Experiment No. 5
Exploring Files and directories: Python program to append data to existing file and then display the entire file
Name : Vaidehi D. Gadag
Branch/Div.: Comps-1 (C47)
Date of Performance: 14/02/2024
Date of Submission: 24/02/2024

Experiment No. 5

Title: Exploring Files and directories: Python program to append data to existing file and then display the entire file

Aim: To Exploring Files and directories: Python program to append data to existing file and then display the entire file

Objective: To Exploring Files and directories

Theory:

Directory also sometimes known as a folder are unit organizational structure in computer's file system for storing and locating files or more folders. Python now supports a number of APIs to list the directory contents. For instance, we can use the Path.iterdir, os.scandir, os.walk, Path.rglob, or os.listdir functions.

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but alike other concepts of Python, this concept here is also easy and short. Python treats file differently as text or binary and this is important. Each line of code includes a sequence of characters and they form text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun. Let's start with Reading and Writing files.

Working of open() function

We use open () function in Python to open a file in read or write mode. As explained above, open ( ) will return a file object. To return a file object we use open() function along with two arguments, that accepts file name and the mode, whether to read or write. So, the syntax being: open(filename, mode). There are three kinds of mode, that Python provides and how files can be opened:

" r ", for reading.

“ w “, for writing.

“ a “, for appending.

“ r+ “, for both reading and writing

Code:

File:

```
f=open("hi1.txt","w")
str=input("Enter a string: ")
f.write(str)
f.close()
f=open("hi1.txt","r")
str1=f.read()
print(str1)
f.close()

f=open("hi2.txt","w")
print("Enter characters till the symbol @ is pressed")
while str!='@':
    str=input()
    if(str !='@'):
        f.write(str+'\n')
    else:
        f.close()

f=open("hi1.txt","a+")
print("Enter characters till the symbol @ is pressed")
while str!='@':
    str=input()
    if(str !='@'):
        f.write(str+'\n')
    else:
        f.seek(0,0)
f=open("hi1.txt","r")
str2=f.read()
print(str2)
f.close()
```

Output:

```
Hello World!

Hello World!
Python is interesting.
```

Dictionary:

```
import os
#creates in C:\
c=os.mkdir('C:/Users\student\Desktop\Veda')
print('New Directory Created')
c1=os.mkdir('C:/Users\student\Desktop\Vaidehi')
print('New Directory Created')
#gets current working directory
print("Current Working Directory:")
print("String format :",os.getcwd())
print("Byte string format :",os.getcwd())
#changing directory
print("Changing the directory:")
print("Previous directory :",os.getcwd())
os.chdir('C:/Users\student\Desktop\Veda')
print("Current directory :",os.getcwd())
#removing a directory
print("Removing a directory")
os.rmdir('C:/Users\student\Desktop\Vaidehi')
#to check whether the directory is removed
print("Is the directory present:")
check=('C:/Users\student\Desktop\Vaidehi')
print(os.path.isdir(check))
#renaming a file
print(os.chdir('C:/Users\student\Desktop\python'))
os.rename('Name.txt','Rename.txt')
```

Conclusion:

In conclusion, working with files and directories in Python is a straightforward process thanks to the built-in modules and functions available. By using the open( ) function, we can open files and specify the mode in which we want to interact with them. Reading and writing to files can be accomplished using methods like read( ) and write( ), respectively.

When working with files, it's essential to release system resources by closing the file after we're done with it. However, Python provides a more convenient and safer way to handle files using context managers with the with statement. This approach ensures that the file is automatically closed when the block of code is exited, even if an exception occurs.

On the other hand, dictionaries are a powerful data structure that allows us to store and access data in a key-value pair format. Dictionaries can be used to store and manipulate data in a more organized and efficient way than using lists or tuples. By combining the use of files and dictionaries, we can create powerful programs that can process, manage, and analyze large datasets.

Overall, understanding how to work with files and dictionaries in Python is crucial for any data processing, file management, or system administration task. With the right knowledge and practice, we can leverage these tools to create efficient and effective programs that can handle complex data processing tasks.