**JAVASCRIPT**

[Operator precedence - JavaScript | MDN (mozilla.org)](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_Precedence)

**Data types:**

1. **Number**
2. **String**
3. **Boolean**
4. **Undefined :**  var with empty value
5. **Null**
6. **Symbol(ES2015):** value that is unique
7. **BigInt(ES2020):** Larger Integers that the Number can hold.

**Javacript has dynamic typing**  we do not have to manually define data type of value stored in variable. Data type are determined auto.

**One error in typeof operator is that it shows type of null as object which is not correct and hence this is a error in javascript**.

**let / const/var :**

* **let** keyword used for variable can change in future.

mutate a variable mean reassigning some new value to already existed value of a variable.

let age = 30;

age = 31;

* we use **const** keyword for variable whose value won’t change in future. i.e; immutable variables which cannot be change.

We cannot declare empty const variables due to immutable.

* **var** is used before ES6, it is mutable

**Basic Operators:**

Assignment Operators : x+=10 // x = x+10 | x\*=2 // x=x\*2

**Template Literal:**  is used to write a string which has variable in it. It starts with backtick (` `)  
with variables being called by ${variable name} sign.

for eg: `I'm ${firstName}, a ${year - birthYear} year old ${job}!`;

**Type conversion :**  is when we manually convert data of one type to another type.

In JS if you want to convert string to number we use Number() function. But the original value doesn’t change.

Eg: const inputYear = ‘1991’;

**Console.log(Number(inputYear));** // this will convert string inputYear into number type but original value of variable inputYear wont be change.

**Console.log(Number(‘Jonas’));** // this will return Not a number (NaN) error because JS cannot convert produce number from the string.

To convert number to string we use String() method in JS.

**Type coercion :** when JavaScript automatically converts type behind the scenes for us.

* In JS if a string is concatenated with number via ‘+’ operator it triggers a type coercion and converts a number to string.

Eg:- console.log(“I am” + 23 + “years old.”) // this will convert 23 to string due to + operator used for concatenation of number and string.

* But for “-” operator its opposite i.e: string is converted into numbers.

Foe eg:- console.log(“23” – “10” -3); // output : 23-10 => 13 -3= 10

here string is concatenated with number via “- ”operator which will convert string into number and the final output will be 10

* For \* and / operator and also for comparison operators string is converted to number and that operation is performed;

Eg:- “23” \* “2” // output : 46 and same for dividation

“23” > “12” // output : true

**Diff b/w == & === :**

=== is strictly non type coercion eg: 18=== 18 // returns true

whereas == is type coercion eg : ‘18’ == 18 // returns true

**JS is backward compatible that is code written in 2020 JS will work in browser supporting older version of JS.**

**JS is not forward compatible that is code written in 2020 will not work in 2089 version of JS.**

**We use Babel to transpile and polyfill your code(converting it to ES5 to ensure browser compatibility for all user).**

* **Strict mode** : allows us to write a more secure JS code and to activate it we use “use strict” at starting of any script file

**Arrays Method:**

**Push():**  adds elements at end of array. It also returns the length of array

**Unshift():** adds elements at beginning of array.

**Pop():** it removes the last element of array. It returns the popped element, removed it from array.

**Shift()**: is used to remove element from the starting of array.

**indexOf():** method gives the location of element on array at any position.

**ES6 function-> includes():** which returns true if the element is present in the array. It uses strict equality for check.

**Objects:** Inside objects keys can be accessed via dot/bracket notation. Bracket for executing some expression and then access key and dot directly to access key of object.

(**GO through lecture or code example final script.js**)

**JavaScript in Browser: Dom and events fundamentals:**

To select an element on a html we use document.querySelector(“.className”) OR document.querySelector(“#idName”).

QuerySector is a method of document object.

To get the text content of that element just use document.querySelector(“.className”).textContent.

**DOM(Document Object Model):**  Dom is a structured representation of HTML documents allows javaScript to access HTML elements and styles to manipulate them.

**document**  is entry point to DOM eg:- document.querySelector

**DOM methods and properties of dom manipulation are not part of JAVASCRIPT.**

**Dom methods and its properties are part of Web APIs which can be accessed by JavaScript**

In order to handle events like (click) on button we use addEventListener() method in javascript which except 2 arguments one event triggered and second the function which needs to be executed on that triggered event.

Eg:- document.querySelector(‘.className’).addEventListener(‘click’, function(){

Some lines of code……

})

* In order to select multiple elements of same class we use **querySelectorAll.**
* To add and remove classes through JavaScript we use document.querySelector(‘className1).classList.add(‘classname2’) **or** classList.remove(‘classsName’)
* Each class acts like a container with multiple properties in it.
* Keyboards events are global events they do not happen on individual element but as a whole on document.
* After any event is added or triggered JavaScript created an object with details of that event , It can be accessed by passing the event argument while defining the function after listening to that event in addEventListener method.

**How JavaScript Works behind the Hood**

In JavaScript everything is a object

JavaScript is a

**high-level, :** where developer does not have to worry about managing resources like managing memory by creating variable because high level language have abstraction which takes care of it.

Downside is program will never be fast as compared to C language.

**prototype-based object-oriented, :** prototype based object oriented approach

**Eg:**

* Prototype example:

Array.prototype.push

Array.prototype.indexOf

Our arrays(which is object in JS) in JavaScript inherits method from prototype.

**multi-paradigm, :** an approach and mindset of structuring the code, which will direct your coding style and technique.

JavaScript can be used in all the paradigms that is :

* Procedural programming
* Object oriented programming
* Functional programming

**interpreted or just-in-time compiled ,**

**dynamic, :** dynamic typed, we don’t defined data type to variable it is already decided when value is passed into it.

Type of variable can be easily be change of we reassign the varaiable.

**single-threaded,**

**garbage-collected(cleans memory)** : is basically an algo inside JavaScript which removes all the unused object automatically

programming language

with **first-class functions:** functions are treated as a variables. we can pass functions into other functions, and we can even return functions from functions.

and a non-blocking event loop

**concurrency model**. : How JavaScript engine handles multiple tasks happening at the same time.

**Why we need that ?**

JavaScript itself runs in **single thread** which means it can only do one thing at a time.

**So what about the long -running tasks ?**

By using the **event loop**  takes long running task into background executes them and puts them back into the main thread once finished.

**And this is called JS non blocking event loop concurrency model with a single thread**.

**Event loop**

* **JS engine** program that runs JS code most popular is chrome V8

**Call Stack Heap**

**Execution Context** :

Environment in which a piece of JS is executed. It also stores all the necessary info required for some code to be executed

Objects in memory

**Where are code Is executed where objects are stored**

* **Modern JS**  uses JIT compilation(Just in Time) where entire code Is converted to machine code as once and executed immediately.
* **JS**  is not a interpreted language

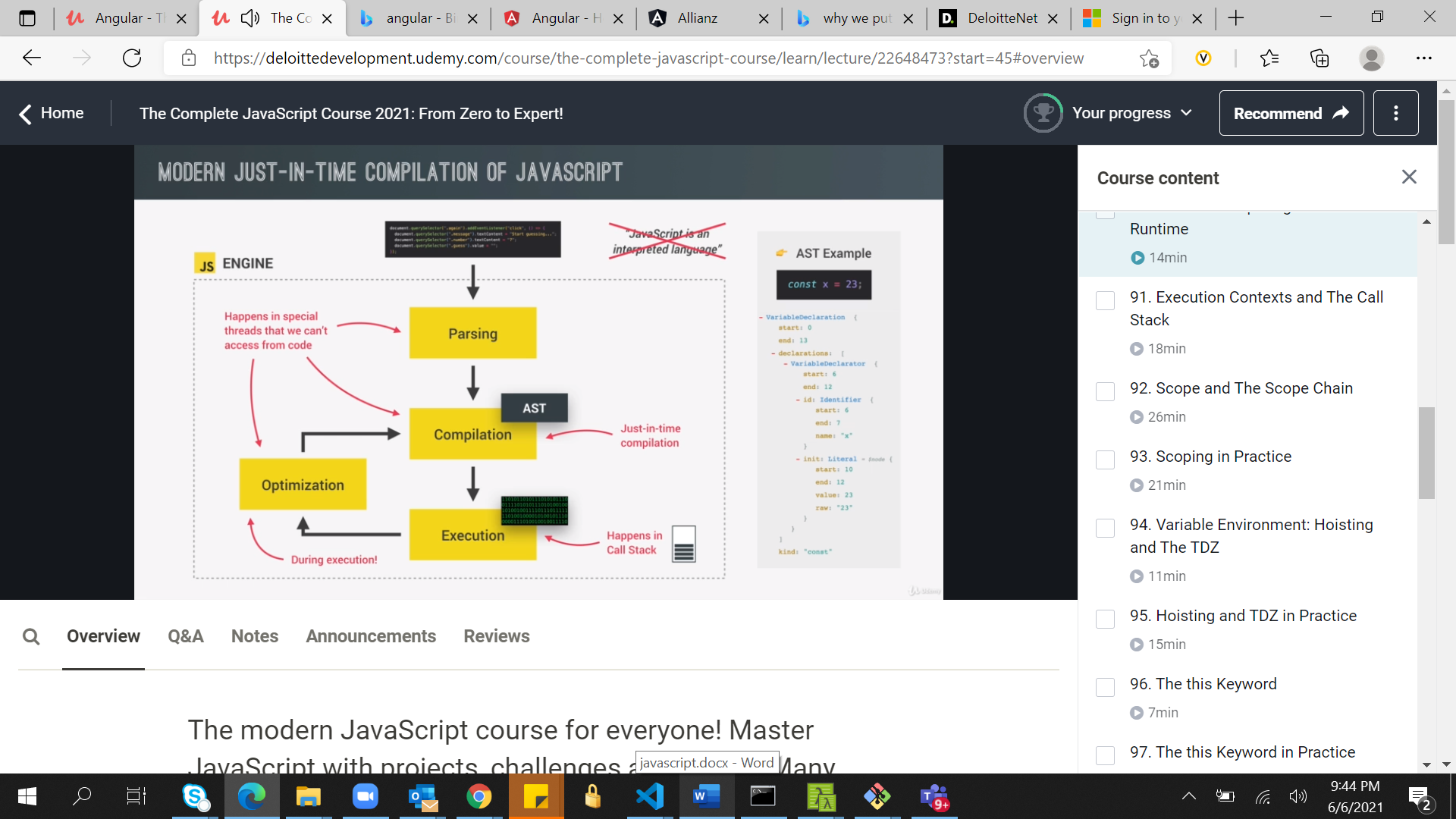
So as a piece of JavaScript code enters the engine the first step is to parse the code

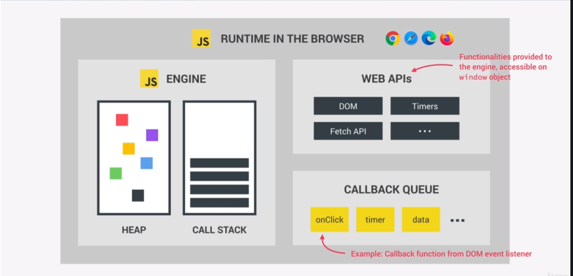
which essentially means to read the code. During the parsing process, the code is parsed into a data structure called the abstract syntax tree or AST.

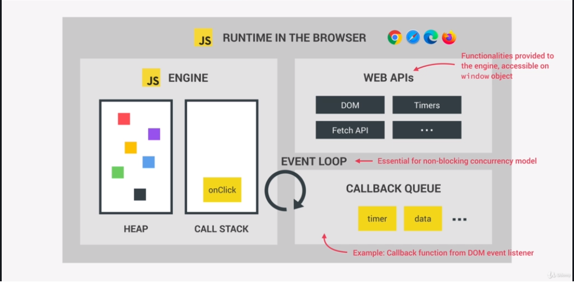
This works by first splitting up each line of code into pieces that are meaningful to the language like the const or function keywords, and then saving all these pieces

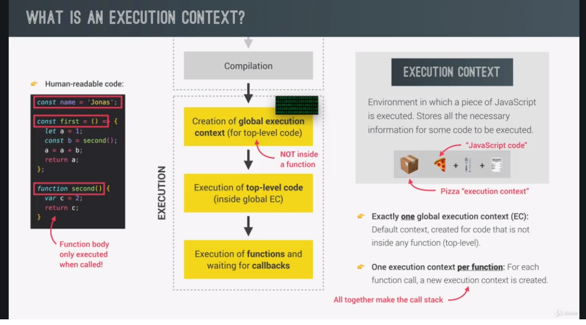
into the tree in a structured way. This step also checks if there are any syntax errors

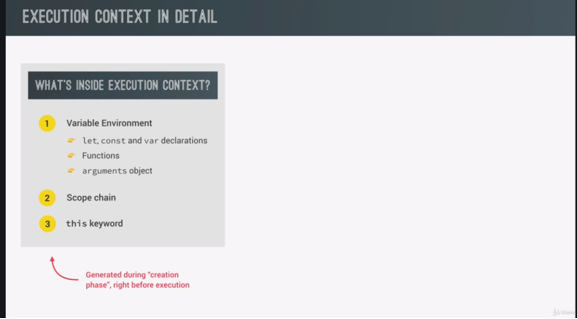
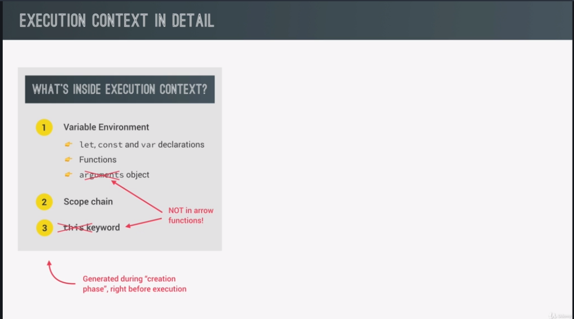
and the resulting tree will later be used to generate the machine code.

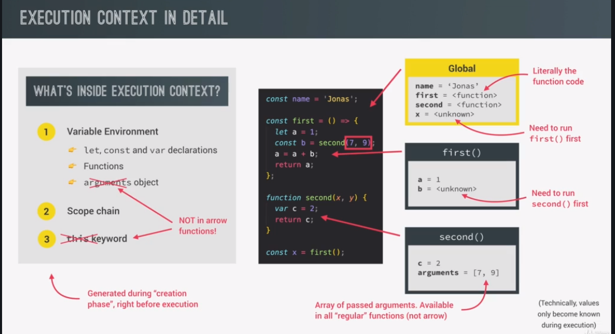


****

****

****

****

****

* **Arrow functions**  do not have **this** keyword and arguments object.

**Basically arrow functions** don't have the arguments object and the this keyword.

Instead, they can use the arguments object, and the this keyword from their closest regular function parent. And this is an extremely important detail to remember

**Scope Chain Concept:**

scoping controls how our program's variables are organized and accessed by the JavaScript engine.

So basically scoping asks the question, where do variables live? Or where can we access a certain variable and where not?

**Lexical Scoping :** lexical scoping means that the way variables are organized and accessed

is entirely controlled by the placement of functions and of blocks in the programs code.

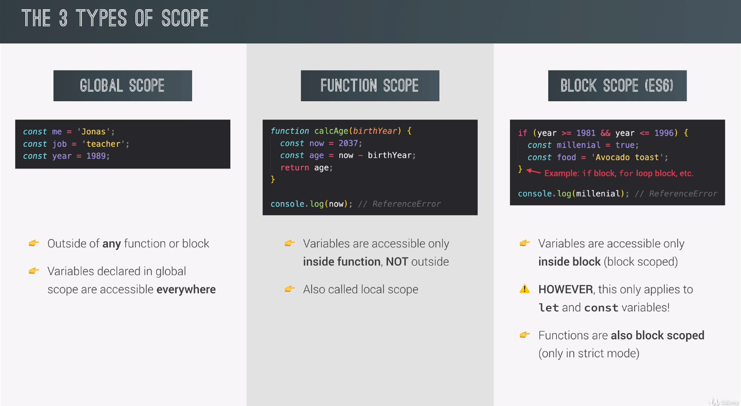
For example, a function that is written inside another function has access to the variables

of the parent function,

**Scope:** Scope is the space or environment in which a certain variable is declared.

In JS we have global scope , function scope and block scope.

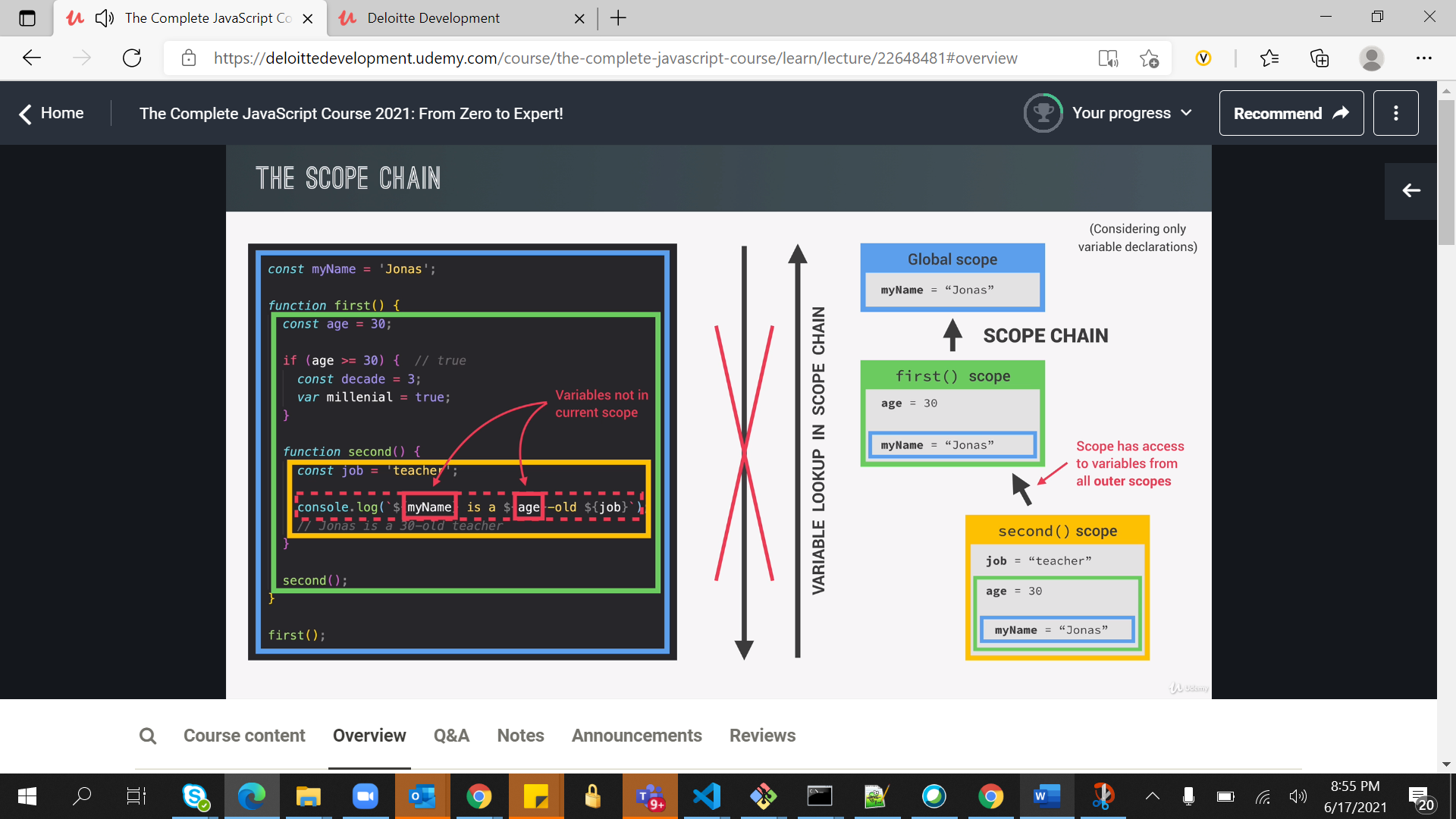
the scope of a variable is basically the entire region of our code, where a certain variable can be accessed.

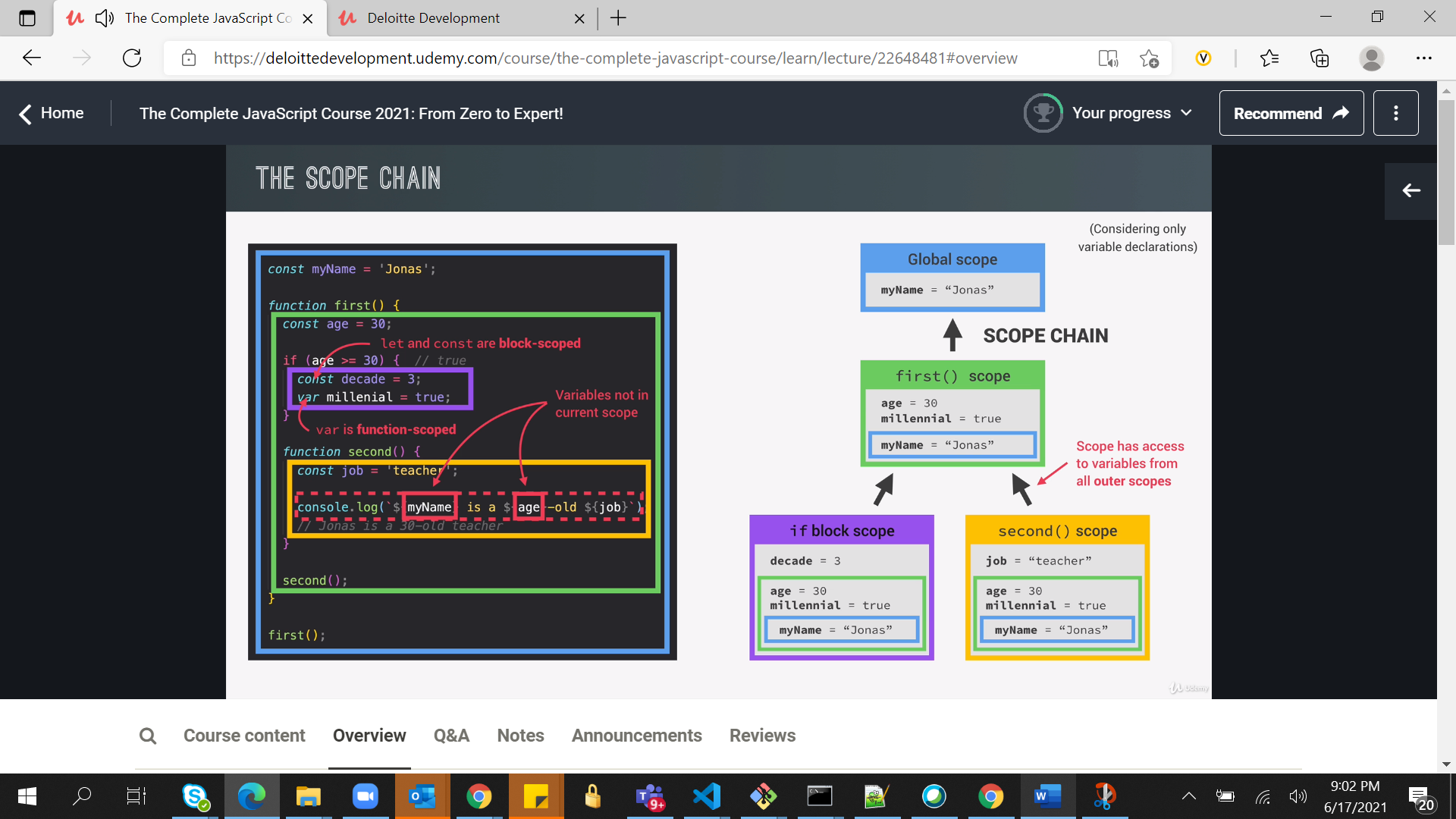


**In block scope only let and const variables applies.  
If we declare a variable with var keyword in block scope then if we can still access it outside block.**

**Var**  is a function scope.

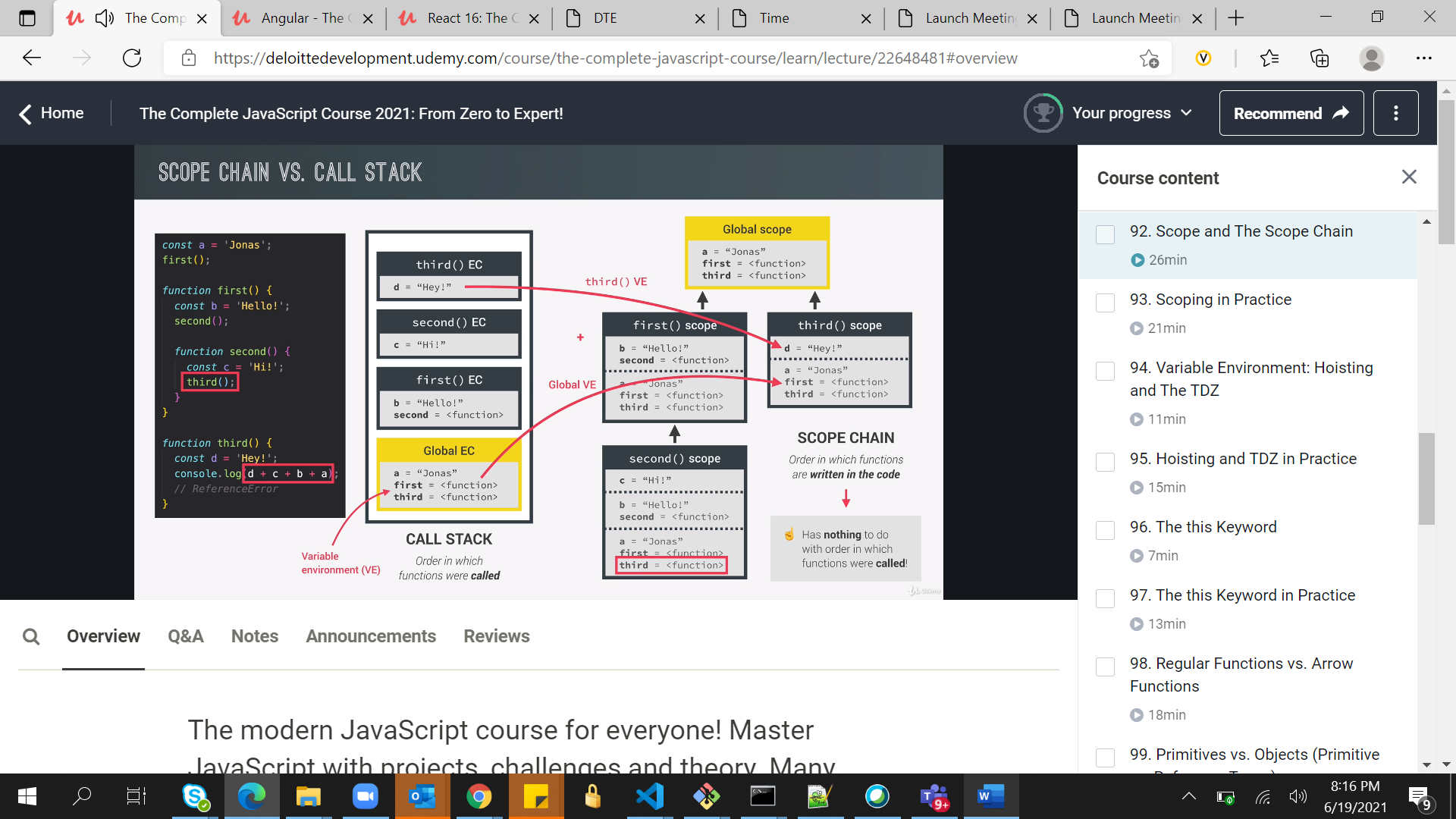
So to recap, let and const variables as well as functions are block scoped in ES 6 in stict mode.



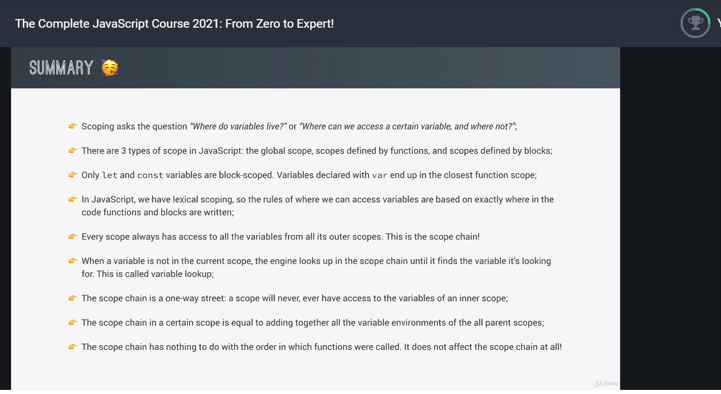


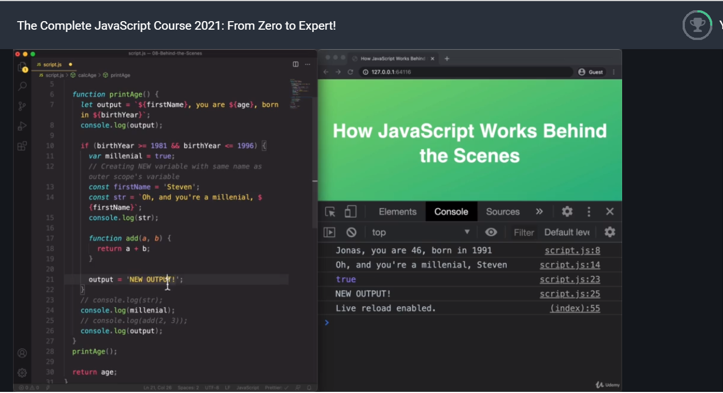
Only let , const are block scoped therefore variable declare using var keyword is not scoped to if block, but as it is function scoped and that means it will be available in function first() scope.

Scope chain has nothing to do with order in which functions were called.



In this variable c is not accessible inside third() even though it is called by second() because third() is called in global EC and even second() is calling it does not care about the order in which func are called.





**Primitives vs Objects Types:**

 it's a misconception that all variables declared with const are immutable.

In fact, that is only true for primitive values, but not for reference values.

In the Friend object, which was declared using a const here. And const is supposed to be for constants. So, for things that we cannot change. However, what actually needs to be constant is the value in the stack. And in this deck, the value only holds the reference,

which we are not actually changing. The only thing that we are changing is the underlying object that is stored in the heap. And that is okay to change, that has nothing to do with const or let, all right? That's only about the value in the stack, but if we change something in the heap

that has nothing to do with const or let.

