**List** is a collection which is **ordered and changeable**. Allows duplicate members.

**Tuple** is a collection which is **ordered and unchangeable.**Allows duplicate members.

Set is a collection which is **unordered and unindexed**. No duplicate members.

Dictionary is a collection which is **ordered and changeable**. No duplicate members.

**Python Lists:**

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

Example

Create a List:

thislist = ["apple", "banana", "cherry"]

print(thislist)

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that lists are ordered, it means that the items have a defined order, and that order will not change.

If you add new items to a list, the new items will be placed at the end of the list.

Changeable

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

Allow Duplicates

Since lists are indexed, lists can have items with the same value:

Example

Lists allow duplicate values:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]

print(thislist)

List Length

To determine how many items a list has, use the len() function:

Example

Print the number of items in the list:

thislist = ["apple", "banana", "cherry"]

print(len(thislist))

List Items - Data Types

List items can be of any data type:

Example

String, int and boolean data types:

list1 = ["apple", "banana", "cherry"]

list2 = [1, 5, 7, 9, 3]

list3 = [True, False, False]

A list can contain different data types:

Example

A list with strings, integers and boolean values:

list1 = ["abc", 34, True, 40, "male"]

type()

From Python's perspective, lists are defined as objects with the data type 'list':

<class 'list'>

Example

What is the data type of a list?

mylist = ["apple", "banana", "cherry"]

The list() Constructor

It is also possible to use the list() constructor when creating a new list.

Example

Using the list() constructor to make a List:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets

print(thislist)

output:

['apple', 'banana', 'cherry']

Python - Change List Items

Change Item Value

To change the value of a specific item, refer to the index number:

Example

Change the second item:

thislist = ["apple", "banana", "cherry"]

thislist[1] = "blackcurrant"

print(thislist)

output:

['apple', 'blackcurrant', 'cherry']

**Change a Range of Item Values:**

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

Example

Change the values "banana" and "cherry" with the values "blackcurrant" and "watermelon":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]

thislist[1:3] = ["blackcurrant", "watermelon"]

print(thislist)

output:

['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']

If you insert more items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

Change the second value by replacing it with two new values:

thislist = ["apple", "banana", "cherry"]

thislist[1:2] = ["blackcurrant", "watermelon"]

print(thislist)

output: ['apple', 'blackcurrant', 'watermelon', 'cherry']

Example

Change the second and third value by replacing it with one value:

thislist = ["apple", "banana", "cherry"]

thislist[1:3] = ["watermelon"]

print(thislist)

output:

['apple', 'watermelon']

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert "watermelon" as the third item:

thislist = ["apple", "banana", "cherry"]

thislist.insert(2, "watermelon")

print(thislist)

output:

['apple', 'banana', 'watermelon', 'cherry']

**Append Items**

To add an item to the end of the list, use the append() method:

Example

Using the append() method to append an item:

thislist = ["apple", "banana", "cherry"]

thislist.append("orange")

print(thislist)

Output:

['apple', 'banana', 'cherry', 'orange']

**Insert Items:**

To insert a list item at a specified index, use the insert() method.

The insert() method inserts an item at the specified index:

Example

Insert an item as the second position:

thislist = ["apple", "banana", "cherry"]

thislist.insert(1, "orange")

print(thislist)

output:

['apple', 'orange', 'banana', 'cherry']

Extend List

To append elements from another list to the current list, use the extend() method.

Example

Add the elements of tropical to thislist:

thislist = ["apple", "banana", "cherry"]

tropical = ["mango", "pineapple", "papaya"]

thislist.extend(tropical)

print(thislist)

output:

['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by refering to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers

thislist = ["apple", "banana", "cherry"]

i = 0

while i <len(thislist):

print(thislist[i])

 i = i + 1

output:

apple

banana

cherry

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

Example

A short hand for loop that will print all items in a list:

thislist = ["apple", "banana", "cherry"]

[print(x) for x in thislist]

Output:

apple

banana

cherry

Remove Specified Item

The remove() method removes the specified item.

Example

Remove "banana":

thislist = ["apple", "banana", "cherry"]

thislist.remove("banana")

print(thislist)

output:

['apple', 'cherry']

Remove Specified Index

The pop() method removes the specified index.

Example

Remove the second item:

thislist = ["apple", "banana", "cherry"]

thislist.pop(1)

print(thislist)

output:

['apple', 'cherry']

If you do not specify the index, the pop() method removes the last item.

Example

Remove the last item:

thislist = ["apple", "banana", "cherry"]

thislist.pop()

print(thislist)

output:

['apple', 'banana']

The del keyword also removes the specified index:

Example

Remove the first item:

thislist = ["apple", "banana", "cherry"]

delthislist[0]

print(thislist)

output:

['banana', 'cherry']

The del keyword can also delete the list completely.

Example

Delete the entire list:

thislist = ["apple", "banana", "cherry"]

delthislist

output:null

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

Example

Clear the list content:

thislist = ["apple", "banana", "cherry"]

thislist.clear()

print(thislist)

output:

[]

Loop Through a List

You can loop through the list items by using a for loop:

Example

Print all items in the list, one by one:

thislist = ["apple", "banana", "cherry"]

for x in thislist:

print(x)

output:

apple

banana

cherry

Loop Through the Index Numbers:

You can also loop through the list items by referring to their index number.

Use the range() and len() functions to create a suitable iterable.

Example

Print all items by referring to their index number:

thislist = ["apple", "banana", "cherry"]

for i in range(len(thislist)):

print(thislist[i])

output:

apple

banana

cherry

Using a While Loop

You can loop through the list items by using a while loop.

Use the len() function to determine the length of the list, then start at 0 and loop your way through the list items by referring to their indexes.

Remember to increase the index by 1 after each iteration.

Example

Print all items, using a while loop to go through all the index numbers

thislist = ["apple", "banana", "cherry"]

i = 0

while i <len(thislist):

print(thislist[i])

 i = i + 1

output:

apple

banana

cherry

Looping Using List Comprehension

List Comprehension offers the shortest syntax for looping through lists:

Example

A short hand for loop that will print all items in a list:

thislist = ["apple", "banana", "cherry"]

[print(x) for x in thislist]

Output:

apple

banana

cherry

Python - Sort Lists:

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

Example

Sort the list alphabetically:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort()

print(thislist)

Output:

['banana', 'kiwi', 'mango', 'orange', 'pineapple']

Example

Sort the list numerically:

thislist = [100, 50, 65, 82, 23]

thislist.sort()

print(thislist)

Output:

[23, 50, 65, 82, 100]

Sort Descending

To sort descending, use the keyword argument reverse = True:

Example

Sort the list descending:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]

thislist.sort(reverse = True)

print(thislist)

output:

['pineapple', 'orange', 'mango', 'kiwi', 'banana']

Example

Sort the list descending:

thislist = [100, 50, 65, 82, 23]

thislist.sort(reverse = True)

print(thislist)

output:

[100, 82, 65, 50, 23]

Case Insensitive Sort

By default the sort() method is case sensitive, resulting in all capital letters being sorted before lower case letters:

Example

Case sensitive sorting can give an unexpected result:

thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.sort()

print(thislist)

output:

['Kiwi', 'Orange', 'banana', 'cherry']

Luckily we can use built-in functions as key functions when sorting a list.

So if you want a case-insensitive sort function, use str.lower as a key function:

Example

Perform a case-insensitive sort of the list:

thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.sort(key = str.lower)

print(thislist)

output:

['banana', 'cherry', 'Kiwi', 'Orange']

Reverse Order

What if you want to reverse the order of a list, regardless of the alphabet?

The reverse() method reverses the current sorting order of the elements.

Example

Reverse the order of the list items:

thislist = ["banana", "Orange", "Kiwi", "cherry"]

thislist.reverse()

print(thislist)

output:

['cherry', 'Kiwi', 'Orange', 'banana']

Python - Copy Lists:

Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

There are ways to make a copy, one way is to use the built-in List method copy().

Example

Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]

mylist = thislist.copy()

print(mylist)

output:

['apple', 'banana', 'cherry']

Another way to make a copy is to use the built-in method list().

Example

Make a copy of a list with the list() method:

thislist = ["apple", "banana", "cherry"]

mylist = list(thislist)

print(mylist)

Output:

['apple', 'banana', 'cherry']

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the + operator.

Example

Join two list:

list1 = ["a", "b", "c"]

list2 = [1, 2, 3]

list3 = list1 + list2

print(list3)

output:

['a', 'b', 'c', 1, 2, 3]

Another way to join two lists are by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

for x in list1:

 list1.append(x)

print(list1)

output:

['a', 'b', 'c', 1, 2, 3]

Example

Use the extend() method to add list2 at the end of list1:

list1 = ["a", "b" , "c"]

list2 = [1, 2, 3]

list1.extend(list2)

print(list1)

output:

['a', 'b', 'c', 1, 2, 3]