[JS: Synchronous vs Asynchronous JavaScript](https://www.notion.so/419d19d6d31743b88890d38ac8caf627)

**Leerdoelen:**

* **Ik kan uitleggen op welke manier JavaScript Synchronous is.**

JS, werkt de opdrachten van boven naar beneden af, dus wat eerst geschreven is wordt   
 ook als eerste uitgevoerd, dan nummer 2, dan 3 etc.

* **Ik kan uitleggen waarom je soms Asynchronous JavaScript nodig zal hebben.**

Wanneer je bijvoorbeeld meerdere opdrachten achter elkaar geeft (ga naar deze webpagina,  
 open die foto, ga naar die andere webpagina  
 dan zou als js synchroon is moeten wachten totdat eerst 1 is uitgevoerd , dan 2 en dan 3  
 de ene opdracht kost meer tijd dan de andere,  
  
 asynchroon, betekend dat opdracht 1 wordt uitgevoerd, en ondertussen dat de computer aan het denken is wordt ook opdracht 2 uitgevoerd en dit geldt ook voor opdracht 3  
 wanneer opdracht 2 eerder klaar is dan 1 voert js dit uit en wanneer opdracht 1 klaar is voert

js deze ook uit, zo ook voor 3 waardoor je niet hoeft te “wachten” totdat eerst 1 klaar is, dan   
 2 en dan 3.

* **Ik kan zelf ook een** [**setTimeout()**](https://www.w3schools.com/jsref/met_win_settimeout.asp) **gebruiken, om async JavaScript te testen**

**setTimeout()** betekend; ik wil deze functie uitvoeren, maar wel een paar msec. In de   
toekomst

vb  
ik geef de pc de opdracht voor een geolocationcall (dus vertel me waar ik ben, geef me de coordinaten)

het antwoord krijg je niet meteen en de volgende code die geschreven is weet hier de uitkomst nog niet van,

ik heb wel een callbackfunction maar deze weet dus niet wanneer het antwoord terug komt

dus er zijn verschillende opdrachten (bv naar servers voor informatie) die je kan geven,   
 alleen je weet niet wanneer je het antwoord terug krijgt.  
 Dus de volgende code heeft nog geen waarde terug gekregen en kan nog nix

Dus zegt js, ik ga vast naar de volgende opdracht en wanneer ik het antwoord terug krijg  
 dan voer ik die uit, maar tot die tijd ga ik verder.

Vb

**Timeout function = asynchonouses function**

|  |  |
| --- | --- |
| let a3 = 100   log (a3); | Hier wodt 100 meteen gelogt op de console |
| let a3 = 100 **setTimeout (**function () {a3++}, 1**)**  log (a3)  let a3 = 100 **setTimeout (**function () {a3++; log (a3)}, 1**)**  log (a3) | Als de code synchroon is dan wordt deze eerst uitgevoerd  Als de code asynchroon is dan wordt eerst log(a3) uitgevoerd en dan pas **setTimeout**  **dus: setTimeout wordt nu 1 ms later uitgevoerd dan log(a3)**  **dus op console nu:** eerst 100 , want dat is de log(a3)  dan  101, want deze funtie wacht 1 ms seconde, logt log(a3) en voert dan a3++; log (a3) uit en dat is 101 |
| let a3 = 100 **setTimeout (**function () {a3++; log (a3)}, 1**)**  log (a3) **setTimeout (**function () {log (a3)}, 1**)** | **dus op console nu:**  100;  -want dat is log (a3),  dan  **setTimeout (**function () {a3++; log (a3)}, 1**)** -want deze functie is als eerste geschreven (let op, ze hebben wel dezelfde tijd gekregen nl 1ms), dan **setTimeout (**function () {log (a3)}, 1**)** -wantdeze function is als tweede geschreven |

**promises = asynchonouses function**

|  |  |
| --- | --- |
| Let p = new Promise (function(){  }); | Lege promise function |
| Let p = new Promise (function(){  });  log(p); | **op console nu:**  *Promise*  en nog voor de 101 (vd setTimeout function)  waarom?  Er komt n keer een waarde terug, maar weet niet wanneer, op dit moment een lege waarde gelogt, maar zodra ik antwoord heb gekregen dan log ik dit.  **Dan setTimeout** uitvoeren  dus op console: Promise (<pending>) **betekend in afwachting** |
| Let p = new Promise (function(resolve, reject){  **resolve(“Yo”);**  });  log(p); | **op console nu:**  *promise (<resolve: “Yo”) meteen op scherm omdat geen vertraging ingesteld.* |
| Let p = new Promise (function(resolve, reject){  **setTimeout(function(){**    **resolve(“Yo”);  }, 0)**    });  log(p); | Nog steeds promise in afwachting |
|  | Alles staat op 0 |
| let a3 = 100 **setTimeout (**function () {a3++; log (a3)}, 0**)**  log (a3) **setTimeout (**function () {log (a3)}, 0**)**  Let p = **new Promise** (function(resolve, reject){  setTimeout(function(){    resolve(“Yo”);  }, 0)    });  log(p); |  |
|  |  |
| let a3 = 100 **setTimeout (**function () {a3++; log (a3)}, 0**)**  log (a3) **setTimeout (**function () {log (a3)}, 0**)**  Let p = **new Promise** (function(resolve, reject){  setTimeout(function(){    resolve(“Yo”);  }, 0)    });  log(p);  setTimeOut (function() {  log (p);  } , **10**) | **Nu op console**  *promise (<pending.)*  *101*  *promise (<resolve: “Yo”)*  *eventueel terug kijken filmpje: va 13:00min* |

[**JS: Callbacks**](https://www.notion.so/ce2910c809e64472ae32b94f0ea61b90)

Een callback is overigens een algemeen JavaScript concept: het is simpelweg een functie die pas uitgevoerd wordt nadat een andere functie is uitgevoerd.

**Leerdoelen:**

* **Ik begrijp wat een Callback is.**het is een functie die pas uitgevoerd wordt **nadat** een andere functie is uitgevoerd.
* **Ik begrijp waarom je een Callback nodig hebt om data asynchronous op te vragen**

* **Ik begrijp de basisprincipes hoe Callbacks gebruikt worden**
* **Ik weet hoe je een Promise maakt, maar belangrijker: hoe je een Promsise "gebruikt"**

**tip:**

Zoals al aangegeven door de docent in de video is data opvragen via een Callback dus net als de Starbucks: je omschrijft heel duidelijk wat je wilt hebben, geeft je naam op, en sluit achteraan in de rij. Als ze je bekertje kwijt zijn dan zijn ze ook je order kwijt. *Dus als er bij het laden iets mis gaat, kan het zijn dat de data niet (op tijd) aankomt waar je verwacht.*

|  |  |
| --- | --- |
| Const posts =[  {title: 'Post One', body: 'This is post one'},  {title: 'Post Two', body: 'This is post two'},  ];  function getPosts () {  setTimeout (function() {  }, 1000);  } | Array  function getPost  setTimeout function  1000 ms = 1 sec |
| **Bovenstaande kan je ook schrijven als**  Const posts =[  {title: 'Post One', body: 'This is post one'},  {title: 'Post Two', body: 'This is post two'},  ];  function getPosts () {  setTimeout (() **=>** {  }, 1000);  } | Arrow => ipv function |
|  |  |
| Const posts =[  {title: 'Post One', body: 'This is post one'},  {title: 'Post Two', body: 'This is post two'},  ];  function getPosts () {  setTimeout (() **=>** {  let output = ' ';  posts.**forEach( (post, index) => {**  **output += `<li>${post.title}</li> ` ;**  **});**  document.body.innerHTML = output;  }, 1000);  }  getPosts(); | Dus vertragen met 1 sec door setTimeout  Door array loopen met forEach, met een callback  Doel is om de Post op de pagina te zetten, dus we maken een variabele met de naam output, en zonder waarde ' '  dan willen we door de posts loopen, dit doen we met **posts.forEach ();**  in posts.forEach kan dan een callback in of een function (hier gebruiken we een arrow function  =>  dan als parameters nemen we post en index, maar dit kan vanalles zijn, maar nu naam post want dat doen we ook en een index  dan willen we voor elke post een output variabele  dus output +=  dan ` ` zodat we template sntax kunne gebruiken  we stoppen er <li> in  met posttitle <li>${post}</li>  **$ {} zodat we een variabele erin kunnen stoppen**  **${post} komt uit de forEach ((post)**  dus we ittereren van uit de array en pakken de title  dan **na** de forEach  dan **na** de forEach  willen we het in de body inserten  dan function callen met getPosts();  Dan na een seconde worden de twee post geladen  dus we krijgen de post van de array  we loopen erdoor  stoppen in de output variabele  dan outputten op de pagina |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |