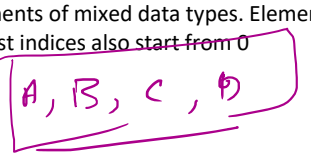
List

07 July 2024 02:06 PM

INTRODUCTION TO LIST

The data type list is an ordered sequence which is mutable and made up of one or more elements. Unlike a string which consists of only characters, a list can have elements of different data types, such as integer, float, string, tuple or even another list. A list is very useful to group together elements of mixed data types. Elements of a list are enclosed in square brackets and are separated by comma. Like string indices, list indices also start from 0



List

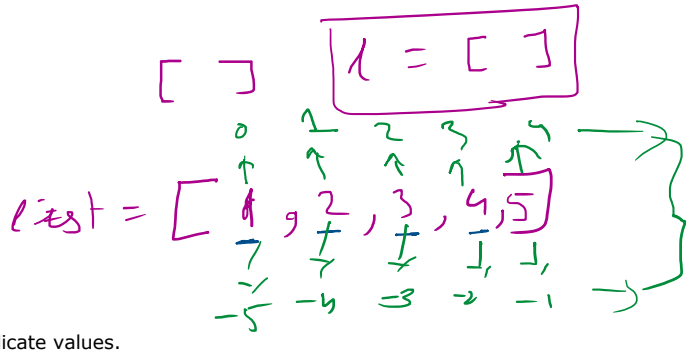
Lists are used to store multiple items in a single variable.
Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.
Lists are created using square brackets:

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.
Note: There are some list methods that will change the order, but in general: the order of the items will not change.

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"]

print(type(mylist))
```

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)
```

Access Items

List items are indexed and you can access them by referring to the index number:

Example

Print the second item of the list:

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

print(thislist[1])
```

Note: The first item has index 0.

Negative Indexing

Negative indexing means start from the end
`-1` refers to the last item, `-2` refers to the second last item etc.

Example

Print the last item of the list:

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

print(thislist[-1])
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.
When specifying a range, the return value will be a new list with the specified items.

Example

Return the third, fourth, and fifth item:

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

print(thislist[2:5])
```

Note: The search will start at index 2 (included) and end at index 5 (not included).
Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to, but NOT including, "kiwi":

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

print(thislist[:4])
```

By leaving out the end value, the range will go on to the end of the list:

Example

This example returns the items from "cherry" to the end:

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

print(thislist[2:])
```

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from "orange" (-4) to, but NOT including "mango" (-1):

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

print(thislist[-4:-1])
```

Check if Item Exists

To determine if a specified item is present in a list use the `in` keyword:

Example

Check if "apple" is present in the list:

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

if "apple" in thislist:

  print("Yes, 'apple' is in the fruits list")
```

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)
```

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)
```

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)
```

Note: The length of the list will change when the number of items inserted does not match the number of items replaced.
If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

Example

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

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

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

print(thislist)
```

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)
```

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)
```

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)
```

Extend List

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

Example

Add the elements of `tropical` to `thislist`:

The elements will be added to the *end* of the list.

Add Any Iterable

The `extend()` method does not have to append *lists*, you can add any iterable object (tuples, sets, dictionaries etc.).

Example

Add elements of a tuple to a list:

```
thislist = ["apple", "banana", "cherry"]
thistuple = ("kiwi", "orange")
thislist.extend(thistuple)
print(thislist)
```

Remove Specified Item

The `remove()` method removes the specified item.

Example

Remove "banana":

```
thislist = ["apple", "banana", "cherry"]
thislist.remove("banana")
print(thislist)
```

If there are more than one item with the specified value, the `remove()` method removes the first occurrence:

Example

Remove the first occurrence of "banana":

```
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Remove Specified Index

The `pop()` method removes the specified index.

Example

Remove the second item:

```
thislist = ["apple", "banana", "cherry"]
thislist.pop(1)
print(thislist)
```

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)
```

The `del` keyword also removes the specified index:

Example

Remove the first item:

```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```

The `del` keyword can also delete the list completely.

Example

Delete the entire list:

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

del thislist
```

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)
```

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)
```

Try it Yourself »

Example

Sort the list numerically:

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

thislist.sort()

print(thislist)
```

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)
```

Example

Sort the list descending:

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

thislist.sort(reverse = True)

print(thislist)
```

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)
```

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)
```

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)
```

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)
```

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)
```

List Methods

Python has a set of built-in methods that you can use on lists.

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list