List in Python

List

- List is a mutable data type
- 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.
- 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:

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.

Defining a List

Lists are written within square brackets []

z =	[3,	7,	4,	2]
index	0	1	2	3

Defining a List. The second row in this table index is how you access items in the list.

```
# Define a list
z = [3, 7, 4, 2]
```

Create a List

There are several ways to create a new list; the simplest is to enclose the values in square brackets `[]`

```
# A list of integers
L = [1, 2, 3]

# A list of strings
L = ['red', 'green', 'blue']
```

The items of a list don't have to be the same type. The following list contains an integer, a string, a float, a complex number, and a boolean.

```
# A list of mixed datatypes
L = [ 1, 'abc', 1.23, (3+4j), True]
```

A list containing zero items is called an **empty list** and you can create one with emptybrackets `[]`

```
# An empty list
L = []
```

Access Values in a List

Each item in a list has an assigned index value. It is important to note that python is a zero indexed based language. All this means is that the first item in the list is at index 0.

z =	[3,	7,	4,	2]
index	0	1	2	3

Access item at index 0 (in blue)

```
# Define a list
z = [3, 7, 4, 2]

# Access the first item of a list at index 0
print(z[0])
```

```
print(z[0])
```

```
3
```

Python also supports negative indexing. Negative indexing starts from the end. It can be more convienient at times to use negative indexing to get the last item in the list because you don't have to know the length of the list to access the last item.

z =	[3,	7,	4,	2]
index	0	1	2	3
negative index	-4	-3	-2	-1

Accessing the item at the last index.

```
# print last item in the list
print(z[-1])
```

As a reminder, you could also access the same item using positive indexes (as seen below).

```
print(z[-1])
```

```
2
```

```
print(z[3])
```

```
2
```

Alternative way of accessing the last item in the list z

Python List Slicing

To access a range of items in a list, you need to slice a list. One way to do this is to use the simple slicing operator :

With this operator you can specify where to start the slicing, where to end and specify the step.

Slicing a List :

If L is a list, the expression L [start : stop : step] returns the portion of the list from index `start` to index `stop`, at a step size `step`.

Syntax

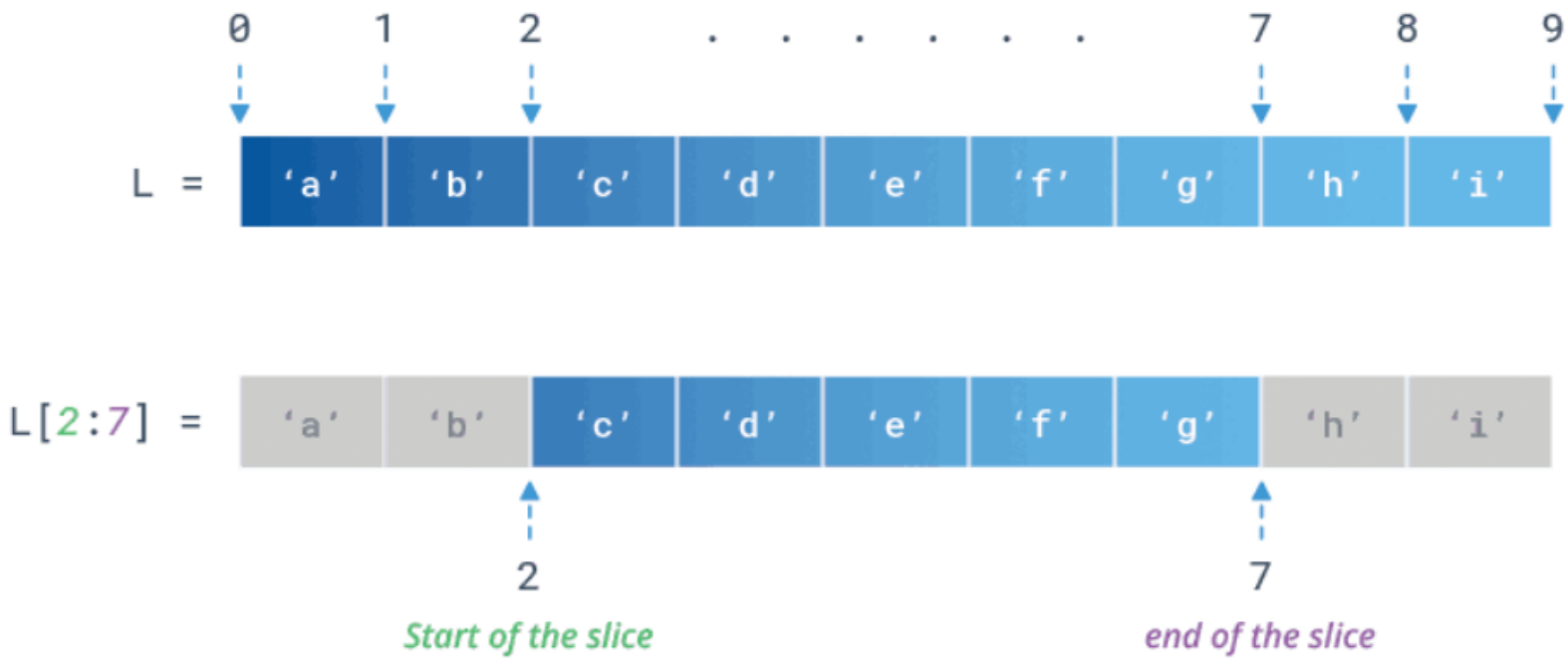
```
L[start:stop:step]
```

Start position *End position* *The increment*

Basic Example

Here is a basic example of list slicing.

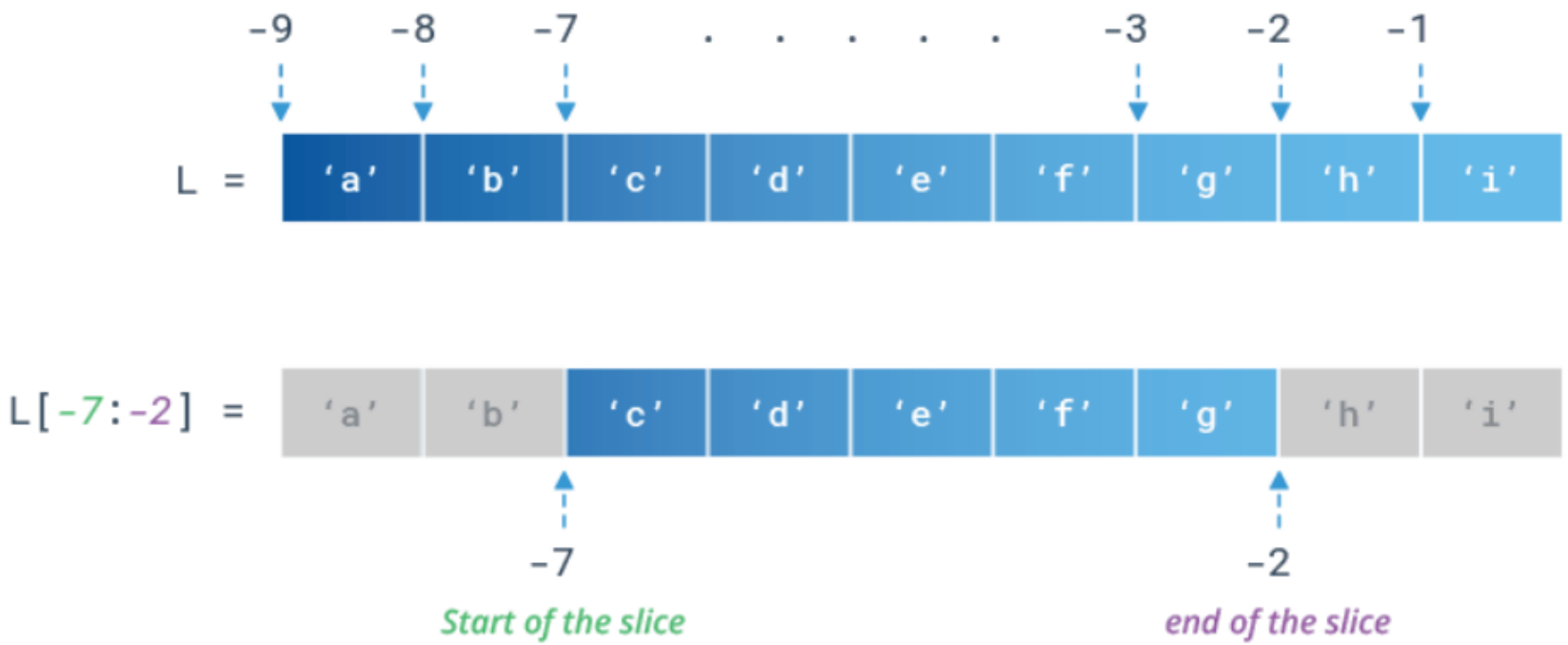
```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[2:7])
# Prints ['c', 'd', 'e', 'f', 'g']
```



Slice with Negative Indices

You can also specify negative indices while slicing a list.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[-7:-2])
# Prints ['c', 'd', 'e', 'f', 'g']
```



Slice with Positive & Negative Indices

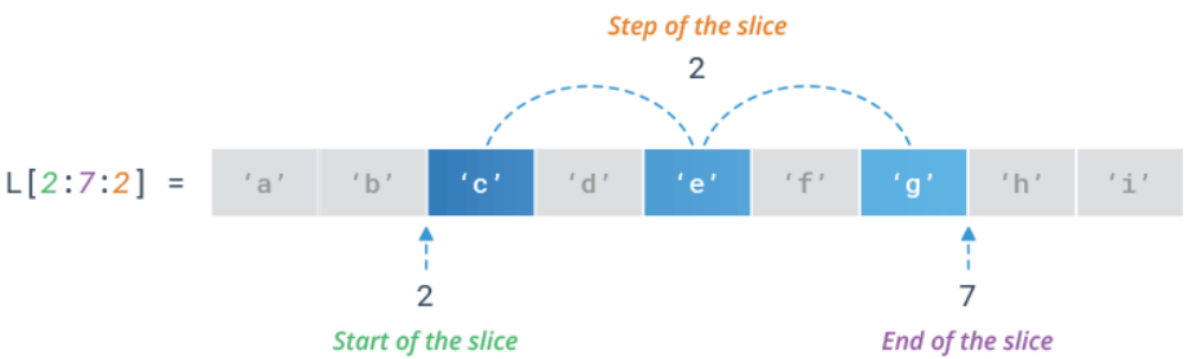
You can specify both positive and negative indices at the same time.

```
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[2:-5])
# Prints ['c', 'd']
```

Specify Step of the Slicing

You can specify the step of the slicing using `step` parameter. The `step` parameter is optional and by default 1.

```
# Return every 2nd item between position 2 to 7
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[2:7:2])
# Prints ['c', 'e', 'g']
```



Negative Step Size

You can even specify a negative step size.

```
# Return every 2nd item between position 6 to 1
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[6:1:-2])
# Prints ['g', 'e', 'c']
```

Slice at Beginning & End

Omitting the `start` index starts the slice from the index 0. Meaning, `L[:stop]` is equivalent to `L[0:stop]`

```
# Slice the first three items from the list
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[:3])
# Prints ['a', 'b', 'c']
```

Whereas, omitting the `stop` index extends the slice to the end of the list. Meaning, `L[start:]` is equivalent to `L[start:len(L)]`

```
# Slice the last three items from the list
L = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i']
print(L[6:])
# Prints ['g', 'h', 'i']
```

Reverse a List

You can reverse a list by omitting both `start` and `stop` indices and specifying a `step` as -1.

```
L = ['a', 'b', 'c', 'd', 'e']
print(L[::-1])
# Prints ['e', 'd', 'c', 'b', 'a']
```

Modify Multiple List Values

You can modify multiple list items at once with slice assignment. This assignment replaces the specified slice of a list with the items of assigned iterable.

```
# Modify multiple list items
L = ['a', 'b', 'c', 'd', 'e']
L[1:4] = [1, 2, 3]
print(L)
# Prints ['a', 1, 2, 3, 'e']
```

```
# Replace multiple elements in place of a single element
L = ['a', 'b', 'c', 'd', 'e']
L[1:2] = [1, 2, 3]
print(L)
# Prints ['a', 1, 2, 3, 'c', 'd', 'e']
```

Insert Multiple List Items

You can insert items into a list without replacing anything. Simply specify **a zero-length slice.**

```
# Insert at the start
L = ['a', 'b', 'c']
L[:0] = [1, 2, 3]
print(L)
# Prints [1, 2, 3, 'a', 'b', 'c']

# Insert at the end
L = ['a', 'b', 'c']
L[len(L):] = [1, 2, 3]
print(L)
# Prints ['a', 'b', 'c', 1, 2, 3]
```

You can insert items into the middle of list by keeping both the `start` and `stop` indices of the slice same.

```
# Insert in the middle
L = ['a', 'b', 'c']
L[1:1] = [1, 2, 3]
print(L)
# Prints ['a', 1, 2, 3, 'b', 'c']
```

Delete Multiple List Items

You can delete multiple items out of the middle of a list by assigning the appropriate slice to an empty list.

```
L = ['a', 'b', 'c', 'd', 'e']
L[1:5] = []
print(L)
# Prints ['a']
```

You can also use the del statement with the same slice.

```
L = ['a', 'b', 'c', 'd', 'e']
del L[1:5]
print(L)
# Prints ['a']
```

Clone or Copy a List

When you execute `new_List = old_List`, you don't actually have two lists. The assignment just copies the reference to the list, not the actual list. So, both `new_List` and `old_List` refer to the same list after the assignment.

You can use slicing operator to actually copy the list (also known as a shallow copy).

```
L1 = ['a', 'b', 'c', 'd', 'e']
L2 = L1[:]
print(L2)
# Prints ['a', 'b', 'c', 'd', 'e']
print(L2 is L1)
# Prints False
```

Slice of Lists

Slices are good for getting a subset of values in your list. For the example code below, it will return a list with the items from index 0 up to and not including index 2.

z =	[3,	7,	4,	2]
index	0	1	2	3

First index is inclusive (before the :) and last (after the :) is not

```
# Define a list
z = [3, 7, 4, 2]

print(z[0:2])
```

```
print(z[0:2])
```

```
[3, 7]
```

z =	[3,	7,	4,	2]
index	0	1	2	3

```
# everything up to but not including index 3
print(z[:3])
```

```
print(z[:3])
```

```
[3, 7, 4]
```

The code below returns a list with items from index 1 to the end of the list

```
# index 1 to end of list
print(z[1:])
```

```
print(z[1:])
```

```
[7, 4, 2]
```

Lists allow duplicate values:

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

Combine Lists

You can merge one list into another by using extend() method. It takes a list as an argument and appends all of the elements.

```
L = ['red', 'green', 'yellow']
L.extend([1,2,3])
print(L)
# Prints ['red', 'green', 'yellow', 1, 2, 3]
```

Alternatively, you can use the concatenation operator `+` or the augmented assignment operator `+=`

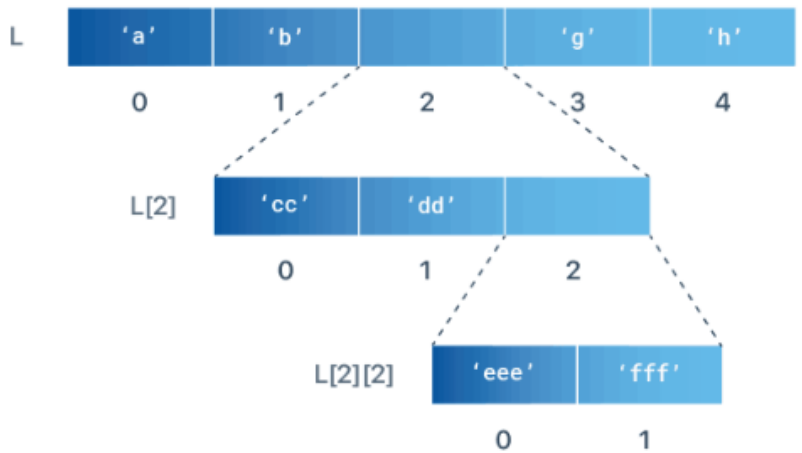
```
# concatenation operator
L = ['red', 'green', 'blue']
L = L + [1,2,3]
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]

# augmented assignment operator
L = ['red', 'green', 'blue']
L += [1,2,3]
print(L)
# Prints ['red', 'green', 'blue', 1, 2, 3]
```

Access Nested List Items

Similarly, you can access individual items in a nested list using multiple indexes. The first index determines which list to use, and the second indicates the value within that list.

The indexes for the items in a nested list are illustrated as below:




```
L = ['a', 'b', ['cc', 'dd', ['eee', 'fff']], 'g', 'h']

print(L[2][2])
# Prints ['eee', 'fff']

print(L[2][2][0])
# Prints eee
```