Data Types - Lists, Tuples and Sets

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are Tuple, Set, and Dictionary, all with different qualities and usage.

Lists are created using square brackets:

```
thislist = ['apple', 'banana', 'cherry']
```

```
type(thislist)
```

```
list
```

```
Start coding or generate with AI.
```

List Items

List items are ordered, changeable, and allow duplicate values.

List items are indexed, the first item has index [0], the second item has index [1] etc.

Since lists are indexed, lists can have items with the same value:

```
fruits = ['apple', 'mango', 'pawpaw', 'papaya', 'apple']
```

```
fruits[-1:-3]
```

```
[]
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
fruits[0]
```

```
'apple'
```

List Length

To determine how many items a list has, use the len() function:

```
len(fruits)
```

```
5
```

```
Start coding or generate with AI.
```

List Items - Data Types

List items can be of any data type:

```
num = [1, 2, 3, 4]
num
```

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

```
list1 = ['Abc', 1, 15, True, False]
list1
```

```
['Abc', 1, 15, True, False]
```

```
type(list1)
```

```
list
```

```
Start coding or generate with AI.
```

type()

From Python's perspective, lists are defined as objects with the data type 'list'

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Using the list() constructor to make a List:

```
x = (1, 2.4, 4, 6)
x
```

```
(1, 2.4, 4, 6)
```

```
type(x)
```

```
tuple
```

```
y = list(x)
y
```

```
[1, 2.4, 4, 6]
```

Access Items

```
fruits
```

```
['Blackcurrant', 'Carrot', 'mango', 'pawpaw', 'papaya', 'watermelon']
```

```
fruits[2]
```

```
'mango'
```

```
Start coding or generate with AI.
```

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

```
fruits[-1]
```

```
'watermelon'
```

```
fruits[-3]
```

```
'pawpaw'
```

```
Start coding or generate with AI.
```

Range of Indexes

You can specify a range of indexes by specifying where to start and where to end the range.

When specifying a range, the return value will be a new list with the specified items.

```
fruits[0:3]
```

```
['Blackcurrant', 'Carrot', 'mango']
```

```
fruits
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

```
fruits[-3 : ]
```

```
['pawpaw', 'pineapple', 'watermelon']
```

Check if Item Exists

To determine if a specified item is present in a list use the in keyword:

```
'egg' in fruits
```

```
False
```

```
'apple' in fruits
```

```
True
```

Change Item Value

To change the value of a specific item, refer to the index number

```
fruits
```

```
['apple', 'mango', 'pawpaw', 'papaya', 'apple']
```

```
fruits[0] = 'cherry'
```

```
fruits
```

```
['cherry', 'mango', 'pawpaw', 'papaya', 'apple']
```

```
fruits[-1] = 'watermelon'
```

```
fruits
```

```
['cherry', 'mango', 'pawpaw', 'papaya', 'watermelon']
```

Change a Range of Item Values

To change the value of items within a specific range, define a list with the new values, and refer to the range of index numbers where you want to insert the new values:

```
fruits
```

```
['cherry', 'mango', 'pawpaw', 'papaya', 'watermelon']
```

```
fruits[0:2] = ['Blackcurrant', 'Carrot']
```

```
fruits
```

```
['Blackcurrant', 'Carrot', 'mango', 'pawpaw', 'papaya', 'watermelon']
```

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

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Insert Items

To insert a new list item, without replacing any of the existing values, we can use the insert() method.

The insert() method inserts an item at the specified index

```
thislist
```

```
['apple', 'banana', 'cherry']
```

```
thislist.insert(1, 'pineapple')
```

```
thislist
```

```
['apple', 'pineapple', 'banana', 'cherry']
```

```
Start coding or generate with AI.
```

Append Items

To add an item to the end of the list, use the append() method

```
thislist.append('Carrot')
```

```
thislist
```

```
['apple', 'pineapple', 'banana', 'cherry', 'Carrot']
```

```
thislist.append('mango')
```

```
thislist
```

```
['apple', 'pineapple', 'banana', 'cherry', 'Carrot', 'mango']
```

Extend List

To append elements from another list to the current list, use the extend() method.

```
thislist
```

```
['apple', 'pineapple', 'banana', 'cherry', 'Carrot', 'mango']
```

```
fruits
```

```
['Blackcurrant', 'Carrot', 'mango', 'pawpaw', 'papaya', 'watermelon']
```

```
new_list = thislist + fruits
```

```
new_list
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'Carrot',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

```
thislist.extend(fruits)
```

```
thislist
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'Carrot',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

```
Start coding or generate with AI.
```

Add Any Iterable

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

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Remove Specified Item

The remove() method removes the specified item.

```
new_list
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'Carrot',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

```
new_list.remove('Carrot')
```

```
new_list
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

Remove Specified Index

The pop() method removes the specified index.

```
new_list.pop(0)
```

```
'apple'
```

```
new_list
```

```
['pineapple',
 'banana',
 'cherry',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

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

```
new_list.pop()
```

```
'watermelon'
```

```
new_list
```

```
['pineapple',
 'banana',
 'cherry',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya']
```

The `del` keyword also removes the specified index:

```
new_list
```

```
['pineapple',
 'banana',
 'cherry',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya']
```

```
del new_list[2]
```

```
new_list
```

```
['pineapple',
 'banana',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya']
```

The `del` keyword can also delete the list completely.

```
del new_list
```

```
new_list
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[140], line 1
----> 1 new_list

NameError: name 'new_list' is not defined
```

```
Start coding or generate with AI.
```

Clear the List

The clear() method empties the list.

The list still remains, but it has no content.

```
fruits
```

```
['Blackcurrant', 'Carrot', 'mango', 'pawpaw', 'papaya', 'watermelon']
```

```
fruits.clear()
```

```
fruits
```

```
[]
```

Loop Through a List

You can loop through the list items by using a for loop:

```
thislist
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'Carrot',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

```
fruits = thislist
```

```
fruits
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

```
for fruit in fruits:
    print(fruit)
```

```
Blackcurrant
Carrot
Carrot
apple
banana
```

```
cherry
mango
mango
papaya
pawpaw
pineapple
watermelon
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```python
for x in thislist:
    print(x)
```

```
apple
pineapple
banana
cherry
Carrot
mango
Blackcurrant
Carrot
mango
pawpaw
papaya
watermelon
```

Loop Through the Index Numbers

You can also loop through the list items by referring to their index number.

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

```python
for x in range(len(thislist)):
    print(x)
```

```
0
1
2
3
4
5
6
7
8
9
10
11
```

```python
len(thislist)
```

```
12
```

Using a While Loop

You can loop through the list items by using a while loop.

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.

```python
thislist
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'Carrot',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

```
i = 0
while i < len(thislist):
    print(thislist[i])
    i = i+1
```

```
apple
pineapple
banana
cherry
Carrot
mango
Blackcurrant
Carrot
mango
pawpaw
papaya
watermelon
```

```
Start coding or generate with AI.
```

Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default

```
thislist
```

```
['apple',
 'pineapple',
 'banana',
 'cherry',
 'Carrot',
 'mango',
 'Blackcurrant',
 'Carrot',
 'mango',
 'pawpaw',
 'papaya',
 'watermelon']
```

```
thislist.sort()
```

```
thislist
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

```
num = [1, 23, 4, 18, 0.5]
```

```
num.sort()
```

```
num
```

```
[0.5, 1, 4, 18, 23]
```

Sort Descending

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

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

```
num
```

```
[23, 18, 4, 1, 0.5]
```

```
Start coding or generate with AI.
```

Case Insensitive Sort

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

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

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().

```
thislist
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

```
y = thislist
```

```
y
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

```
z = thislist.copy()
```

```
z
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Set, and Dictionary, all with different qualities and usage.

A tuple is a collection which is ordered and unchangeable.

Tuples are written with round brackets.

```
fruit = ['cherry']
fruit
```

```
['cherry']
```

```
fruit = ('cherry',)
```

```
fruit
```

```
('cherry',)
```

Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

Allow Duplicates

Since tuples are indexed, they can have items with the same value

```
Start coding or generate with AI.
```

Create Tuple With One Item

To create a tuple with only one item, you have to add a comma after the item, otherwise Python will not recognize it as a tuple.

```
tuple1 = ('a', 'b', 'c')
```

```
tuple1.append(4)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[194], line 1
----> 1 tuple1.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```

```
Start coding or generate with AI.
```

Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are unchangeable, or immutable as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

```
tuple1
```

```
('a', 'b', 'c')
```

```
x = list(tuple1)
x
```

```
['a', 'b', 'c']
```

```
x.append('d')
```

```
x
```

```
['a', 'b', 'c', 'd']
```

```
tuple1 = tuple(x)
```

```
tuple1
```

```
('a', 'b', 'c', 'd')
```

Add Items

Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple.

1. Convert into a list: Just like the workaround for changing a tuple, you can convert it into a list, add your item(s), and convert it back into a tuple.

```
Start coding or generate with AI.
```

2. Add tuple to a tuple. You are allowed to add tuples to tuples, so if you want to add one item, (or many), create a new tuple with the item(s), and add it to the existing tuple:

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Remove Items

Tuples are unchangeable, so you cannot remove items from it, but you can use the same workaround as we used for changing and adding tuple items:

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

Or you can delete the tuple completely

The del keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")

thistuple
```

```
('apple', 'banana', 'cherry')
```

```
del thistuple
```

```
thistuple
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[212], line 1
----> 1 thistuple

NameError: name 'thistuple' is not defined
```

Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are List, Tuple, and Dictionary, all with different qualities and usage.

A set is a collection which is unordered, unchangeable*, and unindexed.

```
# Sets are written with curly brackets.
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

Duplicates Not Allowed

Sets cannot have two items with the same value.

```
thisset = {"apple", "banana", "cherry", "apple"}

print(thisset)
```

```
{'apple', 'cherry', 'banana'}
```

True and 1 are considered the same value in sets, and are treated as duplicates

```
thisset = {"apple", "banana", "cherry", True, 1, 2}

print(thisset)
```

```
{'apple', True, 2, 'cherry', 'banana'}
```

False and 0 are considered the same value in sets, and are treated as duplicates

```
thisset = {"apple", "banana", "cherry",0, False, True, 0}

print(thisset)
```

```
{'apple', 0, True, 'cherry', 'banana'}
```

Get the Length of a Set

To determine how many items a set has, use the len() function.

```
len(thisset)
```

```
5
```

A set can contain the same or different data types

```
set1 = {"apple", "banana", "cherry"}
```

```
set2 = {"abc", 34, True, 40, "male"}
```

```
set2
```

```
{34, 40, True, 'abc', 'male'}
```

```
Start coding or generate with AI.
```

```
Start coding or generate with AI.
```

```
thislist
```

```
['Blackcurrant',
 'Carrot',
 'Carrot',
 'apple',
 'banana',
 'cherry',
 'mango',
 'mango',
 'papaya',
 'pawpaw',
 'pineapple',
 'watermelon']
```

```
thislist.sort()
```

```
import keyword
```

```
keyword.kwlist
```

```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

```
Start coding or generate with AI.
```

```
x = [1, 2, 3, 4, 6, 7, 8, -3]
```

```
x[0:3] + x[5:7]
```

```
[1, 2, 3, 7, 8]
```

```
x[::2]
```

```
[1, 3, 6, 8]
```

```
Start coding or generate with AI.
```