# Web Development II

## Hoofdstuk 02: Functies – Arrays - Modules

HO GENT

# Inhoud

- Functies
  - declaratie & gebruik
  - default parameters
  - hoisting
  - built-in functions & globalThis
  - function expressions
- Arrays
  - declaratie en gebruik
  - Array-functies
- Modules
  - doel en structuur
  - import & export



HO GENT

# 02 Functies – Arrays - Modules

## Functies

HO
GENT

# Functies

A function in JavaScript contains a set of statements that perform a task.

Functions are one of the fundamental building blocks in JavaScript. It takes some input and returns an output where there is some obvious relationship between the input and the output.

To use a function, you must define it somewhere in the scope from which you wish to call it.

- functies
  - leiden tot herbruikbare code
  - laten toe om dubbele code te elimineren
  - laten toe om oplossingen voor grotere problemen op te delen in kleine stukjes



HO GENT

# Functies

- declaratie
  - **function** keyword
  - **naam**
  - lijst van **parameters**
    - tussen ronde haakjes, gescheiden door komma's
  - de JavaScript **statements** die de functie definiëren
    - tussen accolades
    - gebruik het **return** statement om een waarde te retourneren

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

```
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
```

*merk op : geen datatype voor parameters en geen returntype!*

# Functies

- aanroep
  - gebruik de **naam** van de functie
  - voorzie **argumenten** voor de parameters
    - tussen ronde haakjes, gescheiden door komma's
- merk op
  - een functie retourneert altijd een waarde
  - indien een functie niet expliciet een waarde retourneert dan retourneert de functie **undefined**

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

```
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

# 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? (**)
```

- aanroep
  - bij aanroep worden de **waarden van de argumenten** doorgegeven aan de parameters – pass by value
  - binnen de functie gebruik je de parameters net als lokale variabelen
    - je kan deze in de functie body dus mogelijks wijzigen
    - de scope van de parameters is beperkt tot de function body
- merk op:
  - een variabele binnen de haken van een functie **declaratie** noemt men een parameter
  - een waarde die wordt doorgegeven aan een functie bij **aanroep** noemt men een argument.



HO GENT

# Functies

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

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

- default parameters
  - bij declaratie kan aan een parameter een waarde worden toegekend – default value
  - indien bij aanroep geen waarde, of undefined, wordt doorgegeven voor dergelijke parameter dan krijgt deze als waarde de default value
  - indien er voor een parameter **geen default value** is gedefinieerd en er wordt tijdens aanroep **geen waarde** doorgegeven voor de parameter dan krijgt deze parameter de waarde **undefined**



HO GENT

# Functies

- default parameters
  - merk op dat parameters **positioneel** worden ingesteld: left-to-right

```
function f(x = 1, y) {
  return [x, y];
}

f(); // [1, undefined]
f(2); // [2, undefined]
```



HO GENT

# Functies

- default parameters
  - door gebruik van default parameters kan je vermijden om instellingen voor parameters in de body van de functie te doen

```
function multiply(a, b) {
  b = typeof b !== "undefined" ? b : 1;
  return a * b;
}

multiply(5, 2); // 10
multiply(5); // 5
```

```
function multiply(a, b = 1) {
  return a * b;
}

multiply(5, 2); // 10
multiply(5); // 5
multiply(5, undefined); // 5
```



HO GENT

# Hoisting

Hoisting is a JavaScript mechanism where declarations are moved to the top of their scope before code execution.

As a result, you can use a hoisted declaration in its scope before the line it is declared on.

- in JavaScript worden **functie declaraties gehoist**

```
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!!*

# Hoisting

- Hoisting van let/const variables
  - er is een andere vorm van hoisting **voor de let/const declaratie** van variabelen
  - deze vorm van hoisting **laat niet toe dat je een variabele gebruikt vóór zijn declaratie**!

```
'use strict';

const x = 10;
{
  console.log(x); // reference error
  const x = 20;
}
```

*de declaratie const x = 20 creeert een variabele met een block scope (code block afgebakend door de accolades)*

*deze declaratie wordt gehoist: binnen het code block is er dus geweten dat er een **lokale** variabele x is*

*de regel met console.log maakt **geen** gebruik van de globale x, net omdat door de hoisting geweten is dat er een lokale variable x is*

*de lokale variabele mag je echter niet gebruiken vóór zijn declaratie: reference error*



GENT

# Predefined functions

- Javascript kent verschillende top-level built-in functions
  - isNaN, parseInt, parseFloat



HO GENT

# globalThis

- JavaScript’s globalThis
  - deze property bevat de globale this waarde, het verwijst naar het globale object
  - dit is een object die steeds is gedefinieerd en bestaat in de global scope
  - in browser omgevingen is deze globalThis gelijk aan window
  - dit is dus een referentie naar het browser venster die het DOM bevat
- indien je methodes of properties van dit globale object wenst te gebruiken hoef je deze niet te laten voorafgaan door globalThis. of window.
- het gebruik van methodes van window lijkt dan ook sterk op het gebruik van built-in functies



HO GENT

# globalThis

- voorbeeld: alert() – confirm() – prompt()

`Window.alert()`

Displays an alert dialog.

`Window.prompt()`

Returns the text entered by the user in a prompt dialog.

`Window.confirm()`

Displays a dialog with a message that the user needs to respond to.

  - we kunnen deze methodes gebruiken op dezelfde manier als built-in functies



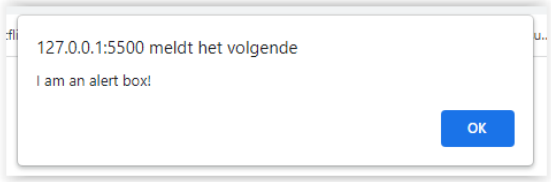
HO GENT

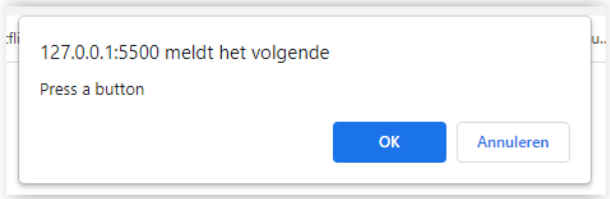
# globalThis

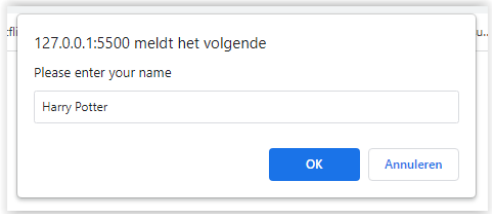
```
function doAlert  () {
    alert('I am an alert box!');
}

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

# globalThis

- voorbeeld 2
  - de window property console

`Window.console` Read only

Returns a reference to the console object which provides access to the browser's debugging console.

  - deze property heeft een log() methode

The `console.log()` method outputs a message to the web console. The message may be a single string (with optional substitution values), or it may be any one or more JavaScript objects.

    - dit is de methode die we gebruiken om gegevens naar de console te schrijven schrijven

```
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? (**)
```



HO GENT

# Functies

- tot nu toe hebben we steeds functies gedeclareerd via een zogenaamde *functie declaratie*.
- functies kunnen ook worden gedeclareerd als *functie expressies*
  - *de declaratie van een functie wordt toegekend aan een variabele*

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

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



HO GENT

# Functie expressie

- in een volgend hoofdstuk wordt dieper ingegaan op functie expressies
- merk nu alvast op dat er geen functienaam meer wordt gebruikt bij de declaratie van de functie
- dit is ook niet nodig, aangezien de functie aan een variabele is toegekend en dat we de naam van de variabele kunnen gebruiken om de functie aan te roepen.

```
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

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

De gestapelde lege pizzadozen


Een array kan je zien als een verzameling doosjes 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 kan op twee manieren

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

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



HO GENT

# Arrays

- arrays kunnen ook via een literal notatie worden gedeclareerd en geinitializeerd

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

- voorbeeld een array met elementen van verschillend type

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



HO GENT

# Arrays

- waarden toekennen aan array elementen
  - mogelijks van verschillende types

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

  - wijzigen

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




HO GENT

# Arrays

- raadplegen van een element adhv de index
  - index start vanaf 0, gebruik [ ] - notatie
  - 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

- itereren over een array

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

  - *for(...; ...; ...)*

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

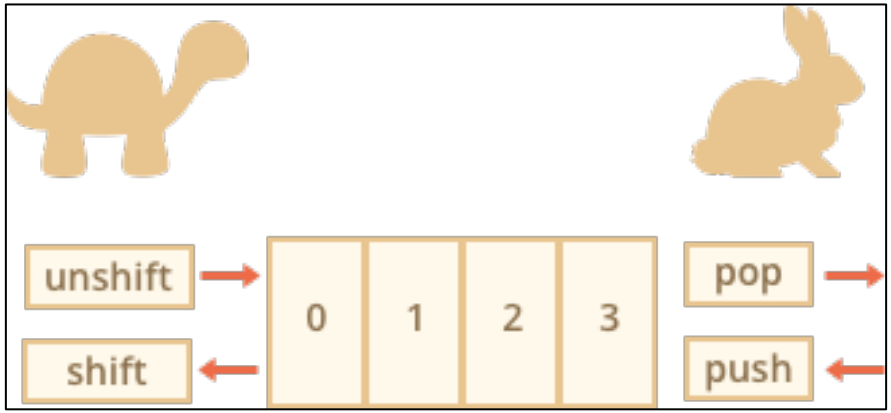
  - *for( ... of ....)*

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

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

# Arrays

- elementen verwijderen via pop & shift
- elementen toevoegen via push & unshift





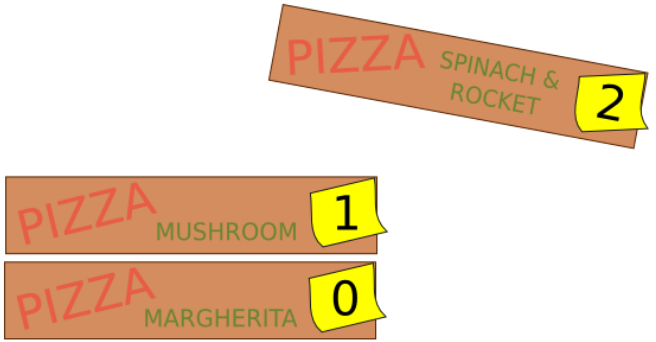

HO GENT

# Arrays

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

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

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



# Arrays

- 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

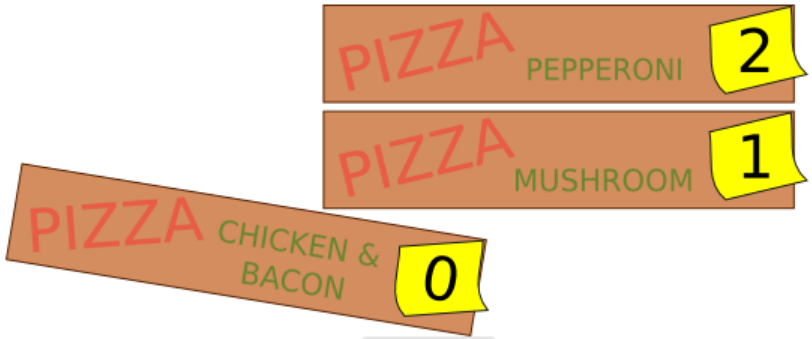
- shift
  - verwijdert de eerste waarde in array en returnt deze waarde

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

# Arrays

- 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

- in arrays kunnen ‘gaten’ onstaan
  - voorbeeld 1 – toekenning aan een element op een index voorbij de length van de array

```
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

- in arrays kunnen ‘gaten’ onstaan
  - voorbeeld 2 – gebruik van de delete operator

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



HO GENT

# Arrays

- indexOf: zoeken of een waarde voorkomt in een array
  - 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

- andere interessante array methodes
  - concat() - voegt 2 arrays samen en retourneert het resultaat
  - 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



HO GENT

# Arrays

- mutating methods vs copying methods:
  - de methodes reverse() en splice(...) **muteren** de array
    - de veranderingen aan de array gebeuren “in-place”
    - de methode retourneert een referentie naar de originele (gewijzigde) array
  - de methodes toReversed() en toSpliced(...) **muteren** de array **niet**
    - de originele array blijft ongewijzigd
    - de methode retourneert een referentie naar een shallow copy met het resultaat



HO GENT

# Arrays

- voorbeelden:

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

mutating

```
const pizzas =  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn’];
const reversed = pizzas.toReversed();
console.log(pizzas); // ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
console.log(pizzas); // ['Pineapple & Sweetcorn', 'Spinach & Rocket', 'Mushroom', 'Margherita'];
```

copying

```
const pizzas =  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
pizzas.splice(1, 2, 'Hawai');
console.log(pizzas); // // ['Margherita', 'Hawai', 'Pineapple & Sweetcorn’];
```

mutating

```
const pizzas =  ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn’];
const splicedPizzas= pizzas.splice(1, 2, 'Hawai');
console.log(pizzas); // ['Margherita', 'Mushroom', 'Spinach & Rocket', 'Pineapple & Sweetcorn'];
console.log(splicedPizzas); // // ['Margherita', 'Hawai', 'Pineapple & Sweetcorn’];
```

copying



HO GENT

# Arrays

- andere interessante array methodes
  - sort(), toSorted() - 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

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

# 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]);
```



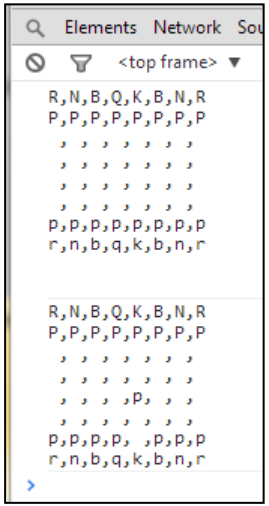
HO GENT

# Arrays

- voorbeeld 2 dimensionale array
  - Een schaakbord, voorgesteld als een 2 dimensionele 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.



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

- voorbeeld
  - beschouw twee bestanden: sayHi.js en main.js:

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

sayHi.js

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

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

main.js

  - HTML:
    - in het script element wordt enkel naar main.js verwezen
    - je moet expliciet aanduiden dat je met modules werkt: type="module"

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



HO GENT

# Modules

- 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
  - *export* directive
    - om aan te geven wat je toegankelijk wil maken voor andere modules
  - *import* directive
    - om aan te geven wat je wil gebruiken uit een andere module
  - een module wordt één keer geëvalueerd, namelijk bij het importeren



HO GENT

# Modules

- voorbeeld:

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

user.js

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

hello.js

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

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

user.js

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

hello.js

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

```
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 mag 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 elementen (variabelen, functies, klassen, …) uit een andere module te gebruiken.
  - specifieer deze in een lijst (tussen accolades)
  - 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 geïmporteerd worden.
  - je moet dan een alias (=object) voor * ingeven (na keyword 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

- as: je kan zowel voor de import als de export een alias gebruiken voor de geëxporteerde/geïmporteerde functionaliteiten

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

- nog een voorbeeld

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*.
  - het gebruik van de accolades bij de import is dan een beetje overbodig en verwarrend
- hiervoor wordt de default export geïntroduceerd.
  - let op: er kan maar één default export 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

HO
GENT