# List Statement

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)
```

**Output**

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

## 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:

# 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))
```

**Output**

3

# 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]
```

# 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))
```

**Output**

```
<class 'list'>
```

# List Comprehension

**List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.**

### Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a `for` statement with a conditional test inside:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []
for x in fruits:
  if "a" in x:
    newlist.append(x)
print(newlist)
```

**Output**

```
['apple', 'banana', 'mango']
```

With list comprehension you can do all that with only one line of code:

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = [x for x in fruits if "a" in x]
print(newlist)
```

**Output**

['apple', 'banana', 'mango']

# The Syntax

```
newlist =
[expression for item in iterable if condition == True]
```

The return value is a new list, leaving the old list unchanged.

**Condition**

The *condition* is like a filter that only accepts the items that valuate to `True`.

## Example

Only accept items that are not "apple":

```
newlist = [x for x in fruits if x != "apple"]
```

**Output**

['banana', 'cherry', 'kiwi', 'mango']

The condition `if x != "apple"` will return `True` for all elements other than "apple", making the new list contain all fruits except "apple".

The *condition* is optional and can be omitted:

## Example

With no `if` statement:

```
newlist = [x for x in fruits]
```

['apple', 'banana', 'cherry', 'kiwi', 'mango']

### Iterable

The *iterable* can be any iterable object, like a list, tuple, set etc.

## Example

You can use the `range()` function to create an iterable:

```
newlist = [x for x in range(10)]
```

**Output:**

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Same example, but with a condition:

## Example

Accept only numbers lower than 5:

```
newlist = [x for x in range(10) if x < 5]
```

**Output**

[0, 1, 2, 3, 4]

### Expression

The *expression* is the current item in the iteration, but it is also the outcome, which you can manipulate before it ends up like a list item in the new list:

## Example

Set the values in the new list to upper case:

```
newlist = [x.upper() for x in fruits]
```

**Output**

['APPLE', 'BANANA', 'CHERRY', 'KIWI', 'MANGO']

You can set the outcome to whatever you like:

## Example

Set all values in the new list to 'hello':

```
newlist = ['hello' for x in fruits]
```

**Output**

['hello', 'hello', 'hello', 'hello', 'hello']

The *expression* can also contain conditions, not like a filter, but as a way to manipulate the outcome:

## Example

Return "orange" instead of "banana":

```
newlist = [x if x != "banana" else "orange" for x in fruits]
```

**Output**

['apple', 'orange', 'cherry', 'kiwi', 'mango']

The *expression* in the example above says:

***"Return the item if it is not banana, if it is banana return orange".***

# Access List Items

## 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])
```

Output

banana

## 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])
```

Output

cherry

## 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", "mang
o"]
print(thislist[2:5])
```

**Output**

['cherry', 'orange', 'kiwi']

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", "mang
o"]
print(thislist[:4])
```

**Output**

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

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", "mang
o"]
print(thislist[2:])
```

**Output**

['cherry', 'orange', 'kiwi', 'melon', 'mango']

## 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])
```

Output

['orange', 'kiwi', 'melon']

# 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")
```

**Output**

Yes, 'apple' is in the fruits list

# List Methods

## 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 |
| 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 |

# Join Lists

## 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]

## Example

Append list2 into list1:

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

**Output**

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

# Python List append() Method

Definition and Usage

The `append()` method appends an element to the end of the list.

## Syntax

```
list.append(elmnt)
```

## Parameter Values

| Parameter | Description |
|---|---|
| *elmnt* | Required. An element of any type (string, number, object etc.) |

### Example

Add an element to the `fruits` list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.append("orange")
```

**Output**

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

### Example

Add a list to a list:

```
a = ["apple", "banana", "cherry"]
b = ["Ford", "BMW", "Volvo"]
a.append(b)
```

**Output**

['apple', 'banana', 'cherry', ["Ford", "BMW", "Volvo"]]

# Python List clear() Method

## Example

Remove all elements from the `fruits` list:

```
fruits = ['apple', 'banana', 'cherry', 'orange']

fruits.clear()
```

**Output**

```
[]
```

# Python List insert() Method

Definition and Usage

The `insert()` method inserts the specified value at the specified position.

## Syntax

```
list.insert(pos, elmnt)
```

## Parameter Values

| Parameter | Description |
|---|---|
| *pos* | Required. A number specifying in which position to insert the value |
| *elmnt* | Required. An element of any type (string, number, object etc.) |

## Example

Insert the value "orange" as the second element of the `fruit` list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.insert(1, "orange")
```

**Output**

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

### Python List remove() Method

Definition and Usage

The `remove()` method removes the first occurrence of the element with the specified value.

## Syntax

```
list.remove(elmnt)
```

## Parameter Values

| Parameter | Description |
|---|---|
| *elmnt* | Required. Any type (string, number, list etc.) The element you want to remove |

## Example

Remove the "banana" element of the `fruit` list:

```
fruits = ['apple', 'banana', 'cherry']
fruits.remove("banana")
```

**Output**

['apple', 'cherry']

# Python List reverse() Method

Definition and Usage

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

## Syntax

```
list.reverse()
```

## Parameter Values

No parameters

### Example

Reverse the order of the `fruit` list:

```
fruits = ['apple', 'banana', 'cherry']

fruits.reverse()
```

**Output**

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