COURSE UNIT 6: WEEK 8

LISTS AND TUPLES

OBJECTIVES:

1. Classify the python list and tuples.
2. Practice programs using python list and tuples.
3. Construct a python program using list and tuples.

PYTHON COLLECTIONS (ARRAYS)

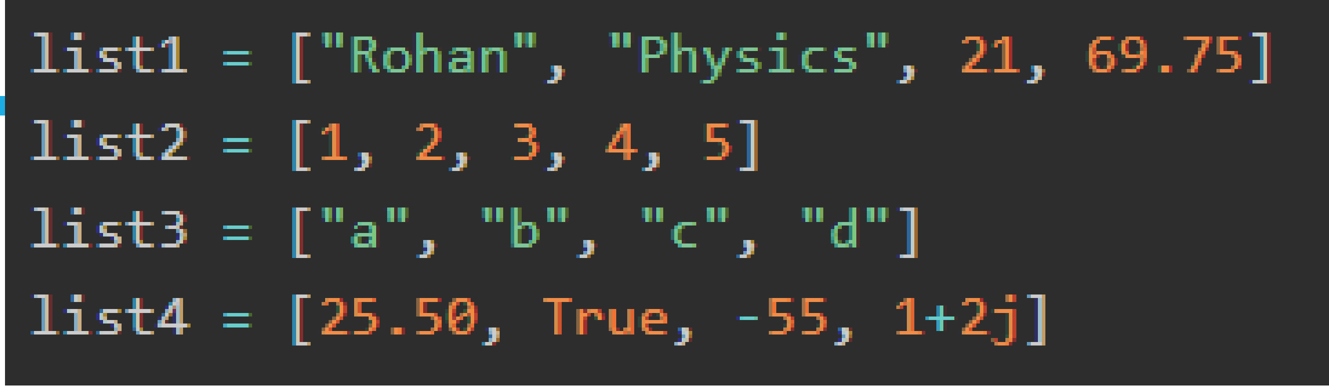
Four Collection Data Types in the Python programming language:

- 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, unchangeable*, and unindexed. No duplicate members.
- Dictionary is a collection which is ordered** and changeable. No duplicate members.

*Set *items* are unchangeable, but you can remove and/or add items whenever you like.
**As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.

PYTHON - LISTS

PYTHON - LISTS

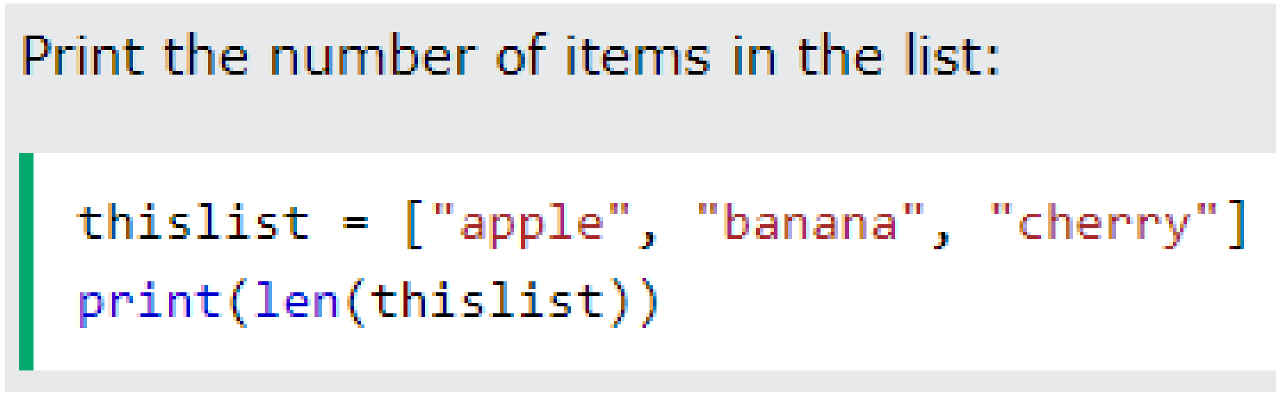


```python
list1 = ["Rohan", "Physics", 21, 69.75]
list2 = [1, 2, 3, 4, 5]
list3 = ["a", "b", "c", "d"]
list4 = [25.50, True, -55, 1+2j]
```

- one of the built-in data types in Python
- used to store multiple items in a single variable
- created using square brackets [ ]
- ordered, changeable, and allow duplicate values
- indexed, the first item has index [0]
- list item can be of any data type

PYTHON - LISTS

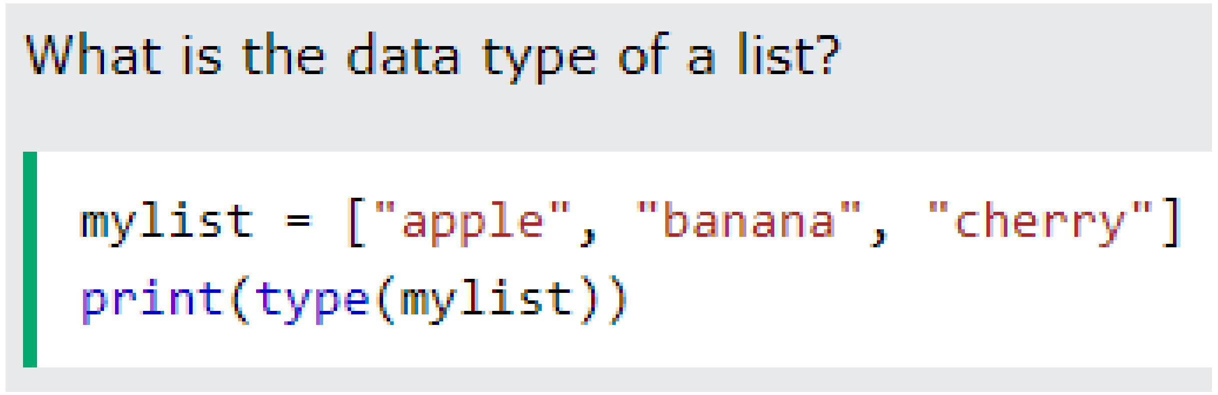
List Length



Print the number of items in the list:

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

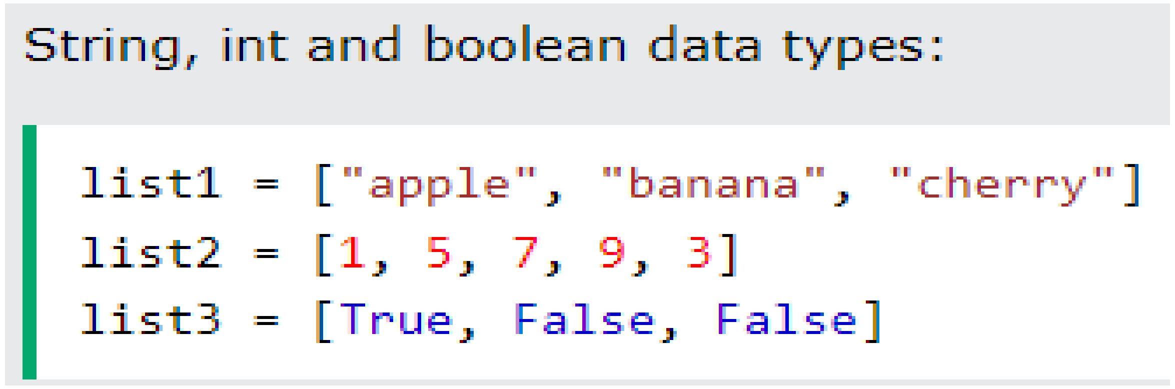
type()



What is the data type of a list?

```
mylist = ["apple", "banana", "cherry"]
print(type(mylist))
```

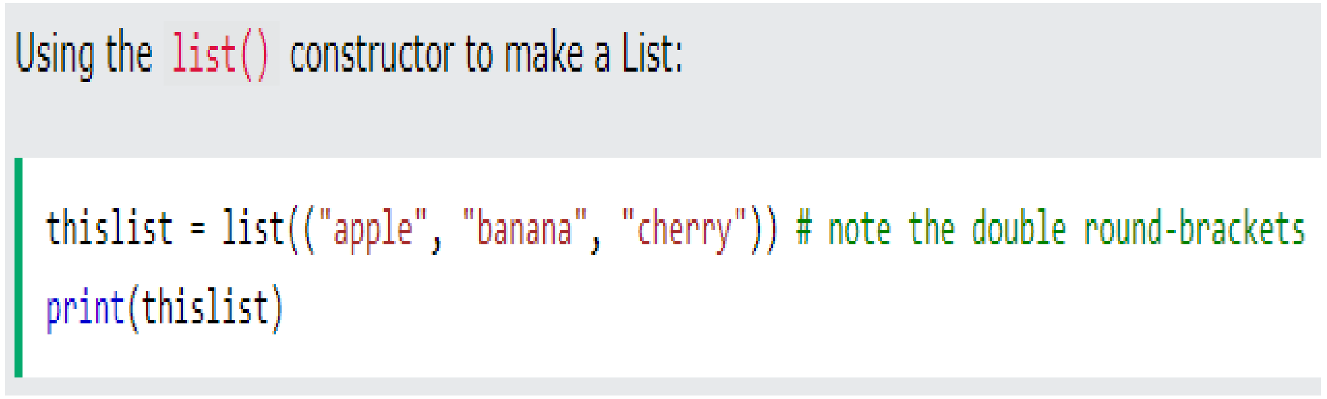
List Items - Data Types



String, int and boolean data types:

```
list1 = ["apple", "banana", "cherry"]
list2 = [1, 5, 7, 9, 3]
list3 = [True, False, False]
```

The list() Constructor

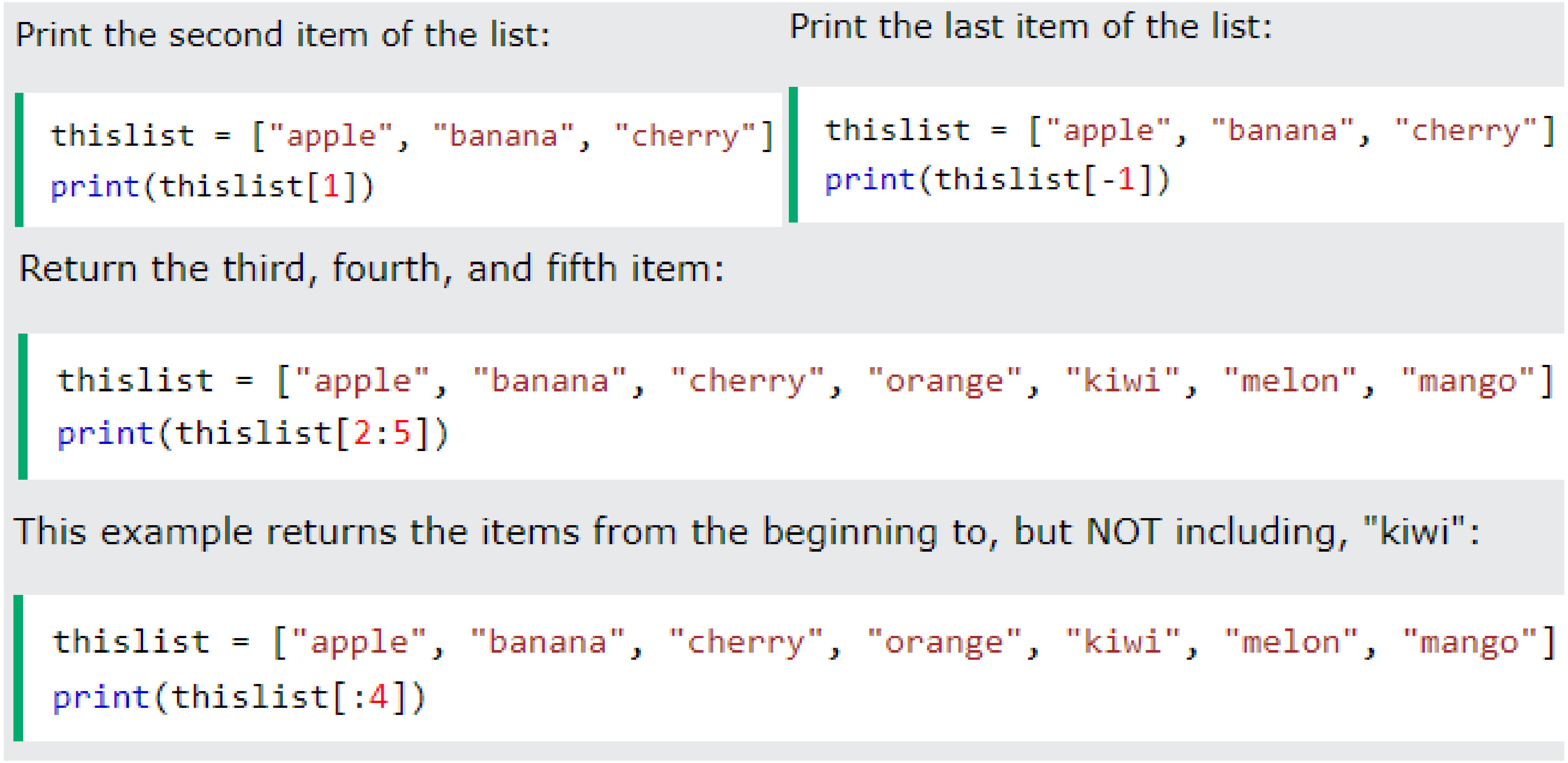


Using the `list()` constructor to make a List:

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

ACCESS LIST ITEMS

Access List Items



Print the second item of the list:

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

Print the last item of the list:

```python
thislist = ["apple", "banana", "cherry"]
print(thislist[-1])
```

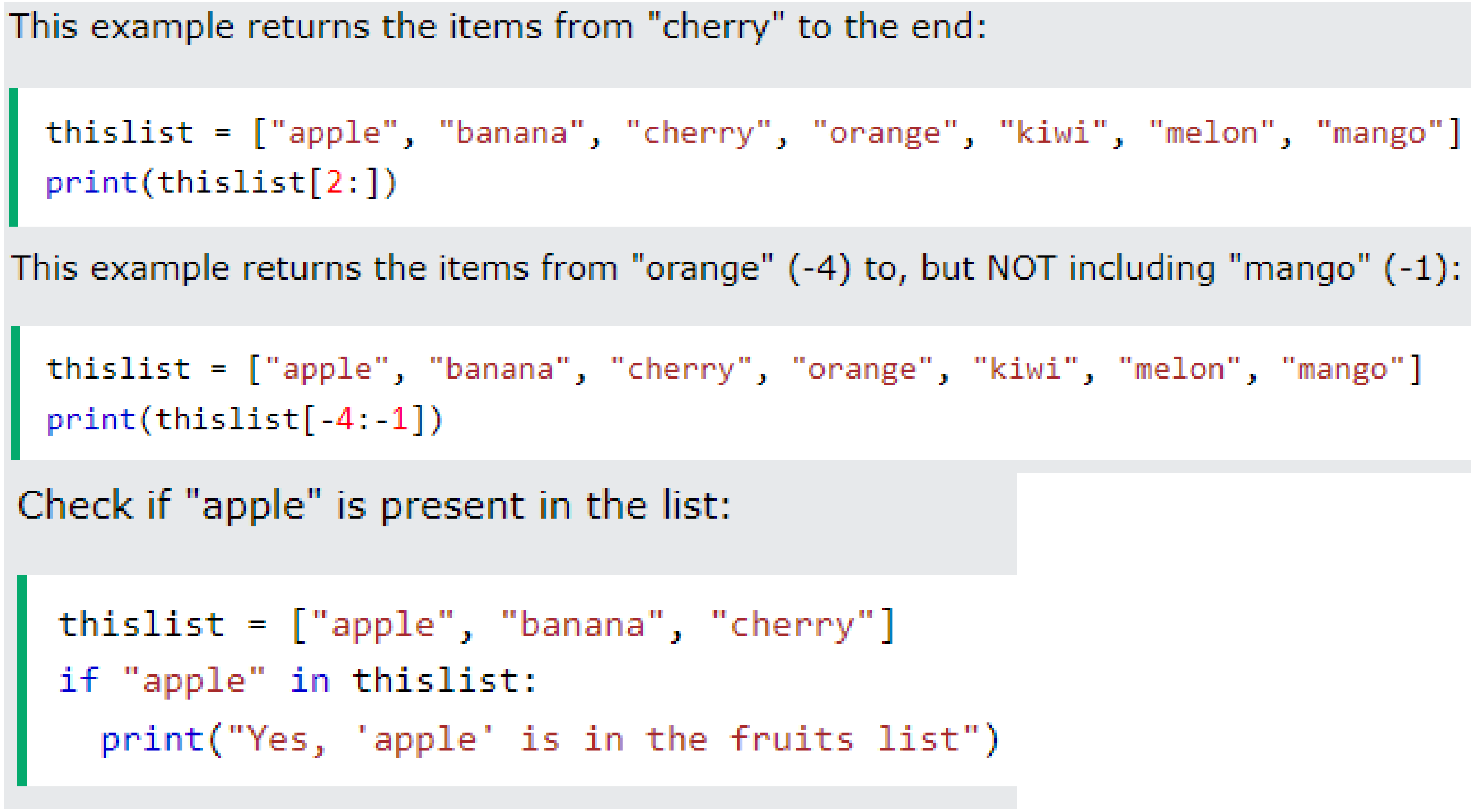
Return the third, fourth, and fifth item:

```python
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:5])
```

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

```python
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[:4])
```

Access List Items



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

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[2:])
```

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

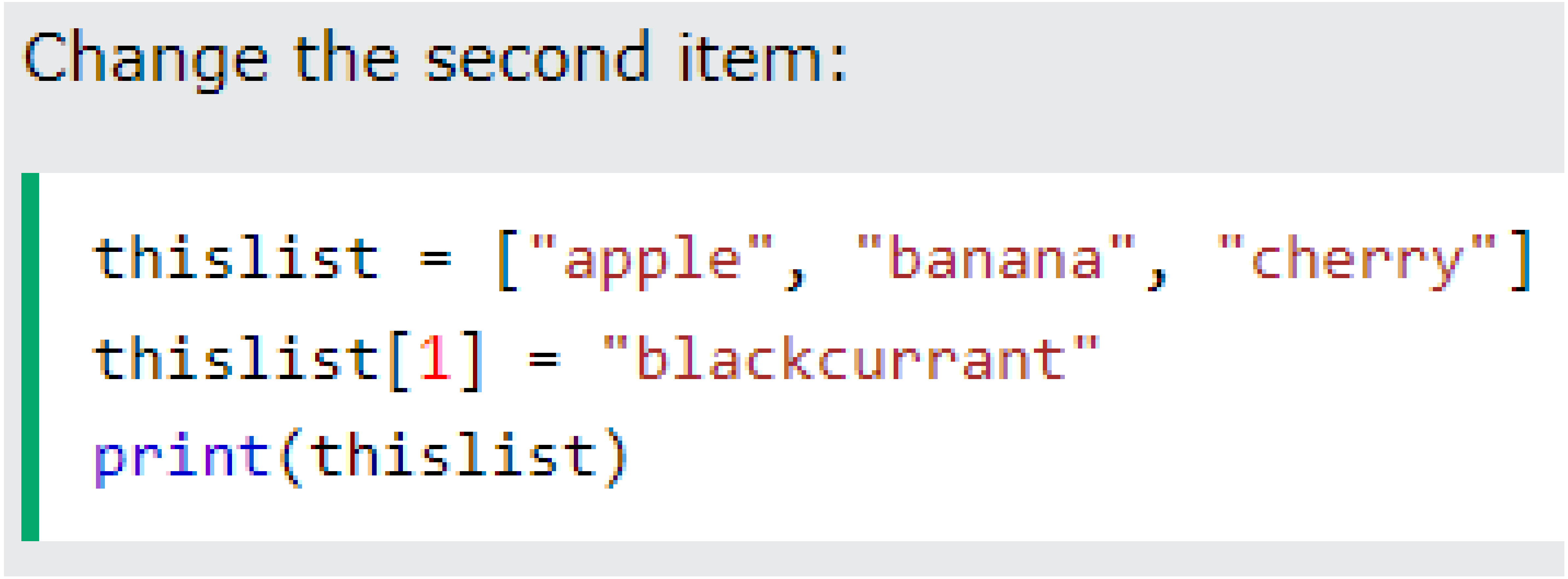
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(thislist[-4:-1])
```

Check if "apple" is present in the list:

```
thislist = ["apple", "banana", "cherry"]
if "apple" in thislist:
  print("Yes, 'apple' is in the fruits list")
```

CHANGE LIST ITEMS

Change Item Value



Change the second item:

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

Change a Range of Item Values

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

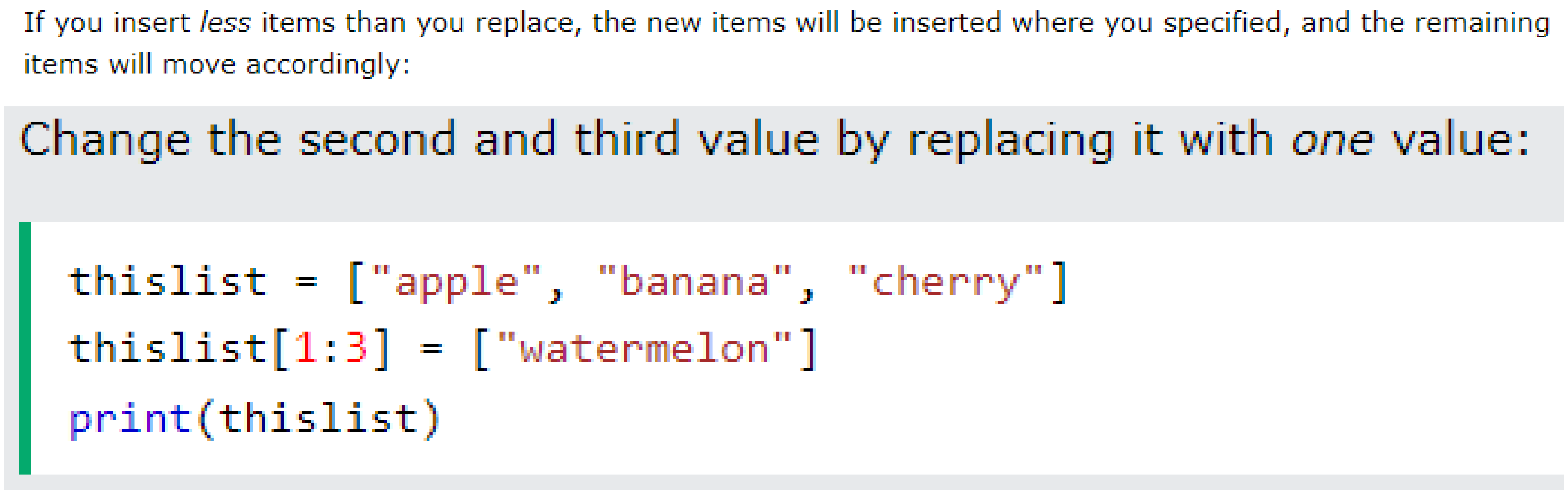
```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

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.

Change a Range of Item Values



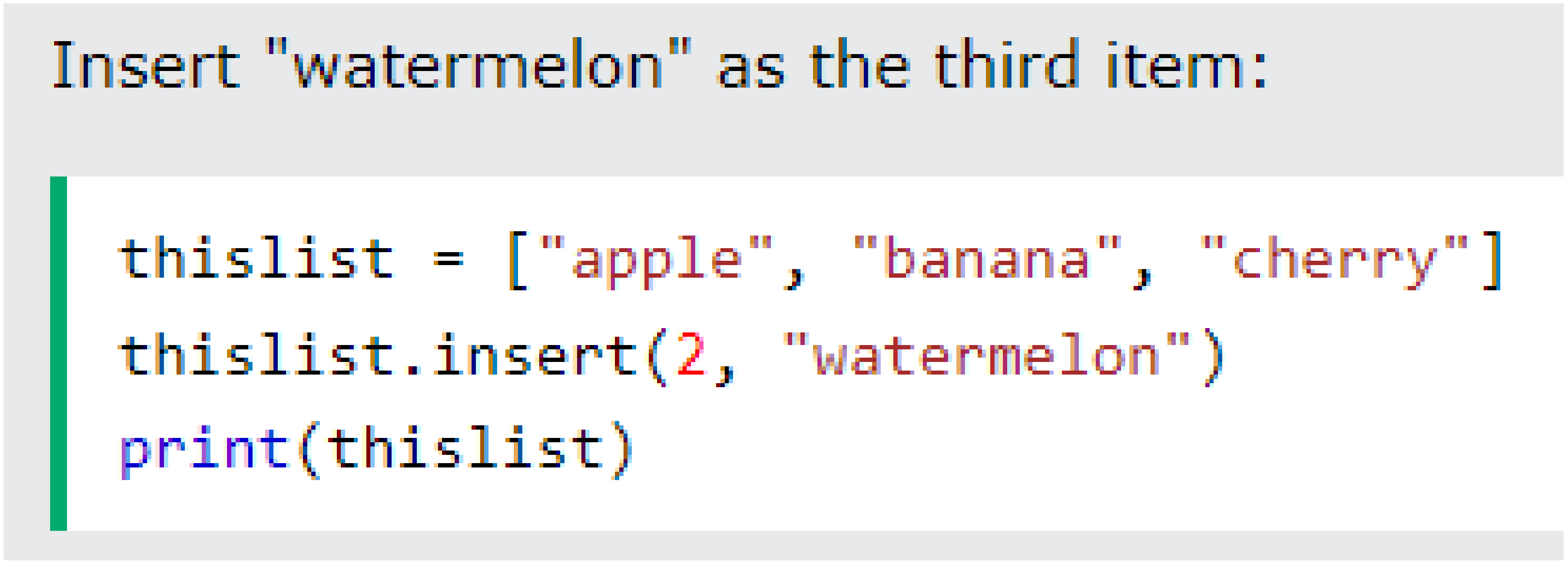
If you insert *less* items than you replace, the new items will be inserted where you specified, and the remaining items will move accordingly:

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.

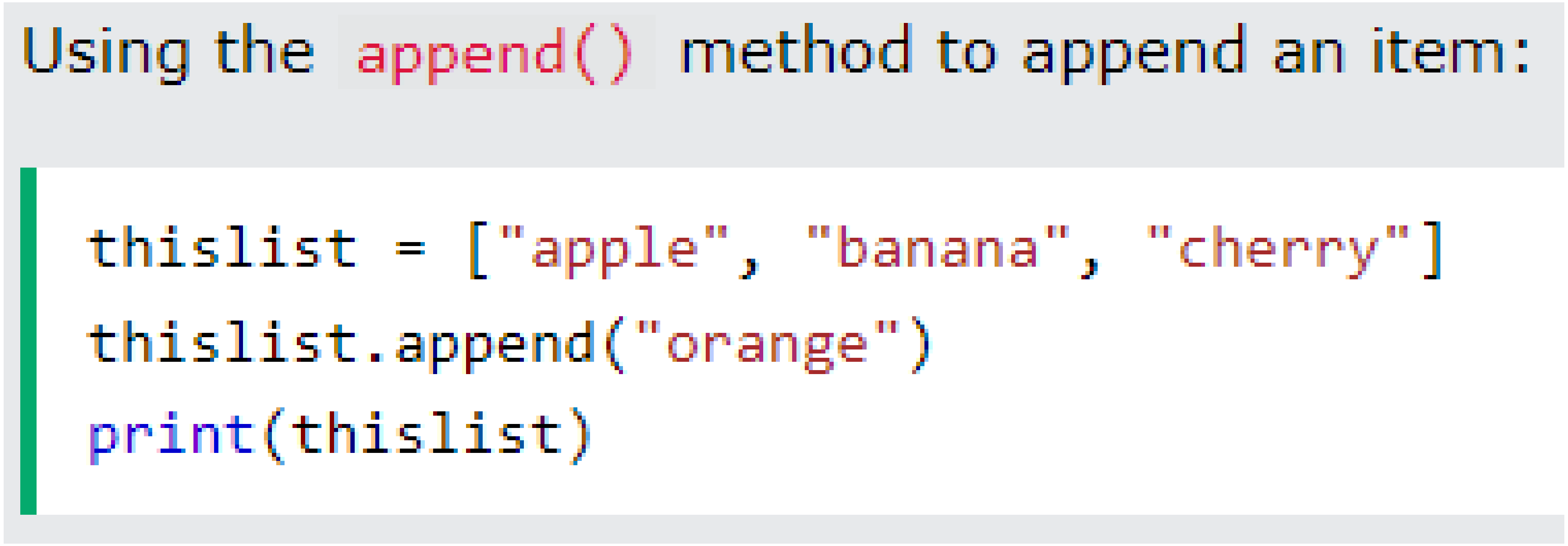


Insert "watermelon" as the third item:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

ADD LIST ITEMS

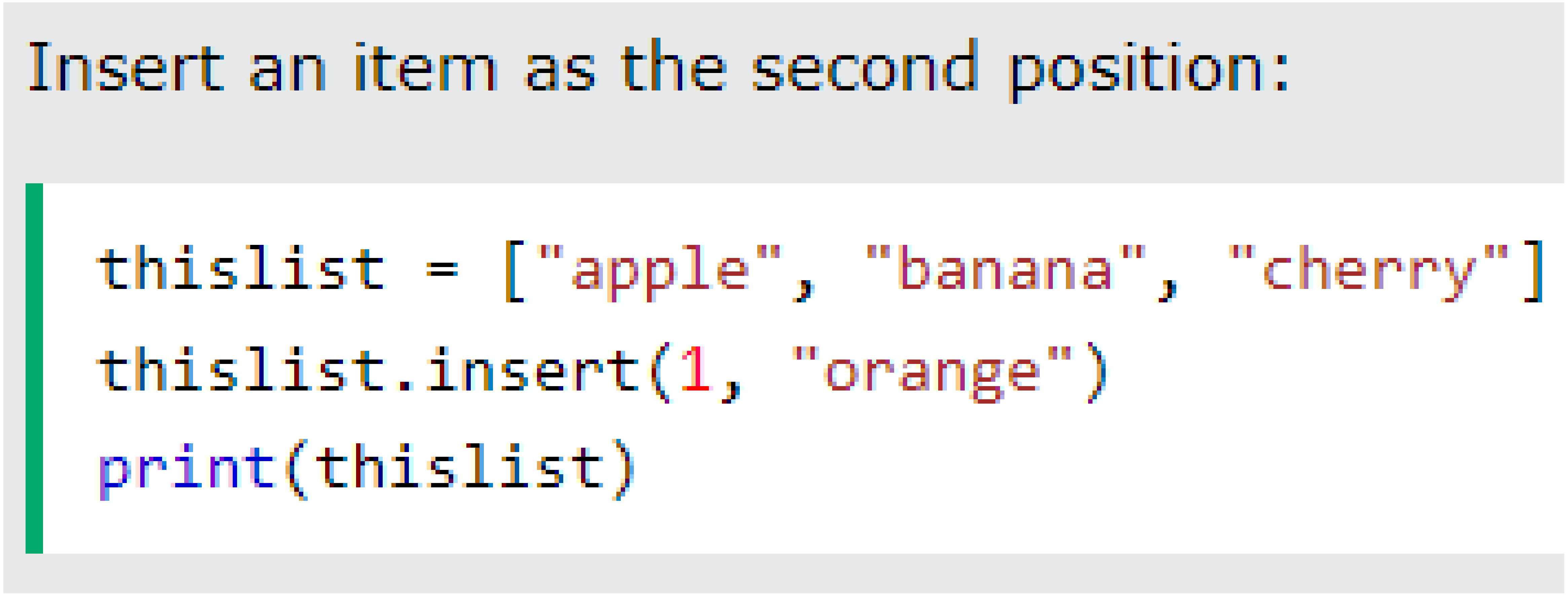
Append Items



Using the `append()` method to append an item:

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

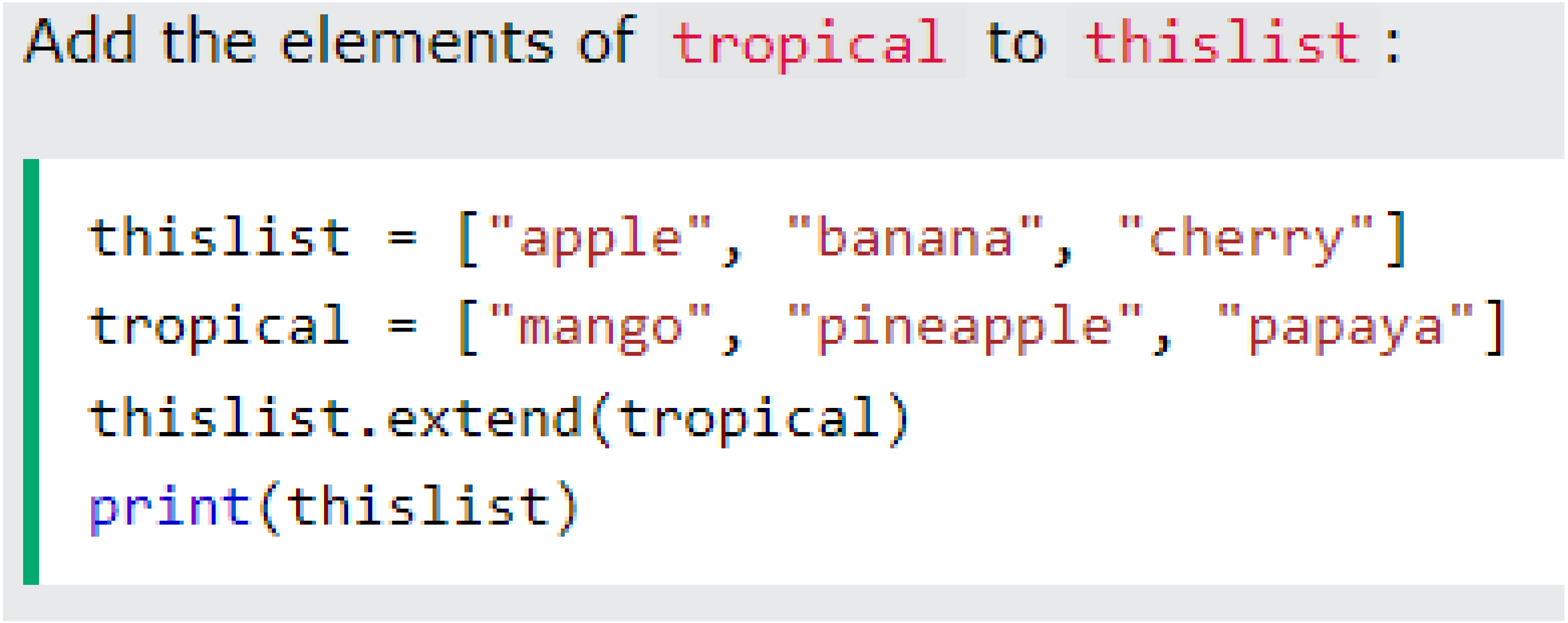
Insert Items



Insert an item as the second position:

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

Extend List

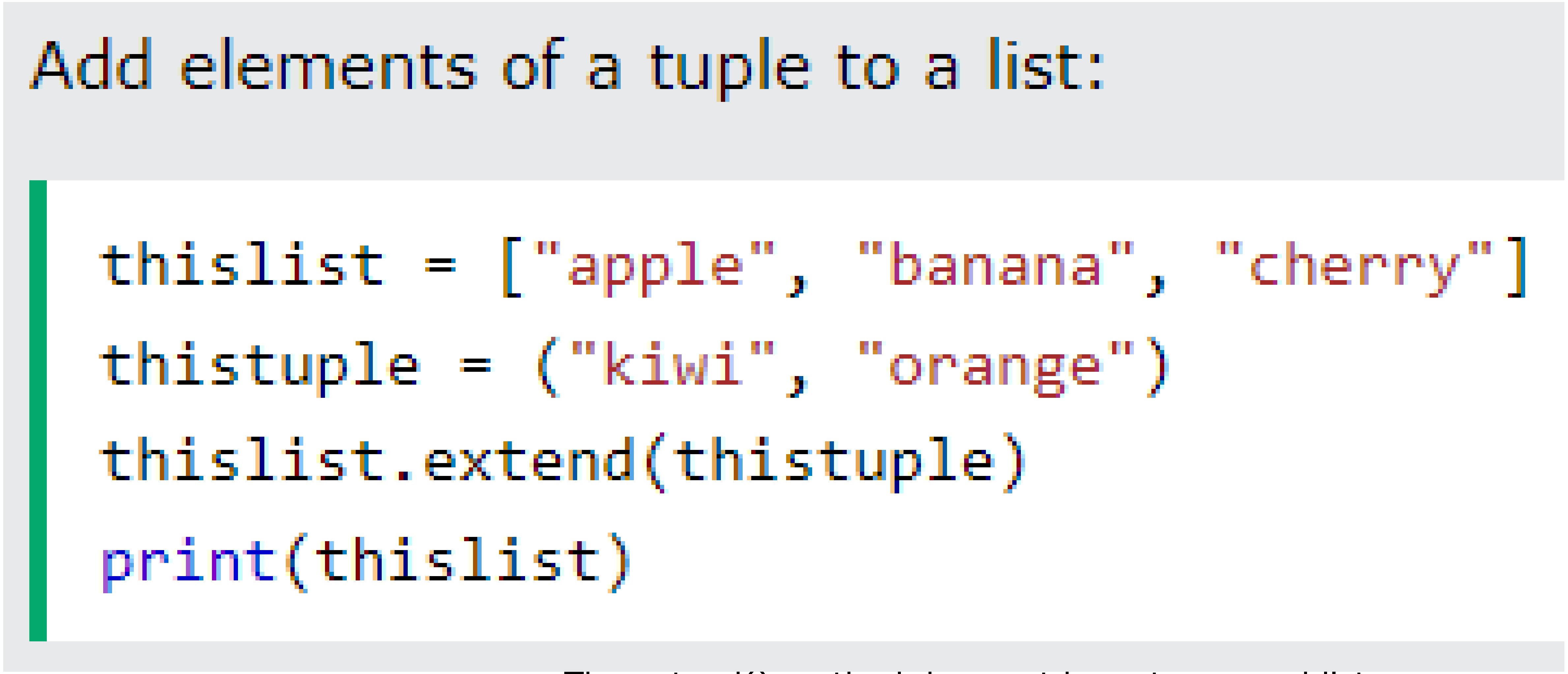


Add the elements of `tropical` to `thislist`:

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

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

Add Any Iterable



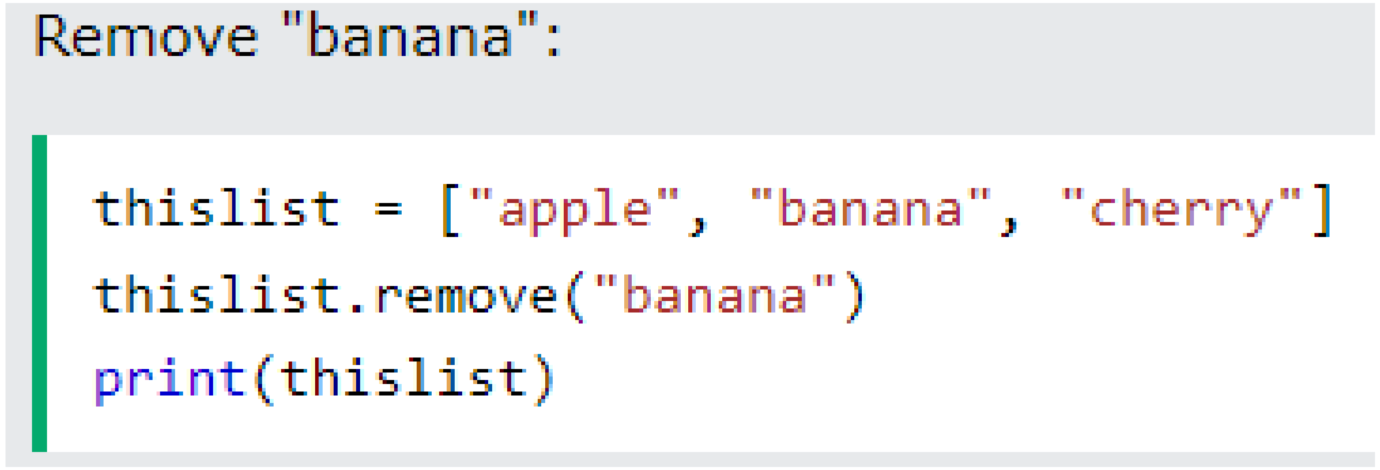
Add elements of a tuple to a list:

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

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

REMOVE LIST ITEMS

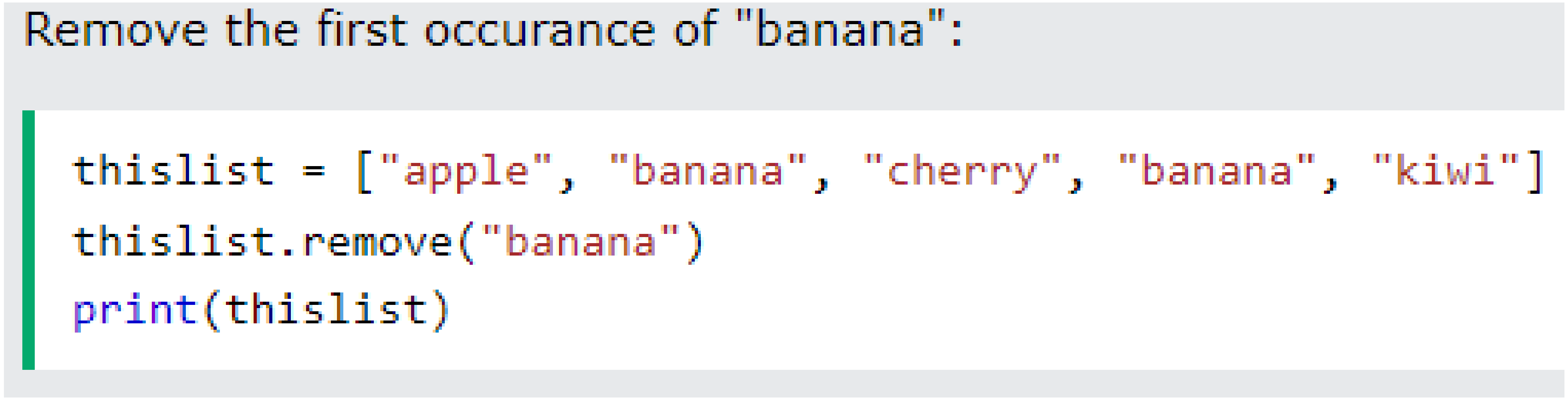
Remove Specified Item



Remove "banana":

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

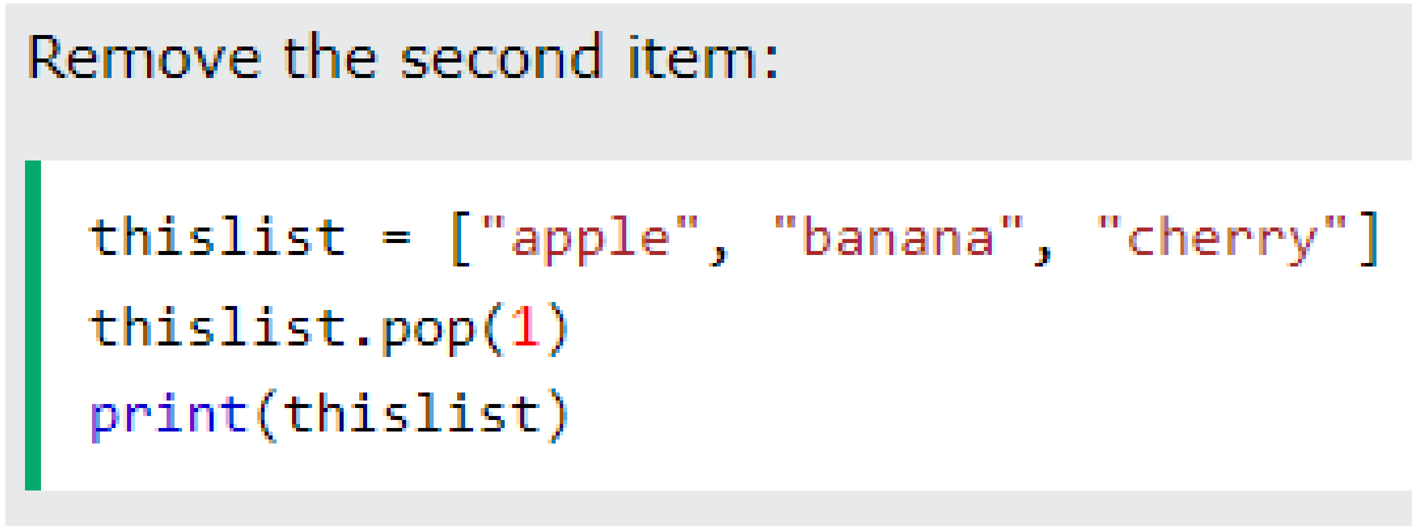
If there are more than one item with the specified value, the remove() method removes the first occurance



Remove the first occurance of "banana":

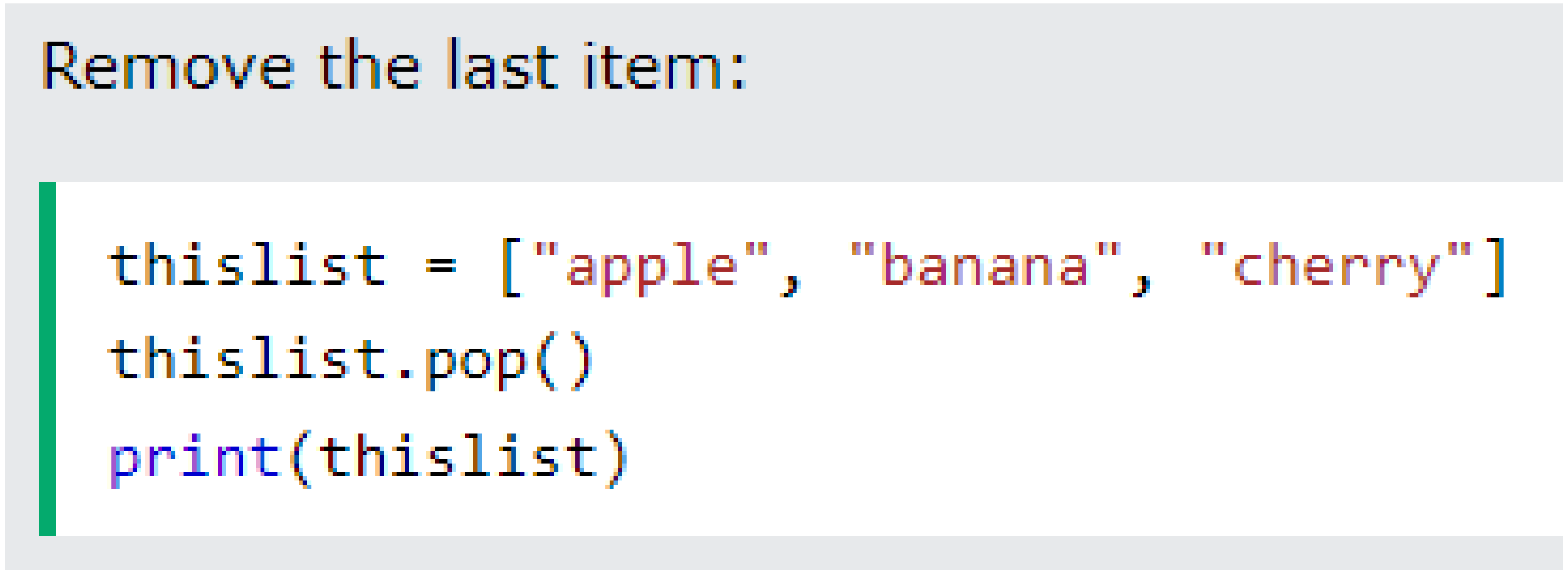
```python
thislist = ["apple", "banana", "cherry", "banana", "kiwi"]
thislist.remove("banana")
print(thislist)
```

Remove Specified Index



Remove the second item:

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

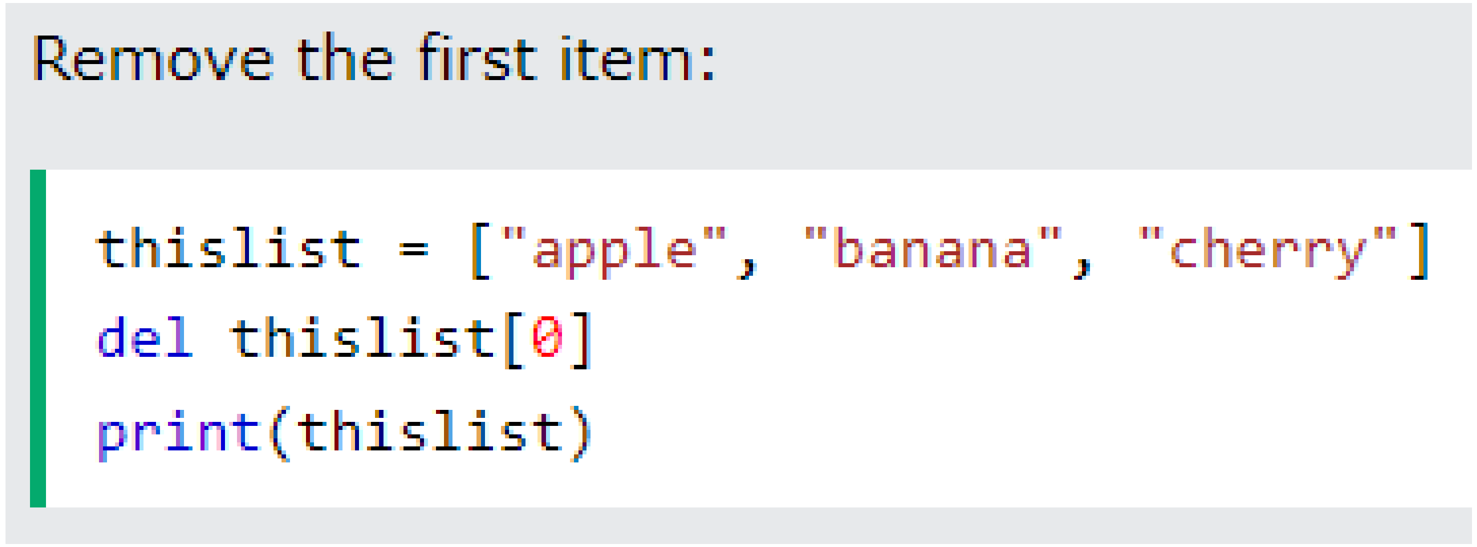


Remove the last item:

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

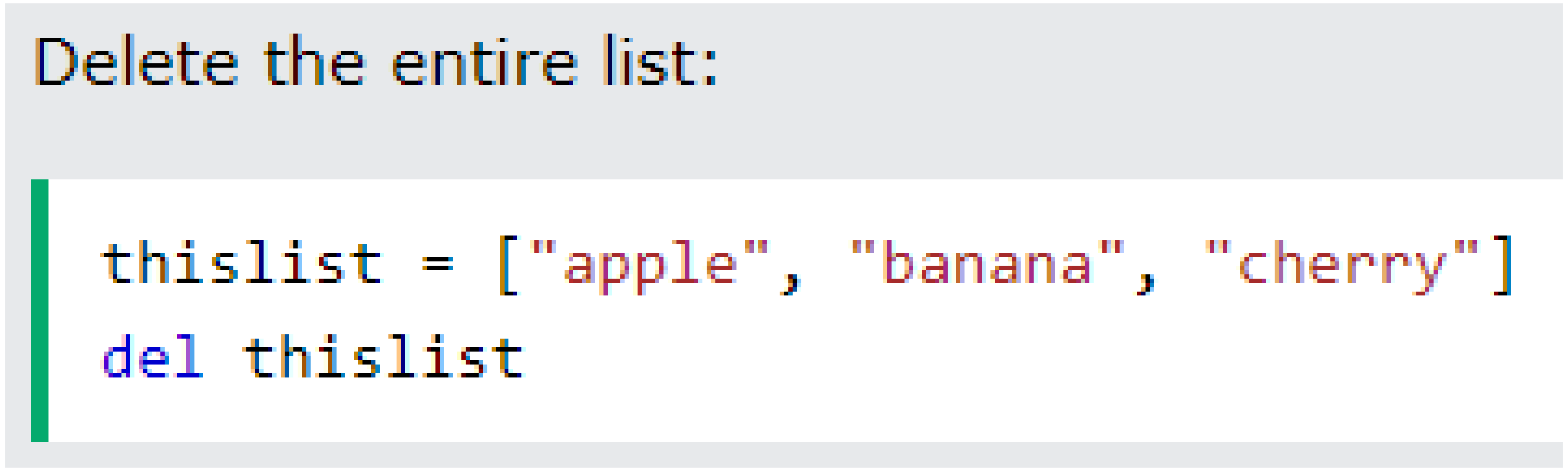
If you do not specify the index, the pop() method removes the last item.

Remove Specified Index



Remove the first item:

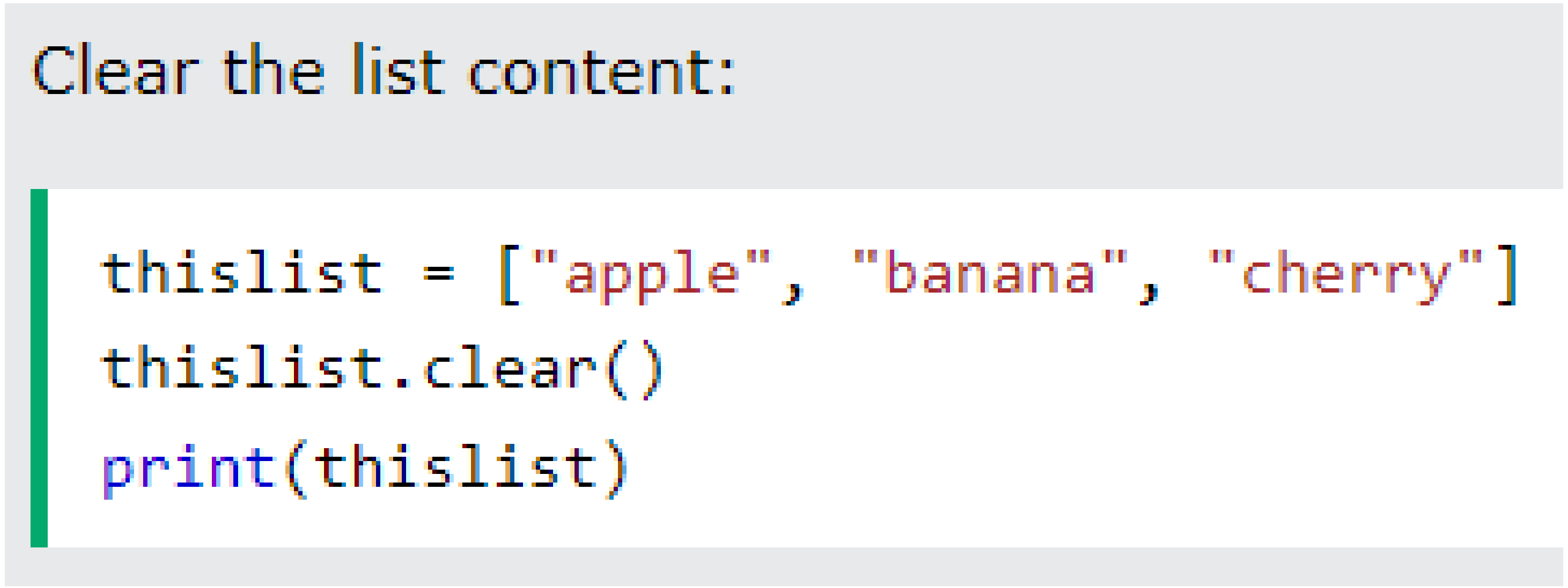
```
thislist = ["apple", "banana", "cherry"]
del thislist[0]
print(thislist)
```



Delete the entire list:

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

Clear the List

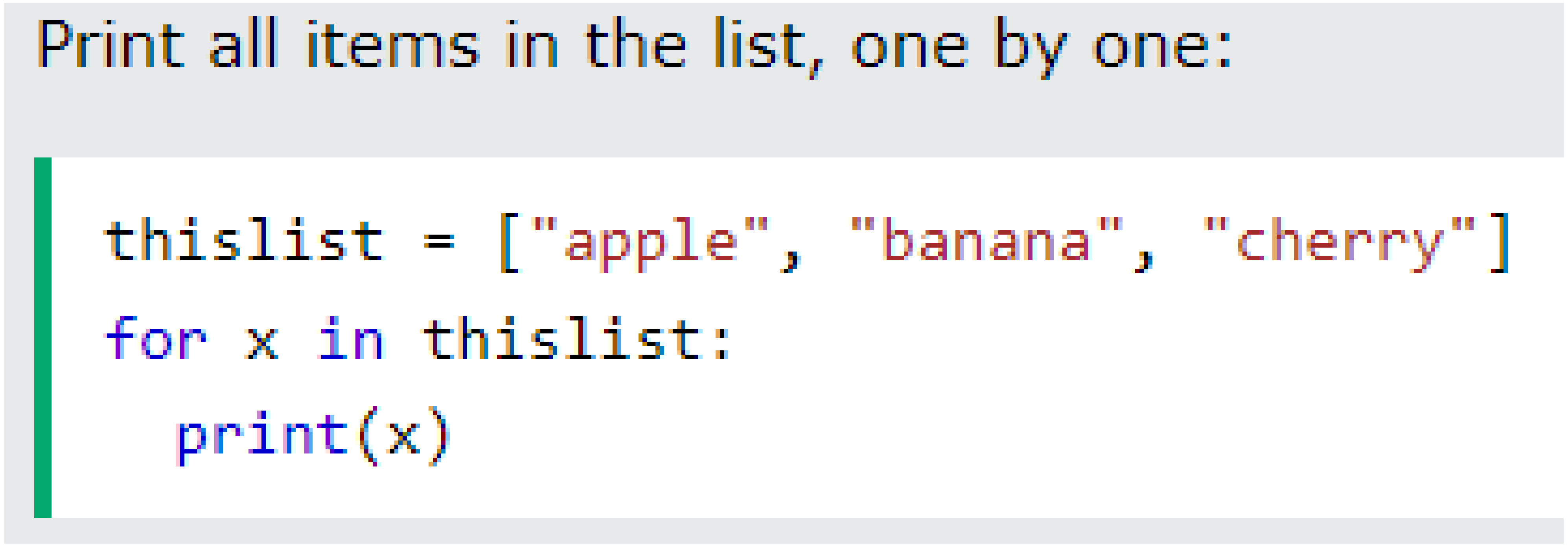

Clear the list content:

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

The clear() method empties the list. The list still remains, but it has no content.

LOOP LISTS

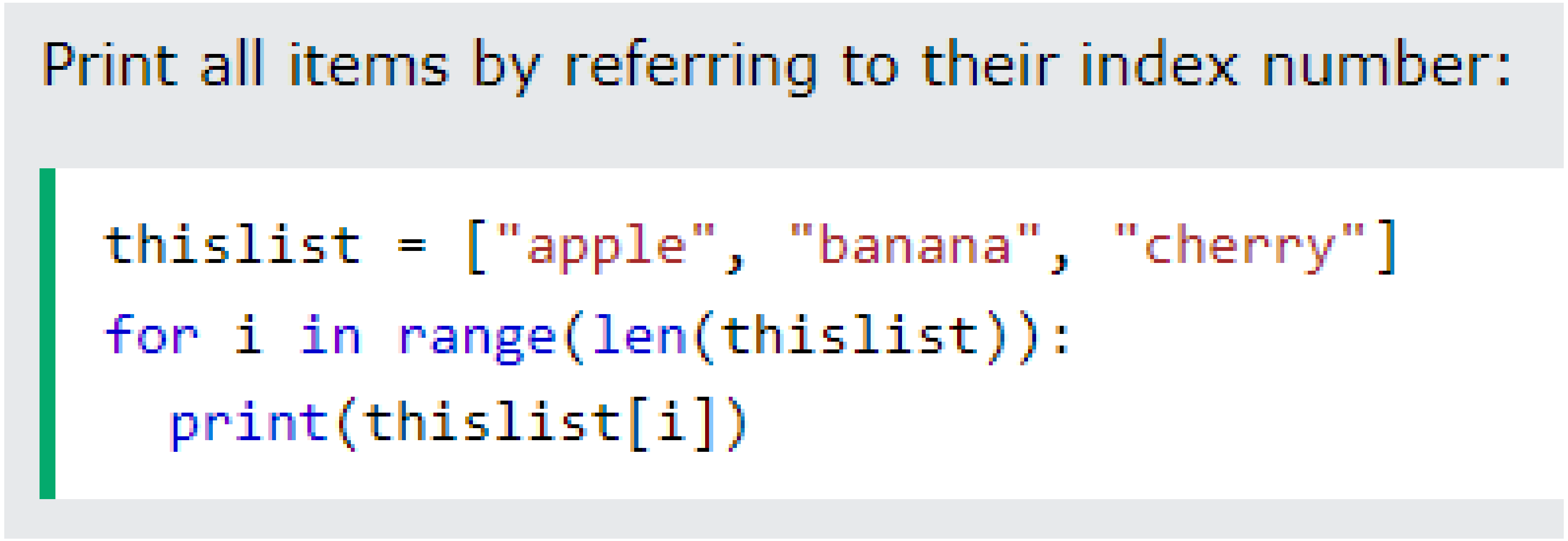
Loop Through a List



Print all items in the list, one by one:

```
thislist = ["apple", "banana", "cherry"]
for x in thislist:
  print(x)
```

Loop Through the Index Numbers

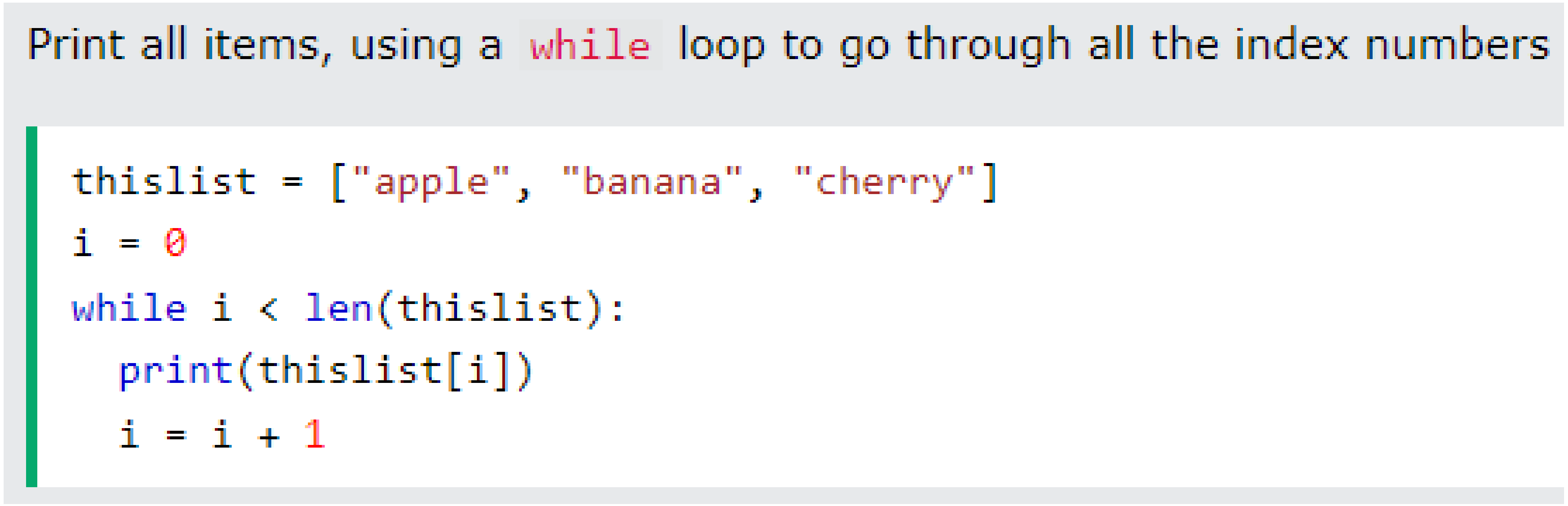


Print all items by referring to their index number:

```
thislist = ["apple", "banana", "cherry"]
for i in range(len(thislist)):
  print(thislist[i])
```

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

Using a While Loop



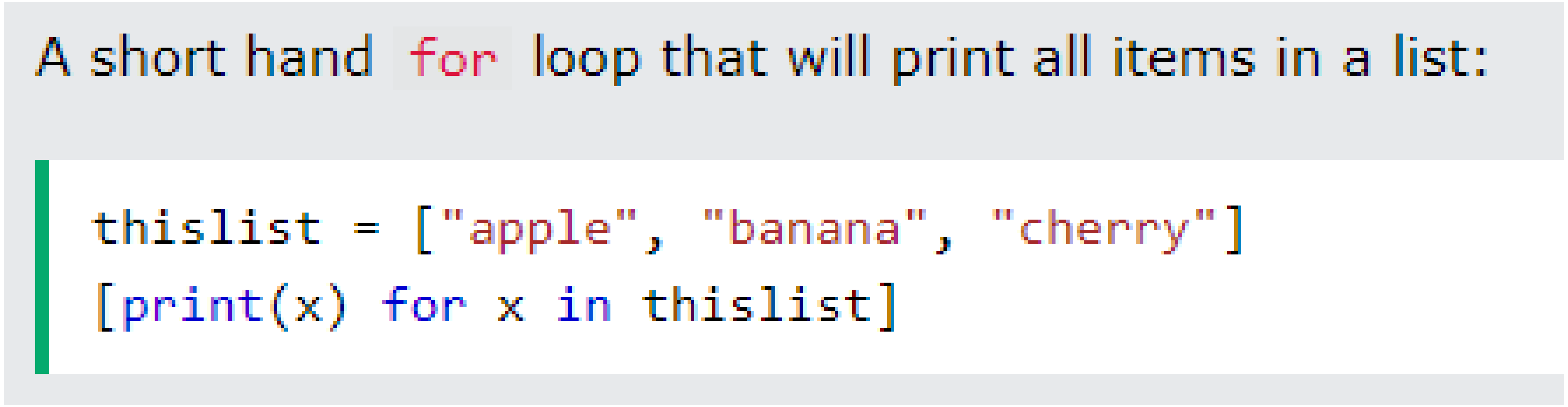
Print all items, using a `while` loop to go through all the index numbers

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
  print(thislist[i])
  i = i + 1
```

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.

Looping Using List Comprehension



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

```
thislist = ["apple", "banana", "cherry"]
[print(x) for x in thislist]
```

LIST COMPREHENSION

List Comprehension

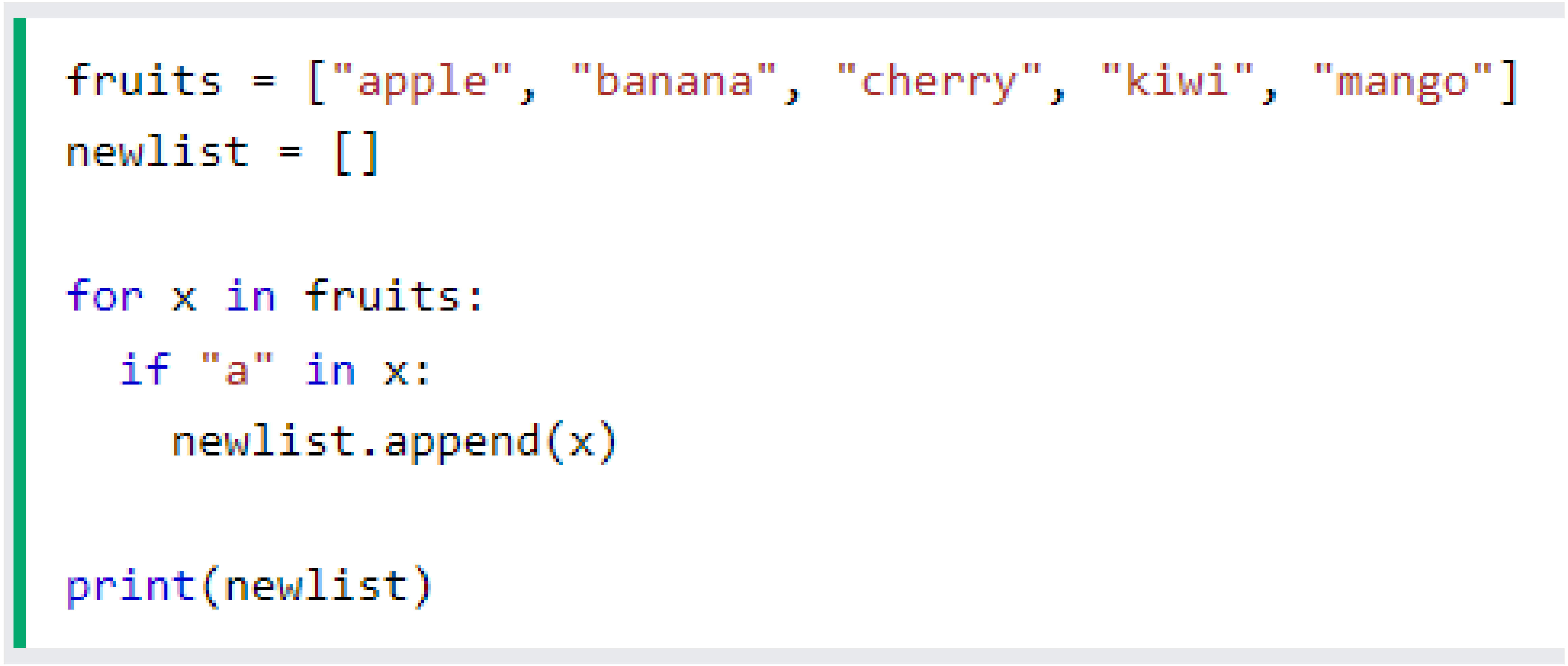
List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

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:

List Comprehension



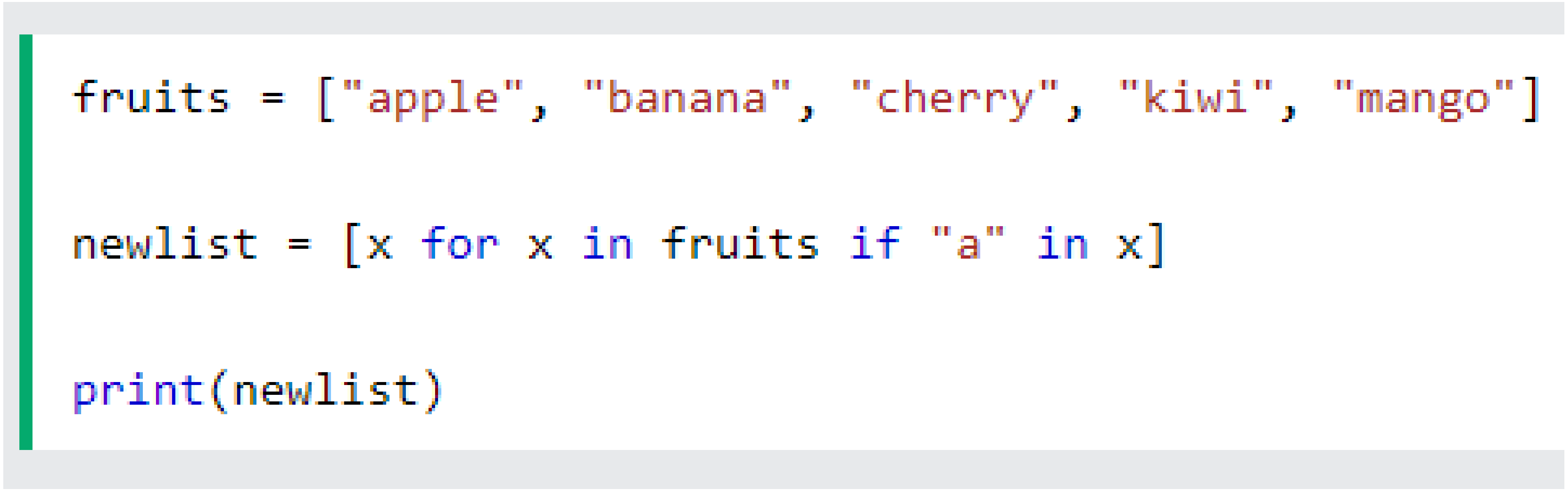
```python
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

print(newlist)
```

List Comprehension

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)
```

List Comprehension

The Syntax

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

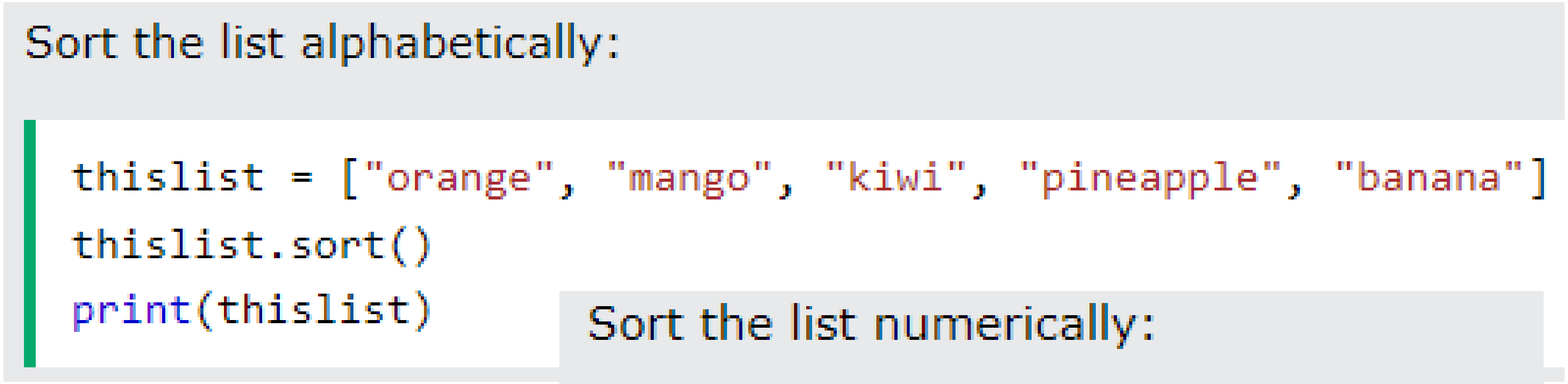
The return value is a new list, leaving the old list unchanged.

- *condition* is like a filter that only accepts the items that valuate to True.
- *iterable* can be any iterable object, like a list, tuple, set etc.
- *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

SORT LISTS

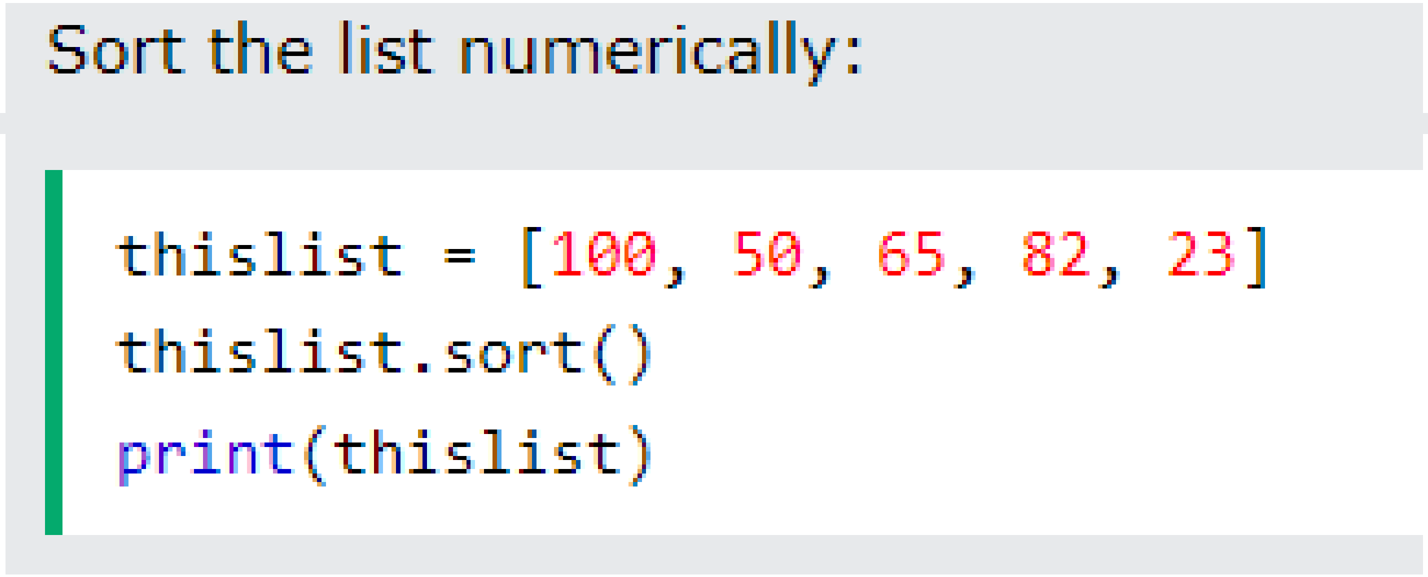
Sort List Alphanumerically

sort() method that will sort the list alphanumerically, ascending, by default



Sort the list alphabetically:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

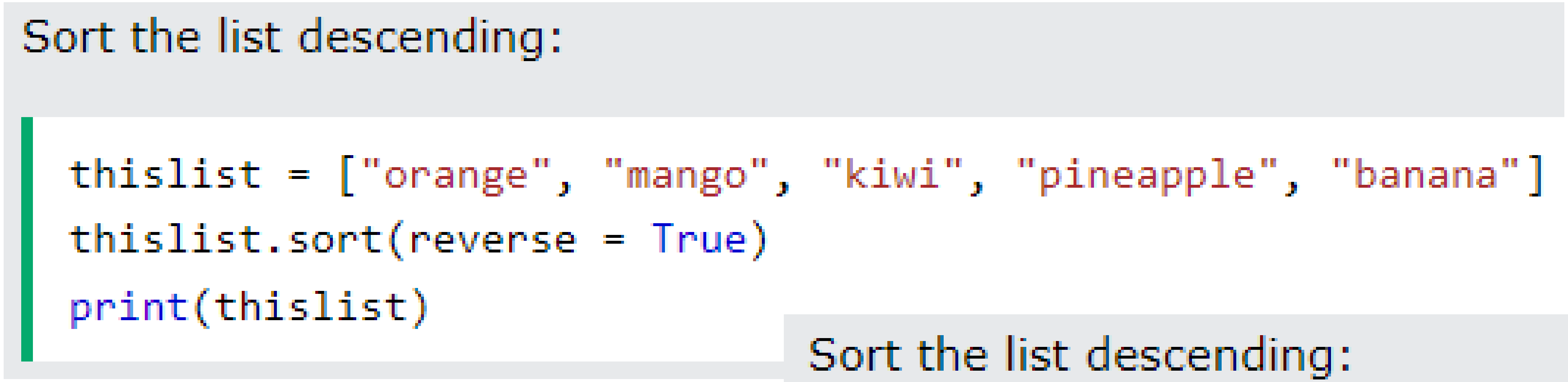


Sort the list numerically:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort()
print(thislist)
```

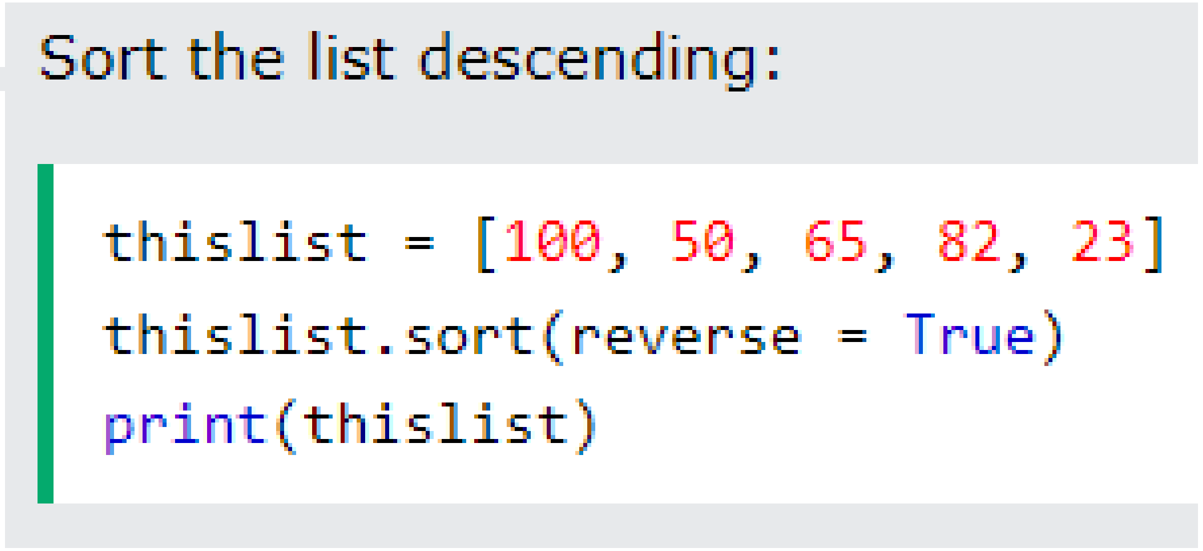
Sort Descending

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



Sort the list descending:

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

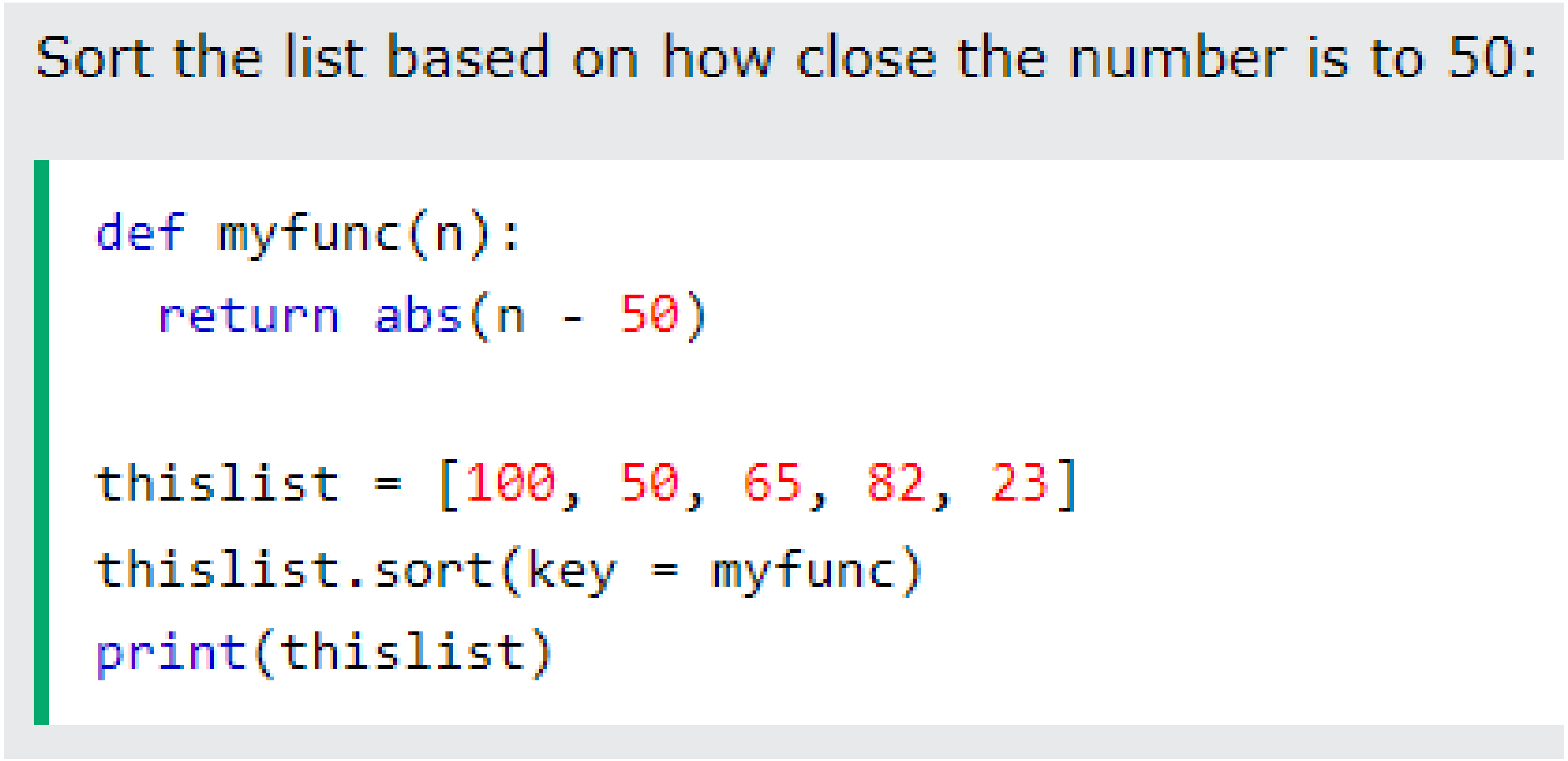


Sort the list descending:

```
thislist = [100, 50, 65, 82, 23]
thislist.sort(reverse = True)
print(thislist)
```

Customize Sort Function

customize your own function by using the keyword argument key = function
The function will return a number that will be used to sort the list (the lowest number first)

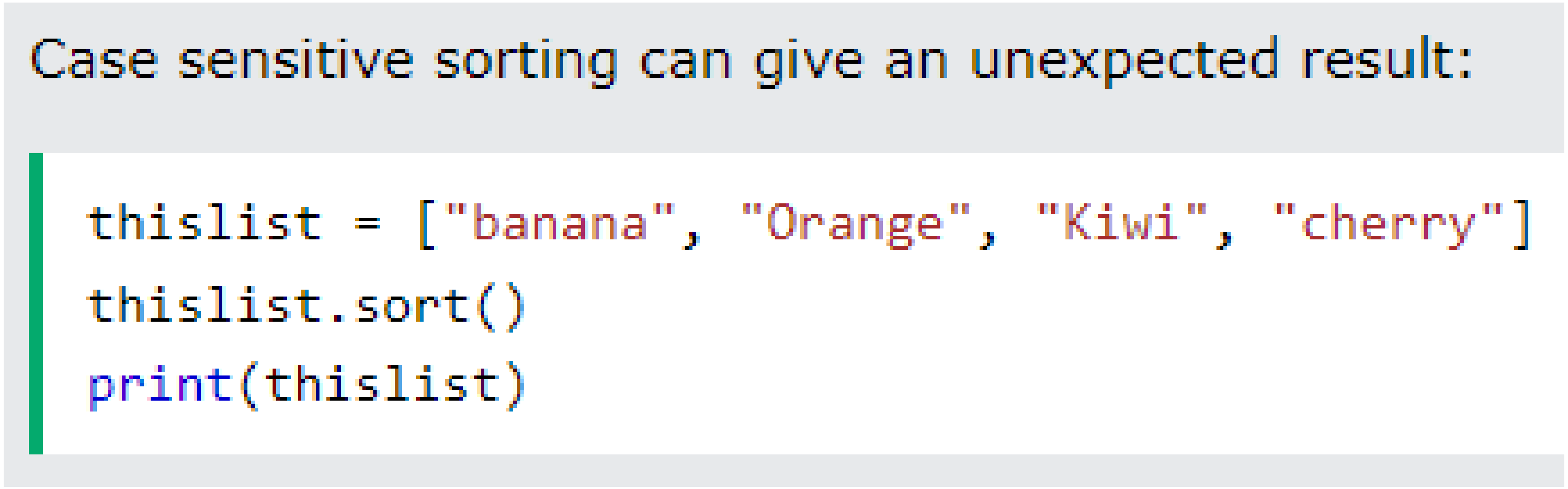

Sort the list based on how close the number is to 50:

```
def myfunc(n):
  return abs(n - 50)

thislist = [100, 50, 65, 82, 23]
thislist.sort(key = myfunc)
print(thislist)
```

Case Insensitive Sort

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

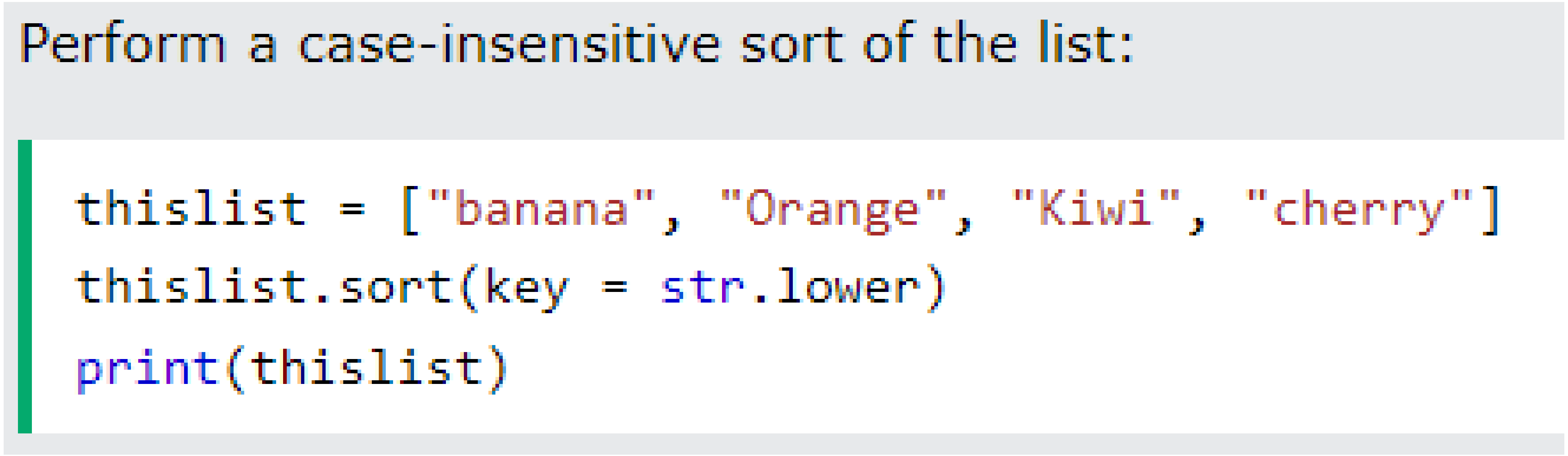


Case sensitive sorting can give an unexpected result:

```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.sort()
print(thislist)
```

Case Insensitive Sort

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

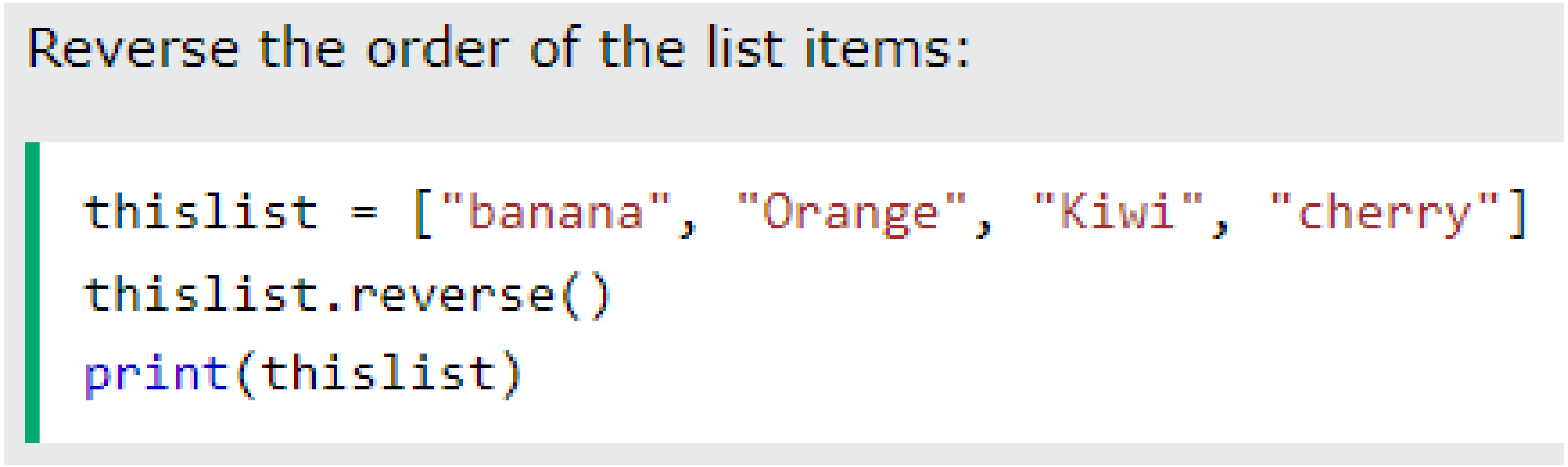


Perform a case-insensitive sort of the list:

```
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.



Reverse the order of the list items:

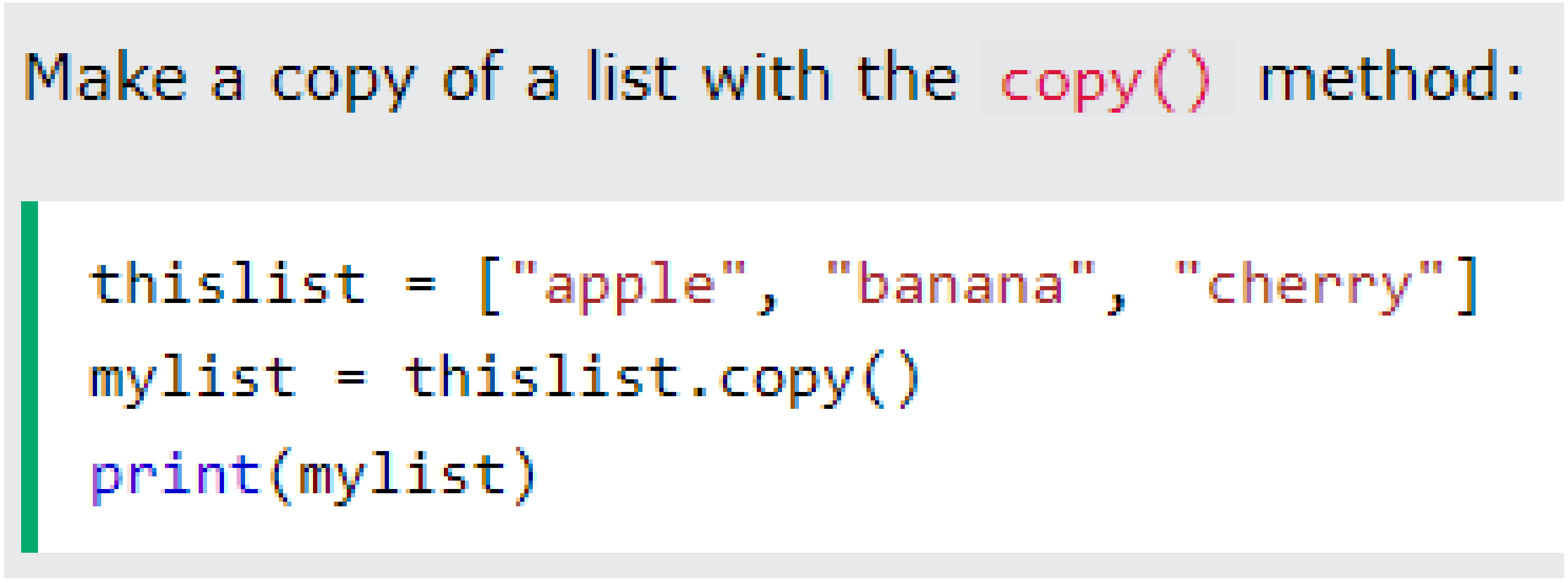
```
thislist = ["banana", "Orange", "Kiwi", "cherry"]
thislist.reverse()
print(thislist)
```

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().

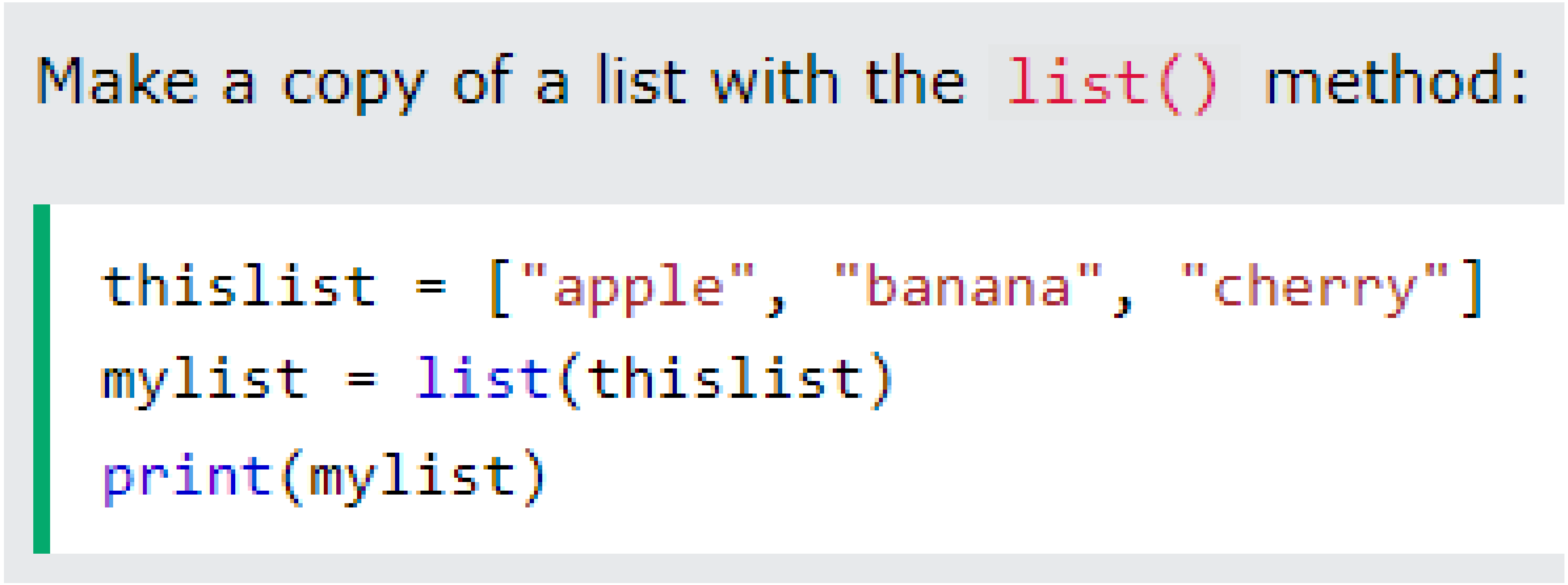


Make a copy of a list with the `copy()` method:

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

Copy a List

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

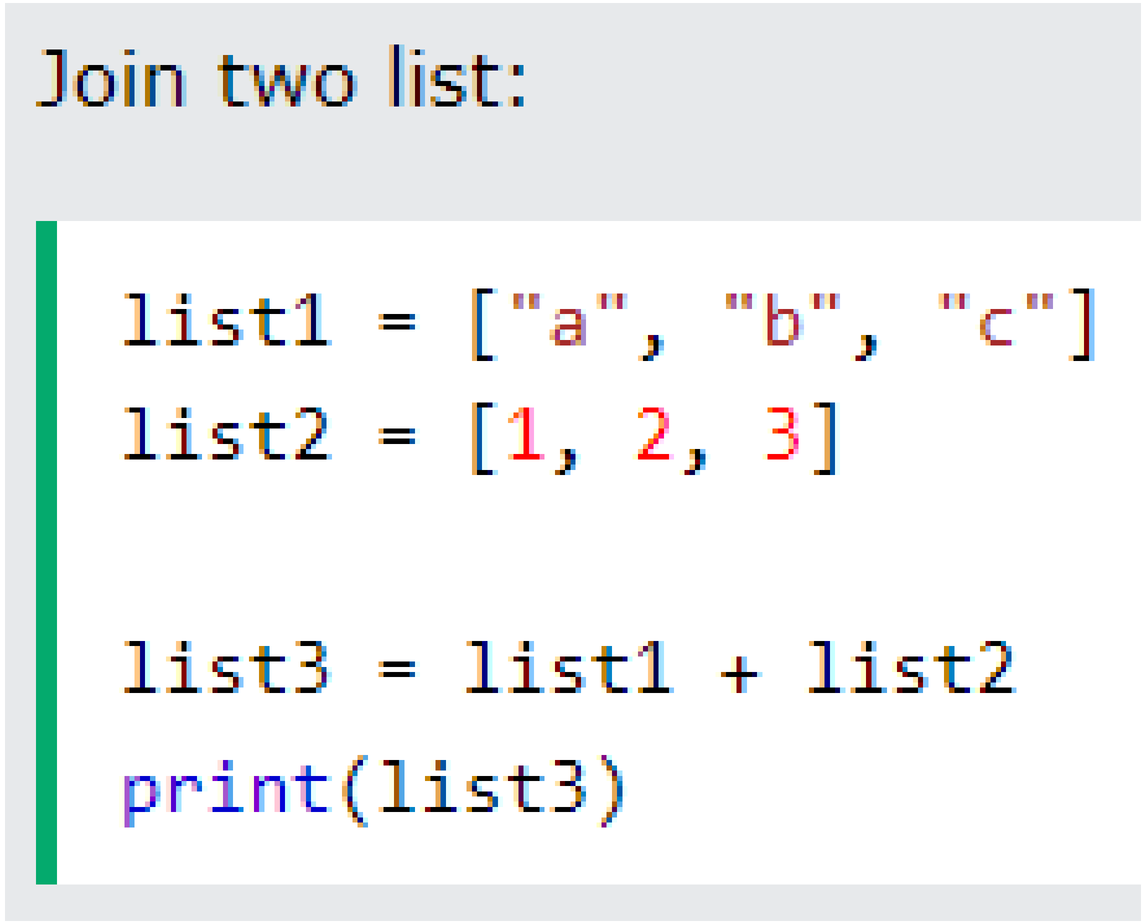

Make a copy of a list with the `list()` method:

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

JOIN LISTS

Join Two Lists

using the + operator

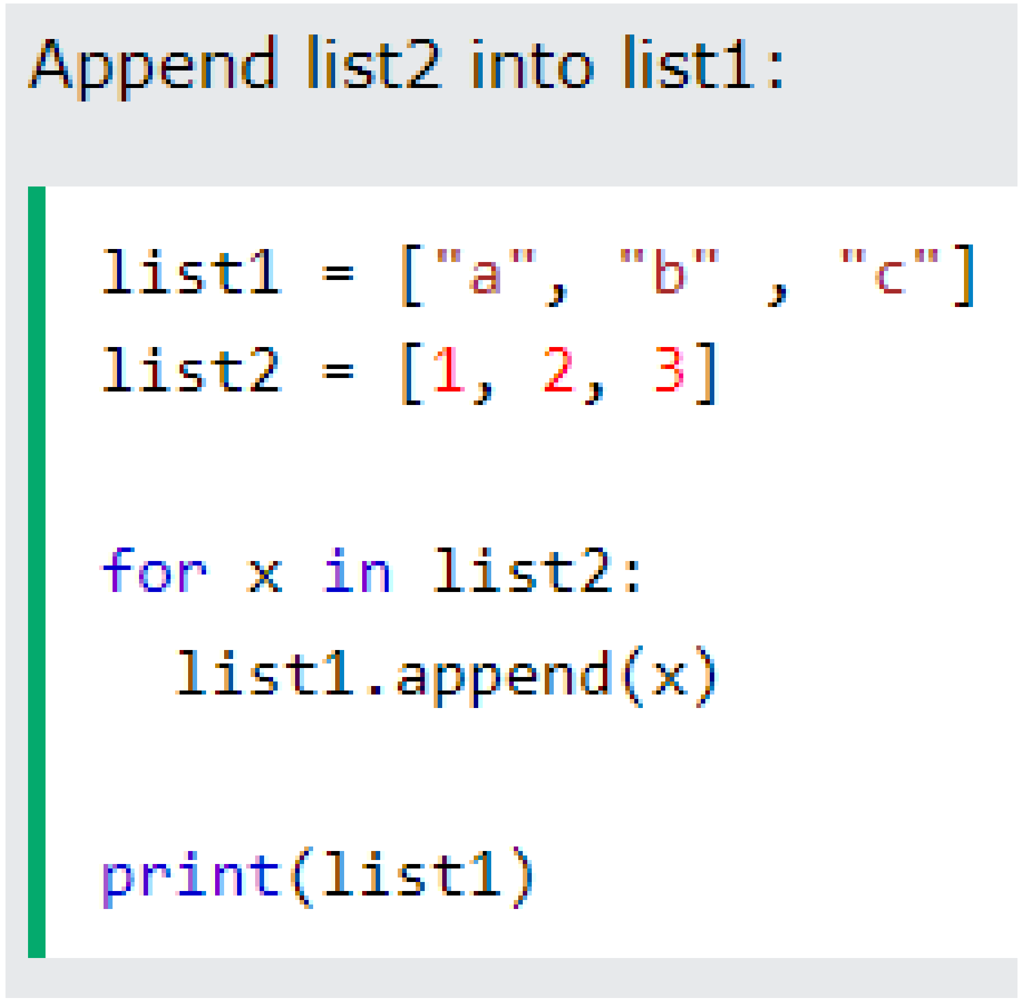


Join two list:

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

list3 = list1 + list2
print(list3)
```

appending all the items from list2 into list1, one by one



Append list2 into list1:

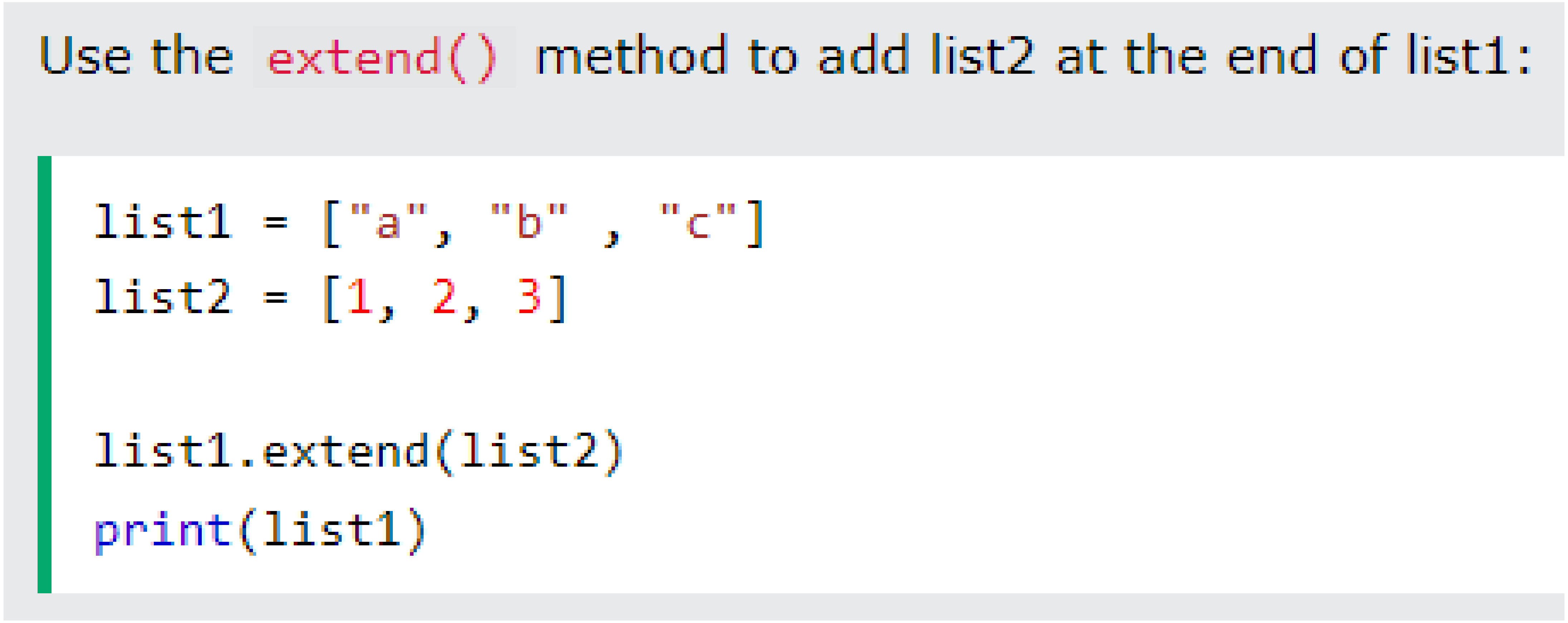
```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]

for x in list2:
  list1.append(x)

print(list1)
```

Join Two Lists

extend() method, where the purpose is to add elements from one list to another list



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)
```

LIST METHODS

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

Applications???

Any questions???

REFERENCES:

- Learn Python Programming. (2023). https://www.tutorialsteacher.com/python
- Python Tutorial. (2022). https://www.w3resource.com/python/python-tutorial.php
- Python Tutorial. (n.d.). https://www.tutorialspoint.com/python/index.htm
- Python Tutorial. (n.d.). https://www.w3schools.com/python/default.asp