UNIT III: File, Exception and Database Handling

A) File handling in python

- File handling is an important part of any web application.
- Python has several functions for opening, creating, reading, updating, and deleting files.

1. File open:

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:
    1. **"r"** - Read - Default value. Opens a file for reading, error if the file does not exist
    2. **"a"** - Append - Opens a file for appending, creates the file if it does not exist
    3. **"w"** - Write - Opens a file for writing, creates the file if it does not exist
    4. **"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**

1. **"t"** - Text - Default value. Text mode
2. **"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("testfile.txt")
```

The code above is the same as:

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

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

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("testfile.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("testfile.txt", "r")
print(f.read(5))
```

Read Lines

You can return one line by using the **readline()** method:
Example: Read one line of the file

```
f = open("testfile.txt", "r")
print(f.readline())
```

By calling readline() two times, you can read the two first lines:
Example: Read two lines of the file

```
f = open("testfile.txt", "r")
print(f.readline())
print(f.readline())
```

By looping through the lines of the file, you can read the whole file, line by line:

Loop through the file line by line:

```
f = open("testfile.txt", "r")
for x in f:
  print(x)
```

2. File Close:

It is a good practice to always close the file when you are done with it.

Example: Close the file when you are finish with it:

```
f = open("testfile.txt", "r")
print(f.readline())
f.close()
```

3. 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 "testfile2.txt" and append content to the file:

```
f = open("testfile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("testfile2.txt", "r")
print(f.read())
```

Example: Open the file "testfile3.txt" and overwrite the content:

```
f = open("testfile3.txt", "w")
f.write("deleted the content")
```

```
f.close()
#open and read the file after the appending:
f = open("testfile3.txt", "r")
print(f.read())
```

4. Create a New File

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

1. **"x" -** Create - will create a file, returns an error if the file exist
2. **"a"** - Append - will create a file if the specified file does not exist
3. **"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")
```

5. Delete a File

To delete a file, you must import the OS module, and run its **os.remove()** function:
Example:Remove the file "testfile.txt":

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

6. 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("testfile.txt"):
        os.remove("testfile.txt")
else:
        print("The file does not exist")
```

7. Delete Folder

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

Example:Remove the folder "myfolder":

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

8. Python File Seek(): Move File Pointer Position

What is seek() in Python

The seek() function **sets the position of a file pointer** and the tell() function **returns the current position** of a file pointer.

A **file handle or pointer denotes the position from which the file contents will be read or written**. File handle is also called as file pointer or cursor.

For example, when you open a file in write mode, the file pointer is placed at the 0^{th} position, i.e., at the start of the file. However, it changes (increments) its position as you started writing content into it.Or, when you read a file line by line, the file pointer moves one line at a time.

Sometimes we may have to read only a specific portion of the file, in such cases use the seek() method to move the file pointer to that position.

For example, use the seek() function to do the file operations like: –

How to Use seek() Method

seek() method sets the file's current position, and then we can read or write to the file from that position.

f.seek(offset, whence)

The allowed values for the whence argument are: –

- A whence value of **0** means from the beginning of the file.

- A whence value of **1** uses the current file position
- A whence value of **2** uses the end of the file as the reference point.
  The default value for the whence is the beginning of the file, which is **0**

Seek Operation	Meaning
f.seek(0)	Move file pointer to the beginning of a File
f.seek(5)	Move file pointer five characters ahead from the beginning of a file.
f.seek(0, 2)	Move file pointer to the end of a File
f.seek(5, 1)	Move file pointer five characters ahead from the current position.
f.seek(-5, 1)	Move file pointer five characters behind from the current position.
f.seek(-5, 2)	Move file pointer in the reverse direction. Move it to the 5th character from the end of the file

tell() Function To Get File Handle Position
file_object.tell()

9. File Rename

Steps to Rename File in Python

To rename a file, Please follow these steps:

1. **Find the path of a file to rename**
   To rename a file, we need its path. The path is the location of the file on the disk. An **absolute path** contains the complete directory list required to locate the file. A **relative path** contains the current directory and then the file name.

2. **Decide a new name**
   Save an old name and a new name in two separate variables.
   old_name='details.txt'
   new_name = 'new_details.txt'
3. **Use rename() method of an OS module**
   Use the os.rename() method to rename a file in a folder. Pass both the old name and a new name to the os.rename(old_name, new_name) function to rename a file.

import os

```
# Absolute path of a file
old_name = r"E:\demos\files\reports\details.txt"
new_name = r"E:\demos\files\reports\new_details.txt"

# Renaming the file
os.rename(old_name, new_name)
```

10. Delete (Remove) Files and Directories in Python

- Deleting file using the os module and pathlib module
- Deleting files from a directory
- Remove files that match a pattern (wildcard)
- Delete empty directory
- Delete content of a directory (all files and sub directories)

How to Delete a File in Python

Python provides strong support for file handling. We can delete files using different methods and the most commonly used one is the os.remove() method. Below are the steps to delete a file.

1. **Find the path of a file**
   We can delete a file using both relative path and absolute path. The path is the location of the file on the disk.
   An **absolute path** contains the complete directory list required to locate the file. And A **relative path** includes the current directory and then the file name.
   For example, /home/Pynative/reports/samples.txt is an absolute path to discover the samples.txt.

2. **Use os.remove() function to delete File**
   The OS module in Python provides methods to interact with the Operating System in Python. The remove() method in this module is used to remove/delete a file path. First, import the os module and Pass a **file path** to the os.remove('file_path') function to delete a file from a disk
3. **Use the rmtree() function of shutil module to delete a directory**
   Import the shutil module and pass the directory path to shutil.rmtree('path') function to delete a directory and all files contained in it.

Remove file with relative path

```
import os
# removing a file with relative path
os.remove("sales_1.txt")
```

Remove file with absolute path

```
import os
# remove file with absolute path
os.remove(r"E:\demos\files\sales_2.txt")
```

11. file copy

Steps to Copy a File in Python

Python provides strong support for file handling. We can copy single and multiple files using different methods and the most commonly used one is the shutil.copy() method. The below steps show how to copy a file from one folder to another.

1. **Find the path of a file**
   We can copy a file using both relative path and absolute path. The path is the location of the file on the disk.
   An **absolute path** contains the complete directory list required to locate the file. For example, /home/Pynative/samples.txt is an absolute path to discover the samples.txt.
2. **Use the shutil.copy() function**
   The shutil module offers several functions to perform high-level operations on files and collections of files. The copy() function in this module is used to copy files from one directory to another.First, import the shutil module and Pass a source file path and destination directory path to the copy(src, dst) function.
3. **Use the os.listdir() and shutil copy() function to copy all files**
   Suppose you want to copy all files from one directory to another, then use the os.listdir() function to list all files of a source folder, then iterate a list using a for loop and copy each file using the copy() function.
4. **Use copytree() function to copy entire directory**
   The shutil.copytree(src, dst) recursively copy an entire directory tree rooted at src to a directory named dst and return the destination directory

Example: Copy Single File

```
import shutil

src_path = r"E:\demos\files\report\profit.txt"
dst_path = r"E:\demos\files\account\profit.txt"
shutil.copy(src_path, dst_path)
print('Copied')
```

Copy All Files From A Directory

Sometimes we want to copy all files from one directory to another. Follow the below steps to copy all files from a directory.

- Store the source and destination directory path into two variables
- Get the list of all files present in the source folder using the os.listdir() function. It returns a list containing the names of the files and folders in the given directory.
- Iterate over the list using a for loop to get the individual filenames
- In each iteration, concatenate the current file name with the source folder path
- Now use the shutil.copy() method to copy the current file to the destination folder path.

Copy Entire Directory

Sometimes we need to copy an entire folder, including all files and subfolders contained in it. Use the copytree() method of a shutil module to copy the directory recursively

```
import shutil

source_dir = r"E:\demos\files\reports"
destination_dir = r"E:\demos\files\account"
shutil.copytree(source_dir, destination_dir)
```

B) Database handling using python

1. Create Connection

Start by creating a connection to the database.

Use the username and password from your MySQL database:

demo_mysql_connection.py:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

print(mydb)
```

2.Creating a Database

To create a database in MySQL, use the "CREATE DATABASE" statement:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE mydatabase")
```

3.Check if Database Exists

```
import mysql.connector

mydb = mysql.connector.connect(
```

```
  host="localhost",
  user="yourusername",
  password="yourpassword"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW DATABASES")

for x in mycursor:
  print(x)
```

4.Creating a Table

To create a table in MySQL, use the "CREATE TABLE" statement.

Make sure you define the name of the database when you create the connection

Create a table named "customers":

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address
VARCHAR(255))")
```

5.Check if Table Exists

You can check if a table exist by listing all tables in your database with the "SHOW TABLES" statement:

```
import mysql.connector

mydb = mysql.connector.connect(
```

```
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("SHOW TABLES")
for x in mycursor:
  print(x)
```

6.Primary Key

When creating a table, you should also create a column with a unique key for each record.

This can be done by defining a PRIMARY KEY.

We use the statement "INT AUTO_INCREMENT PRIMARY KEY" which will insert a unique number for each record. Starting at 1, and increased by one for each record.

Create primary key when creating the table:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(255), address VARCHAR(255))")
```

Create primary key on an existing table:

```
import mysql.connector
```

```
mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)
mycursor = mydb.cursor()
mycursor.execute("ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT
PRIMARY KEY")
```

7.Insert Into Table

```
sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)
mydb.commit()
print(mycursor.rowcount, "record inserted.")
```

Select From a Table

```
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

8.Selecting Columns

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

```
mycursor.execute("SELECT name, address FROM customers")
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

Using the fetchone() Method

If you are only interested in one row, you can use the fetchone() method.

The fetchone() method will return the first row of the result:

```
mycursor.execute("SELECT * FROM customers")
myresult = mycursor.fetchone()
print(myresult)
```

9.Select With a Filter

When selecting records from a table, you can filter the selection by using the "WHERE" statement:

```
sql = "SELECT * FROM customers WHERE address ='kolh'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

10.Wildcard Characters

You can also select the records that starts, includes, or ends with a given letter or phrase.

Use the % to represent wildcard characters:

```
sql = "SELECT * FROM customers WHERE address LIKE '%way%'"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

Prevent SQL Injection

When query values are provided by the user, you should escape the values.

This is to prevent SQL injections, which is a common web hacking technique to destroy or misuse your database.

The mysql.connector module has methods to escape query values:

```
sql = "SELECT * FROM customers WHERE address = %s"
adr = ("Yellow Garden 2", )
mycursor.execute(sql, adr)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

11.Python MySQL Order By

Use the ORDER BY statement to sort the result in ascending or descending order.

The ORDER BY keyword sorts the result ascending by default. To sort the result in descending order, use the DESC keyword.

```
sql = "SELECT * FROM customers ORDER BY name"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

12.ORDER BY DESC

Use the DESC keyword to sort the result in a descending order.

```
sql = "SELECT * FROM customers ORDER BY name DESC"
mycursor.execute(sql)
myresult = mycursor.fetchall()
for x in myresult:
  print(x)
```

13.Delete Record

You can delete records from an existing table by using the "DELETE FROM" statement:

```
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted")
```

14.Delete a Table

You can delete an existing table by using the "DROP TABLE" statement:

```
mycursor = mydb.cursor()
sql = "DROP TABLE customers"
mycursor.execute(sql)
```

15.Drop Only if Exist

If the table you want to delete is already deleted, or for any other reason does not exist, you can use the IF EXISTS keyword to avoid getting an error.

```
mycursor = mydb.cursor()
sql = "DROP TABLE IF EXISTS customers"
mycursor.execute(sql)
```

16.Update Table

You can update existing records in a table by using the "UPDATE" statement:

```
sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) affected")
```

17.Python MySQL Limit

Limit the Result

You can limit the number of records returned from the query, by using the "LIMIT" statement:

```
mycursor.execute("SELECT * FROM customers LIMIT 5")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```