* More LWC
* Directives
  + Special HTML attributes that act like markers on the DOM that will manipulate it or add data to it
    - If true
    - For each
  + They give us more power to manipulate the DOM in the markup itself
  + Render DOM elements conditionally
    - Also known as conditional directives
    - To render HTML conditionally, add the if:true/if:false directive to a nested <template> tag that encloses the conditional content
    - The if:true|false = {property} directive binds data to the template and removes and inserts DOM elements based on if the data is a truthy or a falsy
      * It can be added to a nested template tag inside of the already existing template tag
      * The root template tag cannot have any attributes on it
        + Instead, nest the template tags
  + Render lists
    - HTML templates also render data to the DOM using simple syntax to declaratively bind a components template to data in the components JS class that can render a list of items to the DOM
    - The for:each directive (also known as the iterator directive) can e used to iterate over an array that is added to a nested template tag tat encloses the HTML elements you want to repeat
    - Regardless of which directive you use, you must use a key directive to assign a unique id to each item
    - You assign this key to the element that we wish to bind the current iteration value to in the nested template
      * Key={uniqueId}
    - The for:each directive
      * When using, use for:item – “currentItem” to set the name to access the current iteration’s item
    - Iterator directive
      * Tp apply a special behabior to the first or last item in a list, iterator:iteratorName={array}
      * This has to be used on a nested template tag
      * Has several properties
        + Iterator name is used to access some properties, like value
  + Render multiple templates
    - You may want to render a component with more than one look or fell but you don’t want to mix he HTML in one file
    - You can do this by importing multiple HTML templates and write business logic that readers them conditionally
    - Create multiple HTML files in the component bundle
      * Import them all and add a condition in the render() method to return the correct template
    - The returned value from the render() must be a template reference, which is the imported default export from the HTML file
* Component life cycle
  + LWC have a life cycle that is managed for us by the framework
  + The framework creates components, inserts them into the DOM, renders them and removes them from the DOM for us
    - It also monitors all of the components for the property changes
  + Removing a component is much shorter than adding one
* Controller constructors
  + The constructor fires when a component instance is created
  + Do not add attributes to the host element during construction
    - Add attributes to the host element in any other lifecycle hook
  + The constructor flows from parent to child
  + The first statement must be super() with no parameters
    - This call establishes the correct protype chain and value for this
* connectedCallback and disconnectedCallback Hooks
  + The connectCallback() lifecycle hook fires when a component is inserted into the DOM
  + The disconnectedCallback() fires when a component is removed from the DOM
  + Use the connectedCallback() to interact with a components environment
    - To establish communication with the current document or container
    - Coordinate behavior with the environment
    - Perform init tasks
      * Like fetching data you want to use
    - Setting up caches
    - Listen for some form of event
    - Subscribe/unsubscribe from a message channel
    - Etc
  + The connectedCallback() hook is invoked with the initial properties passed to a component
  + disconnectedCallback() to clean up work done in the connectedCallback()
  + The renderCallback() hook us unique to LWC
    - It is used to perform logic after a component has finished the rendering phase
    - This hook flows from child to parent
    - After a component is connected and rendered, a change to the components state marks the component as dirty and enqueues a microtask to rerender the component
  + Reactivity
    - The system at the core of the LWC framework
    - The framework itself observes changes to the values of fields and properties
    - When it sees a change, it reacts to it
      * It reevalues all the expressions used in the template and rerenders the component to display the new values
    - Field and properties are almost interchangeable terms
      * A component author declares files in a class
      * An instance of the class has properties
    - In LWC only fields that a component author decorates with @api are publicly available to consumers as object properties
  + A component is usually rendered many times during its lifecycle
    - To use this hook to perform one-time operations, use a Boolean field to track weather renderedCallback() has been executed
  + It is best to add event listeners declaratively in an HTML template
    - However if you do need to add one programmatically, do it in the renderedCallback() function
  + When a template is rerendered, the LWC engine attempts to reuse existing elements
    - Elements created using the for:each directive
    - The decision to use the iteration elements depends on the key attributes
    - If the key changes, the element may be rerendered
      * If it doesn’t, the element isn’t rerendered
    - Elements received as slot:content, the engine attempts to reuse the element in a slot tag
    - If you update the state of your component in the renderedCallback() function, this can cause an infinite loop
  + The errorCallback hook
    - Unique to LWC
    - Implement it to create an error boundary component that captures errors in all the descendent components in its tree
    - It captures errors that occur in the lifecycle hooks or during an event handler declared in an HTML element
    - You can code the error boundary component to log stack information abd render an alternative view to tell users what happened
* Decorators
  + LWC has 3 decorators that add additional functionality to a property or a function
  + Not exclusive to JavaScript
  + @api
    - Used to expose a public property
    - Decorate a filed with @api
    - Public properties help define the API for the component
    - The public properties in our template are reactive
    - The expressions used in the template are reevaluated and the renderedCallback() lifecycle hook is executed
    - You can import the @api decorator from the LWC module
    - We can expose methods/functions as public so that out parent components can see the child methods
      * To do this we decorate the function with @api
      * Public methods become part of the component’s API
      * To communicate down the hierarchy, parent components can call JS methods on child functions
  + @track
    - If a field’s value changes and the field is used in a template or in a getter of a property that is used in a template, the component rerenders and displays the new value
    - If a field is assigned to an object or an array, the framework observes some changes to the internal of that object or array such as when you assign a new value
    - To tell the framework to observe changes to the properties of an object, decorate the field with @track so that the system will track that change
  + @wire
    - Is the reactive wire service in the LWC
    - This allows us to read salesforce data
    - Used to specify a wire adaptor or apex method
* XML Configuration Files
  + Each LWC folder must have a configuration file name after it
  + It defines the metadata values for the component
  + It can include targets and design the configuration for the lightning app builder and experience builder
  + Targets tell where LWC can be used
    - This is similar to the way aura was available for record page/ home page etc