1. Explain what SQL injection attacks are and how to prevent them in a web application. Write a sample code snippet in Python that demonstrates how to avoid SQL injection attacks in a web application

PREREQUISITES:

- Python 3.x
- Flask framework (can be installed using pip)
- MySQL database

SETUP INSTRUCTIONS:

1. Install Python 3.x from the official Python website.
2. Install Flask framework by running the command "pip install flask" in your command prompt or terminal.
3. Install MySQL database on your system.
4. Install *mysql.connector* module by running the command " pip install mysql-connector-python " in yourcommand prompt or terminal.
5. Create a new database named "mydatabase" in MySQL.
6. Create a new user named "mydatabaseuser" with the password "mypassword" in MySQL and grant all privileges to the "mydatabase" database.
7. Copy the given code and paste it into a new Python file on your system.
8. Run the Python file in your command prompt or terminal by navigating to its directory and running the command "python filename.py".
9. Open your web browser and navigate to "http://localhost:5000/profile?user_id=<user_id>" to view the result of the program.

ANSWER:

SQL injection attacks are a type of web attack that exploits vulnerabilities in web applications to execute malicious SQL statements. An attacker can use SQL injection to access or modify sensitive data, execute unauthorized operations, or even take control of the underlying database server.

To prevent SQL injection attacks, a web application should use parameterized queries or prepared statements, which allow the application to separate user input from the SQL statement and avoid any potential injection attacks.

Here is an example implementation in Python using Flask and the SQL Alchemy ORM:

CODE:

```
from flask import Flask, request
import mysql.connector

# Initialize the Flask application
app = Flask(__name__)

# Connect to the MySQL database
try:
    conn = mysql.connector.connect(
        host="localhost",
        database="mydatabase",
        user="mydatabaseuser",
        password="mypassword"
    )
except mysql.connector.Error as err:
    print("Unable to connect to the database:", err)

# Create a cursor for executing database queries
cursor = conn.cursor()

# Define the route for displaying the user's profile
@app.route('/profile')
def profile():
    # Get the user ID from the query string
    user_id = request.args.get('user_id')

    # Check that the user ID is a number
    if not user_id.isdigit():
        return "Invalid user ID"

    # Use parameterized queries to execute the database query
    try:
        cursor.execute("SELECT * FROM users WHERE id=%s", (user_id,))
    except mysql.connector.Error as err:
        return "Unable to execute the database query:", err

    # Fetch the result of the database query
    result = cursor.fetchone()

    # Check if the result is None, which means that no user was found with
the given ID
    if result is None:
        return "User not found"

    # Return the result as a string
    return str(result)

# Start the Flask application
if __name__ == '__main__':
    app.run()
```

EXPLANATION:

This is a Flask application that connects to a MySQL database and defines a route for displaying a user's profile based on their ID.

The `mysql.connector` module is imported to allow communication with the database. The Flaskapplication is initialized with `Flask(__name__)`.

Then, the code attempts to connect to the database using the `mysql.connector.connect()` function with the database name, host, username, and password as arguments. If the connection is unsuccessful, an error message is printed.

A cursor is created for executing database queries. The cursor is created outside of the `profile` function so that it can be reused for multiple queries.

The `profile` function is defined with the route `@app.route('/profile')`. The user's ID is obtained from the query string using `request.args.get('user_id')`. The code checks that the user ID is a valid number using the `isdigit()` function.

The query to retrieve the user's information is executed using the cursor's `execute()` method with the user ID as a parameter. The `%s` in the query string is a placeholder that is replaced with the user ID in the second argument of `execute()`.

The result of the query is fetched using the cursor's `fetchone()` method. If the result is `None`, it means that no user was found with the given ID, so an error message is returned. If the result is not `None`, it is returned as a string.

Finally, the Flask application is started using `app.run()`.

PROGRAM OUTCOMES:

- This program sets up a connection to a MySQL database using the mysql.connector module in Python.
- It defines a route for displaying a user's profile by querying the "users" table in the "mydatabase" database using a parameterized query.
- It retrieves the user ID from the query string and checks that it is a valid number.
- It fetches the result of the database query and returns it as a string.
- If the user ID is invalid or the database query fails, the program returns an appropriate error message.