Unit IV

Strings and Lists: String as a compound data type, Length, Traversal and the for loop, **String slices**, String comparison, Looping and counting, List values, Accessing elements, List length, List membership, Lists and for loops, List operations, List deletion. Cloning lists, Nested lists .

Strings are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Creating a String

Strings in Python can be created using single quotes or double quotes or even triple quotes.

Strings in python are surrounded by either single quotation marks, or double quotation marks.'hello' is the same as "hello".You can display a string literal with the print() function:

String Length

To get the length of a string, use the len() function.

a = "Hello, World!" print(len(a))	13

Strings indexing and splitting

Like other languages, the indexing of the Python strings starts from 0. For example, The string "HELLO" is indexed as given in the below figure.

str = "HELLO"

H	E	L	L	O
0	1	2	3	4

str[0] = 'H'

str[1] = 'E'

str[2] = 'L'

str[3] = 'L'

str[4] = 'O'

String Slicing

To access a range of characters in the String, method of slicing is used. Slicing in a String is done by using a Slicing operator (colon).

Python slicing is about obtaining a sub-string from the given string by slicing it respectively from start to end.

Python slicing can be done in two ways.

- slice() Constructor
- Extending Indexing

slice() Constructor

The slice() constructor creates a slice object representing the set of indices specified by range(start, stop, step).

Syntax:

- *slice(stop)*
- *slice(start, stop, step)*

Parameters:

start: *Starting index where the slicing of object starts.*
stop: *Ending index where the slicing of object stops.*
step: *It is an optional argument that determines the increment between each index for slicing.*

String ='ASTRING' s1 = slice(3) s2 = slice(1, 5, 2) print(String[s1]) print(String[s2])	String slicing AST SR GITA

Extending indexing

In Python, indexing syntax can be used as a substitute for the slice object. This is an easy and convenient way to slice a string both syntax wise and execution wise.

Syntax
string[start:end:step]

start, end and step have the same mechanism as slice() constructor.

String ='ASTRING' print(String[:3]) print(String[1:5:2])	AST SR

String Comparison in Python

Method 1: Using **Relational Operators**

The relational operators compare the Unicode values of the characters of the strings from the zeroth index till the end of the string. It then returns a boolean value according to the operator used.

Example: "Geek" == "Geek" will return True as the Unicode of all the characters are equal

print("Geek" == "Geek") print("Geek" < "geek") print("Geek" > "geek") print("Geek" != "Geek")	True True False False

Looping and Counting

We will implement a few of the methods that we described earlier to show how they can be done. This program demonstrates another pattern of computation called a *counter*. The variable count is initialized to 0 and then incremented each time an "r" is found. When the loop exits, count contains the result: the total number of r's.

word = 'raspberry'
count = 0
for letter in word:
if letter == 'r':
count = count + 1
print(count)
	3

# Python program to illustrate 	
# while loop 	
count = 0
while (count < 3):
count = count + 1
print("Hello Geek")
	Hello Geek
	Hello Geek
	Hello Geek

word = 'banana'
count = 0
for letter in word:
if letter == 'a':
count = count + 1
print(count)
	3

Traversal and the for loop

A lot of computations involve processing a string one character at a time. Often they start at the beginning, select each character in turn, do something to it, and continue until the end. This pattern of processing is called a **traversal**.

fruit = "fruit" index = 0 while index < len(fruit): letter = fruit[index]	f r u i

print(letter) index = index + 1	t
string = "Apple" i = 0 one = string[0:i+1] two = string[0:i+2] three = string[0:i+3] four = string[0:i+4] five = string[0:i+5] print(one) print(two) print(three) print(four) print(five)	A Ap App Appl Apple
for name in ["raj", "Amit", "Kiran", "Aman"]: **show = "Hi " + name + ". welcome to Bikaner!"** **print(show)**	**Hi raj. welcome to Bikaner!** **Hi Amit. welcome to Bikaner!** **Hi Kiran. welcome to Bikaner!** **Hi Aman. welcome to Bikaner!**

List

A list in Python is used to store the sequence of various types of data. Python lists are mutable type its mean we can modify its element after it created. However, Python consists of six data-types that are capable to store the sequences, but the most common and reliable type is the list.

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].Lists are used to store multiple items in a single variable.

Characteristics of Lists

The list has the following characteristics:

- The lists are ordered.
- The element of the list can access by index.
- The lists are the mutable type.
- The lists are mutable types.
- A list can store the number of various elements.

thislist = ["apple", "banana", "cherry"] print(thislist)	['apple', 'banana', 'cherry']

List Items

List items are ordered, changeable, and allow duplicate values.List items are indexed, the first item has index [0], the second item has index [1] etc.

List Length

To determine how many items a list has, use the len() function:

thislist = ["apple", "banana", "cherry"] print(len(thislist))	3

Access Items

List items are indexed and you can access them by referring to the index number:

thislist = ["apple", "banana", "cherry"] print(thislist[1])	banana

Negative Indexing

Negative indexing means start from the end

-1 refers to the last item, -2 refers to the second last item etc.

thislist = ["apple", "banana", "cherry"] print(thislist[-1])	cherry

Loop Through a List

You can loop through the list items by using a for loop: List is equivalent to arrays in other languages, with the extra benefit of being dynamic in size. In Python, the list is a type of container in Data Structures, which is used to store multiple data at the same time. Unlike Sets, lists in Python are ordered and have a definite count. There are multiple ways to iterate over a list in Python.

thislist = ["apple", "banana", "cherry"] for x in thislist: print(x)	apple banana cherry

# Python3 code to iterate over a list 	
list = [1, 3, 5, 7, 9]
 	
# Using for loop 	
for i in list:
  print(i)
	1
	3
	5
	7
	9

Removing elements from the list

Python provides the **remove()** function which is used to remove the element from the list.

To remove a list element, you can use either the del statement if you know exactly which element(s) you are deleting or the remove() method

list1 = ['physics', 'chemistry', 1997, 2000]; del list1[2]; print list1	['physics', 'chemistry', 2000]

# Creating a List 	
List = [1, 2, 3, 4, 5]
List.remove(2)
print(List)
	[1, 3, 4, 5]

Cloning Lists

If we want to modify a list and also keep a copy of the original, we need to be able to make a copy of the list itself, not just the reference. This process is sometimes called **cloning**, to avoid the ambiguity of the word copy.

list1 = [10, 22, 44, 23, 4] list2 = list(list1) print(list1) print(list2)	[10, 22, 44, 23, 4] [10, 22, 44, 23, 4]

Nested list

A nested list is a list within a list. Python provides features to handle nested list gracefully and apply common functions to manipulate the nested lists.

Creating a Matrix

Creating a matrix involves creating series of rows and columns. We can use for loop for creating the matrix rows and columns by putting one python list with for loop inside another python list with for loop.We can use the list comprehension with filtering feature by using the for loop within the sub-lists. Below we have a 2 dimensional list with one layer of sub-list inside a bigger list. We access selective elements from each of these nested lists. By using a filter condition.

years = [['January', 'February', 'March'], ['April', 'May', 'June'], ['July', 'August', 'September'],['October','November','December']]
# Nested List comprehension with an if condition 	
years = [years for sublist in years for years in **sublist if len(years) <= 4]**
print(years)

['May', 'June', 'July']

Membership Operators

Membership operators are operators used to validate the membership of a value. It test for membership in a sequence, such as strings, lists, or tuples.Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

not in Returns True if a sequence with the specified value is not present in the object x not in y

1. **in operator :** The 'in' operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a variable in the specified sequence and false otherwise.

x = [1,2,3,4,5] print(8 in x)	False
x = [1,2,3,4,5] print(3 in x)	True

not in' operator- Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

x = [1,2,3,4,5] print(4 not in x)	False
x = [1,2,3,4,5] print(8 not in x)	True

List operations

append() Method : The append() method appends an element to the end of the list.

name = ["Amit", "Jai", "Sonu"] name.append("Mohit") print(name)	['Amit', 'Jai', 'Sonu', 'Mohit']

clear() Method : The clear() method removes all the elements from a list.

name = ["Amit", "Jai", "Sonu"] name.clear() print(name)	[]

Copy() :

The copy() method returns a copy of the specified list.

name = ["amit", "karan", "charu"] x = name.copy() print(x) print(name)	['amit', 'karan', 'charu'] ['amit', 'karan', 'charu']

count()

The count() method returns the number of elements with the specified value.

name = ["amit", "karan", "charu","amit"] x = name.count("amit") print(x)	2

extend()

The extend() method adds the specified list elements (or any iterable) to the end of the current list.

list1 = ["amit", "karan"] list2 = ["naman","raj"] list1.extend(list2) print(list1)	['amit', 'karan', 'naman', 'raj']

sort()

the sort() method sorts the list ascending by default.You can also make a function to decide the sorting criteria(s).

name = ['raj', 'amit', 'jiya'] name.sort() print(name)	['amit', 'jiya', 'raj']