Unit 5 String Processing and File Handling

5.1 Introduction to String

- ✓ A string is a sequence of characters.
- ✓ Strings can be created by enclosing characters inside a single quote or double-quotes.
- ✓ A string that contains no characters are called empty string.

Example:

s='Hello' print(s) s= "Hello" print(s) s = "'Hello'" print(s) s = """Hello, welcome to the world of Python""" print(s)	Output: Hello Hello Hello Hello, welcome to the world of Python

String Operation

Create list of strings

- ✓ To create a list of strings, first use square brackets [ ] to create a list.
- ✓ Then place the list items inside the brackets separated by commas.
- ✓ strings must be surrounded by quotes.
- ✓ use = to store the list in a variable.

Ex:

```
Name = ["Ami", "Neha", "Dharmesh"]
```

Ex:

```
Name = [
   "Ami ",
   " Neha ",
   " Dharmesh "
   ]
```

Print list of strings

- ✓ To print a whole list of strings in one line, you can simply call the built-in print function,

Name = ["Ami", "Neha", "Dharmesh"] print(Name)	Output ['Ami', 'Neha', 'Dharmesh']

Add strings to list

- ✓ When you already have a list of strings and you want to add another string to it, you can use the append method:

Name = ["Ami", "Neha", "Dharmesh"]

```
Name.append("Aesha")
print(Name)
```

Concatenate lists of strings

✓ You can use the + operator to concatenate two lists of strings. For example:

Name1 = ["Ami", "Neha"] Name2 = ["Aesha", "Monika"] Name= Name1 + Name2 print(Name)	Output: ['Ami', 'Neha', 'Aesha', Monika']

Sort list of strings

✓ To sort a list of strings, you can use the sort method:

colors = ["red", "white","blue"] colors.sort() print(colors)	Output: ['blue', 'red', 'white']

✓ You can also use the sorted() built-in function:

colors = ["red", "white","blue"] colors=sorted(colors) print(colors)	Output: ['blue', 'red', 'white']

Join list of strings

✓ To join a list of strings by another string, you need to call the join method on the string, giving your list as an argument. For example, if we have this list:

colors = ["red", "white","blue"] print("".join(colors)) print(" ".join(colors)) print("|".join(colors))	Output: redwhiteblue red white blue red|white|blue

5.2 Access String elements using index operator

✓ You can use a for loop,range in python and slicing operator to traverse a character in a string.

Using for loop to traverse a string

str="python" for s in str: print(s)	Output: p y t h o

	n

Using range to traverse a string:

str="python" for s in range(len(str)): print(str[s])	Output: p y t h o n

Using slicing operator to traverse a string partially

- ✓ A segment of a string is called a slice.
- ✓ Selecting a slice is similar to selecting a character:
- ✓ Subsets of strings can be taken using the slice operator **([ ] and [:])** with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.
- ✓ Slice out substrings, sub lists, sub Tuples using index.

Syntax:[Start: stop: steps]

- ✓ Slicing will start from index and will go up to **stop** in **step** of steps.
- ✓ Default value of start is 0,
- ✓ Stop is last index of list
- ✓ And for step default is 1

str = 'Hello World!' print (str) print(str[0]) print (str[2:4]) print (str[2:]) print (str * 2)	Output Hello World! H ll llo World! Hello World!Hello World!
str="Scripting Language python" for s in str[0:6:1]: print(s)	Output S c r i p t
str="Scripting Language python" for s in str[::3]: print(s)	Output Sii na tn
str="python" for s in str[::-1]: print(s)	Output nohtyp
str="Python" print(str[::-1])	Output nohtyp

Using indexing ,print backward string

str="Python"
l=len(str)-1
while l>=0:
  print(str[l])
  l=l-1
	Output
	n
	o
	h
	t
	y
	p

5.3 String functions

- **Basic functions: len, max, min**

len( ): It returns number of characters in the string

syntax: variable name=len(string)

Example:

```
s=”python”
l=len(s)
print(l)
```

Output:6

min( ): It finds the smallest element in the sequence.

For one string, the character that has the smallest code is selected in the string.

For multiple string, a string is selected which,in lexicographic order (dictionary order) ,follows the first alphabetically.

Syntax:

ans=min(s)

Or

ans=min(s1,s2,…sn)

s="python" print(min(s)) s='abc+-0' print(min(s)) s='012345' print(min(s)) s='aAbB' print(min(s))	Output: h + 0 A
s=("def","abc","xyz") print(min(s)) s=('+','-','*','/') print(min(s))	Output: abc *

max( ): It finds the largest element in the sequence.

For one string, the character that has the largest code is selected in the string.

For multiple string, a string is selected which, in lexicographic order, follows the last alphabetically.

Syntax:

```
ans=max(s)
        Or
ans=max(s1,s2,…sn)
```

s="python" print(max(s)) s='abc+-0' print(max(s)) s='012345' print(max(s)) s='aAbB' print(max(s))	Output: y c 5 b
s=("def","abc","xyz") print(max(s)) s=('+','-','*','/') print(max(s))	Output: xyz /

● **Testing functions: isalnum, isalpha, isdigit, isidentifier, islower, isupper, and isspace**

isalnum(): returns True if all the characters in the string is alphanumeric (a string which contains either number or alphabets or both). Otherwise return False

s=" " print(s.isalnum()) s="abc" print(s.isalnum()) s='123' print(s.isalnum()) s='abc123' print(s.isalnum()) s='+-' print(s.isalnum()) s='a b' print(s.isalnum())	Output: False True True True False False

isalpha: returns True if all the characters in the string are alphabets. Otherwise return False

s=" " print(s.isalpha()) s="abc" print(s.isalpha()) s='123' print(s.isalpha()) s='abc123' print(s.isalpha()) s='+-' print(s.isalpha())	Output: False True False False False

Isdigit: returns True if all the characters in the string are digits. Otherwise return False.

print("abc123".isdigit()) print("123".isdigit()) print("101 ".isdigit()) print("101.129".isdigit())	Output: False True False False

Isidentifier: returns True if string is an identifier or a keyword that is valid according to the python syntax Otherwise return False.

print("hello".isidentifier()) print("a1".isidentifier()) print("1a".isidentifier()) print("for".isidentifier())	Output: True True False True

islower: returns True if all the characters in the string are in lowercase. Otherwise return False.

s = 'Python' print(s.islower()) print("abc".islower())	Output: False True

isupper: returns True if all the characters in the string are in uppercase. Otherwise return False.

s = 'PYthon' print(s.isupper()) print("ABC".isupper())	Output: False True

isspace : returns True if all the characters in the string are whitespace characters. Otherwise return False.

print("\n\t".isspace()) print(" \n\t".isspace()) print("@ \n\t".isspace()) print("123".isspace())	Output: True True False False

- **Searching functions: endswith, startswith, find, rfind, count**

endswith: Syntax: endswith(suffix, start, end)

✓ Returns True when a string ends with the characters specified by suffix.

✓ You can limit the check by specifying a starting index using start or an ending index using end.

✓ If start and end indexes are not provided then, by default it takes 0 and length-1 as starting and ending indexes

s="abcdef" print(s.endswith('ef')) print("abcdef".endswith('ef',2,6)) print("abcdef".endswith('ef',2,4)) s1=('ab','cd','ef') s2="abcd 123 ef" print(s2.endswith(s1,0,len(s2)))	Output: True True False True

startswith: Syntax:startswith(prefix, start, end):

- ✓ Returns True when a string begins with the characters specified by prefix.
- ✓ You can limit the check by specifying a starting index using start or an ending index using end.
- ✓ If start and end indexes are not provided then, by default it takes 0 and length-1 as starting and ending indexes.

s="Hello" print(s.startswith('He')) print(s.startswith('lo')) print(s.startswith('He',0)) print(s.startswith('He',0,len(s))) print(s.startswith('He',2)) print("12345".startswith('123',0,4)) print("12345".startswith('123',2,4))	Output: True False True True False True False

Find: Syntax: find(str, start, end):

- ✓ Check whether str occurs in a string and outputs the index of the location.
- ✓ You can limit the search by specifying starting index using start or a ending index using end.
- ✓ If start and end indexes are not provided then, by default it takes 0 and length-1 as starting and ending indexes .
- ✓ If search not found then it returns -1.

s = 'Hello World' ans = s.find('Hello') print("Substring found at index:", ans) print(s.find('Wo')) print(s.find('l', 2)) print(s.find('a', 2)) print(s.find('o', 2, 10))	Output: Substring found at index: 0 6 2 -1 4

rfind: Syntax: rfind(str, start, end):

- ✓ Provides the same functionality as find(), but searches backward from the end of the string instead of the starting.
- ✓ You can limit the search by specifying starting index using start or a ending index using end.
- ✓ If start and end indexes are not provided then, by default it takes 0 and length-1 as starting and ending indexes .
- ✓ If search not found then it returns -1.

s = 'Hello World' ans = s.rfind('Hello') print("Substring found at index:", ans) print(s.rfind('Wo')) print(s.rfind('l', 2)) print(s.rfind('a', 2)) print(s.rfind('o', 2, 10))	Output: Substring found at index: 0 6 9 -1 7

count: Syntax: count(str, start, end):

✓ Counts how many times str occurs in a string.
✓ You can limit the search by specifying a starting index using start or an ending index using end.
✓ If start and end indexes are not provided then, by default it takes 0 and length-1 as starting and ending indexes.

s= "Hello Students Hello" print(s.count("Hello", 0, 5)) print(s.count("Hello", 0, 20)) print(s.count('l',0,20)) print(s.count('c',0,20))	Output: 1 2 4 0

● **Manipulation functions:**

capitalize( ): It returns a copy of the original string and converts the first character of the string to a capital **(uppercase)** letter, while making all other characters in the string **lowercase** letters.

Syntax: string_name.capitalize()

s= "hello world" print(s.capitalize()) print("heLLo".capitalize()) print('123'.capitalize())	Output: Hello world Hello 123

lower( ): This function converts all uppercase characters in a string into lowercase characters and returns it.

Syntax**:** string.lower()

s= "AMI Patel" print("Original string:",s) print("Converted string:"+s.lower())	Output: Original string: AMI Patel Converted string:ami patel

upper ( ): This function converts all lowercase characters in a string into uppercase characters and returns it.

Syntax**:** string.upper()

s= "AMI Patel" print("Original string:",s) print("Converted string:"+s.upper())	Output: Original string: AMI Patel Converted string:AMI PATEL

title ( ): It converts the first character in each word to uppercase and the remaining characters to lowercase in the string and returns a new string.

Syntax**:** string.title()

s= "hELLo good morning" print("Original string:",s) print("Converted string:"+s.title())	Output: Original string: hELLo good morning Converted string:Hello Good Morning

swapcase: The string swapcase() method converts all uppercase characters to lowercase and lowercase characters to uppercase of the given string, and returns it.

Syntax**:** string.swapcase()

s= "hELLo GOOD morning" print("Original string:",s) print("Converted string:"+s.swapcase())	Output: Original string: hELLo GOOD morning Converted string: HellO good MORNING

replace: It returns a copy of the string where all occurrences of a substring are replaced with another substring.

Syntax: string.replace(old_string, new_string, count)

s="Hello World" s1=s.replace("l", "L") s2=s.replace("o","O",1) print(s) print(s1) print(s2)	Output: Hello World HeLLo WorLd HellO World

lstrip: It returns a copy of the string with leading characters removed (based on the string argument passed). If no argument is passed, it removes leading spaces.
Syntax: string.lstrip([characters])

Here,characters [optional]: A set of characters to remove as leading characters. By default, it uses space as the character to remove.

s=" Hello" print(s.lstrip()) s="++..--$Hello" print(s.lstrip("+.-"))	Output: Hello $Hello

rstrip: It returns a copy of the string with trailing characters removed (based on the string argument passed). If no argument is passed, it removes trailing spaces.

Syntax**:** string.rstrip([characters])
Parameters**:** characters (optional) – a string specifying the set of characters to be removed.

s="Hello " print(s.rstrip()) s="Hello+..+&.." print(s.rstrip('.+&'))	Output: Hello Hello

strip: It is an inbuilt function in the Python programming language that returns a copy of the string with both leading and trailing characters removed.
Syntax: string.strip([characters])

s=" Hello World..." print(s) print(s.strip()) print(s.strip('.'))	Output: Hello World... Hello World... Hello World

casefold: Python String casefold() method is used to convert string to lower case. It is similar to lower() string method, but case removes all the case distinctions present in a string.

- ✓ For example, the German letter ß is already lowercase so, the lower() method doesn't make the conversion.
- ✓ But the casefold() method will convert ß to its equivalent character ss.

Syntax: string.casefold()

s="HELLO" print(s.casefold()) print(s.lower())	Output: hello hello

- **Formatting functions:**

format: The format() method formats the specified value(s) and insert them inside the string's placeholder. The placeholder is defined using curly brackets: {}. It is used for handling complex string formatting more efficiently.

Syntax: string.format(value1, value2...)

a=5 b=10 s='{0}+{1}={2}'.format(a,b,a+b) print(s)	Output: 5+10=15
a=5 s='{0}**{1}={2}'.format(a,2,a**2) print(s)	Output: 5**2=25
s="My name is {fname}".format(fname = "abc") print(s)	Output: My name is abc

center: It creates and returns a new string that is padded with the specified character.

Syntax: string.center(length[, fillchar])

length: length of the string after padding with the characters.

fillchar: (optional) characters which need to be padded. If it's not provided, space is taken as the default argument.

s="Hello World" print(s.center(20)) print(s.center(5)) print(s.center(20,"#")) print(s.center(5,"#"))	Output: Hello World Hello World ####Hello World##### Hello World

ljust: This method left aligns the string according to the width specified and fills the remaining space of the line with blank space if 'fillchr' argument is not passed.

Syntax: ljust(len, fillchr)

len: The width of string to expand it.

fillchr (optional): The character to fill in the remaining space.

s = "hello" print(s.ljust(10)) print(s.ljust(10, '-')) print(s.ljust(10, '#'))	Output: hello hello----- hello#####

rjust: This method left aligns the string according to the width specified and fills the remaining space of the line with blank space if 'fillchr' argument is not passed.

Syntax: rjust(len, fillchr)

len: The width of string to expand it.

fillchr (optional): The character to fill in the remaining space.

s = "hello" print(s.rjust(10)) print(s.rjust(10, '-'))	Output: hello -----hello

print(s.rjust(10, '#'))	#####hello

5.4 Introduction to Text files

- ✓ Python provides inbuilt functions for creating, writing, and reading files.
- ✓ There are two types of files that can be handled in python, normal text files and binary files (written in binary language, 0s, and 1s).

- ✓ Text files: In this type of file, Each line of text is terminated with a special character called EOL (End of Line), which is the new line character ('\n') in python by default.

- ✓ Binary files: In this type of file, there is no terminator for a line, and the data is stored after converting it into machine-understandable binary language.

- ✓ **Difference between Text files and Binary files:**

Text files	Binary files
Data stores in ASCII or Unicode .	Contains raw data
Content written in text files is human readable.	Content written in binary files is not human readable
Slower than binary file.	Faster than text file.
Text files are used to store data more user friendly.	Binary files are used to store data more compactly.

Files

- ✓ Files are named locations on disk to store related information.
- ✓ They are used to permanently store data in a non-volatile memory (e.g. hard disk).
- ✓ When we want to read from or write to a file, we need to open it first.
- ✓ When we are done, it needs to be closed so that the resources that are tied with the file are freed.
- ✓ In Python, a file operation takes place in the following order:
  - Open a file
  - Read or write (perform operation)
  - Close the file

5.5 File Handling functions:

• Basic functions: open, close

open File:

- ✓ Before performing any operation on the file like reading or writing, first, we have to open that file.
- ✓ For this, we should use Python's inbuilt function open()
- ✓ At the time of opening, we have to specify the mode, which represents the purpose of the opening file.

Syntax: file object = open(filename, access mode)
Example: f = open("a.txt", "r")

Where the following mode is supported:

r	open an existing file for a read operation.

w	open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
a	open an existing file for append operation. It will not override existing data.
r+	To read and write data into the file. The previous data in the file will be overridden
w+	To write and read data. It will override existing data.
a+	To append and read data from the file. It will not override existing data.

close File:

- ✓ close() function closes the file and frees the memory space acquired by that file.
- ✓ It is used at the time when the file is no longer needed or if it is to be opened in a different file mode.

Syntax: File object.close()

Example: f = open("a.txt", "r")
f.close()

• Reading file:

- ✓ To read a file in Python, we must open the file in reading (r) mode.
- ✓ There are three ways to read data from a text file.
  read, readline and readlines.

read( ) : Returns the read bytes in form of a string. It reads n bytes, if no n specified, reads the entire file.

Syntax: File object.read([n])

f=open("a.txt","r") print(f.read( )) f.close()	Output: hello everyone Good morning

readline( ) : Reads a line of the file and returns in form of a string. For specified n, reads at most n bytes. It does not reads more than one line.

Syntax: File object.readline([n])

f=open("a.txt","r") print(f.read(5)) f.close()	Output: hello

Readlines( ): Reads all the lines and return them as each line a string element in a list.

Syntax: File_object.readlines()

f=open("a.txt","r") print(f.readlines( )) f.close()	Output: ['hello everyone\n', 'Good morning\n', 'Have a nice day\n']

• writing file: write, append, writelines

- ✓ To write a file in Python, we must open the file in writing (w) mode or append mode(a).

write(): The write() method writes a string to a text file.

Syntax: File_object.write(str1)

Inserts the string str1 in a single line in the text file.

f=open("E:/a.txt","w") f.write("Hello World")	Output: Open a.txt file ….. **Hello World** is display in text

f.close()	file.

append: To append to a text file, you need to open the text file for appending mode.
When the file is opened in append mode, the handle is positioned at the end of the file. The data being written will be inserted at the end, after the existing data.

f=open("E:/a.txt","a") f.write("Good Morning") f.close()	Output: Open a.txt file ….. **Hello World Good Morning** is display in text file.

writelines(): The writelines() method write a list of strings to a file at once.

Syntax: File_object.writelines(L) for L = [str1, str2, str3]

For a list of string elements, each string is inserted in the text file. Used to insert multiple strings at a single time.

f=open("E:/a.txt","w") lines=["Hello\n","good\n","morning"] f.writelines(lines) f.close()	Output: Open a.txt file …..Below content is displayed in text file. Hello good morning