



DICTIONARIES AND SETS

Dictionaries:

- A dictionary is an object that stores a collection of data. Each element in a dictionary has two parts: a key and a value. You use a key to locate a specific value.
- A dictionary is a collection which is ordered, changeable and does not allow duplicates.
- Dictionaries are written with curly brackets, and have keys and values

Creating a Dictionary:

- You can create a dictionary by enclosing the elements inside a set of curly braces (`{ }`).
- `phonebook = {'Chris':'555-1111', 'Katie':'555-2222', 'Joanne':'555-3333'}`
- The first element is `'Chris':'555-1111'`. In this element, the key is `'Chris'` and the value is `'555-1111'`.
- The second element is `'Katie':'555-2222'`. In this element, the key is `'Katie'` and the value is `'555-2222'`.
- The third element is `'Joanne':'555-3333'`. In this element, the key is `'Joanne'` and the value is `'555-3333'`.

Ordered or Unordered?:

- When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change.
- Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.

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.

Example:

- Duplicate values will overwrite existing values:

```
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:

Example

- Print the number of items in the dictionary:
- `print(len(thisdict))`

Dictionary Items - Data Types

- The values in dictionary items can be of any data type:

Example

- String, int, boolean, and list data types.

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

Accessing Items

Example

- Get the value of the "model" key:

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

- There is also a method called `get()` that will give you the same result:
- `x = thisdict.get("model")`

Get Keys

- The `keys()` method will return a list of all the keys in the dictionary.
- `x = thisdict.keys()`
- The list of the keys is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.

Example

- 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  
}  
  
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.
- `x = thisdict.values()`
- The list of the values is a *view* of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list

◦ Example

- Make a change in the original dictionary, and see that the values list gets updated as well:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
x = car.values()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change
```

Get Items

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

Example

- Make a change in the original dictionary, and see that the items list gets updated as well:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

```
x = car.items()
```

```
print(x) #before the change
```

```
car["year"] = 2020
```

```
print(x) #after the change
```

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
```

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

- Update the "year" of the car by using the update() method.

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

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)
```

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.
- The argument must be a dictionary, or an iterable object with key:value pairs.

Example

- Add a color item to the dictionary by using the `update()` method:

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

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.

```
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
```

- 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)
```

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

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

Example

- Print all *values* in the dictionary, one by one.

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

Example

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

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

Example

- 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)
```

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)
```

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`.

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 contain 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 add three dictionaries into a new dictionary.

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  
}
```

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
<code>setdefault()</code>	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value
<code>update()</code>	Updates the dictionary with the specified key-value pairs
<code>values()</code>	Returns a list of all the values in the dictionary

Python Sets

- 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*.
- *** Note:** Set *items* are unchangeable, but you can remove items and add new items.

Example

- Create a Set

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