A Basic introduction to jupyter notebook and some libraries useful for Datascience and Machine learning

Here are good links to explain how to install Anaconda in various operating systems: 1) Ubuntu 2) Mac 3) Windows

Once anaconda is installed, just run the command "jupyter notebook" from the terminal/anaconda shell to get the jupyter notebook interface open in your browser. Here you can create new notebooks and save them.

This tutorial is based on the tutorial by Justin Jhonson (Stanford 231n).

Distributions of the jupyter notebook

Jupyter notebook is not included with the basic python installation. There are several ways to install it as there are many distributions offering this. The installation linked above is the Anaconda installation. But Anaconda is only one of the distributions. It is recommended because Anaconda comes with pre installed scientific packages that we will be using and they need not be installed separately. Another popular distribution is using the pip tool itself. A command like "pip install jupyter" will install the jupyter notebook.

Introduction to Python

Basic Data types in python

Data types supported by python include integers, floats, booleans, and strings.

In [1]:

```python
x = 3
print(type(x)) # Prints "<class 'int'>"
print(x)       # Prints "3"
print(x + 1)   # Addition; prints "4"
print(x - 1)   # Subtraction; prints "2"
print(x * 2)   # Multiplication; prints "6"
print(x ** 2)  # Exponentiation; prints "9"
x += 1
print(x)  # Prints "4"
x *= 2
print(x)  # Prints "8"
y = 2.5
print(type(y)) # Prints "<class 'float'>"
print(y, y + 1, y * 2, y ** 2) # Prints "2.5 3.5 5.0 6.25"
```

```
<class 'int'>
3
4
2
6
9
4
8
<class 'float'>
2.5 3.5 5.0 6.25
```

Python does not support increment and decrement operators.

Increment and decrement operators like i++, ++i, i-- and --i are not supported in python

Booleans

In [4]:

```
t = True
f = False
print(type(t)) # Prints "<class 'bool'>"
print(t and f) # Logical AND; prints "False"
print(t or f)  # Logical OR; prints "True"
print(not t)   # Logical NOT; prints "False"
print(t != f)  # Logical XOR; prints "True"
```

```
<class 'bool'>
False
True
False
True
```

Strings and String Objects

In [1]:

```
a = "Hello, World!"
print(a[1])
b = "Hello, World!"
print(b[2:5])
a = " Hello, World! "
print("[Runs strip] ",a.strip()) # returns "Hello, World!"
a = "Hello, World!"
print("[Runs length] ", len(a))
a = "Hello, World!"
print("[Runs lower] ", a.lower())
a = "Hello, World!"
print("[Runs Upper] ", a.upper())
a = "Hello, World!"
print("[Runs replace] ", a.replace("H", "J"))
a = "Hello, World!"
print("[Runs split] ", a.split(",")) # returns ['Hello', ' World!']
```

```
e
llo
[Runs strip]  Hello, World!
[Runs length]  13
[Runs lower]  hello, world!
[Runs Upper]  HELLO, WORLD!
[Runs replace]  Jello, World!
[Runs split]  ['Hello', ' World!']
```

In [2]:

```
hello = 'hello'    # String literals can use single quotes
world = "world"    # or double quotes; it does not matter.
print(hello)       # Prints "hello"
print(len(hello))  # String length; prints "5"
hw = hello + ' ' + world  # String concatenation
print(hw)  # prints "hello world"
hw12 = '%s %s %d' % (hello, world, 12)  # sprintf style string formatting
print(hw12)  # prints "hello world 12"
```

```
hello
5
hello world
hello world 12
```

Containers

Data structure that contains objects.

Lists

The data structure closest to arrays in the C family. There are still many differences with arrays and it is more powerful than that.

In [37]:

```
# empty list
my_list = []

# list of integers
my_list = [1, 2, 3]

# list with mixed datatypes
my_list = [1, "Hello", 3.4]

# nested list
my_list = ["mouse", [8, 4, 6], ['a']]

# List indexes


my_list = ['p','r','o','b','e']
# Output: p
print(my_list[0])

# Output: o
print(my_list[2])

# Output: e
print(my_list[4])

# Error! Only integer can be used for indexing
# my_list[4.0]

# Nested List
n_list = ["Happy", [2,0,1,5]]

# Nested indexing

# Output: a
print(n_list[0][1])

# Output: 5
print(n_list[1][3])
```

```
p
o
e
a
5
```

Slicing

In [ ]:

```
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]"
```

Changing, adding and deleting elements

In [1]:

```
# mistake values
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

# Output: [1, 4, 6, 8]
print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

# Output: [1, 3, 5, 7]
print(odd)

odd = [1, 3, 5]

# Output: [1, 3, 5, 9, 7, 5]
print(odd + [9, 7, 5]) #Concatenation

#Output: ["re", "re", "re"]
print(["re"] * 3) #Repetition

## Removing elements

my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```

```
[1, 4, 6, 8]
[1, 3, 5, 7]
[1, 3, 5, 9, 7, 5]
['re', 're', 're']
['r', 'o', 'b', 'l', 'e', 'm']
o
['r', 'b', 'l', 'e', 'm']
m
['r', 'b', 'l', 'e']
[]
```

LOOPS

Most loops available in other languages are supported (and similar) in python. Make a careful note of the change in syntax. The do-while loop is not supported by python.

In [10]:

```
animals = ['cat', 'dog', 'monkey']
```

```
for animal in animals:
    print(animal)
```

```
cat
dog
monkey
```

In [11]:

```
# Uses enumerate to get each element.
animals = ['cat', 'dog', 'monkey']
for idx, animal in enumerate(animals):
    print('#%d: %s' % (idx + 1, animal))
```

```
#1: cat
#2: dog
#3: monkey
```

In [42]:

```
#Break statement

fruits = ["apple", "banana", "cherry"]
for x in fruits:
  print(x)
  if x == "banana":
    break
```

```
apple
banana
```

In [44]:

```
# While Looping
i = 1
while i < 6:
  print(i)
  i += 1
```

```
1
2
3
4
5
```

Dictionaries

A dictionary maps a set of objects (keys) to another set of objects (values). A Python dictionary is a mapping of unique keys to values. Dictionaries are mutable, which means they can be changed. The values that the keys point to can be any Python value.

In [13]:

```
d = {'cat': 'cute', 'dog': 'furry'}  # Create a new dictionary with some data
print(d['cat'])       # Get an entry from a dictionary; prints "cute"
print('cat' in d)     # Check if a dictionary has a given key; prints "True"
d['fish'] = 'wet'     # Set an entry in a dictionary
print(d['fish'])      # Prints "wet"
# print(d['monkey'])  # KeyError: 'monkey' not a key of d
print(d.get('monkey', 'N/A'))  # Get an element with a default; prints "N/A"
print(d.get('fish', 'N/A'))    # Get an element with a default; prints "wet"
del d['fish']         # Remove an element from a dictionary
print(d.get('fish', 'N/A')) # "fish" is no longer a key; prints "N/A"
```

```
cute
True
wet
N/A
wet
N/A
```

Sets

A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements. Python's set class represents the mathematical notion of a set.

The major difference between sets and lists is that sets, cannot have multiple occurrences of the same element and store unordered values. Thus it makes sets useful for some mathematical operations that does not allow duplicates.

In [15]:

```
animals = {'cat', 'dog'}
print('cat' in animals)   # Check if an element is in a set; prints "True"
print('fish' in animals)  # prints "False"
animals.add('fish')       # Add an element to a set
print('fish' in animals)  # Prints "True"
print(len(animals))       # Number of elements in a set; prints "3"
animals.add('cat')        # Adding an element that is already in the set does nothing
print(len(animals))       # Prints "3"
animals.remove('cat')     # Remove an element from a set
print(len(animals))       # Prints "2"
```

```
True
False
True
3
3
2
```

Tuples

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets

In [5]:

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5 )
tup3 = "a", "b", "c", "d"

print("tup1[0]: ", tup1[0])
print("tup2[1:5]: ", tup2[1:5])
```

```
tup1[0]:  physics
tup2[1:5]:  (2, 3, 4, 5)
```

In [7]:

```
#Updating Tuples.

tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2
print(tup3)
```

```
(12, 34.56, 'abc', 'xyz')
```

There are many other built in functions and operations that you can do with tuples.

Functions

In [16]:

```
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)  # Prints "HELLO, FRED!"
```

```
Hello, Bob
HELLO, FRED!
```

In [46]:

```
# Passing list as parameters

def my_function(food):
  for x in food:
    print(x)

fruits = ["apple", "banana", "cherry"]

my_function(fruits)
```

```
apple
banana
cherry
```

In [47]:

```
# Return Values

def my_function(x):
  return 5 * x

print(my_function(3))
print(my_function(5))
print(my_function(9))
```

```
15
25
45
```

Classes

In [17]:

```
class Greeter(object):

    # Constructor
    def __init__(self, name):
        self.name = name  # Create an instance variable

    # Instance 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

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

In [2]:

```
import numpy as np

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

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

```
<class 'numpy.ndarray'>
(3,)
1 2 3
[5 2 3]
(2, 3)
1 2 4
```

Numpy has many built in functions. Take a look at numpy to get an idea of all the possibilities.

In [3]:

```
import numpy as np

a = np.zeros((2,2))    # Create an array of all zeros
print(a)               # Prints "[[ 0.  0.]
                       #          [ 0.  0.]]"

b = np.ones((1,2))     # Create an array of all ones
print(b)               # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)   # Create a constant array
print(c)                # Prints "[[ 7.  7.]
                        #          [ 7.  7.]]"

d = np.eye(2)          # Create a 2x2 identity matrix
print(d)               # Prints "[[ 1.  0.]
                       #          [ 0.  1.]]"

e = np.random.random((2,2))   # Create an array filled with random values
print(e)                      # Might print "[[ 0.91940167  0.08143941]
                              #               [ 0.68744134  0.87236687]]"
```

```
[[0. 0.]
 [0. 0.]]
[[1. 1.]]
[[7 7]
 [7 7]]
[[1. 0.]
 [0. 1.]]
[[0.086986   0.10609101]
 [0.3012547  0.52741328]]
```

Array indexing in Numpy

In [4]:

```
import numpy as np
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"
```

```
2
77
```

Array Math

Now we come to the most important use of numpy

In [5]:

```
import numpy as np

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

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))

# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

Matplotlib

It is the most useful plotting library. Here we discuss some examples of the same.

In [6]:

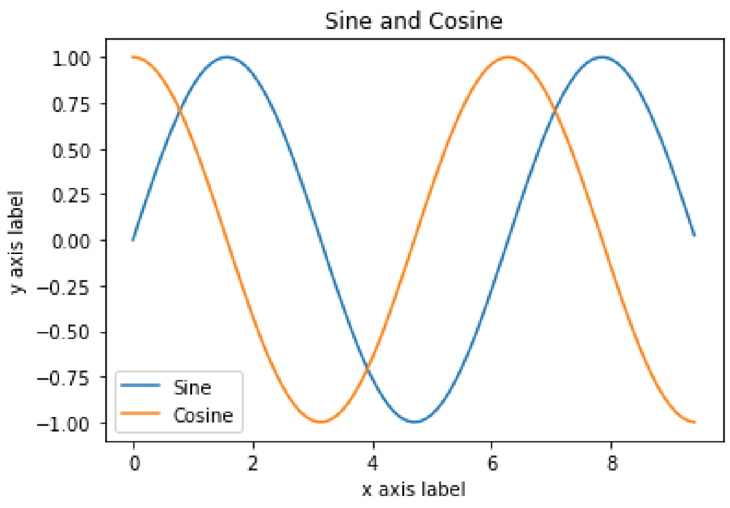
```
import numpy as np
import matplotlib.pyplot as plt

# 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.
```

```
<Figure size 640x480 with 1 Axes>
```

Plotting with title, legends, scale and axis labels

In [7]:

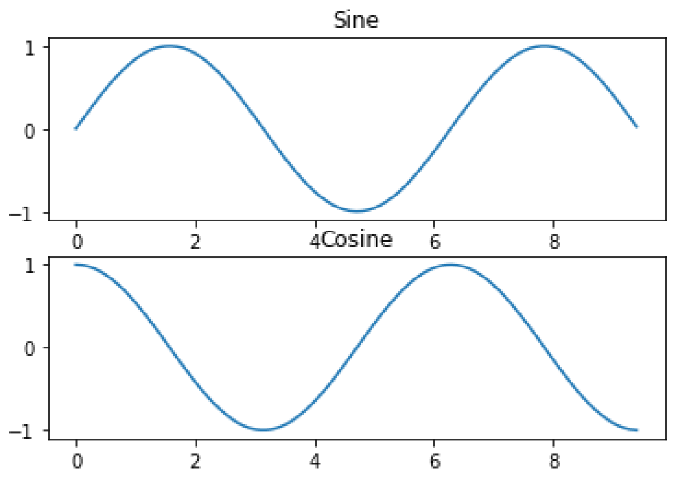
```
import numpy as np
import matplotlib.pyplot as plt

# 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()
```



Subplots

In [8]:

```
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)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

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



An important library to take note of: Pymc. Try installing the library and see examples from [here](). Another good tutorial is given [here](). This library will be used in the course and I might have a separate tutorial for that.

In [ ]: