Lecture 8

Working with files

Need for a data file

- To Store data in organized manner
- To store data permanently
- To access data faster
- To Search data faster
- To easily modify data later on

File Handling

- File handling in Python enables us to create, update, read, and delete the files stored on the file system through our python program.
- In Python, File Handling consists of following three steps:
  - Open the file.
  - Process file i.e perform read or write operation.
  - Close the file.

Opening a file and File object function: **open()**

- Before any reading or writing operation of any file, it must be opened first.
- A file can be opened with a built-in function called open(). This function takes in the file's address and the access_mode and returns a file object.
- A file object allows us to use, access and manipulate all the user accessible files.
- Syntax: file_object = open(file_name, access_mode)
  - File_name is the full name of the file with its path written between quates
  - Access_mode determines what kind of operations can be performed with file,like read,write etc.
- Note : for text file operation file extension should be .txt

File access modes

- A mode indicates how a file is opened, such as whether or not writing to the file is allowed, if existing contents of the file are overwritten or appended, etc.

- There are different types of access_modes:

Mode	Description	Allow read?	Allow write?	Create missing file?	Overwrite file?
'r'	Open the file for reading.	Yes	No	No	No
'w'	Open the file for writing. If the file does not exist, then the file is created. Contents of an existing file are overwritten.	No	Yes	Yes	Yes
'a'	Open the file for appending. If the file does not exist, then the file is created. Writes are added to the end of existing file contents.	No	Yes	Yes	No

Read/write text file

- The read() Method
  - It reads the entire file and returns it contents in the form of a string.
  - Reads at most size bytes or less if end of file occurs. if size not mentioned then read the entire file contents
- The write() Method
  - It writes the contents to the file in the form of string.
  - It does not return value. Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

Reading file function: **read([size])**

- read() function reads the entire file and returns it contents in the form of a string.
- Reads at most size bytes from the file (less if the read hits EOF before obtaining size bytes).
- If the size argument is negative or omitted, read all data until EOF is reached.

```
f = open("a.txt", 'a')
text=f.read()
print(text)
```

Reading a line from the file function: **readline(size)**

- readline() function reads the first line of the file i.e till a newline character or an EOF in case of a file having a single line and returns a string.
- If the size argument is present and non-negative, it is a maximum byte count (including the trailing newline) and an incomplete line may be returned.
- An empty string is returned only when EOF is encountered immediately.

```
f = open("a.txt", 'w')
line1 = 'Welcome to python'
f.write(line1)
line2="\nlecture 6"
f.write(line2)
f.close()
f = open("a.txt", 'r')
text = f.readline()
print(text)
text = f.readline()
print(text)
f.close()
```

```
Welcome to python

lecture 6
```

Writing to a file function: **write(“string”)**

- **write(“string”) function** writes the contents of string to the file.
- It has no return value.
- Due to buffering, the string may not actually show up in the file until the flush() or close() method is called.

```
f = open("a.txt", 'w')
line1 = 'Welcome to python'
f.write(line1)
line2="\nWorking with file,  lecture - 6"
f.write(line2)
f.close()
f = open("a.txt", 'r')
text = f.read()
print(text)
f.close()
```

```
Welcome to python
Working with file,  lecture - 6
```

Writing to a file function: **write("string")**

- Numeric values must be converted to strings to be able to write it into a file

```
num1 = 5
num2 = 7.5
num3 = num1 + num2

f = open('myfile.txt', 'w')
f.write(num1)
f.write(' + ')
f.write(num2)
f.write(' = ')
f.write(num3)
f.close()

f=open('myfile.txt', 'r')
text=f.read()
print(text)
f.close()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback
/tmp/ipython-input-191069089.py in <cell line: 0>()
      4
      5 f = open('myfile.txt', 'w')
----> 6 f.write(num1)
      7 f.write(' + ')
      8 f.write(num2)

TypeError: write() argument must be str, not int
```

```
num1 = 5
num2 = 7.5
num3 = num1 + num2

f = open('myfile.txt', 'w')
f.write(str(num1))
f.write(' + ')
f.write(str(num2))
f.write(' = ')
f.write(str(num3))
f.close()

f=open('myfile.txt', 'r')
text=f.read()
print(text)
f.close()
```

```
5 + 7.5 = 12.5
```

%%writefile

- The %%writefile magic command is specific to Jupyter Notebook and IPython environments. It allows you to write the contents of a cell directly to a file. However, this command does not work in a regular Python script or interactive Python shell.
- Syntax: %%writefile filename.txt

```
%%writefile mydata.txt
5
100
30
20
```

```
Writing mydata.txt
```

```
f=open('mydata.txt')
text=f.read()
print(text)
f.close()
```

```
5
100
30
20
```

Closing a file function: **close()**

- close() function is used to close an open file.
- After using this method, an opened fi will be closed and a closed file canno be read or written any more.

```
f = open("a.txt", 'a')
print(f.closed)
print("Name of the file is",f.name)
f.close()
print(f.closed)
```

```
False
Name of the file is a.txt
True
```

Iterating over lines in a file

- Example:

```
f = open("a.txt", 'w')
line1 = 'Welcome to python'
f.write(line1)
line2="\nlecture 6"
f.write(line2)

f = open("a.txt", 'r')
for text in f.readlines():
  print(text)
f.close()
```

- Output:

```
Welcome to python

lecture 6
```

Example: iterating and calculating average

```
%%writefile mydata.txt
5
100
30
20
```

```
Writing mydata.txt
```

```
# Read file contents
print ('Reading in data....')
f = open('mydata.txt')
lines = f.readlines()
f.close()

# Iterate over each line
print('\nCalculating average....')
total = 0
for ln in lines:
    total += int(ln)

# Compute result
avg = total/len(lines)
print(f'Average value: {avg}')
```

```
Reading in data....

Calculating average....
Average value: 38.75
```

flush() method

- It is a file method can be called to force the interpreter to flush the output buffer to disk.
-

```
import os

# Open a file with default line-buffering.
f = open('myfile.txt', 'w')

# No newline character, so not written to disk immediately
f.write('Write me to a file, please!')

# Force output buffer to be written to disk
f.flush()
os.fsync(f.fileno())
# The fileno() method returns the file descriptor of the stream, as a number.
# The os.fsync() method forces write of file with file descriptor fd to disk.
```

The 'with' statement

- A with statement can be used to open a file, execute a block of statements, and automatically close the file when complete.
- The with statement creates a context manager, which manages the use of a resource, such as a file, by performing set-up and teardown operations.

```
with open('myfile.txt', 'r') as myfile:
    # Statement-1
    # Statement-2
    # ....
    # Statement-N
```

Example

```
# Open a file for reading and appending
with open('myfile.txt', 'r') as f:
    # Read in two integers
    num1 = int(f.readline())
    num2 = int(f.readline())

    product = num1 * num2

    # Write back result on own line
    f.write('\n')
    f.write(str(product))

# No need to call f.close() - f closed automatically
print('Closed myfile.txt')
```

Handling File Not Found Error

- If a file does not exist, trying to open it in read mode causes a **FileNotFoundError**. We can handle this using **try-except**
  - The **try** block attempts to open **non_existent_file.txt** in read mode.
  - Since the file does not exist, a **FileNotFoundError** is raised.
  - The **except** block catches this error and prints a user-friendly message.

```
try:
    # Attempt to open a non-existent file
    file = open("non_existent_file.txt", "r")
    content = file.read()
    file.close()
except FileNotFoundError:
    print("Error: The file does not exist.")
```

```
Error: The file does not exist.
```

Handling Permission Error for File

- Trying to open a file without the necessary permissions raises a **PermissionError**.
  - The **try** block attempts to open **/root/protected_file.txt**, which may require administrator privileges.
  - If the script lacks the necessary permissions, a **PermissionError** is raised.
  - The **except** block catches the error and informs the user.

```
try:
    # Attempt to open a restricted file
    file = open("/root/protected_file.txt", "r")
    content = file.read()
    file.close()
except PermissionError:
    print("Error: You do not have permission to access this file.")
```