1. Setup Your Trailhead Playground
    a. Go to https://sfdc.co/AuraToLWC
2. Deploy the Aura Component
    a. Open a command prompt.
    b. Clone the HelloWorld git repository:

```
git clone https://github.com/sfdc-cdev/AuraToLWC.git
```

    c. Navigate to the new AuraToLWC directory:

```
cd AuraToLWC
```

    d. Authorize your org with the Salesforce CLI, save it with an AuraToLWC alias and set the current user as the default user:

```
sfdx force:auth:web:login -s -a AuraToLWC
```

    e. When a browser window with the Salesforce login page opens, enter your credentials.
    f. Deploy the app code to the org:

```
sfdx force:source:deploy -p force-app/main/default
```

    g. Assign the Hello_World permission set to the current user:

```
sfdx force:user:permset:assign --permsetname Hello_World
```

    h. Open the org in a browser:

```
sfdx force:org:open
```

    i. In the browser, go to the app named Hello Improved and the tab Hello World.

3. Create Lightning web component

a. Open VS Code.
b. Add the project folder you just cloned from GitHub by clicking File > Open and navigating to the **AuraToLWC** folder.
c. In the sidebar, expand the **force-app/main/default folder**.
d. Right-click the **lwc** folder, click **SFDX: Create Lightning Web Component** and name the component **lwcHello**.
e. Within the <template> tags of the **lwcHello.html** copy the markup from the auraHello.cmp and let's look at how the different elements are transformed:
f. Make the component available in App Builder:

lwcHello.html

```
<template>
  <!--
    <aura:component implements="flexipage:availableForAllPageTypes"
      access="global">
      ……
    </aura:component>
  -->
</template>
```

lwcHello.js-meta.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<LightningComponentBundle xmlns="http://soap.sforce.com/2006/04/metadata"
fqn="lwcHello">
  <apiVersion>45.0</apiVersion>
  <isExposed>true</isExposed>
  <targets>
    <target>lightning__AppPage</target>
  </targets>
</LightningComponentBundle>
```

g. Remove the <aura:component> tags from lwcHello.html:

lwcHello.html

```
<template>
  <!--
  <aura:component>
    <aura:attribute name="greeting" type="String" default="world" />
    <lightning:card title="Aura_Hello" iconName="action:announcement">
      <aura:set attribute="actions">
        <lightning:button label="Count" onclick="{!c.count}" />
      </aura:set>
      <div class="slds-m-around_medium">
        <p>Hello, {!v.greeting}!</p>
        <lightning:input label="Name" value="{!v.greeting}" />
      </div>
    </lightning:card>
  </aura:component>
  -->
</template>
```

h. The aura attribute becomes a Lightning web component property:

lwcHello.html

```
<template>
  <!--
    <aura:attribute name="greeting" type="String" default="world" />
    <lightning:card title="Aura_Hello" iconName="action:announcement">
      <aura:set attribute="actions">
        <lightning:button label="Count" onclick="{!c.count}" />
      </aura:set>
      <div class="slds-m-around_medium">
        <p>Hello, {!v.greeting}!</p>
        <lightning:input label="Name" value="{!v.greeting}" />
      </div>
    </lightning:card>
  -->
</template>
```

lwcHello.js

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

export default class LwcHello extends LightningElement {
      @track greeting = "world";

}
```

i. <lightning:card> becomes <lightning-card>:

lwcHello.html

```
<template>
    <lightning-card title="LWC_Hello" icon-name="action:announcement">
      . . .
    </lightning-card>
</template>
```

j. <lightning:button> becomes <lightning-button>.
k. <aura:set> becomes slot=”action”.
l. The syntax of calling the **count** function has changed:

lwcHello.html

```
<template>
    <lightning-card title="LWC_Hello" icon-name="action:announcement">
      <aura:set attribute="actions">
        <lightning-button label="Count" onclick={count} slot="action">
        </lightning-button>
      </aura:set>
    </lightning-card>
      . . .
</template>
```

m. Notice the syntax difference in the way we bind Lightning web components properties:

lwcHello.html

```
<template>
  <lightning-card title="LWC_Hello" icon-name="action:announcement">
    <lightning-button label="Count" onclick={count} slot="actions">
    </lightning-button>
    <div class="slds-m-around_medium">
      <p>Hello, {greeting}!</p>
      . . .
    </div>
  </lightning-card>
</template>
```

n. <lightning:input> becomes <lightning-input>:
o. When an onchange event occurs, we call the handleChange function:

lwcHello.html

```
<template>
  <lightning-card title="Aura_Hello" icon-name="action:announcement">
    <lightning-button label="Count" onclick={count} slot="actions">
    </lightning-button>
    <div class="slds-m-around_medium">
      <p>Hello, {greeting}!</p>
      <lightning-input label="Name" value={greeting} onchange={handleChange}>
      </lightning-input>
    </div>
  </lightning-card>
</template>
```

p. In the lwcHello.js file, we define the count() and handleChange() functions using standard JavaScript:

lwcHello.js

```
import { LightningElement, track } from 'lwc';
export default class LwcHello extends LightningElement {
  @track greeting = "world";
      count() {
            // eslint-disable-next-line no-alert
            alert(this.greeting.length + " letters");
      }
      handleChange(event) {
            this.greeting = event.target.value;
      }
}
```