NAME : GINI CHACKO

ROLL : 8942

CLASS : SE COMPS B

BATCH : B

TOPIC : PYTHON EXPERIMENT 3

Aim: Write python programs to understand different file handling operations.

1. Write a python program which will count the spaces, tabs, and lines in any given text file. Check if the file exists. If file does not exists then handle that exception.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
from collections import Counter

try:
    with open("drive/MyDrive/Colab Notebooks/content.txt") as f:
        text = f.read()
        count = Counter(text)
except:
    print("File not Found")
else:

    spaces = count[" "]
    tabs = count["\t"]
    lines = count["\n"]

    print("The total number of spaces present in this file is ", spaces)
    print("The total number of tabs present in this file is ", tabs)
    print("The total number of lines present in this file is ", lines)
```

```
The total number of spaces present in this file is  115
The total number of tabs present in this file is  5
The total number of lines present in this file is  6
```

2. The file cities_and_times.txt contains city names and times. Each line contains the name of the city, followed by the name of the day ("Sun") and the time in the form hh:mm. Read in the file and create an alphabetically ordered list of the form [('Amsterdam', 'Sun', (8, 52)),

('Anchorage', 'Sat', (23, 52)), ('Ankara', 'Sun', (10, 52)), ('Athens', 'Sun', (9, 52)), ('Atlanta', 'Sun', (2, 52)), ('Auckland', 'Sun', (20, 52)), ('Barcelona', 'Sun', (8, 52)), ('Beirut', 'Sun', (9, 52)), ... …. ('Toronto', 'Sun', (2, 52)), ('Vancouver', 'Sun', (0, 52)), ('Vienna', 'Sun', (8, 52)), ('Warsaw', 'Sun', (8, 52)), ('Washington DC', 'Sun', (2, 52)), ('Winnipeg', 'Sun', (1, 52)), ('Zurich', 'Sun', (8, 52))] Finally, the list should be dumped for later usage with the pickle module.

```
import pickle

lines = open("drive/MyDrive/Colab Notebooks/cities_and_times.txt").readlines()
lines.sort()

cities = []
for line  in lines:
    *city, day, time = line.split()
    hours, minutes = time.split(":")
    cities.append((" ".join(city), day, (int(hours), int(minutes)) ))

dump = open("cities_and_times.pkl", "bw")
pickle.dump(cities, dump)
dump.close()

load=open("dump","rb")
print(*pickle.load(load) , sep = '\n')
```

```
('Amsterdam', 'Sun', '08:52')
('Anchorage', 'Sat', '23:52')
('Ankara', 'Sun', '10:52')
('Athens', '', 'Sun', '09:52')
('Atlanta', 'Sun', '02:52')
('Auckland', 'Sun', '20:52')
('Barcelona', '', '', 'Sun', '08:52')
('Beirut', '', 'Sun', '09:52')
('Berlin', '', 'Sun', '08:52')
('Boston', '', 'Sun', '02:52')
('Brasilia', 'Sun', '05:52')
('Brussels', '', '', '', 'Sun', '08:52')
('Bucharest', '', '', 'Sun', '09:52')
('Budapest', '', '', '', 'Sun', '08:52')
('Cairo', '', '', 'Sun', '09:52')
('Calgary', 'Sun', '01:52')
('Cape', 'Town', '', '', 'Sun', '09:52')
('Casablanca', '', 'Sun', '07:52')
('Chicago', 'Sun', '01:52')
('Columbus', 'Sun', '02:52')
('Copenhagen', '', 'Sun', '08:52')
('Dallas', '', 'Sun', '01:52')
('Denver', 'Sun', '01:52')
('Detroit', 'Sun', '02:52')
('Dubai', '', '', 'Sun', '11:52')
('Dublin', '', 'Sun', '07:52')
('Edmonton', 'Sun', '01:52')
('Frankfurt', '', '', 'Sun', '08:52')
```

```
        ('Halifax', 'Sun', '03:52')
        ('Helsinki', '', '', '', 'Sun', '09:52')
        ('Houston', 'Sun', '01:52')
        ('Indianapolis', '', '', '', 'Sun', '02:52')
        ('Istanbul', 'Sun', '10:52')
        ('Jerusalem', '', '', 'Sun', '09:52')
        ('Johannesburg', '', '', '', 'Sun', '09:52')
        ('Kathmandu', '', '', 'Sun', '13:37')
        ('Kuwait', 'City', 'Sun', '10:52')
        ('Las', 'Vegas', 'Sun', '00:52')
        ('Lisbon', '', 'Sun', '07:52')
        ('London', '', 'Sun', '07:52')
        ('Los', 'Angeles', 'Sun', '00:52')
        ('Madrid', '', 'Sun', '08:52')
        ('Melbourne', 'Sun', '18:52')
        ('Miami', '', '', 'Sun', '02:52')
        ('Minneapolis', 'Sun', '01:52')
        ('Montreal', '', '', '', 'Sun', '02:52')
        ('Moscow', '', 'Sun', '10:52')
        ('New', 'Orleans', 'Sun', '01:52')
        ('New', 'York', '', '', '', 'Sun', '02:52')
        ('Oslo', '', '', '', 'Sun', '08:52')
        ('Ottawa', '', 'Sun', '02:52')
        ('Paris', '', '', 'Sun', '08:52')
        ('Philadelphia', '', '', '', 'Sun', '02:52')
        ('Phoenix', 'Sun', '00:52')
        ('Prague', '', 'Sun', '08:52')
        ('Reykjavik', '', '', 'Sun', '07:52')
        ('Riyadh', '', 'Sun', '10:52')
        ('Rome', '', '', '', 'Sun', '08:52')
        ('Salt', 'Lake', 'City', '', 'Sun', '01:52')
```

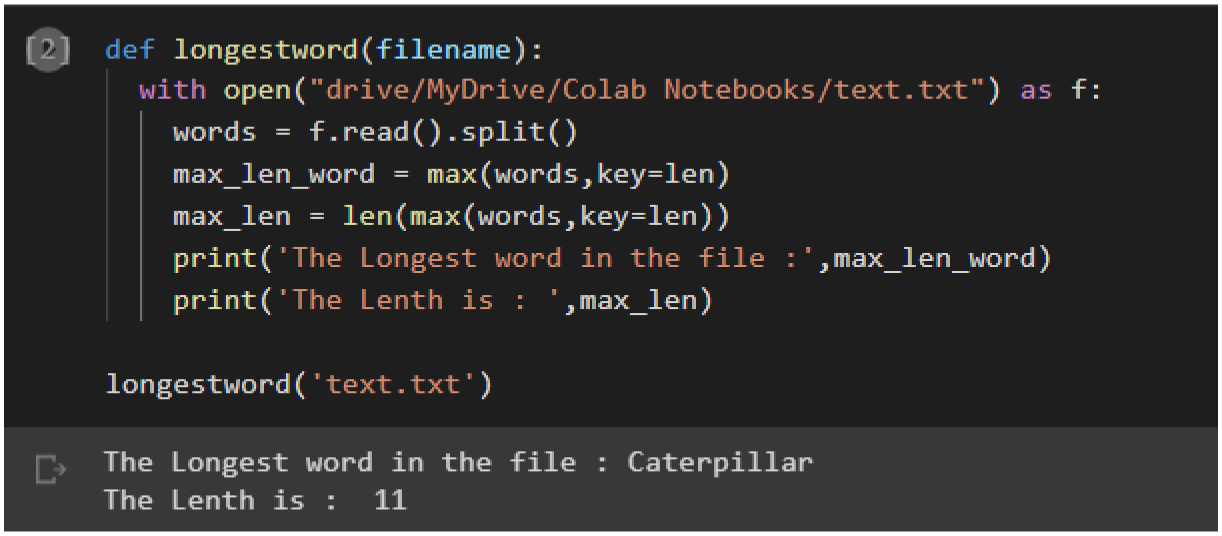
POSTLABS (EXPERIMENT – 3)

1. Write a program to find the longest word in a text file.
Ans :

CODE:

```
def longestword(filename):
    with open(“drive/MyDrive/Colab Notebooks/text.txt”) as f:
            words = f.read().split()
            max_len_word = max(words,key=len)
            max_len = len(max(words,key=len))
            print('maximum lenth word in file :',max_len_word)
            print('lenth is : ',max_len)

longestword(‘text.txt’)
```



```
[2] def longestword(filename):
      with open("drive/MyDrive/Colab Notebooks/text.txt") as f:
        words = f.read().split()
        max_len_word = max(words,key=len)
        max_len = len(max(words,key=len))
        print('The Longest word in the file :',max_len_word)
        print('The Lenth is : ',max_len)

    longestword('text.txt')
```

```
The Longest word in the file : Caterpillar
The Lenth is :  11
```

GINI CHACKO 8942

2. Explain seek() and tell() methods on file in python.

Ans:

a.] seek()

Python file method **seek()** sets the file's current position at the offset. The whence argument is optional and defaults to 0, which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

There is no return value. Note that if the file is opened for appending using either 'a' or 'a+', any seek() operations will be undone at the next write.

If the file is only opened for writing in append mode using 'a', this method is essentially a no-op, but it remains useful for files opened in append mode with reading enabled (mode 'a+').

If the file is opened in text mode using 't', only offsets returned by tell() are legal. Use of other offsets causes undefined behavior.

Syntax :

Following is the syntax for **seek()** method −

fileObject.seek(offset[, whence])

b.] tell()

tell() returns the current position of the file pointer from the beginning of the file. tell() method returns current position of file object. This method takes no parameters and returns an integer value. Initially file pointer points to the beginning of the file(if not opened in append mode).

Syntax :

file_object.tell()

3. **How are renaming and deleting performed on a file? Give the syntax for each.**

Ans:

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then you can call any related functions.

The rename() Method

The rename() method takes two arguments, the current filename and the new filename.

Syntax

```
os.rename(current_file_name, new_file_name)
```

Example

Following is the example to rename an existing file test1.txt −

```
#!/usr/bin/python

import os

# Rename a file from test1.txt to test2.txt

os.rename( "test1.txt", "test2.txt" )
```

The remove() Method

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.

Syntax

```
os.remove(file_name)
```

Example

Following is the example to delete an existing file test2.txt −

```
#!/usr/bin/python
import os
# Delete file test2.txt
os.remove("text2.txt")
```

4. Explain "pickle" in python.

Ans:

Python pickle module is used for serializing and de-serializing a Python object structure. Any object in Python can be pickled so that it can be saved on disk. What pickle does is that it "serializes" the object first before writing it to file. Pickling is a way to convert a python object (list, dict, etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script.

Advantages of using Pickle Module:

1. **Recursive objects (objects containing references to themselves):** Pickle keeps track of the objects it has already serialized, so later references to the same object won't be serialized again.

2. **Object sharing (references to the same object in different places):** This is similar to self- referencing objects; pickle stores the object once, and ensures that all other references point to the master copy. Shared objects remain shared, which can be very important for mutable objects.

3. **User-defined classes and their instances:** Marshal does not support these at all, but pickle can save and restore class instances transparently. The class definition must be importable and live in the same module as when the object was stored.