HTML Tags: Anchor, Form, Table, Image, List Tags, Paragraph, Break, Label

Q1.Introduction to HTML and its structure.

Ans:-

HTML (Hyper Text Markup Language) is the standard language used to create webpages. It structures content using tags.

```
<!DOCTYPE html>
<html>
<head>
        <title>Page Title</title>
</head>
<body>
        <h1>Heading</h1>
        <p>Paragraph text.</p>
</body>
</html>
```

Q2.Explanation of key tags:

Ans:-

1. <a> – Anchor Tag

Definition: Used to create hyperlinks that link to other pages or websites.

- **Example**:

```
<a href="https://google.com">Google</a>
```

---

2. <form> **– Form Tag**

- **Definition**: Used to create a form for user input like text fields, checkboxes, etc.
- **Example**:

```
<form action="/submit" method="post">
  <input type="text" name="username">
  <input type="submit" value="Submit">
</form>
```

---

3. <table> – Table Tag

- **Definition**: Used to display data in rows and columns.
- **Example**:

```
<table border="1">
```

```
 <tr>
  <th>Name</th>
  <th>Age</th>
</tr>
 <tr>
<td>Alice</td>
<td>25</td>
</tr>
</table>
```

---

4. <img> – Image Tag

- **Definition**: Used to display an image on the web page.
- **Example**:<img src="photo.jpg" alt="My Photo">

---

5. List Tags: **<ul>**, **<ol>**, **<li>**

- **Definition**:
  - : Unordered list (● bullet points)
  - : Ordered list (1, 2, 3...)
  - : List item

**6. ** – Paragraph Tag

- **Definition**: Used to define a paragraph of text.
- **Example**: This is a paragraph

**7.
** – Line Break Tag

Definition: This is a paragraph

- **Example**:

  Line 1
Line 2

---

8. <label> – Label Tag

- **Definition**: Used to define a label for an input element in a form. Helps improve accessibility.
- **Example**:

```
<label for="email">Email:</label>

<input type="email" id="email" name="email">
```

CSS: Inline CSS, Internal CSS, External CSS

Q1.Overview of CSS and its importance in web design.

Ans:-

- CSS (Cascading Style Sheets) is used to style HTML elements—like colors, fonts, layout, and spacing.
- CSS (Cascading Style Sheets) is essential in web design for controlling the visual layout and style of web pages, ensuring consistency, responsiveness, and an enhanced user experience.

Q2.Types of CSS:

Ans:-

There are **three main types of CSS** used to style HTML documents.

1. Inline CSS

- **Definition**: CSS styles are written directly inside the HTML element using the style attribute.
- **Usage**: For quick styling of a single element.
- **Example**:

```
<p style="color: blue; font-size: 16px;">This is a blue paragraph.</p>
```

2. Internal CSS

- **Definition**: CSS rules are written inside a <style> tag within the <head> section of the HTML document.
- **Usage**: When styling a single HTML page.
- **Example**:

```
html
CopyEdit
<html>
<head>
 <style>
  p {
   color: green;
   font-size: 18px;
  }
 </style>
</head>
```

```
<body>
  <p>This paragraph is green.</p>
</body>
</html>
```

3. External CSS

- **Definition**: CSS is written in a separate .css file and linked to the HTML file using the <link> tag.
- **Usage**: Best for styling multiple web pages using the same stylesheet.
- **Example**:

HTML File (index.html):

```
html
<head>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <p>This is styled by external CSS.</p>
</body>
```

CSS File (style.css):

```
css
p {
  color: red;
  font-size: 20px;
}
```

CSS: Margin and Padding

Q1.Definition and difference between margin and padding.

Ans:-

Margin:

- The space **outside** the element's border.
- It creates space **between** the element and other elements.

Padding:

- The space **inside** the element's border.
- It creates space **between** the element's content and its border.

```
<div style="margin: 20px; padding: 10px; background-color: lightblue;">
  This box has margin and padding.
</div>
```

Q2.How margins create space outside the element and padding creates space inside.

Ans:-

Padding – Space Inside the Element

- Padding adds space **inside the box**, between the content and the border.
- It pushes the content **inward**.

Example:

```
<div style="padding: 20px; background-color: lightblue;">
  Hello!
</div>
```

Margin – Space Outside the Element

- Margin adds space **outside the box**, between this element and any other element.
- It pushes the whole box **away** from others.

Example:

```
<div style="margin: 30px; background-color: lightgreen;"> I'm pushed away from others!

</div>
```

CSS: Pseudo-Class

Q1.Introduction to CSS pseudo-classes like :hover, :focus, :active, etc.

Ans:- A **pseudo-class** is a keyword added to a selector that lets you style an element **when it is in a specific state**, like when the mouse is over it, when it's clicked, or when it's focused.

1. :hover

- **Definition**: Applies styles when the user hovers (places mouse) over an element.
- **Example**:

```
button:hover
{
  background-color: yellow;
}
```

2. :focus

- **Definition**: Applies styles when an element (like a text box) gets focus (e.g., clicked or tabbed into).
- **Example**:

```
input:focus {
  border: 2px solid blue;
}
```

3. :active

- **Definition**: Applies styles when an element is being **clicked** (during the active state).
- **Example**:

```
a:active {
  color: red;
}
```

Q2.Use of pseudo-classes to style elements based on their state.

Ans:-

4. :visited – When a link has already been clicked

- **Use**: Style visited links differently.
- **Example**:

```
a:visited
{
  color: purple;
}
```

5. :checked – When a checkbox or radio button is selected

- **Use**: Style label or input when checkbox is checked.
- **Example**:

```
input:checked + label
 {
  color: green;
  font-weight: bold;
}
```

CSS: ID and Class Selector

Q1.Difference between id and class in CSS.

Ans:-

ID (#id)

- **Definition :** The id is used to identify a **single unique element** on the page.

- It must be **unique** within the page, meaning no two elements should have the same id.
- It is targeted with the # symbol in CSS.
- **Example:**

```
<div id="header">This is a header</div>
```

Class (.class)

- **Definition**: The class is used to style **multiple elements** that share the same class.
- It can be reused on **many elements** on the page.
- It is targeted with the . symbol in CSS.
- **Example**:

```
<div class="card">This is a card</div>
<div class="card">This is another card</div>
```

Q2.Usage scenarios for id (unique) and class (reusable).

Ans:-

1. **Usage of** id **(Unique)**

- The **id** attribute is used for **unique** elements, meaning it should only appear **once** per page.
- It's great for targeting a **single element** that requires a specific style or JavaScript functionality.

2. **Usage of** class **(Reusable)**

- The `class` attribute is used for **styling multiple elements** that share the same characteristics.
- It can be reused on many elements, making it perfect for grouping elements that need similar styles.

Introduction to Client-Server Architecture

Q1.Overview of client-server architecture.

Ans:-

Client-server architecture is a model where clients (users or devices) request services or resources, and servers provide those services.

Key Components:

- Client: Sends requests (e.g., browser requesting a webpage).
- Server: Receives requests, processes them, and sends back responses.

How it works:

1. Client initiates a request.
2. Server processes the request.
3. Server sends a response back to the client.

Examples:

- Web browsers (clients) communicating with web servers.
- Mobile apps accessing cloud services.

Advantages:

- Centralized management
- Scalability
- Easy maintenance

Q2.Difference between client-side and server-side processing.

Ans:-

1. Client-Side Processing

Definition:
Client-side processing refers to operations or tasks that are performed on the user's device (usually in the browser).

The client is typically a web browser running on the user's computer or mobile device.

Key Characteristics:

- **Executed on the client** (browser).
- **Faster interaction** because no server request is needed for UI updates (except for asynchronous operations like AJAX).
- **Languages Used**: JavaScript, HTML, CSS (mainly).
- **Advantages**:
  - Faster feedback to users (e.g., instant form validation, animations).
  - Reduces load on the server since tasks are handled locally.
- **Disadvantages**:
  - **Security risks**: Code is visible to users, so sensitive logic should not be handled client-side.
  - Dependent on the **user's browser** and device.

- **Animations and Interactions**: Using JavaScript for interactive elements (e.g., dropdown menus, image sliders).
- **AJAX Requests**: Fetching data without refreshing the page, like updating a news feed or chat.

2. Server-Side Processing

Definition:
Server-side processing refers to operations or tasks that are performed on the web server before sending the response to the client. The server processes the request, runs business logic, and generates the appropriate response.

Key Characteristics:

- **Executed on the server**.
- **Slower interaction** because each action might require a round trip to the server (e.g., submitting a form and waiting for a response).
- **Languages Used**: PHP, Python, Ruby, Java, C#, Node.js, etc.
- **Advantages**:
  - **Security**: Sensitive data processing and logic remain hidden from the user.
  - **Powerful**: Can handle complex operations (like database interactions) that the client-side cannot.
- **Disadvantages**:
  - Server load increases with heavy processing or large-scale traffic.
  - **Slower user interaction** compared to client-side processing, since it requires network round trips.

Q3.Roles of a client, server, and communication protocols.

Ans:-

1. Client

- **Definition**: The **client** is the device or application used by the **user** to make requests and view content.
- **Role**: It **requests** data or services from the server and **displays** the response to the user.
- **Example**: When you open a website on your **web browser**, that browser acts as the **client**.

---

2. Server

- **Definition**: The **server** is a computer or system that **provides** data or services when requested by the client.
- **Role**: It **receives** requests from the client, processes them, and **sends back** the information or resources.
- **Example**: When you visit a website, the **server** stores the website and sends it to your browser (client).

---

3. Communication Protocols

- **Definition**: Communication protocols are the **set of rules** that the client and server use to exchange information.
- **Role**: These protocols ensure that both the client and server understand each other and exchange data correctly.
- **Example**: **HTTP** (HyperText Transfer Protocol) is a common protocol that the browser (client) and server use to communicate when loading websites.

Q1.Introduction to the HTTP protocol and its role in web communication.

Ans:-

- **HTTP (HyperText Transfer Protocol)** is the foundation of data communication on the **World Wide Web**.
- **HTTP (HyperText Transfer Protocol)** is a set of rules used by **web browsers** and **web servers** to **communicate** with each other.
- It is the foundation of data exchange on the **internet**.

Key Features:

- **Stateless**: Each request is treated separately; the server doesn't remember anything from before.
- **Text-based**: Easy to read and understand.
- **Request-Response System**:
  - The **client (browser)** sends a request.
  - The **server** sends back a response.

Common HTTP Methods

- **GET** – Used to **retrieve** data from the server (like loading a web page).
- **POST** – Used to **submit** data to the server (like sending a form).
- **PUT** – Used to **update** existing data on the server.
- **DELETE** – Used to **remove** data from the server.

How HTTP Works in Web Communication

1. **User types a URL** in the browser or **clicks a link** on a webpage.
2. The **browser sends an HTTP request** to the web server asking for the content.
3. The **server receives the request**, processes it, and sends back an **HTTP response** (like an HTML page).
4. The **browser gets the response** and **displays the content** to the user.

Q2.Explanation of HTTP request and response headers.

Ans:-

HTTP headers **are extra pieces of information sent along with an HTTP request or response.**
They help the client (like a browser or Java program) **and the** server **understand how to process the data.**

- Request Headers**: Sent from the client to the server (e.g., browser type, accepted data).**
- Response Headers**: Sent from the server to the client (e.g., content type, server info, cookies).**

J2EE Architecture Overview

Q1.Introduction to J2EE and its multi-tier architecture.

Ans:-

Multi-Tier Architecture in J2EE
J2EE applications typically follow a multi-tier architecture, dividing the application into layers, each with a specific role. This helps in modularity, maintainability, and scalability.

1. Client Tier

- Role: Interacts with the end user.
- Examples: Web browser, mobile app, desktop client.
- Technology: HTML, JavaScript, JSP, or JavaFX.

2. Web Tier

- Role: Handles client requests and manages presentation logic.
- Components: Servlets, JSP (JavaServer Pages).
- Function: Receives HTTP requests, processes them, and sends responses.

3. Business Tier

- Role: Contains the core business logic.
- Components: EJB (Enterprise JavaBeans), POJOs (Plain Old Java Objects), or Spring Beans.
- Function: Handles processing, calculations, and decision-making.

4. EIS (Enterprise Information System) Tier

- Role: Manages communication with external systems or databases.
- Components: JDBC, JPA, JMS, or connectors to ERP/legacy systems.
- **Function**: Responsible for data storage, retrieval, and integration with other enterprise resources.

2)Role of web containers, application servers, and database servers.

Ans:-

Roles of Web Containers, Application Servers, and Database Servers

In a multi-tier enterprise application (like those built using J2EE), different types of servers handle different responsibilities to ensure scalability, modularity, and performance.

1. Web Container (Servlet Container)

- **Role**: Manages and runs web components like Servlets and JSPs.
- **Function**:
  - Handles HTTP requests from clients.
  - Manages session tracking, request routing, and response generation.
  - Ensures the servlet lifecycle (init, service, destroy).
- **Examples**: Apache Tomcat, Jetty.

2. Application Server

- **Role**: Provides a full runtime environment for enterprise applications, including business logic.
- **Function**:
  - Hosts and manages EJBs, transaction handling, security, messaging, and web services.
  - Coordinates between the web tier and the backend systems.
- **Examples**: JBoss (WildFly), GlassFish, WebLogic, WebSphere.

3. Database Server

- **Role**: Stores and manages persistent data.
- **Function**:
  - Executes SQL queries, handles transactions, indexes, data storage, and retrieval.
  - Works with JDBC or JPA APIs for communication with the application server.
- **Examples**: MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

Web Component Development in Java (CGI Programming)

Q1.Introduction to CGI (Common Gateway Interface).

Ans:-

- CGI (Common Gateway Interface) is a standard protocol used to enable web servers to interact with external programs (such as scripts or applications) to dynamically generate web content.
- It acts as a bridge between the web server and the backend applications, enabling the server to send and receive information from these programs.

2)Process, advantages, and disadvantages of CGI programming.

Ans:-

Process of CGI Programming

1. **Client Request**: A client (browser) sends a request to the web server (e.g., a form submission).

2. **Web Server Calls CGI Script:** The server invokes the appropriate CGI program to handle the request.
3. **CGI Execution:** The script processes the request, interacts with databases if needed, and generates dynamic content (e.g., HTML).
4. **Response Sent Back:** The server sends the generated content back to the client.

Advantages of CGI Programming

- **Language Agnostic:** Supports multiple programming languages (e.g., Perl, Python, C).
- **Simple to Implement:** Easy to set up for small-scale or simple dynamic applications.
- **Flexibility:** Can handle various dynamic tasks like form processing, database querying, etc.
- **Compatibility:** Works with most web servers and platforms.

Disadvantages of CGI Programming

- **Performance Issues:** Spawns a new process for each request, which can slow down performance, especially with high traffic.
- **Resource Intensive:** High resource consumption due to process creation for every request.
- **Concurrency Problems:** Difficult to handle multiple simultaneous requests efficiently.
- **Limited Scalability:** Not well-suited for large, high-traffic applications.

Servlet Programming: Introduction, Advantages, and Disadvantages

1)Introduction to servlets and how they work.

Ans:-

- Servlet is a server side technology , which is used to handle the client request , process the request , and generate the dynamic response.

How Servlets Work:

1. Client sends request to the server (e.g., via a web browser).
2. The web server forwards the request to the Servlet Container.
3. The Servlet processes the request (e.g., reading form data, interacting with databases) and generates a response.
4. The response is sent back to the client through the web server.

Key Points:

- Efficient: Servlets run in the server's process, avoiding the overhead of creating new processes for each request.
- Platform-independent: Java-based, so they can run on any server with Java support.
- Scalable: Can handle multiple concurrent requests efficiently.

2)Advantages and disadvantages compared to other web technologies.

Ans:-

1. **Efficiency:** Servlets run within the server's process, offering better performance than CGI.
2. **Scalability:** Can handle multiple requests concurrently without creating new processes, unlike CGI.
3. **Platform Independence:** Java-based, so they work across any platform supporting Java.
4. **Session Management:** Supports session management, ideal for applications like e-commerce.
5. **Integration:** Easily integrates with other Java EE technologies like EJB, JSP, and JDBC.

Servlet Versions, Types of Servlets

Q1.History of servlet versions.

Ans:-

1. **Servlet 1.0 (1997):**
   - First version, introduced the concept of using Java to handle HTTP requests and generate dynamic content.
2. **Servlet 2.0 (1999):**
   - Added session management and request/response filters, improving flexibility.
3. **Servlet 2.3 (2001):**
   - Introduced **filters**, **listeners**, and better JSP integration.
4. **Servlet 2.4 (2003):**
   - Added **annotations** and enhanced configuration support via **web.xml**.
5. **Servlet 3.0 (2009):**
   - Introduced **asynchronous processing**, **annotations**, and **multipart file upload**.
6. **Servlet 3.1 (2013):**
   - Improved **asynchronous processing** and introduced better **lifecycle management**.
7. **Servlet 4.0 (2017):**
   - Added support for **HTTP/2** to improve web performance.
8. **Servlet 5.0 (2020):**
   - Transitioned to **Jakarta EE**, modernizing the API for cloud-native and modern web applications.

Q2.Types of servlets: Generic and HTTP servlets.

1. **Generic Servlets:**
   - **Purpose:** General-purpose servlets that handle different protocols, not just HTTP.
   - **Extends:**javax.servlet.GenericServlet.
   - **Use Case:** Used for custom protocols like FTP or SMTP.
2. **HTTP Servlets:**
   - **Purpose:** Specialized for handling **HTTP requests** in web applications.
   - **Extends:**javax.servlet.http.HttpServlet.

- **Use Case:** Used for typical web tasks like form submissions, user authentication, and content rendering.

Difference between HTTP Servlet and Generic Servlet

Q1.Detailed comparison between HttpServlet and GenericServlet.
Ans:-

Definition:

- **GenericServlet**: A protocol-independent servlet that handles any type of request.
- **HttpServlet**: A subclass of GenericServlet designed specifically to handle HTTP requests.

Usage:

- **GenericServlet**: Used for custom protocols (rarely used in web development).
- **HttpServlet**: Used for handling HTTP requests in web applications (common in web development).

Method Handling:

- **GenericServlet**: You override the service() method to handle requests.
- **HttpServlet**: You override doGet(), doPost(), etc., to handle specific HTTP requests.

Protocol Support:

- **GenericServlet**: Does not have built-in support for HTTP features.
- **HttpServlet**: Fully supports HTTP methods, headers, sessions, and cookies.

Inheritance:

- **GenericServlet**: Implements Servlet interface directly.
- **HttpServlet**: Extends GenericServlet to support HTTP.

Example Use:

- **GenericServlet**: Used for non-HTTP protocols.
- **HttpServlet**: Used for HTTP-based web applications.

Servlet Life Cycle

Q1.Explanation of the servlet life cycle: init(), service(), and destroy() methods

Ans:-

1) Loading & Instantiation

- When server is started , servlet class is loaded in the memory and servlet object is created.

2) Init() [Intialization]

- Server object will be initialized by invoking init() method.

3)Request Handling :- service()

- It will handle or serve the client request.
- In this phase service() method invoked.

4)Destroy()

- When the server shutdown , destroy() method will be executed and servlet object will be deleted.

Creating Servlets and Servlet Entry in web.xml

1) How to create servlets and configure them using web.xml.

Ans:-

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

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

      PrintWriter out = response.getWriter();

      out.println("<html><body><h1>Hello, this is my first servlet!
</h1>
</body></html>");
  }
```

2. **Create a** web.xml **File (Servlet Configuration)**

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                  http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
      version="3.0">

  <!-- Servlet Configuration -->
```

```
  <servlet>
    <servlet-name>MyFirstServlet</servlet-name>
    <servlet-class>MyFirstServlet</servlet-class>
  </servlet>

  <!-- Servlet Mapping -->
  <servlet-mapping>
    <servlet-name>MyFirstServlet</servlet-name>
    <url-pattern>/hello</url-pattern> <!-- The URL pattern for this servlet -->
  </servlet-mapping>

</web-app>
```

Logical URL and ServletConfig Interface

Q1.Explanation of logical URLs and their use in servlets.

Ans:-

Logical URL in Servlets (Short Explanation)

- A logical URL is the URL pattern you define in web.xml (or via annotations) to access a servlet.
- It doesn't have to match the servlet class name or file path.

Q2.Overview of ServletConfig and its methods.

Ans:-

ServletConfig is an interface that allows a servlet to access initialization parameters defined in web.xml.

- getInitParameter(String name)
- getInitParameterNames()
- getServletName()
- getServletContext()

RequestDispatcher Interface: Forward and Include Methods

Q1.Explanation of RequestDispatcher and the forward() and include() methods.

Ans:-

- RequestDispatcher is used to forward a request to another resource (like a servlet, JSP, or HTML) on the server or include its content in the response

- forward(request, response)
- include(request, response)

ServletContext Interface and Web Application Listener

Q1.Introduction to ServletContext and its scope.

Ans:-

- ServletContext is an interface that allows servlets to share data and access web app-wide resources.

Scope of ServletContext

- Application-wide scope (lives until the app is undeployed or server stops).
- Used to:
    - Share data between servlets
    - Access app-level parameters
    - Read files/resources from the webapp directory

Q2.How to use web application listeners for lifecycle events.

Ans:-

Listeners in a web app are used to monitor and respond to lifecycle events (like app start, session creation, etc.).

- ServletContextListener
- HttpSessionListener
- ServletRequestListener