SNIST
Sreenidhi Institute of Science and Technology
Because, Life is all about taking the right decisions
AUTONOMOUS

PYTHON PROGRAMMING

Lecture 3 Unit 2

LISTS and TUPLES

Name: Mr. Dheeraj Sundaragiri

Assistant Professor

Department of Computer Science and Engineering

SNIST

LISTS

- Accessing list
- Operations
- Working with lists Function and Methods

LISTS

- The list is the most versatile datatype available in Python, which can be written as a list of comma-separated values (items) between square brackets.
- Important thing about a list is that the items in a list need not be of the same type.

Example

list1= ['physics', 'chemistry', 1997, 2000]

ACCESSING LISTS

- To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

```
list1 = ['physics', 'chemistry', 1997, 2000]
list2 = [1, 2, 3, 4, 5, 6, 7 ]
print ("list1[0]: ", list1[0])      # list1[0]: physics
print ("list2[1:5]: ", list2[1:5]) # list2[1:5]: [2, 3, 4, 5]
```

UPDATING LISTS

Example:

```
list = ['physics', 'chemistry', 1997, 2000]
print ("Value available at index 2 : ", list[2])
list[2] = 2001
print ("New value available at index 2 : ", list[2])
```

Output:

```
Value available at index 2 : 1997
New value available at index 2 : 2001
```

Delete List Elements

Example:

```
list = ['physics', 'chemistry', 1997, 2000]
print (list)
del list[2]
print ("After deleting value at index 2 : ", list)
```

Output:

```
['physics', 'chemistry', 1997, 2000]
After deleting value at index 2 : ['physics', 'chemistry', 2000]
```

Basic List Operations

Python Expression	Results	Description
len([1,2,3])	3	Length
[1,2,3]+[4,5,6]	[1,2,3,4,5,6]	Concatenation
[“hi”]*3	[“hi”,“hi”,“hi”]	Repetition
3 in [1,2,3]	True	Membership
for x in [1,2,3] : print (x,end=' ')	1 2 3	Iteration

Indexing, Slicing and Matrixes

l=['c','python','java']		
Python Expression	Results	Description
l[2]	'java'	Offset starts at zero
l[-2]	'python'	Negative: count from right
l[1:]	['python', 'java']	slicing fetches sections

For matrix

```
matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
>>> matrix[1]                #[4, 5, 6]
>>> matrix[1][1]             #5
>>> matrix[2][0]             #7
>>> matrix = [[1, 2, 3],
... [4, 5, 6],
... [7, 8, 9]]
>>> matrix[1][1]             #5
```

more Indexing

```
>>> L = ['spam', 'Spam', 'SPAM!']
>>> L[1] = 'eggs'        # Index assignment
>>> L
Output      ['spam', 'eggs', 'SPAM!']
>>> L[0:2] = ['eat', 'more']   # Slice
assignment:          delete+insert
>>> L # Replaces items 0,1
['eat', 'more', 'SPAM!']
```

List Iteration and Comprehensions

Example:

```
new_range = []
for i in range(5):
    if i % 2 == 0:
        new_range.append(i*i)
```

List comprehension:

```
new_range  = [i * i  for i in range(5)   if i % 2 == 0]
```

List Methods

append
extend
insert
remove
pop
clear
index
count
sort
reverse
copy

append() # list.append(item)

Add an element to the end of the list

```
>>> a=[1,2,3]
>>> a.append(2)
>>> a   #  [1, 2, 3, 2]
>>> a.append([21,22])
>>> a   #  [1, 2, 3, 2, [21, 22]]
```

List Methods

~~append~~
extend
insert
remove
pop
clear
index
count
sort
reverse
copy

extend() # list1.extend(list2)

Add all elements of a list to the another list

```
>>> a=[1,2,3]
>>> b=[4,5,6]
>>> a.extend(b)
>>> a
[1,2,3,4,5,6]
```

List Methods

~~append~~
~~extend~~
insert
remove
pop
clear
index
count
sort
reverse
copy

insert() # list.insert(index, element)

Insert an item at the defined index

```
>>> a=[1,2,3]
>>> a.insert(1,11)# [1, 11, 2, 3]
```

remove() # list.remove(element)

Removes an item from the list

```
>>> a.remove(11) # [1, 2, 3]
```

List Methods

~~append~~
~~extend~~
~~insert~~
~~remove~~
pop
clear
index
count
sort
reverse
copy

pop() # list.pop(index)

Removes and returns an element at the given index

clear() #list.clear()

Removes all items from the list

```
>>> a=[1,2,3]
>>> a.pop(1)     #  o/p: 2, a=[1, 3]
>>> a.clear()    #  []
```

List Methods

~~append~~
~~extend~~
~~insert~~
~~remove~~
~~pop~~
~~clear~~
index
count
sort
reverse
copy

index() # list.index(element)

Returns the index of the first matched item

```
>>> a=[1,2,3,1,2,3]
>>> a.index(2)  # 1
```

count() # list.count(element)

Returns the count of number of items passed as an argument

```
>>> a.count(1) # 2
```

List Methods

~~append~~
~~extend~~
~~insert~~
~~remove~~
~~pop~~
~~clear~~
~~index~~
~~count~~
sort
reverse
copy

sort() # list.sort(key=...,reverse=...)

- Sort items in a list in ascending order
- If true, the sorted list is reversed (or sorted in Descending order)

```
>>> a=[1,2,3,1,2,3]
>>> a.sort() # [1, 1, 2, 2, 3, 3]
>>> a.sort(reverse=True)
#  [3, 3, 2, 2, 1, 1]
```

List Methods

~~append~~
~~extend~~
~~insert~~
~~remove~~
~~pop~~
~~clear~~
~~index~~
~~count~~
~~sort~~
reverse
copy

reverse() # list.reverse()

Reverse the order of items in the list

```
>>> a=[1,2,3]
>>> a.reverse()  #    [3, 2, 1]
```

copy()

Returns a shallow copy of the list

```
>>> b = a          #    [3, 2, 1]
>>> c=a.copy()     #    [3, 2, 1]
```

TUPLES

- Accessing Tuples
- Operations
- Working with Tuples Function and Methods

TUPLES

- Tuples construct simple groups of objects. They work exactly like lists, except that tuples can’t be changed in place (they’re **immutable**) and are usually written as a series of items in parentheses, not square brackets.

Example

```
tuple1= ('physics', 'chemistry', 1997, 2000)
```

PROPERTIES OF TUPLES

- Ordered collections of arbitrary objects
- Accessed by offset
- Of the category “immutable sequence”
- Fixed-length, heterogeneous, and arbitrarily nestable
- Arrays of object references

Example

tuple2= (1,'a', [1,2,3],('a','b','c'))

ACCESSING TUPLES

- To access values in tuples, use the square brackets for slicing along with the index or indices to obtain value available at that index.

```
tuple1 = ('physics', 'chemistry', 1997, 2000)
tuple2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tuple1[0]: ", tuple1[0]) # tuple1[0]: physics
print ("tuple2[1:5]: ", tuple2[1:5]) # tuple2[1:5]: [2, 3, 4, 5]
```

UPDATING TUPLES

Example:

```
tuple1 = ['physics', 'chemistry', 1997, 2000]
print ("Value available at index 2 : ", tuple1[2])
#   Value available at index 2 : 1997
tuple1[2] = 2001
#...TypeError: 'tuple' object does not support item assignment
del tuple1[2]
#...TypeError: 'tuple' object doesn't support item deletion
```

Basic Tuple Operations

Python Expression	Results	Description
len((1,2,3))	3	Length
(1,2,3)+(4,5,6)	(1,2,3,4,5,6)	Concatenation
("hi",)*3	('hi', 'hi', 'hi')	Repetition
3 in (1,2,3)	True	Membership
for x in (1,2,3) : print (x,end=' ')	1 2 3	Iteration

Indexing, Slicing and Matrixes

l=('c','python','java')		
Python Expression	Results	Description
l[2]	'java'	Offset starts at zero
l[-2]	'python'	Negative: count from right
l[1:]	('python', 'java')	slicing fetches sections

Tuple Iteration and Comprehensions

Example:

```
(i * i  for i in range(5)   if i % 2 == 0)

<generator object <genexpr> at 0x7fbbf67dfa98>
```

Using a Generator object

```
for value in (i * i for i in range(5) if i%2==0):
...     print(value)
```

Tuple Methods

count(x) : Returns the number of items x

index(x) : Returns the index of the first item that is equal to x

```
my_tuple = ('a','p','p','l','e',)

print(my_tuple.count('p'))  # Output: 2

print(my_tuple.index('l'))  # Output: 3
```