Manipulate Lists

by Sophia

WHAT'S COVERED

In this lesson, you will learn about various functions and methods that can be used to manipulate lists. Specifically, this lesson covers:

1. Operators

Lists are especially useful as there are a number of operators that can be used with lists. We covered many of these operators when we were working with string variables. As we will see, these operators can work with any object within Python. Let's see what these common operators can do with lists.

1a. in Operator

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

The `in` operator that we learned about in a previous lesson also works on lists. We can check the contents of a list to see whether it already contains an element or not. This can be quite helpful to determine if an element is included in a list. For example, if we want to check if “dog” is one of the entries in a list called `petsList`, we can check for that.

TRY IT

Directions: Add the following code to the IDE and run to see if "dog" is in the `petsList` list.

```
petsList = ['dog', 'cat', 'fish']
print('dog' in petsList)
```

As we can see, this comes back as True; "dog" is on the list.

```
True
```

In the code above, we have our list called `petsList` with three strings containing "dog", "cat", and "fish". On the next line, we are checking if the string “dog” is in the list `petsList`. If it is, it returns True; if it is not, it returns False. Let’s check for an element that’s not in there, like “monkey”. We should see the result being False.

TRY IT

Directions: Change the code to look for “monkey” and run it.

```
petsList = ['dog', 'cat', 'fish']
print('monkey' in petsList)
```

As we expected, there was no "monkey" in the list.

```
False
```

1b. = Operator

If you recall, the `=` operator (equal sign) is an assignment operator that is used to assign values to variables. We can assign values to lists as easily as we can assign a value to a variable. We can also replace elements in the list using the same operator. We just have to ensure that we include the index number in square brackets to indicate which element we want to change.

TRY IT

Directions: Enter the following code and run it to see the element replacement.

```
petsList = ['dog', 'cat', 'fish']
petsList[1] = 'hamster'
print(petsList)
```

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

The assignment operator changed the value.

```
['dog', 'hamster', 'fish']
```

In the example above, we changed the element in index 1 (which is the second element) to "hamster". Remember the index starts at 0. If we wanted to switch out "dog" for "hamster", we would have needed to use `petsList[0]` as the index.

Note that if we forget to include the index, the `petsList` list now becomes a single string that contains "hamster" and is no longer a list. Python thinks we wanted to replace the full list with one animal.

TRY IT

Directions: Try it and see if you “accidentally” remove all pets for a single "hamster".

```
petsList = ['dog', 'cat', 'fish']
petsList = 'hamster'
print(petsList)
```

Here is that output.

```
hamster
```

Maybe now we can cut back on all the pet food...

1c. + Operator

If we recall, the `+` operator—or addition operator—can also serve as a concatenation operator with string variables, which means it joins the variables by linking them end to end. This operator can also concatenate lists together, which can be useful if we need to combine them. Note that when we use the `+` operator, the second list is concatenated to the end of the first list.

TRY IT

Directions: Go ahead and try concatenating the following lists.

```
firstList = [1, 2, 3]
secondList = [4, 5, 6]
thirdList = firstList + secondList
print(thirdList)
```

Output with a joined list.

```
 [1, 2, 3, 4, 5, 6]
```

Did you get a larger list now? Super!

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

1d. * Operator

We can use the `*` operator, or multiplication operator, as well. Remember, this operator also serves as a replication operator when used with strings and integers. We can replicate or repeat a list a given number of times. In this case, we'll declare `myList` as a list with one element that contains the value 0. Then we'll use the `*` operator to repeat it 4 times and place the result in `resultList`.

TRY IT

Directions: Go ahead and try the `*` operator in this code snippet.

```
myList = [0]
resultList = myList* 4
print(resultList)
```

With the following output.

```
[0, 0, 0, 0]
```

Next, this example uses a list that has three strings and repeats the list three times, to contain nine strings in the end.

Directions: Try this operator again.

```
petsList = ['dog', 'cat', 'fish']
resultList = petsList * 3
print(resultList)
```

Did you get a longer list?

```
['dog', 'cat', 'fish', 'dog', 'cat', 'fish', 'dog', 'cat', 'fish']
```

1e. slice Operator

The last operator we will cover is the `slice` operator. The **slice operator** also works on lists to allow us to return or update specific elements within the list. The `slice` operator works on any data collection types that are ordered, so this operator will work with lists and tuples. To return a list with specific elements, we would use a format like the following, where the list name is first used, then "a" is the first index to start the slice, and "b" is the last index to end the slice. The `slice` operator uses a colon (:) between the first and last index to separate them.

`listname`[a:b]

Remember that the first index number will continue to use 0 as the starting index and will use the length of the string as the last index. Also if including the last index with the `slice` operator, we can think of it as b - 1 in the

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

example above since the last index number will use 1 as the reference of the beginning of the string.

TRY IT

Directions: Try the `slice` operator on the code below.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petSlice = petsList[1:3]
print(petSlice)
```

The updated output.

```
['cat', 'fish']
```

The `slice` returns only "cat" and "fish" from the `petsList` list.

If we don't have the first index, the `slice` starts from the beginning.

Directions: Try the `slice` operator with the first index missing.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petSlice = petsList[:3]
print(petSlice)
```

The first index missing from the `slice` operator gives this output.

```
['dog', 'cat', 'fish']
```

Next, if we don't have the second index listed, the `slice` goes to the end.

Directions: Try the `slice` operator with the second index missing.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petSlice = petsList[3:]
print(petSlice)
```

The second index missing from the `slice` operator gives this output.

```
['rabbit', 'hamster', 'bird']
```

Lastly, if we don't include either of the first or last indexes and just have the colon in between, the `slice` is just a copy of the whole list.

Directions: Try the `slice` operator with neither index identified.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petSlice = petsList[:]
print(petSlice)
```

Here is that output with no indexes.

```
['dog', 'cat', 'fish', 'rabbit', 'hamster', 'bird']
```

As expected, we get the full list of pets since there was no `slice` indicated.

We can also use the slices to replace and update multiple elements within a list at once.

Directions: Try the `slice` operator below to replace elements.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petsList[1:3] = ['dinosaur','robot']
print(petsList)
```

The replacement output.

```
['dog', 'dinosaur', 'robot', 'rabbit', 'hamster', 'bird']
```

"Cat" and "fish" were replaced with "dinosaur" and "robot" since we used indexes [1:3]. Using those indexes, we swapped the second element "cat" (in position 1 since "dog" is in position 0) and the third element ("fish" was in position 2). Since 3 was the second index, the `slice` works up to that value but does not include that value. So, the fourth element "rabbit" in position 3 was not affected.

slice

The `slice` operator also works on lists to allow us to return or update specific elements within the list. The `slice` operator works on any data collection types that are ordered, so this operator will work with lists and tuples.

2. Methods and Functions

Python also has built-in methods that can be used to work with lists. So far, our lists have just been declared and defined. However, after a list is created we may want to add or remove elements from the list in the same way we would do with an online shopping cart.

2a. .append() Method

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

One such method is the **.append() method** which allows us to add a new element to the end of a list or other data collection type.

TRY IT

Directions: Try using the `.append()` method by adding "dinosaur" to the end of the `petsList` list.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petsList.append('dinosaur')
print(petsList)
```

Here is that output.

```
['dog', 'cat', 'fish', 'rabbit', 'hamster', 'bird', 'dinosaur']
```

In this example, we had a list of pets already. When we appended the "dinosaur" and then output the list to the screen, it was placed at the end of the list.

2b. .insert() Method

However, there are other times when we may want to place something in a specific order within the list. To do that, we can use the `.insert()` method. The **.insert() method** allows us to add an element to the list (or other data collection type) in any position.

TRY IT

Directions: Try using the `.insert()` method by adding "dinosaur" to a specific location within the `petsList` list.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petsList.insert(3,'dinosaur')
print(petsList)
```

Output with "dinosaur" inserted.

```
['dog', 'cat', 'fish', 'dinosaur' 'rabbit', 'hamster', 'bird']
```

In the example above, we are using the `.insert()` method on the `petsList` list. As part of the method parameters, we pass in the index position where we want to insert the element and the element that we want to insert. As such, we inserted the "dinosaur" into index 3 (which is the 4th element in the list). If we wanted to insert it into the first position, we would use 0 for the position.

Directions: Try using the `.insert()` method again, this time adding "dinosaur" to the beginning of the list.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petsList.insert(0,'dinosaur')
print(petsList)
```

Output with insertion at beginning of the list.

```
['dinosaur', 'dog', 'cat', 'fish', 'rabbit', 'hamster', 'bird']
```

2c. .extend() Method

The **.extend() method** takes a list (or other data collection type) as an argument and then appends (or adds) all of those elements to the end of another list or data collection.

Directions: Try using the `.extend()` method by extending the `secondPetsList` list onto the `firstPetsList` list.

```
firstPetsList = ['dog', 'cat', 'fish']
secondPetsList = ['rabbit','hamster']
firstPetsList.extend(secondPetsList)
print(firstPetsList)
```

Output with `secondPets` extended into `firstPets` list.

```
['dog', 'cat', 'fish', 'rabbit', 'hamster']
```

Note that the `secondPetsList` list is unchanged.

2d. .sort() Method

The **.sort() method** arranges all of the elements in the list (or other data collection type) from the lowest to highest. If the values are strings, the result will be a list of strings in alphabetical order.

Directions: Try using the `.sort()` method on the `petsList`.

```
petsList = ['dog', 'cat', 'fish','rabbit','hamster','bird']
petsList.sort()
print(petsList)
```

Here is that output.

```
['bird', 'cat', 'dog', 'fish', 'hamster', 'rabbit']
```

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

This results in "bird" to "rabbit" in alphabetical order.

If the list contains numerical values, they would be ordered from smallest to largest.

Directions: Try using the `.sort()` method again on a numerical list.

```
numList = [2, 45, 9, 17, 1, 2]
numList.sort()
print(numList)
```

Output with `.sort()` method on numerical list.

```
[1, 2, 2, 9, 17, 45]
```

Did you get the lists to show up in the expected order?

2e. .pop() Method

If we need to remove elements from a list, there are some different ways that this can be used. The **.pop() method** modifies the list (or other data collection type) and returns the element that was removed. This is useful if we need to do something specific with the element that was removed.

TRY IT

Directions: Try using the `.pop()` method to remove and output an element from the `petsList` list.

```
petsList = ['dog', 'cat', 'fish', 'rabbit','hamster','bird']
removedPet = petsList.pop(3)
print("Remaining: ", petsList)
print("Removed: ", removedPet)
```

Output with removed value.

```
Remaining:  ['dog', 'cat', 'fish', 'hamster', 'bird']
Removed:  rabbit
```

We see that "rabbit" was removed and what is still included in the `petsList` list. We added a variable called `removedPet` so we could print out what element was removed.

If we don't pass in an index, the list simply deletes and returns the last element in the list.

Directions: Try using the `.pop()` method again, this time without an index number.

```
petsList = ['dog', 'cat', 'fish', 'rabbit','hamster','bird']
removedPet = petsList.pop()
```

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
print("Remaining: ", petsList)
print("Removed: ", removedPet)
```

Output without using an index number.

```
Remaining:  ['dog', 'cat', 'fish', 'rabbit', 'hamster']
Removed:  bird
```

2f. .del Statement

We can use the `del` statement (delete) if we don't need to worry about the element that was removed. The **del** statement deletes an element but does not return anything.

TRY IT

Directions: Try using the `del` statement and permanently remove an element.

```
petsList = ['dog', 'cat', 'fish', 'rabbit','hamster','bird']
del petsList[3]
print("Remaining: ", petsList)
```

Output using the `del` statement.

```
Remaining:  ['dog', 'cat', 'fish', 'hamster', 'bird']
```

We no longer have a "rabbit" as part of our `petsList` list.

2g. .remove Method

We can use the **.remove() method** if we know the element we wish to remove (but not the index). The `.remove()` method takes away the first instance that the element occurs. If the element does not exist, using this method will raise an error.

TRY IT

Directions: Try using the `.remove()` method to find and remove an element.

```
petsList = ['dog', 'cat', 'fish', 'rabbit','hamster','bird']
petsList.remove('rabbit')
print("Remaining: ", petsList)
```

With this output.

```
Remaining:  ['dog', 'cat', 'fish', 'hamster', 'bird']
```

The element "rabbit" was found and removed.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

Let's try to see what it looks like with a second element of the same value in the list.

Directions: Try using the `.remove()` method again to find a duplicated element in the list.

```
petsList = ['dog', 'cat', 'fish', 'rabbit','cat','bird']
petsList.remove('cat')
print("Remaining: ", petsList)
```

With the following output.

```
Remaining:  ['dog', 'fish', 'rabbit', 'cat', 'bird']
```

Notice that only the "cat" in index 1 (position 2) is removed and not the one in index 4 (position 5).

There are a number of built-in functions that can be used on lists (or other data collection types) that allow us to quickly look through a list without writing our own code to do so. As a reminder on functions and methods:

- Methods are called from an object using the dot after the object. In the current case, these are lists, which is why we have the list name prior to the method name.

EXAMPLE

We use .append() method to append an item to a list.

```
petsList.append('bunny')
```

- Functions, on the other hand, do not include the list name prior to the function but take in the list as a parameter.

EXAMPLE

This is a function of getting the length of a list.

```
len(petsList)
```

Some built-in functions that are useful on lists include:

- The **max() function** (stands for maximum) returns the largest value in a list (or other data collection type).
- The **min() function** (stands for minimum) returns the smallest value in a list (or other data collection type).
- The **len() function** (stands for length) returns the number of elements in a list (or other data collection type).
- The **sum() function** returns the sum of all values in the list (or other data collection type).

We can combine them in other calculations, like finding the average of a list by dividing the sum of the list by the length of the list.

TRY IT

Directions: Enter the code below and see what each of the functions do.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
numList = [2, 45, 9, 17, 1, 4]
print("Max: ",max(numList))
print("Min: ",min(numList))
print("Length: ",len(numList))
print("Sum: ",sum(numList))
print("Average: ",sum(numList)/len(numList))
```

Output with all function usage.

```
Max:  45
Min:  1
Length:  6
Sum:  78
Average:  13.0
```

Did all your built-in functions work as intended?

Directions: Try changing the elements of a list, and adding some elements. Then try testing all the ways you just learned in this lesson to manipulate lists.

.append()

The `.append()` method allows us to add a new element to the end of a list or other data collection type.

.insert()

The `.insert()` method allows us to add an element to a list or other data collection type, in any position.

.extend()

The `.extend()` method takes a list (or other data collection type) as an argument and then appends (or adds) all of those elements to the end of another list or data collection.

.sort()

The `.sort()` method arranges all of the elements in a list (or other data collection type) from the lowest to highest. If the values are strings, the result will be a list of strings in alphabetical order.

.pop()

The `.pop()` method modifies the list (or other data collection type) and returns the element that was removed.

del

`del` is a reserved keyword and used as the del statement to delete an element. It does not return anything.

.remove()

Use the `.remove()` method if we know the element of a list (or other data collection type) we wish to remove (but not the index); the .remove() method takes away the first instance that the element occurs. If the element does not exist, using this method will raise an error.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

max()
The `max` function (stands for maximum) returns the largest value in a list (or other data collection type).

min()
The `min` function (stands for minimum) returns the smallest value in a list (or other data collection type).

len()
The `len` function (stands for length) returns the number of elements in a list (or other data collection type).

sum()
The `sum` function returns a number, sum of all values in a list (or other data collection type).

---

3. Iterables and Iterators

The next two functions we are going to discuss are the `iter()` and `next()`. First, we need to explain what iterators and iterables are. Lists and the other data collection types (sets, tuples, and dictionaries, which we will go into in more detail in the next lesson) are iterable types and can make use of iterators.

↪ EXAMPLE
For example, say we wanted to output each element of this `myPets` list.

```
myPets = ['hampster', 'cat', 'dog', 'horse', 'cow', 'chickens']
```

However, we wish to see the elements of this list one element at a time. Since a list is an iterable type, we can output like this.

```
hampster
cat
dog
horse
cow
chickens
```

What is an iterable?

The dictionary definition of **iteration** is the repetition of a process. For programming, that is basically the repetitive execution of code to do something. So, to be iterable means that the object can be used in iteration.

Iterable is just a fancy name for a Python object that is capable of going through its members one at a time. A list (or other data collection type) is an iterable object; it can return its elements one at a time.

What is an iterator? An **iterator**, on the other hand, is an object that contains a countable number of values, meaning we can figure out how many elements are in the object. More simply, the iterator is an object that can be used to traverse and move through all its values (elements) within an iterable object.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

All iterable objects have two specific built-in functions that are used with iterators:

- The **iter() function**, which is used to create an iterator by initializing the object that was passed to it.
- The **next() function**, which is used to move to the next value (element) in an iterable object.

We'll generally use them together.
To see an example of these two functions, first, let's create and define a list.

```
numList = [2, 45, 9, 17, 1, 4]
```

In the code above, we've created a list called `numList` and initialized it with 6 integers. Next, we'll create a variable called `listElement` and set it to the return value of the `iter()` function when we pass in the `numList` variable. This return value will point to the start of the list.

```
listElement = iter(numList)
```

Then, we can use the `next()` function to iterate through the `numList` list, with the first time we call the `next()` function returning the first element in the list.

```
print(next(listElement))
```

The output showing 2.

```
2
```

We have the first element of the list output to the screen.

So the complete program is the following code.

```
numList = [2, 45, 9, 17, 1, 4]
listElement = iter(numList)
print(next(listElement))
```

Each time we call the `next()` function, we'll move up by one element. Let's try it out and see what happens.

TRY IT

Directions: Enter the code below and see the `next()` function move through the iterable list numList.

```
numList = [2, 45, 9, 17, 1, 4]
listElement = iter(numList)
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
```

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

```
print(next(listElement))
print(next(listElement))
```

The output with each of the `numList` elements on each row as the output.

```
2
45
9
17
1
4
```

KEY CONCEPT

The easiest way to remember what the `iter()` and `next()` functions are doing is to think of lines of people and roll call. Say we have three lines of ten people. We can name the lines the red, blue, and green lines. Each line is made up of people (elements) and since we are conducting roll call, we are looking for the values of their names. Peter is at the front of the red line, Mary at the front of the blue, and Jane at the front of the green line. If we use the `iter()` function we identify the line we want to start roll call with. So, if we use iter(green), that identifies the green line and we immediately look to the front of the line, so we are looking directly at Jane. If we were to ask next(name) to sound off, Jane goes first since she has not yet had the chance to sound off. "Jane," she says. Then we can ask for next(name), and the second person in the green line would sound off. So on and so on, until the end of the line.

This example uses lines of people, and just like a list or other data collection type, you can use iteration (a repetitive process that in our case is a roll call) on this line of people. That makes our lines of people iterable groups (objects).

Note, though, that if we move to the next element and that element doesn't exist, we will get an error that is displayed.

TRY IT

Directions: Try entering an extra `next()` function that will not have an element.

```
numList = [2, 45, 9, 17, 1, 4]
listElement = iter(numList)
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
```

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

The output has an error.

```
2
45
9
17
1
4
Trackback (most recent call last):
   File "/home/main.py", line 9, in <module>
     print(next(listElement))
StopIteration
```

Also, note that strings are also iterable objects as they contain a sequence of characters. Let's try the same thing and see what happens.

TRY IT

Directions: Enter the code below with a string and see the `next()` function move through the string.

```
myString = "sophia"
listElement = iter(myString)
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
print(next(listElement))
```

Output with "sophia" broken down by letter.

```
s
o
p
h
i
a
```

We'll revisit this code when we cover loops in the next challenge and see some more efficient and effective ways to handle iterators, especially with the use of other types of data collections.

TERMS TO KNOW

Iteration

Iteration is the repetition of a process. For programming, that is basically the repetitive execution of code to do

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

something.

Iterable
Iterable sounds complex but it is just a fancy name for a Python object that is capable of going through its members one at a time.

Iterator
Iterator is a type of object that contains a countable number of values. More simply, the iterator is an object that can be used to traverse and move through all its values (elements) within an iterable object.

iter()
The `iter()` function is used to create an iterator by initializing the object that was passed to it.

next()
The `next()` function is used to move to the next value (element) in an iterable object.

SUMMARY

In this lesson, we learned about the **operators**, **methods**, and **built-in functions** that can be used to manipulate lists. Using these tools, we learned various ways that we can add or remove elements from lists, sort the lists, and perform calculations on lists, such as getting the minimum, maximum, sum, and length from lists that contain numeric elements. We also learned what being called **iterable** means and how Python provides functions to create **iterators** to be able to traverse through an iterable object one element at a time.

Best of luck in your learning!

Source: THIS CONTENT AND SUPPLEMENTAL MATERIAL HAS BEEN ADAPTED FROM “PYTHON FOR EVERYBODY” BY DR. CHARLES R. SEVERANCE ACCESS FOR FREE AT www.py4e.com/html3/ LICENSE: CREATIVE COMMONS ATTRIBUTION 3.0 UNPORTED.

TERMS TO KNOW

.append()
The .append() method allows us to add a new element to the end of a list or other data collection type.

.extend()
The .extend() method takes a list (or other data collection type) as an argument and then appends (or adds) all of those elements to the end of another list or data collection.

.insert()
The .insert() method allows us to add an element to a list or other data collection type, in any position.

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

.pop()

The .pop() method modifies the list (or other data collection type) and returns the element that was removed.

.remove()

Use the .remove() method if we know the element of a list (or other data collection type) we wish to remove (but not the index); the .remove() method takes away the first instance that the element occurs. If the element does not exist, using this method will raise an error.

.sort()

The .sort() method arranges all of the elements in a list (or other data collection type) from the lowest to highest. If the values are strings, the result will be a list of strings in alphabetical order.

Iterable

Iterable sounds complex but it is just a fancy name for a Python object that is capable of going through its members one at a time.

Iteration

Iteration is the repetition of a process. For programming, that is basically the repetitive execution of code to do something.

Iterator

Iterator is a type of object that contains a countable number of values. More simply, the iterator is an object that can be used to traverse and move through all its values (elements) within an iterable object.

del

del is a reserved keyword and used as the del statement to delete an element. It does not return anything.

iter()

The iter() function is used to create an iterator by initializing the object that was passed to it.

len()

The len function (stands for length) returns the number of elements in a list (or other data collection type).

max()

The max function (stands for maximum) returns the largest value in a list (or other data collection type).

min()

The min function (stands for minimum) returns the smallest value in a list (or other data collection type).

next()

The next() function is used to move to the next value (element) in an iterable object.

slice

The slice operator also works on lists which allows us to return or update specific elements within the list. The slice operator works on any data collection types that are ordered, so this operator will work with lists and tuples

sum()

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.

The sum function returns a number, sum of all values in a list (or other data collection type).

© 2025 SOPHIA Learning, LLC. SOPHIA is a registered trademark of SOPHIA Learning, LLC.