**Lists and Tuples in Python**

>>> a = ['foo', 'bar', 'baz', 'qux']

>>> print(a)

['foo', 'bar', 'baz', 'qux']

>>> a

['foo', 'bar', 'baz', 'qux']

### **Lists Are Ordered**

>>> a = ['foo', 'bar', 'baz', 'qux']

>>> b = ['baz', 'qux', 'bar', 'foo']

>>> a == b

False

>>> a is b

False

>>> [1, 2, 3, 4] == [4, 1, 3, 2]

False

### **Lists Can Contain Arbitrary Objects**

>>> a = [21.42, 'foobar', 3, 4, 'bark', False, 3.14159]

>>> a

[21.42, 'foobar', 3, 4, 'bark', False, 3.14159]

Lists can even contain complex objects, like functions, classes, and modules,

>>> int

<class 'int'>

>>> len

<built-in function len>

>>> def foo():

... pass

...

>>> foo

<function foo at 0x035B9030>

>>> import math

>>> math

<module 'math' (built-in)>

>>> a = [int, len, foo, math]

>>> a

[<class 'int'>, <built-in function len>, <function foo at 0x02CA2618>,

<module 'math' (built-in)>]

You can specify a stride—either positive or negative:

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[0:6:2]

['foo', 'baz', 'quux']

>>> a[1:6:2]

['bar', 'qux', 'corge']

>>> a[6:0:-2]

['corge', 'qux', 'bar']

The [:] syntax works for lists. However, there is an important difference between how this operation works with a list and how it works with a string.

>>> s = 'foobar'

>>> s[:]

'foobar'

>>> s[:] is s

True

If s is a string, s[:] returns a reference to the same object:

>>> s = 'foobar'

>>> s[:]

'foobar'

>>> s[:] is s

True

Conversely, if a is a list, a[:] returns a new object that is a copy of a:

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[:]

['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[:] is a

False

The [in and not in](https://realpython.com/python-in-operator/) operators:

>>> a

['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> 'qux' in a

True

>>> 'thud' not in a

True

The [concatenation](https://realpython.com/python-string-concatenation/) (+) and replication (\*) operators:

>>> a

['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a + ['grault', 'garply']

['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'grault', 'garply']

>>> a \* 2

['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'foo', 'bar', 'baz',

'qux', 'quux', 'corge']

The [len()](https://realpython.com/len-python-function/), [min(), and max()](https://realpython.com/python-min-and-max/) functions:

>>> a

['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> len(a)

6

>>> min(a)

'bar'

>>> max(a)

'qux'

#### **Modifying a Single List Value**

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a

['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[2] = 10

>>> a[-1] = 20

>>> a

['foo', 'bar', 10, 'qux', 'quux', 20]

you can’t do this with a string:

>>> s = 'foobarbaz'

>>> s[2] = 'x'

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'str' object does not support item assignment

#### **Modifying Multiple List Values**

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[1:4]

['bar', 'baz', 'qux']

>>> a[1:4] = [1.1, 2.2, 3.3, 4.4, 5.5]

>>> a

['foo', 1.1, 2.2, 3.3, 4.4, 5.5, 'quux', 'corge']

>>> a[1:6]

[1.1, 2.2, 3.3, 4.4, 5.5]

>>> a[1:6] = ['Bark!']

>>> a

['foo', 'Bark!', 'quux', 'corge']

You can insert multiple elements in place of a single element—just use a slice that denotes only one element:

>>> a = [1, 2, 3]

>>> a[1:2] = [2.1, 2.2, 2.3]

>>> a

[1, 2.1, 2.2, 2.3, 3]

You can also insert elements into a list without removing anything. Simply specify a slice of the form [n:n] (a zero-length slice) at the desired index:

>>> a = [1, 2, 7, 8]

>>> a[2:2] = [3, 4, 5, 6]

>>> a

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

Additional items can be added to the start or end of a list using the + concatenation operator or the += [augmented assignment operator](https://realpython.com/python-assignment-operator/#augmented-assignment-operators-in-python):

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a += ['grault', 'garply']

>>> a

['foo', 'bar', 'baz', 'qux', 'quux', 'corge', 'grault', 'garply']

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a = [10, 20] + a

>>> a

[10, 20, 'foo', 'bar', 'baz', 'qux', 'quux', 'corge']

[a.append(<obj>)](https://realpython.com/python-append/) appends object <obj> to the end of list a:

>>> a = ['a', 'b']

>>> a.append(123)

>>> a

['a', 'b', 123]

Remember, list methods modify the target list in place. They do not return a new list:

>>> a = ['a', 'b']

>>> x = a.append(123)

>>> print(x)

None

>>> a

['a', 'b', 123]

The .append() method does not work that way! If an iterable is appended to a list with .append(), it is added as a single object:

>>> a = ['a', 'b']

>>> a.append([1, 2, 3])

>>> a

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

a.extend(<iterable>)

Yes, this is probably what you think it is. .extend() also adds to the end of a list, but the argument is expected to be an iterable. The items in <iterable> are added individually:

>>> a = ['a', 'b']

>>> a.extend([1, 2, 3])

>>> a

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

>>> a = ['a', 'b']

>>> a += [1, 2, 3]

>>> a

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

a.insert(<index>, <obj>)

Inserts an object into a list.

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a.insert(3, 3.14159)

>>> a[3]

3.14159

>>> a

['foo', 'bar', 'baz', 3.14159, 'qux', 'quux', 'corge']

a.remove(<obj>)

Removes an object from a list.

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a.remove('baz')

>>> a

['foo', 'bar', 'qux', 'quux', 'corge']

>>> a.remove('Bark!')

Traceback (most recent call last):

File "<pyshell#13>", line 1, in <module>

a.remove('Bark!')

ValueError: list.remove(x): x not in list

a.pop(index=-1)

Removes an element from a list.

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a.pop()

'corge'

>>> a

['foo', 'bar', 'baz', 'qux', 'quux']

>>> a.pop()

'quux'

>>> a

['foo', 'bar', 'baz', 'qux']

### **Lists Are Dynamic**

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[2:2] = [1, 2, 3]

>>> a += [3.14159]

>>> a

['foo', 'bar', 1, 2, 3, 'baz', 'qux', 'quux', 'corge', 3.14159]

>>> a = ['foo', 'bar', 'baz', 'qux', 'quux', 'corge']

>>> a[2:3] = []

>>> del a[0]

>>> a

['bar', 'qux', 'quux', 'corge']