CHAPTER 8

Data Structures

KEY Concepts

• Creating, Accessing, Updating, and Cloning Lists • List Methods and Functions • Functional Programming • Creating, Accessing, Updating, and Deleting Tuples • Working with Sets and Dictionaries • Nested Lists, Sets, Tuples, and Dictionaries • List and Dictionary Comprehensions

8.1 SEQUENCE

A *data structure* is a group of data elements that are put together under one name. Data structure defines a particular way of storing and organizing data in a computer so that it can be used efficiently.

Note All data structures discussed in this chapter are compound data structure as they are made of simple elements. For example, if we have defined a list as List = [1,2,3,4,5], then List is a compound data structure having integers 1, 2, 3, 4, and 5, which are the simple or basic elements.

Sequence is the most basic data structure in Python. In the sequence data structure, each element has a specific index. This index value starts from zero and is automatically incremented for the next element in the sequence. In Python, sequence is the generic term for an ordered set. For example, we have already studied strings which are a sequence of characters. In this chapter, we will learn about lists and tuples, which are also a type of sequence.

Python has some basic built-in functions that help programmers to manipulate elements that form a part of a sequence. These functions include finding the length of a sequence, finding the largest and smallest elements in a sequence, etc. Other operations that can be performed on a sequence include indexing, slicing, adding, multiplying, and checking for membership.

8.2 LISTS

List is a versatile data type available in Python. It is a sequence in which elements are written as a list of comma-separated values (items) between square brackets. The key feature of a list is that it can have elements that belong to different data types. Let us create lists by writing different comma-separated values between square brackets.

The syntax of defining a list can be given as,

```
List_variable = [val1, val2,...]
```



```
>>> list_A = [1,2,3,4,5]
>>> print(list_A)
[1, 2, 3, 4, 5]
```

```
>>> list_B = ['A', 'b', 'C', 'd', 'E']
>>> print(list_B)
['A', 'b', 'C', 'd', 'E']
```

```
>>> list_C = ["Good", "Going"]
>>> print(list_C)
['Good', 'Going']
```

```
>>> list_D = [1, 'a', "bcd"]
>>> print(list_D)
[1, 'a', 'bcd']
```

Note List is mutable which means that value of its elements can be changed.

8.2.1 Access Values in Lists

Similar to strings, lists can also be sliced and concatenated. To access values in lists, square brackets are used to slice along with the index or indices to get value stored at that index. If you can recollect from the last chapter, the syntax for the slice operation is given as,

```
seq = List[start:stop:step]
```

For example,

```
seq = List[::2]  # get every other element, starting with index 0
seq = List[1::2] # get every other element, starting with index 1
```

Example 8.1 Program to demonstrate the slice operations used to access the elements of the list

```
num_list = [1,2,3,4,5,6,7,8,9,10]
print("num_list is : ", num_list)
print("First element in the list is ", num_list[0])
print("num_list[2:5] = ", num_list[2:5])
print("num_list[::2] = ", num_list[::2])
print("num_list[1::3] = ", num_list[1::3])
```

OUTPUT

```
num_list is :  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
First element in the list is  1
num_list[2:5] =  [3, 4, 5]
num_list[::2] =  [1, 3, 5, 7, 9]
num_list[1::3] =  [2, 5, 8]
```

8.2.2 Updating Values in Lists

Once created, one or more elements of a list can be easily updated by giving the slice on the left-hand side of the assignment operator. You can also append new values in the list and remove existing value(s) from the list using the `append()` method and `del` statement respectively as shown in the following code.

Example 8.2 Program to illustrate updating values in a list

```
num_list = [1,2,3,4,5,6,7,8,9,10]
print("List is : ", num_list)
num_list[5] = 100
print("List after udpation is : ", num_list)
num_list.append(200)
print("List after appending a value is ", num_list)
del num_list[3]
print("List after deleting a value is ", num_list)
```

Programming Tip: `append()` and `insert()` methods are list methods. They cannot be called on other values such as strings or integers.

OUTPUT

```
List is :  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List after udpation is :  [1, 2, 3, 4, 5, 100, 7, 8, 9, 10]
List after appending a value is  [1, 2, 3, 4, 5, 100, 7, 8, 9, 10, 200]
List after deleting a value is  [1, 2, 3, 5, 100, 7, 8, 9, 10, 200]
```

Note If you know exactly which element(s) to delete, use the `del` statement, otherwise use the `remove()` method to delete the unknown elements.

Example 8.3 Programs to illustrate deletion of numbers from a list using `del` statements

```
num_list = [1,2,3,4,5,6,7,8,9,10]   # a list is defined
del num_list[2:4]          # deletes numbers at index 2 and 3
print(num_list)
```

OUTPUT

```
[1, 2, 5, 6, 7, 8, 9, 10]
```

```
num_list = [1,2,3,4,5,6,7,8,9,10]   # a list is defined
del num_list[:]          # deletes all the numbers from the list
print(num_list)          # an empty list is printed
```

OUTPUT

```
[]
```

Can you now imagine what will happen if you write `del num_list`? Yes, the entire variables will be deleted. If you make any attempt to use this variable after the `del` statement, then an error will be generated. This is very much evident from the code given in the following example.

Example 8.4 Program to illustrate deletion of a list

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

Programming Tip: When using slice operation, an IndexError is generated if the index is outside the list.

OUTPUT

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 3, in <module>
    print(num_list)
NameError: name 'num_list' is not defined
```

To insert items from another list or sequence at a particular location, you can use the *slice operation*. This will result in the creation of a list within another list. The program given below demonstrates this concept.

Example 8.5 Program to insert a list in another list using the slice operation

```
num_list = [1, 9,11,13,15]
print("Original List : ", num_list)
num_list[2] = [3,5,7]
print("After inserting another list, the updated list is : ", num_list)
```

OUTPUT

```
Original List :  [1, 9, 11, 13, 15]
After inserting another list, the updated list is :  [1, 9, [3, 5, 7], 13, 15]
```

8.2.3 Nested Lists

Nested list means a list within another list. We have already said that a list has elements of different data types which can include even a list. For example, in the following code, `list1` is a list that has another list at index 3.

Example 8.6 Program to illustrate nested list

```
list1 = [1, 'a', "abc", [2,3,4,5], 8.9]
i=0
while i<(len(list1)):
    print("List1[",i,"] = ",list1[i])
    i+=1
```

OUTPUT

```
List1[0] =  1
List1[1] =  a
List1[2] =  abc
List1[3] =  [2, 3, 4, 5]
List1[4] =  8.9
```

Remember that you can specify an element in the nested list by using a set of indices. For example, to print the second element of the nested list, we will write `print(list[3][1])`. The first index specifies the starting location of the nested list in the main list and the second index specifies the index of the element within the nested list.

8.2.4 Cloning Lists

If you want to modify a list and also keep a copy of the original list, then you should create a separate copy of the list (not just the reference). This process is called *cloning*. The slice operation is used to clone a list.

Example 8.7 Program to create a copy as well as the clone of the original list

```
list1 = [1,2,3,4,5,6,7,8,9,10]
list2 = list1                    #copies a list using reference
print("List1 = ", list1)
print("List2 = ", list2)         #both lists point to the same list
list3 = list1[2:6]
print("List3 = ", list3)         #list is a clone of list1
```

OUTPUT

```
List1 =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List2 =  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List3 =  [3, 4, 5, 6]
```

8.2.5 Basic List Operations

Lists behave in the similar way as strings when operators like + (concatenation) and * (repetition) are used. It works similar in case of operations discussed in Table 8.1.

Table 8.1 Operations on Lists

Operation	Description	Example	Output
len	Returns length of list	`len([1,2,3,4,5,6,7,8,9,10])`	`10`
concatenation	Joins two lists	`[1,2,3,4,5] + [6,7,8,9,10]`	`[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
repetition	Repeats elements in the list	`"Hello", "World"*2`	`['Hello', 'World', 'Hello', 'World']`
in	Checks if the value is present In the list	`'a' in ['a','e', 'i','o','u']`	`True`
not in	Checks if the value is not present In the list	`3 not in [0,2,4,6,8]`	`True`
max	Returns maximum value in the list	`>>> num_list = [6,3,7,0,1,2,4,9]` `>>> print(max(num_list))`	`9`
min	Returns minimum value in the list	`>>> num_list = [6,3,7,0,1,2,4,9]` `>>> print(min(num_list))`	`0`
sum	Adds the values in the list that has numbers	`num_list = [1,2,3,4,5,6,7,8,9,10]` `print("SUM = ", sum(num_list))`	`SUM = 55`

Contd

Table 8.1 *Contd*

Operation	Description	Example	Output
all	Returns True if all elements of the list are true (or if the list is empty)	`>>> num_list = [0,1,2,3]` `>>> print(all(num_list))`	`False`
any	Returns True if any element of the list is true. If the list is empty, returns False	`>>> num_list = [6,3,7,0,1,2,4,9]` `>>> print(any(num_list))`	`True`
list	Converts an iterable (tuple, string, set, dictionary) to a list	`>>> list1 = list("HELLO")` `>>> print(list1)`	`['H', 'E', 'L', 'L', 'O']`

sorted	Returns a new sorted list. The original list is not sorted.	`>>> list1 = [3,4,1,2,7,8]`
		`>>> list2 = sorted(list1)`
		`>>> print(list2)`
			`[1, 2, 3, 4, 7, 8]`

Let us try to see some more examples to understand how indexing, slicing, and other operations are performed on lists.

```
>>> list_A = ["Hello", "World", "Good", "Morning"]
>>> print(list_A[2])   # index starts at 0
Good
>>> print(list_A[-3])  # 3rd elemnt from the end
World
>>> print(list_A[1:])  # prints all elements starting
from index 1
['World', 'Good', 'Morning']
```

Programming Tip: An error is generated if you try to delete an element from the list that is not present in the list.

8.2.6 List Methods

Python has various methods to help programmers work efficiently with lists. Some of these methods are summarized in Table 8.2.

Table 8.2 **List Methods**

Method	Description	Syntax	Example	Output

`append()`	Appends an element to the list. In insert(), if the index is 0, then element is inserted as the first element and if we write, list.insert(len(list), obj), then it inserts obj as the last element in the list. That is, if index= len(list) then insert() method behaves exactly same as append() method.	`list.append(obj)`	`num_list = [6,3,7,0,1,2,4,9]`
			`num_list.append(10)`
			`print (num_list)`
				`[6,3,7,0,1,2,4,9,10]`

`count()`	Counts the number of times an element appears in the list.	`list.count (obj)`	`print(num_list.count(4))`	`1`
`index()`	Returns the lowest index of obj in the list. Gives a ValueError if obj is not present in the list.	`list.index(obj)`	`>>> num_list = [6,3,7,0,3,7,6,0]` `>>> print(num_list.index(7))`	`2`

Contd

Table 8.2 *Contd*

Method	Description	Syntax	Example	Output
insert()	Inserts obj at the specified index in the list.	list.insert(index, obj)	>>> num_list = [6,3,7,0,3,7,6,0] >>> num_list.insert(3, 100) >>> print(num_list)	[6,3,7,100,0,3,7,6,0]
pop()	Removes the element at the specified index from the list. Index is an optional parameter. If no index is specified, then removes the last object (or element) from the list.	list.pop([index])	num_list = [6,3,7,0,1,2,4,9] print(num_list.pop()) print(num_list)	9 [6, 3, 7, 0, 1, 2, 4]
remove()	Removes or deletes obj from the list. ValueError is generated if obj is not present in the list. If multiple copies of obj exists in the list, then the first value is deleted.	list.remove(obj)	>>> num_list = [6,3,7,0,1,2,4,9] >>> num_list.remove(0) >>> print(num_list)	[6,3,7,1,2,4,9]
reverse()	Reverse the elements in the list.	list.reverse()	>>> num_list = [6,3,7,0,1,2,4,9] >>> num_list.reverse() >>>print(num_list)	[9, 4, 2, 1, 7, 3, 6]
sort()	Sorts the elements in the list.	list.sort()	>>> num_list = [6,3,7,0,1,2,4,9] >>> num_list.sort() >>> print(num_list)	[9, 4, 2, 1, 0, 7, 3, 6]
extend()	Adds the elements in a list to the end of another list. Using + or += on a list is similar to using extend().	list1.extend(list2)	>>> num_list1 = [1,2,3,4,5] >>> num_list2 = [6,7,8,9,10] >>> num_list1.extend(num_list2) >>>print(num_list1)	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Key points to remember

- insert(), remove(), and sort() methods only modify the list and do not return any value. If you print the return values of these methods, you will get None. This is a design principle that is applicable to all mutable data structures in Python. The code given below illustrates this point.

Example 8.8 To print the return values

```
>>> num_list = [100, 200, 300, 400]
>>> print(num_list.insert(2, 250))
```

OUTPUT

```
None
```

Programming Tip: It is safer to avoid aliasing when you are working with mutable objects.

- When one list is assigned to another list using the assignment operator (=), then a new copy of the list is not made. Instead, assignment makes the two variables point to the one list in memory. This is also known as ***aliasing***.

Example 8.9 Program that uses the assignment operator to assign one list to another list variable

```
num_list1 = [1,2,3,4,5]
num_list2 = num_list1
print(num_list2)
```

OUTPUT

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

In the above code, the two lists, num_list1 and num_list2, point to the same memory location but are identified using two different names. This means that the two lists are **aliased**. Since lists are mutable, changes made with one alias affect the other.

Note An *alias* is a second name for a piece of data. In Python, aliasing happens whenever one variable's value is assigned to another variable.

- The sort() method uses ASCII values to sort the values in the list. This means that uppercase letter comes before lowercase letters and numbers comes even before the uppercase letters. The functionality of the sort() method is clearly evident from the code given below.

Example 8.10 Program to show the sort() mentioned

```
list1 = ['1', 'a', "abc", '2', 'B', "Def"]
list1.sort()
print(list1)
```

OUTPUT

```
[1, 2, 'B', 'Def', 'a', 'abc']
```

- Items in a list can also be deleted by assigning an empty list to a slice of elements as shown below.

Example 8.11 Program to delete items using empty list

```
list = ['p','r','o','g','r','a','m']
list[2:5] = []
print(list)
```

OUTPUT

```
['p', 'r', 'a', 'm']
```

8.2.7 Using Lists as Stack

Stack is an important data structure which stores its elements in an ordered manner. We will explain the concept of stacks using an analogy. You must have seen a pile of plates where one plate is placed on top of another as shown in Figure 8.1. Now, when you want to remove a plate, you remove the topmost plate first. Hence, you can add and remove an element (i.e. a plate) only at/from one position which is the topmost position.

Figure 8.1 A stack of plates

Stack is a linear data structure which uses the same principle, i.e., the elements in a stack are added and removed only from one end. Hence, a stack is called a **LIFO** (Last-In-First-Out) data structure, as the element that was inserted last is the first one to be taken out.

Now the question is, where do we need stacks in computer science? The answer is in function calls. Consider an example, where we are executing function A. In the course of its execution, function A calls another function B. Function B in turn calls another function C, which calls function D. In order to keep track of the returning point of each active function, a special stack called system stack or call stack is used. Whenever a function calls another function, the calling function is pushed onto the top of the stack. This is because after the called function gets executed, the control is passed back to the calling function. Look at Figure 8.2 which shows this concept.

Figure 8.2 Calling function from another function

Now, when function E is executed, function D will be removed from the top of the stack and executed. Once function D gets completely executed, function C will be removed from the stack for execution. The whole procedure will be repeated until all the functions get executed. Let us look at the stack after each

function is executed. This is shown in Figure 8.3. The system stack ensures a proper execution order of functions. Therefore, stacks are frequently used in situations where the order of processing is very important, especially when the processing needs to be postponed until other conditions are fulfilled.

Figure 8.3 Returning from called functions

A stack supports three basic operations: *push, pop,* and *peep (or peek).* The push operation adds an element at the end of the stack. The pop operation removes the last element from the stack. And, the peep operation returns the value of the last element of the stack (without deleting it). In Python, the list methods make it very easy to use a list as a stack. For example, to push an element in the stack, you will use the append( ) method, to pop an element use the pop( ) method, and for peep operation use the slicing operation as illustrated in the program given below.

Example 8.12 Program to illustrate operations on a stack

```
stack = [1,2,3,4,5,6]
print("Original stack is : ", stack)
stack.append(7)
print("Stack after push operation is : ", stack)
stack.pop()
print("Stack after pop operation is : ", stack)
last_element_index = len(stack) - 1
print("Value obtained after peep operation is : ",
stack[last_element_index])
```

OUTPUT

```
Original stack is :  [1, 2, 3, 4, 5, 6]
Stack after push operation is :  [1, 2, 3, 4, 5, 6, 7]
Stack after pop operation is :  [1, 2, 3, 4, 5, 6]
Value obtained after peep operation is :  6
```

Note The del statement and the pop( ) method does the same thing. The only difference between them is that pop( ) returns the removed item.

8.2.8 Using Lists as Queues

Queue is an important data structure which stores its elements in an ordered manner. For example, consider the analogies given below.

- People moving on an escalator. The people who got on the escalator first will be the first one to step out of it.
- People waiting for a bus. The first person standing in the line will be the first one to get into the bus.
- People standing outside the ticketing window of a cinema hall. The first person in the line will get the ticket first and thus will be the first one to move out of it.
- Luggage kept on conveyor belts. The bag which was placed first will be the first to come out at the other end.
- Cars lined at a toll bridge. The first car to reach the bridge will be the first to leave.

In all these examples, we see that the element at the first position is served first. Same is the case with queue data structure. A queue is a **FIFO** (First-In-First-Out) data structure in which the element that is inserted first is the first one to be taken out. The elements in a queue are added at one end and removed from the other end. In computer systems, the operating system makes full use of queues for the following tasks.

- To maintain waiting lists for a single shared resource like printer, disk, CPU, etc.
- To transfer data asynchronously (data not necessarily received at same rate as sent) between two processes (IO buffers), e.g., pipes, file IO, and sockets.
- As buffers on MP3 players and portable CD players, iPod playlist, etc.
- Handling interrupts. When programming a real-time system that can be interrupted, for example, by a mouse click, it is necessary to process the interrupts immediately; before proceeding with the current job. If the interrupts have to be handled in the order of arrival, then a FIFO queue is the appropriate data structure.
- Queues are also used in the playlist of jukebox to add songs to the end and play from the front of the list.

Queue supports three basic operations—*insert, delete,* and *peep (or peek)*. In Python, you can easily implement a queue by using the `append()` method to insert an element at the end of the queue, `pop()` method with an index `0` to delete the first element from the queue, and slice operation to print the value of the last the element in the queue. The program given below illustrates this concepts.

Example 8.13 Program to show the implementation of a queue using list data structure

```
queue = [1,2,3,4,5,6]
print("Original queue is : ", queue)
queue.append(7)
print("Queue after insertion is : ", queue)
queue.pop(0)
print("Queue after deletion is : ", queue)
print("Value obtained after peep operation is : ", queue[(len(queue) - 1)])
```

OUTPUT

```
Original queue is :  [1, 2, 3, 4, 5, 6]
Queue after insertion is :  [1, 2, 3, 4, 5, 6, 7]
Queue after deletion is :  [2, 3, 4, 5, 6, 7]
Value obtained after peep operation is :  7
```

8.2.9 List Comprehensions

Till now, we know that to create an empty list, we need to write, `List = []`. Just check how a list of cubes is created in the program given below.

Example 8.14 Program to make a list of cubes

```
cubes = []  # an empty list
for i in range(11):
    cubes.append(i**3)
print("Cubes of numbers from 1-10 : ", cubes)
```

OUTPUT

```
Cubes of numbers from 1-10 :  [0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

Note You can also create an empty list by using the built-in list type object. For example, by writing `L = list()`, an empty list `L` is created.

Python also supports computed lists called *list comprehensions* having the following syntax.

```
List = [expression for variable in sequence]
```

Where, the expression is evaluated once, for every item in the sequence.

List comprehensions help programmers to create lists in a concise way. This is mainly beneficial to make new lists where each element is the obtained by applying some operations to each member of another sequence or iterable. List comprehension is also used to create a subsequence of those elements that satisfy a certain condition.

Note An iterable is an object that can be used repeatedly in subsequent loop statements, say for example, for loop.

Example 8.15 Program to combine three lines of code into one

```
>>> cubes = [i**3 for i in range(11)]
>>> print(cubes)
```

OUTPUT

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729, 1000]
```

You can also use the list comprehension to combine the elements of two lists. For example, observe the code given below.

Example 8.16 Program to combine and print elements of two list using list comprehension

```
print([(x, y) for x in [10,20,30] for y in [30,10,40] if x != y])
```

OUTPUT

```
[(10, 30), (10, 40), (20, 30), (20, 10), (20, 40), (30, 10), (30, 40)]
```

In the code, two values, one from each list is used to create a new list only if the two values are not same.

8.2.10 Looping in Lists

Python's for and in constructs are extremely useful especially when working with lists. The for var in list statement is an easy way to access each element in a list (or any other sequence). For example, in the following code, the for loop is used to access each item in the list.

```
for i in list:
    print(i)
```

Example 8.17 Program to find the sum and mean of elements in a list

```
num_list = [1,2,3,4,5,6,7,8,9,10]
sum = 0
for i in num_list:
    sum += i
print("Sum of elements in the list = ", sum)
print("Average of elements in the list = ",
float(sum/float(len(num_list))))
```

OUTPUT

```
Sum of elements in the list =  55
Average of elements in the list =  5.5
```

Python offers multiple ways to access a list. Some of them are discussed in this section.

- **Using the enumerate() function:** This is used when you want to print both index as well as an item in the list. The enumerate() function returns an enumerate object which contains the index and value of all the items of the list as a tuple.

Example 8.18 Program to illustrate the use of enumerate() to print an individual item and its index in the list

```
num_list = [1,2,3,4,5]
for index, i in enumerate(num_list):
    print(i, " is at index : ", index)
```

OUTPUT

```
1  is a t index :  0
2  is a t index :  1
3  is a t index :  2
4  is a t index :  3
5  is a t index :  4
```

- **Using the range() function:** If you need to print index, then you can use the range() function as shown in the code given below.

Example 8.19 Program to print the index of values in a list

```
num_list = [1,2,3,4,5]
for i in range(len(num_list)):
    print("index : ", i)
```

Programming Tip: The index must be an integer. If you specify a non-integer number as the index, then TypeError will be generated

OUTPUT

```
index :  0
index :  1
index :  2
index :  3
index :  4
```

- **Using an iterator:** You can create an iterator using the built-in iter() function. The iterator is used to loop over the elements of the list. For this, the iterator fetches the value and then automatically points to the next element in the list when it is used with the next() method.

Example 8.20 Program to print the elements in the list using an iterator

```
num_list = [1,2,3,4,5]
it = iter(num_list)
for i in range(len(num_list)):
    print("Element at index ", i, " is : ", next(it))
```

OUTPUT

```
Element at index  0  is :  1
Element at index  1  is :  2
Element at index  2  is :  3
Element at index  3  is :  4
Element at index  4  is :  5
```

Note An iterator is often used to wrap an iterable and return each item of interest. All iterators are iterable, but all iterables are *not* iterators. An iterator can only be used in a single for loop, whereas an iterable can be used repeatedly in subsequent for loops.

8.3 FUNCTIONAL PROGRAMMING

Functional programming decomposes a problem into a set of functions. The map(), filter(), and reduce() functions which we will discuss in this section form a part of functional programming tools that work on all list items. However, it is recommended to use list comprehensions instead of these functions where possible.

8.3.1 filter() Function

The filter() function constructs a list from those elements of the list for which a function returns True. The syntax of the filter() function is given as,

```
filter(function, sequence)
```

As per the syntax, the `filter()` function returns a sequence that contains items from the sequence for which the function is True. If *sequence* is a string, Unicode, or a tuple, then the result will be of the same type; otherwise, it is always a list.

Example 8.21 Program to create a list of numbers divisible by 2 or 4 using list comprehension

```
def check(x):
    if (x % 2 == 0 or x % 4 == 0):
        return 1
# call check() for every value between 2 to 21
evens = list(filter(check, range(2, 22)))
print(evens)
```

Programming Tip: Do not add or remove elements from the list during iteration.

OUTPUT

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

From the output of the above program, we see that the `filter()` function returns `True` or `False`. Functions that return a *boolean* value are called *predicates*. Only those values in the range that are divisible by 2 or 4 are included in the newly created list.

8.3.2 `map()` Function

The `map()` function applies a particular function to every element of a list. Its syntax is same as the filter function.

```
map(function, sequence)
```

After applying the specified function on the sequence, the `map()` function returns the modified list. The `map()` function calls `function(item)` for each item in the sequence and returns a list of the return values.

Example 8.22 Program that adds 2 to every value in the list

```
def add_2(x):
    x += 2
    return x
num_list = [1,2,3,4,5,6,7]
print("Original List is : ", num_list)
new_list = list(map(add_2, num_list))
print("Modified List is : ", new_list)
```

OUTPUT

```
Original List is :  [1, 2, 3, 4, 5, 6, 7]
Modified List is :  [3, 4, 5, 6, 7, 8, 9]
```

Note that in the above code, the `map()` function calls `add_2()` which adds 2 to every value in the list. You can even pass more than one sequence in the `map()` function. But in this case, remember two things.

- First, the function must have as many arguments as there are sequences.
- Second, each argument is called with the corresponding item from each sequence (or `None` if one sequence is shorter than another).

Example 8.23 Program to pass more than one sequence to the `map()` function

```
def add(x,y):
    return x+y
list1 = [1,2,3,4,5]
list2 = [6,7,8,9,10]
list3 = list(map(add, list1, list2))
print("Sum of ", list1, " and ", list2, " = ", list3)
```

OUTPUT

```
Sum of  [1, 2, 3, 4, 5]  and  [6, 7, 8, 9, 10]  =  [7, 9, 11, 13, 15]
```

8.3.3 `reduce()` Function

The `reduce()` function with syntax as given below returns a single value generated by calling the function on the first two items of the sequence, then on the result and the next item, and so on.

```
reduce(function, sequence)
```

Example 8.24 Program to calculate the sum of values in a list using the `reduce()` function (Refer Figure 8.4)

```
import functools #functools is a module that contains the function reduce()
def add(x,y):
    return x+y
num_list = [1,2,3,4,5]
print("Sum of values in list = ")
print(functools.reduce(add, num_list))
```

OUTPUT

```
Sum of values in list =  15
```

Figure 8.4 `reduce()` function

Key points to remember

- If there is only one item in the sequence, then its value is returned.
- If the sequence is empty, an exception is raised.
- Creating a list in a very extensive range will generate a `MemoryError` or `OverflowError`. For example,

```
List = [5*i for i in range(100**100)]
```

When you execute the above statement, you will get the system overflow problem. Python window will stop responding and you will have to press `Ctrl+C` to come out of this state.

PROGRAMMING EXAMPLES

Program 8.1 Write a program that creates a list of numbers from 1–20 that are either divisible by 2 or divisible by 4 without using the filter function.

```
div_2_4 = []
for i in range(2, 22):
    if(i%2 == 0 or i%4 == 0):
        div_2_4.append(i)
print(div_2_4)
```

OUTPUT

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Program 8.2 Write a program using filter function to a list of squares of numbers from 1–10. Then use the for...in construct to sum the elements in the list generated.

```
def square(x):
    return(x**2)
squares = []
squares = list(filter(square, range(1, 11)))
print("List of squares in the range 1-10 = ", squares)
sum = 0
for i in squares:
    sum += i
print("Sum of squares in the range 1-10 = ", sum)
```

OUTPUT

```
List of squares in the range 1-10 =  [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
Sum of squares in the range 1-10 =  385
```

Program 8.3 Write a program the defines a list of countries that are a member of BRICS. Check whether a country is a member of BRICS or not.

```
country = ["Brazil", "India", "China", "Russia", "Sri Lanka"]
is_member = input("Enter the name of country : ")
if is_member in country:
    print(is_member, "has also joined BRICS")
else:
    print(is_member, " is not a member of BRICS")
```

OUTPUT

```
Enter the name of country : Pakistan
Pakistan is not a member of BRICS
```

Program 8.4 Write a program to create a list of numbers in the range 1 to 10. Then delete all the even numbers from the list and print the final list.

```
num_list = []
for i in range(1, 11):
    num_list.append(i)
print("Original List : ", num_list)
```



```
for index, i in enumerate(num_list):
    if(i%2==0):
        del num_list[index]
print("List after deleting even numbers : ",num_list)
```

OUTPUT

```
Original List :  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
List after deleting even numbers :  [1, 3, 5, 7, 9]
```

Program 8.5 Write a program to print index at which a particular value exists. If the value exists at multiple locations in the list, then print all the indices. Also, count the number of times that value is repeated in the list.

```
num_list = [1,2,3,4,5,6,5,4,3,2,1]
num = int(input("Enter the value to be searched : "))
i=0
count = 0
while i<len(num_list):
    if num == num_list[i]:
        print(num, " found at location", i)
        count += 1
    i += 1
print(num, " appears ", count, " times in the list")
```

OUTPUT

```
Enter the value to be searched : 4
4  found at location 3
4  found at location 7
4  appears  2  times in the list
list_words = []
```

Program 8.6 Write a program that creates a list of words by combining the words in two individual lists.

```
list_words = []
for x in ["Hello ", "World "]:
    for y in ["Python", "Programming"]:
        word = x + y
        list_words.append(word)
print("List combining the words in two individual lists is : ", list_words)
```

OUTPUT

```
List combining the words in two individual lists is :  ['Hello Python', 'Hello
Programming', 'World Python', 'World Programming']
```

Program 8.7 Write a program that forms a list of first character of every word in another list.

```
list1 = ["Hello", "Welcome", "To", "The", "World", "Of", "Python"]
letters = []
for word in list1:
    letters.append(word[0])
print(letters)
```

OUTPUT

```
['H', 'W', 'T', 'T', 'W', 'O', 'P']
```

Program 8.8 Write a program to remove all duplicates from a list.

```
num_list = [1,2,3,4,5,6,7,6,5,4]
print("Original List : ", num_list)
i=0
while i<len(num_list):
    num = num_list[i]
    for j in range(i+1, len(num_list)):
        val = num_list[j]
        if val == num:
            num_list.pop(j)
    i = i + 1
print("List after removing duplicates : ", num_list)
```

OUTPUT

```
Original List :  [1, 2, 3, 4, 5, 6, 7, 6, 5, 4]
List after removing duplicates :  [1, 2, 3, 4, 5, 6, 7]
```

Program 8.9 Write a program to create a list of numbers in the specified range in particular steps. Reverse the list and print its values.

```
num_list = []
m = int(input("Enter the starting of the range : "))
n = int(input("Enter the ending of the range : "))
o = int(input("Enter the steps in the range : "))
for i in range(m,n, o):
    num_list.append(i)
print("Original List :", num_list)
num_list.reverse()
print("Reversed List : ", num_list)
```

OUTPUT

```
Enter the starting of the range : 2
Enter the ending of the range : 30
Enter the steps in the range : 3
Original List : [2, 5, 8, 11, 14, 17, 20, 23, 26, 29]
Reversed List :  [29, 26, 23, 20, 17, 14, 11, 8, 5, 2]
```


Program 8.10 Write a program that creates a list of 10 random integers. Then create two lists—Odd List and Even List that has all odd and even values in the list respectively.

```
import random
num_list = []
for i in range(10):
    val = random.randint(1, 100)
    num_list.append(val)
print("Original List : ", num_list)
even_list = []
odd_list = []
for i in range(len(num_list)):
    if(num_list[i] % 2 == 0):
        even_list.append(num_list[i])
    else:
        odd_list.append(num_list[i])
print("Even Numbers List = ", even_list)
print("Odd Numbers List = ", odd_list)
```

OUTPUT

```
Original List :  [93, 27, 9, 68, 68, 88, 14, 33, 64, 21]
Even Numbers List =  [68, 68, 88, 14, 64]
Odd Numbers List =  [93, 27, 9, 33, 21]
```

Program 8.11 Write a program to create a list of first 20 odd numbers using the shortcut method.

```
odd = [2*i + 1 for i in range(20)]
print(odd)
```

OUTPUT

```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39]
```

Program 8.12 Write a program that passes a list to a function that scales each element in the list by a factor of 10. Print the list values at different stages to show that changes made to one list is automatically reflected in the other list.

```
def change(list1):
    for i in range(len(list1)):
        list1[i] = list1[i] * 10
    print("After change in function, List is : ", list1)
num_list = [1,2,3,4,5,6]
print("Original List is : ", num_list)
change(num_list)
print("List after change is : ", num_list)
```

Programming Tip: Creating a list in a very extensive range will result in a **OverflowError**. This can be corrected by using generators.

OUTPUT

```
Original List is :  [1, 2, 3, 4, 5, 6]
After change in function, List is :  [10, 20, 30,
40, 50, 60]
List after change is :  [10, 20, 30, 40, 50, 60]
```

Program 8.13 Write a program that has a list of both positive and negative numbers. Create another list using `filter()` that has only positive values.

```
def is_positive(x):
    if x>=0:
        return x
num_list = [10, -20, 30, -40, 50, -60, 70, -80, 90, -100]
List = []
List = list(filter(is_positive, num_list))
print("Positive Values List = ",List)
```

OUTPUT

```
Positive Values List =  [10, 30, 50, 70, 90]
```

Program 8.14 Write a program that converts strings of all uppercase characters into strings of all lowercase characters using the `map()` function.

```
def to_lower(str):
    return str.lower()
list1 = ["HELLO", "WELCOME", "TO", "PYTHON"]
list2 = list(map(to_lower,list1))
print("List in lowercase characters is : ", list2)
```

OUTPUT

```
List in lowercase characters is :  ['hello', 'welcome', 'to', 'python']
```

Program 8.15 Write a program using `map()` function to create a list of squares of numbers in the range 1–10.

```
def squares(x):
    return x*x
sq_list = list(map(squares, range(1,11)))
print("List of squares from 1-10 : ", sq_list)
```

OUTPUT

```
List of squares from 1-10 :  [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Program 8.16 Write a program to combine values in two lists using list comprehension. Combine only those values of a list that are multiples of values in the first list.


```
print([(x, y) for x in [10,20,30,50] for y in [35,40,55,60] if y % x == 0 or x%y == 0])
```

OUTPUT

```
[(10, 40), (10, 60), (20, 40), (20, 60), (30, 60)]
```

Program 8.17 Write a program that converts a list of temperatures in Celsius into Fahrenheit.

```
def convert_to_F(Temp_C):
    return ((float(9)/5)*Temp_C + 32)
Temp_in_C = (36.5, 37, 37.5,39)
Temp_in_F = list(map(convert_to_F, Temp_in_C))
print("List of temperatures in Celsius : ", Temp_in_C)
print("List of temperatures in Fahrenheit : ", Temp_in_F)
```

OUTPUT

```
List of temperatures in Celsius :  (36.5, 37, 37.5, 39)
List of temperatures in Fahrenheit :  [97.7, 98.60000000000001, 99.5, 102.2]
```

Program 8.18 Write a program to find largest value in a list using `reduce()` function.

```
import functools
def max_ele(x,y):
    return x>y
num_list = [4,1,8,2,9,3,0]
print("Largest value in the list is : ", functools.reduce(max, num_list))
```

OUTPUT

```
Largest value in the list is :  9
```

Program 8.19 Write a program that has a list of functions that scales a number by a factor of 2, 3, and 4. Call each function in the list on a given number.

```
L = [lambda x: x * 2,lambda x: x * 3, lambda x: x * 4]
for f in L:
    print(f(5))
print("\n Multiplying the value of 100 by 2 we get : ",(L[0](100)))
```

OUTPUT

```
10 15 20
Multiplying the value of 100 by 2 we get :  200
```

Program 8.20 Write a program to generate in the Fibonacci sequence and store it in a list. Then find the sum of the even-valued terms.

```
a = 0
b = 1
n = int(input("Enter the number of terms : "))
i=2
```

```
List = [a,b]
while i<n:
    s = a + b
    List.append(s)
    a = b
    b = s
    i += 1
print(List)
i=0
sum = 0
while i<n:
    sum += List[i]
    i += 2
print("SUM = ", sum)
```

OUTPUT

```
Enter the number of terms : 10
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
SUM =  33
```

Program 8.21 Write a program to add two matrices (using nested lists).

```
X = [[2,5,4],
     [1 ,3,9],
     [7 ,6, 2]]
Y = [[1,8,5],
     [7,3,6],
     [4,0,9]]
result = [[0,0,0],
          [0,0,0],
          [0,0,0]]
for i in range(len(X)):
    for j in range(len(X[0])):
        result[i][j] = X[i][j] + Y[i][j]
for r in result:
    print(r)
```

OUTPUT

```
[3, 13, 9]
[8, 6, 15]
[11, 6, 11]
```

Program 8.22 Write a program to find the median of a list of numbers.

```
List = []
n = int(input("Enter the number of elements to be inserted in the list : "))
for i in range(n):
```



```
    print("Enter number ", i + 1, " : ")
    num = int(input())
    List.append(num)
print("Sorted List is......")
List = sorted(List)
print(List)
i = len(List) - 1
if n%2 != 0:
    print("MEDIAN = ", List[i//2])
else:
    print("MEDIAN = ", (List[i//2] + List[i+1//2])/2)
```

OUTPUT

```
Enter the number of elements to be inserted in the list : 6
Enter number  1  :  2
Enter number  2  :  9
Enter number  3  :  1
Enter number  4  :  7
Enter number  5  :  4
Enter number  6  :  8
Sorted List is......
[1, 2, 4, 7, 8, 9]
MEDIAN =  6.5
```

Program 8.23 Write a program to calculate distance between two points.

```
import math
p1 = []
p2 = []
x1 = int(input("Enter the x co-ordinate of starting point : "))
y1 = int(input("Enter the y co-ordinate of starting point : "))
x2 = int(input("Enter the x co-ordinate of ending point : "))
y2 = int(input("Enter the y co-ordinate of ending point : "))
p1.append(x1)
p1.append(x2)
p2.append(x2)
p2.append(y2)
distance = math.sqrt( ((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2) )
print("DISTANCE = %f" %distance)
```

OUTPUT

```
Enter the x co-ordinate of starting point : 2
Enter the y co-ordinate of starting point : 4
Enter the x co-ordinate of ending point : 7
Enter the y co-ordinate of ending point : 9
DISTANCE = 5.385165
```

8.4 TUPLE

Like lists, tuple is another data structure supported by Python. It is very similar to lists but differs in two things.

- First, a tuple is a sequence of ***immutable*** objects. This means that while you can change the value of one or more items in a list, you cannot change the values in a tuple.
- Second, tuples use parentheses to define its elements whereas lists use square brackets.

8.4.1 Creating Tuple

Creating a tuple is very simple and almost similar to creating a list. For creating a tuple, generally you need to just put the different comma-separated values within a parentheses as shown below.

`Tup1 = (val 1, val 2,...)`, where `val` (or values) can be an integer, a floating number, a character, or a string. Consider the following examples and observe their outputs.

Example 8.25 Programs to show how to create the different types of tuples

```
Tup1 = ()   # Creates an empty tuple
print(Tup1)
```

OUTPUT

```
()
```

```
Tup1 = (5)   # Creates a tuple with a single element
print(Tup1)
```

OUTPUT

```
5
```

```
Tup1 = (1,2,3,4,5)   # Creates a tuple of integers
print(Tup1)
Tup2 = ('a','b','c','d')    # Creates a tuple of characters
print(Tup2)
Tup3 = ("abc","def","ghi")  #Creates a tuple of strings
print(Tup3)
Tup4 = (1.2,2.3,3.4,4.5)  #Creates a tuple of floating point numbers
print(Tup4)
Tup5 = (1,"abc",2.3,'d')  #Creates a tuple of mixed values
print(Tup5)
```

OUTPUT

```
(1, 2, 3, 4, 5)
('a', 'b', 'c', 'd')
('abc', 'def', 'ghi')
(1.2, 2.3, 3.4, 4.5)
(1, 'abc', 2.3, 'd')
```

Key points to remember

- Any set of multiple, comma-separated values written without an identifying symbol like brackets [ ] (because it specifies a list) and parentheses ( ) (for tuples), etc., are treated as tuples by default. Some examples of such tuples are given below.


```
# Tuple with parentheses
print('a', "bcd", 2, 4.6)
```

OUTPUT

```
a bcd 2 4.6
```

```
# Default tuple without parentheses
a,b = 10, 20
print(a,b)
```

OUTPUT

```
10 20
```

- If you want to create a tuple with a single element, then you must add a comma after the element. In the absence of a comma, Python treats the element as an ordinary data type.

Example 8.26 Programs to demonstrate the necessity of having a comma in the tuple

```
Tup = (10,) # comma after first element
print(type(Tup))
```

OUTPUT

```
<type 'tuple'>
```

```
Tup = (10) # comma missing
print(type(Tup))
```

OUTPUT

```
<type 'int'>
```

8.4.2 Utility of Tuples

In real-world applications, tuples are extremely useful for representing records or structures as we call in other programming languages. These structures store related information about a subject together. The information belongs to different data types. For example, a tuple that stores information about a student can have elements like `roll_no`, `name`, `course`, `total_marks`, `avg`, etc. If you carefully observe, these individual elements can have different data types. For example, `roll_no` can be an integer or an alphanumeric value, `name` and `course` will of course be `string`, and `total_marks` and `avg` can be floating point numbers.

Some built-in functions return a tuple. For example, the `divmod()` function returns two values—quotient as well as the remainder after performing the divide operation.

Example 8.27 Program to illustrate the use of `divmod()` function

```
quo, rem = divmod(100,3)
print("Quotient = ",quo)
print("Remainder = ", rem)
```

OUTPUT

```
Quotient =  33
Remainder =  1
```

8.4.3 Accessing Values in a Tuple

Like other sequences (strings and lists) covered so far, indices in a tuple also starts at `0`. You can even perform operations like slice, concatenate, etc. on a tuple. For example, to access values in tuple, slice operation is used along with the index or indices to obtain value stored at that index.

Example 8.28 Program to illustrate the use of slice operation to retrieve value(s) stored in a tuple

```
Tup1 = (1,2,3,4,5,6,7,8,9,10)
print("Tup[3:6] = ", Tup1[3:6])
print("Tup[:8] = ", Tup1[:4])
print("Tup[4:] = ", Tup1[4:])
print("Tup[:] = ", Tup1[:])
```

OUTPUT

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

8.3.4 Updating Tuple

We have already learnt that tuple is immutable and so, the value(s) in the tuple cannot be changed. You can only extract values from a tuple to form another tuple.

Example 8.29 Program to extract values from a tuple

```
Tup1 = (1,2,3,4,5)
Tup2 = (6,7,8,9,10)
Tup3 = Tup1 + Tup2
print(Tup3)
```

OUTPUT

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

8.4.5 Deleting Elements in Tuple

Since tuple is an immutable data structure, you cannot delete value(s) from it. Of course, you can create a new tuple that has all elements in your tuple except the ones you don't want (those you wanted to be deleted). Observe the code given in the following example and note the error generated.

Example 8.30 Program to illustrate that tuples are immutable

```
Tup1 = (1,2,3,4,5)
del Tup1[3]
print(Tup1)
```

OUTPUT

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2, in <module>
    del Tup1[3]
TypeError: 'tuple' object doesn't support item deletion
```


However, you can always delete the entire tuple by using the del statement. This is done in the code given below.

Example 8.31 Program to delete a tuple

```
Tup1 = (1,2,3,4,5)
del Tup1
print(Tup1)
```

OUTPUT

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 3, in <module>
    print Tup1
NameError: name 'Tup1' is not defined
```

Note that this exception is raised because you are now trying to print a tuple that has already been deleted.

8.4.6 Basic Tuple Operations

Like strings and lists, you can also perform operations like concatenation, repetition, etc. on tuples. The only difference is that a new tuple should be created when a change is required in an existing tuple. Table 8.3 summarizes some operations on tuples.

Table 8.3 Operations on Tuples

Operation	Expression	Output
Length	len((1,2,3,4,5,6))	6
Concatenation	(1,2,3) + (4,5,6)	(1, 2, 3, 4, 5, 6)
Repetition	("Good..")*3	"Good..Good..Good.."
Membership	5 in (1,2,3,4,5,6,7,8,9)	True
Iteration	for i in (1,2,3,4,5,6,7,8,9,10): print(i,end=" ")	1,2,3,4,5,6,7,8,9,10
Comparison (Use >, <, ==)	Tup1 = (1,2,3,4,5) Tup2 = (1,2,3,4,5) print(Tup1>Tup2)	False
Maximum	max(1,0,3,8,2,9)	9
Minimum	min(1,0,3,8,2,9)	0
Convert to tuple (converts a sequence into a tuple)	tuple("Hello") tuple([1,2,3,4,5])	("H", "e", "l", "l", "o") (1, 2, 3, 4, 5)

8.4.7 Tuple Assignment

Tuple assignment is a very powerful feature in Python. It allows a tuple of variables on the left side of the assignment operator to be assigned values from a tuple given on the right side of the assignment operator. Each value is assigned to its respective variable.

In case, an expression is specified on the right side of the assignment operator, first that expression is evaluated and then assignment is done. This feature makes tuple assignment quite versatile. Look at the code given below.

Example 8.32 Program to show the different ways of tuple assignment

```
# an unnamed tuple of values assigned to values of another unnamed tuple
(val1, val2, val3) = (1,2,3)
print(val1, val2, val3)
Tup1 = (100, 200, 300)
(val1, val2, val3) = Tup1    # tuple assigned to another tuple
print(val1, val2, val3)
# expressions are evaluated before assignment
(val1, val2, val3)= (2+4, 5/3 + 4, 9%6)
print(val1, val2, val3)
```

OUTPUT

```
1 2 3
100 200 300
6 5.666667 3
```

Note that while assigning values to a tuple, you must ensure that number of values on both the sides of the assignment operator are same otherwise, an error will be generated as shown below.

```
(val1, val2, val3) = (1,2)
print(val1, val2, val3)
```

OUTPUT

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 1, in <module>
    (val1, val2, val3) = (1,2)
ValueError: need more than 2 values to unpack (expected 3, got 2)
```

8.4.8 Tuples for Returning Multiple Values

We have learnt that a function can return only a single value. But at times, we need to return more than one value from a function. In such situations, it is preferable to group together multiple values and return them together.

Example 8.33 Program to return the highest as well as the lowest values in the list

```
def max_min(vals):
    x = max(vals)
    y = min(vals)
    return (x,y)
vals = (99, 98, 90, 97, 89, 86, 93, 82)
(max_marks, min_marks) = max_min(vals)
print("Highest Marks = ", max_marks)
print("Lowest Marks = ", min_marks)
```

Programming Tip: You can't delete elements from a tuple. Methods like `remove()` or `pop()` do not work with a tuple.

OUTPUT

```
Highest Marks =  99
Lowest Marks =  82
```

Note Unlike lists, tuples do not support `remove()`, `pop()`, `append()`, `sort()`, `reverse()`, and `insert()` methods.

8.3.9 Nested Tuples

Python allows you to define a tuple inside another tuple. This is called a *nested tuple*. Consider the program code given below. We have a list of students who have topped in their respective courses. We store the name, course, and aggregate of three students as tuples inside the tuple `Toppers`.

Example 8.34 Program to demonstrate the use of nested tuples

```
Toppers = (("Arav", "BSc",92.0), ("Chaitanya", "BCA", 99.0),
("Dhruvika", "Btech", 97))
for i in Toppers:
    print(i)
```

OUTPUT

```
('Arav', 'BSc', 92.0)
('Chaitanya', 'BCA', 99.0)
('Dhruvika', 'Btech', 97)
```

- You can even specify a list within a tuple. The code is given below.

Example 8.35 Program to print the name of the topper and her marks in 4 subjects wherein the marks are specified as a list in the tuple `Topper`

```
Topper = ("Janvi",[94, 95, 96, 97])
print("Class Topper : ", Topper[0])
print("Highest Scores in 4 Subjects : ", Topper[1:])
```

OUTPUT

```
Class Topper :  Janvi
Highest Scores in 4 Subjects :  ([94, 95, 96, 97],)
```

8.4.10 Checking the Index: `index()` method

The index of an element in the tuple can be obtained by using the `index()` method. If the element being searched is not present in the list, then error is generated. The syntax of `index()` is given as,

```
list.index(obj)
```

where, `obj` is the object to be found out.

Consider the examples given below.

Example 8.36 Program to demonstrate the use of `index()` method

```
Tup = (1,2,3,4,5,6,7,8)
print(Tup.index(4))
```

OUTPUT

```
3
```

Example 8.37 Program to print the location at which an element is present in the list using the `index()` method

```
students = ("Bhavya", "Era", "Falguni","Huma")
index = students.index("Falguni")
print("Falguni is present at location : ", index)
index = students.index("Isha")
print("Isha is present at location : ", index)
```

OUTPUT

```
Falguni is present at location :  2
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 4, in <module>
    index = students.index("Isha")
ValueError: tuple.index(x): x not in tuple
```

8.4.11 Counting the Elements: count() Method

The `count()` method is used to return the number of elements with a specific value in a tuple.

Example 8.38 Program to count the number of times letter `'x'` appears in the specified string

```
tup = "abcdxxxabcdxxxabcdxxx"
print("x appears ", tup.count('x'), " times in ", tup)
```

OUTPUT

```
x appears  9  times in  abcdxxxabcdxxxabcdxxx
```

8.4.12 List Comprehension and Tuples

We have already studied list comprehension to create a new list in the earlier sections of this chapter. You can use the same concept to manipulate the values in one tuple to create a new tuple.

Example 8.39 Consider the program given below passes a tuple as an argument to a function `double()`. The function scales each value in the tuple by a factor of two and places the scaled values in another tuple.


```
def double(T):
    return ([i*2 for i in T])
Tup = 1,2,3,4,5
print("Original Tuple : ", Tup)
print("Double Values: ",double(Tup))
```

OUTPUT

```
Original Tuple :  (1, 2, 3, 4, 5)
Double Values:  [2, 4, 6, 8, 10]
```

Note If a sequence is specified without parenthesis, it is treated to be a tuple by default.

8.4.13 Variable-length Argument Tuples

Many built-in functions like `max()`, `min()`, `sum()`, etc. use variable-length arguments since these functions themselves do not know how many arguments will be passed to them. Variable-length arguments tuple is a striking feature in Python. It allows a function to accept a variable (different) number of arguments. This is especially useful in defining functions that are applicable to a large variety of arguments. For example, if you have a function that displays all the parameters passed to it, then even the function does not know how many values it will be passed. In such cases, we use a variable-length argument that begins with a `'*'` symbol. Any argument that starts with a `'*'` symbol is known as ***gather*** and specifies a variable-length argument.

Example 8.40 Program to manipulate efficiently each value that is passed to the tuple using variable-length arguments

```
def display(*args):
    print(args)
Tup = (1,2,3,4,5,6)
display(Tup)
```

OUTPUT

```
((1, 2, 3, 4, 5, 6),)
```

The opposite of gather is ***scatter***. So, in case you have a function that accepts multiple arguments but not a tuple, then the tuple is scattered to pass individual elements. Look at the code given below which demonstrates this concept.

Example 8.41 Programs to illustrate scatter in terms of Tuple

```
Tup = (56, 3)
quo, rem = divmod(Tup)
print(quo, rem)
```

```
Tup = (56, 3)
#values are now scattered and passed
quo, rem = divmod(*Tup)
print(quo, rem)
```

```
OUTPUT
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2, in
<module>
    quo, rem = divmod(Tup)
TypeError: divmod expected 2 arguments,
got 1
```

```
OUTPUT
18      2
```

Programming Tip: If a negative value is used for the step, the slice is done backwards.

In the first code, only Tup was passed (a single argument) but the `divmod()` expects two arguments, hence, the error occurs. While in the second code, the symbol * denotes that there may be more than one argument. So Python, extracts the values (scatters them) to obtain two values on which the operation can be applied.

8.4.14 The `zip()` Function

`zip()` is a built-in function that takes two or more sequences and "zips" them into a list of tuples. The tuple thus, formed has one element from each sequence. The code given below illustrates this concept.

Example 8.42 Program to show the use of `zip()` function

```
Tup = (1,2,3,4,5)
List1 = ['a','b','c','d','e']
print(list((zip(Tup, List1))))
```

```
OUTPUT
[(1, 'a'), (2, 'b'), (3, 'c'), (4, 'd'), (5, 'e')]
```

From the output, we see that the result of `zip()` function is a list of tuples where each tuple contains a character from the list and an integer from the tuple. In the above example, there are equal number of values in the list and tuple. But if the two sequences have different length, then the result has the length of the shorter one as illustrated in the code given below.

Example 8.43 Program to use `zip()` function on variable-length sequences

```
Tup = (1,2,3)
List1 = ['a','b','c','d','e']
print(list(zip(Tup, List1)))
```

```
OUTPUT
[(1, 'a'), (2, 'b'), (3, 'c')]
```

- You can even print the elements in a tuple using the `for` statement as shown below.

Example 8.44 Program to print elements in a tuple using `for` loop

```
Tup = ((1, 'a'), (2, 'b'), (3, 'c'))
for i, char in Tup:
    print(i, char)
```

Programming Tip: Slicing can be done on tuples.

```
OUTPUT
1 a
2 b
3 c
```


- To traverse the elements of a sequence and also print their indices, use the built-in function `enumerate()`.

Example 8.45 Program that uses `enumerate()` function to print elements as well as their indices

```
for index, element in enumerate('ABCDEFG'):
    print(index, element)
```

```
OUTPUT
0 A
1 B
2 C
3 D
4 E
5 F
6 G
```

Key points to remember

- Tuples can be converted into lists, and vice versa using the built-in `tuple()` function that takes a list and returns a tuple with the same elements. Similarly, the `list()` function takes a tuple and returns a list.
- You cannot divide or subtract tuples. If you try to do so you will get a `TypeError` with `"unsupported operand type"`.
- Since tuples are immutable, they do not support methods like `sort()` and `reverse()`, as these methods modify existing lists. However, Python has a built-in function `sorted()` which takes any sequence as a parameter and returns a new list with the same elements but in a different order. For example, the code given in the following example illustrates this concept.

Programming Tip: Tuples are faster than lists, but they cannot be changed.

Example 8.46 Program to sort a tuple of values

```
Tup = (5,1,0,2,8,3,9)
print(sorted(Tup))
```

```
OUTPUT
[0, 1, 2, 3, 5, 8, 9]
```

If you write,

```
tup = ("abc","def")
x,y = tup
```

```
print(x, y)
```

OUTPUT

```
abc def
```

then syntactically, it works as assigning x with value `tup[0]` and y with value `tup[1]`.

- You can use string formatting feature to print values in the Tuple. This is shown in the code given below.

Example 8.47 Program to illustrate string formatting function with tuple

```
Tup = ("Heena", 89, 82,4)
print("%s got %d marks in CSA and her aggregate was
%.2f" %(Tup[0], Tup[1], Tup[2]))
```

Programming Tip: Reassigning a value in a tuple causes a TypeError.

OUTPUT

```
Heena got 89 marks in CSA and her aggregate was 82.00
```

8.4.16 Advantages of Tuple over List

Although tuples are similar to lists, there are some advantages of implementing a tuple over a list. Some of these advantages are listed below.

- Tuples are used to store values of different data types. Lists can however, store data of similar data types.
- Since tuples are immutable, iterating through tuples is faster than iterating over a list. This means that a tuple performs better than a list.
- Tuples can be used as key for a dictionary but lists cannot be used as keys. We will learn about dictionaries in the next section.
- Tuples are best suited for storing data that is write-protected (you can read the data but cannot write to it).
- Tuples can be used in place of lists where the number of values is known and small.
- If you are passing a tuple as an argument to a function, then the potential for unexpected behavior due to aliasing gets reduced.
- Multiple values from a function can be returned using a tuple.
- Tuples are used to format strings.

Program 8.24 Write a program to swap two values using tuple assignment.

```
val1 = 10
val2 = 20
print "val1 = ",val1, " val2 = ",val2
(val1,val2) = (val2,val1)
print("val1 = ",val1, " val2 = ",val2)
```

OUTPUT

```
val1 =  10  val2 =  20
val1 =  20  val2 =  10
```


Program 8.25 Write a program using a function that returns the area and circumference of a circle whose radius is passed as an argument.

```
PI = 3.14
def cal_a_r(r):
    return (PI*r*r, 2*PI*r)
radius = float(input("Enter the radius : "))
(area, circumference) = cal_a_r(radius)
print("Area of the circle with radius", radius, " = ", area)
print("Circumference of the circle with radius ", radius, " = ", circumference)
```

OUTPUT

```
Enter the radius : 7
Area of the circle with radius 7.0  =  153.86
Circumference of the circle with radius  7.0  =  43.96
```

Program 8.26 Write a program that has a nested list to store toppers details. Edit the details and reprint the details.

```
Toppers = (("Arav", "BSc",92.0), ("Chaitanya", "BCA",
99.0), ("Dhruvika", "Btech", 97))
for i in Toppers:
    print(i)
choice = input("Do you want to edit the details : ")
if choice == 'y':
    name = input("Enter the name of the students whose details are to be
edited : ")
    new_name = input("Enter the correct name : ")
    new_course = input("Enter the correct course : ")
    new_aggr = input("Enter the correct aggregate : ")
    i = 0
    new_Toppers = ()
    while i<len(Toppers):
        if Toppers[i][0] == name:
            new_Toppers += (new_name, new_course, new_aggr)
        else:
            new_Toppers += Toppers[i]
        i+=1
for i in new_Toppers:
    print(i,end = ' ')
```

Programming Tip: You cannot add elements to a tuple. Methods like `append()` or `extend()` does not work with tuple.

OUTPUT

```
('Arav', 'BSc', 92.0)
('Chaitanya''Chiatanya', 'BCA', 99.0)
('Dhruvika', 'Btech', 97)
Do you want to edit the details : y
Enter the name of the students whose details are to be edited : Chaitanya
Enter the correct name : Chaitanya
Enter the correct course : BCA
```

```
Enter the correct aggregate : 100
Arav BSc 92.0 Chaitanya BCA 100 Dhruvika Btech 97
```

Program 8.27 Write a program that scans an email address and forms a tuple of user name and domain.

```
addr = 'abc@gmail.com'
user_name, domain_name = addr.split('@')
print ("User Name : ", user_name)
print("Domain Name : ", domain_name)
```

OUTPUT

```
User Name :  abc
Domain Name :  gmail.com
```

Program 8.28 Write a program that has a list of numbers (both positive as well as negative). Make a new tuple that has only positive values from this list.

```
Tup = (-10,1,2,-9,3,4,-8,5,6)
newTup = ()
for i in Tup:
    if i>0:
        newTup += (i,)
print(newTup)
```

OUTPUT

```
(1, 2, 3, 4, 5, 6)
```

Program 8.29 Write a program that accepts different number of arguments and return sum of only the positive values passed to it.

```
def sum_pos(*args):
    tot = 0
    for i in args:
        if i>0:
            tot += i
    return tot
print("sum_pos(1,-9,2,-8,3,-7,4,-6,5) = ", sum_pos(1,-9,2,-8,3,-7,4,-6,5))
```

OUTPUT

```
sum_pos(1,-9,2,-8,3,-7,4,-6,5) =  15
```

Program 8.30 Write a program that has two sequences. First which stores some questions and second stores the corresponding answers. Use the `zip()` function to form a valid question answer series.

```
Ques = ["Roll_No", "Name", "Course"]
Ans = [7, "Saesha", "BSc"]
for q,a in zip(Ques, Ans):
    print("What is your", q, "?")
    print("My", q, "is : ", a)
```

Programming Tip: If the index specified in the tuple slice is too big, then an IndexError exception is raised.

OUTPUT

```
What is your Roll_No ?
My Roll_No is :  2
What is your Name ?
My Name is :  Saesha
What is your Course ?
My Course is :  BSc
```


8.5 SETS

Sets is another data structure supported by Python. Basically, sets are same as lists but with a difference that sets are lists with no duplicate entries. Technically, a set is a mutable and an unordered collection of items. This means that we can easily add or remove items from it.

8.5.1 Creating a Set

A set is created by placing all the elements inside curly brackets {}, separated by comma or by using the built-in function `set()`. The syntax of creating a set can be given as,

```
set_variable = {val1, val2, ...}
```

For example, to create a set you can write,

```
>>> s = {1,2.0,"abc"}
>>> print( s)
set([1, 2.0,'abc'])
```

A set can have any number of items and they may be of different data types. For example, the code given below creates a set using the `set()` function. The code converts a list of different types of values into a set.

Example 8.48 Program to convert a list of values into a set

```
s = set([1,2,'a','b',"def",4.56])
print(s)
```

Programming Tip: If we add the same element multiple times in a set, they are removed because a set cannot have duplicate values.

OUTPUT

```
set(['a', 1, 2, 'b', 4.56, 'def'])
```

The program given below demonstrates different ways of creating sets, that is, sets being created by using a list, tuples or string.

Example 8.49 Program to create a set

```
List1 = [1,2,3,4,5,6,5,4,3,2,1]
print(set(List1))  # list is converted into a set
```

```
Tup1 = ('a','b','c','d','b','e','a')
print(set(Tup1))# tuple is converted into a set
str = "abcdefabcdefg"
print(set(str))    # string is converted into a set
# forms a set of words
print(set("She sells sea shells on the sea shore".split()))
```

OUTPUT

```
set([1, 2, 3, 4, 5, 6])
set(['a', 'c', 'b', 'e', 'd'])
set(['a', 'c', 'b', 'e', 'd', 'g', 'f'])
set(['on', 'shells', 'shore', 'She', 'sea', 'sells', 'the'])
```

Like in case of mathematical sets, Python sets are also a powerful tool as they have the ability to calculate differences and intersections between other sets. Figures 8.5(a–d) given below demonstrate various set operations.

Union of two sets includes all elements from both the sets

Intersection of two sets includes elements that are common to both the sets

Difference of A and B (A–B) includes all elements that are in A but not in B

Symmetric difference of A and B includes all elements that are in A and B but not the one that are common to A and B

Figure 8.5 Set operations

Example 8.50 Program to find intersection, union, and symmetric difference between two sets

```
Coders = set(["Arnav","Goransh","Mani","Parul"])
Analysts = set(["Krish","Mehak","Shiv","Goransh","Mani"])
print("Coders : ", Coders)
print("Analysts : ", Analysts)
print("People working as Coders as well as Analysts :
", Coders.intersection(Analysts))
print("People working as Coders or Analysts :
", Coders.union(Analysts))
print("People working as Coders but not Analysts :
```



```
", Coders.difference(Analysts))
print("People working as Analysts but not Coders :
", Analysts.difference(Coders))
print("People working in only one of the groups :
", Coders.symmetric_difference(Analysts))
```

OUTPUT

```
Coders :  {'Arnav', 'Mani', 'Goransh', 'Parul'}
Analysts :  {'Shiv', 'Mani', 'Krish', 'Goransh', 'Mehak'}
People working as coders as well as Analysts :  {'Mani', 'Goransh'}
People working as Coders or Analysts :  {'Goransh', 'Parul', 'Krish', 'Mehak',
'Shiv', 'Arnav', 'Mani'}
People working as Coders but not Analysts : {'Arnav', 'Parul'}
People working as Analysts but not Coders : {'Shiv', 'Krish', 'Mehak'}
People working in only one of the groups : {'Krish', 'Mehak', 'Shiv', 'Arnav', 'Parul'}
```

Some other operations that can be performed on sets are discussed in Table 8.4. But before going through this table, just remember the following points.

Programming Tip: The `update()` method can take tuples, lists, strings, or other sets as its argument.

- Two sets are equal if and only if every element in each set is contained in the other.
- A set is less than another set if and only if the first set is a subset of the second set.
- A set is greater than another set if and only if the first set is a superset of the second set.

Table 8.4 **Set Operations**

Operation	Description	Code	Output

`s.update(t)`	Adds elements of set t in the set s provided that all duplicates are avoided	`s = set([1,2,3,4,5])`
		`t = set([6,7,8])`
		`s.update(t)`
		`print(s)`
			`(1,2,3,4,5,6,7,8)`

`s.add(x)`	Adds element x to the set s provided that all duplicates are avoided	`s = set([1,2,3,4,5])`
		`s.add(6)`
		`print(s)`
			`(1,2,3,4,5,6)`

`s.remove(x)`	Removes element x from set s. Returns KeyError if x is not present	`s = set([1,2,3,4,5])`
		`s.remove(3)`
		`print(s)`
			`(1,2,4,5)`

`s.discard(x)`	Same as `remove()` but does not give an error if x is not present in the set	`s = set([1,2,3,4,5])`
		`s.discard(3)`
		`print(s)`
			`(1,2,4,5)`

`s.pop()`	Removes and returns any arbitrary element from s. KeyError is raised if s is empty	`s = set([1,2,3,4,5])`
		`s.pop()`
		`print(s)`
			`(2,3,4,5)`

`s.clear()`	Removes all elements from the set	`s = set([1,2,3,4,5])`
		`s.clear()`
		`print(s)`
			`set()`

Contd

Table 8.4 *Contd*

Operation	Description	Code	Output
len(s)	Returns the length of set	s = set([1,2,3,4,5]) print(len(s))	5
x in s	Returns True is x is present in set s and False otherwise	s = set([1,2,3,4,5]) print(3 in s)	True
x not in s	Returns True is x is not present in set s and False otherwise	s = set([1,2,3,4,5]) print(6 not in s)	True
s.issubset(t) or s<=t	Returns True if every element in set s is present in set t and False otherwise	s = set([1,2,3,4,5]) t = set([1,2,3,4,5,6,7,8,9,10]) print(s<=t)	True
s.issuperset(t) or s>=t	Returns True if every element in t is present in set s and False otherwise	s = set([1,2,3,4,5]) t = set([1,2,3,4,5,6,7,8,9,10]) print(s.issuperset(t))	False
s.union(t) or s|t	Returns a set s that has elements from both sets s and t	s = set([1,2,3,4,5]) t = set([1,2,3,4,5,6,7,8,9,10]) print(s|t)	(1,2,3,4,5,6,7,8,9,10)
s.intersection(t) or s&t	Returns a new set that has elements which are common to both the sets s and t	s = set([1,2,3,4,5]) t = set([1,2,3,4,5,6,7,8,9,10]) z = s&t print(z)	(1,2,3,4,5)
s.intersection_update(t)	Returns a set that has elements which are common to both the sets s and t	s = set([1,2,10,12]) t = set([1,2,3,4,5,6,7,8,9,10]) s.intersection_update(t) print(s)	(1,2,10)
s.difference(t) or s-t	Returns a new set that has elements in set s but not in t	s = set([1,2,10,12]) t = set([1,2,3,4,5,6,7,8,9,10]) z = s-t print(z)	(12)
s.difference_update(t)	Removes all elements of another set from this set	s = set([1,2,10,12]) t = set([1,2,3,4,5,6,7,8,9,10]) s.difference_update(t) print(s)	(12)
s.symmetric_difference(t) or s^t	Returns a new set with elements either in s or in t but not both	s = set([1,2,10,12]) t = set([1,2,3,4,5,6,7,8,9,10]) z = s^t print(z)	(3,4,5,6,7,8,9,12)

Contd

Table 8.4 *Contd*

Operation	Description	Code	Output
s.copy()	Returns a copy of set s	s = set([1,2,10,12]) t = set([1,2,3,4,5,6,7,8,9,10]) print(s.copy())	(1,2,12,10)
s.isisjoint(t)	Returns True if two sets have a null intersection	s = set([1,2, 3]) t = set([4,5,6]) print(s.isdisjoint(t))	True
all(s)	Returns True if all elements in the set are True and False otherwise	s = set([0,1,2,3,4]) print(all(s))	False
any(s)	Returns True if any of the elements in the set is True. Returns False if the set is empty	s = set([0,1,2,3,4]) print(any(s))	True
enumerate(s)	Returns an enumerate object which contains index as well as value of all the items of set as a pair	s = set(['a','b','c','d']) for i in enumerate(s): print(i,end= ' ')	(0, 'a') (1, 'c') (2, 'b') (3, 'd')
max(s)	Returns the maximum value in a set	s = set([0,1,2,3,4,5]) print(max(s))	5
min(s)	Returns the minimum value in a set	s = set([0,1,2,3,4,5]) print(min(s))	0
sum(s)	Returns the sum of elements in the set	s = set([0,1,2,3,4,5]) print(sum(s))	15
sorted(s)	Return a new sorted list from elements in the set. It does not sorts the set as sets are immutable.	s = set([5,4,3,2,1,0]) print(sorted(s))	[0,1,2,3,4,5]
s == t and s != t	s == t returns True if the two set are equivalent and False otherwise. s!=t returns True if both sets are not equivalent and False otherwise	s = set(['a','b','c']) t = set("abc") z = set(tuple('abc')) print(s == t) print(s!=z)	True False

Key points to remember

- A set cannot contain other mutable objects (like lists). Therefore, the following code will give an error when executed.

```
s = {10,20,[30,40]}
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 1, in <module>
    s = {10,20,[30,40]}
TypeError: unhashable type: 'list'
```

- To make an empty list you write, `List1 = []`, to make an empty tuple you write `Tup = ()`, but to make an empty set you cannot write `s = {}`, because Python will make this as a dictionary. Therefore, to create an empty set use the `set()` as shown below.

```
>>> s = set()
>>> print(s)
set()
print(type(s))
<class 'set'>
```

```
>>> t = {}
>>> print(type(t))
<class 'dict'>
```

- Since sets are unordered, indexing have no meaning.
- Set operations do not allow users to access or change an element using indexing or slicing. This is illustrated by code given below.

Example 8.51 Program to illustrate updating of a set

```
s = {1,2,3,4,5}
print(s[0])
```

OUTPUT

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2, in <module>
    print(s[0])
TypeError: 'set' object does not support indexing
```

Note A set can be created from a list but a set cannot contain a list.

- You can iterate through each item in a set using a `for` loop as shown in the code given below.

Example 8.52 Program to iterate through a set

```
s = set("Hello All, Good Morning")
for i in s:
    print(i,end = ' ')
```

OUTPUT

```
A   e d G i H M l o , g r n
```

- The `copy()` method makes a shallow copy of the set. This means that all the objects in the new set are references to the same objects as the original set.

Note To add a single element in the set use the `add()` method and to add multiple elements in the set, use the `update()` method.

Program 8.31 Write a program that generates a set of prime numbers and another set of odd numbers. Demonstrate the result of union, intersection, difference, and symmetric difference operations on these sets.

```
odds = set([x*2+1 for x in range(1,10)])
print(odds)
primes = set()
for i in range(2, 20):
    j = 2
    flag = 0
    while j<i/2:
        if i%j == 0:
            flag = 1
        j+=1
    if flag == 0:
        primes.add(i)
print(primes)
print("UNION : ",odds.union(primes))
print("INTERSECTION : ", odds.intersection(primes))
print("SYMMETRIC DIFFERENCE : ", odds.symmetric_difference(primes))
print("DIFFERENCE : ", odds.difference(primes))
```

OUTPUT

```
{3, 5, 7, 9, 11, 13, 15, 17, 19}
{2, 3, 4, 5, 7, 11, 13, 17, 19}
UNION :  {2, 3, 4, 5, 7, 9, 11, 13, 15, 17, 19}
INTERSECTION :  {3, 5, 7, 11, 13, 17, 19}
SYMMETRIC DIFFERENCE :  {2, 4, 9, 15}
DIFFERENCE :  {9, 15}
```

Program 8.32 Write a program that creates two sets. One of even numbers in range 1-10 and the other has all composite numbers in range 1–20. Demonstrate the use `all()`, `issuperset()`, `len()`, and `sum()` functions on the sets.

```
evens = set([x*2 for x in range(1,10)])
print("EVENS : ", evens)
composites = set()
for i in range(2, 20):
    j = 2
    flag = 0
    while j<=i/2:
        if i%j == 0:
            composites.add(i)
        j+=1
print("COMPOSITES : ", composites)
print("SUPERSET : ", evens.issuperset(composites))
print("ALL : ", all(evens))
print("LENGTH OF COMPOSITES SET : ", len(composites))
print("SUM OF ALL NUMBERS IN EVENS SET : ", sum(evens))
```

OUTPUT

```
EVENS :  {2, 4, 6, 8, 10, 12, 14, 16, 18}
COMPOSITES :  {4, 6, 8, 9, 10, 12, 14, 15, 16, 18}
SUPERSET :  False
ALL :  True
LENGTH OF COMPOSITES SET :  10
SUM OF ALL NUMBERS IN EVENS SET :  90
```

Program 8.33 Write a program that creates two sets—squares and cubes in range 1–10. Demonstrate the use of update(), pop(), remove(), add() and clear() functions.

```
squares = set([x**2 for x in range(1,10)])
cubes = set([x**3 for x in range(1,10)])
print("SQUARES : ", squares)
print("CUBES : ", cubes)
squares.update(cubes)
print("UPDATE : ", squares)
squares.add(11*11)
squares.add(11*11*11)
print("ADD : ", squares)
print("POP : ", squares.pop())
squares.remove(1331)
print("REMOVE : ", squares)
squares.clear()
print("CLEAR : ", squares)
```

OUTPUT

```
SQUARES :  {64, 1, 4, 36, 9, 16, 49, 81, 25}
CUBES :  {64, 1, 512, 8, 343, 216, 729, 27, 125}
UPDATE :  {64, 1, 512, 4, 36, 8, 9, 16, 49, 81, 729, 343, 216, 25, 27, 125}
ADD :  {64, 1, 512, 121, 4, 36, 8, 9, 16, 49, 81, 1331, 729, 343, 216, 25, 27, 125}
POP :  64
REMOVE :  {1, 512, 121, 4, 36, 8, 9, 16, 49, 81, 729, 343, 216, 25, 27, 125}
CLEAR :  set()
```

Program 8.34 Write a program that has a list of countries. Create a set of the countries and print the names of the countries in sorted order.

```
countries = ['India', 'Russia', 'China', 'Brazil', 'England']
C_set = sorted(set(countries))
print(C_set)
```

OUTPUT

```
['Brazil', 'China', 'England', 'India', 'Russia']
```

8.6 DICTIONARIES

Dictionary is a data structure in which we store values as a pair of key and value. Each key is separated from its value by a colon (:), and consecutive

Programming Tip: Using a mutable object as a dictionary key causes a TypeError.

items are separated by commas. The entire items in a dictionary are enclosed in curly brackets({}). The syntax for defining a dictionary is

```
dictionary_name = {key_1: value_1, key_2: value_2, key_3: value_3}
```

If there are many keys and values in dictionaries, then we can also write just one key-value pair on a line to make the code easier to read and understand. This is shown below.

```
dictionary_name = {key_1: value_1,
                   key_2: value_2,
                   key_3: value_3,
                   }
```

While, keys in the dictionary must be unique and be of any immutable data type (like strings, numbers, or tuples), there is no stringent requirement for uniqueness and type of values. That is, value of a key can be of any type. Remember that *dictionaries are not sequences, rather they are mappings*. **Mappings** are collections of objects that store objects by key instead of by relative position.

Note Dictionary keys are case-sensitive. Two keys with the same name but in different case are not the same in Python.

8.6.1 Creating a Dictionary

The syntax to create an empty dictionary can be given as,

```
dictionary_variable = []
```

The syntax to create a dictionary with key-value pairs is:

```
dictionary_variable = {key1 : val1, key2 : val2, ...}
```

- To create an empty dictionary, just write the following line of code.

```
Dict = {}
print(Dict)
```

OUTPUT

```
{}
```

- A dictionary can be also created by specifying key-value pairs separated by a colon in curly brackets as shown below. Note that one key value pair is separated from the other using a comma.

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print(Dict)
```

OUTPUT

```
{'Roll_No': '16/001', 'Name': 'Arav', 'Course': 'BTech'}
```

- To create a dictionary with one or more key-value pairs you can also use the `dict()` function. The `dict()` creates a dictionary directly from a sequence of key value pairs. For example, the line of code given below creates a dictionary using a list of key-value pairs.

Programming Tip: Hash table is an array whose indexes are obtained using a hash function on the keys. A hash function distributes the keys evenly in the array and minimizes collisions.

```
print(dict([('Roll_No', '16/001'), ('Name','Arav'), ('Course','BTech')]))
```

OUTPUT

```
{'Roll_No': '16/001', 'Name': 'Arav', 'Course': 'BTech'}
```

- Dictionary comprehensions is another way of creating a dictionary. A *dictionary comprehension* is a syntactic construct which creates a dictionary based on existing dictionary. The syntax can be given as,

```
D = {expression for variable in sequence [if condition]}
```

According to the syntax, we place the dictionary comprehension within curly brackets. It has three parts – `for` loop, condition, and expression. First, the `for` loop is used to go through the sequence. The `if` condition is optional and if specified, only those values in the sequence are evaluated using the expression which satisfy the condition.

Note The expression generates elements of dictionary from items in the sequence that satisfy the condition.

Example 8.53 Program to create 10 key-value pairs where key is a number in the range 1-10 and the value is twice the number

```
Dict = {x : 2*x for x in range(1,10)}
print(Dict)
```

OUTPUT

```
{1: 2, 2: 4, 3: 6, 4: 8, 5: 10, 6: 12, 7: 14, 8: 16, 9: 18}
```

8.6.2 Accessing Values

To access values in a dictionary, square brackets are used along with the key to obtain its value.

Example 8.54 Program to access values stored in a dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Dict[ROLL_NO] = ", Dict['Roll_No'])
print("Dict[NAME] = ", Dict['Name'])
print("Dict[COURSE] = ", Dict['Course'])
```

OUTPUT

```
Dict[ROLL_NO] =  16/001
Dict[NAME] =  Arav
Dict[COURSE] =  BTech
```


Note that if you try to access an item with a key, which is not specified in the dictionary, a `KeyError` is generated. For example,

```
Dict = {}
print("Dict[MARKS] = ", Dict['Marks'])
```

Programming Tip: Collision means two or more keys pointing to the same location.

then, we will get the output as,

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 2, in <module>
    print "Dict[MARKS] = ", Dict['Marks']
KeyError: 'Marks'
```

8.6.2 Adding and Modifying an Item in a Dictionary

To add a new entry or a key-value pair in a dictionary, just specify the key-value pair as you had done for the existing pairs. The syntax to add an item in a dictionary is given as,

```
dictionary_variable[key] = val
```

Example 8.55 Program to add a new item in the dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Dict[ROLL_NO] = ", Dict['Roll_No'])
print("Dict[NAME] = ", Dict['Name'])
print("Dict[COURSE] = ", Dict['Course'])
Dict['Marks'] = 95      # new entry
print("Dict[MARKS] = ", Dict['Marks'])
```

Programming Tip: Trying to index a key that isn't part of the dictionary returns a KeyError.

OUTPUT

```
Dict[ROLL_NO] =  16/001
Dict[NAME] =  Arav
Dict[COURSE] =  BTech
Dict[MARKS] =  95
```

8.6.3 Modifying an Entry

To modify an entry, just overwrite the existing value as shown in the following example.

Example 8.56 Program to modify an item in the dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Dict[ROLL_NO] = ", Dict['Roll_No'])
print("Dict[NAME] = ", Dict['Name'])
print("Dict[COURSE] = ", Dict['Course'])
Dict['Marks'] = 95      # new entry
print("Dict[MARKS] = ", Dict['Marks'])
```

```
Dict['Course'] = 'BCA'
print("Dict[COURSE] = ", Dict['Course']) #entry updated
```

OUTPUT

```
Dict[Roll_NO] =  16/001
Dict[NAME] =  Arav
Dict[COURSE] =  BTech
Dict[MARKS] =  95
Dict[COURSE] =  BCA
```

Note Dictionary is an associative array also known as hashes since any key of the dictionary can be associated or mapped to a value.

8.6.4 Deleting Items

You can delete one or more items using the `del` keyword. To delete or remove all the items in just one statement, use the `clear()` function. Finally, to remove an entire dictionary from the memory, we can again use the `del` statement as `del Dict_name`. The syntax to use the `del` statement can be given as,

```
del dictionary_variable[key]
```

Example 8.57 Program to demonstrate the use of `del` statement and `clear()` function

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Dict[Roll_NO] = ", Dict['Roll_No'])
print("Dict[NAME] = ", Dict['Name'])
print("Dict[COURSE] = ", Dict['Course'])
del Dict['Course']     # deletes a key-value pair
print("After deleting course : ", Dict)
Dict.clear()        # deletes all entries
print("After clear(), Dictionary has no items : ", Dict)
del Dict            # deletes the variable Dict from memory
print("Dict does not exist.......")
print(Dict)
```

Programming Tip: Only immutable objects can be used as keys to dictionaries.

OUTPUT

```
Dict[Roll_NO] =  16/001
Dict[NAME] =  Arav
Dict[COURSE] =  BTech
After deleting course :  { 'Roll_No': '16/001', 'Name': 'Arav'}
After clear(), Dictionary has no items :  {}
Dict does not exist.......
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 11, in <module>
    print(Dict)
NameError: name 'Dict' is not defined
```


- You can also use the `pop()` method to delete a particular key from the dictionary. The syntax of the `pop()` method is given as,

```
dict.pop(key [, default])
```

As the name suggests, the `pop()` method removes an item from the dictionary and returns its value. If the specified key is not present in the dictionary, then the default value is returned. Since default is optional, if you do not specify the default value and the key is also not present in the dictionary, then a `KeyError` is generated. Another method `dict.popitem()` randomly pops and returns an item from the dictionary. The use of these methods are illustrated in the program given below.

Example 8.58 Program to randomly `pop()` or remove an element from a dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Name is : ", Dict.pop('Name'))    # returns Name)
print("Dictionary after popping Name is : ", Dict)
print("Marks is :", Dict.pop('Marks', -1))   # returns default value
print("Dictionary after popping Marks is : ", Dict)
print("Randomly popping any item : ",Dict.popitem())
print("Dictionary after random popping is : ", Dict)
print("Aggregate is :", Dict.pop('Aggr'))    # generates error
print("Dictionary after popping Aggregate is : ", Dict)
```

OUTPUT

```
Name is :  Arav
Dictionary after popping Name is :  {'Course': 'BTech', 'Roll_No': '16/001'}
Marks is : -1
Dictionary after popping Marks is :  {'Course': 'BTech', 'Roll_No': '16/001'}
Randomly popping any item :  ('Course', 'BTech')
Dictionary after random popping is :  {'Roll_No': '16/001'}
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 8, in <module>
    print("Aggregate is :", Dict.pop('Aggr'))
KeyError: 'Aggr'
```

Key points to remember

- Keys must have unique values. Not even a single key can be duplicated in a dictionary. If you try to add a duplicate key, then the last assignment is retained. This is shown in the example given below.

Example 8.59 Program to illustrate the use of duplicate keys in a dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech', 'Name' :
'Kriti'}
print("Dict[Roll_NO] = ", Dict['Roll_No'])
print("Dict[NAME] = ", Dict['Name'])
```

```
print("Dict[COURSE] = ", Dict['Course'])
```

OUTPUT

```
Dict[ROll_NO] =  16/001
Dict[NAME] =  Kriti
Dict[COURSE] =  BTech
```

- In a dictionary, keys should be strictly of a type that is immutable. This means that a key can be of strings, number, or tuple type but it cannot be a list which is mutable. In case you try to make your key of a mutable type, then a `TypeError` will be generated as shown below.

Example 8.60 Program to illustrate the use of mutable keys in a dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("Dict[ROll_NO] = ", Dict['Roll_No'])
print("Dict[NAME] = ", Dict['Name'])
print("Dict[COURSE] = ", Dict['Course'])
```

OUTPUT

```
Dict[ROll_NO] =  16/001
Dict[NAME] =  Arav
Dict[COURSE] =  BTech
```

Programming Tip: In Python "None" is a special value like null or nil which means no value.

- Tuples can be used as keys only if they contain immutable objects like strings, numbers, or other tuples. If a tuple used as key contains any mutable object either directly or indirectly, then an error is generated. This is shown in the code given in the following example.

Example 8.61 Program to use tuple as keys

```
Dict = {(1,2),([4,5,6])}
print(Dict)
```

OUTPUT

```
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 1, in <module>
    Dict = {(1,2),([4,5,6])}
TypeError: unhashable type: 'list'
```

- The `in` keyword can be used to check whether a single key is present in the dictionary.

Example 8.62 Program to check single key in a dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
if 'Course' in Dict:
```



```
    print(Dict['Course'])
```

OUTPUT

```
BTech
```

Programming Tip: A KeyError occurs on an invalid access of a key like when a key that is used is not present in the dictionary.

8.6.5 Sorting Items in a Dictionary

The `keys()` method of dictionary returns a list of all the keys used in the dictionary in an arbitrary order. The `sorted()` function is used to sort the keys as shown below.

Example 8.63 Program to sort keys of a dictionary

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print(sorted(Dict.keys()))
```

OUTPUT

```
['Course', 'Name', 'Roll_No']
```

8.6.6 Looping over a Dictionary

You can loop over a dictionary to access only values, only keys, and both using the `for` loop as shown in the code given below.

Example 8.64 Program to access items in a dictionary using `for` loop

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
print("KEYS : ", end = ' ')
for key in Dict:
    print(key, end = ' ')   # accessing only keys
print("\nVALUES : ", end = ' ')
for val in Dict.values():
    print(val, end = ' ')   # accessing only values
print("\nDICTIONARY : ", end = ' ')
for key, val in Dict.items():
    print(key, val, "\t", end = ' ')  # accessing keys and values
```

OUTPUT

```
KEYS :  Roll_No Course Name
VALUES :  16/001 BTech Arav
DICTIONARY :  Roll_No 16/001  Course BTech    Name Arav
```

8.6.7 Nested Dictionaries

You can also define a dictionary inside another dictionary. The program given below demonstrates this concept.

Example 8.65 Program to illustrate nested dictionary (i.e., use of one dictionary inside another)

```
Students = {'Shiv' : {'CS':90, 'DS':89, 'CSA':92},
            'Sadhvi' : {'CS':91, 'DS':87, 'CSA':94},
            'Krish' : {'CS':93, 'DS':92, 'CSA':88}}
for key, val in Students.items():
    print(key, val)
```

OUTPUT

```
Sadhvi {'CS': 91, 'CSA': 94, 'DS': 87}
Krish {'CS': 93, 'CSA': 88, 'DS': 92}
Shiv {'CS': 90, 'CSA': 92, 'DS': 89}
```

8.6.8 Built-in Dictionary Functions and Methods

Table 8.5 discusses some methods and functions that can be used on dictionaries in Python.

Table 8.5 Methods and Functions of Dictionaries

Operation	Description	Example	Output
len(Dict)	Returns the length of dictionary. That is, number of items (key-value pairs)	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print(len(Dict1))	3
str(Dict)	Returns a string representation of the dictionary	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print(str(Dict1))	{'Name': 'Arav', 'Roll_No': '16/001', 'Course': 'BTech'}
Dict.clear()	Deletes all entries in the dictionary	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} Dict1.clear() print(Dict1)	{}
Dict.copy()	Returns a shallow copy of the dictionary, i.e., the dictionary returned will not have a duplicate copy of Dict but will have the same reference	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} Dict2 = Dict1.copy() print("Dict2 : ", Dict2) Dict2['Name'] = 'Saesha' print("Dict1 after modification : ", Dict1)	Dict2 : {'Course': 'BTech', 'Name': 'Arav', 'Roll_No': '16/001'} Dict1 after modification: {'Course': 'BTech',

Contd

Table 8.5 *Contd*

Operation	Description	Example	Output
		print("Dict2 after modification : ",Dict2)	'Name': 'Arav', 'Roll_No': '16/001'} Dict2 after modification: {'Course': 'BTech', 'Name': 'Saesha', 'Roll_No': '16/001'}
Dict.fromkeys(seq[,val])	Create a new dictionary with keys from seq and values set to val. If no val is specified then, None is assigned as default value	Subjects = ['CSA', 'C++', 'DS', 'OS'] Marks = dict.fromkeys(Subjects,-1) print(Marks)	{'OS': -1, 'DS': -1, 'CSA': -1, 'C++': -1}
Dict.get(key)	Returns the value for the key passed as argument. If the key is not present in dictionary, it will return the default value. If no default value is specified then it will return None	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print(Dict1.get('Name'))	Arav
Dict.has_key(key)	Returns True if the key is present in the dictionary and False otherwise	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print('Marks' in Dict1)	False
Dict.items()	Returns a list of tuples (key-value pair)	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print(Dict1.items())	[('Course', 'BTech'), ('Name', 'Arav'), ('Roll_No', '16/001')]
Dict.keys()	Returns a list of keys in the dictionary	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print(Dict1.keys())	['Course', 'Name', 'Roll_No']
Dict.setdefault(key, value)	Sets a default value for a key that is not present in the dictionary	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}	Arav has got marks = 0

Contd

Table 8.5 *Contd*

Operation	Description	Example	Output
		Dict1. setdefault('Marks',0) print(Dict1['Name'], "has got marks = ", Dict1. get('Marks'))	
Dict1.update(Dict2)	Adds the key-value pairs of Dict2 to the key-value pairs of Dict1	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} Dict2 = {'Marks' : 90, 'Grade' : 'O'} Dict1.update(Dict2) print(Dict1)	{'Grade':'O', 'Course': 'BTech', 'Name': 'Arav', 'Roll_No': '16/001', 'Marks': 90}
Dict.values()	Returns a list of values in dictionary	Dict1 = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print(Dict1.values())	['BTech', 'Arav', '16/001']
Dict.iteritems()	Used to iterate through items in the dictionary	Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} for i,j in Dict. iteritems(): print(i, j)	Course BTech Name Arav Roll_No 16/001
in and not in	Checks whether a given key is present in dictionary or not	Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'} print('Name' in Dict) print('Marks' in Dict)	True False

8.6.9 Difference between a List and a Dictionary

There are two main differences between a list and a dictionary.

- First, a list is an ordered set of items. But, a dictionary is a data structure that is used for matching one item (key) with another (value).
- Second, in lists, you can use indexing to access a particular item. But, these indexes should be a number. In dictionaries, you can use any type (immutable) of value as an index. For example, when we write Dict['Name'], Name acts as an index but it is not a number but a string.
- Third, lists are used to look up a value whereas a dictionary is used to take one value and look up another value. For this reason, dictionary is also known as a *lookup table*.

In fact, the main advantage of a dictionary is that you don't need to search for a value one by one in the entire set of values, you can find a value instantly.

- Fourth, the key-value pair may not be displayed in the order in which it was specified while defining the dictionary. This is because Python uses complex algorithms (called hashing) to provide fast access to the items stored in the dictionary. This also makes dictionary preferable to use over a list of tuples.

8.6.10 String Formatting with Dictionaries

Python also allows you to use string formatting feature with dictionaries. So you can use %s, %d, %f, etc. to represent string, integer, floating point number, or any other data.

Example 8.66 Program that uses string formatting feature to print the key-value pairs stored in the dictionary

```
Dict = {'Sneha' : 'BTech', 'Mayank' : 'BCA'}
for key, val in Dict.items():
   print("%s is studying %s" % (key,val))
```

OUTPUT

```
Sneha is studying BTech
Mayank is studying BCA
```

8.6.11 When to use which Data Structure?

- Use lists to store a collection of data that does not need random access.
- Use lists if the data has to be modified frequently.
- Use a set if you want to ensure that every element in the data structure must be unique.
- Use tuples when you want that your data should not be altered.

8.6.12 List vs Tuple vs Dictionary vs Set

- *Tuples are lists which cannot be edited.* While tuples are immutable, lists on the other hand are mutable. Hence, they can be easily edited. *Tuples have fixed size*, so you cannot add or delete items from it. But you can easily add or delete elements in a list. Look at the codes given below which demonstrates the concept.

```
# Editing and Inserting value in List
sports = ['cricket', 'tennis']
print(sports)
sports.append('baseball')
print(sports)
sports[2] = 'basketball'
print(sports)
```

OUTPUT

```
['cricket', 'tennis']
['cricket', 'tennis', 'baseball']
['cricket', 'tennis', 'basketball']
```

```
# Editing and Inserting value in Tuple
sports = ('cricket', 'tennis')
print(sports)
# Does not support the append method
# sports.append('baseball')
sports[2] = 'basketball'
print(sports)
```

OUTPUT

```
('cricket', 'tennis')
Traceback (most recent call last):
  File "C:\Python34\Try.py", line 5, in
<module>
    sports[2] = 'basketball'
TypeError: 'tuple' object does not
support item assignment
```

- Due to the mutability difference, *tuples are easier on memory and processor in comparison to lists*. This means that you can easily achieve performance optimization by using tuples, wherever possible. Moreover, tuples are best used as heterogeneous collections while lists are best used as homogenous collections (where heterogeneous means that the items contained in a tuple may belong to different types or concepts).
- Sets are used to store unordered values and *do not have index*. Unlike tuples and lists, sets can *have no duplicate data*. However, like lists and unlike tuples, you can use the `add()` function to add an element to a set and the `update()` function to edit the elements in the set.

Example 8.67 Program to add an item in a set

```
sports = set(['cricket', 'tennis'])
print(sports)
sports.add('baseball')
print(sports)
```

OUTPUT

```
{'cricket', 'tennis'}
{'cricket', 'tennis', 'baseball'}
```

- Dictionary is used to store key-value pairs. Its underlying concept and usage is absolutely different from that of list, tuple, or set. Dictionaries are best data structure for frequent lookup operations. Consider the code given below.

Example 8.68 Program that uses a dictionary to return the name of the employee of an organization when his project name is given

```
Dict = {'ProjectA' : 'Manav', 'ProjectB' : 'Raghav', 'ProjectC' : 'Harsh',
'ProjectD' : 'Vineet'}
print(Dict['ProjectC'])
```

OUTPUT

```
Harsh
```

Program 8.35 Write a program that has a dictionary of states and their codes. Add another state in the pre-defined dictionary, print all the items in the dictionary, and try to print code for a state that does not exist. Set a default value prior to printing.

```
states = {'Delhi' : 'DL', 'Haryana' : 'HR', 'Maharashtra' : 'MH', 'Rajasthan' : 'RJ'}
states['Tamil Nadu'] = 'TN'      # add another state
states.setdefault('Karnataka','Sorry, no idea')
print("Code for Rajasthan is : ", states['Rajasthan'])
print("-" * 5, "CODES", "-" * 5)
for i in states.items():
    print(i)
print("Code for Karnataka : ", states.get('Karnataka'))
```


OUTPUT

```
Code for Rajasthan is :  RJ
----- CODES -----
('Karnataka', 'Sorry, no idea')
('Haryana', 'HR')
('Delhi', 'DL')
('Rajasthan', 'RJ')
('Maharashtra', 'MH')
('Tamil Nadu', 'TN')
Code for Karnataka :  Sorry, no idea
```

Program 8.36 Write a program that creates a dictionary of radius of a circle and its circumference.

```
print("Enter -1 to exit....")
Circumference = {}
while True:
    r = float(input("Enter raidus : "))
    if r == -1:
        break
    else:
        Dict = {r:2*3.14*r}
        Circumference.update(Dict)
print(Circumference)
```

OUTPUT

```
Enter -1 to exit....
Enter raidus : 5
Enter raidus : 7
Enter raidus : 8
Enter raidus : -1
{8.0: 50.24, 5.0: 31.40000000000002, 7.0: 43.96}
```

Program 8.37 Write a program that creates two dictionaries. One that stores conversion values from meters to centimeters and the other that stores values from centimeters to meters.

```
m_cm = {x : x*100 for x in range(1,11)}
temp = m_cm.values()
cm_m = {x : x/100 for x in temp}
print("Meters : Centimeters", m_cm)
print("Centimeters : Meters", cm_m)
```

OUTPUT

```
Meters : Centimeters {1: 100, 2: 200, 3: 300, 4: 400, 5: 500, 6: 600, 7: 700,
8: 800, 9: 900, 10: 1000}
```

```
Centimeters : Meters {800: 8, 100: 1, 200: 2, 300: 3, 400: 4, 1000: 10, 500: 5,
600: 6, 900: 9, 700: 7}
```

Program 8.38 Write a program that has a set of words in English language and their corresponding words in Hindi. Define another dictionary that has a list of words in Hindi and their corresponding words in Urdu. Take all words from English language and display their meanings in both the languages.

```
E_H = {'Friend' : 'Mitr', 'Teacher' : 'Shikshak', 'Book' : 'Pustak',
'Queen' : 'Rani'}
H_U = {'Mitr' : 'Dost', 'Shikshak' : 'Adhyapak', 'Pustak' : 'Kitab',
'Rani' : 'Begum'}
for i in E_H:
  print(i, "in Hindi means", E_H[i], "and in Urdu means", H_U[E_H[i]])
```

OUTPUT

```
Book in Hindi means Pustak and in Urdu means Kitab
Teacher in Hindi means Shikshak and in Urdu means Adhyapak
Friend in Hindi means Mitr and in Urdu means Dost
Queen in Hindi means Rani and in Urdu means Begum
```

Program 8.39 Write a program that calculates fib(n) using a dictionary.

```
Dict = {0: 0, 1: 1}
def fib(n):
    if n not in Dict:
        val = fib(n-1) + fib(n-2)
        Dict[n] = val
    return Dict[n]
n = int(input("Enter the value of n : "))
print("Fib(", n, ") = ", fib(n))
```

OUTPUT

```
Enter the value of n : 10
Fib( 10 ) =  55
```

Program 8.40 Write a program that creates a dictionary of cubes of odd numbers in the range 1–10.

```
Dict = {x:x**3 for x in range(10) if x%2==1}
print(Dict)
```

OUTPUT

```
{1: 1, 3: 27, 9: 729, 5: 125, 7: 343}
```

Program 8.41 Write a program that prompts the user to enter a message. Now count and print the number of occurrences of each character.


```
def count(message):
    letter_counts = {}
    for letter in message:
        letter_counts[letter] = letter_counts.get(letter, 0) + 1
    print(letter_counts)
message = input("Enter a message : ")
count(message)
```

OUTPUT

```
Enter a message : Good Morning Friends
{' ': 2, 'e': 1, 'd': 2, 'G': 1, 'F': 1, 'i': 2, 'M': 1, 'o': 3, 'n': 3, 'g': 1,
's': 1, 'r': 2}
```

Program 8.42 Write a program to store a sparse matrix as a dictionary.

```
matrix = [[0,0,0,1,0],
          [2,0,0,0,3],
          [0,0,0,4,0]]
Dict = {}
print("Sparse Matrix")
for i in range(len(matrix)):
    print("\n")
    for j in range(len(matrix[i])):
        print(matrix[i][j], end = ' ')
        if matrix[i][j]!=0:
            Dict[(i,j)] = matrix[i][j]
print("\n\nSparse Matrix can be efficiently represented as Dictionary : ")
print(Dict)
```

OUTPUT

```
Sparse Matrix
0 0 0 1 0
2 0 0 0 3
0 0 0 4 0
Sparse Matrix can be efficiently represented as Dictionary :
{(0, 3): 1, (2, 3): 4, (1, 0): 2, (1, 4): 3}
```

Program 8.43 Write a program that inverts a dictionary. That is, it makes key of one dictionary value of another and vice versa.

```
Dict = {'Roll_No' : '16/001', 'Name' : 'Arav', 'Course' : 'BTech'}
inverted = {}
for key, val in Dict.items():
    inverted[val] = key
print("Dict : ", Dict)
print("Inverted Dict : ", inverted)
```

OUTPUT

```
Dict :  {'Course': 'BTech', 'Name': 'Arav', 'Roll_No': '16/001'}
Inverted Dict :  {'BTech': 'Course', 'Arav': 'Name', '16/001': 'Roll_No'}
```

Program 8.44 Write a program that has dictionary of names of students and a list of their marks in 4 subjects. Create another dictionary from this dictionary that has name of the students and their total marks. Find out the topper and his/her score.

```
Marks = {'Neha' : [97,89,94, 90], 'Mitul' : [92,91,94,87], 'Shefali' : [67,99, 88,90]}
tot=0
Tot_Marks = Marks.copy()
for key, val in Marks.items():
   tot = sum(val)
   Tot_Marks[key] = tot
print(Tot_Marks)
max = 0
Topper = ''
for key, val in Tot_Marks.items():
   if val>max:
      max = val
      Topper = key
print("Topper is : ", Topper, "with marks = ", max)
```

OUTPUT

```
{'Neha': 370, 'Mitul': 364, 'Shefali': 344}
Topper is :  Neha with marks =  370
```

Program 8.45 Write a program that print a histogram of frequencies of characters occurring in a message.

```
msg = 'Hello All, Good Morning... Welcome to the World of Python'
msg = msg.lower()
Dict = dict()
for word in msg:
    if word not in Dict:
        Dict[word] = 1
    else:
        Dict[word] = Dict[word] + 1
print(Dict)
for key, val in Dict.items():
    print(key, '\t', '*' * val)
```

OUTPUT

```
{'a': 1, ' ': 9, 'c': 1, 'e': 4, 'd': 2, 'g': 2, 'f': 1, 'i': 1, 'h': 3, 'm': 2,
'l': 6, 'o': 9, ',': 1, '.': 3, 'r': 2, 't': 3, 'w': 2, 'y': 1, 'n': 3, 'p': 1}
```



```
a   *
    *********
c   *
e   ****
d   **
g   **
f   *
i   *
h   ***
m   **
l   ******
o   *********
,   *
.   ***
r   **
t   ***
w   **
y   *
n   ***
p   *
```

Program 8.46 Write a program that prompts the user to enter a filename. Open the file and print the frequency of each word in it.

```
filename = input('Enter the file name: ')
file = open(filename)
counts = dict()
for line in file:
    words = line.split()
    for word in words:
        if word not in counts:
            counts[word] = 1
        else:
            counts[word] += 1
print(counts)
```

OUTPUT

```
Enter the file name: File1.txt
{'a': 1, 'and': 1, '#': 1, 'language': 1, 'Python': 1, 'of': 1, 'is': 1, 'welcome':
1, 'Programming': 1, 'to': 1, 'interesting': 1, 'very': 1, 'world': 1, 'the': 1,
'Reading': 1, 'Hello': 1, 'simple': 1, 'Happy': 1}
```

Program 8.47 Write a program to count the numbers of characters in the string and store them in a dictionary data structure.

```
str = "Good Morning World"
len = len(str)
```

```
Dict = {str:len}
print(Dict)
```

OUTPUT

```
{'Good Morning World': 18}
```

Program 8.48 Write a program that combines the lists to a dictionary.

```
keys = ['Name', 'Age', 'Marital Status']
values = ["Om", 38, "Married"]
details = zip(keys, values)
Dict = dict(details)
print(Dict)
```

OUTPUT

```
{'Name': 'Om', 'Age': 38, 'Marital Status': 'Married'}
```

Summary

- Slice operation can be performed on strings, tuples, and lists.
- If you want to modify a list and also keep a copy of the original list, then you should create a separate copy of the list (not just the reference). This process is called *cloning*.
- The `sort()` method uses ASCII values to sort the values in the list.
- List comprehensions help programmers to create lists in a concise way.
- The `enumerate()` function is used when you want to print both index as well as an item in the list.
- The `filter()` function constructs a list from those elements of the list for which a function returns True.
- The function `map()` applies a particular function to every element of a list.
- You cannot edit, insert, or delete values from a tuple. But you can always perform operations like concatenation, repetition, etc., on tuples.
- If a sequence is specified without parenthesis, it is treated to be a tuple by default.
- Variable-length arguments allows a function to accept a variable (different) number of arguments.
- `zip()` is a built-in function that takes two or more sequences and `"zips"` them into a list of tuples.
- Tuples can be used as key for a dictionary but lists cannot be used as keys.
- A *set* is a mutable and an unordered collection of items.

Glossary

Data structure An organization of data to make it easier to use.

Dictionary A collection of key-value pairs that maps from keys to values. While keys can be of any immutable type, there is no such restriction on its associated value which can be of any type.

Dictionary comprehension A syntactic construct which creates a dictionary based on an existing dictionary.

Hash function A function used to compute the location for a key.

Immutable data Data which cannot be modified. Assigning values to elements or slices of immutable data results in a run-time error.

Key Data that is mapped to a value in a dictionary. Keys are unique data items that are used to look up values in a dictionary.

Key-value pair A pair of items in a dictionary. Key is used to lookup for a value stored in the dictionary.

List A mutable data structure that can have elements that belong to different data types.