



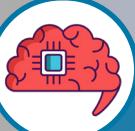
Sint-Lievenscollege

COMPUTATIONEEL DENKEN & JAVASCRIPT

Voornaam: _____

Achternaam: _____

Klas: _____



Computationeel Denken ...

Het begon zo'n slordige 70 jaar geleden. Wetenschappers ontwikkelden een nieuwe wereld. Een digitale wereld! Een wereld van enkele grote computersystemen die met elkaar in verbinding stonden. Computers die een gehele kamer vulden en code die stukje per stukje met de hand werd geschreven. Draai de klok zeventig jaar vooruit en we bevinden ons middenin de digitale revolutie! Onze samenleving wordt steeds meer beïnvloed en gestuurd door computersystemen, apps en algoritmes. Nagenoeg iedereen heeft een krachtig device in de broekzak zitten of om de pols. Talloze problemen, nu en in de toekomst, worden opgelost via die krachtige devices! Maar als jij wil meewerken aan die toekomst, zelf oplossingen uitwerken en jouw steentje bijdragen aan die digitale revolutie, moet je eerst leren '**computationeel denken**'!



Is probleemoplossend denken!

'**Computationeel denken**' is een manier van **probleemoplossend denken**. Een aanpak om problemen op te lossen door het goed gebruik van talloze digitale methodes die ons omringen. Dus door te leren om computers in te zetten als werkmiddelen of 'tools'.

Maar voordat je een computer kan gebruiken als hulpmiddel, moet je begrijpen hoe je dat kan doen. Een computer, groot of klein, is een strikt logische machine die werkt met heel duidelijke instructies en acties. Vriendelijke vragen aan een computer om een probleem op te lossen zal je niet ver brengen. Dat zal de computer niet begrijpen. Maar jouw probleem opdelen in korte acties en deze vertalen naar concrete stappen of code ... bingo!

Met deze syllabus brengen we jou naar **level 2.0 van het computationeel denken**. Dit doen we door ons te verdiepen in de programmeertaal **Javascript!**

Level 2.0 met Javascript!

Javascript is een programmeertaal zoals Python of Object C dat zijn. Het is een taal die wij kunnen gebruiken om instructies te geven aan een computer. Wanneer we instructies zouden schrijven in het Nederlands, zou de computer dat niet begrijpen. Maar een programmeertaal is een taal die wij en onze computers begrijpen.

Javascript kan je terugvinden in een aantal toepassingen, zoals bij het leren programmeren via Blockly, Micro:Bit of binnenin de Minecraft Education Edition spelomgeving. Dit maakt dat het een goede stapsteen is om onze vaardigheden en kennis binnen het **computationeel denken** naar een nieuw level te brengen.



In deze syllabus herhalen we de gekende stappen van het **computationeel denken** en verdiepen we ons in het ontwerpen van een algoritme. Wanneer we ons **algoritme ontwerpen** en gebruikmaken van Javascript, kunnen we **programmeerconcepten** gebruiken. Wanneer je **concepten** goed gebruikt,



let op de **syntax, variabelen, datatypes, operatoren** ... kan je saaie statische websites tot leven laten komen, rekentools bouwen of zelfs jouw eigen game ontwerpen!

Inhoudsopgave



- 1) Herhaling Computationeel Denken - pagina 4
- 2) Hoe bouw ik een goed algoritme?
- 3) Aan de slag met Dodona en Papyros
- 4) Hoofdstuk 1: Hello, World!
- 5) Hoofdstuk 2: Datatypes en de Sequentie
- 6) Hoofdstuk 3: De Selectie
- 7) Hoofdstuk 4: De Begrenste Herhaling
- 8) Hoofdstuk 5: De Voorwaardelijke Herhaling
- 9) Hoofdstuk 6: Uitbreidingsoefeningen

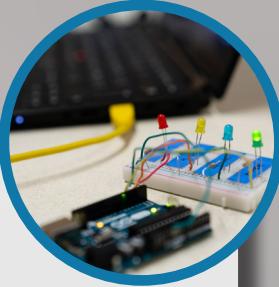
Lesdoelen



Na de opdrachten binnen deze cursus kan je:

- o De stappen van het computationeel denken opsommen en duiden;
- o De stappen van het computationeel denken correct uitvoeren;
- o Een algoritme schematisch uitwerken;
- o Een algoritme uitwerken met behulp van Javascript-code;
- o De syntax van Javascript-code correct hanteren;
- o Gebruikmaken van gekende programmeerconcepten zoals sequentie, selectie, herhalingen ...
- o Gebruikmaken van variabelen;
- o Datatypes (string, number, boolean ...) herkennen en correct gebruiken;
- o Een algoritme ontwerpen dat schematisch functioneel en leesbaar is.

Horizontale en verticale leerlijnen



De kennis en vaardigheden binnen deze lessenreeks komen het best tot hun recht wanneer deze ingebed zijn in verticale- en horizontale leerlijnen, in tegenstelling tot een losstaande cursus.

Verticaal:

Deze lesmaterialen boven verder op bestaande kennis die leerlingen hebben over computationeel denken. Dit via de inleidende lessenreeks 'Computationeel Denken', 'Micro:Bit' of equivalent project. Elementen binnen deze lesmaterialen worden vervolgd binnen de lessenreeks 'Game Design', 'Programmeren met Python' en/of 'Bouw een Slimme Vuilnisbak'.

Horizontaal:

Binnen deze lessenmaterialen komen, naast belangrijke **programmeerconcepten**, ook **concepten uit de lessen wiskunde** aan bod. Concepten zoals volgorde van bewerkingen, operatoren, oppervlakte- en volumeberekeningen, gemiddelde, mediaan ... Deze concepten worden gebruikt om problemen digitaal, met een computer, op te lossen.



Deze opdrachten zijn '**unplugged**' en voer je uit **zonder** computer.



Deze opdrachten zijn '**plugged**' en voer je uit **mét** computer.



Scan de QR-code en krijg zo een **lesvideo** met een woordje uitleg.

Computationeel Denken: Herhaling!



Wanneer we computationeel denken, benaderen we een probleem op zo een manier zodat een computer het kan oplossen. We delen het probleem op, ontwerpen een algoritme en vertalen die naar instructies die een computer kan uitvoeren. Computationeel denken is dus een strategie om probleemoplossend te denken en volgt een aantal stappen, namelijk:

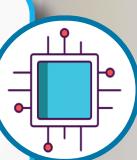
1 - Decompositie:

Bij een **decompositie** wordt een **complex probleem opgedeeld in kleinere deelproblemen of stapjes**. Die deelproblemen zijn veel meer behapbaar voor een computer en lossen we zo een voor een op.



2 - Patroonherkennen

Wanneer we complexe zaken ontleden door **decompositie**, vinden we vaak **patronen** terug. Patronen opsporen en begrijpen kan ons helpen om onze oplossing efficiënter te maken! Zo moet je niet telkens hetzelfde uitvinden!



3 - Abstraheren:

Concrete voorbeelden omzetten in wiskundetaal heet **abstraheren**. Zo kunnen we bijvoorbeeld gebruikmaken van **variabelen** om acties of getallen in het geheugen van de computer te bewaren.



4 - Algoritme Ontwerpen

Een **algoritme** is een **stappenplan** dat ons naar de juiste oplossing brengt. Een goed algoritme voldoet aan een aantal voorwaarden, zoals: **schematisch, functioneel, leesbaar, programmeerconcepten, wiskundeconcepten en creativiteit**.



5 - Debugging

Een goed ontworpen algoritme zal ons tot de oplossing leiden ... maar wat als dat niet het geval is? We spreken dan van een '**bug**'. Dat is een fout in ons ontwerp. **Het opmerken van deze fouten, ze herkennen en de fout uit de oplossing verwijderen noemen we 'debuggen'**.



Blad-Steen-Schaar

Blad-Steen-Schaar is een spelletje met een set eenvoudige regels. Wanneer je dit wil spelen met een menselijke tegenspeler, is vaak geen speluitleg nodig. De eerste die 2 van de 3 spelletjes wint, is de eindwinnaar! Als je wil spelen tegen een computer, zoals dat kan met een Micro:Bit, zal dat niet zo eenvoudig verlopen. We moeten de stappen van het spel omzetten naar instructies die de computer wel kan begrijpen. Daarvoor gebruiken de stappen van het **computationeel denken!**



Opgave 1 - Decompositie:

Werk in duo's of klassikaal. Leg de spelregels van 'blad-steen-schaar' mondeling uit.
Noteer de stappen in volzinnen.

- 1) _____
- 2) _____
- 3) _____
- 4) _____

Opgave 2 - Patroonherkenning:

Bekijk bovenstaande stappen, na de decompositie, gronding. Kan je hier een patroon in opsporen? (tip: een voorbeeld van een patroon is een selectie of herhaling).

Opgave 3 - Abstraheren:

Bij het abstraheren vertalen we een concreet voorbeeld naar iets abstract, zoals wiskunde. We maken hier gebruik van **variabelen**.

Kies een toepasselijke naam voor de variabele: _____

Welke waarden kan die variabele bevatten: _____

Opgave 4 - Algoritme uitwerken:

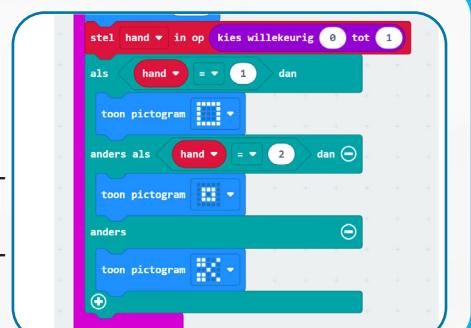
Bij het uitwerken van een algoritme werken we schematisch. Via instructies, die stap voor stap uitgevoerd worden, bereiken we de oplossing. Vervolledig het algoritme hieronder:

Stap 1:
Tel af: 3, 2, 1

Stap 2:
Kies cijfer van ... tot ...

Opgave 5 - Debugging:

Bekijk het algoritme hiernaast.
Wat verloopt niet goed in dit algoritme?



Computationeel Denken - EV-Reizen met Fred

Komende vakantie gaat Fred een uitdaging aan. Hij onderneemt een autoreis met een prototype elektrische auto. Fred wil een tool ontwikkelen om te berekenen hoeveel energie de batterij minimaal moet bevatten om veilig zijn bestemming te bereiken, zonder onderweg te moeten herladen. Fred rijdt het nieuwste prototype Southpole. Deze heeft een batterij van 60.5 kWh. De amerikaanse fabrikant geeft aan dat elke 62 mijl de auto 26% batterij verbruikt. Fred wil een programma ontwerpen dat hem vraagt naar de afstand (in km), zijn batterijpercentage en kan vertellen of de bestemming te bereiken is met 1 "tank." Het programma moet ook vertellen hoeveel procent van zijn batterij verbruikt zal worden.



Opgave 1 - Decompositie

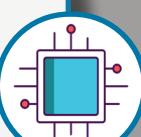
Duid met kleur de hoofdzaken aan in bovenstaande opgave.



Opgave 2 - Patroonherkenning

Bevat bovenstaande opgave een patroon zoals een herhaling, selectie ...

Motiveer jouw antwoord.



Opgave 3 - Abstraheren

Bevat bovenstaande opgave gegevens die we kunnen opslaan als een variabele (= opslaan in het geheugen van de computer om bv. mee te rekenen)? Indien ja, welke?



Opgave 4 - Algoritme Ontwerpen

Maak hieronder een schematische voorstelling van het algoritme (=stappenplan) voor deze opdracht. Zorgt ervoor dat jouw algoritme leesbaar en functioneel is. Debug nadien samen met een buur.





Wat is een “goed” algoritme?

Een **algoritme** is een **stappenplan** dat ons naar de juiste oplossing brengt. Je vindt ze bij een LEGO-bouwdoos of een IKEA-meubel. Bij het stappenplan is het belangrijk dat je de **instructies in de juiste volgorde, een voor een, uitvoert!** De volgorde wijzigen zal eindigen in een heel vreemde IKEA-kast ... Een goed algoritme voldoet aan volgende kenmerken:

- 1) Schematisch
- 2) Functioneel
- 3) Leesbaar
- 4) Programmeerconcepten
- 5) Wiskundeconcepten
- 6) Creativiteit

| Andere bookmarks | Leeslijst |



1) Schematisch

Een goed ontwerp voor een algoritme is **schematisch**. Bekijk volgende voorstellingen van een algoritme. Welke van deze twee verkies je? Motiveer jouw keuze.

var snelheid = 100; var remafstand = snelheid / 10; remafstand = remafstand * remafstand / 2; console.log(remafstand)

A

B

```
when green flag clicked
  [set snelheid to (100)
  set Remafstand to (snelheid) ÷ (10)
  set Remafstand to (Remafstand) × (Remafstand) ÷ (2)]
  say (Remafstand) [Remafstand]
```

2) Functioneel

Een goed ontwerp voor een algoritme is **functioneel**. Dat betekent dat het de gewenste uitkomst moet bereiken. Doet het dat niet, dan hebben we te maken met een **bug**. Welke van deze twee algoritmes verkies je? Motiveer jouw keuze.

```
when green flag clicked
  [set snelheid to (100)
  set Remafstand to (snelheid) ÷ (10)
  set Remafstand to (Remafstand) × (Remafstand) ÷ (2)]
  say (Remafstand) [Remafstand]
```

A

```
when green flag clicked
  [set snelheid to (100)
  set Remafstand to (snelheid) ÷ (10)
  set Remafstand to (Remafstand) × (Remafstand) ÷ (2)]
  say (Remafstand) [Remafstand]
```

B

3) Leesbaar

Een goed ontwerp voor een algoritme is **leesbaar**. Bekijk volgende vier voorstellingen van een algoritme. Twee van deze vier zijn een goed voorbeeld.

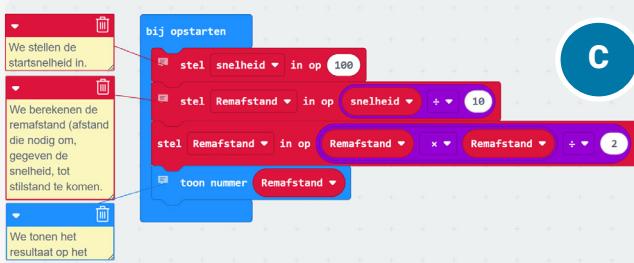
Opgave: Welke twee van deze vier voorstellingen is goed? Motiveer jouw keuze.



A

```
1 let snelheid = 100
2 let Remafstand = snelheid / 10
3 Remafstand = Remafstand * Remafstand / 2
4 basic.showNumber(Remafstand)
5 |
```

B



C

```
1 // We stellen de startsnelheid in.
2 let snelheid = 100
3 // We berekenen de remafstand (afstand die nodig om, gegeven de snelheid, tot stilsta-
4 let Remafstand = snelheid / 10
5 Remafstand = Remafstand * Remafstand / 2
6 // We tonen het resultaat op het scherm.
7 basic.showNumber(Remafstand)
8 |
```

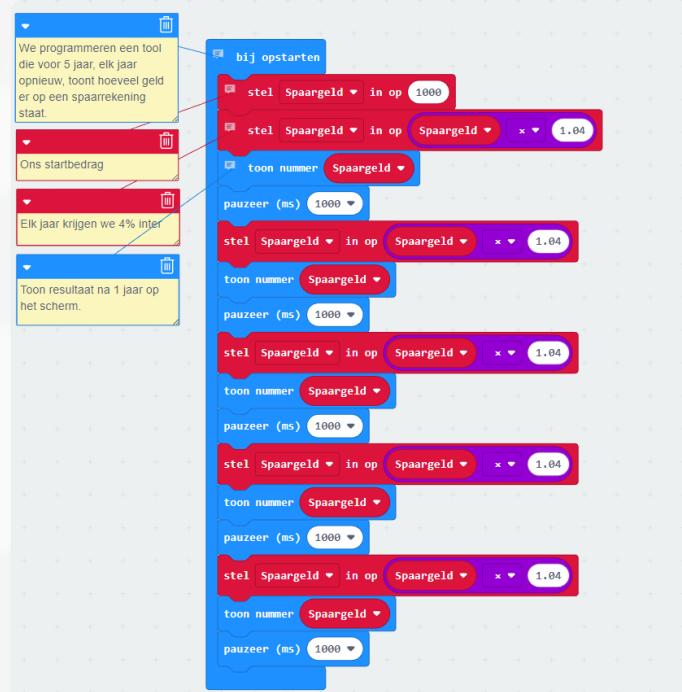
D

4) Programmeerconcepten

Een goed ontwerp voor een algoritme maakt handig gebruik van **programmeerconcepten**.

Dit zijn selectiefuncties zoals IF of ELSE, begrenste herhaling, voorwaardelijke herhaallus ...

Opgave: Hoe kan je onderstaan algoritme verbeteren? Motiveer jouw antwoord.



5) Wiskundeconcepten

Computers zijn strikt logische machines. Grote rekenmachines met veel rekenkracht, als het ware. Een goed ontwerp voor een algoritme maakt gebruik die rekenkracht door correct toepassen van **wiskundeconcepten** zoals formules en rekenregels.

Opgave: Bekijk onderstaande opgave en bijhorend algoritme.

Gegeven:

De code hiernaast geeft klanten die tien of meer cupcakes kopen 10% korting. Kopen ze er twintig of meer, krijgen ze 20% korting. Een enkele cupcake kost 2 euro.

Code:

```
var aantal_cupcakes = ... ;  
var prijs_per_cupcake = 2;  
var prijs = (aantal_cupcakes * prijs_per_cupcake);  
ALS aantal_cupcakes gelijk aan 10 {  
    totaalprijs = prijs / 100 * 110 }  
ALS aantal_cupcakes gelijk aan 20 {  
    totaalprijs = prijs / 100 * 120 }  
//print de uitkomst naar het scherm met console.log  
console.log(totaalprijs);
```

Aan welke **wiskundeconcept(en)** wordt hier gezondigd?

Debug de code! Duid de fout(en) aan met kleur. Noteer jouw correcties hieronder.

6) Creativiteit

Creativiteit bij het maken van een algoritme of programmeren kan op verschillende manieren tot uiting komen, namelijk via **alternatieve oplosmethodes** en **extra elementen**.

Alternatieve oplosmethode:

- o Werd de oplossing op een andere manier gevonden?
- o Is deze oplossing even efficiënt?
- o Is deze oplossing even leesbaar?

Extra elementen:

- o Eigen elementen toevoegen in een bestaande oplossing.
- o "Easter eggs"

Easter eggs ken je misschien uit de game wereld. Ontwikkelaars voegen deze heel graag toe in hun games. Voor sommige gamers is het een ware zoektocht om ze allemaal te vinden.

Opgave: Geef een eigen voorbeeld van zo'n easter egg.

Aan de slag met Dodona en Papyros!



Wanneer je de stappen van het **computationeel denken** en het **ontwerpen van een goed algoritme** onder de knie hebt, is het tijd voor de **plugged activiteiten**. Wanneer je een algoritme van getekend schema naar de computer wil brengen, zal je de instructies moeten uitschrijven in een **programmeertaal**. Dat is een taal die logica, functies, variabelen ... bevat en waarmee de computer aan de slag kan gaan. Om de programmeertaal Javascript (en later Python) aan te leren, maken we gebruik van de digitale tools **Dodona en Papyros van de UGent**.

Dodona: Onze Digitale Leeromgeving

Dodona is een online leeromgeving waar je tal van oefeningen terugvindt. Het platform geeft je snel een overzicht van de oefeningen die je wel of niet hebt gemaakt. Wanneer je oefeningen indient, kan de leerkracht feedback geven en zien waar het goed- of fout loopt. Dodona kan je ook basishints geven wanneer je fouten maakt tegen de **syntax** van de programmeertaal.

- o Wanneer je een oefening indient, komt die automatisch bij jouw docent terecht. Een stuk handiger dan werken met een uploadzone!
- o Wanneer je een oefening indient, zal Dodona een kleine controle uitvoeren op de syntax. Dodona kan je dus vertellen of de syntax van jouw code in orde is.
- o **Opgelet:** Wanneer je oplossing syntaxis juist is, betekent niet dat je volledige oplossing inhoudelijk correct is! Daarvoor dient de docent!

The screenshot shows a dark-themed interface for 'Oefeningenreeksen'. At the top, it says 'Hoofdstuk 1 - Hello, world!' with a graduation cap icon. Below that, it says 'In dit hoofdstuk leren we de eerste regels code schrijven en maken we zo kennis met een eerste datatype, nl. D'. A list of exercises follows:

- 1.0 - Welkom
- 1.1 - Hello, world!
- 1.2 - Naam, familienaam?
- 1.3 - Datatype String
- 1.4 - Concateneren
- 1.5 - Goedemorgen, Robot!

Papyros: Om te testen en debuggen!

Wat Dodona voorlopig nog niet kan, is het uittesten van jouw geschreven code en jou het resultaat tonen. Wanneer we onze code willen schrijven, testen en vervolgens debuggen, maken we gebruik van de Papyros-omgeving. Eenmaal gedebugged, kan je jouw oplossing indienen via de Dodona-leeromgeving! Als alternatief voor Papyros kan ook JSfiddle gebruikt worden.

Code:

```
1 var dag = 0;
2 var startwaarde = 1000000;
3 while (startwaarde > 500000) {
4     startwaarde = startwaarde * 0.964;
5     dag = dag + 1;
6     console.log(dag)
7 }
8 console.log('Na ' + dag + ' dagen is de helft weggesmolten.');
```

Run



Hoofdstuk 1: Hello, world!



In dit hoofdstuk leer je jouw eerste regel Javascriptcode schrijven. Je maakt kennis met de console.log-functie en leert jouw eerste **datatype** kennen, de 'string'!

1.1 - Hello, world!

Gebruik invulvelden zoals hieronder om belangrijke informatie, bordschema's, het schema van jouw algoritme of de syntaxis uit de les op te schrijven. Wanneer je schrijft, zal je nieuwe kennis beter vastzetten in jouw langetermijngeheugen. Dat heet effectief leren!

1.2 - Naam, familienaam?

Noteer hieronder jouw oplossing / code:

1.3 - Datatype String

Noteer hieronder jouw oplossing / code.

Waarop moet je letten bij de aanhalingstekens van de string?

1.4 - Concateneren

Noteer hieronder jouw oplossing / code.

Markeer het teken of operator die je moet gebruiken wanneer je 2 strings wil concateneren.

1.5 - Goedemorgen, Robot!

Noteer hieronder jouw oplossing / code.



Wat hebben we geleerd?

In dit eerste hoofdstuk maakte je kennis met jouw eerste functie, eerste datatype en maakte jouw eerste eigen variabele binnen de programmeertaal Javascript. We zetten een aantal fundamentele zaken nog eens op een rij:

De console.log-functie

Met een console.log-functie kunnen we een resultaat van onze code op het scherm laten verschijnen. Zonder deze functie zal ons algoritme uitgevoerd worden, maar ziet de gebruiker niets verschijnen op het scherm. De **syntaxis** van deze functie: **console.log("boodschap");**

Let dus op met hoofdletters. Javascript is een hoofdlettergevoelige taal.

Bijvoorbeeld:

console.log("Sara's gele auto"); // dit is een correcte coderegel.

Console.log("Sara's gele auto"); // dit is geen correcte coderegel!

Datatype 'string'

Wanneer je een boodschap naar het scherm wil printen met de console.log-functie, zoals de boodschap "hello, world", moeten we deze tussen enkele of dubbele aanhalingstekens zetten. Wanneer data tussen die aanhalingstekens staat, dan krijgt het als datatype string. Een string is een reeks aan karakters. Karakters zoals letters, cijfers of tekens. Aanhalingstekens mag je niet mengen!

Bijvoorbeeld:

console.log("Sara's gele auto"); // dit is een correcte coderegel.

console.log('Sara's gele auto'); // dit is geen correcte coderegel!

console.log("Sara's gele auto'); // dit is geen correcte coderegel!

Een variabele declareren

Wanneer we een gegeven meermaals willen gebruiken, om naar het scherm te schrijven of om bewerkingen op uit te voeren, kunnen we dat beter bewaren in het geheugen van de computer als een variabele. Een variabele heeft altijd een naam en een datatype. In dit hoofdstuk leerde je het datatype 'string' kennen. Zo kunnen we een reeks karakters bewaren in het geheugen van de computer. Wanneer je een variabele aanmaakt of declareert, gebruik je een enkelvoudig gelijkheidsteken(=).

Bijvoorbeeld:

var sterk_wachtwoord = '3h0%iKe*8fguvZy3%X#@5@B';

var gebruikersnaam = 'eddie.vedder@mail.be';

Concatenatie / concateneren van strings

Wanneer we meerdere variabelen tegelijk naar het scherm willen printen met één console.log-functie, dan kunnen we deze concateneren. Hiermee koppelen we deze aan elkaar. Om te concateneren maak je gebruik van een enkel plusteken (+)

Bijvoorbeeld:

var voornaam = 'Eddie';

var achternaam = 'Vedder';

console.log(voornaam + ' ' + achternaam); //tussen de aanhalingstekens staat een spatieteken.

Hoofdstuk 2 - Datatypes en de Sequentie



In dit hoofdstuk maak je kennis met andere datatypes zoals numbers, gaat aan de slag met operatoren om data te verwerken en combineert zo Javascript en wiskundeconcepten!

2.0 - Wat is jouw datatype?

Wat loopt er mis bij de oplossing hieronder?

```
var rekening = '500';
var storting = '275';
var rekening = rekening + storting
console.log('Uw nieuw saldo bedraagt ' + rekening + ' euro.)
```

2.1 - De Sequentie

Noteer hieronder jouw oplossing / code:

2.2 - Operatoren in Javascript

Vul onderstaande comments aan. Welke bewerking steekt achter onderstaande operatoren?

```
console.log(x + y); //som maken
```

```
console.log(x - z); //verschil maken
```

```
console.log(x * y); // -----
```

```
console.log(x / z); // -----
```

```
console.log(x ** y); // -----
```

2.3 - Voetbalveld

Noteer hieronder jouw oplossing / code:

2.4 - Zwembad

Noteer hieronder jouw oplossing / code:

2.5 - Korting in de Supermarkt

Noteer hieronder jouw oplossing / code:

2.6 - Gemiddelde Autorit

Noteer hieronder jouw oplossing / code:

2.7 - Huurwaarborg

Noteer hieronder jouw oplossing / code:

2.8 - Energiefactuur

Noteer hieronder jouw oplossing / code:

2.9 - Raakt Fred op zijn bestemming?

Noteer hieronder jouw oplossing / code:



Wat hebben we geleerd?

In dit tweede hoofdstuk maakte je kennis met een nieuw datatype, numbers, verkende je de werking van verschillende operatoren en maakte je oefeningen waarbij je een algoritme moest ontwerpen. Algoritmes die stap voor stap te gingen en waarbij je wiskundeconcepten correct moet toepassen.

De console.log-functie

Met een console.log-functie kunnen we een resultaat van onze code op het scherm laten verschijnen. Zonder deze functie zal ons algoritme uitgevoerd worden, maar ziet de gebruiker niets verschijnen op het scherm. De **syntaxis** van deze functie: **console.log("boodschap");**

Let dus op met hoofdletters. Javascript is een hoofdlettergevoelige taal.

Bijvoorbeeld:

console.log("Sara's gele auto"); // dit is een correcte coderegel.

Console.log("Sara's gele auto"); // dit is geen correcte coderegel!

Datatype 'string'

Wanneer je een boodschap naar het scherm wil printen met de console.log-functie, zoals de boodschap "hello, world", moeten we deze tussen enkele of dubbele aanhalingstekens zetten. Wanneer data tussen die aanhalingstekens staat, dan krijgt het als datatype string. Een string is een reeks aan karakters. Karakters zoals letters, cijfers of tekens. Aanhalingstekens mag je niet mengen!

Bijvoorbeeld:

console.log("Sara's gele auto"); // dit is een correcte coderegel.

console.log('Sara's gele auto'); // dit is geen correcte coderegel!

console.log("Sara's gele auto'); // dit is geen correcte coderegel!

Een variabele declareren

Wanneer we een gegeven meermaals willen gebruiken, om naar het scherm te schrijven of om bewerkingen op uit te voeren, kunnen we dat beter bewaren in het geheugen van de computer als een variabele. Een variabele heeft altijd een naam en een datatype. In dit hoofdstuk leerde je het datatype 'string' kennen. Zo kunnen we een reeks karakters bewaren in het geheugen van de computer. Wanneer je een variabele aanmaakt of declareert, gebruik je een enkelvoudig gelijkheidsteken(=).

Bijvoorbeeld:

var sterk_wachtwoord = '3h0%iKe*8fguvZy3%X#@5@B';

var gebruikersnaam = 'eddie.vedder@mail.be';

Concatenatie / concateneren van strings

Wanneer we meerdere variabelen tegelijk naar het scherm willen printen met één console.log-functie, dan kunnen we deze concateneren. Hiermee koppelen we deze aan elkaar. Om te concateneren maak je gebruik van een enkel plusteken (+)

Bijvoorbeeld:

var voornaam = 'Eddie';

var achternaam = 'Vedder';

console.log(voornaam + ' ' + achternaam); //tussen de aanhalingstekens staat een spatieteken.



Hoofdstuk 3 - De Selectie

Wanneer de computer een stuk code uitvoert, krijgen we steeds een oplossing.

Maar we kunnen de code zodanig ontwerpen dat meerdere uitkomsten mogelijk zijn! Hiervoor gebruiken we voorwaarden. Aan de hand van de voorwaarden kan de computer selecteren welke stap er gezet moet worden. Daarvoor gebruiken we een IF-ELSE-functie.

3.0 - De Selectie Deel 1: De IF-functie

Noteer jouw oplossing / code hieronder. Noteer het verschil tussen = en == in jouw oplossing.

3.1 - De Selectie Deel 2 - else-functie

Noteer hieronder jouw oplossing / code:

3.2 - De Selectie Deel 3: De IF ELSE-Functie

Noteer hieronder jouw oplossing / code:

3.4 - Aanmelden

Noteer hieronder jouw oplossing / code:

3.5 - Madame Boulangère

Noteer hieronder jouw oplossing / code:

3.6 - Hittegolf

Noteer hieronder jouw oplossing / code:

3.7 - Korting Museumbezoek

Noteer hieronder jouw oplossing / code:

3.8 - Afsluiten Pleinen Gentse Feesten

Noteer hieronder jouw oplossing / code:



Wat hebben we geleerd?

In dit derde hoofdstuk maakte je kennis met de selectie. Wanneer je een oplossing of algoritme ontwerpt, kan je door gebruik te maken van voorwaarden en de if, if else of else-functie meerdere uitkomsten bereiken. Je dient dan wel goed op te letten met de accolades.

De if-functie

Wanneer we een ALS-functie of een IF-functie willen gebruiken, moeten we daarvoor de correcte **syntaxis** gebruiken. Deze gaat als volgt:

```
if (voorwaarde WAAR is) {  
    actie uitvoeren;  
}
```

```
var getal = X;  
if (getal % 2 == 0 ) {  
    console.log('Het getal ' + getal + ' is een even getal.');//  
}
```

De else-functie

Naast de IF-functie kunnen we ook gebruikmaken van de ELSE-functie. Wanneer niet voldaan wordt aan de voorwaarde van de IF-functie, kan de computer de acties uitvoeren die horen bij de else-functie. **Dat wil dus zeggen dat de else-functie geen eigen voorwaarde heeft.**

```
if (voorwaarde WAAR is) {  
    actie uitvoeren;  
}
```

```
else {  
    actie uitvoeren;  
}
```

De else if functie

Naast de IF-functie en ELSE-functie bestaat er ook een IF ELSE-functie. Soms wil je meerdere IF-Functies na elkaar gebruiken. Dan kan een IF ELSE-functie van pas komen! **Elke else if-functie heeft een eigen voorwaarde.** De **syntaxis** van de else if-functie gaat als volgt:

```
if (voorwaarde WAAR is) {  
    actie uitvoeren;  
}  
else if (nieuwe voorwaarde) {  
    actie uitvoeren;  
}
```

Combineren van voorwaarden

Bij het opstellen van een functie met een voorwaarde, kan je twee of meerdere voorwaarden combineren! Daarvoor gebruik je **&& operator** tussen de twee voorwaarden. Enkel wanneer aan beide voorwaarden is voldaan, voorwaarde_1 EN voorwaarde_2, dan zal de actie uitgevoerd worden.

```
var dag1 = 25.6;  
var dag2 = 30.09  
if (dag1 >= 25 && dag2 >= 25) {  
    console.log('Je gebruikt best zonnecreme!');  
}
```

3.7 - Korting Museumbezoek

Noteer hieronder jouw oplossing / code:

3.8 - Afsluiten Pleinen Gentse Feesten

Noteer hieronder jouw oplossing / code:



Hoofdstuk 4 - De Begrensde Herhaling

In dit hoofdstuk maak je kennis met de begrensde herhaling. Wanneer we computationeel denken, ontleden we een probleem en doen we aan patroonherkenning. Een van de patronen die we zo kunnen herkennen is een begrensde herhaling.

4.0 - De Begrensde Herhaling

Noteer hieronder jouw oplossing / code. Markeer de for-functie met kleur.

4.1 - Etappelijst

Noteer hieronder jouw oplossing / code. Markeer de prompt-functie met kleur.

4.2 - Etappelijst in Kilometer

Noteer hieronder jouw oplossing / code.

4.3 - Bosbranden in Frankrijk

Noteer hieronder jouw oplossing / code:

4.4 - Interest Per Jaar

Noteer hieronder jouw oplossing / code:

4.5 - Bevolkingsgroei

Noteer hieronder jouw oplossing / code:

4.6 - Hittegolfalarm

Noteer hieronder jouw oplossing / code:

4.7 - Gemiddelde Score Kerstrappart

Noteer hieronder jouw oplossing / code:



Wat hebben we geleerd?

In dit vierde hoofdstuk maakte je kennis met de begrensde herhaling. Wanneer je een oplossing of algoritme ontwerpt, kan je patronen herkennen en een herhaling gebruiken om jouw oplossing efficiënter te maken. Een van die herhalingen is de **begrensde herhaling**.

De for-functie

De Begrensde Herhaling is een herhaling waarvan wij, **op voorhand**, precies weten hoeveel keer een of meerdere acties moet worden uitgevoerd. Wanneer we een for-functie of begrensde herhaling willen gebruiken, moeten we daarvoor de correcte **syntaxis** gebruiken. Deze gaat als volgt:

```
for (var teller = startgetal; teller < eindgetal; teller++) {  
    actie;  
}
```

Hierbij geldt:

- o **teller** is de naam van onze variabele waarmee we het aantal herhalingen zullen bijhouden;
- o **startgetal** is de startwaarde van onze teller. We beginnen het aantal herhalingen met dit getal;
- o **teller < eindgetal**: we blijven herhalen zolang de teller kleiner is dan het eindgetal;
- o **teller++**: elke keer we de actie hebben uitgevoerd, verhogen we de teller met +1;
- o **teller--**: elke keer we de actie hebben uitgevoerd, verlagen we de teller met -1;

Bijvoorbeeld:

```
for (var teller = 10; teller > 0; teller--) {  
    console.log(teller);  
}
```

Tip: vereenvoudigde notatie bij bewerkingen

Tijdens het schrijven van code in Javascript (of in bv. Python), maak je tal van bewerkingen. Zo maakten we in dit hoofdstuk oefeningen op interest bij een spaarrekening. Daarbij namen we een variabele, verhoogden de waarde met 3% en sloegen die op in dezelfde variabele.

Voorbeeld 1:

```
var spaarrekenening = spaarrekening * 1.03
```

Wanneer je deze code van rechts naar links zou lezen, en niet als een wiskundige vergelijking, lijkt dit een vrij logische bewerking. Deze bewerking kan je eenvoudiger noteren.

Voorbeeld 2:

```
var spaarrekenening *= 1.03
```

Bovenstaande code uit voorbeeld 2 zal dezelfde uitkomst hebben als voorbeeld 1; maar ik heb minder tekens moeten schrijven om dat resultaat te bereiken.

Opgelet:

Leesbaarheid is belangrijk bij het ontwerpen van een goed algoritme. Wanneer je een algoritme schrijft voor een persoon die zelf niet goed kan programmeren, kies je beter voor de notatie uit voorbeeld 1.

Hoofdstuk 5 - De Voorwaardelijke Herhaling



In dit hoofdstuk maak je kennis met de begrensde herhaling. Wanneer we computationeel denken, ontleden we een probleem en doen we aan patroonherkenning. Een van de patronen die we zo kunnen herkennen is een voorwaardelijke herhaling. De voorwaardelijke herhaling of de zolang-functie is een herhaling waarvan wij, niet op voorhand, weten hoeveel keer een of meerdere acties moet worden uitgevoerd.

5.0 - De Voorwaardelijke Herhaling

Noteer hieronder jouw oplossing / code. Markeer de while-functie of zolang-functie met kleur.

5.1 - Etappelijst Versie 2.0

Noteer jouw oplossing / code hieronder:

5.2 - Bosbranden Versie 2.0

Noteer hieronder jouw oplossing / code:

5.3 - Hittegolfalarm Versie 2.0

Noteer hieronder jouw oplossing / code:

5.4 - Interest Per Jaar Versie 2.0

Noteer hieronder jouw oplossing / code:

5.5 - Bevolkingsgroei Versie 2.0

Noteer hieronder jouw oplossing / code:

5.6 - Maximale Reistijd

Noteer hieronder jouw oplossing / code:

Wat hebben we geleerd?

In dit vijfde hoofdstuk maakte je kennis met de voorwaardelijke herhaling. Wanneer je een oplossing of algoritme ontwerpt, kan je patronen herkennen en een herhaling gebruiken om jouw oplossing efficiënter te maken. Een van die herhalingen is de **voorwaardelijke herhaling**.



De while-functie

De voorwaardelijke herhaling is een herhaling waarvan wij, niet op voorhand, weten hoeveel keer een of meerdere acties moet worden uitgevoerd. We weten enkel wanneer de herhaling moet stoppen. **Zolang** aan onze voorwaarde wordt voldaan, gaat het herhalen door.

Voorbeeld 1:

Elk jaar stijgt het aantal pinguïns in de kolonie met 10%. De wetenschappers en verzorgers moeten pas ingrijpen wanneer de populatie groter wordt dan 300 dieren.

Syntaxis:

```
var populatie = 100;  
while (populatie < 300) {  
    var populatie *= 1.10;  
}  
console.log('Alarm, teveel dieren in de kolonie!')
```

Hoofdstuk 6 - Uitbreidingsoefeningen

In deze reeks vind je uitbreidingsoefeningen terug. Deze oefeningen kan je maken om jouw kennis van het computationeel denken en programmeren in Javascript te oefenen. In deze reeks vind je oefeningen terug die alle programmeerconcepten (sequentie, selectie, begrensde herhaling en voorwaardelijke herhaling) toetsen.



6.1 - Sequentie Opdracht 1

Noteer hieronder jouw oplossing / code.

6.2 - Sequentie Opdracht 2

Noteer jouw oplossing / code hieronder:

6.3 - Selectie Opdracht 1

Noteer hieronder jouw oplossing / code:

6.4 - Selectie Opdracht 2

Noteer hieronder jouw oplossing / code:

6.5 - Begrenste Herhaling Opdracht 1

Noteer hieronder jouw oplossing / code:

6.6 - Begrenste Herhaling Opdracht 2

Noteer hieronder jouw oplossing / code:

6.7 - Voorwaardelijke Herhaling Opdracht 1

Noteer hieronder jouw oplossing / code:

6.8 - Voorwaardelijke Herhaling Opdracht 2

Noteer hieronder jouw oplossing / code:

Dit lesproject werd uitgewerkt voor educatief gebruik door Robbe Wulgaert (Sint-Lievenscollege), met oog op het bereiken van lesdoelen rond computationeel denken, ontwerpen van algoritmes en programmeerconcepten hanteren met de programmeertaal Javascript. Het maakt gebruik van frameworks rond computationeel denken van de UGent (Tom Neutens, 2022).

Dit lesmateriaal maakt gebruik van Dodona, Papyros en JS-fiddle. Hiervoor heb je een computer en browser nodig.

Heb je vragen over dit lesproject? Dan kan je contact opnemen via www.aiindeklas.be of www.robbewulgaert.be

Bronvermeldingen:

- Dodona: www.dodona.be
- Papyros: www.papyros.dodona.be
- JSfiddle: www.jsfiddle.net
- Iconen (licentie): www.flat-icons.com
- Foto's Dodona-opdrachten en GitHub: [Pexel](#)
- Foto syllabus: [Robbe Wulgaert](#)