Unit-2

Write notes on String slicing. Illustrate how string slicing is done with suitable example(16)

Sequence of characters enclosed in single, double or triple quotation marks are called strings.

Basics of String:

- Strings are immutable in python. It means it is unchangeable. At the same memoryaddress, the new value cannot be stored.

- Each character has its index or can be accessed using its index.

- String in python has two-way index for each location. (0, 1, 2, ……. In the forwarddirection and -1, -2, -3, in the backward direction.)

Example:

0	1	2	3	4	5	6	7
k	e	n	d	r	i	y	a
-8	-7	-6	-5	-4	-3	-2	-1

- The index of string in forward direction starts from 0 and in backward direction starts from -1.
- The size of string is total number of characters present in the string. (If there are n characters in the string, then last index in forward direction would be n-1 and last index in backward direction would be –n.)

- String are stored each character in contiguous location.
- The character assignment is not supported in string because strings are immutable.Example :

  ```
  str = “kendriya”
  str[2] = ‘y’      # it is invalid. Individual letter assignment not allowed in python
  ```

6.2 Traversing a String:

Access the elements of string, one character at a time.str =

```
“kendriya”

for ch in str :

        print(ch, end= ‘  ‘)
```

Output:

kendriya

6.3 String Operators:

a. Basic Operators (+, *)

b. Membership Operators (in, not in)

c. Comparison Operators (==, !=, <, <=, >, >=)

a. **Basic Operators**: There are two basic operators of strings:

i. String concatenation Operator (+)

ii. String repetition Operator (*)

i. **String concatenation Operator**: The + operator creates a new string by joining the twooperand strings.

Example:

```
>>>”Hello”+”Python”
‘HelloPython’
>>>’2’+’7’
’27’
>>>”Python”+”3.0”
‘Python3.0’
```

Note: You cannot concate numbers and strings as operands with + operator.Example:

>>>7+’4’ # unsupported operand type(s) for +: 'int' and 'str'It is
invalid and generates an error.

ii. **String repetition Operator:** It is also known as **String replication operator**. It requirestwo types of operands- a string and an integer number.

Example:

```
>>>"you" * 3
'youyouyou'
>>>3*"you"
'youyouyou'
```

Note:You cannot have strings as n=both the operands with * operator.Example:

```
>>>"you" * "you"          # can't multiply sequence by non-int of type 'str'It is
```

invalid and generates an error.

b. Membership Operators:

in – Returns **True** if a character or a substring exists in the given string; otherwise **False**

not in - Returns **True** if a character or a substring does not exist in the given string; otherwise **False**

Example:

```
>>> "ken" in "Kendriya Vidyalaya"False
>>> "Ken" in "Kendriya Vidyalaya"True
>>>"ya V" in "Kendriya Vidyalaya"True
>>>"8765" not in "9876543"
False
```

c. Comparison Operators: These operators compare two strings character by characteraccording to their ASCII value.

Characters	ASCII (Ordinal) Value
‘0’ to ‘9’	48 to 57
‘A’ to ‘Z’	65 to 90
‘a’ to ‘z’	97 to 122

Example:

```
>>> 'abc'>'abcD'
False
>>> 'ABC'<'abc'
True
>>> 'abcd'>'aBcD'
True
>>> 'aBcD'<='abCd'
True
```

Slice operator with Strings:

The slice operator slices a string using a range of indices.

Syntax:

string-name[start**:**end]

where **start** and **end** are integer indices. It returns a string from the index **start** to **end-1**.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
d	a	t	a		s	t	r	u	c	t	u	r	e
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example:

```
>>> str="data structure"

>>> str[0:14]

'data structure'

>>> str[0:6]

'data s'

>>> str[2:7]

'ta st'

>>> str[-13:-6]

'ata str'

>>> str[-5:-11]

' '            #returns empty string

>>> str[:14]          # Missing index before colon is considered as 0.'data

structure'

>>> str[0:]              # Missing index after colon is considered as 14. (length of string)'data

structure'

>>> str[7:]

'ructure'

>>> str[4:]+str[:4]'

structuredata'

>>> str[:4]+str[4:]         #for any index str[:n]+str[n:] returns original string'data

structure'

>>> str[8:]+str[:8]

'ucturedata str'

>>> str[8:], str[:8]

('ucture', 'data str')
```

Slice operator with step index:

Slice operator with strings may have third index. Which is known as step. It is optional.

Syntax:

string-name[start:end:step]

Example:

```
>>> str="data structure"

>>> str[2:9:2]

't tu'

>>> str[-11:-3:3]

'atc'

>>> str[: : -1]            #

reverses a string'erutcurts

atad'
```

Interesting Fact: Index out of bounds causes error with strings but slicing a string outside the index does not cause an error.

Example:

```
>>>str[14]

IndexError: string index out of range

>>> str[14:20]        # both indices are outside

the bounds'            '# returns empty string

>>> str[10:16]

'ture'
```

Reason: When you use an index, you are accessing a particular character of a string, thus the index must be valid and out of bounds index causes an error as there is no character to returnfrom the given index.

But slicing always returns a substring or empty string, which is valid sequence.

Built-in functions of string:

S. No.	Function	Description	Example
1	len( )	Returns the length of a string	>>>print(len(str))

			14
2	capitalize( )	Returns a string with its **first character** capitalized.	>>> str.capitalize() 'Data structure'

3	find(sub,start,end)	Returns the **lowest index** in the string where thesubstring **sub** is found within the slice range. Returns -1 if **sub** is not found.	>>> str.find("ruct",5,13)7
			>>> str.find("ruct",8,13)
			-1

4	isalnum( )	Returns **True** if the characters in the string are alphabets or numbers. **False** otherwise	>>>s1.isalnum( )True >>>s2.isalnum( )True >>>s3.isalnum( )True >>>s4.isalnum( )False >>>s5.isalnum( )False
5	isalpha( )	Returns **True** if all characters in the string are alphabetic. **False** otherwise.	>>>s1.isalpha( )False >>>s2.isalpha( )True >>>s3.isalpha( )False >>>s4.isalpha( )False >>>s5.isalpha( )False
6	isdigit( )	Returns **True** if all the characters in the string aredigits. **False** otherwise.	>>>s1.isdigit( )False >>>s2.isdigit( )False >>>s3.isdigit( )True >>>s4.isdigit( )False >>>s5.isdigit( )False

7	islower( )	Returns **True** if all the characters in the string arelowercase. **False** otherwise.	>>> s1.islower()
			True
			>>> s2.islower()
			True
			>>> s3.islower()
			False

			>>> s4.islower() False >>> s5.islower() True
8	isupper( )	Returns **True** if all the characters in the string areuppercase. **False** otherwise.	>>> s1.isupper() False >>> s2.isupper() False >>> s3.isupper() False >>> s4.isupper() False >>> s5.isupper() False
9	isspace( )	Returns **True** if there are only whitespacecharacters in the string. **False** otherwise.	>>> " ".isspace() True >>> "".isspace() False
10	lower( )	Converts a string in lowercase characters.	>>> "HeLlo".lower() 'hello'
11	upper( )	Converts a string in uppercase characters.	>>> "hello".upper() 'HELLO'
12	lstrip( )	Returns a string after removing the leadingcharacters. (Left side). if used without any argument, it removes theleading whitespaces.	>>> str="data structure" >>> str.lstrip('dat')' structure' >>> str.lstrip('data')' structure' >>> str.lstrip('at') 'data structure'

			>>> str.lstrip('adt')' structure' >>> str.lstrip('tad')' structure'
13	rstrip( )	Returns a string after removing the trailingcharacters. (Right side). if used without any argument, it removes thetrailing whitespaces.	>>> str.rstrip('eur') 'data struct' >>> str.rstrip('rut') 'data structure' >>> str.rstrip('tucers') 'data '

14	split( )	breaks a string into words and creates a list out of it	>>> str="Data Structure"
			>>> str.split( )
			['Data',
			'Structure']

Explain in detail about Python Files, its types, functions and operations that can be performed on files with examples. (16)

File:

A file is some information or data which stays in the computer storage devices. You already know about different kinds of file , like your music files, video files, text files. Python gives you easy ways to manipulate these files.

Types of File

- **Text File**: Text file usually we use to store character data. For example, test.txt
- **Binary File**: The binary files are used to store binary data such as images, video files, audio files, etc. These files are readable only by the computer
- **CSV File**: Comma Separated value file. E.g temp.csv
- **TSV File**- Tab Separated value file. E.g temp1.tsv

Data File handling takes place in the following order.

1- Opening a file.
2- Performing operations (read, write) or processing data.
3- Closing the file.

Read File

To read or write a file, we need to open that file. For this purpose, Python provides a built-in function `open()`.

Pass file path and access mode to the `open(file_path, access_mode)` function. It returns the file object. This object is used to read or write the file according to the access mode.

Access mode represents the purpose of opening the file. For example, R is for reading and W is for writing.

we will use the test.txt file for manipulating all file operations. Create a text.txt on your machine and write the below content in it to get started with file handling.

```
Welcome to PYnative.com

This is a sample.txt

Line 3

Line 4

Line 5
```

Example:

```python
# Opening the file with absolute path
fp = open(r'E:\demos\files\sample.txt', 'r')
# read file
print(fp.read())
# Closing the file after reading
fp.close()

# path if you using MacOs
# fp = open(r"/Users/myfiles/sample.txt", "r")
```

Output:

```
Welcome to PYnative.com
```

```
This is a sample.txt

Line 3

Line 4

Line 5
```

File Access Modes

The following table shows different access modes we can use while opening a file in Python.

Mode	Description
r	It opens an existing file to read-only mode. The file pointer exists at the beginning.
rb	It opens the file to read-only in binary format. The file pointer exists at the beginning.
r+	It opens the file to read and write both. The file pointer exists at the beginning.
rb+	It opens the file to read and write both in binary format. The file pointer exists at the beginning of the file.
w	It opens the file to write only. It overwrites the file if previously exists or creates a new one if no file exists with the same name.
wb	It opens the file to write only in binary format. It overwrites the file if it exists previously or creates a new one if no file exists.
w+	It opens the file to write and read data. It will override existing data.

wb+	It opens the file to write and read both in binary format
a	It opens the file in the append mode. It will not override existing data. It creates a new file if no file exists with the same name.
ab	It opens the file in the append mode in binary format.
a+	It opens a file to append and read both.
ab+	It opens a file to append and read both in binary format.

Writing to a File

To write content into a file, Use the access mode w to open a file in a write mode.

Note:

- If a file already exists, it truncates the existing content and places the filehandle at the beginning of the file. A new file is created if the mentioned file doesn’t exist.
- If you want to add content at the end of the file, use the access mode a to open a file in append mode

Example

```
text = "This is new content"
# writing new content to the file
fp = open("write_demo.txt", 'w')
fp.write(text)
print('Done Writing')
fp.close()
```

Copy Files

There are several ways to cop files in Python. The shutil.copy() method is used to copy the source file's content to the destination file.

Example

```
import shutil

src_path = r"E:\demos\files\report\profit.txt"
dst_path = r"E:\demos\files\account\profit.txt"
shutil.copy(src_path, dst_path)
print('Copied')
```

Rename Files

In Python, the os module provides the functions for file processing operations such as renaming, deleting the file, etc. The os module enables interaction with the operating system.

The os module provides rename() method to rename the specified file name to the new name. The syntax of rename() method is shown below.

Example

```
import os

# Absolute path of a file
old_name = r"E:\demos\files\reports\details.txt"
new_name = r"E:\demos\files\reports\new_details.txt"

# Renaming the file
os.rename(old_name, new_name)
```

Delete Files

In Python, the os module provides the remove() function to remove or delete file path.

```
import os

# remove file with absolute path
```

```
os.remove(r"E:\demos\files\sales_2.txt")
```

Python File Methods

Python has various method available that we can use with the **file object**. The following table shows file method.

Method	Description
read()	Returns the file content.
readline()	Read single line
readlines()	Read file into a list
truncate(size)	Resizes the file to a specified size.
write()	Writes the specified string to the file.
writelines()	Writes a list of strings to the file.
close()	Closes the opened file.
seek()	Set file pointer position in a file
tell()	Returns the current file location.
fileno()	Returns a number that represents the stream, from the operating system's perspective.
flush()	Flushes the internal buffer.

Let’s see each file methods one by one.

read() Method:-

Syntax:

```
file_object.read(size)
```

- The size represents the **number of bytes to read** from a file. It returns file content in a string object.
- If size is not specified, it **reads all content from a file**
- If the size argument is negative or not specified, read all data until EOF is reached.
- An empty string is returned when EOF is encountered immediately.
- When used in non-blocking mode, less data than requested may be returned, even if no size parameter was given.

Example:

```
# read(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # read 14 bytes
    # fp is file object
    print(fp.read(14))

    # read all remaining content
    print(fp.read())
```

Output:

```
My First Line

My Second Line

My Third Line
```

readline() Method:-

Syntax:

```
file_object.readline(size)
```

- Read one line from a file at a time. It returns the line in a string format.

- If the size is given it reads the number of bytes (including the trailing newline) from a file.
- If the size argument is negative or not specified, it read a single line
- An empty string is returned when EOF is encountered immediately.

Example:

```
# readline(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # read single line
    print(fp.readline())

    # read next line
    print(fp.readline())
```

Output:

```
My First Line

My Second Line
```

readlines() Method:-

Syntax:

```
file_object.readlines(size)
```

- Read all lines from a file and return them in the form of a list object.
- If the sizehint argument is present, instead of reading the entire file, whole lines totaling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read

Example:

```
# read(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # read all lines
    print(fp.readlines())
```

Output:

```
['My First Line\n', 'My Second Line\n', 'My Third Line']
```

readable() Method:-

It checks whether the file stream can be read or not.

Example:

```
# read(size)
with open(r'E:\pynative\files\test.txt', 'r') as fp:
    # check if file object is readable
    print(fp.readable())
# Output True
```

truncate(size) Method:-

Use the truncate() method to make the file empty.

- If the optional size argument is present, the file is truncated to (at most) that size. So, for example, if you specify 10 bytes, truncate() will remove the first ten bytes from a file.
- The size defaults to the current position of a file pointer
- The current file position is not changed. Note that if a specified size exceeds the file's current size, the result is platform-dependent: possibilities include that the file may remain unchanged, increase to the specified size as if zero-filled, or increase to the specified size with undefined new content. Availability: Windows, many Unix variants.

Example:

```
with open(r'E:\pynative\files\test.txt', 'a') as fp:
    fp.truncate(5)
```

write() Method:-

Write a string to the file. If buffering is used, the line may not show up in the file until the flush() or close() method is called.

Example:

```
with open(r'E:\pynative\files\test.txt', 'w') as fp:
    fp.write('My New Line')
```

Read More: Complete guide on write to file in Python

writelines() Method:-

- Write a list of strings to the file. Use to write multiple lines at a time to a file. You can write any iterable object producing strings, typically a list of strings.
- Note: writelines() do not add line separators.

Example:

```
with open(r'E:\pynative\files\test.txt', 'w') as fp:
    data = ['line1\n', 'line2\n', 'line3\n']
    fp.writelines(data)
```

writable() Method:-

It checks whether the file can be written to or not.

Example:

```
# read(size)
with open(r'E:\pynative\files\test.txt', 'w') as fp:
    # check if file object is readable
    print(fp.writeable())
# Output True
```

close() Method:-

Close the opened file. A closed file cannot be read or written anymore. The ValueError will be raised if you try to read or write a closed file.

Example:

```
fp = open(r'E:\pynative\files\test.txt', 'r')
print(fp.read())
# close file
fp.close()
```

Note: It is good practice to open a file using the with statement. It automatically closes the file and ensures that all the resources tied up with the file are released.

seek() and tell() method:-

The seek() function **sets the position of a file pointer** and the tell() function **returns the current position** of a file pointer.

Example:

```
with open(r'E:\pynative\files\test.txt', "r") as fp:
    # Moving the file handle to 6th character
    fp.seek(6)
    # read file
    print(fp.read())

    # get current position of file pointer
    print(fp.tell())
```

Explain the process of Reading and writing CSV/TSV files with Python(16)

Reading and Writing CSV Files in Python

CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.

Reading and Writing CSV Files in Python

It is supported by a wide range of applications. A CSV file stores tabular data in which each data field is separated by a *delimiter*(comma in most cases).

To represent a CSV file, it must be saved with the ***.csv*** file extension.

Reading and Writing CSV Files in Python

Writing to CSV file

csv.writer class is used to insert data to the CSV file. This class returns a writer object which is responsible for converting the user’s data into a delimited string. A CSV file object should be opened with newline=” otherwise, newline characters inside the quoted fields will not be interpreted correctly.

Syntax:

csv.writer(csvfile, dialect='excel', **fmtparams)csv.writer class provides two methods for writing to CSV. They are writerow() and writerows().

writerow(): This method writes a single row at a time. Field row can be written using this method.
Syntax:writerow(fields)

writerows(): This method is used to write multiple rows at a time. This can be used to write rows list.
Syntax:

writerows(rows)

. **Example 1: Write into CSV files with csv.writer()**

```
import csv

with open('innovators.csv', 'w', newline='') as file:

    writer = csv.writer(file)

    writer.writerow(["SN", "Name", "Contribution"])

    writer.writerow([1, "Linus Torvalds", "Linux Kernel"])

    writer.writerow([2, "Tim Berners-Lee", "World Wide Web"])

    writer.writerow([3, "Guido van Rossum", "Python Programming"])
```

Read from CSV File

```
 filename = 'university_records.csv'

infile = open(filename, 'r')

data = infile.read()
```

```
print(data)

infile.close()
```

Example 1: Writing Multiple Rows with writerows()

```
import csv

row_list = [["SN", "Name", "Contribution"],

            [1, "Linus Torvalds", "Linux Kernel"],

            [2, "Tim Berners-Lee", "World Wide Web"],

            [3, "Guido van Rossum", "Python Programming"]]

with open('protagonist.csv', 'w', newline='') as file:

   writer = csv.writer(file)
```

Example 2: Read CSV files with csv.reader() writer.writerows(row_list)

```
import csv

with open('innovators.csv', 'r') as file:

   reader = csv.reader(file)

   for row in reader:

      print(row)
```

Reading and writing CSV/TSV files with Python

- CSV and TSV formats are essentially text files formatted in a specific way: the former one separates data using a comma and the latter uses tab \t characters. Thanks to this, they are really portable and facilitate the ease of sharing data between various platforms.

How to read a TSV file in Python

- A TSV file is a file containing tab separated values, USE csv.reader() TO READ A TSV FILE

Tsvfile.py

```
import csv

with open('output.tsv', 'w') as out_file:
    tsv_writer = csv.writer(out_file, delimiter='\t')

    tsv_writer.writerow(['name', 'field'])

    tsv_writer.writerow(['Dijkstra', 'Computer Science'])

    tsv_writer.writerow(['Shelah', 'Math'])

    tsv_writer.writerow(['Aumann', 'Economic Sciences'])

tsv_file = open("output.tsv")

read_tsv = csv.reader(tsv_file, delimiter="\t")

for row in read_tsv:

  print(row)
```

Python OS module

Renaming the file

The Python **os** module enables interaction with the operating system. The os module provides the functions that are involved in file processing operations like renaming, deleting, etc. It provides us the rename() method to rename the specified file to a new name. The syntax to use the **rename()** method is given below.

Syntax:

1. rename(current-name, new-name)

The first argument is the current file name and the second argument is the modified name. We can change the file name bypassing these two arguments.

Example 1:

```
import os
#rename file2.txt to file3.txt
os.rename("file2.txt","file3.txt")
```

Output:

The above code renamed current **file2.txt** to **file3.txt**

Removing the file

The os module provides the **remove()** method which is used to remove the specified file. The syntax to use the **remove()** method is given below.

```
remove(file-name)
```

Example 1

```
import os;
#deleting the file named file3.txt
os.remove("file3.txt")
```

Creating the new directory

The **mkdir()** method is used to create the directories in the current working directory. The syntax to create the new directory is given below.

Syntax:

```
mkdir(directory name)
```

Example 1

```
import os
#creating a new directory with the name new
os.mkdir("new")
```

The getcwd() method

This method returns the current working directory.

The syntax to use the getcwd() method is given below.

Syntax

```
os.getcwd()
```

Example

```
import os
os.getcwd()
```

Output:

```
'C:\\Users\\DEVANSH SHARMA'
```

Changing the current working directory

The chdir() method is used to change the current working directory to a specified directory.

The syntax to use the chdir() method is given below.

Syntax

```
chdir("new-directory")
```

Example

```
import os
# Changing current directory with the new directiory
os.chdir("C:\\Users\\DEVANSH SHARMA\\Documents")
#It will display the current working directory
os.getcwd()
```

Output:

```
'C:\\Users\\DEVANSH SHARMA\\Documents'
```

Deleting directory

The rmdir() method is used to delete the specified directory.

The syntax to use the rmdir() method is given below.

Syntax

```
os.rmdir(directory name)
```

Example 1

```
import os
#removing the new directory
os.rmdir("directory_name")
```

It will remove the specified directory.

Python - sys Module(8 Mark)

- The sys module provides functions and variables used to manipulate different parts of the Python runtime environment.

1.sys.argv

sys.argv returns a list of command line arguments passed to a Python script. The item at index 0 in this list is always the name of the script. The rest of the arguments are stored at the subsequent indices.

```
test.py
import sys
 print("You entered: ",sys.argv[1], sys.argv[2], sys.argv[3])
```

OUTPUT

```
C:\python36> python test.py Python C# Java
You entered: Python C# Java
```

2.sys.exit

This causes the script to exit back to either the Python console or the command prompt. This is generally used to safely exit from the program in case of generation of an exception.

3.sys.path

This is an environment variable that is a search path for all Python modules.

```
>>>sys.path
```

4.sys.version

This attribute displays a string containing the version number of the current Python interpreter.

```
>>>sys.version
```