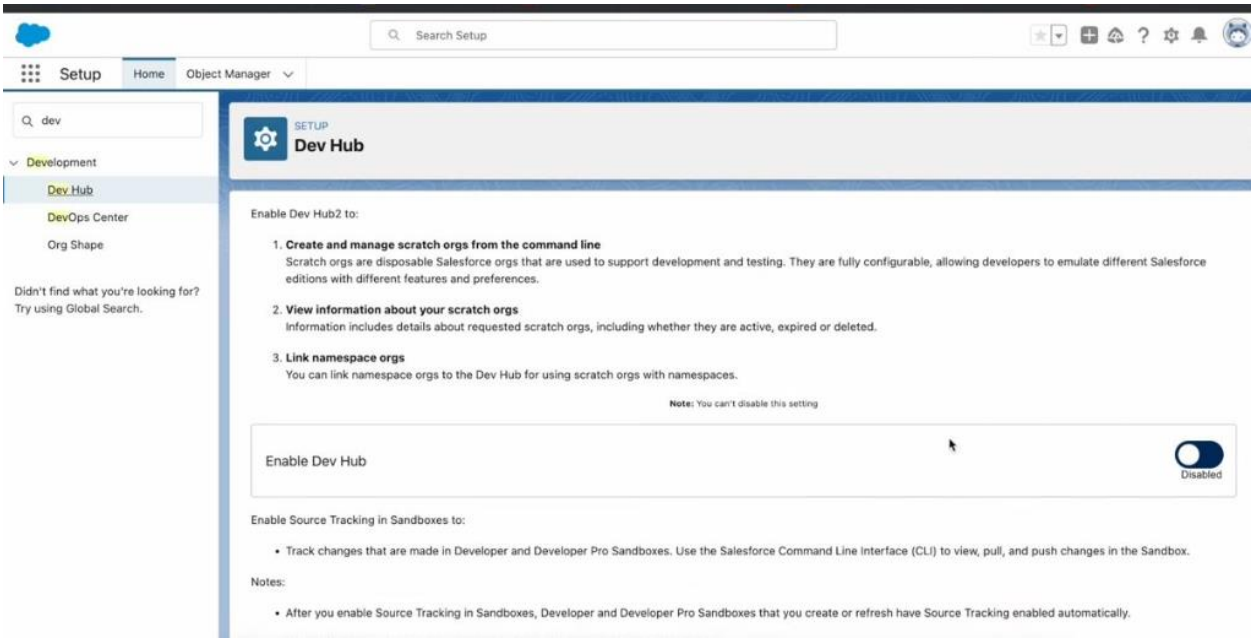
Crack the Code: 100 Interview Questions on Lightning Web Components!

What is the use of Dev Hub?

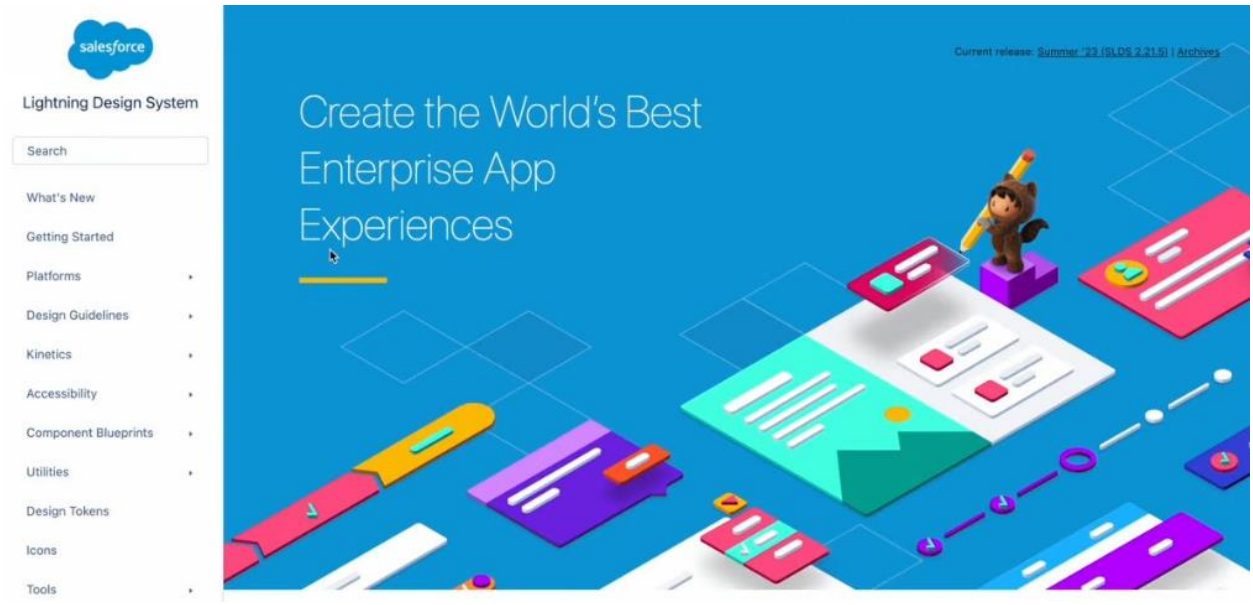
Enabling the dev Hub in your org allows for the creation of multiple scratch orgs.



What is SLDS?

SLDS is Salesforce Lightning Design System. The Developer Library and Lightning Web Component are essential resources for those working on code and design in the Salesforce platform.

Crack the Code: 100 Interview Questions on Lightning Web Components!



Explain Data Binding in LWC?

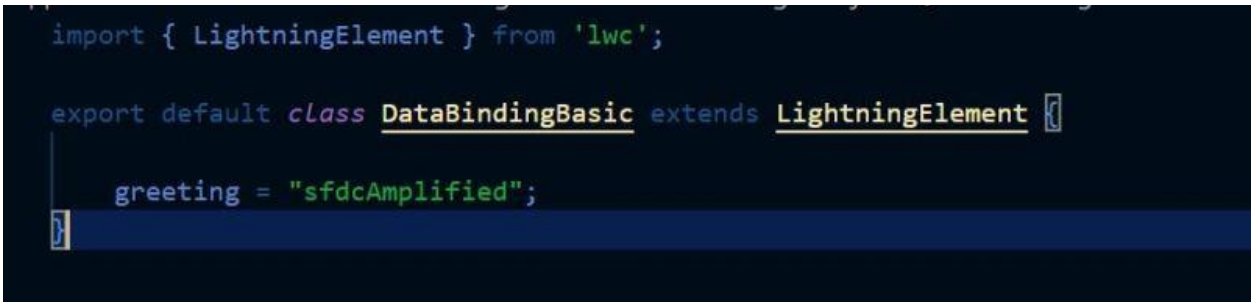
LWC is designed with a one-way data binding approach by default. To

understanding Data Binding we need to understand property and attributes

When we declare a **value** in JavaScript file, we say it as property.

in the below image *greeting* is the property which holds text sfdcamplified.

🌟 Crack the Code: 100 Interview Questions on Lightning Web Components!



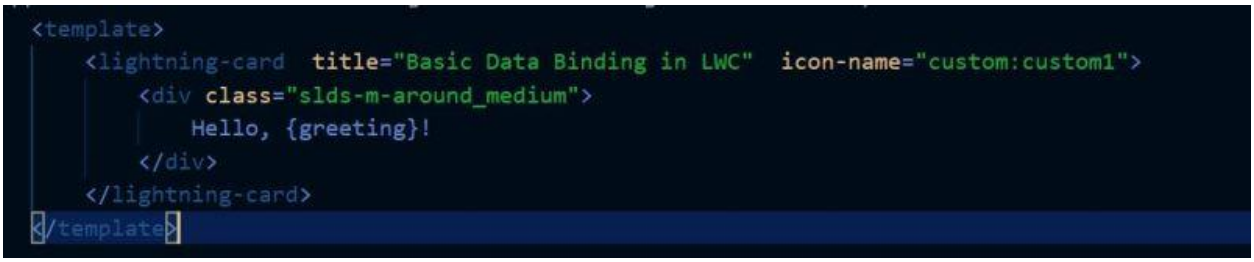
```
import { LightningElement } from 'lwc';

export default class DataBindingBasic extends LightningElement {

    greeting = "sfdcAmplified";
}
```

To access any JavaScript property (in this case – greeting) in template, surround that property with curly braces - {property}

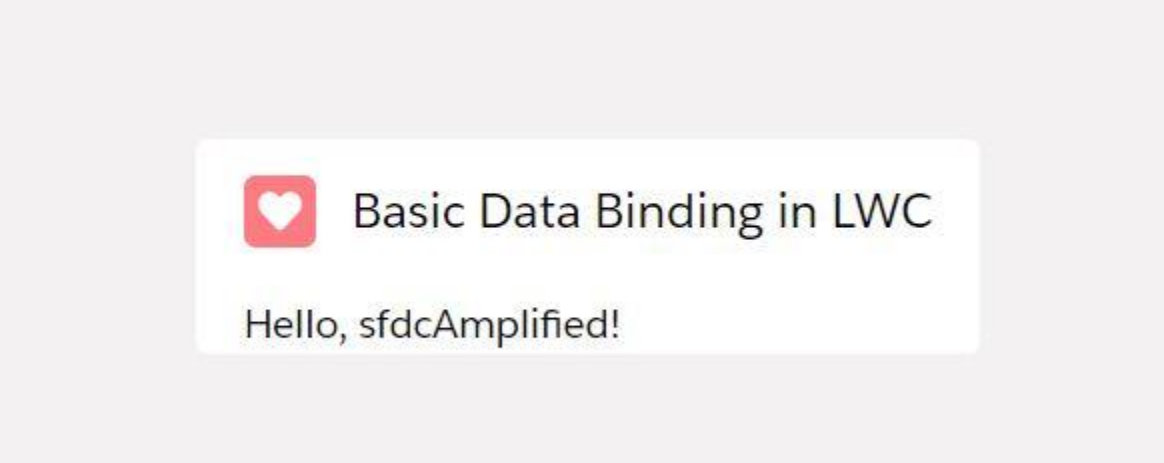
When we use property in HTML, we call it as attribute.



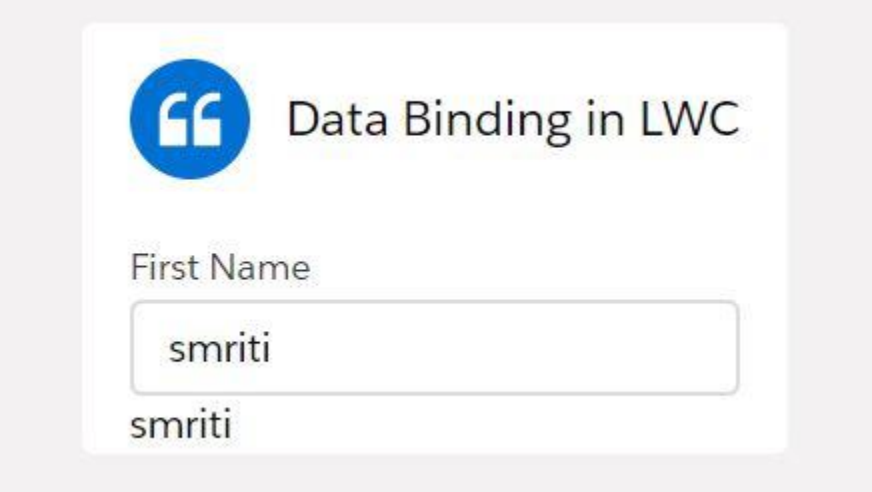
```
<template>
    <lightning-card  title="Basic Data Binding in LWC"  icon-name="custom:custom1">
        <div class="slds-m-around_medium">
            Hello, {greeting}!
        </div>
    </lightning-card>
</template>
```

Output

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



If we want based on user input automatically UI gets updated, then how will we do it in LWC?



✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

the answer is Event Listener - we must fire an event and handle that event to update the code in your controller.

HTML File

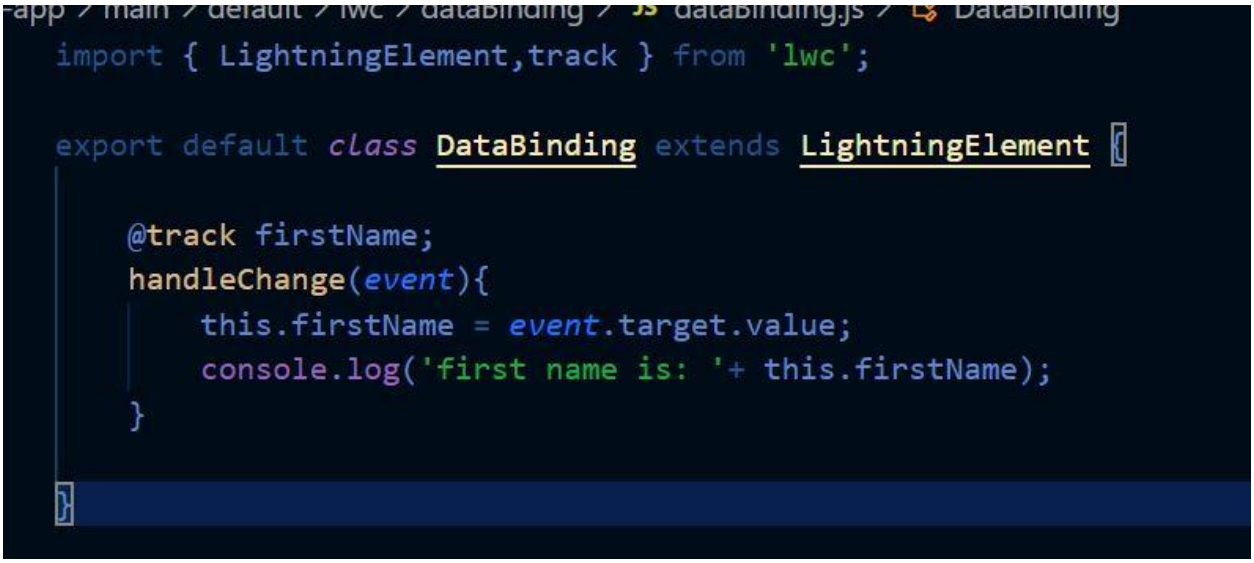
We have defined the function **handleChange(event){}** in the lightning input

```
<template>
    <lightning-card title="Data Binding in LWC"  icon-name="action:script">

     <div class="slds-m-around_medium">
    <lightning-input label="First Name" value={firstName} onchange={handleChange}>
    </lightning-input>
    {firstName}
    </div>
</lightning-card>
</template>
```

JavaScript File - we are handling **onchange HTML event** in the javascript with the function, hence we have written **(event)** in brackets. Basically whenever action happens it dispatches an event.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



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

export default class DataBinding extends LightningElement {

    @track firstName;
    handleChange(event){
        this.firstName = event.target.value;
        console.log('first name is: '+ this.firstName);
    }

}
```

What is the meaning of this keyword?

Eg - this.firstName = event.target.value

If property is inside the class, then use this. It specifies the current context.

this.firstName : The variable **name** which we have defined, we are just recalling it in the function to assign its changed value.

event.target: targets element where change is fired

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

event.target.value : This means from the onchange event I want the targeted value

Meaning**,** whatever is there in value take that.

What is Conditional Rendering?

If you want to hide some of the components from the HTML and show it based on conditions, then use conditional rendering.

How to iterate over multiple items in LWC?

To iterate over list we can use two directives

for:each

Iterator

Whenever we use for:each or Iterator we need to use key directive on the element on which we are doing iteration. Key gives unique id to each item.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Remember, without key we cannot do iteration.When a list changes, the

framework uses the key to rerender only the item that changed.

How can you render multiple templates in LWC?

In LWC, we can display multiple templates conditionally based on the component's state using the **if:true** directive.

Here's an example of how to display multiple templates in LWC:

```
<template>
    <template if:true={showTemplate1}>
        <p>This is template 1</p>
    </template>
    <template if:true={showTemplate2}>
        <p>This is template 2</p>
    </template>
</template>
```

In this example, the **if:true** directive is used to conditionally render each template based on the
component **'showTemplate1'** and **'showTemplate2'** properties. When the **'showTemplate1'** property is true, the first template is rendered, and when the **'showTemplate2'** property is true, the second template is rendered.

Explain Component Communication In LWC?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

lightning web component can be nested and there are 4 possibilities for communication between components.

- Parent to child communication
- Child to parent communication
- Communication between two separate components.
- Lightning Web Component to aura component

What is a decorator?

Decorator is a part of ECMA script and is used to add extra functionality in your function or methods.

- These Decorators dynamically change the functionality of property or function.
- They are identified with the symbol '@' as prefixed before a method or a variable name.
- Whenever we want to use any decorator, we must need to explicitly import it from '*lwc*'

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

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

There are some built-in decorators provided by LWC that we can use to define properties and methods in components:

- **@api:** This annotation is used to define a public property that can be accessed by other components.
- **@wire:** This annotation is used to connect a component to an Apex method or a wire adapter.
- **@track:** This annotation defines a reactive property that causes the component to re-render when the property changes.

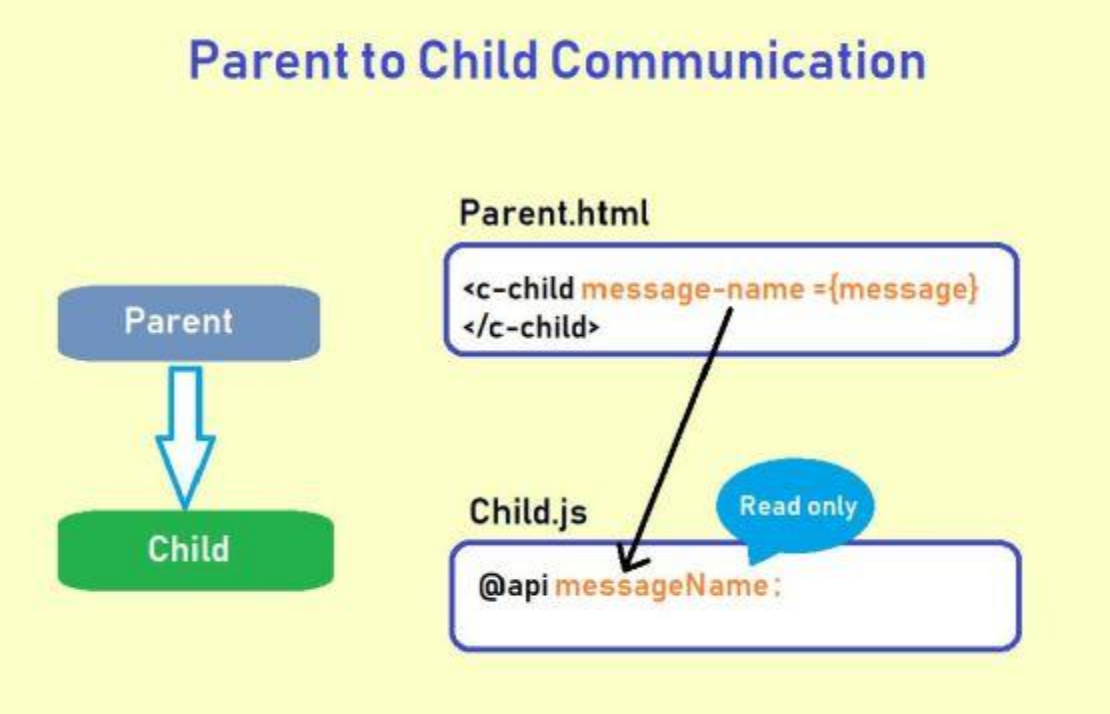
How to pass data from Parent to child communication?

To pass data down in the component hierarchy, the **child component must declare a public API**. There are two forms in the public API of a component.

- public properties
- public methods

Define parent to child communication using Public Property?

✸ Crack the Code: 100 Interview Questions on Lightning Web Components!



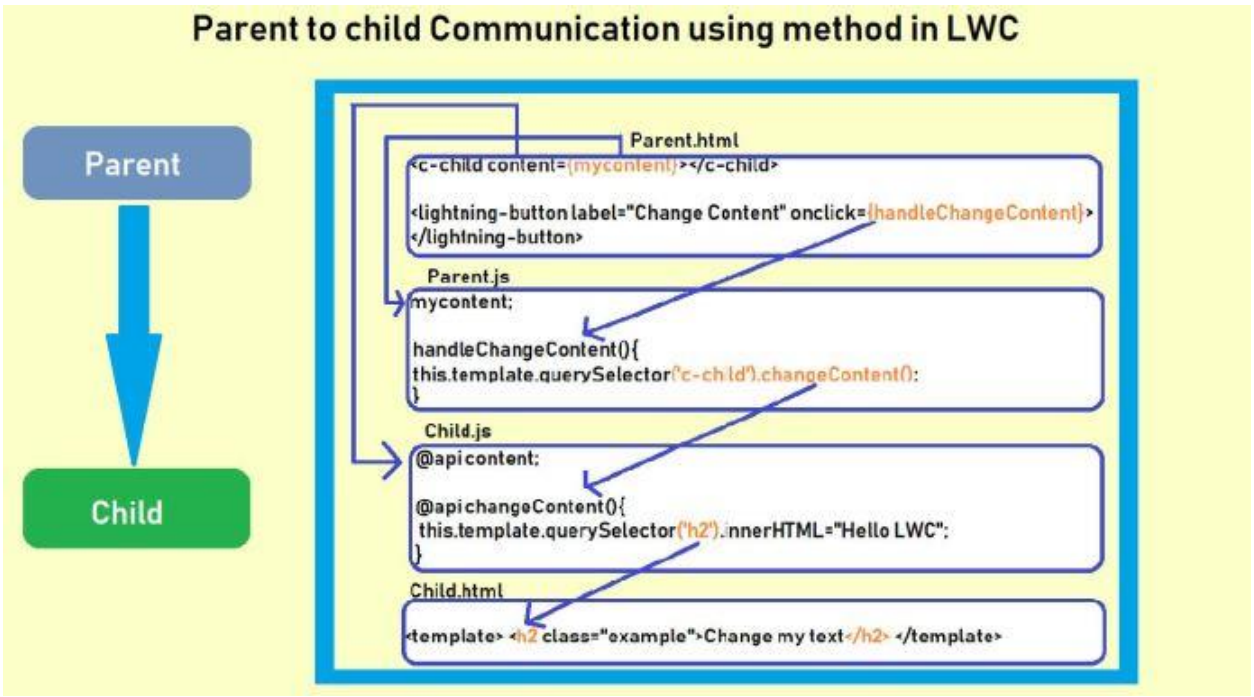
If you want to make the property to be called from another component, you need to declare this property with **@api decorator in the calling component**. Along with decoration you have to import this in your js file as:

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

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Along with this, if the value changes on this property, the component will be re-rendered automatically.

Define parent to child communication using Public Property?



Created a method and used api decorator before it so as to expose it in parent component.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

What are lifecycle hooks in Lightning Web Component?

Lightning web components have callback method, that are used to handle lifecycle of the components. These are called lifecycle hooks. You can override these hooks to modify the behavior.

Note: All lifecycle hooks are part of HTML custom element specification except renderedCallback() and errorCallback() which are specific to LWC.

- constructor()
- connectedCallback()
- rendered()
- renderedCallback()
- disconnectedCallback()
- errorCalllbback()

Explain Lifecycle in LWC?

1.Constructor is called

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

2.Public property are set on parent

3.Parent is inserted to DOM

4.Once parent is inserted to DOM, Connected callback is called on parent

5.From this connected callback as parent is already inserted, you can reference different DOM elements.

6.Parent is rendered

7.Once parent is rendered, Constructor is called on the child

8.Public property of child are set

9.Child is inserted to DOM

10.Once child is inserted to DOM, Connected callback is called on child

11.Child is rendered

12.Once child is rendered, rendered callback is called

13.Once child components are rendered on the screen, then parents rendered callback is called.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Too hard to remember?

Let's use the Domino's Pizza website scenario to illustrate this in a more relatable way:

Imagine you're visiting the Domino's Pizza website to order a pizza. The process you go through on the website is like how components are rendered in LWC. Here's how it works:

1. Constructor is called (Parent & Child):

This is like you deciding to order pizza from Domino's and opening their website. The website's main page (parent) starts loading, and this is analogous to the constructor method being called. The main page is setting up resources, initializing state.

2. Public property is set on the parent:

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Now, the website asks for your location to deliver the pizza. This is like setting a public property on the parent; the website can't proceed without this info, much like a component might need input data to function properly.

3. Parent is inserted into the DOM:

With your location set, the website displays the Domino's branch closest to you along with the menu. This stage is like the parent component being inserted into the DOM; it's now part of the webpage and you can interact with it.

4. Connected callback is called on the parent:

As you browse the menu, the website might load special offers based on your location or previous orders. This is like the connected callback; now that the parent component is in the DOM, it can handle tasks like loading data or setting up event listeners.

5. Parent is rendered:

You're now looking at the menu and deciding what pizza to order. The parent component (the main page) is fully rendered and visible.

6. Constructor is called on the child:

You decide on a pizza and click on it. This action loads a new component: the pizza customization page. This is like the child component's constructor being called; the child (pizza customization page) begins its initial setup.

7. Public property of child is set:

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- The customization page knows which pizza you selected and shows you the correct toppings and price. This is like public property being set on the child; it needs this data to display the correct information.

8. Child is inserted into the DOM:

You're now viewing the customization options and perhaps choosing different toppings or a different crust. The child component is now in the DOM.

9. Connected callback is called on the child:

As you interact with the options, the page might display calorie counts or update the price in real-time. This is like the child's connected callback; it can now react to changes and update itself accordingly.

10. Child is rendered:

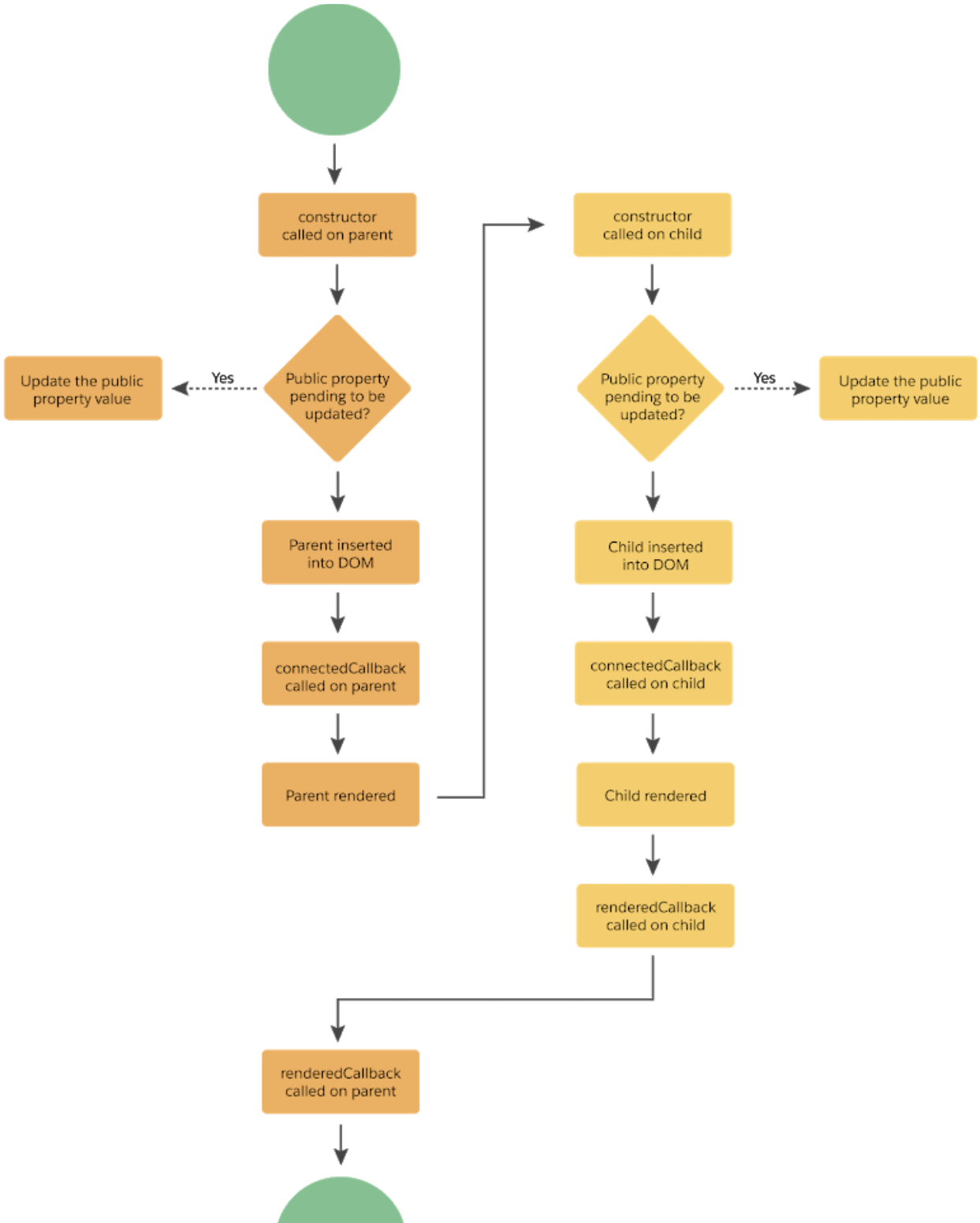
You finalize your pizza with your preferred toppings and it's now ready to be added to the cart. This is like the child component being fully rendered.

11. Rendered callback is called (Child & Parent):

After adding the pizza to your cart, maybe you decide to add a drink too. The page updates the cart total. This is like the rendered callback; the components have been rendered and now any post-render activities occur. First, this happens in the child (updating the cart), then in the parent (maybe updating the total items in the header's cart icon).

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Crack the Code: 100 Interview Questions on Lightning Web Components!



✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Explain Lifecycle hooks with real world example?

let's imagine we're dealing with a "Pizza Tracker" component on the Domino's Pizza website. This component allows customers to see the real-time status of their pizza order. Here's how lifecycle hooks would work in this scenario:

1. Order Initiation (Construction): As soon as you place your order for a delicious pizza, the "Pizza Tracker" component is created. At this point, the tracker is like a blank screen ready to display stages like 'Preparing,' 'Baking,' 'Quality Check,' 'Out for Delivery,' and 'Delivered.'

2. Tracker Activation (Connected Callback): The blank "Pizza Tracker" is now live on your screen, but it hasn't started tracking your order just yet. It's like having the tracker board up with all the stages, but no progress indicated. So, the website sends a request to Domino's system asking.

3. Progress Update (Rendered Callback): As the Domino's kitchen gets busy with your order, the system sends updates back to the "Pizza Tracker." First, the 'Preparing' stage lights up, followed by 'Baking,' and so

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

on. Your screen reflects these updates in real-time, getting you more excited as the delivery time approaches. If there's a delay or a rush, the tracker adjusts the progress display accordingly.

4. Unexpected Hiccups (Error Callback): If something goes wrong — let's say the 'Quality Check' fails, and they decide to remake your pizza — the tracker might not get the usual progress data. This stage catches such issues and might display a message like, "Your pizza is being remade to ensure top quality. Thank you for your patience!" This keeps you informed and reassured, rather than just seeing the progress halt with no explanation.

5. Pizza Arrival (Disconnected Callback): Hooray! Your pizza arrives, you're ready to eat, and you close the "Pizza Tracker." The tracker understands that its job is done so it shuts off its lights and goes to rest.

Explain constructor?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- Hook that fires when a **component instance is created.**
- The first statement must be super() with no parameters. This call establishes the correct prototype chain and value for this. Always call super before touching this.
- Don’t use a return statement inside the constructor body, unless it is a simple early-return (return or return this).
- **At that point, the component properties won’t be ready yet.**
- It flows from parent to child.

Do not assign any property in Constructor because they are not ready at this point. The flow of Constructor is flows from **parent to child**. It means if there are child components in LWC, then first parent constructor will be called, followed by child constructor.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {

constructor() {
super();
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
console.log('In Constructor');
}
```

What is connectedCallback?

- Fires when a component is inserted into the DOM.
- The component properties will be ready, but the **child elements won't be yet.**
- **It can fire more than once.** For example, if you remove an element and then insert it into another position, such as when you reorder a list, the hook fires several times. If you want code to run one time, write code to prevent it from running twice.
- It flows **from parent to child.**
- The equivalent in Aura is the init() event.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {

connectedCallback(){
console.log('In connectedCallback');
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
}
}
```

What is the use of ConnectedCallback()

We generally use it when we need to do something when the component is loaded. Example: query some data when the component is loaded.

The execution flow of connectedCallback is parent to child. So we cannot access child elements in the connectedCallback, because they are not inserted yet.

app.html

```
<template>
<lightning-input label="Name" value={greeting}>
</lightning-input>
</template>
```

app.js

Crack the Code: 100 Interview Questions on Lightning Web Components!

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

export default class App extends LightningElement {

    connectedCallback(){
        console.log('connectedcallback');
        this.greeting ='sfdcAmplified';
    }
}
```

What is render?

- Hook that overrides the standard rendering functionality.
- Gets invoked after connectedCallback() and must return a valid HTML template.
- It can fire more than once.
- It flows from **parent to child.**

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {

render(){
console.log('In render');
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
}
}
```

What is Use Case of render method?

We have a salesTemplate and on click of which we are able to redirect to service template.

In this use case we have following file structure:

- salesTemplate.html
- serviceTemplate.html
- salesTemplate.js

What is renderedCallback?

- Fires when a component rendering is done.
- It can fire more than once.
- It flows from child to parent.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

The renderedCallback() is unique to Lightning Web Components. In simple words, if you are dealing with **nested parent child component**, renderedCallback() method in child component will get fired first.

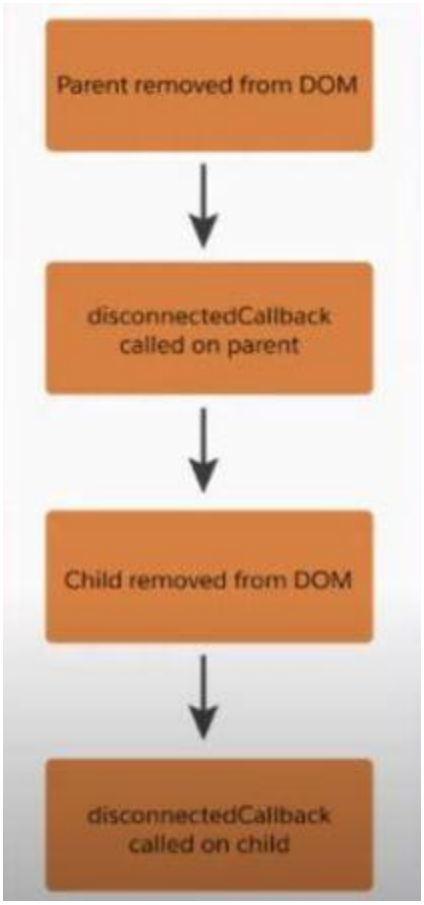
Note, You can set the properties in renderedCallback but best practice is do not use renderedCallback() to change the state of a component, such as loading values or setting properties. Use getters and setters instead.

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement {

renderedCallback(){
console.log('In renderedCallback');
}
}
```

What is Disconnected Callback. How is it used?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



disconnectedCallback()

- Fires when a component is removed from the DOM.
- It can fire more than once.
- It flows from parent to child.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- The equivalent in aura was the unrender() methods of the custom renderer file.

When this component removed form DOM, it will fire disconnectedCallback and clear array.

1. Parent is removed from DOM
2. Disconnected Callback is called on parent
3. Child removed from DOM
4. Disconnected Callback on the child is called

```
import { LightningElement } from 'lwc';
export default class App extends LightningElement
{
disconnectedCallback(){
console.log('In disconnectedCallback');
}
}
```

What is errorCallback?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Captures errors that may happen in all the descendant components lifecycle hooks.

```
errorCallback(error, stack) {
alert(error);
}
```

What is Use of lifecycle hooks?

Dynamic Rendering of Components (using the render hook):

Imagine you're using Domino's website to order your pizza. Depending on the time of day, the website might show you different deals; for example, a breakfast special if it's morning, a lunch deal around noon, and a dinner combo in the evening.

What happens in LWC: Here's where the lifecycle hook comes in. Before the Domino's website shows you any deals, it checks the time. Based on this condition (morning, afternoon, or evening), the lifecycle hook method

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

decides which template to load (breakfast, lunch, or dinner deals). So, you see the most relevant offers when you're browsing, making your experience smoother and more personalized.

2. Handling Errors Gracefully (using the errorCallback hook):

Let’s say you're tracking your order on the Domino's website, but suddenly there's a system glitch, and the tracker can't get your pizza's status. Without proper handling, this could confuse you – you might wonder if your order got lost or cancelled.

What happens in LWC: This situation is where `errorCallback` plays a crucial role. Instead of the tracker just freezing or showing a technical error that might confuse you, the `errorCallback` function steps in. It catches this issue and could display a friendly message like, "We're currently experiencing a hiccup in our system, but don’t worry, your pizza is still on its way!"

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Difference between Render and rendered call back?

In the context of Domino's "Pizza Tracker" example, the `render` and `renderedCallback` methods in the Lightning Web Component (LWC) lifecycle have specific roles that help the tracker function properly and provide real-time updates to the user.

1. render method: This method is a part of the rendering lifecycle and is responsible for rendering the template of the component. In our "Pizza Tracker," the `render` method controls **what the user sees initially**. This could be the **tracker interface with the different stages like 'Preparing,' 'Baking,' 'Quality Check,' 'Out for Delivery,' and 'Delivered,' but with no active status yet**.

- In simpler terms, the `render` **method sets up the stage**, but the actors (status indicators) haven't started performing.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

2. renderedCallback method: This method is invoked after the component is rendered and whenever it rerenders. Every time there's a new update from the kitchen about the pizza's status, the `renderedCallback` method updates the UI to reflect this. So, when your pizza moves from 'Preparing' to 'Baking,' this method is what updates the tracker on your screen, lighting up the new status.

- Think of the `renderedCallback` as the stage manager in a play. When a new scene begins (your order status updates), the stage manager ensures the correct props are in place and the actors are ready (the tracker reflects the new status).

In summary, the `render` method sets up the "Pizza Tracker" display, while the `renderedCallback` ensures this tracker accurately and promptly reflects each update to your pizza's journey from the oven to your doorstep.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

In Salesforce's Lightning Web Components (LWC), data from Salesforce is often retrieved using either the `wire` service or imperative Apex. While both approaches facilitate communication with the server and can call Apex methods to retrieve data, they operate differently. Let's use the Domino's website as an analogy to explain these concepts:

What is difference between wire and imperative in LWC?

1. Wire Service (Automatic and Reactive):

Imagine the Domino's website has a feature where the menu updates automatically whenever a new pizza is added to their system. You, as a customer, just open the menu page, and it always shows the current offerings without needing to refresh the page or click anything.

In LWC: This is similar to the `wire` service. When you "wire" a function or property, Salesforce automatically listens for changes that affect the data

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

and retrieves fresh data when necessary. It's reactive; it updates the component's data whenever the relevant Salesforce data changes.

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

import getPizzas from '@salesforce/apex/Dominos.getPizzas';

export default class PizzaMenu extends LightningElement {

    @wire(getPizzas) pizzas;

}
```

In this example, `pizzas` gets updated automatically if the data in Salesforce changes.

2. Imperative Apex (Manual and More Control):

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Now, imagine you're on Domino’s website, and you see a "Refresh Menu" button. The menu only updates when you click this button. You have control over when it happens, and it's not automatic.

In LWC: This is like using imperative Apex. You call Apex methods directly at the specific times you decide, like in response to a user action (e.g., clicking a button), or a lifecycle event, etc.

Differences:

Reactivity: `wire` is reactive and automatic, while imperative calls are manual and only happen when specifically invoked.

Control: Imperative calls give you more control over error handling, complex logic, and can be used for more than just fetching data (e.g., sending data, calling multiple methods consecutively, etc.), whereas `wire` is mainly used for directly wiring data to properties or functions.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

`wire` is best when data should stay in sync with the server, and imperative is better when you need more control over when and how data is fetched or manipulated.

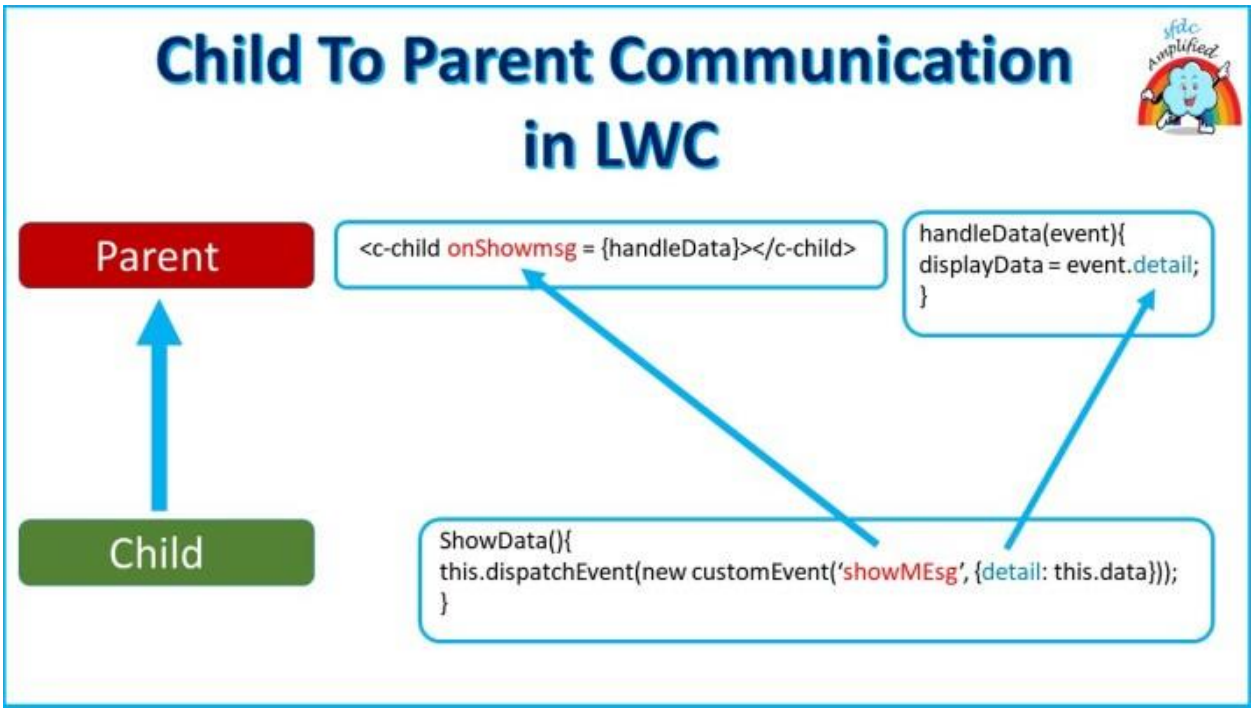
What are Events?

Events are actions that happen in the system we are programming — the system produces a signal when an event occurs and provides a mechanism by which action can be automatically taken when the event occurs.

For example, in an airport when the runway is clear for a plane to take off, a signal is communicated to the pilot, and as a result, they commence piloting the plane.

How to pass data from Child to Parent?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



Data must be passed up using Events. Events are just standard JavaScript events that we fire from the child component and listen into the parent.

Note: To communicate **down** the component containment hierarchy, we can pass properties to the **child via HTML attributes, or public methods**. To communicate down the component hierarchy we don't need events.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

How do we communicate between different components which don't have a relation?

To communicate between two different components that do not have any direct relation, we use publish-subscribe utility.

What is Difference between event in Aura VS LWC?

Unlike component or application events in Aura , LWC uses **Standard DOM events.**

In Aura component, we need to create a separate event file and register the event, where event is fired.

Moreover, to handle the event, a handler is needed in component hierarchy. In LWC, registering of events is not required.

Step 1: Create an Event

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
new customEvent(eventName, props);
```

We need custom event to communicate up the component hierarchy.

Step 2: Dispatch an Event

Then, we can make an event target dispatch the created event invoking the dispatchEvent standard method.

```
this.dispatchEvent(new customEvent(eventName, props);
```

Step 3 Handle an Event:

There are two ways to listen to an event.

- Declaratively from the component's HTML template
- Programmatically using an imperative JavaScript API

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

It’s better to listen from the HTML template since it reduces the amount of code we need to write. To handle events, define methods in the component’s JavaScript class

How to attach an Event Listener Declaratively?

Declare the listener in html of parent component
parent.html

```
<template>
        <c-child-component oneventName={listenerHandler}></c-child-component >
</template>
```

Define the handler function listenerHandler, JavaScript file.

parent.js

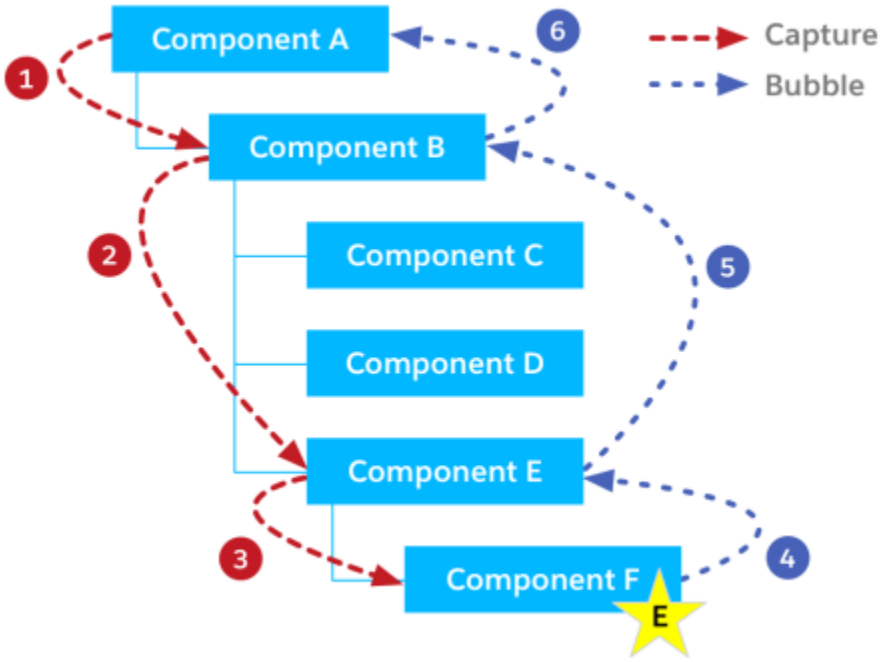
```
import { LightningElement } from 'lwc';
export default class Parent extends LightningElement {
    listenerHandler(){
        // Code runs when event is received
    }
}
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

What is Event Propagation?

Event propagation is a mechanism that defines how events propagate or travel through the DOM tree to arrive at its target and what happens to it afterward

What is event bubbling and Capturing?



✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

The meaning of bubbling in real life is bubble goes from bottom of glass to the top of the glass when we pour soda to the glass.

On a web page bubbling means when we have an event on an element, the execution goes from lowest element to the top.

Example: We click a button; we not only click that button but you actually clicking parent and its parent all the way to the body.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

What is meaning of bubbles : false and composed : false (Default behavior)?

- This is the default behavior, and bubbles and composed are not required to be defined explicitly.
- When this is in effect, the event would not bubble up the DOM, as well as it would not cross shadow boundary.
- We can listen these event only via defining the event handler to component that fires the event.

What happens when bubbles : true and composed : false ?

In this behavior, the event bubbles up the DOM, but don’t cross the shadow boundary. In simple terms, It bubbles up inside the current template, in which event is fired, but it would not cross it’s boundary.

For example :- There are two component, where parent component contain child component. If event is fired inside child component, the event only

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

bubbles up inside child component, and it would not cross it's boundary, and parent component will not be able to listen this event.

How to attach an Event Listener Programmatically?

In this case we will define listener and handler function in the JavaScript file itself.

```
import { LightningElement } from 'lwc';
export default class Parent extends LightningElement {
  constructor() {
    super();
    this.template.addEventListener('listenerHandler', this.handleListener);
  }
  handleListener = () => {};
}
```

Step 1 – Create event- new customEvent(eventName, props);

Step 2 – dispatch event- this.dispatchEvent(new customEvent(eventName, props);

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

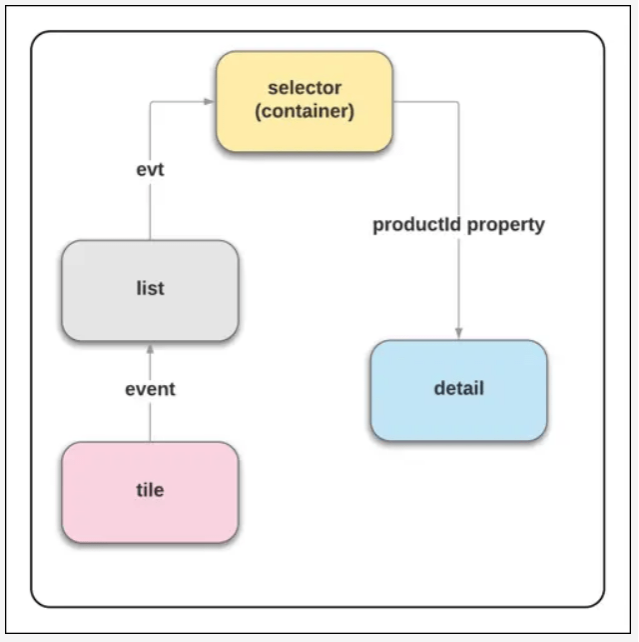
Step 3: Attach Event Listener using addEventListener.

Step 4: Handle the event – Define the handler function handleCustomEvent in JavaScript file.

Point to Remember: Events Up, Properties Down

In a complex component (one that contains several parent and child components) we recommend you propagate th**e event up through the component hierarchy,** so parent components can respond to child events. If you have other child components (not the one firing the event) you can **pass a property down** to those children in response to the event.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



When don’t we need apex to get data from salesforce?

For below three approaches we don’t need to write apex to do get data:

- Use Lightning Data Service to access Salesforce Data
- Use Base Lightning Component to work with Salesforce records.
- Use Wire Service to get Salesforce Data

How to Use Apex to Access Salesforce Data?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Step 1: call apex Method from apex class into JavaScript

Lightning web components can import methods from Apex classes.

Note: Before you use an Apex method, make sure that there isn't an easier way to get the data

Syntax

```
import apexMethodName from
'@salesforce/apex/Namespace.Classname.apexMethodReference';
```

Step 2: Call an Apex method as function into Lightning Web Component

- Wire a property
- Wire a function
- Call a method imperatively

How to Expose Apex Methods to Lightning Web Components?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

To expose an Apex method to a Lightning web component

- method must be static
- method must be either global or public
- Annotate the method with @AuraEnabled

What is the use of Client-Side Caching?

- Annotating the Apex method with @AuraEnabled(cacheable=true) **improves the run time performance.**
- To set cacheable=true, a method must only get data, **it can't mutate (change) data.**
- Marking a method as **cacheable improves component's performance** by quickly **showing cached data** from client-side storage without waiting for a server trip.
- If the cached data is stale, the framework retrieves the latest data from the server. Caching is especially beneficial for users on high latency, slow, or unreliable connections.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- To use @wire to call an Apex method, you must set cacheable=true

How to Wire an Apex Method to a Property?

If an Apex method is annotated with @AuraEnabled(cacheable=true), we can invoke it from a component via the wire service.

```
import apexMethodName from
'@salesforce/apex/Namespace.Classname.apexMethodReference';
 @wire(apexMethodName,{ apexMethodParams })propertyOrFunction;
```

- apexMethodName—A symbol that identifies the Apex method.
- apexMethodReference—The name of the Apex method to import.
- Classname—The name of the Apex class.
- Namespace—The namespace of the Salesforce organization.
- apexMethodParams—An object with parameters for the apexMethod, if needed. If a parameter value is null, the method is called.
- propertyOrFunction—A private property or function that receives the stream of data from the wire service.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- If a property is decorated with @wire, the results are *returned to the* ***property's*** *data property or error property.*
- If a function is decorated with @wire, the results are *returned in an* ***object*** *with a data property or an error property.*

Scenario – Get the list of contact records from Salesforce using wire?

- To get contact data, the component wires an Apex method "getContactList". The Apex method makes a SOQL query that returns a list of contacts.
- **Method must be static, and global or public.** The method must be decorated with @AuraEnabled(cacheable=true).

```
public with sharing class ContactMaster {
    @AuraEnabled(cacheable=true)
    public static List<Contact> getContactList() {
        List<Contact> conList= [SELECT Id, Name, Title, Phone, Email FROM
Contact LIMIT 10];
        return conList;
    }
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
}
```

contactList.js

- The component's JavaScript code imports the Apex method and invokes it through the wire service.
- The wire service either gets the list of contacts to the **contacts.data property**, or returns an error to the contacts.error property.

```
import { LightningElement, wire } from 'lwc';
import getContactList from
'@salesforce/apex/ContactMaster.getContactList';

export default class contactList extends LightningElement {
    @wire(getContactList)
    contacts;
}
```

contactList.html

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

The template uses **the if:true directive to check whether the contacts.data property is true**. If it is, it iterates over it and renders the name of each contact.

```
<template>
    <lightning-card title="ApexWireMethodToProperty" icon-
name="custom:custom63">
        <div class="slds-m-around_medium">
            <template if:true={contacts.data}>
                <template for:each={contacts.data} for:item="contact">
                    <p key={contact.Id}>{contact.Name}</p>
                </template>
            </template>
            <template if:true={contacts.error}>
                <c-error-panel errors={contacts.error}></c-error-panel>
            </template>
        </div>
    </lightning-card>
</template>
```

Output

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Write a scenario to Wire an Apex Method to a Function?

ContactMaster.apxc

- This component calls the Apex method as apexWireMethodToProperty.
- The method must be static, and global or public. The method must be decorated with @AuraEnabled(cacheable=true).

```
public with sharing class ContactMaster {
    @AuraEnabled(cacheable=true)
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
    public static List<Contact> getContactList() {
       List<Contact> conList= [SELECT Id, Name, Title, Phone, Email FROM
Contact LIMIT 10];
        return conList;
    }
}
```

contactDisplay.js

- The component's JavaScript code imports the Apex method and invokes it via the wire service.
- The wire service gets the results to the wiredContacts() function **via an object with either an error or data property.**
- If the wire service provisions data, it's assigned to this.contacts, which is used in the template.
- If it provisions error, it's assigned to this.error, which is also used in the template. If the value of these properties change, the template rerenders.

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

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
import getContactList from
'@salesforce/apex/ContactMaster.getContactList';

export default class contactDisplay extends LightningElement {

    contacts;
    error;

    @wire(getContactList)
    wiredContacts({ error, data }) {
        if (data) {
            console.log('data: ', data);
            this.contacts = data;
            console.log('this.contacts: ', this.contacts);
            this.error = undefined;
            console.log('this.error: ', this.error);

        } else if (error) {
            console.log('error ', error);
            this.error = error;
            console.log('this.error: ', this.error);
            this.contacts = undefined;
            console.log('this.contacts : ', this.contacts);
        }
    }
}
```

contactDisplay.html

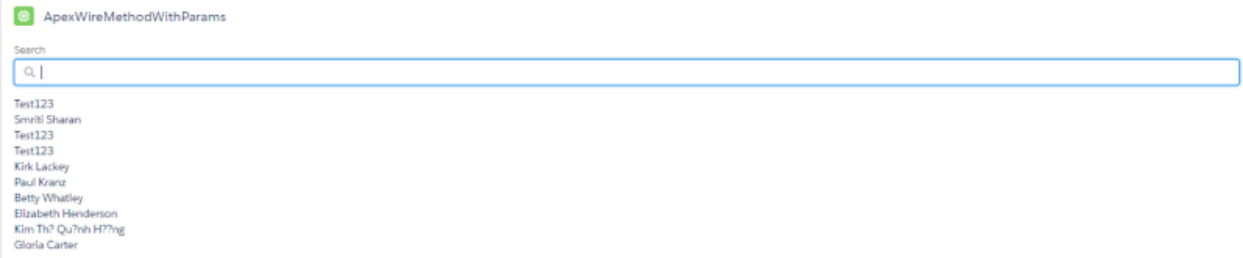
✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- The template uses the if:true directive to check for the JavaScript contacts property. If it exists, it iterates over it and renders the name of each contact.
- If the error property exists, the component renders <c-error-panel>.

```
<template>
    <lightning-card title="Apex Wire Method To Function" icon-
name="custom:custom63">
        <div class="slds-m-around_medium">
            <template if:true={contacts}>
                <template for:each={contacts} for:item="contact">
                    <p key={contact.Id}>{contact.Name}</p>
                </template>
            </template>
            <template if:true={error}>
                <c-error-panel errors={error}></c-error-panel>
            </template>
        </div>
    </lightning-card>
</template>
```

Scenario – Find contacts from Salesforce?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



ContactMaster.apxc

The Apex method “findContacts” takes a string parameter “searchKey” and returns a list of contacts

```
public with sharing class ContactMaster {

    @AuraEnabled(cacheable=true)
   public static list<Contact> findContacts(String searchKey){
            String key = '%' + searchKey + '%';
       List<Contact> conlist = [SELECT Id,Name from Contact WHERE
Name LIKE :key LIMIT 10 ];
       return conlist;
   }
}
```

contactDisplay.html

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

The template uses “searchKey” as value in input field

```
<template>
    <lightning-card
        title="Apex Wire Method With Params"
    >
        <div class="slds-var-m-around_medium">
            <lightning-input
                type="search"
                onchange={handleKeyChange}
                class="slds-var-m-bottom_small"
                label="Search"
                value={searchKey}
            ></lightning-input>
            <template if:true={contacts.data}>
                <template for:each={contacts.data} for:item="contact">
                    <p key={contact.Id}>{contact.Name}</p>
                </template>
            </template>
            <template if:true={contacts.error}>
                <c-error-panel errors={contacts.error}></c-error-panel>
            </template>
        </div>


    </lightning-card>
</template>
```

contactDisplay.js

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

- JavaScript shows the value of the **searchkey parameter with $ to indicate that it's dynamic and reactive.**
- It references a property of the component instance. If its value changes, the template rerenders.

```
import { LightningElement, wire } from 'lwc';
import findContacts from '@salesforce/apex/ContactMaster.findContacts';


export default class ContactDisplaywithParameters extends
LightningElement {

    searchKey = '';

    @wire(findContacts, { searchKey: '$searchKey' })
    contacts;

    handleKeyChange(event) {
        const searchValue = event.target.value;
            this.sexarchKey = searchValue;
    }
}
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

How do you create a Lightning App that embeds an LWC component?

```
<aura:application extends="force:slds">
    <c:myLwcComponent />
</aura:application>
```

In the above snippet, we have the component **(myLwcComponent)**. Suppose we need to embed this component in the lightning app. So, here we can use the **<c:>** syntax used to reference the component within the app.

How to enable the LWC component cache?

We need to use the **@wire** decorator with the **getRecord** or **getListUi** wire adapters. These adapters automatically cache the results of the wire call in the browser's cache, which can help improve performance.

How do you use the Salesforce REST API in LWC? What are the best practices for integrating with external systems?

To use the Salesforce REST API in LWC, we can use the built-in fetch method or a third-party library like **axios** or jQuery.

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

export default class MyComponent extends LightningElement {
    connectedCallback() {
        const endpoint =
'/services/data/v53.0/query?q=SELECT+Name+FROM+Account';
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
        fetch(endpoint, {
            method: 'GET',
            headers: {
                'Authorization': 'Bearer ' + authToken
            }
        })
        .then(response => {
            return response.json();
        })
        .then(data => {
            console.log(data);
        })
        .catch(error => {
            console.error(error);
        });
    }
}
```

Can you explain how to use the Lightning Message Service in LWC and give an example?

To use the Lightning Message Service in LWC, we are required to follow these steps:

- **Define a message channel:** A message channel defines the topic or theme of the messages that will be published and subscribed to. We can define a message channel using the “lightning/messageService” module.

```
import { LightningElement } from 'lwc';
import { createMessageChannel } from 'lightning/messageService';
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
export default class MyComponent extends LightningElement {
    channelName = 'myChannel';

    connectedCallback() {
        this.channel = createMessageChannel({
            channelName: this.channelName,
            isSubscribe: true
        });
    }
}
```

Publish a message: After defining a message channel, we can publish messages on that channel using the **publish()** method.

```
import { LightningElement } from 'lwc';
import { createMessageChannel, publish } from
'lightning/messageService';

export default class MyComponent extends LightningElement {
    channelName = 'myChannel';

    handleClick() {
        const message = {
            recordId: '001XXXXXXXXXXXXXX'
        };
        publish(this.channel, message);
    }
}
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

Subscribe to a message: Finally, to receive and act upon messages published on a message channel, we can use the **subscribe()** method.

```
import { LightningElement } from 'lwc';
import { createMessageChannel, subscribe } from
'lightning/messageService';

export default class MyComponent extends LightningElement {
    channelName = 'myChannel';

    connectedCallback() {
        this.channel = createMessageChannel({
            channelName: this.channelName,
            isSubscribe: true
        });

        this.subscription = subscribe(
            this.channel,
            message => {
                console.log(message.recordId);
            }
        );
    }

    disconnectedCallback() {
        unsubscribe(this.subscription);
    }
}
```

In this example, we define a method called **connectedCallback()** that sets up a subscription to the **"myChannel"** message channel. When a

Crack the Code: 100 Interview Questions on Lightning Web Components!

message is received, the **console.log()** method is called with the recordId property of the message.

To clean up the subscription when the component is removed from the DOM, we define a **disconnectedCallback()** method that unsubscribes from the channel.

What is the purpose of the Lightning Data Service in LWC?

The purpose of the Lightning Data Service in LWC is to provide a declarative and efficient way to perform CRUD (Create, Read, Update, Delete) operations on Salesforce records. It is a framework-provided service that works as a data layer between the LWC component and the Salesforce database.

What are the benefits of The Lightning Data Service?

- **Declarative data binding:** With the Lightning Data Service, you can declaratively bind your component's UI elements to record data without writing any Apex code or SOQL queries. This simplifies the code and reduces the development time.
- **Caching and automatic record updates:** The Lightning Data Service caches record data locally on the client side and automatically updates it when changes occur in the database. This improves performance and reduces the number of round trips to the server.
- **Automatic CRUD operations:** The Lightning Data Service provides a set of methods to create, read, update, and delete records. These methods are automatically implemented and available for use in your LWC component.

Crack the Code: 100 Interview Questions on Lightning Web Components!

- **Automatic sharing and field-level security:** The Lightning Data Service **automatically enforces sharing rules and field-level security** when reading or modifying records.

What is the role of the Shadow DOM in LWC, and how does it differ from the traditional DOM?

In LWC, the Shadow DOM is used to encapsulate the component's DOM hierarchy and prevent CSS styles and JavaScript code from bleeding out of the component and interfering with the rest of the page.

When we create an LWC component, its HTML template, and CSS styles are automatically encapsulated in the Shadow DOM. **The JavaScript code can access the Shadow DOM via the "this.template" property**, but it cannot access the rest of the page's DOM directly.

Why do we use $ when passing property in wire function, what does it mean?

$ prefix tells the wire service to treat values passed as a property of the class and evaluate it as this.propertyName and the property is reactive. If the property's value changes, new data is provisioned and the component renders.

@wire(getcoffee,{boatTypeId : '$coffeeTypeId})

Is wire method called multiple times during lifecycle of component ?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

True

What is the difference between event.StopPropogation() and Event.preventDefault()?

stopPropagation prevents further propagation of the current event in the capturing and bubbling phases.

preventDefault prevents the default action the browser makes on that event.

What are type of Quick actions in LWC?

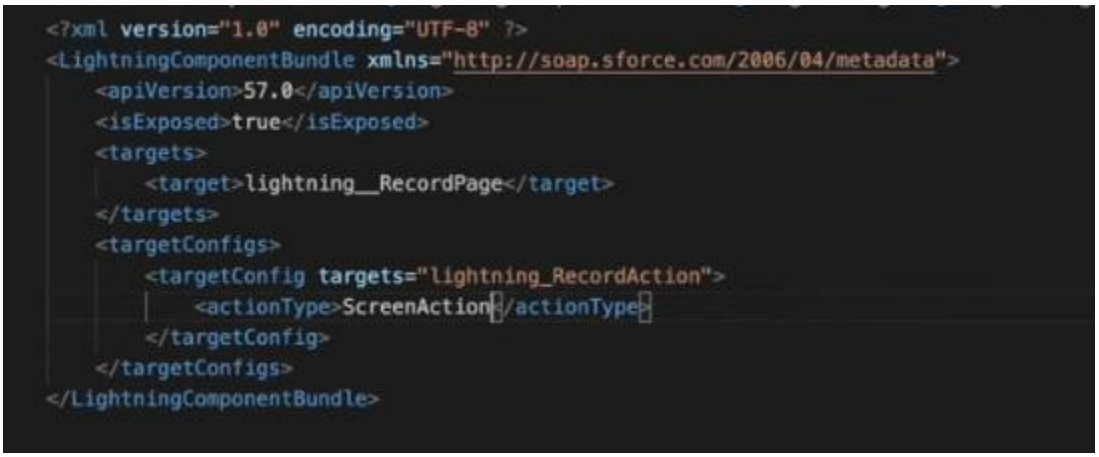
There are two types of quick actions in LWC :

Screen actions : Normal action which open a modal

Headless actions : Which don’t have any html file rather just logic written in js

We mention action type as Screen action or Headless Action in xml file to define type of action.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!



```
<?xml version="1.0" encoding="UTF-8" ?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata">
    <apiVersion>57.0</apiVersion>
    <isExposed>true</isExposed>
    <targets>
        <target>lightning__RecordPage</target>
    </targets>
    <targetConfigs>
        <targetConfig targets="lightning_RecordAction">
            <actionType>ScreenAction</actionType>
        </targetConfig>
    </targetConfigs>
</LightningComponentBundle>
```

Why do we extend Lightning Element ?

LightningElement is custom wrapper on HTMLElement which actually contains all the lifecycle hooks methods over which Salesforce did some customization.

If we parent component A and there are two component B and C as child components. How can we communicate between B and C ?

We should fire up event from child component B to parent A then from A call attribute / function exposed (as @api) and pass data into C component.

What does composed = true does in an event ?

These types of events can bubble up inside dom and also able to cross shadow boundary.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

When do I use @track on a property? Do I still need it considering all properties are by default reactive now?

After Spring 20 all the properties are made by default reactive so we don't need @track for primitive properties. We still need it for array or object type properties.

Can I use multiple decorators on one property?

No we cant use multiple decorators on same property.

What is LMS?

LMS is defined as the standard publish-subscribe library that enables communication with DOM across the components be it Visualforce Pages, Aura components, and Lightning Web Components (LWC) all can use it to publish message and listen to messages published by others.

Do we have application events in LWC?

Ans: We dont have application event as such in LWC like Aura rather we have LMS in LWC to communicate between components which are not part of same hierarchy.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

How can we navigate user from LWC component to record detail page?

We can do it using NavigationMixin service

Can I get current user ID in LWC without apex?

Yes we can get current user ID without apex by simply importing

import Id from '@salesforce/user/Id'

Can i call function annotated with @AuraEnabled(cacheable= true) imperatively ?

Ans: Yes

Can we do DML in method annotated with @AuraEnabled(cacheable= true)?

Ans: No we can't do DML inside any method annotated with cacheable = true , you will receive an error as DMLLimit Exception.

How to refresh cache when calling method imperatively?

We have to use getRecordNotifyChange(RecordIds) which refreshes the LDS cache providing you the latest data this will work only if cacheable = true was there. Otherwise we will have to call the function again from our js to get the latest data.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

When do we face error of “Cant assign to Read only property” in LWC?

This error occurs when you are trying to make changes to a property which is marked as @api , ideally you should clone the value then make the changes to it.

How to query all lightning-input , combobox, radio buttons using one querySelector or do I have to use multiple ?

We can query all of them using one query selector only no need to write multiple for each tag. We can pass all tags (,) separated to query all.

```
const allValid = […this.template.querySelectorAll(‘lightning-input,lightning-combobox,lightning-radio-group’)];
```

(…) spread operator before querySelectorAll because querySelector returns you the nodelist and using spread operator we convert it to array of items otherwise we wouldn’t be able to use array functions like map or reduce.

How do you handle errors and exceptions in LWC?

🌟 Crack the Code: 100 Interview Questions on Lightning Web Components!

Error handling in LWC can be done using try-catch blocks or by leveraging the onError lifecycle hook. You can catch exceptions and display appropriate error messages to the user, ensuring a smooth user experience.

```
<template>
try {
  // Code that may throw an exception
} catch (error) {
  // Handle the error and display an error message
}
</template>
```

How do you make an HTTP callout from a Lightning Web Component?

You can use the fetch API to make HTTP callouts in LWC. Import the fetch method and use it to send requests to external services.

How can you make a Lightning Web Component available for use in the Salesforce App Builder?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

You need to define a custom component in the meta.xml file of your LWC and set the isExposed attribute to true.

What is the purpose of the lightning-record-edit-form component in LWC?

lightning-record-edit-form is used to create, view, or edit a record's fields using the Salesforce Lightning Data Service.

How can you communicate between sibling components in LWC?

You can use custom events and properties defined in a common parent component to facilitate communication between sibling components.

What is the role of the lightning/navigation module in LWC?

The lightning/navigation module allows you to navigate users to different pages within Salesforce, such as records, lists, or custom pages.

How do you unit test a Lightning Web Component?

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

You can write unit tests for LWC using Jest framework, which is supported by Salesforce. Test components, properties, and methods using Jest functions.

How can we track the database changes in the Lightning web component and refresh UI on data updates?

If we need to track the changes done by **Lightning Data Service** then we can use the **getRecordChangeNotifiy()** function from the LDS.

But If you need to track data changed from other places like Apex Trigger, Flow, API integrations then we need to use the **Change Data Capture (CDC). Or Platform event.**

How to query elements based on a data attribute and its value?

We can query the components if the attribute is defined on the component like below.

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

this.template.querySelector('[data-id]');

Also, we can query the component if we need to query the component with the value of the attribute like this.

this.template.querySelector(`[data-id='id-value']`);

You can also pass the string templates to the query selector function.

this.template.querySelector(`[data-id='${id}']`);

What is the role of the `lightning-datatable` component in LWC?

`lightning-datatable` component is used to display tabular data in a customizable and user-friendly format. Here's an example:

```
1<!-- datatableExample.html -->
2<template>
3    <lightning-datatable data="{data}" columns="{columns}" key-field=
4</lightning-datatable>
 </template>
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
1// datatableExample.js
2import { LightningElement } from 'lwc';
3
4export default class DatatableExample extends LightningElement {
5    data = [
6        { Id: '001', Name: 'Account 1', Industry: 'Finance' },
7        { Id: '002', Name: 'Account 2', Industry: 'Technology' },
8        { Id: '003', Name: 'Account 3', Industry: 'Healthcare' },
9    ];
10
11    columns = [
12        { label: 'Account Name', fieldName: 'Name', type: 'text' },
13        { label: 'Industry', fieldName: 'Industry', type: 'text' },
14    ];
15}
```

Scenario: Suppose you have written a LWC which has the @wire decorator to get the data from apex. But you are unable to get the data from the server. What could be one of the reason?

check whether you have used **cacheable=true** along with @AuraEnabled annotation in the apex method.

How can i reference record ID of page in my component which is fired from quick action?

To reference the record ID of the page in your Lightning Web Component (LWC) fired from a Quick Action, you can use

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

the `**lightning/pageReferenceUtils**` module. This module provides utility functions to extract parameters, including the record ID, from the page URL.

Here's how you can do it:

In your LWC component's JavaScript file (`**yourComponent.js**`), import the `**lightning/pageReferenceUtils**` module.

```
1import { LightningElement, wire } from 'lwc';
2import { getRecordId } from 'lightning/pageReferenceUtils';
```

Use the `**getRecordId**` function to extract the record ID.

```
1export default class YourComponent extends LightningElement {
2  recordId;
3
4  connectedCallback() {
5    this.recordId = getRecordId();
6  }
7}
```

The `**getRecordId**` function will automatically fetch the record ID of the page where the component is loaded, regardless of whether it is launched from a Quick Action or any other context.

Now, the `**recordId**` property will hold the record ID, and you can use it in your component's logic as needed.

When you place this component on a record page and trigger it from a Quick Action, it will automatically fetch and display the record ID of the

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

page where the Quick Action is invoked.

Can I call function annotated with @AuraEnabled(cacheable= true) imperatively ?

Yes

How to ensure that your LWC respects FLS?

1. Use Standard Components Where Possible: Use Lightning Data Service components

like `lightning-record-form`, `lightning-record-view-form`, and `lightning-record-edit-form`, respect FLS automatically. Whenever possible, use these components to handle data input/display.

```
<lightning-record-view-form record-id={recordId} object-api-name="Account">
    <lightning-output-field field-name="Name"></lightning-output-field>
    <!-- other fields -->
</lightning-record-view-form>
```

In this example, FLS is enforced for the "Name" field of the "Account" object, and if a user doesn't have permission to view the "Name" field, it won't be displayed.

2. Check FLS in Apex:

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

When writing Apex controllers that are called from your LWC, always check FLS. Use the `**SecurityEnforced**` annotation.

```
public with sharing class AccountController {

    @AuraEnabled(cacheable=true)

    public static Account getAccount(Id accountId) {

        return [SELECT Id, Name FROM Account WHERE Id = :accountId
WITH SECURITY_ENFORCED];

    }

}
```

The `WITH SECURITY_ENFORCED` clause ensures that the query respects FLS, throwing an exception if the user lacks the necessary permissions.

3. Use Schema Methods in Apex:

Utilize Schema methods in Apex to check if the current user has read, create, or edit access to a field. This is more manual but allows for fine-grained control.

```
public with sharing class SecurityCheck {

    public static Boolean isFieldAccessible() {
```

✵ Crack the Code: 100 Interview Questions on Lightning Web Components!

```
        return Schema.sObjectType.Account.fields.Name.isAccessible();
    }
}
```

Here, `isAccessible()` checks if the current user has read access to the Account's Name field.