Web Development II

Hoofdstuk 02: Functies – Arrays - Modules

HO GENT

Inhoud

- Functies:
  - Doel
  - Syntax
  - Hoisting
- Arrays
- Modules

HO GENT

02 Functies – Arrays - Modules

Functies

HO
GENT

Functies

- efficiëntere code
- herbruikbaar
- elimineert dubbele code
- grote problemen opdelen in kleinere eenheden.
- er bestaan built-in functies zoals alert, Math.round(),...

HO GENT

Functies

- Declaratie

```
function functionname(par1,par2,...,parX) {
   statements
};
```

Merk op : ook hier geen datatype voor parameters en geen returntype!

- Uitvoeren

```
functionname(arg1, arg2,….argx);
```

- Een functie kan een waarde retourneren via het return statement. Indien geen return statement aanwezig wordt undefined geretourneerd.

```
function getAvatar(points) {
    let avatar;
    if (points < 100) {
        avatar = 'Mouse';
    } else if (points < 1000) {
        avatar = 'Cat';
    } else {
        avatar = 'Gorilla';
    }
    return avatar;
}
const myAvatar = getAvatar(335);
console.log(`my avatar : ${myAvatar}`); //Cat
```

Functies

- Via parameters kunnen waarden worden doorgegeven aan functies.

```
function showMessage(from, text) { // parameters: from, text
      console.log(`${from}: ${text}`);
}

showMessage('Ann', 'Hello!'); // Ann: Hello! (*)
showMessage('Ann', "What's up?"); // Ann: What's up? (**)
```

- Als de functie wordt aangeroepen worden de waarden toegewezen aan de lokale variabelen *from* en *text.*
  De functie zal deze lokale variabelen dan gebruiken. Deze lokale variabelen mogen gewijzigd worden binnen de functie.
- Merk op:
  - De variabele binnen de haken van een functie is een *parameter* (wordt gedeclareerd).
  - Een waarde die wordt doorgegeven aan een functie via een parameter is een *argument.*

HO GENT

Functies

- Er kan aan een parameter van een functie een default value worden toegekend.
  Indien de functie wordt aangeroepen zonder alle argumenten dan wordt de default value gebruikt.

```
function showMessage(from, text = "empty message") {
      console.log(`${from}: ${text}`););
}

showMessage("Ann"); // Ann: empty message
```

- Indien geen default value wordt voorzien kan ook gebruikt worden van de *nullish coalescing operator ??*

```
function showMessage(from, text) {
      text = text ?? "empty message";
       console.log(`${from}: ${text}`);
}
showMessage("Ann"); // Ann: no text given
```


HO GENT

Functies

Een variabele gedeclareerd binnen de functie block is enkel zichtbaar binnen de functie.

In het voorbeeld is de variabele avatar enkel zichtbaar binnen de functie getAvatar().

Let op: een variabele (global variable) gedeclareerd buiten de functie is ook zichtbaar binnen de functie. Hier wordt in een later hoofdstuk dieper op ingegaan.

```
function getAvatar(points) {
    let avatar;
    if (points < 100) {
        avatar = 'Mouse';
    } else if (points < 1000) {
        avatar = 'Cat';
    } else {
        avatar = 'Gorilla';
    }
    return avatar;
}
const myAvatar = getAvatar(335);
console.log(`my avatar : ${myAvatar}`); //Cat
```


HO GENT

Hoisting

"Hoisting is a JavaScript mechanism where **variables and function declarations** are **moved to the top of their scope** before code execution."

HO
GENT

Hoisting

- functie en variabele declaraties worden eerst gecompileerd alvorens de browser de script code uitvoert
- na de compilatie fase zijn de variabelen en functies gekend
- **Hoisting en function declarations**

```
sayHi('Bob'); //alert: Hi, my name is Bob

function sayHi(name) {
   alert(`Hi, my name is ${name}`);
}

sayHi('Bob'); //alert: Hi, my name is Bob
```

Een functie op dergelijke manier gedeclareerd, kan overal in de code gebruikt worden, zelfs vóór zijn declaratie!!

HO GENT

Hoisting

Hoisting en let/const variables

- hoisting gebeurt ook **voor de declaratie** van variabelen gedeclareerd met let en const
- hoisting gebeurt **niet voor de initialisatie** van de variabelen!

```
'use strict';

x = 10;
console.log(x);
```

```
Uncaught ReferenceError: x is not defined
    at variabelen.js:3
```

klassiek: in strict mode moet je je variabelen declareren...

```
'use strict';

x = 10;
let x = 20;
console.log(x);
```

```
Uncaught ReferenceError: Cannot access 'x' before initialization
    at variabelen.js:3
```

de variabele is gedeclareerd en de declaratie is gehoist, de initialisatie is niet gehoist en je mag niet naar de variabele verwijzen alvorens ze werd geinitialiseerd

Functies

Window built-in functions: dit zijn methodes (functions) van het window object.

window.alert() - window.confirm() - window.prompt(,)
(window mag worden weggelaten)

alert box: *alert ("sometext");*

confirm box: *confirm("sometext");*
ok: retourneert true – cancel: retourneert false

prompt box: *prompt("sometext","defaultvalue");*
als geen defaultValue voorzien wordt null geretourneerd

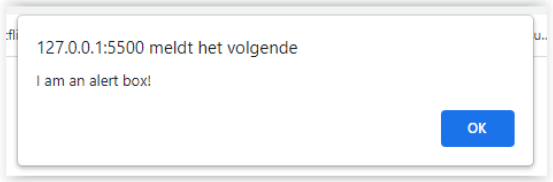
HO GENT

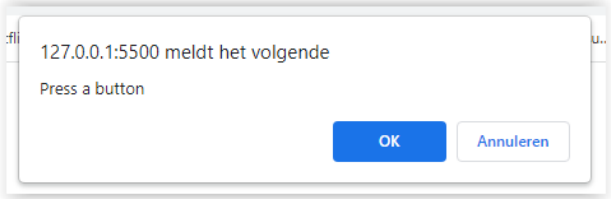
Functies

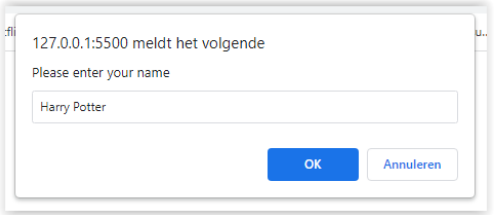
```
function doAlert  () {
    alert('I am an alert box!');
}

function doConfirm () {
    const antwoord = confirm('Press a button');
    if (antwoord === true)
        alert('You pressed OK!');
    else
        alert('You pressed Cancel!');
}

 function doPrompt () {
    const name = prompt('Please enter your name', 'Harry Potter');
    if (name !== null && name !== "") {
        alert(`Hello ${name}! How are you today?`);
    }
```







HO GENT

Functies

- Alle functies tot nog toe zijn gecreëerd door een zogenaamde *functie declaratie.*
- Functies kunnen ook gecreëerd worden door een expressie: *functie expressie*

```
function sayHi() {
     alert( "Hello" );
}
```

```
const sayHi = function() {
     alert( "Hello" );
};
```


HO GENT

Functie expressie

Men kan een functie ook als een waarde (type function) beschouwen die je aan een variabele toekent.
Dit is het geval bij functie expressies.
Meer hierover in volgend hoofdstuk.

Merk op:

- er wordt geen functienaam meer toegekend aan de functie. Dit is ook niet nodig, aangezien de functie aan een variabele is toegekend.

```
const sayHi = function() {
     alert( "Hello" );
};

sayHi();
```


HO GENT

02 Functies – Arrays - Modules

Arrays

HO
GENT

Arrays

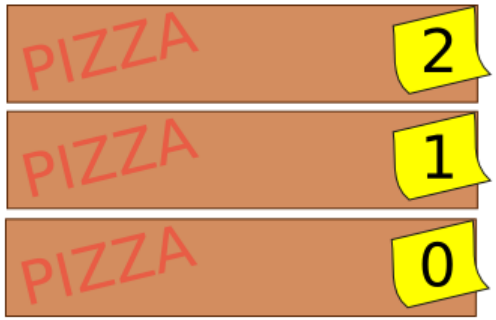
- een **array** is een geordende verzameling van **elementen**, deze mogen van een **verschillend type** zijn.
- elk element heeft een genummerde positie in de array, **index** genaamd: het eerste element heeft index 0.
- arrays zijn **dynamisch**: er wordt geen grootte gegeven bij creatie, ze kunnen groeien/krimpen
- array notatie: **[ ]**

HO GENT

Arrays

- De Ninja Pizzeria maakt pizza's en stopt ze dan in dozen, klaar om te leveren.

  De gestapelde lege pizzadozen



  Een array is een verzameling lege dozen waar je waarden kan in stoppen.

HO GENT

Referentie : Jones, D. (2014). *JavaScript: Novice to Ninja* (p. 500). SitePoint.

Arrays

- declaratie van een **lege array** op twee manieren

```
const pizzas = [];
```

```
const pizzas = new Array();
```


HO GENT

Arrays

waarden toekennen aan een array

mogelijks van verschillende types

```
pizzas[0] = 'Margherita';
pizzas[1] = 'Mushroom';
pizzas[2] = 'Spinach & Rocket';
```

wijzigen

```
pizzas[0] = 'Ham & Pineapple';
```

HO GENT

Arrays

aanmaken van array via array literal

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
```

voorbeeld met elementen van verschillend type

```
const mixedArray = [null, 1, 'two', true, undefined ];
```


HO GENT

Arrays

opvragen 1 element adhv de index:

- **index** start vanaf 0. Gebruik [ ]
- **length**: aantal elementen in array

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];

console.log(pizzas[2]); //Spinach & Rocket

console.log(`eerste pizza ${pizzas[0]}`); //Margherita

console.log(`laatste pizza ${pizzas[pizzas.length-1]}`); //Pineapple & Sweetcorn
```

de volledige array weergeven:

```
console.log(pizzas); // Array(4) ['Margherita', 'Mushroom', 'Spinach & Rocket',
'Pineapple & Sweetcorn'];
```


HO GENT

Arrays

verwijderen van een element in de array

```
const pizzas =  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
delete pizzas[2];
console.log(pizzas); // ['Margherita', 'Mushroom', undefined, 'Pineapple & Sweetcorn'];
```

verwijdert de waarde op deze positie, maar de ruimte bestaat nog steeds en bevat nu de waarde **undefined**.

HO GENT

Arrays

een array itereren:

```
const pizzas =  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
```

for(…; …; …)

```
for (let i = 0; i < pizzas.length; i++) {
        console.log(pizzas[i]);
}
```

for(… of ….)

```
for (let value of pizzas) {
        console.log(value);
}
```

```
Margherita
Mushroom
Spinach & Rocket
Pineapple & Sweetcorn
```


HO GENT

Arrays

Gaten creëren.

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
pizzas[30] = 'Vegetarian'; //er wordt een waarde toegevoegd op index 30
console.log(pizzas.length); // 31
console.log(pizzas[20]); //undefined
```


HO GENT

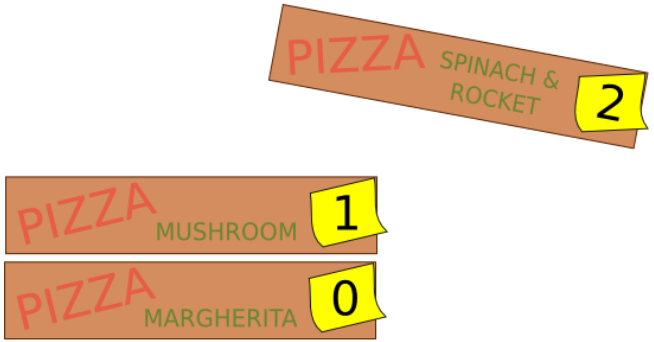
Arrays

pop, push, shift en unshift

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];
```

pop verwijdert het laatste element uit array en retourneert dit element.

```
pizzas.pop(); // << 'Spinach & Rocket‘
console.log(pizzas); // ['Margherita', 'Mushroom']
```



Arrays

pop, push, shift en unshift

push voegt één of meerdere waarden toe aan het einde van de array en retourneert de nieuwe lengte van de array.

```
pizzas.push('Pepperoni'); // << 3
```

Arrays

pop, push, shift en unshift

shift : verwijdert de eerste waarde in array en returnt deze waarde.

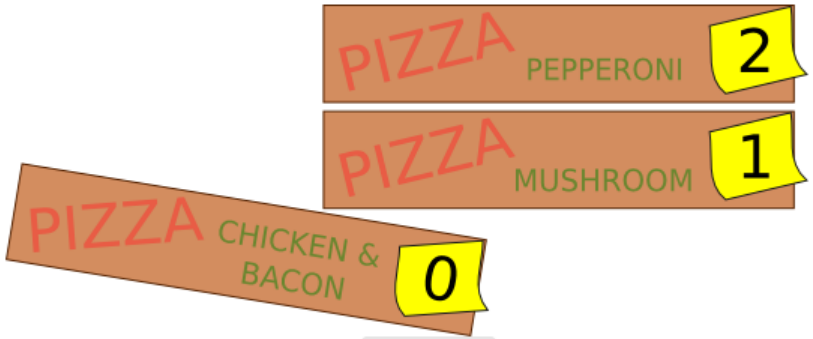
```
pizzas.shift(); //<< 'Margherita'
```

Arrays

pop, push, shift en unshift

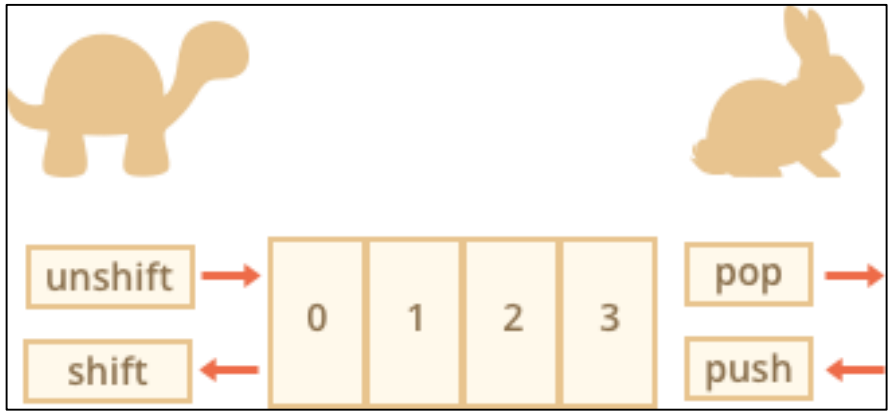
unshift : voegt één of meerdere waarden toe aan het begin van het array en returnt de nieuwe lengte van de array.

```
pizzas.unshift('Chicken & Bacon'); //<< 3
```



Arrays

pop, push, shift en unshift



de array vooraan laten groeien en krimpen is een trager proces omdat overige elementen in de array moeten verplaatst worden...

HO GENT

Arrays

zoeken of een waarde voorkomt in een array

indexOf : retourneert index van eerste voorkomen of -1 als waarde niet voorkomt.

```
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket];
pizzas.indexOf('Spicy Beef'); //<< -1
pizzas.indexOf('Margherita'); //<< 0
```


HO GENT

Arrays

Meer informatie over arrays:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array

Andere vaak gebruikte array methodes

concat(): voegt 2 arrays samen

reverse(): keert de volgorde van de array elementen om

slice(start_index, upto_index): returnt een nieuw array als een stuk van de oorspronkelijke array met als argumenten de begin- en een eindpositie.

splice(start_index, numberofitemsToRemove, waarde1,..., waardex): verwijdert numberofItemsToRemove waarden uit de array startend op positie start_index en voegt dan de nieuwe waarden toe waarde1... waardex

Arrays

andere vaak gebruikte array methodes

- ***sort()*** : sorteert de elementen in de array
- ***indexOf(searchElement[, fromIndex])*** : de index van het eerste voorkomen van het element vanaf fromIndex
- ***lastIndexOf(searchElement[, fromIndex])*** : idem indexOf maar begint achteraan
- ***join()***: converteert alle elementen van een array tot 1 lange string

HO GENT

Arrays

```
// Array built-in methods
let pizzas = ["Chicken & Bacon", "Mushroom", "Pepperoni"];
console.log(pizzas);
pizzas = pizzas.concat(["Spicy Beef", "Chicken & Mushroom"])
console.log(pizzas);
console.log(pizzas.join());
console.log(pizzas.slice(2, 4));
console.log(pizzas.splice(2, 1, "Chicken & Pepper", "Veggie Deluxe"));
console.log(pizzas);
console.log(pizzas.reverse());
console.log(pizzas.sort());
```

Arrays

Destructuring assignment

is een manier om meerdere waarden te extraheren uit een array en toe te kennen aan variabelen

```
//Variabele declaraties
//ophalen van eerste en tweede item uit een array
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Chicken & Bacon'];
const [eerstePizza, tweedePizza] = pizzas;
console.log(eerstePizza); // Margherita
console.log(tweedePizza); // Mushroom

//ophalen van derde item uit een array
const [, , derdePizza] = pizzas;
console.log(derdePizza); //Spinach & Rocket
```


HO GENT

Arrays

Destructuring assignment

```
//Destructuring Assignment
const pizzas = ['Margherita', 'Mushroom', 'Spinach & Rocket'];
let pizza1, pizza2;
[pizza1, pizza2] = pizzas;
console.log(pizza1); // Margherita
console.log(pizza2); // Mushroom

//default values
pizzas = ['Margherita'];
[pizza1, pizza2 = 'Mushrooms' ] = pizzas;
console.log(pizza1); // Margherita
console.log(pizza2); // Mushrooms
```


HO GENT

Arrays

Destructuring assignment

```
// Swapping variables in ECMAScript 5
let a = 1,b = 2, tmp;
tmp = a;
a = b;
b = tmp;
console.log(a); // 2
console.log(b); // 1

// Swapping variables vanaf ECMAScript 6
a = 1;
b = 2;
[ a, b ] = [ b, a ];
console.log(a); // 2
console.log(b); // 1
```


HO GENT

Arrays

meerdimensionele arrays

een array van een array = 2 dimensionale array.

een array van een array van een array = 3 dimensionale array.

voorbeeld : een array die de kaarten van 2 poker hands bevat

```
const hands = [];
hands[0] = [5,'A',3,'J',3];
hands[1] = [7,'K',3,'J',3];
console.log ('2de kaart, 2de hand : ' + hands[1][1]);
```


HO GENT

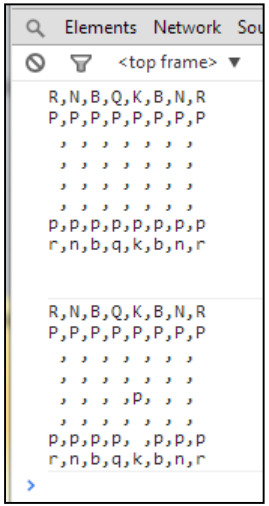
Arrays

2 dimensionale array:

Een schaakbord, voorgesteld als een 2 dimensionale array van strings. De eerste zet verplaatst ‘p’ van positie (6,4) naar (4,4). 6,4 wordt op blanco geplaatst.

```
function playChess() {
    const board = [
        ['R', 'N', 'B', 'Q', 'K', 'B', 'N', 'R'],
        ['P', 'P', 'P', 'P', 'P', 'P', 'P', 'P'],
        [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
        [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
        [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
        [' ', ' ', ' ', ' ', ' ', ' ', ' ', ' '],
        ['p', 'p', 'p', 'p', 'p', 'p', 'p', 'p'],
        ['r', 'n', 'b', 'q', 'k', 'b', 'n', 'r']];

    console.log(board.join('\n') + '\n\n');
    board[4][4] = board[6][4]; // Move King's Pawn forward 2
    board[6][4] = ' ';
    console.log(board.join('\n'));
}
```



HO GENT

02 Functies – Arrays - Modules

Modules

HO
GENT

Modules

Zolang scripts beperkt zijn in size is één js-bestand voldoende voor je applicatie.

Indien de code uitgebreid en complex (meerdere functies – klassen(zie later)) wordt, kan het gebruik van aparte bestanden interessant worden om je code overzichtelijk en mooi gestructureerd te houden.

Ook het hergebruik van klassen en functies kan het gebruik van meerdere aparte bestanden interessant zijn.

Zo een apart bestand is een module.

Het gebruik van modules is standaard voorzien sinds 2015.

HO GENT

Modules

Een module is een js-bestand.

Modules kunnen elkaar laden, daarvoor wordt gebruik gemaakt van de directives:

- *export*: variabelen, functies, klassen worden toegankelijk buiten de huidige module
- *import*: staat toe om functionaliteit (variabelen, functies, klassen) van andere modules te importeren en gebruiken

HO GENT

Modules

Beschouw twee bestanden: sayHi.js en main.js:

sayHi.js

```
export function sayHi(user) {
    console.log(`Hello, ${user}!`);
}
```

main.js

```
import { sayHi } from './sayHi.js';

console.log(sayHi); // function...
sayHi('John'); // Hello, John!
```

Let op: enkel de main.js moet in de html gedeclareerd worden, maar moet wel aangeduid worden als module met het attribuut *type="module"*

```
<body>
    <script type="module" src="js/main.js"></script>
</body>
```


HO GENT

Modules

Belangrijkste kenmerken modules:

- modules werken steeds in strict mode.
- elke module heeft zijn eigen scope (top-level): variabelen, functies en klassen zijn niet automatisch zichtbaar voor ander modules.
- Gebruik de *export* directive om aan te geven wat toegankelijk is van buiten de module en de *import* directive voor wat er nodig is.
- Een module wordt maar één keer geëvalueerd, namelijk bij het importeren.

HO GENT

Modules

Beschouw twee bestanden: user.js en hello.js:

```
<body>
  <script type="module" src="js/user.js"></script>
  <script type="module" src="js/hello.js"></script>
</body>
```

user.js

```
const user = "John";
```

```
export const user = "John";
```

hello.js

```
console.log(user);
```

```
import { user } from "./user.js";
console.log(user);
```

```
Live reload enabled.                                    index.html:40
Uncaught ReferenceError: user is not defined            hello.js:1
    at hello.js:1:13
>
```

```
John                                                    hello.js:2
>
```

Modules: export

export wordt vaak voor de declaraties geplaatst.

```
// export an array
export let months = ['Jan', 'Feb', 'Mar','Apr', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
// export a constant
export const MODULES_BECAME_STANDARD_YEAR = 2015;
// export a class
export class User {
     constructor(name) {
          this.name = name;
     }
}
```


HO GENT

Modules: export

export kan ook na de declaraties geplaatst worden

```
function sayHi(user) {
        alert(`Hello, ${user}!`);
}

function sayBye(user) {
        alert(`Bye, ${user}!`);
}

export {sayHi, sayBye}; // a list of exported variables
```


HO GENT

Modules: import

import wordt gebruikt om functionaliteiten (variabelen-functies-klassen-...) beschikbaar te stellen. Na import volgt een lijst (tussen acolades) van beschikbare code en dan de relatieve verwijzing (begint met ./) naar de module.

```
import {sayHi, sayBye} from './say.js';

sayHi('John'); // Hello, John!
sayBye('John'); // Bye, John!
```


HO GENT

Modules: import

import kan ook met een * gebruikt worden. Dit gebeurt als er veel moet geïmporteeerd worden. Je moet dan een alias (=object) voor * ingeven (na *as*). Je kan dan de geïmporteerde functionaliteiten benaderen als een object.

```
import * as say from './say.js';

say.sayHi('John'); // Hello, John!
say.sayBye('John'); // Bye, John!
```


HO GENT

Modules: export – import

Je kan zowel voor de import als de export een alias gebruiken voor de geëxporteerde/ geïmporteerde functionaliteiten: gebruik het keyword *as.*

sayHi.js

```
function sayHi(user) {
	alert(`Hello, ${user}!`);
}

function sayBye(user) {
	alert(`Bye, ${user}!`);
}

export {sayHi, sayBye}; // a list of exported variables
```

main.js

```
import {sayHi as hi, sayBye as bye} from './say.js';

hi('John'); // Hello, John!
bye('John'); // Bye, John!
```


HO
GENT

Modules: export – import

Dit kan ook.

sayHi.js

```
function sayHi(user) {
        alert(`Hello, ${user}!`);
}

function sayBye(user) {
        alert(`Bye, ${user}!`);
}

export {sayHi as hi, sayBye as bye}; // a list of exported variables
```

main.js

```
import {hi, bye} from './say.js';

hi('John'); // Hello, John!
bye('John'); // Bye, John!
```


HO GENT

Modules: export default

Vaak wordt een klasse in een apart bestand geplaatst en beschikbaar gesteld via een *export*.
Dan wordt het gebruik van de accolades bij de import een beetje overbodig en verwarrend om deze klasse te gebruiken.
Hiervoor wordt de default export geïntroduceerd.
Let op: er kan maar één maal default gebruikt worden per js-bestand.

```
// 🗀 user.js
export default class User { // just add "default"
  constructor(name) {
    this.name = name;
  }
}
```

```
// 🗀 main.js
import User from './user.js'; // not {User}, just User

new User('John');
```

Named export	Default export
`export class User {...}`	`export default class User {...}`
`import {User} from ...`	`import User from ...`

HO GENT