| |
|---|
| Experiment No. 5 |
| Exploring Files and directories: Python program to append data to existing file and then display the entire file |
| Date of Performance: |
| Date of Submission: |

# 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:**

```
def append_data_to_file(file_name, data):
    """Append data to an existing file."""
    with open(file_name, 'a') as file:
        file.write(data + '\n')


def display_file_contents(file_name):
    """Display the entire contents of a file."""
    with open(file_name, 'r') as file:
        contents = file.read()
        print("Contents of the file:")
        print(contents)


# File name
file_name = "example.txt"


# Data to append
data_to_append = "This is new data to append."
```

```
# Append data to file

append_data_to_file(file_name, data_to_append)


# Display entire file

display_file_contents(file_name)
```

**RESULTS:**

```
C:\Users\anand\PycharmProjects\pythonProject\.venv\Scripts\python.exe C:\Users\anand\PycharmProjects\pythonProject\main.py
Contents of the file:
This is new data to append.


Process finished with exit code 0
```

**Conclusion:**The exploration of directories and files in Python provides a comprehensive understanding of file handling operations and directory navigation within the language. Through functionalities like appending data to existing files and displaying file contents, developers gain valuable insights into managing file I/O operations effectively. This exploration equips programmers with the necessary skills to manipulate and interact with files and directories, facilitating the development of robust and scalable applications that incorporate file management capabilities seamlessly. Python's intuitive file handling mechanisms empower developers to handle various file-related tasks efficiently, enhancing productivity and enabling the creation of versatile software solutions across different domains.