Web Components

Five Problems, One Solution

1. Undescriptive markup.
2. No component standard.
3. Style conflicts.
4. No native templates.
5. No bundling.

Web Components are built using a combination of four new technologies:

- Templates: Allow us to define inert, reusable markup using web standars.
- Custom Elements: Give us the power to extend HTML by defining our own elements.
- Shadow DOM: Provides encapsulation of our markup and our styling.
- Imports: Support bundling HTML, JS and CSS files together into a single, simple, reusable package.

#1 Templates

Where do we put this?

```
<tr>
    <td class="first-name"></td>
    <td class="last-name"></td>
    <td class="address"></td>
    <td class="phone"></td>
</tr>
```

Now we can use the **<template>** tag.

```
<template>
    <tr>
        <td class="first-name"></td>
        <td class="last-name"></td>
        <td class="address"></td>
        <td class="phone"></td>
    </tr>
</template>
```

Templates Characteristics

1. All markup in the <template> tag is inert; it doesn't render; any script inside does not run, it does nothing. The markup inside is totally inert until it's cloned and utilized on the page. Also an <img> tag with a bad path wouldn't throw a 404 error, and any kind of javacript won't run a video wouldn't play,

etc.

2. The content of the <template> tag is hidden from a traditional selection techniques. You can't use JS to select any child nodes in a template, and CSS selectors won't affect anything.
3. You can put the <template> tag anywhere in the page, in the <head>, <body>, a <select>, etc.

Template Activation

A template is inert until it is activated by importing the node into the DOM. So... How can we activate a template?

It is a 3-phase process similar to cloning any other HTML element:

1. First, you need to got a reference to the template:

```
var template = document.querySelector('#myTemplate');
```

2. Second, you need to use the importNode method to create a clone of the template's content.

```
var clone = document.importNode(template.content, true);
```

The content property of the template object returns everything found bettween the opening and closing template tags. The second parameter determines whether to do a deep copy; in other words, this boolean determines whether the descendents will be copy (tipically set to true, to assure that all template's content are copied).

3. Add the content to the page as desired.

```
document.body.appendChild(clone);
```

Injecting Dynamic Data

Inject data before cloning by manipulating the template clone:

```
<body>
    <template>
        <span class="verb"></span>
    </template>
</body>
```

Steps:

```
// (1) Get a reference to the template
var template = document.getelementById('myTemplate');

// (2) Use docuemnt.importNode to clone the templates's content
var clone = document.importNode(template.content, true);

// (3) Change the target element within the template as desired
clone.querySelector('.verb').textContent = 'Awesome!';

// (4) Append element to page
document.body.appendChild(clone);
```

Nested Templates

```
<template id="header">
    <div>Header</div>
    <template id="subHeader">
        <div>Sub header</div>
    </template>
</template>
```

In this cases, each template has to be cloned and added to the page manually. For example, in the example above, the #subHeader template won't be clones on to the page automatically, we still need to clone the #subHeader template as well.

```
var template = document.getelementById('header');
var clone = document.importNode(template.content, true);
document.body.appendChild(clone);

var template = document.getelementById('subHeader');
var clone = document.importNode(template.content, true);
document.body.appendChild(clone);
```