

Python Collections (Arrays)

There are 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 and unindexed. No duplicate members.
- **Dictionary** is a collection which is unordered, changeable and indexed. No duplicate members.

When choosing a collection type, it is useful to understand the properties of that type. Choosing the right type for a particular data set could mean retention of meaning, and, it could mean an increase in efficiency or security.

List

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example

Create a List:

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

Access Items

You access the list items by referring to the index number:

Example

Print the second item of the list:

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

Negative Indexing

Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

Example

Print the last item of the list:

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

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.

Example

Return the third, fourth, and fifth item:

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

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

By leaving out the start value, the range will start at the first item:

Example

This example returns the items from the beginning to "orange":

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

By leaving out the end value, the range will go on to the end of the list:

Example

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

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

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the list:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

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

Change Item Value

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

Example

Change the second item:

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

Loop Through a List

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

Example

Print all items in the list, one by one:

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

Check if Item Exists

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

Example

Check if "apple" is present in the list:

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

List Length

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

Example

Print the number of items in the list:

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

Add Items

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

Example

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

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

To add an item at the specified index, use the `insert()` method:

Example

Insert an item as the second position:

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

Remove Item

There are several methods to remove items from a list:

Example

The `remove()` method removes the specified item:

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

Example

The `pop()` method removes the specified index, (or the last item if index is not specified):

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

The `del` keyword removes the specified index:

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

Example

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

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

Example

The `clear()` method empties the list:

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

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()`.

Example

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

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

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

Example

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

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

Join Two Lists

There are several ways to join, or concatenate, two or more lists in Python.

One of the easiest ways are by using the `+` operator.

Example

Join two list:

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

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

Another way to join two lists are by appending all the items from list2 into list1, one by one:

Example

Append list2 into list1:

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

for x in list2:
    list1.append(x)

print(list1)
```

Or you can use the `extend()` method, which purpose is to add elements from one list to another list:

Example

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)
```

The list() Constructor

It is also possible to use the `list()` constructor to make a new list.

Example

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

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

List Methods

Method	Description
<code>append()</code>	Adds an element at the end of the list
<code>clear()</code>	Removes all the elements from the list

<code>copy()</code>	Returns a copy of the list
---------------------	----------------------------

<code>count()</code>	Returns the number of elements with the specified value
----------------------	---

<code>extend()</code>	Add the elements of a list (or any iterable), to the end of the current list
-----------------------	--

<code>index()</code>	Returns the index of the first element with the specified value
----------------------	---

<code>insert()</code>	Adds an element at the specified position
-----------------------	---

<code>pop()</code>	Removes the element at the specified position
--------------------	---

<code>remove()</code>	Removes the item with the specified value
-----------------------	---

<code>reverse()</code>	Reverses the order of the list
------------------------	--------------------------------

<code>sort()</code>	Sorts the list
---------------------	----------------

Python has a set of built-in methods that you can use on lists.

Python Tuples

Tuple

A tuple is a collection which is ordered and **unchangeable**. In Python tuples are written with round brackets.

Example

Create a Tuple:

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

Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

Example

Print the second item in the tuple:

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

Negative Indexing

Negative indexing means beginning from the end, `-1` refers to the last item, `-2` refers to the second last item etc.

Example

Print the last item of the tuple:

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

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 tuple with the specified items.

Example

Return the third, fourth, and fifth item:

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

Note: The search will start at index 2 (included) and end at index 5 (not included).

Remember that the first item has index 0.

Range of Negative Indexes

Specify negative indexes if you want to start the search from the end of the tuple:

Example

This example returns the items from index -4 (included) to index -1 (excluded)

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

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.

Example

Convert the tuple into a list to be able to change it:

```
x = ("apple", "banana", "cherry")
y = list(x)
y[1] = "kiwi"
x = tuple(y)

print(x)
```

Loop Through a Tuple

You can loop through the tuple items by using a `for` loop.

Example

Iterate through the items and print the values:

```
thistuple = ("apple", "banana", "cherry")
for x in thistuple:
    print(x)
```

Check if Item Exists

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

Example

Check if "apple" is present in the tuple:

```
thistuple = ("apple", "banana", "cherry")
if "apple" in thistuple:
    print("Yes, 'apple' is in the fruits tuple")
```

Tuple Length

To determine how many items a tuple has, use the `len()` method:

Example

Print the number of items in the tuple:

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

Add Items

Once a tuple is created, you cannot add items to it. Tuples are **unchangeable**.

Example

You cannot add items to a tuple:

```
thistuple = ("apple", "banana", "cherry")
thistuple[3] = "orange" # This will raise an error
print(thistuple)
```

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.

Example

One item tuple, remember the comma:

```
thistuple = ("apple",)
print(type(thistuple))

#NOT a tuple
thistuple = ("apple")
print(type(thistuple))
```

Remove Items

Note: You cannot remove items in a tuple.

Tuples are **unchangeable**, so you cannot remove items from it, but you can delete the tuple completely:

Example

The `del` keyword can delete the tuple completely:

```
thistuple = ("apple", "banana", "cherry")
del thistuple
print(thistuple) #this will raise an error because the tuple no longer exists
```

Join Two Tuples

To join two or more tuples you can use the `+` operator:

Example

Join two tuples:

```
tuple1 = ("a", "b" , "c")
tuple2 = (1, 2, 3)

tuple3 = tuple1 + tuple2
print(tuple3)
```

The tuple() Constructor

It is also possible to use the `tuple()` constructor to make a tuple.

Example

Using the `tuple()` method to make a tuple:

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

Tuple Methods

Python has two built-in methods that you can use on tuples.

Method	Description
<code>count()</code>	Returns the number of times a specified value occurs in a tuple
<code>index()</code>	Searches the tuple for a specified value and returns the position of where it was found

Python Sets

Set

A set is a collection which is unordered and unindexed. In Python, sets are written with curly brackets.

Example

Create a Set:

```
thisset = {"apple", "banana", "cherry"}  
print(thisset)
```

Note: Sets are unordered, so you cannot be sure in which order the items will appear.

Access Items

You cannot access items in a set by referring to an index or a key.

But you can loop through the set items using a `for` loop, or ask if a specified value is present in a set, by using the `in` keyword.

Example

Loop through the set, and print the values:

```
thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
    print(x)
```

Change Items

Once a set is created, you cannot change its items, but you can add new items.

Add Items

To add one item to a set use the `add()` method.

To add more than one item to a set use the `update()` method.

Example

Add an item to a set, using the `add()` method:

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)
```

Example

Add multiple items to a set, using the `update()` method:

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.update(["orange", "mango", "grapes"])  
  
print(thisset)
```

Get the Length of a Set

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

Example

Get the number of items in a set:

```
thisset = {"apple", "banana", "cherry"}  
  
print(len(thisset))
```

Remove Item

To remove an item in a set, use the `remove()`, or the `discard()` method.

Example

Remove "banana" by using the `remove()` method:

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)
```

Note: If the item to remove does not exist, `remove()` will raise an error.

Example

Remove "banana" by using the `discard()` method:

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.discard("banana")  
  
print(thisset)
```

Note: If the item to remove does not exist, `discard()` will **NOT** raise an error.

You can also use the `pop()` method to remove an item, but this method will remove the *last* item. Remember that sets are unordered, so you will not know what item that gets removed.

The return value of the `pop()` method is the removed item.

Example

Remove the last item by using the `pop()` method:

```
thisset = {"apple", "banana", "cherry"}  
  
x = thisset.pop()  
  
print(x)  
  
print(thisset)
```

Note: Sets are *unordered*, so when using the `pop()` method, you will not know which item that gets removed.

Example

The `clear()` method empties the set:

```
thisset = {"apple", "banana", "cherry"}  
  
thisset.clear()  
  
print(thisset)
```

Example

The `del` keyword will delete the set completely:

```
thisset = {"apple", "banana", "cherry"}  
  
del thisset  
  
print(thisset)
```

Join Two Sets

There are several ways to join two or more sets in Python.

You can use the `union()` method that returns a new set containing all items from both sets, or the `update()` method that inserts all the items from one set into another:

Example

The `union()` method returns a new set with all items from both sets:

```
set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)
```

Example

The `update()` method inserts the items in set2 into set1:

```
set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}
```

```
set1.update(set2)  
print(set1)
```

Note: Both `union()` and `update()` will exclude any duplicate items.

There are other methods that joins two sets and keeps ONLY the duplicates, or NEVER the duplicates, check the full list of set methods in the bottom of this page.

The set() Constructor

It is also possible to use the `set()` constructor to make a set.

Example

Using the `set()` constructor to make a set:

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

Set Methods

Python has a set of built-in methods that you can use on sets.

Method	Description
<u>add()</u>	Adds an element to the set
<u>clear()</u>	Removes all the elements from the set
<u>copy()</u>	Returns a copy of the set
<u>difference()</u>	Returns a set containing the difference between two or more sets
<u>difference_update()</u>	Removes the items in this set that are also included in another, specified set
<u>discard()</u>	Remove the specified item
<u>intersection()</u>	Returns a set, that is the intersection of two other sets
<u>intersection_update()</u>	Removes the items in this set that are not present in other, specified set(s)

<u>isdisjoint()</u>	Returns whether two sets have a intersection or not
<u>issubset()</u>	Returns whether another set contains this set or not
<u>issuperset()</u>	Returns whether this set contains another set or not
<u>pop()</u>	Removes an element from the set
<u>remove()</u>	Removes the specified element
<u>symmetric_difference()</u>	Returns a set with the symmetric differences of two sets
<u>symmetric_difference_update()</u>	inserts the symmetric differences from this set and another
<u>union()</u>	Return a set containing the union of sets
<u>update()</u>	Update the set with the union of this set and others

Python Dictionaries

Dictionary

A dictionary is a collection which is unordered, changeable and indexed. In Python dictionaries are written with curly brackets, and they have keys and values.

Example

Create and print a dictionary:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
print(thisdict)
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets:

Example

Get the value of the "model" key:

```
x = thisdict["model"]
```

There is also a method called `get()` that will give you the same result:

Example

Get the value of the "model" key:

```
x = thisdict.get("model")
```

Change Values

You can change the value of a specific item by referring to its key name:

Example

Change the "year" to 2018:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["year"] = 2018
```

Loop Through a Dictionary

You can loop through a dictionary by using a `for` loop.

When looping through a dictionary, the return value are the *keys* of the dictionary, but there are methods to return the *values* as well.

Example

Print all key names in the dictionary, one by one:

```
for x in thisdict:  
    print(x)
```

Example

Print all *values* in the dictionary, one by one:

```
for x in thisdict:  
    print(thisdict[x])
```

Example

You can also use the `values()` method to return values of a dictionary:

```
for x in thisdict.values():  
    print(x)
```


Example

Loop through both *keys* and *values*, by using the `items()` method:

```
for x, y in thisdict.items():  
    print(x, y)
```

Check if Key Exists

To determine if a specified key is present in a dictionary use the `in` keyword:

Example

Check if "model" is present in the dictionary:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
if "model" in thisdict:  
    print("Yes, 'model' is one of the keys in the thisdict dictionary")
```

Dictionary Length

To determine how many items (key-value pairs) a dictionary has, use the `len()` function.

Example

Print the number of items in the dictionary:

```
print(len(thisdict))
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

Example

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)
```

Removing Items

There are several methods to remove items from a dictionary:

Example

The `pop()` method removes the item with the specified key name:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)
```

Example

The `popitem()` method removes the last inserted item (in versions before 3.7, a random item is removed instead):

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.popitem()  
print(thisdict)
```

Example

The `del` keyword removes the item with the specified key name:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
del thisdict["model"]  
print(thisdict)
```

Example

The `del` keyword can also delete the dictionary completely:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer  
exists.
```

Example

The `clear()` method empties the dictionary:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
thisdict.clear()  
print(thisdict)
```

Copy a Dictionary

You cannot copy a dictionary simply by typing `dict2 = dict1`, because: `dict2` will only be a *reference* to `dict1`, and changes made in `dict1` will automatically also be made in `dict2`.

There are ways to make a copy, one way is to use the built-in Dictionary method `copy()`.

Example

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

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)
```

Another way to make a copy is to use the built-in function `dict()`.

Example

Make a copy of a dictionary with the `dict()` function:

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)
```

Nested Dictionaries

A dictionary can also contain many dictionaries, this is called nested dictionaries.

Example

Create a dictionary that contain three dictionaries:

```
myfamily = {  
    "child1" : {  
        "name" : "Emil",  
        "year" : 2004  
    },  
    "child2" : {
```

```

        "name" : "Tobias",
        "year" : 2007
    },
    "child3" : {
        "name" : "Linus",
        "year" : 2011
    }
}

```

Or, if you want to nest three dictionaries that already exists as dictionaries:

Example

Create three dictionaries, then create one dictionary that will contain the other three dictionaries:

```

child1 = {
    "name" : "Emil",
    "year" : 2004
}
child2 = {
    "name" : "Tobias",
    "year" : 2007
}
child3 = {
    "name" : "Linus",
    "year" : 2011
}

myfamily = {
    "child1" : child1,
    "child2" : child2,
    "child3" : child3
}

```

The dict() Constructor

It is also possible to use the `dict()` constructor to make a new dictionary:

Example

```

thisdict = dict(brand="Ford", model="Mustang", year=1964)
# note that keywords are not string literals
# note the use of equals rather than colon for the assignment
print(thisdict)

```

Dictionary Methods

Python has a set of built-in methods that you can use on dictionaries.

Method	Description
<code>clear()</code>	Removes all the elements from the dictionary
<code>copy()</code>	Returns a copy of the dictionary
<code>fromkeys()</code>	Returns a dictionary with the specified keys and value
<code>get()</code>	Returns the value of the specified key
<code>items()</code>	Returns a list containing a tuple for each key value pair
<code>keys()</code>	Returns a list containing the dictionary's keys
<code>pop()</code>	Removes the element with the specified key
<code>popitem()</code>	Removes the last inserted key-value pair

[setdefault\(\)](#) Returns the value of the specified key. If the key does not exist: insert the key, with the specified value

[update\(\)](#) Updates the dictionary with the specified key-value pairs

[values\(\)](#) Returns a list of all the values in the dictionary