List, Tuples, Sets & Dictionaries

UNIT - III

Unit-3: List, Tuples, Sets & Dictionaries

CO3 - Implement data structures lists, tuples, sets and dictionaries to solve the given problems.

3.1 Lists and operations on Lists

3.2 Tuples and operations on Tuples

3.3 Sets and operations on Sets

3.4 Dictionaries and operations on Dictionaries

List

- Python does not have the concept of an Array, but there are certain types which supports the concept of Sequence, Examples are: List, Tuple, Set and Dictionary
- List is a type which is used to store the sequence of Non-Homogeneous (Heterogeneous) elements, Non-Homogeneous means you can store any type of data for each element.
- Index starting from 0.
- Lists are created using Square Brackets [ ]
- Example: Creation of List

```
list1 = [10, 20.5, 'a', 'abc']
```

- How to Print List

```
print(list1)
[10, 20.5, 'a', 'abc']
```

Example:

```
# empty list
my_list = [] OR  my_list = list()

# list with mixed data types
my_list = [1, "Hello", 3.4]

# nested list
my_list = ["mouse", [8, 4, 6], ['a']]
```

Characteristic of List

Iteratable	• List elements can be iterated one by one in a loop or a function where iteratable values are accepted.
Heterogeneous	• All the elements can be of any Type. • No restriction of storing elements of same types
Ordered / Indexed	• List items have a defined order, and that order will not change. • If you add new items to a list, the new items will be placed at the end of the list.
Mutable	• It can change, add, and remove items in a list after it has been created.
Allows Duplicates	• Since lists are indexed, lists can have items with the same value.

Access List Items

- List items are indexed and you can access them by referring to the index number:
- Example:

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

Output: banana

- Range of Indexes in List
- Example:

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

Output: ['cherry', 'orange', 'kiwi']

Check if Item Exists in List or Not

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

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

Output: Yes, 'apple' is in the fruits list

Change List Items

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

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

Output: ['apple', 'blackcurrant', 'cherry']

- 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:
- Example:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]
thislist[1:3] = ["blackcurrant", "watermelon"]
print(thislist)
```

Output: ['apple', 'blackcurrant', 'watermelon', 'orange', 'kiwi', 'mango']

Insert Items in List

- 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:
- Example:

```
thislist = ["apple", "banana", "cherry"]
thislist.insert(2, "watermelon")
print(thislist)
```

Output: ['apple', 'banana', 'watermelon', 'cherry']

Append Items in List

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

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

Output: ['apple', 'banana', 'cherry', 'orange']

Extend List

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

```
thislist = ["apple", "banana", "cherry"]
tropical = ["mango", "pineapple", "papaya"]
thislist.extend(tropical)
print(thislist)
```

Output: ['apple', 'banana', 'cherry', 'mango', 'pineapple', 'papaya']

Remove List Items

- Remove Specified Item

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

Output: ['apple', 'cherry']

- Remove Specified Index, If you do not specify the index, the pop() method removes the last item

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

Output: ['apple', 'cherry']

- The del keyword also removes the specified index:

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

Output: ['banana', 'cherry’]

- The del keyword can also delete the list completely.

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

Output: #this will cause an error because you have successfully deleted "thislist".

Clear List

- The clear() method empties the list, The list still remains, but it has no content.
- Example:

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

Output: [ ]

Repetition of List

- List Repetition can be done to repeat the List, for that * operator is used
- Example:

```
l1 = ["apple", "banana", "cherry"]
l2 = l1 * 2
print(l2)
Output: ["apple", "banana", "cherry", "apple", "banana", "cherry"]
```

Looping in List

- Loop Through a List (for loop)

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

Output:

```
apple
banana
cherry
```

- Loop Through a List (while loop)

```
thislist = ["apple", "banana", "cherry"]
i = 0
while i < len(thislist):
    print(thislist[i])
    i = i + 1
```

Output:

```
apple
banana
cherry
```

- Loop Through the Index Numbers

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

Output:

```
apple
banana
cherry
```

Sorting in List

- Sort List Alphanumerically

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort()
print(thislist)
```

Output:

```
['banana', 'kiwi', 'mango', 'orange', 'pineapple']
```

- Sort Descending

```
thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]
thislist.sort(reverse = True)
print(thislist)
```

Output:

```
['pineapple', 'orange', 'mango'
```

Reverse List

- Reverse Order

```
    thislist = ["banana", "Orange", "Kiwi",
"cherry"]
    thislist.reverse()
    print(thislist)
```

Output:

```
['cherry', 'Kiwi', 'Orange',
'banana']
```

Copy 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.
- Example:

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

Output:

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

Join List

- Join Two List

```
list1 = ["a", "b", "c"]
list2 = [1, 2, 3]
list3 = list1 + list2
print(list3)
```

Output:

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

- Join Two List Using extend Method

```
list1 = ["a", "b" , "c"]
list2 = [1, 2, 3]
list1.extend(list2)
print(list1)
```

Output:

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

List Methods (Summary)

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

Tuple

- Tuples are used to store multiple items in a single variable, so it is an another type of Sequence separated by commas.
- A tuple is a collection which is ordered and unchangeable.
- Tuples are written with round brackets ( )
- Tuple items are indexed, the first item has index [0], the second item has index [1] etc.
- Example:

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

Output: ('apple', 'banana', 'cherry')

```
# empty tuple
my_tuple = () OR my_tuple = tuple()
# tuple with mixed data types
my_tuple = (1, "Hello", 3.4)
# nested tuples
my_tuple = ("mouse", (8, 4, 6), ('a'))
```

Characteristics of Tuple

Iteratable	• Tuple elements can be iterated one by one in a loop or a function where iteratable values are accepted.
Heterogeneous	• All the elements can be of any Type. No restriction of storing elements of same types • No restriction of storing elements of same types
Ordered / Indexed	• Tuple items have a defined order, and that order will not change.
Immutable	• We cannot change, add, and remove items in a tuple after it has been created.
Allows Duplicates	• Since tuples are indexed, they can have items with the same value.

Length of Tuple

- To determine how many items a tuple has, use the len() function:
- Example:

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

Output: 3

Access Tuple Items

- You can access tuple items by referring to the index number, inside square brackets: thistuple = ("apple", "banana", "cherry")

```
print(thistuple[1])
```

Output: banana

- Range of Indexes in Tuples

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

Output: ('cherry', 'orange', 'kiwi')

Check if Item Exists in Tuples or Not

- To determine if a specified item is present in a Tuples use the in keyword:
- Example:

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

Output: Yes, 'apple' is in the fruits tuple

Update Tuples Values/Items

- 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

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

Output: ("apple", "kiwi", "cherry")

Insert Items in Tuples

- Since tuples are immutable, they do not have a built-in append() method, but there are other ways to add items to a tuple.
- Method 1: 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.

Example:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.append("orange")
thistuple = tuple(y)
print(thistuple)
```

Output: ('apple', 'banana', 'cherry', 'orange')

- Method 2: 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:

Example:

```
thistuple = ("apple", "banana", "cherry")
y = ("orange",)
thistuple += y
print(thistuple)
```

Output: ('apple', 'banana', 'cherry', 'orange')

Remove Tuple 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:
- Method 1: Convert the tuple into a list, remove "apple", and convert it back into a tuple:

```
thistuple = ("apple", "banana", "cherry")
y = list(thistuple)
y.remove("apple")
thistuple = tuple(y)
print(thistuple)
```

Output: ('banana', 'cherry')

- Method 2: you can delete the tuple completely using del keyword:

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

Repetition of Tuple

- Tuple Repetition can be done to repeat the tuple, for that * operator is used
- Example:

```
t1 = (4, 5, 6)
t2 = t1 * 2
print(t2)
Output: (4, 5, 6, 4, 5, 6)
```

Unpacking Tuple

- When we create a tuple, we normally assign values to it. This is called "packing" a tuple:
- Example: Packing a tuple:

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

- But, in Python, we are also allowed to extract the values back into variables. This is called "unpacking":
- Example: Unpacking a tuple:

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

```
Output: apple
        banana
        cherry
```

Looping in Tuples

- Loop Through a tuple (for loop)

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

Output:

```
apple
banana
cherry
```

- Loop Through a Tuple (while loop)

```
thistuple = ("apple", "banana", "cherry")
i = 0
while i < len(thistuple):
  print(thistuple[i])
  i = i + 1
```

Output:

```
apple
banana
```

- Loop Through the Index Numbers

```
thistuple = ("apple", "banana", "cherry")
for i in range(len(thistuple)):
  print(thistuple[i])
```

Output:

```
apple
banana
cherry
```

Join 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)
```

Output:

```
('a', 'b', 'c', 1, 2, 3)
```

Tuples Methods

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

Comparisons List v/s Tuple

- What are common characteristics?
  - Both store arbitrary data objects
  - Both are of sequence data type
- What are differences?
  - Tuple doesn't allow modification
  - Tuple doesn't have methods
  - Tuple supports format strings
  - Tuple supports variable length parameter in function call.
  - Tuples slightly faster

Set

- Sets are used to store multiple items in a single variable.
- A set is a collection which is unordered, unchangeable*, and unindexed.
- Tuples are written with curly brackets { }
- The items in sets will appear in a random order every time you refreshed.
- Example:

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

Output: {'cherry', 'apple', 'banana'}

```
# empty set
my_set = set()
# set with mixed data types
my_set = {1, "Hello", 3.4}
```

Characteristic of Set

Iteratable	• Set elements can be iterated one by one in a loop or a function where iteratable values are accepted.
Heterogeneous	• All the elements can be of any Type. Only restriction is we can only add Hashable type elements in Set. So List, Dictionary, set and bytearray cannot be an element of set.
Unordered / Non-Indexed	• Set items does not have a defined order. • Items just has a participation in the set so its sequence may vary every time.
Mutable	• It can change, add, and remove items in a list after it has been created.
No Duplicates	• As the set is not Indexed, same element cannot participate in the set more than once so duplicates won't be considered as 2 different items.

Length of Set

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

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

Output: 3

Access Set 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:

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

```
Output: banana
        apple
        cherry
```

Check if Item Exists in Set or Not

- To determine if a specified item is present in a sets use the in keyword:
- Example:

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

Output: True

Update Set Values/Items

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

Insert Items in Set

- To add one item to a set use the add() method.
- Example:

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

Output: {'orange', 'cherry', 'banana', 'apple'}

- To add items from another set into the current set, use the update() method.
- Example: Add elements from tropical into thisset:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset.update(tropical)
print(thisset)
```

Output: {'apple', 'mango', 'cherry', 'pineapple', 'banana', 'papaya'}

Remove Set Items

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

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

Output: {'apple', 'cherry'}

- Example:

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

Output: {'apple', 'cherry'}

- The del keyword will delete the set completely:
- Example:

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

Output: this will raise an error because the set no longer exist

Clear Set

- The clear() method empties the set:
- Example:

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

Output: set()

Looping in Set

- Loop Through a set (for loop)

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

Output:

```
apple
banana
cherry
```

Join Sets : Union and Update

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

Output:

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

```
set1 = {"a", "b" , "c"}

set2 = {1, 2, 3}

set1.update(set2)

print(set1)
```

Output:

```
{2, 1, 'b', 'c', 3, 'a'}
```

Sets Intersection

- The intersection() method will return a new set, that only contains the items that are present in both sets.
- Example:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.intersection(y)
print(z)
```

Output:

```
{'apple'}
```

Sets Difference

- Return a set that contains the items that only exist in set x, and not in set y.]
- Example:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.difference(y)
print(z)
```

Output:

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

Sets Symmetric Difference

- The symmetric_difference() method will return a new set, that contains only the elements that are NOT present in both sets.
- Example:

```
x = {"apple", "banana", "cherry"}
y = {"google", "microsoft", "apple"}
z = x.symmetric_difference(y)
print(z)
```

Output:

```
{'google', 'banana', 'microsoft', 'cherry'}
```

Set Methods

Method	Description
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

Dictionary

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- Dictionaries are written with curly brackets { }, and have keys and values.
- Example:

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

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Characteristic of Dictionary

Iteratable	• Dictionary elements can be iterated one by one in a loop or a function where iteratable values are accepted.
Hashable Keys	• Keys can be of any type but they must be Hashable and each key can be of different type
Heterogeneous – Mutable Values	• All the Values can be of any Type. No Restriction around that but per key only 1 value is allowed to be entered.
Ordered / Indexed	• Dictionary items are Ordered and they are defined by their Keys.
Allow Duplicates	• As Dictionary item is defined by their keys, the values can have duplicated but keys must be unique.

Access Dictionary Items

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

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

Output: Mustang

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

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

Output: Mustang

Get Keys of Dictionary

- The keys() method will return a list of all the keys in the dictionary.
- Example:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
    }
x = thisdict.keys()
print(x)
    Output: dict_keys(['brand', 'model', 'year'])
```

Get Values of Dictionary

- The values() method will return a list of all the values in the dictionary.
- Example:

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

Output: dict_values(['Ford', 'Mustang', 1964])

Get Items of Dictionary

- The items() method will return each item in a dictionary, as tuples in a list.

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

Output: dict_items([('brand', 'Ford'), ('model', 'Mustang'), ('year', 1964)])

Check if Key Exists in Dictionary or Not

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

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

Output: Yes, 'model' is one of the keys in the

Change Dictionary Items

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

```
thisdict ={
      "brand": "Ford",
      "model": "Mustang",
      "year": 1964
}
thisdict["year"] = 2018
print(thisdict)
      Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 2018}
```

Update Dictionary

- The update() method will update the dictionary with the items from the given argument.
- The argument must be a dictionary, or an iterable object with key:value pairs.
- Example:

```
thisdict = {
    "brand": "Ford",
    "model": "Mustang",
    "year": 1964
}
thisdict.update({"year": 2020})
print(thisdict)
```

Output: {'brand': 'Ford', 'model': 'Mustang', 'year':

Add Items in Dictionary

- 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)
```

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964, 'color': 'red'}

Remove Items From Dictionary

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

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

- The popitem() method removes the last inserted item.
- Example:

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

- The del keyword delete Dictionary completely.
- Example:

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

Output: this will cause an error because "thisdict" no longer exists.

Clear Dictionary

- The clear() method empties the dictionary.
- Example:

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

Output: { }

Looping in 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

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

Output: brand
model
year

- Example: Print all values in the dictionary, one by one

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

Output: Ford
Mustang
1964

- Example: Loop through both keys and values, by using the items() method:

```
thisdict ={
      "brand": "Ford",
      "model": "Mustang",
      "year": 1964
}
for x, y in thisdict.items():
      print(x, y)
```

Output: brand Ford
model Mustang
year 1964

Copy 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.
- Method-1: Using built-in Dictionary method copy().
- Example:

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

Output: {'brand': 'Ford', 'model': 'Mustang',

- Method-2: Using dict() function
- Example:

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

Output: {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}

Nested Dictionary

- A dictionary can contain dictionaries, this is called nested dictionaries.
- Example:

```
myfamily = {
        "child1" : {
                "name" : "Emil",
                "year" : 2004
        },
        "child2" : {
                "name" : "Tobias",
                "year" : 2007
        },
        }
print(myfamily)
```

Output: {'child1': {'name': 'Emil', 'year': 2004}, 'child2': {'name': 'Tobias', 'year': 2007}}

Access Items in Nested Dictionary

- To access items from a nested dictionary, you use the name of the dictionaries, starting with the outer dictionary:
- Example:

```
myfamily = {
        "child1" : {
                "name" : "Emil",
                "year" : 2004
        },
        "child2" : {
                "name" : "Tobias",
                "year" : 2007
        }
        }
print(myfamily["child2"]["name"])
```

Output:Tobias

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