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

```
# 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.


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

Ordered

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

Changeable

Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

#Duplicates Not Allowed

Dictionaries cannot have two items with the same key:


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

Dictionary Length

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


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

```python
# Dictionary Items - Data Types
# The values in dictionary items can be of any data type:

thisdict = {
  "brand": "Ford",
  "electric": False,
  "year": 1964,
  "colors": ["red", "white", "blue"]
}
print(thisdict)


# type()
# From Python's perspective, dictionaries are defined as objects with the data type 'dict':
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(type(thisdict))


# The dict() Constructor
# It is also possible to use the dict() constructor to make a dictionary.

thisdict = dict(name = "John", age = 36, country = "Norway")
print(thisdict)
```

```python
# Accessing Items
# You can access the items of a dictionary by referring to its key name, inside square brackets:

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


# There is also a method called get() that will give you the same result:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
x = thisdict.get("model")
print(x)


# Add a new item to the original dictionary, and see that the keys list gets updated as well:
car = {
"brand": "Ford",
"model": "Mustang",
"year": 1964
}
```

```python
x = car.keys()

print(x) #before the change

car["color"] = "white"

print(x) #after the change

# Get Values
# The values() method will return a list of all the values in the dictionary.
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}

x = thisdict.values()

print(x)

# Get Items
# 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()
```

```python
  print(x)


# Check if Key Exists
# To determine if a specified key is present in a dictionary use the in keyword:


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


# Change Values
# You can change the value of a specific item by referring to its key name:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}


thisdict["year"] = 2018
print(thisdict)


# Update Dictionary
# The update() method will update the dictionary with the items from the given argument.
thisdict = {
```

```python
  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

thisdict.update({"year": 2020})


print(thisdict)
```

Adding Items

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

```python
thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

thisdict["color"] = "red"

print(thisdict)
```

Update Dictionary

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

```python
thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

thisdict.update({"color": "red"})
```

```python
# Removing Items
# There are several methods to remove items from a dictionary:


# 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)


# The del keyword removes the item with the specified key name:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
del thisdict["model"]
print(thisdict)


# The clear() method empties the dictionary:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
```

```python
  "year": 1964
}
thisdict.clear()
print(thisdict)


# Loop Through a Dictionary


# Print all key names in the dictionary, one by one:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(x)


# Print all values in the dictionary, one by one:
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
for x in thisdict:
  print(thisdict[x])


# You can also use the values() method to return values of a dictionary:
thisdict = {
  "brand": "Ford",
```

```python
  "model": "Mustang",

  "year": 1964

}

for x in thisdict.values():

  print(x)


# you can use the keys() method to return the keys of a dictionary:

thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

for x in thisdict.keys():

  print(x)


# 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)


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

thisdict = {

  "brand": "Ford",
```

```python
  "model": "Mustang",

  "year": 1964

}

mydict = thisdict.copy()

print(mydict)


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

thisdict = {

  "brand": "Ford",

  "model": "Mustang",

  "year": 1964

}

mydict = dict(thisdict)

print(mydict)


# Nested Dictionaries
# A dictionary can contain dictionaries, this is called nested dictionaries.

myfamily = {

  "child1" : {

    "name" : "Emil",

    "year" : 2004

  },

  "child2" : {

    "name" : "Tobias",

    "year" : 2007

  },

  "child3" : {

    "name" : "Linus",
```

```python
    "year" : 2011
  }
}
print(myfamily)


# Access Items in Nested Dictionaries
# To access items from a nested dictionary, you use the name of the dictionaries,
starting with the outer dictionary:


myfamily = {
  "child1" : {
    "name" : "Emil",
    "year" : 2004
  },
  "child2" : {
    "name" : "Tobias",
    "year" : 2007
  },
  "child3" : {
    "name" : "Linus",
    "year" : 2011
  }
}
print(myfamily["child2"]["name"])


# Loop Through Nested Dictionaries
# You can loop through a dictionary by using the items() method like this:
myfamily = {
```

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

for x, obj in myfamily.items():
   print(x)

   for y in obj:
     print(y + ':', obj[y])
```