# Python Lists

Lists are one of the 4 data types in Python used to store collections of data.

```
['John', 'Peter', 'Debora', 'Charles']
```

## Getting values with indexes

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> furniture[0]
# 'table'

>>> furniture[1]
# 'chair'

>>> furniture[2]
# 'rack'

>>> furniture[3]
# 'shelf'
```

## Negative indexes

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> furniture[-1]
# 'shelf'

>>> furniture[-3]
# 'chair'

>>> f'The {furniture[-1]} is bigger than the {furniture[-3]}'
# 'The shelf is bigger than the chair'
```

## Getting sublists with Slices

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> furniture[0:4]
# ['table', 'chair', 'rack', 'shelf']

>>> furniture[1:3]
# ['chair', 'rack']

>>> furniture[0:-1]
# ['table', 'chair', 'rack']

>>> furniture[:2]
```

```
# ['table', 'chair']

>>> furniture[1:]
# ['chair', 'rack', 'shelf']

>>> furniture[:]
# ['table', 'chair', 'rack', 'shelf']
```

Slicing the complete list will perform a copy:

```
>>> spam2 = spam[:]
# ['cat', 'bat', 'rat', 'elephant']

>>> spam.append('dog')
>>> spam
# ['cat', 'bat', 'rat', 'elephant', 'dog']

>>> spam2
# ['cat', 'bat', 'rat', 'elephant']
```

## Getting a list length with len()

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> len(furniture)
# 4
```

## Changing values with indexes

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> furniture[0] = 'desk'
>>> furniture
# ['desk', 'chair', 'rack', 'shelf']

>>> furniture[2] = furniture[1]
>>> furniture
# ['desk', 'chair', 'chair', 'shelf']

>>> furniture[-1] = 'bed'
>>> furniture
# ['desk', 'chair', 'chair', 'bed']
```

## Concatenation and Replication

```
>>> [1, 2, 3] + ['A', 'B', 'C']
# [1, 2, 3, 'A', 'B', 'C']

>>> ['X', 'Y', 'Z'] * 3
# ['X', 'Y', 'Z', 'X', 'Y', 'Z', 'X', 'Y', 'Z']
```

```
>>> my_list = [1, 2, 3]
>>> my_list = my_list + ['A', 'B', 'C']
>>> my_list
# [1, 2, 3, 'A', 'B', 'C']
```

## Using for loops with Lists

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> for item in furniture:
...     print(item)
# table
# chair
# rack
# shelf
```

## Getting the index in a loop with enumerate()

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']

>>> for index, item in enumerate(furniture):
...     print(f'index: {index} - item: {item}')
# index: 0 - item: table
# index: 1 - item: chair
# index: 2 - item: rack
# index: 3 - item: shelf
```

## Loop in Multiple Lists with zip()

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> price = [100, 50, 80, 40]

>>> for item, amount in zip(furniture, price):
...     print(f'The {item} costs ${amount}')
# The table costs $100
# The chair costs $50
# The rack costs $80
# The shelf costs $40
```

## The in and not in operators

```
>>> 'rack' in ['table', 'chair', 'rack', 'shelf']
# True

>>> 'bed' in ['table', 'chair', 'rack', 'shelf']
# False

>>> 'bed' not in furniture
# True
```

```
>>> 'rack' not in furniture
# False
```

## The Multiple Assignment Trick

The multiple assignment trick is a shortcut that lets you assign multiple variables with the values in a list in one line of code. So instead of doing this:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> table = furniture[0]
>>> chair = furniture[1]
>>> rack = furniture[2]
>>> shelf = furniture[3]
```

You could type this line of code:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> table, chair, rack, shelf = furniture

>>> table
# 'table'

>>> chair
# 'chair'

>>> rack
# 'rack'

>>> shelf
# 'shelf'
```

The multiple assignment trick can also be used to swap the values in two variables:

```
>>> a, b = 'table', 'chair'
>>> a, b = b, a
>>> print(a)
# chair

>>> print(b)
# table
```

## The index Method

The `index` method allows you to find the index of a value by passing its name:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> furniture.index('chair')
# 1
```

## Adding Values

### append()

`append` adds an element to the end of a `list`:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> furniture.append('bed')
>>> furniture
# ['table', 'chair', 'rack', 'shelf', 'bed']
```

### insert()

`insert` adds an element to a `list` at a given position:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> furniture.insert(1, 'bed')
>>> furniture
# ['table', 'bed', 'chair', 'rack', 'shelf']
```

## Removing Values

### del()

`del` removes an item using the index:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> del furniture[2]
>>> furniture
# ['table', 'chair', 'shelf']

>>> del furniture[2]
>>> furniture
# ['table', 'chair']
```

### remove()

`remove` removes an item with using actual value of it:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> furniture.remove('chair')
>>> furniture
# ['table', 'rack', 'shelf']
```

### pop()

By default, `pop` will remove and return the last item of the list. You can also pass the index of the element as an optional parameter:

```
>>> animals = ['cat', 'bat', 'rat', 'elephant']

>>> animals.pop()
'elephant'

>>> animals
```

```
['cat', 'bat', 'rat']

>>> animals.pop(0)
'cat'

>>> animals
['bat', 'rat']
```

## Sorting values with sort()

```
>>> numbers = [2, 5, 3.14, 1, -7]
>>> numbers.sort()
>>> numbers
# [-7, 1, 2, 3.14, 5]

furniture = ['table', 'chair', 'rack', 'shelf']
furniture.sort()
furniture
# ['chair', 'rack', 'shelf', 'table']
```

You can also pass `True` for the `reverse` keyword argument to have `sort()` sort the values in reverse order:

```
>>> furniture.sort(reverse=True)
>>> furniture
# ['table', 'shelf', 'rack', 'chair']
```

If you need to sort the values in regular alphabetical order, pass `str.lower` for the key keyword argument in the sort() method call:

```
>>> letters = ['a', 'z', 'A', 'Z']
>>> letters.sort(key=str.lower)
>>> letters
# ['a', 'A', 'z', 'Z']
```

You can use the built-in function `sorted` to return a new list:

```
>>> furniture = ['table', 'chair', 'rack', 'shelf']
>>> sorted(furniture)
# ['chair', 'rack', 'shelf', 'table']
```

## The Tuple data type

```
>>> furniture = ('table', 'chair', 'rack', 'shelf')

>>> furniture[0]
# 'table'

>>> furniture[1:3]
# ('chair', 'rack')
```

```
>>> len(furniture)
# 4
```

The main way that tuples are different from lists is that tuples, like strings, are immutable.

## Converting between list() and tuple()

```
>>> tuple(['cat', 'dog', 5])
# ('cat', 'dog', 5)

>>> list(('cat', 'dog', 5))
# ['cat', 'dog', 5]

>>> list('hello')
# ['h', 'e', 'l', 'l', 'o']
```