INDIAN INSTITUTE OF TECHNOLOGY MADRAS

सिद्धिर्भवति कर्मजा

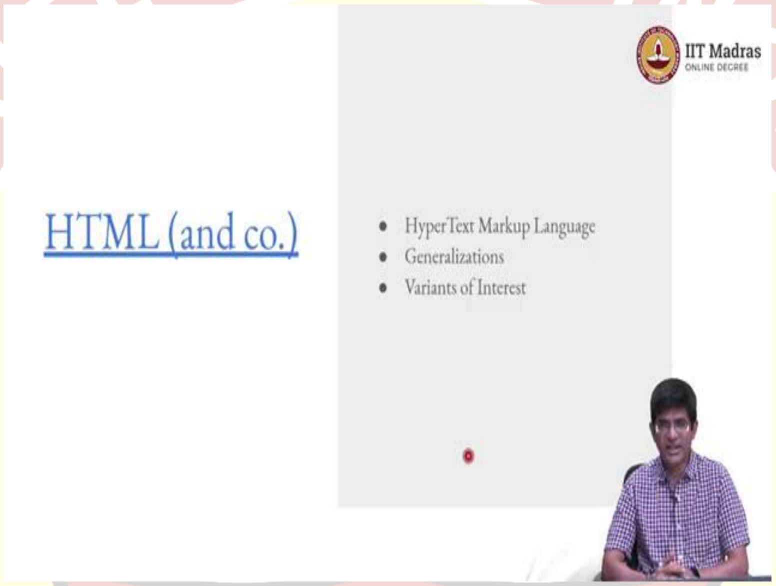
IIT Madras

ONLINE DEGREE

Modern Application Development - I
Professor. Nitin Chandrachoodan
Department of Electric Engineering
Indian Institute of Technology, Madras
Introduction to HTML

Hello, everyone, and welcome to this course on Modern Application Development.

(Refer Slide Time: 0:16)

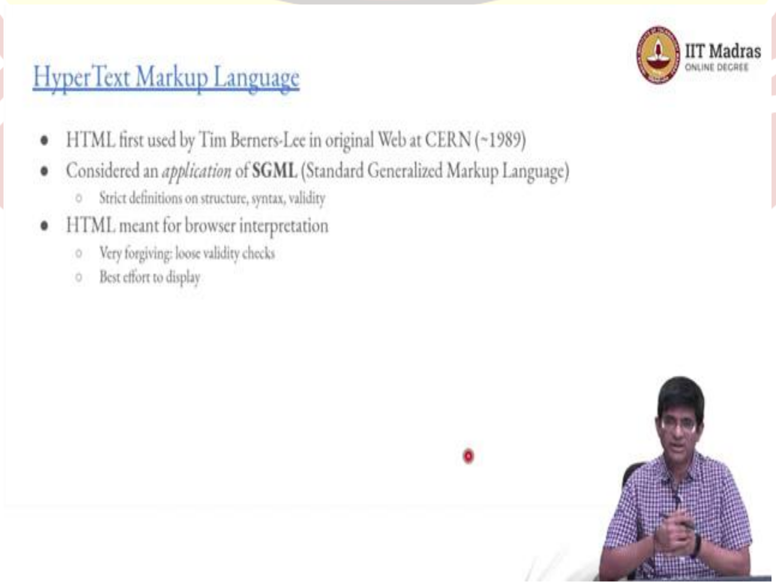


HTML (and co.)

- HyperText Markup Language
- Generalizations
- Variants of Interest

So, now with all of that behind us, we are now going to look a little bit more into detail on HTML and its variants, what is HTML, Hypertext Markup Language.

(Refer Slide Time: 0:27)



HyperText Markup Language

- HTML first used by Tim Berners-Lee in original Web at CERN (~1989)
- Considered an *application* of **SGML** (Standard Generalized Markup Language)
  - Strict definitions on structure, syntax, validity
- HTML meant for browser interpretation
  - Very forgiving: loose validity checks
  - Best effort to display

So, HTML in some form as not quite the form that we know it now was first used by Tim Berners-Lee in the original web at CERN, the European Center for Nuclear Research roughly in 1989. I believe that, HTML was in development, but even before that.

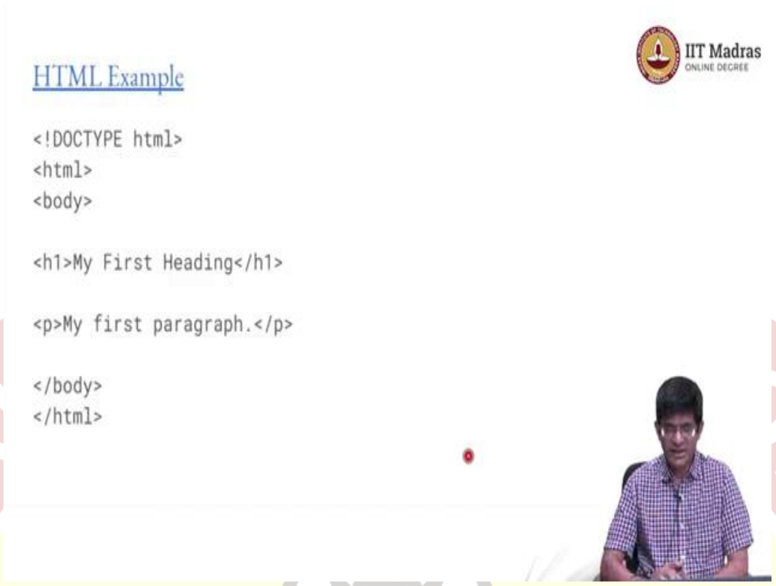
Now, it is not that it came out of nowhere HTML, the idea of markup language is already existed. And in fact, there was something called the Standard Generalized Markup Language SGML, which was meant as a way for creating other markup languages which were more sort of domain specific. Now, SGML has a number of very strict restrictions on the structure, on the syntax, the validity and so on. HTML, on the other hand, was meant primarily for browsers.

And right from the beginning one of the I think conscious decisions taken or there which has had a very interesting impacts further down the line is that since it was meant for browser interpretation, the approach that the original inventors took was that, look, it has to be forgiving of mistakes, because after all a person typing an HTML document is entirely likely to leave out some tag here or there. And let us say if I try loading that into a browser and only thing that I keep seeing are error messages, I am not going to really use this.

On the other hand, most of the time, if I know that, a person started, let us say, a title or something over here and then they jump to a new paragraph, I know that the title has ended before I started a new paragraph. So, why not just let the browser be forgiving of these kinds of mistakes, and do our best effort interpretation. So, HTML was a markup language along the lines of things like SGML, which already existed at that time. It was, it had a few things that were very specific to how you would mark up something for display on a screen and basically for annotating hypertext, so to say.

And, given the fact, the context in which it was being used, the fact that it was meant for some kind of interaction between different people, they decided that the browser is also need to be sort of as forgiving as possible and do our best effort to display what can be displayed.

(Refer Slide Time: 2:53)


HTML Example

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

This is an example of an HTML document, sort of pretty much the simplest document that you can think of. This first line that you can see or hear, I mean, this is a modern HTML document, which looks very similar to what HTML would have looked like 30 years ago. It is pretty much the same. But the point is, this line over here DOCTYPE HTML nowadays is considered more or less a requirement, pretty much standardized and say, look, if you do not have that it is not really valid HTML.

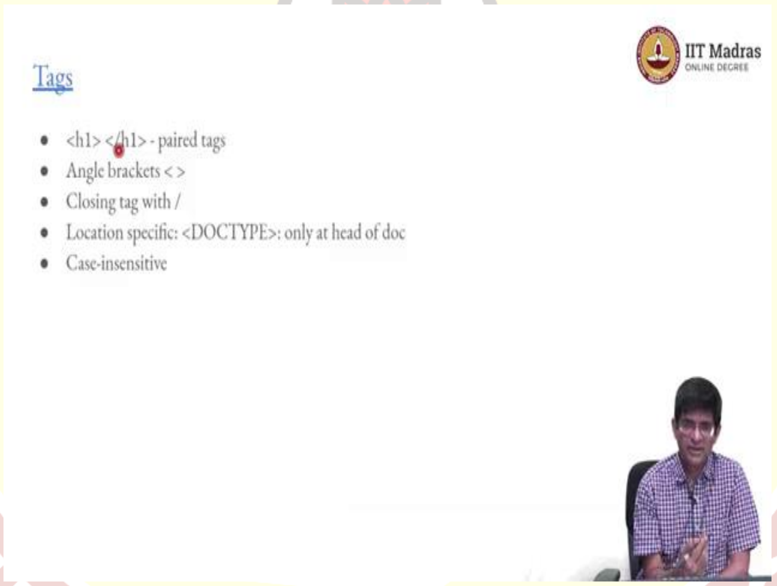
In previous days, there were different variants of this. It would not quite be this DOCTYPE HTML, it would be some other, they had things called DTD Document Type Definitions and so on, which were trying to sort of stick with SGML, but were neither here nor there. Now, what we have is, in some sense, a simplified version. More important is the structure of the document itself. What it says is, look, this HTML pairs up with this slash(/) HTML at the bottom. Everything in between is your main document.

Now, within that document, you have a body which starts here and ends with the slash(/) body. So, what you can see is that every one of these so called tags has some name, in this case body, in this case HTML. And how is a tag specified by putting these angle brackets(< >) the less than and greater than signs around it. No spaces nothing else around this, just the tag immediately surrounded by these two angle brackets(< >).

Now, this <body> and </body> is a structural element. It says that everything within this is the main body of the text. Within that I have <h1> and </h1>. H1 is a first level heading, which means that it usually will be in larger font size, etcetera. But how you display it, it is up to you. Similarly, you have another structural element called a paragraph. So, now you can see where HTML is going. It is basically trying to sort of take the overall text and add these tags around it.

The main text that you have is just this, my first heading, my first paragraph. You could have just written these two lines, and you are done. Everything else around here is markup, which is doing two things. One is it is telling other readers or even machines that look at this document, how to interpret what is in different parts of it. And also it is specifying that there is semantic content meaning to this. This is a heading. This is a paragraph. Therefore, display them appropriately.

(Refer Slide Time: 5:33)



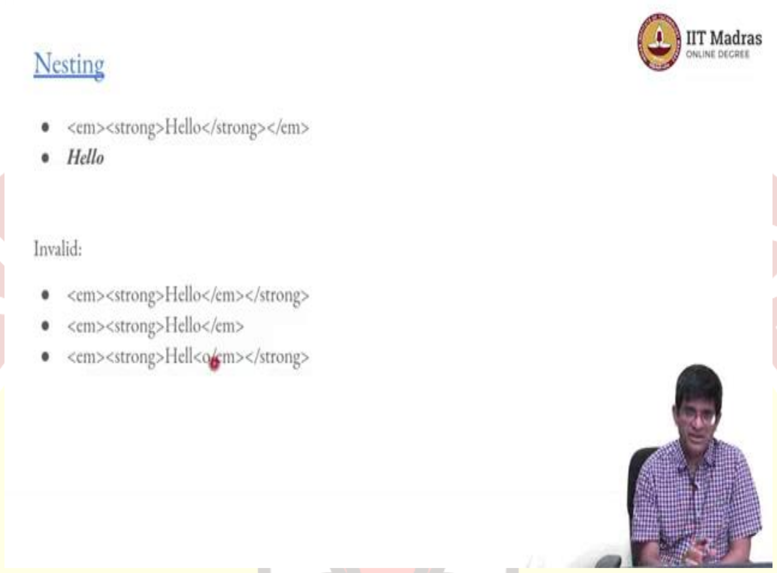
Tags

- <h1> </h1> - paired tags
- Angle brackets < >
- Closing tag with /
- Location specific: <DOCTYPE>: only at head of doc
- Case-insensitive

So, we always have paired tags like this. <h1> is always matched with the </h1>. The angle brackets(< >) are used in order to indicate tags. And the closing tag as you can see, the </h1> usually has a slash associated with it. Some of the tags are location specific, for example, that DOCTYPE tag only makes sense at the heading, at the pretty much the first line of the document. One other thing is that all of these things are usually case insensitive.

You could use either capital H1 or small h1. You could even have capital H over here and smaller h over here and the standard allows you to just mix and match. This generally not considered good style, simply because it makes it harder to read. But there is nothing preventing you from doing it.

(Refer Slide Time: 6:24)



Nesting

- `<em><strong>Hello</strong></em>`
- ***Hello***

Invalid:

- `<em><strong>Hello</em></strong>`
- `<em><strong>Hello</em>`
- `<em><strong>Hell<o/em></strong>`

Can you nest these tags? Sure. So, for example, if I do something like this, <em> is used for emphasis. This is a logical tag, because emphasis actually speaking does not really tell you how it is to be displayed. If I say emphasis, does it mean that I should make it blink or should I make it like in a bright neon color? What is usually done is to put it in italics, use an italic font. What about strong?

<Strong> indicates that whatever comes over here should be somehow, it should stand out. Once again, what is usually used is to make it bold font. But people sometimes confuse it and think that just by indicating strong it means bold font, it does not. Strong just means that this is a, strong is a logical markup, whereas bold font is a physical markup.

In this case, since usually <strong> stands for bold and <em> stands for italic, what will happen is the actual text which is between these tags will get displayed something like this. Why like this? First, it will apply the strong, which makes it bold. And then on top of that, it will apply <em> which will make it italic and the result will be both bold and italic. Now, automatically, you might also realize that this means that you can have invalid combinations of tags.

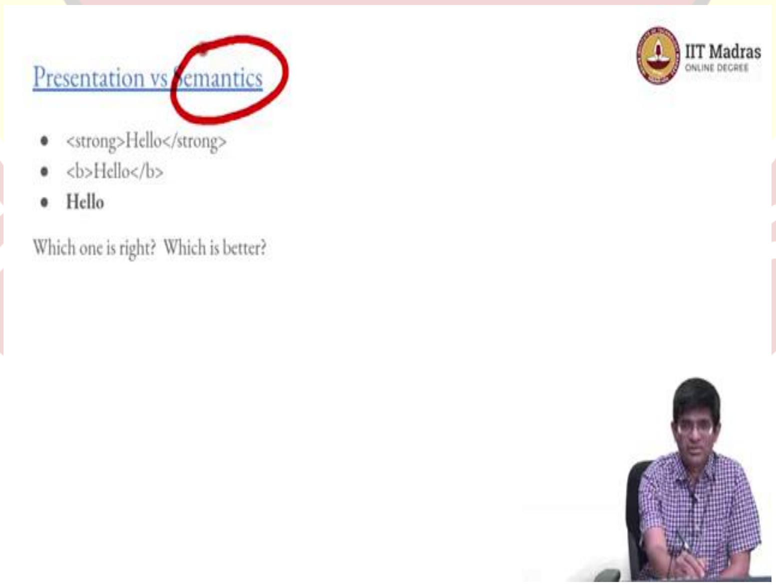
Supposing I write it like this, I start with <em>, say strong, then give the word to be written, close the </em> tag, and then close the </strong>, there is actually a mismatch, because this

<em> goes with this and this strong goes with this. They are not nested properly. Since <strong> was the innermost tag, it should also have been the first one to be closed before I went and close the other one. In other words, as far as the document standard is concerned, the order in which you do these things does matter.

Now, like I said, most browsers are quite forgiving. So, if you do write markup like this, there is a good chance that it will go through. At some point, it will indicate there was invalid HTML over here, but it will still try and understand what you wanted to do and display something which it thinks makes sense. What happens if you do not close one of the tags? You do not close the <strong> tag. You straightaway go and close the <em> tag. Is this valid? Technically, no.

Once again, what will the browser do, most likely try and make a best effort and realize that since you have closed the outer tag, the inner one must also have been closed. What about outright typos or spelling mistakes that we have over here? Yes, I mean, this will most likely result in something which even the browser does not know what to do with. Because something like this, it does not even know whether it is a starting tag or an ending tag or what it corresponds to. It might indicate an error, but it is unlikely to display it on the main page. It will probably just not format this the way that you expect.

(Refer Slide Time: 9:30)



Presentation vs Semantics

- <strong>Hello</strong>
- <b>Hello</b>
- **Hello**

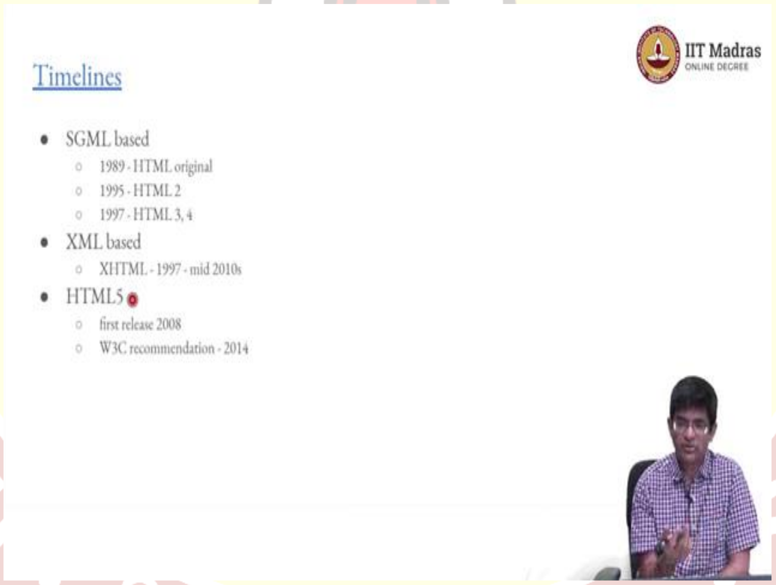
Which one is right? Which is better?

So, like I said, there is a difference between the presentation and the semantics, both <strong> and <b> will result in exactly the same output. Which one is correct? Which one is better? It is not an easy question to answer. Which one is correct? Both may be correct. It is not very clear what you wanted to do. Did you just want something to be in bold font? If so, both of them are okay.

But strong is actually telling you something more than that. It is trying to convey that to the reader that you want to emphasize this in some way. That is why you have put it in strong. Whereas b is simply saying, put it in bold font. It is up to the reader to decide whether it is important or not. So, b, in other words, is just doing presentation without worrying about the meaning, whereas here you are concerned with meaning.

So, when you do use tags in order to indicate functionality, be aware of that. You should, as far as possible, try and use them in a context where the semantics, the meaning makes sense. Ultimately, that is the more long lasting and the more important aspect of markup.

(Refer Slide Time: 10:43)



Timelines

- SGML based
  - 1989 - HTML original
  - 1995 - HTML 2
  - 1997 - HTML 3, 4
- XML based
  - XHTML - 1997 - mid 2010s
- HTML5
  - first release 2008
  - W3C recommendation - 2014

Some rough timelines on the evolution of HTML, like I said, the original SGML based HTML started from the late 80s, basically, around 1990 or so, the first sort of formal definition, once the popularity of the web started taking off, HTML 2 was defined in 1995. HTML 3, as well as 4 were actually defined in 1997. And then for quite a long time it was pretty much just HTML 4 that continued. Except that after a while people sort of started switching over to something called XHTML.

Now, what is XHTML? This was based on thing that there was SGML by its nature and the way that it was constructed is actually a very strict set of specifications. Whereas XML is something called the Extensible Markup Language, it was meant to sort of reduce a lot of the difficulties associated with the use of SGML, make it easier to define new types of tags, variations on tags, encapsulate different kinds of data, while at the same time being simpler to use than SGML. And this is something that you very often come across in computer science.

I mean, the original definitions are done by people who have put in a lot of thought. So, it is not that SGML was unnecessarily complicated for the sake of being complicated. It had all that structure because of all the mathematical underpinnings, how it should be passed, how it should be interpreted, what kind of tree structure should it have. A lot of thought went into that process and that is why the entire SGML definitions were made in a certain way. But that also did make them difficult to sort of modify and understand easily.

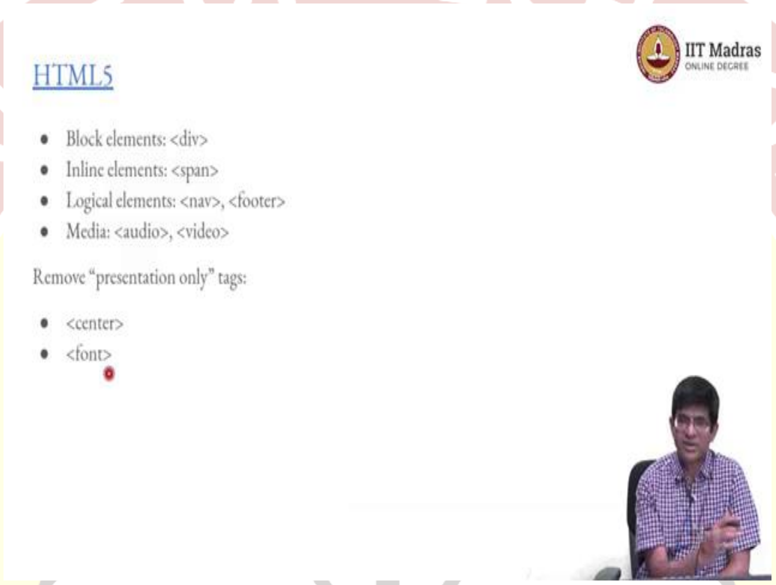
XML sort of took several learnings over the years. It was like, more than 10 years of SGML, HTML use that came into play and XML sort of tried to relax many of those problems or make it a little bit easier to use and understand. Based on XML, once again, XML has very specific kind of meanings and interpretations. XHTML said, now we will once again look at it in the context of the browser. What kind of problems can be forgiven? What kind cannot? What should be expected of the HTML?

But at the same time, bring the X factor into the, into play, and allow you to have slightly different representations of things. That remain for quite a while, almost till the mid-2010s. And finally, now, what we have is the dominant form is what is called HTML 5. This was sort of in development since around 2008 or so. It formally became a W3C recommendation in 2014. And nowadays is pretty much the standard everywhere.

And part of the reason why HTML 5 really took off was in between, before the rise of HTML 5, but after HTML 4, and so on, there were a lot of websites that used so called flash animations. So, flash was a technology from Adobe, which allowed you to do dynamic animations on websites. Those websites, they had some nice features. They looked quite cool. They also allowed you to do more fancy interactions than possible just with HTML. But it was not really standardized properly. You needed a separate plug-in. It was not part of the document standard.

So, it ended up being sort of heavy and not standardized. It would work on Windows machines, not well on Linux, it would not work on the same way on different variants of browsers on the same machine. HTML 5 attempted to sort of say, let us address those issues and bring in some standards for how things can be done. And since about 2014, it is pretty much the dominant standard seen on the web.

(Refer Slide Time: 14:45)


HTML5

- Block elements: <div>
- Inline elements: <span>
- Logical elements: <nav>, <footer>
- Media: <audio>, <video>

Remove "presentation only" tags:

- <center>
- <font>

So, what it does is that it also introduced a couple of logical elements, things like block elements. So, you will come across this <div>, which is a very common element that you are going to see when you write HTML documents. It essentially means this is a block something like a paragraph. You could have inline elements which just are a <span>. So, it basically spans across a few characters and say this is how this should be interpreted or displayed.

Now, you could again, abuse this in the previous manner. I mean, what we said about the difference between <bold> and <strong>, and just say, a span, this span of text has this characteristic, this font size, this and so on. Ideally, you should not, but it might happen, you could do something of that sort. It allows you to have fine grained control over what happens when. But to a large extent, always what we are looking for when we have any kind of document like this is, how do we specify logical structure rather than exact presentation.

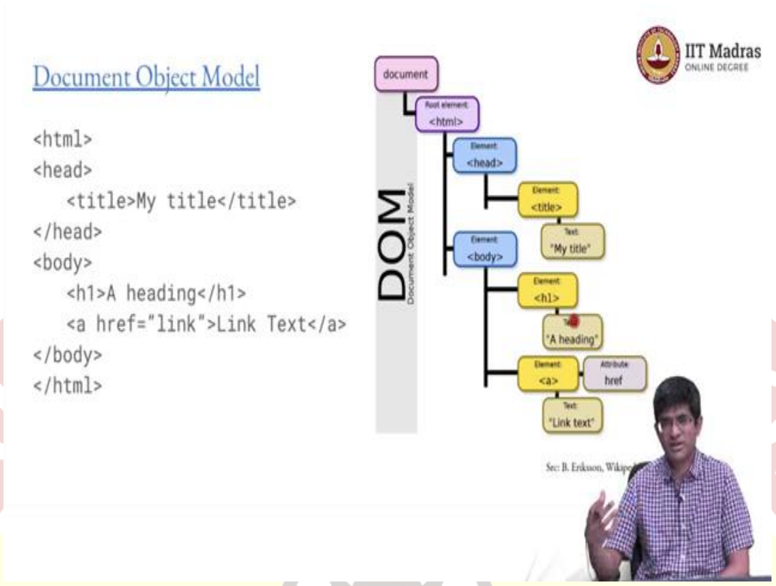
One additional aspect, as far as that was concerned was the introduction of elements like <nav>, so the navigation bar, the <footer> and so on. Now, these things are especially important in the context of accessible websites. What we mean by accessible is, let us say a person is visually impaired, and cannot read everything that is on the screen, is there still a way they can interact with this website. And there are screen readers. Things that will take the text that is on the screen and read it out to you.

Now, if this text is just one big blob of text, it becomes very difficult for a screen reader to work properly. But if you have used <nav> tags and footers and the divs and spans properly, the screen reader can actually do a very good job of finding out which is the important part. Let us say you have one <div>, which says main content, you can directly focus on that, or there is another <div> that says abstract that can be read out first. The <nav> can straightaway tell which is the next link, which is the upper link, any related links and so on. So, that is where the logical markup becomes more important.

HTML 5 also introduced some new tags like <audio> and <video> in order to interact with media. Basically, you can directly embed an audio segment or a video in mp3 or an mp4 file directly into a webpage, and it will play automatically. So, it defines how the browser should respond when it sees those tags and so on. Certain other tags which are mainly only for presentation, something like center, there used to be a tag called center which would explicitly allow you to center a certain thing on the page. But think about it.

I mean, going by our original definitions that is direct physical presentation, it is not logical markup. Similarly, there was a font tag, which would allow you to change the font that was used in part of the text. Once again, they said, look, take this out of the HTML standard. There are other ways by which we can do this.

(Refer Slide Time: 17:55)



Document Object Model

```
<html>
<head>
   <title>My title</title>
</head>
<body>
   <h1>A heading</h1>
   <a href="link">Link Text</a>
</body>
</html>
```

So, HTML, now that we have defined what an HTML document looks like, what the tags are, and so on, it also defines something called a document object model. So, like we said, this is an HTML document, I have left out the DOCTYPE in the beginning, but it has this HTML and slash HTML. Within that it has two sub-structures. There is a heading and a body. Inside the header, there is a title. Inside the body, there is a first level heading and a link. All of this essentially corresponds to a single HTML document.

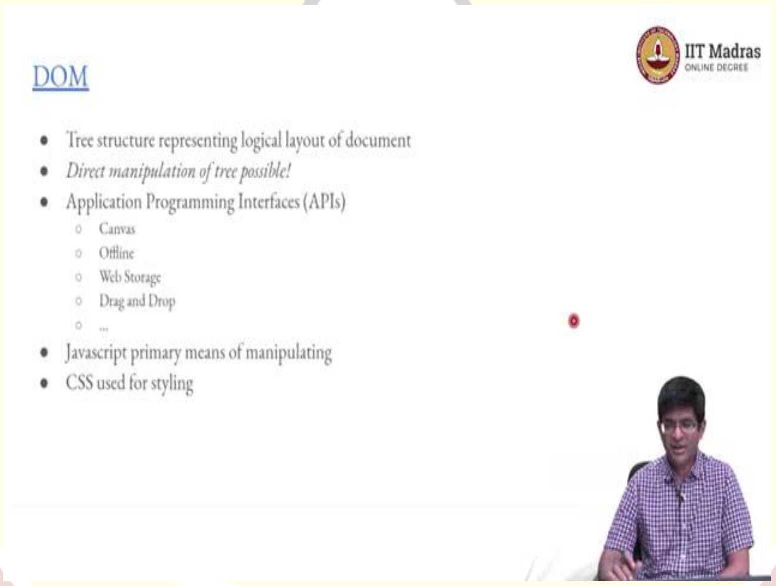
Now, this is the document as we see it before it is actually displayed. The first thing that the browser needs to do is to parse this that is to say take out all of these tags and say the tags are actually telling me something about structure, about how the entire thing is to be interpreted. And that is where the document object model comes into the picture. What it says is each of these tags can, this entire structure can be interpreted as a tree like this.

You have a top level document. Inside the document, your root element is this <html>, which basically corresponds to everything which is inside the document. Inside <html>, I have <head> and I have <body>. I have an element <head> and I have another element <body>. Inside <head>, I have a <title>. And inside <title>, I have the text of the title. Similarly, inside <body>, I have a heading <h1> and inside the <h1>, I have the text of that heading. I also have a link that is a tag. It has an attribute. This is not part of the text. It is an attribute. It is an href. And it also has the actual link itself, this link text which is available over here.

So, this HTML, the first thing that the browser does when it sees it is to try and parse it or interpret it as a tree structure like this. Why is this useful? Because ultimately you need to understand this document object model in the context of web apps, because you might have a way by which you can directly manipulate parts of the DOM.

So, there might be something where in a programmatic fashion you can go and directly change this <h1> element which is inside <body> or you could set the style corresponding to body h1 and say, this is what this heading should get displayed as. So, although it is not very important for us right at the beginning, understanding the DOM is important in terms of how we would modify the structure of a page or how it gets displayed.

(Refer Slide Time: 20:54)



DOM

- Tree structure representing logical layout of document
- *Direct manipulation of tree possible!*
- Application Programming Interfaces (APIs)
  - Canvas
  - Offline
  - Web Storage
  - Drag and Drop
  - ...
- Javascript primary means of manipulating
- CSS used for styling

So, it is a tree structure representing the logical layout of the document. What that means is the direct manipulation of this tree is now possible, because computer scientists for a long time have a number of algorithms that operate on trees. It means that we know how to sort of move trees around, restructure, change the parent element of a given element, change the contents of one, go down a tree all the way and change all the elements inside it in a certain fashion, all of that can be done nicely using algorithms that are already known. So, the moment you have this kind of a nice structure to the document, all those things get opened.

Now, in addition to that, a number of APIs are also defined on the DOM, the Document Object Model. One of them is the so called Canvas, which basically allows for drawing arbitrary things on the screen. There is an offline API which actually tells you that certain websites, you might have come across sites that say, click here to install it as an app. What that means is that the HTML corresponding to that site gets stored on your local machine. And the next time you access it, it will run even if you are not connected to the internet.

There are now APIs for web storage which basically allow you to store certain information on your local machine, which means that it can become persistent. The next time that you go back to the site, it might still remember where you left off and be able to sort of work with that. There are sort of utility APIs, like drag and drop, which basically allow you to drag a file from your operating system folders and drop it into a certain location.

So, in other words, it brings this whole drag and drop that people are already used to on the desktop into the web browser. So, all of those are defined as APIs, Application Programming Interfaces, which the browser sort of standardizes and says, if you use these APIs, you will be able to get this kind of functionality.

Now, in most of the cases, when we talk about manipulating the DOM, we use JavaScript. That is not something that we are going to be getting into in any level of detail now or in the near future. But it is something to keep in mind if you are interested in advanced development of these applications.

And the other thing is basically styling. How do I also control the representation of different aspects of the DOM that is where CSS comes into the picture. And what is CSS is something that we do need to understand in a bit more detail.