# File Operations (or) File Handling

File operations contains operations like opening a file, reading from it, writing into it, closing it, renaming a file, deleting a file, and various file methods.

To store data temporarily and permanently, we use files. A file is the collection of data stored on a disk in one unit identified by filename.

Common File Operations Include:

Python File Handling
1. Create Files
2. Read Files
3. Write to Files
1. List Files From Directory
2. Copy, Rename, Delete Files from Directory
3. Copy, Delete Directories

## Create File in Python:

Below is the list of access modes for creating an a file.

| File Mode | Meaning |
|---|---|
| `w` | Create a new file for writing. If a file already exists, it truncates the file first. Use to create and write content into a new file. |
| `x` | Open a file only for exclusive creation. If the file already exists, this operation fails. |

| File Mode | Meaning |
|---|---|
| `a` | Open a file in the append mode and add new content at the end of the file. |
| `b` | Create a binary file |
| `t` | Create and open a file in a text mode |

**Note**:

- The file is created in the same directory where our program/script is running.
- If you have not specified any specific path(directory location), the file is created in the working directory. It is known as creating a file using the **relative path**. A relative path contains the current directory and then the file name.

```
# Create File in Python
# create a empty text file
# in current directory
fp = open('sales.txt', 'x')
fp.close()


#Use access mode w if you want to create and write content into a
file.
# create a empty text file
fp = open('sales_2.txt', 'w')
fp.write('first line')
fp.close()
```

```python
import os

# list files from a working directory
print(os.listdir())

# verify file exist
print(os.path.isfile('sales.txt'))
```

## ‘with’ keyword in Python:

**with** statement in Python is used in exception handling to make the code cleaner and much more readable. It simplifies the management of common resources like file streams. Observe the following code example on how the use of with statement makes code cleaner.

```python
# 1) without using with statement
file = open('file_path', 'w')
file.write('hello world !')
file.close()

# 2) without using with statement
file = open('file_path', 'w')
try:
    file.write('hello world')
finally:
    file.close()

# using with statement
with open('file_path', 'w') as file:
    file.write('hello world !')
```

**Explanation:** Notice that unlike the first two implementations, there is no need to call file.close() when using with statement. The with statement itself ensures proper acquisition and release of resources. An exception during the file.write() call in the first implementation can prevent the file from closing properly which may introduce several bugs in the code, i.e. many changes in files do not go into effect until the file is properly closed.

The second approach in the above example takes care of all the exceptions but using the with statement makes the code compact and much more readable. Thus, with statement helps avoiding bugs and leaks by ensuring that a resource is properly released when the code using the resource is completely executed. The with statement is popularly used with file streams, as shown above and with Locks, sockets, subprocesses and telnets etc.

```
# Create File In A Specific Directory
# create a text file for writing
with open(r'E:\santosh\reports\profit.txt', 'w') as fp:
    fp.write('This is first line')
    pass



# using the absolute path.
import os

# list files a directory
print(os.listdir(r'E:\santosh\reports'))
# output ['sample.txt', 'sales.txt', 'sales_2.txt' 'profit.txt']

# verify file exist
print(os.path.isfile(r'E:\santosh\reports\profit.txt'))
# output True
```

Also, you can **join directory path and file name** to create file at the specified location.

If you have a directory path and file name in two variables, use the `os.path.join()` function to construct a full path. This function accepts the directory path and file name as arguments and constructs an absolute path to create a file.

```python
import os

# Specify the directory path
path = r'F:\santosh\account'
file_name = 'revenue.txt'

# Creating a file at specified folder
# join directory and file path
with open(os.path.join(path, file_name), 'w') as fp:
    # uncomment below line if you want to create an empty file
    fp.write('This is a new line')
```

```python
# create file if not exists.

import os

file_path = r'E:\santosh\account\profit.txt'
if os.path.exists(file_path):
    print('file already exists')
else:
    # create a file
    with open(file_path, 'w') as fp:
        # uncomment if you want empty file
        fp.write('This is first line')
```

```python
# Use file access mode x

# The access mode x open a file for exclusive creation. If the file
already exists, this operation fails with FileExistsError. Use try-
except block to handle this error.

try:
    file_path = r'E:\santosh\account\profit.txt'
    # create file
    with open(file_path, 'x') as fp:
        pass
except:
    print('File already exists')
```

***Note**: When exclusive creation is specified, it means that this mode will not create a file if the file with the specified name already exists. In the x mode, the file is only writeable, but in x+ mode, the file is opened as both readable and writeable.*

```
# Create File with a DateTime
from datetime import datetime

# get current date and time
x = datetime.now()

# create a file with date as a name day-month-year
file_name = x.strftime('%d-%m-%Y.txt')
with open(file_name, 'w') as fp:
    print('created', file_name)

# with name as day-month-year-hours-minutes-seconds
file_name_2 = x.strftime('%d-%m-%Y-%H-%M-%S.txt')
with open(file_name_2, 'w') as fp:
    print('created', file_name_2)

# at specified directory
file_name_3 = r"E:\demos\files_demos\account\\" + x.strftime('%d-%m-
%Y-%H-%M-%S.txt')
with open(file_name_3, 'w') as fp:
    print('created', file_name_3)
```