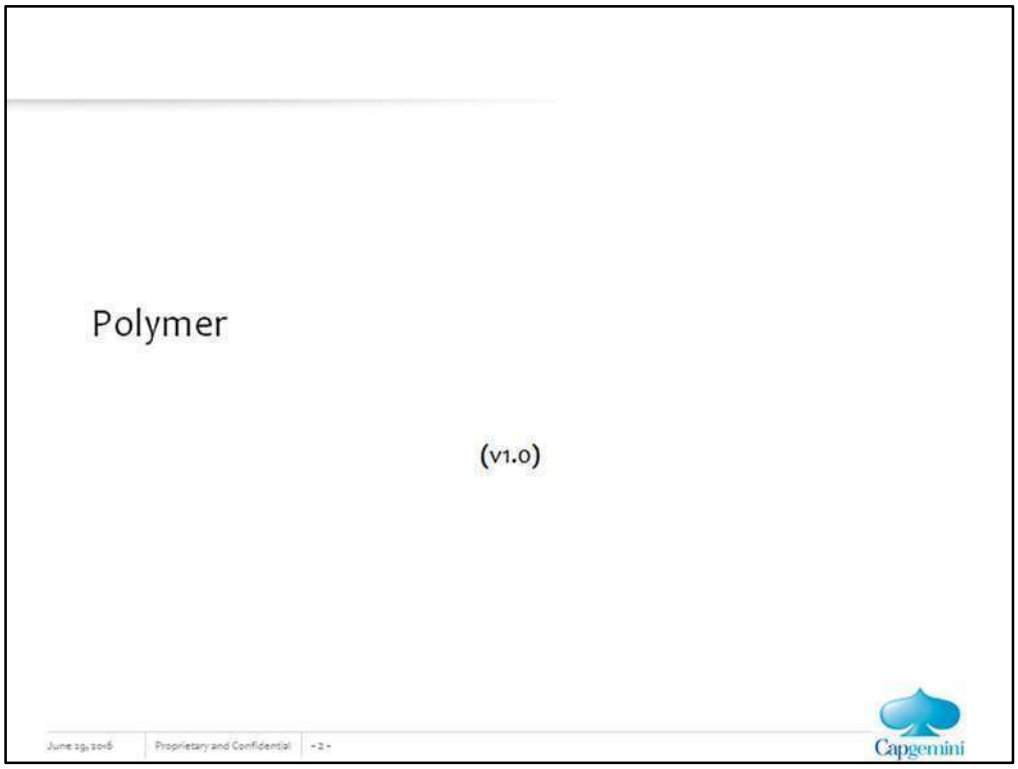
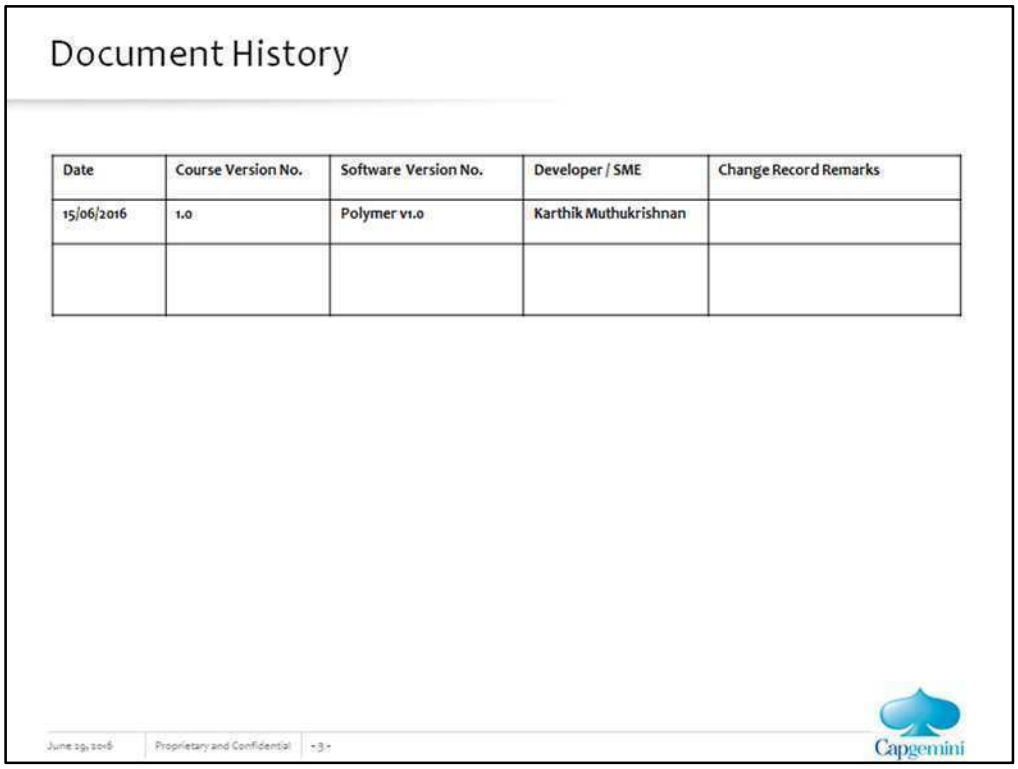
Instructor Notes:

Instructor Notes:



Instructor Notes:

Add instructor notes here.



Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
15/06/2016	1.0	Polymer v1.0	Karthik Muthukrishnan	

June 29, 2016 | Proprietary and Confidential | - 3 -

Capgemini

Instructor Notes:

Add instructor notes here.

Course Goals and Non Goals

- **Course Goals**
  - Understand Web Components fundamentals
  - Understanding the Polymer library fundamentals
  - Building responsive web applications by componentizing app

- **Course Non Goals**
  - Using Polymer in Node Environment
  - Unit Testing
  - Deploying Polymer web applications

Capgemini

Instructor Notes:

Add instructor notes here.

Instructor Notes:

Add instructor notes here.

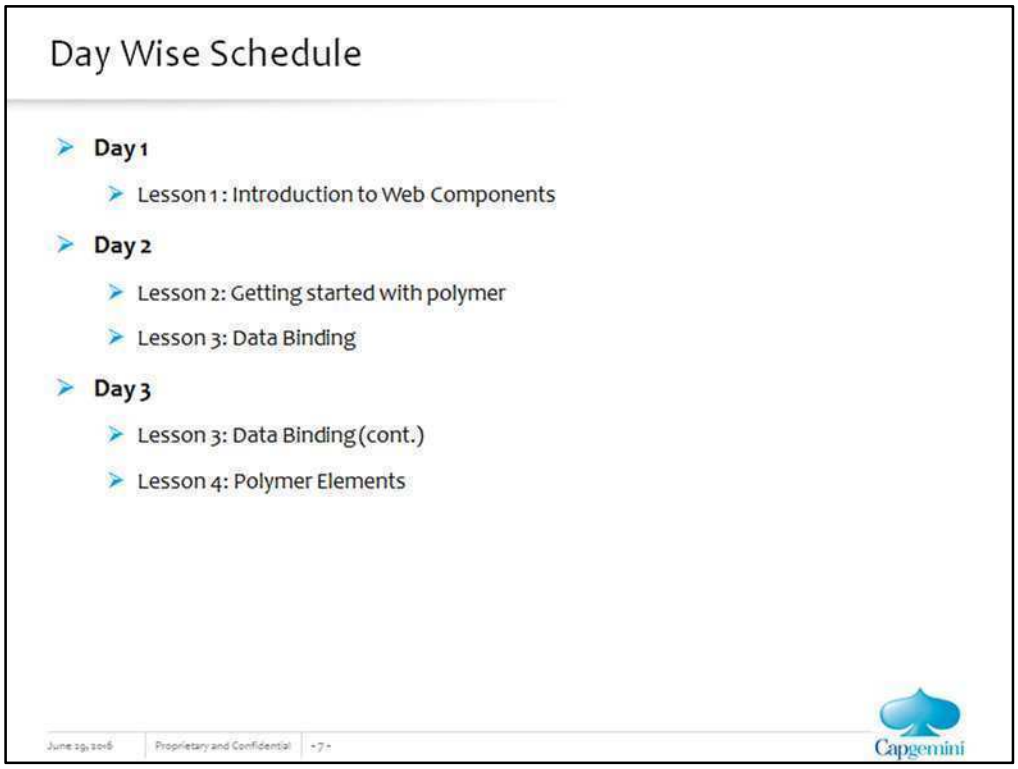
Intended Audience

- **Web application developers**

Capgemini

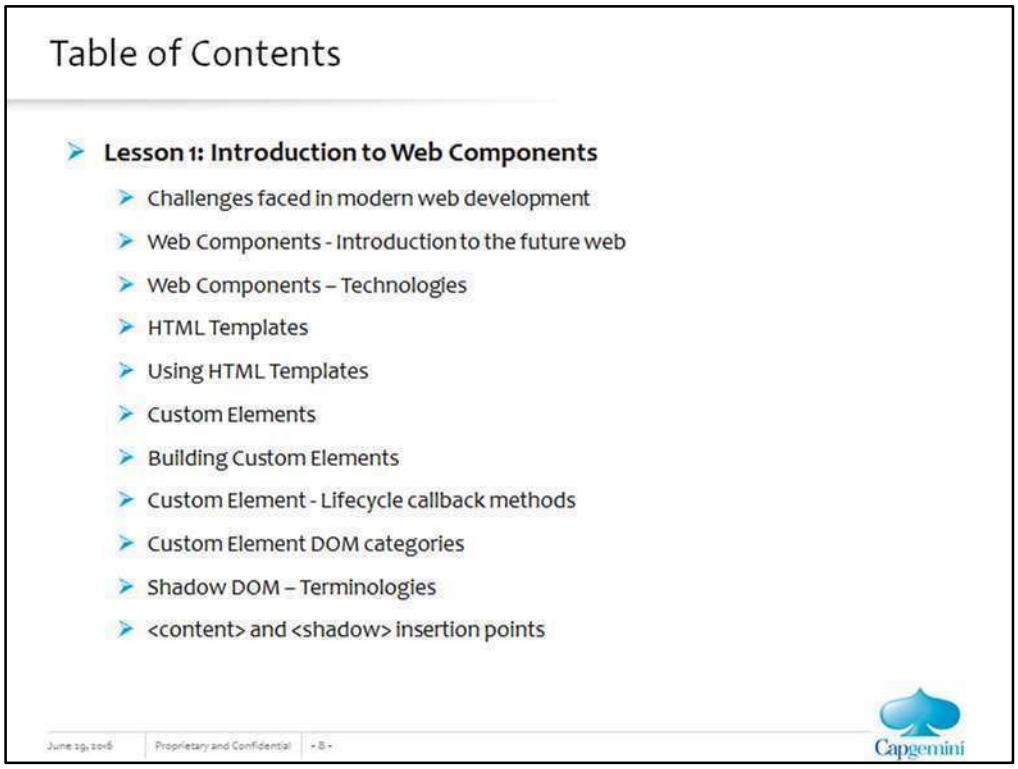
Instructor Notes:

Add instructor notes here.



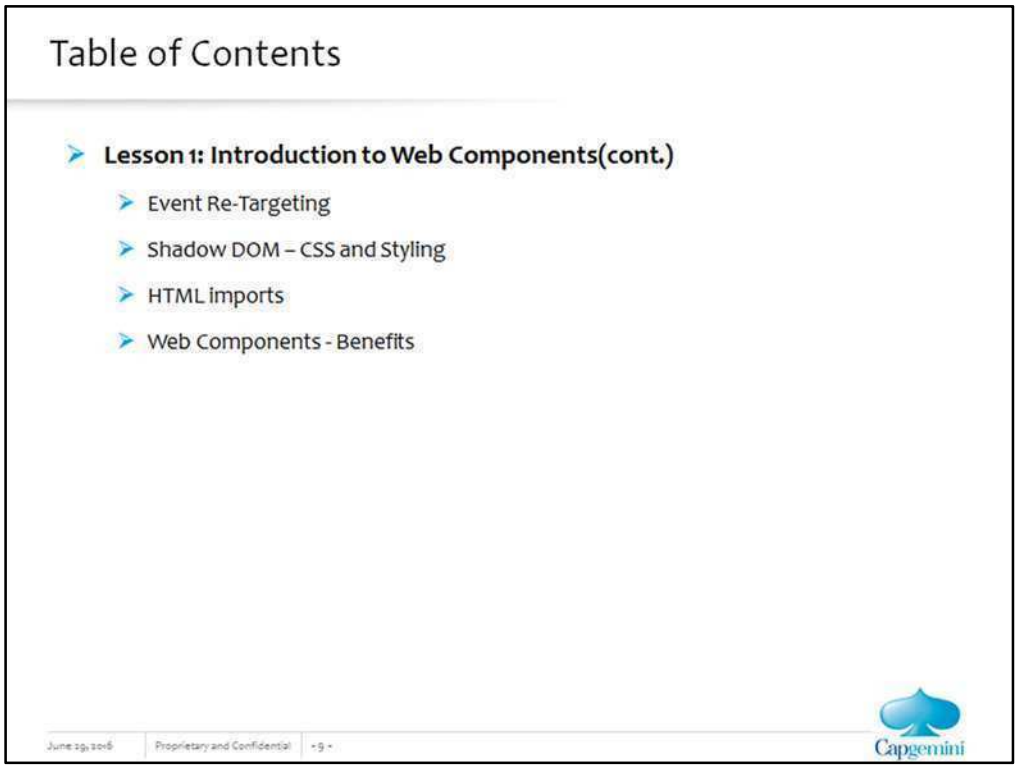
Instructor Notes:

Add instructor notes here.



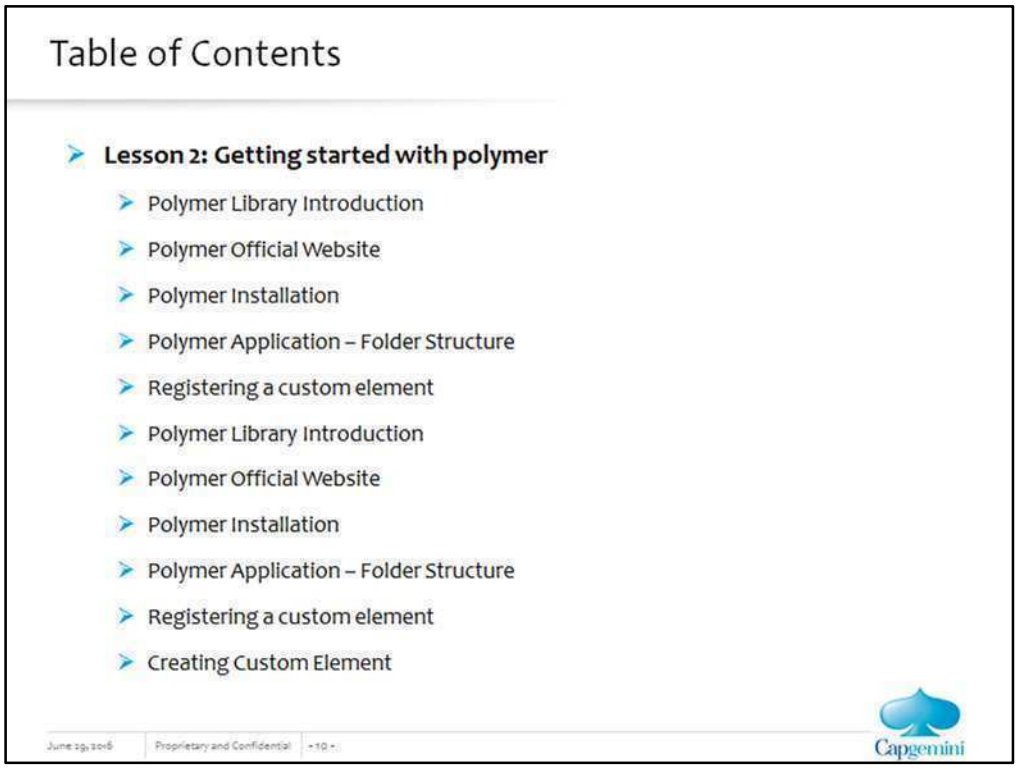
Instructor Notes:

Add instructor notes here.



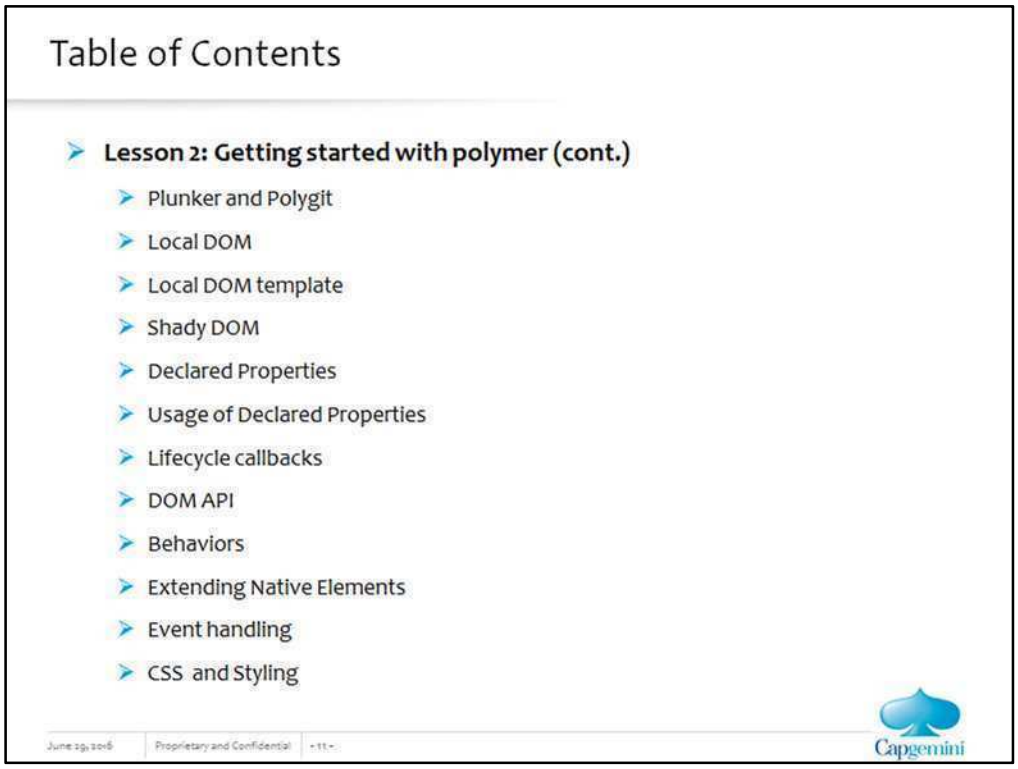
Instructor Notes:

Add instructor notes here.



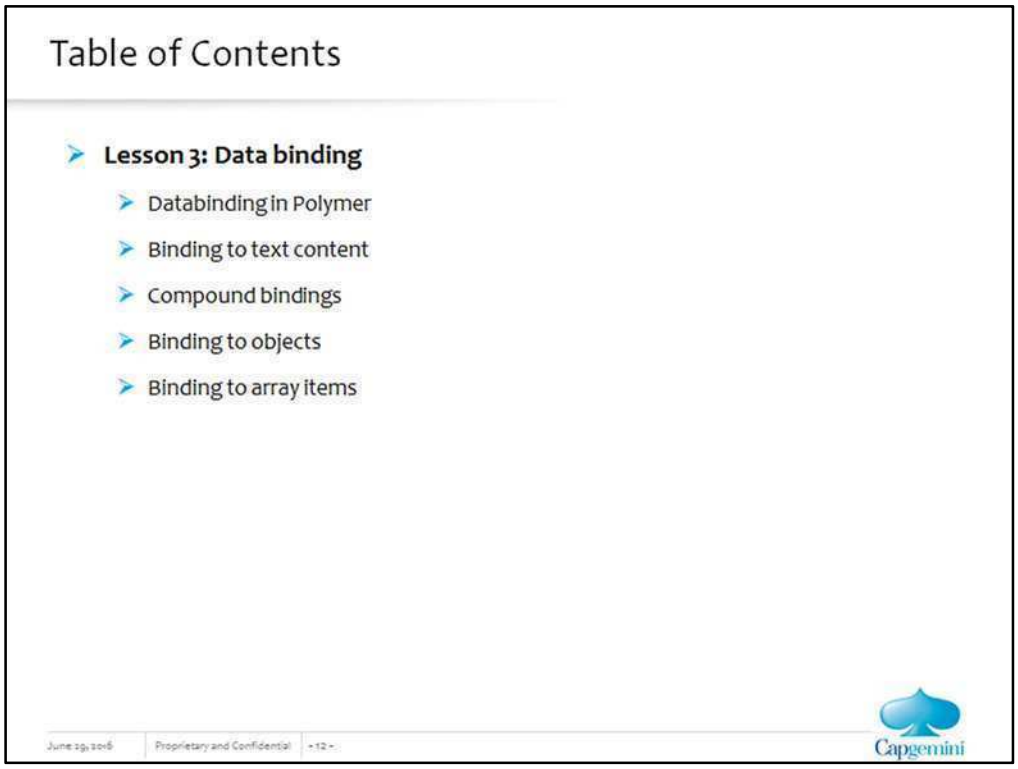
Instructor Notes:

Add instructor notes here.



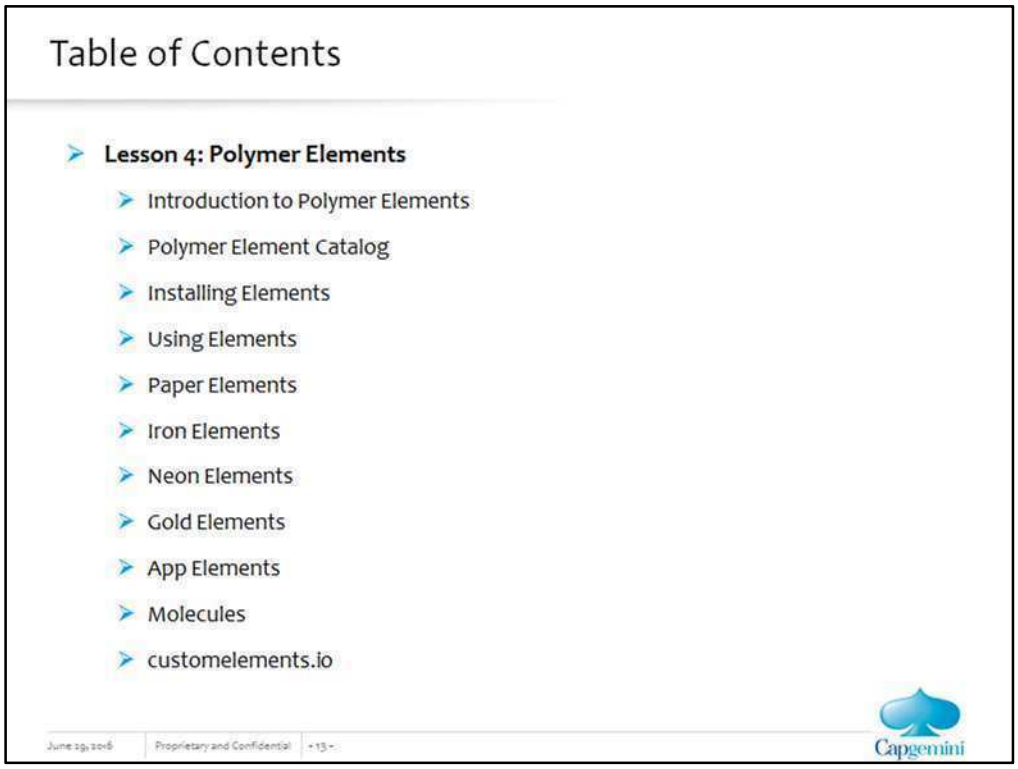
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

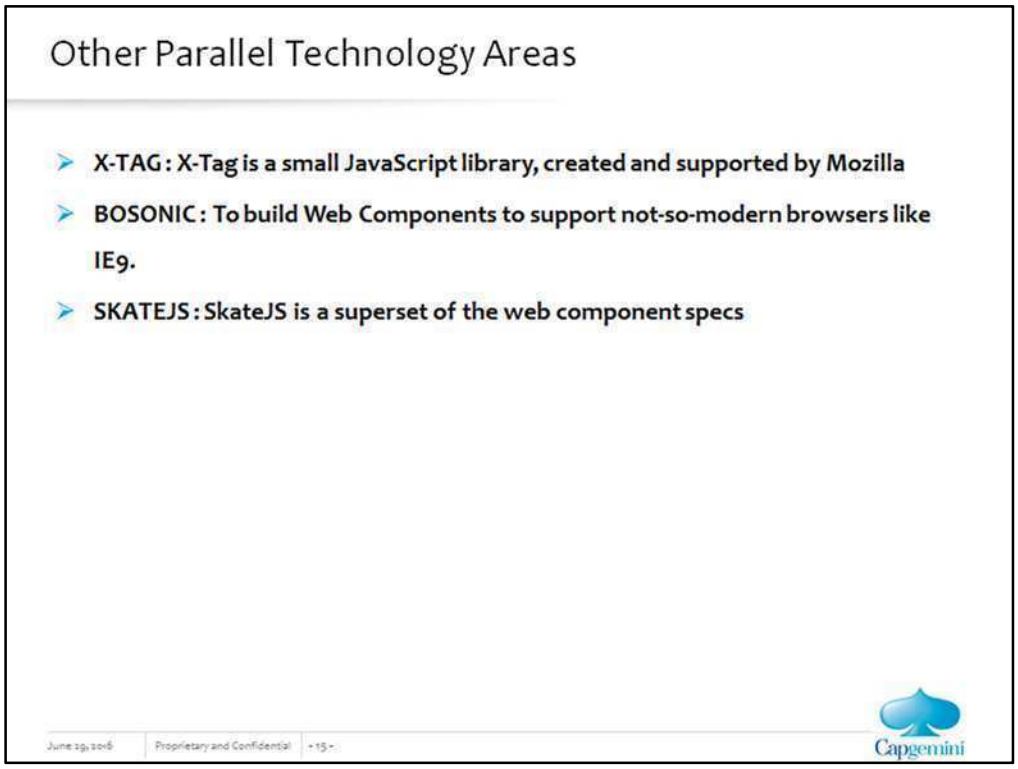
References

- www.html5rocks.com
- https://www.polymer-project.org/1.0/

Capgemini

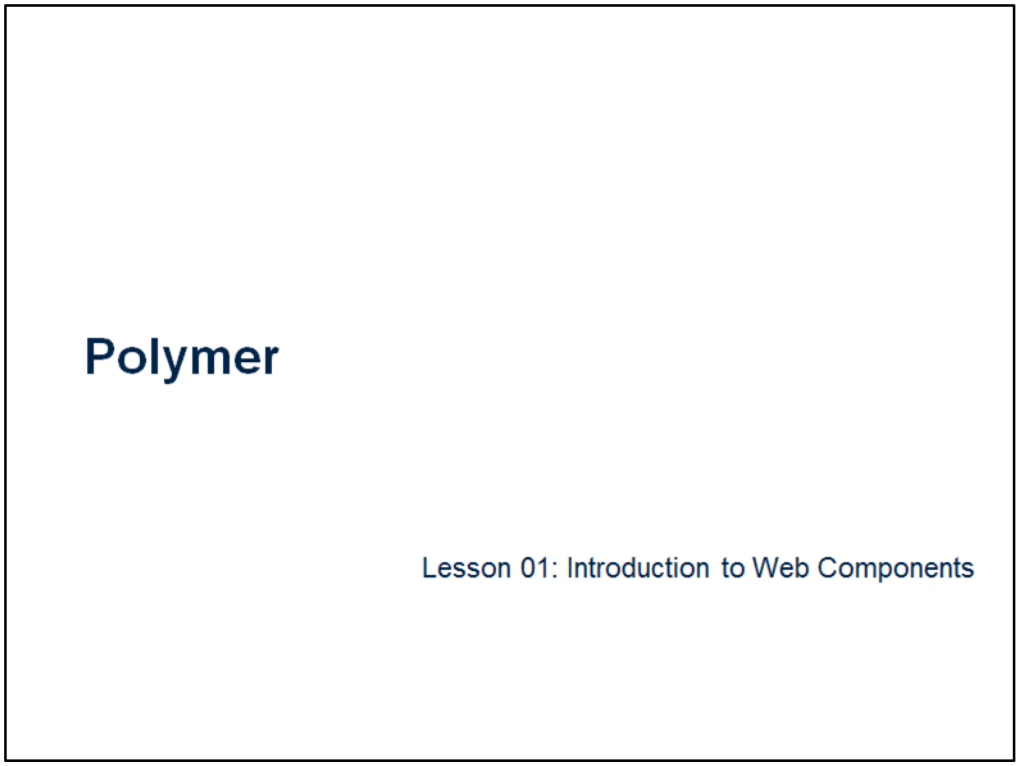
Instructor Notes:

Add instructor notes here.



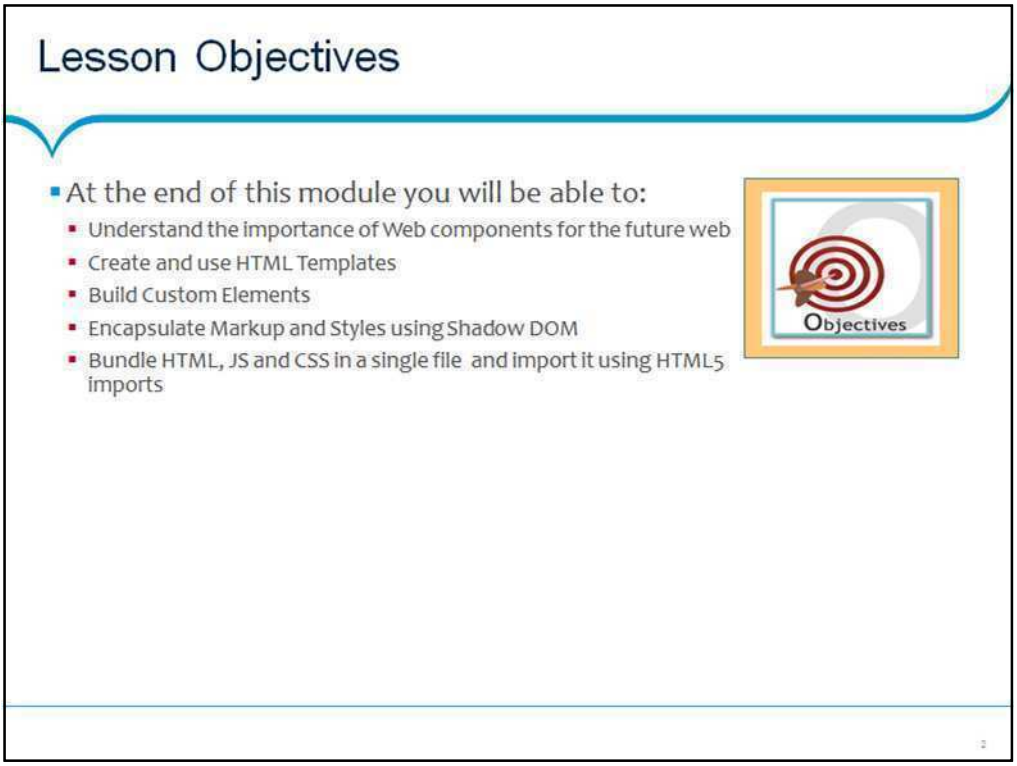
Instructor Notes:

Add instructor notes here.



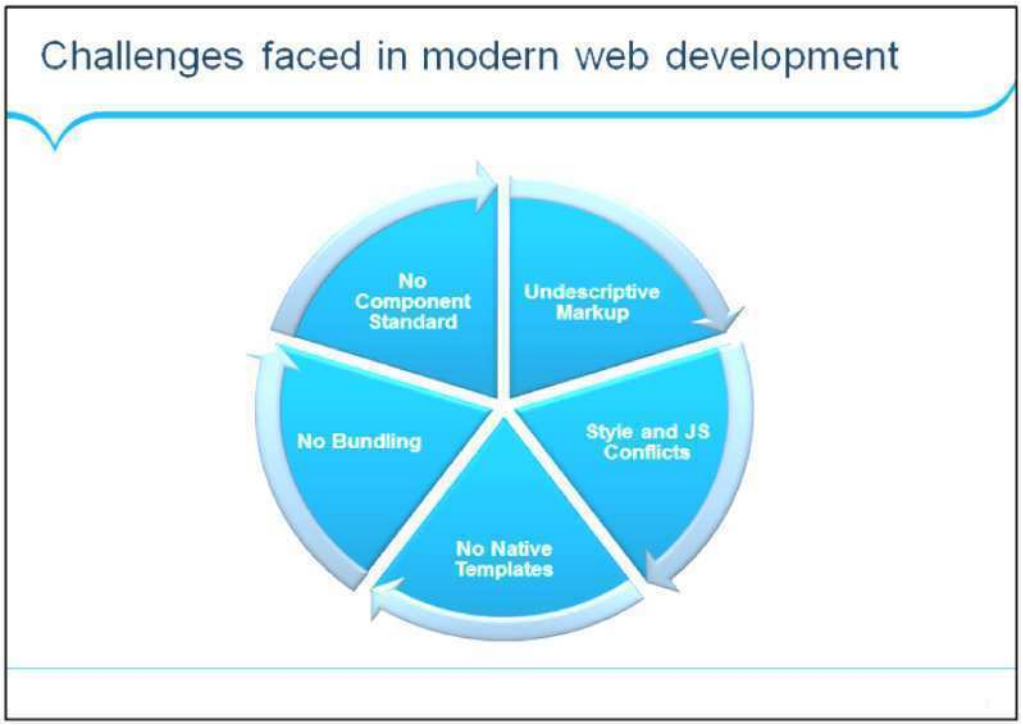
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Undescriptive Markup

It's hard to read and navigate hierarchy. Div tags are generic containers that doesn't convey the nature of the data inside. Using HTML 5 Semantics may helps us to convey the meaning of the content.

Style and JS Conflicts

Need to write complex, prefixed CSS selectors to avoid conflicts. !important need to be used to force styles, but still there is not guarantee that another style won't conflict.

```
#splId {
  color:red;
}
.splClass{
  color:blue!important;/*will have high importance than id selector*/
}
<div id="splId" class="splClass">Hi</div>
```

No markup encapsulation i.e. JavaScript can also be accidentally manipulate as well.

No Native Templates

There is no natural way to store HTML for templates. Approaches followed now create confusion, lead to XSS vulnerabilities, or aren't truly inert. For instance handlebars deliver a template to the browser by including **type="text/x-handlebars-template"** it in a <script> tag.

No Bundling

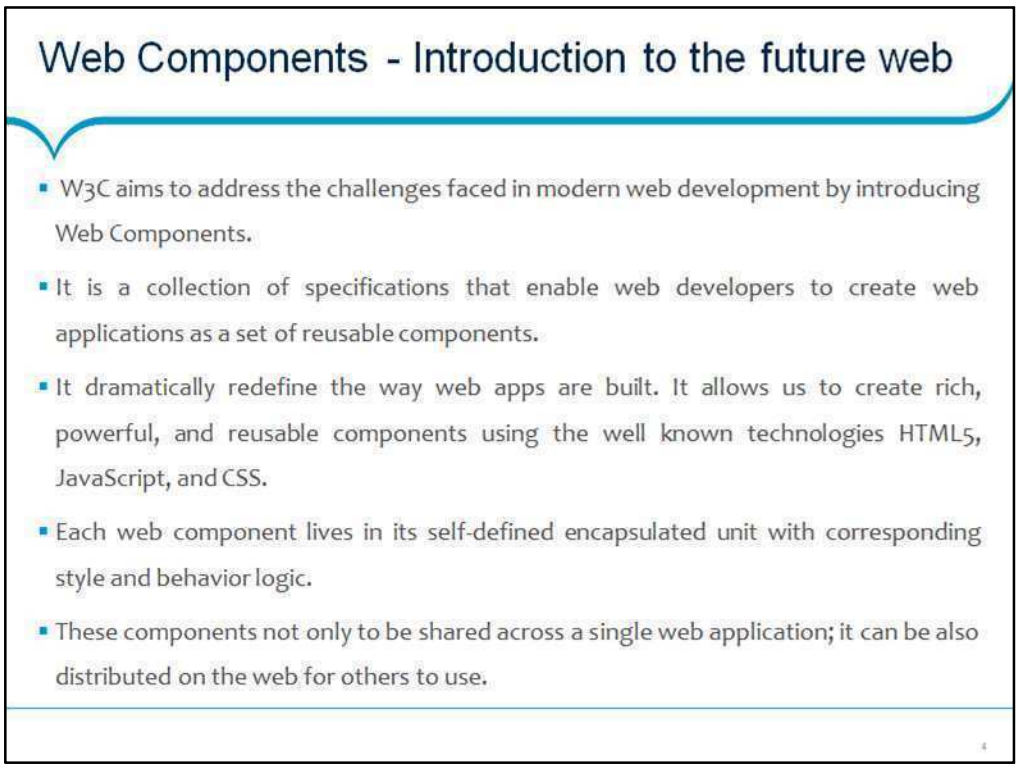
There's no simple way to import a component with a single line of code. Need to reference the necessary JS and CSS in the right order and add proprietary markup.

No Component Standard

Switching between today's proprietary component solutions is painful. jQueryUI, KendoUI, Angular, etc. all have their own style.

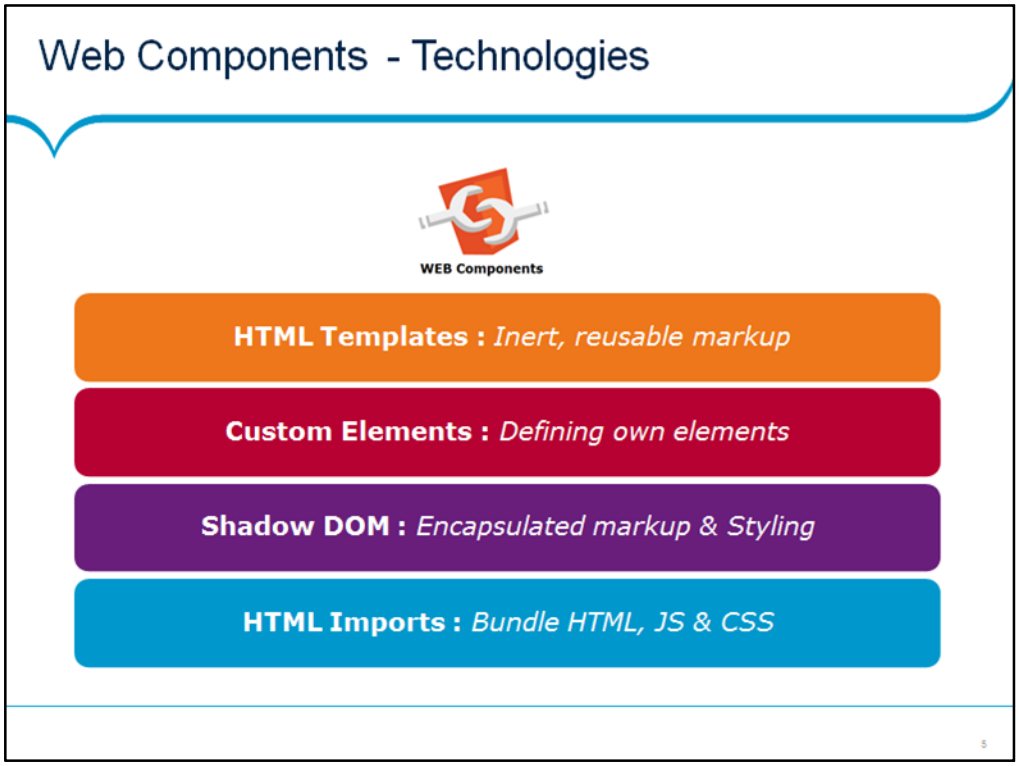
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Web Components consists of four technologies : Custom Elements, HTML Templates Shadow DOM and HTML Imports.

Custom Elements : Enable web developers to create their own elements that are relevant to their design as part of the DOM structure with the ability to style/script them like any other HTML tag.

HTML Templates : Allows to define fragments of markup that stay consistent across web pages with the ability to inject dynamic content using JavaScript.

Shadow DOM : Abstract all the complexities from the markup by defining functional boundaries between the DOM tree and the subtrees hidden behind a shadow root.

HTML Imports : Similar to importing CSS files; It allows to include and reuse HTML documents in other HTML documents.

Each of these pieces is useful individually. But when combined as a whole package gives:

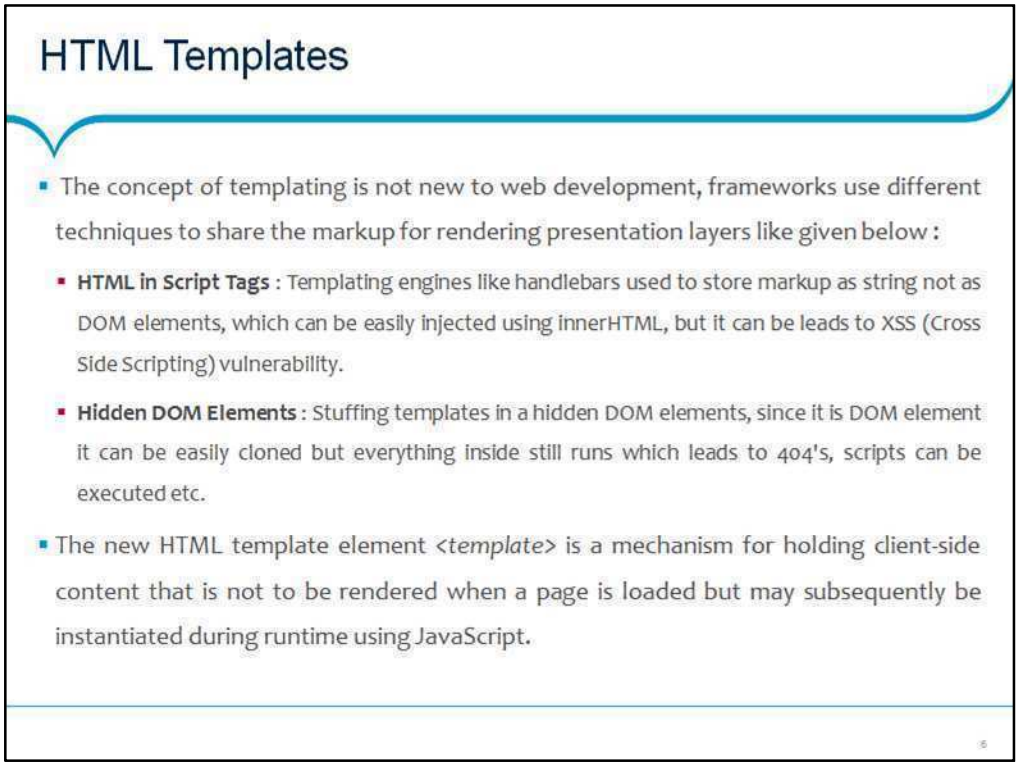
Composability : Being able to create whole sites and apps by putting different elements together.

Encapsulation : Isolating markup, style, and behavior logic so they don't leak into the rest of the page.

Reusability : Extending existing elements to create new elements, allowing you to stop reinventing the wheel.

Instructor Notes:

Add instructor notes here.



Handlebars delivers a template to the browser by including it in a <script> tag.

```
<script id="entry-template" type="text/x-handlebars-template">
 <div class="entry">
  <h1>{{title}}</h1>
  <div class="body">
   {{body}}
  </div>
 </div>
</script>
```

Template Characteristics :

Inert : Does nothing until cloned

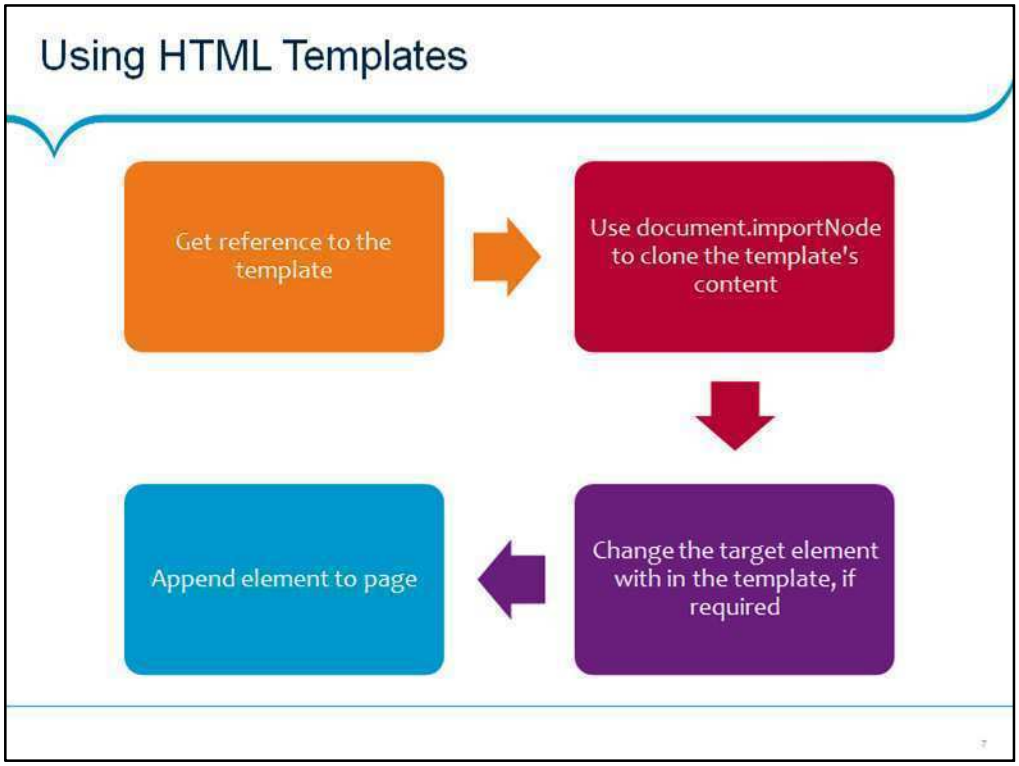
Hidden from Selectors: We cannot locate the elements placed with in the template tag using document.getElementById

Flexible placement : Template tag can be placed anywhere on the page <head>, <body> etc.

```
<template>
  <p id="para">Paragraph element inside the template</p>
  <script> alert("Won't be executed"); </script>
</template>
```

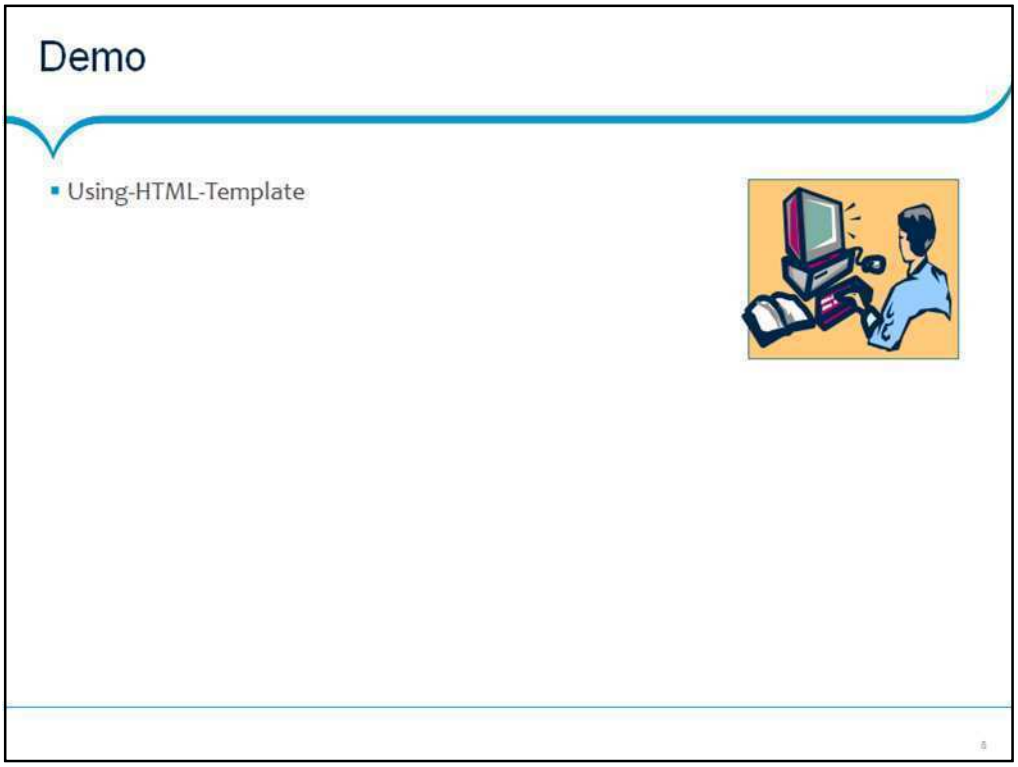
Instructor Notes:

Add instructor notes here.



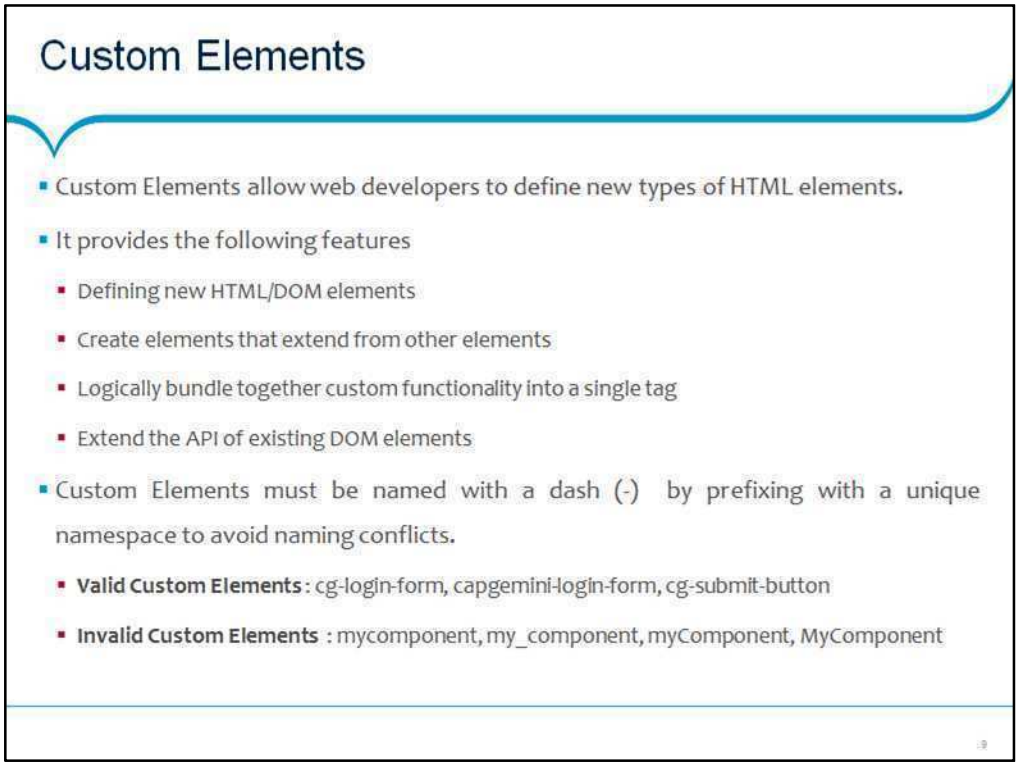
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

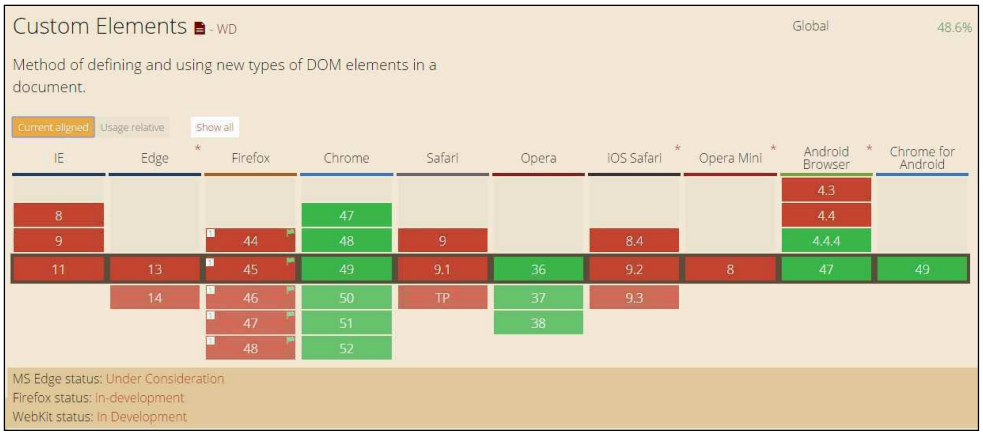


Custom Elements enable developers to create their own custom HTML tags and allow them to use those tags in their sites and apps, and enable easier component reuse.

Naming rules :

It should have at least one '-' inside the name of a custom element. Before the dash (-) prefix custom element with Company Name, Application Name, Brand Name which makes the developer to understand the usage of Custom element and avoid naming conflicts.

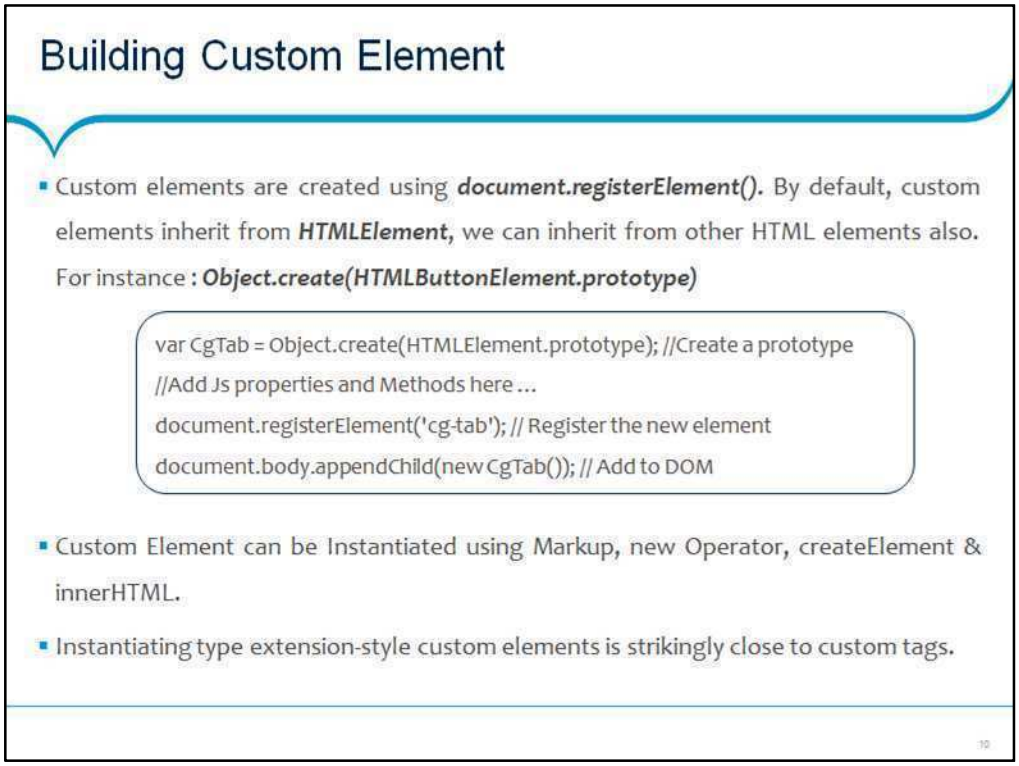
Supported browsers : At present, **c**ustom Elements are supported by Chrome, Firefox and Android browsers. For polyfilling other browsers, we can use ***webcomponents.js***



Source : http://caniuse.com/

Instructor Notes:

Add instructor notes here.



Creating Custom element and adding JS methods and Properties to it.

```
var XFooProto = Object.create(HTMLElement.prototype);
// 1. Give x-foo a foo() method.
XFooProto.foo = function() {    alert('foo() called');  }

// 2. Define a property read-only "bar".
Object.defineProperty(XFooProto, "bar", {value: 5});

// 3. Register x-foo's definition.
var XFoo = document.registerElement('x-foo', {prototype: XFooProto});

// 4. Instantiate an x-foo.
var xfoo = document.createElement('x-foo');

// 5. Add it to the page.
document.body.appendChild(xfoo);
```

Instantiating Custom Element

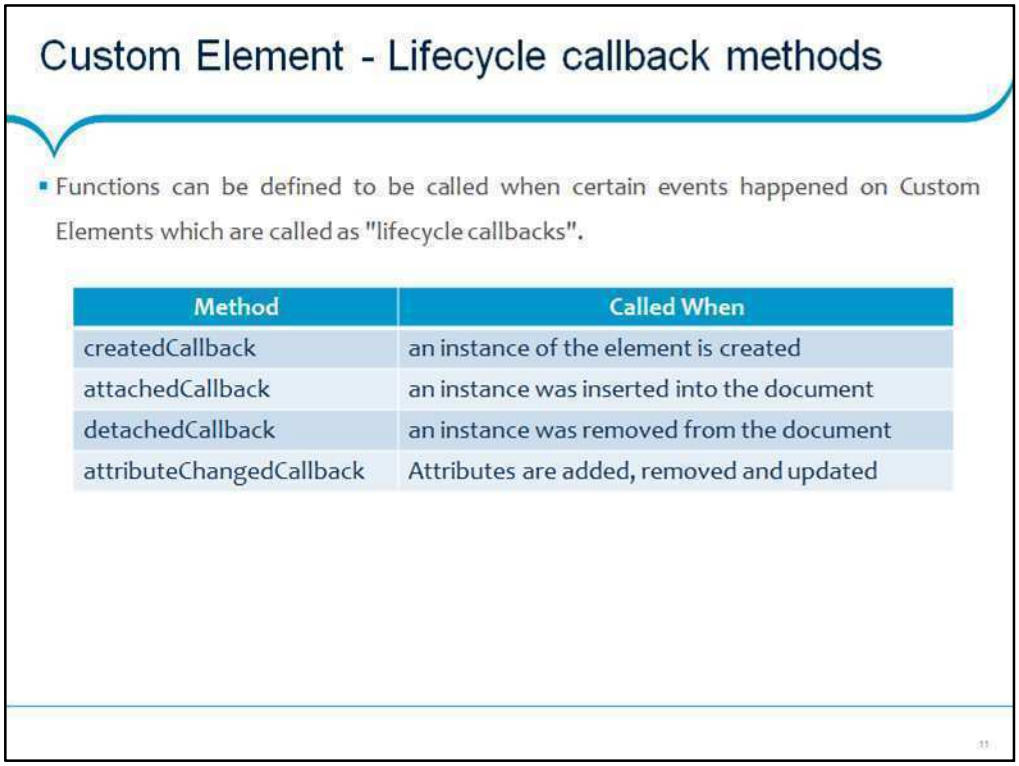
1. **Markup** : <x-foo>
2. **new Operator** : var el = new Xfoo();
3. **createElement** : var el = document.createElement('x-foo');
4. **innerHTML** : el.innerHTML = '<x-foo />'

Instantiating Extended Element

1. **Markup :** <button is="spl-button"/>
2. **JavaScript :** var button = document.createElement('button','spl-button');
3. **New Operator :** document.body.appendChild(new SplButton());

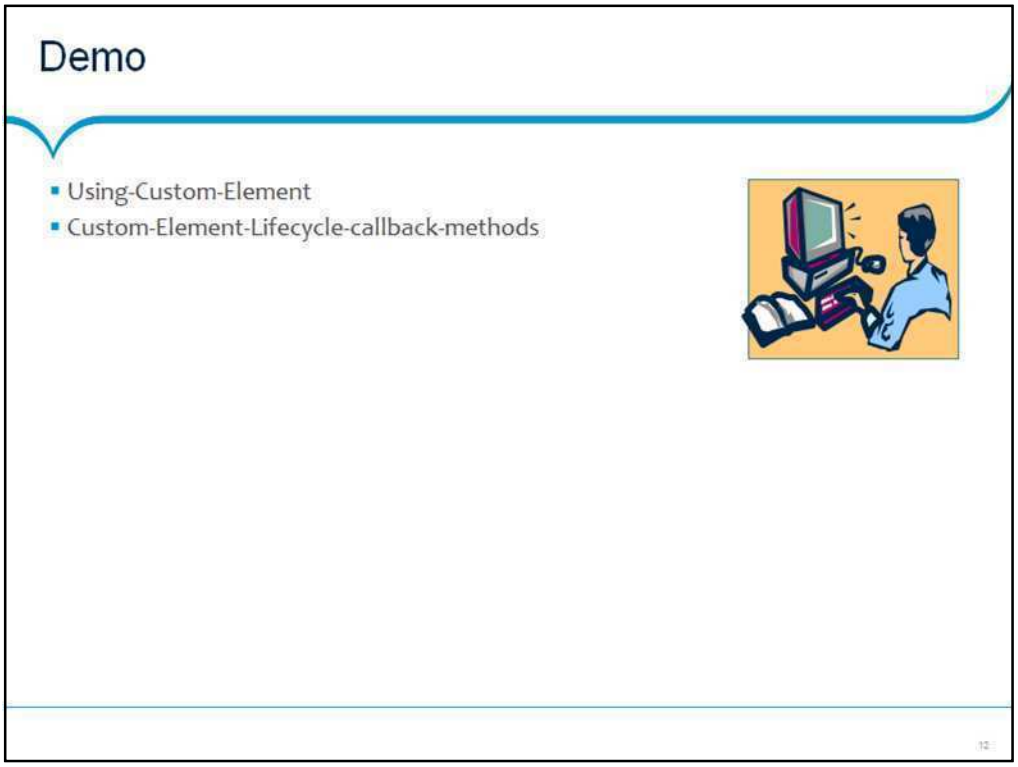
Instructor Notes:

Add instructor notes here.

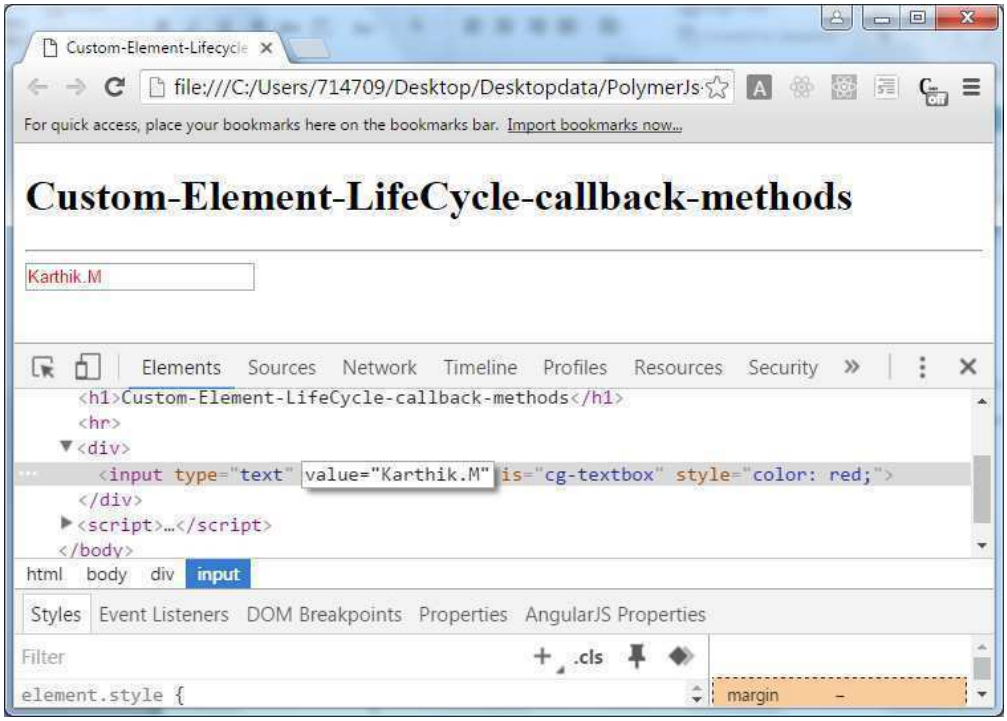


Instructor Notes:

Add instructor notes here.

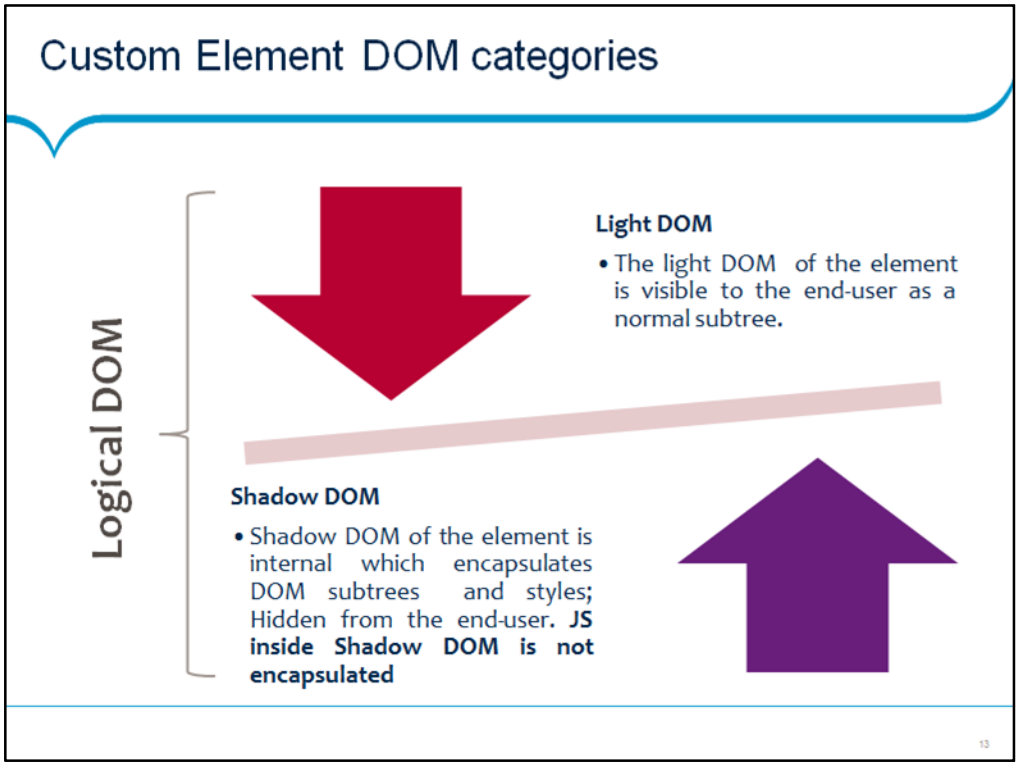


Changing the value for the value attribute using Chrome Developer Tool



Instructor Notes:

Add instructor notes here.



Light DOM

Custom element supplies the light DOM. The light DOM of <my-custom-element> is visible to the end-user of the element as a normal subtree. They can access .childNodes, .children, .innerHTML, or any other property or method that gives information about a node's subtree.

```
<my-custom-element>
  <q>Hello World</q> <!-- part of my-custom-element's light DOM -->
</my-custom-element>
```

Shadow DOM

<my-custom-element> may define shadow DOM by attaching a shadow root to itself. Shadow DOM is internal to the element and hidden from the end-user. Its nodes are not children of <my-custom-element>

```
#shadow-root
  <!-- everything in here is my-custom-element's shadow DOM -->
  <span>People say: <content></content></span>
  <footer>sometimes</footer>
```

Together, the light DOM and shadow DOM are referred to as the logical DOM. This is the DOM that the developer interacts with. The composed DOM is what the browser sees and uses to render the pixels on the screen.

Instructor Notes:

Add instructor notes here.

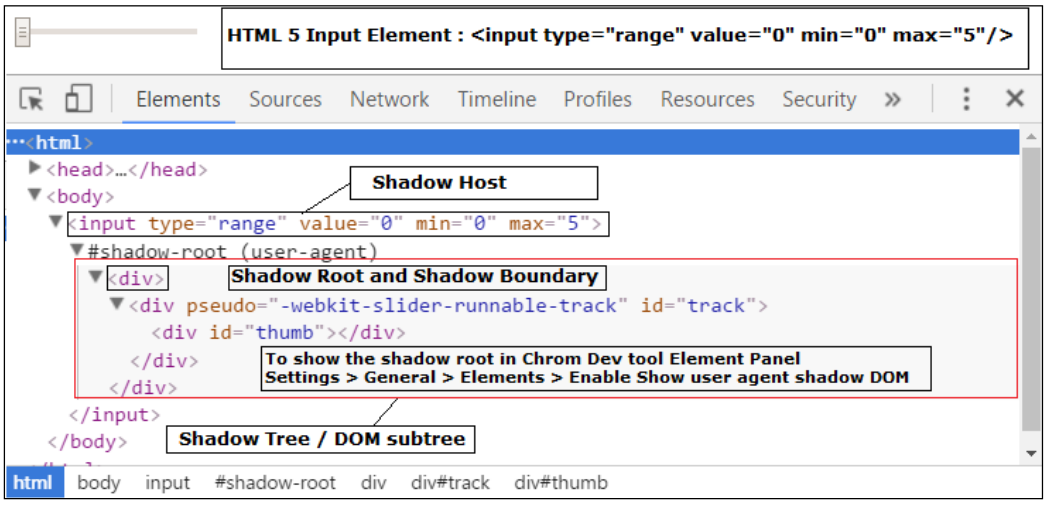
Shadow DOM - Terminologies

- **Shadow DOM**
  - Shadow DOM addresses the lack of true DOM tree encapsulation when building components. It allows multiple DOM subtrees to be composed into one larger tree.
- **Shadow Host**
  - DOM element which is hosting the Shadow DOM subtree or it is the DOM node which contains the Shadow Root.
- **Shadow Root**
  - Root of the DOM subtree containing the shadow DOM nodes. It is a special node, which creates the boundary between the normal DOM nodes and the Shadow DOM nodes. It is this boundary, which encapsulates the Shadow DOM nodes from any HTML or CSS code on the consuming page.

Shadow Boundary :
A Barrier which separates Light DOM from the Shadow DOM. It Encapsulates DOM sub tree. It enhances the readability and avoids accidental styling.

CSS Selectors inside a shadow root only match / style the Shadow DOM.

JavaScript in the Shadow DOM is not encapsulated. We need to use module patterns if encapsulation required.



Instructor Notes:

Add instructor notes here.

<content> and <shadow> insertion points

- <content></content> : Serves as a placeholder for the host content in shadow DOM.
- <shadow></shadow> : They are hosts for other shadow trees.

```
<div id="shadowHost">Light DOM</div>
<script>
var container = document.querySelector('# shadowHost ');
var root1 = container.createShadowRoot();
var root2 = container.createShadowRoot();
root1.innerHTML = '<div>Root 1</div><content></content>';
root2.innerHTML = '<div>Root 2</div><shadow></shadow>';
</script>

Output :
ROOT 2
ROOT1
Light DOM
```

Greedy Insertion Points : Below mentioned <content> placeholders are considered a wildcard selection and it will grab any content in the shadow host that is left over

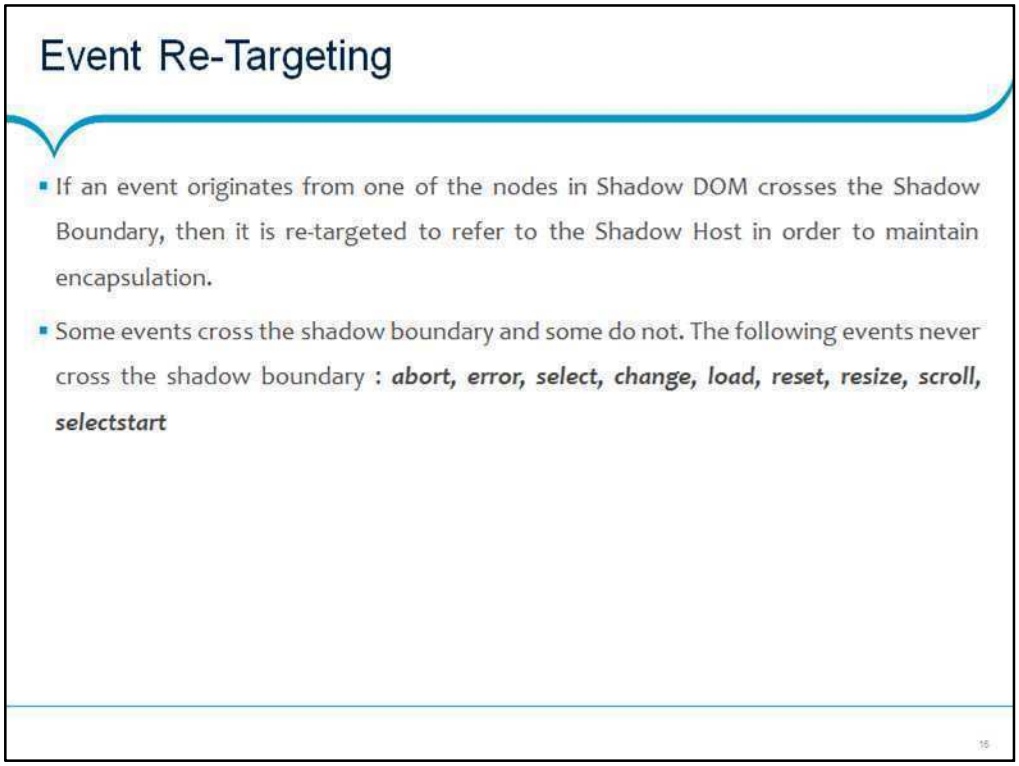
```
<content></content>
<conent select=""></conent>
<content select="*"></content>
```

Shadow trees added to a host are stacked in the order they're added, starting with the most recent first. The last one added is the one that renders.

If multiple <shadow> insertion points exist in a shadow tree, only the first is recognized. The rest are ignored.

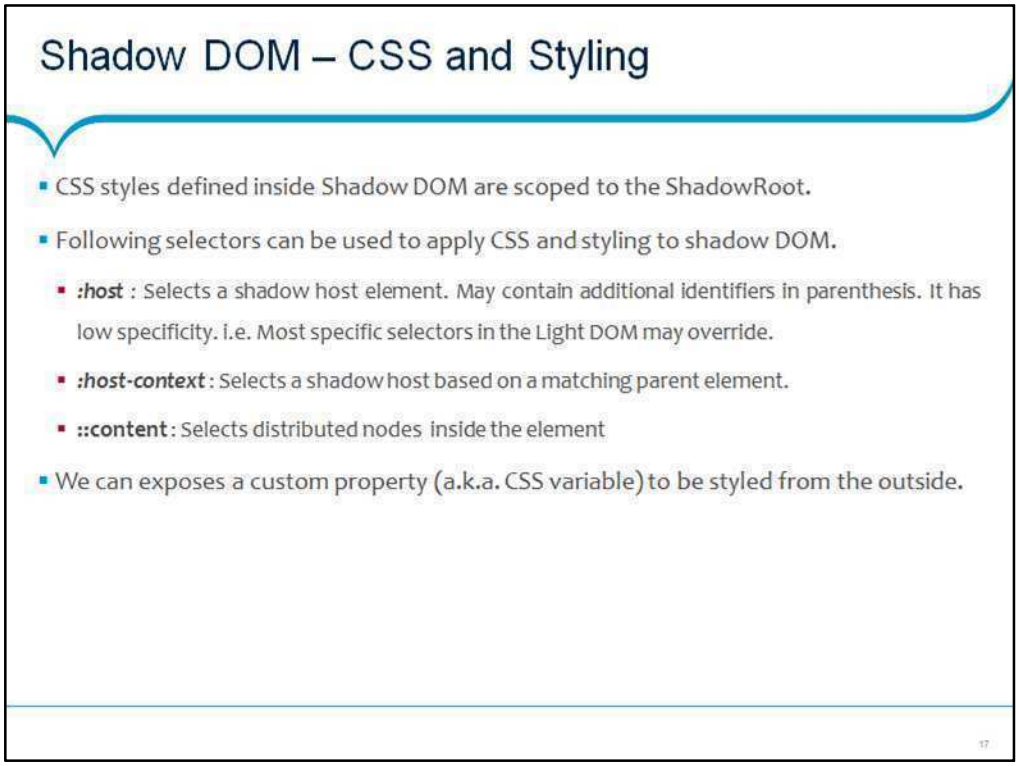
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



CSS Specificity Rules:

Traditional CSS specificity rules apply for :host

From most to least specific:

1.	Inline Style	<p style="color:red">
2.	ID	#target{color:red}
3.	Class, pseudo-class, attribute	.spl{ color : red}
4.	Elements	p {color: red}

Instructor Notes:

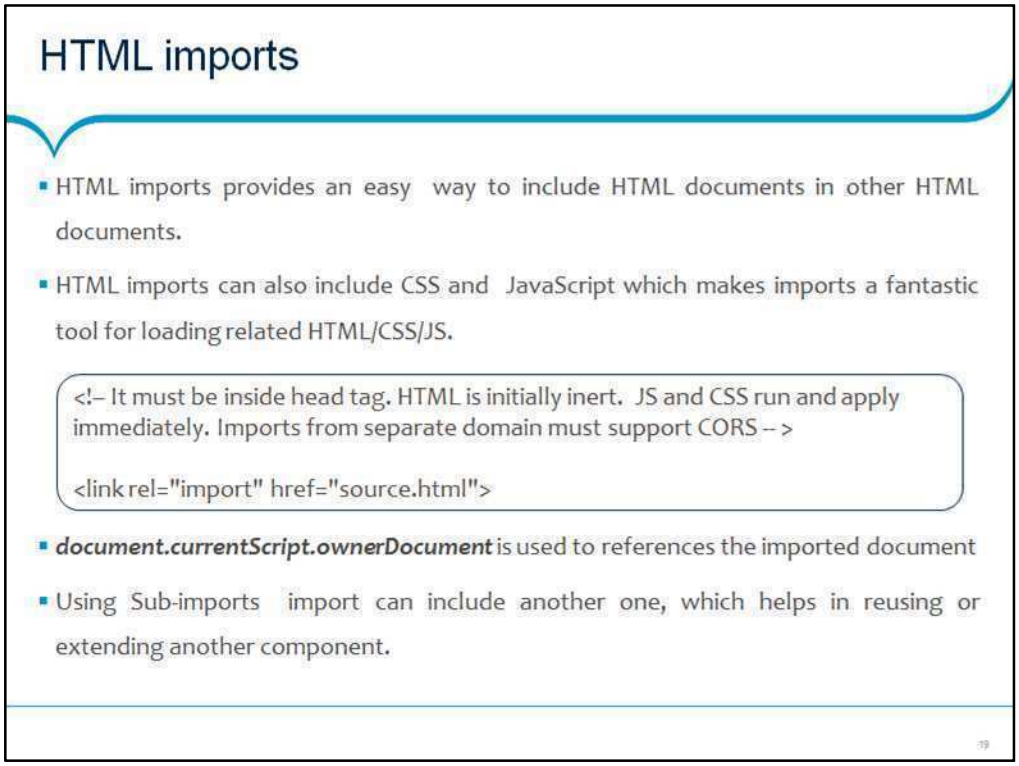
Add instructor notes here.

Demo

- ShadowDOM
- InsertionPoints
- Event-Re-Targeting
- ShadowDOM-Styling

Instructor Notes:

Add instructor notes here.



HTML Imports allow bundling HTML/CSS/JS as a single resource

An import's mimetype is text/html.

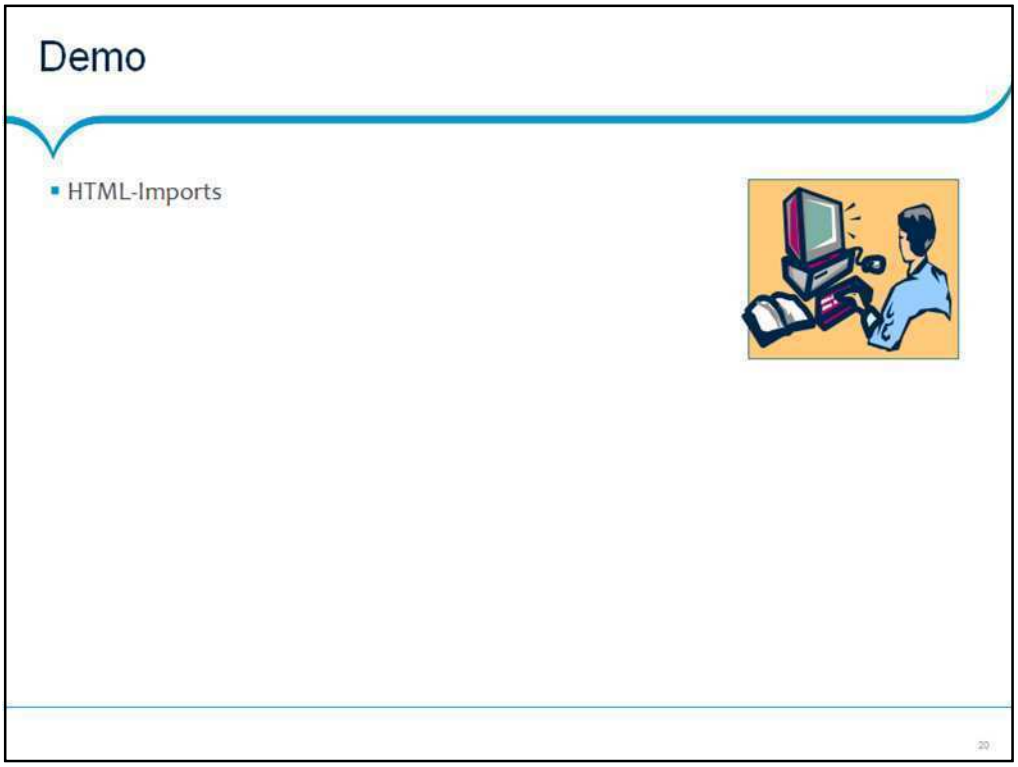
Resources from other origins need to be CORS-enabled.

Imports from the same URL are retrieved and parsed once. That means script in an import is only executed the first time the import is seen.

Scripts in an import are processed in order, but do not block the main document parsing.

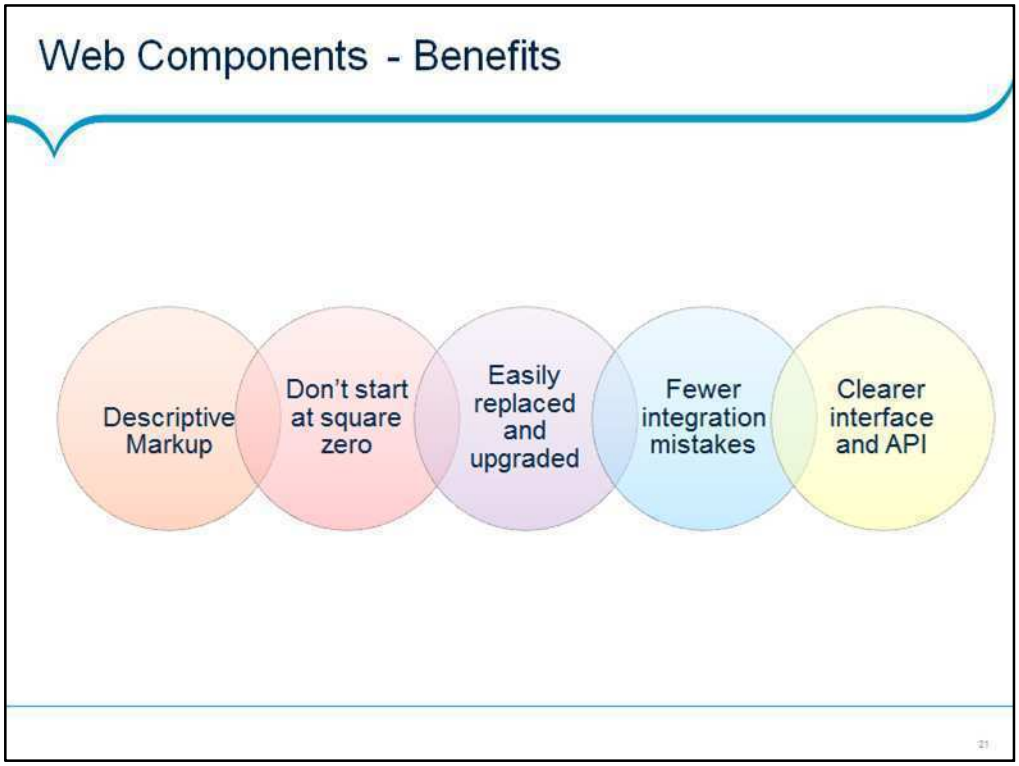
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Descriptive markup : Web components makes the markup easier to read, navigate and understand. It helps to replace the generic tags like div and span tags with higher level and more descriptive markup, which can be navigate and maintained easily.

Don't start at square zero : No need to start from the beginning on each new project, we can easily leverage the components created in the previous projects or use the components created by the community.

Easily replaced and upgraded : Components can be easily replaced and upgraded when bugs have been found; since it is properly bundled, encapsulated and centralized.

Fewer Integration mistakes : HTML imports makes it very easy to import bundle of dependencies, which leads to have less integration problems in projects. Just we need to add single line at the top of file and get everything we need to run the component.

Clearer interface and API : API for interacting with the components will be much clear than the current frameworks. Components will feel native , which greatly reduces the learning curve when using the new components created by other developers.

Instructor Notes:

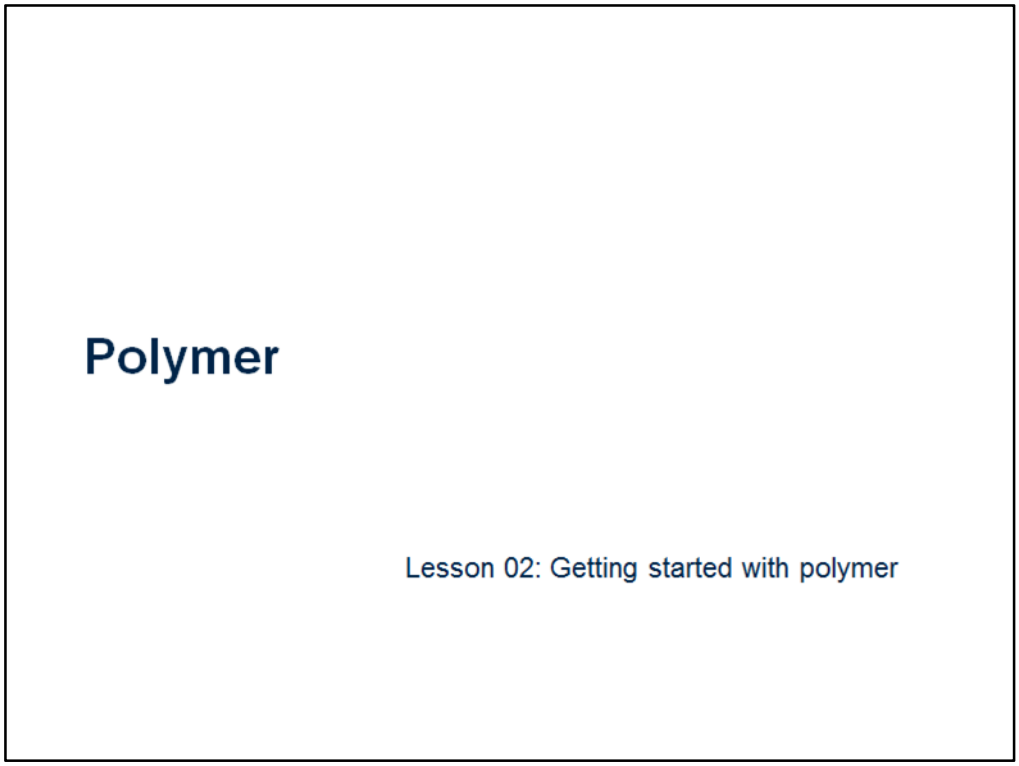
Add instructor notes here.

Summary

- Web components are a collection of specifications that enable developers to create their web applications as a set of reusable components
- HTML Templates provides a native way to declare templates. All the contents inside does nothing until activated via cloning.
- Custom Elements lets us to create rich, reusable and highly descriptive markup by defining our own html element.
- Shadow DOM provides encapsulation for DOM and styles but JS is not encapsulated.
- Like <template> tag any HTML in HTML5 import is inert initially.
- HTML imports bundle up HTML, JS, CSS and deliver the components.
- webcomponents.js is a set of polyfills built on top of the Web Components specifications. It makes it possible for developers to use these standards today across all modern browsers.

Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

Lesson Objectives

- At the end of this module you will be able to:
  - Understand the basic concepts of polymer.js
  - Create custom element
  - Explain the concept of Local DOM, Shadow DOM and Shady DOM
  - Create Local DOM using <dom-module> and the <template> element with in it.
  - Create properties for custom element.
  - Understand Polymer components Lifecycle.
  - Use Polymer DOM API
  - Create and use Behaviors
  - Understand Event handling In Polymer
  - Style Polymer components using CSS

Instructor Notes:

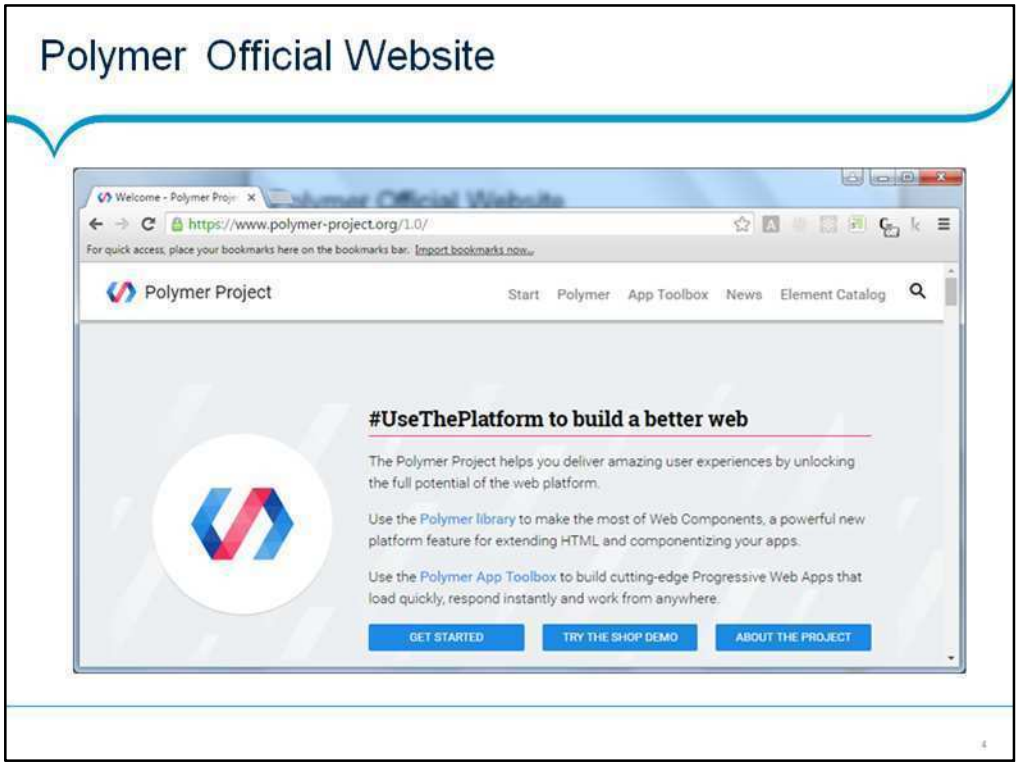
Add instructor notes here.

Polymer Library Introduction

- Web components provide a new standard for building web UI components, and Polymer makes it easy to build reusable components for your web applications.
- Polymer is a new type of library for the web, built on the top of web components, and designed to leverage the evolving web platform on modern browsers.

Instructor Notes:

Add instructor notes here.

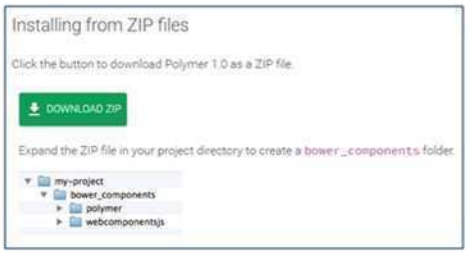


Instructor Notes:

Add instructor notes here.

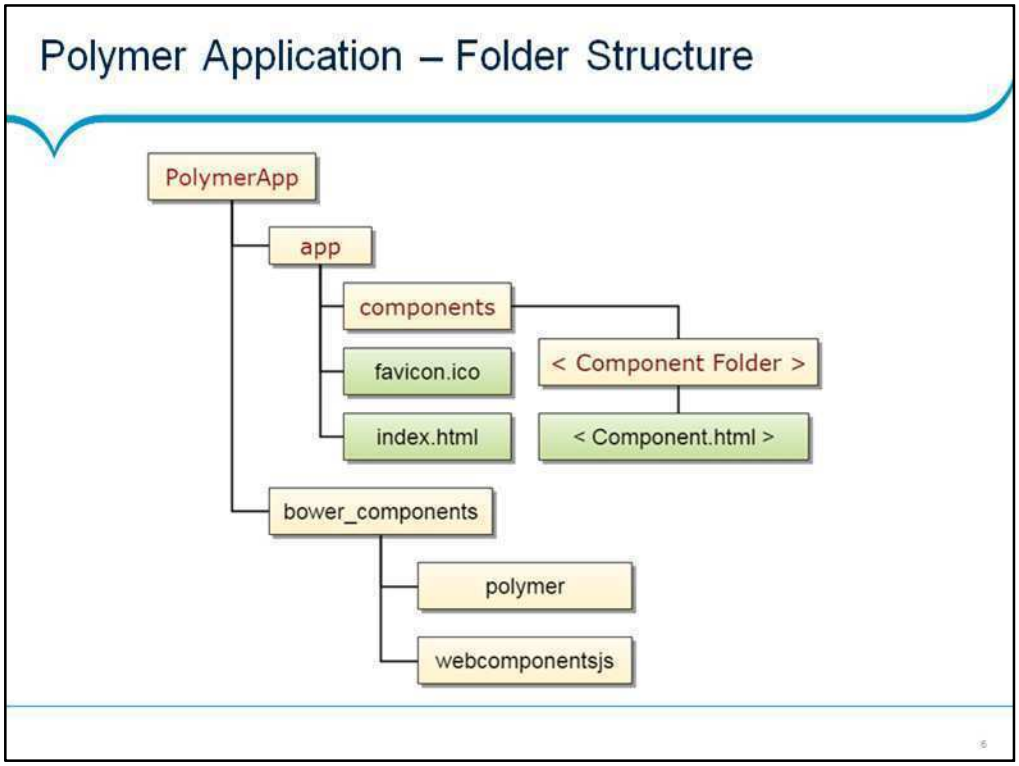
Polymer Installation

- Polymer library can be installed in two ways
  - Installing with Bower (Requires node setup)
  - Installing from ZIP files
- **Installing from Zip files** : When Polymer downloaded as a ZIP file, we will get all of the dependencies bundled into a single archive.
- It's a great way to get started there is no need to install any additional tools.



Instructor Notes:

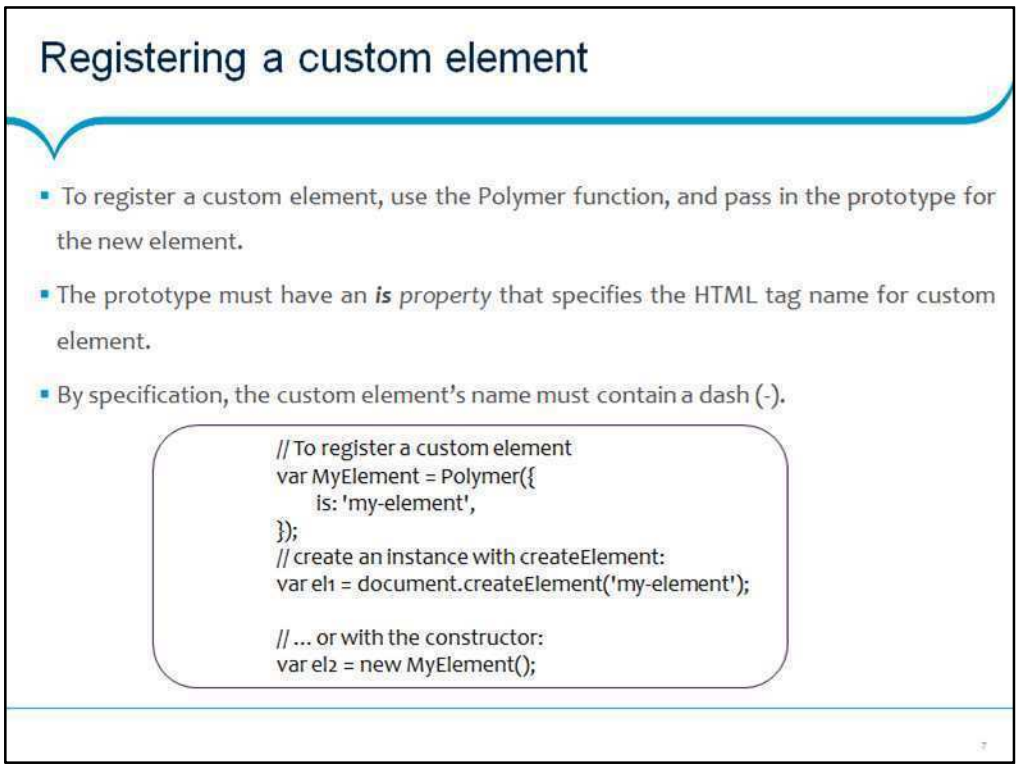
Add instructor notes here.



components folder hold our custom components

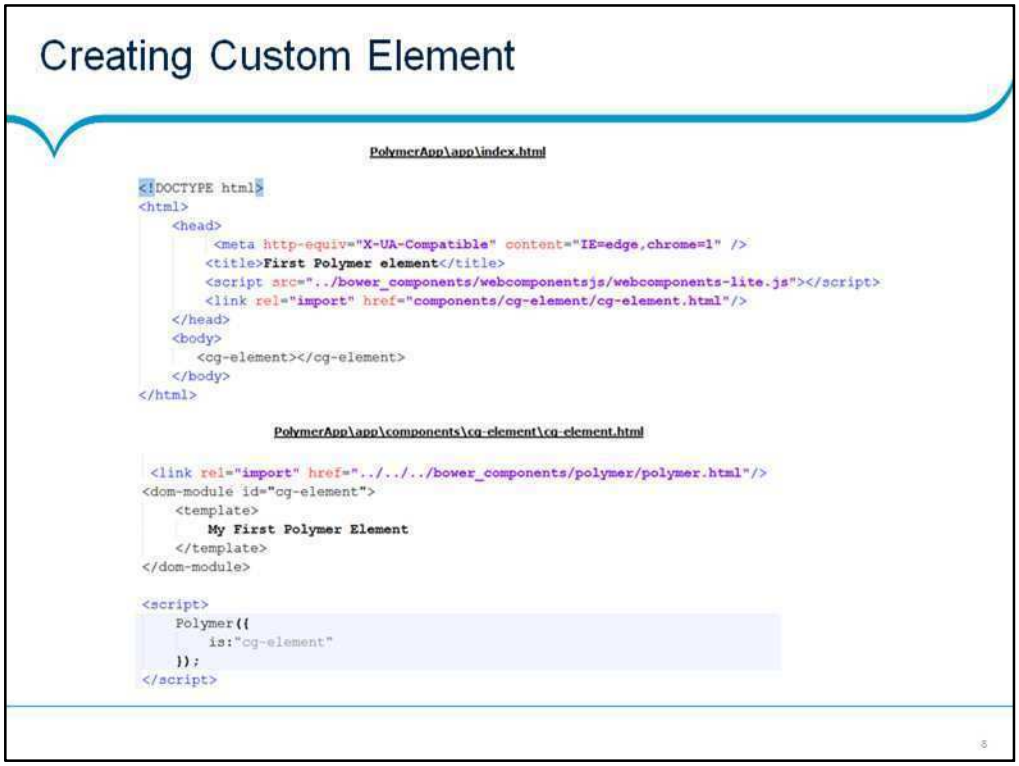
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



To run the program in Internet Explorer
<meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
- IE=edge means IE should use the latest (edge) version of its rendering engine
- chrome=1 means IE should use the Chrome rendering engine if installed

Polyfill for the old browsers which doesn't support rel="import" attribute
<script src="../bower_components/webcomponentsjs/webcomponents-lite.js"></script>

Including the element declaration
<link rel="import" href="components/cg-element/cg-element.html"/>

Adding custom element
<cg-element></cg-element>

importing polymer
<link rel="import" href="../../../bower_components/polymer/polymer.html"/>

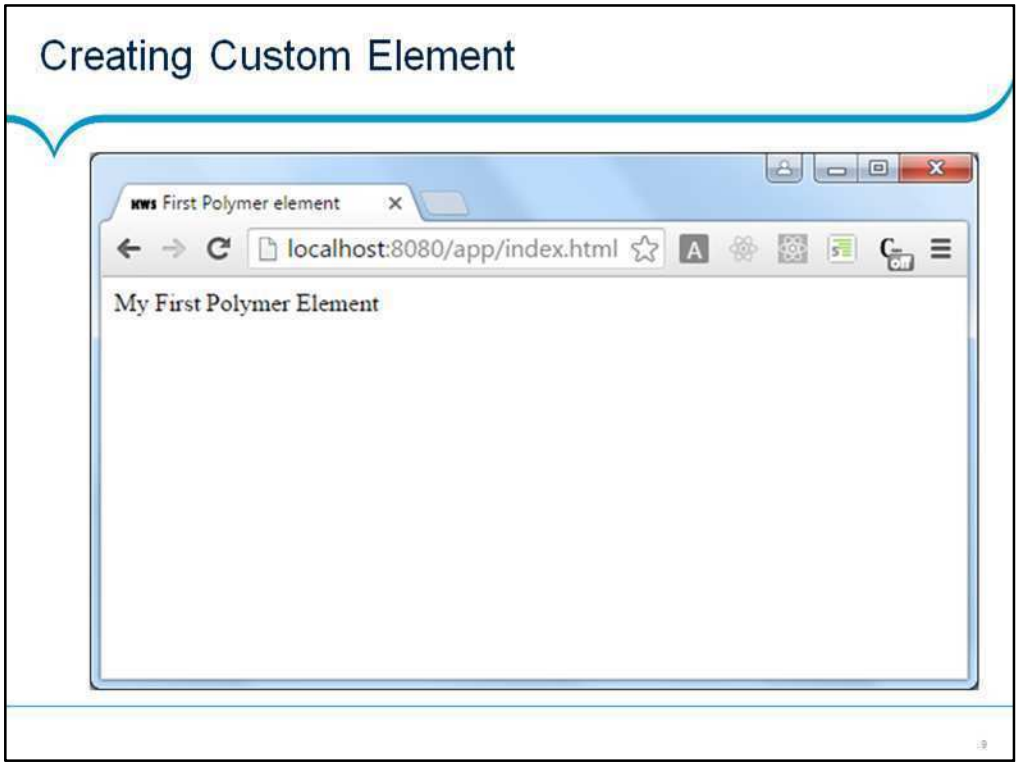
dom module element is used to declare the element
<dom-module id="cg-element">

details in the template tag will be used as a content for the element
<template> My First Polymer Element </template>
</dom-module>

<script>
Adding the element to the polymer scope which will parse the dom module
Polymer({ is:"cg-element" });
</script>

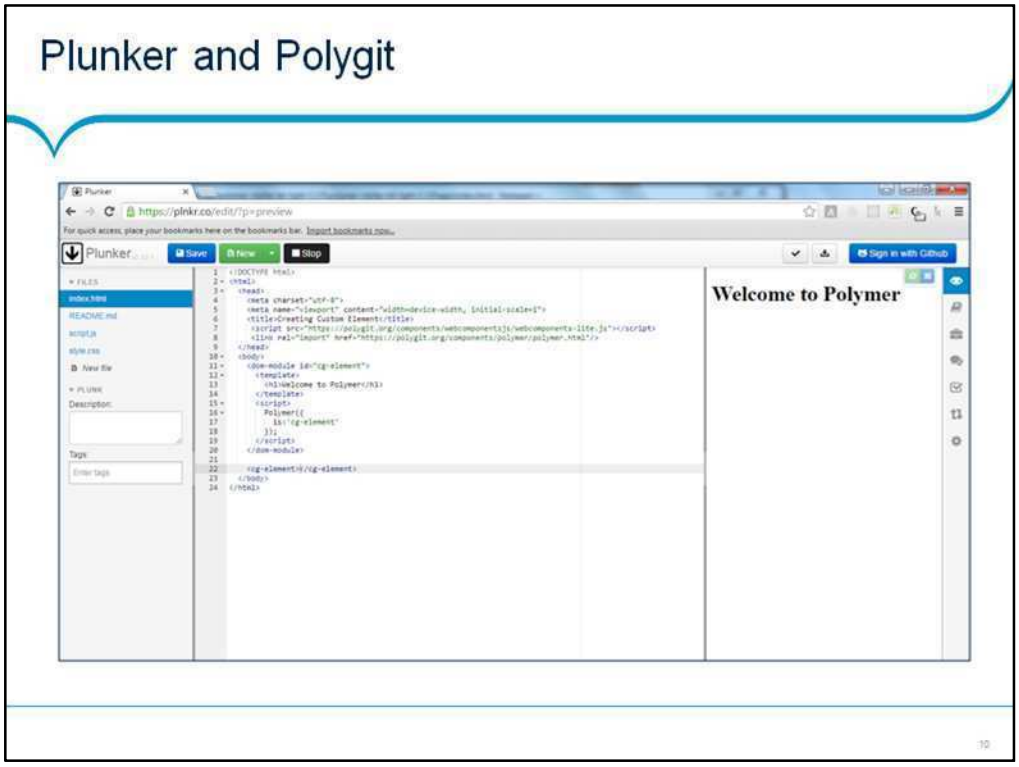
Instructor Notes:

Add instructor notes here.



Instructor Notes:

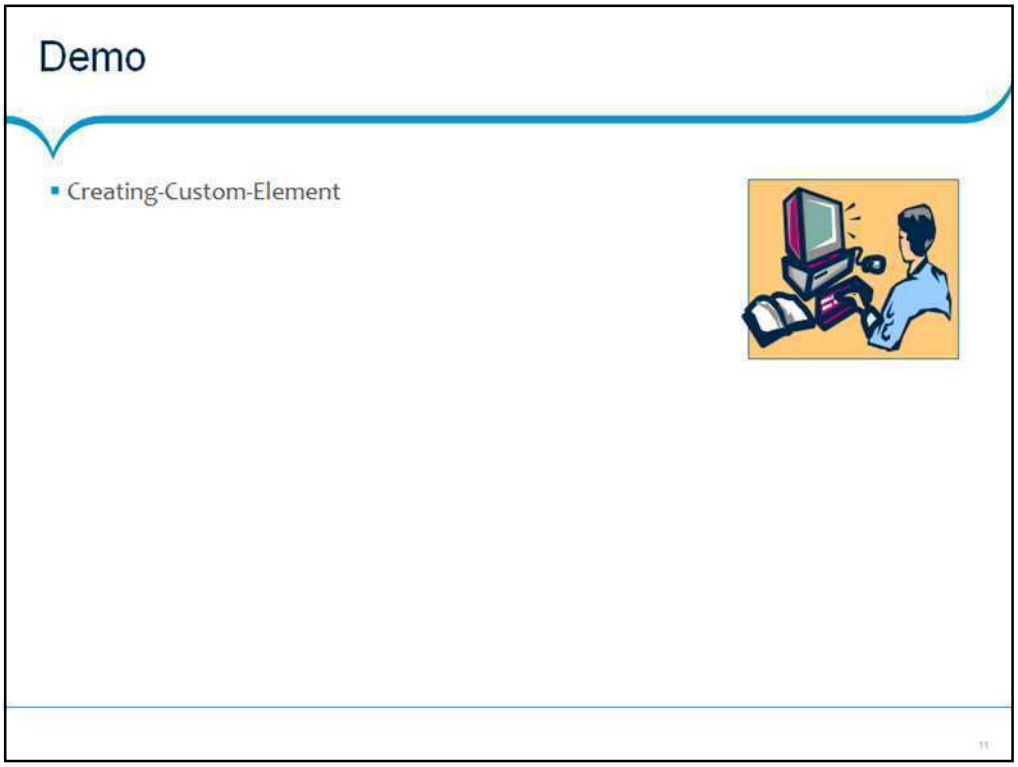
Add instructor notes here.




```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="utf-8">
   <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1" />
   <title>Creating Custom Element</title>
    <script src="https://polygit.org/components/webcomponentsjs/webcomponents-l te.js"></script>
    <link rel="import" href="https://polygit.org/components/polymer/polymer.html"/>
</head>
<body>
                <dom-module id="cg-element">
                    <template>
                            <h1>Welcome to Polymer</h1>
                    </template>
                     <script>
                            Polymer({
                                is:'cg-element'
                             });
                     </script>
                </dom-module>

                <cg-element></cg-element>
</body>
</html>
```

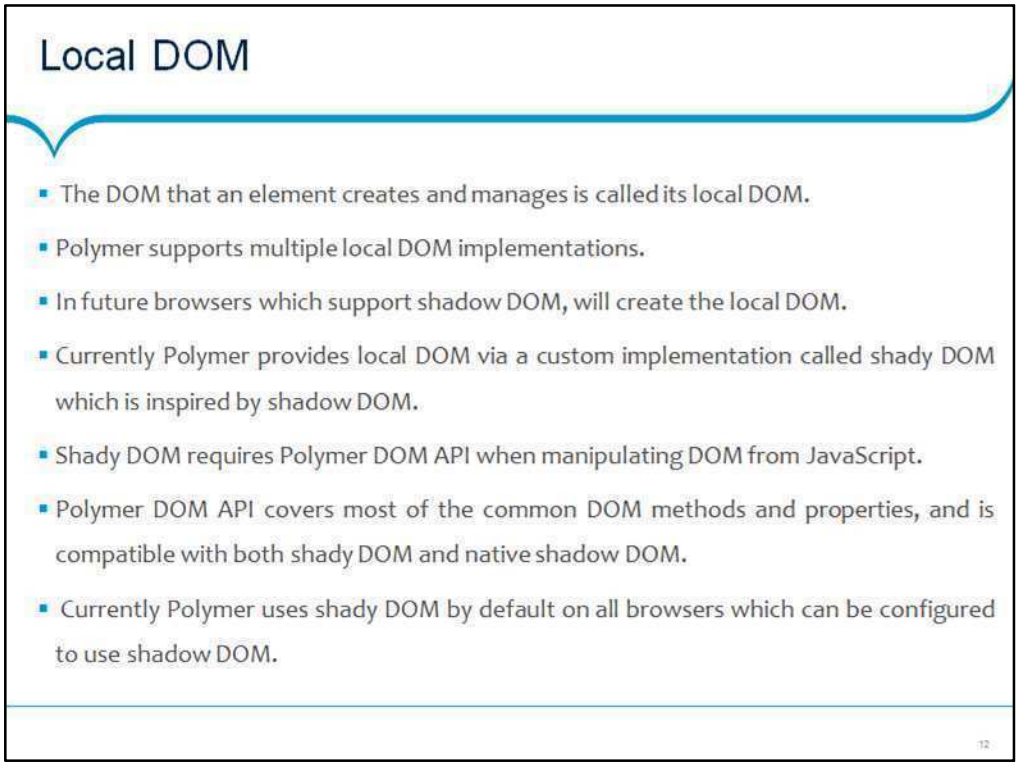
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Document-level global Polymer settings can be set by creating a Polymer object on window before importing the Polymer library

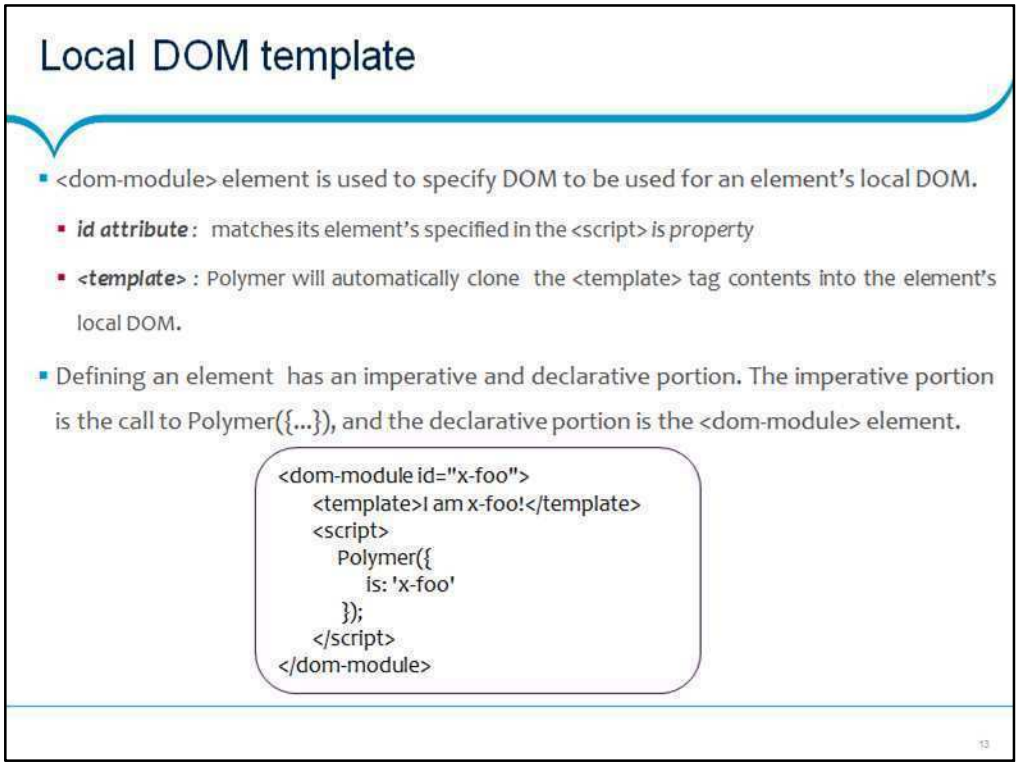
```
<html>
<head>
 <meta charset="utf-8">
 <script src="components/webcomponentsjs/webcomponents-lite.js"></script>
 <script>
  /* this script must run before Polymer is imported */
  window.Polymer = {
    dom: 'shadow',  //shady (current default)
    lazyRegister: true
  };
 </script>
 <!-- import a component that relies on Polymer -->
 <link rel="import" href="elements/my-app.html">
</head>
<body>

 ...
```

Settings can also be switched on the URL query string: http://example.com/test-app/index.html?dom=shadow

shady: All local DOM is rendered using shady DOM, even where shadow DOM is supported (current default).
shadow: Local DOM is rendered using shadow DOM where supported (this will be the default in the future).

Instructor Notes:

Add instructor notes here.



The imperative and declarative portions of an element's definition may be placed in the same html file or in separate files.

The <script> tag can be inside or outside of the <dom-module> element.

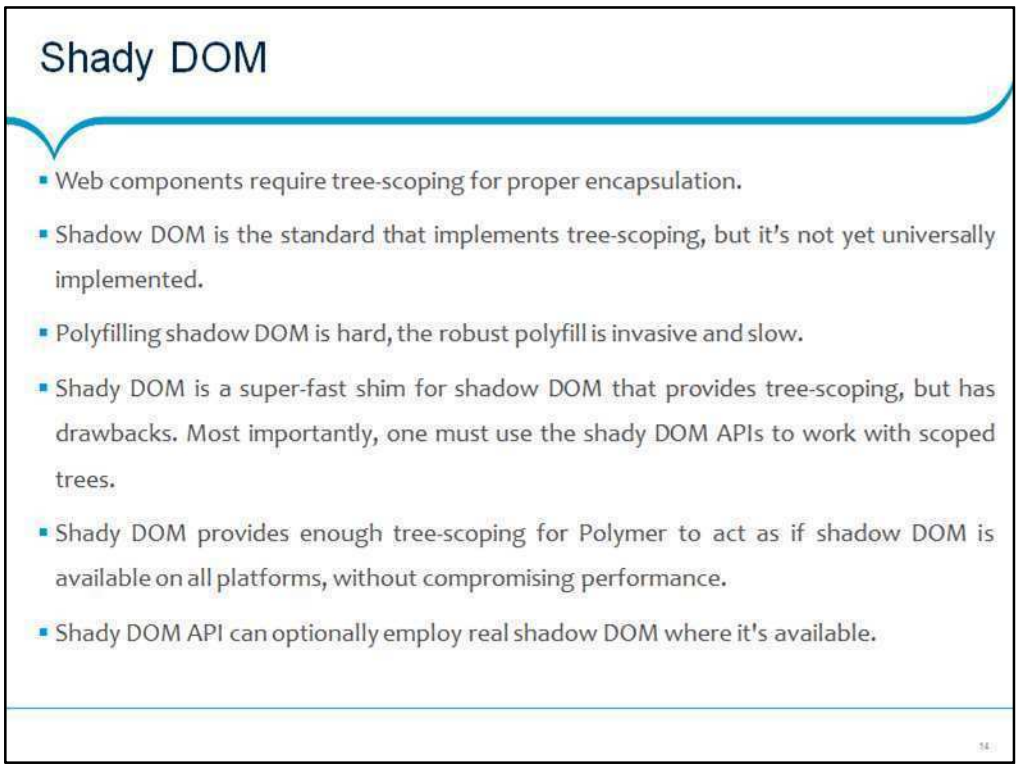
The element's template must be parsed before the call to Polymer.

The children specified with in the element is called as **Light DOM**

```
<dom-module id="x-foo">
    <template>I am x-foo!</template>
    <script>
        Polymer({
            is: 'x-foo'
        });
    </script>
</dom-module>

<body>
    <x-foo>
      <p>Child element placed inside element</p><!– Light DOM --- >
    </x-foo>
</body>
```

Instructor Notes:

Add instructor notes here.



Many projects cannot afford the Shadow DOM Polyfill, for the reasons given below.

- It's a lot of code.
- It's slow to indirect all the DOM API.
- Structures like NodeList can simply not be emulated.
- There are certain accessors that cannot be overwritten (for example, window.document, window.document.body).
- The polyfill returns objects that are not actually Nodes, but Node proxies, which can be very confusing.

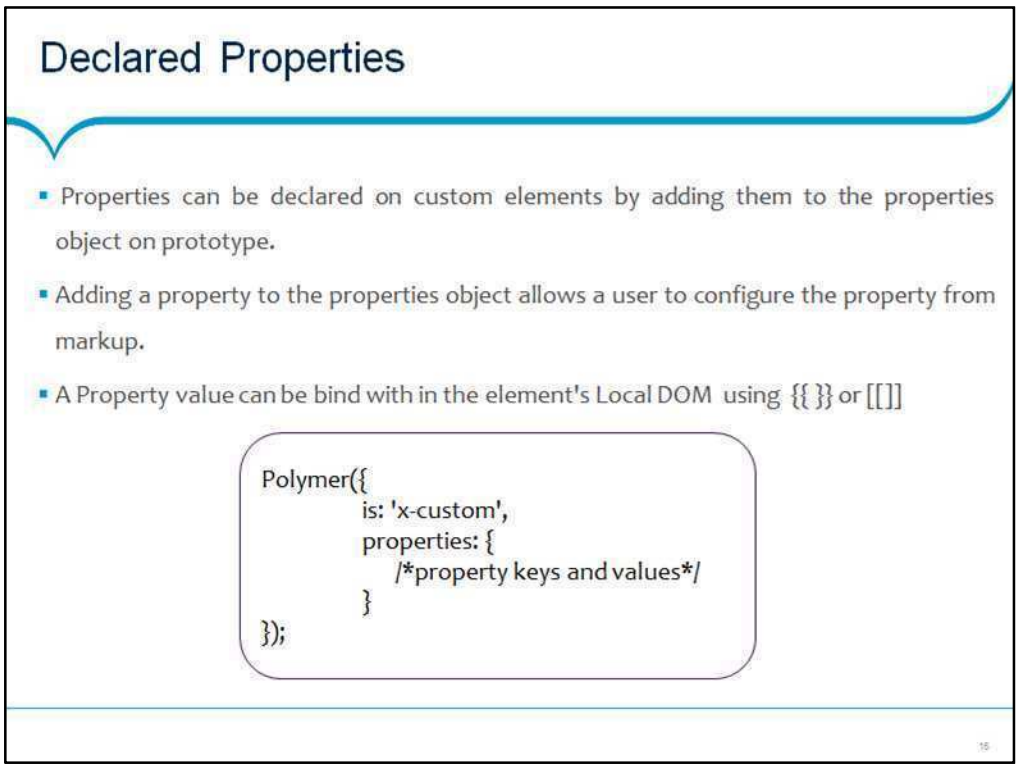
```
> $0
<-   <cg-element>
            My First Polymer Element
       </cg-element>
> $0.textContent
<- "
          My First Polymer Element
      "
> Polymer.dom($0).textContent
<- ""
```

Instructor Notes:

Add instructor notes here.

Instructor Notes:

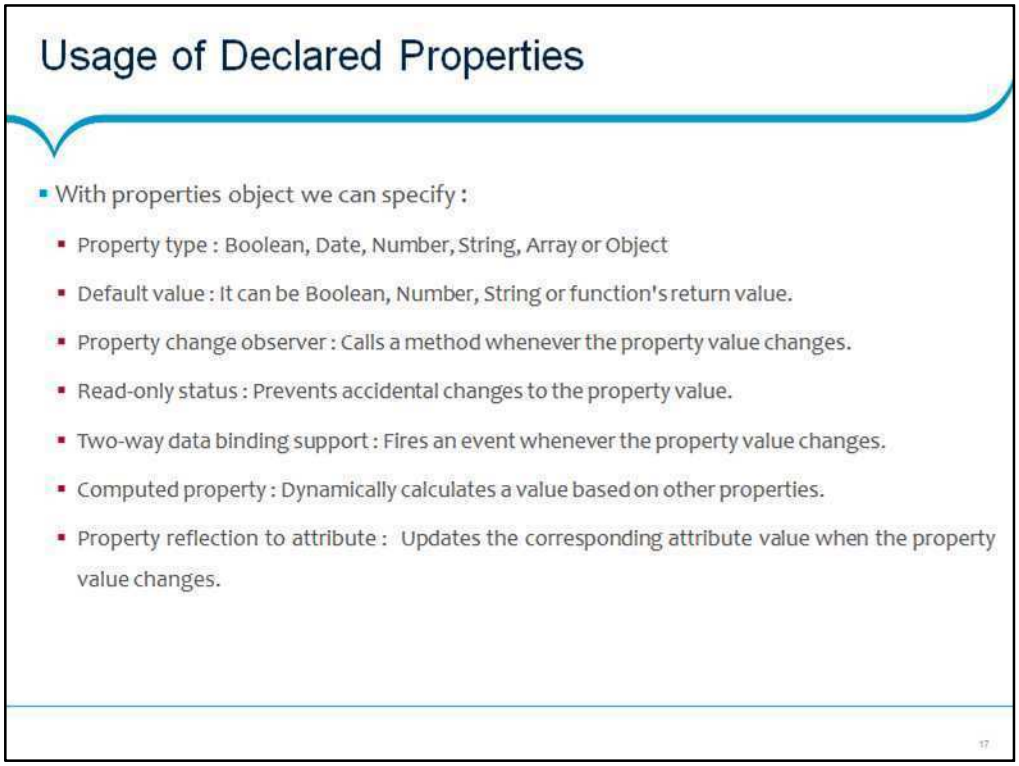
Add instructor notes here.



Using properties we can define properties in element.

Instructor Notes:

Add instructor notes here.

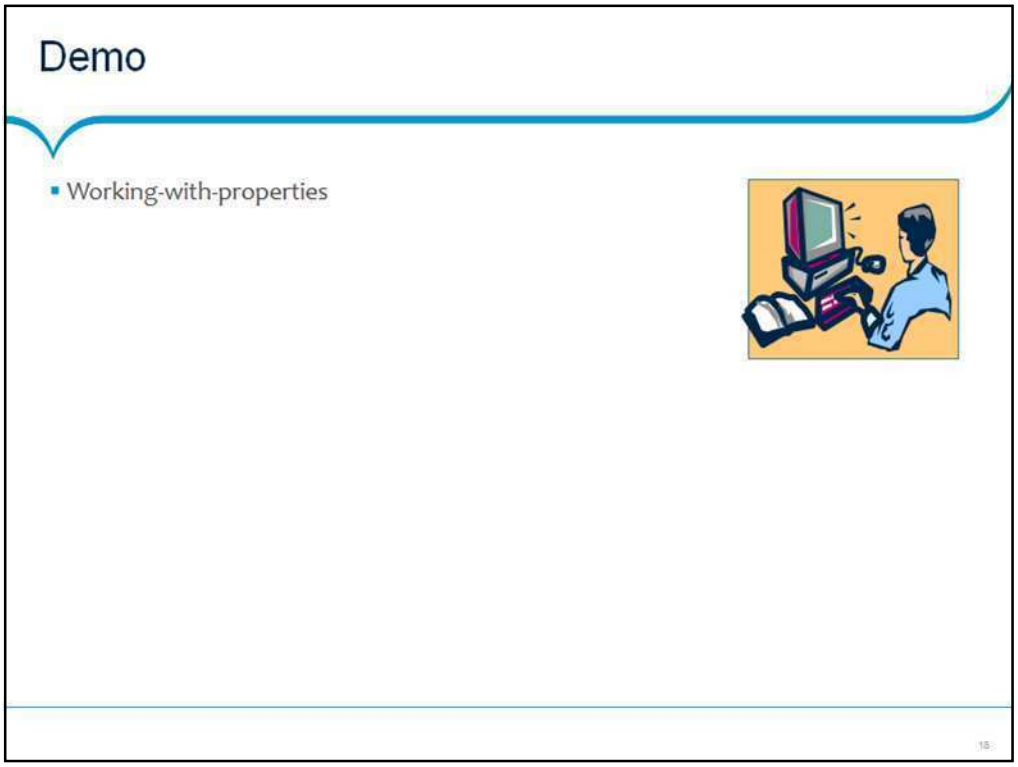


The properties object supports the following keys for each property:

Key	Value
type	Used for deserializing from an attribute. It can be Boolean, Date, Number, String, Array or Object
value	Sets the Default value for the property(boolean, number, string or function). If value is a function, the function is invoked and the return value is used as the default value of the property. If the default value should be an array or object unique to the instance, create the array or object inside a function.
reflectToAttribute	Set to true to cause the corresponding attribute to be set on the host node when the property value changes. If the property value is Boolean, the attribute is created as a standard HTML boolean attribute
readOnly	If true, the property can't be set directly by assignment or data binding
notify	If true, the property is available for two-way data binding. In addition, an event, property-name-changed is fired whenever the property changes.
computed	The value is interpreted as a method name and argument list. The method is invoked to calculate the value whenever any of the argument values changes. Computed properties are always read-only.
observer	The value is interpreted as a method name to be invoked when the property value changes

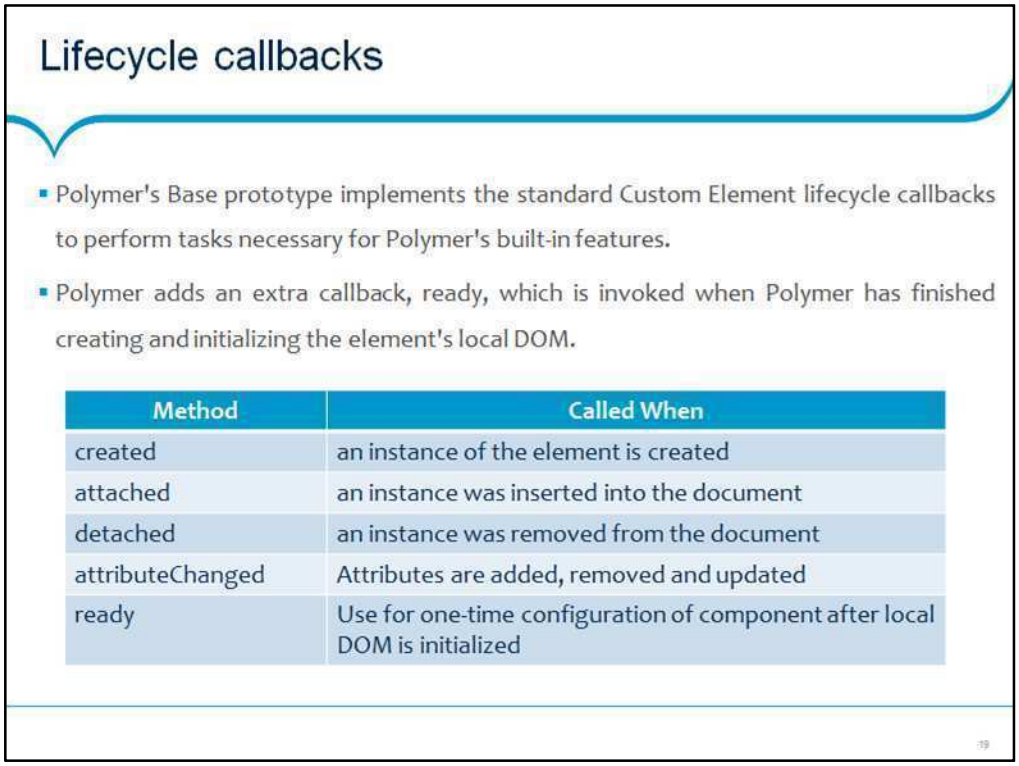
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

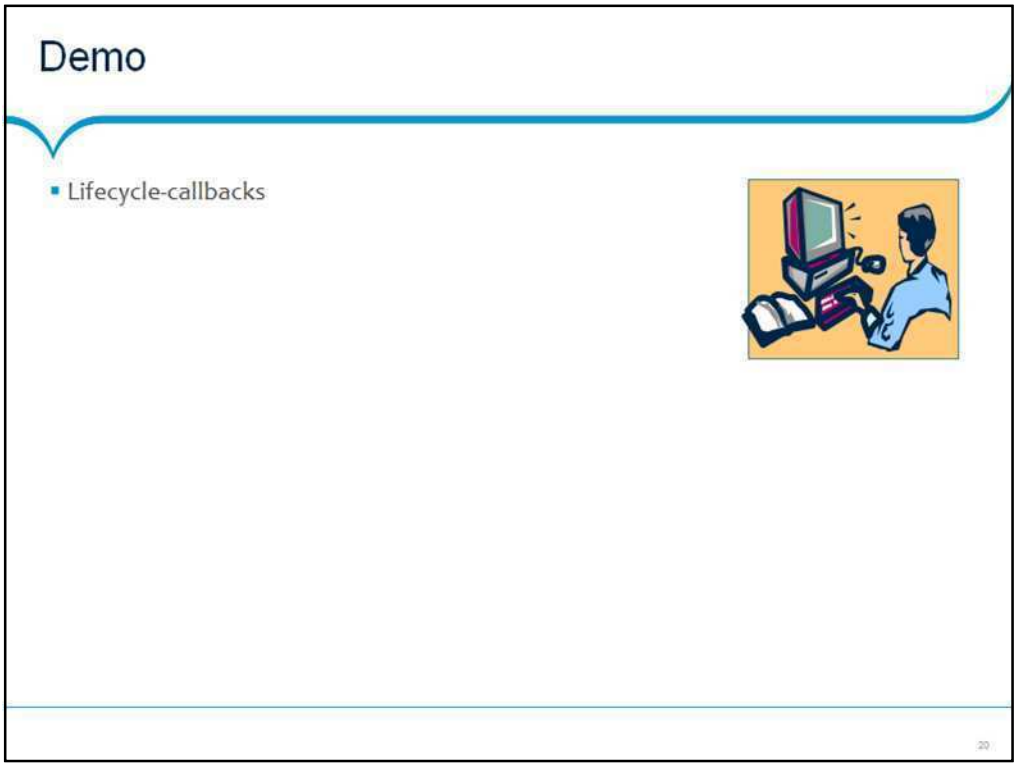


The properties object supports the following keys for each property:

Key	Value
type	Used for deserializing from an attribute. It can be Boolean, Date, Number, String, Array or Object
value	Sets the Default value for the property(boolean, number, string or function). If value is a function, the function is invoked and the return value is used as the default value of the property. If the default value should be an array or object unique to the instance, create the array or object inside a function.
reflectToAttribute	Set to true to cause the corresponding attribute to be set on the host node when the property value changes. If the property value is Boolean, the attribute is created as a standard HTML boolean attribute
readOnly	If true, the property can't be set directly by assignment or data binding
notify	If true, the property is available for two-way data binding. In addition, an event, property-name-changed is fired whenever the property changes.
computed	The value is interpreted as a method name and argument list. The method is invoked to calculate the value whenever any of the argument values changes. Computed properties are always read-only.
observer	The value is interpreted as a method name to be invoked when the property value changes

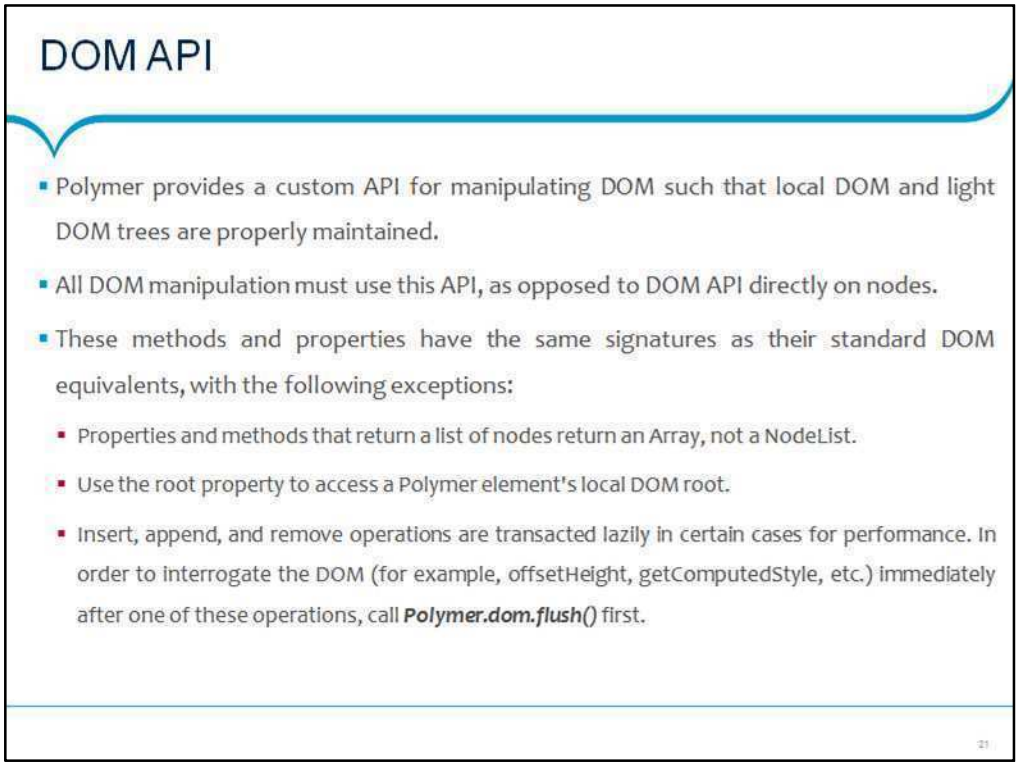
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Use Polymer.dom(this) to access the light dom and Polymer.dom(this.root) to access element's local dom.

The following methods and properties are provided.

Adding and removing children:

Polymer.dom(parent).appendChild(node)
Polymer.dom(parent).insertBefore(node, beforeNode)
Polymer.dom(parent).removeChild(node)
Polymer.dom.flush()

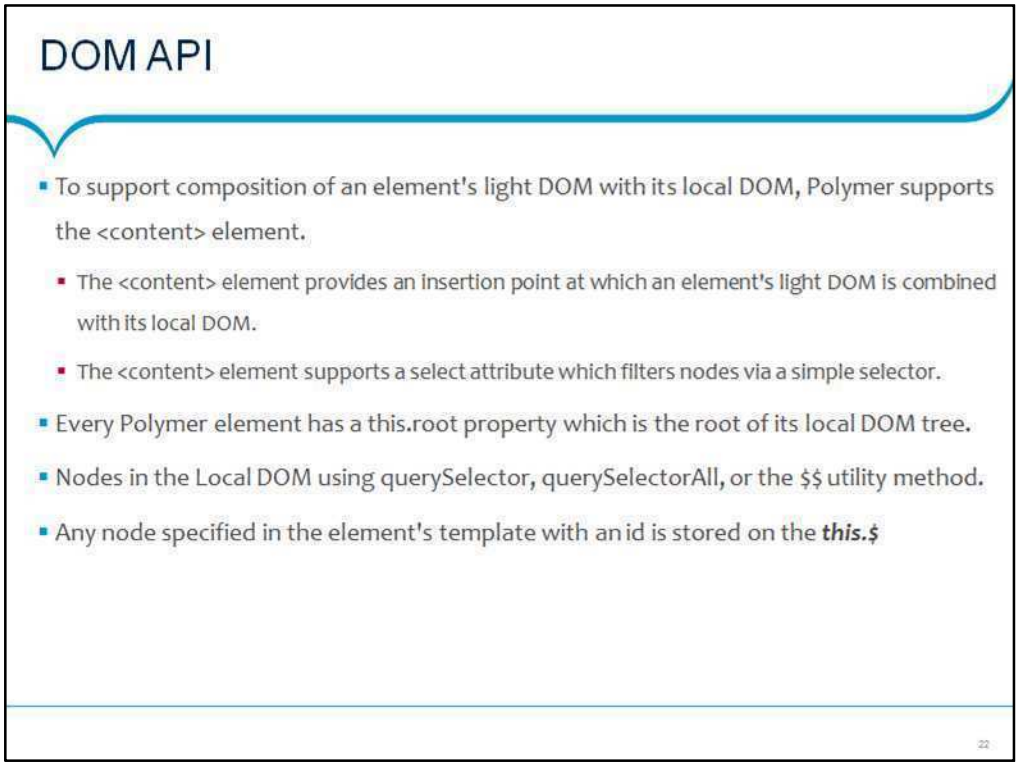
Calling append/insertBefore adds the node to parent's light DOM. In order to insert/append into the local DOM of a custom element, use a node in the local DOM as a parent (or this.root, which is the root of the local DOM).

Parent and child APIs:

Polymer.dom(parent).childNodes
Polymer.dom(parent).children
Polymer.dom(node).parentNode
Polymer.dom(node).firstChild
Polymer.dom(node).lastChild
Polymer.dom(node).firstElementChild
Polymer.dom(node).lastElementChild
Polymer.dom(node).previousSibling
Polymer.dom(node).nextSibling
Polymer.dom(node).textContent
Polymer.dom(node).innerHTML

Instructor Notes:

Add instructor notes here.



Query selector:

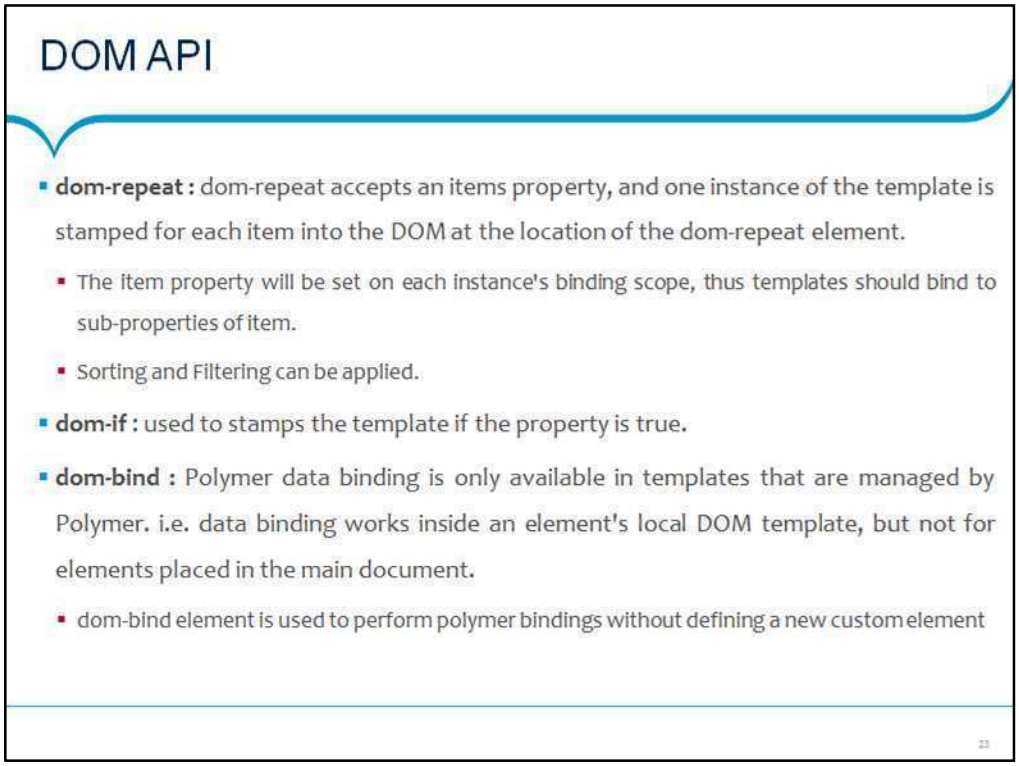
Polymer.dom(parent).querySelector(selector)
Polymer.dom(parent).querySelectorAll(selector)

Content APIs:
Polymer.dom(contentElement).getDistributedNodes()
Polymer.dom(node).getDestinationInsertionPoints()

Node mutation APIs:
Polymer.dom(node).setAttribute(attribute, value)
Polymer.dom(node).removeAttribute(attribute)
Polymer.dom(node).classList

Instructor Notes:

Add instructor notes here.



Instructor Notes:

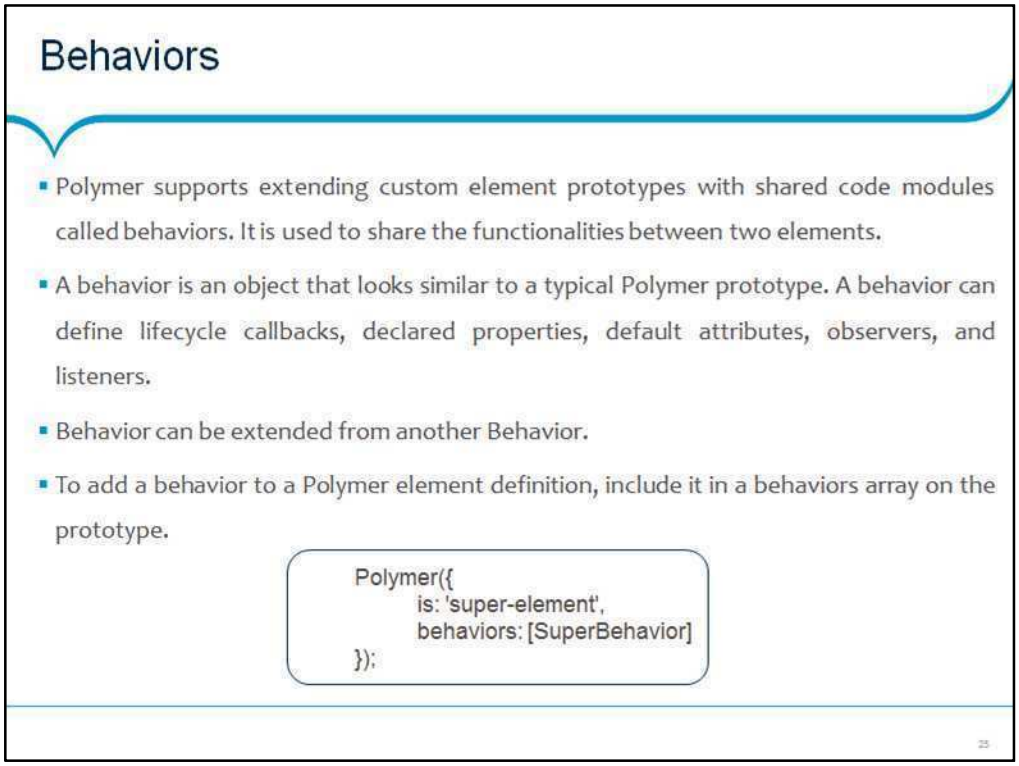
Add instructor notes here.

Demo

- Polymer-DOM-API
- Polymer-Api
- InsertionPoints

Instructor Notes:

Add instructor notes here.



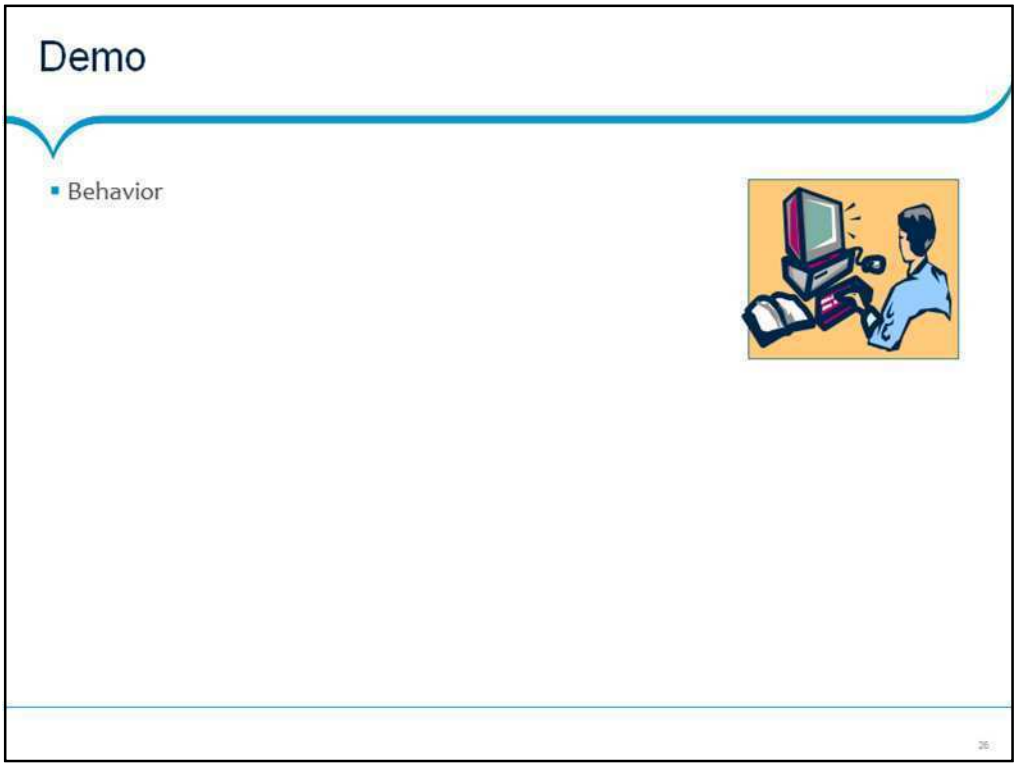
For lifecycle events, the lifecycle callback is called for each behavior in the order given in the behaviors array, followed by the callback on the prototype.

Any non-lifecycle functions on the behavior object are mixed into the base prototype, unless the prototype already defines a function of the same name. If multiple behaviors define the same function, the last behavior in the behaviors array takes precedence over other behaviors.

To extend a behavior, or create a behavior that includes an existing behavior, define a behavior as an array of behaviors.

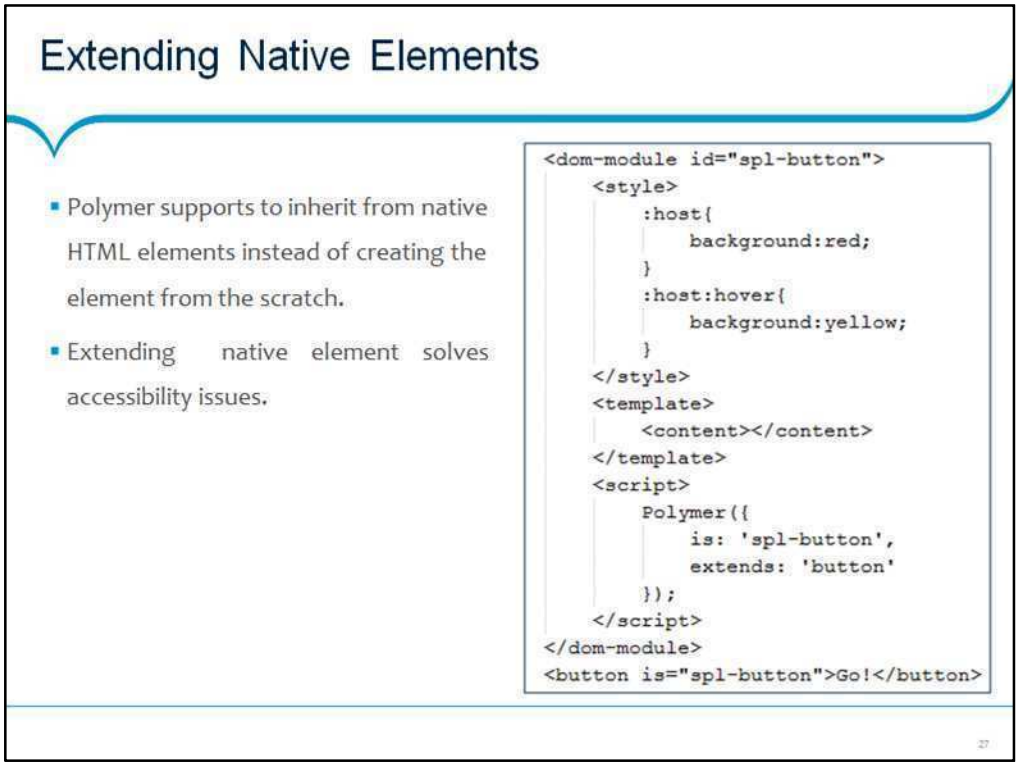
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



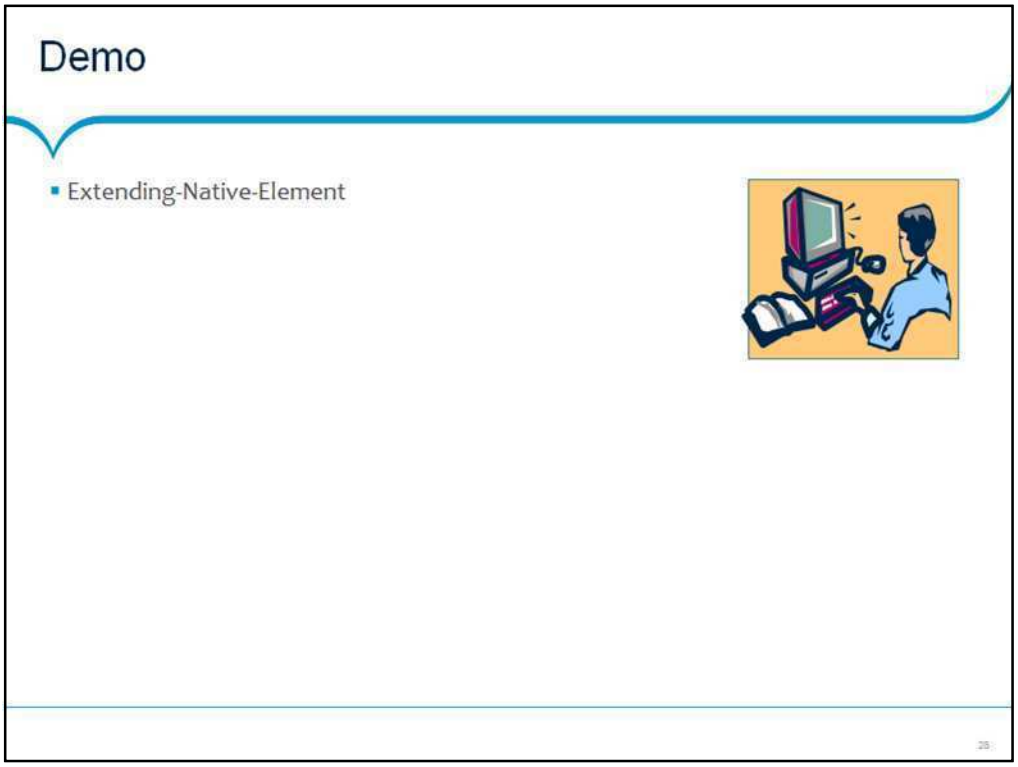
For lifecycle events, the lifecycle callback is called for each behavior in the order given in the behaviors array, followed by the callback on the prototype.

Any non-lifecycle functions on the behavior object are mixed into the base prototype, unless the prototype already defines a function of the same name. If multiple behaviors define the same function, the last behavior in the behaviors array takes precedence over other behaviors.

To extend a behavior, or create a behavior that includes an existing behavior, define a behavior as an array of behaviors.

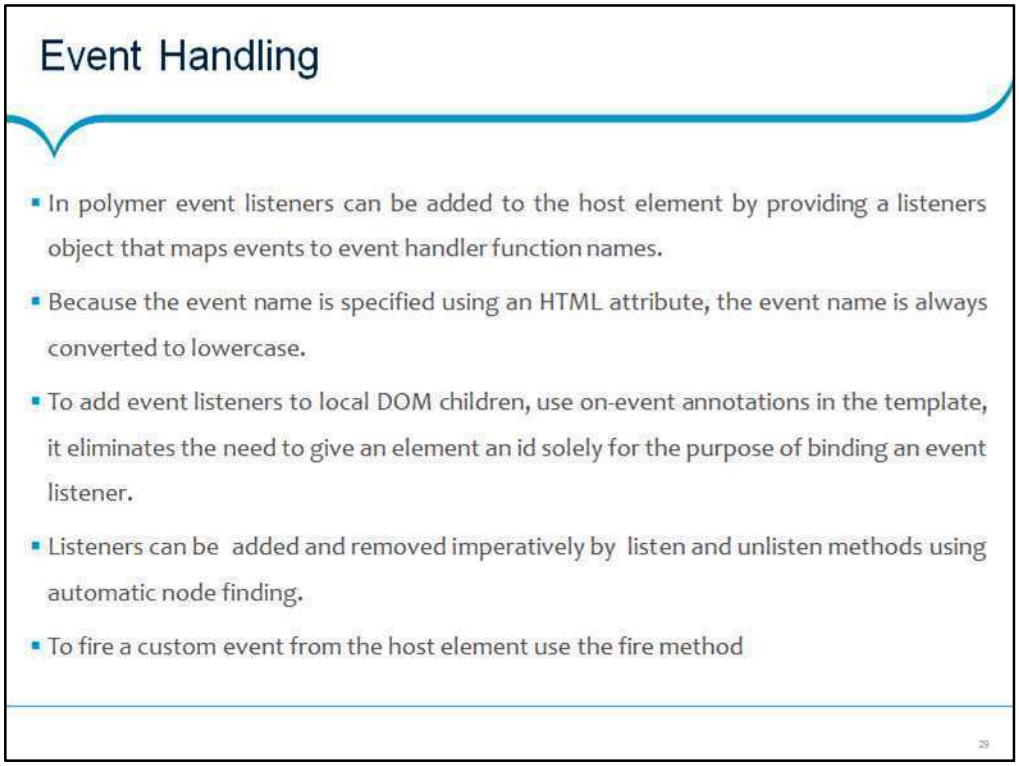
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



If listeners are added imperatively, then it need to be removed imperatively alone. This is commonly done in the attached and detached callbacks. If the listeners are used in object or annotated event listeners, Polymer automatically adds and removes the event listeners.

Event retargeting:
Shadow DOM has a feature called "event retargeting" which changes an event's target as it bubbles up, such that target is always in the same scope as the receiving element.

Shady DOM doesn't do event regargeting for events as they bubble, because the performance cost would be prohibitive.

Use Polymer.dom(event) to get a normalized event object that provides equivalent target data on both shady DOM and shadow DOM. Specifically, the normalized event has the following properties:

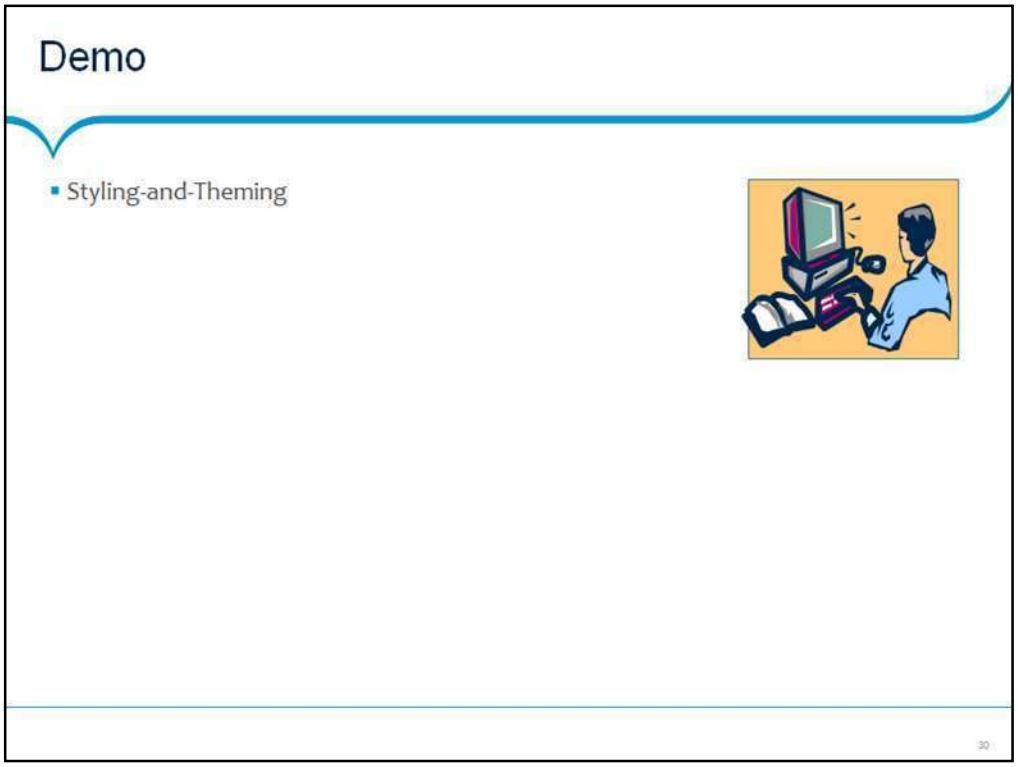
rootTarget: The original or root target before shadow retargeting (equivalent to event.path[0] under shadow DOM or event.target under shady DOM).

localTarget: Retargeted event target (equivalent to event.target under shadow DOM). This node is always in the same scope as the node where the listener was added.

path: Array of nodes through which event will pass (equivalent to event.path under shadow DOM).

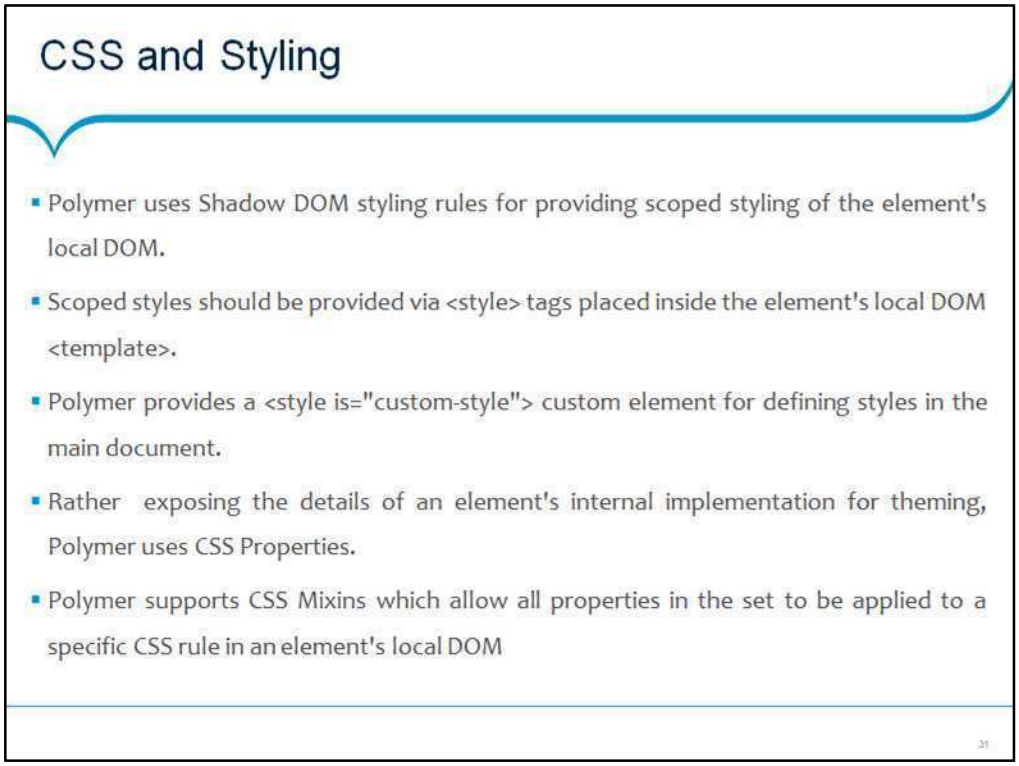
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

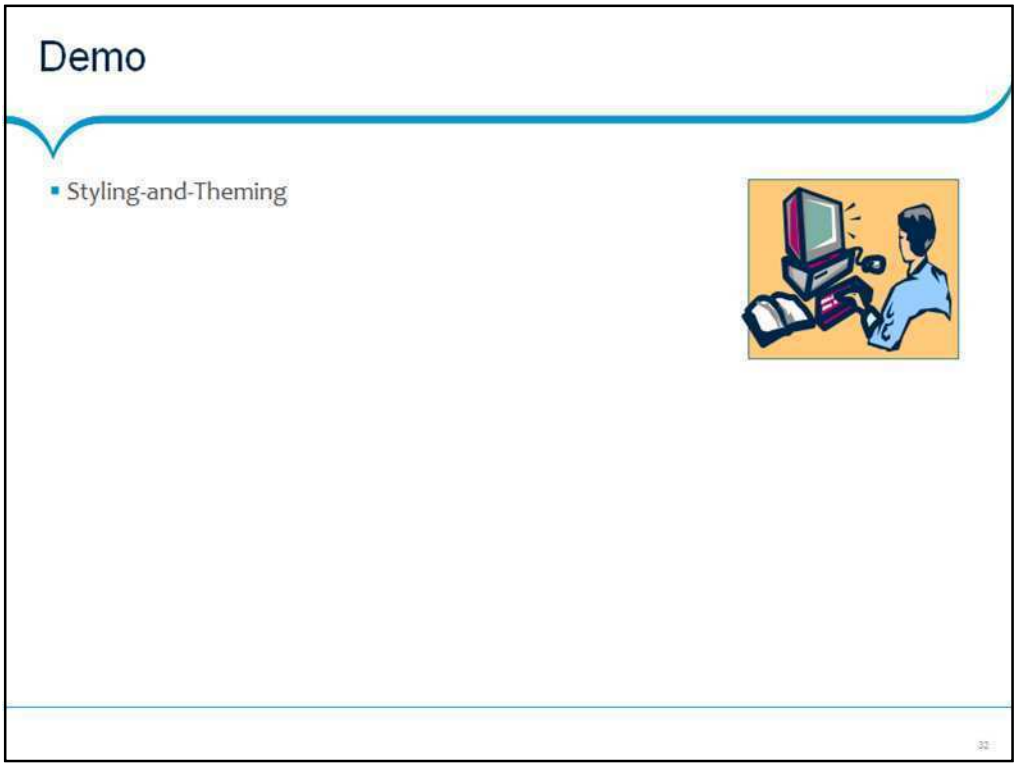


Document styles defined in a custom-style are shimmed to ensure they do not leak into local DOM when running on browsers without native Shadow DOM.

Styling	
:host	style host element
:host ::content	style distributed content
--custom-property: value	create css property 'custom-property' and give it a 'value'
--mixin-name: {mixin contents}	create mixin 'mixin-name' with contents 'mixin contents'
color: var(--my-color, red)	apply custom property 'my color', set default to red
color: @apply(--mixin-name)	apply mixin 'mixin-name' to a property 'color'

Instructor Notes:

Add instructor notes here.



Instructor Notes:

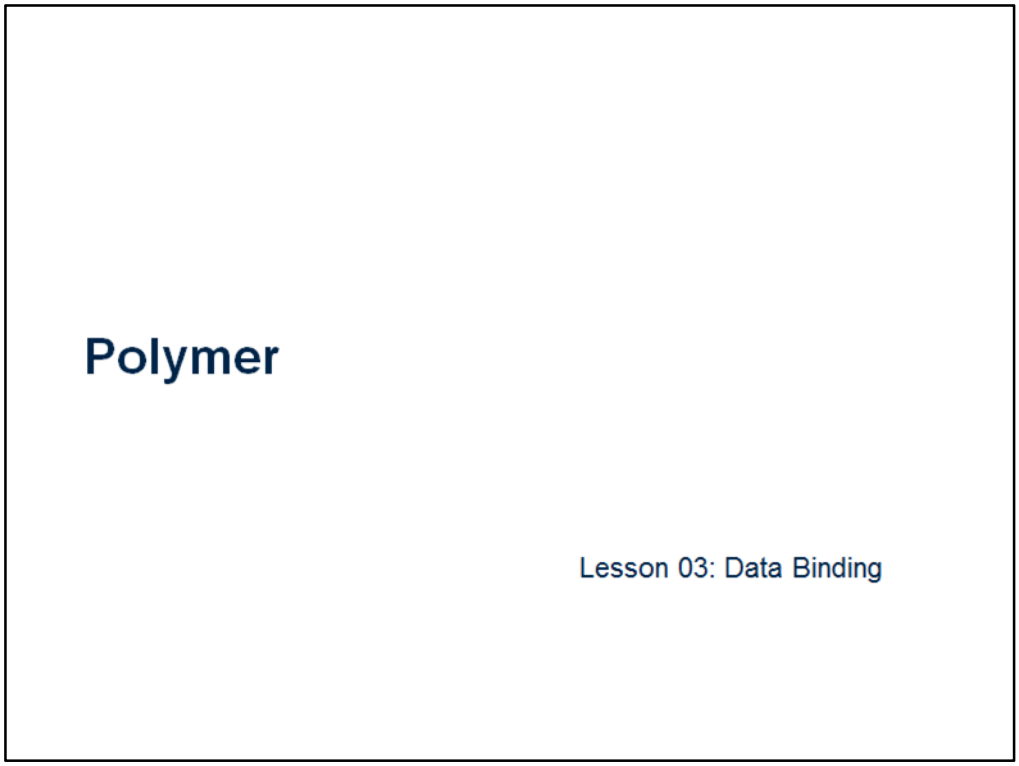
Add instructor notes here.

Summary

- Polymer is a new type of library for the web, built on the top of web components.
- By specification, the custom element's name must contain a dash (-).
- The DOM that an element creates and manages is called its local DOM.
- Shady DOM requires Polymer DOM API when manipulating DOM from JavaScript.
- <dom-module> element is used to specify DOM to be used for an element's local DOM.
- Property value's type can be a Boolean, Date, Number, String, Array or Object.
- A Property value can be bind with in the element's Local DOM using {{ }} or [[]]

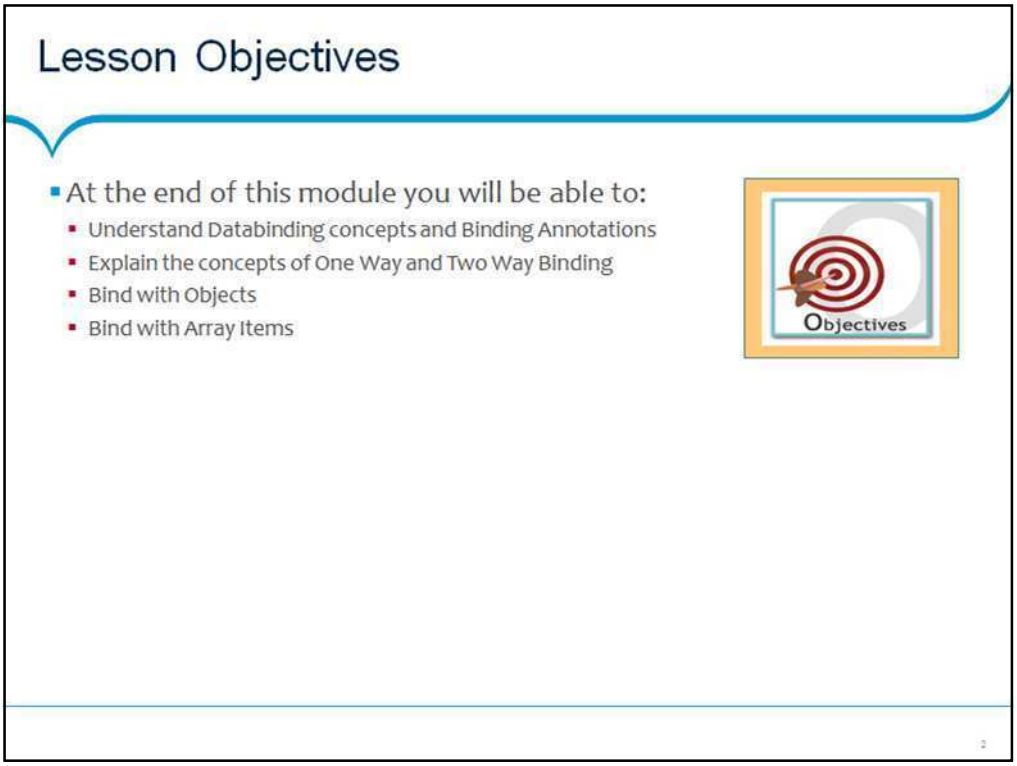
Instructor Notes:

Add instructor notes here.



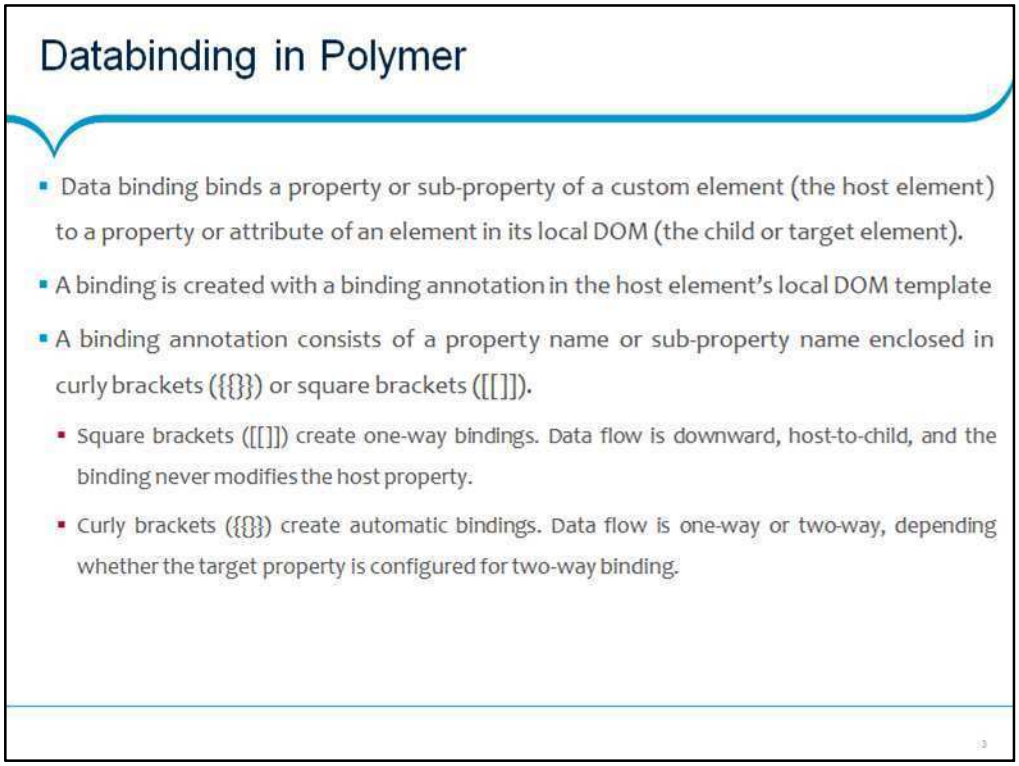
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



To bind to a child property, specify the attribute name that corresponds to the property, with an annotation as the attribute value:

<child-element name="{{myName}}"></child-element>

The above example binds the child element's name property to the host element's myName property.

Property name to attribute name mapping :
For data binding, deserializing properties from attributes, and reflecting properties back to attributes, Polymer maps attribute names to property names and the reverse.

When mapping attribute names to property names:

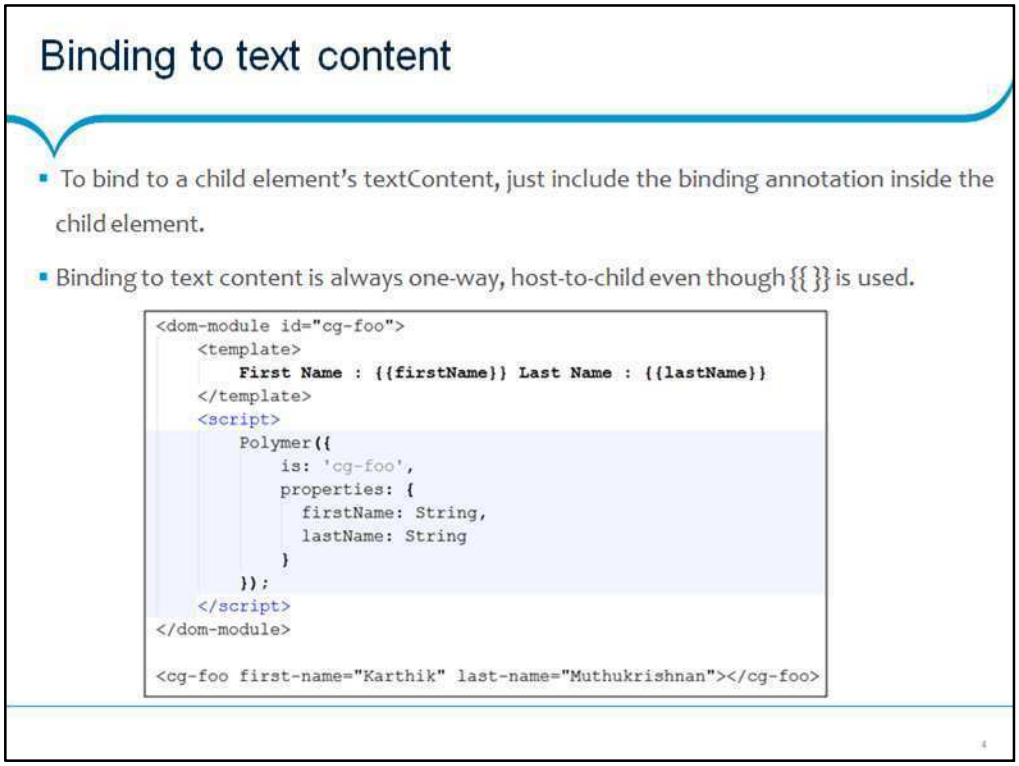
- Attribute names are converted to lowercase property names. For example, the attribute firstName maps to firstname.

- Attribute names with dashes are converted to camelCase property names by capitalizing the character following each dash, then removing the dashes. For example, the attribute first-name maps to firstName.

The same mappings happen in reverse when converting property names to attribute names (for example, if a property is defined using reflectToAttribute: true.)

Instructor Notes:

Add instructor notes here.



When binding to style, href, class, for or data-* attributes, it is recommend to use attribute binding syntax.

To bind to an attribute, use $= rather than =. This results in a call to:

element.setAttribute(attr, value);

As opposed to:

element.property = value;

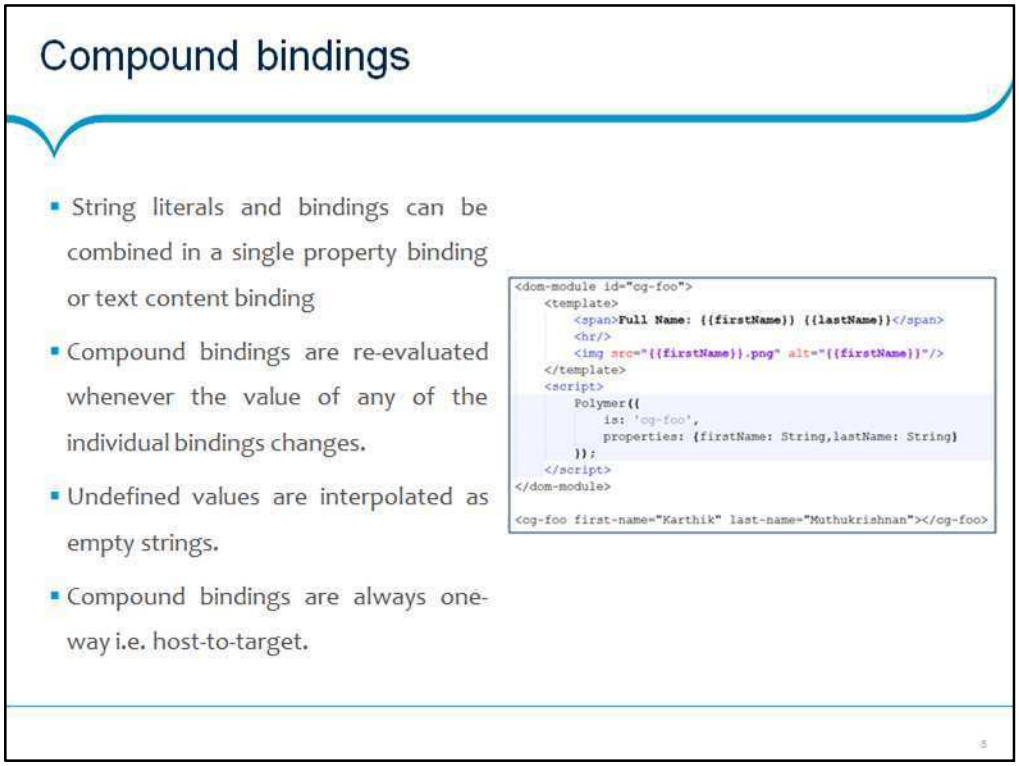
How configuration and binding annotation affect direction of data binding:

Child property config	**notify: false readOnly: false**	**notify: true* readOnly: false**	**notify: true* readOnly: true**	**notify: false readOnly: true**
Binding				
{{curly}}	host --> child	host <--> child	host <-- child	no binding
[[square]]	host --> child	host --> child	silly	no binding

So using curly braces only takes effect when the child's property is notify: true, and setting the child to notify: true only takes effect when the host uses curly braces.

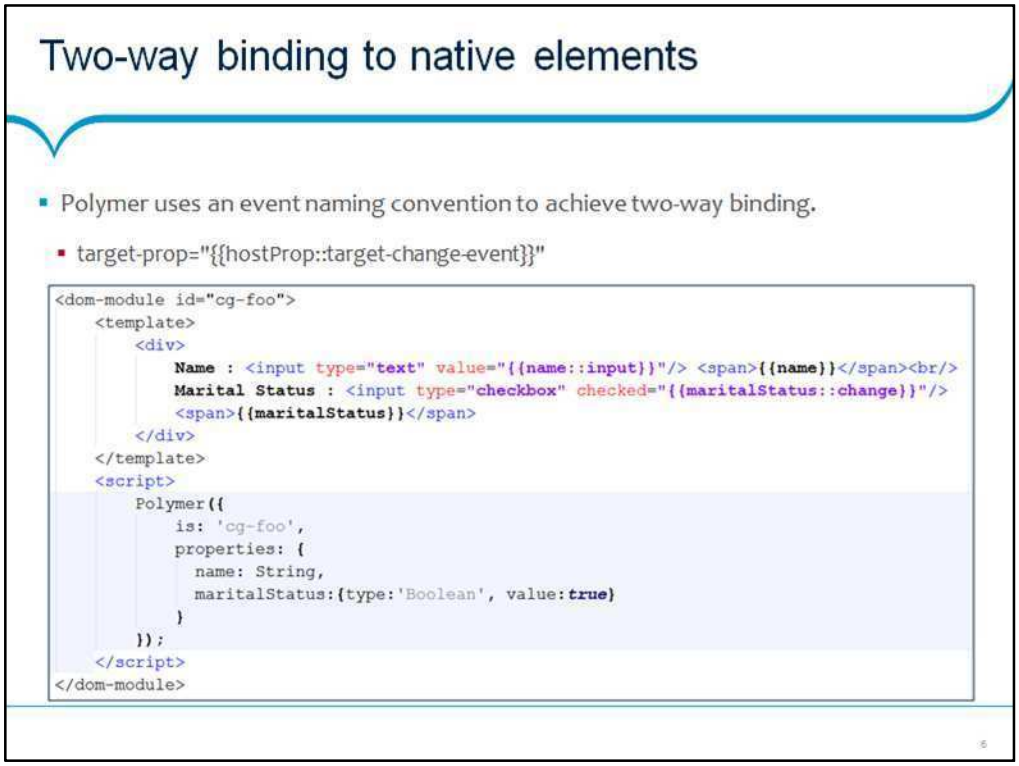
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



To avoid two-way binding, use "square-brace" syntax ([[property]]), which results in only one-way (downward, host-to-child) data-binding.

To summarize, two-way data-binding is achieved when both the host and the child agree to participate, satisfying these three conditions:

- The host must use curly-brace {{property}} syntax. (Square-brace [[property]] syntax results in one-way downward binding, regardless of how the child property is configured.)
- The child property being bound to must be configured with the notify flag set to true (or otherwise send a <property>-changed custom event). (If the property being bound does not have the notify flag set, only one-way (downward) binding will occur.)
- The child property being bound to must not be configured with the readOnly flag set to true. (If the child property is notify: true and readOnly: true, and the host binding uses curly-brace syntax, the binding is one-way, upward (child-to-host).)

Instructor Notes:

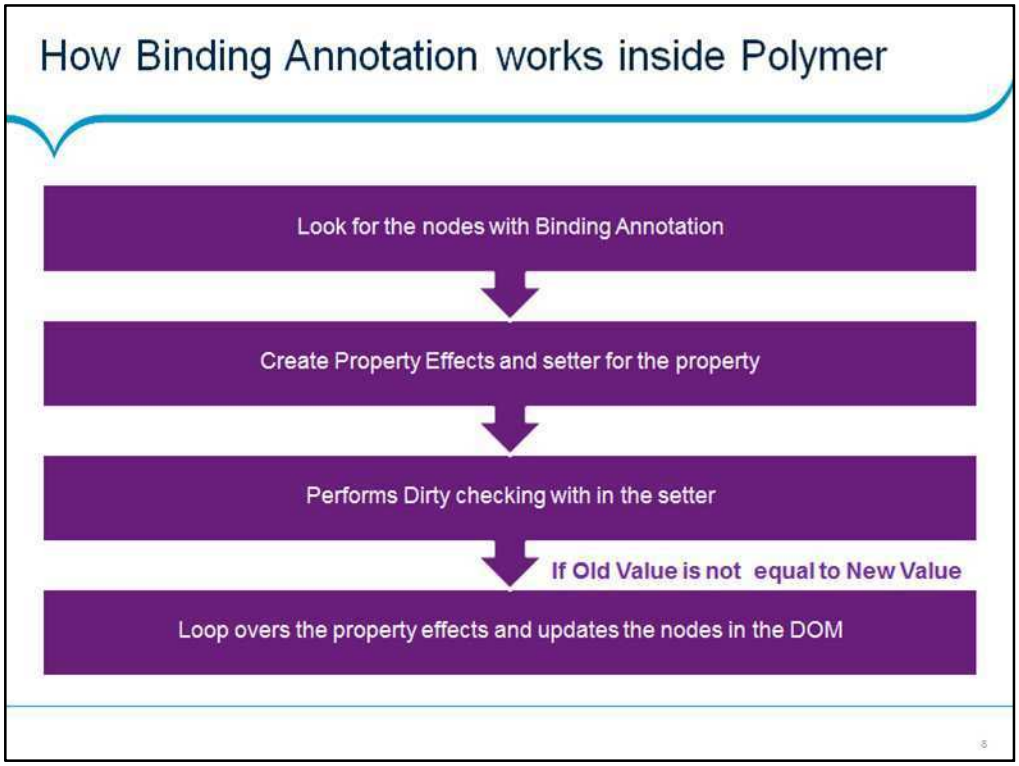
Add instructor notes here.

Demo

- OneWay-Compound-TwoWay
- OneWay-TwoWay-Pasing-values

Instructor Notes:

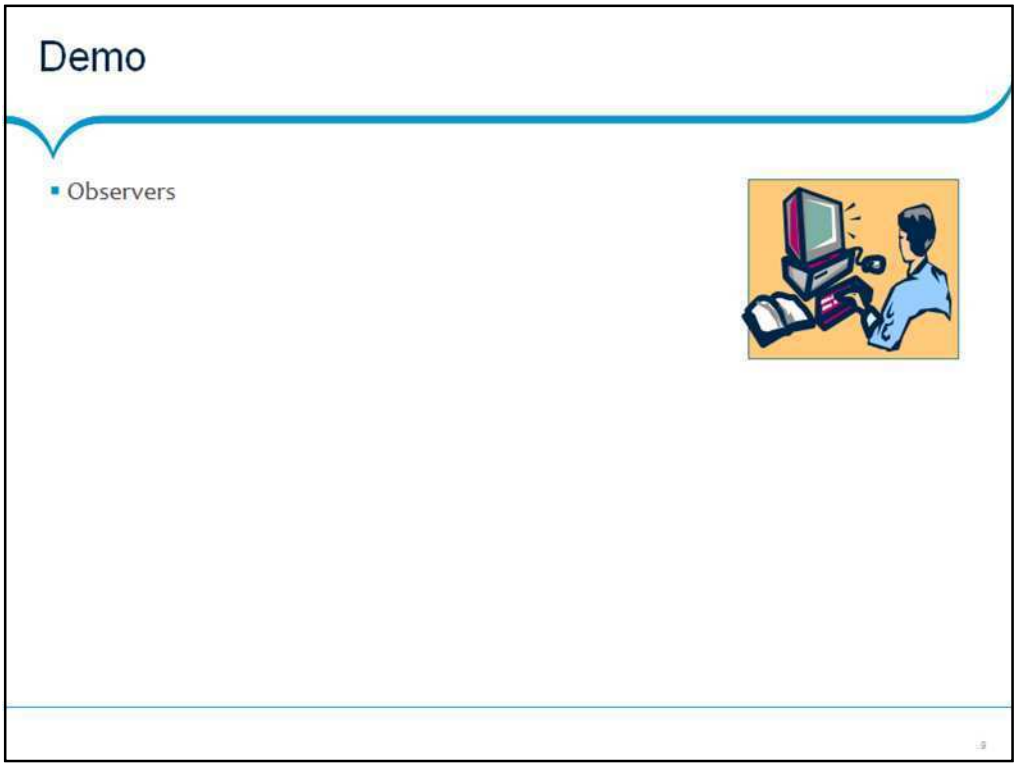
Add instructor notes here.




```
<div>{{foo}}</div>
<paper-input value="{{foo}}">
</paper-input>

_____________________________________________

Polymer({
  is:'cg-binding',
  property :{
    foo:{
        observer:'fooChanged',
        notify:true
    }
  },
  set foo(val){
    /*Dirty checking*/
    var old = this.__data__.foo;
    if(old !== val){
        this.__data__.foo = val;
        //Loops over the property effects
        //and execute their actions
        this.fooChanged();
        this.fire('foo-changed',{value:val})
    }
  }
});
```

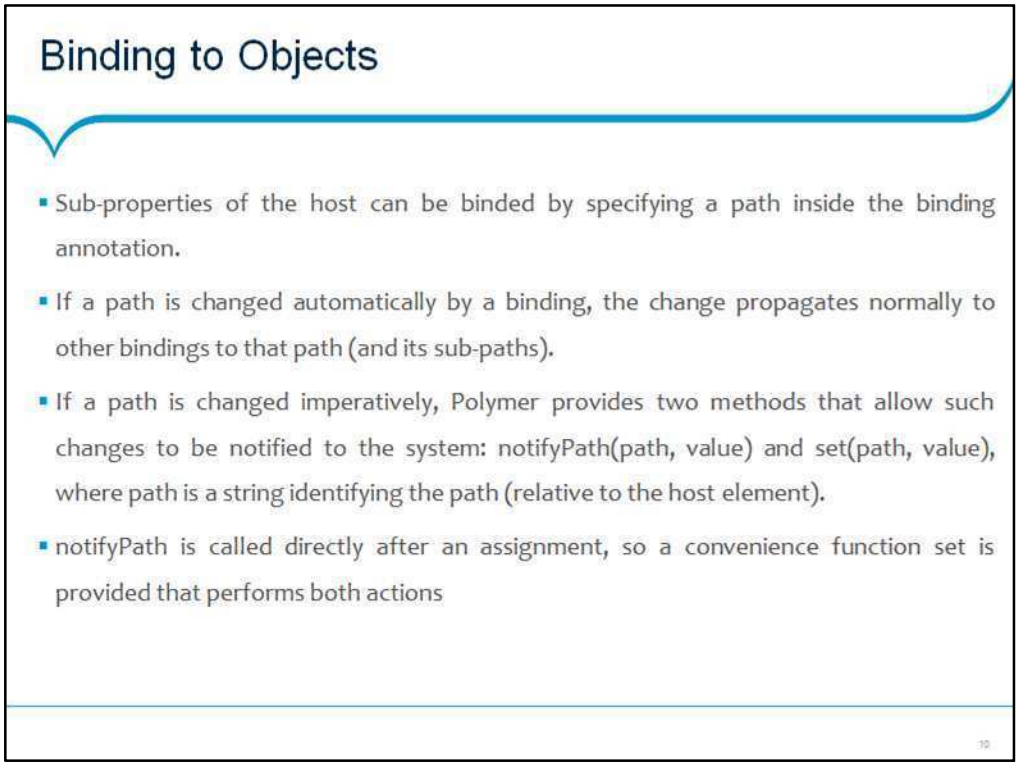
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Path change notification

Two-way data-binding and observation of paths in Polymer is achieved using a similar strategy to 2-way property binding:

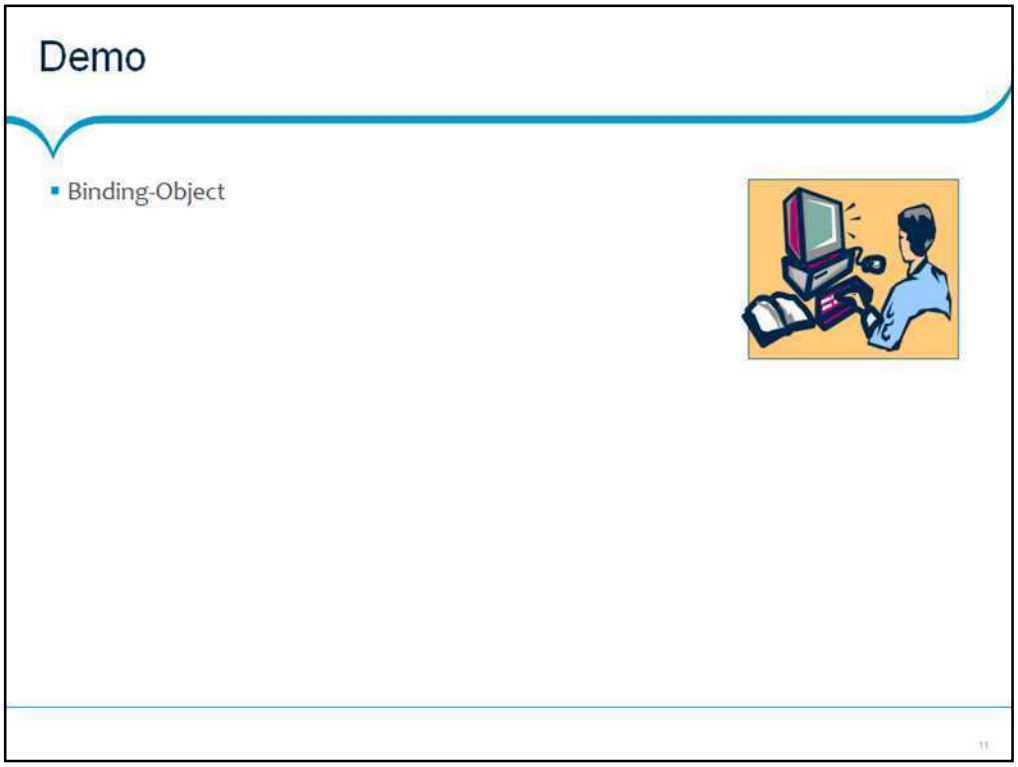
When a sub-property of a property configured with type: Object changes, an element fires a non-bubbling <property>-changed DOM event with a detail.path value indicating the path on the object that changed.

Elements that have registered interest in that object (either via binding or change handler) may then take the appropriate action.

Finally, those elements will forward the notification on to any children they have bound the object to, and will also fire a new <property>-changed event where property is the root object, to notify any hosts that may have bound root object down.

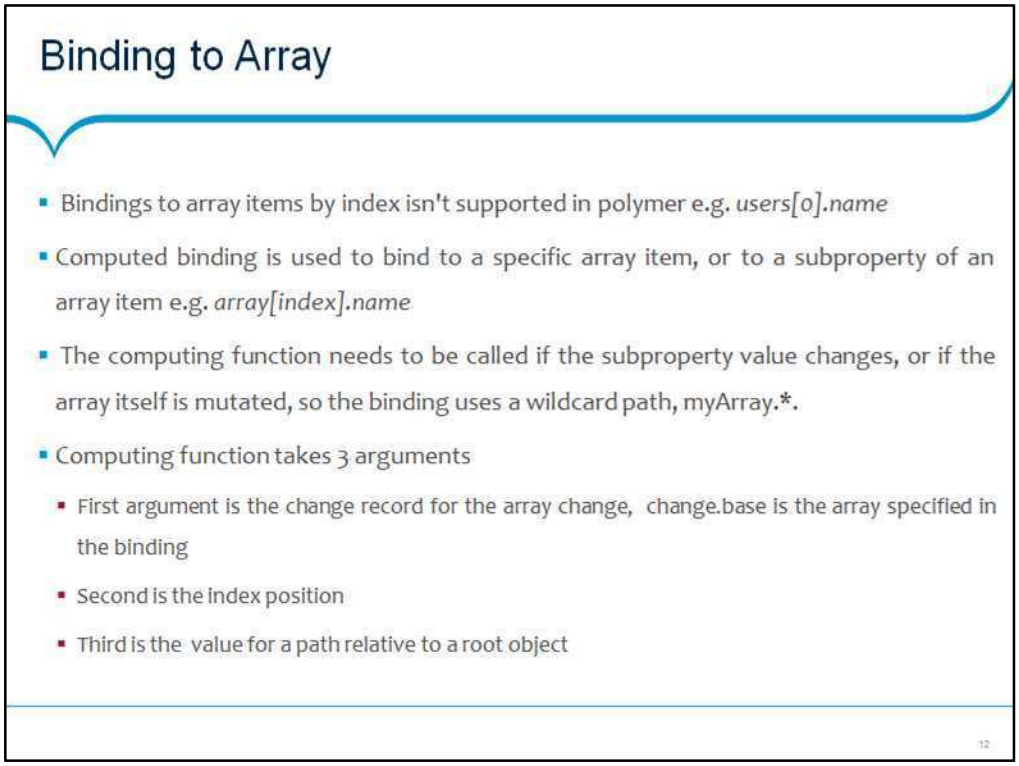
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



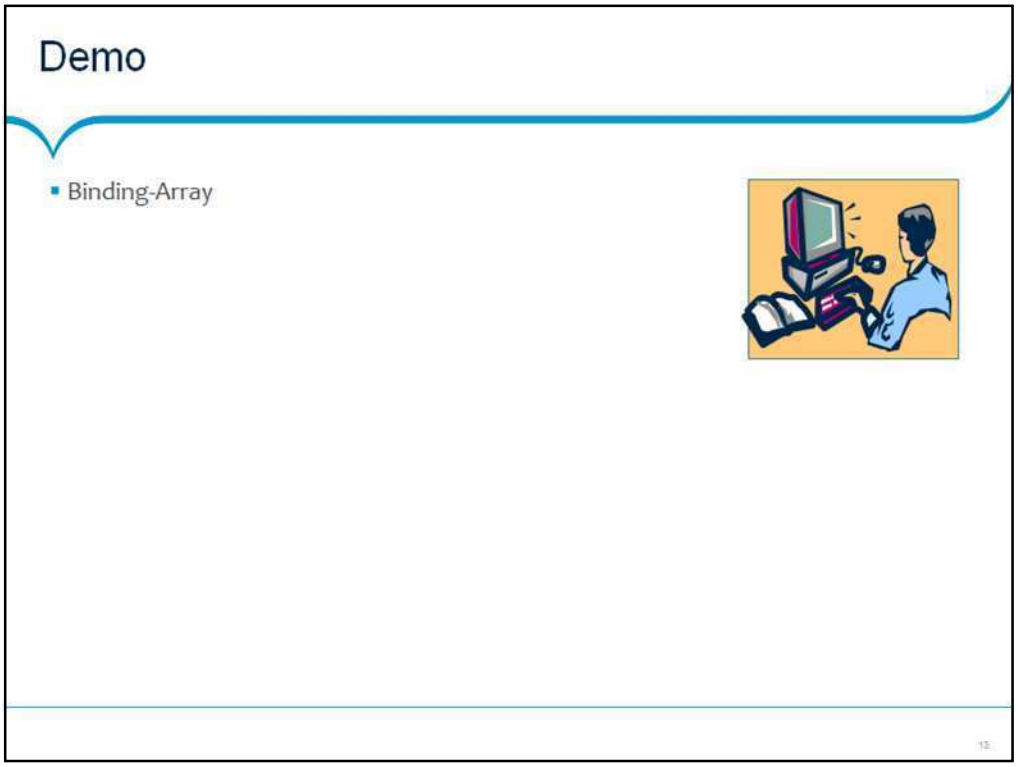
To avoid two-way binding, use "square-brace" syntax ([[property]]), which results in only one-way (downward, host-to-child) data-binding.

To summarize, two-way data-binding is achieved when both the host and the child agree to participate, satisfying these three conditions:

- The host must use curly-brace {{property}} syntax. (Square-brace [[property]] syntax results in one-way downward binding, regardless of how the child property is configured.)
- The child property being bound to must be configured with the notify flag set to true (or otherwise send a <property>-changed custom event). (If the property being bound does not have the notify flag set, only one-way (downward) binding will occur.)
- The child property being bound to must not be configured with the readOnly flag set to true. (If the child property is notify: true and readOnly: true, and the host binding uses curly-brace syntax, the binding is one-way, upward (child-to-host).)

Instructor Notes:

Add instructor notes here.



Instructor Notes:

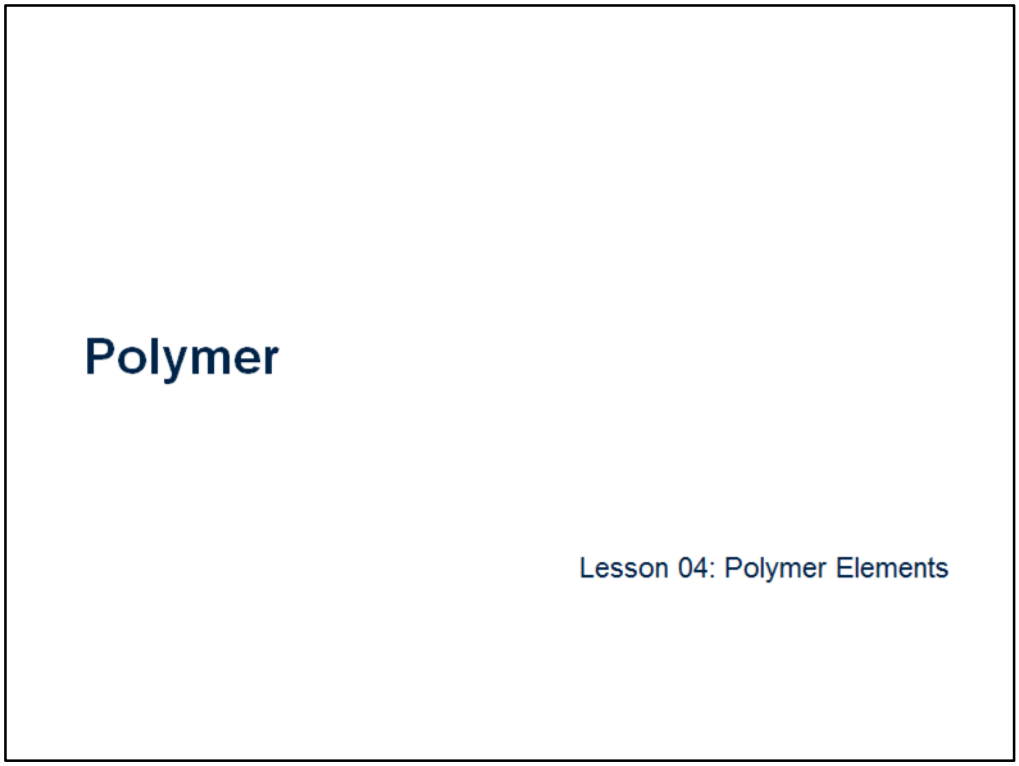
Add instructor notes here.

Summary

- Polymer supports one way and two way binding using the binding annotation [[ ]] and {{ }} respectively.
- When binding to style, href, class, for or data-* attributes, it is recommend that you use attribute binding syntax.
- Two-way bind to native elements specifies a custom change event name in the binding annotation.
- Sub-properties of the host can be binded by specifying a path inside the binding annotation.
- Paths do not support array access notation (such as users[2]). String keys (such as users[bob]) can be replaced with dotted paths (users.bob).

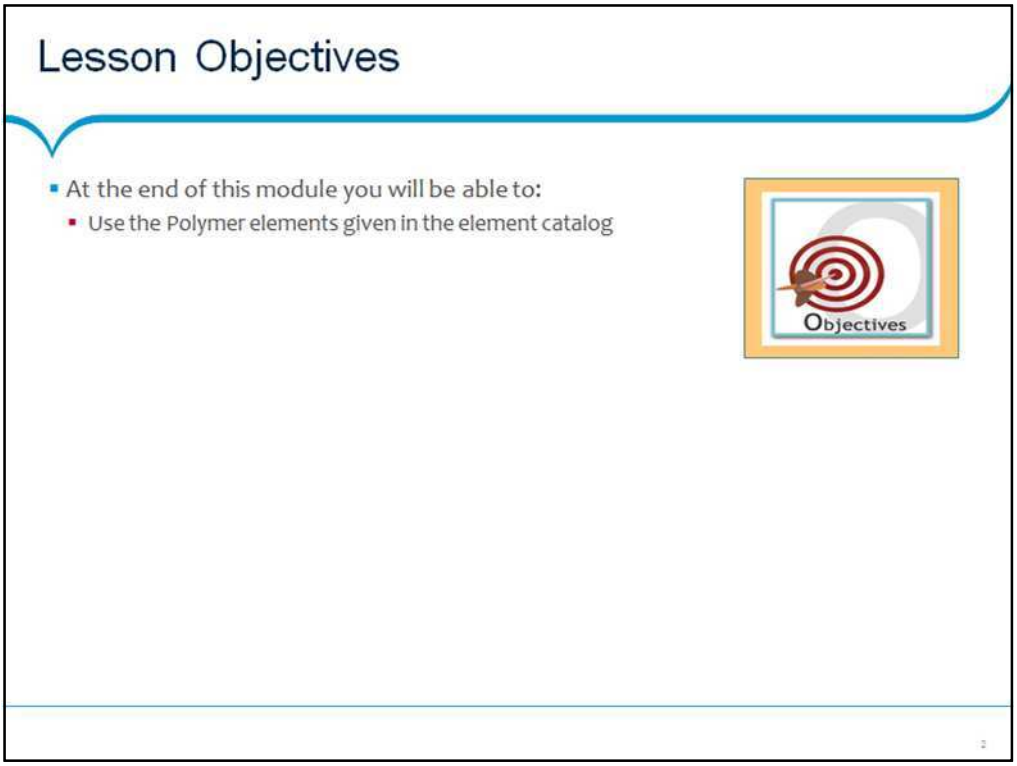
Instructor Notes:

Add instructor notes here.



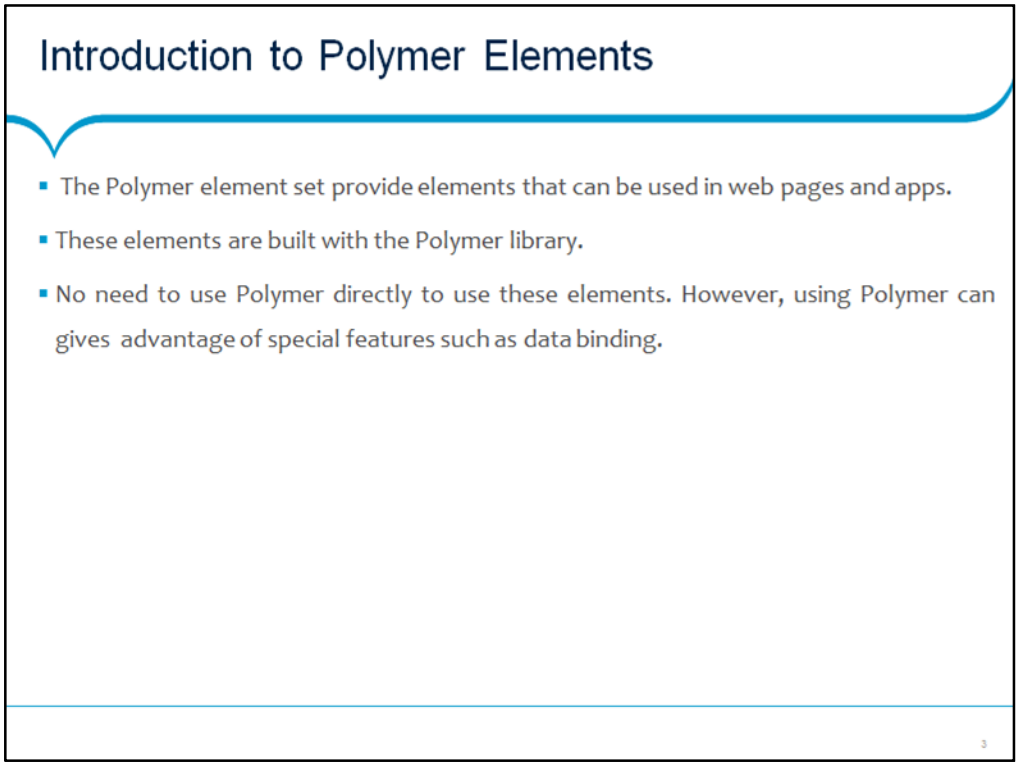
Instructor Notes:

Add instructor notes here.



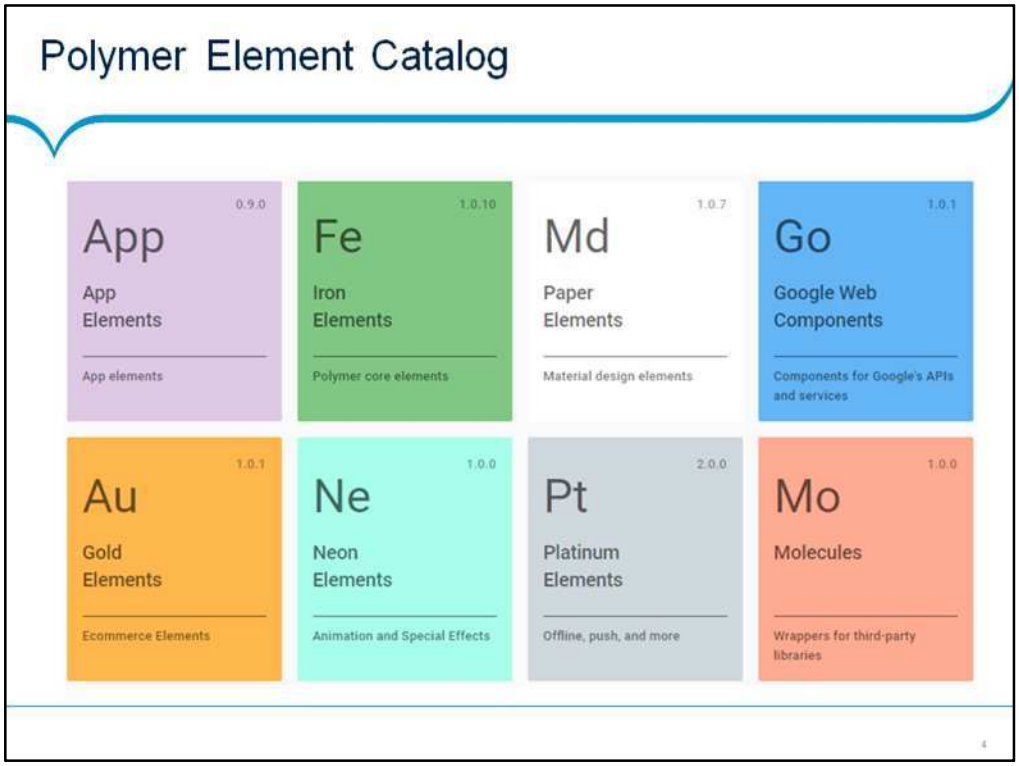
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Polymer contains a few primary collections of elements:

App Elements: It enables building full web apps out of modular custom elements.

Iron elements : A set of utility elements including generic UI elements (such as icons, input and layout components), as well as non-UI elements providing features like AJAX, signaling and storage.

Paper elements : A set of UI elements that implement the material design system.

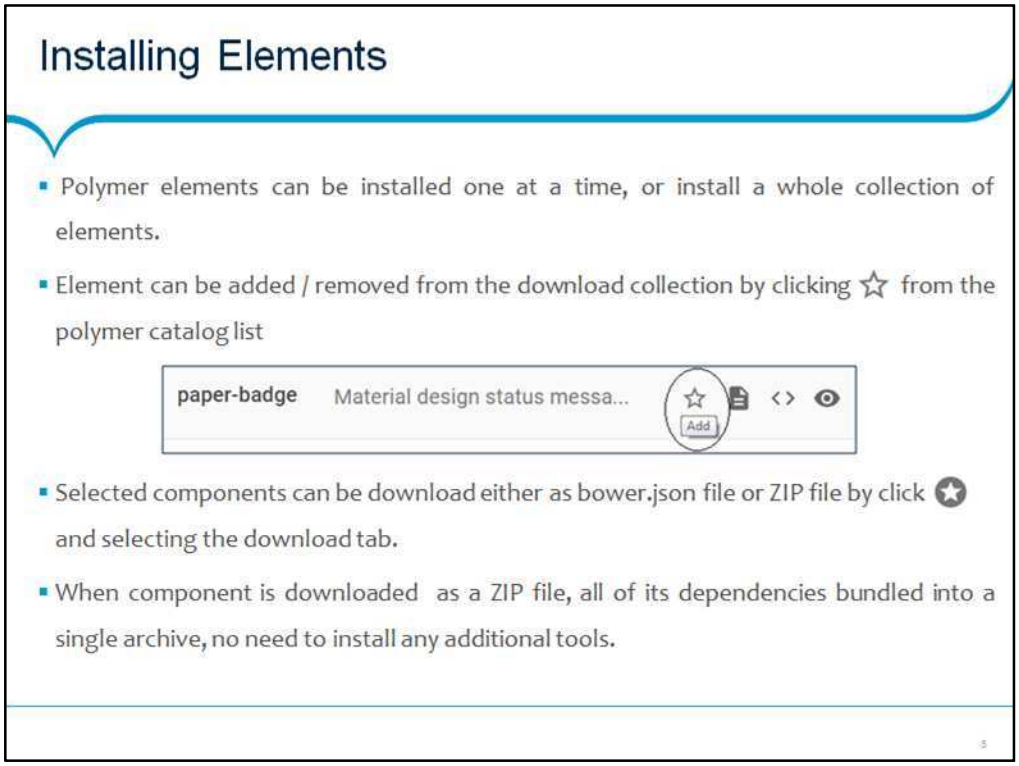
Gold elements: Form elements for ecommerce.

Neon elements: Animation-related elements.

Platinum elements : Elements for app-like features, like push notifications, offline caching and bluetooth.

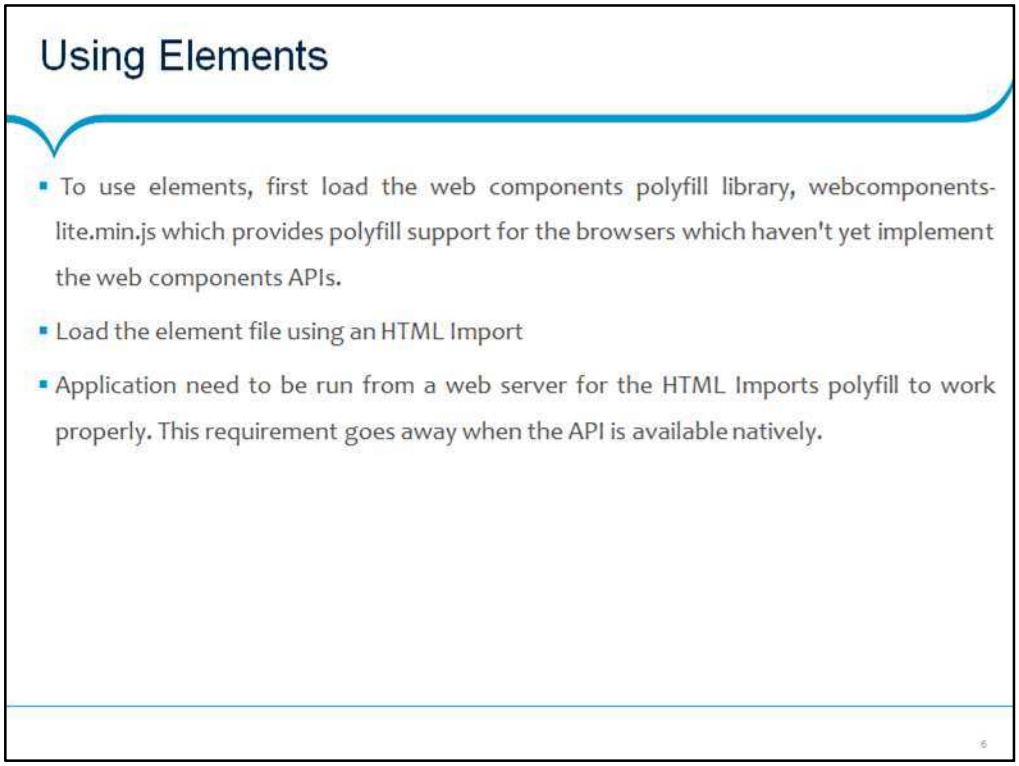
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

To use Polymer elements effectively, it's important to be familiar with a few of the common Polymer element APIs.

The Polymer element docs list element properties, but not element attributes.

When mapping attribute names to property names:
- Attribute names are converted to lowercase property names.
- Attribute names with dashes are converted to camelCase property names by capitalizing the character following each dash, then removing the dashes.

HTML attributes are string values, but many Polymer elements understand attribute values that are a JSON-serialized object or array.

Polymer elements also support data binding but only In the local DOM template of a Polymer element or Inside an autobinding template.

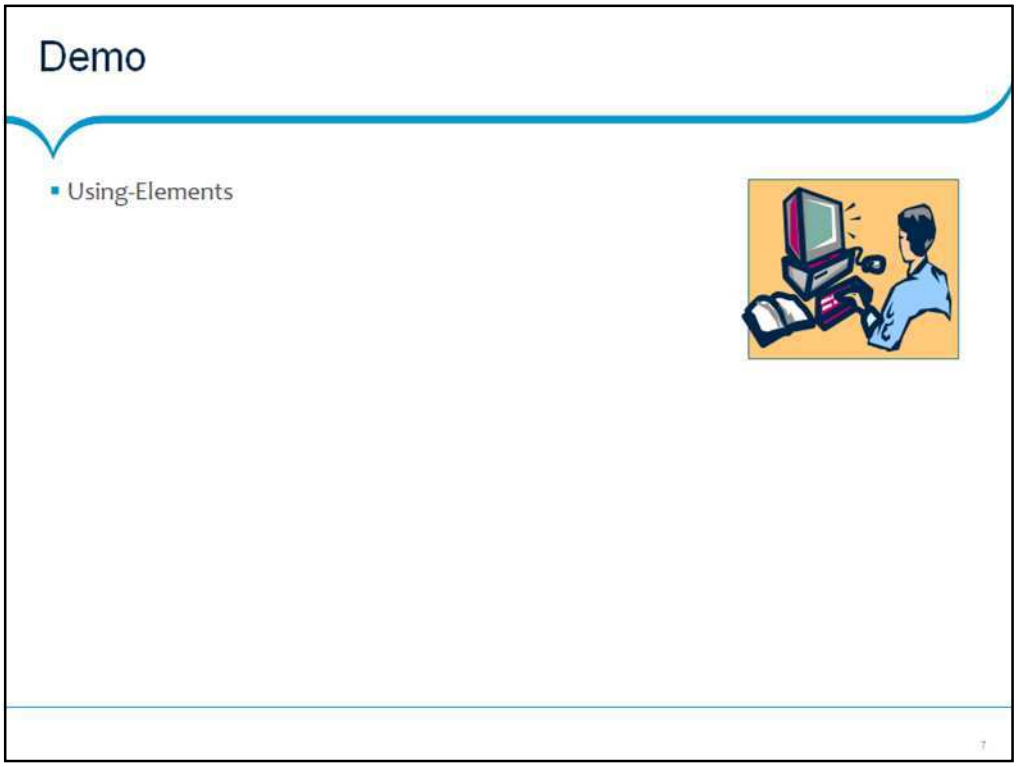
Polymer element, need to use the Polymer.dom APIs to maintain the correct DOM structure. This is particularly important when adding or removing children from a Polymer element

Many of the Polymer elements can be styled using custom CSS properties and mixins.

Property and mixin values can be specified:
- Inside a Polymer element's <style> tag to specify values scoped to that element.
- Inside a custom-style element to specify values in the document scope.

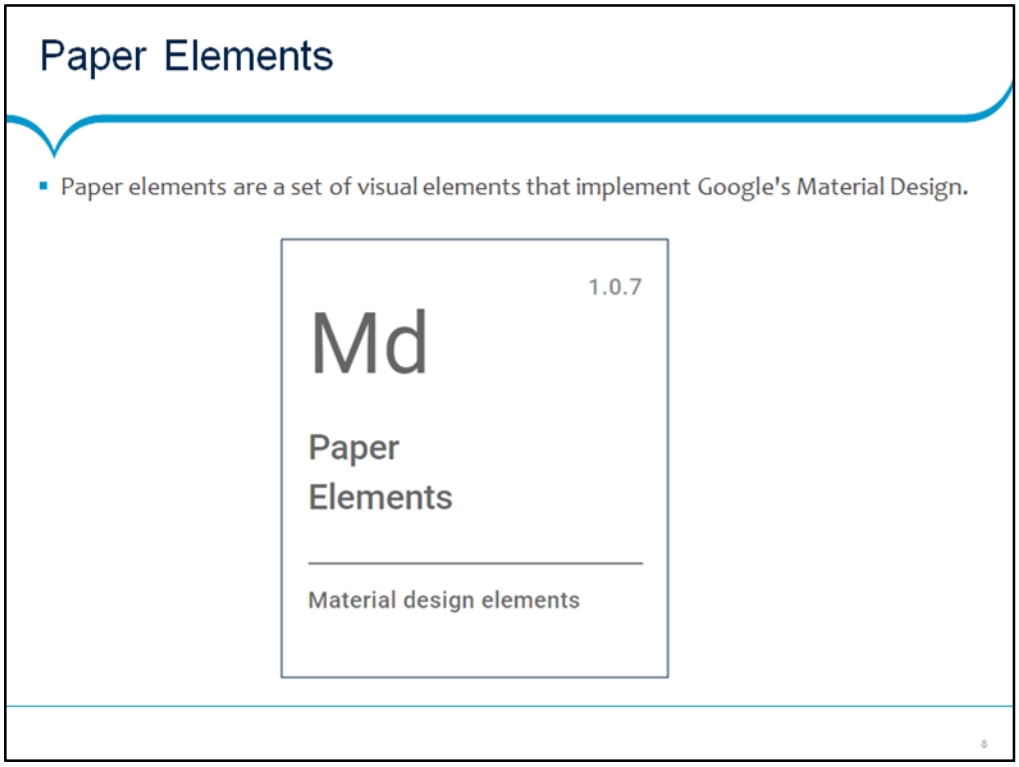
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



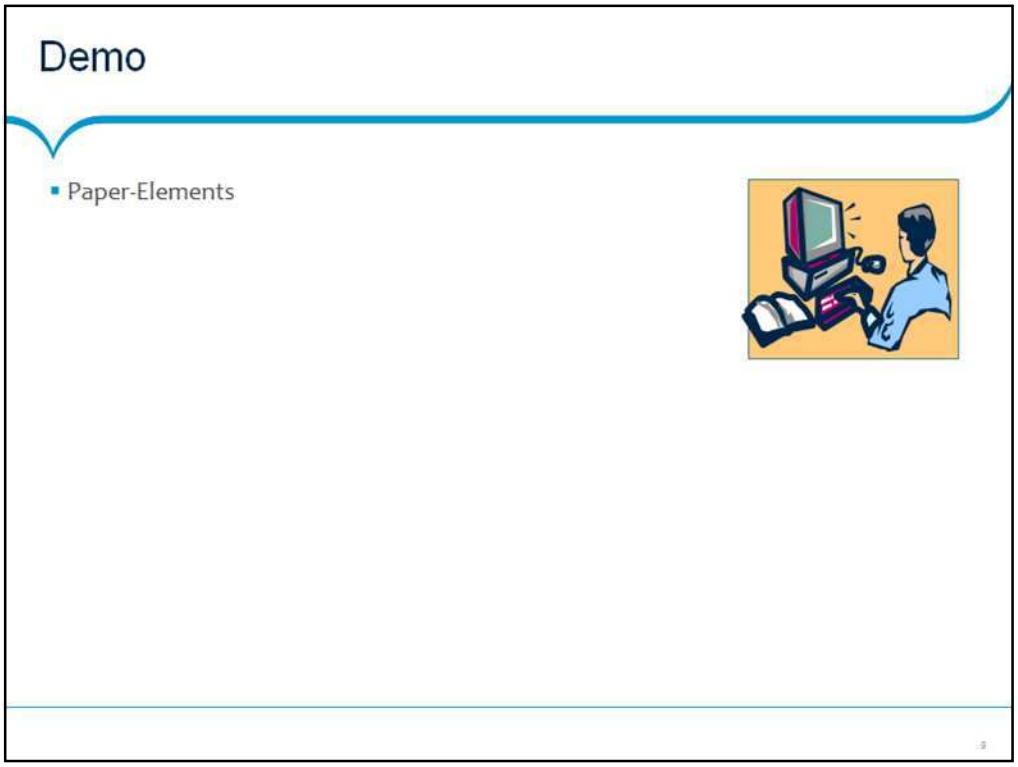
The Paper Elements is a group of Material Design elements.

Material Design is Google design language to makes user interface and experience across Google platforms both the Web and Android apps more visually consistent.

Some elements that are unique to Material Design are Paper and Floating Action Button (FAB).

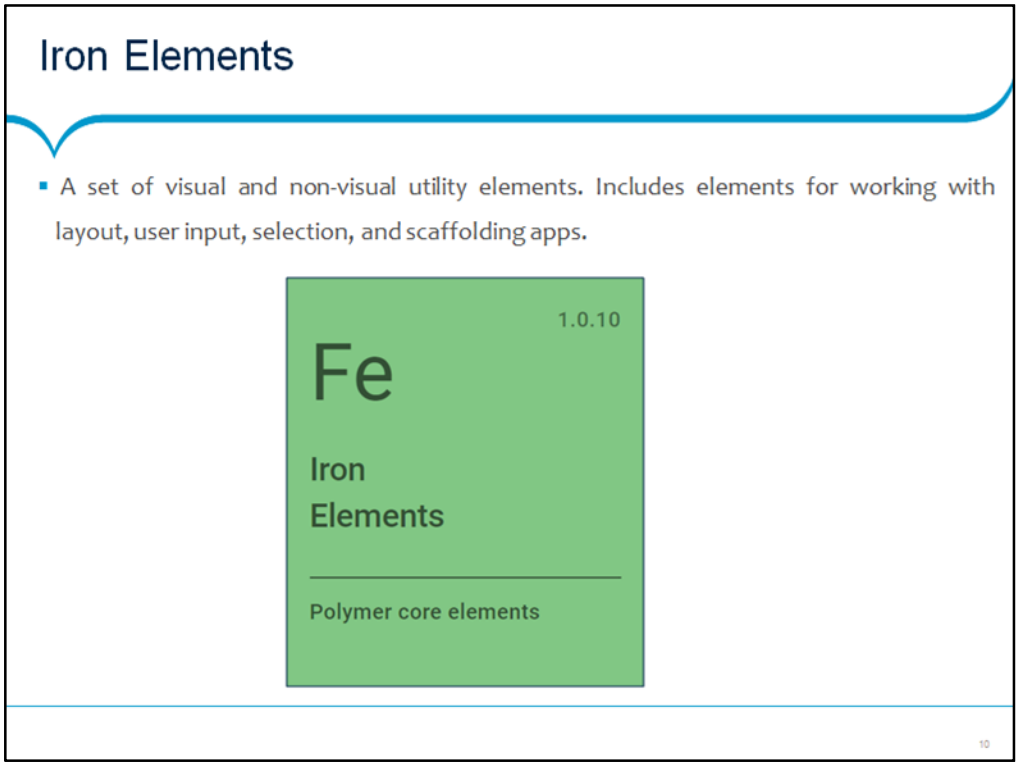
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Iron Elements is a collection of basic elements. These basic elements are what we normally use to build a web page such as an input, form and image. The difference is Polymer adds some extra powers to these elements.

Instructor Notes:

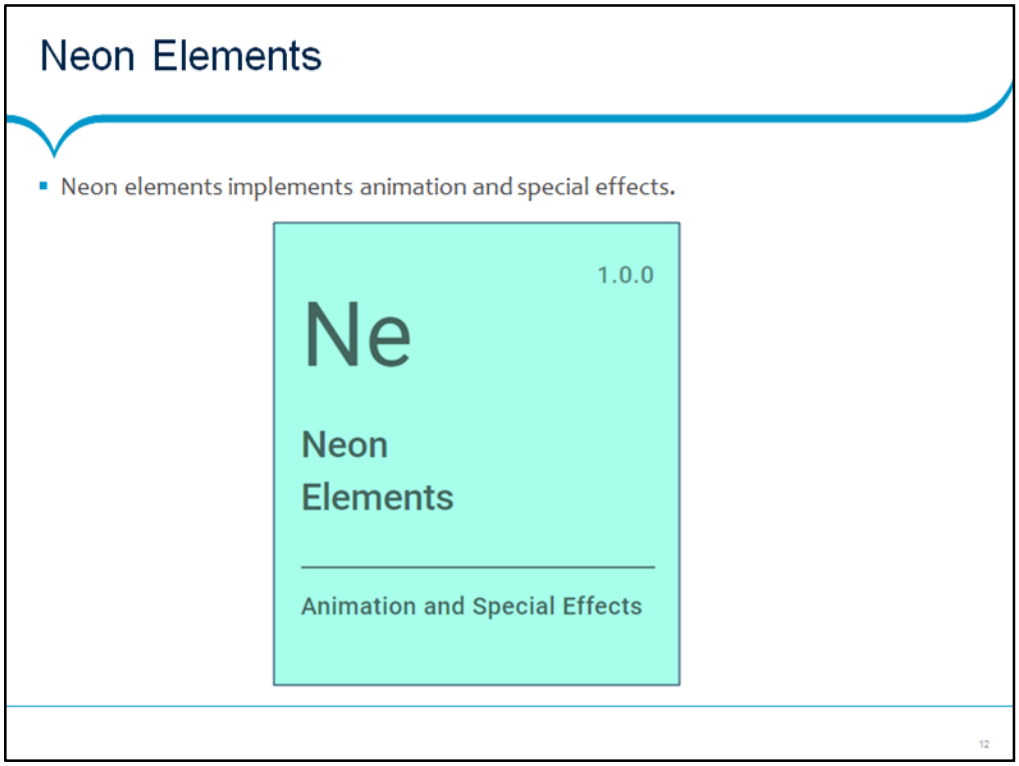
Add instructor notes here.

Demo

- Iron-Elements
- dom-bind-demo
- iron-ajax-demo
- iron-iconset-svg-demo

Instructor Notes:

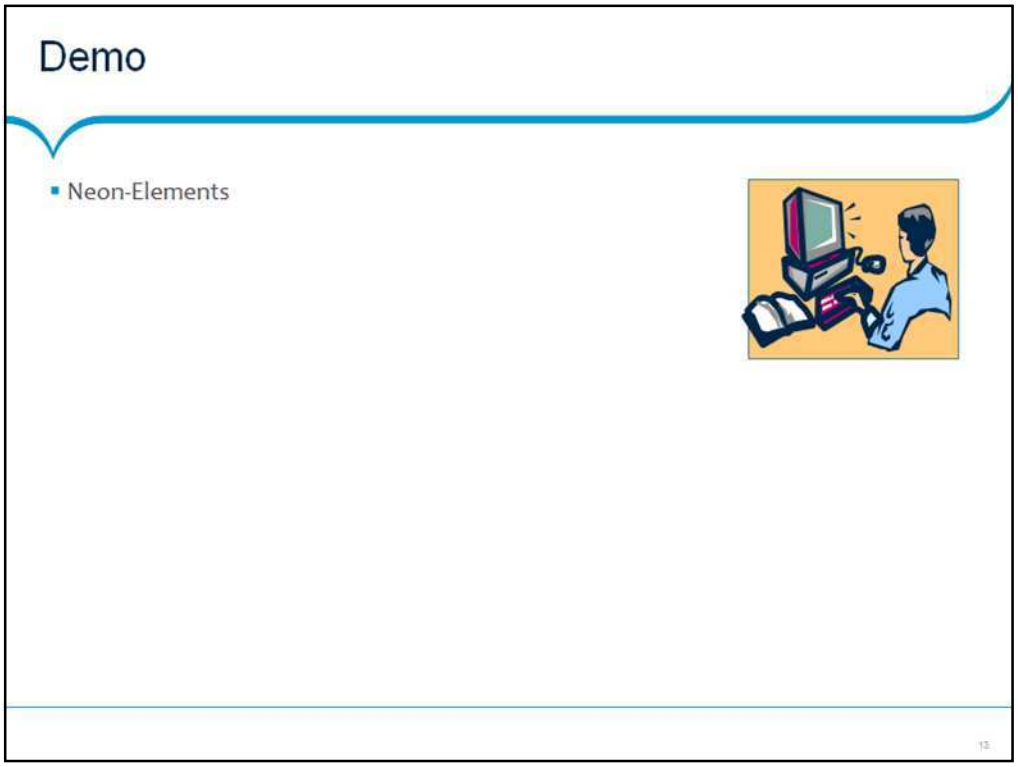
Add instructor notes here.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

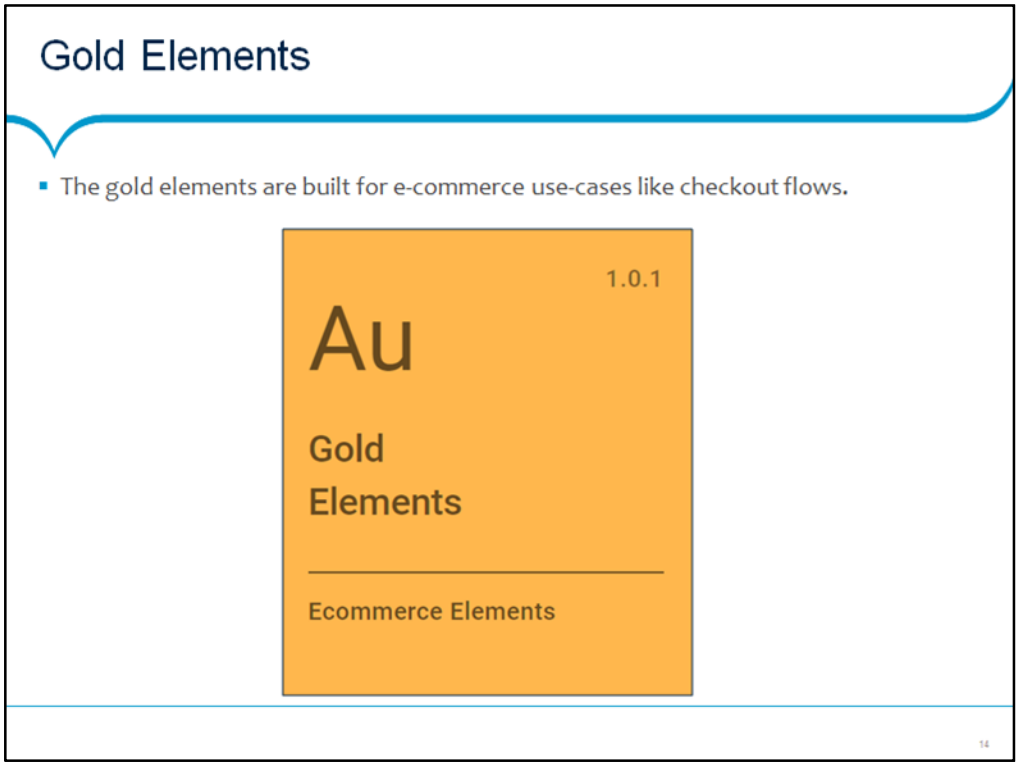
Instructor Notes:

Add instructor notes here.



Instructor Notes:

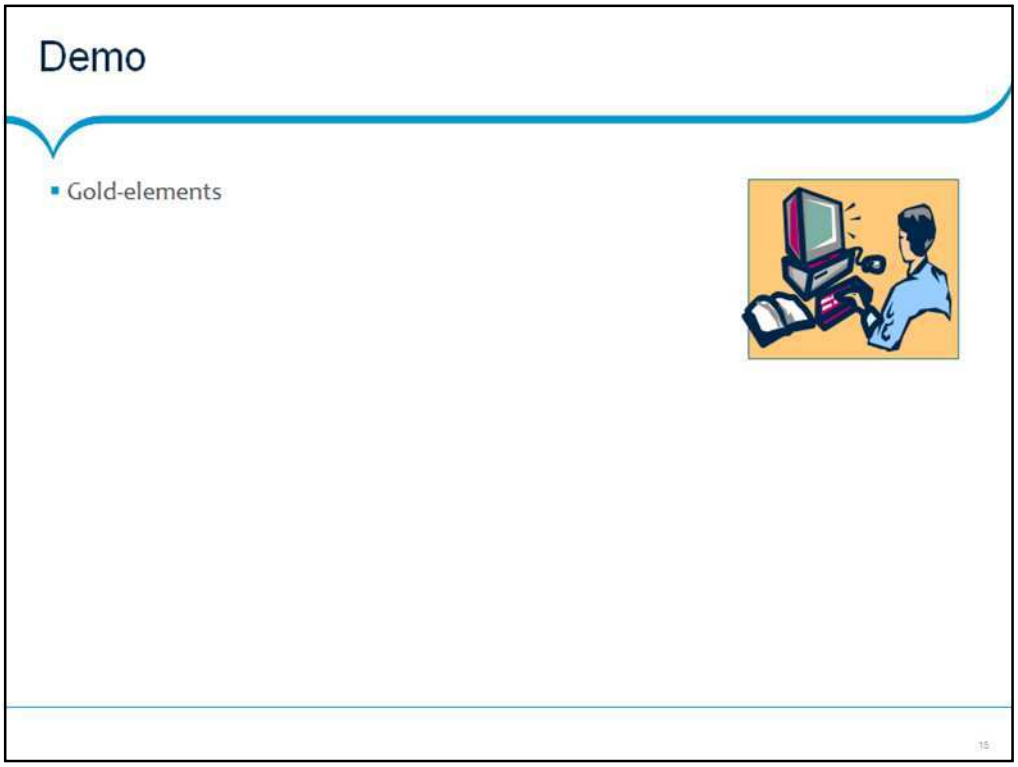
Add instructor notes here.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

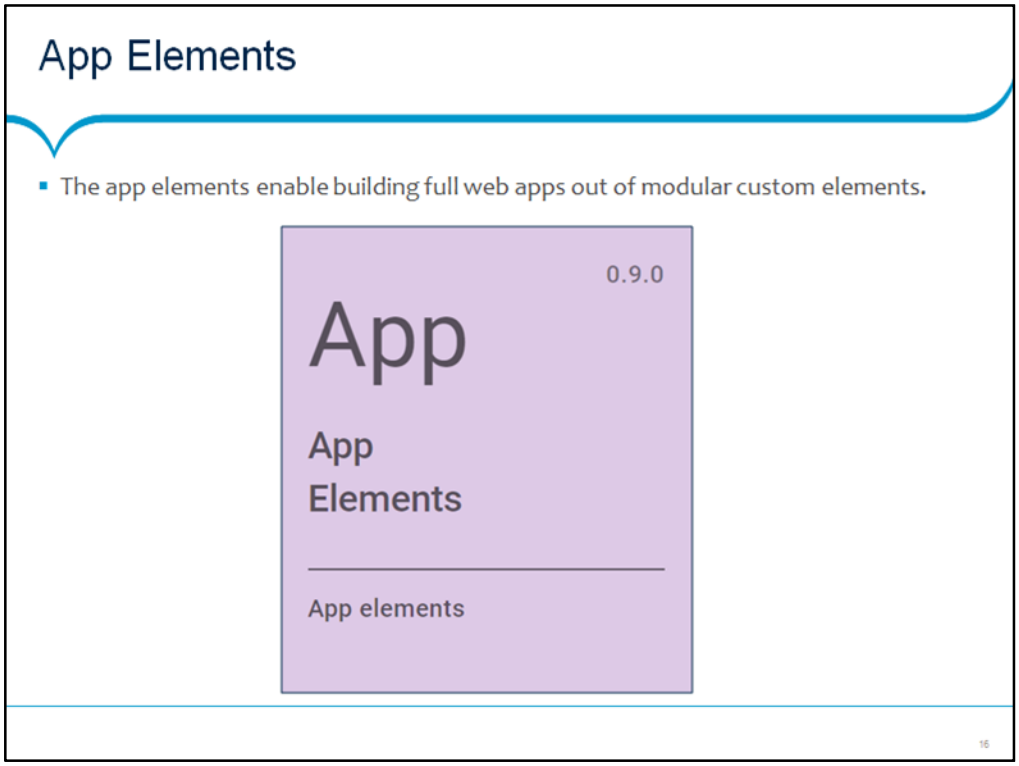
Instructor Notes:

Add instructor notes here.



Instructor Notes:

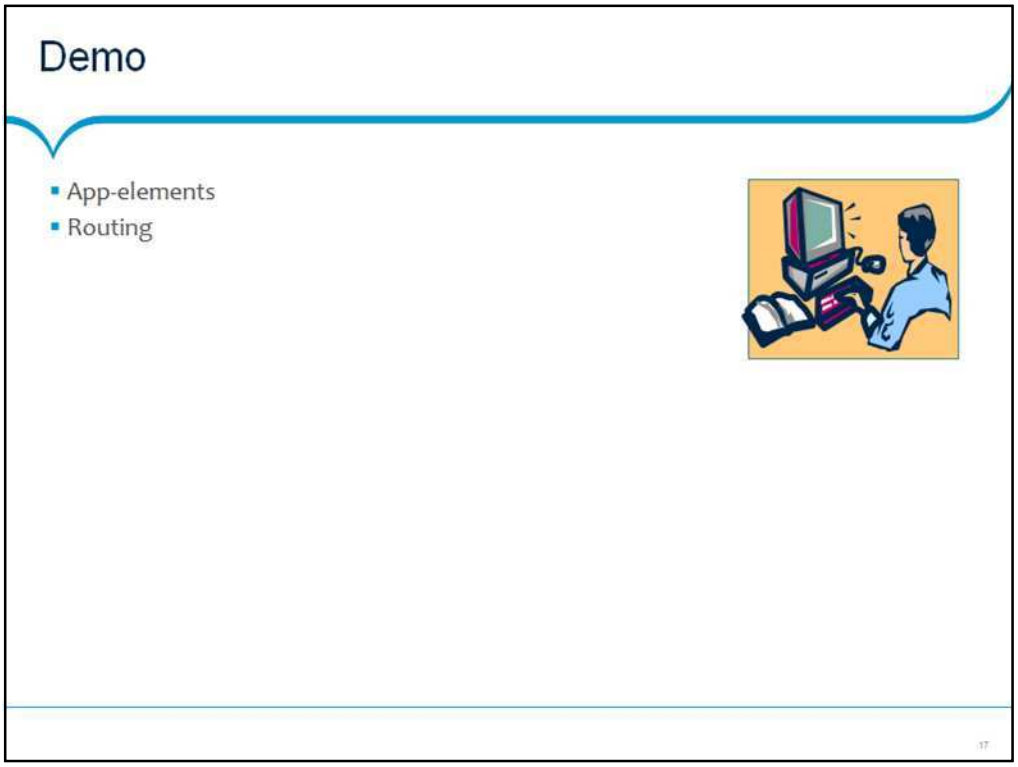
Add instructor notes here.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

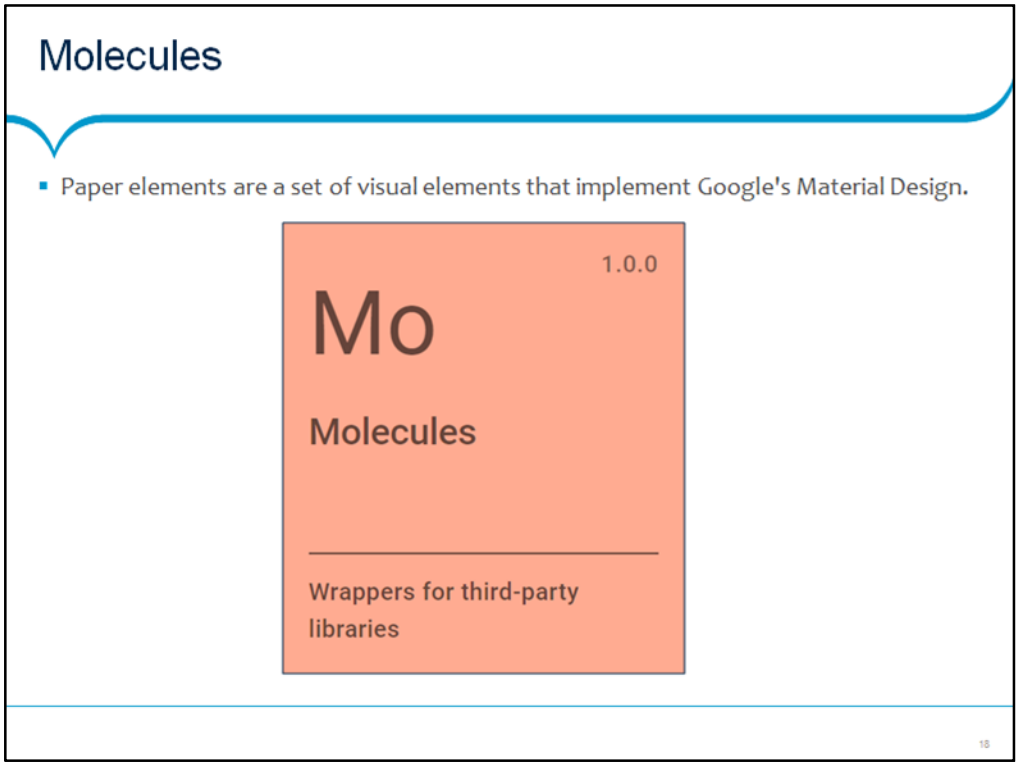
Instructor Notes:

Add instructor notes here.



Instructor Notes:

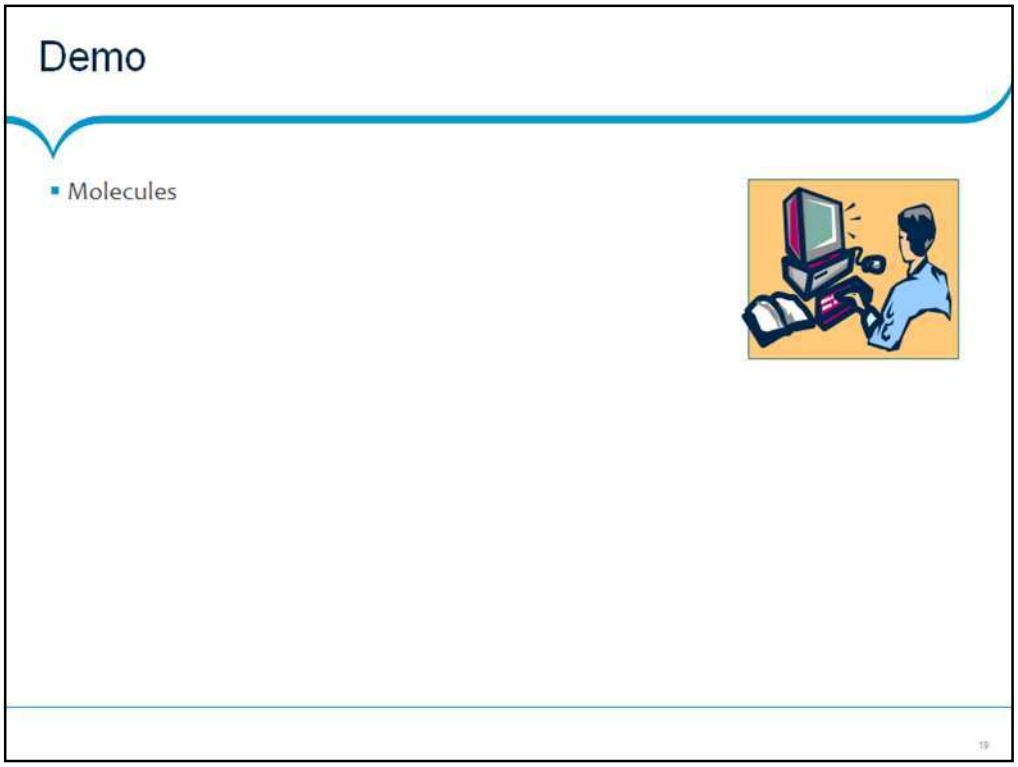
Add instructor notes here.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

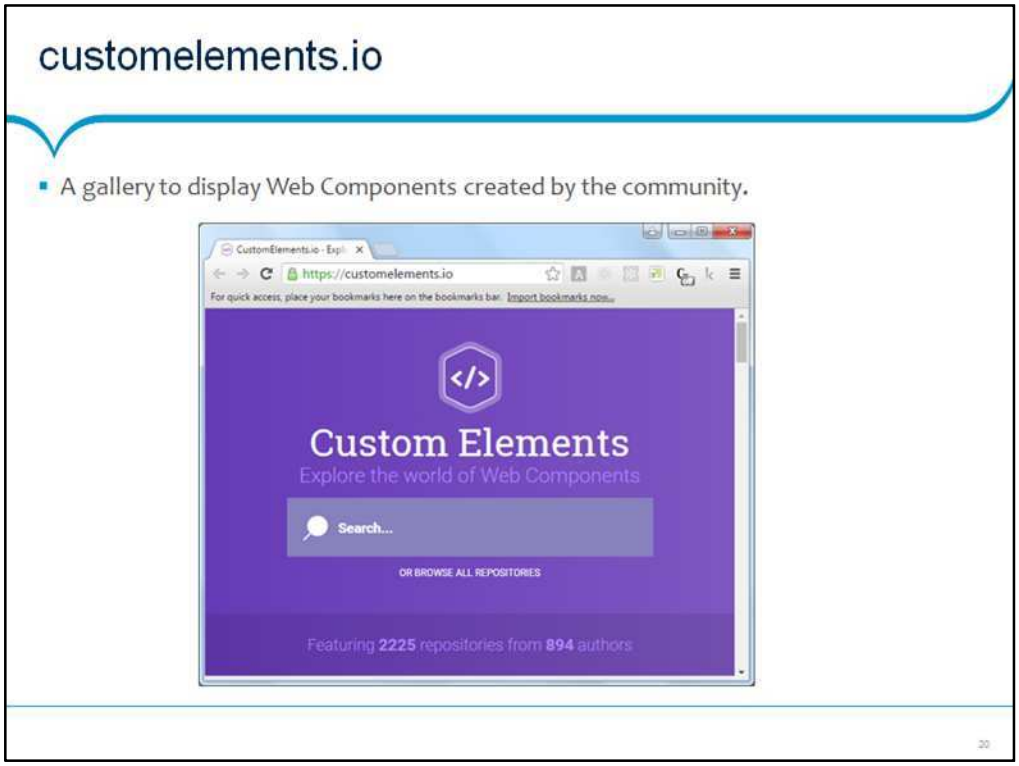
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.



Be sure to include *webcomponents-lite.min.js* before any code that touches the DOM.

Instructor Notes:

Add instructor notes here.

Summary

- Material design UI elements implemented using Polymer Paper elements.
- Iron Elements is a collection of basic elements.
- Gold Elements are the elements designed specifically for e-commerce apps.
- Neon elements are used for animation and special effects.
- Platinum elements for offline and push notifications.
- Google Web Components are special elements that cope with Google APIs and services such as Google Maps, Youtube, as well as Google Feed.
- CustomElements.io is a hub where we can find custom elements built with Web Components.