Polymer

A JavaScript Library for Designing Custom Elements using Web Components

Why Use Web Components?

```
<body>
 <!-- menu --->
 <ul>
   <li><a href=”#home”>Home</a></li>
   <li><a href=”#promo”>Weekly Deals</a></li>
   <li><a href=”#search”>Search</a></li>
   <li><a href=”#orders”>Orders</a></li>
   <li><a href=”#login”>Signin</a></li>
 </ul>
 <div id=”homescreen”>
   <!-- details of home screen here -->
   <table>
     <tr>_____</tr>
     <tr>_____</tr>
   </table>
 </div>
 <div id=”promoscreen”>
   <!-- details of home screen here →
   <span>Don’t miss this one-time offer:</span>
   <ol>
     <li>
   </ol>
 </div>
 <div id=”searchscreen”>
   <span>What are you looking for?</span>
   <form _____>
   </form>
 </div>
</body>
```

```
<body>
  <main-menu>
    <menu-item>Home</menu-item>
    <menu-item>Promotion</menu-item>
    <menu-item>Search</menu-item>
    <menu-item>Orders</menu-item>
    <menu-item>Signin</menu-item>
  </main-menu>
  <page-tabs>
    <tab-item><home-screen></tab-item>
    <tab-item><promo-screen></tab-item>
    <tab-item><search-prod></tab-item>
    <tab-item><order-list></tab-item>
    <tab-item><sign-in></tab-item>
  <page-tabs>
</body>
```

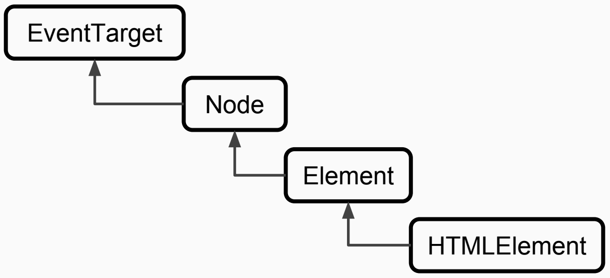
Web Components APIs

1. Custom DOM Elements
2. Shadow DOM
3. HTML imports
4. HTML templates

1. Custom Elements

HTML builtin elements

Tag Name	JavaScript *Class* Name (not CSS class)	Parent Class
`<body>`	`HTMLBodyElement`	`HTMLElement`
`<p>`	`HTMLParagraphElement`	`HTMLElement`
`<input>`	`HTMLInputElement`	`HTMLElement`
`<div>`	`HTMLDivElement`	`HTMLElement`


$$\begin{tikzcd}
\text{EventTarget} & & & \\
& \text{Node} \arrow[ul] & & \\
& & \text{Element} \arrow[ul] & \\
& & & \text{HTMLElement} \arrow[ul]
\end{tikzcd}$$

DOM Custom Elements

- Define a new class that inherits `HTMLElement` (or its descendants)
- Associate the class with a (unique) tag name
- Use the tag name in your HTML document

DOM Custom Elements

```
/* Step 1: Definition (in JavaScript) */
class MyElement extends HTMLElement {
  /* detailed definition goes here */
}
```

```
/* Step 2: Registration (in JavaScript) */
window.customElements.define(‘my-element’, MyElement);
```

Browser support
- Chrome
- Opera
- Safari

```
<!--  Step 3: Usage (in HTML) -->
<body>

  <my-element></my-element>

</body>
```

Custom Element Lifecycle Methods

- Constructor: *called when the element is **created***.
  - Should be used to setup initial state and default values, and event listeners
  - In general, initialization work should be deferred to connectedCallback() as much as possible (especially work that requires fetching resources or rendering)
- connectedCallback(): *called when the element is **inserted** into the DOM*
- disconnectedCallback: *called when the element is **removed** from the DOM. Practical use: run cleanup code*
- attributeChangedCallback(): *called when an **attribute changes** (added, removed, updated)*

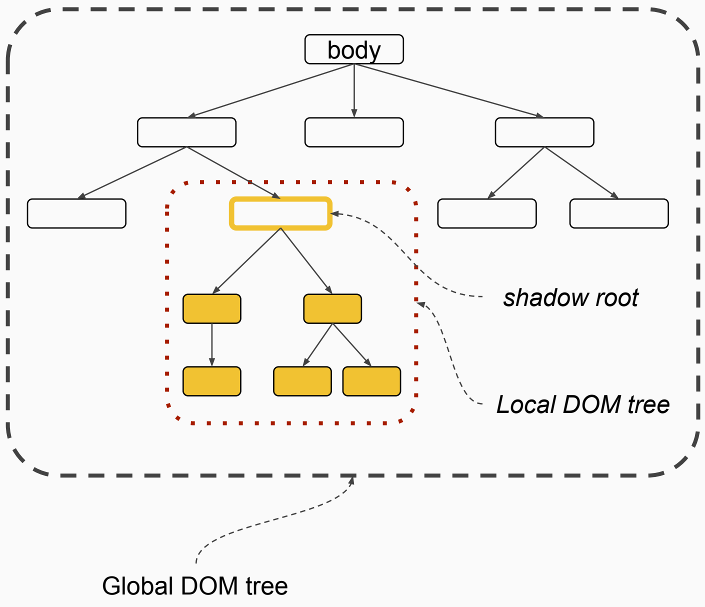
Rules on Custom Elements

1. The name of the associated tag (not the class) must be in (*and begin with*) a lowercase, contains a dash (-),
   a. Invalid tag names: `mylist`, `shopping_cart`, `-fancy-icon`
   b. Valid tag names: `collapsible-list`, `game-badge`, `my_game-badge`
2. Can't register the same tag name more than once
3. Custom elements cannot be self-closing, you must use the tag with its closing tag

2. Shadow DOMs

Shadow DOMs

- Global vs. Local DOM Trees
- Ability to insert a DOM (sub)tree into a node of the global DOM tree
- 2-Way Isolation & *Localized operations*



2-way Isolation & Localized Operations

- `document.getElement___()` on the global DOM will not return a node in a local DOM
- CSS styles defined in a local DOM do not leak out to the global DOM (and vice versa)
- Creating a shadow DOM/root: ________`.attachShadow()`

Example

```html
<html lang="en">
<head>
  <script src="./main.js"></script>
  <style>
  b { color: red }
  </style>
</head>
<body>
  <hello-world></hello-world>
  <b>Bonjour le Monde!</b>
</body>
</html>
```

```javascript
/* in main.js */
class HelloWorld extends HTMLElement {
  constructor() {
    super();    /* call the super class constructor */
    this.attachShadow ({mode:'open'});
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = "<b>Hello World!</b>";
  }
}
//------------------
customElements.define ('hello-world', HelloWorld);
```

Hello World! Bonjour le Monde!

<b> inside the shadow DOM is not affected by the global style

"Closed" Shadow Root

```html
<html lang="en">
<head>
  <script src="./main.js"></script>
  <style>
  b { color: red }
  </style>
</head>
<body>
  <hello-world></hello-world>
  <b>Bonjour le Monde!</b>
</body>
</html>
```

```javascript
/* in main.js */
class HelloWorld extends HTMLElement {
  constructor() {
    super();
    this.attachShadow ({mode:'closed'});
  }

  connectedCallback() {
    this.shadowRoot.innerHTML = "<b>Hello World!</b>";
  }
}
//------------------
customElements.define ('hello-world', HelloWorld);
```

`this.shadowRoot` is null

3. HTML imports

What's the Problem?

So many different ways for "loading" external contents into an HTML page

Contents to "load"	Technique
JavaScript code	`<script src="___">`
Stylesheet	`<link rel="stylesheet" href="___">`
Image	`<img src="_____">`

Video	`<video width="640" height="480">`
	`  <source="_____">`
	`</video>`

Another HTML doc	**`AJAX!`**

Need AJAX to load HTML docs?

Really???

17

Using HTML import

```
<!-- simplest use case -->
<head>
  <link rel=”import” href=”/path/to/files/stuff.html”>
</head>
```

```
<!-- with event handling →
<script>
function handleCompleted(ev) {
  console.log(“Loaded “, ev.target.href);
}

function handleError(ev) {
  console.log(“Import error “, ev.target.href);
}

</script>

<head>
  <link rel=”import” href=”/path/to/files/stuff.html”
    onload=”handleCompleted(event)”  onerror=”handleError(event)”>
</head>
```

18

`<link rel="import" ...>` does NOT mean copy here

You need to write a script to use the imported contents

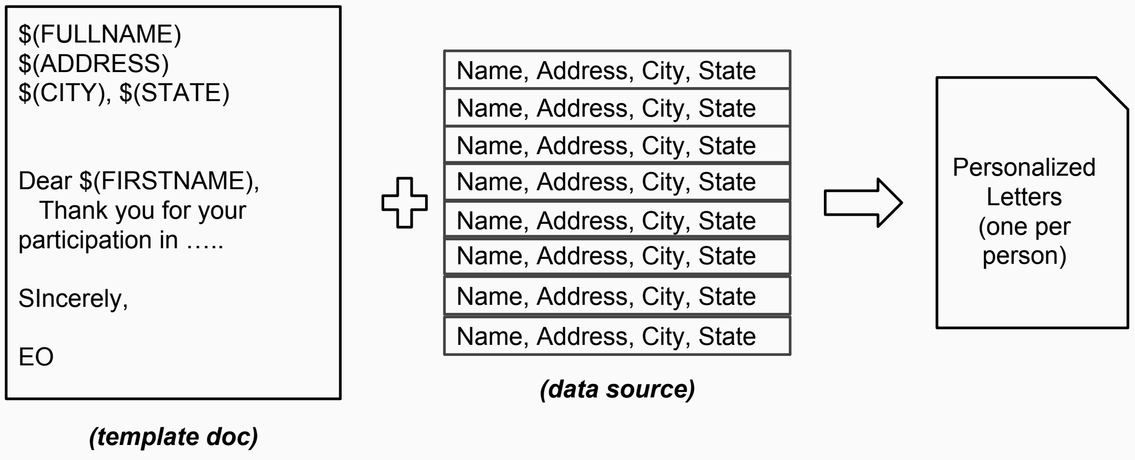
HTML import example

```
<!-- extra.html -->
<span id="top">Wing Span</span>
```

```
<html>  <!-- main.html -->
<head>
  <link rel="import" href="extra.html">
</head>
<body>
  <script>
    // (1) get the the link
    var ext = document.querySelector('link[href="extra.html"]');
    // ext.import is a full-fledged HTML doc (includes html, head, body, etc)
    // (2) target node of imported content
    var topws = ext.import.getElementById("top");
    document.body.appendChild(topws.cloneNode(true));     // true: deep clone
  </script>
</body>
</html>
```

4. HTML templates

Personalized Letter of Appreciation?


$(FULLNAME)
$(ADDRESS)
$(CITY), $(STATE)
Dear $(FIRSTNAME),
Thank you for your participation in …..
SIncerely,
EO
(template doc)
Name, Address, City, State
Name, Address, City, State
Name, Address, City, State
Name, Address, City, State
Name, Address, City, State
Name, Address, City, State
Name, Address, City, State
Name, Address, City, State
(data source)
Personalized Letters (one per person)

HTML Template

- Imported HTML Contents
- Parse as HTML, but contents are *unused* at **load time**
  - Scripts don't run
  - Images don't load
  - Audio/Video files don't play
  - Contents are considered NOT to exist in the document
- Actual contents are instantiated at **run-time**
  - Scripts will run, images will load, ….

HTML template example

```
<html>  <!-- main.html -->
<body>
  <template id="sample">
    Attempt: <b>plate</b>
  </template>
  <script>
    // (1) get the the link
    var tmp = document.getElementById("sample");
    // (2) instantiate the content
    var plt = document.importNode(tmp.content, true);    // true: deep clone
    document.body.appendChild(plt);
  </script>
</body>
</html>
```

Beware of mixing up function names

```
// HTML imports
var tmp = document.getElement______();
var imp = tmp.import.getElement_____();
var node = imp.cloneNode(true);

document.appendChild(node);
```

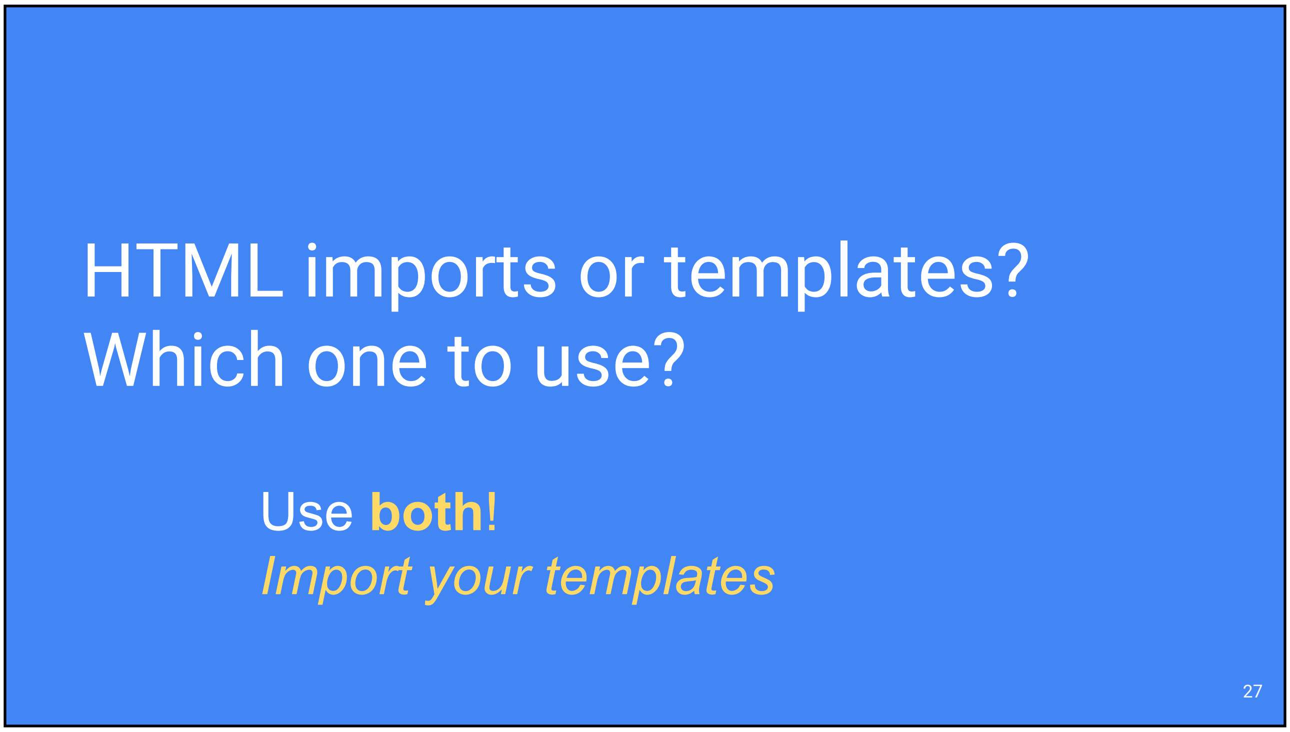
```
// HTML templates
var tmp = document.getElement_____();

var node = document.importNode (tmp.content,
                               true);
document.appendChild(node);
```

Import vs. Template

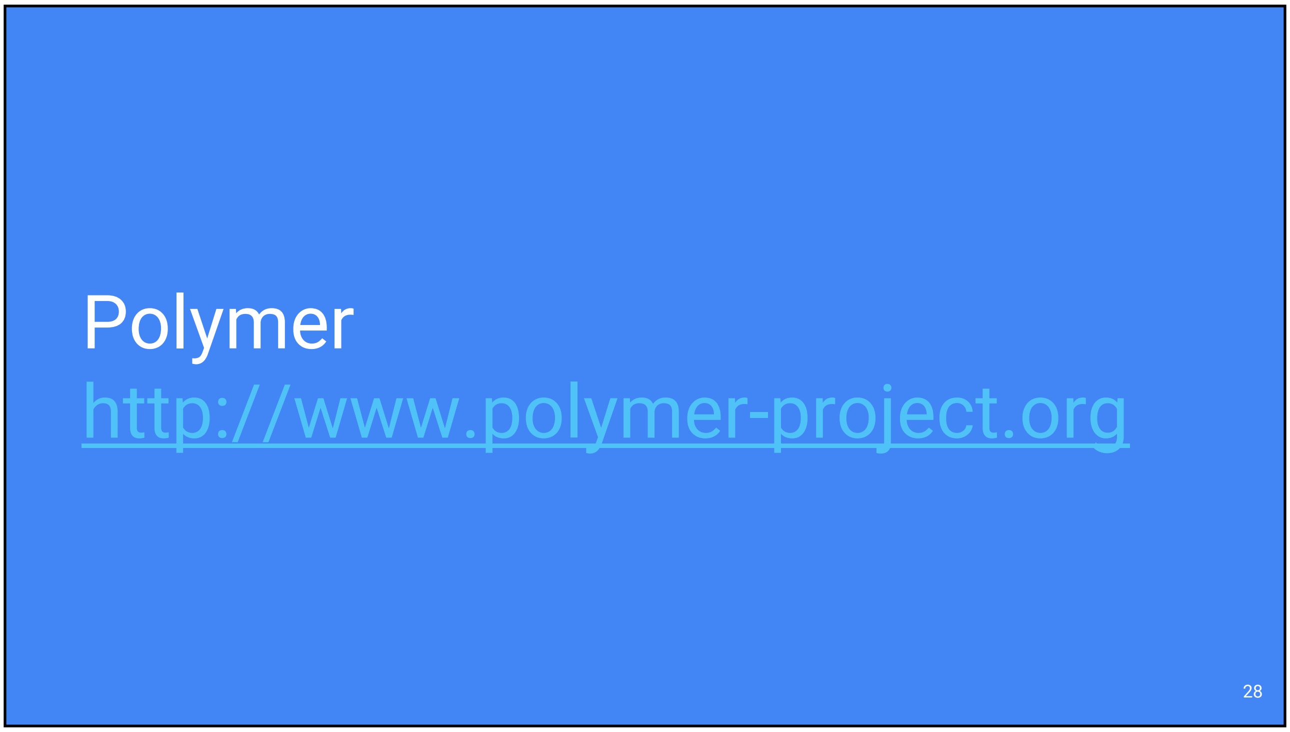
- Import a **full-fledged** HTML doc (default)
- Use `<link rel=”import” _____>`
- document.getElement___ or document.querySelector____ to obtain a reference to the `<link>`
- `_______.`**`import`** is the imported content
- Invoke `____.cloneNode(____)` to instantiate contents

- Define a **snippet** of HTML doc
- Use `<template>________</template>`
- document.getElement___ or document.querySelector____ to obtain a reference to the `<template>`
- `______.`**`content`** is the template content
- Invoke `document.importNode(____)` to instantiate contents



HTML imports or templates? Which one to use?

Use **both**!
Import your templates



Polymer

http://www.polymer-project.org