

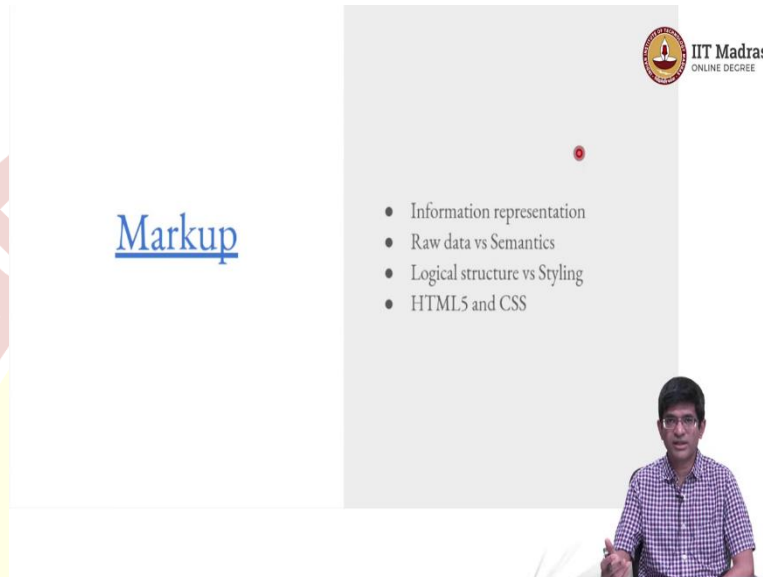
IIT Madras

ONLINE DEGREE

Modern Application Development – I
Professor Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology, Madras
Information Representation in Machine

Hello, everyone, and welcome to this course on modern application development.

(Refer Slide Time: 00:16)



We are now going to look at markup, and what it means, and how it is used in order to control how something gets displayed on a screen. Now, this is a very primary or a main part of what is called the user interface, the part that faces the user, and how we interact with the computer in the first place.

So, markup is something that is used in order to change the way something is shown to the user and ultimately deals with the aesthetics or the sense of beauty or the sense of how nice something looks. And obviously, as you would all be familiar by now, the better that a site looks, there are some sort of indefinable ways in which you can clearly say that one site looks better than another. In some cases, it is obvious in other cases, it is less obvious, but there is a concept of aesthetics that finally determines how we interact with a website.

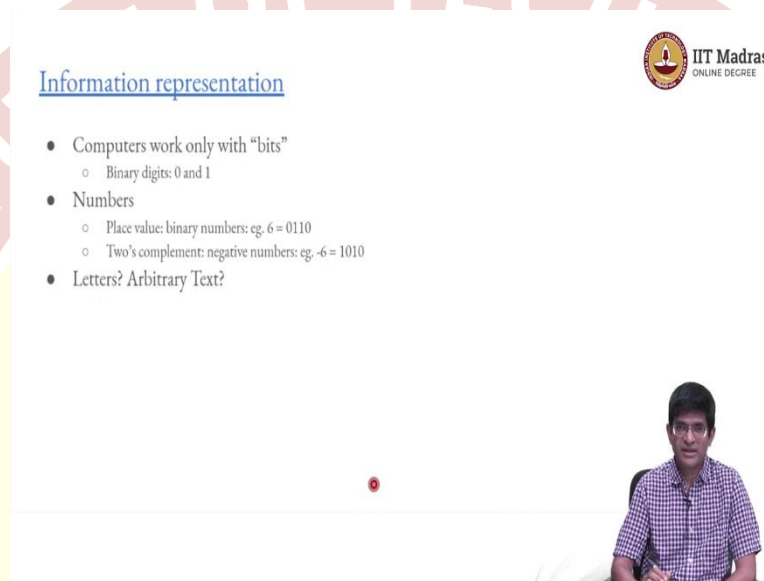
So, markup plays a very important role in that because ultimately, that is how the developer, the person who is developing the website can make changes to the website in such a way that the way that it gets displayed to the user changes. So, let us try and understand what it is.

So, we are going to look at a few things. First of all, how do we represent information inside the computer? There is also a distinction between the, so called raw data versus the

semantics. What does it mean? We will also try and understand the distinction between logical structure of a document versus the styling, how it is presented to you.

And finally, we will look at HTML5 and CSS, but only at a very high level, primarily, because those are the technologies that we will be using as part of this course. Because as far as the web is concerned, those are the primary underlying presentation technologies. The parts that determine how a page looks.

(Refer Slide Time: 02:10)



So, first things first. What do we mean by information representation? As you are all familiar, computers work only with bits, because ultimately, they are digital logic systems, which means that you are essentially everything inside the computer. Any, not just computation, but even information is stored in the form of bits, either a 0 or a 1.

This works out nicely, because from an electrical standpoint you have either 0 volts, which is essentially the ground level, or a 1, which is some kind of a high voltage level, high could possibly be something like 1 volt event, so it is not really a high voltage, it is just something which is clearly distinguishable from the 0.

Why do we do this, because it turns out that not only can you sort of build up a nice algebra and way of manipulating all the bits so that more involved computations can be performed, they are also very robust to noise. In other words, if there is any disturbance or uncertainty or temperature fluctuations none of those are going to affect the system, at least not easily. So, computers and pretty much any digital logic that we have today work with bits. Those are essentially a bit is a binary digit means that it can take on only two values 0 or 1.

So, unlike the alphabet, for example, where I can write ABCD, up to Z, 26 letters or the Indian alphabets most of which have somewhere in the region of 50 characters or so 50 alphabets, we have here only two, 0 and 1, which means that everything inside the computer has to be represented using these bits.

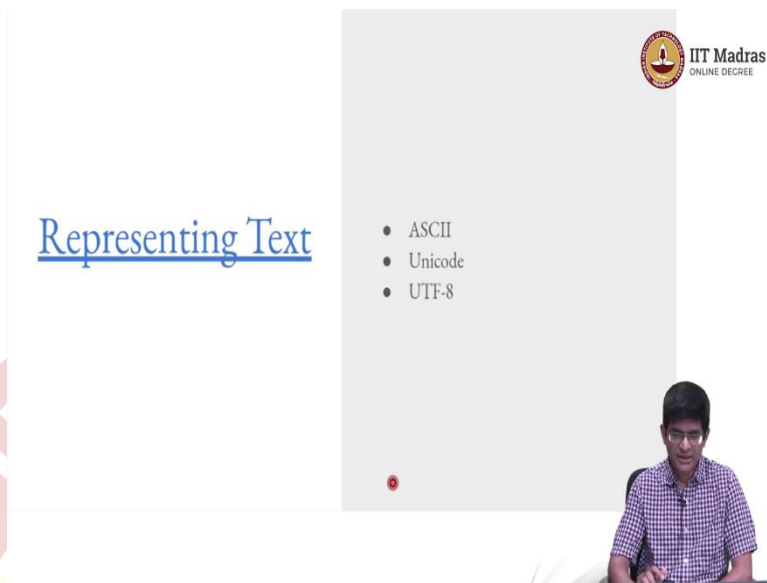
What that in turn means is, I cannot just look at a 0 or a 1 somewhere arbitrarily and decide what it means, I need to follow certain conventions. I need to say if a 0 or 1 appears in this particular location or in this particular place or in this particular context, I will interpret it in a certain manner. So, we know already that if I have decided that a certain set of bits represents a number, then I can interpret that number.

For example, we know that we can represent binary numbers, for example, the sequence 0110 would represent the number 6. This is based on the place value system. So, it is 0 into 2 to the power of 0 plus 1 into 2 to the power of 1 plus 1 into 2 to the power of 2 plus 0 into 2 to the power of 3, which finally gives us 6. Now, what about negative numbers? Remember, we have only 0 and 1 so we cannot even afford a minus sign, which is like the 2s complement representation comes into the picture.

Again, it is a question of interpretation. If I see the sequence 1010, I know that if the number that I am talking about is a 4-bit 2s complement number, then this value, the value corresponding to 1010 is minus 6. Instead, if you told me that it was a 4-bit unsigned number, it would actually have the value plus 10 or if I told you, I do not know what this is, then the best that you can say is this is a sequence 1010, you do not really know what it means.

But then comes a question what about? So, numbers so far, so good, we have a handle on how we might be able to represent numbers. The concept of context and how we interpret these numbers is still important, but at least the coding itself is easy. But what about letters? There is no natural coding which says that a particular number or a particular sequence of bits should correspond to A or another sequence should correspond to B.

(Refer Slide Time: 05:54)

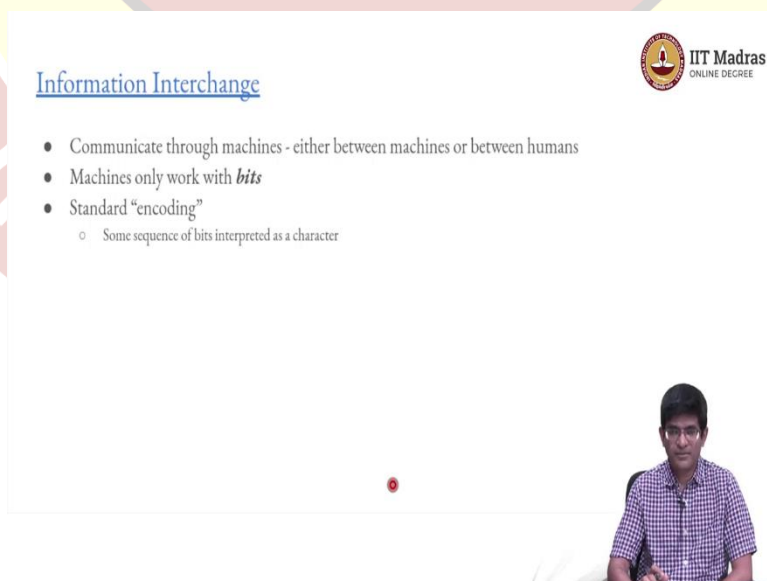


Representing Text

- ASCII
- Unicode
- UTF-8

So, that basically takes us to the question of how do we represent text? And we will take text to be the sort of the most important thing that we need to represent. Because any other thing can be similarly encoded, it is only a question of that we understand in what context we are trying to represent the sequences that we have. So, there are a few different concepts that we need to sort of keep in mind over here. And I will be going over those in the coming slides. So, representing text.

(Refer Slide Time: 06:30)



Information Interchange

- Communicate through machines - either between machines or between humans
- Machines only work with *bits*
- Standard “encoding”
 - Some sequence of bits interpreted as a character

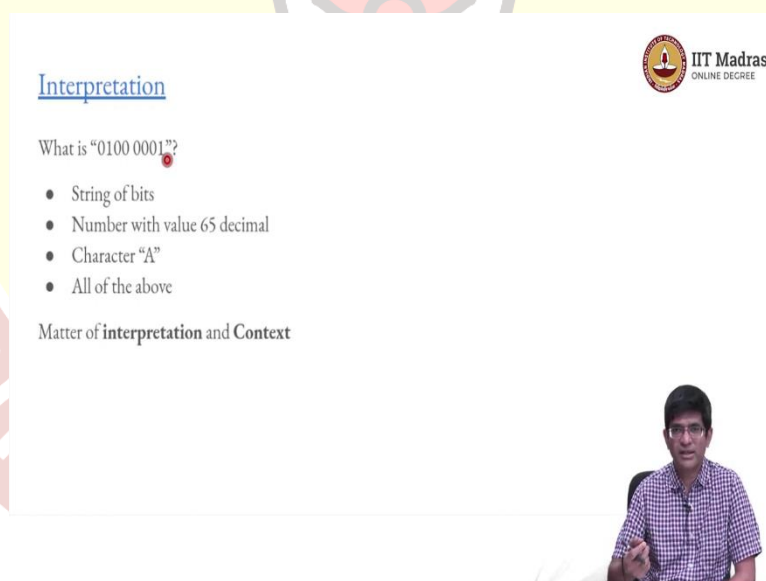
The core idea that people recognize a long time ago, pretty much as soon as digital computers came into existence. Why am I saying digital computers? Because there were analog

computers before that. I mean, there were things, which basically did some kind of differential equation solving and so on, which in some sense for analog computers. So, a computer is anything that can compute.

A digital computer is something that works with digital logic. Now, what that means is, you need to be able to exchange information between such systems, such computers. And this information interchange could either be between machines directly, one machine talking to another. Let us say a computer talking to a printer or between humans and machines. I mean, between, directly between humans, we are unlikely to have to represent letters as bits, but when a machine is trying to display something to a human or a human is trying to communicate something to a machine you need to be able to talk in a common language.

And the machine is one that is less flexible. It requires the language to be something that it can understand. Machines only work with bits, 0 and 1 are the only alphabets so to speak available to you, which means that we need to decide on some kind of a standard encoding and say that this sequence of bits represents a certain character.

(Refer Slide Time: 07:59)



Interpretation

What is "0100 0001"?

- String of bits
- Number with value 65 decimal
- Character "A"
- All of the above

Matter of interpretation and Context

IIT Madras
ONLINE DEGREE

So, this brings us to the question. Supposing, I just put down this sequence of bits in front of you. 0100 0001. It is a perfectly valid binary sequence, and the question we are asking is, what does that binary sequence represent? First, is it a string of bits? Of course, it is a string of bits, that is undeniable. It is 0s and 1s, so there is no problem with that it can always be interpreted as a string of bits that is not very helpful.

But if I take this the, so called, binary system with the place values and so on, and say I am going to interpret this as a number 65 decimal, yes, that is another perfectly valid interpretation of the sequence of bits. But then, I come along and say, I am instead going to interpret this as the character, capital A, it is a single letter, capital A.

Which of these is correct? Of course, all of the above. And what it means is that it is basically a matter of how we interpret the data and more importantly, context. If I just saw the sequence 0100 0001 as a set of random bits scribbled on a wall, I do not know what to make of it. But if I see it, in let us say a maths book or in something, which is discussing binary arithmetic I am almost certainly going to interpret it as the number 65. Whereas, if I see it somewhere else, which is talking about character encodings or how we can represent letters using bits then it makes sense that it is probably the code for A.

So, the context in which we see this makes a big difference to how we interpret bits. In the same way inside a computer there is a large memory layout, there is a lot of place where bits are stored, the program itself is not randomly going around picking up arbitrary locations in memory and deciding what to do with them, it goes through them in sequence, which means that as far as the program, the execution of the program is concerned, certain memory locations have specific meanings.

So, you might have declared a variable as a string constant and the internal program after compilation would have said that the pointer to this memory location is such and such, which means that if I go to that memory location I have context, which tells me that this string of bits should be interpreted as characters.

(Refer Slide Time: 10:38)

ASCII



- American Standard Code for Information Interchange
- 7-bits: 128 different entities
 - 'a' .. 'z'
 - 'A' .. 'Z'
 - '0' .. '9'
 - Special characters: !@#\$%^&*() ...
- Why 7-bits?
- What about other characters? अ अ अ अ अ
 - 1000s of characters needed



So, with all of that in mind, the U.S. which was sort of the leader as far as the digital technology in the 60s and 70s, at least was concerned, they propose this thing called the American Standard Code for Information Interchange, Information Interchange. It is basically for how we can transfer information between devices or between humans and machines. And that is what we now know as ASCII. You might have heard the term ASCII codes this is precisely what ASCII is.

So, it is a standardized code, which basically describes how specific strings of bits should be interpreted. And the original ASCII was a 7-bit code. Why 7 bits, because it was a time before even 8-bit computers were standardized. The number of bits was something precious, you did not really want to use up more bits than necessary, so people actually tried to optimize the number of bits. And the fact that you could use 7 bits for something probably made more sense than unnecessarily using 8-bits.

So, it is a 7-bit code. Among other things, it had encodings for all the small letters A to Z, and the capital letters A to Z. Note that these are actually treated differently. ASCII, of course, did a nice thing in the sense that the codes for I mean between capital A and small a differ only in a single bit, but that is just for our convenience. As far as the computer is concerned, it probably does not make too much difference, they are independent codes anyway and a computer does not really understand the letter A, it just sees a code and needs to know what to do with it.

Apart from these, of course, you also need the numbers 0 to 9. Several especially characters, exclamation marks the at symbol, which we are very familiar with now, but which was not really used all that much, then, but things like dollar signs, percentage signs ampersands those are very common, of course, meaning that there are special characters. So, what do we have?

We have 26 alphabets, 26 uppercase alphabets, 52 another 10 digits 62. A number of special characters probably takes you into the 70s or so something for a space, something for full stop, something for a comma and a number of escape characters. I mean, so something like just extra characters, which are sort of special characters that may not really have a specific meaning outside of additional computer, like Enter or Return or the carriage return line feed those kinds of characters.

End result was, somewhere in the region of more than 60, but less than 100 characters that you need to represent. What is the minimum number of bits you need 7 with 7 bits you can represent up to 128 entities with 6 bits, you would have managed only 64, it would have been a tight squeeze, I mean, basically not possible if you also wanted special characters, so 7 bits.

But the question naturally arose after some time. I mean, for a long time ASCII was pretty much a good standard. Of course, after some time, the Europeans got fed-up, because European languages have a lot of special characters of their own. They have like the accents on top of E and A and various other things and they needed encodings, which could represent those, which meant that pretty much something like an extended ASCII format came into the picture, which added one more bit.

It became an 8-bit code, you can now go up to 256 characters, that was enough for pretty much all scripts that are derived from the Latin alphabet. The Roman alphabet or Latin alphabet the ABCD that we are familiar with, but there are a lot of other languages. So, for example, within India itself, you have Hindi, you have Tamil, you have Malayalam. Outside India, you have Thai, you have Chinese, Japanese, various other languages. You start adding all of these together, and you find very quickly that 256 is just not going to cut it, you need thousands of characters to be represented.

(Refer Slide Time: 14:46)

Unicode



- Allow codes for more scripts, characters
- How many?
 - All living languages? All extinct languages? All future languages?
- “Universal Character Set” encoding - UCS
 - UCS-2: 2 bytes per character - max 65,536 characters
 - UCS-4: 4 bytes per character: 4 Billion+ characters



So, eventually, after a lot of, I mean, the whole point over here is that you need to have a standard representation. Everybody cannot come and develop their own representations because it means that you cannot communicate with each other after a while. So, the Unicode was sort of created as a consortium. Many different companies, universities, various interested parties came together and created a standard, which allows codes for many more scripts and basically for different characters for letters. You still have the question how many should we have. Should you support all living languages? All living as well as past possibly extinct languages?

What about hieroglyphics? What about the Sumerian cuneiform script? Should you have the actual Unicode letters corresponding to each one of those, even though we have not even interpreted those languages completely. What about future languages? Maybe some new script comes along, which is found to be more efficient or somehow different or we meet aliens at some point, does not matter.

The point is, how do you stop. So, at least the idea behind Unicode was to come up with some kind of a universal character set encoding. The original UCS was sort of a 2 by 10 coding, 2 bytes per character. And at that point, they pretty much had this restriction that 2 bytes means a maximum of 2 to the power of 16 or 65,536 different permutations, different combinations of those bits.

Now, in fact, the original Unicode while it was being defined, they pretty much explicitly said that the idea here was if you have past languages or things which are not really commonly used, they will not be encoded in Unicode, think of some other ways of encoding them. A valid way of looking at it, part of the reason being that if you want to encode more,

the first things you do not know how many might come and the second is, you are going to use more bytes per character. You already have 2 bytes per character over here, which means that you have doubled the space required to store even a simple text document. Let us say that you had a small document with like a 100 words in it normally that would be 100 into 5 around 500 bytes or so.

Whereas, with UCS-2, 2 bytes per character, you now need 1 kilobyte exactly double. But anyway, at some point people said look there are enough languages that the 65,000 can be a problem, it makes it too tight. So, let us look at UCS-4, which is 4 bytes per character. Now, you can go up to at least 4 billion characters. Is that enough? Well, we never know. At least right now it looks like it might be enough, but in future will we encounter a situation where we do not have enough encodings possible.

Now, at present out of these 4 billion possibilities somewhere around I think, more than a 100,000 encoding spaces are defined. So, in other words, out of the possible 4 billion combinations of bits of 32 bits, that is 4 bytes, about a 100,000 or so actually have Unicode definitions. The rest are pretty much undefined. So, you can see that it is fairly sparse. Only a small number of them actually have definition. But it does not matter, it was a conscious choice between the efficiency of the encoding and the capability of encoding more information.

