Explain the n-tier and MVC architecture of enterprise computing.

N-Tier Architecture: Divides an application into multiple tiers. It is also called multi-tier architecture.

3 Tier: Divides an application into 3 tiers:

- **Presentation Tier:** Handles user interface and interaction
- **Business Logic Tier:** Implements business rules and processes data
- **Data Tier:** Manages communication with the database and data storage

2 Tier: There is no Business Logic tier between client and server

1 Tier: Presentation, Business Logic, and Data tiers are all located on a single machine

Benefits of N-Tier Architecture

- **Improved modularity:** Promotes cleaner code structure and easier maintenance.
- **Scalability:** Allows individual layers to scale independently based on demand.
- **Security:** Enforces separation of concerns, enhancing data security and integrity.

MVC Architecture: Separates the application into three distinct parts:

- **Model:** Represents the data and business logic of the application.
- **View:** Responsible for displaying the user interface and user interaction.
- **Controller:** Handles user input, interacts with the model, and updates the view accordingly.

Benefits:

- **Cleaner Code:** Promotes cleaner code
- **Flexibility:** Allows independent development and changes to each component.
- **Maintainability:** Easier to modify the user interface without affecting the underlying business logic.

Write the code of a servlet that displays "GGSIP UNIVERSITY, DELHI" on a web browser in BLUE color, italics and center aligned.

```
import java.io.*;
import javax.servlet.*;
```

```
public class DisplayTextServlet extends HttpServlet {
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        out.println("<html>");
        out.println("<body style='color: blue; font-style: italic; text-align:
center;'>");
        out.println("<p>GGSIP UNIVERSITY, DELHI</p>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Define Scriptlet, Directives and Action of JSP.

1. **Scriptlet:** A scriptlet is a block of Java code embedded within `<%` and `%>` tags in a JSP page. It allows you to execute Java code directly in the HTML page. Example:

```
<%
   String message = "Hello, JSP!";
   out.println(message);
%>
```

2. **Directives:** Directives are special commands that provide global information about a JSP page.   Example:

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
import="java.util.*" %>
```

3. **Actions:** JSP actions are enclosed in `<jsp:...>` tag and are used to perform specific tasks within a JSP page. Examples:

```
<jsp:include page="header.jsp" />
<jsp:forward page="anotherPage.jsp" />
```

Using JDBC, write the code for a java program to fetch and display the details employee from the 'employee' table already stored in the database

```
import java.sql.*;

public class FetchEmployeeDetails {

  public static void main(String[] args) throws Exception {
    // Database connection information
    String url = "jdbc:mysql://localhost:3306/your_database";
    String username = "your_database_username";
    String password = "your_database_password";

    // Driver class name
    String driverClassName = "com.mysql.cj.jdbc.Driver";

    // Load the JDBC driver
    Class.forName(driverClassName);

    // Connect to the database
    Connection connection = DriverManager.getConnection(url, username,
password);

    // Define the SQL query
    String query = "SELECT * FROM employee";

    // Create a statement object
    Statement statement = connection.createStatement();

    // Execute the query and store the result set
    ResultSet resultSet = statement.executeQuery(query);

    // Display the employee details
    while (resultSet.next()) {
      int id = resultSet.getInt("id");
      String name = resultSet.getString("name");
      String email = resultSet.getString("email");
      System.out.printf("ID: %d, Name: %s, Email: %s\n", id, name, email);
    }

    // Close the resources
    resultSet.close();
    statement.close();
    connection.close();
  }
}
```

JAAS

JAAS (Java Authentication and Authorization Service) is a framework within the Java platform that provides pluggable APIs for authentication and authorization. It allows developers to implement custom authentication and authorization mechanisms into their Java applications independent of the underlying security implementation.

Components:

- **LoginModule:** Responsible for performing user authentication.
- **Subject:** Represents a user or service within the system

- **Principal:** Represents the identity of the user or service.
- **Credential:** Represents the user's authentication information like password or token.
- **Policy:** Defines the access control rules for different resources and actions.

JMS

JMS (Java Message Service) is an API that provides a standard way for Java applications to send and receive messages asynchronously. This enables loose coupling between applications, as they don't need to know each other's details to communicate.

Types of JMS Messages:

- **TextMessage:** Contains text data.
- **BytesMessage:** Contains byte data.
- **ObjectMessage:** Can contain any serializable Java object.
- **MapMessage:** Stores key-value pairs.

Benefits of JMS: Flexibility, Reliability, Scalability and Loose Coupling

Using HTML, design a Web Page for Retired Employees having fields like Employee ID, Employee Name, Employee Age, Last Designation held, Employee Address, Employee Phone number, Employee Email Id, Date of Retirement, Amount of Pension, etc. with appropriate JAVASCRIPT Validations.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Retired Employees Information</title>
</head>
<body>
    <h2>Retired Employees Information</h2>
    <form id="retiredEmployeeForm" onsubmit="return validateForm()">
        <label for="employeeId">Employee ID:</label>
        <input type="text" id="employeeId" name="employeeId" required>

        <label for="employeeName">Employee Name:</label>
        <input type="text" id="employeeName" name="employeeName" required>

        <label for="employeeAge">Employee Age:</label>
        <input type="number" id="employeeAge" name="employeeAge" required>

        <label for="designation">Last Designation held:</label>
        <input type="text" id="designation" name="designation" required>
```

```html
        <label for="employeeAddress">Employee Address:</label>
        <input type="text" id="employeeAddress" name="employeeAddress" required>

        <label for="employeeEmail">Employee Email Id:</label>
        <input type="email" id="employeeEmail" name="employeeEmail" required>

        <label for="dateOfRetirement">Date of Retirement:</label>
        <input type="date" id="dateOfRetirement" name="dateOfRetirement" required>

        <label for="pensionAmount">Amount of Pension:</label>
        <input type="number" id="pensionAmount" name="pensionAmount" required>

        <button type="submit">Submit</button>
    </form>

    <script>
        function validateForm() {
            // JavaScript validation
            var age = document.getElementById("employeeAge").value;
            if (age < 18 || age > 100) {
                alert("Please enter a valid age between 18 and 100.");
                return false;
            }
            return true;
        }
    </script>
</body>
</html>
```