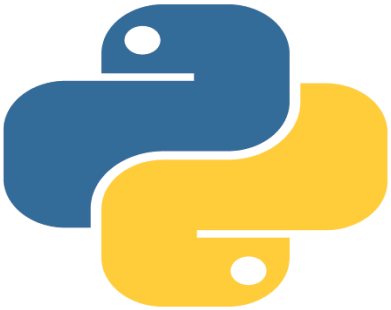

Python

Part 2

Python List

List is one of the built-in data types in Python. A Python list is a sequence of comma separated items, enclosed in square brackets [ ]. The items in a Python list need not be of the same data type.

```
list1 = ["python", "java", 80, 20.6,”c++”]
list2 = [1, 2, 3, 4, 5]
list3 = ["a", "b", "c", "d"]
list4 = [76.7, -88, True, 3+2j]
```

Example:

```
list1 = ["python", "java", 80, 20.6,”c++”]
print(list1)
list2=[1,2,,4,5]
print(list2)
```

Output:

```
[‘python’,’java’,80,20.6,’c++’]
[1,2,3,4,5]
```

Access element from list

To access value in list, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example:

```
list1 = ["python", "java", 80, 20.6,”c++”]
list2 = [1, 2, 3, 4, 5, 6]
print ("list1[0]: ", list1[0])
print(“list2[2]:”,list[2])
```

Output:

```
List[0]:python
List[2]:3
```

Updating list

Example:

```
list=[89,56,32,89,43,90]
print("list before update:" ,list)
list[2]=67
print("new list:" ,list)
```

Output:

```
List before update:[89,56,32,89,43,90]
New list:[89,56,67,89,43,90]
```

Access List Items with Negative Indexing

Negative indexing in Python is used to access elements from the end of a list, with -1 referring to the last element, -2 to the second last, and so on.

Example:

```
list=[89,56,32,89,43,90]
print("negative index value:" ,list[-1] )
print("negative index value:" ,list[-2])
```

Output:

```
negative index value: 90
negative index value: 43
```

Access List Items with Slice Operator

The slice operator in Python is used to fetch one or more items from the list. We can access list items with the slice operator by specifying the range of indices we want to extract

```
[start:stop]
```

Where

start is the starting index (inclusive).
stop is the ending index (exclusive).

Example:

```
list1 = ["a", "b", "c", "d"]

print ("Items from index 1 to last in list1: ", list1[1:])
print ("Items from index 0 to 1 in list2: ", list2[:2])
print ("Items from index 0 to index last in list3", list3[:])
print("Items from index 0 to1 in list "list[0:2])
```

Output:

```
Items from index 1 to last in list1:  ['b', 'c', 'd']
Items from index 0 to 1 in list2:  ['a','b']
Items from index 0 to index last in list3 ['a',b','c','d']
Items from index 0 to1 in list ['a','b']
```

Add list items

Adding list items in Python implies inserting new elements into an existing list. Lists are mutable, meaning they can be modified after creation, allowing for the addition, removal, or modification of their elements.

Adding List Items Using append() Method

The append() method in Python is used to add a single element to the end of a list.

Example:

```
list1=[1,2,3,4,5]
print("list before adding items:",list1)
list1.append(6)
print("list after adding items",list1)
```

Output:

```
List before adding items:['1','2,'3','4','5']
List after adding items:['1','2','3','4','5','6']
```

Adding List Items Using insert() Method

The insert() method in Python is used to add an element at a specified index (position) within a list,shifting existing elements to
accommodate

Example:

```
list1=[1,2,3,4,5,6]
list1.insert(2,8)
list2=("a","l","q","w")
list2.insert(3,"f")
print(list1)
print(list2)
```

Output:

```
['1','2','8','3','4','5','6']
['a','l',q','f','w']
```

Adding List Items Using extend() Method

The extend() method in Python is used to add multiple elements from an iterable (such as another list) to the end of a list.

Add list items using the extend() method by passing another iterable containing the elements we want to add, like **my_list.extend(iterable)**, which appends each element from the iterable to the end of **my_list**.

Example:

```
list=[1,2,3,4]
another_list=[5,6,7]
list.extend(another_list)
print(list)
```

Output:

```
[1,2,3,4,5,6,7]
```

Removing List Items

Removing list items in Python implies deleting elements from an existing list. Lists are ordered collections of items, and sometimes you need to remove certain elements from them based on specific criteria or indices. When we remove list items, we are reducing the size of the list or eliminating specific elements.

Remove List Item Using remove() Method

The remove() method in Python is used to remove the first occurrence of a specified item from **a list.**

Example:

```
list=[1,2,3,4,5,6,7,8,9,10]
print("list before removing operation",list)
list.remove(2)
print("list after removing operation",list)
```

Output:

```
List before removing operation [1,2,3,4,5,6,7,8,9,10]
List after removing list operation [1,2,4,5,6,7,8,9,10]
```

Remove List Item Using pop() Method

The pop() method in Python is used to removes and returns the last element from a list if no index is specified, or removes and returns the element at a specified index, altering the original list.

Example:

```
list=[6,12,6,7,8,9,10]
print("list before pop operation",list)
list.pop(4)
print("list after pop operation",list)
```

Output:

```
list before pop operation [6,12,6,7,8,9,10]
list after pop operation [6,12,6,7,9,10]
```

Remove List Item Using clear() Method

The clear() method in Python is used to remove all elements from a list, leaving it empty.

Example:

```
list=[1,2,3,4,5]
list.clear()
print("clear list:",list)
```

Output:

```
clear list:[]
```

Remove List Item Using del Keyword

The del keyword in Python is used to delete element either at a specific index or a slice of indices from memory.

Example:

```
list=['a','b','c','d','e','f','g']
del.list[3]
print("list after delete element",list)
```

Output:

```
List after delete element ['a','b',c','e','f','g']
```

Python - Loop Lists

Looping through list items in Python refers to iterating over each element within a list. We do so to perform the desired operations on each item. These operations include list modification, conditional operations, string manipulation, data analysis, etc.

Example:

```
list=[1,2,3,4,5,6,7]
for item in list:
    print(item,end=' ')
```

Output:

```
1 2 3 4 5 6 7
```

Example:

Output:

```
1
2
3
4
5
```

```
my_list = [1, 2, 3, 4, 5]
index = 0

while index < len(my_list):
   print(my_list[index])
   index += 1
```

Example:

```
lst = [98,65,34,67,89,12]
indices = range(len(lst))
for i in indices:
   print ("lst[{}]: ".format(i), lst[i])
```

Output:

```
lst[0]: 98
lst[1]: 65
lst[2]: 34
lst[3]: 67
lst[4]: 89
lst[5]: 12
```

Python - Sort Lists

Sorting a list in Python is a way to arrange the elements of the list in either ascending or descending order based on a defined criterion, such as numerical or lexicographical order.

Sorting Lists Using sort() Method

The python sort() method is used to sort the elements of a list in place. This means that it modifies the original list and does not return a new list.

Syntax

```
list_name.sort(key=None, reverse=False)
```

list_name is the name of the list to be sorted.
key (optional) is a function that defines the sorting criterion. If provided, it is applied to each element of the list for sorting. Default is None.
reverse (optional) is a boolean value. If True, the list will be sorted in descending order. If False (default), the list will be sorted in ascending order.

Example:

```
sort_list=["python","c++","java","csharp"]
sort_list.sort()
print("list after sorted",sort_list)
```

Output:

```
list after sorted ['c++', 'csharp', 'java', 'python']
```

Example:

```
sort_list=[34,56,12,32,23]
sort_list.sort()
print("list after sorted",sort_list)
```

Output:

```
list after sorted [12,23,32,34,56]
```

Python - Tuples

Tuple is one of the built-in data types in Python. A Python tuple is a sequence of comma separated items, enclosed in parentheses (). The items in a Python tuple need not be of the same data type.Tuple is immutable sequential datatypes.meaning once a tuple is created, its elements cannot be changed, added, or removed.

```
tup1=(“python”,”c++”,1,2,4)
tup2=(1,2,3,4,5,6)
tup3=(1,2.3,6,7.8,1+2j)
```

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain values available at that index.

Example:

```
tup1=(1,2,3,4,5,6,7)
print(tup1)
print(“Third indexed value:”,tup[3])
```

Output:

```
(1,2,3,4,5,6,7)
Third indexed value:4
```

Accessing Tuple Items with Negative Indexing

Negative indexing in Python is used to access elements from the end of a tuple, with **-1** referring to the last element, **-2** to the second last, and so on.

Example:

```
tup=(1,2,3,4,5,6,7,8,9,10)
print(tup[-1])
print(tup[-3])
```

Output:

```
10
8
```

Access Tuple Items with Slice Operator

The slice operator in Python is used to fetch one or more items from the tuple. We can access tuple items with the slice operator by specifying the range of indices we want to extract. It uses the following syntax

Syntax

```
[start:stop]
```

Example:

```
tup=(1,2,3,4,5,6,7,8,9,10)
print(tup[:])
print(tup[2:])
print(tup[:3])
print(tup[2:5])
```

Output:

```
(1,2,3,4,5,6,7,8,9,10)
(3,4,5,6,7,8,9,10)
(1,2,3)
(3,4,5)
```

Python - Loop Tuples

Looping through tuple items in Python refers to iterating over each element in a tuple sequentially.

Loop Through Tuple Items with For Loop

Syntax:

```
for item in tuple:
    #code block to execute
```

Example:

```
tup=(1,2,3,4,5.6.7.8.9,10)
for item in tup:
    print(tup,end=" ")
```

Output:

```
1 2 3 4 5 6 7 8 9 10
```

Loop Through Tuple Items with While Loop

A while loop in Python is used to repeatedly execute a block of code as long as a specified condition evaluates to "True".

```
while condition:
```

```
#code block to execute
```

Example:

```
my_tup = (1, 2, 3, 4, 5)
index = 0


while index < len(my_tup):
  print(my_tup[index])
  index += 1
```

Updating Tuples in Python

Example:

```
list1=[1,2,3,4,5]
list2=[“python,”c++”,”c”,”java”]
list_update=list1+list2
print(list_update)
```

Output:

```
[1,2,3,4,5,’python’,’c++’,’c’,’java’]
```

Set in python

In Python, a set is an unordered collection of unique elements. Unlike lists or tuples, sets do not allow duplicate values i.e. each element in a set must be unique. Sets are mutable, meaning you can add or remove items after a set has been created.

Sets are defined using curly braces **{}** or the built-in **set()** function.

Example:

```
set={1,2,3,4}
print(set)
```

Output:

```
{1,2,3,4}
```

Using the set() Function

Create a set using the set() function by passing an iterable (like a list or a tuple) containing the elements you want to include in the set.

Example:

```
my_set=set([1,2,3,4,5])
print(my_set)
```

Output:

```
{1,2,3,4,5}
```

Duplicate Elements in Set

Sets in Python are unordered collections of unique elements. If you try to create a set with duplicate elements, duplicates will be automatically removed

Example:

```
set={1,2,3,3,4,5,5,6,7,8}
print(set)
```

Output

```
{1,2,3,4,5,6,7,8}
```

Access Set Items

In Python, sets are unordered collections of unique elements, and unlike sequences (such as lists or tuples), sets do not have a positional index for their elements. This means that you cannot access individual elements of a set directly by specifying an index.

Access Set Items Using For Loop

A for loop in Python is a control flow statement used for iterating over a sequence and executing a block of code for each element in the sequence. The loop continues until all elements have been processed.

Example:

```
langs = {"C", "C++", "Java", "Python"}
# Accessing set items using a for loop
for lang in langs:
  print (lang)
```

Output:

```
C
C++
Java
Python
```

Adding Elements in a Set

The add() method in Python is used to add a single element to the set. It modifies the set by inserting the specified element if it is not already present. If the element is already in the set, the add() method has no change in the set.

Syntax

Following is the syntax to add an element to a set

```
set.add(obj)
```

Example:

```
set_= set()
# Adding elements to the set using add() method
language.add("C")
language.add("C++")
language.add("Java")
language.add("Python")


# Printing the updated set
print("Updated Set:", language)
```

Output:

```
Updated Set: {'C','C++','Java','Python'}
```

Add Set Items Using the update() Method

In Python, the update() method of set class is used to add multiple elements to the set. It modifies the set by adding elements from an iterable (such as another set, list, tuple, or string) to the current set. The elements in the iterable are inserted into the set if they are not already present.

Syntax

Following is the syntax to update an element to a set

```
set.update(obj)
```

Adding a Single Set Item

In the example below, we initialize a set called "my_set". Then, we use the update() method to add the element "4" to the set

Example:

```
my_set = {1, 2, 3}
# Adding element to the set
my_set.update([4])
# Print the updated set
print("Updated Set:", my_set)
```

Output:

```
Update Set: {1,2,3,4}
```

Python - Remove Set Items

Removing set items implies deleting elements from a set. In Python, sets are mutable, unordered collections of unique elements, and there are several methods available to remove items from a set based on different criteria.

We can remove set items in Python using various methods such as remove(), discard(), pop(), clear(), and set comprehension.

Remove Set Item Using remove() Method

The remove() method in Python is used to remove the first occurrence of a specified item from a set.

Example:

```
my_set = {"Chemistry", "Physics", 21, 69.75}
print ("Original set: ", my_set)


my_set.remove("Physics")
print ("Set after removing: ", my_set)
```

Output:

```
Original set:{'Chemistry','Physics',21,69,75}
Set after removing: {'Chemistry',21,69,75}
```

Remove Set Item Using discard() Method

The discard() method in set class is similar to remove() method. The only difference is, it doesn't raise error even if the object to be removed is not already present in the set collection.

Example:

```
my_set = {"Rohan", "Physics", 21, 69.75}
print ("Original set: ", my_set)


# removing an existing element
my_set.discard(21)
print ("Set after removing Physics: ", my_set)
```

Output:

```
Original set: {'Rohan','Physics',21,69,75}
Set after removing Physics: {'Rohan','Physics,69,75}
```

Remove Set Item Using pop() Method

We can also remove set items using the pop() method by removing and returning an arbitrary element from the set. If the set is empty, the pop() method will raise a KeyError exception.

Example:

```
my_set = {1, 2, 3, 4, 5}
# removing and returning an arbitrary element from the set
removed_element = my_set.pop()


# Printing the removed element and the updated set
print("Removed Element:", removed_element)
print("Updated Set:", my_set)
```

Output:

```
Removed Element: 1
Updated Set: {2, 3, 4, 5}
```

Remove Set Item Using clear() Method

The clear() method in set class removes all the items in a set object, leaving an empty set.

Example:

```
# Defining a set with multiple elements
my_set = {1, 2, 3, 4, 5}


# Removing all elements from the set
my_set.clear()
# Printing the updated set
print("Updated Set:", my_set)
```

Output:

```
Updated Set:set()
```

Python - Loop Sets

Loop Through Set Items

Looping through set items in Python refers to iterating over each element in a set. We can later perform required operations on each item. These operations include list printing elements, conditional operations, filtering elements etc.

Example:

```
my_set = {25, 12, 10, -21, 10, 100}
for item in my_set:
  # Performing operations on each element
  print("Item:", item)
```

Output:

```
Item: 100
Item: 25
Item: 10
Item: -21
Item: 12
```

Frozen Sets

In Python, a frozen set is an immutable collection of unique elements, similar to a regular set but with the distinction that it cannot be modified after creation. Once created, the elements within a frozen set cannot be added, removed, or modified, making it a suitable choice when you need an immutable set.

You can create a frozen set in Python using the frozenset() function by passing an iterable (such as a list, tuple, or another set) containing the elements you want to include in the frozen set.

Example:

```
my_frozen_set = frozenset([1, 2, 3])
print(my_frozen_set)
my_frozen_set.add(4)
```

Output:

```
frozenset({1, 2, 3})
Traceback (most recent call last):
  File "/home/cg/root/664b2732e125d/main.py", line 3, in <module>
    my_frozen_set.add(4)
AttributeError: 'frozenset' object has no attribute 'add'
```

Dictionaries in Python

In Python, a dictionary is a built-in data type that stores data in key-value pairs. It is an unordered, mutable, and indexed collection. Each key in a dictionary is unique and maps to a value. Dictionaries are often used to store data that is related, such as information associated with a specific entity or object, where you can quickly retrieve a value based on its key.

```
capitals={"Maharashtra":"Mumbai","Gujarat":"Gandhinagar","Telangana":"Hyderabad",
"Karnataka":"Bengaluru"}
numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}
marks = {"Savita":67, "Imtiaz":88, "Laxman":91, "David":49}
```

Example:

```
capitals={"Maharashtra":"Mumbai","Gujarat":"Gandhinagar","Telangana":"Hyderabad",
```

```
"Karnataka":"Bengaluru"}
print(capitals)
```

Output:

```
{‘Maharashtra’:’Mumbai’,’Gujarat’:’Gandhinagar’,’Telangana’:’Hyderabad’,
‘Karnataka’:’Bengaluru’}
```

Key Features of Dictionaries

Unordered − The elements in a dictionary do not have a specific order. Python dictionaries before version 3.7 did not maintain insertion order. Starting from Python 3.7, dictionaries maintain insertion order as a language feature.

Mutable − You can change, add, or remove items after the dictionary has been created.

Indexed − Although dictionaries do not have numeric indexes, they use keys as indexes to access the associated values.

Unique Keys − Each key in a dictionary must be unique. If you try to assign a value to an existing key, the old value will be replaced by the new value.

Heterogeneous − Keys and values in a dictionary can be of any data type.

Example:

```
d1 = {"city":[“Bangalur","Hydrabad"], "country":["India", "USA"]}
print (d1)
```

Output:

```
{"Fruit":["Mango","Banana"], "Flower":["Rose", "Lotus"]}
```

Access Dictionary Items

Accessing dictionary items in Python involves retrieving the values associated with specific keys within a dictionary data structure. Dictionaries are composed of key-value pair.

Example:

```
student1={"name":"Rahul","college":"Rajagiri","age":20}
print("Name of student:",student1["name"])
```

Output:

```
Name of student:Rahul
```

Access Dictionary Items Using get() Method

The get() method in Python's dict class is used to retrieve the value associated with a specified key. If the key is not found in the dictionary, it returns a default value (usually None) instead of raising a KeyError.

Syntax:

Following is the syntax of the get() method in Python

```
Val = dict.get("key")
```

Example:

```
student1={"name":"Rahul","College":"Rajagiri","Age":20}
print("Age of student::",student1l.get("age"))
```

Output:

```
Age of student: 20
```

Example:

```
number={“odd”:[1,3,5],”even”:[2,4,6]}
print(number)
```

Output:

```
{‘odd’:[1,3,5],’even’:[2,4,6]}
```

Example

In this example, we are retrieving all the keys from the dictionary "student_info" using the keys() method

```
student1={“name”:”Reshma”,”age”:24,”salary”:20000}
all_keys = employ_detail.keys()
print("Keys:", all_keys)
```

Output:

```
Keys: dict_keys(['name', 'age', 'salary'])
```

Change Dictionary Items

Changing dictionary items in Python refers to modifying the values associated with specific keys within a dictionary. This can involve updating the value of an existing key, adding a new key-value pair, or removing a key-value pair from the dictionary.

Modifying Dictionary Values

Modifying values in a Python dictionary refers to changing the value associated with an existing key. To achieve this, you can directly assign a new value to that key.

Example:

```
student1={"Name":"Rahul","College":"Rajagiri","Age":20}
print(“dictionary before modifying”,student1)
```

```
student1[“Age”]=21
print(“dictionary after modifying”,student1)
```

Output:

```
{‘Name’:’Rahul’,’College’:’Rajagiri’,’Age’:20}
```

Updating Multiple Dictionary Values

If you need to update multiple values in a dictionary at once, you can use the update() method. This method is used to update a dictionary with elements from another dictionary or an iterable of key-value pairs.

Example:

```
employ1={“Name”:”Amal”}
employ1.update({”Qualification”:”BTech”,”Salary”:20000})
print(employ1)
```

Output:

```
{‘Name’:’Amal’:’Qualification’:’BTech,’Salary’:20000}
```

Modify Dictionary by Adding New Key-Value Pairs

Adding new key-value pairs to a Python dictionary refers to inserting a new key along with its corresponding value into the dictionary.

Example:

```
employ1={“Name”:”Amal”,”Qualification”:”BTech”}
employ1[‘Salary’]=20000
print(employ1)
```

Output:

```
{'Name': 'Amal',’Qualification’:’BTech’, 'Salary':20000}
```

Modify Dictionary by Removing Key-Value Pairs

Removing key-value pairs from a Python dictionary refers to deleting specific keys along with their corresponding values from the dictionary.

Example: Using Assignment Operator

You can add a new key-value pair to a dictionary by directly assigning a value to a new key as shown below.

Example:

```
employ1={“Name”:”Amal”,”Qualification”:”BTech”}
employ1[‘Salary’]=20000
print(employ1)
```

Output:

```
{‘Name’:’Amal’,’Qualification’:’BTech’,Salary:20000}
```

Modify Dictionary by Removing Key-Value Pairs

Removing key-value pairs from a Python dictionary refers to deleting specific keys along with their corresponding values from the dictionary.

Example: Using the del Statement

Use the **del** statement to remove a specific key-value pair from a dictionary.

```
employ1={“Name”:”Amal”,”Qualification”:”BTech”,”Salary”:20000}
del.employ1[‘Name’]
print(employ1)
```

Output:

```
{‘Qualification’:’BTech’,’Salary’:20000}
```

Example: Using the pop() Method

Use the **pop()** method to remove a specific key-value pair from a dictionary and return the value associated with the removed key.

Example:

```
employ1={“Name”:”Amal”,”Qualification”:”BTech”,”Salary”:20000}
employ1.pop(‘Salary’)
print(employ1)
```

Output:

```
{‘Name’:’Amal’,’Qualication’:’BTech’,’Salary’:20000}
```

Example: Using the popitem() Method

Use the **popitem()** method as well to remove the last key-value pair from a dictionary and return it as a tuple

Example:

```
employ1={“Name”:”Amal”,”Qualification”:”BTech”,”Salary”:20000}
removed_item.popitem()
print(employ1)
print(“removed item”,removed_item)
```

Output:

```
{‘Name’:’Amal’,’Qualication’:’BTech’}
```

```
('Salary':20000)
```

Add Dictionary Items

Adding dictionary items in Python refers to inserting new key-value pairs into an existing dictionary. Dictionaries are mutable data structures that store collections of key-value pairs, where each key is associated with a corresponding value.

Add Dictionary Item Using Square Brackets

The square brackets **[]** in Python is used to access elements in sequences like lists and strings through indexing and slicing operations.

Example:

In this example, we are creating a dictionary named "marks" with keys representing names and their corresponding integer values.

Example:

```
sub_mark={"Physics":65,"Chemistry":70,"Biology":61}
print("Dictionary before adding",sub_mark)
sub_mark['Maths']=71
print("Dictionary after adding",sub_mark)
```

Output:

```
{'Physics':65,'Chemistry':70,'Biology':61,'Marhs':61}
```

Add Dictionary Item Using the update() Method

The update() method in Python dictionaries is used to merge the contents of another dictionary or an iterable of key-value pairs into the current dictionary.

Example:

```
sub_mark={"Physics":65,"Chemistry":70,"Biology":61}
print("Dictionary before adding",sub_mark)
```

```
sub_mark.update({'Maths': 58, 'Zoology': 90})
print ("Dictionary after new addition: ", sub_mark)
```

Output:

```
{‘Physics’:65,’Chemistry’:70,’Biology’:61,’Maths’:58,’Zoology’:90}
```

Python - Remove Dictionary Items

Removing dictionary items in Python refers to deleting key-value pairs from an existing dictionary. Dictionaries are mutable data structures that hold pairs of keys and their associated values.

Remove Dictionary Items Using del Keyword

The del keyword in Python is used to delete objects. In the context of dictionaries, it is used to remove an item or a slice of items from the dictionary, based on the specified key(s).

Example :

```
sub_mark={“Physics”:65,”Chemistry”:70,”Biology”:61}
print(“Dictionary before del operation”,sub_mark)
Del sub_mark[‘physics’]
print(“Dictionary after del operation”,sub_mark)
```

Output:

```
{’Chemistry’:70,’Biology’:61,}
```

Example:

The del keyword, when used with a dictionary object, removes the dictionary from memory −

```
sub_mark={“Physics”:65,”Chemistry”:70,”Biology”:61}
print(“Dictionary before del operation”,sub_mark)
Del sub_mark
print(“Dictionary after del operation”,sub_mark)
```

Output:

```
NameError: name 'sub_mark' is not defined
```

Remove Dictionary Items Using pop() Method

The pop() method in Python is used to remove a specified key from a dictionary and return the corresponding value.

Example:

```
sub_mark={“Physics”:65,”Chemistry”:70,”Biology”:61}
print(“Dictionary before del operation”,sub_mark)
sub_mark.pop(‘Chemistry’)
print(“Dictionary after del operation”,sub_mark)
```

Output:

```
{‘Physics’:65,’Biology’:61‘}
```

Remove Dictionary Items Using popitem() Method

The popitem() method in Python is used to remove and return the last key-value pair from a dictionary.

Example:

```
sub_mark={“Physics”:65,”Chemistry”:70,”Biology”:61}
print(“Dictionary before del operation”,sub_mark)
sub_mark.popitem()
print(“Dictionary after del operation”,sub_mark)
```

Output:

```
{‘Physics’:65,’Chemistry’:70,’Biology’:61}
```

Remove Dictionary Items Using clear() Method

The clear() method in Python is used to remove all items from a dictionary. It effectively empties the dictionary, leaving it with a length of 0.

Example:

```
sub_mark={“Physics”:65,”Chemistry”:70,”Biology”:61}
print(“Dictionary before del operation”,sub_mark)
sub_mark.popitem()
print(“Dictionary after del operation”,sub_mark)
```

Output:

```
{‘Physics’:65,’Chemistry’:70,’Biology’:61}
```

Loop Through Dictionaries

Looping through dictionaries in Python refers to iterating over key-value pairs within the dictionary and performing operations on each pair.

Example:

```
sub_mark={“Physics”:65,”Chemistry”:70,”Biology”:61}
For key in sub_mark:
        print(key,student[key])
```

Output:

```
Physics 65
Chemistry 70
Biology 61
```

Example:

In this approach, the loop iterates over the key-value pairs using the items() method of the dictionary. Each iteration provides both the key and its corresponding value.

Example:

```
student = {"name": "John", "age": 21, "major": "BTech"}
for key, value in student.items():
  print(key, value)
```

Output:

```
Name John
Age 21
Major BTech
```

Loop Through Dictionary Using dict.keys() Method

The dict.keys() method in Python is used to return a view object that displays a list of keys in the dictionary. This view object provides a dynamic view of the dictionary's keys, allowing you to access and iterate over them.

Example:

```
student = {"name": "John", "age": 21, "major": "BTech"}
# Looping through keys
for key in student.keys():
```

```
print(key)
```

Output:

```
Name
Age
Major
```

Loop Through Dictionary Using dict.values() Method

The dict.values() method in Python is used to return a view object that displays a list of values in the dictionary.

Example:

```
student = {"name": "John", "age": 21, "major": "BTech"}
# Looping through keys
for values in student.values():
  print(values)
```

Output:

```
John
21
BTech
```