# Dictionary

- Dictionaries are **mutable** built in types,
- Each element is a **key : value** pair
- **Keys are unique** within a dictionary while **values may not be**.
- ***Keys*** must be of an ***immutable data type*** such as strings, numbers, or tuples.
- **Declaration of dictionary→**
- **d={}** **#Empty dictionary**
- **dict1 = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}**
  # Dictionary with 3 elements

Accessing Values in Dictionary:

- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
- print(dict['Name'])   # prints ‘Zara’
- print (dict['Age'])   # prints 7
- print (dict[‘Class'])   # prints First
- Instead of index only key is required

Updating Dictionary

- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
- dict['Age'] = 12 # update existing entry
- dict['School'] = "DPS School" # Add new entry
- If key is not found adds new element with specified key:value pair
- print(dict['Age']) #prints 12
- print(dict['School']) #prints "DPS School"

- d={7 : ['ab',34,45.67], 23 : ['rt', 'yt', 'lo']}
- d[7][1]=90
- print(d)
- 'd={‘ab’ : {‘a’ : 2, ‘b’ : 34}, ‘bc’ : {‘x’ : 89, 90 : ‘n’}}
- #d['ab'] itself is a dictionary
- # d['ab']['a'] has value 2 in the dictionary
- d['ab']['a']=4 # d['ab']['a'] assigned value 4 now
- print(d)

update() method

- The dict1.update(dict2) adds dictionary dict2's key-values pairs in to dict1. This function does not return anything.
- dict1 = {'Name': 'Zara', 'Age': 7}
- dict2 = {'Sex': 'female' }
- dict1.update(dict2)
- ---
- d1 = {'a': 1, 'b': 2}
- d2 = {'b': 3, 'c': 4}
- d3={**d1,**d2} # {'a': 1, 'b': 3, 'c': 4}

Nested dictionaries

- d={'ab':{'a':2,'b':34},'bc':{'x':89,90:'n'}}
- print(d['ab']) # {'a': 4, 'b': 34}
- print(d['ab']['a']) # 2
- d['ab']['a'] = 4
- print(d)

fromkeys(seq[, value])

- dict.fromkeys(seq[, value])
- #creates a new dictionary with keys from seq (list, tuple, set, string)and values set to value.
- --
- seq = ('name', 'age', ‘mobile')
- dict = dict.fromkeys(seq)
- # {‘mobile': None, 'name': None, 'age': None}
- --
- dict = dict.fromkeys(seq, 10)
- # {'Mobile': 10, 'name': 10, 'age': 10}
- --
- d1=d1.fromkeys(“abcd”)
- # {'a': None, 'b': None, 'c': None, 'd': None}

Deleteting Dictionary Elements

- Remove individual dictionary elements
- dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
- del dict['Name'] # remove entry with key
- dict.pop(‘Name’) # removes specified key and returns corresponding value ‘Zara’
- dict.pop(‘School’, d) # returns d since ‘School’ key is not in dict
- dict.popitem() # returns any (key, value) pair tuple e.g. ('Age‘, 7)
- dict.clear() # remove all entries in dict leaving empty dictionary
- del dict # deletes entire dictionary object

Note that

- More than one entry per key not allowed. Which means no duplicate key is allowed.
- When duplicate keys encountered during assignment, the last assignment wins.
- For example –
- dict = {'Name': 'Zara', 'Age': 7, 'Name': ‘John'}
- print (dict['Name']) # prints ‘John’

Dictionary Methods

- **dict.clear()** **# Removes all elements**
- **dict.copy()** **# Returns a deep copy**
- **dict.get(key, default=None)** **# return a value for key**
- **key** - This is the Key to be searched in the dictionary.
- **default** - This is the Value to be returned in case key does not exist.
- **d.setdefault(k[,d])**
- **#d.get(k,d), also set D[k]=d if k not in D**
- **d.setdefault("Joe", 48)**
- **#returns 48 and key added--value 48**

Dr Harsh Dev

Dictionary Methods continued

- dict.items() # Returns a list of *dict*'s (key, value) tuple pairs
- dict.keys() # Returns list of dictionary keys
- dict.values() # Returns list of dictionary values

Generate a dictionary from tuples

- It’s often useful to generate a dictionary from a list of tuples.
- **pairs = [('key1', 'val1'), ('key2', 'val2'), ('key3', 'val3')]**
- **d = dict(pairs)**
- **print(d)**
- **# Out: {'key3’ : 'val3', 'key2‘ : 'val2', 'key1’ : 'val1'}**
- # Generate the same list of tuples via **list comprehension**
- **pairs = [('key{0}'.format(x), 'value{0}'.format(x))**
  for x in range(1, 4)]

Sorting

- d={'Jhon': 34, 'Ram': 23, 'Harry': 25, 'Joe’: 48, 'James': 97, 'Alice': 56}
- sorted(d)   # returns sorted list of keys
- #['Alice', 'Harry', 'James', 'Jhon', ‘Joe', 'Ram']
- sorted(d.keys()) # returns list of keys
- #['Alice', 'Harry', 'James', 'Jhon‘, ‘Joe', 'Ram']
- sorted(d.items())   # returns list of sorted key value pairs
- #[('Alice', 56), ('Harry', 25), ('James', 97), ('Jhon', 4), ('Joe', 48), ('Ram', 23)]
- sorted(d.values()) # returns list of values
- #[23, 25, 34, 48, 56, 97]

Dr Harsh Dev

Sorting

- sorted(d.items(), key=lambda x : x[1])
  # sort dictionary on value
- #[('Ram', 23), ('Harry', 25), ('Jhon', 34), ('Joe', 48), ('Alice', 56), ('James', 97)]
- sorted(d.items(), key=lambda x : x[0])
  # sort dictionary on keys
- #[('Alice', 56), ('Harry', 25), ('James', 97), ('Jhon', 34), ('Joe', 48), ('Ram', 23)]

efficient dictionary loops

- for key, value in my_dict.items():
-         print (key, value)
-
- for key in my_dict.keys():
-         print (key)
-
- for value in my_dict.values():
-         print (value)

Problem Description: Given below is a Dictionary Customer details representing customer Details from Retail Application. Customer **Id is key** and Customer **Name is value**

customer_details = { 1001 : "John", 1004 : "Jill", 1005: "Joe", 1003 : "Jack" }

Write Python code to perform below mentioned operations:

- Print details of Customers

  Ans: print(customer_details)
- Print number of Customers

  Ans: print(len(customer_details))
- Print Customer names in ascending order

  Ans: print(*sorted*(customer_details.values()))

- Delete the details of customer with customer id = 1005 and print updated dictionary

  Ans: **del(customer_details[1005])** or

  customer_details.pop(1004)

- Update the name of customer with customer id = 1003 to “Mary” and print updated dictionary

  Ans: **customer_details[1003] = "Mary"**

- Check whether details of customer with customer id 1002 exists in the dictionary.

  Ans: **print(1002 in customer_details)**