Webscripting

# Hoofdstuk 3

## Functions

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

# Definitie via binding

**Binding tussen naam en stuk code**

De functie is enkel beschikbaar na de definitie
**const: niet wijzigbaar door toekenning**
let: wel wijzigbaar

scope
square

```
const square = function( x ) {
    return x * x;
};
console.log( square( 2 ) );
```

# Definitie via binding

**Binding tussen naam en stuk code**

Functie enkel beschikbaar na de definitie
const: niet wijzigbaar door toekenning
**let: wel wijzigbaar**

```
let safeMode = true;
let launchMissiles = function() {
   console.log("launching missiles");
};
if ( safeMode ) {
   launchMissiles = function() {
      console.log( "nothing happens" );
   };
}
launchMissiles();
```

# Definitie via declaration

**Function declaration is 'hoisted' (omhooggebracht)**
**Je mag de functie aanroepen voor de declaratie:**

```
future();
function future() {
    console.log("You'll never have flying cars");
}
```

```
You'll never have flying cars
```

```
future();
const future = function() {
    console.log("You'll never have flying cars");
}
```

```
ReferenceError: future is not defined
```

# Definitie via arrow-notation

```
const power = (base, exponent) => {
   let result = 1;
   for (let i = 0; i < exponent ; i++) {
      result *= base;
   }
   return result;
}
console.log(power(2,3)); // 8
```

```
const square = (x) => {return x*x;}
console.log(square(2)); // 4
```

```
// 1 argument: haakjes mogen weg
const square2 = x => {return x*x;}
```

```
// zonder accolades: expression w. teruggeg.
const square3 = (x) => x*x;
```

# Optional arguments

**Extra argumenten worden genegeerd**

```
function square(x) {
   return x * x;
};
console.log(square(2,"a",12223));      // 4
```

**Niet gespecificeerd bij aanroepen: undefined**

```
function minus( a, b ) {
   if ( b == undefined ) {
      return -a;
   } else {
      return a - b;
   }
};
console.log(minus(3));     // -3
console.log(minus(3,4));// -1
```

# Scope van bindings

**var (nooit gebruiken!)**

- scope is function indien var in function
  global anders

- hoisting: voor executie worden enkel de declaratie
  omhoog gebracht (niet de initialisatie)

```
console.log(a);
var a = 1;
console.log(a);
```

→

```
var a;
console.log(a); // undefined
a = 1;
console.log(a); // 1
```

# Scope van bindings

**var (nooit gebruiken!)**

- scope is function indien var in function
  global anders

- hoisting: voor executie worden enkel de declaratie
  omhoog gebracht (niet de initialisatie)

```
function test(){

      console.log(a);
      var a = 1;
      console.log(a);
}
test();
```

```
function test(){
   var a;
   console.log(a); //undef.
   a = 1;
   console.log(a); // 1
}
test();
```

```
var x = 1;
function test(){
    console.log(x);
}
test();
```

```
var x;
x = 1;
function test(){
    console.log(x); //1
}
test();
```

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

    console.log(x);
    if (1==2){
        // not executed
        var x = -1;
    }
}
test();
```

```
var x;
x = 1;
function test(){
    var x;
    console.log(x); //undefined
        if (1==2){
            // not executed
            x = -1
        }
    }
test();
```

# Scope

**let, const:**

- scope is code block indien let/const in code block
  global anders
- hoisting: voor executie worden enkel de declaraties omhoog gebracht
- ReferenceError indien gebruikt voor declaratie (temporal dead zone, tdz)

```
console.log(a);
let a;
console.log(a);
a = 1;
console.log(a);
```

```
let a;
console.log(a);// referenceError

console.log(a);
a = 1;
```

```
ReferenceError: a is not defined
```

# Scope

**let, const:**

- scope is code block indien let/const in code block
  global anders
- hoisting: voor executie worden enkel de declaraties omhoog gebracht
- ReferenceError indien gebruikt voor declaratie

```
let a;
console.log(a);
a = 1;
console.log(a);
```

```
let a;
console.log(a); // undefined
a = 1;
console.log(a); // 1
```

```
let a = 1;
console.log(a);
{
    console.log(a);
}
```

```
let a;
a = 1;

{
console.log(a); // 1
}
```

```
let a = 1;
{

    console.log(a);
    let a = 2;
    console.log(a);
}
console.log(a);
```

```
let a;
a = 1;
{
    let a;
    console.log(a);
    a = 2;
    console.log(a);
}
console.log(a);
```

```
ReferenceError: a is not defined
```

# Scope

```
let a = 1;
{
    let a = 2;
    console.log(a);
}
console.log(a);
```

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

# Nested scope

**Function ingredient nested in function hummus**

```
const hummus = function(factor) {
    const ingredient = function(amount, unit, name) {
        let ingredientAmount = amount * factor;
        if (ingredientAmount > 1) {
            unit += "s";
        }
        console.log(`${ingredientAmount} ${unit} ${name}`);
    };
    ingredient(1, "can", "chickpeas");
    ingredient(0.25, "cup", "tahini");
    ingredient(0.25, "cup", "lemon juice");
};
hummus(3);
```

```
3 cans chickpeas
0.75 cup tahini
0.75 cup lemon juice
```

# Nested scope

```
const hummus = function(factor) {  factor: 3
    const ingredient = function(amount, unit, name) {
        let ingredientAmount = amount * factor;
        if (ingredientAmount > 1) {
            unit += "s";
        }
        console.log(`${ingredientAmount} ${unit} ${name}`);
    };
    ingredient( amount: 1,  unit: "can",  name: "chickpeas");
    ingredient( amount: 0.25,  unit: "cup",  name: "tahini");
    ingredient( amount: 0.25,  unit: "cup",  name: "lemon juice");
};

hummus( factor: 3);
```

hummus()

Console

Variables

Local

factor = 3

ingredient = function (amount, unit, name) {

this = global

Global = global

hummus = outer function
2 bindings in hummus:
factor & ingredient

# Nested scope

demo.js | index.html

```
1  const hummus = function(factor) {  factor: 3
2      const ingredient = function(amount, unit, name) {  amount: 1  unit: "can"  name: "chickpeas"
3          let ingredientAmount = amount * factor;  ingredientAmount: 3  amount: 1  factor: 3
4          if (ingredientAmount > 1) {  ingredientAmount: 3
5              unit += "s";  unit: "can"
6          }
7          console.log(`${ingredientAmount} ${unit} ${name}`);  console: Console {_stdout: ,_stderr:
8      };
9      ingredient( amount: 1,  unit: "can",  name: "chickpeas");
10     ingredient( amount: 0.25,  unit: "cup",  name: "tahini");
11     ingredient( amount: 0.25,  unit: "cup",  name: "lemon juice");
12 };
13
14 hummus( factor: 3);
```

hummus() › ingredient()

Console

Variables

- Local
  - amount = 1
  - console = Console {_stdout: ,_stderr: ,_times: }
  - factor = 3
  - ingredientAmount = 3
  - name = "chickpeas"
  - unit = "can"
  - this = global
- Closure
- Global = global

ingredient is inner function

4 bindings in ingredient: amount, unit, name, ingredientAmount & factor

<u>factor afkomstig van outer function!</u>

# Closures

Bij  een nested function heeft de inner-function ook toegang tot de variabelen van de outer function.

| | |
|---|---|
| generateMultiplier | outer function<br>geeft function terug |
| returned function | inner function<br>heeft toegang tot binding factor |

```
const generateMultiplier = function(factor) {
    return function(number) {
        return number * factor
    };
};

const twice = generateMultiplier(2);
const threeTimes = generateMultiplier(3);
console.log(twice(5)); // 10
console.log(threeTimes(6)); // 18
```

# Closures

demo.js | index.html

```
1   const generateMultiplier = function (factor) {
2       return function(number) {
3           return number * factor
4       };
5   };
6
7   const twice = generateMultiplier( factor: 2);
8   const threeTimes = generateMultiplier( factor: 3);
9   console.log(twice(5));
10  console.log(threeTimes(6));
```

Console

Variables

- Local
  - factor = 3
  - number = 6
  - this = global
- Closure
  - factor = 3
- Global = global

# Recursion

**Functie die zichzelf aanroept**

Machtsverheffing (power)

base^exponent = base * base^(exponent-1)

base^0 = 1

```
function power(base, exponent) {
   if (exponent == 0) {
      return 1;
   }
   return base * power(base, exponent - 1);
}
console.log(power(2, 3));
```

# Recursion

**Functie die zichzelf aanroept**

faculteit (factorial)

```
n! = n * (n-1)!
0!=1
```

```
function factorial(number) {
   if (number == 0) {
      return 1;
   }
   return number * factorial(number - 1);
}
console.log(factorial(5));
```

# Recursion (extra)

**zoek getal beginnend van 1 adhv de operaties: x + 3 of x * 3**
**13 = (((1 * 3) + 5) + 5)**

```
function findSolution(target) {
    function find(current, history) {
        console.log(`find : ${current} = ${history}`);
        if (current == target) {
            return history;
        } else if (current > target) {
            return null;
        } else {
            return find(current + 5, `(${history} + 5)`) ||
                find(current * 3, `(${history} * 3)`);
        }
    }
    return find(1, "1");
}
console.log(findSolution(13));
```

```
find : 1 = 1
find : 6 = (1 + 5)
find : 11 = ((1 + 5) + 5)
find : 16 = (((1 + 5) + 5) + 5)
find : 33 = (((1 + 5) + 5) * 3)
find : 18 = ((1 + 5) * 3)
find : 3 = (1 * 3)
find : 8 = ((1 * 3) + 5)
find : 13 = (((1 * 3) + 5) + 5)
(((1 * 3) + 5) + 5)
```

# Besluit

**Declaratie van functions:**

```
const twice = function (number) {
   return number * 2;
}
```

```
function theeTimes (number) {
   return number * 3;
}
```

```
const fourTimes = number => number*4;
```

```
const fiveTimes = (number) => {return number * 5;};
```

# Besluit

**let, const**

- scope is code block indien let/const in code block
  global anders
- hoisting: voor executie worden enkel de declaraties omhoog gebracht
- ReferenceError indien binding gebruikt voor declaratie

**closure**

Bij een nested function heeft de inner-function ook toegang tot de variabelen van de outer function.