1. **Advanced topics in Python**

- 1. **Advanced topics in Python**
- 2. **Python Database**
- 3. MySQL
    - 3.1. Downloads
    - 3.2. Install
    - 3.3. **Connection to database**
        - 3.3.1. Open Database Connection
    - 3.4. Running Queries using Python
        - 3.4.1. Create Database
        - 3.4.2. Create a Table
        - 3.4.3. Primary Key
        - 3.4.4. Insert into table
        - 3.4.5. fetchone()
        - 3.4.6. fetchall()
        - 3.4.7. rowcount
        - 3.4.8. Update
        - 3.4.9. Sort
        - 3.4.10. Delete
        - 3.4.11. Drop
- 4. **MongoDB**
    - 4.1. Create a Database
    - 4.2. Check if database exists
    - 4.3. Creating a Collection
    - 4.4. Check if the collection exists
    - 4.5. Insert
    - 4.6. Return the _id Field
    - 4.7. Insert Multiple Documents, with Specified IDs
    - 4.8. Find One
    - 4.9. Find All
    - 4.10. Return Only Some Fields
    - 4.11. Filter the Result
    - 4.12. Sort the Result
    - 4.13. Sort Descending
    - 4.14. Delete Document
    - 4.15. Delete All Documents in a Collection
    - 4.16. Delete Collection
    - 4.17. Update Collection
    - 4.18. Limit the Result
- 5. **Python File Open**
    - 5.1. File Handling
        - 5.1.1. **File opening methods**
    - 5.2. open() function

- 5.3. File Methods
    - 5.3.1. read()
    - 5.3.2. readline()
    - 5.3.3. Close()
- 5.4. Write to an Existing File
- 5.5. Create a New File
- 5.6. Delete file
    - 5.6.1. Check if File exist
- 5.7. Delete Folder

2. **Python Database**

Python Database API supports a wide range of database servers such as −

- GadFly
- mSQL
- MySQL
- PostgreSQL
- Microsoft SQL Server 2000
- Informix
- Interbase
- Oracle
- Sybase

PostgreSQL and MySQL are two of the most common open source databases for storing Python web applications data.

3. MySQL

3.1. Downloads

Downloads MySQL database at https://www.mysql.com/downloads/.

3.2. Install

Install `python -m pip install mysql-connector`

3.3. **Connection to database**

3.3.1. Open Database Connection

To create a connection to database use the username and password from your MySQL database.

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password"
    )

print(db)
```

3.4. Running Queries using Python

3.4.1. Create Database

The **"CREATE DATABASE"** *statement is used to create a database in MySQL*

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# execute SQL query using execute() method.
cur.execute("CREATE DATABASE information")
```

```
# disconnect from server
db.close()
```

3.4.2. Create a Table

The **"CREATE TABLE"** *statement is used to create a table in MySQL*
and also define the name of the database when you create the connection

- **Example :**
  Create a table named "EMPLOYEE"

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# execute SQL query using execute() method.
cur.execute("CREATE TABLE EMPLOYEE (FIRST_NAME VARCHAR(255), LAST_NAME VARCHAR(255),AGE
INT, INCOME FLOAT")

# disconnect from server
db.close()
```

3.4.3. Primary Key

It is a unique key for each record. Create a column with a unique key for each record.
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. If the table already exists, use the **ALTER TABLE** *keyword.*

- **Example :**
  Create primary key on an existing table

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )
```

```
# prepare a cursor object using cursor() method
cur = db.cursor()

# execute SQL query using execute() method.
cur.execute("ALTER TABLE EMPLOYEE ADD COLUMN ID INT AUTO_INCREMENT PRIMARY KEY")

# disconnect from server
db.close()
```

3.4.4. Insert into table

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME,
       LAST_NAME, AGE, INCOME)
       VALUES ('%s', '%s', '%d', '%d' )" %
       ('David', 'son', 25, 20000)

try:
   # Execute the SQL command
   cur.execute(sql)
   # Commit your changes in the database
   db.commit()

except:
   # Rollback in case there is any error
   db.rollback()

# disconnect from server
db.close()
```

3.4.5. fetchone()

The fetchone() method is used if you are only interested in one row, this method will return the first row of the result.

- **Example :**
  Fetch only one row

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Execute the command
cur.execute("SELECT * FROM EMPLOYEE")

# fetch one row
result = cur.fetchone()

# Now print fetched result
print(result)

# disconnect from server
db.close()
```

3.4.6. fetchall()

The fetchall() method, which fetches all rows from the last executed statement

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Execute the command
cur.execute("SELECT * FROM EMPLOYEE")

# fetch all
result = cur.fetchall()

# Now print fetched result
for x in result:
  print(x)
```

```
# disconnect from server
db.close()
```

3.4.7. rowcount

This is a read-only attribute and returns the number of rows that were affected by an execute() method.

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Prepare SQL query to INSERT a record into the database.
sql = "INSERT INTO EMPLOYEE(FIRST_NAME,
       LAST_NAME, AGE, INCOME)
       VALUES ('%s', '%s', '%d', '%d' )" %
       ('David', 'son', 25, 20000)

# Execute the SQL command
cur.execute(sql)

# Commit your changes in the database
db.commit()

# using rowcount method
print(mycursor.rowcount, "record inserted.")
```

3.4.8. Update

UPDATE Operation on any database means to update one or more records, which are already available in the database.

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )
```

```
# prepare a cursor object using cursor() method
cur = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET AGE = 35 WHERE INCOME = '%d'" % (12000)

# Execute the SQL command
cur.execute(sql)

# Commit your changes in the database
db.commit()

# disconnect from server
db.close()
```

3.4.9. Sort

The ORDER BY statement is used 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.

- **Example :**
  Sort the result alphabetically by name

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Prepare SQL query to UPDATE required records
sql = "SELECT * FROM EMPLOYEE ORDER BY FIRST_NAME"


# Execute the SQL command
cur.execute(sql)

result = cur.fetchall()

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

3.4.10. Delete

DELETE operation is required to delete some records from your database. Here is an example to delete all the records from EMPLOYEE where INCOME is more than 20000 −

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE WHERE INCOME > '%d'" % (20000)

# Execute the SQL command
cur.execute(sql)

# Commit your changes in the database
db.commit()

# disconnect from server
db.close()
```

3.4.11. Drop

The "DROP TABLE" statement is used to delete an existing table.

Example
Delete the table "EMPLOYEE":

```
import mysql.connector

# Open database connection
db = mysql.connector.connect(
    host = "local_host",
    user = "your_username",
    passwd = "your_password",
    database = "information"
    )

# prepare a cursor object using cursor() method
cur = db.cursor()

# Prepare SQL query to DELETE table
```

```
sql = "DROP TABLE EMPLOYEE"

# Execute the SQL command
cur.execute(sql)

# Commit your changes in the database
db.commit()

# disconnect from server
db.close()
```

4. **MongoDB**

It is a document-oriented, open-source database program that is platform- independent. MongoDB, stores its data in documents using a JSON structure.

4.1. Create a Database

To create a database in MongoDB, start by creating a MongoClient object, then specify a connection URL with the correct ip address and the name of the database you want to create.

> Note : MongoDB will create the database if it does not exist, and make a connection to it.

- **Example :**
  Create a database called "information"

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["information"]
```

4.2. Check if database exists

```
import pymongo

client = pymongo.MongoClient('mongodb://localhost:27017/')

dblist = client.list_database_names()
if "information" in dblist:
  print("The database exists.")
```

4.3. Creating a Collection

To create a collection in MongoDB, use database object and specify the name of the collection you want to create. MongoDB will create the collection if it does not exist.

- **Example :**
  Create a collection called "employee":

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")

db = client["information"]

coll = db["employee"]
```

> Note : In MongoDB, a collection is not created until it gets content!

4.4. Check if the collection exists

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]

cllist = db.list_collection_names()

if "employee" in cllist:
  print("collection exists.")
```

4.5. Insert

The insert_one() method is used to insert a record, or document into a collection.

- **Example :**
  Insert a record in the "employee" collection

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

dict = { "name": "David", "income": 37000 }

x = coll.insert_one(dict)
print(x)
```

4.6. Return the _id Field

The insert_one() method returns a InsertOneResult object, which has a property, inserted_id, that holds the id of the inserted document.

- **Example :**
  Insert another record in the "customers" collection, and return the value of the _id field

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

dict = { "name": "David", "income": 37000 }

x = coll.insert_one(dict)
print(x.inserted_id)
```

4.7. Insert Multiple Documents, with Specified IDs

If you do not want MongoDB to assign unique ids for your document, you can specify the _id field when you insert the document(s).

> **Note :** Remember that the values has to be unique. Two documents cannot have the same _id.

- **Example :**

```
import pymongo
client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

list  = [
  { "_id": 1, "name": "David", "income": 20000},
  { "_id": 2, "name": "Peter", "income": 37000},
  { "_id": 3, "name": "John", "income": 25000},
  { "_id": 4, "name": "Amy", "income": 35000},
  { "_id": 5, "name": "Michael", "income": 15000},
  { "_id": 6, "name": "Sandy", "income": 22000},
  { "_id": 7, "name": "Ruhi", "income": 20000},
  { "_id": 8, "name": "Navi", "income": 22000},
  { "_id": 9, "name": "Susan", "income": 17000},
  { "_id": 10, "name": "Vicky", "income": 30000},
  { "_id": 11, "name": "Savi", "income": 44000},
  { "_id": 12, "name": "William", "income": 41000},
  { "_id": 13, "name": "Robin", "income": 38000},
  { "_id": 14, "name": "Jim", "income": 10000}
]

x = coll.insert_many(list)

print(x.inserted_ids)
```

4.8. Find One

The find_one() method is used to select data from a collection. The find_one() method returns the first occurrence in the selection.

- **Example :**
  Find the first document in the customers collection.

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

x = coll.find_one()
```

```
print(x)
```

4.9. Find All

The find() method is used to select data from a table in MongoDB. The find() method returns all occurrences in the selection.

- **Example :**
  Return all documents in the "customers" collection, and print each document

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

for x in coll.find():
  print(x)
```

4.10. Return Only Some Fields

If you want to return only some fields then describe which fields to include in the result.

- **Example :**
  Return only the names and income, not the _ids

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

for x in coll.find({},{ "_id": 0, "name": 1, "income": 1 }):
  print(x)
```

> Note : You are not allowed to specify both 0 and 1 values in the same object (except if one of the fields is the _id field). If you specify a field with the value 0, all other fields get the value 1, and vice versa:

- **Example :**
  This example will exclude "income" from the result

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]
```

```
for x in coll.find({},{ "income": 0 }):
  print(x)
```

4.11. Filter the Result

When finding documents in a collection, result can be filter by using a query object.

- **Example :**
  Find document(s) with the income 20000

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

query = { "income": "20000" }

doc = coll.find(query)

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

4.12. Sort the Result

The sort() method is used to sort the result in ascending or descending order.
The sort() method takes one parameter for "fieldname" and one parameter for "direction" (ascending is the default direction).

- **Example :**
  Sort the result alphabetically by name

```
import pymongo

import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

doc = coll.find().sort("name")

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

4.13. Sort Descending

The value -1 is used as the second parameter to sort descending.

sort("name", 1) #ascending
sort("name", -1) #descending

- **Example :**
  Sort the result reverse alphabetically by name:

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

doc = coll.find().sort("name", -1)

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

4.14. Delete Document

The delete_one() method is used to delete one document.

> **Note :** If the query finds more than one document, only the first occurrence is deleted.

- **Example :**
  Delete the document with the income "20000"

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

query = { "income": 20000 }

coll.delete_one(query)
for x in coll.find():
  print(x)
```

4.15. Delete All Documents in a Collection

Pass an empty query object to the delete_many() method to delete all documents in a collection.

- **Example :**
  Delete all documents in the "employee" collection

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
```

```
coll = db["employee"]

x = coll.delete_many({})

print(x.deleted_count, " documents deleted.")
```

4.16. Delete Collection

The drop() method is used to collection in MongoDB.

- **Example :**
  Delete the "employee" collection

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

coll.drop()
```

> **Note :** The drop() method returns true if the collection was dropped successfully, and false if the collection does not exist.

4.17. Update Collection

The update_one() method is used to update a record.

> **Note :** If the query finds more than one record, only the first occurrence is updated.

The second parameter is an object defining the new values of the document.

- **Example :**
  Change the income from "20000" to "30000"

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

query = { "income": 20000 }
newvalues = { "$set": { "income": 30000 } }

coll.update_one(query, newvalues)

for x in coll.find():
  print(x)
```

4.18. Limit the Result

The limit() method is used to limit the result in MongoDB.

- **Example :**
  Limit the result to only return 5 documents

```
import pymongo

client = pymongo.MongoClient("mongodb://localhost:27017/")
db = client["information"]
coll = db["employee"]

result = coll.find().limit(5)

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

5. **Python File Open**

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

5.1. File Handling

The open() function is the key function for working with files in Python and it takes two parameters; filename, and mode.

5.1.1. **File opening methods**

> - "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
> - "t" - Text - Default value. Text mode
> - "b" - Binary - Binary mode (e.g. images)

5.2. open() function

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

- **Example :**

```
f = open("<file name>", "r")
print(f.read())
```

5.3. File Methods

5.3.1. read()

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("<file name>", "r")
print(f.read(5))
```

5.3.2. readline()

- *You can return one line by using the readline() method*

  - **Example :**
    Read one line of the file

```
f = open("<file name>", "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("<file name>", "r")
print(f.readline())
print(f.readline())
```

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

  - **Example :**
    Loop through the file line by line

```
f = open("<file name>", "r")
for x in f:
  print(x)
```

5.3.3. Close()

Always close the file when you are done with it, in some cases, due to buffering, changes made to a file may not show until you close the file.

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

```
f = open("<file name>", "r")
print(f.readline())
f.close()
```

5.4. Write to an Existing File

you must add a parameter to the open() function to write to an existing file

> - "a" - Append - will append to the end of the file
> - "w" - Write - will overwrite any existing content

- **Example :**
  Open the file "" and append content to the file

```
f = open("<file name>", "a")
f.write("Now the file has more content!")
f.close()
```

- open and read the file after the appending

```
f = open("<file name>", "r")
print(f.read())
```

- **Example :**
  Open the file "" and overwrite the content

```
f = open("<file name>", "w")
f.write("content deleted by mistake!")
f.close()
```

- open and read the file after the appending

```
f = open("<file name>", "r")
print(f.read())
```

5.5. 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 ""

```
f = open("<newfile>", "x")
Result: a new empty file is created!
```

- **Example :**
  Create a new file if it does not exist

```
f = open("<newfile>", "w")
```

5.6. Delete file

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

- **Example :**
  Remove the file ""

```
import os
os.remove("<file name>")
```

5.6.1. 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("<file name>"):
  os.remove("<file name>")
else:
  print("The file does not exist")
```

5.7. Delete Folder

The os.rmdir() method is used to delete an entire folder but You can only remove empty folders

- **Example :**
  Remove the folder "myfolder"

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