

**IIT Madras**  
ONLINE DEGREE

**Modern Application Development-I**  
**Professor. Nitin Chandrachoodan**  
**Department of Electrical Engineering**  
**Indian Institute of Technology, Madras**  
**Custom Elements**

(Refer Slide Time: 00:16)

**Custom Elements**

**Adding custom elements**

- XML allows arbitrary namespaces and tag definitions
  - Applications can be defined on top of XML tag definitions
- But HTML5 does not use the same approach
- Requirement for elements:
  - Meaning: what does a tag mean - <title>, <h1> etc OK, but is <my-button> actually a button?
  - Rendering: how should a tag be shown: provide display details for each tag
  - States: checkbox can be checked or blank - what about custom tags?
  - Customized built-in element or Autonomous custom element?

<https://html.spec.whatwg.org/multipage/custom-elements.html>

Hello, everyone, and welcome to this course on modern application development. So, so far we have looked at a high level overview of JavaScript, and we now come to the most important part, why did we bring JavaScript into the picture? We were talking about HTML to start with. And the reason why JavaScript comes into the picture is because we want to define custom elements.

So, what are custom elements? Now, XML, as we saw earlier, allows arbitrary namespaces and tag definitions. But HTML5 does not use the same approach. What it does, is that, it has specific requirements for elements, meaning that, there are elements like title, h1 and so on. And HTML or rather, let me rephrase that. HTML5 has specific elements, like title, h1, etc. But if I am allowed to define an arbitrary custom element, then should I really be allowed to say my-button? And what is the browser supposed to infer from that?

Just because the word button is there inside it does it automatically infer that, yes, this is a type of button or does it say, this is a tag, I have no idea what it means. Is it a heading? Is it a list item? Is it a button? Is it a link? Is it just a picture that's displayed on the screen? Nothing is clear from the name.

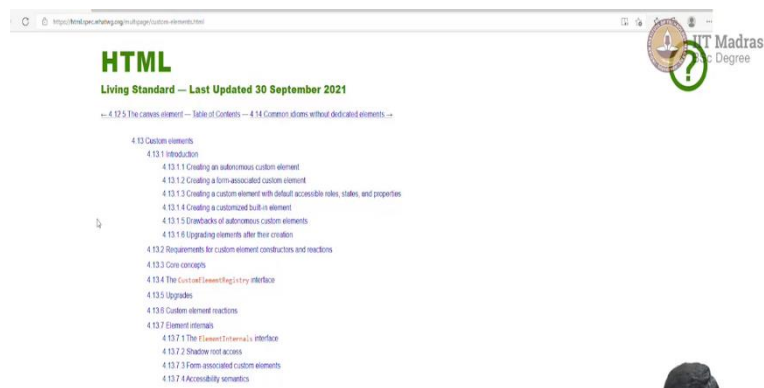
As a human being when I read the name my-button, I think it must be a button, but there is no real need. There is no nothing compulsory about that. On the other hand, there is another part to a tag element, which is to say, how should a tag be shown? This is the part that JavaScript can actually solve best for you. It basically gives you display details. In other words, if you provide a tag, then how should that tag get displayed on the screen?

Now, elements could also have states. I mean, in general, HTTP is a stateless protocol, but that does not mean that there are, you cannot have certain elements on the screen that have a state associated with them. The simple example is a checkbox. So, on screen, I might have a form where I have something to be checked off or a radio button or a checkbox. A checkbox or radio button has state associated with it. Is it checked or is it not checked?

Now, if you are defining a new custom tag of your own, maybe, it is something similar to a checkbox or maybe it is something else. The point is, it might have some custom state of its own. How do I sort of keep track of that state? How do I use that state in a useful manner? That is also part of how you define an element.

And finally, a question that we could ask is, should I restrict my customization to only taking built-in elements like headings or list items and adding functionality to them or should I be able to build a completely new autonomous custom element, meaning that, it has no relationship to any elements that are already defined or at least, it is not directly related to any of them and is completely standalone, in some sense. Now, the full details on how to use this is specified at this webpage.

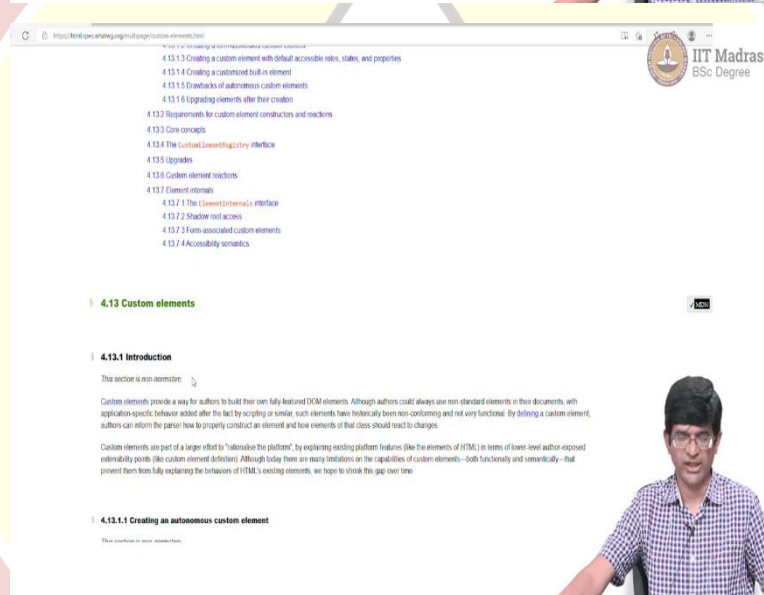
(Refer Slide Time: 03:43)



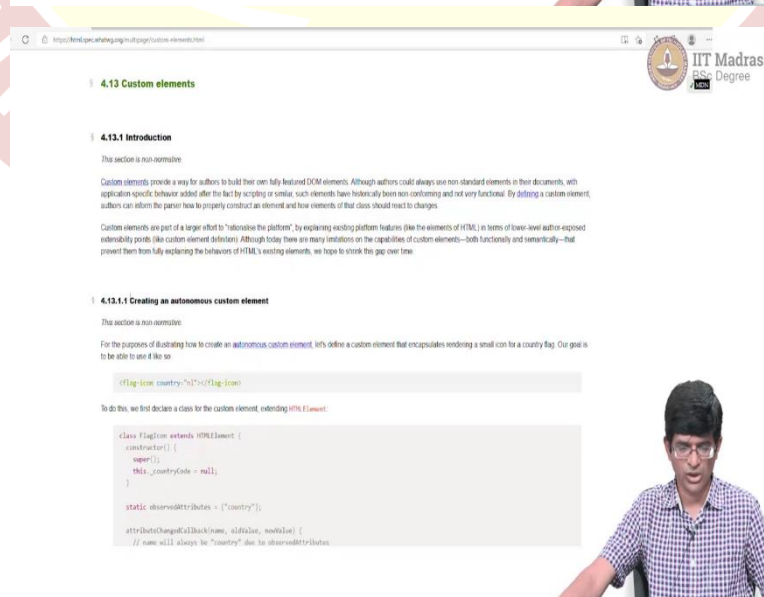
The screenshot shows the 'HTML Living Standard' website, last updated on 30 September 2021. The table of contents lists various sections, with '4.13 Custom elements' highlighted in green. The list includes sub-sections like '4.13.1 Introduction', '4.13.1.1 Creating an autonomous custom element', and '4.13.1.2 Creating a form-associated custom element'.



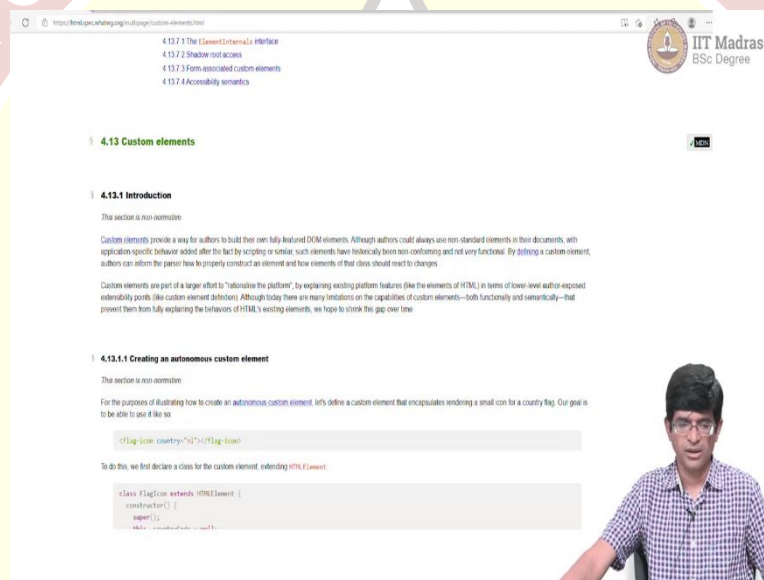
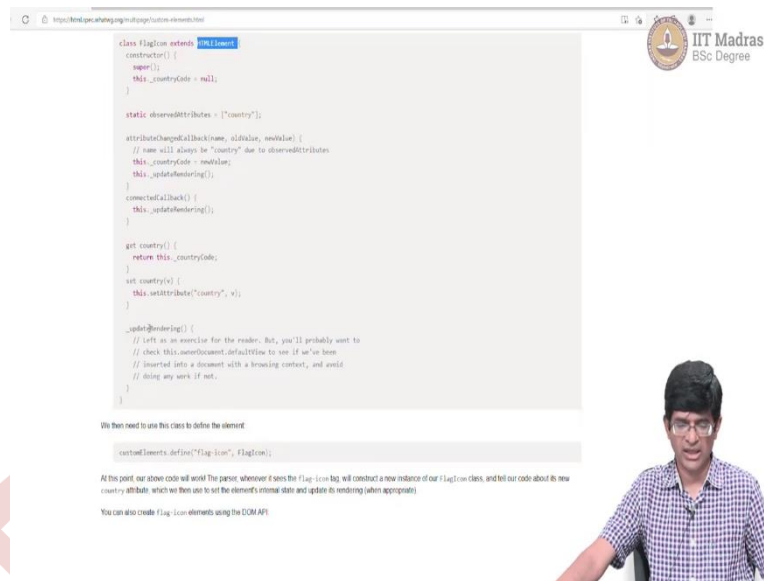
This screenshot shows the '4.13 Custom elements' section of the HTML Living Standard. It includes the sub-section '4.13.1 Introduction' and a note stating 'This section is non-normative'.



This screenshot shows the '4.13.1 Introduction' section. It explains that custom elements provide a way for authors to build their own fully featured DOM elements. It also mentions that custom elements are part of a larger effort to 'modernize the platform'.



This screenshot shows the '4.13.1.1 Creating an autonomous custom element' section. It provides a code example for creating a custom element that encapsulates rendering a small icon for a country flag. The code defines a `FlagIcon` class that extends `HTMLElement` and includes a constructor, a static `observedAttributes` array, and an `attributeChangedCallback` method.



We have this this particular webpage is what I had referred to earlier, this is the HTML living standard. And within that living standard, what we are looking at over here is section 4.13 which refers to custom elements. So, this HTML living standard, you can see that it actually shows, this was last updated on thirtieth. September. I am recording this on the first of October, which literally means that this document was updated yesterday.

On other days, it is not that it is updated every day, it might actually have been changed a few days back. And if you go look at this thing, you will find that, yes, there are a number of updates, all the changes are recorded and tracked, so you can go back and find out what it looked like at a different point in time. But the ultimate thing is that whatever exists as of now is what you need to take as the living standard, so to say.



Now, one issue with this document is that in pretty much every section you will find this disclaimer saying that this section is non-normative. Normative generally means, it is actually part of a standard. And when you say non-normative, it means, that it is potentially subject to change. Now they have to do that because part of the reason for being a living standard is that you cannot really say that this is a standard, it is not going to change from onwards, but it also means that you always have that concern that is this really what I should be following?

Anyway, the point is, this is the standard that is followed today, and it is a good sort of document. It is also a somewhat dense document, not very easy to fully understand, but it is worth sort of at least skimming through to start with. It gives you a lot of useful information about how you can create an autonomous custom element. So, for example, I could create something called a flag icon a tag that does not exist in HTML.

Now, what would this do? When I read it as a human being, I think, most likely what is going to happen is I will take the country, I will find out what flag that should be there, get a picture of that flag and display it somewhere. But how does a browser really know what to do about it? That is where you basically have to go and declare everything in JavaScript.

You have to say that there is a class FlagIcon, which extends HTML element. It has certain attributes, it has callbacks, blah, blah, blah. And the most important thing is, it also tells you how to update the rendering, that is to see what should be displayed on the screen. In this particular case, it does not actually tell you what to do so you cannot use this piece of code. But you could sort of imagine that, maybe I would need to go find one central location where the flags of all countries are there, find a picture corresponding to the flag of whichever country I need to display and put that image on the screen.

Once you have that piece of JavaScript, I can just call this `customElements.define`. It is a function called, it is basically calling the custom elements API, the application programming interface. So, the custom elements API, which is present inside any browser, that supports JavaScript allows you to define saying that flag icon this new tag will call this JavaScript function, or this class corresponds to this class FlagIcon.

Which means that from there onwards, you know, after you have done all of this, you can then go there and directly have a piece of HTML in your code, which shows `<flag-icon country="nl"> </flag icon>`. So, this document, in other words, gives you the basics of how that is supposed to be done. But it is not very easy to, I mean, it is not like a tutorial. You cannot really read this and understand how to use it.

(Refer Slide Time: 07:30)

**Web Components**

- Custom elements
  - JS API to create custom element tags
- Shadow DOM
  - API to keep styling of component separate from rest of page
- HTML Templates
  - `<template>` and `<slot>` tags to write markup templates

So, what people did is, rather than just sticking to the custom elements API, which is present over there within JavaScript, they said, let us try and make this a bit more useful. Add some more components to it, and they came up with something called a web component definition. So, a web component is something, which makes use of three aspects of HTML essentially. One is the use of custom elements, which we already saw how you can define as part of the HTML5 and JavaScript APIs. It is basically a JavaScript API that allows you to create custom element tags.

But on top of that, let us add some more thing to make it easier to use because, if I am adding a new tag I probably want to display something on the screen. And it means that I should be able to control which parts of the screen are going to change. I do not want a style change that I make over here to suddenly go and affect the entire page. I would like to be able to affect just the part that I am interested in. That is done by using another API called the Shadow DOM, Shadow Document Object Model.

And essentially, what it does at a high level, high enough level, you can think about it, it allows you to keep the styling of the component that you are defining separate from the rest of the page. It is sort of an encapsulation. And finally, there is this concept called HTML templates. Template and slot were two new tags that were introduced in HTML5, that allow you to basically do something at a high enough level, you can think of it like Jinja templates, it is not exactly the same thing, but in principle it is just allowing you to replace some text with something else.

It does not have like the kind of templating language that Jinja has, but you can do it in other ways because after all, you have access to JavaScript over here. So, the template tag basically allows you to say that, this is roughly what a template for displaying something should look like, use that in order to actually display certain things.

(Refer Slide Time: 09:33)

The image shows a presentation slide titled "Web Component Examples" with a list of five URLs. The slide is part of a video lecture, as evidenced by the presenter's video feed in the bottom right corner. The slide also features the IIT Madras BSc Degree logo in the top right corner. Below the slide, a screenshot of a web browser displays the article "An Introduction to Web Components" by @dmitrybaranovskiy. The article discusses the challenges of building complex frameworks and introduces a five-part series. The first part, "An Introduction to Web Components (This post)", is highlighted. The article also mentions that Web Components consist of three separate technologies: Custom Elements, Shadow DOM, and HTML Templates.

### Web Component Examples

- <https://github.com/mdn/web-components-examples>
- <https://mdn.github.io/web-components-examples/editable-list/>
- <https://mdn.github.io/web-components-examples/edit-word/>
- <https://mdn.github.io/web-components-examples/word-count-web-component/>
- <https://css-tricks.com/an-introduction-to-web-components/>

**Article Series:**

- Part 1: An Introduction to Web Components (This post)
- Part 2: Crafting Reusable HTML Templates
- Part 3: Creating a Custom Element from Scratch
- Part 4: Encapsulating Style and Structure with Shadow DOM
- Part 5: Advanced Tooling for Web Components

**What are Web Components, anyway?**

Web Components consist of three separate technologies that are used together:

- Custom Elements
- Shadow DOM
- HTML Templates



The collage consists of three screenshots from a video lecture, overlaid with a large, semi-transparent watermark that reads "INDIAN INSTITUTE OF TECHNOLOGY MADRAS".

- Top Screenshot:** A VS Code editor window titled "Custom elements demo". It shows three panels: HTML, CSS, and JS. The HTML panel contains a custom element definition and some text. The JS panel contains a class definition for a custom element.
- Middle Screenshot:** A web browser window showing the output of the custom element demo. It displays the text "Hello world" three times, each on a new line.
- Bottom Screenshot:** A web browser window showing a page titled "Introduction to web-components/". The main content area has a heading "Custom elements" and a paragraph explaining that custom elements are HTML elements, like `<div>`, `<h1>`, or `<img>`, but something you can name wherever that are. The sidebar on the right has a section "OUR LEARNING PARTNER" with a logo and text.

The screenshot shows a web browser window. The main content area has a dark background with white text. The heading 'Custom elements' is prominently displayed. Below it, there is a paragraph of text. To the right, there is a sidebar with a dark background. It features a logo for 'Frontend Masters' and a section titled 'Need front-end development training?' with a link below it. The browser's address bar shows a URL starting with 'https://www.frontendmasters.com/'.

The screenshot shows a web browser window. The main content area has a dark background with white text. The heading 'Custom elements' is prominently displayed. Below it, there is a paragraph of text. To the right, there is a sidebar with a dark background. It features a logo for 'Frontend Masters' and a section titled 'Need front-end development training?' with a link below it. The browser's address bar shows a URL starting with 'https://www.frontendmasters.com/'.

The **shadow DOM** is an encapsulated version of the DOM. This allows authors to effectively isolate DOM fragments from one another, including anything that could be used as a CSS selector and the styles associated with them. Generally, any content inside of the document's scope is referred to as the **light DOM**, and anything inside a shadow root is referred to as the shadow DOM.

When using the light DOM, an element can be selected by using `document.querySelector("selector")` or by targeting any element's children by using `document.querySelector("selector")` in the same way, a shadow root's children can be targeted by calling `shadowRoot.querySelector` where `shadowRoot` is a reference to the document fragment. The difference being that the shadow root's children will not be select-able from the light DOM. For example, if we have a shadow root with a `button` inside of it, calling `shadowRoot.querySelector("button")` would return our button, but no invocation of the document's query selector will return that element because it belongs to a different Document Object Model instance. Style selectors work in the same way.

In this respect, the shadow DOM works sort of like an `iframe` where the content is cut off from the rest of the document; however, when we create a shadow root, we still have total control over that part of our page, but scoped to a content. This is what we call **encapsulation**.

If you've ever written a component that renders the same `id` or relies on either CSS-in-JS tools or CSS naming strategies (like [BEM](#)), shadow DOM has the solution for you.

OUR LEARNING PARTNER

**Frontend Masters**

Need front-end development training?

Frontend Masters is the best place to go. They have courses on all the most important front-end technologies, from React to CSS, from Node.js and backend with Node.js and Full Stack.

### Shadow DOM style encapsulation demo

```
HTML: <div id="myDiv">This will use the CSS background color</div>
      <div id="myDiv">Not inside</div>
```

```
CSS: .myDiv { background-color: red; }
```

```
JS: const shadowRoot = document.querySelector("#myDiv").attachShadow({ mode: "open" });
    shadowRoot.innerHTML = `
    <div id="myDiv">
      background: red;
      color: white;
    </div>
    <div id="myDiv">Not inside</div>`;
```

Not inside

### Custom elements demo

```
HTML: <my-component>This is some other text</my-component>
      <my-component>Hello world</my-component>
      <my-component>Hello world</my-component>
      <my-component>Hello world</my-component>
      <my-component>Some more text</my-component>
```

```
JS: class MyComponent extends HTMLElement {
  constructor() {
    this.innerHTML = "Hello world!";
  }
}
customElements.define("my-component", MyComponent);
```

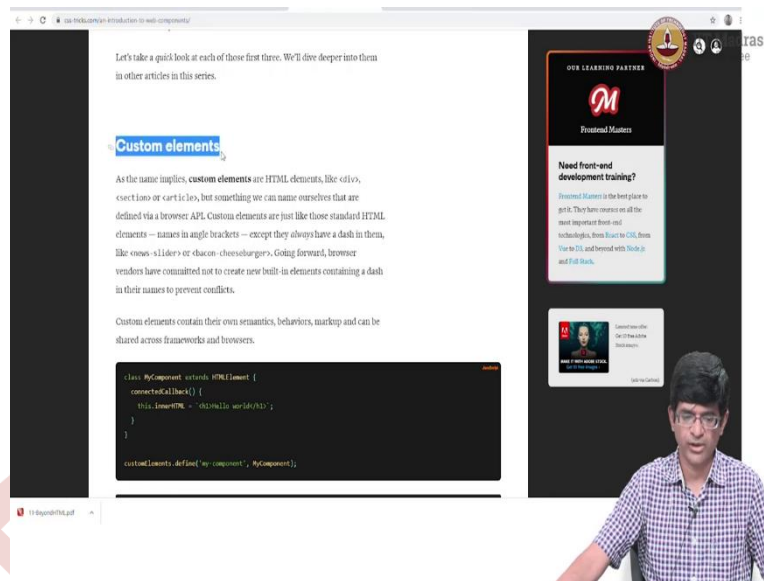
This is some other text

Hello world

Hello world

Hello world

Some more text



Now, probably the best way to explain this is once again through examples, and csstricks.com website actually has a couple of good examples that I am just going to walk through here. So, this webpage an introduction to web components has some nice examples that you can look at, right? And in fact, you can play around with.

So, the first example that we have here. And what we can do, the nice thing is you can actually open it on this website called codepen. And code pen essentially allows you to have something like this, which is basically a snippet of HTML, and CSS and JavaScript and it displays the final output for you down here. So, let us look at our HTML, it just has one line, which is just basically the new component that you have defined you have got this thing called my component and it shows you what that is.

Now, what is my component, I have this line at the bottom down here saying customElements.in, the JavaScript side of things, I have this thing saying customElements.define my\_component, and it will call the class my component with capital M and capital C. And what does my component itself do? It just basically adds the text or rather, it says, this.innerHTML =hello world.

Now, let us try playing around with it a little bit, I am going to add some text over here. I am going to try some more text. And it automatically goes in and displays whatever is there in the text. So, what you can see is that it has basically taken the inner HTML of the page that you have, wherever you had this mic component and replaced it with the text, hello world, which means that, now what you get corresponding to the my component is this hello world that is present out here.

And then after that, you have basically added on this some more text. Now, what happens if I, so what you can see is I have some text before that. I have my component over here, and I have some more text. I can also go further and make another copy of that. So, I will just basically make two more copies of this my component. And as you can see, it renders again down here, and it basically shows it three times. So, what did this do? This was just basically a very simple example of the most basic element, basic part of a web component, the fact that you can create a custom element.

(Refer Slide Time: 12:40)

The top screenshot shows the 'Shadow DOM' section of a web component introduction. It explains that the shadow DOM is an encapsulated version of the DOM, allowing authors to effectively isolate DOM fragments from one another. It also mentions that when using the light DOM, an element can be selected by using `document.querySelector("selector")` or by targeting any element's children by using `element.querySelector("selector")`.

The bottom screenshot shows the 'HTML templates' section. It explains that the `<template>` element allows us to stamp out re-usable templates of code inside a normal HTML flow that won't be immediately rendered, but can be used at a later time. It includes a code snippet for a template and a JavaScript snippet for creating and inserting content from the template.

```
<template id="book-template">
  <div class="title"></div> <div class="author"></div>
</template>

<div id="books"></div>
```

```
const fragment = document.getElementById("book-template");
const books = [
  { title: "The Great Gatsby", author: "F. Scott Fitzgerald" },
  { title: "A Farewell to Arms", author: "Ernest Hemingway" },
  { title: "Catch 22", author: "Joseph Heller" }
];

books.forEach(book => {
  // Create an instance of the template content
  const instance = document.importNode(fragment.content, true);
```

### HTML templates

The aptly named HTML <template> element allows us to stamp out re-usable templates of code inside a normal HTML flow that won't be immediately rendered, but can be used at a later time.

```
<template id="book-template">
  <div class="title"><span book.title></span></div>
</template>

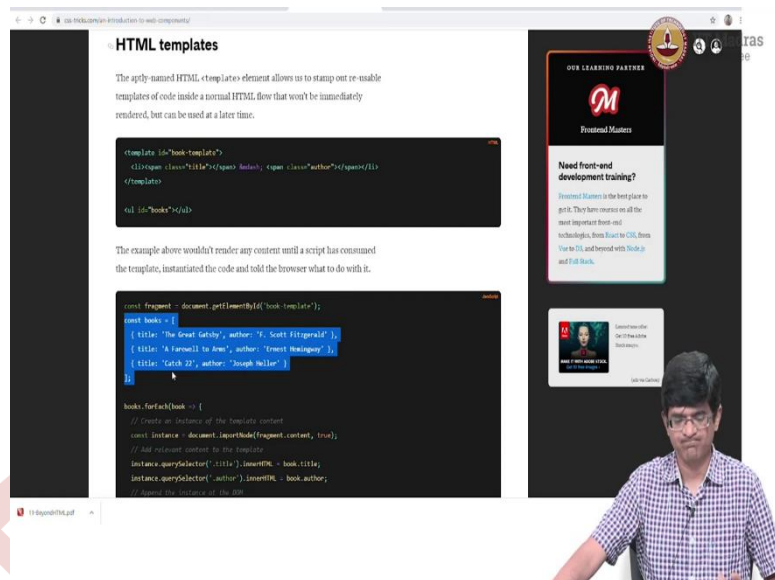
<div id="books"></div>
```

The example above wouldn't render any content until a script has consumed the template, instantiated the code and told the browser what to do with it.

```
const fragment = document.getElementById('book-template');
const books = [
  { title: 'The Great Gatsby', author: 'F. Scott Fitzgerald' },
  { title: 'A Farewell to Arms', author: 'Ernest Hemingway' },
  { title: 'Catch 22', author: 'Joseph Heller' }
];

books.forEach(book => {
  // Create an instance of the template content
  const instance = document.importNode(fragment.content, true);
  // Add relevant content to the template
  instance.querySelector('.title').innerHTML = book.title;
  instance.querySelector('.author').innerHTML = book.author;
  // Append the instance of the DOM
  document.getElementById('books').appendChild(instance);
});
```

19 depend.html.pdf



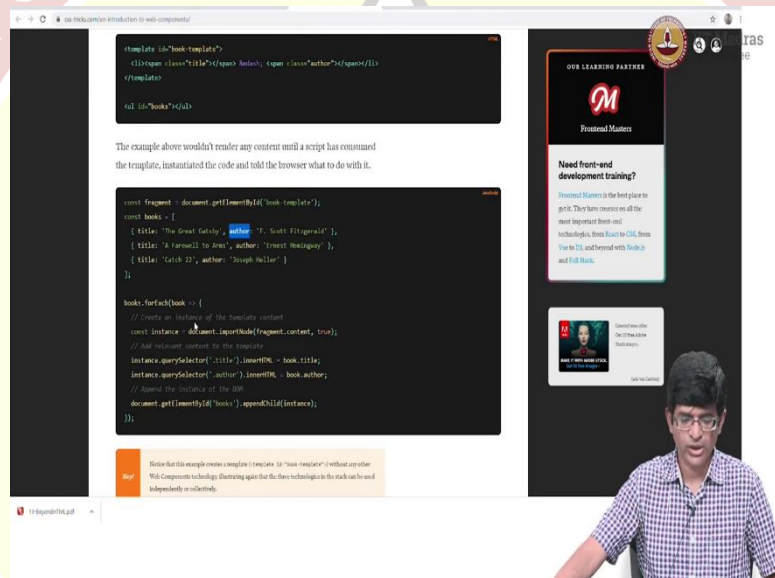
The example above wouldn't render any content until a script has consumed the template, instantiated the code and told the browser what to do with it.

```
const fragment = document.getElementById('book-template');
const books = [
  { title: 'The Great Gatsby', author: 'F. Scott Fitzgerald' },
  { title: 'A Farewell to Arms', author: 'Ernest Hemingway' },
  { title: 'Catch 22', author: 'Joseph Heller' }
];

books.forEach(book => {
  // Create an instance of the template content
  const instance = document.importNode(fragment.content, true);
  // Add relevant content to the template
  instance.querySelector('.title').innerHTML = book.title;
  instance.querySelector('.author').innerHTML = book.author;
  // Append the instance of the DOM
  document.getElementById('books').appendChild(instance);
});
```

19 depend.html.pdf

Notice that this example creates a template (in "book-template") without any other Web Components technology. Illustrating again that the three technologies in this stack can be used independently or collectively.



Often, the consumer of a service that utilizes the template API could write a template of any shape or structure that could be created at a later time. Another page on a site might use the same service, but structure the template this way:

```
<template id="book-template">
  <div class="author"><span>'s class="name"><span class="title"></span></div>
</template>

<div id="books"></div>
```

HTML CSS JS

Result

- The Great Gatsby — F. Scott Fitzgerald
- A Farewell to Arms — Ernest Hemingway
- Catch 22 — Joseph Heller

Choose template

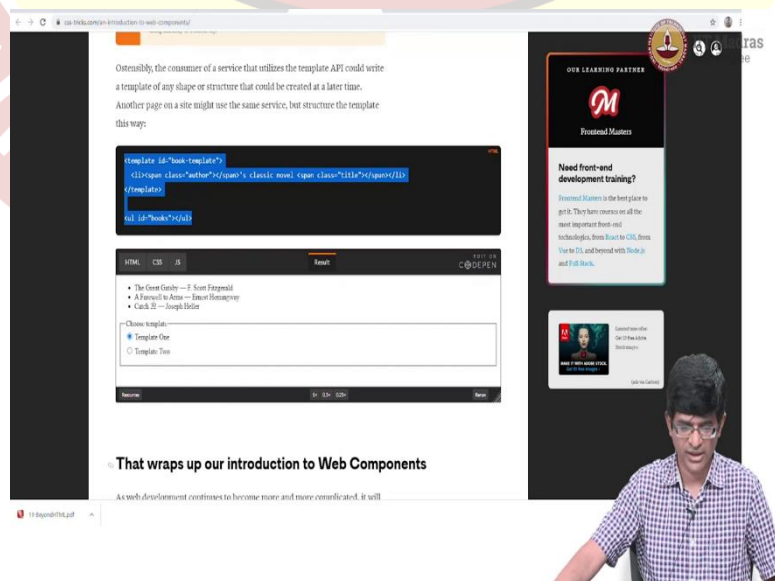
• Template One

• Template Two

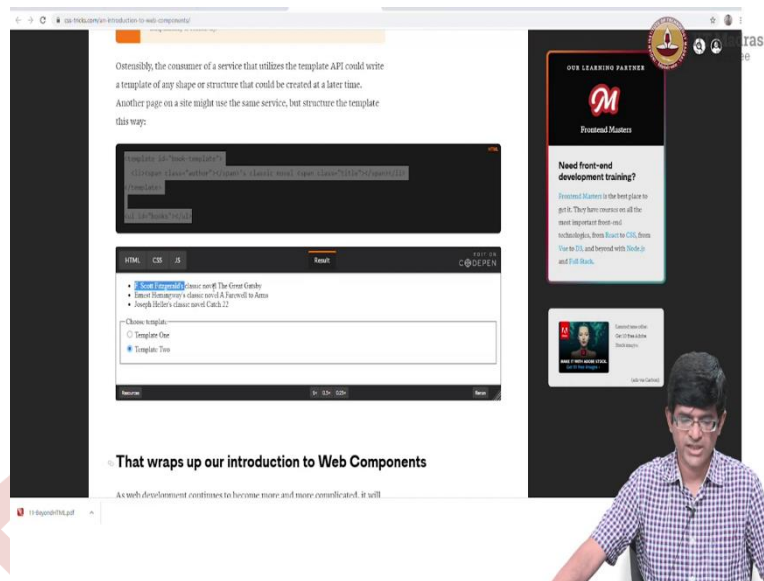
19 depend.html.pdf

### That wraps up our introduction to Web Components

As web development continues to become more and more complicated, it will







It goes further and says how you can use the Shadow DOM. Now the Shadow DOM is a little bit more tricky to explain what it says over here. And in fact, the example that you have, sort of shows that there is some text out here, with a div in the HTML part, which basically says this will use the CSS background, and then there is a button that says not tomato which you can see down here

Now, what does a JavaScript itself do? If you look at the HTML text, what it said was, I mean, if you think about it, what it should have displayed was this will use the CSS background. That is all. But what the JavaScript does is, it basically goes and modifies that slightly, adds background tomato as the style. It also adds the text tomato and it basically converts this entire thing that you had out here into a button.

So, in other words, what it has done is, it has managed to add some extra styling, and also convert the whatever you had out here into a button without affecting the rest of the page itself. In other words, this shadow route that we created out here using these commands, restricted itself only to this part. So, this style that we created even though we gave the background tomato, that button background did not go and modify this button. So, the not-tomato remains as it is.

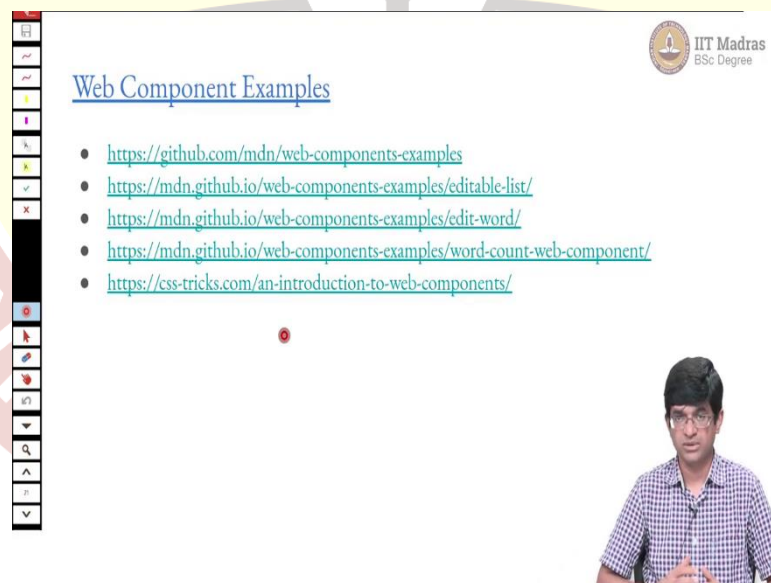
The fact that I went and modified the button background class the button background style over here to tomato applies only to the first button not to the second one. A little bit tricky to understand, but it is worth understanding how it works if you are planning to get into this in more detail, at least.

And finally, there is this notion of HTML templates. So, what does an HTML template do? You know, the example that they are given out here basically shows you that you might have information where you let us say you have a piece of JavaScript, which contains a list of books and each book basically has a title and an author. Now the question becomes, how are you going to display it.

And what is being done over here, is there are like templates that are created out here. And in fact, the demonstration can be shown right now. The way that it is showing it right now is basically it shows the title of the page puts a dash, and shows the author of the page. And now, by just clicking on this one radio button that says template two it changes around and says, F. Scott Fitzgerald's classic novel, The Great Gatsby, Ernest Hemingway's classic novel, Farewell to Arms, and so on.

In other words, it has changed the template, the template now became the title of the book, space, classic novel or rather the author's name space classic novel space the title of the novel. So, just by making a change in the template, you were able to modify the way that this got displayed.

(Refer Slide Time: 16:11)



The screenshot shows a presentation slide titled "Web Component Examples" with the IIT Madras BSc Degree logo in the top right corner. The slide contains a list of five URLs:

- <https://github.com/mdn/web-components-examples>
- <https://mdn.github.io/web-components-examples/editable-list/>
- <https://mdn.github.io/web-components-examples/edit-word/>
- <https://mdn.github.io/web-components-examples/word-count-web-component/>
- <https://css-tricks.com/an-introduction-to-web-components/>

In the bottom right corner of the slide, there is a video inset showing a man with glasses and a checkered shirt, who appears to be the speaker.

github.com/nidhi/web-components-examples

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search Sign in Sign up

nidhi / web-components-examples Public

Notifications Star 13k Fork 518

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 0 tags Go to file Code About

chiradevml Merge pull request #10 from nidhi/patch-4 ✓ 1 checker on May 25, 2020 68 commits

composed composed path	Use <code>slot</code> , instead of <code>Set</code>	3 years ago
defined pseudo-class	Use <code>shadowroot</code> , <code>count</code>	3 years ago
edit word	Improved edit word demo	3 years ago
editable list	WIP	3 years ago
element details	Use <code>count</code> instead of <code>set</code>	4 years ago
expanding list web-component	Expanding list element made more self-contained	2 years ago
host selectors	Use template literals and <code>slot</code> , instead of <code>Set</code>	3 years ago
life cycle callbacks	Handle callbacks Simplified update cycle	3 years ago
popup info box external stylesheet	adding external stylesheet shadow DOM example	2 years ago
popup info box web-component	adding external stylesheet shadow DOM example	2 years ago
shadow part	Update index.html	17 months ago
simple template	Normative extension	3 years ago
storage	Use <code>for</code> block, template literals and <code>slot</code> , instead of <code>Set</code>	3 years ago
styled pseudo-element	Use template literals and <code>slot</code> , instead of <code>Set</code>	3 years ago
word count web-component	Use template literals and <code>slot</code> , instead of <code>Set</code>	3 years ago
CODE_OF_CONDUCT.md	adding CODE file	3 years ago

11 dependITM.pdf

11 dependITM.pdf

github.com/nidhi/web-components-examples

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search Sign in Sign up

nidhi / web-components-examples Public

Notifications Star 13k Fork 518

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 0 tags Go to file Code About

chiradevml Merge pull request #10 from nidhi/patch-4 ✓ 1 checker on May 25, 2020 68 commits

defined pseudo-class. A very simple example that shows how the `defined pseudo-class` works. See `defined-pseudo-class` file.

- editable list** — `editable-list`: A simple example showing how elements can be consolidated to create a list with addable/removable items. Items are added by using a `list-item` attribute or by entering text and clicking the plus sign. See `editable-list` file.
- edit word** — `edit-word`: Wrapping one or more words in this element means that you can then click/focus the element to reveal a text input that can then be used to edit the words. See `edit-word` file.
- element details** — `element-details`: Displays a box containing an HTML element name and description. Provides an example of an autonomous custom element that gets its structure from a `template` element (that also has its own styling defined), and also contains `slot` elements populated at runtime. See `element-details` file.
- expanding list web-component** — `expanding-list`: Creates an unordered list with expandable/collapsible children. Provides an example of a customized built-in element (the class inherits from `HTMLListElement`, rather than `HTMLElement`). See `expanding-list` file.
- life cycle callbacks** — `isolate-square` (`isolate-square`): A trivial example web component that creates a square colored box on the page. The demo also includes buttons to create, destroy and change attributes on the element, to demonstrate how the web components life cycle callbacks work. See `life-cycle-callbacks` file.
- popup info box web-component** — `popup-info` (`popup-info`): Creates an info box that when focused displays a popup info box. Provides an example of an autonomous custom element that takes information from its attributes, and defines structure and basic style in an attached shadow DOM. See `popup-info` file.
- simple template** — A very simple trivial example that quickly demonstrates usage of the `template` and `slot` elements. See `simple-template` file.
- storage example** — `summary-display`: An example that takes as its two slot values a list of possible choices, and a description for the selected choice. Multiple paragraphs are included inside the element containing all the possible descriptions when a choice is clicked. Its corresponding description paragraph is given an appropriate `slot` attribute so that it appears in the second slot. This example is written to demonstrate usage of the `storage` attribute, and features of the `HTMLSlotElement` interface. See the `storage` example file.
- styled pseudo-element**: A very simple example that shows how the `::slotted` pseudo-element works. See `styled-pseudo-element` file.
- word count web-component** — `word-count` (`word-count`): When added to an element, counts all the words inside that shadow DOM. It also contains an internal that periodically updates the `word-count` attribute. See `word-count` file.

11 dependITM.pdf

11 dependITM.pdf

github.com/nidhi/web-components-examples

Why GitHub? Team Enterprise Explore Marketplace Pricing

Search Sign in Sign up

nidhi / web-components-examples Public

Notifications Star 13k Fork 518

Code Issues Pull requests Actions Projects Wiki Security Insights

master 1 branch 0 tags Go to file Code About

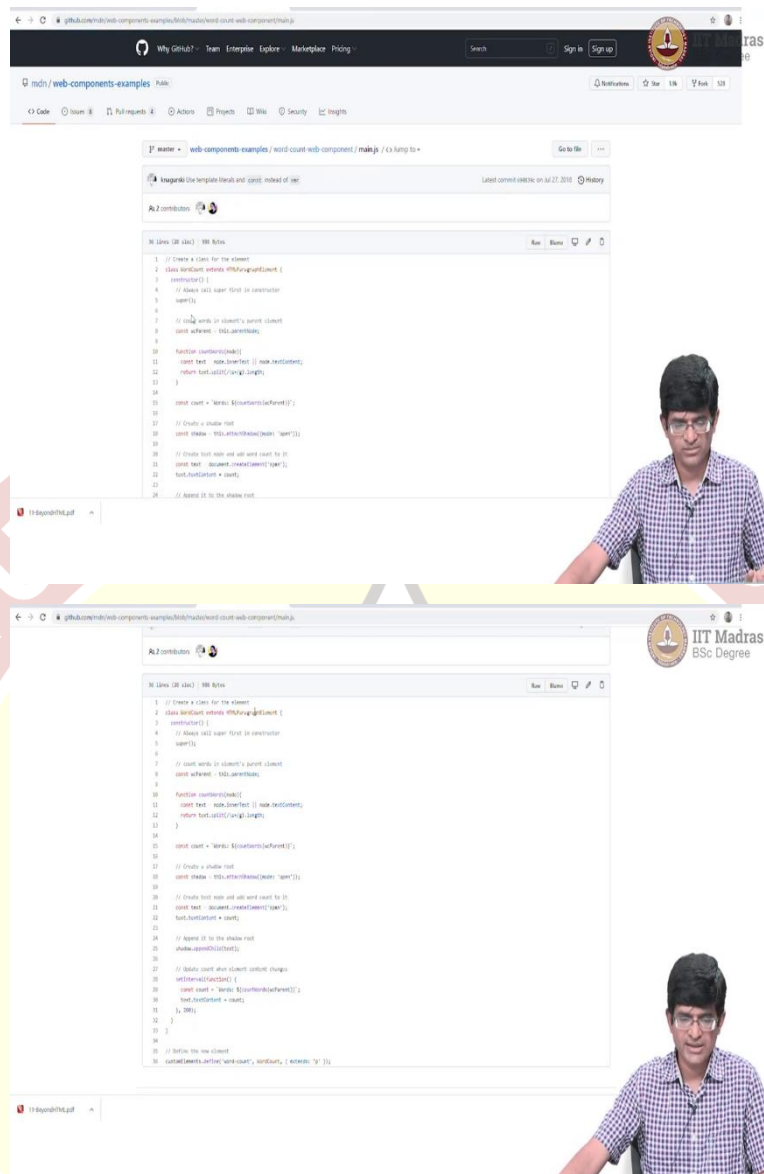
chiradevml Merge pull request #10 from nidhi/patch-4 ✓ 1 checker on May 25, 2020 68 commits

defined pseudo-class. A very simple example that shows how the `defined pseudo-class` works. See `defined-pseudo-class` file.

- editable list** — `editable-list`: A simple example showing how elements can be consolidated to create a list with addable/removable items. Items are added by using a `list-item` attribute or by entering text and clicking the plus sign. See `editable-list` file.
- edit word** — `edit-word`: Wrapping one or more words in this element means that you can then click/focus the element to reveal a text input that can then be used to edit the words. See `edit-word` file.
- element details** — `element-details`: Displays a box containing an HTML element name and description. Provides an example of an autonomous custom element that gets its structure from a `template` element (that also has its own styling defined), and also contains `slot` elements populated at runtime. See `element-details` file.
- expanding list web-component** — `expanding-list`: Creates an unordered list with expandable/collapsible children. Provides an example of a customized built-in element (the class inherits from `HTMLListElement`, rather than `HTMLElement`). See `expanding-list` file.
- life cycle callbacks** — `isolate-square` (`isolate-square`): A trivial example web component that creates a square colored box on the page. The demo also includes buttons to create, destroy and change attributes on the element, to demonstrate how the web components life cycle callbacks work. See `life-cycle-callbacks` file.
- popup info box web-component** — `popup-info` (`popup-info`): Creates an info box that when focused displays a popup info box. Provides an example of an autonomous custom element that takes information from its attributes, and defines structure and basic style in an attached shadow DOM. See `popup-info` file.
- simple template** — A very simple trivial example that quickly demonstrates usage of the `template` and `slot` elements. See `simple-template` file.
- storage example** — `summary-display`: An example that takes as its two slot values a list of possible choices, and a description for the selected choice. Multiple paragraphs are included inside the element containing all the possible descriptions when a choice is clicked. Its corresponding description paragraph is given an appropriate `slot` attribute so that it appears in the second slot. This example is written to demonstrate usage of the `storage` attribute, and features of the `HTMLSlotElement` interface. See the `storage` example file.
- styled pseudo-element**: A very simple example that shows how the `::slotted` pseudo-element works. See `styled-pseudo-element` file.
- word count web-component** — `word-count` (`word-count`): When added to an element, counts all the words inside that shadow DOM. It also contains an internal that periodically updates the `word-count` attribute. See `word-count` file.

11 dependITM.pdf

11 dependITM.pdf



Ultimately, what happens, as a result is that, you can have more and more complicated word, such web components that are built up and start composing an entire webpage out of such components. So, once again, the Mozilla Developer Network, they have this GitHub page that contains a number of different examples.

So, this, in fact this MDN, the [github.com/mdn](https://github.com/mdn/web-components-examples), they contain a number of web components examples which you can also see live. You can actually see what it looks like. So, in fact, some of them are quite instructive and useful to sort of see how they work. So, let us just dive into one of them, which is a very simple example. What it says is, the JavaScript corresponding to this, it looks a little bit long, but it is quite small compared to most of the other JavaScript, it is worth just understanding what it does.

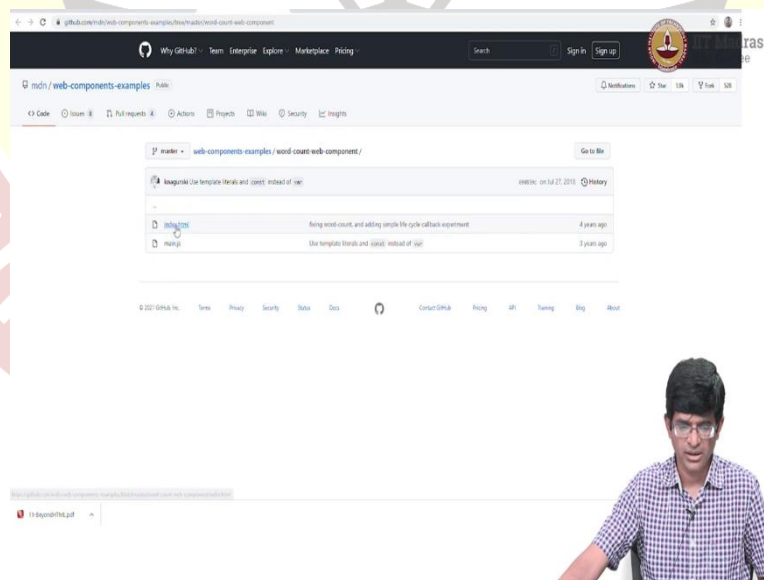
It is now defining a new class, which extends the HTML paragraph element. So, interesting, what it is doing is it is sort of creating an extension on top of an element that already exists, it is not defining a new element by itself. All this part about the constructor and so on is just like the initialization, the init function that you call in Python.

The important part is, it creates a shadow root and it is trying to create the text corresponding to that root. And what is the text? It is going to be equal to some variable called count. And what is this variable count? It is going to be by running this function count words, which is defined out here.

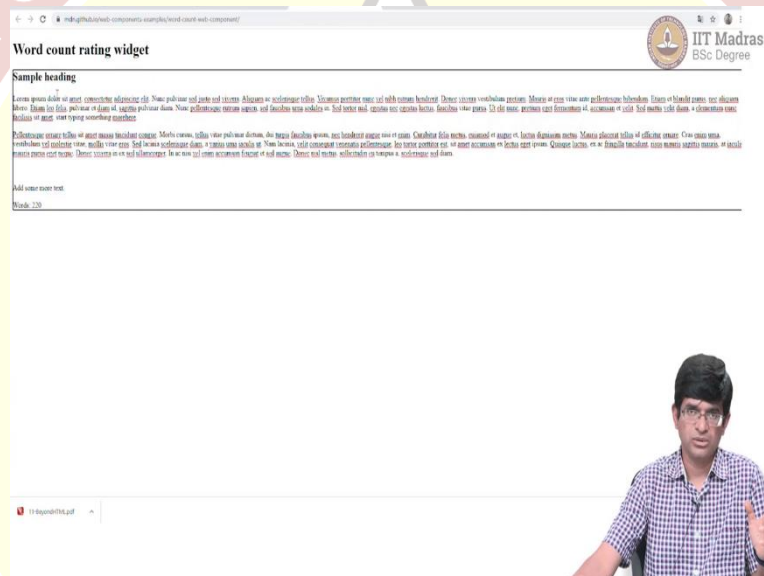
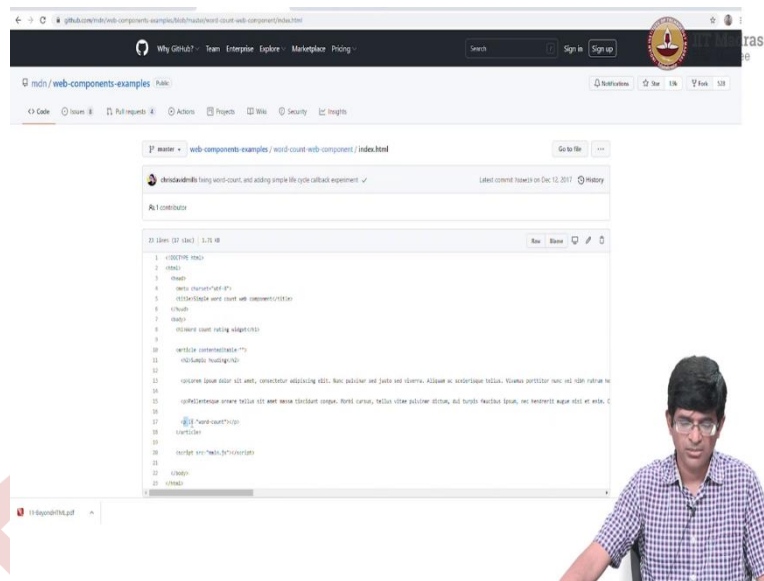
So, this function count words has been defined in order to count the number of words in a given piece of text, in fact, specifically in the inner text corresponding to a given node. Which means that, I can then call count words on the parent of this node, which I get again from the JavaScript over here, set that as the text of something that I want to display and also set it so that every 200 milliseconds, I will update this word count.

And finally, I define a custom element. The word count is defined as calling this class capital word count and it extends p. It was defined as an extension of HTML paragraph element and here we say that it extends p. That is to say, the p element type.

(Refer Slide Time: 19:13)







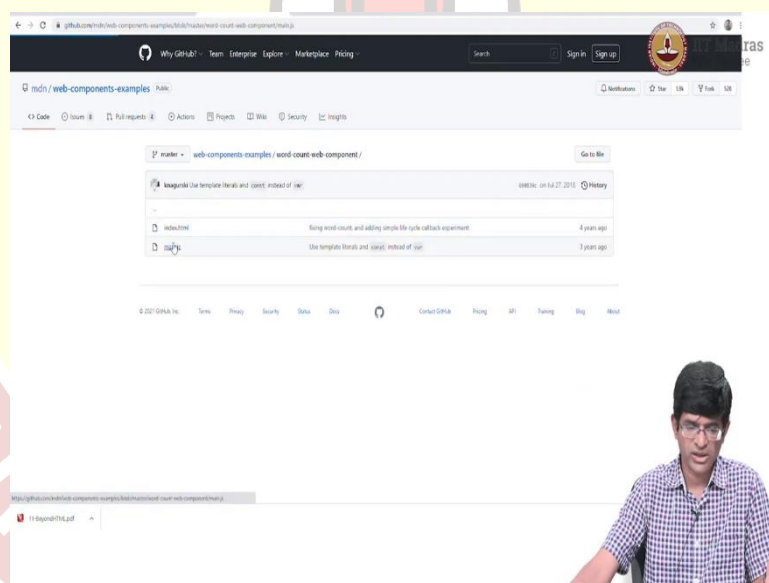
What does the index.html look like? It is like fairly regular HTML up to this point. It just declares the content editable article. You can ignore that for now, it basically means it is sort of like a text area that you can type text into. It has some random text generated in there. And this last line 17 that you see or hear basically says there is a word-count paragraph. In other words, it is a p, element of type p paragraph, but with the extension word-count applied to it.

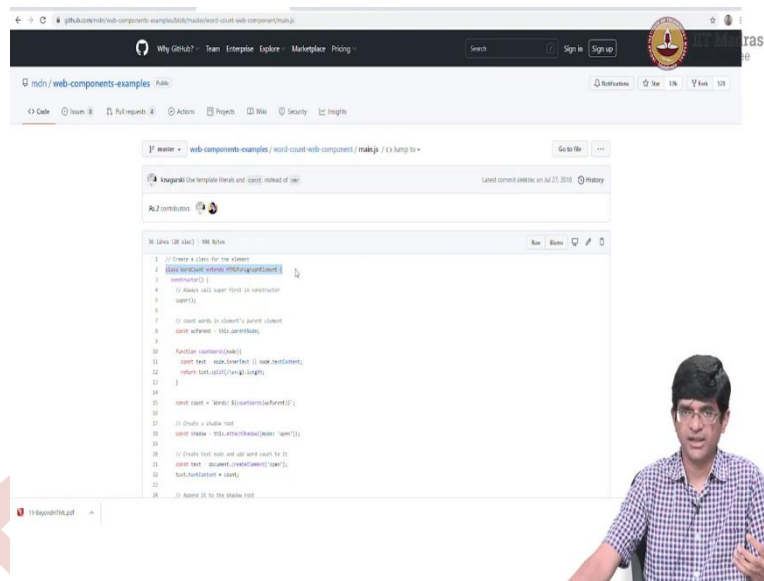
What does this look like when you actually go to a page like that, it basically says word count rating widget. And when I click somewhere there, I find that I can actually edit it. Now look at this number down here saying words:2012. Let me go and start typing something more here. And you can see that as I type, it starts changing this number of words that have become 216.

And I can actually go in there and now, perhaps, add some more text. All of this works, but you will realize that, you cannot actually select this words:220 and delete it, it is not part of the paragraph, it just sits over there, because it was created by the JavaScript, it is not something you typed in. So, no matter where else you type, it will automatically add this at the end.

So, is this useful? Think about it. Let us say that you are creating an editor for somebody to fill in a form you create one standard widget like this, which basically takes an HTML area where you can type text and puts a word count at the bottom. It may not just be a word count, it might even be something which if you go beyond a certain value will actually raise an alert and stop you from typing or you could have something which basically puts a timer out there. And after you have typed for a certain amount of time it counts down and says, stop. All that functionality could have been added in just by modifying that main.js, which defines the web component.

(Refer Slide Time: 21:29)





Now, where this becomes really useful is think about how the web component was defined, it was just done in this main.js, which basically had this class wordcount, which extended paragraph element. It did not have any relationship to this particular website, it did not have any other dependencies on what is being done over here, which means that, I could have taken it, put it on my website, you could have taken it, put it on your website.

Somebody could have installed it on GitHub or some other place, and I could have just referenced it from there and it would work, which means, it becomes reusable. And that is sort of the key behind web components. The fact that you are creating reusable user interface components.

(Refer Slide Time: 22:13)

### Summary

- Custom Elements: API to extend HTML5 element/tag capabilities
- Shadow DOM: restrict scope of styling or modification of content
- HTML Templates

Combined: Web Components

- Goal: reuse
- Problem: limited standardization

So, to summarize, what is the point of web components? It basically is an API that can allow you to extend HTML5 element or tag capabilities. It uses something extra called the Shadow DOM, which restricts the scope of styling and allows you to create self-contained encapsulated units and it uses templates that allows you to write no reusable code.

You combine all these three together and you have this definition of something called web components. And the primary goal of this is reuse of widgets. Widget is just a generic term for something that you might want to use in different parts of your screen of your user interface. So, one of the biggest problems, of course, with this entire approach is this, unfortunately, the web component part is not standardized.

And if you think about it, you can understand, why because, the API is standardized, how to create a new custom element, but how should I use a shadow DOM, how should I use templates and so on is largely a matter of subjective interpretation. So, there is a very good chance that people might object and say, look, that is not the way I think about it. That is not how I want to do it.

