DICTIONARIES

- Unordered collection of items.
- Key/value pair.
- Keys will be a single element.
- Values can be a list or list within a list, numbers, etc.
- Mutable.

Creating Python Dictionary

- Curly braces { } separated by comma.
- Pair (key : value)

Example:

```
d = {} # empty dictionary
d = {1: 'apple', 2: 'ball'} # dictionary with integer keys
d = {'name': 'Abhishek', 1: [2, 4, 3]} # dictionary with mixed keys

# using dict()
d = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
d = dict([(1,'apple'), (2,'ball')])
```

Properties of Dictionary Keys

- More than one entry per key is not allowed (no duplicate key is allowed).
- The values in the dictionary can be of any type, while the keys must be immutable like numbers, tuples, or strings.
- Dictionary keys are case sensitive- Same key name but with the different cases are treated as different keys in Python dictionaries.

ACCESSING VALUES IN DICTIONARIES

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

Example:

```
# get vs [] for retrieving elements
d = {'name': 'Ram', 'age': 26}
print(d['name'])
print(d.get('age'))
```

```
# Trying to access keys which doesn't exist throws error
print(d.get('address')) #None
print(d['address']) # KeyError
```

WORKING WITH DICTIONARIES

Operations on Dictionaries

1. Changing and Adding Dictionary Elements

- Mutable
- Assignment operator

Example:

```
d = {'name': 'Ram', 'age': 26}
d['age'] = 29   # update value
print(d)
d['address'] = 'Vadodara'   # add item
print(d)
```

2. Removing Elements from Dictionary

- Particular item – **pop()** method
- Arbitrary – **popitem()** method
- All the items removed at once – **clear()**
- We can delete the entire dictionary using **del** keyword.

Example:

```
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
print(squares) # Throws Error
```

DICTIONARY METHODS

- clear()
- copy()
- fromkeys(seq[, v])
- get(key [, d])
- items
- keys()
- values()
- update()
- pop()
- popitem()
- setdefault(key, [,d])

OPERATIONS ON DICTIONARY

1. Membership Test

Example:

```
squares = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
print(1 in squares) # True
print(6 in squares) # False
print(6 not in squares) # True
print(25 in squares) # membership tests for key only not value
```

2. Iterating through a Dictionary

Example:

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

Converting Lists into Dictionary

- Two steps involved
- Step 1: Create a **'zip' class** object by passing the two lists to **zip() function**.
- Step 2: Convert the zip object into a dictionary by using the **dict() function.**
- The zip() function is useful to convert the sequences into a zip class object.

Example:

```
countries = ["USA","India","Germany","France"] #List 1
cities = ['New York','Delhi','Berlin','Paris'] #List 2

#Making a dictionary
z=zip(countries,cities)
d=dict(z)

#display key-value pairs from dictionary d
print('{:15s} --- {:15s}'.format('COUNTRY','CAPITAL'))
for k in d:
   print('{:15s} --- {:15s}'.format(k,d[k]))
```

Converting Strings into Dictionary

- Three steps involved
- Step 1: Split the string into pieces where a comma is found using **spilit() method** and then break the string at equals symbol.
- Step 2: Store these pieces into list using **append() method**.
- Step 3: Convert the list into dictionary.

Example:

```
str= "Ram=23, Pooja=21, Vivek=19, Khushi=24"

#brake the string and store the pieces into a list
lst = []
for x in str.split(','):
   y=x.split('=')
   lst.append(y)

#Convert the list into dictionary
d=dict(lst)

#Create new dictionary
d1={}
```

```
for k, v in d.items():
   d1[k]=int(v)

#Display
print(d1)
```

Passing Dictionaries into Functions

Example:

```
def fun(dictionary):
   for i,j in dictionary.items():
      print(i, '--', j)

d={'a':'apple', 'b':'ball', 'c':'cat', 'd':'door'}
fun(d)
```

Ordered Dictionaries

- The elements of the dictionary are not ordered. It means the elements are not stored into the same order as they were entered into the dictionary.
- Keep the order of the elements.
- **OrderedDict() method** of **'collection'** module.

Example:

```
from collections import OrderedDict
d=OrderedDict() #d is ordered dictinary
d[10]='Apple'
d[20]='Ball'
d[30]='Cat'
d[40]='Door'
d[50]='Eye'

for i, j in d.items():
   print(i,j)
```