# Python – Dictionary

Python's dictionaries are kind of hash-table type. They work like associative arrays or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept':
'sales'}
print (dict['one']) # Prints value for 'one'
key
print (dict[2]) # Prints value for 2 key
print (tinydict) # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the
values
```

# Python – Accessing values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class':
'First'}
print ("dict['Name']: ", dict['Name'])
print ("dict['Age']: ", dict['Age'])
```

Key error when we try to access the value for key that is not present eg:

print(dict[“School”]

PRIME INTUIT
Finishing School

# Python – Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below:

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School" # Add new entry
print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School'])
```

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.
To explicitly remove an entire dictionary, just use the **del** statement.

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name'] # remove entry with key 'Name'
dict.clear() # remove all entries in dict
del dict # delete entire dictionary
print ("dict['Age']: ", dict['Age'])
print ("dict['School']: ", dict['School’])

TypeError: 'type' object is unsubscriptable
```

# Python – Properties

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.
There are two important points to remember about dictionary keys-
**(a)** More than one entry per key is not allowed. This means no duplicate key is allowed. When duplicate keys are encountered during assignment, the last assignment wins.

```
dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print ("dict['Name']: ", dict['Name'])
```

**(b)** Keys must be immutable. This means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed.

```
dict = {['Name']: 'Zara', 'Age': 7}
print ("dict['Name']: ", dict['Name'])
```

PRIME INTUIT
Finishing School

# Built-in Dictionary Functions & Methods

| SN | Functions with Description |
|---|---|
| 1 | **cmp(dict1, dict2)**<br>No longer available in Python 3. |
| 2 | **len(dict)**<br>Gives the total length of the dictionary. This would be equal to the number of items in the dictionary. |
| 3 | **str(dict)**<br>Produces a printable string representation of a dictionary. |
| 4 | **type(variable)**<br>Returns the type of the passed variable. If passed variable is dictionary, then it would return a dictionary type. |

PRIME INTUIT
Finishing School

# Built-in Dictionary Functions & Methods

| SN | Methods with Description |
|---|---|
| 1 | **dict.clear()**<br>Removes all elements of dictionary *dict.* |
| 2 | **dict.copy()**<br>Returns a shallow copy of dictionary *dict.* |
| 3 | **dict.fromkeys()**<br>Create a new dictionary with keys from seq and values *set* to *value*. |
| 4 | **dict.get(key, default=None)**<br>For *key* key, returns value or default if key not in dictionary. |

PRIME INTUIT
Finishing School

# Built-in Dictionary Functions & Methods

| | |
|---|---|
| 5 | **dict.has_key(key)**<br>Removed, use the **in** operation instead. |
| 6 | **dict.items()**<br>Returns a list of *dict*'s (key, value) tuple pairs. |
| 7 | **dict.keys()**<br>Returns list of dictionary dict's keys. |
| 8 | **dict.setdefault(key, default=None)**<br>Similar to get(), but will set dict[key]=default if *key* is not already in dict. |
| 9 | **dict.update(dict2)**<br>Adds dictionary *dict2*'s key-values pairs to *dict.* |
| 10 | **dict.values()**<br>Returns list of dictionary *dict*'s values. |

PRIME INTUIT
Finishing School

# Python – Sets

Python's sets are used to store collections of data / multiple items in a single variable. Sets Is a collection of data which is unordered, unchangeable and **unindexed,** Set items do not allow duplicate values

setss are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).

```
dict = {}
dict['one'] = "This is one"
dict[2] = "This is two"
tinydict = {'name': 'john','code':6734, 'dept':
'sales'}
print (dict['one']) # Prints value for 'one'
key
print (dict[2]) # Prints value for 2 key
print (tinydict) # Prints complete dictionary
print (tinydict.keys()) # Prints all the keys
print (tinydict.values()) # Prints all the
values
```