PYTHON PROGRAMMING LANGUAGE
CS226
Semester-II

Unit-1

Strings:

A string is a group of characters or sequence of characters.

Create and initializing a string

Strings can be created by enclosing characters inside a single quote or double quotes. Even triple quotes can be used in Python but generally used to represent multiline strings and docstrings.

Example 1:

```
>> ‘Hello’

‘Hello’



>> “Hello”
‘Hello’



>> ‘‘‘Hello’’’

‘Hello’



>>>myfirst=“Save Earth”
>>>print (myfirst)

Save Earth

>>>print (“A friend in need is a friend indeed”)

  A friend in need is a friend indeed


>>>my_string1 = raw_input("Enter a string")
```

```
>>>print(my_string1)



>>>my_string2 = """Hello, welcome to the world of Python"""

>>>print(my_string2)
```

Access elements in a string:
Individual characters in a string are accessed using the subscript[] operator. The expression in brackets is called an **index**. The index of the first character is 0 and that of the last character is n-1 where n is the number of characters in the string. Trying to access a character out of index range will raise an IndexError. Python allows negative indexing for its sequences.

String A	H	E	L	L	O
Positive Index	0	1	2	3	4
Negative Index	-5	-4	-3	-2	-1

Example 2:

A= "HELLO"

To access the first character of the string

```
>>>print A[0]

H
```

To access the fourth character of the string

```
>>>print A[3]

L
```

To access the last character of the string

```
>>print A[-1]
```

O

To access the third last character of the string

```
>>print A[2]

L

>>print A[-3]

L
```

Both indexing elements referring same elements.

Important points about accessing elements in the strings using subscripts

- Positive subscript helps in accessing the string from the beginning
- Negative subscript helps in accessing the string from the end.
- Subscript 0 or –ve n(where n is length of the string) displays the first element.
  Example : A[0] or A[-5] will display "H"
- Subscript 1 or –ve (n-1) displays the second element.

Strings are immutable

Strings are immutable means that the contents of the string cannot be changed after it is created.

Let us understand the concept of immutability with help of an example.

Example 3

```
>>>str='honesty'

>>>str[2]='p'

TypeError: 'str' object does not support item assignment
```

Python does not allow the programmer to change a character in a string. As shown in the above example, str has the value "honesty". An attempt to replace "n" in the string by "p" displays a TypeError.

Traversing a string

Traversing a string means accessing all the elements of the string one after the other by using the subscript. A string can be traversed using: for loop or while loop.

String traversal using for loop	String traversal using while loop
Program: A="Welcome" for i in A: print (i) Output: W e l c o m e	**Program:** A="Welcome" i=0 while (i<len(A)): print (A[i]) i=i+1 Output: W e l c o m e

Strings Operations

Operator	Description	Example
+ (Concatenation or Addition)	The + operator joins the text on both sides of the operator	Example4: >>> "Save"+"Earth" Save Earth Example 5:

		str1 = 'Hello'
		
		str2 ='World!'
		
		# using +
		
		print('str1 + str2 = ', str1 + str2)

*** (Repetition)**	The * operator repeats the string on the left hand side times the value on right hand side.	Example 6: >>>3*"Save Earth" "Save Earth Save Earth Save Earth" Example 7: str= "Hello" print (3*str) Output: HelloHelloHello

in (Membership)	The operator displays 1 if the string contains the given character or the sequence of characters.	>>>A="Save Earth"
		
		>>> "S" in A
		
		True
		
		>>> "Save" in A
		
		True
		
		>> "SE" in A
		
		False

not in	The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of in operator discussed above)	>>>"SE" not in "Save Earth" True >>>"Save" not in "Save Earth" False
range (start, stop[, step])	This function is already discussed in previous chapter.	
Slice[n:m]	The Slice[n : m] operator extracts sub	>>>A="Save Earth"

	parts from the strings.	>>> print A[1:3] av The print statement prints the substring starting from subscript 1 and ending at subscript 3 but not including subscript 3

Slice Operation:
A substring of a string is called a slice. The slice operation is used to extract sub-parts of strings.

Example 8: Program to demonstrate slice operation on strings.

```
x= “PYTHON”
print (x[1:5])
# output: YTHO
print (x[:6])

# output: PYTHON

print (x[1:])

# output: YTHON

print (x[:])

# output: PYTHON

print (x[1:20])

# output: YTHON

print (x[-1])

# output: N

print (x[-6])

# output: P

print (x[-2:])

# output: ON
```

```
print (x[:-2])

# output: PYTH

print (x[-5:-2])

# output: YTH
```

Slicing string stride:
In the slice operation, we can specify a third argument as the stride, which refers to the number of characters to move forward after the first character is retrieved from the string. The default value of stride is 1.

Example 9:

```
name= “Welcome to the world of Python”
print (name[2:10])

print (name[2:10:1])

print (name[2:10:2])

print (name[2:13:4])
```

```
Output:
lcome to

lcome to

loet

le
```

Example 10:

```
test = “Welcome to the world of Python”

print (test[::3])
```

```
Output:
WceohwloPh
```

Example 11:

```
st = “Welcome to the world of Python”

print (st[::-1])
```

Output:

```
nohtyP fo dlrow eh tot emocleW
```

Example 12:

```
A = “Welcome to the world of Python”

print (A[::-3])
```

Output:

```
nt    r ttml
```

Strings using relations operators:

Symbol/Operator	Description	Example
<	Less than	>>> “Hello”< “Goodbye” False >>>'Goodbye'< 'Hello' True
>	Greater than	>>>”Hello”> “Goodbye” True >>>'Goodbye'> 'Hello' False

<=	less than equal to	>>>"Hello"<= "Goodbye" False >>>'Goodbye' <= 'Hello' True
>=	greater than equal to	>>>"Hello">= "Goodbye" True >>>'Goodbye' >= 'Hello' False
! =	not equal to	>>>"Hello"!= "HELLO" True >>> "Hello" != "Hello" False
==	equal to	>>>"Hello" == "Hello" True >>>"Hello" == "Good Bye" False

Escape Sequence:

An escape sequences is nothing but a special character that has a specific function.

Escape sequence	Description	Example
\n	New line	>>> print "Hot\nCold" Hot Cold
\t	Tab space	>>>print "Hot\tCold" Hot Cold

\	Backslash	>>>print("\\") \
\’	Single quote	>>>print("\' ") ‘
\”	Double quote	>>>print (("\"") “

Raw String:
If you want to specify a string that should not handle any escape sequences and want to display exactly as specified, then you need to specify that string as a raw string.

A Raw string is specified by prefixing r or R to the string.

Example 13:

```
>>>print (R “What\’s your name?”)
What\’s your name?
```

String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family.

%s string conversion via str() prior to formatting

%d signed decimal integer

%f floating point real number

Example 14:

```
print "My name is %s and weight is %d kg!" % ('student', 45.256)
```

Output

```
My name is student and weight is 45 kg!
```

String Output formatting:

Sometimes we would like to format our output to make it look attractive. This can be done by using the str.format() method. This method is visible to any string object.

Example 15:

```
x = 5; y = 10

print ('The value of x is {} and y is {}'.format(x,y))
```

Output:- The value of x is 5 and y is 10

Here the curly braces {} are used as placeholders. We can specify the order in which it is printed by using numbers (tuple index).

Example 15:

```
print('I love {0} and {1}'.format('bread','butter'))

# Output: I love bread and butter


print('I love {1} and {0}'.format('bread','butter'))

# Output: I love butter and bread
```

We can even use keyword arguments to format the string.

```
print('Hello {name}, {greeting}'.format(greeting = 'Goodmorning', name = 'John'))

# Output: Hello John, Goodmorning
```

String Functions and Methods:

Syntax	Description	Example
len()	Returns the length of the string Syntax: len(string_variable)	>>>A="Save Earth" >>> print len(A) 10
capitalize()	Returns the exact copy of the string with the first letter in upper case Syntax: string_variable.capitalize()	>>>str="welcome" >>>print str.capitalize() Welcome
find(sub[, start[, end]])	The function is used to search the first occurrence of the substring in the given string. It returns the index at which the substring starts. It returns -1 if the substring does occur in the string. Syntax: string_variable. find(substring,start,end)	>>>str='mammals' >>>str.find('ma') 0 On omitting the start parameters, the function starts the search from the beginning. >>>str.find('ma',2) 3 >>>str.find('ma',2,4) -1 Displays -1 because the substring could not be found between the index 2 and 4-1 >>>str.find('ma',2,5) 3

isalnum()	Returns True if the string contains only letters and digit. It returns False ,If the string contains any special character like _ , @,#,* etc. syntax:string_variable.isalnum( )	>>>str='Save Earth' >>>str.isalnum() False The function returns False as space is an alphanumeric character. >>>'Save1Earth'.isalnum() True

isalpha()	Returns True if the string contains only letters. Otherwise return False. syntax: string_variable.isalpha()	>>> 'Click123'.isalpha()
		
		False
		
		>>> 'python'.isalpha()
		
		True

isdigit()	Returns True if the string contains only numbers. Otherwise it returns False. Syntax: string_variable.isdigit()	>>>str="SAVE EARTH" >>>print str.isdigit() False Message= "007" print (Message.isdigit()) Output: True
lower()	Returns the exact copy of the string with all the letters in lowercase. syntax: string_variable.lower()	>>> str = "SAVE EARTH" >>>print str.lower() save earth
islower()	Returns True if the string is in lowercase syntax: string_variable.islower()	str="save earth" >>>print str.islower() True
isupper()	Returns True if the string is in uppercase.	str="save earth"

	syntax:string_variable.isupper()	>>> print str.isupper() False
upper()	Returns the exact copy of the string with all letters in uppercase. syntax: string_variable.upper()	>>> str= “welcome” >>>print str.upper() WELCOME
strip()	Removes all leading and trailing whitespaces in a string. syntax:string_variable.strip()	>>>str=” hello “ >>>str.strip() Hello

lstrip()	Returns the string after removing the space(s) on the left of the string.
	syntax:string.lstrip()
		>>> print str
		
		Save Earth
		
		>>>str.lstrip()
		
		'Save Earth'
		
		>>>str='Teach India Movement'
		
		>>> print str.lstrip("T")
		
		each India Movement
		
		>>> print str.lstrip("Te")
		
		ach India Movement
		
		>>> print str.lstrip("Pt")
		
		Teach India Movement
		
		If a string is passed as argument to the lstrip() function, it removes those characters from the left of the string

rstrip()	Returns the string after removing the space(s) on the right of the string. syntax:string_variable.rstrip()	>>>str='Teach India Movement‟ >>> print str.rstrip() Teach India Movement

isspace()	Returns True if the string contains only white spaces and False even if it contains one character. syntax:string_variable.isspace()	>>> str=' '
		
		>>> print str.isspace()
		
		True
		
		>>> str='p'
		
		>>> print str.isspace()
		
		False

istitle()	Returns True if the string is title cased. Otherwise returns False syntax: string_variable.istitle()	>>> str='The Green Revolution'
		
		>>> str.istitle()
		
		True
		
		>>> str='The green revolution'
		
		>>> str.istitle()
		
		False

replace(old, new)	The function replaces all the occurrences of the old string with the new string syntax:string_variable.replace(old,new)	>>>str="hello"
		
		>>> print str.replace('l','%')
		
		he%%o
		
		>>> print str.replace('l','%%')
		he%%%%o

join (list)	Returns a string in which the string elements have been joined by a separator.
	
	Syntax: string_variable.join(list)
		>>> str1=('jan', 'feb' ,'mar')
		>>>str="&"
		
		>>> str.join(str1)
		
		'jan&feb&mar

swapcase()	Returns the string with case changes
	
	Syntax:string_variable.swapcase()
		>>> str='UPPER'
		
		>>> print str.swapcase()
		
		upper
		
		>>> str='lower'
		
		>>> print str.swapcase()

		LOWER

split([sep[, maxsplit]])	The function splits the string into substrings using the separator. The second argument is optional and its default value is zero. If an integer value N is given for the second argument, the string is split in N+1 strings.
	
	Syntax: string_variable.split(substring,value)
		>>>str='The$earth$is$what$we$all$have$in$common.'
		
		>>> str.split($,3)
		
		SyntaxError: invalid syntax
		
		>>> str.split('$',3)
		
		['The', 'earth', 'is', 'what$we$all$have$in$common.']
		>>> str.split('$')
		
		['The', 'earth', 'is', 'what', 'we', 'all', 'have', 'in', 'common.']
		
		>>> str.split('e')
		
		['Th', ' Gr', '', 'n R', 'volution']
		
		>>> str.split('e',2)
		
		['Th', ' Gr', 'en Revolution']

partition(sep)	The function partitions the strings at the first occurrence of separator, and returns the strings partition in three parts i.e. before the separator, the separator itself, and the part after the separator. If the separator is not found, returns the string itself, followed by two empty strings syntax:string_variable.partition (substring)	>>>str='The Green Revolution'
		
		>>> str.partition('Rev')
		
		('The Green ', 'Rev', 'olution')
		
		>>> str.partition('pe')
		
		('The Green Revolution', '', '')
		
		>>> str.partition('e')
		
		('Th', 'e', ' Green Revolution')

count(str,beg,end)	Count number of times str occurs in a string. You can specify be as 0 and end as the length of the message to search the entire string or use any other value to just search a part of the string. syntax: string_variable.count(str,beg,en	>>>str="he" >>>message="helloworldhellohello" >>>message.count(str,0,len(message)) 3

	d)	
find(str,beg,end)	Checks if str is present in the string. If found it returns the position at which str occurs in a string, otherwise returns -1. You can either set beg=0 and end equal to the length of the message to search entire string or use any other value to search a part of it. Syntax:string_variable.find(str, beg,end)	>>>message= “she is my best friend” >>>message.find(“my”,0,len(message)) 7 >>>message.find(“no”,0,len(message)) -1
index(str,beg,end)	Same as find but raises an exception if str is not found Syntax:string_variable.index(substring, beg, end)	>>>message= “she is my best friend” >>>message.index(“mine”,0,len(message)) valueError: substring not found
rfind(str,beg,end)	Same as find but starts searching from the end. Syntax: string_variable.rfind(str,beg,end)	>>>str= “Is this your bag?” >>>str.rfind(“is”,0,len(str)) 5
rindex(str,beg,end)	Same as index but start searching from the end and raises an exception if str is not found. Syntax: string_variable.rindex(str,beg,end)	>>>str= “Is this your bag?” >>>str.rindex(“you”,0,len(str)) 8
encode(encoding,errors)	This function is used to encode the string using the provided encoding. Syntax: string_variable.encode(encoding, errors)	>>str = "this is string example....wow!!!"; >>str.encode('base64','strict') dGhpcyBpcyBzdHJpbmcgZXhhbXBsZS4uLi53b3chISE=

decode(encoding,errors)	This function is used decode the encoded string. syntax: string_variable.decode(encoding,errors)	>>str = "this is string example....wow!!!"; >>str=str.encode('base64','strict') >>str.decode('base64','strict') this is string example....wow!!!";

String Constants:

The string module consists of a number of useful constants, classes and functions. These functions are used to manipulate strings.

String constants some constants defined in the string module are:

1) string.ascii_letters: The command/function displays a string containing uppercase characters and lowercase characters.

Example 16:

```
>>>import string

>>>print (string.ascii_letters)

'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

2) string.ascii_uppercase: The command/function displays a string containing uppercase characters.

Example 17:

```
>>>import string

>>>print (string.ascii_uppercase)

'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

3) string.ascii_lowercase: The command displays a string containing all lowercase characters.

Example 18:

```
>>>import string

>>>print (string.ascii_lowercase)

'abcdefghijklmnopqrstuvwxyz'
```

4) string.ascii_digits: The command displays a string containing digits.

Example 19:

```
>>>import string

>>>print (string.ascii_digits)

'0123456789'
```

5) string.hexdigits: The command displays a string containing hexadecimal characters.

Example 20:

```
>>>import string

>>>print (string.hexdigits)

'0123456789abcdefABCDEF'
```

6) string.octdigits: The command displays a string containing octal characters.

Example 21:

```
>>>import string

>>>print (string.octdigits)

'01234567'
```

7) string.punctuations: The command displays a string containing all the punctuation characters.

Example 22:

```
>>>import string

>>>print (string.punctuations)

'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}-'
```

8) string.whitespace: The command displays a string containing all ASCII characters that are considered whitespace. This includes the characters space, tab, linefeed, return, formfeed, and vertical tab.

Example 23:

```
>>>import string

>>>print (string.whitespace)

'\t\n\x0b\x0c\r’
```

9) string.printable: The command displays a string containing all characters which are considered printable like letters, digits, punctuations and whitespaces.

Example 23:

```
>>>import string

>>>print (string.printable)

'0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ! "#$%&\'()*+,-
./:;<=>?@[\\]^_`{|}- \t\n\r\x0b\x0c'
```

Regular expressions and Pattern matching

A Regular expression is a sequence of characters and special characters that helps to match or find strings in another strings. In python, regular expression can be accessed using the **re** module which comes as a part of the stand library.

Regular expressions are a powerful tool for various kinds of string manipulation. These are basically a special text string that is used to describing a search pattern to extract information from the text such as code, files, log, spreadsheets, or even documents.

In a regular expression special characters also called meta characters. The sequence formed by using meta characters and letters can be used to represent a group of patterns.

Example 24:

str= “Ram$”

The pattern “Ram$” is known as a regular expression. The expression has the meta character “$”. Meta character “$” is used to match the given regular expression at the end of the string. So the regular expression would match the string “SitaRam” or “HeyRam” but will not match the string “Raman”.

Consider the following codes:

import re	import re

string1='SitaRam'
 	
if re.search('Ram$',string1):
 	
        print "String Found"
 	
else :
 	
        print" No Match"
 	
Output:
 	
String Found
	string1='SitaRam'
	
	if re.search('Sita$',string1):
	
	        print "String Found"
	
	else :
	
	        print" No Match"
	
	Output
	
	No Match

As shown in the above examples, Regular expressions can be used in python for matching a particular pattern by importing the re module.

Note: re module includes functions for working on regular expression.

How the meta characters are used to form regular expressions.

S.No	Meta character	Usage	Example
1	[ ]	Used to match a set of characters.	[ram] The regular expression would match any of the characters r, a, or m. [a-z] The regular expression would match only lowercase characters.
2	^	Used to complementing a set of characters	[^ram] The regular expression would match any other characters than r, a or m.
3	$	Used to match the end of string only	Ram$ The regular expression would match Ram in SitaRam but will not

			match Ram in Raman
4	*	Used to specify that the previous character can be matched zero or more times.	wate*r The regular expression would match strings like watr, wateer, wateeer and so on.
5	+	Used to specify that the previous character can be matched one or more times.	wate+r The regular expression would match strings like water, wateer, wateeer and so on.
6	?	Used to specify that the previous character can be matched either once or zero times	wate?r The regular expression would only match strings like watr or water
7	{ }	The curly brackets accept two integer value s. The first value specifies the minimum no of occurrences and second value specifies the maximum of occurrences	Wate{1,4}r The regular expression would match only strings water, wateer, wateeer or wateeeer

Few Functions from re module:

1) re.match():

The match function is used to determine if the regular expression (RE) matches at the beginning of the string.

Example 25:

The following example demonstrate the use of match() function.

Import re

string=“She sells sea shells on the sea shore”

pattern=“sells”

```
if re.match(pattern,string):

        print (“Match found ”)

else:

        print (“Match not found”)

pattern1= “She”

if re.match(pattern,string):

        print (“Match Found”)

else:   print (“Match not found”)
```

In above program, “sells” is present in the string but still we got the output as match not found. This is because the re.match() function finds a match only at the beginning of the string.

2) re.search():

The search function is used to determine if the regular expression (RE) matches at anywhere in the string.

Example 26:

The following example demonstrate the use of search() function.

```
import re

string=“She sells sea shells on the sea shore”

pattern=“sells”

result= re.search(pattern,string)

if result:

        print (“Match found ”)
```

```
else:
        print (“Match not found”)
```

3) re.start()

The start function returns the starting position of the match.

4) re.end()

The end function returns the end position of the match.

Example 27:

The following example demonstrate the use of start() and end() function.

```
import re

string="She sells sea shells on the sea shore"

pattern="sells"

result= re.search(pattern,string)

if result:

        print (result.start())

        print (result.end())

else:

        print ("Match not found")
```

Output:

```
4

9
```

5) re.span() :

The span function returns the tuple containing the (start, end) positions of the match.

Example 28:

The following example demonstrate the use of span() function.

```
import re

string="She sells sea shells on the sea shore"

pattern="sells"

result= re.search(pattern,string)

if result:

        print (result.span())

else:

        print ("Match not found")
```

Output:

```
(4,9)
```

6) re.findall():

The function determines all substrings where the RE matches, and returns them as a list.

Example 29:

```
import re

string="hello my favorite words hellooooo world"

pattern="hell*o"

result= re.findall(pattern,string)

if result:
        print (result)

else:

        print ("Match not found")
```

Output:

```
['hello', 'hello']
```

7) re.finditer()

The function determines all substrings where the RE matches, and returns them as an iterator.

Example 30:

```
import re

string="hello my favorite words hellooooo world"

pattern="hell*o"

result= re.finditer(pattern,string)

if result:

    for i in result:

        print (i.span())

else:

    print ("Match not found")
```

Output:

```
(0, 5)

 (24, 29)
```

8) **re.group():**

The group function is used to return the string matched the RE.

Example 31:

```
import re

string = '39801 356, 2102 1111'

# Three digit number followed by space followed by two digit number

pattern = '(\d{3}) (\d{2})'

# match variable contains a Match object.

match = re.search(pattern, string)

if match:

        print(match.group())

        print (match.group(1))

        print (match.group(2))

        print (match.group(1,2))

        print (match.groups())

else:

        print("pattern not found")
```

Output:

```
801 35
```

```
801

35

('801', '35')

('801', '35')
```

Exercise:

1. Write a Python program to copy the string using for loop(using range and without using range function).
2. Write a program to print the following pattern.
   A
   AB
   ABC
   ABCDE
   ABCDEF
3. Write a python that takes user's name and PAN card number as input. Validate the information using is X function and print the details.
4. Write a program that encrypts a message by adding a key value to every character.
   Hint: say, if key=3, then add 3 to every character
5. Write a program to that uses split() to split multiline string.
6. Write a program that accepts a string from user and redisplays the same string after removing vowels from it.
7. Write a program that finds whether a given character is present in a string or not. In case it is present it prints the index at which it is present. Do not use built-in find functions to search the character.
8. Write a program that finds a character in a string from ending of string to beginning that like rfind function.
9. Write a python program that count the occurrences of a character in a string and also counting from the specified location . Do not use string count function.
10. Write a program to reverse of string.
11. Write a program to find whether the given string is palindrome string or not.
12. Write a Python program to get a string from a given string where all occurrences of its first char have been changed to '$', except the first char itself.
    Sample String : 'restart' Expected Result : 'resta$t
13. Write a Python program to add 'ing' at the end of a given string (length should be at least 3). If the given string already ends with 'ing' then add 'ly' instead. If the string length of the given string is less than 3, leave it unchanged.

14. Write a Python program to remove the characters which have odd index values of a given string.
15. Write a Python script that takes input from the user and displays that input back in upper and lower cases.
    Sample Output: What's your favourite language? english
    My favourite language is ENGLISH
    My favourite language is english
16. Write a Python function to reverse a string if it's length is a multiple of 4.
17. Write a Python program to count and display the vowels of a given text.
18. Write a Python program to lowercase first n characters in a string.
19. Write a Python program to reverse words in a string.
20. Write a Python program to count occurrences of a substring in a string.

Assignment –I

1. Write any 20 Functions from strings with syntax and example?
2. Write short notes on String Constants?

Unit-2

Lists:

List is data type available in python. It is a sequence in which elements are written as a list of comma separated values between square brackets. List is mutable data type which means the value of its elements can be changed.

Syntax:

list_variable=[var1,var2, ...]

Creating and initializing a list:

In Python programming, a list is created by placing all the items (elements) inside a square bracket [ ], separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

Examples:

```
>>L=[1,2,3,4,5]

>>print (L)

[1,2,3,4]

>>B=['a', 'b', 'c', 'd', 'e']

>>print (B)

['a', 'b', 'c', 'd', 'e']

# empty list

>>my_list = []

>>print (my_list)

[]

# list with mixed datatypes

>>my_list = [1, "Hello", 3.4]

>>print (my_list)

[1, "Hello", 3.4]

# nested list

>>my_list = ["mouse", [8, 4, 6], ['a']]

>>print (my_list)

["mouse", [8, 4, 6], ['a']]
```

Accessing an element of list:

The elements/values in a list are accessed using indexes. The index of the first element is 0 and that of the last element is n-1, where n is the total number of elements in the list. Trying to access a character out of index range will raise an IndexError. Like strings, can also use the slice, concatenation and repetition operations on lists. Python allows negative indexing for its sequences.

Example:

```
n_list = ["Happy", [2,0,1,5]]

print(n_list[1])

# Output: [2, 0, 1, 5]

print(n_list[0][1])

# Output: a

print(n_list[1][3])

# Output: 5

my_list = ['p','r','o','b','e']

print(my_list[-1])

# Output: e

print(my_list[-5])

# Output: p
```

Slice lists

We can access a range of items in a list by using the slicing operator (colon).
Syntax:
seq = L [start: stop: step]

Example

```
my_list = ['p','r','o','g','r','a','m','m','e']
print(my_list[2:5])
# Output: ['o', 'g', 'r']

print(my_list[:-5])
# Output: ['p', 'r', 'o', 'g']

print(my_list[5:])
# Output: ['a', 'm', 'm', 'e']
```

```
print(my_list[:])
# Output: ['p', 'r', 'o', 'g', 'r', 'a', 'm', 'm', 'e']
```

<u>change an element in a list/Updating values in a list:</u>

List are mutable, meaning, their elements can be changed unlike string or tuple. We can use assignment operator (=) to change an item or a range of items.

Example

```
even = [2, 4, 6, 8]
# change the 1st item
even[0] = 1
print(even)
# Output: [1, 4, 6, 8]

# change 2nd to 4th items
even[1:4] = [3, 5, 7]
print(even)
# Output: [1, 3, 5, 7]
```

Example

```
L=[1,2,3,4,5,6,7,8,9,10]
print ("List is", L)
L[5]=100
print ("List after updation is:",  L)
# The del statement is used to delete element(s) from the list.
del L[3]
print ("List after deleting a value is ", L)

 #Output:
List is: [1,2,3,4,5,6,7,8,9,10]
List after updation is: [1,2,3,4,5,100,7,8,9,10]
List after deleting a value is: [1,2,3,5,100,7,8,9,10]
```

Example:

Deletion of elements from a list using del statement

```
L= [1,2,3,4,5,6,7,8,9,10]
del L[2:4]      # deletes numbers at index 2 and 3
print (L)
#Output:
[1,2, 5,6,7,8,9,10]
```

```
L= [1,2,3,4,5,6,7,8,9,10]
del L[:]      # deletes all the elements from the list
print (L)
#Output:
[]
```

```
L= [1,2,3,4,5,6,7,8,9,10]
del L      # The entire list variable will be deleted. Try to use the deleted list variable, then an error will be generated.
print (L)
#Output:
Name Error: name L is not defined
```

Nested List:

Nested list means a list within another list.

Example:

Test=[1,2,3, "hari", [100,200,500], 8.5]

Traversing a List:

Traversing a list means accessing all the elements of the list one after the other. A list can be traversed using: for loop or while loop.

Example:

List traversal using for loop	List traversal using while loop

Program without using range function:
A=[1,2,3,4,5]
 	
for i in A:
 	
    print (i)
 	
Output:
1
 	
2
 	
3
 	
4
	Program:
	A=[1,2,3,4,5]
	
	i=0
	
	while (i<len(A)):
	
	    print (A[i])
	
	    i=i+1
	
	Output:
	1

5 **Program without using range function:** A=[1,2,3,4,5] for i in range(len(A)): print (A[i]) Output: 1 2 3 4 5	2 3 4 5

Note: Here **len( ) function** is used to get the length of list.

Lists as arguments:

When a list is passed to the function, the function gets a reference to the list. So if the function makes any changes in the list, they will be reflected back in the list.

Example

```
def add_Const (L):
 for i in range (len (L)):
            L [i] += 10
 X = [1, 2, 3, 4, 5]
add_Const (X)
 print (X)
```

Output:

```
 [11, 12, 13, 14, 15]
```

List operations:

Operator	Description	Example
+ (Concatenation)	Joins two lists	Example:

		>>[1,2,3,4,5]+[6,7,8,9,10] [1,2,3,4,5,6,7,8,9,10] >>l=[1,2,3] >>m=[4,6,8] >>x=l+m >>x [1,2,3,4,6,8]
*** (Repetition)**	Repeat elements in the list	Example : L= [1,2,3] print (3*L) Output: [1,2,3,1,2,3,1,2,3]
in (Membership)	Checks if the value is present in the list	>>> 3 in [5,6,7,9,15] False >>L=[1,2,3,4,5,6,7] >>4 in L True
not in	Checks if the value is not present in the list	>>> 3 not in [5,6,7,9,15] True >>L=[1,2,3,4,5,6,7] >>4 not in L False
Slice[n:m]	The Slice[n : m] operator extracts sub parts from the list.	>>>A=[1,2,3,4,5,6] >>> print A[1:3]

		[2,3] The print statement prints the sublist starting from subscript 1 and ending at subscript 3 but not including subscript 3

List Functions:

Function	Description	Example
len()	Returns length of list syntax: len(list)	Example: >>a=[1,5,7,20,2,44] >>len(a) 6
max()	Returns maximum value in the list syntax:max(list)	Example : >>L= [1,3,5,20,55,60,4,33] >>max(L) 60
min	Returns minimum value in the list syntax: min(list)	Example : >>L= [1,3,5,20,55,60,4,33] >>min(L) 1
sum	Add the values in the list that has only numbers syntax: sum(list)	>>L=[1,2,3,4,5,6,7,8,9,10] >>sum(L) 55
all	Returns True if all elements of the list are true	>>L=[1,0,20,60]

	syntax: all(list)	>>all(L) False
any	Returns True if any element of the list is true. syntax: any(list)	>>L=[6,3,7,0,1,2,4,9] >>any(L) True
list	Converts tuple or strings or set or dictionary to a list syntax: list(tuple/string/sets/dictionary)	>>x= "Hello" >>L=list(x) >>L ['H', 'e', 'l', 'l', 'o']
sorted	Returns a new sorted list. syntax: sorted(list)	>>L=[3,4,1,2,5] >>X=sorted(L) >>X [1,2,3,4,5]

List Methods:

Method	Description	Example
append	Appends an element to the end of the list. Syntax: list.append(element)	>>a=[1,5,7,20,2,44] >>a.append(66) >>a [1,5,7,20,2,44,66]
count	Counts the number of times an element appears in the list. Syntax:list.count(element)	>>a=[1,5,7,20,5,44,66,5] >>a.count(5) 3
index	Returns the index value of element in the list. If the element is present more than once, it returns lowest index value of the element in the list. Gives a ValueError if	>>a=[1,5,7,20,5,44,66,5] >>a.index(5)

	element is not present in the list. syntax: list.index(element)	1
insert	Insert element at the specified index in the list. Syntax: list.insert(index,element)	>>a=[1,5,7,20,2,44] >>a.insert(2,100) >>a [1,5,100,7,20,2,44,66]
pop	Removes the last element from the list. And also removes the element at the specified index from the list if index value is provided. Index is optional parameter. Syntax: list.pop()	>>L=[1,0,20,60,85,100] >>L.pop() >>L [1,0,20,60,85] >> L.pop(2) >>L [1,0,60,85]
remove	Removes or deletes element from the list. If multiple elements present in the list, then the first element will be deleted. Syntax: list.remove(element)	>>L=[6,3,7,0,1,2,4,9] >>L.remove(9) >>L [6,3,7,0,1,2,4]
reverse	Reverse the elements in the list. Syntax: list.reverse()	>>L=[6,3,7,0,1,2,4,9] >>L.reverse() >>L [9,4,2,1,0,7,3,6]
sort	Sorts the elements in the list. Syntax: list.sort()	>>L=[3,4,1,2,5] >>L.sort() >>L [1,2,3,4,5]

extend	Add the elements in the list to the end of another list. Using + is on list is similar to extend() Syntax:list1.extend(list2)	>>L=[1,2,3] >>m=[10,20,30] >>L.extend(m) >>L [1,2,3,10,20,30]

Example: **Write a python program product of matrices.**

```
l=int(input("Enter first matrix row size: "))
m=int(input("Enter first matrix col size: "))
x=int(input("Enter second matrix row size: "))
y=int(input("Enter second matrix col size: "))
if m==x:
    A=[]
    for i in range(l):
        C=[]
        for j in range(m):
            C.append(int(input("Enter A matrix element: ")))
        A.append(C)
    print ("A matrix is", A)
    B=[]
    for i in range(x):
        C=[]
        for j in range(y):
            C.append(int(input("Enter B matrix element: ")))
        B.append(C)
    print ("B matrix is", B)
    R=[]
    for i in range(l):
        C=[]
        for j in range(y):
            C.append(0)
        R.append(C)
    for i in range(l):
        for j in range(y):
            for k in range(m):
                R[i][j]=R[i][j]+A[i][k]*B[k][j]
    print ("Result Matrix is", R)
else:
    print ("Matrix multiplication is not possible")
```

Example: Write a python program add two matrices.(windows version)

```
l=int(input("Enter first matrix row size: "))
m=int(input("Enter first matrix col size: "))
x=int(input("Enter second matrix row size: "))
y=int(input("Enter second matrix col size: "))
if l==x and m==y:
    A=[]
    for i in range(l):
        C=[]
        for j in range(m):
            C.append(int(input("Enter A matrix element: ")))
        A.append(C)
    print ("A matrix is", A)
    B=[]
    for i in range(x):
        C=[]
        for j in range(y):
            C.append(int(input("Enter B matrix element: ")))
        B.append(C)
    print ("B matrix is", B)
    R=[]
    for i in range(l):
        C=[]
        for j in range(y):
            C.append(0)
        R.append(C)
    for i in range(l):
        for j in range(y):
                R[i][j]=A[i][j]+B[i][j]
    print ("Result Matrix is", R)
else:
    print ("Matrix Addition is not possible")
```

Exercise:

1. Create a list named wordList that contains the following words: "washington", "lee", "generals", "arlington", "RGUKT" and perform the following operations.
   i) Print out the first element in wordList
   ii) Print out the length of the list?
   iii) Print out the last element in wordList?
   iv) Print out the index of the word "generals" in the list?
2. Write a program to that creates a list of numbers from 1-20 that are either divisible by 2 or 4.
3. Write a program to generate a list of squares from 1-10.
4. Write a python program the defines a list of countries that are a member of BRICS. Check whether a country is a member of BRICS or not.
5. Write a python 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.
6. 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 the value is repeated in the list.

7. Write a program that creates a list of words by combining the words in two individual lists.
8. Write a program that forms a list of first character of every word in the another list.
9. Write a python program to remove all duplicates from the a list.
10. Write a program that creates a list of 10 random integers. Then create two lists-odd list even list that has all odd and even values in the list.

Assignment II

1. Define list. How to create a list with syntax and example.
2. Write list functions with syntax and example.
3. Write list methods with syntax and example.
4. Write a python program add two matrices.
5. Write a python program product of matrices.

Unit-3

Tuples:

Tuples is data type available in python. Tuples are sequence of elements separated by comma enclosed in parenthesis. They are immutable data type which means values cannot be changed in tuple.

Syntax:

tuple_variable=(val1,val2,…..)

where val can be integer, floating number, character or string.

Creating and initializing a tuple:

In Python programming, a tuple is created by placing all the items (elements) inside a parenthesis (), separated by commas.

It can have any number of items and they may be of different types (integer, float, string etc.).

Examples:

```
>>L=(1,2,3,4,5)

>>print (L)
```

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

>>B=(‘a’, ‘b’, ‘c’, ‘d’, ‘e’)

>>print (B)

(‘a’, ‘b’, ‘c’, ‘d’, ‘e’)

# empty list

>tup = ()

>>print (tup)

()

# tuple with mixed datatypes

>>tup1 = (1, "Hello", 3.4)

>>print (tup1)

(1, "Hello", 3.4)

# nested tuple

>>mytup= ("mouse", (8, 4, 6), ('a’, ‘b’))

>>print (mytup)

("mouse", (8, 4, 6), ('a’, ‘b’))
```

Note: Single element of a tuple

Create a tuple with a single element, must add a coma after the element. If the absence of comma, Python treats the element as an ordinary value and not consider as tuple.

Example tup=(1,) #comma after first element	Example tup=(1) #comma missing

print (type(tup)) Output: <type ‘tuple’>	print (type(tup)) Output: <type ‘int’>

Accessing an element of tuple:

The elements/values in a tuple are accessed using indexes. The index of the first element is 0 and that of the last element is n-1, where n is the total number of elements in the tuple. Trying to access a character out of index range will raise an IndexError. Like strings or lists, can also use the slice, concatenation and repetition operations on tuple. Python allows negative indexing for its sequences.

Example:

```
N = ("Happy", (2,0,1,5))

print(N[1])

# Output: (2, 0, 1, 5)

print(N[0][1])

# Output: a

print(N[1][3])

# Output: 5

tup = ('p','r','o','b','e')

print(tup[-1])

# Output: e

print(tup[-5])

# Output: p
```

Slice tuple

We can access a range of items in a tuple by using the slicing operator (colon - :).
Syntax:

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

Example

```
my_tuple = ('p','r','o','g','r','a','m')
print(my_tuple[1:4])
# Output: ('r', 'o', 'g')

print(my_tuple[:-5])
# Output: ('p', 'r')

print(my_tuple[5:])
# Output: ('a', 'm')

print(my_tuple[:])
# Output: ('p', 'r', 'o', 'g', 'r', 'a', 'm')
```

change an element in a tuple/Updating values in a tuple:

Unlike lists, tuples are immutable.This means that elements of a tuple cannot be changed once it has been assigned. But, if the element is itself a mutable datatype like list, its nested items can be changed. We can also assign a tuple to different values (reassignment).

Example

```
my_tuple = (4, 2, 3, [6, 5])
my_tuple[3][0] = 9
print(my_tuple)
#Ouptput
(4, 2, 3, [9, 5])

my_tuple[1]=40  #TypeError: 'tuple' object does not support item assignment
print(my_tuple)
```

Note: Tuple is an immutable data type. That cannot allow to delete values from tuple.

Example:

```
Tup=(1,2,3,4,5)
del (Tup[2])
print (Tup)
#Ouptput
TypeError: tuple object doesn’t support item deletion
```

Note: we cannot delete or remove items from a tuple.But deleting a tuple entirely is possible using the keyword del.

Example:

```
Tup=(1,2,3,4,5)
del (Tup)
print (Tup)
#Ouptput
TypeError: name Tup is not defined.
```

<u>Nested tuple:</u>

Python allows to define a tuple inside a another tuple. This is is called a nested tuple.

Example:

```
Test=(1,2,3, "hari", (100,200,500), 8.5)
for i in Test:
        print (i)
#Ouptput
1
2
3
hari
(100, 200, 500)
8.5
```

<u>Tuple Assignment:</u>

If we want to interchange (swap) any two variable values, we have to use temporary variable. For example;

```
>>> A=10
>>> B=20
>>> print A,B
10  20
>>> T=A
>>> A=B
>>> B=T
>>> print A,B
20 10
```

But in python, tuple assignment is more elegant:

Example

```
>>> T1=(10,20,30)
>>> T2=(100,200,300,400)
>>> print T1
(10, 20, 30)
>>> print T2
(100, 200, 300, 400)
>>> T1,T2=T2,T1      # swap T1 and T2
>>> print T1
(100, 200, 300, 400)
>>> print T2
(10, 20, 30)
```

The left side is a tuple of variables, while the right side is a tuple of expressions. Each value is assigned to its respective variable. All the expressions on the right side are evaluated before any of the assignments. The number of variables on the left and the number of values on the right have to be the same:

Example

```
 >>> T1=(10,20,30)
>>> T2=(100,200,300)
>>> t3=(1000,2000,3000)
>>> T1,T2=T2,T1,t3
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
T1,T2=T2,T1,t3
ValueError: too many values to unpack
```

Here, two tuples are in the left side and three tuples are in right side. That is why, we get errors. Thus, it is required to have same number of tuples in both sides to get the correct result.

Example

```
>>> T1,T2,t3=t3,T1,T2
>>> print T1
(1000, 2000, 3000)
>>> print T2
(10, 20, 30)
>>> print t3
(100, 200, 300)
```

Tuple operations:

Operator	Description	Example

+ (Concatenation)	Joins two tuples	Example: >>(1,2,3,4,5)+(6,7,8,9,10) (1,,2,3,4,5,6,7,8,9,10) >>l=(1,2,3) >>m=(4,6,8) >>x=l+m >>x (1,2,3,4,6,8)
*** (Repetition)**	Repeat elements in the tuple	Example : L= (1,2, “Good”) print (3*L) Output: (1,2, “Good”, 1,2, “Good”, 1,2, “Good”)
in (Membership)	Checks if the value is present in the tuple	>>> 3 in (5,6,7,9,15) False >>L=(1,2,3,4,5,6,7) >>4 in L True
not in	Checks if the value is not present in the tuple	>>> 3 not in (5,6,7,9,15) True >>L=(1,2,3,4,5,6,7) >>4 not in L False

Slice[n:m]	The Slice[n : m] operator extracts sub parts from the tuple.	>>>A=(1,2,3,4,5,6) >>> print A[1:3] (2,3) The print statement prints the subtuple starting from subscript 1 and ending at subscript 3 but not including subscript 3

Tuple Functions:

Function	Description	Example
len()	Returns length of tuple syntax: len(tuple)	Example: >>a=(1,5,7,20,2,44) >>len(a) 6
max()	Returns maximum value in the tuple syntax: max(tuple)	Example : >>t= (1,3,5,20,55,60,4,33) >>max(t) 60
min()	Returns minimum value in the tuple syntax: min(tuple)	Example : >>t= (1,3,5,20,55,60,4,33) >>min(t) 1
sum()	Return the sum of all elements in the tuple syntax: sum(tuple)	>>t=(1,2,3,4,5,6,7,8,9,10) >>sum(t) 55

all()	Returns True if all elements of the tuple are true syntax: all(tuple)	>>t=(1,0,20,60) >>all(t) False
any()	Returns True if any element of the tuple is true. syntax: any(tuple)	>>t=(6,3,7,0,1,2,4,9) >>any(t) True
tuple()	Converts list or strings or set or dictionary to a tuple syntax: tuple(list/strings/sets/dictionary)	>>x= "Hello" >>t=tuple(x) >>t ('H', 'e', 'l', 'l', 'o')
sorted()	Returns a new sorted tuple. syntax: sorted(tuple)	>>t=(3,4,1,2,5) >>X=sorted(t) >>X (1,2,3,4,5)
reversed()	It returns the reversed of the object. syntax: reversed(tuple)	>>a=(1,2,3,4,5) >>tuple(reversed(tuple)) (5,4,3,2,1)

cmp()	This function is used to check whether the given tuples are same or not. If both are same, it will return 'zero', otherwise return 1 or -1. If the first tuple is big, then it will return 1, otherwise return -1.
	syntax: cmp(t1,t2)
		>>> T1=(10,20,30)
		>>> T2=(100,200,300)
		>>> T3=(10,20,30)
		>>> cmp(T1,T2)
		-1
		>>> cmp(T1,T3)
		0
		>>> cmp(T2,T1)
		1

Tuple Methods:

Method	Description	Example

count	Counts the number of times an element appears in the tuple. Syntax:tuple.count(element)	>>a=(1,5,7,20,5,44,66,5) >>a.count(5) 3
index	Returns the index value of element in the tuple. If the element is present more than once, it returns lowest index value of the element in the tuple. Gives a ValueError if element is not present in the list. syntax: tuple.index(element)	>>a=(1,5,7,20,5,44,66,5) >>a.index(5) 1

Exercise:

1. Write a Python program to create a tuple.
2. Write a Python program to create a tuple with different data types.
3. Write a Python program to create a tuple with numbers and print tuple.
4. Write a Python program to add an item in a tuple.
5. Write a Python program to convert a tuple to a string.
6. Write a Python program to get the 4th element and 4th element from last of a tuple.
7. Write a Python program to find the repeated items of a tuple.

Example: tuplex = 2, 4, 5, 6, 2, 3, 4, 4, 7

Output: 4 is 3 times

8. Write a Python program to check whether an element exists within a tuple.
9. Write a Python program to remove an item from a tuple.
10. Write a Python program to slice a tuple.
11. Write a Python program to find the index of an item of a tuple.
12. Write a Python program to find the length of a tuple.
13. Write a Python program to reverse a tuple.
14. Write a program to swap two values using tuple assignment.
15. Write a program that has a nested tuple to store topper details. Edit the details and reprint the details.
16. 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.

Assignment III

1) Define tuple. Explain tuple functions and methods with syntax and example.
2) Write a program that has a nested tuple to store topper details. Edit the details and reprint the details.

Unit-4

Dictionaries:

Dictionary is a data type available in python which store values as a pair of key and value. Each key is separated from its value by a colon (:), and consecutive items are separated by commas. The entire items in a dictionary are enclosed in curly brackets {}. Dictionary keys must be of immutable type and must be unique.

Syntax: my_dict = {'key1': 'value1','key2': 'value2','key3': 'value3'…'keyn': 'valuen'}

A dictionary as a mapping between a set of indices (which are called keys) and a set of values. Each key maps a value. The association of a key and a value is called a **key-value pair**.

Example:

```
>>> A={1:"one",2:"two",3:"three"}

 >>> print A

 {1: 'one', 2: 'two', 3: 'three'}
```

In the above example, we have created a list that maps from numbers to English words, so the keys values are in numbers and values are in strings.

Example

```
 >>>computer={'input':'keybord','output':'mouse','language':'python','os':'windows8',}

 >>> print computer

 {'input': 'keyboard', 'os': 'windows-8', 'language': 'python', 'output': 'mouse'}
```

In the above example, we have created a list that maps from computer related things with example, so here the keys and values are in strings. The order of the key-value pairs is not in same order (ie. input and output orders are not same). We can get different order of items in different computers. Thus, the order of items in a dictionary is unpredictable.

Example

```
>>>
D={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursday’,'fri':'Friday','sat':'Saturday'}
>>> print D
{'wed': 'Wednesday', 'sun': 'Sunday', 'thu': 'Thursday', 'tue': 'Tuesday', 'mon': 'Monday', 'fri': 'Friday', 'sat': 'Saturday'}
```

<u>Creation, initializing and accessing the elements in a Dictionary:</u>

In Python programming, Creating a dictionary is as simple as placing items inside curly braces {} separated by comma. An item has a key and the corresponding value expressed as a pair, key: value. The function dict ( ) is used to create a new dictionary with no items. This function is called built-in function. Also create a empty dictionary as {}.

Example:

```
>>> D=dict()
>>> print D
{}
```

{} represents empty dictionary. To add an item to the dictionary (empty string), we can use square brackets for accessing and initializing dictionary values.

Example

```
>>> H=dict()
>>> H["one"]="keyboard"
>>> H["two"]="Mouse"
>>> H["three"]="printer"
>>> H["Four"]="scanner"
>>> print H
```

```
{'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
```

Example

```
# empty dictionary
my_dict = {}
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
```

Accessing the elements in a Dictionary:

While indexing is used with other container types to access values, dictionary uses keys. Key can be used either inside square brackets or with the get() method. The difference while using get() is that it returns None instead of KeyError, if the key is not found.

Example

```
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
print my_dict['name']#John
print my_dict.get('age')#None
print my_dict['age']#Keyerror
```

Traversing a dictionary:

Access each element of the dictionary can be done by using 'for' loop only.

Example:

```
 H={'Four': 'scanner', 'three': 'printer', 'two': 'Mouse', 'one': 'keyboard'}
 for i in H:
        print ( i,":", H[i])
```

output:

```
Four: scanner
one: keyboard
three: printer
```

two: Mouse

Appending values to the dictionary and Update element in a Dictionary:

Dictionary are mutable. We can add new items or change the value of existing items using assignment operator. If the key is already present, value gets updated, else a new key: value pair is added to the dictionary.

Syntax: Dictionary_name [key]=value

Example

```
>>> a={"mon":"monday","tue":"tuesday","wed":"wednesday"}
>>> a["thu"]="thursday"
>>> print a
{'thu': 'thursday', 'wed': 'wednesday', 'mon': 'monday', 'tue': 'tuesday'}
```

Example

```
my_dict = {'name': 'John', 1: [2, 4, 3]}
# update value
my_dict['name'] = 'jack'
print(my_dict)
Output:
{'name': 'Jack', 1: [2, 4, 3]}
```

Example

```
{'name': 'Jack', 1: [2, 4, 3]}
# add item
my_dict['address'] = 'Downtown'
print(my_dict)
Output:
{'name': 'Jack', 1: [2, 4, 3], ‘address’: ‘Downtown’}
```

Example

{'name': 'Jack', 1: [2, 4, 3], 'address': 'Downtown'}

my_dict[1][2]=10

print (my_dict)

Output:

{'name': 'Jack', 1: [2, 4, 10], 'address': 'Downtown'}

Removing/Delete an item from dictionary

Remove item from the existing dictionary by using del key word.

Syntax: del dicname[key]

Example

```
>>> A={"mon":"monday","tue":"tuesday","wed":"wednesday","thu":"thursday"}
 >>> del A["tue"]
 >>> print A
 {'thu': 'thursday', 'wed': 'wednesday', 'mon': 'monday'}
```

Dictionary Functions:

Function	Description	Example

len()	Returns length of dictionary. That is number of items(key:value pair)
	syntax: len(Dictionary_name)
		Example:
		
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>len(D)
		
		3

cmp()	This is used to check whether the given dictionaries are same or not. If both are same, it will return 'zero', otherwise return 1 or -1. If the first dictionary having more number of items, then it will return 1, otherwise return -1. syntax: cmp(d1,d2) #d1and d2 are dictionary.	Example : >>D1={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursday','fri':'Friday','sat':'Saturday'} >>D2={'sun':'Sunday','mon':'Monday','tue':'Tuesday','wed':'Wednesday','thu':'Thursd

		ay','fri':'Friday','sat':'Saturday'}
		
		>>>D3={'mon':'Monday','tue':'Tuesday','wed':'Wednesday'}
		
		>>> cmp(D1,D3) #both are not equal
		1
		
		>>> cmp(D1,D2) #both are equal
		
		0
		
		>>> cmp(D3,D1)
		
		-1

Dictionary Methods:

Method	Description	Example

clear()	Delete all items in the dictionary.
	
	Syntax: dictionary.clear()
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.clear()
		
		>>print (D)
		
		{}

get()	Returns the value for the key passed as argument. If the key is not present in dictionary, it will return None.
	Syntax: dictionary.get(key)
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.get('ID')
		
		B181234

has_key()	Returns True if the key is present in the dictionary and False otherwise.
	syntax: dictionary.has_key(key)
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.has_key('Marks')
		
		False

items()	Returns a list of tuples syntax: dictionary.items()	>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}

		>>D.items() [('ID':'B181234'),('name': 'Rajesh'), ('Course': 'BTech')]

keys()	Returns a list of keys in the dictionary.
	
	Syntax: dictionary.keys()
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.keys()
		
		['ID', 'name', 'Course']

values()	Returns a list of values in the dictionary.
	syntax: dictionary.values()
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.values()
		
		['B181234', 'Rajesh, 'BTech']

pop()	Remove a particular item in a dictionary.
	syntax: dictionary.pop(key)
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.pop('name')
		
		{ 'ID':'B181234','Course': 'BTech'}

popitem()	Remove a item from a dictionary.
	
	Syntax: dictionary.popitem()
		>>D={'ID':'B181234','name': 'Rajesh', 'Course': 'BTech'}
		
		>>D.popitem()
		
		{'name': 'Rajesh', 'Course': 'BTech'}

fromkeys()	Create a new dictionary with keys from seq and values set to val. If no value is specified then, None is assigned as default value. syntax: dict.fromkey(sequence,value)	Subjects=['Tel','Hin','Eng',' Mat','Phy','Che','IT'] marks=dict.fromkeys(Subjects,-1) print (marks) Output: {'Tel': -1, 'Che': -1, 'Eng': -1,

		'IT': -1, 'Phy': -1, 'Hin': -1, 'Mat': -1}

update()	Adds the key-value pairs of Dictionary2 to the key-value pairs of Dictionary1
	syntax: dictionary1.update(dictionary2)
		d1={'Tel': -1, 'Che': -1, 'Eng': -1, 'IT': -1, 'Phy': -1, 'Hin': -1, 'Mat': -1}
		
		d2={'id':'B182134','NAME':'RAJESH'}
		
		d1.update(d2)
		
		print (d1)
		
		Output:
		{'Tel': -1, 'Che': -1, 'Eng': -1, 'IT': -1, 'Phy': -1, 'NAME': 'RAJESH', 'id': 'B182134', 'Hin': -1, 'Mat': -1}

Exercise:

1. Write a Python script to add key to a dictionary?

Sample Dictionary : {0: 10, 1: 20}

Expected Result : {0: 10, 1: 20, 2: 30}

```
d = {0:10, 1:20}

print(d)

d.update({2:30})

print(d)
```

2. Write a Python script to concatenate following dictionaries to create a new one?

Sample Dictionary :
dic1={1:10, 2:20}
dic2={3:30, 4:40}
dic3={5:50,6:60}
Expected Result : {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

```
dic1={1:10, 2:20}
```

```
dic2={3:30, 4:40}

dic3={5:50,6:60}

dic4 = {}

for d in (dic1, dic2, dic3):

            dic4.update(d)

print(dic4)
```

3. Write a Python script to check if a given key already exists in a dictionary?

```
d = {1: 10, 2: 20, 3: 30, 4: 40, 5: 50, 6: 60}

x=input("Enter a key")

if x in d:

          print('Key is present in the dictionary')

else:

          print('Key is not present in the dictionary')
```

4. **Write a Python script to print a dictionary where the keys are numbers between 1 and 15 (both included) and the values are square of keys?**
Sample Dictionary
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100, 11: 121, 12: 144, 13: 169, 14: 196, 15: 225}

```
n=input("Enter a number: ")

d=dict()

for x in range(1,n):

          d[x]=x**2

print(d)
```

5. **Write a Python program to remove duplicates from Dictionary.**

Sample input:

```
student_data = {'id1':

  {'name': ['Sara'],

   'class': ['V'],
   'subject_integration': ['english, math, science']
  },
 'id2':
  {'name': ['David'],
   'class': ['V'],
   'subject_integration': ['english, math, science']
  },
 'id3':
   {'name': ['Sara'],
   'class': ['V'],
   'subject_integration': ['english, math, science']
  },
 'id4':
  {'name': ['Surya'],
   'class': ['V'],
   'subject_integration': ['english, math, science']
  },
}
```

Output:

```
{'id2': {'subject_integration': ['english, math, science'], 'class': ['V'], 'name': ['David']}, 'id4': {'subje
ct_integration': ['english, math, science'], 'class': ['V'], 'name': ['Surya']}, 'id1': {'subject_integration'
: ['english, math, science'], 'class': ['V'], 'name': ['Sara']}}
```

6. **Write a Python program to multiply all the items in a dictionary.**

Sample:

my_dict = {'data1':100,'data2':54,'data3':247}

7. Write a Python program to iterate over dictionaries using for loops.

Sample: d = {'Red': 1, 'Green': 2, 'Blue': 3}
Output:

```
Red corresponds to  1

Blue corresponds to  3

Green corresponds to  2
```

8. Write a Python program to sort a dictionary by key.

9. Write a Python program to create a dictionary from a string.

Note: Track the count of the letters from the string.

Sample string : 'rguktbasar'

Expected output: {'r': 2, 'g': 1, 'u': 1, 'k': 1, 't': 1, 'b': 1, 'a': 2, 's': 1}

10. 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.

Assignment IV

1) Define dictionary and write a notes on dictionary functions and methods.

2) 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.

Unit-5

Sets:

Set is an unordered collection of values, of any type, with no duplicate entry. Sets are immutable (which cannot be changed). However, the set itself is mutable. We can add or remove items from it. Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

Creating a set:

A set is created by placing all the elements inside curly braces {}, separated by comma or by using the built-in function set().

Syntax:

set_variable={val1,val2,…}

Example:

```
s={1,2.0, “abc”}
```

A set can have any number of items and they may be of different data types(integer, float, tuple, string etc). But a set cannot have a mutable element, like list, set or dictionary, as its element.

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

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:

```
s=set([1,2, ‘a’, ‘b’, “def”, 55])
print (s)
```

output:

```
set([1,2, ‘a’, ‘b’, “def”, 55])
```

Example:

```
my_set = {1, 2, 3}
print(my_set)

# set of mixed datatypes
my_set = {1.0, "Hello", (1, 2, 3)}
print(my_set)
```

Output:

```
{1, 2, 3}
{1.0, 'Hello', (1, 2, 3)}
```

Example:

```
# set do not have duplicates
# Output: {1, 2, 3, 4}
my_set = {1,2,3,4,3,2}
print(my_set)
```

```
# set cannot have mutable items
# here [3, 4] is a mutable list
# If you uncomment line,
# this will cause an error.
# TypeError: unhashable type: 'list'
my_set = {1, 2, [3, 4]}

# we can make set from a list
# Output: {1, 2, 3}
my_set = set([1,2,3,2])
print(my_set)
output:
{1, 2, 3, 4}
{1, 2, 3}
```

Creating an empty set

Empty curly braces {} will make an empty dictionary in Python. To make a set without any elements we use the set() function without any argument.

Example:

```
# initialize a with {}
a = {}

# check data type of a
# Output: <class 'dict'>
print(type(a))

# initialize a with set()
a = set()

# check data type of a
```

Output: <class 'set'>

print(type(a))

<u>Change a set in Python</u>:
Sets are mutable. But since they are unordered, indexing have no meaning.

We cannot access or change an element of set using indexing or slicing. Set does not support it.

We can add single element using the add() method and multiple elements using the update() method. The update() method can take tuples, lists, strings or other sets as its argument. In all cases, duplicates are avoided.

Example:
s={1,2,3,4,5}

Print (s[0])

Output:
'set' object does not support indexing.

<u>Traversing a set:</u>
We can iterate through each item in a set using a for loop.
Example:

```
s=set("Hello every one, Good morning")
for i in s:

        print (i, end= ' ')
```

Set Operations:

Operator	Description	Example
in (Membership)	Returns True is element is present in set otherwise False.	>>> s=set([1,2,3,4,5]) >>(3 in s) True
not in	Returns True is element is not present in set otherwise False.	>>> s=set([1,2,3,4,5]) >>(3 in s) True

== and !=	== returns True if the two sets are equivalent and False otherwise. != returns True if both sets are not equivalent and False otherwise.	>>>A=set([‘a’, ‘b’, ‘c’]) >>>B=set(“abc”) >>>C=set(tuple(‘abc’)) >>>print (A==B) True >>>print (A!=C) False

<u>Set Functions:</u>

Function	Description	Example
len()	Returns length of set syntax: len(set)	Example: >>a=set([1,5,7,20,2,44]) >>len(a) 6
max()	Returns maximum value in the set syntax: max(set)	Example : >>s= set([1,3,5,20,55,60,4,33]) >>max(s) 60
min()	Returns minimum value in the set syntax: min(set)	Example : >>s= set([1,3,5,20,55,60,4,33]) >>min(s) 1
sum()	Return the sum of all elements in the set syntax: sum(set)	>>s=set([1,2,3,4,5,6,7,8,9,10]) >>sum(s)

		55
all()	Returns True if all elements of the sets are true otherwise False syntax: all(set)	>>s=set([1,0,20,60]) >>all(s) False
any()	Returns True if any element of the set is true. Returns False if the set is empty. syntax: any(tuple)	>>s=set([6,3,7,0,1,2,4,9]) >>any(s) True
sorted()	Returns a new sorted list list from elements in the set. It does not sorts the set as sets are immutable.	>>t=set([3,4,1,2,5]) >>X=sorted(t) >>X [1,2,3,4,5]

Set Methods

Method	Description	Example

add(x)	Add element x to the set.
	Syntax: set_variable.add(x)
		s=set([1,2,3,4,5])
		s.add(6)
		print (s)
		output:
		set([1,2,3,4,5,6])

remove(x)	Removes element x from the set. Returns keyerror if element x is not present.
	syntax: set_variable.remove(x)
		s=set([1,2,3,4,5])
		s.remove(3)
		print (s)

		output: set([1,2,4,5])

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

pop()	Removes element from the set.
	syntax: set_variable.pop()
		s=set([1,2,3,4,5])
		
		s.pop()
		
		print (s)
		
		output:
		
		set([2,3,4,5])

clear()	Remove all elements from the set
	syntax: set_variable.clear()
		s=set([1,2,3,4,5])
		
		s.clear()
		
		print (s)
		
		output:
		
		set()

update()	Add elements of set B in the set A provided that all duplicates are avoided.
	syntax: set1.update(set2)
		A=set([1,2,3,4,5])
		B=set([6,7,8,9,10])
		A.update(B)
		print (A)
		OUTPUT:
		set([1,2,3,4,5,6,7,8,9,10])

copy()	Returns a copy of set.
	syntax: set.copy()
		s=set([1,2,3,4,5])
		
		x=s.copy()
		
		print (x)
		
		output:
		set([1,2,3,4,5])

issubset() or <=	Return True if every element in Set A is present in Set B and False otherwise. syntax: set1.issubset(set2) or set1<=set2	A=set([1,2,3,4,5]) B=set([1,2,3,4,5,6,7,8,9,10]) print (A.issubset(B)) OUTPUT: True
issuperset() or >=	Return True if every element in Set B is present in Set A and False otherwise. syntax: set1.issuperset(set2) or set1>=set2	A=set([1,2,3,4,5]) B=set([1,2,3,4,5,6,7,8,9,10]) print (A.issuperset(B)) OUTPUT: False

**union()
or
|**
	Returns a set that has elements from both sets A and B
	Syntax: set1.union(set2) or set1 | set2
		A=set([1,2,3,4,5])
		B=set([1,2,3,4,5,6,7,8,9,10])
		print (A.union(B))
		OUTPUT:
		set([1,2,3,4,5,6,7,8,9,10])

**intersection()
or
&**
	Returns a new set that has elements which are common to both the set A and Set B
	syntax: set1.intersection(set2) or set1&set2
		A=set([1,2,3,4,5])
		B=set([1,2,3,4,5,6,7,8,9,10])
		print (A.intersection(B))
		OUTPUT:
		set([1,2,3,4,5])

**intersection_updat
e()**
	Returns a set that elements which are common to both set A and Set B and result will be updated in set A.
	Syntax: set1.intersection_update(set2)
		A=set([1,2,10,12)
		B=set([1,2,3,4,5,6,7,8,9,10])
		A.intersection_update(B)
		print (A)
		OUTPUT:
		set([1,2,10])

**difference()
or
 	
-**
	Returns a new set has elements in Set A but not Set B
	Syntax: set1.difference(set2) or set1-set2
		A=set([1,2,10,12)
		B=set([1,2,3,4,5,6,7,8,9,10])
		print (A.difference(B))
		OUTPUT:
		set([12])

**difference_update(
)**
	Returns a set that has elements in set A but not Set B. The result set updated in Set A
	syntax: set1.difference_update(set2)
		A=set([1,2,10,12)
		B=set([1,2,3,4,5,6,7,8,9,10])
		A.difference_update(B)
		print (A)
		OUTPUT:
		set([12])

symmetric_difference() or ^	Returns a new set with elements either in A or in B but not both.
	syntax: set1.symmetric_difference(set2)
	or
	set1^set2
		A=set([1,2,10,12)
		B=set([1,2,3,4,5,6,7,8,9,10])
		print
		(A.symmetric_difference(B))
		
		OUTPUT:
		set([3,4,5,6,7,8,9,12])

Exercise:

1. Write a program to perform union, intersection, difference and symmetric difference operation on two sets. Take one set as prime numbers and second set as odd numbers.

2. 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.

3. Write a program that perform update(), pop(), remove(), add() and clear() operations on two sets.

Assignment V

1) Define set. Write set functions and methods with syntax and example.

Unit-6

Searching and Sorting:

Searching is a process of finding an object among a group of objects. Here the element is also referred to as 'Key'. Searching is one of the important application of Arrays i.e., we can search a particular element present in a list or in a record.

There are 2 important Searching Techniques.

1. Linear Search [Sequential Search]
2. Binary Search [Logarithmic Search]

Linear Search is applied on the unsorted or unordered list when there are fewer elements(small arrays) in a list. Binary search can only be used for sorted arrays(lists), but it's fast as compared to linear search. Use of binary search on an array which is not sorted, then must sort it using some sorting technique.

1) Linear Search

Linear search is a very simple search algorithm. In this type of search, a sequential search is made over all items one by one. Every item is checked and if a match is found then that particular item is returned, otherwise the search continues till the end of the data collection.

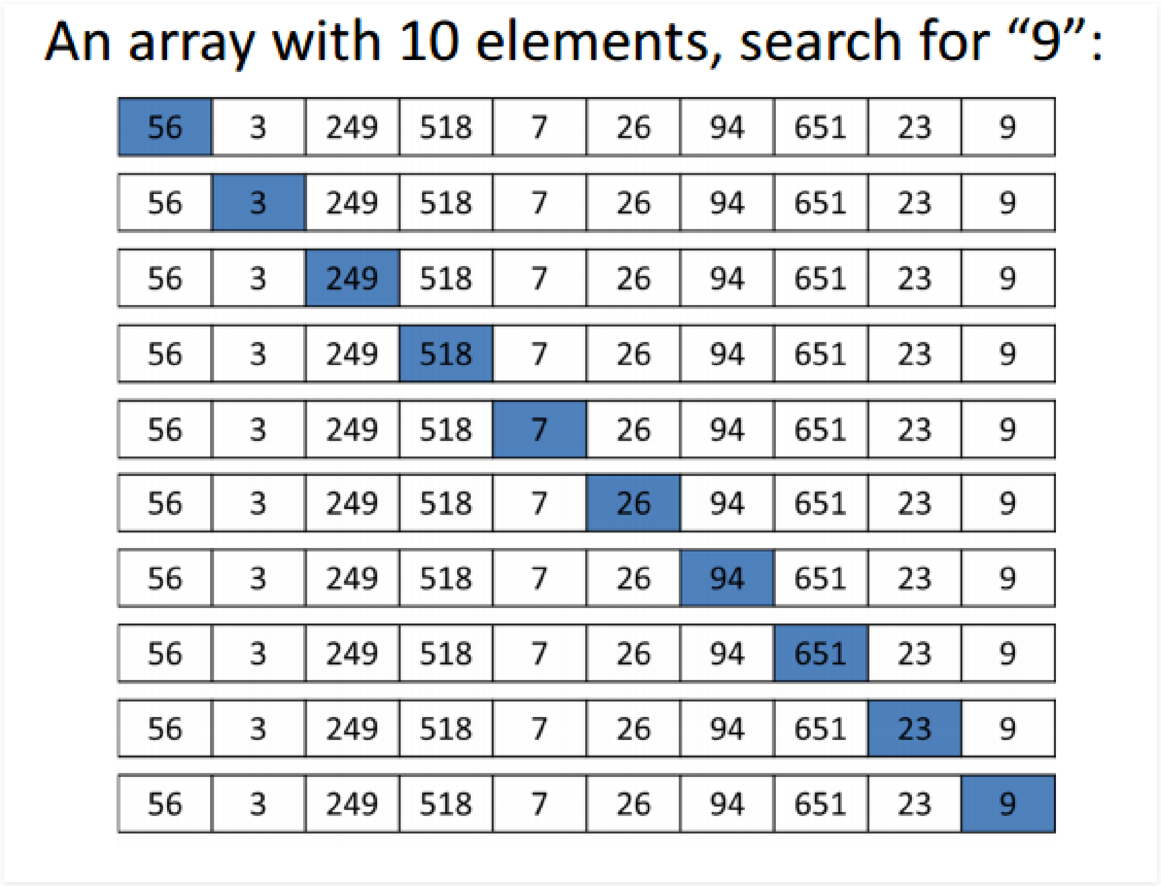
Example: Find a favorite CD in row of Bunch of CD’s

Performance:

1) Suppose that the first element in the array list contains the variable key, then we have performed **one comparison** to find the key.
2) Suppose that the second element in the array list contains the variable key, then we have performed **two comparisons** to find the key.
3) Carry on the same analysis till the key is contained in the last element of the array list. In this case, we have performed N comparisons (where N is the size of the array list) to find the key.
4) Finally if the key is NOT in the array list, then we would have performed N comparisons and the key is NOT found and we would return-1.

Algorithm:

- Start from the leftmost element of arr[]/list and one by one compare x(key) with each element of arr[]/list
- If x(key) matches with an element, return the index.
- If x doesn’t match with any of elements, return -1



56	3	249	518	7	26	94	651	23	9
56	**3**	249	518	7	26	94	651	23	9
56	3	**249**	518	7	26	94	651	23	9
56	3	249	**518**	7	26	94	651	23	9
56	3	249	518	**7**	26	94	651	23	9
56	3	249	518	7	**26**	94	651	23	9
56	3	249	518	7	26	**94**	651	23	9
56	3	249	518	7	26	94	**651**	23	9
56	3	249	518	7	26	94	651	**23**	9
56	3	249	518	7	26	94	651	23	**9**

Program:

```
def search(arr, x):
    for i in range(len(arr)):

        if arr[i] == x:
            return i

    return -1
n=input("Enter list size: ")
arr=[]
for i in range(0,n):
    print "Enter {} Element".format(i+1)
    item=input()
    arr.append(item)
key=input("Enter search element: ")
result=search(arr,key)
if result==-1:
    print "Element not found in the list"
else:
    print "Element found at index: ", result
```

Efficiency of the Linear Search:

It is easy to understand. Easy to implement. Does not require the array to be in sorted order.

The disadvantage is its inefficiency

If there are 20,000 items in the array and what you are looking for is in the 19,999thelement, you need to search through the entire list.

Complexity:

There exist 3 time complexities, namely worst, best & average cases. If there are N elements in the list, then it is obvious that in the worst case i.e., when there is no target element in the list, N comparisons are required. Hence W C TC =O(N).

The best case, in which the 1stcomparison returns a match, it requires a single comparison.
Hence B C TC =O(1)

The average time depends on the prob. that the key will be found in the list. Thus the avg. case roughly requires N/2 comparision to search the element(i.e., depends on N)

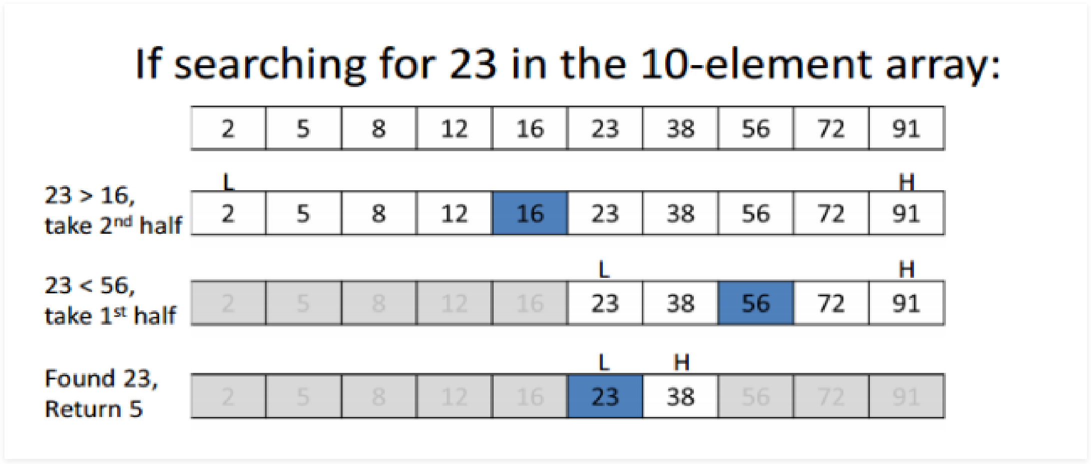
Hence A C TC=O(N)

<u>2) Binary Search</u>

Binary Search is applied on the sorted array or list. In binary search, we first compare the value with the elements in the middle position of the array. If the value is matched, then we return the value. If the value is less than the middle element, then it must lie in the lower half of the array and if it's greater than the element then it must lie in the upper half of the array. We repeat this procedure on the lower (or upper) half of the array. Binary Search is useful when there are large numbers of elements in an array.

Ex: This concept is generally used by Electricians to locate a fused bulb in a serial set & in searching a dictionary, phonebook.

Example:



Algorithm:

1. Compare x(key) with the middle element.
2. If x matches with middle element, we return the mid index.
3. Else If x(key) is greater than the mid element, then x can only lie in right half subarray after the mid element. So we repeated for right half.
4. Else (x is smaller) repeated for the left half.

Program

```
def binarySearch(arr, key):
    minimum=0
    maximum=len(arr)-1
    while minimum <= maximum:

        mid = (minimum + maximum)/2;

        # Check if x is present at mid
        if arr[mid] == key:
            return mid

        # If x is greater, ignore left half
        elif arr[mid] < key:
            minimum = mid + 1

        # If x is smaller, ignore right half
        else:
            maximum = mid - 1

    # If we reach here, then the element
    # was not present
    return -1

n=input("Enter list size: ")
arr=[]
for i in range(0,n):
    print "Enter {} Element".format(i+1)
    item=input()
    arr.append(item)
key=input("Enter search element: ")

result = binarySearch(sorted(arr), key)

if result != -1:
    print "Element is present at index %d" % result
else:
    print "Element is not present in array"
```

Performance & Time complexity:

Best Case performance –The middle element is equal to the “input key” O(1).

Worst Case performance -The “input key” is not present in the list O(logn).

Average Case performance –The “input key” is present, but it’s not the middle element O(logn).

It is faster than Linear Search algorithm, and its performance increases in comparison to LS

Sorting

Sorting is a process that organizes a collection of data into either ascending or descending order. They are many sorting algorithms such as

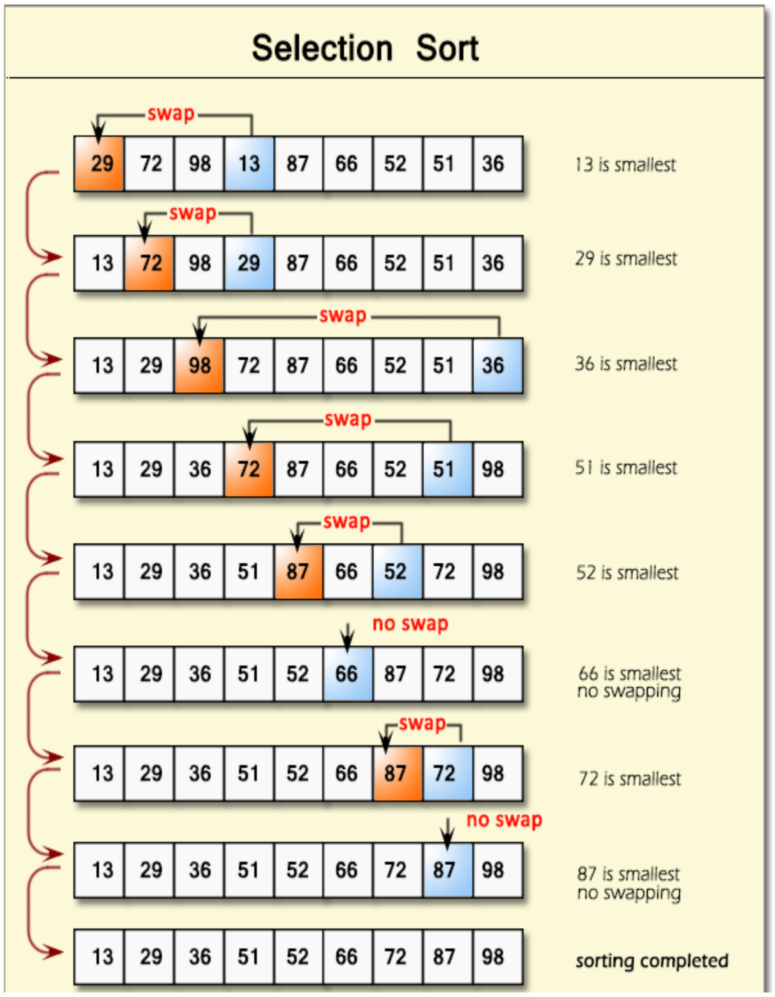
1) Selection Sort
2) Bubble Sort
3) Insertion Sort

Selection Sort:

Selection Sort algorithm is used to arrange a list of elements in a particular order. In selection sort, the first element in the list is selected and it is compared repeatedly with all the remaining elements in the list. If any element is smaller than the selected element (for Ascending order), then both are swapped so that first position is filled with the smallest element in the sorted order. Next, we select the element at a second position in the list and it is compared with all the remaining elements in the list. If any element is smaller than the selected element, then both are swapped. This procedure is repeated until the entire list is sorted.

Algorithm:

Step 1: Select the first element of the list (i.e., Element at first position in the list).
Step 2: Compare the selected element with all the other elements in the list.
Step 3: In every comparision, if any element is found smaller than the selected element then both are swapped.
Step 4: Repeat the same procedure with element in the next position in the list till the entire list is sorted.



Program:

```
def selection_sort(my_list):
    # Function Definition
    for i in range(len(my_list)):
        minimum = i
        for j in range(i+1, len(my_list)):
            if my_list[j] < my_list[minimum]:
                minimum = j
        # Swapping the values
        l[i],l[minimum]=l[minimum],l[i]
    return my_list
size = int( input("Enter a size of the list: ") )
l=[]
for i in range(size):
    l.append(int(input("Enter Element: ")))
print ("Entered elements into list: ",(l) )
print ("After doing SELECTION SORT: ",selection_sort(l))
```

Time Complexities:

Worst Case Complexity: $O(n^2)$

Best Case Complexity: $O(n^2)$

Average Case Complexity: $O(n^2)$

Advantages: The selection sort is easy to understand, easy to write & this makes it easy to implement the algorithm.

Disadvantage:

This algorithm is not suitable for large amount of data.

Bubble Sort

Bubble sort is a simple sorting algorithm. In Bubble sort algorithm, compares the adjacent elements and swaps their positions if they are in wrong order.

Algorithm:

Step1: Starting from the first index, compare the first and the second elements. If the first element is greater than the second element, swap them.

Step2: Now, compare the second and the third elements. Swap them if they are not in order.

Step3: Repeat the same procedure until reach the last element in the list.

Step4: In every comparison, first largest element will placed to last index(n-1th index), second largest element placed at n-2th index and so on.

Step5: Finally list will be sorted.

Time Complexities:

Worst Case Complexity: $O(n)$

Best Case Complexity: $O(n^2)$

Average Case Complexity: $O(n^2)$

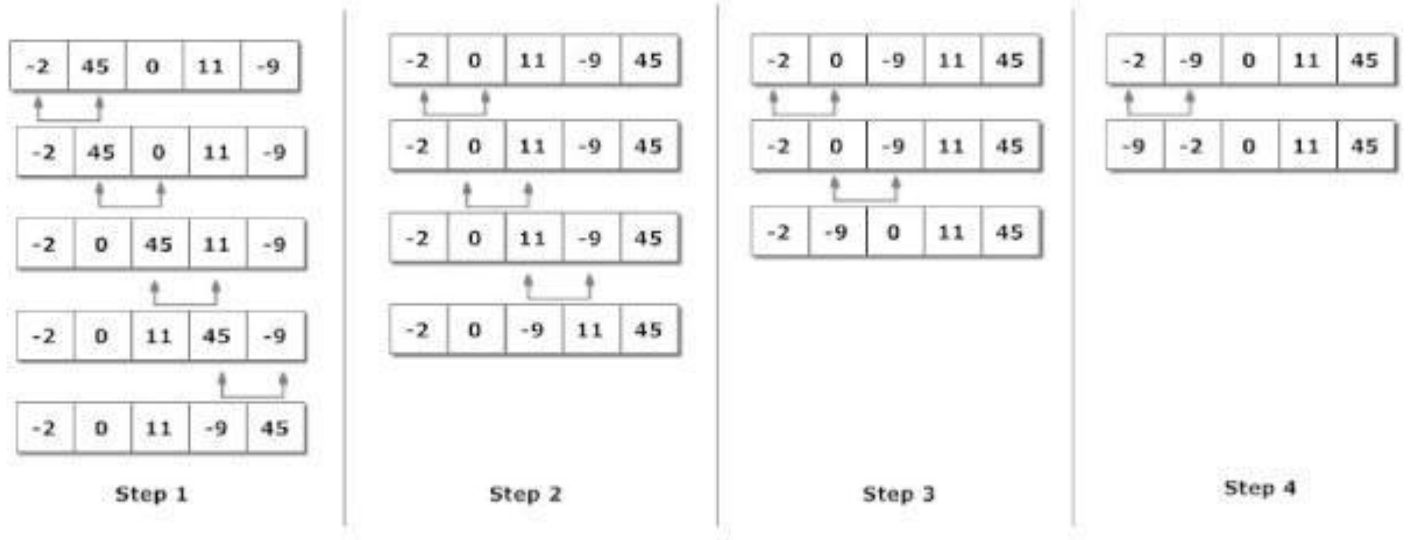


Figure: Working of Bubble sort algorithm

Program:

```
def bubble_sort(l):
    """ Implementation of bubble sort """
    for i in range(len(l)):
        for j in range(len(l)-1-i):
            if l[j] > l[j+1]:
                l[j], l[j+1] = l[j+1], l[j]
                # Swap! return items
    return (l)
size = int(input("Enter a size of the array: "))
l=[]
for i in range(0,size):
    l.append(int(input("Enter a element: ")))
print ("Entered elements into list: ",(l))
print ("After doing BUBBLE SORT: ",bubble_sort(l))
```

Insertion Sort:

Insertion sort is a simple sorting algorithm that works the way we sort playing cards in our hands. In insertion sort algorithm, every iteration moves an element from unsorted portion to sorted portion until all the elements are sorted in the list.

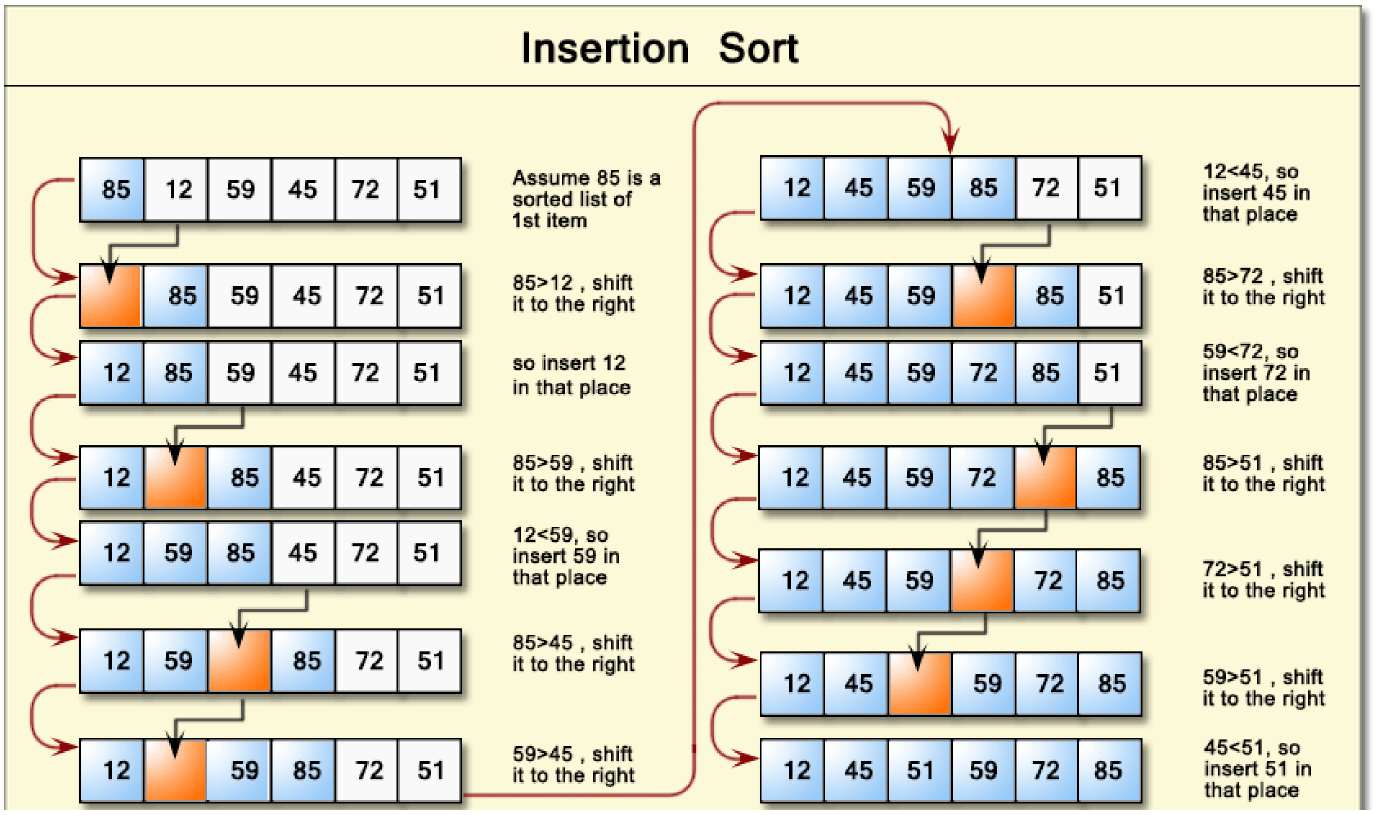
Algorithm:

Step 1: Assume that first element in the list is in sorted portion and all the remaining elements are in unsorted portion.

Step 2: Take first element from the unsorted portion and insert that element into the sorted portion in the order specified.

Step 3: Repeat the above process until all the elements from the unsorted portion are moved into the sorted portion.

Step 4: Finally list will be sorted.


Insertion Sort
Assume 85 is a sorted list of 1st item
85>12 , shift it to the right
so insert 12 in that place
85>59 , shift it to the right
12<59, so insert 59 in that place
85>45 , shift it to the right
59>45 , shift it to the right
12<45, so insert 45 in that place
85>72 , shift it to the right
59<72, so insert 72 in that place
85>51 , shift it to the right
72>51 , shift it to the right
59>51 , shift it to the right
45<51, so insert 51 in that place

Insertion Sort Program

```
# Source code of Insertion Sort
def insertionSort(arr):
    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
            arr[j+1] = arr[j]
            j -= 1
        arr[j+1] = key
    return arr
size = int(input("Enter a size of the array: "))
arr=[]
for i in range(0,size):
    arr.append(int(input("Enter a element: ")))
print ("Entered elements into list: ",(arr))
print ("Sorted array is:",insertionSort(arr))
```

Time complexity:

Best-case:O(n)

Worst-case:O(n^2)

Average-case:O(n^2)

So, Insertion Sort is O(n^2)