1 Inleiding Javascript

1.1 Wat is Javascript?

In het opleidingsonderdeel 'Web Essentials' hebben we voornamelijk onze focus gelegd op de webtalen HTML en CSS. Met deze twee webtalen kunnen we de structuur en de opmaak van een webpagina vastleggen.

In dit opleidingsonderdeel bouwen we verder op onze kennis uit het voorgaande opleidingsonderdeel en gaan we een derde webtaal leren, genaamd Javascript. Met Javascript kunnen we onze webpagina's interactief maken.

We kunnen Javascript gebruiken voor:

- Formuliervalidatie
- Single-page applicatie
- Ajax-webapplicaties
- Aanpassingen aan de opmaak
- Aanpassingen aan de inhoud
- Animaties
- …

Javascript is een 'scripttaal'. De Javascript-programma's hoeven niet gecompileerd te worden, maar kunnen onmiddellijk worden uitgevoerd. Binnen de browser is hiervoor een component (interpreter) aanwezig die de programma's uitvoert. Het grote voordeel is dat wij, als developer, hierdoor sneller onze code kunnen testen. Gewoon het programma in de browser openen volstaat om de werking na te gaan.

1.2 Benodigde software

1.2.1 Integrated Development Environment

Jetbrains WebStorm is een gespecialiseerde Integrated Development Environment (IDE) voor webontwikkelaars om broncode te schrijven.

Webstorm installeren? https://www.jetbrains.com/community/education/#students

1.2.2 Node.js

Node.js is een cross-platform run-time environment gebaseerd op de open-source JavaScript en WebAssembly V8-engine van Google. Met Node.js kunnen we Javascript gebruiken buiten de browsers.

Via de npm (Node Package Manager) kunnen we ook dependencies installeren.

Node.js installeren? https://nodejs.org/en/download/

1.2.3 Google Chrome

Google Chrome is de webbrowser met handige, ingebouwde developer tools.

Google Chrome installeren? https://www.google.com/chrome/

1.2.4 Jetbrains IDE support

Met de Jetbrains IDE support kunnen we Javascript debuggen zonder een nieuw exemplaar te starten in Chrome.

Jetbrain IDE support installeren? https://chrome.google.com/webstore/detail/jetbrains-ide-support/hmhgeddbohgjknpmjagkdomcpobmllji

1.2.5 Git Bash en Github

Met Git Bash en GitHub kunnen we een lokale en remote repository aanmaken om aan versiebeheer doen.

Git Bash installeren? https://git-scm.com/downloads

GitHub account aanmaken? https://github.com/

2 Starten met Javascript

2.1 Javascript toevoegen

We kunnen Javascript op twee manieren toevoegen aan onze website:

- We plaatsen het script integraal in de HTML-code van een webpagina
- We plaatsen het script in een apart Javascript-bestand (.js)

2.1.1 In de HTML-code

We plaatsen ons script integraal op onze webpagina, tussen de <script>-tags.

HTML-code

```
<!DOCTYPE html>
<html lang=”nl”>
<head>
      <meta charset=”utf-8”>
      <title>Dit is een webpagina met een script</title>
      <script>
            alert(“Hello World – vanuit javascript”);
      </script>
<head>
...
```

We zijn niet verplicht om ons <script>-element binnen het <head>-element van onze webpagina te plaatsen. De locatie is afhankelijk van de manier van werken van het script.

Je gaat zien dat we in de meeste oefeningen van deze cursus ons script helemaal onderaan onze webpagina gaan plaatsen, net voor het afsluiten van het <body>-element. Dit zorgt ervoor dat onze HTML- en CSS-code eerst wordt ingeladen en daarna pas onze JS-code.

HTML-code

```
      ...
      </footer>
      <script>
            alert(“Hello World – vanuit javascript”);
      </script>
</body>
```

2.1.2 In een apart bestand

We plaatsen ons script in een apart javascript-bestand (.js) en we leggen een link naar het aparte javascript-bestand via de <script>-tags.

Ook hier kan je de locatie van het <script>-element in de HTML-code bepalen in functie van de werking van het script.

HTML-code

```
      ...
      </footer>
      <script src="apartbestand.js"></script>
</body>
```

JS-code

```
let code = prompt('Vul uw promotiecode in', 'uw code');
let tekst = 'De code die u invoerde was: ' + code;
Document.getElementById('kortingscode').innerHTML = tekst;
```

2.2 Werking script via Defer of Async

Op moderne websites zijn de scripts vaak 'zwaarder' dan de HTML-pagina's. Dit zorgt ervoor dat de script, wanneer deze eerst worden ingeladen, de laadtijd van de volledige website vergroten. Er zijn twee manieren om dit probleem op te lossen:

- We plaatsen, zoals we op de vorige pagina hebben geleerd, ons `<script>`-element aan het einde van de HTML-pagina, net voor het afsluiten van het <body>-element. Maar bij grote HTML-pagina's kan dit nog steeds voor vertragingen zorgen…
- We voegen het 'async' of 'defer' attribuut toe aan het <script>-element.

2.2.1 Defer

De webbrowser moet niet wachten op het script. De webbrowser mag de HTML-pagina al verwerken. Het script laad in de achtergrond en wordt uitgevoerd zodra de HTML-pagina verwerkt is. Defer zorgt ervoor dat, ongeacht de locatie van het script, er geen blokkade is.

HTML-code

```
<script defer src="js/apartscriptbestand.js"></script>
```

2.2.2 Async

*H*et script wordt asynchroon (tegelijkertijd) met de HTML-pagina gedownload en uitgevoerd. Het 'async' attribuut maakt het script volledig onafhankelijk. Het script wacht niet op eventuele andere scripts of op de HTML-pagina. Het script start wanneer het is ingeladen.

HTML-code

```
<script async src="js/apartscriptbestand.js"></script>
```

2.3 Javascript output

Javascript kan op verschillende manieren data 'tonen', al dan niet zichtbaar voor de bezoeker van de website. Volgende selectie van mogelijkheden gaan we regelmatig gebruiken doorheen deze cursus:

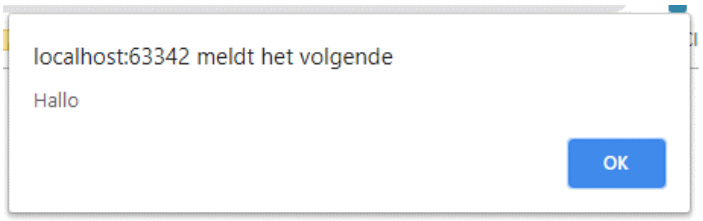
- In een alert-box, via 'alert()'.
- In een browser console, via 'console.log()'.
- In een HTML-element, via 'innerHTML'.

2.3.1 Via alert()

Via de alert() methode kunnen we data weergeven in een alert-venster,

JS-code

```
alert("Hallo");
```

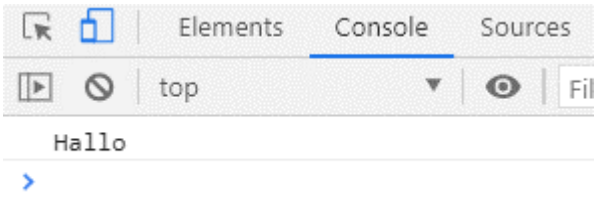


2.3.2 Via console.log()

De console.log() methode is ideaal om te debuggen. De data is niet meteen zichtbaar voor de bezoeker van de webpagina, maar is wel te raadplegen via de DevTools (tab 'Console').

JS-code

```
console.log("Hallo");
```



2.3.3 Via innerHTML

Via 'innerHTML' kunnen we inhoud toevoegen aan een HTML-element.

HTML-code

```
<p id="inhoud"></p>
```

JS-code

```
document.getElementById("inhoud").innerHTML = "Hallo";
```

Om toegang te krijgen tot een HTML-element in Javascript, geven we het element eerst een 'id' in de HTML-code.

2.4 Commentaar in Javascript

We kunnen in Javascript op twee manieren commentaar toevoegen:

- Op één regel.
- Op meerdere regels.

JS-code

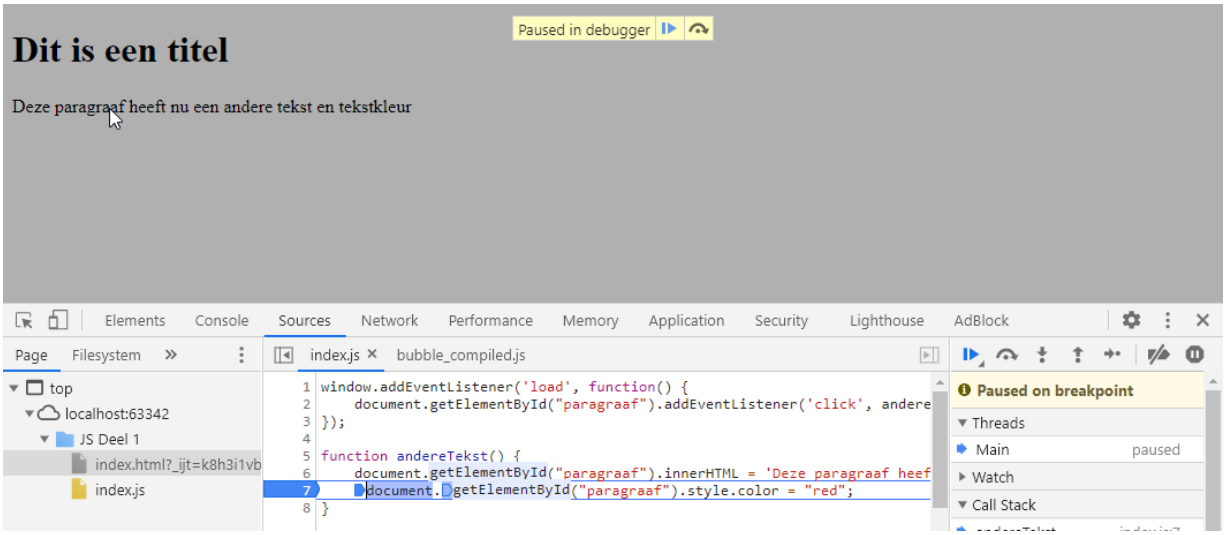
```
// Dit is commentaar op één regel.
/* Dit is commentaar over
   meerdere regels heen */
```

2.5 Javascript Debuggen

Met een debugger kunnen we fouten en onregelmatigheden in onze code eenvoudig opsporen.

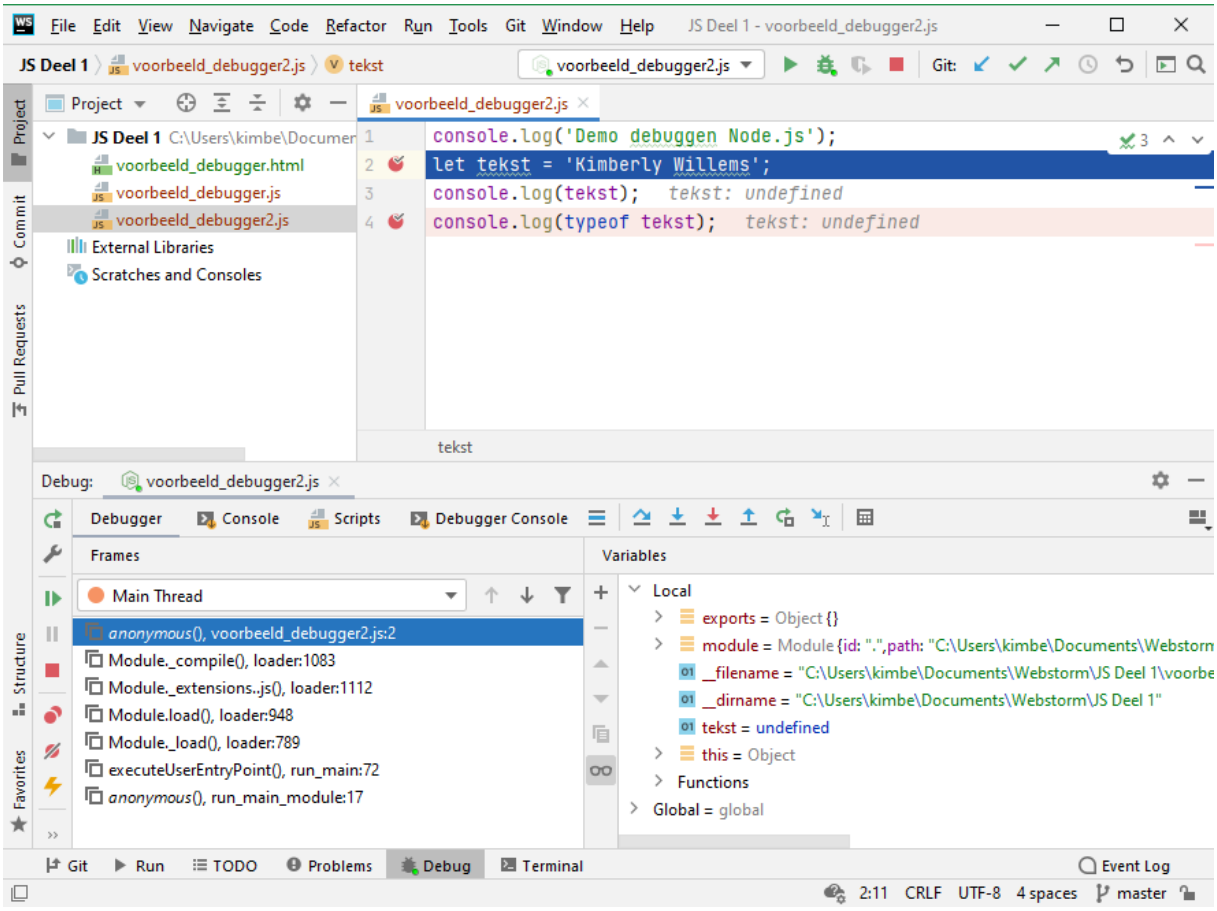
2.5.1 In Google Chrome via de HTML-pagina

1. We openen onze webpagina in Webstorm waaraan ons script is gelinkt.
2. We klikken op de knop 'Run' in Webstorm. Google Chrome opent de webpagina.
3. We openen onze Chrome DevTools.
4. We gaan in de Chrome DevTools naar het tabblad 'Sources'.
5. We openen ons script in de Chrome DevTools.
6. We plaatsen een breekpunt op de plaats waar we een fout verwachten.
7. We laten de debugger het script uitvoeren. Wanneer de debugger de code inleest, stopt deze automatisch bij het breekpunt. We kunnen na de stop de voorgaande stappen inspecteren en de code stapsgewijs verder laten gaan om zo de fout te achterhalen.



2.5.2 In Webstorm via Node.js

1. We openen ons script in Webstorm.
2. We plaatsen een breekpunt op de plaats waar we een fout verwachten.
3. We klikken op de knop 'Debug' in Webstorm. Onderaan verschijnt de debugger.
4. We laten de debugger het script uitvoeren. Wanneer de debugger de code inleest, stopt deze automatisch bij het breekpunt. We kunnen na de stop de voorgaande stappen inspecteren of het script verder laten gaan.


File Edit View Navigate Code Refactor Run Tools Git Window Help
JS Deel 1 - voorbeeld_debugger2.js
JS Deel 1 voorbeeld_debugger2.js tekst
voorbeeld_debugger2.js
Git:
Project
JS Deel 1 C:\Users\kimbe\Documer
voorbeeld_debugger.html
voorbeeld_debugger.js
voorbeeld_debugger2.js
External Libraries
Scratches and Consoles
1 console.log('Demo debuggen Node.js');
2 let tekst = 'Kimberly Willems';
3 console.log(tekst); tekst: undefined
4 console.log(typeof tekst); tekst: undefined
tekst
Debug: voorbeeld_debugger2.js
Debugger Console Scripts Debugger Console
Frames
Main Thread
anonymous(), voorbeeld_debugger2.js:2
Module._compile(), loader:1083
Module._extensions..js(), loader:1112
Module.load(), loader:948
Module._load(), loader:789
executeUserEntryPoint(), run_main:72
anonymous(), run_main_module:17
Variables
Local
exports = Object {}
module = Module {id: ".",path: "C:\Users\kimbe\Documents\Webstorm
__filename = "C:\Users\kimbe\Documents\Webstorm\JS Deel 1\voorbe
__dirname = "C:\Users\kimbe\Documents\Webstorm\JS Deel 1"
tekst = undefined
this = Object
Functions
Global = global
Git Run TODO Problems Debug Terminal
Event Log
2:11 CRLF UTF-8 4 spaces master