Javascript les 6

(en een klein beetje Max)

debuggen

fouten in je code herkennen, opsporen en voorkomen

vaak voorkomende bugs

- typefouten
- scope
- out of index
- volgorde
- teveel of te weinig ;
- = vs == vs ===

typefouten

een typefoutje is snel gemaakt maar kan in sommige gevallen lastig te vinden zijn

```
elipse();
playsound();

ellipse();
playSound();
```

scope

waar in de code maak je een variabele?

```
function setup() {
  let waarde = 10;
}

function draw() {
    console.log(waarde);
}
```

‘waarde’ kan hier niet gevonden worden

out of index

een waarde uit een array halen die niet bestaat

```
let lijst = [1,2,3,4,5]
```

een array met 5 items

```
console.log(lijst[5]);
```

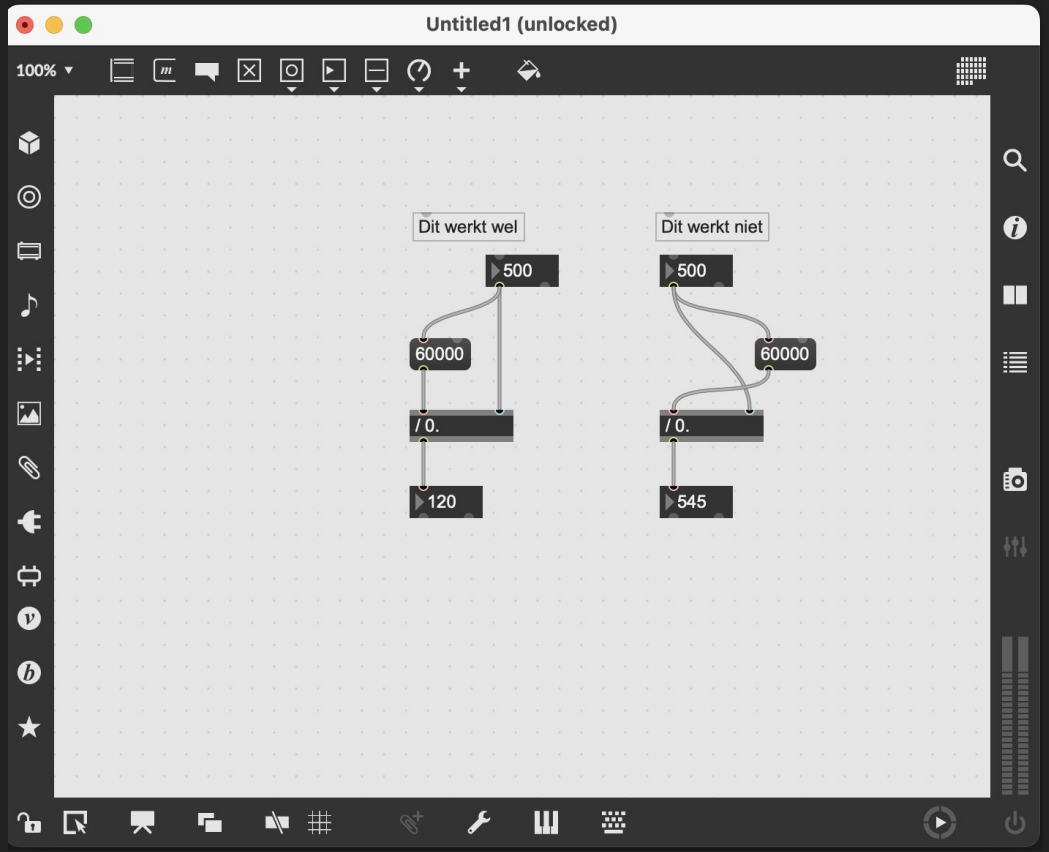
element 5 bestaat niet, array begint met tellen bij 0

volgorde

met name voor kleur, lijndikte etc. belangrijk

```
function draw() {
  ellipse(100,100,50);
  fill('red');
  ellipse(200,200,20);
  fill('yellow');
}
```

volgorde in Max



te veel of te weinig ;

gaat bijna nooit fout, maar hieronder wel

```
let x = 0;
if (x === 0) {;
  console.log(“hoi”);
}
```

het if-statement wordt afgebroken na regel 2

= vs == vs ===

= is gelijkstellen aan
== is vragen of iets (ongeveer) gelijk is
=== is vragen of iets strict gelijk is

```
let x = 0;
if (x = 1) {
  //altijd waar
  //want x wordt 1
}
```

= vs == vs ===

= is gelijkstellen aan
== is vragen of iets (ongeveer) gelijk is
=== is vragen of iets strict gelijk is

```
let x = 0;
if (x == 1) {
  //niet waar
  //want x is 0
}
```

= vs == vs ===

= is gelijkstellen aan
== is vragen of iets (ongeveer) gelijk is
=== is vragen of iets strict gelijk is

```
let x = 0;
if (x === '0') {
  //niet waar
  //want x is 0 en niet '0'
}
```

= vs == vs ===

= is gelijkstellen aan
== is vragen of iets (ongeveer) gelijk is
=== is vragen of iets strict gelijk is

```
let x = 0;
if (x == '0') {
  //waar
  //want x is 0 en dat
  //lijkt ongeveer op '0'
}
```

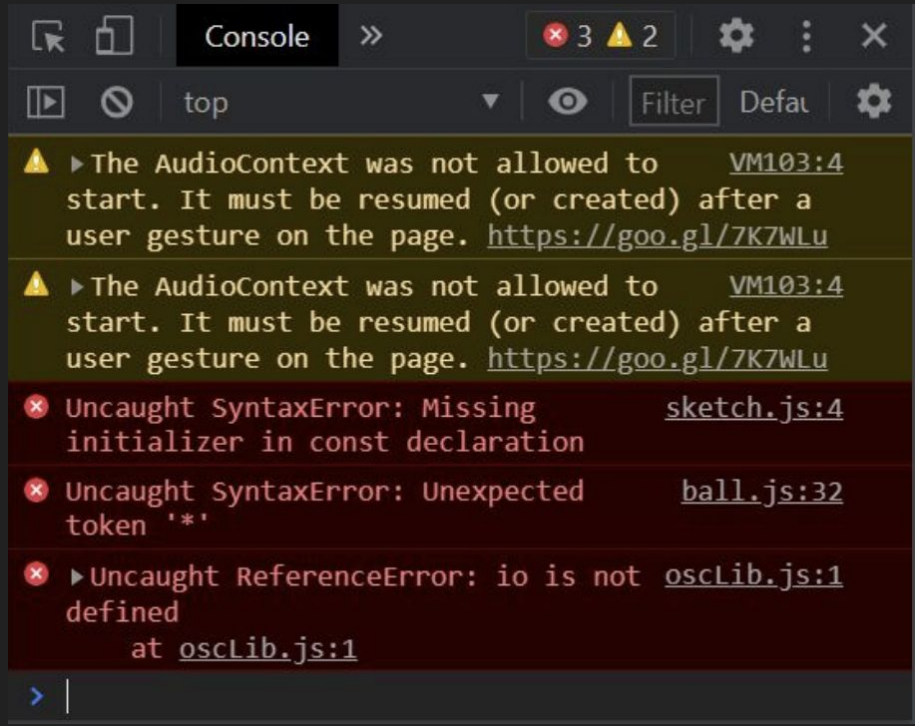
strategieën om te debuggen

- console / Max-window
  - console.log(); / print
  - event probe (Max)
- code isoleren
  - //
- peer review
  - rubber ducking
- hulpbronnen
- heel lang staren

console

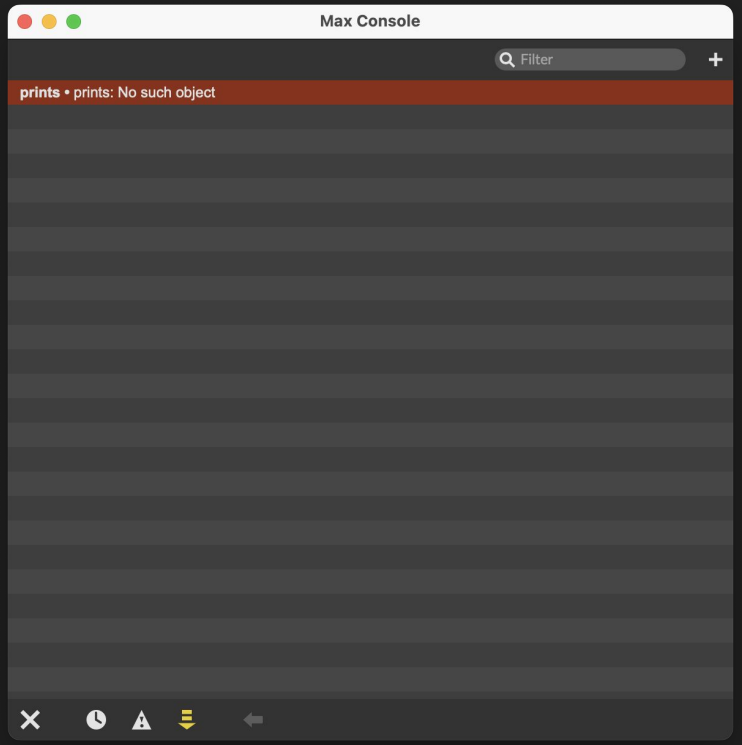
foutmeldingen (incl. regelnummer)

- regelnummer geeft aan waar de fout optreedt
- dit is niet altijd waar de fout ook daadwerkelijk zit



Max-window

foutmeldingen in het rood
- klik op de melding om het betreffende object te highlighten in Max



console.log / print

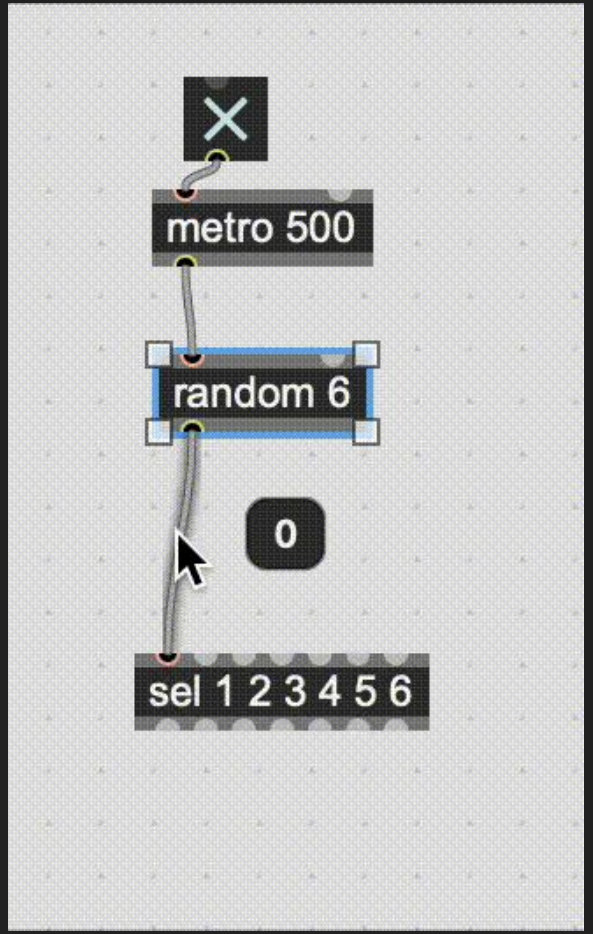
gebruik de logging-functie of het print-object om bepaalde waarden en uitkomsten naar de console te sturen.

zo zie je waar welke waarde uitkomt en of deze is wat jij verwacht.

event probe

gebruik event probes om te zien
wat er uit een lijn komt

zet aan via debug -> Event Probe



code isoleren

gebruik commentaar om stukken code tijdelijk uit te zetten.

- treedt de bug nu nog steeds op?
- is het gedrag anders?
- verandert er eigenlijk helemaal niets?

peer-review

laat iemand meekijken met wat er fout gaat, misschien ziet deze persoon wat er misgaat.

door het probleem te bespreken kom je soms al tot een nieuw inzicht.

zelfs als je het niet met een echt persoon bespreekt.

maar met een `rubber duck`.

hulpbronnen

als je een specifieke foutmelding hebt, maar deze niet begrijpt, zoek dan op taal-gerelateerde fora

zonder foutmelding kan je het ook online proberen, maar heb je een goede formulering nodig.

heel lang staren

kan soms helpen, maar wordt bij grote stukken code snel heel inefficiënt.

bugs voorkomen

- secuur werken
  - spellingcheck
- logische benaming
  - variabelen
  - functies
- opsplitsen van code
- indentatie

secuur werken

- kijk goed of alles wat je typt goed gespeld is
- werk kleine stukjes code uit, werk niet van de hak op de tak
- pas je iets ergens aan, doe dat meteen overal
- gebruik een spelling-checker (vscode)

logische benaming

- gebruik zinvolle namen voor variabelen en functies
- wees consistent
- niet te lang, niet te kort

opsplitsen van code

- verdeel lange regels code over meerdere regels
- maak eventueel een extra variabele aan
- behoud zo het overzicht

indentatie

- zorg voor een inspringing na elke {
- zo is duidelijk welke code bij welk code-block hoort

indentatie

```
function mousePressed() {
if (mouseX > width / 2) {
x = mouseX;
y = mouseY;
}else {
y = mouseX;
x = mouseY;
}
```

indentatie

```
function mousePressed() {
   if (mouseX > width / 2) {
      x = mouseX;
      y = mouseY;
   }
   else {
      y = mouseX;
      x = mouseY;
   }
```

Opdracht