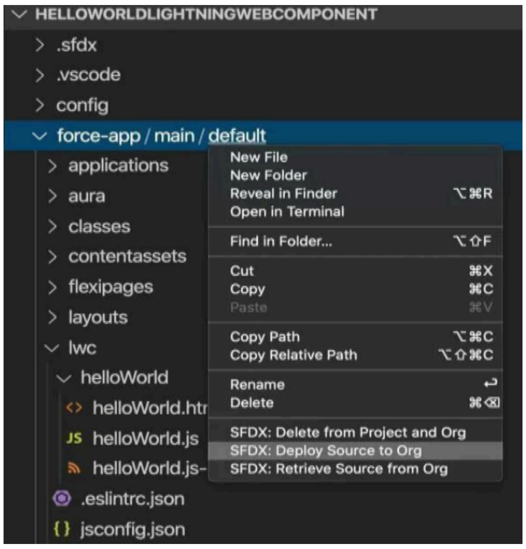
1. a) Write the steps involved in Deploying Lightning Web Component Files.
   b) Briefly write about Lightning Design System Styles with an example.

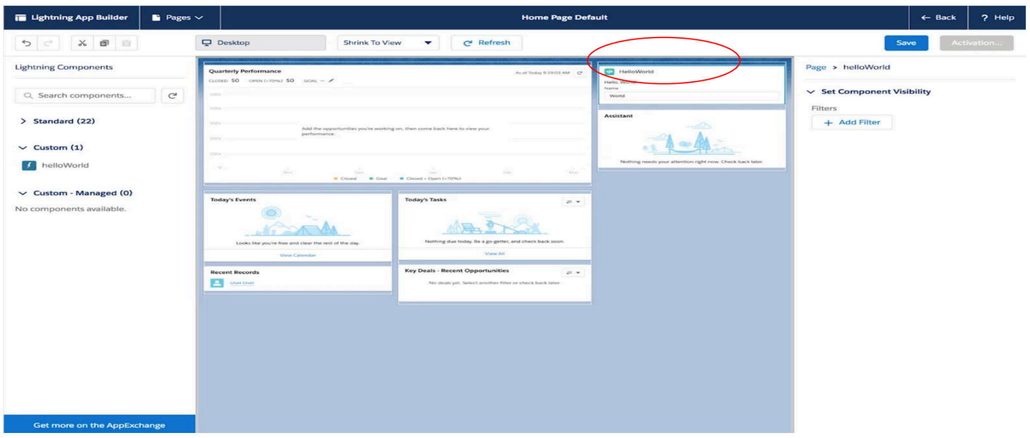
## (a) Steps Involved in Deploying Lightning Web Component Files

1. Create a Project:
   - Use Salesforce CLI or Visual Studio Code to create a SFDX project for your components.
2. Authorize Your Salesforce Org:
   - Open the Command Palette in Visual Studio Code (`Ctrl+Shift+P`).
   - Run `SFDX: Authorize an Org` to log in to your Salesforce org.
3. Create the Component:
   - Run `SFDX: Create Lightning Web Component`.
   - Specify the name and default directory (e.g., `force-app/main/default/lwc`).
4. Develop the Component:
   - Write the required files (`.html`, `.js`, and optional `.css`).
   - Ensure metadata is included in the `-meta.xml` file to expose the component.
5. Deploy the Component:
   - Right-click the default folder under `force-app/main/default` and select `SFDX: Deploy Source to Org`.



   - Verify successful deployment in the Output tab.
6. Add the Component to the App:
   - Log in to the Salesforce Lightning Experience.

- Use the App Builder to drag and drop your component onto a page.



- Save and activate the page.

Start → Create Project → Authorize Org → Create Component → Write Files → Deploy Files → Add to App → End.

---

## (b) Lightning Design System Styles

The **Salesforce Lightning Design System (SLDS)** provides a set of styles and guidelines for consistent UI design across Salesforce applications.

**Features:**

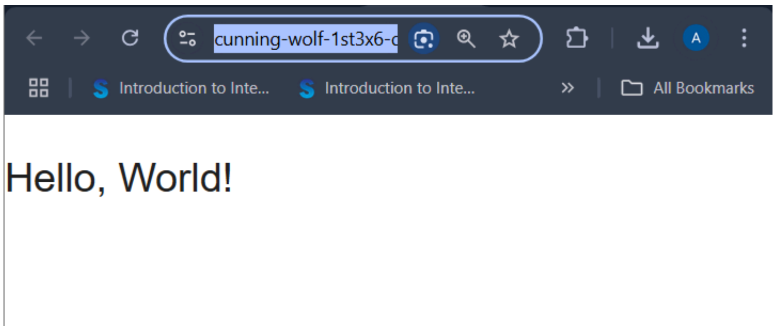
- Predefined styles for buttons, forms, cards, grids, and more.
- Supports responsive design.
- Encourages clean, minimalistic layouts.

### Usage: Example: Styling with SLDS

**Applying SLDS Classes:** Use SLDS classes in your HTML to style elements. For example:

```
<div class="slds-box slds-theme_shade">
    <p class="slds-text-heading_medium">Hello, World!</p>
</div>
```

This snippet applies a shaded box with medium-sized heading text.



Hello, World!

**Custom Styling with CSS:** You can complement SLDS by adding component-specific styles in the `.css` file:

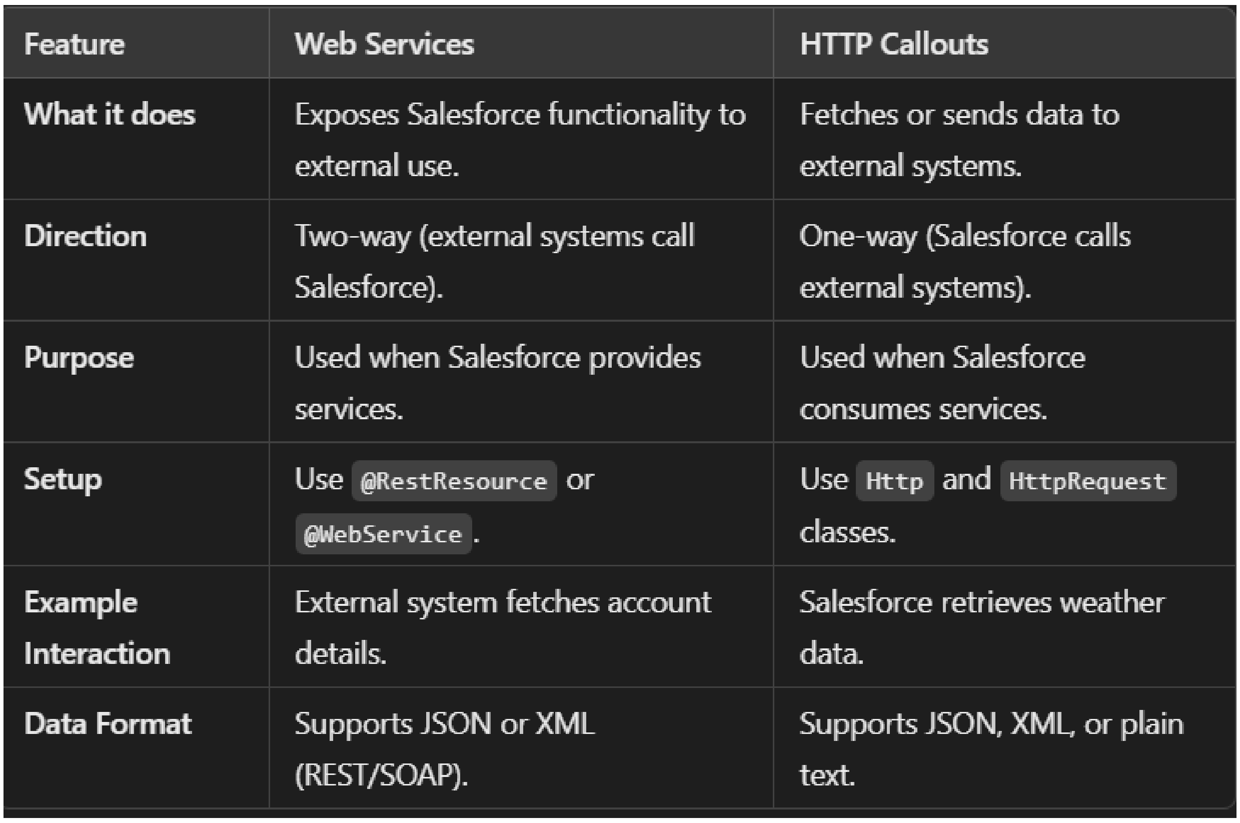
```
.custom-text {
    color: red;
    font-size: 18px;
}
```

Use it in your component's HTML:

```
<p class="custom-text">Custom Red Text</p>
```

---

**2. Describe the difference between web services and HTTP callouts.**



| Feature | Web Services | HTTP Callouts |
|---|---|---|
| **What it does** | Exposes Salesforce functionality to external use. | Fetches or sends data to external systems. |
| **Direction** | Two-way (external systems call Salesforce). | One-way (Salesforce calls external systems). |
| **Purpose** | Used when Salesforce provides services. | Used when Salesforce consumes services. |
| **Setup** | Use `@RestResource` or `@WebService`. | Use `Http` and `HttpRequest` classes. |
| **Example Interaction** | External system fetches account details. | Salesforce retrieves weather data. |
| **Data Format** | Supports JSON or XML (REST/SOAP). | Supports JSON, XML, or plain text. |

Web Services (Salesforce as a Service Provider) example code:

- External system calls Salesforce's API to fetch account details

```
@RestResource(urlMapping='/AccountService/*')
global with sharing class AccountService {
    @HttpGet
    global static Account getAccount() {
        return [SELECT Id, Name FROM Account LIMIT 1];
    }
}
```

HTTP Callouts (Salesforce as a Client) example code:

- Salesforce fetches weather data from an external API.

```
Http http = new Http();
HttpRequest request = new HttpRequest();
request.setEndpoint('https://api.weather.com/v3/weather');
request.setMethod('GET');
HttpResponse response = http.send(request);
System.debug(response.getBody());
```

3. Contact thisContact = new Contact(
FirstName = 'Brian',
LastName = 'Dent',
Phone = '(619) 852-4569',
Department = 'Mission Control',
Title = 'Mission Specialist - Neptune',
Email = 'briandent@trailhead.com');
insert thisContact;

Write step by step process to get the contact's first name and last name. Use the System.debug method to write LastName, FirstName to the Debug log.

Step-by-Step process to retrieve the contact's first name and last name and write them to the Debug log:

---

## Step 1: Insert the Contact

The provided code creates and inserts a new contact into the Salesforce database.

```
Contact thisContact = new Contact(
    FirstName = 'Brian',
    LastName = 'Dent',
    Phone = '(619) 852-4569',
    Department = 'Mission Control',
    Title = 'Mission Specialist - Neptune',
    Email = 'briandent@trailhead.com'
);
insert thisContact;
```

---

## Step 2: Query the Contact

Retrieve the contact you just inserted using SOQL (Salesforce Object Query Language). Use a filter condition, such as the email, to ensure you query the correct contact.

```
Contact retrievedContact = [SELECT FirstName, LastName FROM Contact
WHERE Email = 'briandent@trailhead.com'];
```

---

## Step 3: Debug the Contact Details

Use the `System.debug` method to log the contact's first and last name.

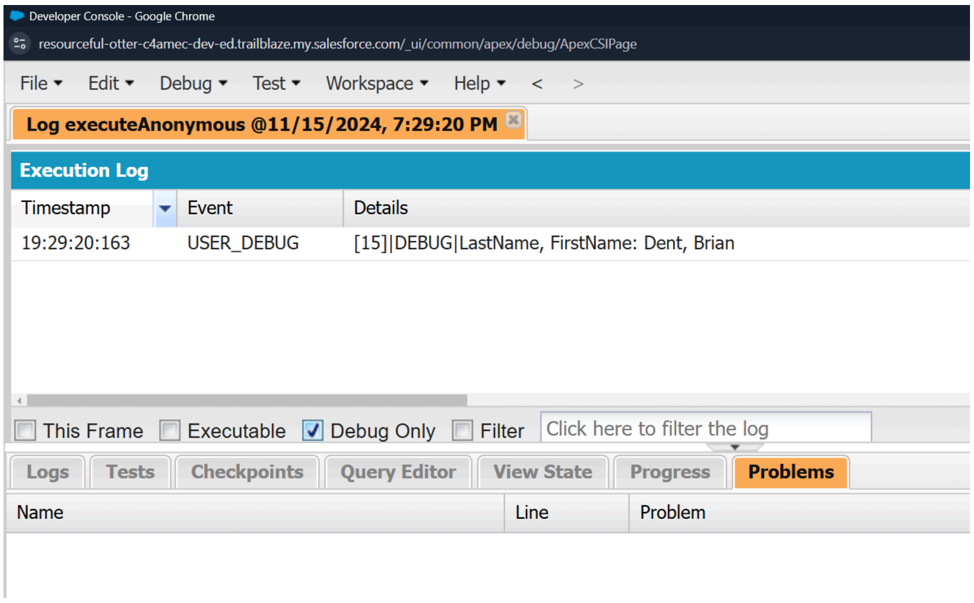
```
System.debug('LastName, FirstName: ' + retrievedContact.LastName +
', ' + retrievedContact.FirstName);
```

---

## Complete Code Example

```
Contact thisContact = new Contact(
    FirstName = 'Brian',
    LastName = 'Dent',
    Phone = '(619) 852-4569',
    Department = 'Mission Control',
    Title = 'Mission Specialist - Neptune',
    Email = 'briandent@trailhead.com'
);
insert thisContact;

Contact retrievedContact = [SELECT FirstName, LastName FROM Contact
WHERE Email = 'briandent@trailhead.com' LIMIT 1];
System.debug('LastName, FirstName: ' + retrievedContact.LastName +
', ' + retrievedContact.FirstName);
```

---


Developer Console - Google Chrome
resourceful-otter-c4amec-dev-ed.trailblaze.my.salesforce.com/_ui/common/apex/debug/ApexCSIPage
File
Edit
Debug
Test
Workspace
Help
Log executeAnonymous @11/15/2024, 7:29:20 PM
Execution Log
This Frame
Executable
Debug Only
Filter
Click here to filter the log
Logs
Tests
Checkpoints
Query Editor
View State
Progress
Problems
Name
Line
Problem

| Timestamp | Event | Details |
|---|---|---|
| 19:29:20:163 | USER_DEBUG | [15]\|DEBUG\|LastName, FirstName: Dent, Brian |

1. **Insert the Contact:** Adds a new record to the database.
2. **Query the Contact:** Retrieves the contact record using SOQL with a WHERE condition for precision.

3. **Debug the Output:** Prints the concatenated last name and first name in the format `LastName, FirstName`.

**4. Describe the function of checkpoints in the Developer Console. Why are they important when debugging complex code?**

### Functions of Checkpoints in the Developer Console

1. **Capture Variable State:**
    - Checkpoints record the values of variables at a specific point during code execution, helping to identify unexpected behavior.
2. **Inspect Object Properties:**
    - Allows developers to view the detailed state of sObjects and other custom objects, including their fields and relationships.
3. **Analyze Heap Usage:**
    - Provides insights into memory usage during execution, enabling optimization and preventing limit errors.
4. **Debug Control Flow:**
    - Validates the logical flow of the code by pausing at checkpoints to confirm that execution proceeds as expected.
5. **Identify Errors:**
    - Helps in pinpointing issues like null pointer exceptions or incorrect calculations by inspecting data at critical points.

### Importance of Checkpoints for Debugging Complex Code

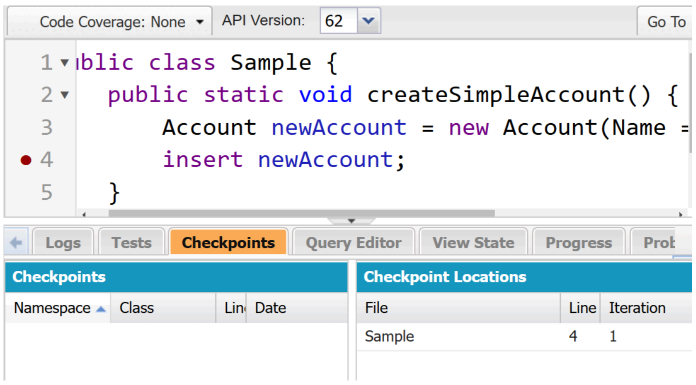
1. **Real-time Variable Inspection:** Checkpoints display variable values at a specific execution point, helping to identify errors in logic.
2. **Object State Analysis:** Developers can inspect the properties of sObjects and custom objects during code execution.
3. **Heap Usage Insights:** They allow you to monitor memory usage to identify and fix performance issues.
4. **Flow Verification:** Ensure the control flow of your code matches expectations.
5. **Efficiency:** Save time by focusing on problematic areas without running the entire code repeatedly.

### Example: Debugging with Checkpoints

```
public class Sample {

    public static void createSimpleAccount() {

        Account newAccount = new Account(Name = 'Test Account');

        insert newAccount;

    }

}
```

## Debug Process:

1. **Set a Checkpoint** at the `insert newAccount;` line.



Code Coverage: None | API Version: 62 | Go To

```
public class Sample {
    public static void createSimpleAccount() {
        Account newAccount = new Account(Name =
●       insert newAccount;
    }
```

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Prob

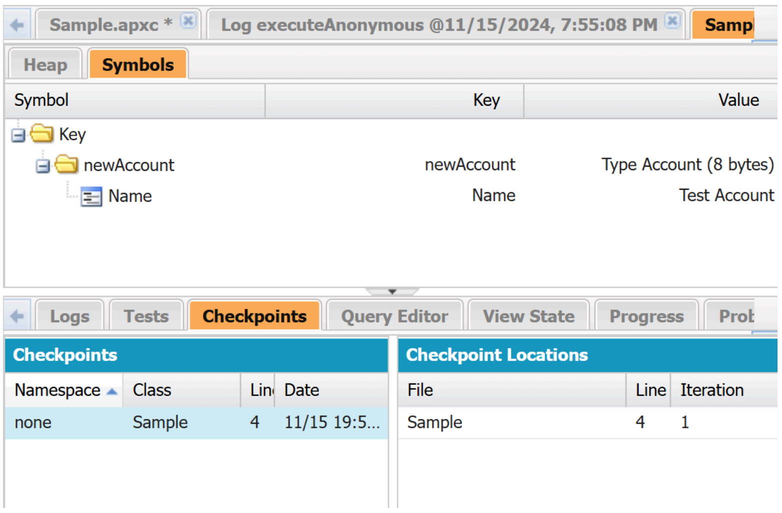
**Checkpoints**

| Namespace | Class | Line | Date |
|---|---|---|---|

**Checkpoint Locations**

| File | Line | Iteration |
|---|---|---|
| Sample | 4 | 1 |

2. **Run the Code** in **Debug > Open Execute Anonymous Window:**
   `Sample.createSimpleAccount();`
3. **Inspect the Checkpoint Data:**
   - View `newAccount` to ensure it has the correct `Name` value.
   - Check for memory usage to ensure there are no issues with the heap.



Sample.apxc * | Log executeAnonymous @11/15/2024, 7:55:08 PM | Samp

Heap | Symbols

| Symbol | Key | Value |
|---|---|---|
| Key | | |
| newAccount | newAccount | Type Account (8 bytes) |
| Name | Name | Test Account |

Logs | Tests | Checkpoints | Query Editor | View State | Progress | Prob

**Checkpoints**

| Namespace | Class | Line | Date |
|---|---|---|---|
| none | Sample | 4 | 11/15 19:5... |

**Checkpoint Locations**

| File | Line | Iteration |
|---|---|---|
| Sample | 4 | 1 |

5. Explain SOAP and REST with examples.

## SOAP (Simple Object Access Protocol)

- **Protocol** for exchanging structured information in web services.
- **Uses XML** for message format and relies on **HTTP/HTTPS** for communication.
- SOAP can support both stateful and stateless operations.
- **Strict** standards, with rules for message format, security, and error handling.
- Often used in enterprise systems due to its high security and reliability.

**Example:** A request to get user details from a server:

```
<soapenv:Envelope
xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
                  xmlns:web="http://www.example.com/webservice">
   <soapenv:Body>
      <web:GetUserDetails>
         <web:UserId>12345</web:UserId>
      </web:GetUserDetails>
   </soapenv:Body>
</soapenv:Envelope>
```

---

## REST (Representational State Transfer)

- **Architecture style** for designing networked applications, not a protocol.
- Uses **HTTP** for communication and supports different message formats like **JSON**, **XML**, and others.
- **Stateless** and lightweight, making it faster and easier to use for web applications.
- **Methods**: It uses HTTP methods like:
  - **GET** (retrieve data)
  - **POST** (create data)
  - **PUT** (update data)

    - **DELETE** (remove data)
- Common in modern web services for mobile apps and web applications.

**Example:** A GET request to retrieve user details:

```
GET /users/12345
Host: www.example.com
```

Response in JSON format:

```
{
  "userId": 12345,
  "name": "John Doe"
}
```

**6. Why do we use Lightning Web Components in Salesforce? Write a program that displays Hello World in an input field.**

### Why Use Lightning Web Components (LWC) in Salesforce?

Lightning Web Components (LWC) are the modern framework for building user interfaces in Salesforce. They provide several advantages over older technologies (like Visualforce and Aura Components). They are:

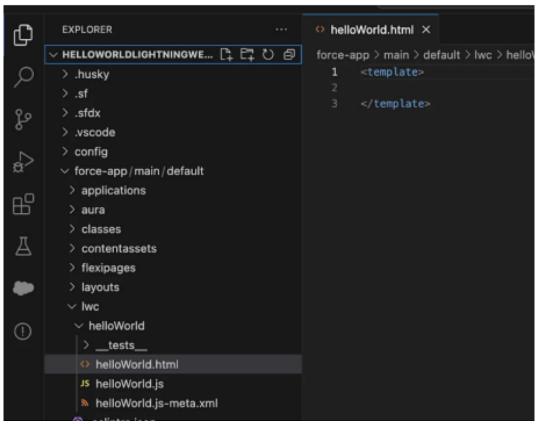
1. **Faster Performance**: LWC uses modern JavaScript and runs natively in the browser, making it faster and more efficient than Aura.
2. **Standardized Web Standards**: LWC is built using Web Components standards, which ensures compatibility with other web frameworks.
3. **Better Development Experience**: LWC has a cleaner syntax that simplifies development and debugging.
4. **Reusability**: Components in LWC are modular, so you can reuse them across multiple pages or applications.
5. **Better Integration**: LWC components integrate seamlessly with the Salesforce platform, using the Lightning Data Service and Apex to handle data manipulation.

---

### Hello World Program in Lightning Web Component

**1. Create the LWC:**

1. Open **VS Code** (with Salesforce Extensions).
2. **Create the Component** In your **SFDX Project:**
   a. Run `SFDX: Create Lightning Web Component` in our command palette.
   b. Specify the name "`helloWorld`" and the default directory (e.g., `force-app/main/default/lwc`).



```
<template>

</template>
```

This will generate three files:

- `helloWorld.html`
- `helloWorld.js`
- `helloWorld.css` (optional)

**2. Write the Code:**

1. **helloWorld.html**: defines the structure of the component.

```
<template>
    <lightning-input
            label="Enter Text" value={mssg} onchange={handleChange}>
      </lightning-input>
    <p>
      Hello, {mssg}!
    </p>
</template>
```

2. **helloWorld.js**: logic file that handles the data and events.

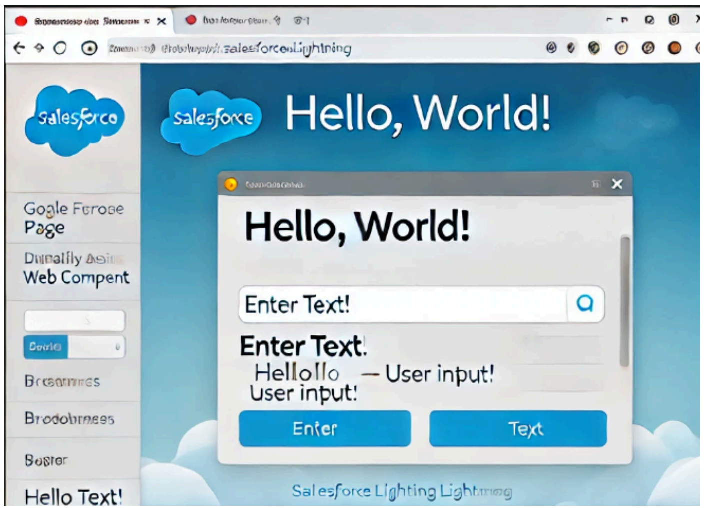
```
import { LightningElement } from 'lwc';
```

```
export default class HelloWorld extends LightningElement {
    mssg = 'World';
    handleChange(event) {
        this.mssg = event.target.value;
    }
}
```

3. **helloWorld.css** (optional): This is for styling your component. You can leave it empty or add some basic styles:

3. Deploy and Test:

1. Deploy the component to Salesforce using VS Code.
2. In Salesforce, go to **App Builder** and add your `helloWorld` component to a Lightning page (e.g., on the Home page).
3. When you view the page, you will see an input field where you can type. The text "Hello, World!" will update as you type into the input field.



## 7. Explain Handle Events in Lightning Web Components (LWC). Give examples.

## Handling Events in Lightning Web Components (LWC)

In Lightning Web Components (LWC), events are used to enable communication between components or to handle user interactions. Events play a vital role in dynamic web applications, allowing components to respond to actions such as button clicks, form submissions, or other interactions.

---

## Types of Events in LWC

1. Standard browser events like `click`, `change`, `mouseover`, etc.
2. Custom events are used to communicate between components in LWC, especially from child to parent components.

---

## Steps to Handle Events in LWC

1. **Listen for Events**: Use standard event attributes in HTML (e.g., `onclick`, `onchange`) or JavaScript event listeners (`addEventListener`) to capture user interactions.
2. **Define Event Handlers**: Create a JavaScript method in the component class to handle the event logic.
3. **Emit Events** (for Custom Events): Use the `CustomEvent` constructor to dispatch events for parent-to-child or child-to-parent communication.

---

## Example 1: Handling a Button Click Event

(**`buttonClick.html`**) : `onclick` attribute binds the button click to the `handleClick` method.

```
<template>
    <lightning-button label="Click Me"
onclick={handleClick}></lightning-button>
    <p>{message}</p>
</template>
```

(**`buttonClick.js`**) : updates the `message` property, which automatically re-renders the UI.

```
import { LightningElement } from 'lwc';
export default class ButtonClick extends LightningElement {
    message = 'Hello, World!';
    handleButtonClick() {
        this.message = 'Button was clicked!';
    }
}
```

---

## Example 2: Using Custom Events (Child-to-Parent Communication)

Child Component: `sendMessage` method creates and dispatches a custom event.

HTML File (**`childComponent.html`**):

```
<template>
    <lightning-button label="Send Message"
onclick={sendMessage}></lightning-button>
</template>
```

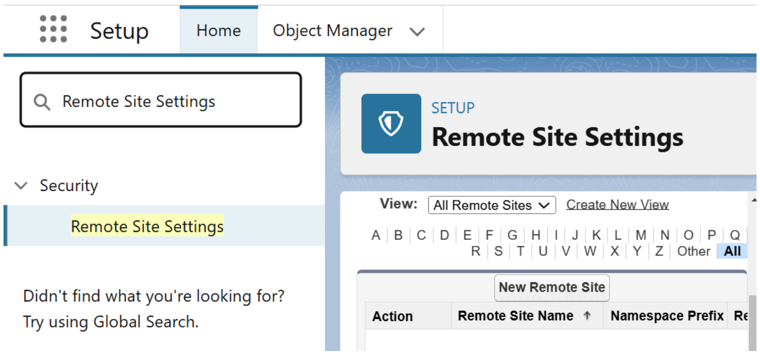
JavaScript File (**`childComponent.js`**): `oncustommessage` attribute listens for the `custommessage`

```
import { LightningElement } from 'lwc';
export default class ChildComponent extends LightningElement {
    sendMessage() {
        const message = 'Hello from Child!';
        const event = new CustomEvent('custommessage', {
            detail: message
        });
        this.dispatchEvent(event);
    }
}
```

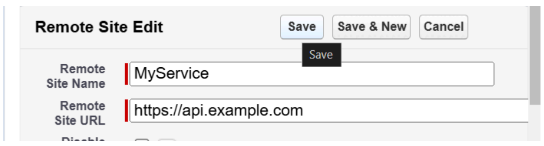
## 8. Write steps to add remote sites to the remote site settings to allow callouts to external sites. Explain REST with examples

### Steps to Add Remote Sites to Remote Site Settings in Salesforce

1. Login to Salesforce
2. Access Remote Site Settings:
    - Click the **Gear Icon** (Setup) in the upper-right corner.
    - In the **Quick Find** box, type **Remote Site Settings** and select it.

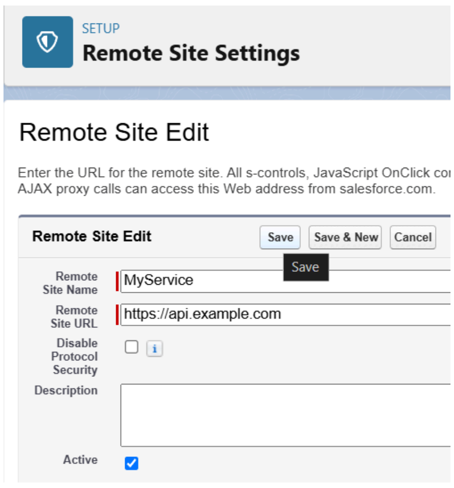


3. Create a New Remote Site:
    - Click on the **New Remote Site** button.
4. Provide Details:
    - **Remote Site Name**: Enter a unique name for the external site (e.g., `MyService`).
    - **Remote Site URL**: Enter the base URL for the external service you want Salesforce to call (e.g., `https://api.example.com`).



Remote Site Name: MyService
Remote Site URL: https://api.example.com

5. Activate the Site:
    - Check the **Active** box to enable the callout functionality.
6. Save:
    - Click the **Save** button to store your configuration.


SETUP
Remote Site Settings

Remote Site Edit

Enter the URL for the remote site. All s-controls, JavaScript OnClick co
AJAX proxy calls can access this Web address from salesforce.com.

Remote Site Edit

| | |
|---|---|
| Remote Site Name | MyService |
| Remote Site URL | https://api.example.com |
| Disable Protocol Security | ☐ |
| Description | |
| Active | ☑ |

This setup ensures that Salesforce

---

## Understanding REST API

**REST (Representational State Transfer)** is a widely used architectural style for designing APIs, enabling communication between systems using standard HTTP methods.

### Core Concepts

1. **HTTP Methods**:
   - **GET**: Fetches data from the server.
   - **POST**: Sends new data to the server.
   - **PUT**: Updates existing data.
   - **DELETE**: Removes data.
2. **Endpoints**:
   - Defined URLs for accessing specific resources (e.g., `/products` or `/users`).
3. **Stateless Communication**:
   - Each request is independent; no session data is retained on the server.
4. **Data Formats**:
   - Typically uses JSON or XML for exchanging data.

### Example: **Retrieve User Data**

Request:

```
GET https://jsonplaceholder.typicode.com/users
```

Response:

```
[
  { "id": 1,
     "name": "Leanne Graham",
     "email": "leanne@example.com" }
]
```

9. Explain in detail about LWC Module, Lifecycle Hooks and Decorators. Support with a program.

10. What is typecasting and Apex Callout with examples?

## Typecasting in Apex

Typecasting is the process of converting a variable from one data type to another in Apex. It can be used to convert between related classes, primitive data types, or objects of different data types.

### Types of Typecasting in Apex

1. **Upcasting (Implicit Casting):**
   - Assigning a child class object to a parent class reference.
   - Automatically handled by Apex.
2. **Downcasting (Explicit Casting):**
   - Assigning a parent class reference back to a child class object.
   - Requires explicit casting using parentheses.

**Examples of Typecasting:**

```
Upcasting
Account acc = new Account(Name = 'Test Account');
SObject sObj = acc;  // Implicit casting from Account to SObject
Downcasting
SObject sObj2 = new Account(Name = 'Test Account');
```

```
Account acc2 = (Account) sObj2;  // Explicit casting from SObject to
Account
```

---

### Apex Callout

An **Apex Callout** is a way for Salesforce to make an external HTTP or SOAP request to a third-party service or API. Callouts are used to fetch or send data to external systems.

- Requires setting up a **remote site setting** to allow the endpoint.
- Must be invoked asynchronously in a trigger context (e.g., using `@future` or queueable classes).
- The `HttpRequest` and `HttpResponse` classes are used for making HTTP callouts.

**Example of an HTTP Callout:**

### Apex Callout Example

```
public class SimpleCallout {
    public static String makeCallout() {
        Http http = new Http();
        HttpRequest request = new HttpRequest();
request.setEndpoint('https://jsonplaceholder.typicode.com/posts/1');
        request.setMethod('GET');
        HttpResponse response = http.send(request);
        return response.getBody();
    }
}
```

**Steps to Enable a Callout:**

1. **Set Remote Site Settings:**
   - Go to **Setup > Remote Site Settings > New Remote Site**.
   - Add the domain or URL of the external service (e.g., `https://api.example.com`).
2. **Test the Callout:**

- Call the `makeCallout` method from the Developer Console to see the results.