> *Java is to JavaScript as ham is to hamster.*
> —Jeremy Keith

Inleiding JavaScript (JS)

1 Wat is JS?

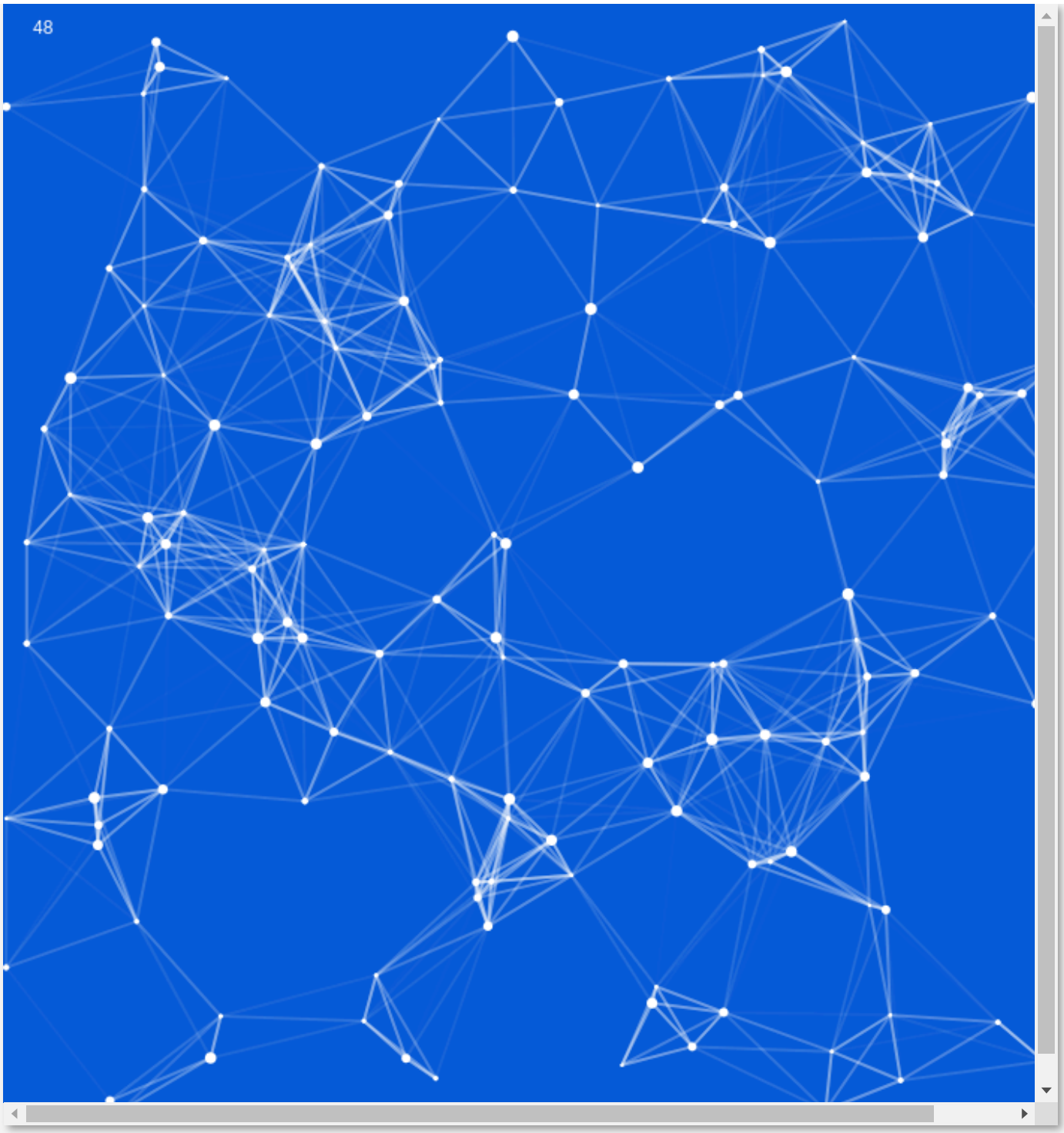
JS is een programmeertaal die ingebouwd zit in elke browser en interactie met een webpagina mogelijk maakt. Vroeger was de taal een client-side scripting taal, wat wil zeggen dat ze dus enkel in de browser werkte. Nu kan je JS (via Node) ook op een server draaien. Je hebt dus niet noodzakelijk nog een browser nodig.

In deze beperkte kennismaking voor dit OPO zullen we ons beperken tot JS in de browser. In het vervolgOPO zul je meer van nabij met JS te maken krijgen.

JS is een scriptingtaal. De browser ontvangt een script (kan zowel in de HTML-code zelf staan als een extern bestand zijn), leest en ontcijfert het (‘parsing’). Daarna converteert de JS-engine van de browser het script naar machinetaal en voert het uit. Je zou denken dat JS door het scriptingkarakter een trage taal is, maar dat is helemaal niet meer het geval.

Onderstaande ‘sketch’ (gemaakt in een JS framework p5JS) berekent 60 keer per seconde (fps) de afstand van 150 objecten tot de 149 andere. Als deze afstand binnen een bepaalde grens ligt, wordt er een verbindingslijn getekend die des te feller is naarmate de afstand tussen beide kleiner is.

In de linkerbovenhoek zie je een getal. Haalt je browser de 60 fps? Dat kan sterk afhangen van browser tot browser. Op mijn computer is Safari de snelste, gevolgd door chrome / edge en tenslotte firefox (die zelfs geen 30 fps haalt).



Just for fun: hieronder vind je de code voor deze sketch:

```
let particles = []
const particleAantal = 150 // mag natuurlijk ook met let of var
const minDist = 120
const vmax = 2
const maxstraal = 4

function setup () {
  createCanvas(750, 750)
}

function draw () {
```

```
  background(8, 90, 215)
  fill(255)
  text(Math.round(frameRate()), 20, 20)
  strokeWeight(2)

  for (let i = particles.length - 1; i >= 0; i--) {
    particles[i].update() // bereken nieuwe positie aan de hand van huidige positie en
    particles[i].toon() // teken het i-de partikel als een kleine cirkel
    if (particles[i].positie.x > width || particles[i].positie.x < 0 || particles[i].p
      particles.splice(i, 1) // verwijder element uit array want partikel is buiten wi
    }
  }

  while (particles.length < particleAantal) {
    particles.push(new Particle()) // maak nieuwe partikels aan tot het aantal = parti
  }

  noFill()
  for (let i = particles.length - 1; i >= 0; i--) {
    let p1 = particles[i]
    for (let j = particles.length - 1; j >= 0; j--) {
      if (i !== j) { // enkel twee verschillende partikels vergelijken
        let p2 = particles[j]
        let d = abs(dist(p1.positie.x, p1.positie.y, p2.positie.x, p2.positie.y))
        if (d <= minDist) { // partikel i en j zijn dicht genoeg bij elkaar om lijnstu
          stroke(255, map(d, 0, minDist, 100, 0))
          line(p1.positie.x, p1.positie.y, p2.positie.x, p2.positie.y)
        }
      }
    }
  }
}

// Constructor voor nieuw particle object
function Particle () {
  this.positie = createVector(random(0, width), random(0, height))
  this.snelheid = createVector(random(-vmax, vmax), random(-vmax, vmax))
  this.straal = random(1, maxstraal)
}

// had ook binnen de constructor kunnen staan met this.update()
Particle.prototype.update = function () {
  this.positie.add(this.snelheid)
}

// had ook binnen constructor kunnen staan met this.toon()
Particle.prototype.toon = function () {
  noStroke()
  fill(255)
```

```
  circle(this.positie.x, this.positie.y, this.straal)
}
```

2 Wat kan JS doen?

Op de site die je nu leest gebruik ik JS voor:

- Het automatisch nummeren van alle titels `h2` en `h3`;
- het genereren van een automatische inhoudsopgave op basis van deze titels;
- bijhouden op welke pagina we zijn in de navigatie, zodat CSS de oranje kleur kan toepassen via een `class`;
- het tonen en verbergen van het antwoord bij een kleine oefening.

Dat zijn enkele voorbeelden van heel kleine scriptjes (vaak slechts enkele tientallen regels lang. De sketch hierboven met de 150 partikels maakt gebruik van het framework p5JS, dat gemakkelijk uit 100.000 regels JS code bestaat.

De editor die we voorstelden om te gebruiken (Visual Studio Code, zie tools) is volledig in HTML, CSS en JS geschreven. Wat je waarschijnlijk niet besefte bij het eerder gebruik van VS Code is dat je eigenlijk in een webpagina bezig bent (via een framework dat electron heet)!

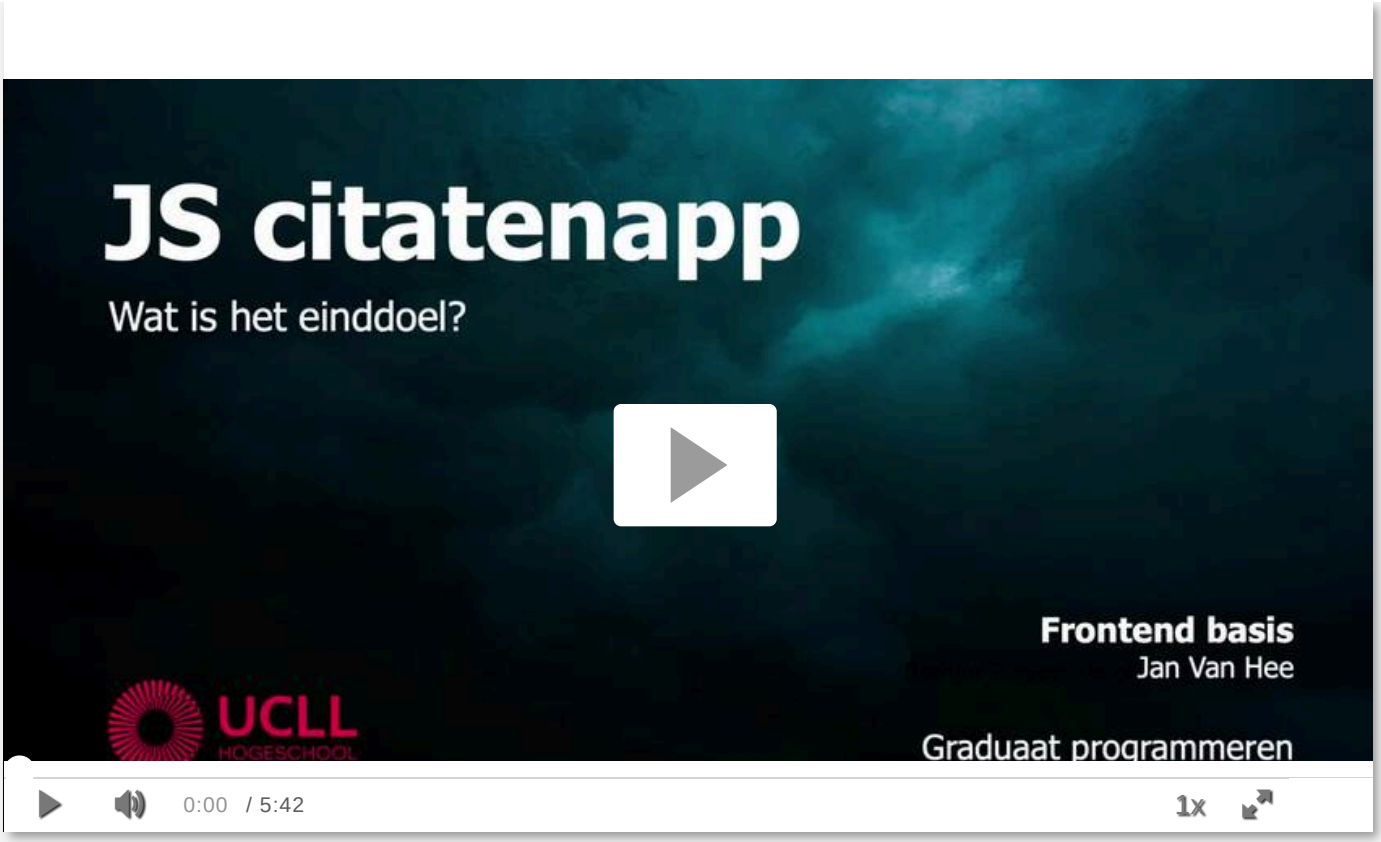
3 Project module 2: citatenapp

In module 1 van dit OPO maakte je een statische site. De klemtoon lag er op HTML, CSS, RWD, bruikbaarheid, toegankelijkheid en een beetje op vormgeving.

De opdracht voor module 2 is een JS-app maken. Vanzelfsprekend gebruik je nog steeds correcte HTML en CSS, maar de klemtoon verschuift naar JS. We leggen eerst alle technieken die je nodig hebt uit m.b.v. een volledig uitgewerkt voorbeeld: een citatenapp.

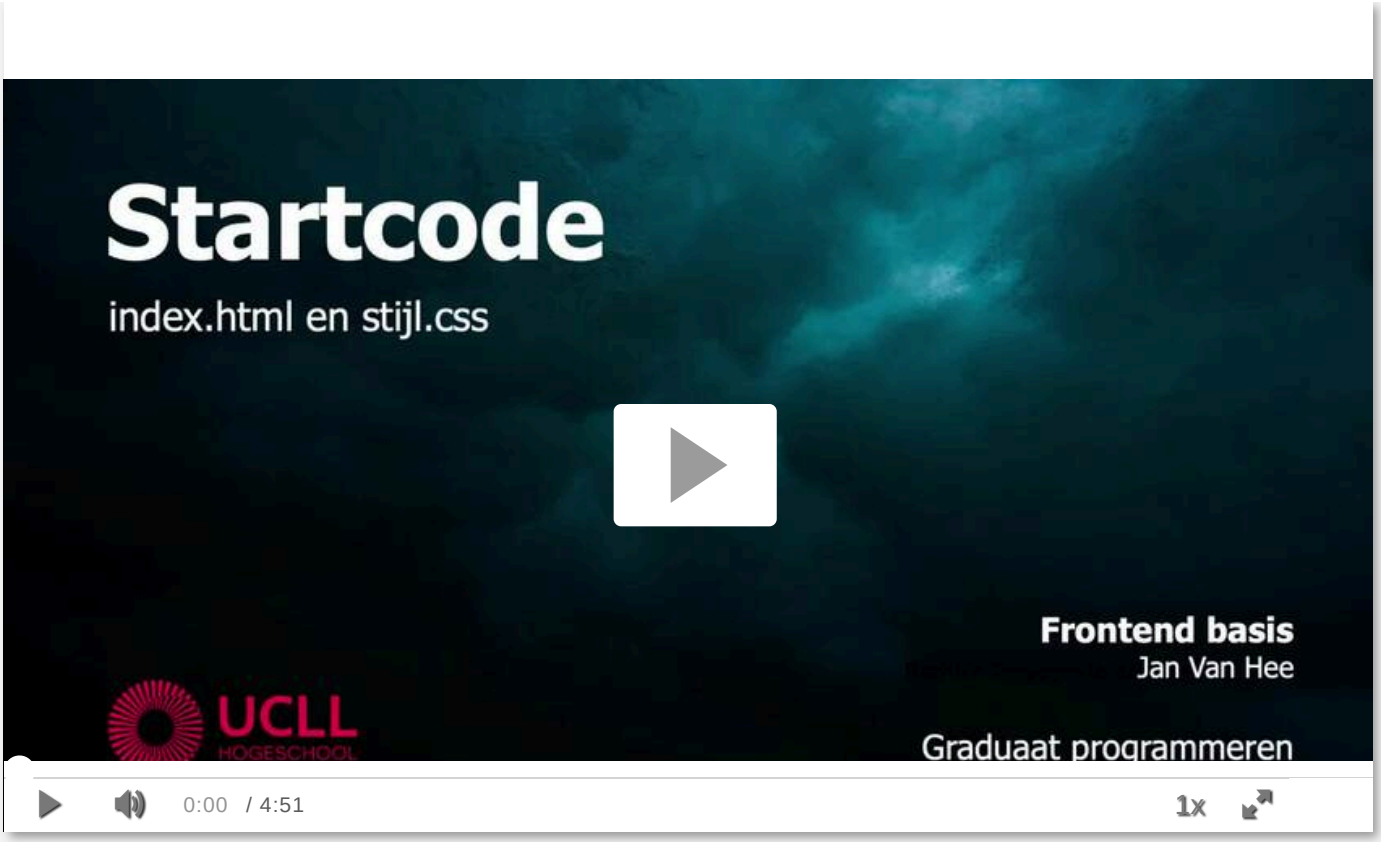
Deze applicatie telt twee pagina's en is geschreven in HTML, CSS en JS. Ze laat toe om beroemde citaten te beheren. De gebruiker kan citaten toevoegen, verwijderen, zoeken op woorden in een citaat enz.

 Onderstaand filmpje toont het einddoel dat je in heel wat kleine stapjes probeert zelf te bouwen. Dit project is opgevat als een doe-tekst.



De volledige opdracht voor module 2 vind je op Toledo. Daar staat ook een beetje startcode die je kan downloaden. Download die startcode en bekijk ze.

Het volgende filmpje overloopt die startcode: index.html, stijl.css, mapje ‘img’ met het logo en een favicon.

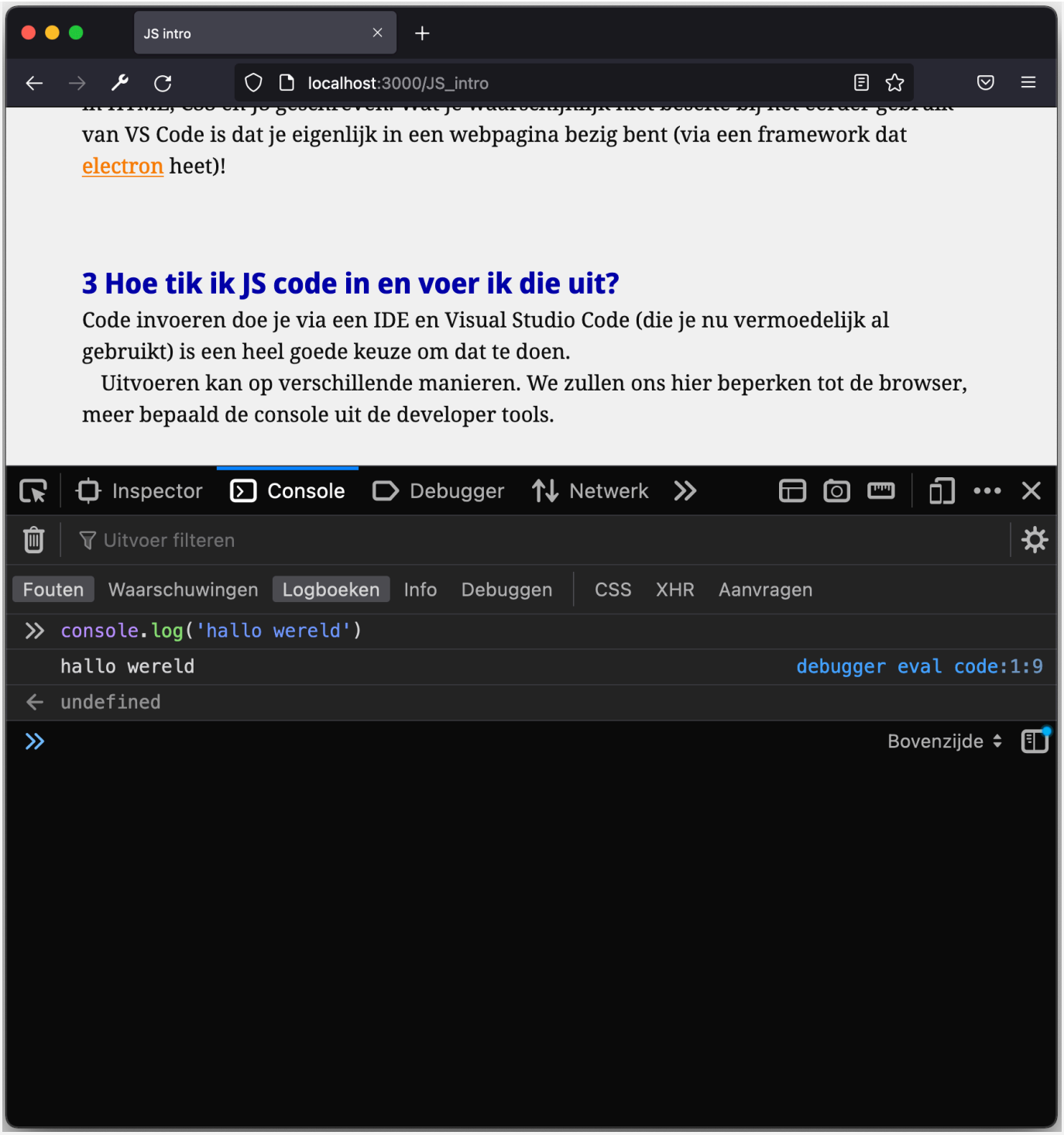


4 Hoe tik ik JS code in en voer ik die uit?

Code *invoeren* doe je via een IDE. Visual Studio Code (die je nu vermoedelijk al gebruikt) is een goede keuze om dat te doen.

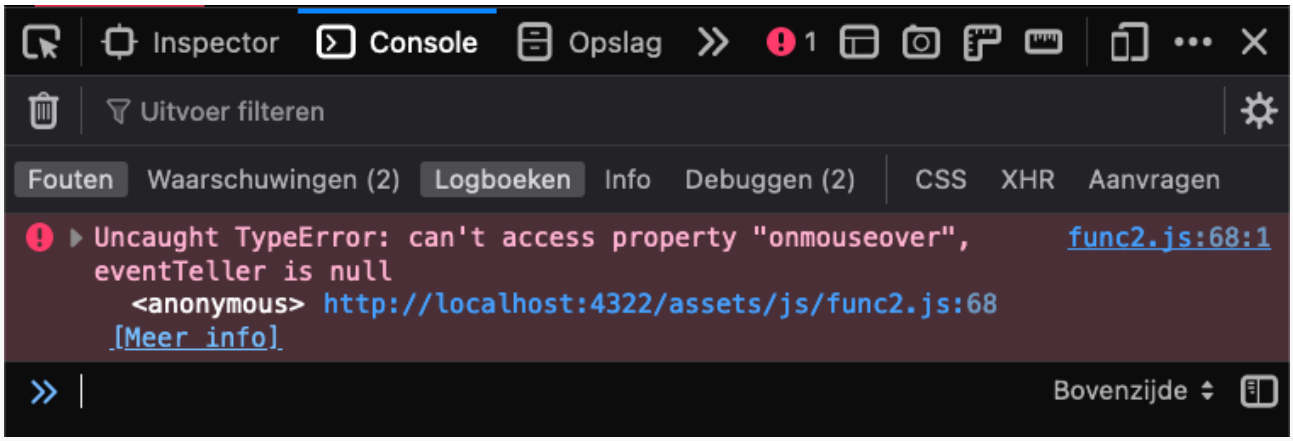
Uitvoeren kan op verschillende manieren. We zullen ons hier beperken tot de browser, meer bepaald de console in de developer tools.

Je opent de developer tools door rechts te klikken op een vrije ruimte in een webpagina en ‘inspecteren’ te kiezen. Klik dan op het tabblad ‘Console’. Dit wordt de omgeving waarin we JS kunnen uittesten. Typ nu in deze console `console.log('hallo wereld')` en bevestig met enter.



Normaal gezien zullen we echter altijd code intikken in VS code. De plaats om op zoek te gaan naar fouten, uitvoer te bekijken enz is echter wel deze console.

Het is een goed idee om tijdens het ontwikkelen in JS **altijd de console open te laten staan** , zodat je direct kan zien of er iets fout is en waar die fout dan zit. Onderstaande afbeelding toont dat de variabele `eventTeller` de waarde ‘null’ heeft (dus eigenlijk geen waarde), waardoor de eigenschap `onmouseover` een fout geeft (op lijn 68 in het bestand ‘func2.js’).



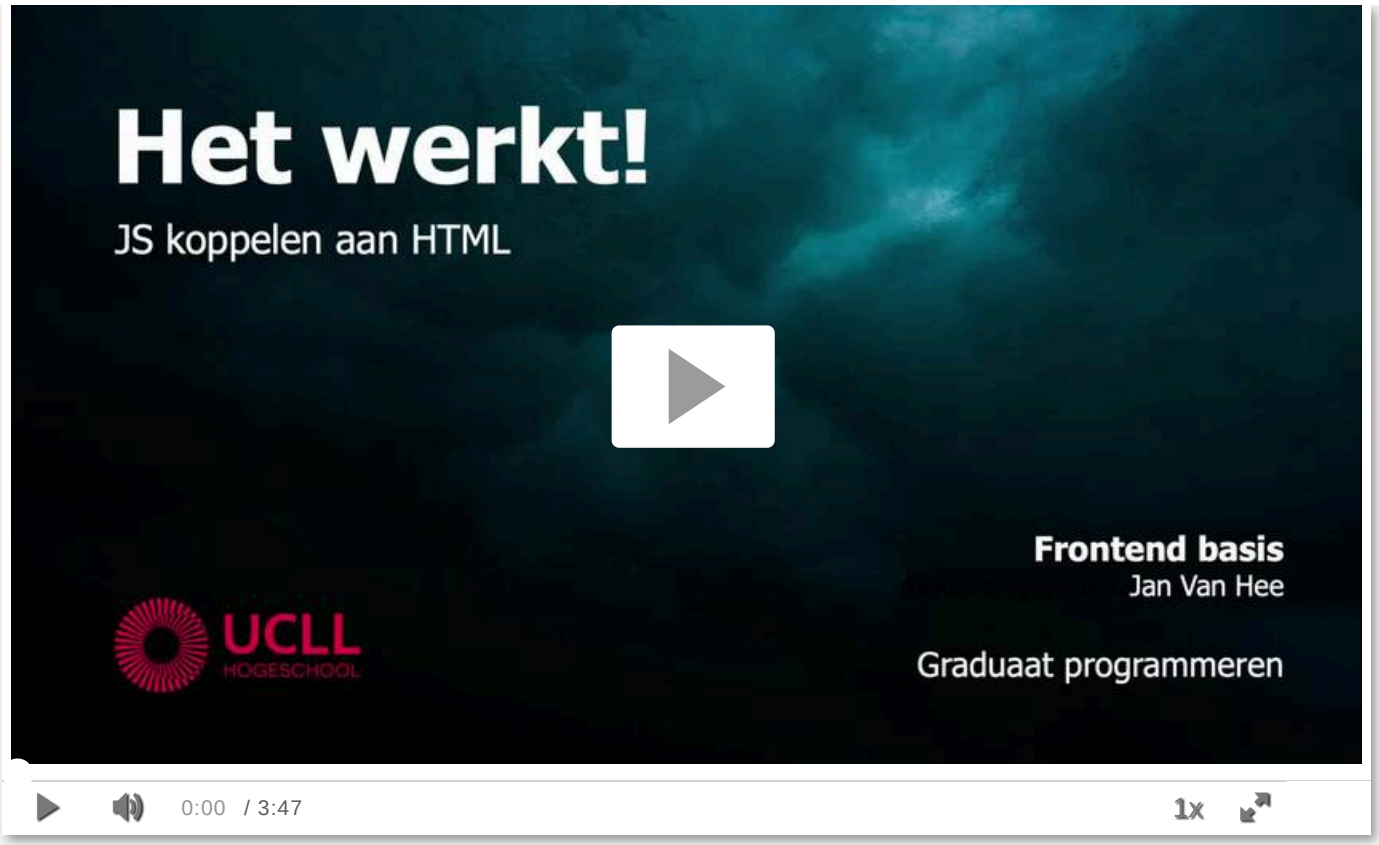
5 JS koppelen aan HTML

Het is een goede gewoonte om je JS-code in een apart bestand (of meerdere bestanden als je modulair wilt werken) te zetten, met de extensie .js (of eventueel .mjs in het geval van modules). Je laadt dit bestand met het `script` element in HTML. Dat kan in de `head` van het HTML-document ofwel in `body`, meestal vlak voor de sluittag `</body>`.

De hoofdreden om het onderaan de `body` te zetten is dat *eerst de volledige HTML moet geladen worden*. De JS gaat namelijk meestal iets wijzigen aan de HTML, en dat kan dus pas als die eerst volledig geladen is!

Je kan echter in de `head` iets soortgelijks bereiken door gebruik te maken van het `defer` attribuut. Dit vertelt de browser dat er moet gewacht worden met het *uitvoeren* van het JS-script tot het volledige HTML-bestand verwerkt ('geparsed') is.

Onderstaand filmpje toont een tweetal manieren om JS-code toe te voegen aan je applicatie: ofwel in het HTML-bestand zelf, ofwel – meestal de betere optie – in een extern bestand. Begin je JS-code altijd met een kleine test in de console ('Het werkt!').



↑ Naar top