# 1 Controlestructuren

Controlestructuren zorgen ervoor dat er in een programma beslissingen worden genomen en berekeningen worden uitgevoerd.

## 1.1 If – else

Deze controlestructuur laat toe om op basis van een voorwaarde of conditie een beslissing te nemen. De voorwaarde staat tussen haakjes en maakt gebruik van een vergelijkingsoperator. Ze geeft als resultaat een booleaanse waarde terug. Afhankelijk van het resultaat wordt één of meerdere statements uitgevoerd. Deze statements moet je groepen met accolades.

*JS-code*

```
// Ben je jonger dan 12 jaar, dan krijgt het een Happy Meal. Ben je ouder
   dan 12 jaar, dan krijg je geen Happy Meal.

let leeftijd = 11;
if (leeftijd <= 12) {
      console.log("Je krijgt een Happy Meal.");
}
else {
      console.log("Je krijgt geen Happy Meal.");
}
```

Je kan meerdere voorwaarden samenvoegen met behulp van logische operatoren.

*JS-code*

```
// Je mag een nieuwe gsm kopen als je voldoende hebt gespaard én je
   huidige gsm stuk is. Indien je onvoldoende hebt gespaard of je huidige
   gsm nog perfect werkt, mag je geen nieuwe gsm kopen.

let spaargeld = 800;
let huidigeGsm = "Stuk";
if (spaargeld >= 750 && huidigeGsm == "Stuk") {
      console.log("Je mag een nieuwe gsm kopen.");
}
else {
      console.log("Nog even wachten...");
}
```

Indien er niets moet gebeuren wanneer een voorwaarde 'false' is, kan je het 'else-gedeelte' weglaten in je code. Dit gedeelte is niet verplicht.

*JS-code*

```
// Indien je ouder bent dan 18, krijg je het label '+18'.

let leeftijd = 19;
if (leeftijd >= 18) {
      console.log("Je krijgt het label +18.");
}
```

*Opgelet! In het hoofdstuk 4 hebben we geleerd over de 'voorwaardelijke operator'. Deze speciale operator is een afkorting van de if-else controlestructuur. Je mag zelf kiezen welke schrijfwijze je wenst te gebruiken.*

## 1.2 Switch

De switch-controlestructuur is een veralgemening van de if-else controlestructuur. Deze laat toe om op een overzichtelijke manier verschillende mogelijke beslissingen te evalueren. De switch-controlestructuur is veel leesbaarder en handelbaarder dan een uitgebreide of geneste if-else controlestructuur.

*JS-code*

```
// Ga na welke dag we vandaag zijn.

let dag;
switch (new Date().getDay()) {
  case 0:
    dag = "zondag";
    break;
  case 1:
    dag = "maandag";
    break;
  case 2:
     dag = "dinsdag";
    break;
  case 3:
    dag = "woensdag";
    break;
  case 4:
    dag = "donderdag";
    break;
  case 5:
```

```
    dag = "vrijdag";
    break;
  case 6:
    dag = "zaterdag";
}
console.log(“Vandaag zijn we “ + dag + “.”);
```

We kunnen in onze switch-controlestructuur ook een ‘default’ toevoegen voor ‘alle andere gevallen’.

*JS-code*

```
// Ga na of de bezoeker een koekje wenst en geef een antwoord.

let koekje = prompt("Wil je een koekje", "Ja, Nee");
let uitkomst;
switch (koekje) {
  case "Ja":
    uitkomst = "Joepie, hier is je koekje!";
    break;
  case "Nee":
    uitkomst = "Oké, hap dan maar wat lucht!";
    break;
  default:
     uitkomst = "Sorry, dat was geen duidelijk antwoord...";
}
document.getElementById(“antwoord”).innerHTML = uitkomst;
```

## 1.3 While

Met de While-controlestructuur kunnen we een lus maken. Zolang er aan een voorwaarde wordt voldaan, worden de bijhorende statements uitgevoerd.

*JS-code*

```
// De lus loopt zolang de teller kleiner is dan 10. Per keer verhoogt de
teller met 1.

let teller = 1;
let uitkomst = “”;
while (teller < 10) {
      uitkomst = uitkomst + “<br>De teller staat op “ + teller + “.”;
      teller++;
}
```

```
console.log(uitkomst);
```

## 1.4 Do-while

De do-while controlestructuur is een variant van de while-controlestructuur. Bij deze controlestructuur worden de statements minstens één keer uitgevoerd, ook wanneer de voorwaarde onwaar is.

*JS-code*

```
// De lus loopt zolang de teller kleiner is dan 10. Per keer verhoogt de
teller met 1.

let teller = 15;
let uitkomst = "";
do {
      uitkomst = uitkomst + "<br>De teller staat op " + teller + ".";
      teller++;
}
while (teller < 10);
console.log(uitkomst);
```

## 1.5 For

Met de for-controlestructuur kunnen we nog eenvoudiger werken met tellers. Deze controlestructuur is aangewezen wanneer we het aantal herhalingen op voorhand kennen. Tussen de haakjes komt eerst de initialisatie waarbij de teller op een waarde wordt ingesteld, daarna volgt de voorwaarde en als laatste de definitie van de stappen, meestal een ophoging in de vorm van teller++.

*JS-code*

```
// De lus loopt zolang de teller kleiner is dan 10. Per keer verhoogt de
teller met 1.

let uitkomst = "";
for(teller = 0; teller < 10; teller++) {
      uitkomst = uitkomst + "<br>De teller staat op " + teller + ".";
}
console.log(uitkomst);
```

# 2 Arrays

Een variabele kan telkens maar één waarde bevatten, maar soms is het nuttig om bepaalde variabelen te koppelen in een reeks van gerelateerde data. In Javascript kunnen we hiervoor een array of reeksvariabele voor gebruiken.

Een array of reeksvariabele is een geordende verzameling van elementen. Elk element heeft een genummerde positie. Het eerste element staat op positie nul en alle genummerde posities samen vormen de index.

## 2.1 Arrays declareren

Het declareren van een array kan op verschillende manieren.

*JS-code*

```
// Een lege reeksvariabele.
let leeg = new Array();


// Een lege reeksvariabele met plaats voor 5 elementen.
let werkdag = new Array(5);


// Een lege array met plaats voor 5 elementen die erna gevuld wordt.
let werkdag = new Array(5);
werkdag[0] = "Maandag";
werkdag[1] = "Dinsdag";
werkdag[2] = "Woensdag";
werkdag[3] = "Donderdag";
werkdag[4] = "Vrijdag";


// Een array met de vijf werkdagen als elementen.
let werkdag = new Array("Maandag", "Dinsdag", "Woensdag", "Donderdag",
"Vrijdag");


// Een array met de vijf werkdagen als elementen.
let werkdag = ["Maandag", "Dinsdag", "Woensdag", "Donderdag", "Vrijdag"];
```

## 2.2 Arrays gebruiken

### 2.2.1 Arrays uitlezen

We kunnen de elementen in een array gemakkelijk opvragen en uitlezen.

*JS-code*

```
// Geef de volledige array terug met waarde en lengte.
console.log(werkdag);


// Geef één element terug op basis van de positie in de index.
console.log(werkdag[0]);


// Slaat een element uit de array op als aparte variabele.
let maandag = werkdag[0];
```

### 2.2.2 Lengte van een array opvragen

Elke array heeft een bepaalde hoeveelheid elementen, maar soms weten we niet op voorhand hoeveel elementen er in een array staan. Via de eigenschap 'length' kunnen we het aantal elementen van een array eenvoudig opvragen.

*JS-code*

```
// Aantal elementen van een array opvragen.
console.log(werkdag.length);
```

### 2.2.3 Arrays eenvoudig aanpassen

We kunnen eenvoudig elementen toevoegen, vervangen of verwijderen in een array.

*JS-code*

```
// Voegt op positie 5 een element toe aan de array.
werkdag[5] = "zaterdag";

// Vervangt op positie 0 een element in de array.
werkdag[0] = "zondag";

// verwijder de elementen in de array door de lengte aan te passen.
werkdag.length = 0;
```

## 2.3 Methoden en functies om arrays aan te passen

Er bestaan verschillende methoden en functies om de inhoud van een array aan te passen. In deze cursus bekijken we enkel de meest gebruikte…

### 2.3.1 Join en split

De join-methode zet alle elementen in de array om naar een string en voegt ze samen tot één lange string.

*JS-code*

```
// Elementen samenvoegen tot één string zonder opgegeven scheidingsteken.
// Maandag,Dinsdag,Woensdag,Donderdag,Vrijdag
console.log(werkdag.join());

// Elementen samenvoegen tot één string met " - " als scheidingsteken.
// Maandag - Dinsdag - Woensdag - Donderdag - Vrijdag
console.log(werkdag.join(" - "));
```

De split-methode doet het omgekeerde van de join-methode. Via de spit-methode kunnen we één lange string omzetten naar een array met meerdere elementen. Bij de split-methode moeten we altijd het scheidingsteken meegeven.

*JS-code*

```
// String splitsen tot een array met meerdere elementen.
let werkdagen = "Maandag,Dinsdag,Woensdag,Donderdag,Vrijdag";
console.log(werkdagen.split(","));
```

### 2.3.2 Sort

Via de sort-methode kunnen we alle elementen in de array sorteren in alfabetische volgorde van A naar z.

*JS-code*

```
// Elementen in array alfabetisch sorteren van A naar z.
// Dinsdag, Donderdag, Maandag, Vrijdag, Woensdag
console.log(werkdag.sort());
```

### 2.3.3 Reverse

Via de reverse-methode kunnen we de volgorde van alle elementen in de array omdraaien.

*JS-code*

```
// Elementen in array in omgekeerde volgorde plaatsen.
// Vrijdag, Donderdag, Woensdag, Dinsdag, Maandag
console.log(werkdag.reverse());

// Elementen in array in omgekeerd alfabetisch sorteren van z naar A.
// Woensdag, Vrijdag, Maandag, Donderdag, Dinsdag
werkdag.sort();
werkdag.reverse();
console.log(werkdag);
```

### 2.3.4 Push en unshift

Via de push-methode kunnen we elementen aan het einde van een array toevoegen. Met de unshift-methode kunnen we elementen aan het begin van een array toevoegen.

*JS-code*

```
// Extra kleuren toevoegen als laatste elementen aan de array
// Rood, Geel, Blauw, Groen, Wit, Zwart
let kleuren = [“Rood”, “Geel”, “Blauw”];
kleuren.push(“Groen”, “Wit”, “Zwart”);
console.log(kleuren);

//Extra kleuren toevoegen als eerste elementen aan de array
// Groen, Wit, Zwart, Rood, Geel, Blauw
let kleuren = [“Rood”, “Geel”, “Blauw”];
kleuren.unshift(“Groen”, “Wit”, “Zwart”);
console.log(kleuren);
```

### 2.3.5 Pop en shift

Via de pop-methode kunnen we één element aan het einde van een array verwijderen en het verwijderde element weergeven. Met de shift-methode kunnen we één element aan het begin van een array verwijderen en het verwijderde element weergeven.

*JS-code*

```
// Laatste kleur verwijderen uit de array en verwijderde kleur teruggeven
// Rood, Geel
let kleuren = [“Rood”, “Geel”, “Blauw”];
kleuren.pop();
console.log(kleuren);

// Eerste kleur verwijderen uit de array en verwijderde kleur teruggeven
// Geel, Blauw
let kleuren = [“Rood”, “Geel”, “Blauw”];
kleuren.shift();
console.log(kleuren);
```

### 2.3.6 Concat

Met de concat-methode kunnen we twee arrays samenvoegen tot een nieuwe array.

*JS-code*

```
// Twee arrays samenvoegen tot een nieuwe array.
// Maandag, Dinsdag, Woensdag, Donderdag, Vrijdag, Zaterdag, Zondag
let werkdag = [“Maandag“, “Dinsdag“, “Woensdag“, “Donderdag“, “Vrijdag“];
let weekenddag = [“Zaterdag”, “Zondag”];
let dagen = werkdag.concat(weekenddag);
console.log(dagen);
```

# 3 Functies

Een functie is een stukje code met een naam. Functies zorgen ervoor dat we bepaalde stukjes code makkelijk meermaals kunnen gebruiken zonder deze opnieuw te moeten schrijven. Dit is onderhoudsvriendelijker, want we moeten slechts op één plaats onze stukjes code beheren.

## 3.1 Functies definiëren

Om een zelfgeschreven functie in Javascript te kunnen gebruiken, moeten we deze definiëren. Er bestaan twee manieren om functies te definiëren:

- De functie expressie
- De functie declaratie

### 3.1.1 Functie expressie

Een *functie expressie* is een gewone variabele declaratie waarbij de waarde van de variabele een functie is. De functie is enkel beschikbaar na de definitie.

*JS-code*

```
let vierkant = function(lengte) {
      return lengte * lengte;
};
console.log(vierkant(5));
```

### 3.1.2 Functie declaratie

Een *functie declaratie* is 'hoisted' (omhooggebracht) en geeft ons meer vrijheid over de locatie in onze code. De functie is hierdoor ook beschikbaar in de code die voorafgaat aan de functie declaratie.

*JS-code*

```
console.log(kubus(5));
function kubus(lengte) {
      return lengte * lengte * lengte;
}
```

## 3.2 Opbouw van functies

Een zelfgeschreven functie bevat *'meestal'* volgende onderdelen:

- Een unieke naam
- Een lijst met nodige parameters
- Een body met statements

*JS-code*

```
function functieNaam(parameter1, parameter2) {
      // statements waarin we de parameters gebruiken
}
```

### 3.2.1 Unieke naam

Voor de naamgeving van een functie moeten we dezelfde regels hanteren als voor de naamgeving van een variabele:

- Het eerste karakter moet een letter of underscore (_) zijn.
- De naam mag zowel hoofdletters als kleine letters bevatten.
- De naam mag letters, getallen, underscore en dollartekens bevatten.
- De naam mag geen spaties of andere lees- of speciale tekens bevatten.
- De naam mag geen gereserveerd woord (gekend woord in de webtaal) zijn.

Een functie kan ook geen naam hebben. We spreken dan van een anonieme functie of IIFE (Immediately-Invoked Function Expression). Deze functie wordt meteen uitgevoerd en kunnen we niet meer hergebruiken.

*JS-code*

```
( function(parameter) {
      // statements waarin we de parameter gebruiken;
}(waardeParameter));


// Bereken de prijs inclusief 21% btw voor een bedrag exclusief btw.
// Uitkomst in console is 121.
( function(bedrag) {
      console.log(bedrag * 1.21);
}(100));
```

### 3.2.2 Lijst met parameter(s)

Een parameter is een waarde die door de functie intern wordt gebruikt. De parameters staat na de unieke functienaam tussen haakjes. Indien er meerdere parameters nodig zijn in een functie, worden deze gescheiden met behulp van een komma.

Parameters zijn niet verplicht in een functie. Sommige functies kunnen perfect zonder parameters werken. De haakjes na de functienaam blijven dan leeg.

*JS-code*

```
// Schrijf 'Test' in een HTML-element.
function schrijfTest() {
      document.getElementById("uitkomst").innerHTML = "Test";
}
schrijfTest();


// Tel twee getallen op.
function optellen(getal1, getal2) {
      return getal1 + getal2;
}
console.log(optellen(12, 15));
console.log(optellen(1, 2));


// Geef een naam op als variabele en gebruik deze via een functie.
let volledigeNaam = "Kimberly Willems";
function weergevenNaam(naam) {
      console.log("Uw naam is " + naam + ".");
}
weergevenNaam(volledigeNaam);
```

Alle typen variabelen en waarden mogen als parameter worden gebruikt. De parameters bestaan zolang de functie wordt uitgevoerd. Als een functie meerdere keren achter elkaar wordt uitgevoerd met telkens verschillende variabelen en waarden, worden de parameters telkens 'ververst'.

## 3.3 Waarden retourneren

Functies kunnen één enkelvoudige waarde (getal, string, boolean, …) als resultaat teruggeven via het statement 'return'.  Na het statement 'return' wordt de functie automatisch beëindigd.

*JS-code*

```
// Bereken het kwadraat van het getal 5.
function kwadraat(getal) {
      return getal * getal;
}
console.log(kwadraat(5));
```

De returnwaarde van een functie kunnen we ook opslaan in een variabele. Deze variabele kunnen we nadien verder gebruiken in ons script.

*JS-code*

```
// Bereken het kwadraat van het getal 5.
function kwadraat(getal) {
      return getal * getal;
}
let uitkomst = kwadraat(5);
console.log(uitkomst);
```

*Opgelet! Wanneer we meerdere waarden willen retourneren uit een functie, moeten we deze wrappen in een object. Objecten zien we in het volgende hoofdstuk.*

# 4 Objecten

In moderne webapplicaties zijn objecten amper nog weg te denken. In dit hoofdstuk gaan we de algemene basis zien van 'objecten' binnen Javascript. In de komende hoofdstukken gaan we verder op deze basis en breiden we onze kennis over objecten verder uit.

## 4.1 Objecten in het echte leven

In het echte leven kunnen we voorwerpen, zoals bijvoorbeeld een auto, zien als een object. Een auto heeft een set eigenschappen en methodes.

| | |
|---|---|
| `auto.merk = Mustang` | `auto.start()` |
| `auto.model = Mach 1` | `auto.stop()` |
| `auto.bouwjaar = 1973` | `auto.rem()` |
| `auto.gewicht = 1547` | `auto.veranderSnelheid()` |
| `auto.kleur = grijs` | `auto.draaiRechts()` |
| … | … |

Alle auto's hebben dezelfde eigenschappen, maar de waarden van de eigenschappen verschillen. Alle auto's hebben dezelfde methodes, maar deze worden uitgevoerd op een verschillend moment.

## 4.2 Objecten in Javascript

In Javascript bestaat een object ook uit een set van eigenschappen (properties) en methoden (methods). Objecten zijn erg handig om bij elkaar horende informatie te groeperen en te organiseren. Ze zijn zelfs eenvoudiger in het gebruik dan arrays omdat ze werken met 'genaamde waarden' in plaats van met een numerieke index.

Objecten worden in Javascript genoteerd met accolades (object literal).

*JS-code*

```
// Notatie op één regel
let object = { naam1: waarde, naam2: waarde, naam3: waarde };

// Notatie op meerdere regels.
let object = {
      naam1: waarde,
      naam2: waarde,
      naam3: waarde
};

// Een voorbeeldobject 'persoon' met drie eigenschappen.
let persoon = {
      voornaam: 'Kimberly',
      achternaam: 'Willems',
      beroep: 'lector'
};
```

### 4.2.1 Object properties

De eigenschappen (properties) in een object hebben elk een naam en een waarde. De naam en de waarde worden met een dubbele punt van elkaar gescheiden. De naam is een string en de waarde kan eender welke geldige Javascript-waarde zijn. De eigenschappen binnen het object worden met komma's van elkaar gescheiden.

De waarde van een eigenschap kan volgende Javascript-waarde zijn:

- Variabelen
- Diverse gegevenstypen
- Functies
- Arrays
- Objecten

*JS-code*

```
// Een voorbeeldobject 'game' met verschillende waarden.
let game = {
      titel: 'Assassin's Creed Valhalla',       // Stringwaarde
      editie: 'Drakkar Edition',                // Stringwaarde
      ean: 3307216169116,                       // Numerieke waarde
      pegi: { leeftijd: 18, inhoud: 'Geweld'}   // Objectwaarde
};
```

We kunnen de eigenschap van een object op twee verschillende manieren uitlezen en gebruiken, namelijk via een:

1. *Puntnotatie:* we schrijven eerst de objectnaam en daarna verbinden we de eigenschapnaam aan de objectnaam met een punt.
2. *Arraynotatie:* we schrijven eerst de objectnaam en daarna schrijven we de eigenschapnaam tussen blokhaken én aanhalingstekens.

*JS-code*

```
// Een voorbeeldobject 'persoon' met drie eigenschappen.
let persoon = {
      voornaam: 'Kimberly',
      achternaam: 'Willems',
      beroep: 'lector'
};

// Puntnotatie
document.getElementById('uitkomst').innerHTML = 'Mijn naam is ' +
persoon.voornaam + ' ' + persoon.achternaam + ' en ik ben ' +
persoon.beroep + ' van beroep.';

// Arraynotatie
document.getElementById('uitkomst').innerHTML = 'Mijn naam is ' +
persoon["voornaam"] + ' ' + persoon["achternaam"] + ' en ik ben ' +
persoon["beroep"] + ' van beroep.';
```

### 4.2.2 For … in lus

Met 'for … in' kan je itereren doorheen de eigenschappen van een object. De lus wordt eenmalig uitgevoerd overheen iedere eigenschap.

*JS-code*

```
let persoon = {voornaam:"Jasper", naam:"Beuls", mailadres:"jb@pxl.be"};
let uitkomst = "";
let eigenschap;
for (eigenschap in persoon) {
  uitkomst = uitkomst + persoon[eigenschap] + " ";
}
document.getElementById("demo").innerHTML = uitkomst;
```

### 4.2.3 Nieuwe eigenschappen toevoegen

We kunnen nieuwe eigenschappen toevoegen aan een bestaand object door het een waarde te geven.

*JS-code*

```
persoon.nationaliteit = "belg";
```

### 4.2.4 Eigenschappen verwijderen

Met de 'delete' operator kunnen we de waarde van een eigenschap en de eigenschap zelf verwijderen uit een object.

*JS-code*

```
delete persoon.mailadres;
```

*Opgelet! Deze operator mogen we NOOIT gebruiken op voorgedefinieerde Javascript objecten zoals window, Math en Date objecten!*

## 4.3 Object methods

De methoden in een object zijn de acties die kunnen worden uitgevoerd op het object. Methoden worden opgeslagen in eigenschappen als functiedefinities.

*JS-code*

```
// Een voorbeeldobject 'game' met eigenschappen en methoden aanmaken.
let game = {
      titel: 'Assassin's Creed Valhalla',
      editie: 'Drakkar Edition',
      prijs: getGamePrice('3307216169116'),     // Waarde uit functieresultaat
      volledige_titel: function() {             // Functiewaarde
            return this.titel + ' - ' + this.editie;
      }
};
```

*<u>Opgelet! Het woord 'This' in de functiedefinitie</u>*
*We gebruiken in de functiedefinitie het woord 'this' om te refereren naar de 'eigenaar van de functie, namelijk het object. De letterlijke vertaling van 'this.titel' en 'this.editie' uit het voorbeeld is hierdoor 'de titel van dit object' en 'de editie van dit object'.*

We kunnen de objectmethode benaderen via een puntnotatie. We schrijven eerst de objectnaam en daarna verbinden we de methodenaam aan de objectnaam met een punt. Achter de methodenaam voegen we nog haakjes toe. Doen we dit niet, krijgen we de functiedefinitie terug.

*JS-code*

```
// Een voorbeeldobject 'game' met eigenschappen en methoden aanmaken.
let game = {
      titel: 'Assassin's Creed Valhalla',
      editie: 'Drakkar Edition',
      volledige_titel: function() {
            return this.titel + ' - ' + this.editie;
      }
};

// Resultaat in html-element bevat de eigenlijke titel en editie.
document.getElementById('uitkomst').innerHTML = game.volledige_titel();

// Resultaat in html-element bevat de functiedefinitie
document.getElementById('uitkomst').innerHTML = game.volledige_titel;
```

### 4.3.1 Nieuwe methode toevoegen

We kunnen nieuwe methoden toevoegen aan een bestaand object op onderstaande manier.

*JS-code*

```
persoon.volledigeNaam = function() {
      return this.voornaam + " " + this.naam;
};
```

## 4.4 Objecten omzetten naar arrays

Elk object kan geconverteerd worden naar een array.

*JS-code*

```
let persoon = {voornaam:"Jasper", naam:"Beuls", mailadres:”jb@pxl.be”};
let persoonArray = Object.values(persoon);
```