Python Take Cares

Ana Bell, Nitish Mittal
MIT 6.00.1x Course on Python Programming
nitish_mittal [at] outlook.com

HELP from Documentation

dir(module)
help()

Important Characters and Sets of Characters

Tab	\t
New Line	\n
Backslash	\\
String	” ” or ’ ’
Boolean	True or False
Line Comment	#comment
Block Comment	”””” ”””” or ”’ ”’

Order of Operations (Emory)

Operator	Description
()	Parentheses (grouping)
f(args...)	Function call
x[index:index]	Slicing
x[index]	Subscription
x.attribute	Attribute reference
**	Exponentiation
+x, -x	Positive, negative
*, /, %	Mul, Div, Modulo
+, -	Addition, subtraction
in, not in, is, is not $<, <=, >, >=, <>, != , ==$	Membership, Identity, Comparison
not x	Boolean NOT
and	Boolean AND

Variable Names

• case sensitive
• cannot start with a number (ex, 1_assd is not allowed)

Six Steps to Defining a Function

1. What should your function do? Type a couple of example calls.
2. Pick a meaningful name (often a verb or verb phrase): What is a short answer to ”What does your function do”?
3. Decide how many parameters the function takes and any return values
4. Describe what your function does and any parameters and return values in the docstring
5. Write the body of the function
6. Test your function. Think about edge cases.

Typecasting and ”ast” module

>>> int(45)
45
>>> int(’45’)
45
>>> str(45)
’45’
>>> str(’45’)
’45’
>>> int(str(45))
45

To convert a string of the form of a list or a dict to its original form, we use the ast module. Important when reading from a file.
$>>> s =' [1, 2, 3]'$
$>>> ast.literal\_eval(s)$
$[1, 2, 3]$

Calling Methods

module_name.function_name(x)
• math.sqrt(x)
• random.randrange(2,5)
• ast.literal_eval(’$\{1 : 2, 3 : 4\}$’)

String Operators

Description	Operator	Example	Output
equality	==	’cat’ == ’cat’	True
inequality	!=	’cat’ != ’Cat’	True
less than	<	’A’ < ’a’	True
greater than	>	’a’ > ’A’	True
less than or equal	<=	’a’ <= ’a’	True
greater than or equal	>=	’a’ >= ’A’	True
contains	in	’cad’ in ’abracadabra’	True
length of str s	len(s)	len(”abc”)	3

String Indexing and Slicing

• s[i] means character at ith position
• s[a:b] means index a to length (b-a) or a to b index but not including b
• s[::-1] gives the reverse of a string
• String is immutable (ex. s[4]=’a’ will not replace ’a’ and index 4 of s)
• Like for strings, slicing and indexing can also be used for lists

List Functions

length of list	len(list)
smallest element in list	min(list)
largest element in list	max(list)
sum of elements of list (list items are numeric)	sum(list)

List Methods

append a value or string	list.append(’a’)
extended by another list	list.extend([’a’, ’b’])
removes the first occurrence of value	list.remove(’a’)
reverses the list	list.reverse()
count the occurrence of value	list.count(’a’)
sort list in increasing order	list.sort()
returns index of first occurrence of value	list.index(’a’)

$>>> a = [5] + [6] + ['a', 7]$
$>>> print(a)$
$[5, 6,' a', 7]$

List Mutability

We say that lists are mutable: they can be modified.

```
>>> lst = [1, 2, 3]
>>> lst[0] =' apple'
>>> lst
['apple', 2, 3]
```

List Aliasing

```
>>> lst1 = [11, 12, 13, 14, 15, 16, 17]
>>> lst2 = lst1
>>> lst1[-1] = 18
>>> lst2
[11, 12, 13, 14, 15, 16, 18]
```

After the second statement executes, *lst1* and *lst2* both refer to the same list. When two variables refer to the same objects, they are aliases. If that list is modified, both of *lst1* and *lst2* will see the change. This is also known as **Deep Copy**.

```
>>> lst1 = [1, 2, 3]
>>> lst2 = lst1[:]
>>> lst3 = lst1.copy()
>>> lst2.remove(2)
>>> lst3.remove(3)
>>> lst1
[1, 2, 3]
```

Both *lst2* and *lst3* are **Shallow Copies** of *list1*.

Be careful about:

```
>>> lst1 = [11, 12, 13, 14, 15, 16, 17]
>>> lst2 = lst1
>>> lst1 = [5, 6] #Reference of lst1 is changed
>>> lst2
[11, 12, 13, 14, 15, 16, 17]
```

Dict

- The values of a dictionary can be of any type, but the keys must be of an immutable data type such as strings, numbers, or tuples.
- Keys can be numbers, strings, Booleans
- A list is unhashable in a dictionary (cannot be used as a key)
- A tuple is hashable in a dictionary (can be used as a key).
- Values can be dicts, strings, numbers, booleans, lists

```
for key in my_dict:
    value = my_dict[key]
```

This is same as:
for key, value in my_dict.items():

Set

A set is a data structure in which all the elements are unique.

```
>>> a = set()
>>> a.add(1)
>>> a.add(2)
>>> a.add(3)
>>> a
{1, 2, 3}
>>> a.add(1)
>>> a
{1, 2, 3}
>>> a.remove(1)
>>> a
{2, 3}
```

Consider two sets:

```
>>> a = {1, 2, 3, 4}
>>> b = {3, 4, 5, 6}
```

Union

```
>>> a.union(b)
{1, 2, 3, 4, 5, 6}
```

Intersection

```
>>> a.intersection(b)
{3, 4}
```

Difference

```
>>> a.difference(b)
{1, 2}
```

Global and Local Variables

Variables defined outside functions are global variables. Their values may be accessed inside functions without declaration.

To modify to a global variable inside a function, the variable must be declared inside the function using the keyword *global*.

```
def x():
    global num
    num = 5
def y():
    num = 4

>>> num = 7
>>> print(num)
7
>>> x()
>>> print(num)
5
>>> y()
>>> print(num)
5
```

Reading CSV Files

Consider a CSV file with the content as:

Name	Gender	Age
John	Male	21
Natasha	Female	20
Scarlett	Female	24

```
>>> import csv
>>> f = open('info.csv',' r')
```

Method - List

```
>>> csv1 = csv.reader(f)
>>> for row in csv1 :
>>>     print(row[0])
Name
John
Natasha
Scarlett
```

Method - Dict

```
>>> csv2 = csv.DictReader(f)
>>> for row in csv2 :
>>>     print(row['Name'])
John
Natasha
Scarlett
```