Day 20: Database Connectivity & Regular Expressions in Python

1. Database Connectivity in Python

Python allows us to interact with databases using libraries like SQLite, MySQL, and PostgreSQL. In this lesson, we will cover how to connect to these databases, perform CRUD operations, and use best practices.

1.1 Introduction to Databases in Python

What is a Database?

A database is a structured collection of data that allows for efficient storage, retrieval, and management of information.

Why Use Databases in Python?

- Store and retrieve structured data efficiently.
- Perform operations on large datasets without loading them into memory.
- Securely manage data with transactions.

Types of Databases

1. **SQL Databases**: SQLite, MySQL, PostgreSQL (Structured data with tables, rows, and columns)
2. **NoSQL Databases**: MongoDB, Firebase (Unstructured or semi-structured data)

1.2 SQLite with Python

SQLite is a lightweight database built into Python. It is useful for small applications and quick prototyping.

1.2.1 Connecting to SQLite

```
import sqlite3


# Connect to database (or create if it doesn't exist)
conn = sqlite3.connect("students.db")


# Create a cursor object to interact with the database
cursor = conn.cursor()


print("Database connected successfully!")
```

✅ **Output**: Database connected successfully!

1.2.2 Creating a Table

```
cursor.execute('''CREATE TABLE IF NOT EXISTS students (
              id INTEGER PRIMARY KEY,
              name TEXT NOT NULL,
              age INTEGER NOT NULL
            )''')


conn.commit()
print("Table created successfully!")
```

✅ **Output**: Table created successfully!

1.2.3 Inserting Data

```
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Alice", 21))

cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Bob", 22))

conn.commit()

print("Data inserted successfully!")
```

✅ **Output**: Data inserted successfully!

1.2.4 Retrieving Data

```
cursor.execute("SELECT * FROM students")

rows = cursor.fetchall()


for row in rows:

    print(row)
```

✅ **Output**:

```
(1, 'Alice', 21)

(2, 'Bob', 22)
```

1.2.5 Updating Data

```
cursor.execute("UPDATE students SET age = ? WHERE name = ?", (23, "Alice"))

conn.commit()

print("Data updated successfully!")
```

✅ **Output**: Data updated successfully!

1.2.6 Deleting Data

```
cursor.execute("DELETE FROM students WHERE name = ?", ("Bob",))

conn.commit()

print("Data deleted successfully!")
```

✅ **Output**: Data deleted successfully!

1.2.7 Closing the Connection

```
conn.close()

print("Database connection closed!")
```

✅ **Output**: Database connection closed!

1.3 MySQL with Python

To connect to **MySQL**, install the connector first:

```
pip install mysql-connector-python
```

Connecting to MySQL

```
import mysql.connector

conn = mysql.connector.connect(

    host="localhost",

    user="root",

    password="your_password",

    database="test_db"

)

cursor = conn.cursor()

print("Connected to MySQL successfully!")
```

2. Regular Expressions (Regex) in Python

Regular Expressions (Regex) are patterns used to match and manipulate strings.

2.1 Common Regex Methods

Importing the re module:

```
import re
```

2.2 Using re.match()

Matches a pattern at the **beginning** of a string.

```
pattern = r"Hello"

text = "Hello World"

match = re.match(pattern, text)

if match:

   print("Match found:", match.group())

else:

   print("No match")
```

✅ **Output**: Match found: Hello

2.3 Using re.search()

Searches for a pattern **anywhere** in the string.

```
pattern = r"World"

text = "Hello World"

match = re.search(pattern, text)

if match:

   print("Pattern found at position:", match.start())
```

✅ **Output**: Pattern found at position: 6

2.4 Using re.findall()

Finds **all occurrences** of a pattern.

```
pattern = r"\d+"  # Find all numbers

text = "There are 3 cats, 4 dogs, and 10 birds."


matches = re.findall(pattern, text)

print("Numbers found:", matches)
```

✅ **Output**: Numbers found: ['3', '4', '10']

2.5 Using re.sub()

Replaces occurrences of a pattern.

```
text = "The color is blue."

updated_text = re.sub(r"blue", "red", text)


print(updated_text)
```

✅ **Output**: The color is red.

2.6 Validating an Email Address

```
email_pattern = r"^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$"

email = "example@email.com"

if re.match(email_pattern, email):

  print("Valid email address")

else:

  print("Invalid email address")
```

✅ **Output**: Valid email address

2.7 Extracting Phone Numbers

```
text = "Contact me at 9876543210 or 123-456-7890"

phone_numbers = re.findall(r"\d{10}|\d{3}-\d{3}-\d{4}", text)

print("Phone numbers found:", phone_numbers)
```

✅ **Output**: Phone numbers found: ['9876543210', '123-456-7890']

Recap of Key Concepts

Feature	**SQLite**	**Regex**
Purpose	Store structured data	Pattern matching in text
Library	sqlite3	re
Common Methods	execute(), fetchall(), commit()	match(), search(), findall(), sub()
Use Case	Data storage and retrieval	Text validation, data extraction

Conclusion

✅ **Database Connectivity**: We learned how to connect Python to SQLite and MySQL, create tables, and perform CRUD operations.

✅ **Regex in Python**: We explored pattern matching with match(), search(), findall(), and text manipulation.