

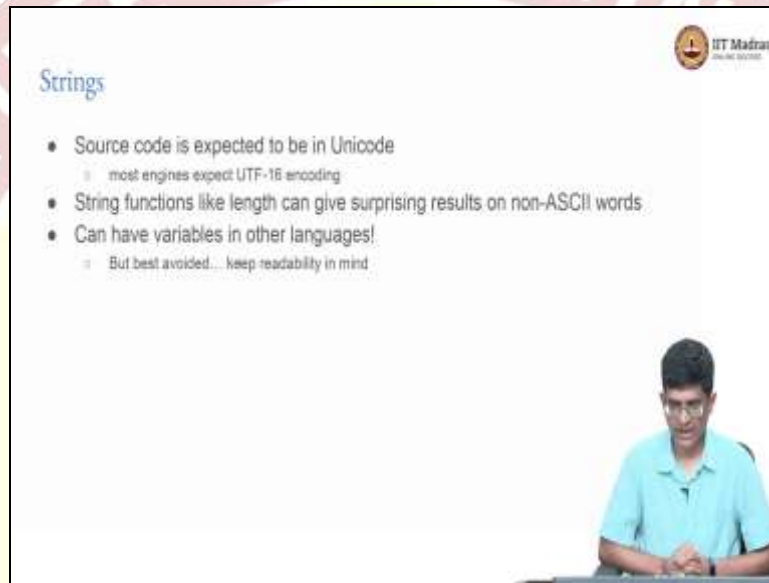


IIT Madras
ONLINE DEGREE

Modern Application Development II
Online Degree Programme
B. Sc in Programming and Data Science
Diploma Level
Prof. Nitin Chandrachoodan
Department of Electrical Engineering
Indian Institute of Technology- Madras

JavaScript - Control Flow and Functions

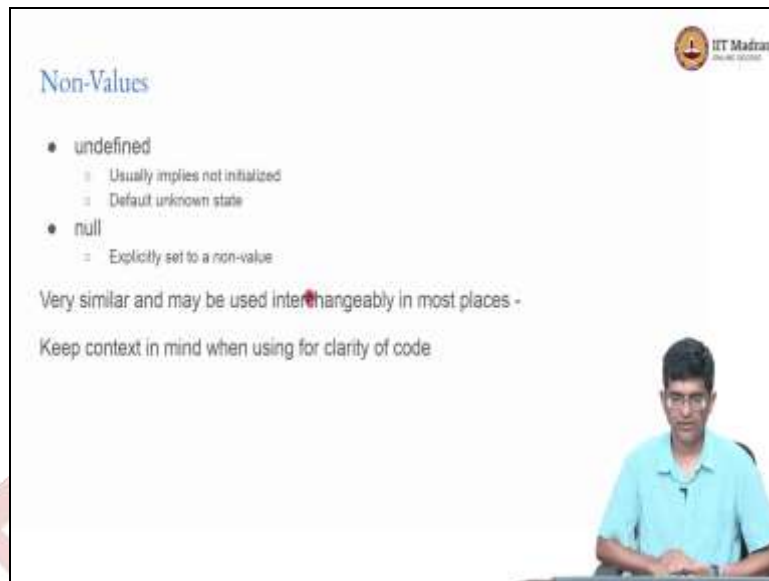
(Refer Slide Time: 00:14)



Hello everyone welcome to modern application development part two. Ok, I had already mentioned or rather demonstrated a little bit about strings, right. As I mentioned in the case of strings most strings expect utf-16 encoding and functions like length can give surprising results. And when I say surprising you need to just keep in mind that you know you need to compute keeping uh in taking into account the fact that it is in Unicode and utf-16 format.

But it is not that it is going to give you random results it is predictable but you need to know how to actually compute it. You can have variables in other languages but please keep readability in mind and generally better to avoid this unless absolutely necessary.

(Refer Slide Time: 00:55)



Non-Values

- **undefined**
 - Usually implies not initialized
 - Default unknown state
- **null**
 - Explicitly set to a non-value

Very similar and may be used interchangeably in most places -

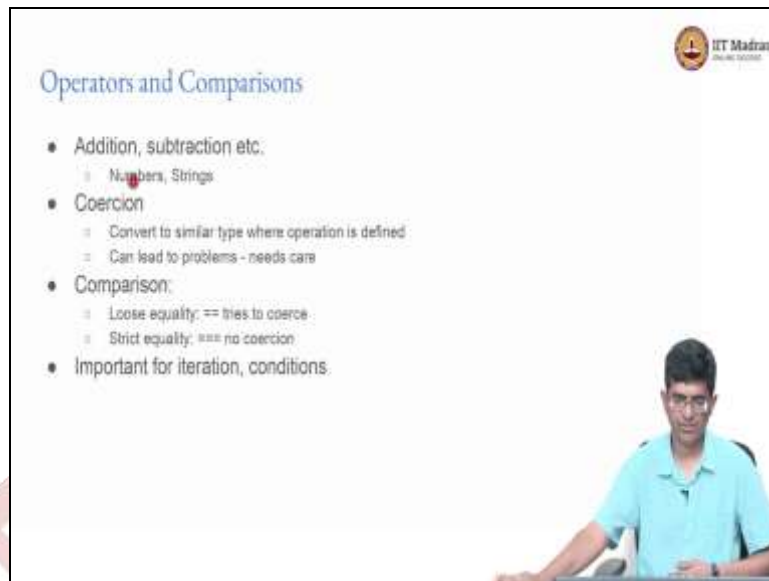
Keep context in mind when using for clarity of code

IT Madras

Undefined and null as I mentioned are the so called non values, right. The difference between these two is that undefined is usually used for things that have not yet been given an initial value, right. So, when I declare an array an empty array with three elements in it the elements in it are undefined. But if I actually declare the array and I put null into the values over there then they are null.

So, that is a slightly different thing you explicitly want it to be null rather than just leaving it undefined and unknown. OK. So, there is a subtle difference in a lot of cases they can be used interchangeably meaning that you can use undefined where you would like to compare against null or vice versa but it's probably good to keep the context in mind when you are using it. So, that your code is clear, right and you know when you want to use undefined or when you want to use null.

(Refer Slide Time: 01:49)

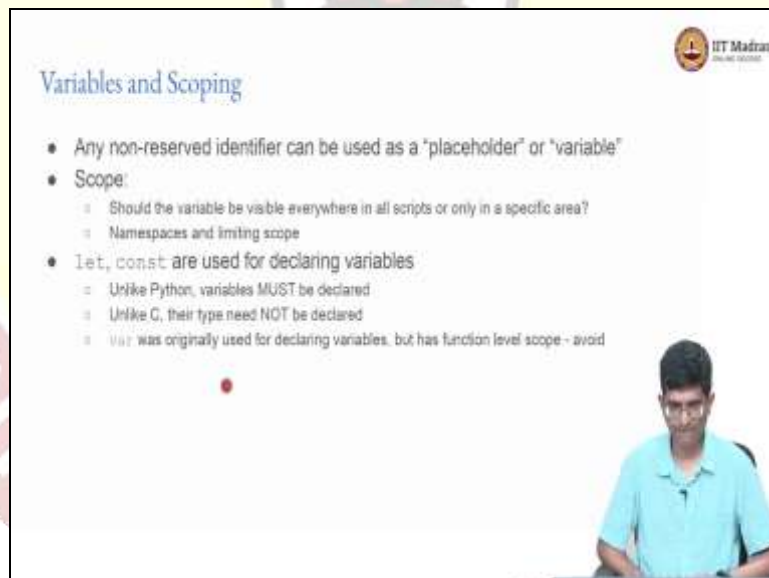


Operators and Comparisons

- Addition, subtraction etc.
 - Numbers, Strings
- Coercion
 - Convert to similar type where operation is defined
 - Can lead to problems - needs care
- Comparison:
 - Loose equality: `==` tries to coerce
 - Strict equality: `===` no coercion
- Important for iteration, conditions

Operators and comparisons as I mentioned right you have addition subtraction coefficient which can be problematic comparison preferable to use strict equality, right wherever possible. So, all of this is important when you are trying to implement iterations or conditions.

(Refer Slide Time: 02:06)

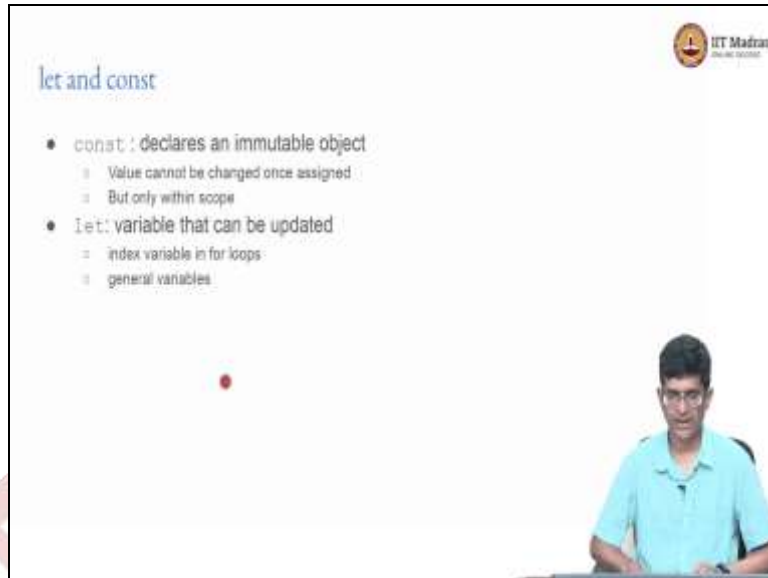


Variables and Scoping

- Any non-reserved identifier can be used as a "placeholder" or "variable"
- Scope:
 - Should the variable be visible everywhere in all scripts or only in a specific area?
 - Namespaces and limiting scope
- `let`, `const` are used for declaring variables
 - Unlike Python, variables MUST be declared
 - Unlike C, their type need NOT be declared
 - `var` was originally used for declaring variables, but has function level scope - avoid

This scoping of variables we already went through in the demo right, and what we saw there was any non-reserved identifier can be used as a variable name. Scope is something that we need to be careful about just. So, that we do not end up declaring a variable with the same name as something else in a library. So, whenever we declare a function or a block you can use `let` in order to have the variables inside it without going and messing up variables that have been declared elsewhere.

(Refer Slide Time: 02:37)



let and const

- `const`: declares an immutable object
 - Value cannot be changed once assigned
 - But only within scope
- `let`: variable that can be updated
 - index variable in for loops
 - general variables

So, let and const are the two mechanism used for declaring variables. As I said const declares what is called an immutable object the idea behind declaring something as constant is that whatever you have assigned into that variable cannot be changed. That's a bit tricky because what it actually means is let's say that you assign a object into a variable something inside the object can still be changed right, that is not restricted by const.

But the object itself cannot be reassigned or or the variable cannot be reassigned let on the other hand can. So, if you are implementing something like a for i equal to 0 to 10 loop i probably as we declared a let variable. So, that it can be updated each time you go through the loop.

(Refer Slide Time: 03:22)



Control Flow

- Conditional execution:
 - if, else
- Iteration
 - for, while
- Change in flow
 - break, continue
- Choice
 - switch

Now having done all of this it's important to also understand a little bit about control flow in Javascript.

(Video Start: 03:33)

And as part of that what we are going to do is you know we already looked at the basic idea of conditions right and an example of this would simply be to say ok you know I can declare a variable I can check if the value of the variable is equal to something else right. And based on which of these got executed right I would say of course 3 is not equal to 5.

On the other hand if I change the value of the variable out here and I run it again then it would print whatever came out in the first one which says basically what like it doesn't matter this is just a demo right. What about iteration this is an example of how we would write iteration in a language like C right and what would it do? It would ah we ran into a problem right I am trying to basically use const over here.

What does it mean I have declared x as a constant and I am trying to do x plus plus meaning that I am trying to change its value right this is where the trouble gets in. So, what do I need to do I can have let and that should then not be a problem. Right if I run this sure enough it prints out 0 1 2 3 4 right. So, x starts from 0 goes up to 4 exactly the way that you know in a C or Java like language this is what a for loop would look like.

But this business with const is not quite as straightforward as that supposing I have something like this const v is equal to 1, 2, 3, 4 and then I do for const x in v. Now look at this instead of let I am using a const over here okay and going by the previous argument it should say that you know x is changing its value each time through the iteration. So, how can I declare it as a const right but this works.

Right why does it work? Well because even though x is changing it is only changing from one iteration to the next right and within the scope of the function that is to say this open brackets close brackets the value of x is not being changed. On the other hand this x plus plus over here actually has to get run within the scope of this function that we have right it has to be incremented and then I need to check whether it is still within the bounds that I am working with.

Okay so, because of that this const over here when I use the const x in v works fine right where v is basically an array we have not yet looked at arrays we look at them later but you can broadly get an idea of what this is right it is just a collection of different values. Now the interesting thing here is you will notice that what got printed was not 1, 2, 3, 4 it was 0, 1, 2, 3. So, in other words what has happened is when I do this in the values that x is getting are not actually the values of v it is getting the index values.

Right so, instead if I print v of x then I would find that I should actually get the values 1, 2, 3, 4. sure enough that's what happens okay but there is another form of iteration right where instead of in I can use off what happens if I run this here the x will actually take the values of v. Okay so, x of v basically means the values of v right not just the values the indices that are present in v. It's a bit of a subtle difference you need to keep it in mind mostly off should be safe for your iterations.

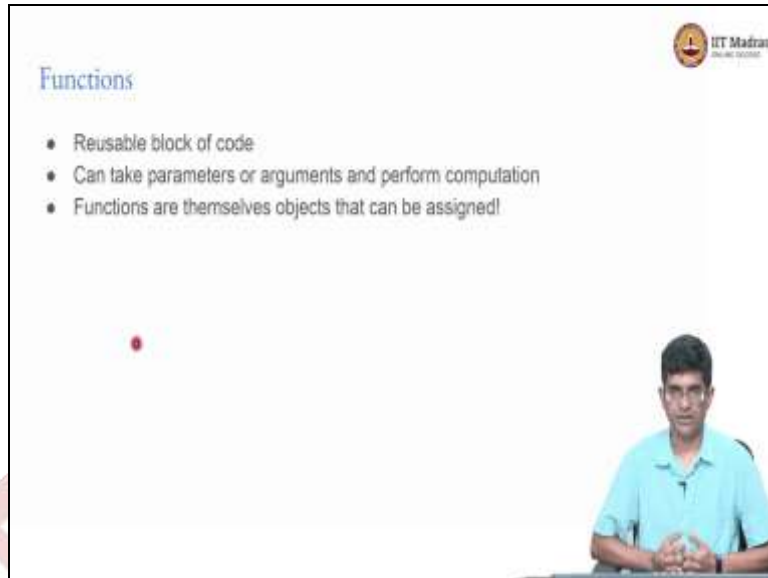
Right there are sort of subtle things that you need to keep in mind at some point but you can probably just keep it simple and off should work in most cases but it will give you the values and not the index right as you can see it runs and prints out 1, 2, 3, 4 okay.

(Video End: 07:39)

what else, so, we have looked at the basics of control flow there are a few other things you know there are while loops there are break and continue very similar to what's used in python right and also something called a switch statement which would be familiar to those of you who are coming from a C or Java background where you can basically compare against different options and instead of doing if else if else if else if you can just do a single switch statement.

I am not getting into the details of these at the moment I would recommend that you look at the language reference to see good examples of those as well.

(Refer Slide Time: 08:11)



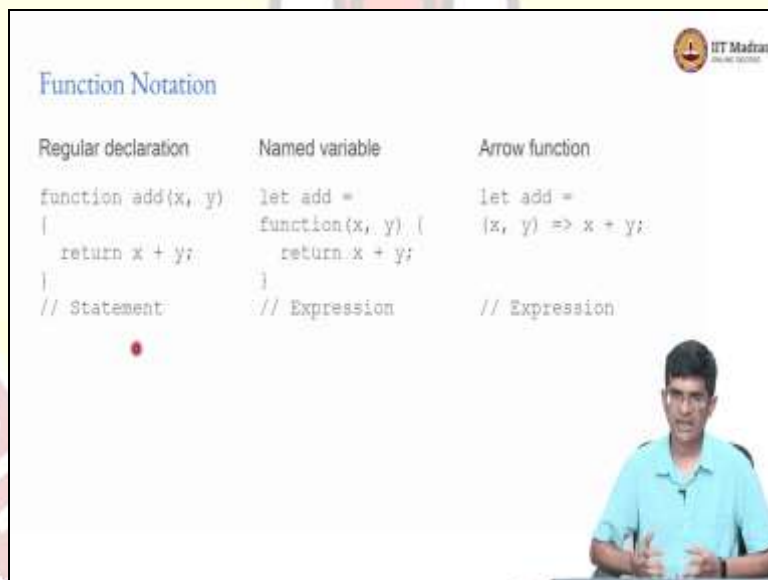
Functions

- Reusable block of code
- Can take parameters or arguments and perform computation
- Functions are themselves objects that can be assigned!

The slide features a small red dot on the left and a video of a man in a light blue shirt in the bottom right corner. The IIT Madras logo is in the top right.

Now what's a function right a function is essentially a reusable block of code right. It can take parameters or arguments and perform certain kinds of computations right and the interesting thing is functions are themselves objects that can be assigned okay.

(Refer Slide Time: 08:27)



Function Notation

Regular declaration	Named variable	Arrow function
<code>function add(x, y) { return x + y; }</code>	<code>let add = function(x, y) { return x + y; }</code>	<code>let add = (x, y) => x + y;</code>
// Statement	// Expression	// Expression

The slide features a small red dot on the left and a video of a man in a light blue shirt in the bottom right corner. The IIT Madras logo is in the top right.

There are multiple different notations that we can use for function the most straightforward one probably is what you know it this would be the equivalent of a def declaration in python. Function add of x y return x plus y this is just a statement that declares a function tells you what the functionality is and you know you can thereafter use add x y just like you would in any other programming language.

This on the in the middle column is something which is actually called a named expression right. If you look at the notation here I am saying let right and I am saying

add is equal to a function. Okay so, in other words I am taking this entire function `x y return x plus y` this statement out here is not actually a statement it's an expression in other words it has a return value. Notice the difference over here is directly function of `x, y` whereas over here it was function `add of x, y` that made it a statement which does not return a value.

Right whatever it returns that directly goes into `add` but over here when I just give function of `x, y` that's actually something called an anonymous function I am declaring the function without giving it a name okay and when I declare an anonymous function that it gives me back the content of the function which I can then assign into a variable out here okay. There is one more notation which is usually called the arrow notation over here for this is preferably used only for very simple kinds of functions.

Although unfortunately you will find that you know there are especially when you are looking at front end code and so on. You will find it being used quite a lot for fairly large and complex looking functions right. But ideally it's best suited for very small and brief functions something like this let `add` equals it takes two inputs implies or gives `x plus y`. Okay so, `add` basically is declared as a function that takes `x y` as input and gives `x plus y` as the output okay.

(Refer Slide Time: 10:35)



The slide is titled "Anonymous functions and IIFEs" and features the IIT Madras logo in the top right corner. It contains two code snippets: `let x = function () { return "hello" } // Anonymous bound` and `(function () { return "hello" })() // Declare and invoke`. Below the code, it asks "Why? Older JS versions did not have good scoping rules." and lists a bullet point: "• Avoid IIFEs in modern code - poor readability". A small inset video of a man in a blue shirt is visible in the bottom right corner of the slide.

Now this notation here right where I am basically saying function returned `hello` is something that's called an anonymous function. I am taking that anonymous function and binding it into a variable called `x` okay. This is a valid way of declaring functions

you can get any kind of function in this way. Now you could go one step further and basically say you know I declare this function I tell what the functionality is and then I immediately invoke the function right.

This parenthesis over here at the end is what is actually called invoking the function I am actually invoking it right there okay. If I did not have this open parenthesis open ah close parenthesis over here this starting with the function would sort of be taken as a statement and would not get evaluated but if I have this notation as seen over here what it is actually doing is it's declaring a function saying this is the functionality.

Immediately invoking it and moving on. So, what happened I cannot reuse this function because I never gave it a name I just declared it and invoked it. This is effectively just the same as creating a block right. I could have just used open parenthesis close parenthesis and you know whatever was in there would have got done I could have just you know returned hell well I cannot do a return unfortunately right.

But it would have sort of probably printed that value out. Right so, this kind of a thing is what's called an IIFE immediately invoked functioning expression or function execution right. Now as far as possible avoid them because they were required for older versions of javascript that did not have good scoping rules. They did not have let and const in particular right. And an IIFE was a very good way at that point of getting around that scoping because var still had function level scope okay.

So, because of that people tended to use IIFE's I think they were called IFFE's or something at that point but nowadays there is really no good excuse for using it right and it there are some people who feel that this makes it easy to read but for the most part especially for someone new to the language it just makes it more difficult to understand what's going on right. It's always better to explicitly declare a function and then call it okay.

(Video Time: 12:59)

So, let's look at some basic examples of functions right like I said I can declare a function. I am also going to use this built in function type of which takes in another identifier or a variable right and gives you some information about that identifier. Right

so, in this case I'm going to say what is the type of add and then also call add with the values 2 and 3 as input. Right so, what happens the first thing is I see that the type of is reported as function good.

So, it explicitly found that this is a function and of course when I called you know add it gave me the result nice to know okay. Now here's the interesting thing I can actually add something to that function right add dot v and this is something strange I mean this is we will get to this later but I am basically creating an object and assigning it as part of v okay and if I ask it okay what is add dot v or if even add dot v dot a right.

All of these things basically run through and I can see that it is going to print that out add dot v gets reported as this object add dot v dot a actually gets this value 3 and that gets printed out. Okay so, this is what I meant by functions can themselves have objects inside them right an object can have functions which are usually called methods but what does it mean if a function has an object inside it, it's not very clear.

The language allows it that doesn't mean there's necessarily a good reason to use it. Okay so probably best avoided, we can also use the yeah in fact we will just let me keep this the basic function declaration I am also going to use the arrow notation out here add one and you know let it also run directly as an anonymous function right where I do not give it a name I just have to give console dot log x y return x plus y and immediately invoke it out here 2 3 right.

So, what happens over here is line 93 basically declares a function says that the function should return x plus y immediately gives the function two parameters two and three evaluates it logs into console and is done. Okay so, this is an immediately invoked function expression right. Was there any need to do this well I could have probably just written x plus y in this case right 2 plus 3.

Right I do not really gain anything because it is not like I can reuse this add function I have already invoked it with the values 2, 2, 3 okay. There are certain scenarios where having this kind of IIFE helps but for the most part is probably easier and clearer to actually declare your functions explicitly right okay. Now similarly we can also have add

2 which is once again declared as a function and then I can look at all three of these add, add 1 and add 2 and I need to uncomment this.

And what we will find is that type of add type of add1 type of add2 they are all functions right even though they were declared in different ways one was using function another one was using the arrow notation another one was using a named bind to an anonymous function right all of them work the same way.

(Video End: 16:27)

