

Dag 5

about:black

console.log(window)

window.innerHeight

Global execution context

window (equals to global object) = object waar alle browsereigenschappen in zitten

this (equals window)

global (equals global object)


Global Execution context →

When the JavaScript interpreter reads our code we enter the **creation phase** the **creation phase** defines the window (global object) and it defines the 'this' object (window).

The next thing that happens is, all of the variables are assigned a value of undefined. See below;

- window (global object)
- this (window)
- **course (undefined)**
- **week (undefined)**
- **getCourse (fn())**

```
1 var course = 'block tech';
2 var week = 1;
3 // function declaration
4 function getCourse() {
5   return {
6     course: course,
7     week: week
8   }
9 }
```



declaratie = variabele aanmaken

assignment = vullen van variabele

In creation phase worden alleen de declaraties van variabelen en functions gelezen (gehoist: naar boven gehaald), niet de assignments. Zo wordt er door JS ruimte gealloceerd voor de variabelen en functies.

De variabelen worden gevuld in de execution phase.

EXECUTION CONTEXT | HOISTING | SCOPES | CLOSURES

Global Execution context

So the steps that take place in the **creation phase** are

- creation of global object
- creation of this
- set up memory space for variables and functions
- assign variables a default value of undefined and place functions directly in memory

This is the end of the **creation phase** we now enter the **execution phase** Here JavaScript starts reading our code line by line, after our code is being run, our variables are assigned there value.

Now that we know all about the global execution context we have a little quiz. What would the following code output?

```
1 console.log(course);
2 console.log(week);
3 console.log(getCourse);
4
5 var course = 'block tech';
6 var week = 1;
7 // function declaration
8 function getCourse() {
9   return {
10     course: course,
11     week: week
12   }
13 }
```

- undefined
- undefined
- function getCourse()

The variables are assigned the value of undefined. And the console.log(getCourse) simply references our in memory stored function.

want??

function:

maakt een arguments object ipv een global object

scope

de huidige context van de execution

hoisting

variabelen en function declarations fysiek verschuiven naar de bovenkant van de code (onderdeel van creation phase)

```
console.log(num);
```

```
var num;
```

```
num = 6;
```

➔ num = undefined, want var wordt gehooft, daarna console.log en daarna num vullen

closure

zorgt ervoor dat de inner function toegang heeft tot de outer function, ook als de execution context van de outer function al is gebeurd

(functie binnen functie (outer & inner function))

global scope vs local scope

global scope: declared once accessible anywhere

local scope: alleen binnen het block waarin t gedeclareerd is

function scope vs block scope

function scope: variables inside the function

block scope: een bepaald block is relevant voor die conditie, bijv. if of for loop statements

let mag je niet opnieuw declareren, maar **var** wel