rd-edition
- Conda: managing Python: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-python.html
- Pandas: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html