



# **JavaScript 2**

## **Lektion => 8**

Utbildare: Mahmud Al Hakim

# **NACKADEMIN**

# Lektionstillfällets mål

## Mål med lektionen

- Exekveringskontext
- Scope
- Closure
- Att läsa: sid. 452-486
- **Arbetsmetod**
  - Teori och praktik varvas under lektionen

**NACKADEMIN**

# Kort summering av föregående lektion

- Vi har gått igenom
  - **Introduktion till AngularJS**

**NACKADEMIN**

# Exekveringskontext (Execution Context)

- Varje sats i JavaScript lever i en ett av dessa tre exekveringskontext.

Global  
Context

Function  
Context

Eval  
Context

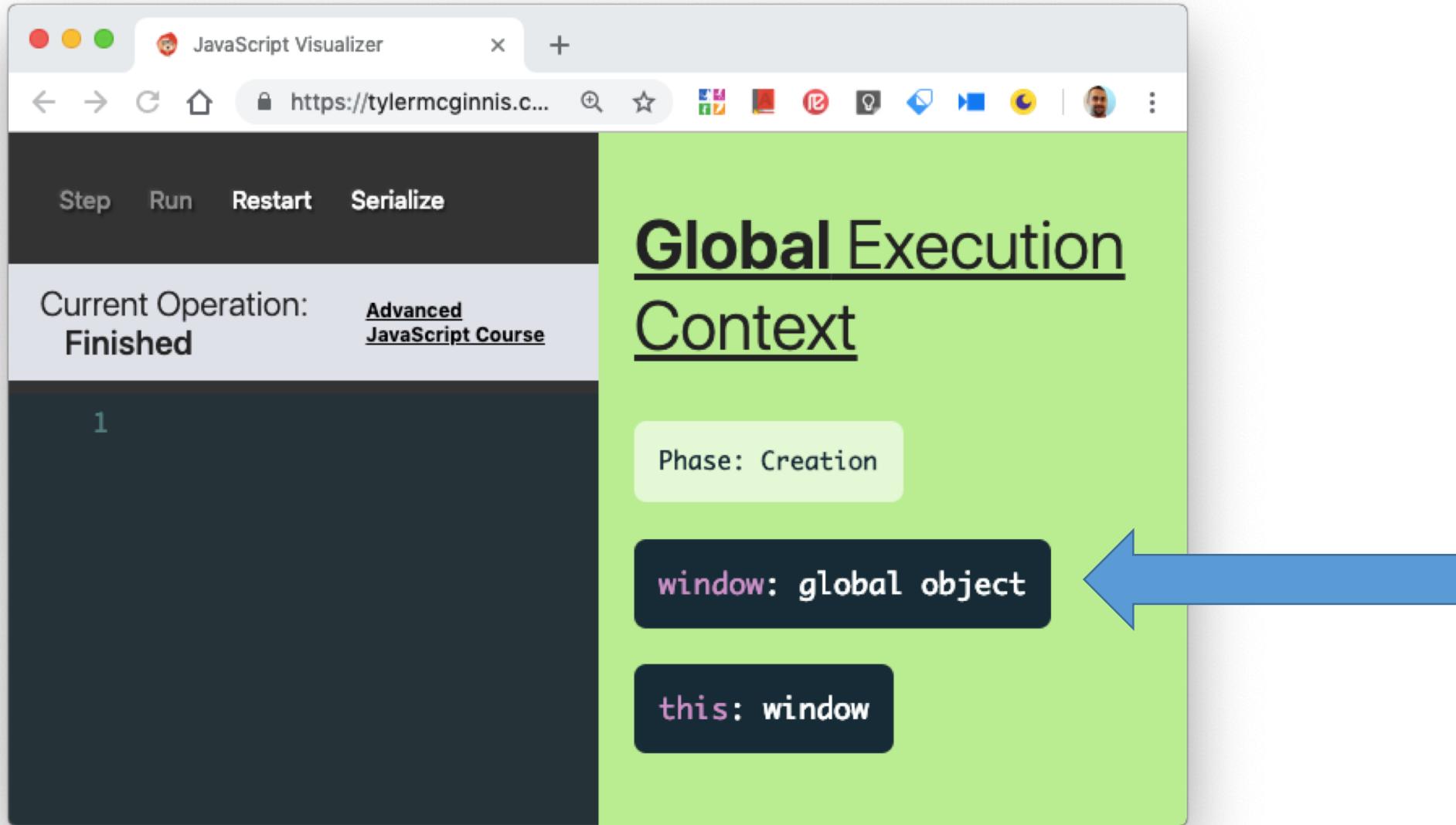
# Javascript Visualizer

<https://tylermcginnis.com/javascript-visualizer>

The screenshot shows the JavaScript Visualizer tool. On the left, there's a dark sidebar with buttons for "Step", "Run", "Restart", and "Serialize". A "Run Speed" slider is set to "Fast". Below it, the "Current Operation" is set to "Advanced JavaScript Course". The main area has a large number "1" at the top left. To the right, the title "JavaScript Visualizer" is displayed in large letters, followed by a subtitle: "A tool for visualizing **Execution Context**, **Hoisting**, **Closures**, and **Scopes** in JavaScript". Below the title are social media sharing icons for Twitter, Facebook, and LinkedIn. At the bottom, instructions are listed:

1. Type **(ES5)** JavaScript in the editor
2. "Step" or "Run" through the code
3. Visualize how your code is interpreted

# Global Execution Context



# Creation Phase – Variabler

The screenshot shows the JavaScript Visualizer interface. On the left, there's a sidebar with controls: 'Step Run' (disabled), 'Restart', 'Serialize', and a 'Run Speed' slider set to 'Fast'. Below that, it says 'Current Operation: Program' and 'Advanced JavaScript Course'. A code editor window contains the line `1 var a = 1;`. On the right, the main area is titled 'Global Execution Context' and shows the 'Phase: Creation'. It lists three variables: 'window: global object', 'this: window', and 'a: undefined'. A blue arrow points from the 'a: undefined' entry towards the bottom of the slide.

Step Run

Restart

Serialize

Run Speed

Slow Fast Faster

Current Operation:  
Program

Advanced JavaScript  
Course

```
1 var a = 1;
```

**Global Execution Context**

**Phase: Creation**

window: global object

this: window

a: undefined

# Execution Phase – Variabler

The screenshot shows the JavaScript Visualizer interface. On the left, there's a sidebar with controls: 'Step' (Run), 'Restart', and 'Serialize'. Below that, it says 'Current Operation: Advanced JavaScript Course' and 'Finished'. A code editor window contains the line `1 var a = 1;`. On the right, a pink panel titled 'Global Execution Context' displays three variable bindings: `window: global object`, `this: window`, and `a: 1`. A blue arrow points from the 'a: 1' binding towards the 'a' variable in the code editor.

Step Run

Restart

Serialize

Run Speed

Slow Fast Faster

Current Operation: Advanced JavaScript  
Course

Finished

```
1 var a = 1;
```

Global Execution Context

Phase: Execution

window: global object

this: window

a: 1

# Creation Phase – Funktioner

The screenshot shows the JavaScript Visualizer interface. On the left, the code being run is:

```
1 var a = 1;
2
3 demo();
4
5 function demo () {
6     var a = 2;
7 }
8
```

The title bar says "Advanced JavaScript Course". The toolbar includes "Step" (which is selected), "Run", "Restart", and "Serialize". A "Run Speed" slider is set to "Fast". The status bar says "Current Operation: Program".

To the right, under the heading "Global Execution Context", the state variables are listed:

- Phase: Creation
- window: global object
- this: window
- a: undefined
- demo: fn()

A large blue arrow points from the "Global Execution Context" area back towards the code editor.

# Function Context – Creation Phase

The screenshot shows the JavaScript Visualizer tool interface. On the left, the code being executed is:

```
1 var a = 1;
2
3 demo();
4
5 function demo () {
6     var a = 2;
7 }
```

The interface has two main sections: the Global Execution Context (purple background) and the demo Execution Context (yellow background). A blue arrow points from the Global context down to the demo context.

**Global Execution Context**

- Phase: Execution
- window: global object
- this: window
- a: undefined
- demo: fn()

**demo Execution Context**

- Phase: Creation
- arguments: { length: 0 }
- this: window
- a: undefined

# Function Context – Execution Phase

The screenshot illustrates the execution phase of a JavaScript function context using the [Advanced JavaScript Course](https://tylermcginnis.com) visualizer.

**Global Execution Context:**

- Phase: Execution
- window: global object
- this: window
- a: 2
- demo: fn()

**demo Execution Context:**

- Phase: Execution
- arguments: { length: 0 }
- this: window
- a: 2

A large blue arrow points from the Global Execution Context down to the demo Execution Context, indicating the scope chain or parent-child relationship between the two contexts.

```
1 var a = 1;
2
3 demo();
4
5 function demo () {
6     var a = 2;
7 }
8
```

# Eval Context

- eval() är en funktion som finns i objektet window.
- Funktionen eval() evaluerar JavaScript-kod.

```
console.log("2+2");           // 2+2  
console.log(eval("2+2")); // 4
```

# Do not ever use eval!

`eval()` is a dangerous function, which executes the code it's passed with the privileges of the caller. If you run `eval()` with a string that could be affected by a malicious party, you may end up running malicious code on the user's machine with the permissions of your webpage / extension. More importantly, a third-party code can see the scope in which `eval()` was invoked, which can lead to possible attacks in ways to which the similar `Function` is not susceptible.

[https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\\_Objects/eval#Do not ever use eval](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval#Do_not_ever_use_eval)

# Övning

Förklara exekveringskontexten för nedanstående kod

```
console.log(window);
console.log(this);
console.log(window === this);

function demo() {
  console.log(window);
  console.log(this);
  console.log(window === this);
}

demo();
```

# Hoisting (Hissa/Lyfta upp)

- Under "Creation Phase" flyttas alla **deklarationer** högst upp, alltså, både deklarationer av variabler och funktioner.
- Detta kallas **Hoisting** i JavaScript.

```
console.log(x); // undefined (Orsakar buggar)
var x = 5;      // OBS! var
console.log(x); // 5
```

# Hoisting – Demo

The screenshot shows a browser window titled "JavaScript Visualizer" with the URL "https://tylermcginnis.com". The interface includes a toolbar with "Step", "Run", "Restart", and "Serialize" buttons, and a status bar indicating "Current Operation: Program" and "Advanced JavaScript Course". The code editor contains the following script:

```
1 console.log(x);
2 var x = 5;
3 |
```

To the right, a yellow panel titled "Global Execution Context" displays the state of variables:

- Phase: Creation
- window: global object
- this: window
- x: undefined

A large blue arrow points from the "x: undefined" entry towards the code editor.

För att lösa problemet med variabel-hoisting  
Använd alltid **let** eller **const** istället för **var**

```
console.log(x);
```

✖ ➔ Uncaught ReferenceError: x is not defined

```
let x = 5;
```

# Övning 1

Förklara exekveringskontexten för nedanstående kod

```
var a = 1;  
console.log(a);  
demo();  
  
function demo() {  
    console.log(a);  
    var a = 2; // OBS var  
    console.log(a);  
}
```

## Övning 2

Förklara exekveringskontexten för nedanstående kod

```
let a = 1;  
console.log(a);  
demo();  
  
function demo() {  
    console.log(a);  
    let a = 2; // OBS let  
    console.log(a);  
}
```

# Function Context – Creation Phase

## Objektet arguments

```
function add(a,b){  
    console.log(arguments);  
    return a+b;  
}  
add(1,2);
```

```
▼ Arguments(2) [1, 2, callee  
  0: 1  
  1: 2  
  ► callee: f add(a,b)  
  length: 2  
  ► Symbol(Symbol.iterator):  
  ► __proto__: Object
```

Step Run Restart Serialize

Run Speed  
Slow Fast Faster

Current Operation: BlockStatement

Advanced JavaScript Course

```
1 function add(a,b){  
2   console.log(arguments);  
3   return a+b;  
4 }  
5 add(1,2);
```

## Global Execution Context

Phase: Execution

window: global object

this: window

add: fn()

## add Execution Context

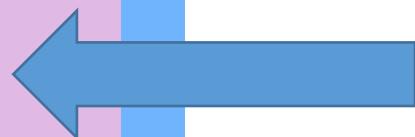
Phase: Creation

arguments: { 0: 1, 1: 2, length: 2 }

this: window

a: 1

b: 2



# this i ett objekt refererar till det aktuella objektet

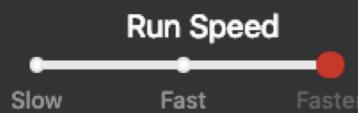
```
let course = {  
    title: "JavaScript",  
    info: function() {  
        console.log(this);  
        console.log(this.title);  
    }  
};  
course.info();
```



```
▼ {title: "JavaScript", info: f}  
  ► info: f ()  
  title: "JavaScript"  
  ► __proto__: Object  
JavaScript
```

Step

Run Restart Serialize



Current Operation: ExpressionStatement

### Advanced JavaScript Course

```
1 var course = {  
2   title: "JavaScript",  
3   info: function() {  
4     console.log(this);  
5   }  
6 };  
7 course.info();
```

## Global Execution Context

Phase: Execution

window: global object

this: window

course: { title: "JavaScript", info:  
fn() }

## anonymous Execution Context

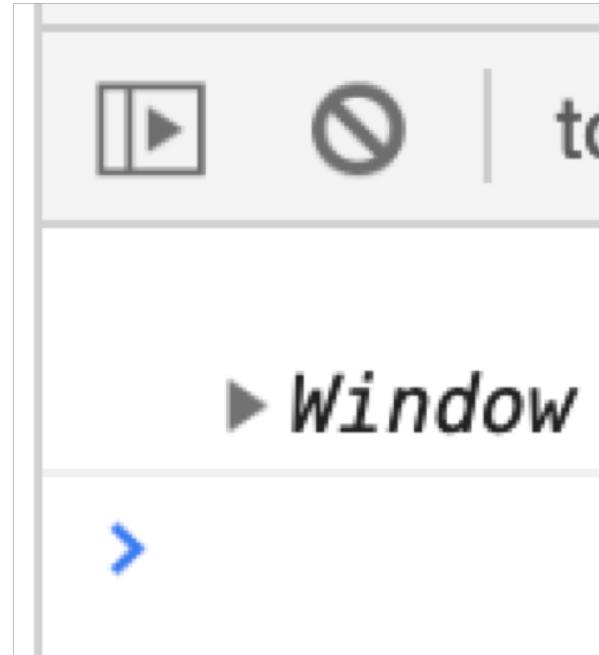
Phase: Execution

arguments: { length: 0 }

this: { title: "JavaScript", info:  
fn() }

**this** i en funktion som inte tillhör ett objekt refererar till det globala objektet (window)

```
let course = {  
    title: "JavaScript",  
    info: function() {  
        function demo (){  
            console.log(this);  }  
        demo();  
    }  
};  
course.info();
```



Step Run Restart Serialize

Run Speed  
Slow Fast Faster

Current Operation: BlockStatement

Advanced JavaScript Course

```
1 var course = {  
2   title: "JavaScript",  
3   info: function() {  
4     function demo (){  
5       console.log(this); // window  
6     }  
7     demo();  
8   }  
9 };  
10 course.info();
```

this: window

course: { title: "JavaScript", info: fn() }

anonymous Execution Context

Phase: Execution

arguments: { length: 0 }

this: { title: "JavaScript", info: fn() }

demo: fn()

demo Execution Context

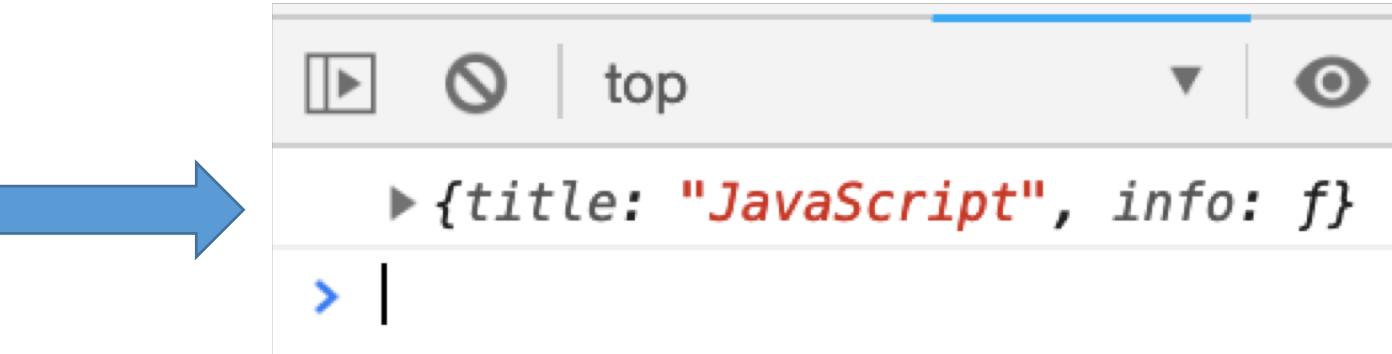
Phase: Creation

arguments: { length: 0 }

this: window

Lösning 1 på föregående problem  
Skapa en referens till det aktuella objektet

```
let course = {  
    title: "JavaScript",  
    info: function() {  
        let self = this;  
        function demo (){  
            console.log(self);  
        }  
        demo();  
    }  
};  
course.info();
```



Lösning 2 på föregående problem  
Arrow-funktioner saknar **this**

```
let course = {  
    title: "JavaScript",  
    info: function() {  
        demo = () => console.log(this);  
        demo();  
    }  
};  
course.info();
```

Övning – Vad får du i console och varför?  
Hur löser du ett sådant problem?

```
let course = {  
    title: "JavaScript",  
    students : ["Mahmud", "Kalle", "Erik"],  
    info: function() {  
        this.students.forEach(function (s) {  
            console.log(s + ' läser ' + this.title);  
        });  
    }  
};  
course.info();
```

# Scope (*räckvidd*)

- Scope innebär den synlighet och levnadstid som variabler har.
- Varje exekveringskontext har ett eget objekt som lagrar alla variabler, funktioner och parametrar.
- Varje exekveringskontext har även tillgång till föräldrarnas objekt.

# Global Scope

- Variabler som definierats utanför en funktion är globala.
- Globala variabler är tillgängliga i alla funktioner!
- Dessa variabler läggs till window-objektet som egenskaper.

# Global Scope – Demo 1

```
var mahmud = "Mahmud";
console.log(mahmud + ' är en global variabel');

function demo() {
    console.log(mahmud + ' finns i demo()');

    function inner() {
        console.log(mahmud + ' finns i inner()');
    }
    inner();
}
demo();
```

# Global Scope – Demo 2

```
var mahmud = "Mahmud";  
  
function demo() {  
    console.log(mahmud);  
    console.log(window.mahmud);  
    console.log(this.mahmud);  
}  
  
demo();
```

# Function Scope & Lexical Scope

- Variabler som deklareras inne i en funktion är lokala.
- Lokala variabler inne i en funktion är inte synliga utanför funktionen.
- Funktioner i JavaScript har något som kallas "**Lexical Scope**".
- Lexical Scope innebär att en funktion är länkad till objektet som har definierat funktionen.
- En funktion har alltså tillgång till sina egna lokala variabler och även till alla variabler som finns i objektet och även till alla objekt ovanför!

# Function Scope – Demo 1

```
function demo() {  
    let mahmud = "Mahmud";  
    console.log(mahmud + ' inne i demo');  
}  
demo();  
  
console.log(mahmud + ' utanför demo');
```

✖ ➔ Uncaught ReferenceError: **mahmud** is not defined

# Function Scope – Demo 2

```
let mahmud = "Al Hakim";  
  
function demo() {  
    let mahmud = "Mahmud"; // Samma variabelnamn  
    console.log(mahmud + ' inne i demo');  
}  
demo();  
  
console.log(mahmud + ' utanför demo');
```

# Övning – Inga felmeddelanden! Varför?

```
function demo() {  
    mahmud = "Mahmud";  
    console.log(mahmud + ' inne i demo');  
}  
demo();  
  
console.log(mahmud + ' utanför demo');
```

Step Pause Restart Serialize

Run Speed  
Slow Fast Faster

Current Operation: Program

Advanced JavaScript Course

```
1 function demo() {  
2   mahmud = "Mahmud";  
3   // OBS! Ingen var eller let  
4   console.log(mahmud + ' inne i demo');  
5 }  
6 demo();  
7  
8 console.log(mahmud + ' utanför demo');  
9 |
```

## Global Execution Context

Phase: Execution

window: global object

this: window

demo: fn()

mahmud: "Mahmud"

# Block Scope

- Variabler som deklarerar med `let` eller `const` är lokala inne i blocket.
- OBS! `var` har inte Block-Scope!

# Block Scope – Demo 1

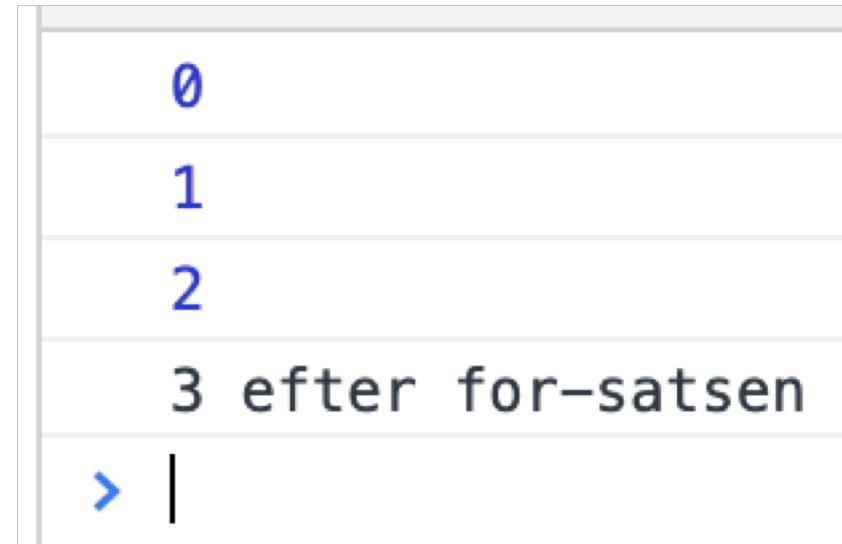
```
function demo() {  
    let mahmud = "Mahmud";  
    console.log(mahmud + ' inne i demo');  
    {  
        let mahmud = "Al Hakim"; // Samma variabelnamn  
        console.log(mahmud + ' inne i ett block');  
    }  
    console.log(mahmud + ' efter blocket');  
}  
demo();
```

# Block Scope – Demo 2

```
function demo() {  
    var mahmud = "Mahmud";  
    console.log(mahmud + ' inne i demo');  
    {  
        var mahmud = "Al Hakim"; // OBS! ORSAKAR PROBLEM  
        console.log(mahmud + ' inne i ett block');  
    }  
    console.log(mahmud + ' efter blocket');  
}  
demo();
```

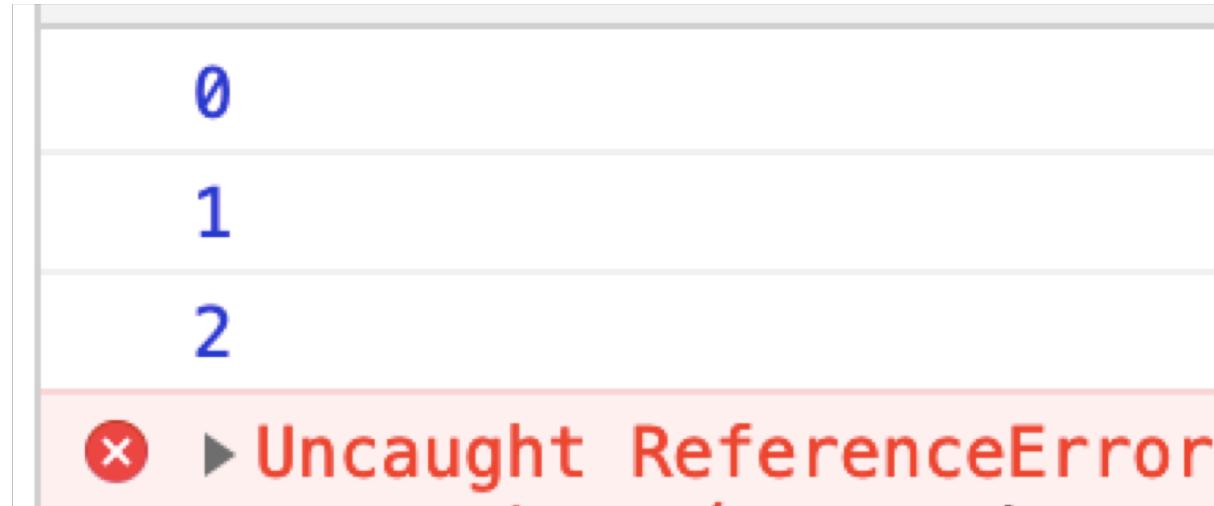
## Block Scope – Demo 3

```
function demo() {  
    for (var index = 0; index < 3; index++) {  
        console.log(index);  
    }  
    console.log(index + ' efter for-satsen');  
}  
demo();
```



## Block Scope – Demo 4

```
function demo() {  
    for (let index = 0; index < 3; index++) {  
        console.log(index);  
    }  
    console.log(index + ' efter for-satsen');  
}  
demo();
```



# Closures

- När du returnerar en funktion inuti en yttre funktion skapas automatiskt en **closure**.
- Den inre funktionen har tillgång till den yttre funktionens variabler även fast den yttre funktionen har körts klart.

# Closures – Demo 1

```
function outer() {  
    let a = 1;  
    return function inner() {  
        console.log(a);  
    };  
}  
let inner = outer();  
inner();
```

# Closure Scope

The screenshot shows the JavaScript Visualizer interface with the title "Execution Context Demo". The "Run Speed" slider is set to "Fast". The current operation is "Program". The code being run is:

```
1 function outer() {  
2     var a = 1;  
3     return function inner() {  
4         console.log(a);  
5     };  
6 }  
7 var inner = outer();  
8 inner();|
```

**Global Execution Context**

Phase: Execution

window: global object

this: window

outer: fn()

inner: fn()

**Closure Scope**

arguments: { length: 0 }

this: window

a: 1

# Closures – Demo 2

```
function outer() {  
    let a = 1;  
    return function (b) {  
        a = a+b;  
        console.log(a);  
    };  
}  
let inner = outer();  
inner(1);  
inner(2);
```

Step Run Restart Serialize

Run Speed  
Slow Fast Faster

Current Operation: ExpressionStatement

Advanced JavaScript Course

```
1 function outer() {  
2     var a = 1;  
3     return function (b) {  
4         a = a+b;  
5         console.log(a);  
6     };  
7 }  
8 var inner = outer();  
9 inner(1);  
10 inner(2);
```

Tracer: Tracer

## Closure Scope

arguments: { length: 0 }

this: window

a: 4

## anonymous Execution Context

Phase: Execution

arguments: { 0: 2, length: 1 }

this: window

b: 2

# Immediately-invoked function expression IIFE (Självanropande funktioner)

```
(function () {  
    console.log("Jag är en IIFE");  
})();
```

- Tips. Bra att läsa:
- <http://benalman.com/news/2010/11/immediately-invoked-function-expression/>

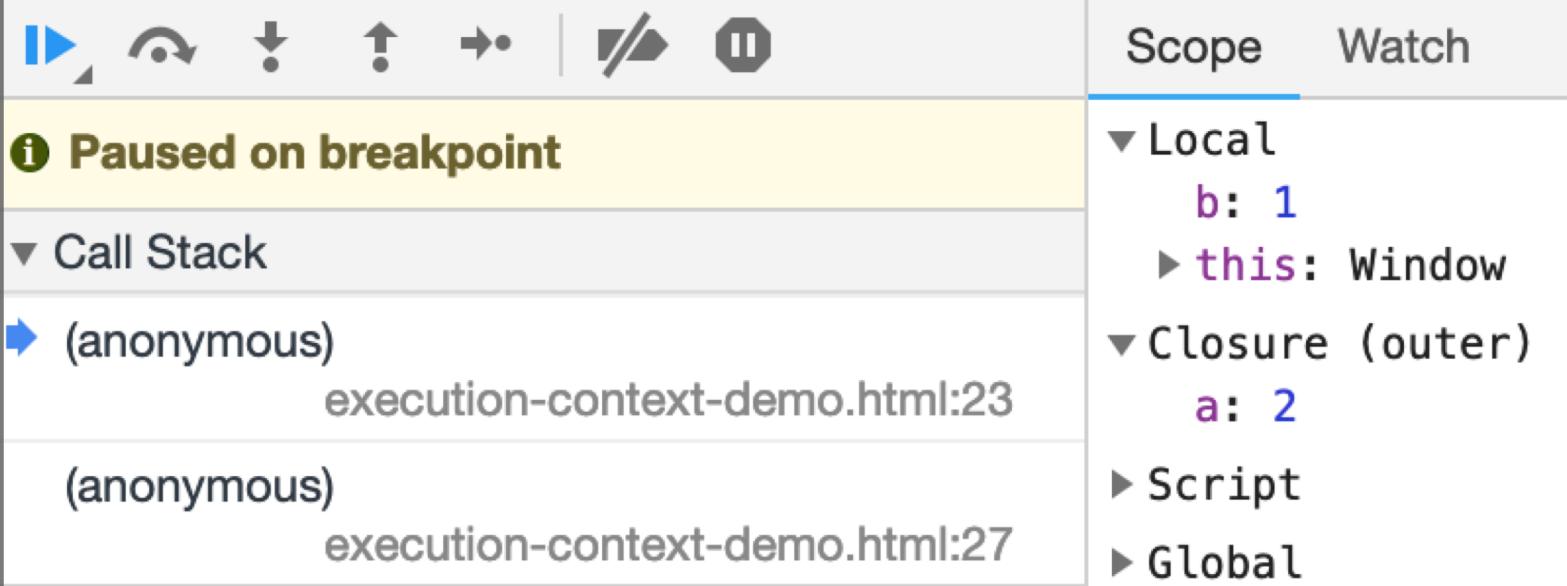
# En closure i en IIFE

```
let inner = (function () {
  let a = 1;
  return function (b) {
    a = a+b;
    console.log(a);
  };
})();
inner(1);
inner(2);
```

# Call Stack och Scope i Chrome DevTools

```
execution-context-demo.html >
19 function outer() {
20     let a = 1;
21     return function (b) {
22         a = a+b;
23         console.log(a);
24     };
25 }
26 let inner = outer();
27 inner(1);
28 inner(2);
```

{ } Line 23, Column 24



# Skapa ett JavaScript-bibliotek

```
let myLib = (function() {
    // Privata variabler
    const info = "myLib.js är ett demo-bibliotek";
    const version = "0.1";

    // Ett objekt som innehåller publika metoder
    let myLib = {
        random: function(min, max) {
            return Math.floor(Math.random() * (max + 1 - min)) + min;
        }
    };
    return myLib;
})();
```

# OBS! Biblioteket finns i en closure

The screenshot shows a browser window with the URL <https://tylermcginnis.com>. The page title is "Advanced JavaScript Course". On the left, there's a code editor with the following JavaScript code:

```
1 /*jshint esversion: 6 */
2 let myLib = (function() {
3     // Privata variabler
4     const info = "myLib.js är ett demo-
      bibliotek";
5     const version = "0.1";
6
7     // Ett objekt som innehåller publika
    metoder
8     let myLib = {
9         random: function(min, max) {
10             return Math.floor(Math.random() * (max
+ 1 - min) + min);
11         }
12     };
13     return myLib;
14 })();
15
16
```

At the top of the code editor, there are buttons for "Step", "Pause" (which is selected), "Restart", and "Serialize". To the right is a "Run Speed" slider set to "Faster". Below the code editor, it says "Current Operation: Program".

The main content area has a light orange background. It displays two sections: "Global Execution Context" and "Closure Scope".

**Global Execution Context**

- Phase: Execution
- window: global object
- this: window
- myLib: { random: fn() }

**Closure Scope**

- arguments: { length: 0 }
- this: window
- myLib: { random: fn() }

# Minifiera biblioteket med UglifyJS

<https://skalman.github.io/UglifyJS-online>

The screenshot shows a web browser window titled "UglifyJS 3: Online JavaScript minifier". The URL in the address bar is <https://skalman.github.io/UglifyJS-online>. The page content includes the title, a code editor on the left containing the original JavaScript code, and a results section on the right showing the minified output.

**UglifyJS 3: Online JavaScript minifier**

```
let myLib = (function() {
  // Privata variabler
  const info = "myLib.js är ett demo-bibliotek";
  const version = "0.1";
  // Ett objekt som innehåller publika metoder
  let myLib = {
    random: function(min, max) {
      return Math.floor(Math.random() * (max + 1 - min) +
min);
    }
  };
  return myLib;
})();
```

The minified output (77 bytes, saved 77.15%)

```
let myLib={random:function(n,o){return Math.floor(Math.random()*(o+1-n)+n)};}
```

Options  As I type **Minify**

# Använd JavaScript-biblioteket – demo

```
<h1>myLib demo</h1>
<button id="btn">Generera ett nummer mellan 1 och 6</button>
<h2 id="random"></h2>

<script src="myLib.js"></script>
<script>
document.getElementById('btn').addEventListener('click' , function(){
document.getElementById('random').innerHTML = myLib.random(1, 6)
})
</script>
```

# Övning

- Skapa ditt eget JavaScript-bibliotek.
- Lägg in några metoder som du oftast använder i dina applikationer.
- Dokumentera allting och skapa en GitHub Repo till biblioteket.
- Dela ditt bibliotek med dina klasskamrater.
- Lycka till.

# The Ultimate Guide to Execution Contexts, Hoisting, Scopes, and Closures in JavaScript

[https://youtu.be/Nt-qa\\_LIUH0](https://youtu.be/Nt-qa_LIUH0)



# Summering av dagens lektion

- Exekveringskontext
- Scope
- Closure
- Att läsa: sid. 452-486
- Reflektioner kring dagens lektion?

**NACKADEMIN**

# Framåtblick inför nästa lektion

- Under nästa lektion kommer vi att jobba med
  - JavaScripts objekt
  - Prototyper och prototypkedja
  - Klasser
- Bra att läsa  
<https://javascript.info/classes>

NACKADEMIN