Python

Unit – 5

Advanced Topics II: Regular Expressions

Regular Expressions (REs) and Python

Regular Expressions (commonly abbreviated as **REs**) are a powerful tool in Python used for matching patterns in text. They are essential for parsing strings, validating inputs, and scraping data from documents like HTML files. Python provides a built-in module called re to work with regular expressions.

📌 1. Introduction to Regular Expressions

Regular Expressions are patterns used to match combinations of characters in strings. These are especially useful in text processing, form validation, and data extraction.

Example Use Cases:

- Validate email addresses
- Find all phone numbers in a document
- Replace patterns like dates or prices
- Extract data from HTML pages

In Python, we use the re module to implement these operations.

🔧 Importing the Module:

```
import re
```

🧬 2. Sequence Characters in Regular Expressions

Sequence characters help in matching specific types of characters.

Pattern	Description	Example Match
\d	Matches any digit (0-9)	'5' in "abc5"
\D	Matches any non-digit character	'a' in "abc"
\w	Matches any alphanumeric char	'a', '1', '_'
\W	Matches any non-alphanumeric	'!', ' '
\s	Matches any whitespace character	space, tab, newline
\S	Matches non-whitespace characters	'a', '9'
.	Matches any character except \n	'a', '3', '@'

Example:

```
re.findall(r'\d+', "There are 24 apples and 8 bananas.")

# Output: ['24', '8']
```

🔁 3. Quantifiers in Regular Expressions

Quantifiers define how many times a character or group should occur.

Symbol	Description	Example
*	0 or more repetitions	a* matches aaa, a, or ""
+	1 or more repetitions	a+ matches a, aa, but not ""
?	0 or 1 repetition	a? matches a or ""
{n}	Exactly n repetitions	a{3} matches aaa
{n,}	At least n repetitions	a{2,} matches aa, aaa, aaaa
{n,m}	Between n and m repetitions	a{2,4} matches aa, aaa, aaaa

Example:

```
re.findall(r'a{2,4}', "caaaandy")  # Output: ['aaaa']
```

🧩 4. Special Characters in Regular Expressions

These characters have special meanings and must be escaped using a backslash \ when meant literally.

Character	Meaning
.	Any character except newline
^	Start of a string
$	End of a string
[]	Matches any one of the characters
`	`
()	Groups regex parts
\	Escape character

Example:

```
re.findall(r'[^a-z]', "abc123")  # Output: ['1', '2', '3']
```

📂 5. Using Regular Expressions on Files

You can use REs to process text files efficiently — for example, to extract specific data from logs or reports.

Example: Finding all emails in a file

```
import re

with open('sample.txt', 'r') as file:
    content = file.read()

emails = re.findall(r'\b[\w.-]+@[\w.-]+\.\w{2,4}\b', content)
print(emails)
```

🌐 6. Retrieving Information from an HTML File

HTML files often contain structured data. We can extract tags, attributes, or text using REs. However, **for complex HTML, it's better to use BeautifulSoup**.

Example: Extract all links (<a href="...">)

```
html = """
<html>
<body>
<a href="http://example.com">Example</a>
<a href="http://test.com">Test</a>
</body>
</html>
"""

links = re.findall(r'href="(.*?)"', html)
print(links)  # Output: ['http://example.com', 'http://test.com']
```

🧪 Case Study: Screen Scraping with Regular Expressions

Objective: Extract product names and prices from a simplified HTML snippet using REs.

Sample HTML:

```
<div class="product">

  <h2>iPhone 14</h2>

  <span class="price">$999</span>

</div>

<div class="product">

  <h2>Galaxy S22</h2>

  <span class="price">$899</span>

</div>
```

Python Code:

```
import re

html = """
<div class="product">
  <h2>iPhone 14</h2>
  <span class="price">₹999</span>
</div>
<div class="product">
  <h2>Galaxy S22</h2>
  <span class="price">₹899</span>
</div>
"""


products = re.findall(r'<h2>(.*?)</h2>\s*<span class="price">\$(.*?)</span>', html)

for name, price in products:
    print(f"Product: {name}, Price: ₹{price}")
```

Output:

```
Product: iPhone 14, Price: ₹999

Product: Galaxy S22, Price: ₹899
```

Python

Unit – 6

Python's Database Connectivity with MySQL

Python provides powerful libraries to connect and interact with databases like MySQL. This allows developers to create dynamic and data-driven applications efficiently.

We'll explore how to connect Python with MySQL, perform operations such as inserting, updating, deleting, and retrieving data, and even how to create database tables programmatically.

🛠️ 1. Verifying the MySQL Database Interface Installation

To connect Python with MySQL, we need a connector. The most commonly used one is:

```
mysql-connector-python
```

✅ Install MySQL Connector

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

✅ Verify Installation in Python

```
import mysql.connector


print("MySQL connector is working!")
```

If no error is shown, the installation is successful.

📥 2. Working with MySQL Database

Before performing operations, ensure that:

- MySQL is installed and running on your system.
- You have a database and user credentials ready.

✅ Connect to MySQL Database

```
import mysql.connector

connection = mysql.connector.connect(
    host="localhost",
    user="root",
    password="your_password",
    database="your_database"
)

if connection.is_connected():
    print("Successfully connected to MySQL database!")
```

🔄 3. Using MySQL from Python

After establishing a connection, use a **cursor** object to execute SQL queries.

```
cursor = connection.cursor()

cursor.execute("SELECT DATABASE();")

data = cursor.fetchone()

print("You're connected to:", data)
```

Use cursor.execute(sql_query) to run SQL statements.

📄 4. Retrieving All Rows from a Table

To fetch records from a table:

✅ Example: Fetch All Records

```
cursor = connection.cursor()

cursor.execute("SELECT * FROM students")

rows = cursor.fetchall()

for row in rows:

    print(row)
```

🔁 Fetch Methods:

- fetchone() – fetches the next row.
- fetchall() – fetches all rows.
- fetchmany(size) – fetches given number of rows.

➕ 5. Inserting Rows into a Table

To add new data into a table:

✅ Example:

```
cursor = connection.cursor()
sql = "INSERT INTO students (name, age, grade) VALUES (%s, %s, %s)"
values = ("Keyur", 21, "A")

cursor.execute(sql, values)
connection.commit()

print(cursor.rowcount, "record inserted.")
```

Note: Always use parameterized queries to avoid SQL injection.

❌ 6. Deleting Rows from a Table

To remove records based on a condition:

✅ **Example:**

```
cursor = connection.cursor()
sql = "DELETE FROM students WHERE name = %s"
val = ("Keyur",)

cursor.execute(sql, val)
connection.commit()

print(cursor.rowcount, "record(s) deleted.")
```

✏️ 7. Updating Rows in a Table

To modify existing data:

✅ **Example:**

```
cursor = connection.cursor()
sql = "UPDATE students SET grade = %s WHERE name = %s"
values = ("A+", "Keyur")

cursor.execute(sql, values)
connection.commit()

print(cursor.rowcount, "record(s) updated.")
```

🧱 8. Creating Database Tables through Python

You can create tables directly from your Python script.

✅ Example: Create students Table

```
cursor = connection.cursor()
cursor.execute("""
   CREATE TABLE IF NOT EXISTS students (
      id INT AUTO_INCREMENT PRIMARY KEY,
      name VARCHAR(100),
      age INT,
      grade VARCHAR(5)
   )
""")
print("Table created successfully.")
```

📚 Complete Example Program

```
import mysql.connector
# Connect
conn = mysql.connector.connect(
   host="localhost",
   user="root",
   password="your_password",
   database="school"
)
cursor = conn.cursor()
# Create Table
cursor.execute("""
CREATE TABLE IF NOT EXISTS students (
   id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100),
   age INT,
   grade VARCHAR(5)
)
""")

# Insert
sql = "INSERT INTO students (name, age, grade) VALUES (%s, %s, %s)"
values = ("Keyur", 20, "B")
cursor.execute(sql, values)
conn.commit()

# Read
cursor.execute("SELECT * FROM students")
for row in cursor.fetchall():
    print(row)

# Update
cursor.execute("UPDATE students SET grade='A' WHERE name='Keyur'")
conn.commit()

# Delete
cursor.execute("DELETE FROM students WHERE name='Keyur'")
conn.commit()

conn.close()
```

✅ Summary

Operation	Function/Method
Connect DB	mysql.connector.connect()
Create Cursor	connection.cursor()
Execute SQL	cursor.execute(query)
Fetch Data	fetchone(), fetchall()
Insert Data	INSERT INTO table_name ...
Update Data	UPDATE table_name SET ...
Delete Data	DELETE FROM table_name WHERE ...
Create Table	CREATE TABLE SQL with cursor.execute()