

# JavaScript Beginner Course Part2

jan.schulz@devugees.org

# Agenda

1. What happens to our code?
2. Execution contexts and the Execution Stack
3. Hoisting
4. Scoping
5. Scoping VS Execution Stacks
6. this - Variable



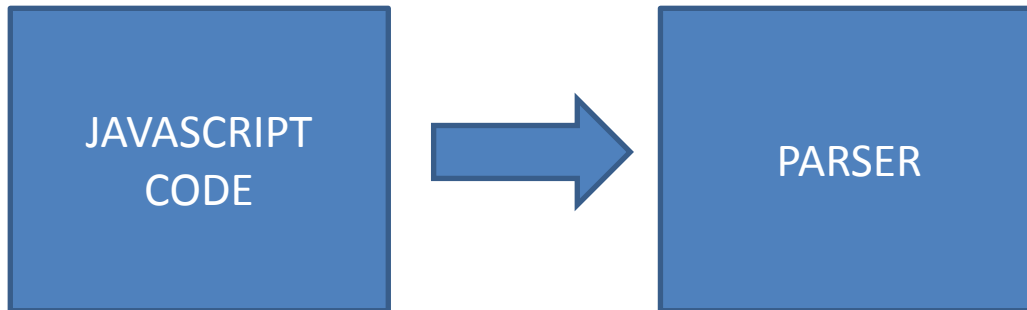
**BEHIND  
THE SCENES**

# 1. What happens to our code?

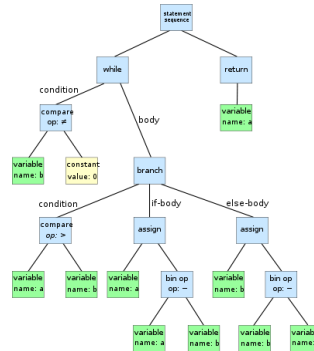
JAVASCRIPT  
CODE

```
1 function quickSort(items, left, right) {  
2   var index = 0;  
3   if (items.length > 1) {  
4     left = typeof left != 'number' ? 0 : left;  
5     right = typeof right != 'number' ? items.length - 1 : right;  
6     index = partition(items, left, right);  
7     if (left < index - 1) {  
8       quickSort(items, left, index - 1);  
9     }  
10    if (index < right) {  
11      quickSort(items, index, right);  
12    }  
13  }  
14  return items;  
15 }  
16 // first call  
17 var result = quickSort(items);
```

# 1. What happens to our code?



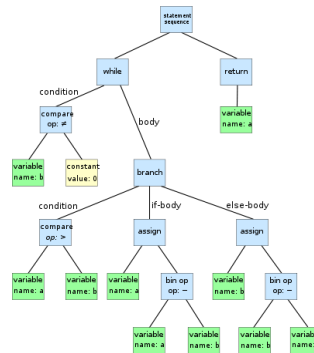
```
1 function quickSort(items, left, right) {  
2   var index = 0;  
3   if (items.length > 1) {  
4     left = typeof left != 'number' ? 0 : left;  
5     right = typeof right != 'number' ? items.length - 1 : right;  
6     index = partition(items, left, right);  
7     if (left < index - 1) {  
8       quickSort(items, left, index - 1);  
9     }  
10    if (index < right) {  
11      quickSort(items, index, right);  
12    }  
13  }  
14  return items;  
15 }  
16 // first call  
17 var result = quickSort(items);  
18
```



# 1.What happens to our code?



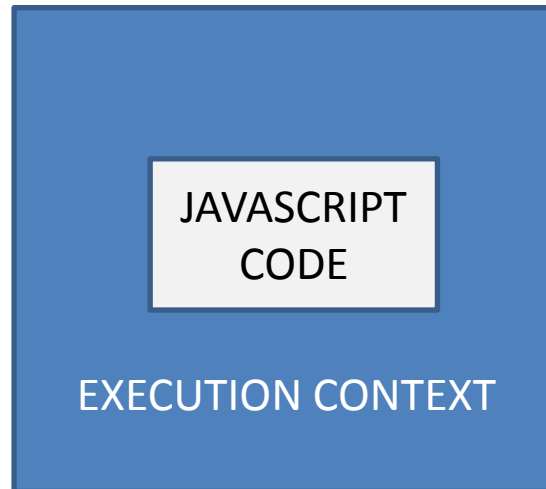
```
1 function quickSort(items, left, right) {  
2   var index = 0;  
3   if (items.length > 1) {  
4     left = typeof left != 'number' ? 0 : left;  
5     right = typeof right != 'number' ? items.length - 1 : right;  
6     index = partition(items, left, right);  
7     if (left < index - 1) {  
8       quickSort(items, left, index - 1);  
9     }  
10    if (index < right) {  
11      quickSort(items, index, right);  
12    }  
13  }  
14  return items;  
15 }  
16  
17 // first call  
18 var result = quickSort(items);
```



```
b8 00 b8 8e c0 d6 20 03 a8 fd 01 bf a2 00 b9 .....6  
02 00 eb 2b b4 06 b2 ff cd 21 3c 71 0f 84 a5 01 .....+...!sq...  
3c 50 b9 a0 00 74 18 3c 48 b9 a0 00 0f 84 d9 00 .....<P...t.<K...  
b9 02 00 3c 4d 74 08 3c 4b 0f 84 cc 00 eb d5 89 .....<Mt.<K...  
3e b5 09 01 cf 89 3e b5 09 a8 87 01 8b 3e b5 09 .....>...>...>...  
b0 20 26 88 05 26 88 45 fe 26 88 85 62 ff 26 88 .....<..4.E..4..b..6  
85 60 ff 26 88 85 5e ff 26 88 85 9e 00 b0 07 26 .....<..6..*..6...<6  
88 45 01 8b 3e b3 09 89 fb 83 eb 02 d1 fb 8a 00 .....E..>...  
26 88 45 fe 89 fb 81 eb a2 00 d1 fb 8a 00 26 88 .....4.E...<6..  
85 5e ff 89 fb 81 eb a0 00 d1 fb 8a 00 26 88 85 .....>...>...>...  
60 ff 89 fb 81 eb 9e 00 d1 fb 8a 00 26 88 85 62 .....<..6...<6..b  
ff 89 fb 81 eb a2 00 d1 fb 8a 00 26 88 85 5e ff .....<..6...<6..>...  
89 fb 83 c3 02 d1 fb 8a 00 26 88 45 02 89 fb 81 .....<..6.E...<6..  
c3 9e 00 d1 fb 8a 00 26 88 85 9e 00 89 fb 81 c3 .....<..6...<6..>...  
a0 00 d1 fb 8a 00 26 88 85 a0 00 89 fb 81 c3 a2 .....<..6...<6..>...  
00 d1 fb 8a 00 26 88 85 a2 00 b0 08 26 88 05 a0 .....<..6.E...<6..>...  
87 09 26 88 45 01 e5 0b ff 89 3e b5 09 29 cf 89 .....<..6...<6..>...  
3e b3 09 e8 bd 00 8b 3e b5 09 b0 20 26 88 05 26 .....<..6...<6..>...  
88 45 02 26 88 85 9e 00 26 88 85 a0 00 26 88 85 .....<..b...<6.E..>...  
a2 00 26 88 85 62 ff b0 07 26 88 45 01 8b 3e b3 .....<..6...<6.E..>...  
09 89 fb 83 eb 02 d1 fb 8a 00 26 88 45 fe 89 fb .....<..6...<6.E..>...  
81 eb a2 00 d1 fb 8a 00 26 88 85 5e ff 89 fb 81 .....<..6...<6...>...  
eb a0 00 d1 fb 8a 00 26 88 85 60 ff 89 fb 81 eb .....<..6...<6...>...  
9e 00 d1 fb 8a 00 26 88 85 62 ff 89 fb 81 eb a2 .....<..6...<6...>...  
00 d1 fb 8a 00 26 88 85 5e ff 89 fb 83 c3 02 d1 .....<..6...<6...>...
```

## 2. Execution contexts and Execution Stack

Execution Context is a Box/ Container/ Wrapper



## 2. Execution contexts and execution stack

```
var name = 'John';  
  
function first() {  
  var greeting = 'Hello! '  
  var x = greeting + name;  
  
  second();  
  
  console.log(x);  
}  
  
function second() {  
  var greeting = 'Hi! '  
  var x = greeting + name;  
  
  third();  
  
  console.log(x);  
}  
  
function third() {  
  var greeting = 'Hey! '  
  var x = greeting + name;  
  
  console.log(x);  
}  
  
first();
```



Global Execution Context

Execution Stack



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Global Execution Context

Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Global Execution Context

Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Global Execution Context

Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

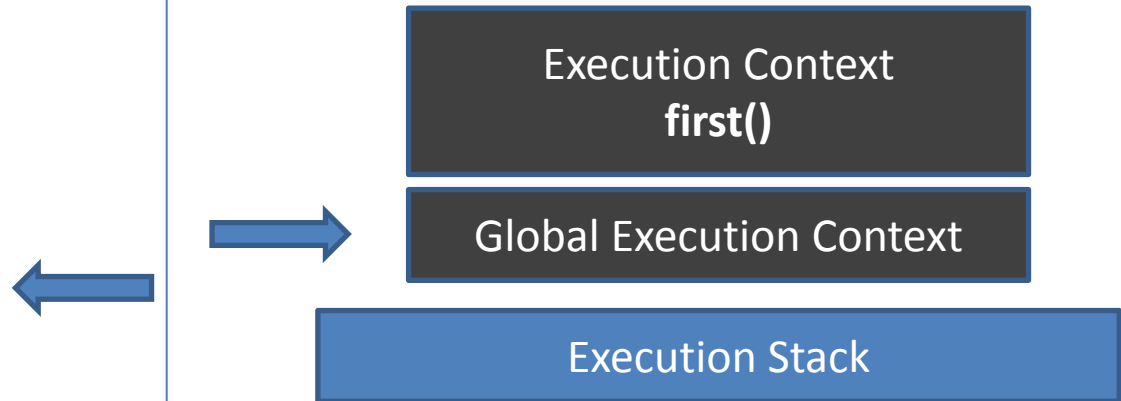
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

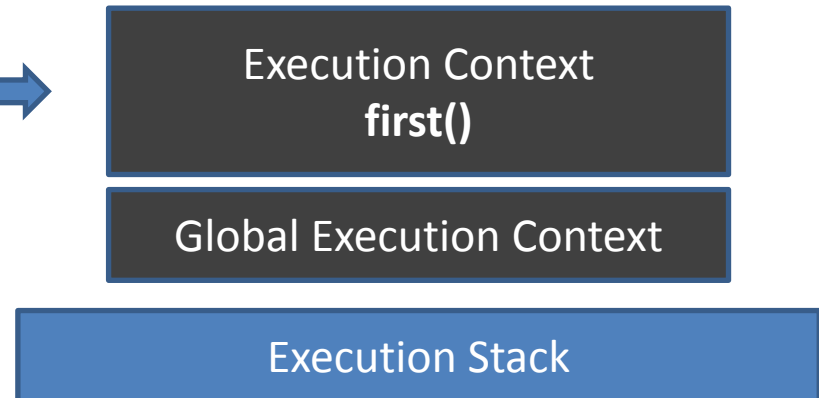
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Execution Context  
**first()**

Global Execution Context

Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Execution Context  
**first()**

Global Execution Context

Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Execution Context  
**second()**

Execution Context  
**first()**

Global Execution Context

Execution Stack



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

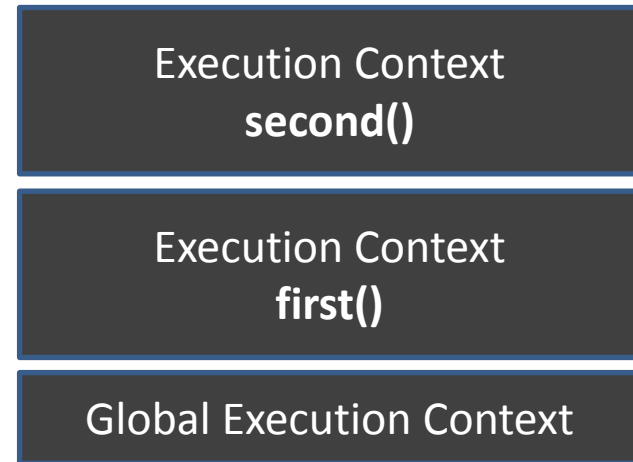
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

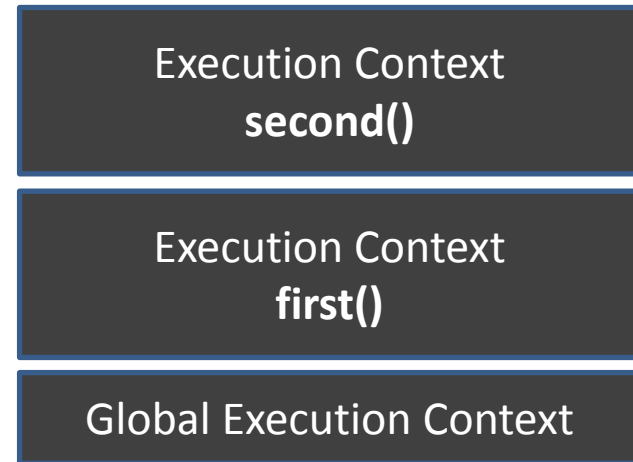
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

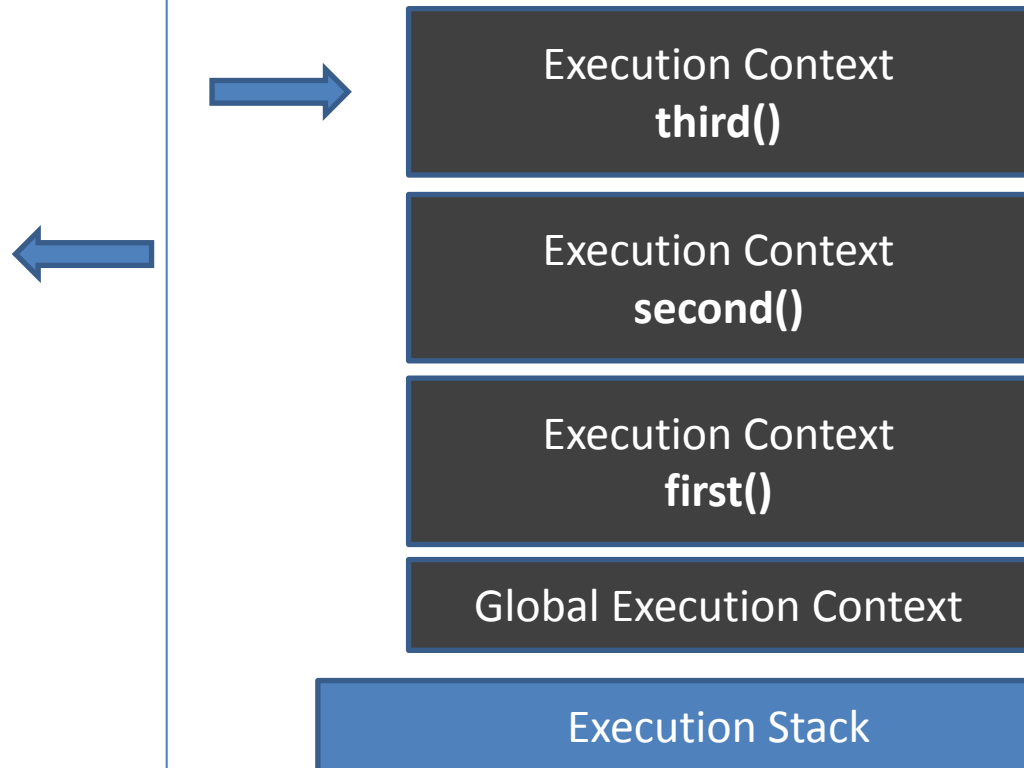
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

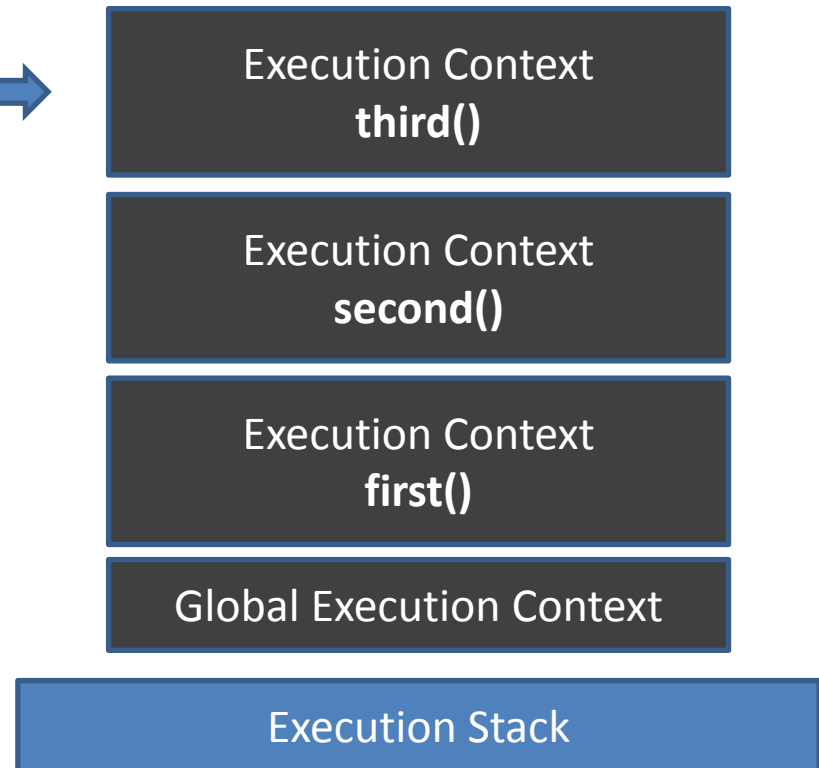
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

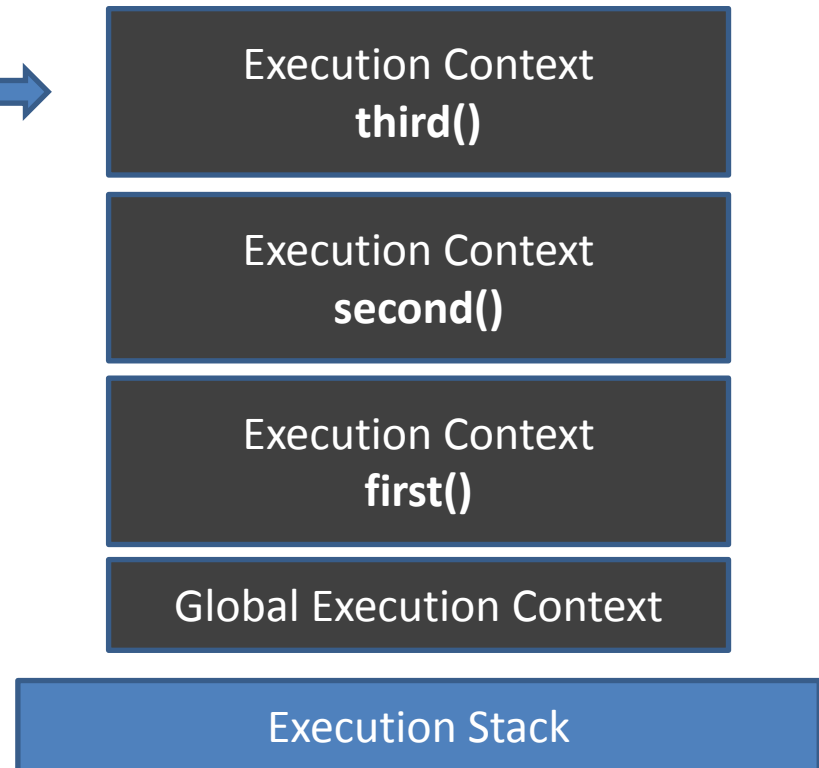
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

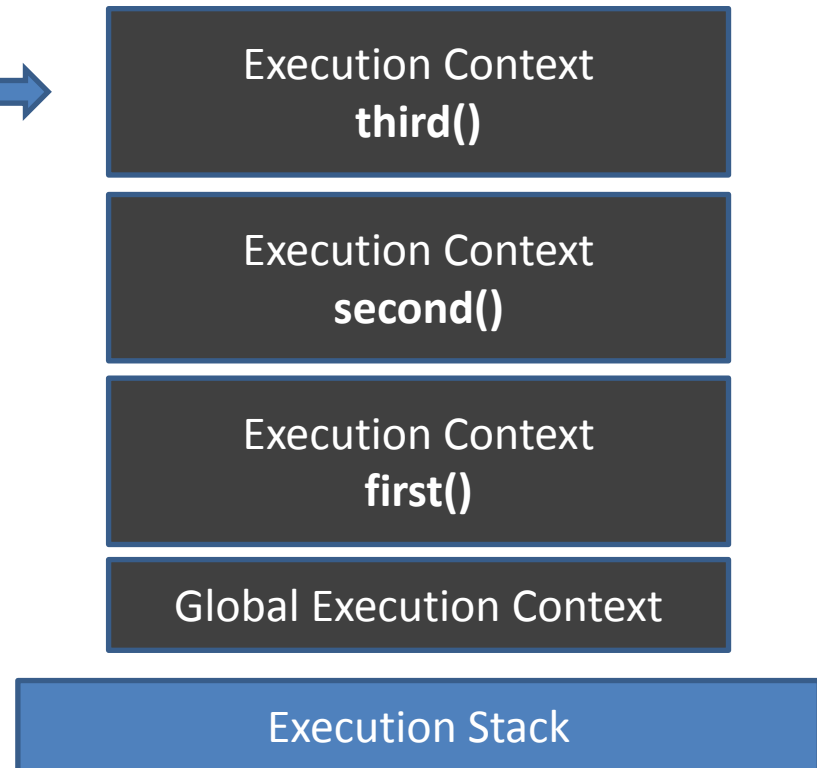
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

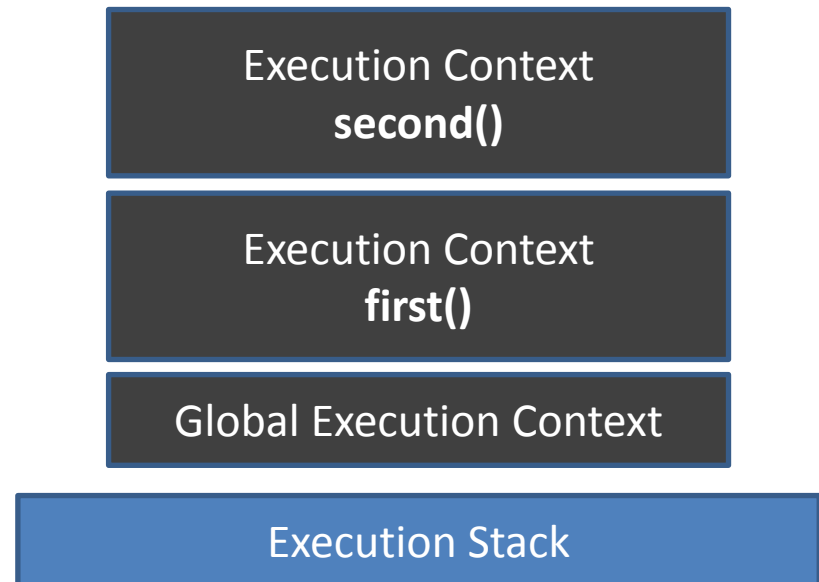
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

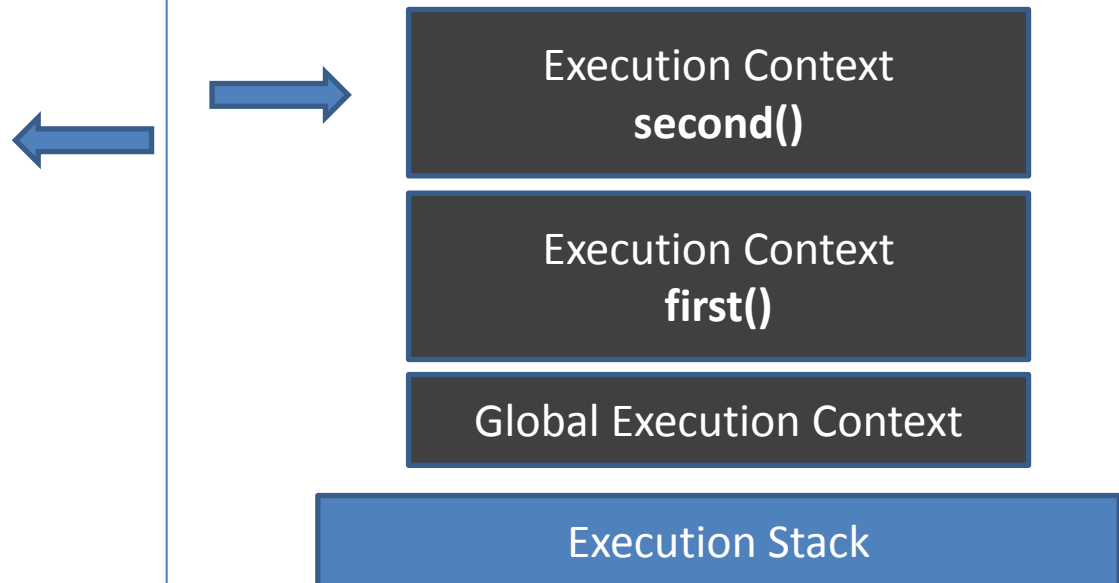
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```





## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

  console.log(x);
}

first();
```



Execution Context  
**first()**

Global Execution Context

Execution Stack

## 2. Execution contexts and execution stack

```
var name = 'John';

function first() {
  var greeting = 'Hello! ';
  var x = greeting + name;

  second();

  console.log(x);
}

function second() {
  var greeting = 'Hi! ';
  var x = greeting + name;

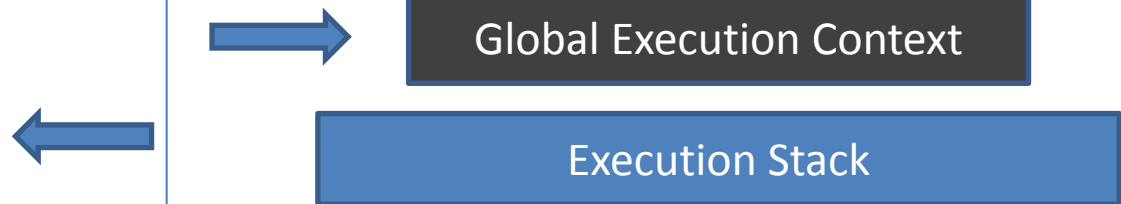
  third();

  console.log(x);
}

function third() {
  var greeting = 'Hey! ';
  var x = greeting + name;

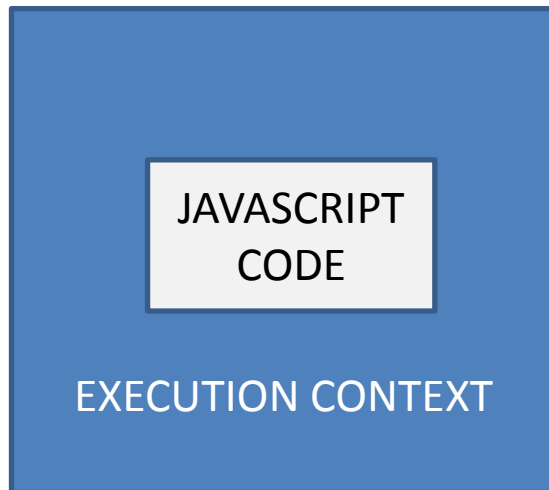
  console.log(x);
}

first();
```



## 2. Execution contexts and execution stack

Execution Context is a Box /Container /Wrapper



= OBJECT

## 2. Execution contexts and execution stack

### Global Execution Context

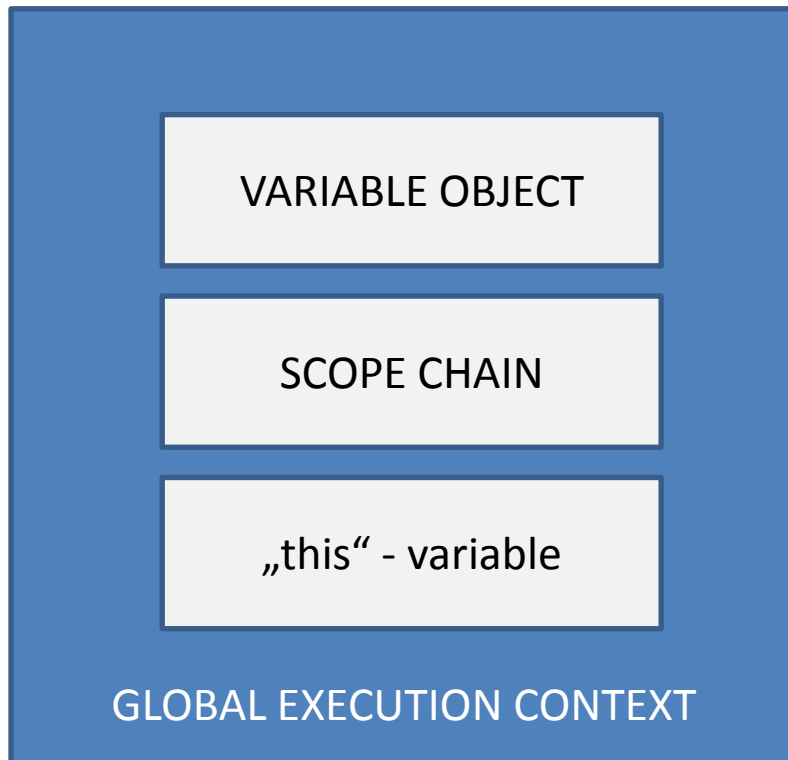


= WINDOW  
OBJECT

- Code that is not inside any of our functions
- Associated with the global object

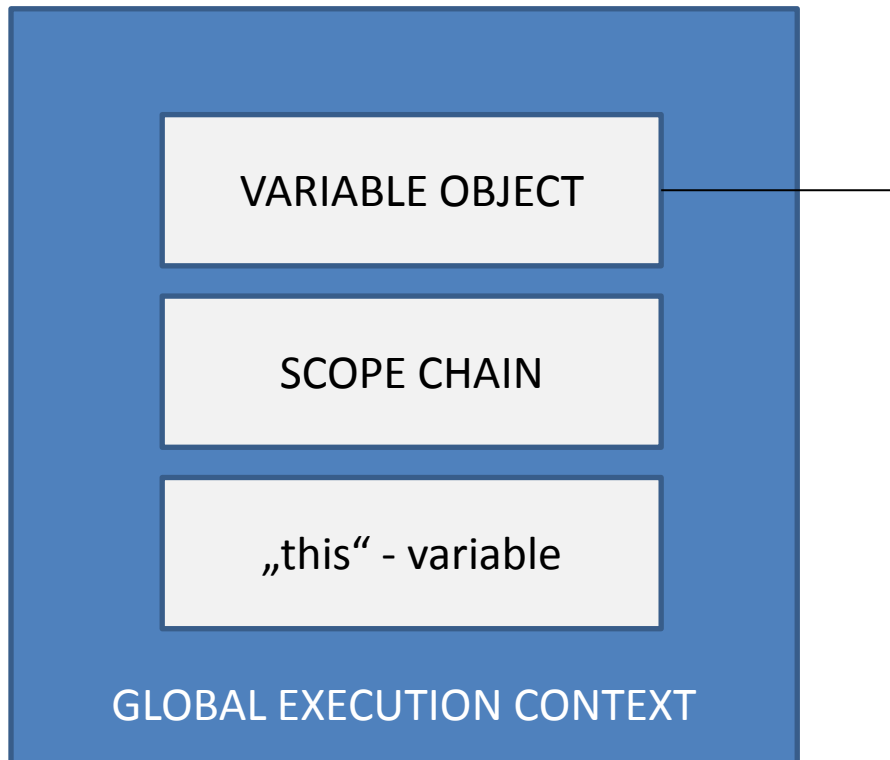
## 2. Execution contexts and execution stack

**Each** Execution Context consists of **three Parts** ...



## 2. Execution contexts and execution stack

### Parts of an Execution Context



When e.g. Google Chrome compiles your JS – Code:

#### HOISTING

Before the code is **executed** line by line, the code is **analyzed** by e.g. Google Chrome which makes

- function declarations
- variable declarations

**available!**

# 3. Hoisting

- „A variable can be declared after it has been used.“
- **Before Execution**, the code is analyzed by the JavaScript Parser of declarations of **functions** and **variables**

```
x = "world";
```

```
var x;
```

```
hello( x );
```

```
function hello(p) {
```

```
    console.log( "hallo " + p );
```

```
}
```

# 3. Hoisting

- „A variable can be declared after it has been used.“
- **Before** Execution, the code is analyzed by the JavaScript Parser of declarations of **functions** and **variables**

```
x = "world";
```

```
var x;
```

```
hello( x );
```

```
function hello(p) {
```

```
    console.log( "hallo " + p );
```

```
}
```



**Variable Object** has

One new property x which is undefined

And

One new function which is hello(p)



# 3. Hoisting

- „A variable is used before it is declared“
- **Before** JavaScript **variables** are

When Chrome executes the code

1. x is set to “world”
2. Hello(x) is called

```
x = "world";  
var x;  
hello( x );
```

```
function hello(p) {  
  console.log( "hallo " + p );  
}
```

is undefined

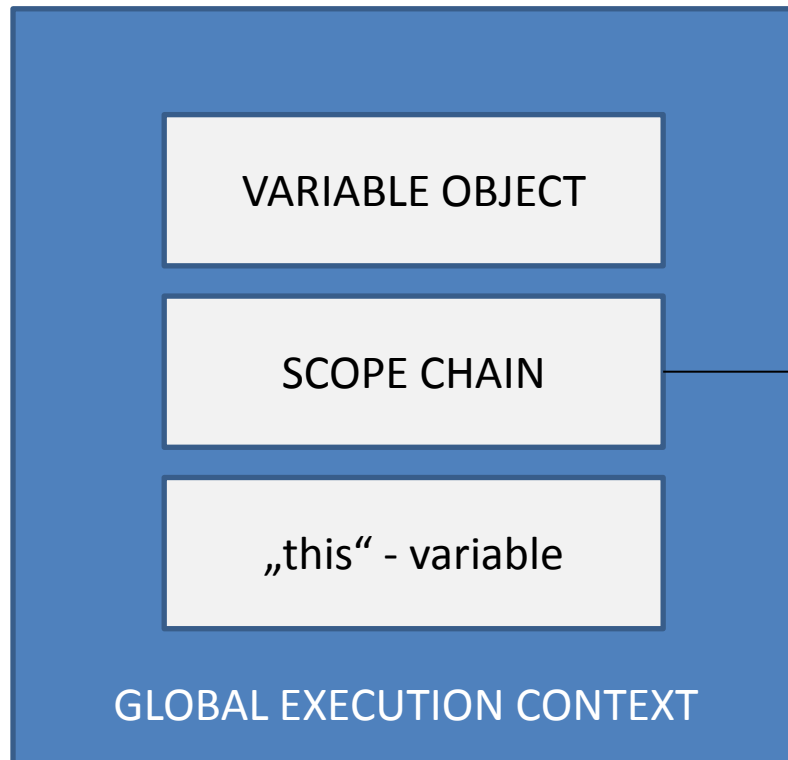
One new function which is hello(p)

## 4. 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

# 4. Scoping

## Parts of an Execution Context



Contains  
Variable Objects of  
all **parents**

# 4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;
```

```
function parent() {  
  var x = 1;
```

```
  function child1() {  
    var z = 1 + x + g;  
    console.log("z is " + z);  
  }
```

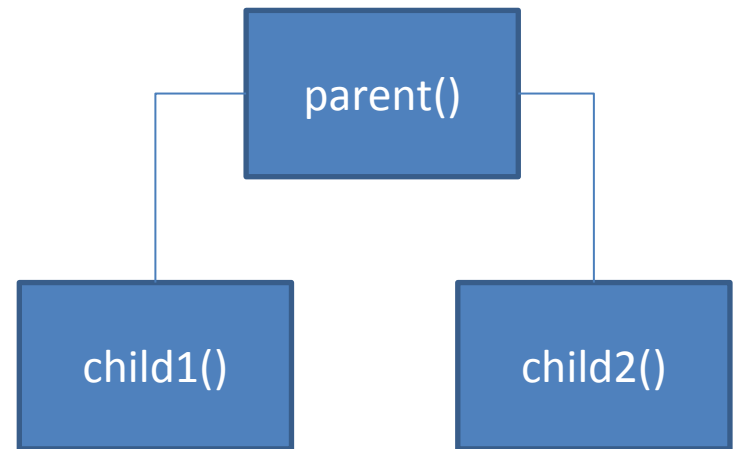
```
  function child2() {  
    var z = 2 + x + g;  
    console.log("z is " + z);  
  }  
}
```

# 4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;
```

```
function parent() {  
  var x = 1;  
  
  function child1() {  
    var z = 1 + x + g;  
    console.log("z is " + z);  
  }  
  
  function child2() {  
    var z = 2 + x + g;  
    console.log("z is " + z);  
  }  
}
```



# 4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;
```

```
function parent() {  
  var x = 1;
```

```
  function child1() {  
    var z = 1 + x + g;  
    console.log("z is " + z);  
  }
```

```
  function child2() {  
    var z = 2 + x + g;  
    console.log("z is " + z);  
  }
```

```
}
```

GLOBAL SCOPE

# 4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;
```

GLOBAL SCOPE

```
function parent() {
```

```
  var x = 1;
```

parent() scope

```
  function child1() {
```

```
    var z = 1 + x + g;
```

```
    console.log("z is " + z);
```

```
  }
```

```
  function child2() {
```

```
    var z = 2 + x + g;
```

```
    console.log("z is " + z);
```

```
  }
```

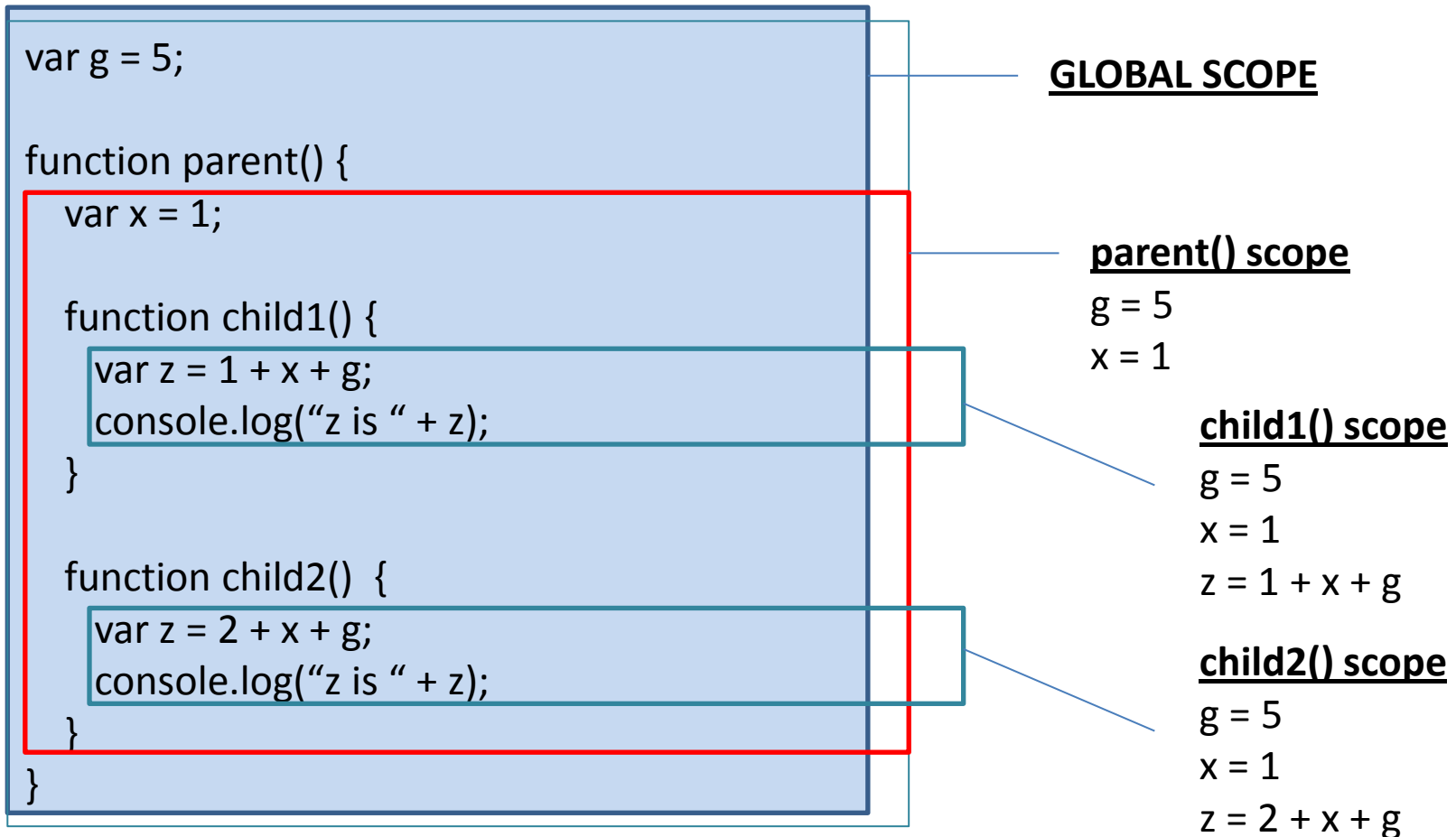
```
}
```

g = 5

x = 1

# 4. Scoping

- What is a „parent“ and a „child“





# 5. Scoping VS Execution Stacks

What is the difference between

**execution stack**

and

**scope chain**

?

# 5. Scoping VS Execution Stacks

Execution Stack:

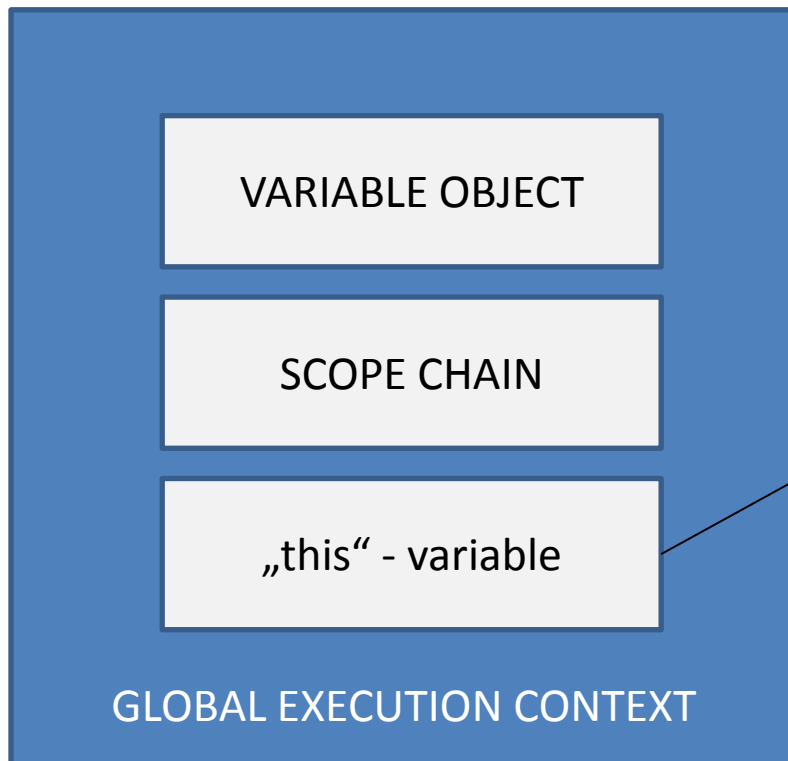
Order in which functions are called.

Scope Chain:

Order in which functions are written lexically.

# 6. this - Variable

- Each Execution Context has



## Function Call

*this* is the global object  
(the window object in the browser)

## Method Call

*this* points to the  
object that is calling the  
method

**this** depends on the  
function/method it is in !