





Part3



Mohammad Reza Gerami Mrgerami@aut.ac.ir gerami@virasec.ir



Python Collections (Arrays)

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.

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



Create a List:

```
list1 = ["apple", "banana", "cherry"]
print(list1)
```

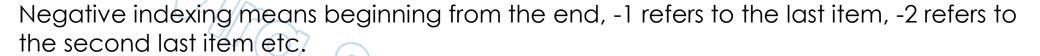
Access Items

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

Print the second item of the list:

```
print(list1[1])
```

Negative Indexing





Print the last item of the list:

list1 = ["apple", "banana", "cherry"]
print(list1[-1])

Security Solid

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.

Return the third, fourth, and fifth item:

```
list1 = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(list1[2:5])
        ['cherry', 'orange', 'kiwi']
```

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:

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

```
list1 = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(list1[:4])
    ['apple', 'banana', 'cherry', 'orange']
```

Range of Indexes

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

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

```
list1 = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"] print(list1[2:])
```

```
['cherry', 'orange', 'kiwi', 'melon', 'mango
```



Security Solidies

Range of Negative Indexes

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

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

```
list1 = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]
print(list1[-4:-1])
```

['orange', 'kiwi', 'melon']

Change Item Value

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

Change the second item:

```
list1 = ["apple", "banana", "cherry"]
list1[1] = "Elballo"
print(list1)
```

```
['apple', 'Elballo', 'cherry']
```



Loop Through a List

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

Print all items in the list, one by one:

```
list1 = ["apple","banana","cherry"]
for x in list1:
   print(x)
```

apple banana cherry



Check if Item Exists

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

Check if "apple" is present in the list:

```
list1 = ["apple","banana","cherry"]
if "apple" in list1:
   print("Yes, 'apple' is in the fruits list")
```

Yes, 'apple' is in the fruits list



List Length

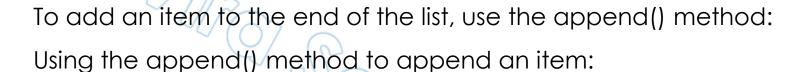
To determine how many items a list has, use the len() function:

Print the number of items in the list:

```
list1 = ["apple", "banana", "cherry"]
print(len(list1))
```



Add Items



```
list1 = ["apple", "banana", "cherry"]
list1.append("orange")
print(list1)
```

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

Insert an item as the second position:

```
list1 = ["apple", "banana", "cherry"]
list1.insert(1, "orange")
print(list1)
```





Remove Item

There are several methods to remove items from a list:

The remove() method removes the specified item:

```
list1 = ["apple", "banana", "cherry"]
list1.remove("banana")
print(list1)

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

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

```
list1 = ["apple", "banana", "cherry"]
list1.pop()
print(list1)
```

The del keyword removes the specified index:

```
list1 = ["apple", "banana", "cherry"]
del list1[0]
print(list1)
```

['banana', 'cherry']

apple', 'banana']

Remove Item



The del keyword can also delete the list completely:

```
list1 = ["apple", "banana", "cherry"]
del list1
```

The clear() method empties the list:

```
list1 = ["apple", "banana", "cherry"]
list1.clear()
print(list1)
```

Copy a List



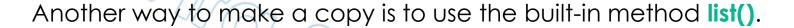
You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

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

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

```
list1 = ["apple", "banana", "cherry"]
mylist = list1.copy()
print(mylist)
```

Copy a List



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

```
list1 = ["apple", "banana", "cherry"]
mylist = list(list1)
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.

Join two list:

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

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



Join Two Lists



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

Append list2 into list1:

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

for x in list2:
    list1.append(x)

print(list1)
```

Join Two Lists



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

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)
```

List Methods

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



Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

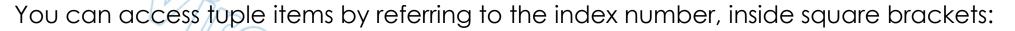
VIII Solution

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

Create a Tuple:

```
tuple1 = ("apple", "banana", "cherry")
print(tuple1)
```

Access Tuple Items





Print the second item in the tuple:

```
tuple1 = ("apple", "banana", "cherry")
print(tuple1[1])
```

Negative Indexing

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



Print the last item of the tuple:

```
tuple1 = ("apple", "banana", "cherry")
print(tuple1[-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.

Return the third, fourth, and fifth item:

```
tuple1 = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(tuple1[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:

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

```
tuple1 = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")
print(tuple1[-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.

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.

Iterate through the items and print the values:

```
tuple1 = ("apple", "banana", "cherry")
for x in tuple1:
   print(x)
```



Check if Item Exists

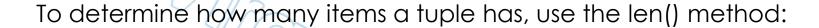




Check if "apple" is present in the tuple:

```
tuple1 = ("apple", "banana", "cherry")
if "apple" in tuple1:
   print("Yes, 'apple' is in the fruits tuple")
```

Tuple Length





```
tuple1 = ("apple", "banana", "cherry")
print(len(tuple1))
```



Add Items



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

You cannot add items to a tuple:

tuple1 = ("apple", "banana", "cherry")
tuple1[3] = "orange" # This will raise an error
print(tuple1)

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.

One item tuple, remember the comma:

```
tuple1 = ("apple",)
print(type(tuple1))

#NOT a tuple
tuple1 = ("apple")
print(type(tuple1))
```



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:

The del keyword can delete the tuple completely:

```
tuple1 = ("apple", "banana", "cherry")
del tuple1
print(tuple1) #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:

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

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



Tuples

The tuple() Constructor



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

Using the tuple() method to make a tuple:

```
tuple1 = tuple(("apple", "banana", "cherry")) # note the double round-brackets
print(tuple1)
```

Tuples

Security Sold

Tuple Methods

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

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



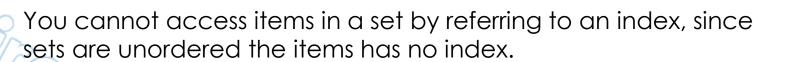
A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

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



Security Solution

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.

Loop through the set, and print the values:

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

for x in thisset: print(x)

Check if "banana" is present in the set:

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

Change Items

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



Add Items

Security Solit

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

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

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

Add Items



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.

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.

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.

Remove Item



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.

Remove Item



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.

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.

Remove Item

Charles Solution of the Courting Solution of t

The clear() method empties the set:

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

The del keyword will delete the set completely:

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

Join Two Sets

Security solding

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:

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)
```



Join Two Sets



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

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

The set() Constructor

Security Solitor

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

Using the set() constructor to make a set:

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

Set

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

Method	Description	1
add()	Adds an element to the set	
clear()	Removes all the elements from the set	
copy()	Returns a copy of the set	
difference()	Returns a set containing the difference between two or more sets	
difference_update()	Removes the items in this set that are also included in another, specified set	
discard()	Remove the specified item	
intersection()	Returns a set, that is the intersection of two other sets	
intersection_update()	Removes the items in this set that are not present in other, specified set(s)	
isdisjoint()	Returns whether two sets have a intersection or not	
issubset()	Returns whether another set contains this set or not	
issuperset()	Returns whether this set contains another set or not	
pop()	Removes an element from the set	
remove()	Removes the specified element	
symmetric_difference()	Returns a set with the symmetric differences of two sets	
symmetric_difference_update()	inserts the symmetric differences from this set and another	
union()	Return a set containing the union of sets	
update()	Update the set with the union of this set and others	

Python Dictionaries

Dictionary



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:

Get the value of the "model" key:

x = thisdict["model"]

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

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:

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.

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

for x in thisdict:
 print(x)

Print all values in the dictionary, one by one:

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



Loop Through a Dictionary



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

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

Loop through both keys and values, by using the items() function:

```
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:

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() method.

Print the number of items in the dictionary:

print(len(thisdict))

Adding Items



```
Security Soliding
```

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

Removing Items

There are several methods to remove items from a dictionary:

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

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



Removing Items

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)
```

Removing Items

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

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



Removing Items

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.
```

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().

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

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

Copy a Dictionary

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

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

```
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. Create a dictionary that contain three dictionaries:

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

Nested Dictionaries

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

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

```
myfamily['child1']
myfamily['child1'].values()
myfamily['child1'].keys()
myfamily
```

```
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



```
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



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

Vira Security Solutions Academy - https://virasec.ir/academy - +989125792641



Visiting Address: No 53 Vafa Manesh Ave

Heravi, Pasdaran Ave, TEHRAN-IRAN

Tel No: 0098 21 22196115-09125792641

Email: info@ virasec.ir

Website: www.virasec.ir

آدرس: تهران، پاسداران، هروی، خیابان وفامنش، پلاک ۵۳ شماره تماس: ۰۲۱۲۲۱۹۶۱۱۵-۰۹۱۲۵۷۹۲۶۴۱