Python Dictionary

Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.

Dictionaries are optimized to retrieve values when the key is known.

Creating Python Dictionary

Creating a dictionary is as simple as placing items inside curly braces { } separated by commas.

An item has a key and a corresponding value that is expressed as a pair (key: value).

While the values can be of any data type and can repeat, keys must be of immutable type (string, number or tuple with immutable elements) and must be unique.

```
# empty dictionary
my_dict = {}
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
# using dict()
my_dict = dict({1:'apple', 2:'ball'})
# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

As you can see from above, we can also create a dictionary using the built-in dict() function.

Accessing Elements from Dictionary

While indexing is used with other data types to access values, a dictionary uses keys.

Keys can be used either inside square brackets [ ] or with the get() method.

If we use the square brackets [ ],KeyError is raised in case a key is not found in the dictionary. On the other hand, the get() method returns None if the key is not found.

```
# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}
# Output: Jack
print(my_dict['name'])
# Output: 26
print(my_dict.get('age'))
# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))
# KeyError
print(my_dict['address'])
```

Changing and Adding Dictionary elements

Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.

If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

```
# Changing and adding Dictionary Elements
my_dict = {'name': 'Jack', 'age': 26}
# update value
```

```
my_dict['age'] = 27
#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)
# add item
my_dict['address'] = 'Downtown'
# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
```

Removing elements from Dictionary

We can remove a particular item in a dictionary by using the pop() method. This method removes an item with the provided key and returns the value

The popitem() method can be used to remove and return an arbitrary (key,value) item pair from the dictionary. All the items can be removed at once, using the clear() method.

We can also use the del() keyword to remove individual items or the entire dictionary itself.


```
# Removing elements from a dictionary
# create a dictionary
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
# remove a particular item, returns its value
# Output: 16
print(squares.pop(4))
# Output: {1: 1, 2: 4, 3: 9, 5: 25}
print(squares)
# remove an arbitrary item, return (key,value)
# Output: (5, 25)
print(squares.popitem())
# Output: {1: 1, 2: 4, 3: 9}
print(squares)
# remove all items
squares.clear()
# Output: {}
print(squares)
# delete the dictionary itself
del squares
# Throws Error
print(squares)
```

Python Dictionary Methods

Methods that are available with a dictionary are tabulated below. Some of them have already been used in the above examples.

Method	Description
clear()	Removes all items from the dictionary.
copy()	Returns a shallow copy of the dictionary.
get(key[,d])	Returns the value of the key. If the key does not exist, returns d
items()	Return a new object of the dictionary's items in (key, value) format.
keys()	Returns a new object of the dictionary's keys.
update([other])	Updates the dictionary with the key/value pairs from other, overwriting existing keys.
values()	Returns a new object of the dictionary's values

Python Dictionary Comprehension

Dictionary comprehension is an elegant and concise way to create a new dictionary from an iterable in Python.

Dictionary comprehension consists of an expression pair (key: value) followed by a for statement inside curly braces { }

Here is an example to make a dictionary with each item being a pair of a number and its square.

```
# Dictionary Comprehension
squares = {x: x*x for x in range(6)}
print(squares)
```

Output

```
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

A dictionary comprehension can optionally contain more for or if statements.

An optional if statement can filter out items to form the new dictionary.

Here are some examples to make a dictionary with only odd items.

```
# Dictionary Comprehension with if conditional
odd_squares = {x: x*x for x in range(11) if x % 2 == 1}
print(odd_squares)
```

Output

```
{1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
```

Other Dictionary Operations

Dictionary Membership Test

We can test if a key is in a dictionary or not using the keyword in Notice that the membership test is only for the keys and not for the values.

```
# Membership Test for Dictionary Keys

squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}

# Output: True

print(1 in squares)

# Output: True

print(2 not in squares)

# membership tests for key only not value

# Output: False

print(49 in squares)
```

Iterating Through a Dictionary

We can iterate through each key in a dictionary using a for loop

```
# Iterating through a Dictionary
squares = {1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
for i in squares:
    print(squares[i])
```

Output

```
1
9
25
49
81
```