# Python unit 4 -part 2

List

## Python List

Python lists are one of the most versatile data types that allow us to work with multiple elements at once. For example,

```
# a list of programming languages
  ['Python', 'C++', 'JavaScript']
```

## Create Python Lists

In Python, a list is created by placing elements inside square brackets `[]`, separated by commas.

```
# list of integers

  my_list = [1, 2, 3]
```

A list can have any number of items and they may be of different types (integer, float, string, etc.).

```
# empty list

my_list = []


# list with mixed data types

  my_list = [1, "Hello", 3.4]
```

**A list can also have another list as an item. This is called a nested list.**

```
# nested list

  my_list = ["mouse", [8, 4, 6], ['a']]
```

## Access List Elements

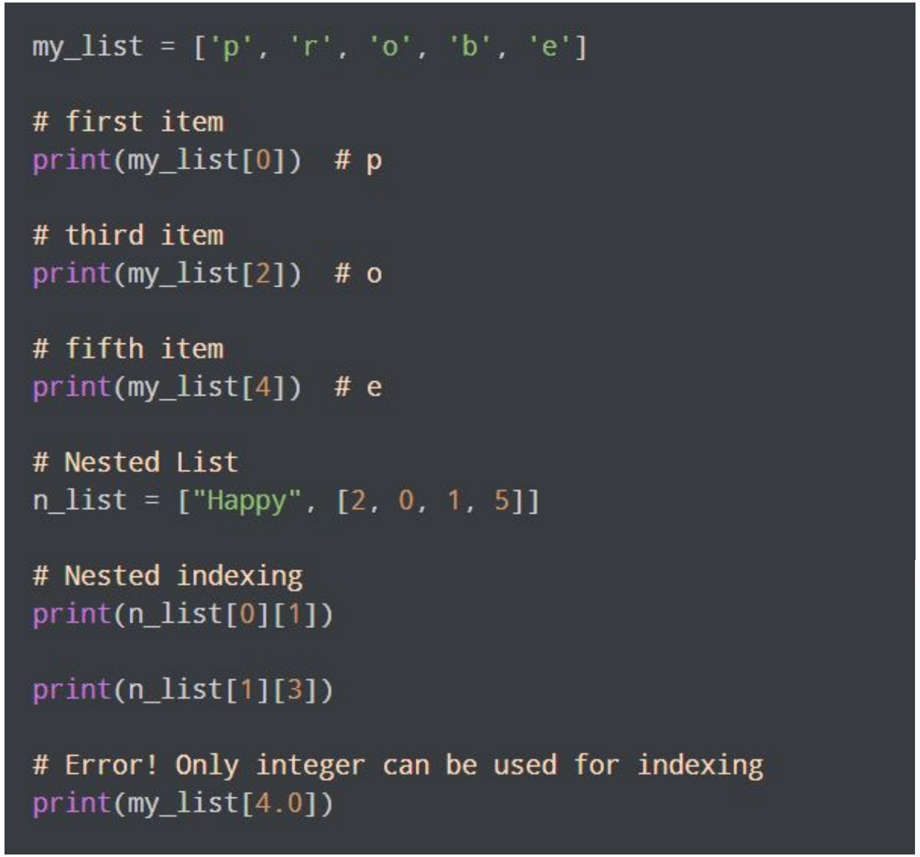
There are various ways in which we can access the elements of a list.

### List Index

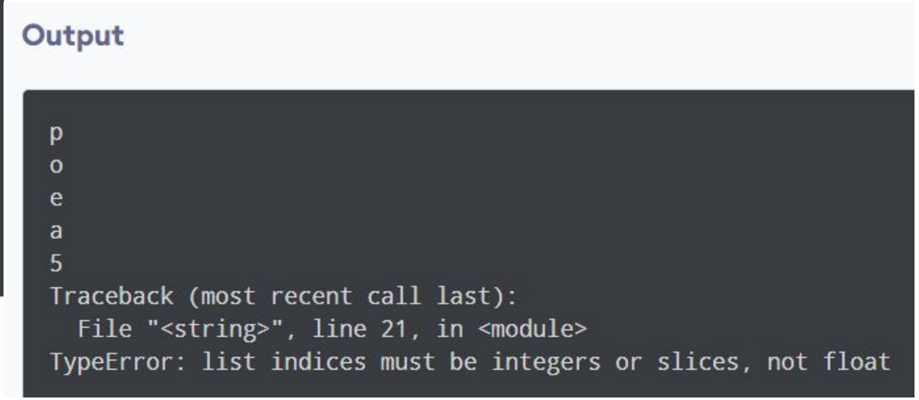
We can use the index operator `[]` to access an item in a list. In Python, indices start at 0. So, a list having 5 elements will have an index from 0 to 4.

Trying to access indexes other than these will raise an `IndexError`. The index must be an integer. We can't use float or other types, this will result in `TypeError`.

Nested lists are accessed using nested indexing.



```python
my_list = ['p', 'r', 'o', 'b', 'e']

# first item
print(my_list[0])  # p

# third item
print(my_list[2])  # o

# fifth item
print(my_list[4])  # e

# Nested List
n_list = ["Happy", [2, 0, 1, 5]]

# Nested indexing
print(n_list[0][1])

print(n_list[1][3])

# Error! Only integer can be used for indexing
print(my_list[4.0])
```



**Output**

```
p
o
e
a
5
Traceback (most recent call last):
  File "<string>", line 21, in <module>
TypeError: list indices must be integers or slices, not float
```

## Negative indexing

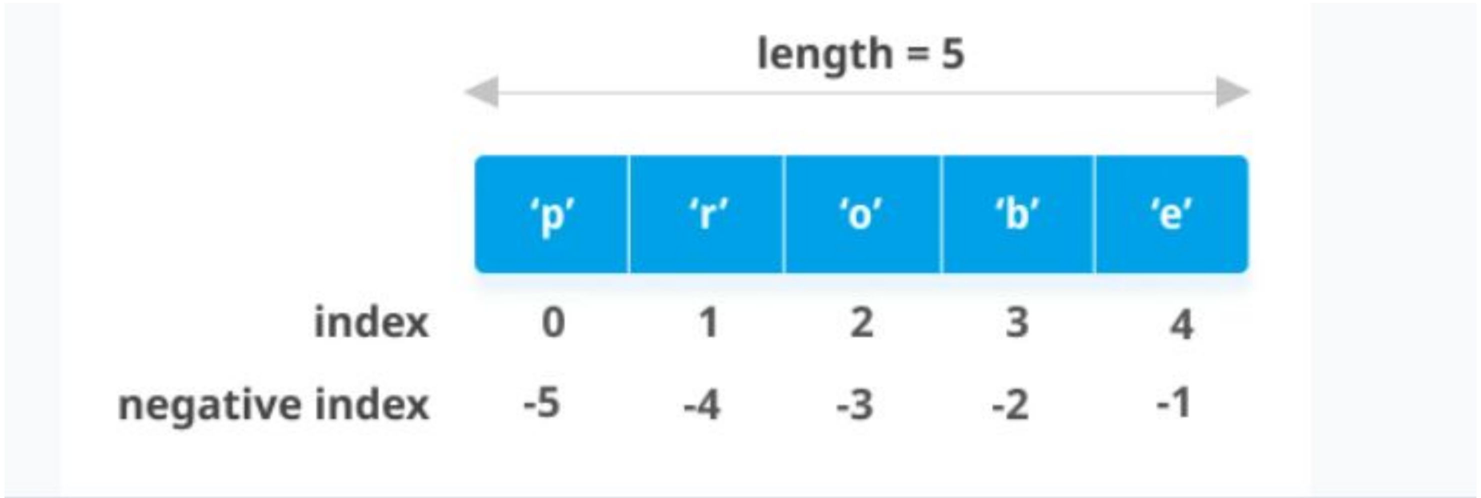
Python allows negative indexing for its sequences. The index of -1 refers to the last item, -2 to the second last item and so on.

```python
# Negative indexing in lists
my_list = ['p','r','o','b','e']

# last item
print(my_list[-1])

# fifth last item
print(my_list[-5])
```

Run Code »

### Output

```
e
p
```

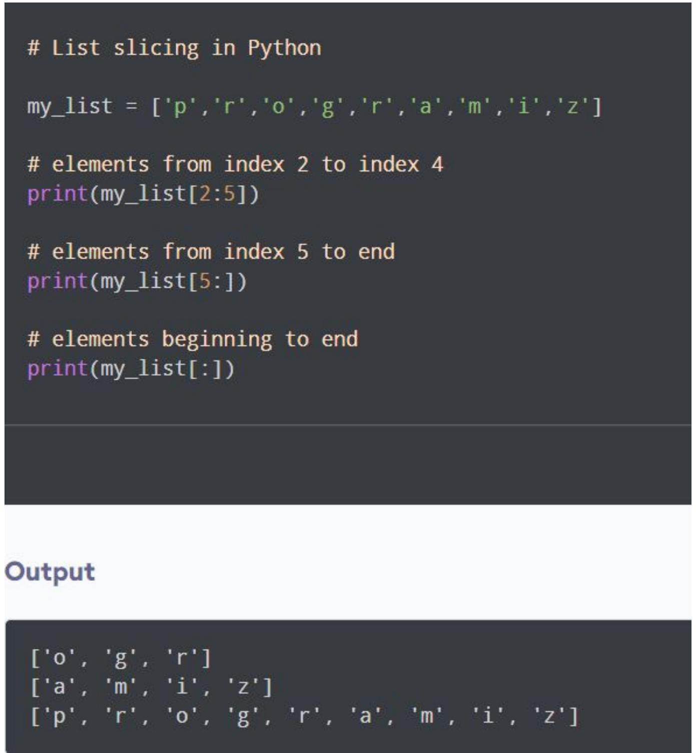


List indexing in Python

## List Slicing in Python

We can access a range of items in a list by using the slicing operator `:`.

Note: When we slice lists, the start index is inclusive but the end index is exclusive. For example, `my_list[2:5]` returns a list with elements at index 2, 3 and 4, but not 5.



```
# List slicing in Python

my_list = ['p','r','o','g','r','a','m','i','z']

# elements from index 2 to index 4
print(my_list[2:5])

# elements from index 5 to end
print(my_list[5:])

# elements beginning to end
print(my_list[:])
```

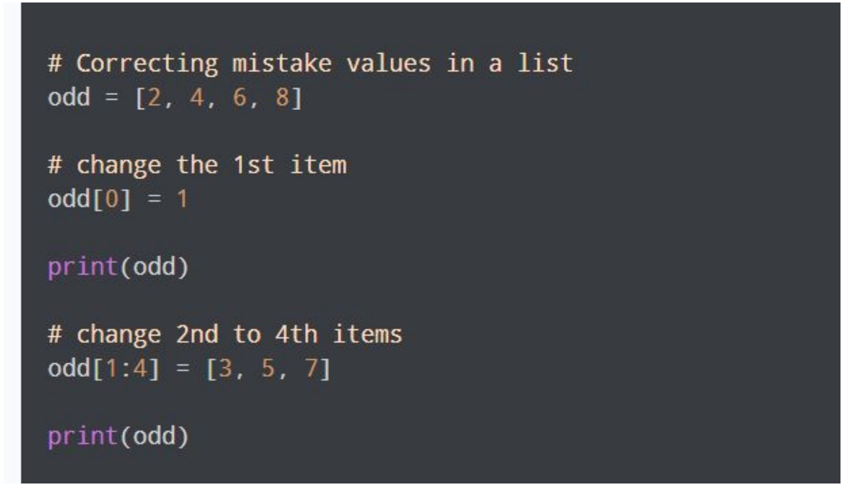
Output

```
['o', 'g', 'r']
['a', 'm', 'i', 'z']
['p', 'r', 'o', 'g', 'r', 'a', 'm', 'i', 'z']
```

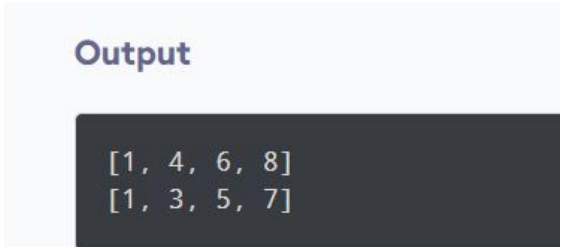
## Add/Change List Elements

Lists are mutable, meaning their elements can be changed unlike string or tuple.

We can use the assignment operator = to change an item or a range of items.



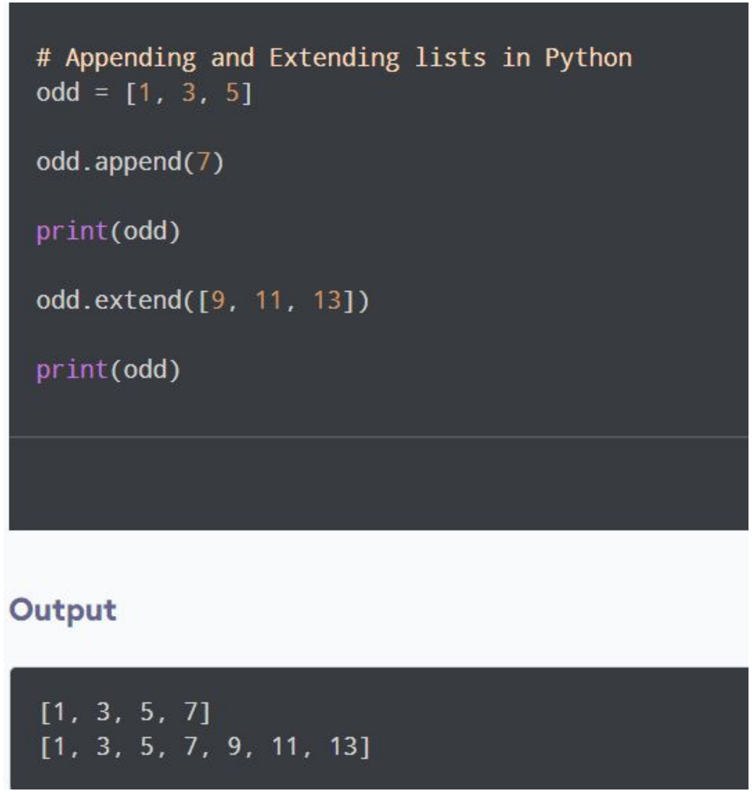
```
# Correcting mistake values in a list
odd = [2, 4, 6, 8]

# change the 1st item
odd[0] = 1

print(odd)

# change 2nd to 4th items
odd[1:4] = [3, 5, 7]

print(odd)
```



Output

```
[1, 4, 6, 8]
[1, 3, 5, 7]
```

We can add one item to a list using the `append()` method or add several items using the `extend()` method.



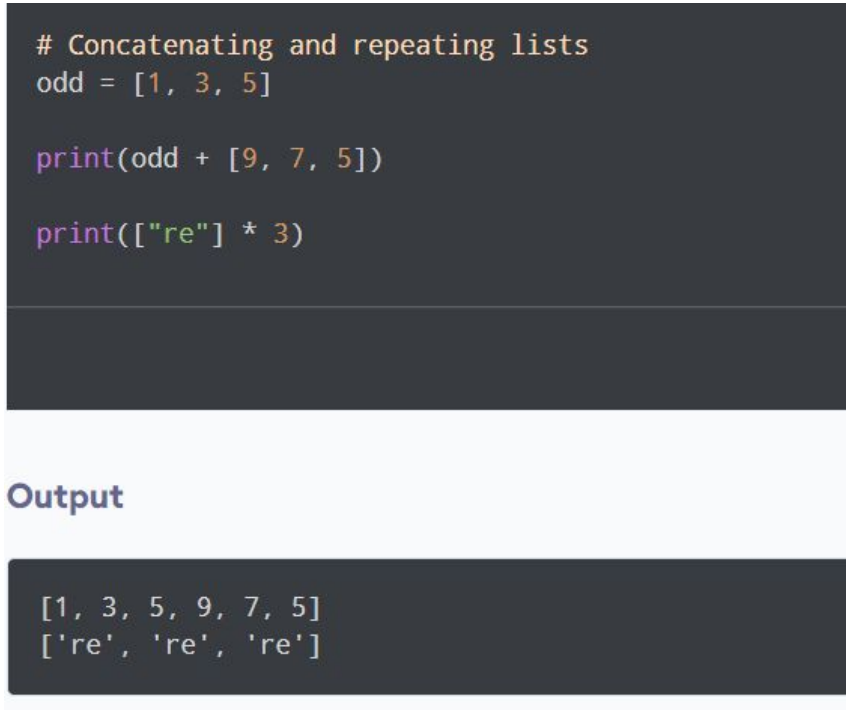
```python
# Appending and Extending lists in Python
odd = [1, 3, 5]

odd.append(7)

print(odd)

odd.extend([9, 11, 13])

print(odd)
```

**Output**

```
[1, 3, 5, 7]
[1, 3, 5, 7, 9, 11, 13]
```

We can also use `+` operator to combine two lists. This is also called concatenation.

The `*` operator repeats a list for the given number of times.

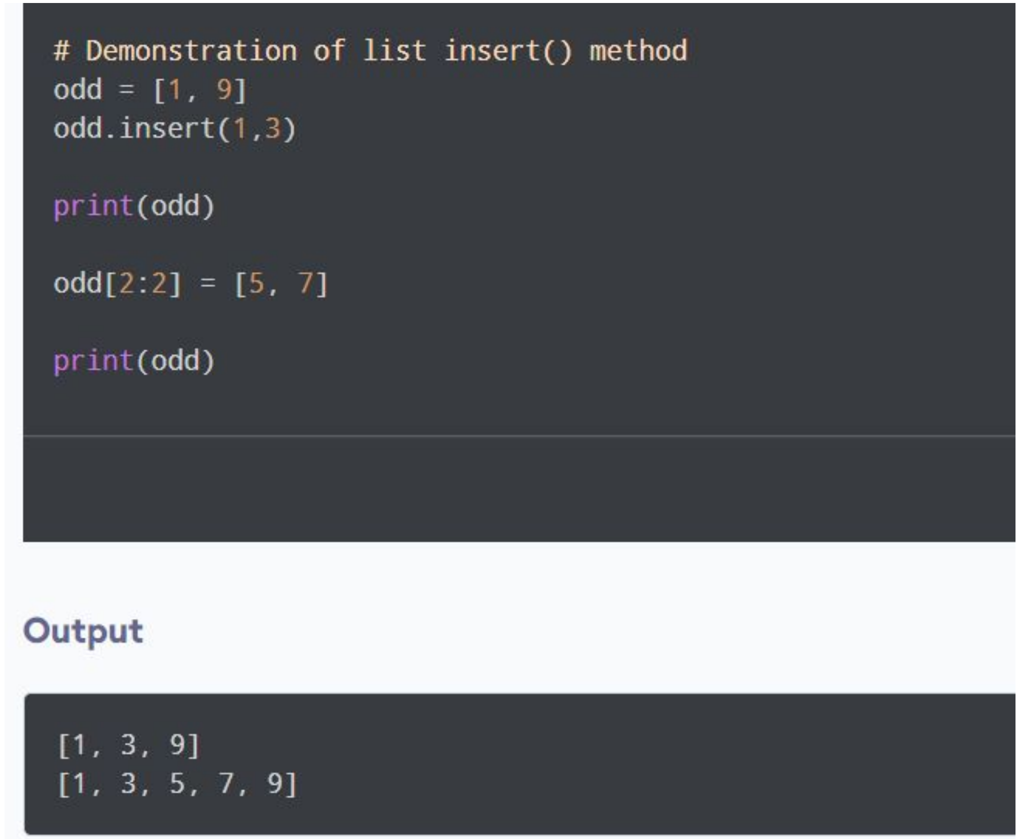


```python
# Concatenating and repeating lists
odd = [1, 3, 5]

print(odd + [9, 7, 5])

print(["re"] * 3)
```

Output

```
[1, 3, 5, 9, 7, 5]
['re', 're', 're']
```

we can insert one item at a desired location by using the method `insert()` or insert multiple items by squeezing it into an empty slice of a list.



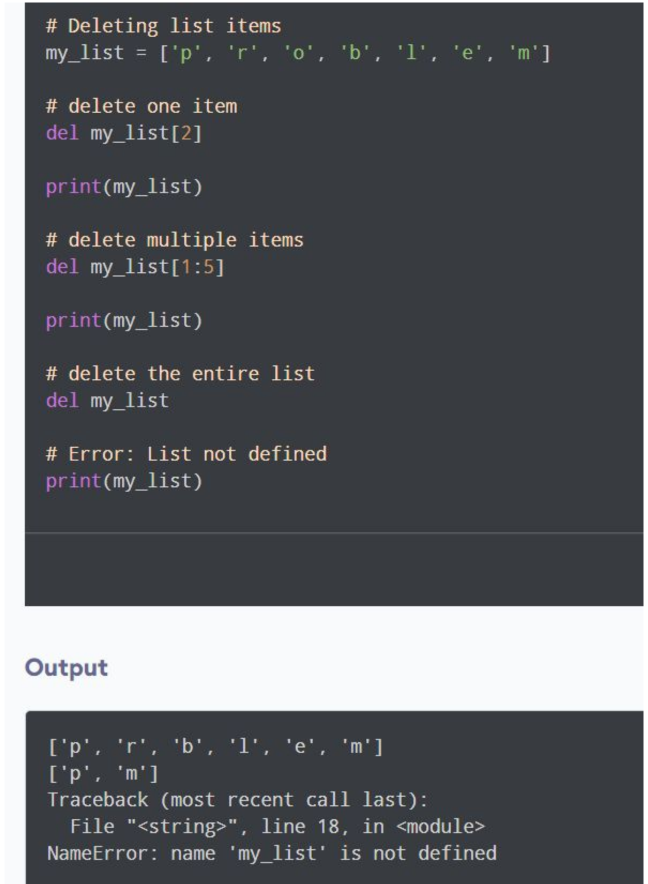
```
# Demonstration of list insert() method
odd = [1, 9]
odd.insert(1,3)

print(odd)

odd[2:2] = [5, 7]

print(odd)
```

Output

```
[1, 3, 9]
[1, 3, 5, 7, 9]
```

## Delete List Elements

We can delete one or more items from a list using the Python del statement. It can even delete the list entirely.



```python
# Deleting list items
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# delete one item
del my_list[2]

print(my_list)

# delete multiple items
del my_list[1:5]

print(my_list)

# delete the entire list
del my_list

# Error: List not defined
print(my_list)
```

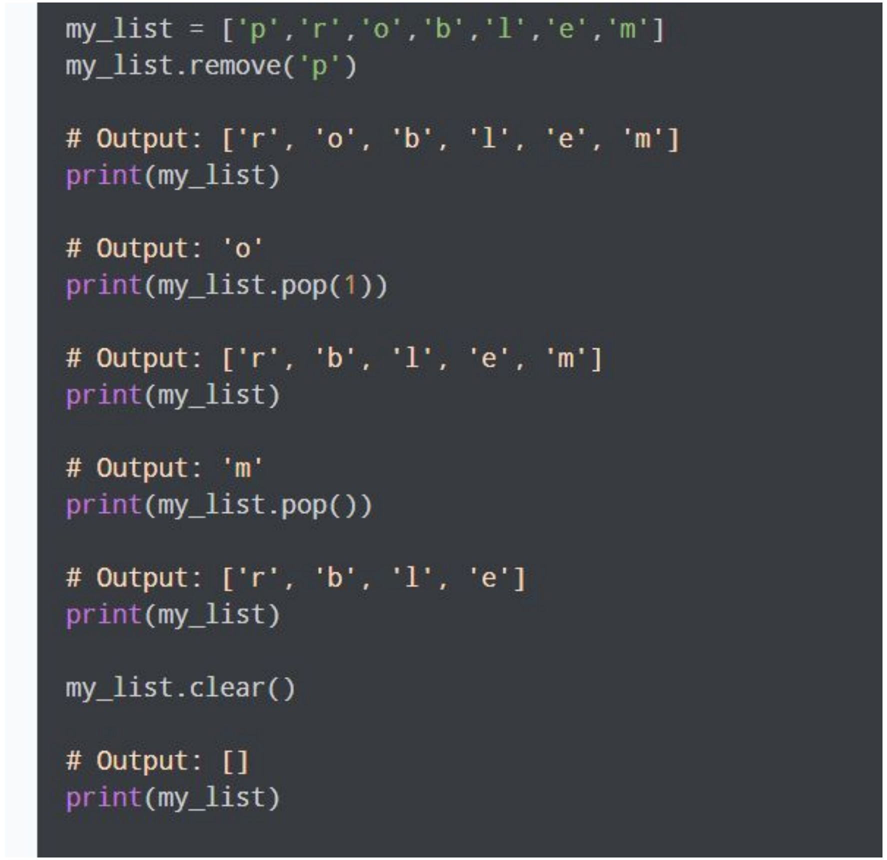
Output

```
['p', 'r', 'b', 'l', 'e', 'm']
['p', 'm']
Traceback (most recent call last):
  File "<string>", line 18, in <module>
NameError: name 'my_list' is not defined
```

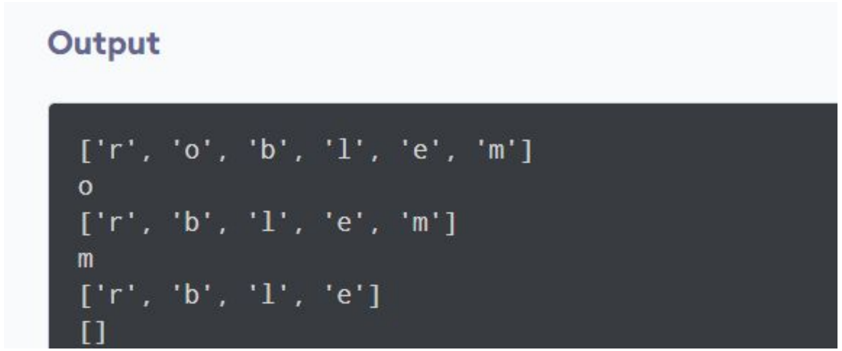
We can use `remove()` to remove the given item or `pop()` to remove an item at the given index.

The `pop()` method removes and returns the last item if the index is not provided. This helps us implement lists as stacks (first in, last out data structure).

And, if we have to empty the whole list, we can use the `clear()` method.

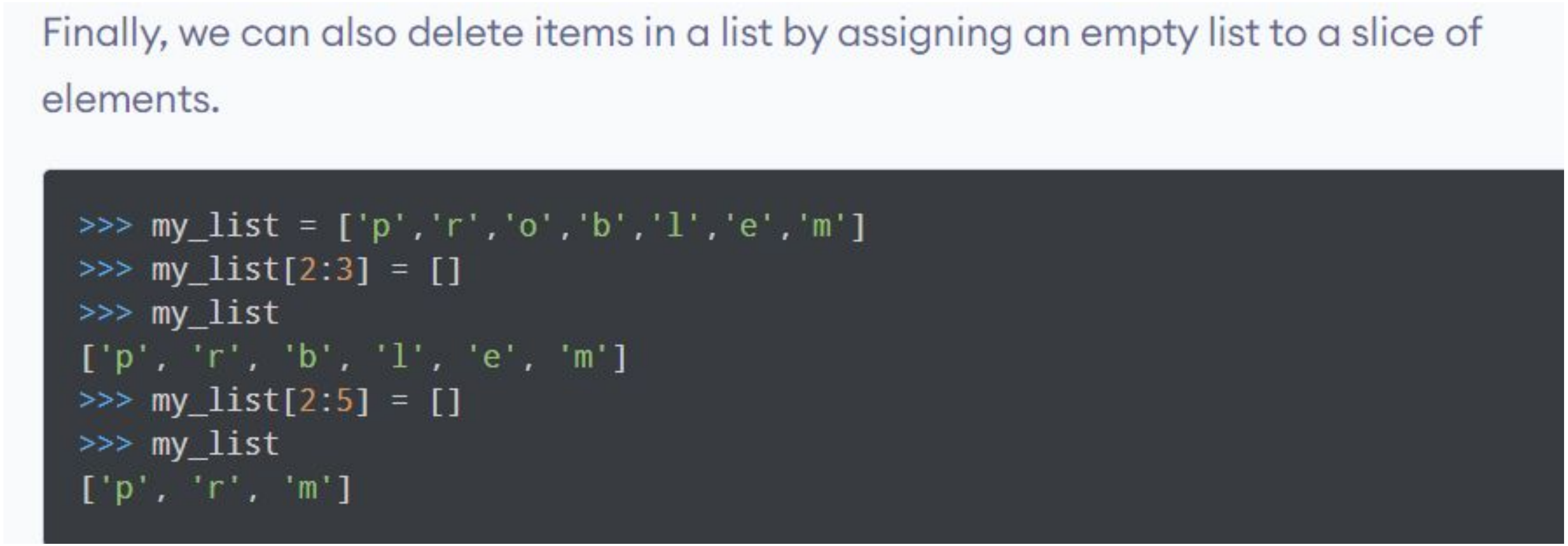


```
my_list = ['p','r','o','b','l','e','m']
my_list.remove('p')

# Output: ['r', 'o', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'o'
print(my_list.pop(1))

# Output: ['r', 'b', 'l', 'e', 'm']
print(my_list)

# Output: 'm'
print(my_list.pop())

# Output: ['r', 'b', 'l', 'e']
print(my_list)

my_list.clear()

# Output: []
print(my_list)
```



**Output**

```
['r', 'o', 'b', 'l', 'e', 'm']
o
['r', 'b', 'l', 'e', 'm']
m
['r', 'b', 'l', 'e']
[]
```

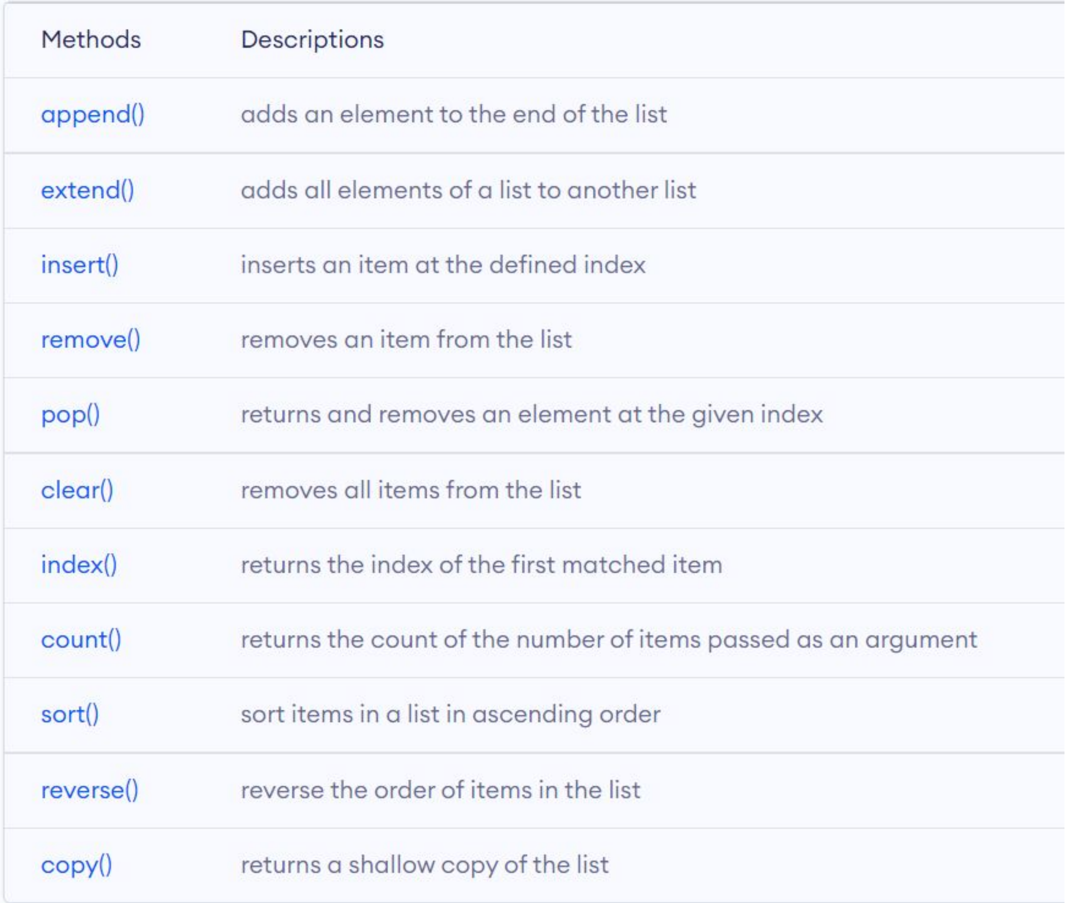


Finally, we can also delete items in a list by assigning an empty list to a slice of elements.

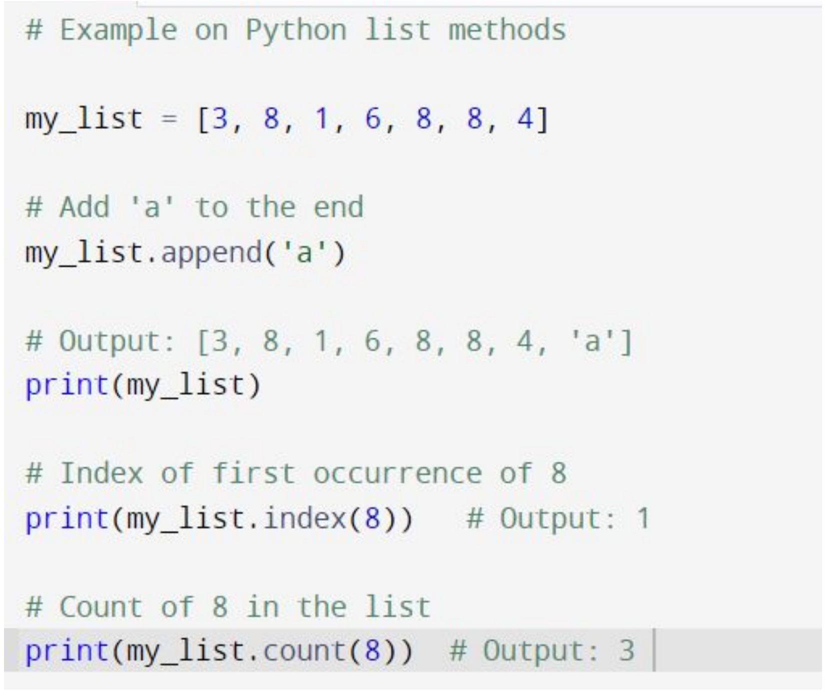
```
>>> my_list = ['p','r','o','b','l','e','m']
>>> my_list[2:3] = []
>>> my_list
['p', 'r', 'b', 'l', 'e', 'm']
>>> my_list[2:5] = []
>>> my_list
['p', 'r', 'm']
```

## Python List Methods

Python has many useful list methods that makes it really easy to work with lists. Here are some of the commonly used list methods.

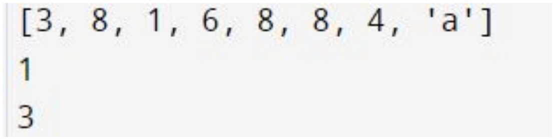


| Methods | Descriptions |
|---|---|
| append() | adds an element to the end of the list |
| extend() | adds all elements of a list to another list |
| insert() | inserts an item at the defined index |
| remove() | removes an item from the list |
| pop() | returns and removes an element at the given index |
| clear() | removes all items from the list |
| index() | returns the index of the first matched item |
| count() | returns the count of the number of items passed as an argument |
| sort() | sort items in a list in ascending order |
| reverse() | reverse the order of items in the list |
| copy() | returns a shallow copy of the list |



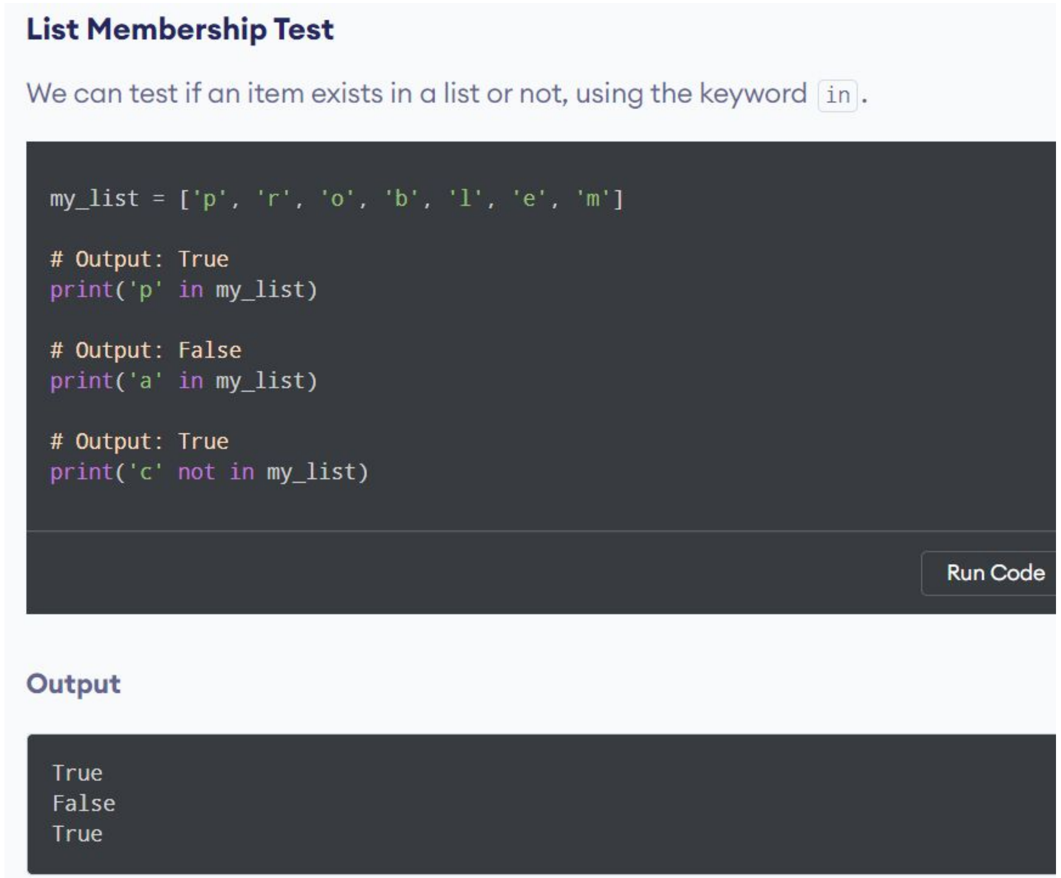
```python
# Example on Python list methods

my_list = [3, 8, 1, 6, 8, 8, 4]

# Add 'a' to the end
my_list.append('a')

# Output: [3, 8, 1, 6, 8, 8, 4, 'a']
print(my_list)

# Index of first occurrence of 8
print(my_list.index(8))   # Output: 1

# Count of 8 in the list
print(my_list.count(8))  # Output: 3
```

output:



```
[3, 8, 1, 6, 8, 8, 4, 'a']
1
3
```

## Other List Operations in Python



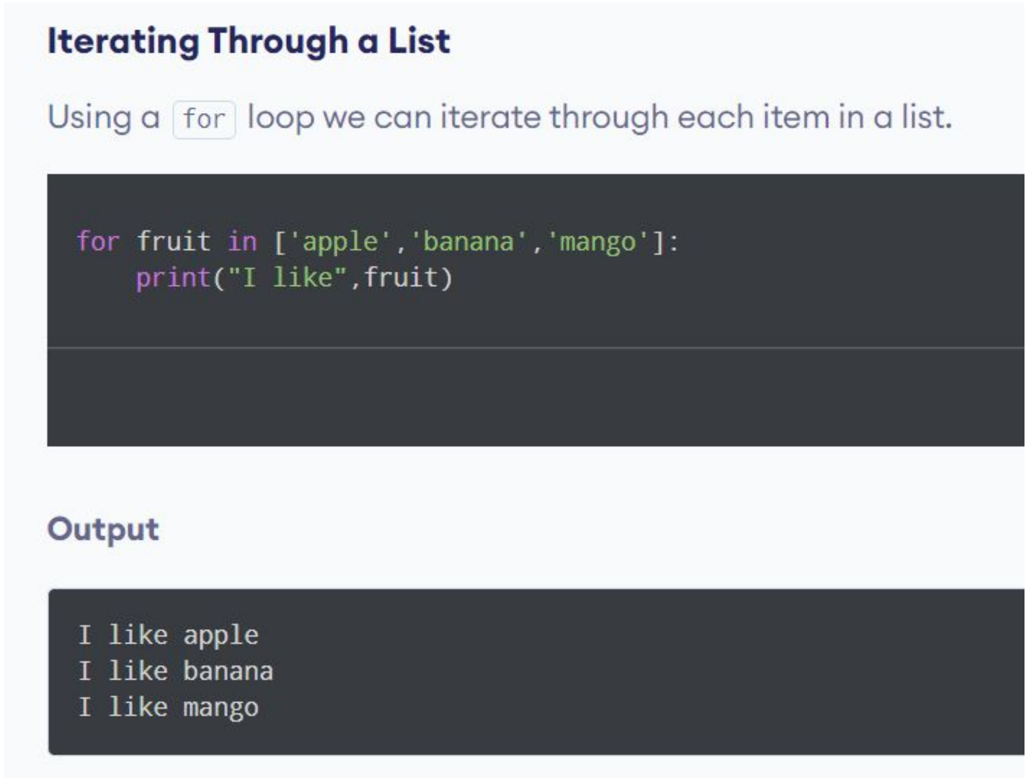
### List Membership Test

We can test if an item exists in a list or not, using the keyword `in`.

```
my_list = ['p', 'r', 'o', 'b', 'l', 'e', 'm']

# Output: True
print('p' in my_list)

# Output: False
print('a' in my_list)

# Output: True
print('c' not in my_list)
```

**Output**

```
True
False
True
```



## Iterating Through a List

Using a `for` loop we can iterate through each item in a list.

```
for fruit in ['apple','banana','mango']:
    print("I like",fruit)
```

**Output**

```
I like apple
I like banana
I like mango
```

# Python List pop()

The `pop()` method removes the item at the given index from the list and returns the removed item.

The syntax of the `pop()` method is:

```
list.pop(index)
```

- The `pop()` method takes a single argument (index).
- The argument passed to the method is optional. If not passed, the default index -1 is passed as an argument (index of the last item).
- If the index passed to the method is not in range, it throws IndexError: pop index out of range exception.

The `pop()` method returns the item present at the given index. This item is also removed from the list.

## Example 1: Pop item at the given index from the list

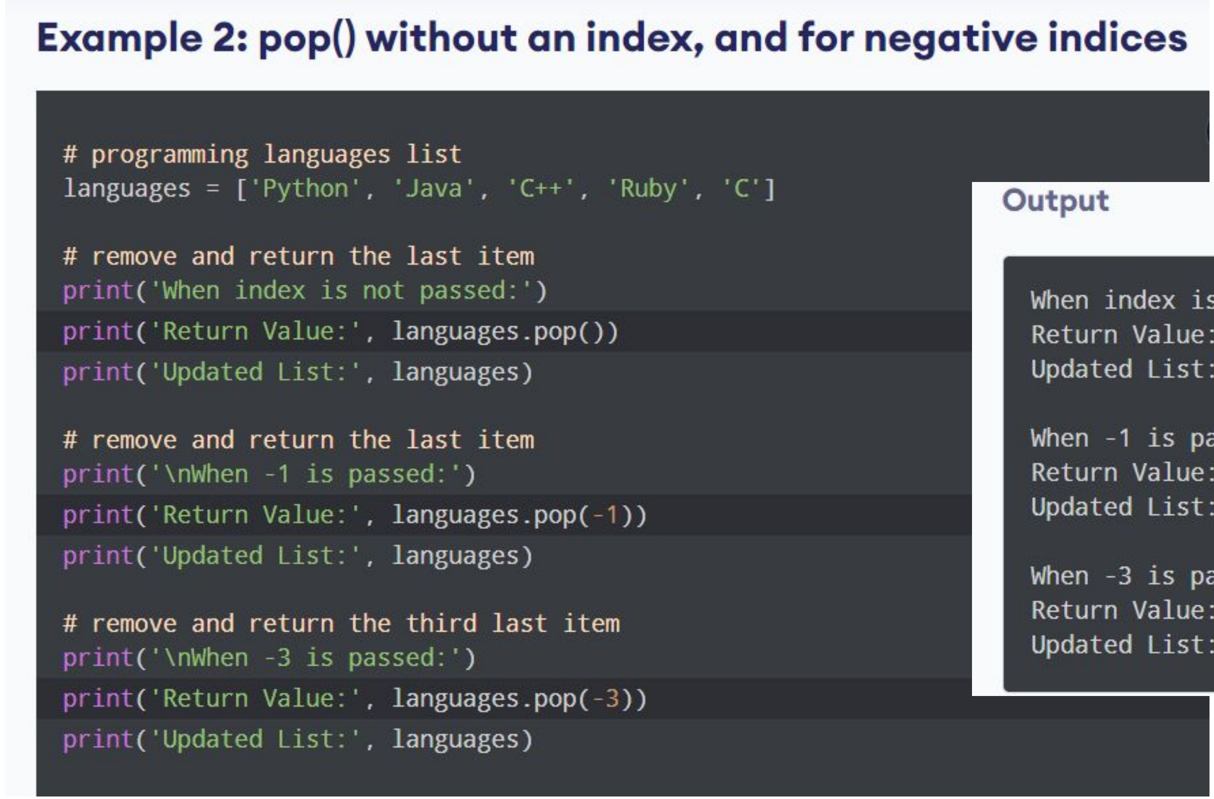
```python
# programming languages list
languages = ['Python', 'Java', 'C++', 'French', 'C']

# remove and return the 4th item
return_value = languages.pop(3)
print('Return Value:', return_value)

# Updated List
print('Updated List:', languages)
```

Run C

### Output

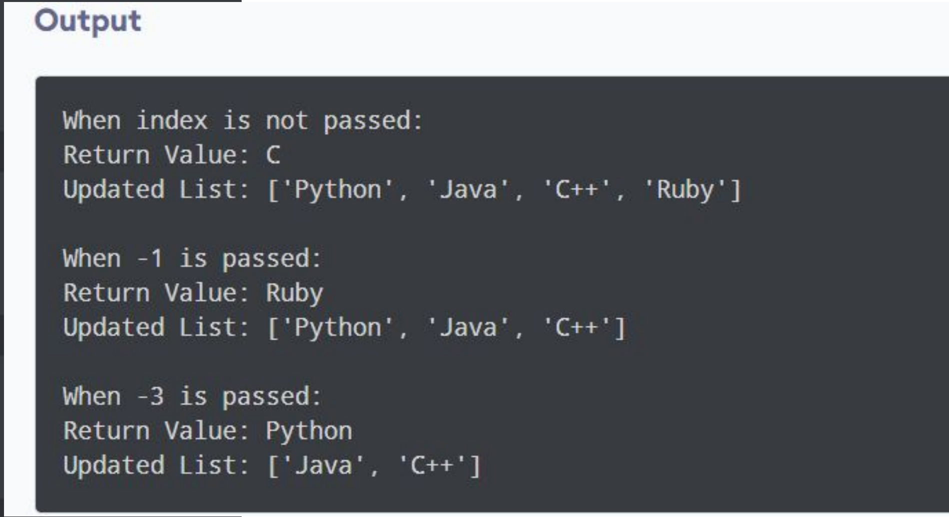
```
Return Value: French
Updated List: ['Python', 'Java', 'C++', 'C']
```

**Note:** Index in Python starts from 0, not 1.



## Example 2: pop() without an index, and for negative indices

```python
# programming languages list
languages = ['Python', 'Java', 'C++', 'Ruby', 'C']

# remove and return the last item
print('When index is not passed:')
print('Return Value:', languages.pop())
print('Updated List:', languages)

# remove and return the last item
print('\nWhen -1 is passed:')
print('Return Value:', languages.pop(-1))
print('Updated List:', languages)

# remove and return the third last item
print('\nWhen -3 is passed:')
print('Return Value:', languages.pop(-3))
print('Updated List:', languages)
```



### Output

```
When index is not passed:
Return Value: C
Updated List: ['Python', 'Java', 'C++', 'Ruby']

When -1 is passed:
Return Value: Ruby
Updated List: ['Python', 'Java', 'C++']

When -3 is passed:
Return Value: Python
Updated List: ['Java', 'C++']
```

If you need to remove the given item from the list, you can use the remove() method.

And, you can use the `del` statement to remove an item or slices from the list.

## Python List sort()

The `sort()` method sorts the elements of a given list in a specific ascending or descending order.

The syntax of the `sort()` method is:

```
list.sort(key=..., reverse=...)
```

Alternatively, you can also use Python's built-in sorted() function for the same purpose.

```
sorted(list, key=..., reverse=...)
```

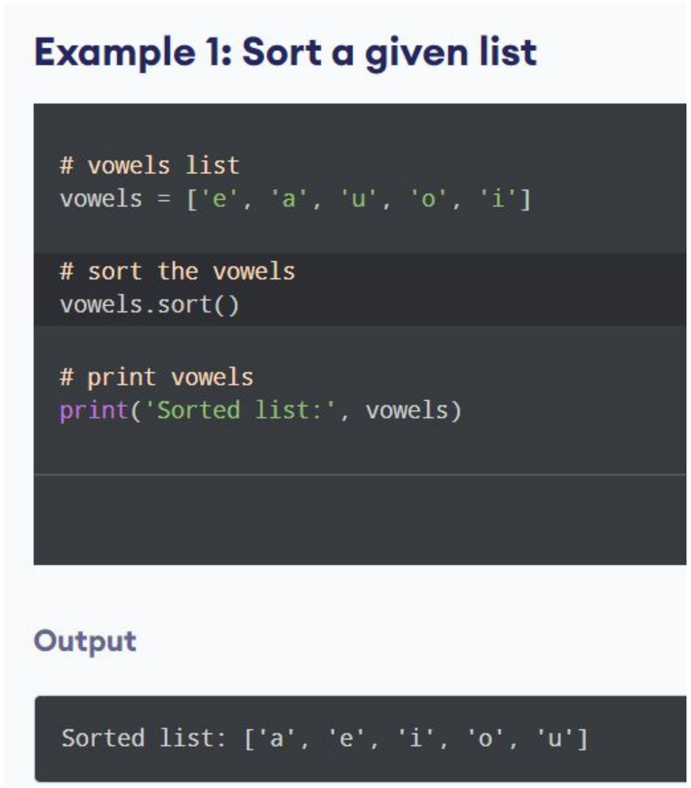
Note: The simplest difference between `sort()` and `sorted()` is: `sort()` changes the list directly and doesn't return any value, while `sorted()` doesn't change the list and returns the sorted list.

By default, `sort()` doesn't require any extra parameters. However, it has two optional parameters:

- reverse - If `True`, the sorted list is reversed (or sorted in Descending order)
- key - function that serves as a key for the sort comparison

The `sort()` method doesn't return any value. Rather, it changes the original list.

If you want a function to return the sorted list rather than change the original list, use `sorted()`.

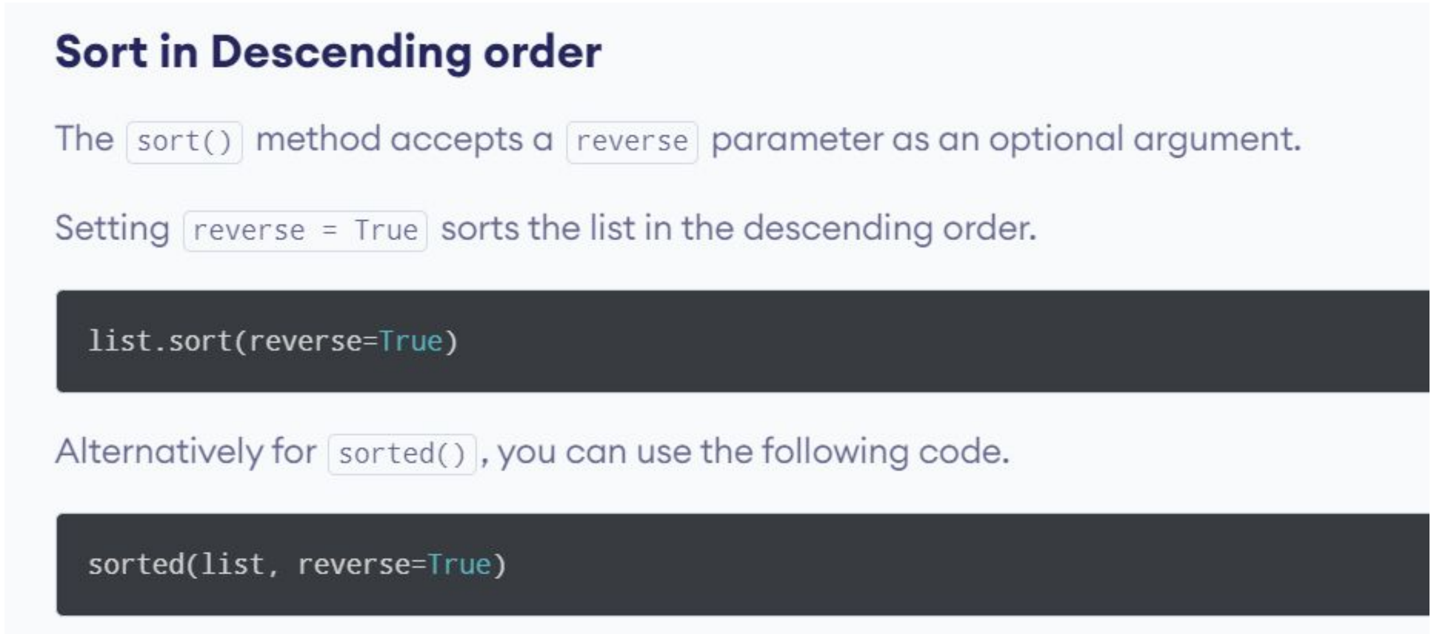


## Example 1: Sort a given list

```python
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']

# sort the vowels
vowels.sort()

# print vowels
print('Sorted list:', vowels)
```

**Output**

```
Sorted list: ['a', 'e', 'i', 'o', 'u']
```



## Sort in Descending order

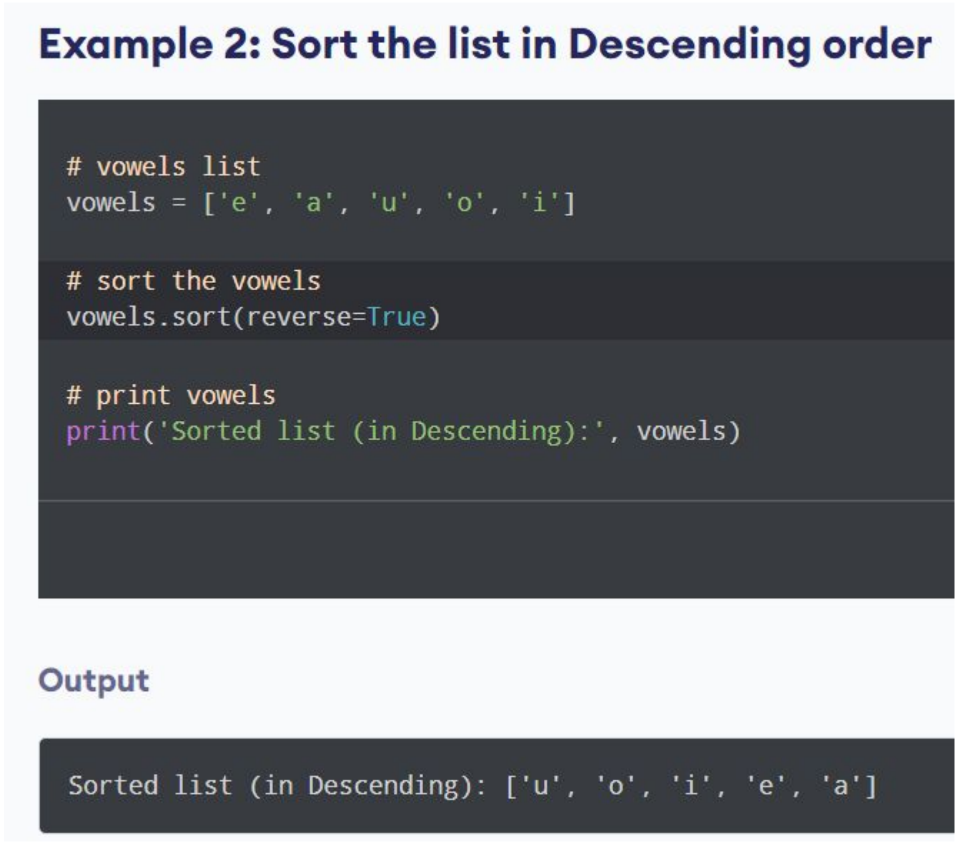
The `sort()` method accepts a `reverse` parameter as an optional argument.

Setting `reverse = True` sorts the list in the descending order.

```
list.sort(reverse=True)
```

Alternatively for `sorted()`, you can use the following code.

```
sorted(list, reverse=True)
```



## Example 2: Sort the list in Descending order

```python
# vowels list
vowels = ['e', 'a', 'u', 'o', 'i']

# sort the vowels
vowels.sort(reverse=True)

# print vowels
print('Sorted list (in Descending):', vowels)
```

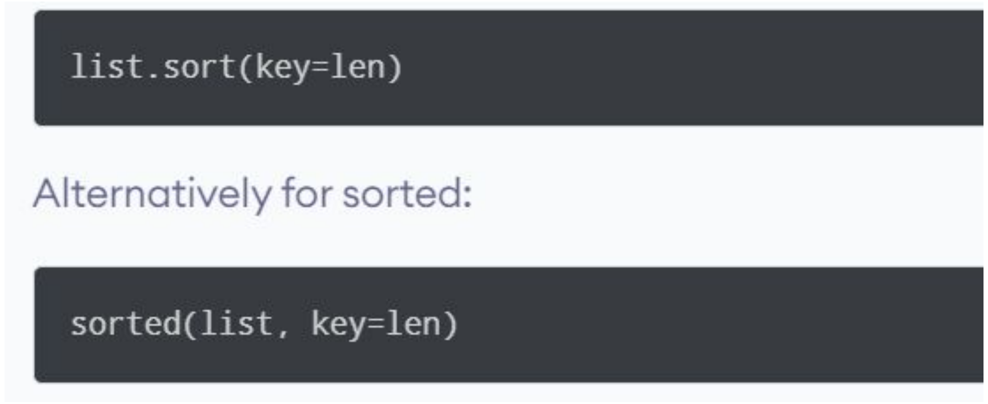
**Output**

```
Sorted list (in Descending): ['u', 'o', 'i', 'e', 'a']
```

## Sort with custom function using key

If you want your own implementation for sorting, the `sort()` method also accepts a `key` function as an optional parameter.

Based on the results of the key function, you can sort the given list.



```
list.sort(key=len)
```

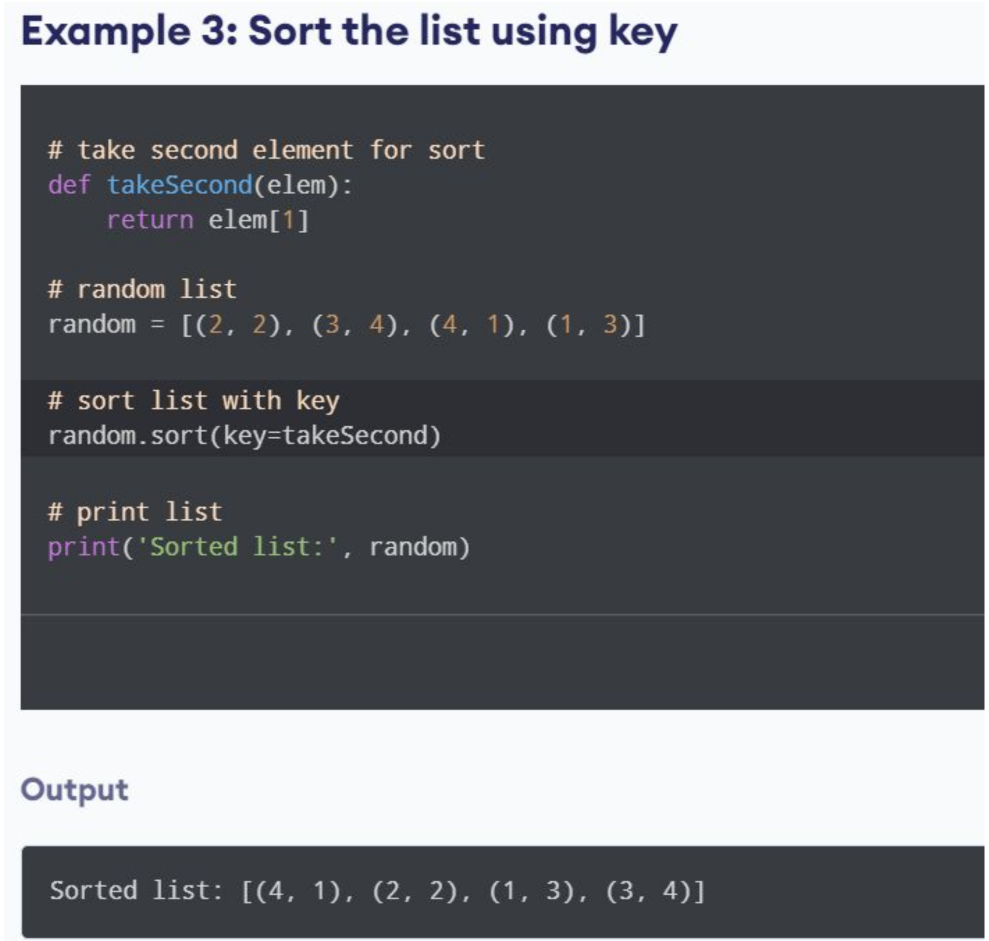
Alternatively for sorted:

```
sorted(list, key=len)
```

Here, `len` is Python's in-built function to count the length of an element.

The list is sorted based on the length of each element, from lowest count to highest.

We know that a tuple is sorted using its first parameter by default. Let's look at how to customize the `sort()` method to sort using the second element.



## Example 3: Sort the list using key

```python
# take second element for sort
def takeSecond(elem):
    return elem[1]

# random list
random = [(2, 2), (3, 4), (4, 1), (1, 3)]

# sort list with key
random.sort(key=takeSecond)

# print list
print('Sorted list:', random)
```

**Output**

```
Sorted list: [(4, 1), (2, 2), (1, 3), (3, 4)]
```

## Python List reverse()

The `reverse()` method reverses the elements of the list.

The syntax of the `reverse()` method is:

```
list.reverse()
```

The `reverse()` method doesn't take any arguments.

The `reverse()` method doesn't return any value. It updates the existing list.

## Example 1: Reverse a List

```python
# Operating System List
systems = ['Windows', 'macOS', 'Linux']
print('Original List:', systems)

# List Reverse
systems.reverse()

# updated list
print('Updated List:', systems)
```

### Output

```
Original List: ['Windows', 'macOS', 'Linux']
Updated List: ['Linux', 'macOS', 'Windows']
```

There are other several ways to reverse a list.

## Example 2: Reverse a List Using Slicing Operator

```python
# Operating System List
systems = ['Windows', 'macOS', 'Linux']
print('Original List:', systems)

# Reversing a list
# Syntax: reversed_list = systems[start:stop:step]
reversed_list = systems[::-1]

# updated list
print('Updated List:', reversed_list)
```

### Output

```
Original List: ['Windows', 'macOS', 'Linux']
Updated List: ['Linux', 'macOS', 'Windows']
```

## Example 3: Accessing Elements in Reversed Order

If you need to access individual elements of a list in the reverse order, it's better to use the `reversed()` function.

```python
# Operating System List
systems = ['Windows', 'macOS', 'Linux']

# Printing Elements in Reversed Order
for o in reversed(systems):
    print(o)
```

Run Code »

### Output

```
Linux
macOS
Windows
```

## Python List extend()

The `extend()` method adds all the elements of an iterable (list, tuple, string etc.) to the end of the list.

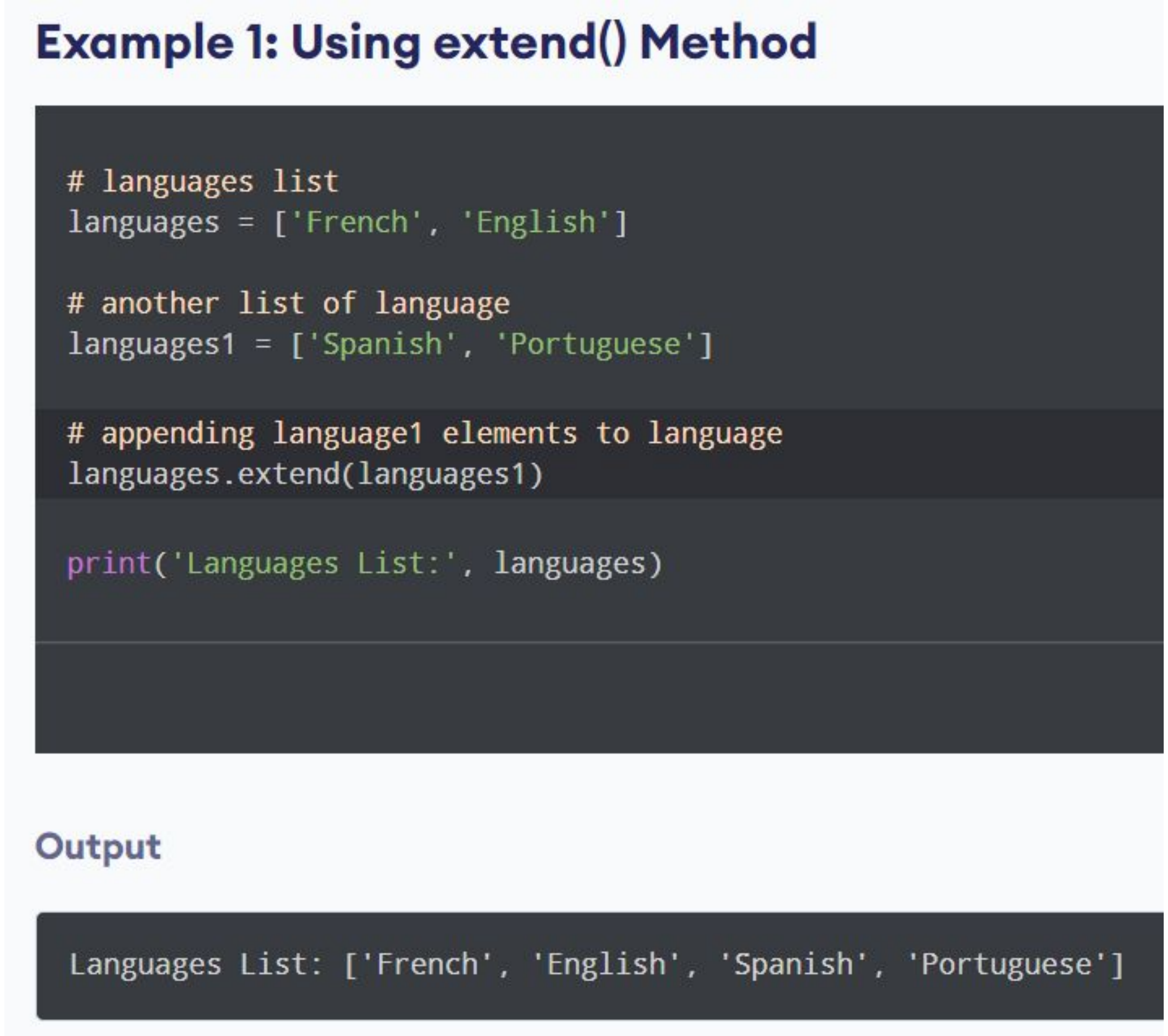
The syntax of the `extend()` method is:

```
list1.extend(iterable)
```

Here, all the elements of `iterable` are added to the end of `list1`.

As mentioned, the `extend()` method takes an iterable such as list, tuple, string etc.

The `extend()` method modifies the original list. It doesn't return any value.



## Example 1: Using extend() Method

```python
# languages list
languages = ['French', 'English']

# another list of language
languages1 = ['Spanish', 'Portuguese']

# appending language1 elements to language
languages.extend(languages1)

print('Languages List:', languages)
```

### Output

```
Languages List: ['French', 'English', 'Spanish', 'Portuguese']
```

## Example 2: Add Elements of Tuple and Set to List

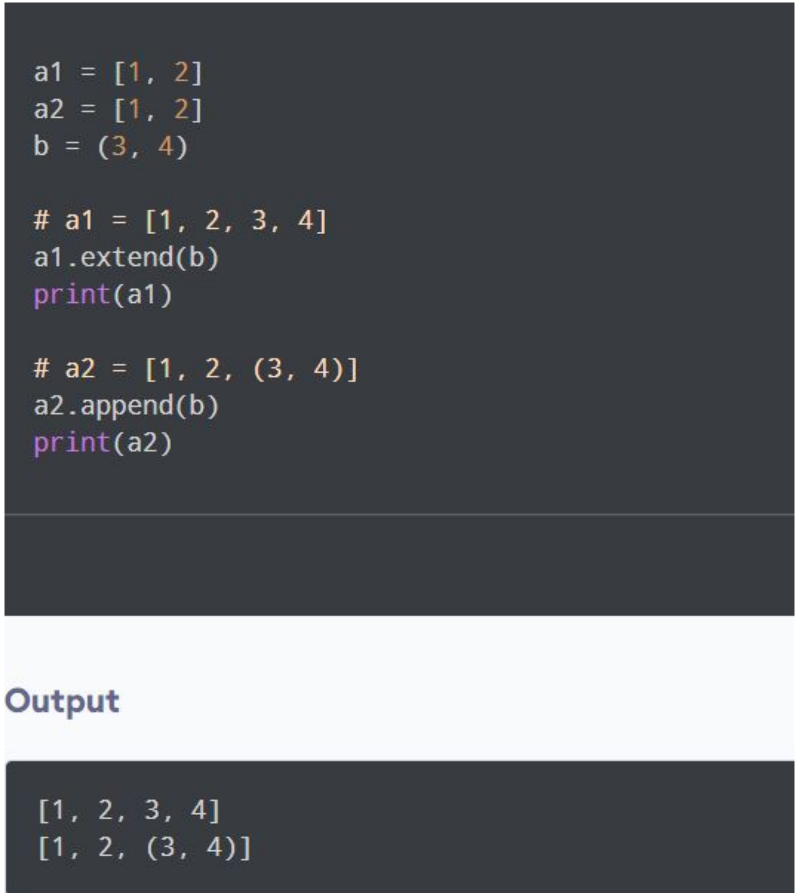
```python
# languages list
languages = ['French']

# languages tuple
languages_tuple = ('Spanish', 'Portuguese')

# languages set
languages_set = {'Chinese', 'Japanese'}

# appending language_tuple elements to language
languages.extend(languages_tuple)

print('New Language List:', languages)

# appending language_set elements to language
languages.extend(languages_set)

print('Newer Languages List:', languages)
```

### Output

```
New Languages List: ['French', 'Spanish', 'Portuguese']
Newer Languages List: ['French', 'Spanish', 'Portuguese', 'Japanese', 'Chinese']
```

## Python extend() Vs append()

If you need to add an element to the end of a list, you can use the `append()` method.



```python
a1 = [1, 2]
a2 = [1, 2]
b = (3, 4)

# a1 = [1, 2, 3, 4]
a1.extend(b)
print(a1)

# a2 = [1, 2, (3, 4)]
a2.append(b)
print(a2)
```

Output

```
[1, 2, 3, 4]
[1, 2, (3, 4)]
```

# Python List insert()

The `insert()` method inserts an element to the list at the specified index.

The syntax of the `insert()` method is

```
list.insert(i, elem)
```

Here, `elem` is inserted to the list at the `ith` index. All the elements after `elem` are shifted to the right.

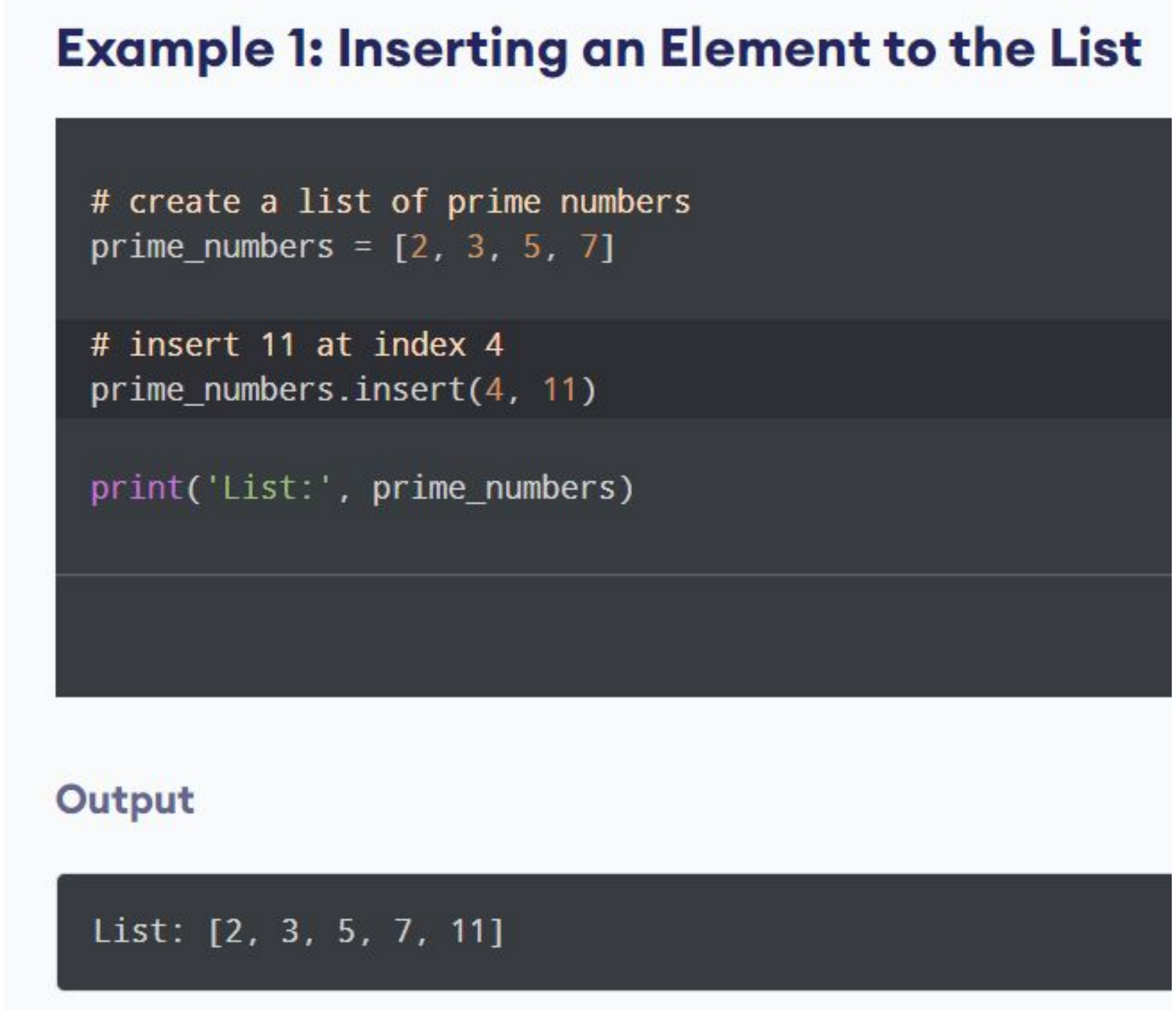
The `insert()` method takes two parameters:

- index - the index where the element needs to be inserted
- element - this is the element to be inserted in the list

Notes:

- If `index` is 0, the element is inserted at the beginning of the list.
- If `index` is 3, the index of the inserted element will be 3 (4th element in the list).

The `insert()` method doesn't return anything; returns `None`. It only updates the current list.



## Example 1: Inserting an Element to the List

```python
# create a list of prime numbers
prime_numbers = [2, 3, 5, 7]

# insert 11 at index 4
prime_numbers.insert(4, 11)

print('List:', prime_numbers)
```

**Output**

```
List: [2, 3, 5, 7, 11]
```

## Example 2: Inserting a Tuple (as an Element) to the List

```python
mixed_list = [{1, 2}, [5, 6, 7]]

# number tuple
number_tuple = (3, 4)

# inserting a tuple to the list
mixed_list.insert(1, number_tuple)

print('Updated List:', mixed_list)
```

Run

### Output

```
Updated List: [{1, 2}, (3, 4), [5, 6, 7]]
```