**Full Stack Developer**

**Leren programmeren**

**T 1.1 - Inleiding**

**Steven De Ryck**

**September 2021**

**INHOUD**

[1 Wat is een computerprogramma? 3](#_Toc52220379)

[1.1 Definitie 3](#_Toc52220380)

[1.2 Vragen 3](#_Toc52220381)

[2 Wat is een algoritme? 4](#_Toc52220382)

[2.1 Definitie 4](#_Toc52220383)

[2.2 Voorbeelden van algoritmes 5](#_Toc52220384)

[2.3 Algoritme van Euclides 5](#_Toc52220385)

[2.4 Algoritme tegenover programma 6](#_Toc52220386)

[2.5 Vragen 6](#_Toc52220387)

[3 Succesvol programmeren 7](#_Toc52220388)

[3.1 Verschillende fasen 7](#_Toc52220389)

[3.2 Nassi-Schneidermann diagrammen 9](#_Toc52220390)

[3.3 Structorizer 11](#_Toc52220391)

[3.3.1 Taal kiezen 11](#_Toc52220392)

[3.3.2 Standaardteksten instellen 11](#_Toc52220393)

# Wat is een computerprogramma?

## Definitie

<https://nl.wikipedia.org/wiki/Computerprogramma>

Een **computerprogramma** is een opeenvolging van **instructies** met als doel om een bepaalde **taak** met een **computer** uit te voeren.

**Computerbroncode** wordt in het algemeen geschreven door computerprogrammeurs. Broncode wordt geschreven in een **programmeertaal**, […] Broncode kan met behulp van een **compiler** worden omgezet in een uitvoerbaar bestand (ook wel een **executable** of binair bestand genoemd) en later worden uitgevoerd door een **centrale verwerkingseenheid**. Als alternatief kunnen computerprogramma's worden uitgevoerd met behulp van een **interpreter**. […]

Een verzameling van computerprogramma's en bijbehorende data wordt **software** genoemd. […]

**Computerprogramma’s** zijn uitgeschreven **algoritmes** die door een computer uitgevoerd kunnen worden.

## Vragen

* **Welke programma’s gebruik je zelf?**

photoshop – illustrator – gimp – indesign – powerpoint – word – outlook – excel – chrome – youtube – spotify – banking apps - …

* **In welke apparaten die je dagelijks gebruikt, of tegenkomt, zit er software?**

computer – smartphone – spelconsole – auto – wasmachine – gsm – tv – audio-installatie – printer – koelkast? – …

# Wat is een algoritme?

## Definitie

<https://nl.wikipedia.org/wiki/Algoritme>

**Een algoritme is een eindige reeks instructies die vanuit een gegeven begintoestand naar een beoogd doel leidt. Algoritmen staan in beginsel los van computerprogramma's, al worden voor de uitvoering van algoritmen vaak computers gebruikt.**

Het **doel** van een algoritme kan van alles zijn met een duidelijk resultaat. De instructies kunnen in het algemeen omgaan met **eventualiteiten** die bij het uitvoeren kunnen optreden.

Algoritmen hebben in het algemeen stappen die zich herhalen (**iteratie**) of die beslissingen (**logica of vergelijkingen**) vereisen om de taak te voltooien.

Eenzelfde taak kan gewoonlijk met **verschillende reeksen instructies** worden opgelost. Het verschil ligt dan meestal in de hoeveelheid tijd, ruimte of inspanning die het algoritme vergt; dit is de **complexiteit** van een algoritme.

Een **algoritme** is een reeks instructies die vanuit een gegeven begintoestand naar een beoogd doel leidt.

## Voorbeelden van algoritmes

* Een **routebeschrijving**: bijvoorbeeld de kortste weg naar school
* Een **recept in de keuken**:  
  om aardappelsalade te maken kan het ene recept de instructie "schil de aardappel" bevatten en daarna de instructie "kook de aardappel". **Bij een ander recept kunnen die twee stappen omgedraaid zijn.** Beide recepten zullen echter vragen deze stappen voor alle aardappelen uit te voeren en het eindresultaat is een lekkere aardappelsalade.
* **In een doos Lego** zitten er heel wat losse stukjes. Het probleem is heel eenvoudig: “Hoe maken we met deze losse stukken dezelfde constructie als op de doos? Hiervoor gebruiken we de handleiding, een reeks van instructies die ons naar het gewenste resultaat begeleid.
* Wanneer je een **nieuwe smartphone** aankoopt, heb je steeds een **handleiding** (al dan niet op internet) om je te helpen probleemsituaties om te zetten in oplossingen, bv.:
* Hoe wis je alle berichten?
* Hoe bewaar ik een contactpersoon?

## Algoritme van Euclides

<https://nl.wikipedia.org/wiki/Algoritme_van_Euclides>

In de getaltheorie, een deelgebied van de wiskunde, is het algoritme van Euclides een efficiënte methode voor het berekenen van de **grootste gemene deler** (**ggd**) van twee positieve gehele getallen.

Het algoritme is vernoemd naar de Oud-Griekse wiskundige **Euclides van Alexandrië**, die het algoritme in de boeken VII en X van zijn Elementen beschreef.[1]

De grootste gemene deler van twee getallen is het grootste getal dat beide getallen zonder rest deelt. Het algoritme van Euclides is **gebaseerd op het principe dat de grootste gemene deler van twee getallen niet verandert als het kleinere getal van het grotere wordt afgetrokken**.

Zo is **21** de grootste gemene deler van **252** en **105** (252 = 21 × 12, 105 = 21 × 5) aangezien de grootste gemene deler van 147 (252 - 105) en 105 ook gelijk is aan 21. Aangezien men in het algoritme van twee getallen steeds het kleinste van het grootste aftrekt, kan men dit proces herhalen totdat een van de twee getallen gelijk wordt aan nul. **De grootste gemene deler is het resterende getal (dat niet nul is).**

## Algoritme tegenover programma

**Elk computerprogramma is eigenlijk één groot algoritme, maar dan omgezet in een programmeertaal**.

Vermits het algoritme van een programma soms iets te ingewikkeld is voor de programmeur om zomaar te programmeren, deelt hij het algoritme op in kleinere **deelalgoritmen**.

Al deze verschillende deelalgoritmen worden vervolgens geprogrammeerd in een programmeertaal en samengevoegd tot één programma.

Enkele voorbeelden van programmeertalen zijn:

* Javascript
* Java
* PHP

## Vragen

* **Welke talen ken je nog (van naam) ?**

C#

Python

Ruby

C++

C

Pascal

Visual Basic

Typescript

Swift

PHP

…

* **Waarom zijn dit geen programmeertalen?**

HTML

CSS

# Succesvol programmeren

## Verschillende fasen

Wanneer we een programma willen schrijven, zullen we altijd **eerst het algoritme uitwerken**.

We willen namelijk **niet in het wilde weg beginnen programmeren**. De verschillende stappen die we nodig hebben om tot een algoritme en vervolgens het programma te komen zijn als volgt:

***Analysefase (Analysis)***

* **Probleemstelling**   
  Wat is het probleem? Wat willen we als resultaat?
* **Probleemanalyse**
  + Welke gegevens worden er ingevoerd door de gebruiker (wat is de **input**)? (bv. een geboortedatum, de straal van een cirkel, …)
  + Welke verwerking moet er gebeuren (**proces**)? (bv. Wat is de formule om de oppervlakte van een cirkel te berekenen?)
  + Wat moet er uitgevoerd/getoond worden (**output**)? (bv. de leeftijd van de gebruiker, de oppervlakte van de cirkel, …)
* **Algoritme** **uitschrijven**
  + Controleer of het probleem moet opgesplitst worden in deelproblemen.
  + Maak voor elk deelprobleem een schema waarin alle stappen van begin tot einde zijn opgenomen.

***Ontwikkelingsfase (Development)***

* **Programmeren**

Zet je algoritme om in een programma. We schrijven het programma (of de code) in een programmeertaal.

* **Documenteren**

De code van een programma kan vrij ingewikkeld worden. Om er voor te zorgen dat jij of andere mensen de code later nog begrijpen, voorzie je **commentaar** in je code, eventueel bijkomende **handleidingen**, technische documenten, enz.

De levensloop van een programma is niet gedaan na de ontwikkelingsfase. Volgende fasen volgen meestal nog:

***Testfase (Testing)***

* **Testfase**

Controleer of je programma werkt door dit op verschillende manieren te testen.

* **Debuggen**

Fouten in een programma opsporen en oplossen, noemt men debuggen (de bugs eruit halen)

***Implementatie (Deployment)***

Het programma effectief in gebruik nemen, noemen we de **implementatie** ervan. Dat kan voor eigen gebruik zijn, maar ook voor andere gebruikers, klanten, het brede publiek, …

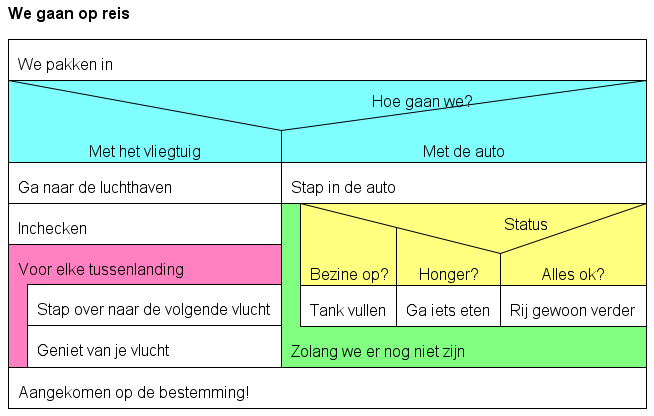
***Onderhoud (Support)***

Ook na de implementatie, tijdens het normale gebruik van programma’s, kunnen er nog onverwachte problemen opduiken die opgelost moeten worden, of kunnen bepaalde operaties al van bij de implementatie voorzien zijn (bv. archiveren, opkuisen, …). Meestal wordt van bij de implementatie voorzien hoe deze problemen aangepakt zullen worden; vaak is dit het onderwerp van een **onderhouds- of supportcontract**.

Ook alle **updates en upgrades** van programma’s vallen hieronder.

## Nassi-Schneidermann diagrammen

**Nassi-Shneiderman diagrammen** zijn een hulpmiddel om algoritmes uit te schrijven vooraleer men begint met programmeren.



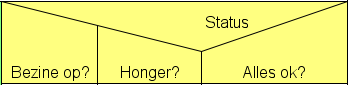
Ze maken gebruik van een zeer beperkt aantal vormen:

* Een **instructie**



Instructies worden van boven naar onder gelezen (en uitgevoerd). Uitzondering: herhalingen (zie hieronder)

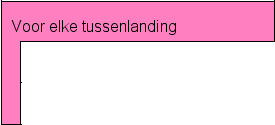
* Een **beslissing** (**if**-**else**-statement, **case**-statement)



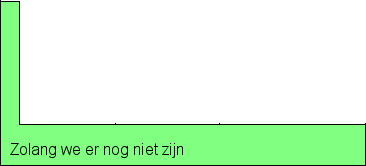
Soms zijn er verschillende mogelijkheden. In de driehoek staat de toestand of conditie die getest wordt. Daaronder staan de verschillende mogelijkheden.

* Een **herhaling** of **iteratie** (**for**-, **while**- of **repeat**-statements)

**0-n herhaling**: deze wordt 0 tot n keer uitgevoerd (dus mogelijk ook helemaal niet). In programmeertalen zijn dit (o.a.) de **for**, **foreach**, **while** loops.



**1-n herhaling**: deze wordt 1 tot n keer uitgevoerd (dus minstens 1 keer). In programmeertalen zijn dit (o.a.) de **do…while**, **until**,…loops.



## Structorizer

Het programma **Structorizer** is speciaal bedoeld om deze diagrammen te tekenen (met de computer).

<https://structorizer.fisch.lu/>

### Taal kiezen

**Preferences** -> **Language**

Kies Dutch

### Standaardteksten instellen

Ga naar **Instellingen** -> **Structuren**

