cosmos_python_2017

July 5, 2017

1 Python as calculator

Note: To convert ipynb to pdf file, use command: ipython nbconvert cosmos_python_2015.ipynb –to latex –post pdf

```
In [3]: 1 + 3
```

```
Out[3]: 4
```

```
In [4]: 3 * 90
```

```
Out[4]: 270
```

```
In [7]: 4 + 5 * 6 - 2
```

```
Out[7]: 32
```

```
In [8]: 2/5.0
```

```
Out[8]: 0.4
```

```
In [9]: 2/5
```

```
Out[9]: 0
```

What happened? Be careful when you divide by an integer!!

1.0.1 New ideas: Modulo and whole value division

```
In [10]: 8 % 3
```

```
Out[10]: 2
```

```
In [11]: 9 % 3
```

```
Out[11]: 0
```

```
In [12]: 8 // 3
```

```
Out[12]: 2
```

```
In [13]: 9 // 3
```

```
Out[13]: 3
```

2 Variables and Assignments

```
In [14]: x = 6
         x

Out[14]: 6

In [15]: x = 'My name is Joe'
         x

Out[15]: 'My name is Joe'

In [19]: type(x)

Out[19]: str
```

Note : variables are not "declared" as one type: variable type can change!

```
In [20]: x = 2.345
         type(x)

Out[20]: float

In [21]: x = 1 - 3j
         type(x)

Out[21]: complex

In [22]: x = True
         type(x)

Out[22]: bool

In [23]: x, y = 3 , 4
         print x,y

3 4

In [24]: x, y = y , x
         print x , y

4 3

In [25]: z = x

In [33]: x = z * y
         print x , y,  z

12 3 4

In [34]: x = z - y
         print x , y, z

1 3 4

In [35]: x = z**y - x
         print x , y, z

63 3 4
```

```
In [36]: x = 2
         y = 'Bob'
         x  + y
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-36-8d8b7d6f72c6> in <module>()
  1 x = 2
  2 y = 'Bob'
----> 3 x  + y


TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

```
In [37]: x = 'Bradley '
         y = 'Cooper'
         x + y
```

```
Out[37]: 'Bradley Cooper'
```

```
In [56]: x = float(raw_input("Enter value for x:"))
         print "The value of x is:  ",x
```

```
Enter value for x:34.5
The value of x is:   34.5
```

```
In [51]: x , y = 2 , 3
         print "The value of x * y = ", x*y
```

```
The value of x * y =  6
```

2.1 Packages and modules

```
In [3]: import numpy as np
        print np.pi, np.cos(np.pi), np.e
```

```
3.14159265359 -1.0 2.71828182846
```

```
In [4]: x = np.cos(np.pi) * np.sin(2*np.pi) + np.log(10)
        print x
```

```
2.30258509299
```

```
In [9]: import random as random
        #help(random)
        random.randint(0,9)
```

```
Out[9]: 6
```

2.2 Lists and Arrays

All variables we have used to this point have had a single value. But we want a variable that supports multiple values at once : Called an "array" in C, python has a few different variable types with multiple values. here we will look at lists and arrays.

Most basic type of container in python is a "list". It's just a list of quantities in a row, one after the other. What's special about python lists, is that all of the "elemnets" in the list do not have to be the same type.

2.2.1 Lists

```
In [62]: r = [ 1, 2, 3, 4 ,5, 10]
         print r

[1, 2, 3, 4, 5, 10]

In [63]: r = [ 'Bob', 6, 78.98, True, 10, 1+8j]
         print r

['Bob', 6, 78.98, True, 10, (1+8j)]
```

To add a new element to end of list, use "append":

```
In [12]: r = [1, 2,3, 4, 5, 10]
         r.append(6)
         r.append('apples')
         print r

[1, 2, 3, 4, 5, 10, 6, 'apples']
```

"Slicing" lets you access elements of the list

```
In [15]: r[3]

Out[15]: 4

In [16]: r[3] = 'oranges'
         print r

[1, 2, 3, 'oranges', 5, 10, 6, 'apples']

In [13]: r[0:7]
         r[:7]

Out[13]: [1, 2, 3, 4, 5, 10, 6]

In [18]: r[-1]

Out[18]: 'apples'

In [19]: r[-1::-1]

Out[19]: ['apples', 6, 10, 5, 'oranges', 3, 2, 1]

In [21]: r_copy = r
         print r_copy

[1, 2, 3, 'oranges', 5, 10, 6, 'apples']

In [78]: r[0] = 86
         print r
         print r_copy

[86, 2, 3, 'oranges', 5, 10, 6, 'apples']
[86, 2, 3, 'oranges', 5, 10, 6, 'apples']
```

What happened ? "r_copy" is not a separate copy of r. It "points" to r. We must be careful... If we want to separate copy of r, we must use slicing, as shown below.

```
In [22]: # To create separate copy of a list
         r_copy = r[:]
         print r_copy
         r[0] = 99
         print r
         print r_copy
```

```
[1, 2, 3, 'oranges', 5, 10, 6, 'apples']
[99, 2, 3, 'oranges', 5, 10, 6, 'apples']
[1, 2, 3, 'oranges', 5, 10, 6, 'apples']
```

```
In [24]: # Use range built-in function
         w = range(10)
         print w
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [2]: w = range(1, 11)
        print w
```

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
In [14]: # get nuber of elements and sum of list
         w = range(1,100,10)
         print w
         print "Number of elements of w = ", len(w)
         print "Sum of all elements of w = ", sum(w)
```

```
[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
Number of elements of w =  10
Sum of all elements of w =  460
```

```
In [10]: #  Delete last element of list
         print w
         w.pop()
         print w
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

2.2.2 Numpy arrays

What if we want a "list" where all the elements are the same type (like in C language)? In python, this is called an array. Arrays use a special module called numpy (for numerical python). Arrays are like vectors/matrices. They are fixed in length and are much faster to work with than lists in python. We will use arrays more than lists.

```
In [28]: # Import numpy module and create array of zeros
         import numpy as np
         array1 = np.zeros(10,float)
         print array1, len(array1)
```

```
[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.] 10
```

```
In [29]: # Create array of ones of type integer
         array2 = np.ones(10,int)
         print array2
```

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

```
In [30]: # Convert list to array ( and array to list)
         r = [1.2, 3.4, -1.5]    # This is a list
         r_array = np.array(r)
         print r_array
```

```
[ 1.2  3.4 -1.5]
```

```
In [33]: # Create 2D array (matrix)
         array3 = np.zeros((2,3))  # Notice (2,3) inside: 2 x 3 matrix
         print array3
```

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

```
In [10]: # Operations on arrays
         array4 = np.array (range(10))
         print "Mean value in array =  " , sum(array4)/ len(array4)
         print "Mean value in array =  " , array4.mean()
```

```
Mean value in array =   4
Mean value in array =   4.5
```

```
In [42]: # Convert array to a list : To add onto array
         list4 = list(array4)
         print list4
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
In [44]: # Arithmetic with arrays
         array5 = np.array([0,1,2, 3,4,5])
         x = array5**2
         print x
```

```
[ 0  1  4  9 16 25]
```

```
In [45]: x = np.sqrt ( sum(array5**2))
         print x
```

```
7.4161984871
```

2.2.3 Dictionary

Dictionaries are special types of python lists which allow lookup. We will use a dictionary when we look at converting DNA sequences to protein.

```
In [32]: # Creating dictionaries
         num_dict = { 1: 'one', 2: 'two', 3: 'three', 4: 'four' }
         num_dict[3]
         #num_dict[5]
         num_dict[5]= 'five'
         num_dict[20] = 'twenty'
         print num_dict.keys()
         print num_dict.values()
```

```
[1, 2, 3, 4, 5, 20]
['one', 'two', 'three', 'four', 'five', 'twenty']
```

2.3 Simple plotting with "matplotlib"

In [26]:
```
# Define x-axis and y-axis with arrays of points
import numpy as np
x = np.arange(0, 2*np.pi, .1) # Like range for lists
print x
y = np.sin(x)
print y
```

```
[ 0.   0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.   1.1  1.2  1.3  1.4
  1.5  1.6  1.7  1.8  1.9  2.   2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9
  3.   3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  4.   4.1  4.2  4.3  4.4
  4.5  4.6  4.7  4.8  4.9  5.   5.1  5.2  5.3  5.4  5.5  5.6  5.7  5.8  5.9
  6.   6.1  6.2]
[ 0.          0.09983342  0.19866933  0.29552021  0.38941834  0.47942554
  0.56464247  0.64421769  0.71735609  0.78332691  0.84147098  0.89120736
  0.93203909  0.96355819  0.98544973  0.99749499  0.9995736   0.99166481
  0.97384763  0.94630009  0.90929743  0.86320937  0.8084964   0.74570521
  0.67546318  0.59847214  0.51550137  0.42737988  0.33498815  0.23924933
  0.14112001  0.04158066 -0.05837414 -0.15774569 -0.2555411  -0.35078323
 -0.44252044 -0.52983614 -0.61185789 -0.68776616 -0.7568025  -0.81827711
 -0.87157577 -0.91616594 -0.95160207 -0.97753012 -0.993691   -0.99992326
 -0.99616461 -0.98245261 -0.95892427 -0.92581468 -0.88345466 -0.83226744
 -0.77276449 -0.70554033 -0.63126664 -0.55068554 -0.46460218 -0.37387666
 -0.2794155  -0.1821625  -0.0830894 ]
```

In [27]:
```
# Now import matplotlib library and plot
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()
```

In [28]:
```
# Make plot fancier with title, labels
plt.plot(x,y, 'r')
plt.title('Plot of x vs sin(x)')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.show()
```

Go to matplotlib gallery on internet for examples !!

In []: