# Complete JavaScript Course

1. Basic Knowledge

* var variableNam = value;

Data types: Number, string, boolean, undefined (for variable does not have a value yet), null (non-existent)

If the variable is defined by “var” before, we don’t need to use var when we want to change its value

* JS has dynamic typing: data types are automatically assigned to variables
* console.log(…); -> print to the console
* alert(message) -> create an alert in the browser
* prompt(message) -> input something to store
* // comments -> single line comment

/\* multiple line comments \*/

* typeof variable -> data type of variable
* short version for if else statements:

var variableName = condition ? whatDoIfTrue : whatDoifFalse;

Exp: var drink = age >= 18 ? ‘beer’ : ‘juice’;

* Switch statement:

var varName = value;

switch (varName) {

case value1:

Todo if varName/value = value1;

break;

case value2:

case value3:

Todo if varName/value = value2 or value3;

break;

case valuen:

Todo if varName/value = valuen;

break;

default:

Todo if none of above cases is not fulfilled

}

* Truthy and Falsy values and equality operator:

Truthy values: Not falsy values

Falsy values: undefined, null, 0, ‘’, NaN

* == and === operators:

== is to compare datatypes

=== is to compare values

* **Array methods:**

ArrayName.push(value) -> add value at the end of the array

ArrayName.unshift(value) -> add value at the beginning of the array

ArrayName.pop() -> remove the last value of the array

ArrayName.shift() -> remove the first value of the array

ArrayName.indexOf(value) -> position/index of the value in the array (if value is not in the array, return -1)

* **Objects and properties** (= dictionary)

each value has its own name

var objectName = { name1 : value1, name2 : value2, name3 : value3,…};

objectName.name1 -> objectName[name1] -> call value assigned to name1

objectName.name2 = value -> objectName[name2] = value -> assign/change value to name2

var objectName2 = new Object(); -> create an empty object

* **Object methods:**

we can add function inside the object

objectName = { name1: function(var1) { ToDo and return value;},….}

objectName.name1(var1) -> add the value to name1 according to the inner function

if we want to use the value inside the object:

objectName = { name1 : value1, name2 : function() { return 2 – this.name1;}, name3 = value3} -> function in name2 will use the name1’s value when using “this”

* **Loops and iterations:**

for (var i = 0; i < n; i++) {TODO}

while (I < n) {TODO}

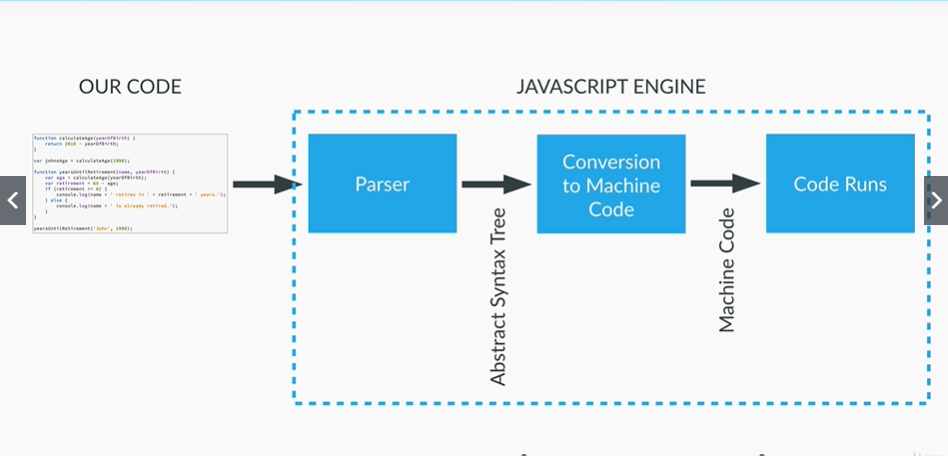
continue -> if the condition is true, continue will break the current run and go to the next implementation: for (…) { if (typeof var !== ‘string’) continue; TODO;} -> if type of var is string, the continue will break the statement and run to the next “I” in the loop.

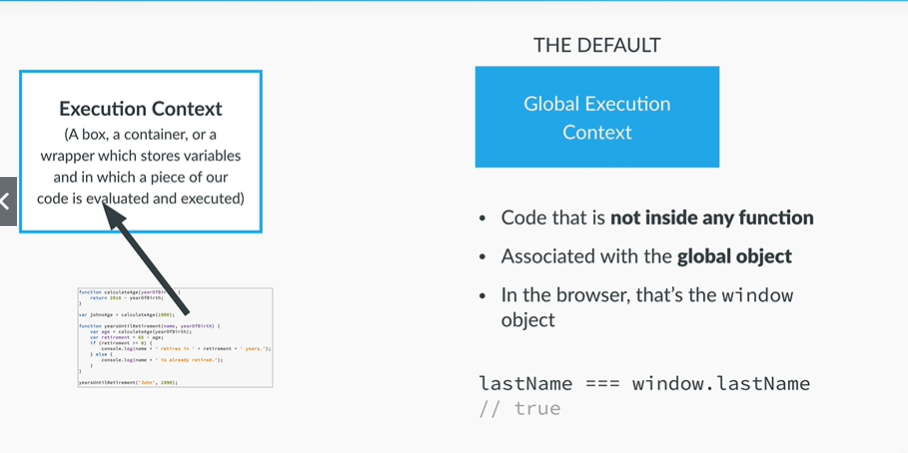
break -> if the condition is true, break breaks out of the entire loop

* **Function:**

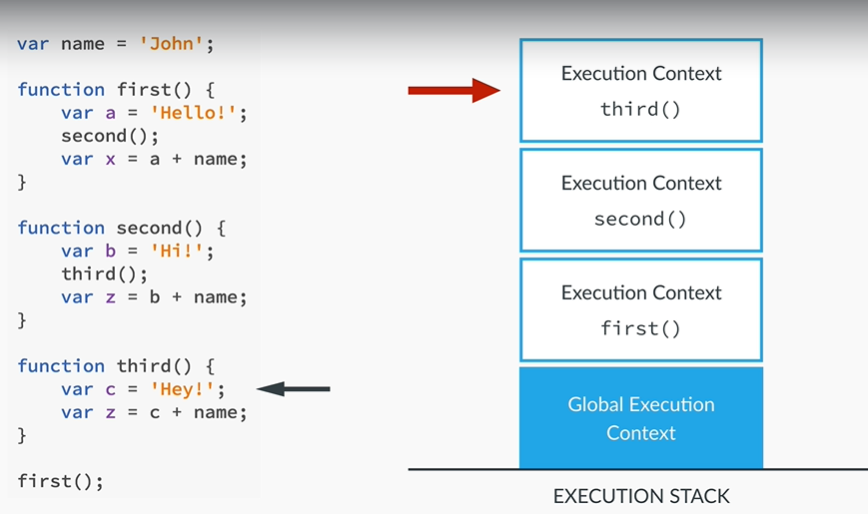
syntax: var funcName = function(var) {TODO} -> function funcName(var) {TODO}

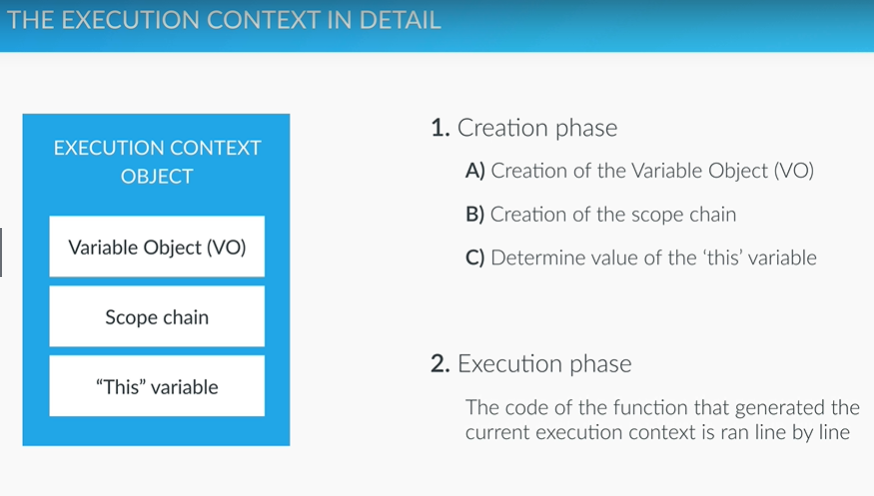
1. How JavaScript works behind the scenes





1. Execution Stack and Execution contexts:





* The argument object is created, containing all the arguments that were passed into the function
* Code is scanned for **function declarations**: for each function, a property is created in the Variable object, **pointing to the function**.
* Code is scanned for **variable declarations**: for each variable, a property is created in the Variable object, and set to undefined
* last 2 points called HOISTING:

**We can use the function even before we declare it**

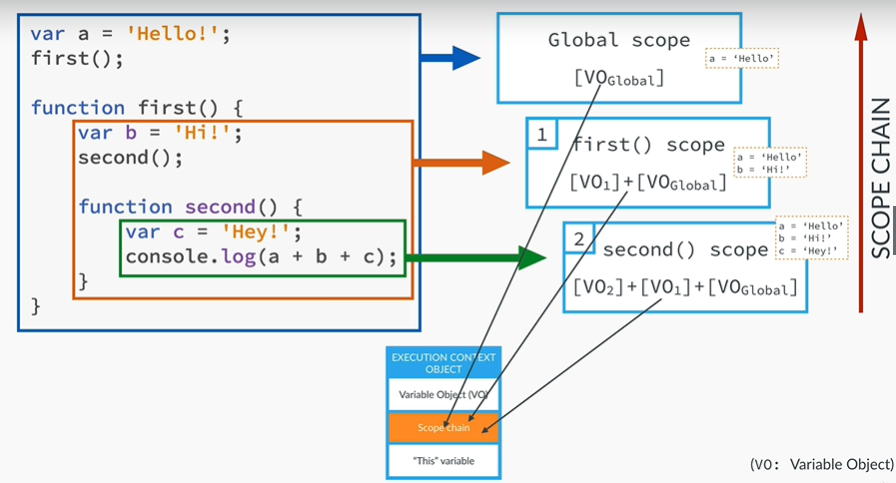
Exp: funcName(var); function funcName(var) {TODO} -> RUN OK

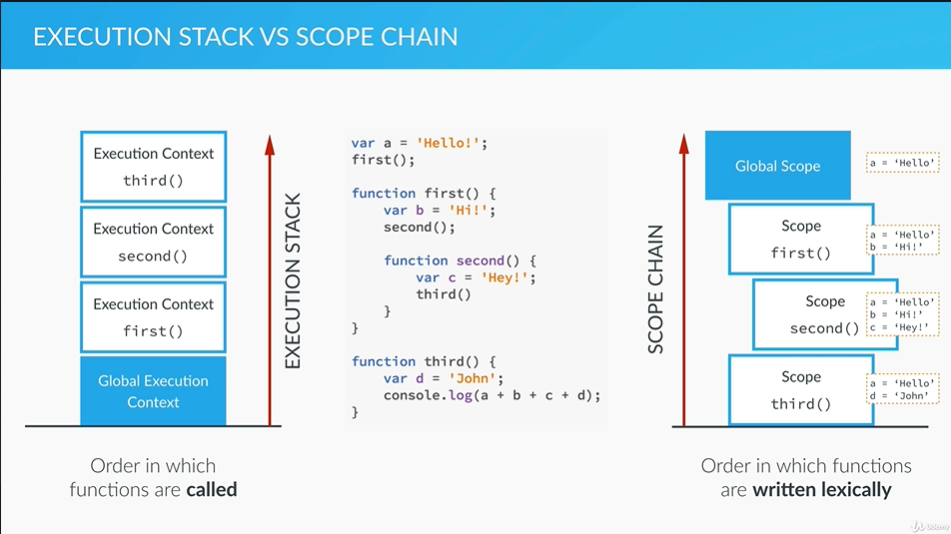
**But if we use function expression (var funcName = function(var) {TODO}, it will not work**

Exp: funcName(var); var funcName = function(var) {TODO} -> RUN ERROR

1. Scoping and scope chain in Javascript
   * Scoping:

* Scoping answers the question “where can we access a certain variable?”
* Each new function creates a scope: the space/environment, in which the variables it defines are accessible
* Lexical scoping: a function that is lexically within another function get access to the scope of the outer function





1. The ‘This variable’:

* Regular function call: the ‘this’ keyword points at the global object (the window object, in the browser).
* Method call: the variable points to the object that is calling the method
* the ‘this’ keyword is not assigned a value until a function where it is defined is actually called

example:

var john = {

name: ‘John’, yearOfBirth: 1999,

calculateAge: function() { console.log(this); console.log(2019 – this.yearOfBirth); }}

* when we call john.calculateAge(); -> the console.log(this) will call the john object. The console.log(2019 – this.yearOfBirth) will call the inside expression where this uses the value of yearOfBirth