intro-to-python-and-numpy-1-1

January 25, 2024

#CSCI 4851 AND CSCI 6951#

1 Basics of Jupyter Notebook

1.0.1 Two types of cells

- Code cell: for coding
- Markdown cells : for text

1.0.2 Keyboard Shortcuts for Jupyter Notebook:

- Shift-Enter :execute a cell
- a: insert cell above
- b: insert cell below
- dd: delete cell
- m: convert cell to markdown (in esc mode)
- y: convert back to code cell (in esc mode)

#Basics of Python# ##Data Types## ###Numbers###

[58]:
```
x = 13
print (x)
print (type(x))
```

```
13
<class 'int'>
```

[59]:
```
y= 5.9
print(y)
print (type(y))
```

```
5.9
<class 'float'>
```

[60]:
```
print (x+1)  #Addition
print (x-1)  #Subtraction
print (x*2)  #Multiplication
print (x/2)  #Division
print (x**2) #Exponentitaion
```

```
print (x%2)  #Modulus
```

```
14
12
26
6.5
169
1
```

[61]:
```
x +=2 #Another way of doing addition
print(x)
x -=2
print(x)
x *=2
print(x)
```

```
15
13
26
```

Question 1: Solve

```
# What will be value of y when it depends upon variable x raised to the power of 3 and the resu
```

[62]:
```
x =4
y = x**3 + 300
print(y)
```

```
364
```

###**Booleans**###

[63]:
```
x=3
y=2
z=3
t= (x==z)
f= (x==y)
print(t)
print(f)
```

```
True
False
```

[64]:
```
print (t and f)  #Logical AND;
print (t or  f)  #Logical OR;
print (not t)    #Logical NOT;
print (t !=f)    #Logical XOR;
```

```
False
True
```

```
False
True
```

###**Strings**###

[65]:
```
h='hello'  #Strings in single quotes
w= "world" #Strings in double quotes
print(h)
print(w)
```

```
hello
world
```

[66]:
```
z= "5"
print(z)
print(type(z))
```

```
5
<class 'str'>
```

Question 2: Solve

```
#Print the text Hello, World ! using the '+' sign for the concatenation of the string
```

[67]:
```
# print(f"{h.capitalize()}, {w.capitalize()} !")
h1 = h.capitalize()
w1 = w.capitalize()
print( h1 + ", " + w1 + " !")
```

```
Hello, World !
```

Useful fuctions and methods of string function

[68]:
```
print(len(h)) #length function returns the number of items in an object, for␣
 ↪string itreturns number of characters
```

```
5
```

[69]:
```
txt1 = "My name is {fname}, I'm {age}".format(fname = "Raya", age = 36)  #The␣
 ↪format() method formats the specified value(s) and insert them inside the␣
 ↪string's placeholder.
print(txt1)
txt2 = "My name is {0}, I'm {1}".format("Raya",36)
print(txt2)
txt3 = "My name is {}, I'm {}".format("Raya",36)
print(txt3)
```

```
My name is Raya, I'm 36
My name is Raya, I'm 36
My name is Raya, I'm 36
```

[70]:
```
fn="raya"
print(fn.capitalize())  # Capitalize a string
print(fn.upper())       # Convert a string to uppercase; prints "HELLO"
print(fn.rjust(7))      # Right-justify a string, padding with spaces
print(fn.replace('l', 'sh'))  # Replace all instances of one substring with␣
 ↪another
```

```
Raya
RAYA
   raya
raya
```

Question 3: Solve

print your name as center justified, with 7 padding spaces

[71]:
```
fn = "raya"
print(fn.center(7))
```

```
 raya
```

You can find a list of all string methods in the documentation.

1.1 Python Containers

###**Lists**### A list is equivalent to an array, but is resizeable and can contain elements of different types

[72]:
```
new_list= [2,4,6] #Creates a list
print(new_list)
```

```
[2, 4, 6]
```

[73]:
```
#Indices of list
print(new_list[0])
print(new_list[2])
print(new_list[-1]) #negative indices starts from the back of the list
```

```
2
6
6
```

[74]:
```
my_list=['banana', 'apple', 1, 2] #list can contain multiplle data types
print(my_list)
print(type(my_list[1]))
print(type(my_list[2]))
```

```
['banana', 'apple', 1, 2]
<class 'str'>
<class 'int'>
```

[75]:
```
my_list.append(3) #add new element at the end of the list
print(my_list)
my_list.pop()  #remove the last element of the list
print(my_list)
```

```
['banana', 'apple', 1, 2, 3]
['banana', 'apple', 1, 2]
```

[76]:
```
#Slicing or accessing the sublist using Python
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[:-2])         # Slice indices can be negative; prints ["0, 1, 2]"
```

```
[0, 1, 2, 3, 4]
[2, 3]
[2, 3, 4]
[0, 1]
[0, 1, 2, 3, 4]
[0, 1, 2]
```

Question 4: Solve

‘Assign a new sublist (8,9) in the original list nums at indices 2 and 3.

[111]:
```
nums[2:4] = [8,9]
print(nums)
```

```
[0, 1, 8, 9, 4]
```

###**Dictionaries**### A dictionary stores a (key, value) pairs. More about dictionaries is in this documentation

[78]:
```
dn={'e': 'elephant', 'd':' dog '} #Creates dictionary with key and it value pair
print(dn['e']) #Get an entry from the key
dn['c']='cat' #Adding key value pair to the dictionary
print(dn['c'])
```

```
elephant
cat
```

[79]:
```
# it is easy to iterate over dictionaries
d = {'person': 2, 'cat': 4, 'spider': 8}
for animal, legs in d.items():
    print('A {} has {} legs'.format(animal, legs))
```

```
A person has 2 legs
A cat has 4 legs
A spider has 8 legs
```

###**Sets**### A set is unorderd collection of elements

[80]:
```
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')
print('fish' in animals)  # prints "False"
animals.remove('cat')    # Remove an element from a set
print(len(animals))
```

```
True
False
True
2
```

#**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

[81]:
```
import numpy
```

[82]:
```
a= numpy.array([1,2,3]) ## array with rank 1
print (a)
```

```
[1 2 3]
```

[83]:
```
import numpy as np #abbreviation eases the calling of the library
a= np.array([1,2,3]) ## array with rank 1
print (a)
print (type(a))
```

```
[1 2 3]
<class 'numpy.ndarray'>
```

##**Arrays**##

[84]:
```
a = np.array(42)
b = np.array([1, 2, 3, 4, 5]) #1 D array
c = np.array([[1, 2, 3], [4, 5, 6]]) #2 D array
d = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) #3 D array
```

```
e = np.array([[[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]],  [[[1, 2, 3],␣
 ↪[4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]])

# Shape = whole Outer, outer, inner
# Dimension = whole 2D twice for 3D, whole 3D twice for 4D and so on

print(a.ndim)
print(b.ndim)
print(c.ndim)
print(d.ndim)
print(e.ndim)
```

```
0
1
2
3
4
```

Bonus Question: Make an array with 5 dimensions

[85]:
```
f = np.array([[[[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]], [[[1, 2, 3],␣
 ↪[4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]],

              [[[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]], [[[1, 2, 3],␣
 ↪[4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]]

              ])
print(f.ndim)
```

```
5
```

[86]:
```
# Functions to create array
a=np.zeros((2,2)) #create an array of all zeros
print(a)
```

```
[[0. 0.]
 [0. 0.]]
```

[87]:
```
b=np.ones((2,2)) #create an array of all one
print(b)
```

```
[[1. 1.]
 [1. 1.]]
```

[88]:
```
c = np.full((2,2), 6) # Create a constant array
print (c)
```

```
[[6 6]
```

```
 [6 6]]
```

[89]:
```
d=np.eye(3)
print(d)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

[90]:
```
e=np.random.random((3,3))
print(e)
```

```
[[0.22900096 0.72092247 0.39853525]
 [0.7051335  0.16168659 0.28735854]
 [0.37854112 0.17079281 0.29577102]]
```

###**Array Indexing**###

[91]:
```
# 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]
print(b)
```

```
[[2 3]
 [6 7]]
```

We pass slice instead of index like this: [start:end].

We can also define the step, like this: [start:end:step].

[92]:
```
print(a[0, 1])
b[0, 0] = 77    # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])  # A slice of an array will modify the original array
```

```
2
77
```

[93]:
```
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
row_r3 = a[[1], :]  # Rank 2 view of the second row of a

print(row_r1, row_r1.shape)
```

```
print(row_r2, row_r2.shape)
print(row_r3, row_r3.shape)
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
[[5 6 7 8]] (1, 4)
```

Question 5: Solve

Write a code and verify it to access rank 1 and rank 2 of the second column of a

[94]:
```
# Rank1 is 1D (0,1) and Rank2 is 2D (1,1)

row_r4 = a[:, 1]
print(row_r4, row_r4.shape)
row_r4 = a[:, 1:2]
print(row_r4, row_r4.shape)
```

```
[77  6 10] (3,)
[[77]
 [ 6]
 [10]] (3, 1)
```

###**Creating Arrays from another Array**###

[95]:
```
a = np.array([[1,2], [3, 4], [5, 6]])
print(a)

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

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

[96]:
```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)  # Find the elements of a that are bigger than 2;
                    # this returns a numpy array of Booleans of the same
                    # shape as a, where each slot of bool_idx tells
                    # whether that element of a is > 2.
```

```
print(bool_idx)
```

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

###**Datatypes**###

[97]:
```
x = np.array([1, 2])  # Let numpy choose the datatype
y = np.array([1.0, 2.0])  # Let numpy choose the datatype
z = np.array([1, 2], dtype=np.int64)  # Force a particular datatype

print(x.dtype, y.dtype, z.dtype)
```

```
int64 float64 int64
```

###**Array Maths**###

[98]:
```
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 same array
print(x + y)
print(np.add(x, y))
```

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

[99]:
```
# Elementwise difference; both produce the array
print(x - y)
print(np.subtract(x, y))
```

```
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
```

[100]:
```
# Elementwise product; both produce the array
print(x * y)
print(np.multiply(x, y))
```

```
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
```

[101]:
```
# Elementwise division;
print(x / y)
print(np.divide(x, y))
```

```
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
```

Question 6:Solve

Find the elementwise square root of x using the sqrt function of numpy

[102]:
```
print(np.sqrt(x))
```

```
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

[103]:
```
v = np.array([9,10])
w = np.array([11, 12])

# Matrix multiplication
print(np.dot(v, w))
print(v.dot(w))
print(v @ w)
```

```
219
219
219
```

[104]:
```
# Performing computations on arrays
x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

```
10
[4 6]
[3 7]
```

[105]:
```
print(x)
print("transpose\n", x.T)
```

```
[[1 2]
 [3 4]]
transpose
 [[1 3]
 [2 4]]
```

List of mathematical functions provided by numpy documentation

Functions:

rint()

[106]:
```
x = np.array([[1.8, 2.9, 3.5], [2.6, 5.8, 8.3]])
np.rint(x)
```

[106]:
```
array([[2., 3., 4.],
       [3., 6., 8.]])
```

trunc()

[107]:
```
x = np.array([-1.7, -1.5, -0.2, 0.2, 1.5, 1.7, 2.0])
np.trunc(x)
```

[107]:
```
array([-1., -1., -0.,  0.,  1.,  1.,  2.])
```

nanprod()

[108]:
```
x = np.array([[1,2,2], [2,3, np.nan]])
np.nanprod(x)
```

[108]:
```
24.0
```

cross()

[109]:
```
x = [1, 2, 0]
y = [4, 5, 6]
np.cross(x, y)
```

[109]:
```
array([12, -6, -3])
```

fabs()

[110]:
```
x = np.array([[-1.6, -2.4, 2.2], [2.5, -3.8, -5.7]])
np.fabs(x)
```

[110]:
```
array([[1.6, 2.4, 2.2],
       [2.5, 3.8, 5.7]])
```