浙江理工大学
ZHEJIANG SCI-TECH UNIVERSITY

3. List, Tuple, String

《Python programming》 / Lecturer：Zhiyi Luo (罗志一)

School of Computer Science and Technology
计算机科学与技术学院

Lists

- Python list is an ordered sequence of items.
- Ordered: Maintain the orde of the data insertion
- Mutable: List is mutable and we can modify items
- Heterogeneous: List can contain data of different types
- Contains duplicate: Allows duplicates data

L = [20, ‘Jessa’, ‘35.75, [30, 60, 90]]

Lists

- The list data structure is very flexible it has many unique inbuilt functionalities like **pop(), append(),** etc which makes it easier, where the data keeps changing.
- As Lists are mutable it is used in applications where the values of the items change frequently.

List Constants

- List constants are surrounded by square brackets and the elements in the list are separated by commas
- A list element can be any Python object – even another list
- A list can be empty

```
>>> print([1, 24, 76])
[1, 24, 76]
>>> print(['red', 'yellow',
'blue'])
['red', 'yellow', 'blue']
>>> print(['red', 24, 98.6])
['red', 24, 98.6]
>>> print([ 1, [5, 6], 7])
[1, [5, 6], 7]
>>> print([])
[]
```

Creating a Python list

- Using **list()** constructor
- Using square bracket **[]**
  - Make an empty list using []

```
>>> empty_list = []
>>> empty_list
[]
>>> another_empty_list = list()
>>> another_empty_list
[]
```

Convert Other Data Type to Lists with list()

- Python’s list() function converts other data types to lists. The following example converts a string to a list of one-character strings:

```
>>> list('cat')
['c', 'a', 't']

>>> a_tuple = ('ready', 'fire', 'aim')
>>> list(a_tuple)
['ready', 'fire', 'aim']
```

Chop a string into a list with split()

- Python’s list() function converts other data types to lists. The following example converts a string to a list of one-character strings:

```
>>> birthday = '1/6/1952'
>>> birthday.split('/')
['1', '6', '1952’]

>>> splitme = 'a/b//c/d///e'
>>> splitme.split('/')
['a', 'b', '', 'c', 'd', '', '', 'e’]

>>> splitme = 'a/b//c/d///e'
>>> splitme.split('//')
['a/b', 'c/d', '/e']
```

Get an Item by using [offset]

- As with strings, you can extract a single value from a list by specifying its offset:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0]
ma'Groucho'
>>> marxes[1]
'Chico'
>>> marxes[2]
'Harpo’
```

```
>>> marxes[-1]
'Harpo'
>>> marxes[-2]
'Chico'
>>> marxes[-3]
'Groucho'
```

Get an Item by using [offset]

- The offset has to be a valid one for this list

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> marxes[-5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Lists of Lists

- Lists can contain elements of different types, including other lists

```
>>> small_birds = ['hummingbird', 'finch']
>>> extinct_birds = ['dodo', 'passenger pigeon', 'Norwegian Blue']
>>> carol_birds = [3, 'French hens', 2, 'turtledoves']
>>> all_birds = [small_birds, extinct_birds, 'macaw', carol_birds]
>>> all_birds
[['hummingbird', 'finch'], ['dodo', 'passenger pigeon', 'Norwegian
Blue'], 'macaw', [3, 'French hens', 2, 'turtledoves']]
>>> all_birds[0]
['hummingbird', 'finch']
```

Looking Inside Lists

- Just like strings, we can get at any single element in a list using an index specified in square brackets


```
>>> friends = [ 'Joseph', 'Glenn', 'Sally' ]
>>> print(friends[1])
Glenn
```

Lists are mutable

- Strings are “immutable” – we cannot change the contents of a string – we must make a new string to make any change
- Lists are “mutable” – we can change an element of a list using the index operator

```
>>> fruit = 'Banana'
>>> fruit[0] = 'b'
Traceback
TypeError: 'str' object does not
support item assignment
>>> x = fruit.lower()
>>> print(x)
banana
```

```
>>> lotto = [2, 14, 26, 41, 63]
>>> print(lotto)
[2, 14, 26, 41, 63]
>>> lotto[2] = 28
>>> print(lotto)
[2, 14, 28, 41, 63]
```

Get a slice to extract items by offset range

- Slicing a list implies, accessing a range of elements in a list.
  - Syntax for slicing: **listname[start_index: end_index: step]**
  - The start_index denotes the index position from where the slicing should begin and the end_index parameter denotes the index positions till which the slicing should be done.
  - The step allows you to take each nth-element within a start_index:end_index range, default as 1.

Get a slice to extract items by offset range

- You can extract a subsequence of a list by using a slice:

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes[0:2]
['Groucho', 'Chico']
```

- As with strings, slices can step by values other than one. The next example starts at the beginning and goes right by 2:

```
>>> marxes[::2]
['Groucho', ‘Harpo']
```

Get a slice to extract items by offset range

- Here, we start at the end and go left by 2:

```
>>> marxes[::-2]
[‘Harpo’, 'Groucho']
```

- And finally, the trick to reverse a list:

```
>>> marxes[::-1]
['Harpo', 'Chico', 'Groucho']
```

The elements in the list can be accessed from right to left by using negative indexing.

Adding elements to the list

- We can add a new element/list of elements to the list using the list methods such as append(), insert() and extend().

Append item at the end of the list

- The append() method will accept only one parameter and add it at the end of the list.
  - **listname.append(object)**

```
>>> my_list = list([5, 8, 'Tom', 7.50])
>>> my_list.append('Emma')
>>> print(my_list)
[5, 8, 'Tom', 7.5, 'Emma']
>>> my_list.append([25, 50, 75]) # append the nested list at the end
>>> print(my_list)
[5, 8, 'Tom', 7.5, 'Emma', [25, 50, 75]]
```

Add item at the specified position in the list

- Use the insert() method to add the object/item at the specified position in the list. The insert method accepts parameters position and object.
  - **listname.insert(index, object)**

```
>>> my_list = list([5, 8, 'Tom', 7.50])
>>> my_list.insert(2, 25) # insert 25 at position 2
>>> print(my_list)
[5, 8, 25, 'Tom', 7.5]
>>> # insert nested list at at position 3
>>> my_list.insert(3, [25, 50, 75])
>>> print(my_list)
[5, 8, 25, [25, 50, 75], 'Tom', 7.5]
```

Using extend()

- The extend method will accept the list of elements and add them at the end of the list. We can even add another list by using this method.
  - **listname.extend(list_object)**

```
>>> my_list = list([5, 8, 'Tom', 7.50])
>>> my_list.extend([25, 75, 100])
>>> print(my_list)
[5, 8, 'Tom', 7.5, 25, 75, 100]
```

Modify the items of a List

- The list is a mutable sequence of iterable objects. It means we can modify the items of a list. Use the index number and assignment operator (=) to assign a new value to an item.
  - Modify the individual item
  - Modify the range of items

```
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> my_list[0] = 20
>>> print(my_list)
[20, 4, 6, 8, 10, 12]
>>> my_list[1:4] = [40, 60, 80] # modify from 1st index to 4th
>>> print(my_list)
[20, 40, 60, 80, 10, 12]
```

Removing elements from a List

- The elements from the list can be removed using the following list methods.

method	Description
`remove(item)`	To remove the first occurrence of the item from the list.
`pop(index)`	Removes and returns the item at the given index from the list.
`clear()`	To remove all items from the list. The output will be an empty list.
`del list_name`	Delete the entire list.

Remove specific item

- Use the remove() method to remove the first occurrence of the item from the list.
- It throws a keyerror if an item not present in the original list.

```
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> my_list.remove(6)  # remove item 6
>>> my_list.remove(8)  # remove item 8
>>> print(my_list)
[2, 4, 10, 12]
```

Remove item present at given index

- Use the pop() method to remove the item at the given index. The pop() method removes and returns the item present at the given index. It will remove the last item from the list if the index number is not passed.

```
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> my_list.pop(2) # remove item present at index 2
6
>>> print(my_list)
[2, 4, 8, 10, 12]
>>> my_list.pop() # remove item without passing index number
12
>>> print(my_list)
[2, 4, 8, 10]
```

Remove the range of items

- Use del keyword along with list slicing to remove the range of items.

```
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> del my_list[2:5] # remove item from index 2 to 5
>>> print(my_list)
[2, 4, 12]

# remove all items starting from index 3
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> del my_list[3:]
>>> print(my_list)
[2, 4, 6]
```

Remove all items

- Use the clear() method to remove all items from the list.

```
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> my_list.clear() # clear list
>>> print(my_list)
[]
>>>
>>> # Delete entire list
>>> del my_list
```

Finding an element in the list

- Use the index() function to find an item in a list.
- The index() function will accept the value of the element as a parameter and returns the first occurrence of the element or returns ValueError if the element does not exist.

```
>>> my_list = list([2, 4, 6, 8, 10, 12])
>>> print(my_list.index(8))
3
```

Concatenation of two lists

- The concatenation of two lists means merging of two lists. There are two ways to do that.
  - Using the + operator.
  - Using the extend() method.

```
>>> my_list1 = [1, 2, 3]
>>> my_list2 = [4, 5, 6]
# Using + operator
>>> my_list3 = my_list1 + my_list2
>>> print(my_list3)
[1, 2, 3, 4, 5, 6]
```

Test for a value with in

- The Pythonic way to check for the existence of a value in a list is using **in**.

```
>>> marxes = ['Groucho', 'Chico', 'Harpo', 'Zeppo']
>>> 'Groucho' in marxes
True
>>> 'Bob' in marxes
False
```

Count occurrences of a value by using count()

- To count how many times a particular value occurs in a list, use count().

```
>>> marxes = ['Groucho', 'Chico', 'Harpo']
>>> marxes.count('Harpo')
1
>>> marxes.count('Bob')
0
```

Convert to a string with join()

- To count how many times a particular value occurs in a list, use count().

```
>>> friends = ['Harry', 'Hermione', 'Ron']
>>> separator = ' * '
>>>
>>> joined = separator.join(friends)
>>> joined
'Harry * Hermione * Ron'
>>> separated = joined.split(separator)
>>> separated
['Harry', 'Hermione', 'Ron']
>>> separated == friends
True
```

How long is a list?

- The len() function takes a list as a parameter and returns the number of elements in the list.
- Actually len() tells us the number of elements of any set or sequence (such as a string…)

```
>>> greet = 'Hello Bob'
>>> print(len(greet))
9
>>> x = [ 1, 2, 'joe', 99]
>>> print(len(x))
4
```

Copying a list

- Using assignment operator (=)

```
>>> l = [1,2,3]
>>> l1 = l
>>> l.append(0)
>>> l
[1, 2, 3, 0]
>>> l1
[1, 2, 3, 0]
```

- Using the slicing [:]

```
>>> l = [1,2,3]
>>> l1 = l[:]
>>> l.append(0)
>>> l
[1, 2, 3, 0]
>>> l1
[1, 2, 3]
```

Copying a list

- Using the copy() method

```
>>> l = [1,2,3]
>>> l1 = l.copy()
>>> l.append(0)
>>> l
[1, 2, 3, 0]
>>> l1
[1, 2, 3]
```

```
>>> l = [[1,2], 3]
>>> l1 = l.copy()
>>> l[0].append(0)
>>> l
[[1, 2, 0], 3]
>>> l1
[[1, 2, 0], 3]
```

```
>>> import copy
>>> l2 = copy.deepcopy(l)
>>> l2
[[1, 2, 0], 3]
>>> l
[[1, 2, 0], 3]
>>> l[0].append(0)
>>> l
[[1, 2, 0, 0], 3]
>>> l1
[[1, 2, 0, 0], 3]
>>> l2
[[1, 2, 0], 3]
```

Sort a list using sort()

- The sort function sorts the elements in the list in ascending order.

```
>>> mylist = [3,2,1]
>>> mylist.sort()
>>> print(mylist)
[1, 2, 3]
```

Reverse a list using reverse()

- The reverse function is used to reverse the elements in the list.

```
>>> mylist = [3, 4, 5, 6, 1]
>>> mylist.reverse()
>>> print(mylist)
[1, 6, 5, 4, 3]
```

Python Built-in functions with List

- In addition to the built-in methods available in the list, we can use the built-in functions as well on the list.
- Using max() & min()

```
>>> mylist = [3, 4, 5, 6, 1]
>>> print(max(mylist)) #returns the maximum number in the list.
6
>>> print(min(mylist)) #returns the minimum number in the list.
1
```

- Using sum()

```
>>> mylist = [3, 4, 5, 6, 1]
>>> print(sum(mylist))
19
```

Python Built-in functions with List

- all()
  - In the case of all() function, the return value will be true only when all the values inside the list are true. Let us see the different item values and the return values.

Item Values in List	Return Value
All Values are True	True
One or more False Values	False
All False Values	False
Empty List	True

Python Built-in functions with List

- any()
  - The any() method will return true if there is at least one true value. In the case of Empty List, it will return false.

Item Values in List	Return Value
All Values are True	True
One or more False Values	True
All False Values	False
Empty List	False

Tuples

- Similar to lists, tuples are sequences of arbitrary items. Unlike lists, tuples are immutable, meaning you can’t add, delete or change items after the tuple is defined. So, a tuple is similar to a constant list.

Creating a Tuple

- Using parenthesis()
  - A tuple is created by common-separated items inside rounded brackets.
- Using a **tuple()** constructor
  - Create a tuple by passing the comma-seperated items inside the tuple()

```
>>> t = (1,2,3)
>>> t
(1, 2, 3)

>>> t = (1,2,3, )
>>> t
(1, 2, 3)
```

```
>>> t = 1,2,3
>>> t
(1, 2, 3)

>>> one_marx = 'Groucho',
>>> one_marx
('Groucho',)
```

Creating a Tuple

- Using parenthesis()
  - A tuple is created by common-separated items inside rounded brackets.
- Using a **tuple()** constructor
  - Create a tuple by passing the comma-seperated items inside the tuple()
  - Make an empty list using ()

```
>>> empty_tuple = ()
>>> empty_
[]
>>> another_empty_list = list()
>>> another_empty_list
[]
```

Packing and Unpacking

- In Python, we can create a tuple by packing a group of variables. Packing can be used when we want to collect multiple values in a single variable. Generally, this operation is referred to as tuple packing.
- Similarly, we can unpack the items by just assigning the tuple items to the same number of variables. This process is called “Unpacking.”

```
# packing variables into tuple
>>> tuple1 = 1, 2, "Hello"
>>> print(tuple1)
(1, 2, 'Hello')
>>> print(type(tuple1))
<class 'tuple'>
```

```
# unpacking tuple into variable
>>> i, j, k = tuple1
>>> print(i, j, k)
```

```
1 2 Hello
```