University of Pittsburgh

ECE 2195: Special Topics – Computers Machine Learning

Introduction to Python

Mai Abdelhakim, PhD
ECE Department
Swanson School of Engineering
University of Pittsburgh
maia@pitt.edu

Python

- General-purpose programming, interpreted language, popular for many data science applications

- Can Interact with code through terminal

- Installation recommended: **Anaconda**
  https://www.anaconda.com/download/
  Choose Python 3.6

  - Includes Spyder: Python development environment
  - Jupyter notebook

Jupyter notebook

- **Jupyter** notebook (http://jupyter.org): interactive web application for coding
  - Incorporate code, text and images in one file

- Supports many programing languages, e.g. Python, R, Julia
  - By separate Kernels

- Start it from anaconda or from the terminal (type jupyter notebook) or from anaconda

Python Libraries

- **Numpy** (http://numpy.org): **multidimensional data** storage and computation
- **SCiPy** (http://scipy.org): **numerical** tools, e.g. interpolation
- **Pandas** (http://pandas.pydata.org): **Dataframe objects**, and tools to manipulate and filter data
- **Matplotlib** (http://matplotlib.org): **visualization**
- **Seaborn**: https://seaborn.pydata.org/
  - Python visualization library based on matplotlib.
- **Scikit-Learn** (http://scikit-learn.org): common **machine learning algorithms**

Python Syntax

- Command line or scripts (.py files), notebooks (.ipynb)
- Comments are indicated by #
  - E.g. # this line is a comment
- The end of line is end of statement (no ‘;’ needed)
  - E.g. x=5
- Semicolon (;) can be used to separate statements on the same line
  - E.g x=5; y=8
- Printing: built-in print function
  - print(x)
  - print("The value of x is equal to", x)

Python Syntax

- Variables have no attached type information
  - No need to define the type of the variable

- Example:
  X=10
  type(x) ➔ output: int

- w='Hello'
  - get type of w?

```
In [11]: x= 10; type(x)
Out[11]: int

In [10]: y= 5.3; type(y)
Out[10]: float
```

Operations

Arithmetic operations

Operator	Name	Description
a + b	Addition	Sum of a and b
a - b	Subtraction	Difference of a and b
a * b	Multiplication	Product of a and b
a / b	True division	Quotient of a and b
a // b	Floor division	Quotient of a and b, removing fractional parts
a % b	Modulus	Remainder after division of a by b
a ** b	Exponentiation	a raised to the power of b
-a	Negation	The negative of a

A whirlwind Tour of Python, by Jake VanderPlas (available online)

Comparison operations, return true/false

Operation	Description
a == b	a equal to b
a != b	a not equal to b
a < b	a less than b
a > b	a greater than b
a <= b	a less than or equal to b
a >= b	a greater than or equal to b

Everything is an Object

- Objects has attributes and methods accessed by **dot operator**

- Example:
  - List.append(4)
  - x.real (real part of a number), y.imag (imaginary part)

Indentation

- Whitespace at the beginning has a meaning in Python
- **Code block**: statements that should be treated as a unit
- Code blocks are **preceded by a colon ":"**
- Amount of indenting must be consistent in the code (typically 4 spaces)

```
if statement:
        #code block, code without indentation will not be part of it

while condition:
        #code block
```

Conditional Statements

- Allow a code block to execute only if a condition is satisfied

```
if x==0:
    print (‘x is equal to 0’)
elif x>0:
    print(‘x is greater than zero’)
else:
    print(‘x is not greater than or equal to zero’)
```

Boolean Operations – Combine Boolean Values

- Combine Boolean values using : **and**, **or** and **not**

  - **Example:**

```
if (x < =9) and (y > 2):
    #code block
If (x>10) or (x%2==0):
```

Loops

- For loops: for repeating a code block a number of times
  for N **in** [0,1,2,3]**:** → This can be any list
      print('N is equal to', N)

- **range(n)** is an **object** that **generates sequence** from **0 to n-1**, and is often used in for loops

```
for N in range(4):
    print('N is equal to', N)
```

```
N is equal to 0
N is equal to 1
N is equal to 2
N is equal to 3
```

Loops

- While loop: condition is checked in each iteration
  while condition:
  print('this code block will be executed when condition is satisfied')

- Break the loop entirely using ***break***

- *Example:*
  x=2
  while x<5**:**
  print(x)
  x=x+1

- https://wiki.python.org/moin/WhileLoop

Lists

- List definitions
- Length of list & sum functions
- Append element
- Append another list
- List with elements of different types

```
L=[1,2,3]; type(L)
```

```
list
```

```
len(L)
```

```
3
```

```
sum(L)
```

```
6
```

```
L.append(4)
```

```
print(L)
```

```
[1, 2, 3, 4]
```

```
L+[5,6,7]
```

```
[1, 2, 3, 4, 5, 6, 7]
```

```
L2=[1,'two',3.14,[4,5,6]]
print(L2)
```

```
[1, 'two', 3.14, [4, 5, 6]]
```

```
print(len(L2[3]))
```

```
3
```

Lists

- Indexing:
  - **Starts from Zero**: e.g. L[0], L[1],…
  - You can access end of list can be through negative sign starting -1
    - L=[1,2,3]; Print(L[-1]) => Output: 3
- More functions: https://docs.python.org/3/tutorial/datastructures.html

Lists – Access specific elements

- Access multiple elements:
  - **List [Start : End : Step]**
    - Start counts from 0
    - End is index of the last element that will **not** be included
  - From the **third** to the **fourth** element: L[**2**:4]
    - Element with index 4 (5th element in the list) will not be included

```
L=[10,12,13,14,15,16,17,18]

print(L[2:4])
```

```
[13, 14]
```

  - If "End" is not specified, default is end of the list
    - L[2:] is the same as L[2:len(L)]
  - If "Start" is left out, then zero is assumed
    - L[:3] is the same as L[0:3]

Lists – Access specific elements with step size

- Step size:
  - Entire list with step size of 2: L[::2]
  - Negative step is possible: counts back from the end
    - L=[1,2,3]; L=[::-1] => output: (3,2,1)

```
L=[10,12,13,14,15,16,17,18]
```

```
print(L[::2])
```

```
[10, 13, 15, 17]
```

```
print(L[::-1]) # negative step
```

```
[18, 17, 16, 15, 14, 13, 12, 10]
```

Dictionaries

- https://docs.python.org/3/tutorial/datastructures.html#dictionaries
- Flexible mapping of keys to values
- Can be created by comma-separated list of **{key:value}** (using braces)

```
Grade={'Alex': 10, 'Peter':15, 'James': 20}
print(Grade)
```

```
{'Alex': 10, 'Peter': 15, 'James': 20}
```

- **Index** is through a valid **key**
  - print(Grade['Peter'])
  - **Order is not important**

Dictionaries: adding element

```
Grade={'Alex': 10, 'Peter':15, 'James': 20}
print(Grade)
```

```
{'Alex': 10, 'Peter': 15, 'James': 20}
```

- Add new element using a new key

```
# add element by a new key
Grade['Jeff']=16
print(Grade)
```

```
{'Alex': 10, 'Peter': 15, 'James': 20, 'Jeff': 16}
```

- Find keys in a dictionary: **Grade.keys()**
- Find values with **Grade.values()**

```
print(Grade.keys())
print(Grade.values())
```

```
dict_keys(['Alex', 'Peter', 'James', 'Jeff'])
dict_values([10, 15, 20, 16])
```

Numpy: Numerical Python – Multidimensional arrays

- Multidimensional arrays storage and efficient manipulation
- https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
- Example: starts by **import** numpy

```
In [1]: import numpy
        x=numpy.array( [ [1,2,3] , [4,5,6] ] )
        print(x)

        [[1 2 3]
         [4 5 6]]
```

- Efficient element-wise operation on data
  - E.g. x*2 will multiply each element in the array by 2

Numpy: Numerical Python

```
In [1]: import numpy
        x=numpy.array( [ [1,2,3] , [4,5,6] ] )
        print(x)
```

```
[[1 2 3]
 [4 5 6]]
```

```
In [2]: type(x)
Out[2]: numpy.ndarray

In [3]: x.shape # get dimension of x
Out[3]: (2, 3)

In [4]: print(x.T) # transpose of x
```

```
[[1 4]
 [2 5]
 [3 6]]
```

```
In [6]: x=x.reshape(3,2)
        print(x)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

Pandas

- Enabled labeled interface for multidimensional in the form of DataFrame object
- Labeled column-oriented data
- Example: starts with import pandas as pd

```
                # “as pd” gives pandas a short name
dataFrame1=pd.DataFrame({‘labelCol1’:[‘a’,’b’,’c’,’d’], ‘labelCol2’: [1,2,3,4]})
```

```
import pandas as pd
dataFrame1=pd.DataFrame({'labelCol1':['a','b','c','d'], 'labelCol2': [1,2,3,4]})
print(dataFrame1)
```

```
  labelCol1  labelCol2
0         a          1
1         b          2
2         c          3
3         d          4
```

Pandas

- dataFrame1=pd.DataFrame({‘labelCol1’:[‘a’,’b’,’c’,’d’], ‘labelCol2’: [1,2,3,4]})

- Efficient way to do operations on each column independently

- Two ways to access column:
  - **dataFrame1 [‘labelCol2’]**
  - **dataFrame1. labelCol2**

- Example: sum the **second** column in dataFrame1 (which has label ‘labelCol2’):

  dataFrame1 [‘labelCol2’].sum() #this sums element of elements with label ‘labelCol2’

Pandas – multidimensional data with Labeled axes

- Can put numpy array into Pandas DataFrame
- Example:

```
import pandas as pd
import numpy as np
x=np.array([[1,2,3],[4,5,6]])
colLabels=['Col1','Col2','Col3']
myDataFrame=pd.DataFrame(x, columns= colLabels )
```

- Column and row labels

```
RowLabel=['r1','r2']
Mydataframe = pd.DataFrame(x, index= RowLabel, columns= colLabels )
Mydataframe.loc['r1','Col2']
```

- Many functions: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html

Matplotlib

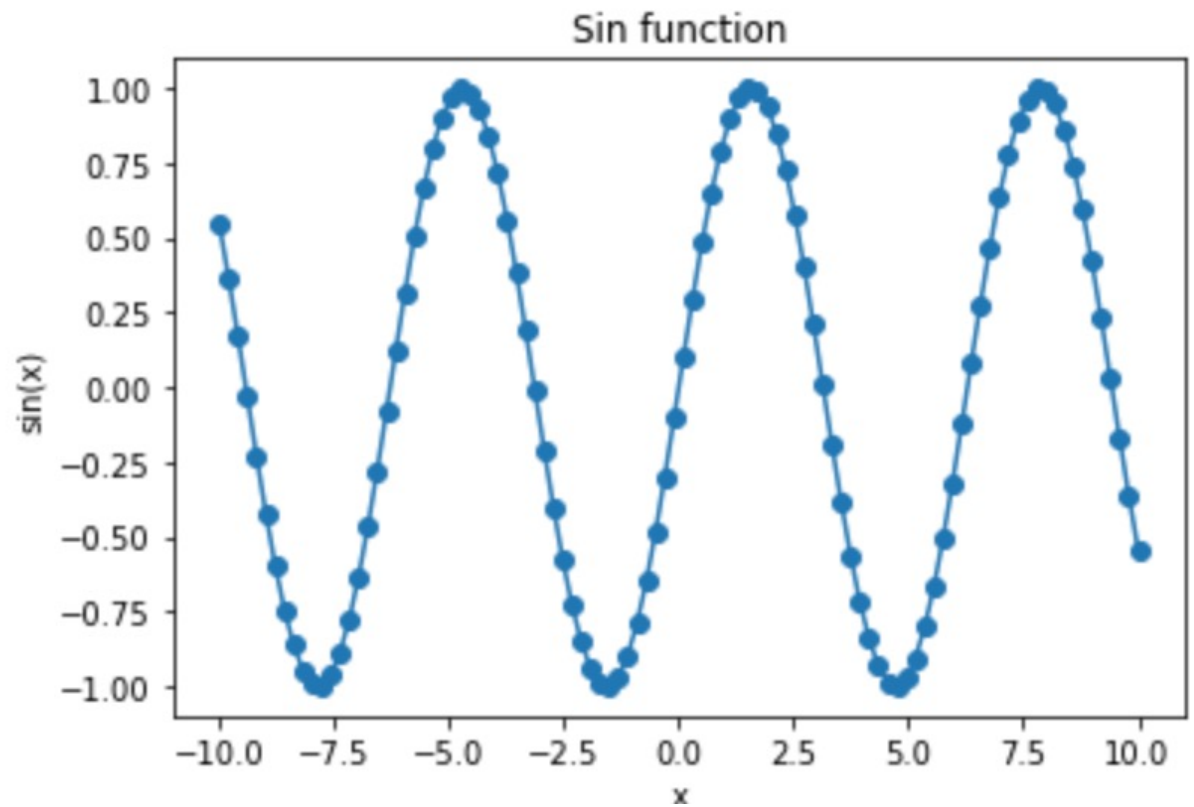
- Visualization tool - create plots
- View examples: http://matplotlib.org/examples/index.html
- When using Jupyter notebook, at the beginning type: %matplotlib inline
- Need to import matplotlib.pyplot as plt

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

x=np.linspace(-10,10,100) #100 is the number of samples in the plot
y=np.sin(x)
plt.plot(x,y,marker='o')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Sin function')
```

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

x=np.linspace(-10,10,100) #100 is the number of samples in the plot
y=np.sin(x)
plt.plot(x,y,marker='o')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Sin function')
```



Scikit-Learn

- Open source project used in academia and industry
- Number of machine learning algorithms
- http://scikit-learn.org/stable/documentation

Exercise 1 – Q1

1. Create a dictionary of score of each team, with keys:values as follows

   Team1: 4,
   Team2: 3,
   Team3: 5,
   Team4: 2,

2. Add to the dictionary ‘Team5’ who has a score ‘5’
3. PRINT all the keys of the dictionary using the keys() method
4. Find the length of VALUES in the dictionary and print it
5. Get the average score of the teams and print it

Exercise 1 – Q2

1. Generate a numpy array with 2 columns, where first column contains numbers from 0 to 5 and second column is [0, 1, 4, 9, 16, 25]
2. Check the shape of your array. It should be 6x2.
3. Put the array into a data frame, with column labels ‘x’ and ‘y’
4. Plot ‘x’ on x-axis versus ‘y’ on y-axis
5. Get average of elements in second column of the data frame