Python Dictionaries

12-2-2024

By Ms. Soofi Shafiya

Dictionary

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

- 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.

Ordered/unordered?

As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are 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 do not have a defined order, you cannot refer to an item by using an index.
- Dictionaries are changeable, meaning that we can change, add or remove items after the dictionary has been created.

- Dictionaries cannot have two items with the same key.
- Duplicate values will overwrite existing values.

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

Len(): length of dictionary

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

Type(): 'dict'

Dict() constructor

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

- We can access the items of a dictionary by referring to its key name, inside square brackets:
- `x = thisdict["model"]` OR
- `x = thisdict.get("model") #using get() method`

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.

```
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.

```
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()'`
- The returned list is a *view* of the items of the dictionary, meaning that any changes done to the dictionary will be reflected in the items list.
-

Check if Key Exists

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

```
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
```

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.

```
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:

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

Removing Items

- Pop() # removes the item with the specified key name

```
thisdict.pop("model")
```

- Popitem() # removes the last inserted item

```
thisdict.popitem()
```

- Del() # removes the item with the specified key name

```
del thisdict["model"]
del thisdict #  delete the dictionary completely.
```

Clear() # method empties the dictionary:

```
thisdict.clear()
```

Loop through a dictionary

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

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

```
for x in thisdict:
  print(thisdict[x]) # Print all values in the dictionary, one by one
```

```
for x in thisdict.values(): #we can also use values() method method to return values of a dictionary:
  print(x)
```

We can use keys() method to return the keys of a dictionary

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

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

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

Copy Dictionaries

We cannot copy a dictionary simply by typing dict2=dict1, because dict2 will only be a reference to dict1. And change made in dict1 will reflect in dict2.

COPY()

```
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()

```
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",
    "year" : 2011
  }
}
```

Or, if you want to add three dictionaries into a new dictionary

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

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

Access Items in Nested Dictionaries

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

```
print(myfamily["child2"]["name"])
```

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