IIT Madras

ONLINE DEGREE

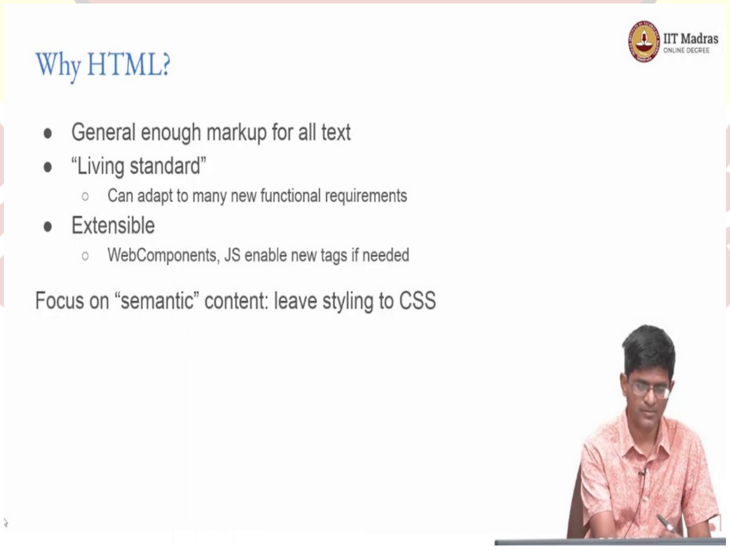
Modern Application Development-II
Professor. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Markup Alternatives

(Refer Slide Time: 0:12)

Modern Application Development - II
Markup Alternatives
Prof. Nitin Chandrachoodan
Department of Electrical Engineering
IIT Madr

Hello, everyone. Welcome to Modern Application Development part 2. We are now going to look at some alternatives to the core idea of Markup.

(Refer Slide Time: 0:20)



Why HTML?

- General enough markup for all text
- "Living standard"
  - Can adapt to many new functional requirements
- Extensible
  - WebComponents, JS enable new tags if needed

Focus on "semantic" content: leave styling to CSS

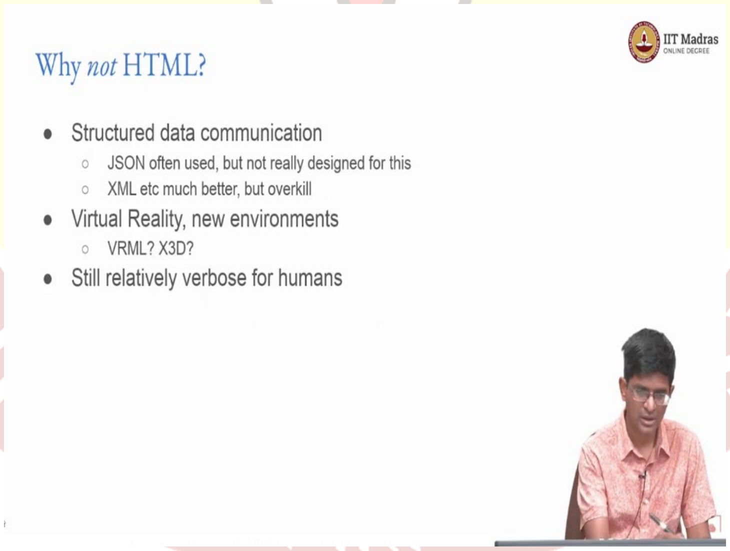
So, the first thing before we start looking at alternatives to HTML, let us first understand why HTML is used. and why is it so popular? The thing is it is a general enough Markup for pretty much all text type of documents at this point. And somewhere along the line, the fact that it morphed into the living standard, meaning that it is continuously updated and can potentially adapt to any new functional requirements meant that it is now in principle, powerful enough to do anything that you want.

You could propose a change that essentially adapted to the changes that you need. But more importantly, it is also extensible at this point in time, meaning that you have web components, you have JavaScript APIs, which basically enable you to, create new tags, if needed and add functionality as required. Which means that with HTML?

It is a simple enough language that you can use it to focus on "semantic" content, leave the styling to the CSS, the cascading style sheets, and not how to deal with as much complexity as something like SGML or even XML. XML is actually a simplified version, relatively, not so difficult to use as SGML, but HTML turns out to be even simpler than that.

It is a very at its most basic it is a very easy language to use. It is also very forgiving because of the nature of the way that web browsers written. But it is also possible to have very well-structured HTML that can be created systematically.

(Refer Slide Time: 1:59)



Why *not* HTML?

- Structured data communication
  - JSON often used, but not really designed for this
  - XML etc much better, but overkill
- Virtual Reality, new environments
  - VRML? X3D?
- Still relatively verbose for humans

Now having said all that, there are also reasons not to use HTML. So, why not HTML? The fundamental place where it breaks down is when you are trying to communicate structure data. That is to say, let us say, I want to actually give some information corresponding to an object. I want to send it back from a server to a client. Mostly what happens is that you would then replace it with something like JSON. Because that is you would essentially say that.

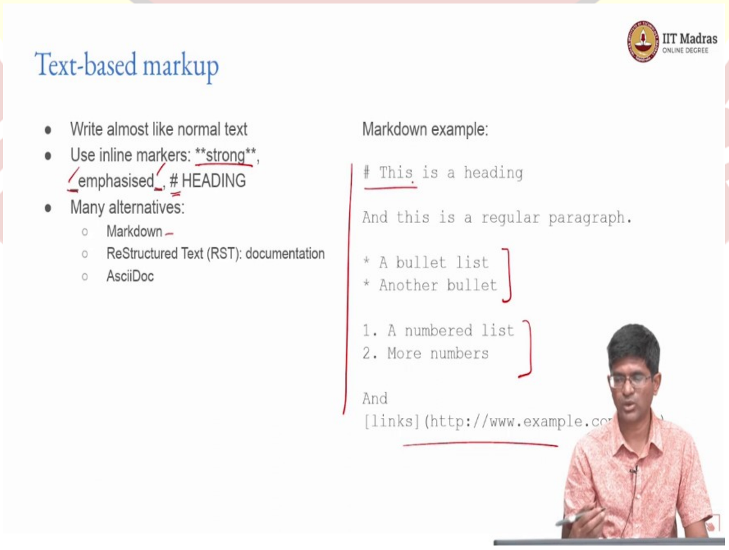
Look, I want to actually convey the information about the object, which means that even on the client side, it is going to be some code. That is going to look at the object not a human being. So, why bother with HTML tags? Let me use something like JSON instead. JSON has its own problems.

As we saw earlier. It is to represent objects and it, at least dictionaries, but type information cannot really be specified as part of JSON without some kinds of hacks on top of it, XML is much better, but very often is overkill. And the point is, JSON is so simple to parse and use that very often that is what gets used in most cases. But what about new forms of Mark-up. Virtual reality. And the interesting thing is, even though I am saying new there was something called VRML virtual reality Markup language that was actually proposed in 1995.

So, it is by no means new and the thing is it is there is VRML, which has been succeeded by X3D. This is for new kinds of environments. As we move forward, we are likely to come up with new user interfaces and new kinds of contexts in which we need to use Markup where HTML may not be the best fit.

And apart from all of this, HTML is still sort of relatively verbose for human beings to write. It is easy enough to understand during you can sort of look at the HTML and say, mostly I understand what is going on, but when I am actually writing a document, maybe I do not even want to see that much.

(Refer Slide Time: 4:03)



Text-based markup

- Write almost like normal text
- Use inline markers: **strong**, _emphasised_, # HEADING
- Many alternatives:
  - Markdown
  - ReStructured Text (RST): documentation
  - AsciiDoc

Markdown example:

```
# This is a heading

And this is a regular paragraph.

* A bullet list
* Another bullet

1. A numbered list
2. More numbers

And
[links](http://www.example.co
```

Now, there is an alternative which is to use what is called text-based Markup, where you can almost write like normal text and you use just some kind of inline markers. So, something like, use asterisks to indicate something as strong or to be displayed in bold. Use something maybe emphasise like this, or, you know using slashes that could be different ways by which it is done.

I use a hash in front of a string in order to indicate that the entire line corresponds to a heading. Now, there are all, this is roughly what corresponds to Markdown, which is one of the alternatives that you might have already heard of. Markdown is sort of a play on words. It is basically same, this is not Markup or rather. It is a different type of Markup. So, it is called Markdown.

What is it trying to do? It is using the inline text, meaning that there are just some indicators, some special characters that are introduced into the text, but those special characters are not something that are going to be problematic, even a person looking at those characters will sort of understand straight away that what is the meaning or what is the idea behind it? Once again, convention.

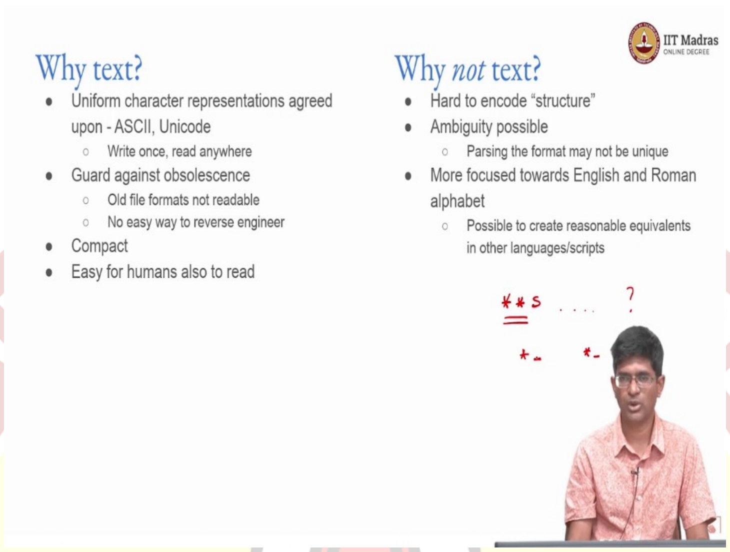
But if I look at something like "star, star, strong, star, star", then even if I do not know anything about it, chances are, I would sort of, at least if I see it in a few different places, I would get the Idea that you are trying to make a strong point over here. You wanted it to be in bold. That is why you are putting it as stars.

So, this is inline special characters. It is not the kind of special characters that are sort of, what is called the presentational Markup. Where you actually hide it. So, word documents, for example, do have characters. That do not get displayed because word automatically hides them and shows you the bold version of the text, for example. But if you look inside the document, you would find that there is actually a character that stays, that says start making it bold from here.

So, Markdown is one of the alternators. There is the RST format, which is often used for documentation, especially in Python code. That is ASCII doc. There are many different variants of what can be used. This is an example of what a Markdown text would look like, and you can look at it and see that it is readable. It is very readable for a human being.

A person reading this word straight away be able to say, look, this is an itemized list. This is enumerated list. This is a link. and yeah. You know, once I know the conventions, I would know that this is a heading and therefore I know to look for headings and subheadings and so on. I can just look at this document even without a special kind of display editor or anything like just a regular text editor, makes it very easy for me to write such documents, so, this is very human friendly.

(Refer Slide Time: 6:54)



Why text?

- Uniform character representations agreed upon - ASCII, Unicode
  - Write once, read anywhere
- Guard against obsolescence
  - Old file formats not readable
  - No easy way to reverse engineer
- Compact
- Easy for humans also to read

Why *not* text?

- Hard to encode "structure"
- Ambiguity possible
  - Parsing the format may not be unique
- More focused towards English and Roman alphabet
  - Possible to create reasonable equivalents in other languages/scripts

So, what are the reasons for going for this kind of approach? The uniform character representations have already been agreed upon. So, there are standard representations for ASCII, for Unicode, which allows us to pretty much represent any character that we have using some display, but there is no Unicode for showing that something is bold, for example, because that is not part of a character encoding. That is part of a styling.

Now, the thing is once you have this character encoding and it is just specified as standard, the chances are that you will not go obsolete. Meaning that even a person looking at this Markdown document 20 years or 30 years from now, will be able to reasonably well make out what you intended. Whereas if we go back around 20 or 30 years, there are word format documents, not Microsoft word, but other formatting documents. That probably were created by software that no longer exists.

It is not, not even going to run on your present systems. What do you do? The software cannot run on your present system, even if it did, it does not have any simple way of exporting it to something that you can use. So, when you have a problem like that, suddenly you can be faced with a situation where you have a document that was created in a certain way. And now nobody can read it. That is what is called obsolescence.

And it can be problematic, especially for archival purposes. The other thing about textures, it is very compact and it is easy for humans to read, . But at the same time, there are some issues. So, why not text? In particular, it is hard to encode structure. So, if I want to have structured data, then you cannot really do it using this, even something as simple as a table.

There are some conventions that are defined for tables, but they are not usually part of standard text-based Markup. And there are many different variants on how to do it. The bigger problem is ambiguity is sometimes possible, meaning that it is not always possible to say what was intended in a particular Markup that has been implemented.

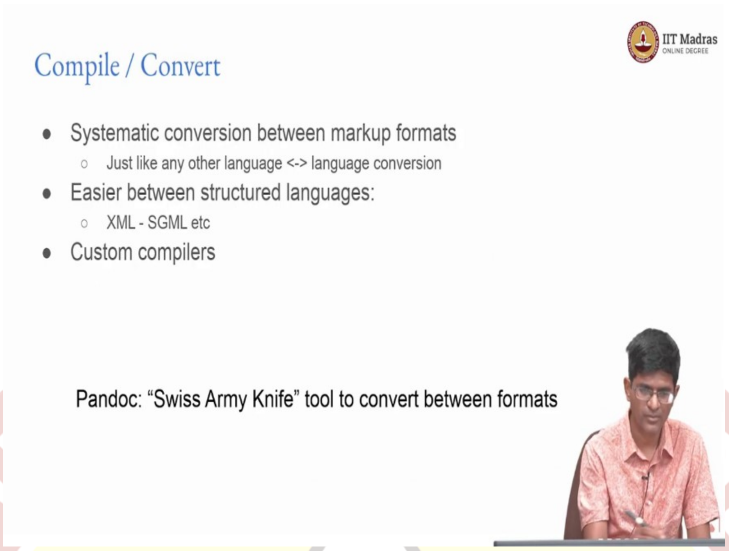
So, for example, if I have something like S and I just leave it like this, was it supposed to be that this was an opening and closing bold and that, you know, after that, I just need to move on. Or, was it that at the end of the day, I forgot to put in the extra asterisks that, close it out and actually make it bold.

If I have something like star and an underscore, how should I interpret it? Should it be actually going? And, you know, I, if I have some other syntax over here, , is this actually wrong? Or, does it just mean that whatever this underscore should also have been made bold which are the ways there are different ways in which you could go about interpreting different Markup styles.

That is not really possible in the case of the explicit tanks where you have opening and closing tags and the nesting rules are very clearly specified. And the other thing is, it is sort of more focused towards the English and the Roman alphabet in general. You could of course use Markdown in Hindi or Tamil or Chinese or any other language, but some of the characters, if you look at them, the fact that we use a hash for this and you know, the 1, 2, 3 numbers in order to indicate and dash for indicating the start of a rather itemized list.

They may not always make sense in different languages. Even something like a full-stop has a different symbol in hindi than it does in English. So, would that have a different interpretation? Should I define a new markdown just for Hindi? What about Chinese? Where a horizontal line basically represents the number 1. Would that be confused against the start of a list, for example? How should I go about doing it? Do I use the same kind of approach or is it simple enough that it can be used? There are potential problems there.

(Refer Slide Time: 11:01)



Compile / Convert

- Systematic conversion between markup formats
  - Just like any other language <-> language conversion
- Easier between structured languages:
  - XML - SGML etc
- Custom compilers

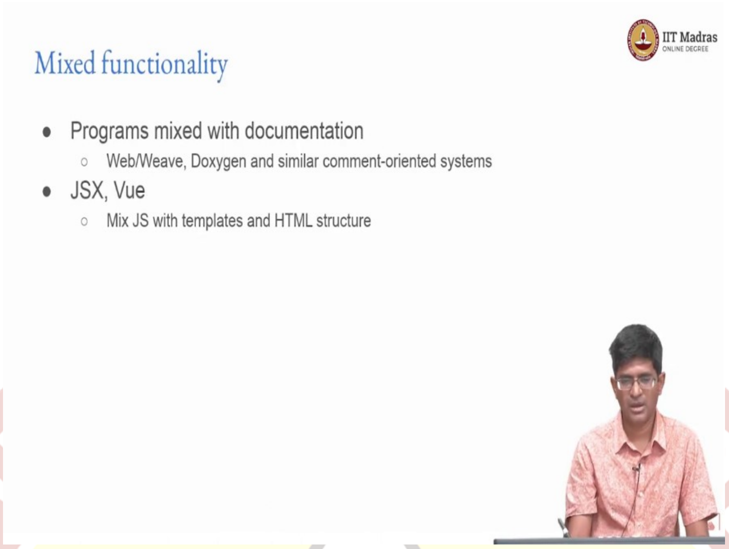
Pandoc: "Swiss Army Knife" tool to convert between formats

Which means of course that one of the things that can be done is you create systematic conversion techniques between different languages and it is actually a compilation process. You are taking one language and converting it into another language with possibly certain kinds of optimizations or improvements somewhere along the way.

But for the most part, you are not really doing optimization. So, probably conversion is a better word than compilation in this case, in most cases, but there could be compilation where, for example, you take something like Latex or Markdown input and convert it all the way into something like PDF, where it is unlikely that you would go back from PDF to Markdown and the PDF has enough extra input over there, that it is something like a compilation step.

And a very useful tool in this context is what is called Pandoc. It is advertises as itself as a "Swiss army knife", a general versatile tool that can be used to convert between formats and it lose up to its name. It is fantastic in terms of being able to convert between various different kinds of formats and very powerful, very fast and useful in working with various text-based formats.

(Refer Slide Time: 12:14)



Mixed functionality

- Programs mixed with documentation
  - Web/Weave, Doxygen and similar comment-oriented systems
- JSX, Vue
  - Mix JS with templates and HTML structure

Now, Markdown or Markup rather could go further. There are sort of what is called mixed functionality where you could have a program mixed with documentation. And one of the earliest was something called Web where literally it was used in order to create so called literate documents or literate programming.

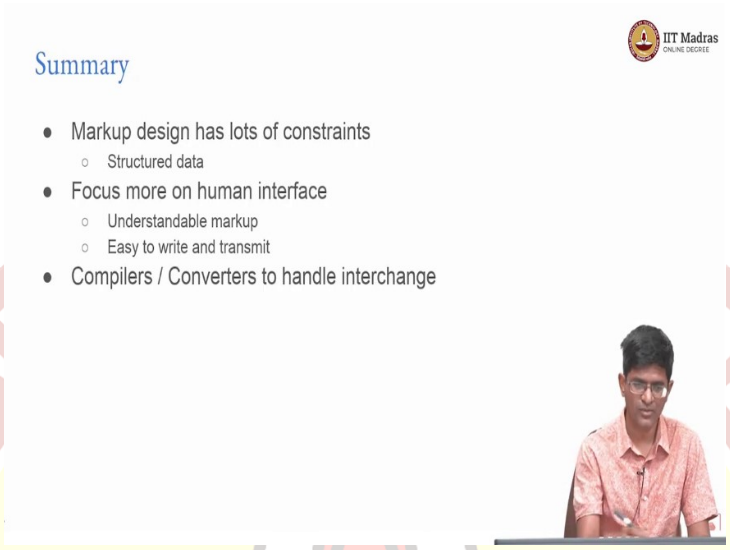
And the idea was that you would write a document and within that doc, document explaining what you are trying to do. And within that document there will be certain parts that correspond to code and there will be a compiler that will be able to extract those parts and convert them into actual code that would run.

So, it had a very nice, at least conceptual approach. It turned out that it was easier to sort of just focus on code and write the documentation separately. And quite frankly, people are poor at writing documentation. So, it never really took off in a big way, but the ideas behind it are fairly solved.

You would see something slightly similar in techniques like Doxygen that are used in order to generate documentation from the comments that are added around functions in code. On the other hand, coming back to frontends that we have been looking at recently view templates, themselves have the ability to sort of mix some amount of templating with the JavaScript code.

And there is also JSX that is used for react and it allows you to much more flexibly mix templates with HTML code. Or HTML structure, and also Java Script code. So, you could have a mix of the HTML template and Java Script all sort of closely intermixed with each other. The point is these are all different extensions of the core idea of Markup that allow you to handle various different kinds of specific functionality in efficient ways.

(Refer Slide Time: 14:08)



Summary

- Markup design has lots of constraints
  - Structured data
- Focus more on human interface
  - Understandable markup
  - Easy to write and transmit
- Compilers / Converters to handle interchange

So, to summarize Markup design, if you really go for it has a lot of constraints, actually specifying structure data needs to be done with care and the focus is for the text-based markup languages is more on the human interface. You want it to be understandable, Markup, that is easy to write and transmit.

And one of the reasons for this is when I am actually writing, let us say a blog post, or I am trying to create documentation for Wiki. I do not want to focus on each and every open and closed tag and whether the tags are nested properly and so on. I would rather write in a fashion that is easy for me to use.

And not only that, it turns out that in a lot of cases, this documentation is being done by programmers, who are not really using word processes. They are not using Microsoft word, which means that the actual Markup is not easy, you have to sort of transition between different documents in order to do this.

If I wanted to have something, which is just part of my code itself, I would just be editing it with the regular text editor, and over here, Markdown and the similar kinds of text-based languages are very powerful because they make it easy to use. The fact that there are converters that handled various kinds of interchanges just means that they have now become a very popular way of approaching this whole Markup problem.