1 Python as calculator

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

\begin{itemize}
\item \texttt{In [3]: 1 + 3}
\item \texttt{Out[3]: 4}
\item \texttt{In [4]: 3 * 90}
\item \texttt{Out[4]: 270}
\item \texttt{In [7]: 4 + 5 * 6 - 2}
\item \texttt{Out[7]: 32}
\item \texttt{In [8]: 2/5.0}
\item \texttt{Out[8]: 0.4}
\item \texttt{In [1]: 2/5}
\item \texttt{Out[1]: 0.4}
\end{itemize}

1.0.1 New ideas: Modulo and whole value division

\begin{itemize}
\item \texttt{In [10]: 8 \% 3}
\item \texttt{Out[10]: 2}
\item \texttt{In [11]: 9 \% 3}
\item \texttt{Out[11]: 0}
\item \texttt{In [12]: 8 // 3}
\item \texttt{Out[12]: 2}
\item \texttt{In [13]: 9 // 3}
\item \texttt{Out[13]: 3}
\end{itemize}
2 Variables and Assignments

In [3]: x = 6
   
   type(x)

Out[3]: int

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 [5]: x, y = 3, 4
   
   print(x,y)

3 4

In [7]: x, y = y, x
   
   print(x,y)

4 3

In [8]: z = x

In [9]: x = z * y
   
   print(x, y, z)

12 3 4
In [34]: x = z - y
    print x, y, z

1 3 4

In [10]: x = z**y - x
    print(x, y, z)

52 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 [12]: x = float(input("Enter value for x:"))
    print("The value of x is: ",x)

Enter value for x:5
The value of x is:   5.0

In [16]: x, y = 2, 3
    print("The value of x * y = \(d\) \(x+y\) \)

The value of x * y = 6
2.1 Packages and modules

In [17]: import numpy as np
   print (np.pi, np.cos(np.pi), np.e)

3.141592653589793 -1.0 2.718281828459045

In [18]: x = np.cos(np.pi) * np.sin(2*np.pi) + np.log(10)
   print ("x = %f" % (x))

x = 2.302585

In [19]: import random as random
   #help(random)
   random.randint(0,9)

Out[19]: 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 "elements" in the list do not have to be the same type.

2.2.1 Lists

In [23]: r = [ 1, 2, 3, 4 ,5, 10]
   r

Out[23]: [1, 2, 3, 4, 5, 10]

In [21]: r = [ 'Bob', 6, 78.98, True, 10, 1+8j]
   r

Out[21]: ['Bob', 6, 78.98, True, 10, (1+8j)]

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

In [24]: r = [1, 2, 3, 4, 5, 10]
   r.append(6)
   r.append('apples')
   r

Out[24]: [1, 2, 3, 4, 5, 10, 6, 'apples']

"Slicing" lets you access elements of the list
In [25]: r[3]
Out[25]: 4
In [26]: r[3] = 'oranges'
   r
Out[26]: [1, 2, 3, 'oranges', 5, 10, 6, 'apples']
In [27]: r[0:7]
Out[27]: [1, 2, 3, 'oranges', 5, 10, 6]
In [28]: r[-1]
Out[28]: 'apples'
In [29]: r[-1::-1]
Out[29]: ['apples', 6, 10, 5, 'oranges', 3, 2, 1]
In [36]: r_copy = r
   r_copy
Out[36]: [99, 2, 3, 'oranges', 5, 10, 6, 'apples']
In [37]: r[0] = 86
   r, r_copy
Out[37]: ([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 [35]: # To create separate copy of a list
   r_copy = r[:]
   r_copy
   r[0] = 99
   r, r_copy
Out[35]: ([99, 2, 3, 'oranges', 5, 10, 6, 'apples'],
   [86, 2, 3, 'oranges', 5, 10, 6, 'apples'])

In [38]: # Use range built-in function
   w = range(10)
   w
Out[38]: range(0, 10)
In [40]: w = list(range(1, 11))
   w
Out[40]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [42]: # get number of elements and sum of list
   : w = list(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 [46]: # Delete last element of list
   : w = list(range(10))
   : 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 work much faster than lists in python. We will use arrays more than lists.

In [47]: # 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 [48]: # 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 [49]: # 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 [50]: # 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 [51]: # 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 =  4.5  
   Mean value in array =  4.5  

In [52]: # 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 [53]: # Arithmetic with arrays
   array5 = np.array([0,1,2, 3,4,5])
   x = array5**2
   print (x)

   [ 0  1  4  9 16 25]

In [54]: x = np.sqrt ( sum(array5**2))
   print (x)

   7.4161984871

2.2.3 Strings

A string is a sequence of characters, usually used to store text

In [60]: the_string = 'Hello World!'  
In [62]: the_string[4]
   the_string[:4]
In [62]: 'Hell'
Out[62]: 'Hell'

In [81]: the_string.split()
Out[81]: ['Hello', 'World!']

In [65]: the_string.split('r')
Out[65]: ['Hello W', 'o!']

In [67]: words = ['this', 'is', 'a', 'list', 'of', "strings"]
    ' '.join(words)
Out[67]: 'this is a list of strings'

In [69]: 'ABC'.join(words)
Out[69]: 'thisABCisABCaABClistABCofABCstrings'

In [71]: ''.join(words)
Out[71]: 'thisisalistofstrings'

2.2.4 Tuples

Tuple consist of a number of values separated by commas. They are useful for ordered pairs and returning several values from a function.

In [89]: point1 = (3, 4, 5)
    print(point1)
    print(point1[0])

    (3, 4, 5)
    3

In [90]: x1, y1, z1 = point1  # Unpacking a tuple
    print(x1, y1, z1)

    3 4 5

2.2.5 Dictionaries

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 [91]: # 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())
dict_keys([1, 2, 3, 4, 5, 20])
dict_values(['one', 'two', 'three', 'four', 'five', 'twenty'])

In [92]: print(num_dict.items())
   for key,value in num_dict.items():
     print (key,value)
l = list(num_dict.items())
print(l)
dict_items([(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four'), (5, 'five'), (20, 'twenty')])
1 one
2 two
3 three
4 four
5 five
20 twenty
[(1, 'one'), (2, 'two'), (3, 'three'), (4, 'four'), (5, 'five'), (20, 'twenty')]

2.3 Simple plotting with "matplotlib"

In [56]: # 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]

In [57]: # Now import matplotlib library and plot
   import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()

In [58]: # 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 !!