TATA ELXSI

Python Scripting

Learning & Development Team

Monday, November 2, 2020

Module – 2 : Data Structures

Disclaimer

- This material is developed for in house training at **TATA ELXSI**.
- *TATA ELXSI is not responsible for any errors or omissions, or for the results obtained from the use of this information. All information in this material is provided "as is", with no guarantee of completeness, timeliness or of the results obtained from the use of this information...*
- *This is a training material and the information contained herein is not intended to be a source of advice or research outcome with respect to the material presented, and the information and/or documents/examples contained in this material do not constitute authors or TATA ELXSI's advice.*
- The training material therefore was solely developed for educational purposes. Currently, commercialization of the prototype does not exist, nor is the prototype available for general usage.
- **TATA ELXSI**, its staff, its students or any other participants **only** can use it as a part of training. This material should not be found on any website.
- For the preparation of this material we have referred from the below mentioned links or books.
- Excerpts of these material has been taken where there is no copy right infringements.

Agenda

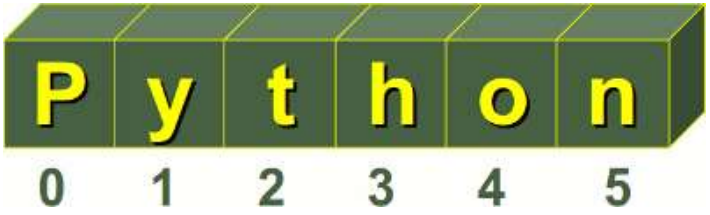
- Sequences
  - Strings
  - Lists
  - Tuples
    - Membership
    - Concatenation
    - Repetition
    - Slices
    - Conversions
    - Built-in functions
  - Implementing Data Structures like Stack and Queue

strings

- Strings are amongst the most popular types in Python.
- We can create them simply by enclosing characters in quotes.
- Python treats single quotes the same as double quotes.

```
>>> x = "tata"
>>> y = "Elxsi"
>>> print("The concat of", x ," & ", y," is : ",x+y)
The concat of tata  &  Elxsi  is :  tataElxsi
```

```
>>> lang = "Python"
>>> lang[2]
't'
```



strings

```
#!/usr/bin/python

var1 = 'Hello World!'
var2 = "Python Programming"

print("var1[0]: ", var1[0])
print("var2[1:5]: ", var2[1:5])
```

Output:

```
var1[0]: H
var2[1:5]: ytho
```

```
#!/usr/bin/python
var1 = 'Hello World!'
Print("Updated String :- ", var1[:6] +
   'Python‘)
```

Output:

```
Updated String :- Hello Python
```

Operator	Description	Example
+	Concatenation - Adds values on either side of the operator	a + b will give HelloPython
*	Repetition - Creates new strings, concatenating multiple copies of the same string	a*2 will give -HelloHello
[]	Slice - Gives the character from the given index	a[1] will give **e**
[:]	Range Slice - Gives the characters from the given range	a[1:4] will give **ell**
in	Membership - Returns true if a character exists in the given string	**H in a** will give 1
not in	Membership - Returns true if a character does not exist in the given string	**M not in a** will give 1
r/R	Raw String - Suppresses actual meaning of Escape characters. The syntax for raw strings is exactly the same as for normal strings with the exception of the raw string operator, the letter "r," which precedes the quotation marks. The "r" can be lowercase (r) or uppercase (R) and must be placed immediately preceding the first quote mark.	**print r'\n'** prints \n and **print R'\n'** prints \n
%	Format - Performs String formatting	See at next section

strings

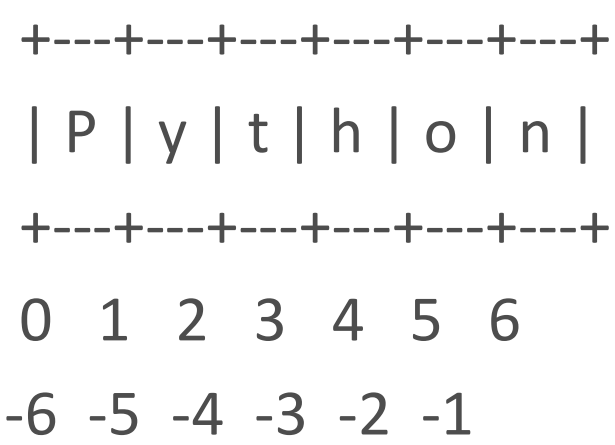
- Strings can be indexed (subscripted), with the first character having index 0.

```
>>> word = 'Python'
>>> word[0]  # character in position 0
'P'
>>> word[5]  # character in position 5
'n'

>>> word[-1]  # last character
'n'
>>> word[-2]  # second-last character
'o'
>>> word[-6]
'P'
```

```
>>> word[0:2]  # characters from position 0 (included) to 2 (excluded)
'Py'
>>> word[2:5]  # characters from position 2 (included) to 5 (excluded)
'tho‘

>>> word[:2] + word[2:]
'Python'
>>> word[:4] + word[4:]
'Python'
```

strings

```
>>> word[:2]  # character from the beginning to
   position 2 (excluded)
'Py'
>>> word[4:]  # characters from position 4
   (included) to the end
'on'
>>> word[-2:] # characters from the second-last
   (included) to the end
'on'
```



String Formatting Operator:

- One of Python's coolest features is the string format operator **%**.

```
#!/usr/bin/python

>>print ("My name is %s and weight is %d kg!" % (‘Akash’, 61))
```

String Formatting Operator:

Format Symbol	Conversion
%c	character
%s	string conversion via str() prior to formatting
%i	signed decimal integer
%d	signed decimal integer
%u	unsigned decimal integer
%o	octal integer
%x	hexadecimal integer (lowercase letters)
%X	hexadecimal integer (UPPERcase letters)
%e	exponential notation (with lowercase 'e')
%E	exponential notation (with UPPERcase 'E')
%f	floating point real number
%g	the shorter of %f and %e
%G	the shorter of %f and %E

Tuples

- key = (lastname, firstname)
- point = x, y, z # parentheses optional
- x, y, z = point # unpack
- lastname = key[0]
- singleton = (1,) # trailing comma!!!
- empty = () # parentheses!
- tuples vs. lists; tuples immutable

List in Python

Python Lists:

- The list is a most versatile data type available in Python which can be written as a list of comma-separated values (items) between square brackets.

- Good thing about a list is that items in a list need not all have the same type.

```
>>> list1 = ['physics', 'chemistry', 1997, 2000]
>>> list2 = [1, 2, 3, 4, 5 ]
>>> list3 = ["a", "b", "c", "d"]

>>>  print "list1[0]: ", list1[0]
>>>  print "list2[1:5]: ", list2[1:5]
```

Python Lists:

```
>>> letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']
>>> letters
['a', 'b', 'c', 'd', 'e', 'f', 'g']

>>> # replace some values
>>> letters[2:5] = ['C', 'D', 'E']

>>> letters
['a', 'b', 'C', 'D', 'E', 'f', 'g']
```

```
>>> # now remove them
>>> letters[2:5] = []
>>> letters
['a', 'b', 'f', 'g']

>>> # clear the list by replacing all the elements
    with an empty list
>>> letters[:] = []
>>> letters
[]
```

Lists

- Flexible arrays, *not* Lisp-like linked lists
  - a = [99, "bottles of beer", ["on", "the", "wall"]]
- Same operators as for strings
  - a+b, a*3, a[0], a[-1], a[1:], len(a)
- Item and slice assignment
  - a[0] = 98
  - a[1:2] = ["bottles", "of", "beer"]

-> [98, "bottles", "of", "beer", ["on", "the", "wall"]]

  - del a[-1] # -> [98, "bottles", "of", "beer"]

More List Operations

```
>>> a = range(5)                # [0,1,2,3,4]
>>> a.append(5)                 # [0,1,2,3,4,5]
>>> a.pop()                     # [0,1,2,3,4]
5
>>> a.insert(0, 42)             # [42,0,1,2,3,4]
>>> a.pop(0)                    # [0,1,2,3,4]
42
>>> a.reverse()                 # [4,3,2,1,0]
>>> a.sort()                    # [0,1,2,3,4]
```

list.extend(L)

- list.extend(L): Extend the list by appending all the items in the given list.

```
>>> numlist=range(0,10,2)
>>> numlist
[0, 2, 4, 6, 8]
>>> numlist.extend([1,3,5])
>>> numlist
[0, 2, 4, 6, 8, 1, 3, 5]
>>>
```

list.append(x)

- list.append(x): Add an item to the end of the list.

```
>>> numlist=range(0,10,2)
>>> numlist.append(range(10,16,2))
>>> numlist
[0, 2, 4, 6, 8, [10, 12, 14]]
>>> numlist[5]
[10, 12, 14]
>>> numlist[5][0]
10
>>> len(numlist)
6
>>>
>>> numlist[5].extend([5,55,555])
>>> numlist
[0, 2, 4, 6, 8, [10, 12, 14, 5, 55, 555]]
>>>
```

list.remove(x):

- **list.remove(x):** Remove the first item from the list whose value is x. It is an error if there is no such item.

```
>>> numlist=[0,2,4,6,8,10]
>>> numlist
[0, 2, 4, 6, 8, 10]
>>> numlist.remove(2)
>>> numlist
[0, 4, 6, 8, 10]
>>> numlist(100)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    numlist(100)
TypeError: 'list' object is not callable
>>>
```

list.clear():

- **list.clear():** Remove all items from the list. Equivalent to del a[:].

```
>>> numlist.clear()
>>> numlist
[ ]
>>>
```

- **list.index(x) :** Return the index in the list of the first item whose value is x. It is an error if there is no such item.

```
>>> numlist
[1, 2, 3, 5, 1, 2, 5, 3, 4, 5]
>>> numlist.index(5)
3
>>> numlist.index(55)
```

list.count(x) and list.reverse()

- **list.count(x) :** Return the number of times x appears in the list.

```
>>> numlist.count(5)
3
>>> numlist.count(50)
0
>>>
```

list.reverse(): Reverse the elements of the list in place.

```
>>> numlist
[1, 2, 3, 5, 1, 2, 5, 3, 4, 5]
>>> numlist.reverse()
>>> numlist
[5, 4, 3, 5, 2, 1, 5, 3, 2, 1]
>>>
```

list.copy()

list.copy() : Return a shallow copy of the list. Equivalent to a[:].

```
>>> numlist
[5, 4, 3, 5, 2, 1, 5, 3, 2, 1]
>>> numlist2=numlist
>>> id(numlist)
36753120
>>> id(numlist2)
36753120
>>> x=numlist.copy()
>>> x
[5, 4, 3, 5, 2, 1, 5, 3, 2, 1]
>>> id(x)
37422544
```

```
>>> numlist[5]=555
>>> numlist
[5, 4, 3, 5, 2, 555, 5, 3, 2, 1]
>>> numlist2
[5, 4, 3, 5, 2, 555, 5, 3, 2, 1]
>>> x
[5, 4, 3, 5, 2, 1, 5, 3, 2, 1]
>>>
```

Lists : example

```
>>>which_one = input("What month (1-12)? ")

>>>months = [’January’, ’February’, ’March’, ’April’, ’May’, ’June’, \ ’July’, ’August’, ’September’, ’October’,
’November’, \ ’December’]

>>>if 1 <= which_one <= 12:
...         print("The month is",months[which_one - 1])
```

Using Lists as Stacks

- To add an item to the top of the stack, use append().
- To retrieve an item from the top of the stack, use pop() without an explicit index.

```
>>> stack = [3, 4, 5]
>>> stack.append(6)
>>> stack.append(7)
>>> stack
[3, 4, 5, 6, 7]
>>> stack.pop()
 7
>>> stack
[3, 4, 5, 6]
>>> stack.pop()
6
>>> stack.pop()
5
>>> stack [3, 4]
```

Using Lists as Queues

- To implement a queue, use collections.deque which was designed to have fast appends and pops from both ends.

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")          # Terry arrives
>>> queue.append("Graham")         # Graham arrives
>>> queue.popleft()    # The first to arrive now leaves 'Eric'
>>> queue.popleft()    # The second to arrive now leaves 'John'
>>> queue              # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

Sets

- Python also includes a data type for sets.
- A set is an unordered collection with no duplicate elements.
- Basic uses include membership testing and eliminating duplicate entries.
- Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Set

- Example:

```
>>> x = set("A Python Tutorial")
>>> x {'A', ' ', 'i', 'h', 'l', 'o', 'n', 'P', 'r', 'u', 't', 'a', 'y', 'T'}
>>> type(x) <class 'set'>
>>>
```

- x = set(["Perl", "Python", "Java","Ruby","Python","Perl"])
- >>> x
- {'Python', 'Java', 'Perl', 'Ruby'}

Set

```
>>> a = set('abracadabra')
>>> b= set('alacazam')
>>> a
{'r', 'c', 'd', 'a', 'b'}
>>> b
{'c', 'l', 'z', 'a', 'm'}
>>> a - b
{'r', 'd', 'b'}              # letters in a but not in b
>>> a | b                   # letters in either a or b
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                   # letters in both a and b
{'a', 'c'}
>>> a ^ b                   # letters in a or b but not both
{'r', 'd', 'b', 'm', 'z', 'l'}
```

Dictionaries

- Dictionaries consist of pairs (called items) of keys and their corresponding values.

- Hash tables, "associative arrays“

```
>>> dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
```

- Lookup:

```
>>>dict[“Beth"] -> “3452"

>>>dict["back"] # raises KeyError exception
```

Delete Dictionary Elements:

- You can either remove individual dictionary elements or clear the entire contents of a dictionary.
- You can also delete entire dictionary in a single operation.

```
#!/usr/bin/python
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}

del dict['Name']; # remove entry with key 'Name'
dict.clear(); # remove all entries in dict
del dict ; # delete entire dictionary

print "dict['Age']: ", dict['Age'];
print "dict['School']: ", dict['School'];
```

More Dictionary Ops

- Keys, values, items:
  - d.keys() -> ["duck", "back"]
  - d.values() -> ["duik", "rug"]
  - d.items() -> [("duck","duik"), ("back","rug")]
- Presence check:
  - d.has_key("duck") -> 1; d.has_key("spam") -> 0
- Values of any type; keys almost any
  - {"name":"Guido", "age":43, ("hello","world"):1,
    42:"yes", "flag": ["red","white","blue"]}

Dictionary Details

- Keys must be immutable:
  - numbers, strings, tuples of immutables
    - these cannot be changed after creation
  - reason is hashing (fast lookup technique)
  - not lists or other dictionaries
    - these types of objects can be changed "in place"
  - no restrictions on values
- Keys will be listed in arbitrary order
  - again, because of hashing

Variables

- No need to declare
- Need to assign (initialize)
- use of uninitialized variable raises exception
- Not typed

```
if friendly: greeting = "hello world"

else: greeting = 12**2

print greeting
```

- Everything is a "variable":
  - Even functions, classes, modules

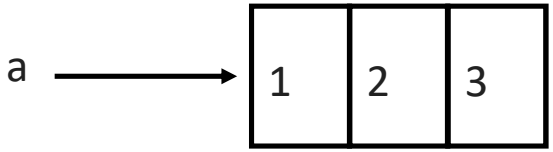
Reference Semantics

- Assignment manipulates references
  - x = y does not make a copy of y
  - x = y makes x reference the object y references
- Very useful; but beware!
- Example:

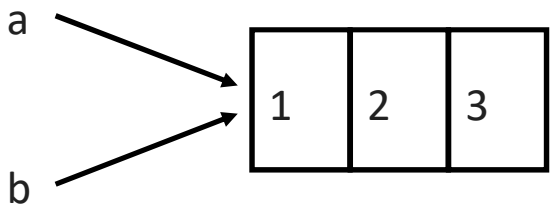
```
>>> a = [1, 2, 3]

>>> b = a

>>> a.append(4)

>>> print b

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

Changing a Shared List

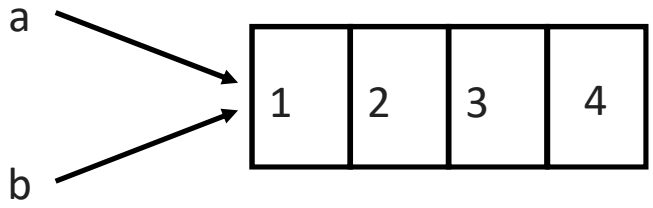
a = [1, 2, 3]



b = a

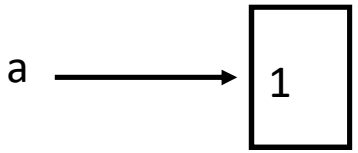


a.append(4)

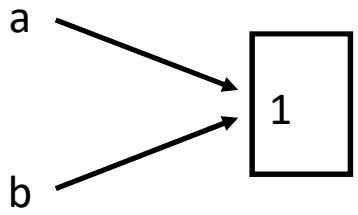


Changing an Integer

a = 1

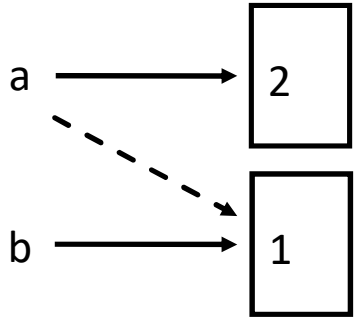


b = a



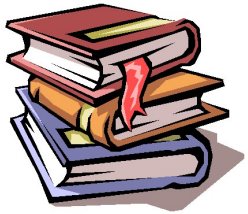
new int object created by add operator (1+1)

a = a+1

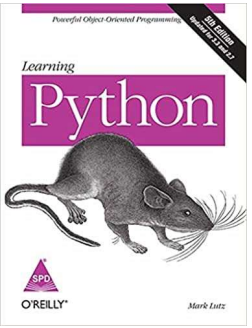


old reference deleted by assignment (a=...)

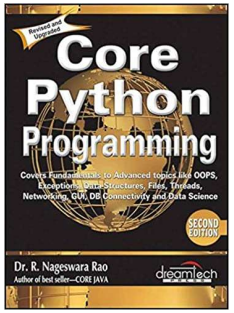
References



- **Python 3.x.x documentation**: https://docs.python.org/3/



- **Learning Python: Powerful Object-Oriented Programming: 5th Edition**



- **Core Python Programming [**by R. Nageswara Rao (Author)]

Thank you

Learning & Development

Tata Elxsi

Bangalore

www.tataelxsi.com

Confidentiality Notice

This document and all information contained herein is the sole property of Tata Elxsi Limited and shall not be reproduced or disclosed to a third party without the express written consent of Tata Elxsi Limited.