Dictionary

Dictionary in Python is an unordered collection of data values,
Dictionaries are used to store data values in key:value pairs.

Creating a Dictionary

A Dictionary can be created by placing a sequence of elements within curly **{}** braces, separated by ‘comma’. Dictionary holds pairs of values, one being the Key and the other corresponding pair element being its **Key:value**. Values in a dictionary can be of any data type and can be duplicated, whereas keys can’t be repeated and must be *immutable*.

Note – Dictionary keys are case sensitive, the same name but different cases of Key will be treated distinctly.

```
#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')])
```

Python 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

Accessing Elements from Dictionary or Indexing

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}
print(my_dict['name'])
print(my_dict.get('age'))
# Trying to access keys which doesn't exist throws error
print(my_dict.get('address'))
# KeyError
print(my_dict['address'])

Output
Jack
26
None
Traceback (most recent call last):
  File "<string>", line 15, in <module>
    print(my_dict['address'])
KeyError: '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'}
print(my_dict)
Output
{'name': 'Jack', 'age': 27}
{'name': 'Jack', 'age': 27, 'address': 'Downtown'}
```

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
print(squares.pop(4))
print(squares)

# remove an arbitrary item, return (key,value)
print(squares.popitem())
print(squares)

# remove all items
squares.clear()
print(squares)

# delete the dictionary itself
del squares

# Throws Error
print(squares)

Output
16
{1: 1, 2: 4, 3: 9, 5: 25}
(5, 25)
{1: 1, 2: 4, 3: 9}
{}
Traceback (most recent call last):
  File "<string>", line 30, in <module>
    print(squares)
NameError: name 'squares' is not defined
```

Iterating through dictionaries

To 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
```

Dictionary Built-in Functions

Function	Description
all()	Return True if all keys of the dictionary are True (or if the dictionary is empty).
any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.
len()	Return the length (the number of items) in the dictionary.
cmp()	Compares items of two dictionaries. (Not available in Python 3)
sorted()	Return a new sorted list of keys in the dictionary.
str(dict)	It converts the dictionary into the printable string representation.
type(variable)	It is used to print the type of the passed variable.

```
# Dictionary Built-in Functions
squares = {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
print(all(squares))
print(any(squares))
print(len(squares))
print(sorted(squares))
print("Equivalent string: %s" %str(squares))
print("variable type: %s" %type(squares))
```

Output

```
False
True
6
[0, 1, 3, 5, 7, 9]
Equivalent string: {0: 0, 1: 1, 3: 9, 5: 25, 7: 49, 9: 81}
variable type: <class 'dict'>
```

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 {}.

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

Output

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

```
# 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}
```