

EXPERIMENT NO : 5A

Python programs to implement File Handling operations in Python

NAME : AKASH RAMKRIT YADAV

ID.NO: VU4F2122016

BATCH : A

BRANCH : IT

DIV : A

Aim :- python programs to implement File Handling operations in Python

THEORY:

OUTPUT:

Python 3.11.0a4 (main, Mar 13 2023, 10:57:32) [MSC v.1929 32 bit (Intel)] on win32

Type "help", "copyright", "credits" or "license()" for more information.

#AKASH YADAV ID.NO:VU4F2122016 EXP:5A DATE:13/3/2023

File Handling in Python:

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 like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a 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 the reading and writing files.

Python File Open

Python has several functions :

=> creating, reading, updating, and deleting files.

File Handling

The key function for working with files in Python is the `open()` function.

The `open()` function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a file:

`"r"` - Read - Default value. Opens a file for reading, error if the file does not exist

`"a"` - Append - Opens a file for appending, creates the file if it does not exist

`"w"` - Write - Opens a file for writing, creates the file if it does not exist

`"x"` - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

`"t"` - Text - Default value. Text mode

`"b"` - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

The code above is the same as:

```
f = open("demofile.txt", "rt")
```

Because `"r"` for read, and `"t"` for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Python File Open

Open a File on the Server

Assume we have the following file, located in the same folder as Python:

```
demofile.txt
```

```
Hello! Welcome to demofile.txt  
This file is for testing purposes.  
Good Luck!
```

To open the file, use the built-in `open()` function.

The `open()` function returns a file object, which has a `read()` method for reading the content of the file:

Example:

```
f = open("demofile.txt", "r")  
print(f.read())
```

If the file is located in a different location, you will have to specify the file path, like this:

Example

Open a file on a different location:

```
f = open("D:\\myfiles\\welcome.txt", "r")  
print(f.read())
```

Read Only Parts of the File

By default the `read()` method returns the whole text, but you can also specify how many characters you want to return:

Example

Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Python File Write

Write to an Existing File

To write to an existing file, you must add a parameter to the `open()` function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Example:

Open the file "demofile2.txt" and append content to the file:

```
f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

Example

Open the file "demofile3.txt" and overwrite the content:

```
f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the overwriting:
f = open("demofile3.txt", "r")
print(f.read())
```

Note: the "w" method will overwrite the entire file.

Create a New File

To create a new file in Python, use the `open()` method, with one of the following parameters:

"x" - Create - will create a file, returns an error if the file exist

"a" - Append - will create a file if the specified file does not exist

"w" - Write - will create a file if the specified file does not exist

Example

Create a file called "myfile.txt":

```
f = open("myfile.txt", "x")
```

Result: a new empty file is created!

Example

Create a new file if it does not exist:

```
f = open("myfile.txt", "w")
```

Python Delete File

Delete a File

To delete a file, you must import the OS module, and run its `os.remove()` function:

Example:

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist:

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Example

Check if file exists, *then* delete it:

```
import os
if os.path.exists("demofile.txt"):
    os.remove("demofile.txt")
else:
    print("The file does not exist")
```

Delete Folder

To delete an entire folder, use the `os.rmdir()` method:

Example

Remove the folder "myfolder":

```
import os  
os.rmdir("myfolder")
```

Note: You can only remove *empty* folders.

Advantages:

- **Versatility:** File handling in Python allows you to perform a wide range of operations, such as creating, reading, writing, appending, renaming, and deleting files.
- **Flexibility:** File handling in Python is highly flexible, as it allows you to work with different file types (e.g. text files, binary files, CSV files, etc.), and to perform different operations on files (e.g. read, write, append, etc.).
- **User-friendly:** Python provides a user-friendly interface for file handling, making it easy to create, read, and manipulate files.
- **Cross-platform:** Python file handling functions work across different platforms (e.g. Windows, Mac, Linux), allowing for seamless integration and compatibility.

Disadvantages:

- **Error-prone:** File handling operations in Python can be prone to errors, especially if the code is not carefully written or if there are issues with the file system (e.g. file permissions, file locks, etc.).
- **Security risks:** File handling in Python can also pose security risks, especially if the program accepts user input that can be used to access or modify sensitive files on the system.
- **Complexity:** File handling in Python can be complex, especially when working with more advanced file formats or operations. Careful attention must be paid to the code to ensure that files are handled properly and securely.

- **Performance:** File handling operations in Python can be slower than other programming languages, especially when dealing with large files or performing complex operations.

CODE:

```
import os

def create_file(filename):
    try:
        with open(filename, 'w') as f:
            f.write('this is AKASH file\nmy id is vu4f2122016\ni am in SE IT A\n')
        print("File " + filename + " created successfully.")
    except IOError:
        print("Error: could not create file " + filename)

def read_file(filename):
    try:
        with open(filename, 'r') as f:
            contents = f.read()
            print(contents)
    except IOError:
        print("Error: could not read file " + filename)

def append_file(filename, text):
    try:
        with open(filename, 'a') as f:
            f.write(text)
```



```
        print("Text appended to file " + filename + " successfully.")

except IOError:

    print("Error: could not append to file " + filename)


def rename_file(filename, new_filename):

    try:

        os.rename(filename, new_filename)

        print("File " + filename + " renamed to " + new_filename + " successfully.")

    except IOError:

        print("Error: could not rename file " + filename)


def delete_file(filename):

    try:

        os.remove(filename)

        print("File " + filename + " deleted successfully.")

    except IOError:

        print("Error: could not delete file " + filename)
```

```
create_file("ar.txt")

read_file("ar.txt")

append_file("ar.txt", "additinal text:i am an engineering student.\n")

read_file("ar.txt")

rename_file('ar.txt', 'new_ar.txt')

read_file("new_ar.txt")

delete_file('new_ar.txt')
```

OUTPUT:

File ar.txt created successfully.

this is AKASH file

my id is vu4f2122016

i am in SE IT A

Text appended to file ar.txt successfully.

this is AKASH file

my id is vu4f21220016

i am in SE IT A

additinal text:i am an engineering student.

File ar.txt renamed to new_ar.txt successfully.

this is AKASH file

my id is vu4f21220016

i am in SE IT A

additinal text:i am an engineering student.

File new_ar.txt deleted successfully.