1_pythonIntroduction

January 11, 2024

Instructors: Prof. Antonio Ortega (aortega@usc.edu)

Teaching Assistant: Jiazhi Li (jiazhil@usc.edu)

Notebooks: Written by Alexios Rustom (arustom@usc.edu)

1 Python Introduction

A brief introduction to operations in Python and the NumPy scientific computing package within Python.

Python is the programming language we will be using in this course. It has a set of numeric data types and arithmetic operations. NumPy is a library that extends the base capabilities of python to add a richer data set including more numeric types, vectors, matrices, and many matrix functions. NumPy and python work together fairly seamlessly. Python arithmetic operators work on NumPy data types and many NumPy functions will accept python data types.

For more information about Python Documentation: Python Documentation

[1]:
```
import numpy as np
import math
import time
import matplotlib.pyplot as plt
```

1.1 Operations on Numbers

[2]:
```
x1 = 1
x2 = 2
x = x1 + x2
print('The type of x is {}'.format(type(x)))
print('{} + {} is {}'.format(x1,x2,x))
```

```
The type of x is <class 'int'>
1 + 2 is 3
```

[3]:
```
y1=17
y2=3
y = y1*y2
print('The product of {} and {} is {}'.format(y1,y2,y))
```

```
The product of 17 and 3 is 51
```

[4]:
```
z1=10
z2=3
z = z1/z2
print('The ratio of {} and {} is {:.3f}'.format(z1,z2,z))
print("The smallest integer greater than {:.2f} is {}".format(z,math.ceil(z)))
print("The greatest integer greater than {:.1f} is {}".format(z,math.floor(z)))
```

```
The ratio of 10 and 3 is 3.333
The smallest integer greater than 3.33 is 4
The greatest integer greater than 3.3 is 3
```

[5]:
```
x=10
y=3
z = x//y
print('The floor devision of {} and {} is {}'.format(x,y,z))
```

```
The floor devision of 10 and 3 is 3
```

[6]:
```
x=10
y=3
z = x%y
print('{} Modulus {} is {}'.format(x,y,z))
```

```
10 Modulus 3 is 1
```

[7]:
```
x=10
y=2
z = x%y
print('{} Modulus {} is {}'.format(x,y,z))
```

```
10 Modulus 2 is 0
```

1.2 Logical Operators

[8]:
```
True
```

[8]: True

[9]:
```
False
```

[9]: False

[10]:
```
not True
```

[10]: False

[11]:
```
True or False
```

[11]: True

[12]:
```
(True or not False) and (1 and not 0)
```

[12]: True

[13]:
```
1 == 2
```

[13]: False

[14]:
```
1<=2
```

[14]: True

1.3 If, elif, else Statements

[15]:
```
x=3
y=2
if (x<y):
    print('x is less than y')
elif (x==y):
    print('x is equal to y')
else:
    print('x is greater than y')
```

```
x is greater than y
```

1.4 Range

[16]:
```
range(0,5)
```

[16]: range(0, 5)

1.5 For Loop

[17]:
```
for i in range(0,3):
    print(i)
```

```
0
1
2
```

1.6 While Loop

[18]:
```
i = 0
while (i<3):
    print(i)
    i = i+1
```

```
0
1
2
```

1.7 Operations on Lists

[19]:
```
x = list(range(0,5))
print('x = ',x)
print('The type of x is {}'.format(type(x)))
```

```
x =  [0, 1, 2, 3, 4]
The type of x is <class 'list'>
```

[20]:
```
x = [1,7,17]
print('x = ',x)
print('The type of x is {}'.format(type(x)))
```

```
x =  [1, 7, 17]
The type of x is <class 'list'>
```

[21]:
```
print('The attributes of a list are: ', dir(x))
```

```
The attributes of a list are:  ['__add__', '__class__', '__class_getitem__',
'__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse',
'sort']
```

[22]:
```
#Create an Empty List
y = []
#Append
y.append(1)
y.append(7)
y.append(17)
print('y = ',y)
```

```
y =  [1, 7, 17]
```

[23]:
```
print('The first element of y is: ', y[0])
print('The last element of y is: ', y[-1])
```

```
The first element of y is:  1
The last element of y is:  17
```

[24]:
```
y.insert(0,'Hello')
print('y = ',y)
```

```
y =  ['Hello', 1, 7, 17]
```

[25]:
```
y.insert(3,'Hello Again')
print('y = ',y)
```

```
y =  ['Hello', 1, 7, 'Hello Again', 17]
```

[26]:
```
y.extend([3, 4])
print('y = ',y)
```

```
y =  ['Hello', 1, 7, 'Hello Again', 17, 3, 4]
```

[27]:
```
del y[0]
print('y = ',y)
```

```
y =  [1, 7, 'Hello Again', 17, 3, 4]
```

[28]:
```
y.append('Hello Again')
print('y = ',y)
```

```
y =  [1, 7, 'Hello Again', 17, 3, 4, 'Hello Again']
```

[29]:
```
y.remove('Hello Again')
print('y = ',y)
```

```
y =  [1, 7, 17, 3, 4, 'Hello Again']
```

[30]:
```
y.remove('Hello Again')
print('y = ',y)
```

```
y =  [1, 7, 17, 3, 4]
```

[31]:
```
y.reverse()
print('The reverse is: ', y)
```

```
The reverse is:  [4, 3, 17, 7, 1]
```

[32]:
```
y.sort()
print('The sorted (ascending) list is: ', y)
```

```
The sorted (ascending) list is:  [1, 3, 4, 7, 17]
```

[33]:
```
y.sort(reverse=True)
print('The sorted (descending) list is: ', y)
```

```
The sorted (descending) list is:  [17, 7, 4, 3, 1]
```

[34]:
```
print('The minimum of y is: ', min(y))
print('The maximum of y is: ', max(y))
print('The index of y[0] is: ', y.index(y[0]))
print('The sum of y is: ', sum(y))
```

```
The minimum of y is:  1
The maximum of y is:  17
The index of y[0] is:  0
The sum of y is:  32
```

[35]:
```
y.append('Hello')
print('y =', y)
```

```
y = [17, 7, 4, 3, 1, 'Hello']
```

[36]:
```
print('Hello' in y)
```

```
True
```

1.8 List Comprehension

[37]:
```
alphabet_letters = []

for letter in 'abcdefghijklmnopqrstuvwxyz':
    alphabet_letters.append(letter)

print('Alphabet letters using for loop: \n', alphabet_letters)
```

```
Alphabet letters using for loop:
 ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

[38]:
```
alphabet_letters_new = [ letter for letter in 'abcdefghijklmnopqrstuvwxyz' ]
print('Alphabet letters using List Comprehension: \n', alphabet_letters_new)
```

```
Alphabet letters using List Comprehension:
 ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z']
```

[39]:
```
tic = time.time()  # capture start time
list = []
upper_limit = 100000000
for integer in range(0,upper_limit):
    list.append(integer)
toc = time.time()  # capture end time
print(f"Duration For loop: {1000*(toc-tic):.4f} ms ")

lst = []
tic = time.time()
lst = [integer for integer in range(upper_limit)]
toc = time.time()
print(f"Duration For List comprehension: {1000*(toc-tic):.4f} ms ")
```

```
Duration For loop: 6907.7294 ms
Duration For List comprehension: 3010.4735 ms
```

1.9 Dictionary

[40]:
```
# A dictionary stores a list of (key, value) pairs
animal_legs = {'cat': 4, 'dog': 4, 'fish': 0, 'bird': 2, 'spider': 8}
print(animal_legs['dog'])
```

```
4
```

[41]:
```
for k, v in animal_legs.items():
    print("A {} has {} legs.".format(k,v))
```

```
A cat has 4 legs.
A dog has 4 legs.
A fish has 0 legs.
A bird has 2 legs.
A spider has 8 legs.
```

- ints, floats, tuples, strings are immutables
- lists, dictionaries are mutables

1.10 Functions

[42]:
```
def my_function(major):
    print("Hello, I am an undegraduate student in", major)
```

[43]:
```
my_function('Electrical & Computer Engineering')
```

```
Hello, I am an undegraduate student in Electrical & Computer Engineering
```

[44]:
```
my_function('Computer Science')
```

```
Hello, I am an undegraduate student in Computer Science
```

In Python, arguments of a function are passed by assignment which behave differently for immutables and mutables.

[45]:
```
x = ['A', 'C', 'D'] # Lists are mutables

def change_list(the_list):
    print("The original list is:",the_list)
    the_list.append('E')
    print("In the funtion: It is changed to\n", the_list)

change_list(x)
print("Outside the function: It is changed to\n", x)
```

```
The original list is: ['A', 'C', 'D']
In the funtion: It is changed to
 ['A', 'C', 'D', 'E']
Outside the function: It is changed to
 ['A', 'C', 'D', 'E']
```

[46]:
```
x = "This is a string." # Strings are immutables

def change_string(the_string):
    print("The original string is:",the_string)
    the_string = the_string + " ABCDE."
    print("In the funtion: It is changed to\n", the_string)

change_string(x)
print("Outside the funtion: It is changed to\n", x)
```

```
The original string is: This is a string.
In the funtion: It is changed to
 This is a string. ABCDE.
Outside the funtion: It is changed to
 This is a string.
```

To avoid unwanted change on mutables, we use deep copy.

[47]:
```
x = [31, 13, 22]
x2 = x.copy()
x2.append(0)
print("x is {}.\nx2 is {}.".format(x,x2))

x3 = x
x3.append(0)
print("x is {}.\nx3 is {}.".format(x,x3))
```

```
x is [31, 13, 22].
x2 is [31, 13, 22, 0].
x is [31, 13, 22, 0].
x3 is [31, 13, 22, 0].
```

1.11 Python Lambda

syntax: lambda argument:operation

[48]:
```
x = lambda a : a + 10
print(x(5))
```

```
15
```

[49]:
```
x = lambda a, b, c : a + b + c
x(5, 6, 2)
```

[49]:
```
13
```

The power of lambda is better shown when you use them as an anonymous function inside another function.

Say you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

[50]:
```
def myfunc(n):
  return lambda a : a * n
```

[51]:
```
mydoubler = myfunc(2)
print(mydoubler(11))
```

```
22
```

[52]:
```
mytripler = myfunc(3)
print(mytripler(11))
```

```
33
```

1.12 Numpy Arrays

[53]:
```
# NumPy routines which allocate memory and fill arrays with value
a = np.zeros(10);
print(f"np.zeros(4) :   a = {a}, shape of a= {a.shape}, data type of a = {a.
 ↪dtype}")
```

```
np.zeros(4) :   a = [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.], shape of a= (10,), data
type of a = float64
```

[54]:
```
print('Type of a: ', type(a))
```

```
Type of a:  <class 'numpy.ndarray'>
```

[55]:
```
a = np.ones((1,2));              print(f"np.ones(1,2) :  a = {a}, shape of a =␣
 ↪{a.shape}, data type of a = {a.dtype}")
```

```
np.ones(1,2) :  a = [[1. 1.]], shape of a = (1, 2), data type of a = float64
```

[56]:
```
a = np.random.rand(3); print(f"np.random.rand(3): a = {a}, shape of a = {a.
 ↪shape}, data type of a = {a.dtype}")
```

```
np.random.rand(3): a = [0.75163266 0.02310556 0.5136052 ], shape of a = (3,),
data type of a = float64
```

[57]:
```
a = np.random.randint(1, 10, 3); print(f"np.random.randint(3, 0, 10): a = {a},␣
 ↪shape of a = {a.shape}, data type of a = {a.dtype}")
```

```
np.random.randint(3, 0, 10): a = [2 9 3], shape of a = (3,), data type of a =
int64
```

[58]:
```
a1 = np.random.randint(1, 10, 3)
print('a1: ', a1)
a2 = np.random.randint(1, 10, 3)
print('a2: ', a2)
```

```
a1:  [9 6 4]
a2:  [5 3 1]
```

[59]:
```
print('Attributes of a1: \n', dir(a1))
```

```
Attributes of a1:
 ['T', '__abs__', '__add__', '__and__', '__array__', '__array_finalize__',
'__array_function__', '__array_interface__', '__array_prepare__',
'__array_priority__', '__array_struct__', '__array_ufunc__', '__array_wrap__',
'__bool__', '__class__', '__class_getitem__', '__complex__', '__contains__',
'__copy__', '__deepcopy__', '__delattr__', '__delitem__', '__dir__',
'__divmod__', '__dlpack__', '__dlpack_device__', '__doc__', '__eq__',
'__float__', '__floordiv__', '__format__', '__ge__', '__getattribute__',
'__getitem__', '__gt__', '__hash__', '__iadd__', '__iand__', '__ifloordiv__',
'__ilshift__', '__imatmul__', '__imod__', '__imul__', '__index__', '__init__',
'__init_subclass__', '__int__', '__invert__', '__ior__', '__ipow__',
'__irshift__', '__isub__', '__iter__', '__itruediv__', '__ixor__', '__le__',
'__len__', '__lshift__', '__lt__', '__matmul__', '__mod__', '__mul__', '__ne__',
'__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__',
'__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__',
'__rlshift__', '__rmatmul__', '__rmod__', '__rmul__', '__ror__', '__rpow__',
'__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__',
'__setattr__', '__setitem__', '__setstate__', '__sizeof__', '__str__',
'__sub__', '__subclasshook__', '__truediv__', '__xor__', 'all', 'any', 'argmax',
'argmin', 'argpartition', 'argsort', 'astype', 'base', 'byteswap', 'choose',
'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumprod', 'cumsum',
'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat',
'flatten', 'getfield', 'imag', 'item', 'itemset', 'itemsize', 'max', 'mean',
'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp',
'put', 'ravel', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted',
'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze', 'std', 'strides',
'sum', 'swapaxes', 'take', 'tobytes', 'tofile', 'tolist', 'tostring', 'trace',
'transpose', 'var', 'view']
```

[60]:
```
print('a1 - a2: ', a1-a2)
```

```
a1 - a2:  [4 3 3]
```

[61]:
```
print('First two elements of a1: ', a1[0:2])
print('Last two elements of a2: ', a2[-2:])
```

```
First two elements of a1:  [9 6]
Last two elements of a2:  [3 1]
```

[62]:
```
upper_limit = 100000000
lst = []
tic = time.time()
lst = [integer for integer in range(upper_limit)]
toc = time.time()
print(f"Duration For List comprehension: {1000*(toc-tic):.4f} ms ")

tic = time.time()
```

```
array_integers = np.arange(0,upper_limit)
toc = time.time()
print(f"Duration For Array Construction: {1000*(toc-tic):.4f} ms ")
```

```
Duration For List comprehension: 3050.6854 ms
Duration For Array Construction: 106.6911 ms
```

[63]:
```
a1 = np.random.randint(1, 10, 3)
print('a1: ', a1)
a2 = np.random.randint(1, 10, 3)
print('a2: ', a2)
```

```
a1:  [1 7 9]
a2:  [7 5 6]
```

[64]:
```
np.set_printoptions(precision=3)
print('Exponential of a1: ', np.exp(a1))
```

```
Exponential of a1:  [2.718e+00 1.097e+03 8.103e+03]
```

[65]:
```
print('Square Root of a1: ',np.sqrt(a1))
```

```
Square Root of a1:  [1.    2.646 3.   ]
```

[66]:
```
print('Cosine of a1: ',np.cos(a1))
```

```
Cosine of a1:  [ 0.54   0.754 -0.911]
```

[67]:
```
np.set_printoptions(suppress=True)
```

[68]:
```
A = np.arange(10)
print('A: ', A)
print('Shape of A: ', A.shape)
```

```
A:  [0 1 2 3 4 5 6 7 8 9]
Shape of A:  (10,)
```

[69]:
```
A_copy = A[0:4].copy()
print('A_copy: \n', A_copy)
print('Shape of A_copy: \n', A_copy.shape)
```

```
A_copy:
 [0 1 2 3]
Shape of A_copy:
 (4,)
```

[70]:
```
A_copy_reshaped = A_copy.reshape(2,2)
print('Shape of A_copy_reshaped: \n', A_copy_reshaped.shape)
print('Shape of A_copy: \n', A_copy.shape)
print('Shape of A: \n', A.shape)
```

```
Shape of A_copy_reshaped:
 (2, 2)
Shape of A_copy:
 (4,)
Shape of A:
 (10,)
```

[71]:
```
x = np.array([1, 2, 3, 4])
print('x: ', x)
```

```
x:  [1 2 3 4]
```

[72]:
```
# multiply a by a scalar
y = 7 * x
print(f"y = 7 * x : {y}")
```

```
y = 7 * x : [ 7 14 21 28]
```

[73]:
```
print('Square of every element in y:', y**2)
```

```
Square of every element in y: [ 49 196 441 784]
```

[74]:
```
a3 = np.array([[1, 2, 3], [3, 4, 5]])
print('Shape of a3: ', a3.shape)
print('a3: \n', a3)
```

```
Shape of a3:  (2, 3)
a3:
 [[1 2 3]
 [3 4 5]]
```

[75]:
```
a3_T = a3.transpose()
print('Shape of a3_T: ', a3_T.shape)
print('The transpose of a3: \n', a3_T)
```

```
Shape of a3_T:  (3, 2)
The transpose of a3:
 [[1 3]
 [2 4]
 [3 5]]
```

[76]:
```
print('The sum of a1 & a2: ', a1+a2)
```

```
The sum of a1 & a2:  [ 8 12 15]
```

[77]:
```
def dot_product_function(a, b):
    x=0
    for i in range(b.shape[0]):
        x = x + a[i] * b[i]
    return x
```

[78]:
```
np.random.seed(1)
a = np.random.rand(50000000)
b = np.random.rand(50000000)

tic = time.time()  # capture start time
dot_product_np = np.dot(a, b)
toc = time.time()  # capture end time

print(f"np.dot(a, b) =  {dot_product_np:.4f}")
#Vector Notation
print(f"duration of np.dot(a, b): {1000*(toc-tic):.4f} ms ")

tic = time.time()  # capture start time
c = dot_product_function(a,b)
toc = time.time()  # capture end time

print(f"dot_product_function =  {c:.4f}")
print(f"duration of dot_product_function: {1000*(toc-tic):.4f} ms ")

del(a);del(b)  #remove these big arrays from memory
```

```
np.dot(a, b) =  12501467.1511
duration of np.dot(a, b): 41.4679 ms
dot_product_function =  12501467.1511
duration of dot_product_function: 9089.4649 ms
```

[79]:
```
A = np.array([[4, 20],[10, 1]])
print('Maximum of A is: ', A.max())
print('Minimum of A is: ', A.min())
```

```
Maximum of A is:  20
Minimum of A is:  1
```

[80]:
```
print('Determinant of A: {:.2f}'.format(np.linalg.det(A)))
```

```
Determinant of A: -196.00
```

[81]:
```
print('Inverse of A: \n', np.linalg.inv(A))
```

```
Inverse of A:
 [[-0.005  0.102]
 [ 0.051 -0.02 ]]
```

[82]:
```
B = np.array([[4, 1],[2, 2]])
print('The shape of A is: ', A.shape)
print('The shape of B is: ', B.shape)
assert A.shape[1] == B.shape[0], f'AB is not defined'
```

```
The shape of A is:  (2, 2)
The shape of B is:  (2, 2)
```

[83]:
```
C = np.matmul(A,B)
print('The matrix product of A & B: \n', C)
```

```
The matrix product of A & B:
 [[56 44]
 [42 12]]
```

[84]:
```
bool_flag = C> 20
print(bool_flag)
```

```
[[ True  True]
 [ True False]]
```

[85]:
```
D = C[bool_flag]
D
```

[85]:
```
array([56, 44, 42])
```

1.13 Pointer behavior: Shallow copy vs deep copy

[86]:
```
# Shallow copy
a = np.arange(3)
print('a = {0}'.format(a))
b = a
print('b = {0}'.format(b))
a[0] = 100
print('a = {0}'.format(a))
print('b = {0}'.format(b))
```

```
a = [0 1 2]
b = [0 1 2]
a = [100   1   2]
b = [100   1   2]
```

[87]:
```
# Deep copy
c = np.zeros(len(a)) #initialize c
c[:] = a[:]
d = np.copy(a)
a[0] = 55
print('a = {0}'.format(a))
print('c = {0}'.format(c))
print('d = {0}'.format(d))
```

```
a = [55  1  2]
c = [100.   1.   2.]
d = [100   1   2]
```

1.14 Plotting

[88]:
```
x= np.arange(0, 10, 0.01)
# corresponding y axis values
y = np.cos(x)
z = np.sin(x)
# plotting the points
plt.plot(x, y, color='green',label = 'Cosine')
plt.plot(x, z, color='blue', label = 'Sine')
# function to show the plot
plt.xlabel('x', fontsize=12)
plt.ylabel('y', fontsize=12)
plt.title('Cosine & Sine plots', fontsize=16)
plt.legend()
plt.show()
```