Basic Python (numpy+matplotlib)

Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

LOAD PACKAGES

```
import numpy as np
print ("Loading package(s)")
```

```
Loading package(s)
```

PRINT function usages

```
print ("Hello, world")

# THERE ARE THREE POPULAR TYPES
# 1. INTEGER
x = 3;
print ("Integer: %01d, %02d, %03d, %04d, %05d"
       % (x, x, x, x, x))
# 2. FLOAT
x = 123.456;
print ("Float: %.0f, %.1f, %.2f, %1.2f, %2.2f"
       % (x, x, x, x, x))
# 3. STRING
x = "Hello, world"
print ("String: [%s], [%3s], [%20s]"
       % (x, x, x))
```

```
Hello, world
Integer: 3, 03, 003, 0003, 00003
Float: 123, 123.5, 123.46, 123.46, 123.46
String: [Hello, world], [Hello, world], [        Hello, world]
```

FOR + IF/ELSE

```
dlmethods = ["ANN", "MLP", "CNN", "RNN", "DAE"];
for alg in dlmethods:
    if alg in ["ANN", "MLP", "CNN"]:
        print ("%s is a feed-forward network." % (alg))
    elif alg in ["RNN"]:
        print ("%s is a recurrent network." % (alg))
    else:
        print ("%s is an unsupervised method." % (alg))
```

```
ANN is a feed-forward network.
MLP is a feed-forward network.
CNN is a feed-forward network.
RNN is a recurrent network.
DAE is an unsupervised method.
```

FOR LOOP WITH INDEX

```
print("FOR loop with index.")
for i, alg in zip(range(len(dlmethods)), dlmethods):
    if alg in ["ANN", "MLP", "CNN"]:
        print ("[%d/%d] %s is a feed-forward network."
               % (i, len(dlmethods), alg))
    elif alg in ["RNN"]:
        print ("[%d/%d] %s is a recurrent network."
               % (i, len(dlmethods), alg))
    else:
        print ("[%d/%d] %s is an unsupervised method."
               % (i, len(dlmethods), alg))
```

```
FOR loop with index.
[0/5] ANN is a feed-forward network.
[1/5] MLP is a feed-forward network.
[2/5] CNN is a feed-forward network.
[3/5] RNN is a recurrent network.
[4/5] DAE is an unsupervised method.
```

FUNCTIONS

```
def sum(a, b):
    return a+b
X = 10.
Y = 20.
# USAGE
print ("%.1f + %.1f = %.1f" % (X, Y, sum(X, Y)))
```

```
10.0 + 20.0 = 30.0
```

STRING

```
head = "Deep learning"
body = "very "
tail = "HARD."
print (head + " is " + body + tail)

# REPEAT
print (head + " is " + body*3 + tail)
print (head + " is " + body*10 + tail)

# IT IS USED IN THIS WAY
print ("\n" + "="*50)
print (" "*15 + "It is used in this way")
print ("="*50 + "\n")

# INDEXING
x = "Hello, world"
for i in range(len(x)):
    print ("Index: [%02d/%02d] Char: %s" % (i, len(x), x[i]))
```

```
Deep learning is very HARD.
Deep learning is very very very HARD.
Deep learning is very very very very very very very very very

==================================================
               It is used in this way
==================================================

Index: [00/12] Char: H
Index: [01/12] Char: e
Index: [02/12] Char: l
Index: [03/12] Char: l
Index: [04/12] Char: o
Index: [05/12] Char: ,
Index: [06/12] Char:
Index: [07/12] Char: w
Index: [08/12] Char: o
Index: [09/12] Char: r
Index: [10/12] Char: l
Index: [11/12] Char: d
```

LIST

```
a = []
b = [1, 2, 3]
c = ["Hello", ",", "world"]
d = [1, 2, 3, "x", "y", "z"]
x = []
print (x)
x.append('a')
print (x)
x.append(123)
print (x)
x.append(["a", "b"])
print (x)
print ("Length of x is %d." % (len(x)))
for i in range(len(x)):
    print ("[%02d/%02d] %s." % (i, len(x), x[i]))
```

```
[]
['a']
['a', 123]
['a', 123, ['a', 'b']]
Length of x is 3.
[00/03] a.
[01/03] 123.
[02/03] ['a', 'b'].
```

DICTIONARY

```
dic = dict()
dic["name"] = "Sungjoon"
dic["age"] = 31
dic["job"] = "Ph.D. Candidate"
print (dic)
```

```
{'name': 'Sungjoon', 'age': 31, 'job': 'Ph.D. Candidate'}
```

CLASS

```
class Greeter:
    # CONSTRUCTOR
    def __init__(self, name):
        self.name = name
    # METHOD
    def greet(self, loud=False):
        if loud:
            print ('HELLO, %s!' % self.name.upper())
        else:
            print ('Hello, %s' % self.name)

g = Greeter('Fred')  # Construct an instance of the Greeter class
g.greet()            # Call an instance method; prints "Hello, Fred"
g.greet(loud=True)   # Call an instance method; prints "HELLO, FRED!"
```

```
Hello, Fred
HELLO, FRED!
```

NUMPY

```
def print_np(x):
    print ("Type is %s" % (type(x)))
    print ("Shape is %s" % (x.shape,))
    print ("Values are: \n%s" % (x))
```

RANK 1 ARRAY

```
x = np.array([1, 2, 3]) # RANK 1 ARRAY
print_np(x)
x[0] = 5
print_np(x)
```

```
Type is <class 'numpy.ndarray'>
Shape is (3,)
Values are:
[1 2 3]
Type is <class 'numpy.ndarray'>
Shape is (3,)
Values are:
[5 2 3]
```

RANK 2 ARRAY

```
y = np.array([[1,2,3], [4,5,6]])   # Create a rank 2 array
print_np(y)
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 3)
Values are:
[[1 2 3]
 [4 5 6]]
```

RANK 2 ZEROS

```
a = np.zeros((2, 2))  # Create an array of all zeros
print_np(a)
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[ 0.  0.]
 [ 0.  0.]]
```

RANK 2 ONES

```
b = np.ones((3, 2))   # Create an array of all ones
print_np(b)
```

```
Type is <class 'numpy.ndarray'>
Shape is (3, 2)
Values are:
[[ 1.  1.]
 [ 1.  1.]
 [ 1.  1.]]
```

RANK 2 EYES (IDENTITY)

```
c = np.eye(3, 3)     # Identity matrix
print_np(c)
```

```
Type is <class 'numpy.ndarray'>
Shape is (3, 3)
Values are:
[[ 1.  0.  0.]
 [ 0.  1.  0.]
 [ 0.  0.  1.]]
```

RANDOM MATRIX (UNIFORM)

```
d = np.random.random((4, 4))     # Sample from uniform distribution
print_np(d)
```

```
Type is <class 'numpy.ndarray'>
Shape is (4, 4)
Values are:
[[ 0.02697245  0.99072777  0.85069875  0.97311501]
 [ 0.06379829  0.72114197  0.53863635  0.23904971]
 [ 0.60742283  0.0590545   0.82516577  0.00290694]
 [ 0.16129955  0.96104575  0.04787153  0.45486264]]
```

RANDOM MATRIX (GAUSSIAN)

```
e = np.random.randn(4, 4)     # Sample for zero-mean unit Gaussain
print_np(e)
```

```
Type is <class 'numpy.ndarray'>
Shape is (4, 4)
Values are:
[[-1.57818909  1.0185437   1.73242661 -0.090831  ]
 [ 0.53168521 -1.35948676 -0.2321504  -0.57783035]
 [-0.10958057  1.71371358  1.28865437 -1.12649039]
 [ 1.73988761  0.95571367  1.90299326 -0.27793377]]
```

ARRAY INDEXING

```
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
print_np(a)
```

```
Type is <class 'numpy.ndarray'>
Shape is (3, 4)
Values are:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

```
b = a[:2, 1:3]
print_np(b)
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[2 3]
 [6 7]]
```

GET NTH ROW(S) FROM MATRIX

```
row_r1 = a[1, :]    # Rank 1 view of the second row of a
print_np(row_r1)
```

```
Type is <class 'numpy.ndarray'>
Shape is (4,)
Values are:
[5 6 7 8]
```

```
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print_np(row_r2)
```

```
Type is <class 'numpy.ndarray'>
Shape is (1, 4)
Values are:
[[5 6 7 8]]
```

```
row_r3 = a[[1], :]  # Rank 2 view of the second row of a
print_np(row_r3)
```

```
Type is <class 'numpy.ndarray'>
Shape is (1, 4)
Values are:
[[5 6 7 8]]
```

ARRAY MATH

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# ELEMENTWISE SUM
print_np(x + y)
print_np(np.add(x, y))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[  6.   8.]
 [ 10.  12.]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[  6.   8.]
 [ 10.  12.]]
```

```
# ELEMENTWISE DIFFERENCE
print_np(x - y)
print_np(np.subtract(x, y))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[-4. -4.]
 [-4. -4.]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[-4. -4.]
 [-4. -4.]]
```

```
# ELEMENTWISE PRODUCT
print_np(x * y)
print_np(np.multiply(x, y))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[  5.  12.]
 [ 21.  32.]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[  5.  12.]
 [ 21.  32.]]
```

```
# ELEMENTWISE DIVISION
print_np(x / y)
print_np(np.divide(x, y))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[ 0.2         0.33333333]
 [ 0.42857143  0.5       ]]
```

```
# ELEMENTWISE SQUARE ROOT
print_np(np.sqrt(x))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[ 1.          1.41421356]
 [ 1.73205081  2.        ]]
```

MATRIX OPERATION ¶

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])
v = np.array([9,10])
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[1 2]
 [3 4]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[5 6]
 [7 8]]
Type is <class 'numpy.ndarray'>
Shape is (2,)
Values are:
[ 9 10]
Type is <class 'numpy.ndarray'>
Shape is (2,)
Values are:
[11 12]
```

VECTORS-VECTORS OPERATION

```
print_np (v.dot(w))
print_np (np.dot(v, w))
```

```
Type is <class 'numpy.int64'>
Shape is ()
Values are:
219
Type is <class 'numpy.int64'>
Shape is ()
Values are:
219
```

MATRIX-VECTOR OPERATION

```
print_np (x.dot(v))
print_np (np.dot(x, v))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2,)
Values are:
[29 67]
Type is <class 'numpy.ndarray'>
Shape is (2,)
Values are:
[29 67]
```

MATRIX-MATRIX OPERATION

```
print_np (x.dot(y))
print_np (np.dot(x, y))
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[19 22]
 [43 50]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[19 22]
 [43 50]]
```

TRANSPOSE

```
x = np.array([[1,2],[3,4]])
print_np (x)
print_np (x.T)
```

```
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[1 2]
 [3 4]]
Type is <class 'numpy.ndarray'>
Shape is (2, 2)
Values are:
[[1 3]
 [2 4]]
```

SUM

```
print_np (np.sum(x))  # Compute sum of all elements
print_np (np.sum(x, axis=0))  # Compute sum of each column
print_np (np.sum(x, axis=1))  # Compute sum of each row
```

```
Type is <class 'numpy.int64'>
Shape is ()
Values are:
10
Type is <class 'numpy.ndarray'>
Shape is (2,)
Values are:
[4 6]
Type is <class 'numpy.ndarray'>
Shape is (2,)
Values are:
[3 7]
```

ZEROS-LIKE

```
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
y = np.zeros_like(x)
print_np (x)
print_np (y)
```

```
Type is <class 'numpy.ndarray'>
Shape is (4, 3)
Values are:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Type is <class 'numpy.ndarray'>
Shape is (4, 3)
Values are:
[[0 0 0]
 [0 0 0]
 [0 0 0]
 [0 0 0]]
```

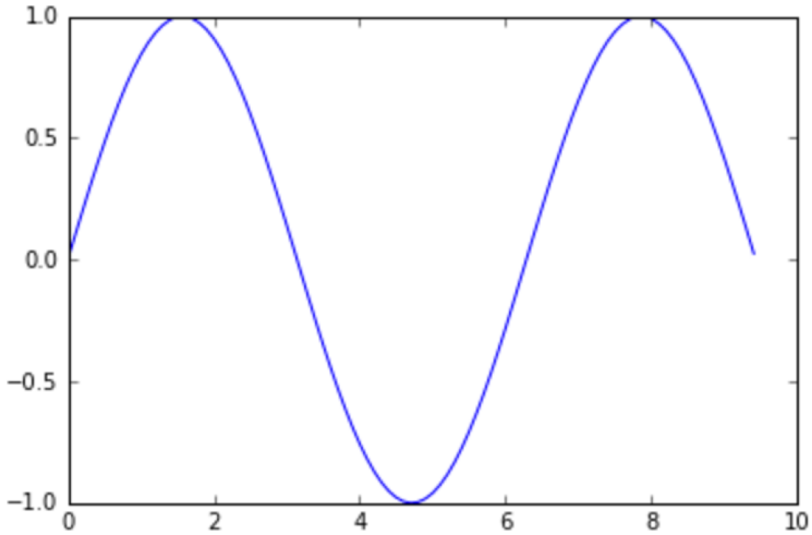
MATPLOTLIB

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

```
# 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)
```

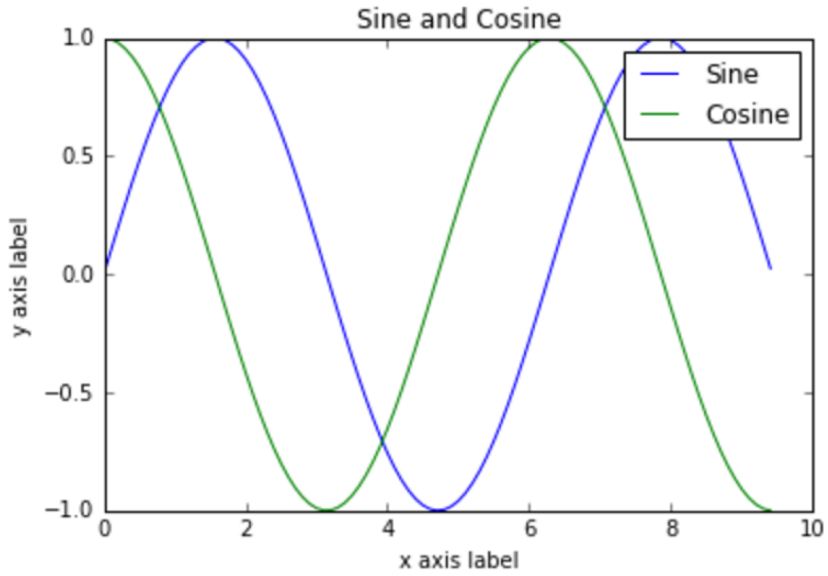
```
[<matplotlib.lines.Line2D at 0x7f1750cb5278>]
```



PLOT TWO GRAPHS IN ONE

```
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'])

# Show the figure.
plt.show()
```



SUBPLOT

```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

plt.subplot(2, 1, 1)
plt.plot(x, y_sin)
plt.title('Sine')

plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

plt.show()
```