**Samenvatting Module 4 Informatica**

1. **Programmeertalen samenvatting**

Programmeertalen kunnen in zes hoofdcategorieën ingedeeld worden:

* Machinecode
* Assembleertalen
* Hogere programmeertalen
* Vierde-generatietalen
* Vijfde-generatietalen
* Objectgeoriënteerde talen
* **Machinecode** is de programmeertaal van de eerste generatie. De instructies bestaan uit nullen en enen. Deze taal wordt tot de lagere programmeertalen gerekend. We zeggen ook wel dat dergelijke talen 'dichter bij de processor' staan.
* Bij **assembleertalen** wordt er gebruik gemaakt van gemakkelijker te onthouden lettercodes in plaats van nullen en enen. Assembleertaal behoort tot de tweede generatie.
* Bij **hogere programmeertalen** (vanaf de derde generatie) worden de instructies door een compiler vertaald naar machinecode. Ze worden ook wel procedurele talen genoemd. Bekende derde-generatietalen zijn C, C++ en Java.
* De meeste talen van de **vierde generatie** zijn niet-procedureel, maar meer probleemgericht. Ze leveren vaak onhandelbaar grote programma's op. Met vierdegeneratiehulpmiddelen kan men wel snel een toepassing of een bepaald gedeelte hiervan genereren.
* Bij **vijfde-generatietalen** beschrijft de programmeur de situatie en het probleem en de oplossing wordt automatisch berekend. Deze manier van programmeren noem je declaratief programmeren.
* Bij **objectoriëntatie** worden gegevens en code gecombineerd in objecten. Objecten hebben eigen verantwoordelijkheden, houden data vast en vertonen gedrag. Belangrijke begrippen zijn: object, methode, attribuut, overerving, compositie en inkapseling.

**Object** = een op zichzelf staand geheel met een eigen taak.

**Attribuut** = waar de data van een object wordt opgeslagen.

**Methode** = als een functie aan een object is gekoppeld is het geen functie maar een methode.

**Overerving** =

**Inkapseling** = het principe dat objecten verantwoordelijkheden ingekapseld hebben, waardoor ze beschermd zijn tegen inkijk van andere objecten.

**Voordelen van objectoriëntatie:**

* Er is een kortere ontwikkelingstijd door hergebruik van programmamodules.
* Moeilijkere problemen zijn te behandelen door betere organisatie van programmacode.
* Het onderhoud van programma's is makkelijk omdat duidelijker is wat elk deel van de software doet.
* Er zijn betere uitwisselingsmogelijkheden door vastgestelde standaards.
* Er is eenvoudigere taakverdeling tussen meerdere programmeurs.

**Nadelen van objectoriëntatie zijn:**

* Het kost tijd en moeite om te leren programmeren in de OO-stijl.
* Voor kleine applicaties wegen de voordelen van objectoriëntatie soms niet op tegen het extra werk.
* Soms kun je te ver doorslaan in de **OO-stijl**.

**Derde-generatietaal** = taal C

Zijn objectgeoriënteerde nakomeling is C++.

**Java** is een OO-programmeertaal die geen platform nodig heeft.

Java bestaat uit drie onderdelen.

Visual Basic.NET is een programmeertaal en programmeeromgeving van Microsoft.

C# is een programmeertaal van Microsoft die kenmerken van C, C++ en vooral Java heeft.

Bij programmeertalen voor het web maken we verschil tussen client-sided en server-sided talen.

Client-sided wordt op de computer van de gebruiker uitgevoerd.

Server-sided wordt uitgevoerd op de server waarop de gevraagde website zich bevindt.

**Client-sided** zijn opmaaktalen (HTML en CSS) en scriptingtalen (waaronder JavaScript).

**Server-sided** is PHP.

1. **PSD’s maken samenvatting**

**PSD** = techniek waarmee je schema’s kunt maken

PSD = Programma Structuur Diagram

PSD wordt ook wel Nassi-Shneiderman Diagram genoemd.

PSD = schema dat de samenhang (de structuur) van instructies binnen een programma weergeeft.

PSD’s bestaan uit:

* Rechthoeken
* Driehoeken
* Tekst

PSD is een voorbereiding op het coderen in een programmeertaal.

Ze zijn programmeertaalonafhankelijk.

Er gelden geen strenge regels: **de probleemoplossing staat centraal.**

Alogritme = een set opdrachten om een berekening of handeling stapsgewijs een bepaalde volgorde uit te voeren.

Schrijven van opdrachten in een programmeertaal is de laatste stap in het systeemontwikkelingproces.

PSD’s vormen een hulpmiddel voor wanneer je een oplossing van het probleem moet bedenken.

Er zijn **3 basisstructuren**, als je die goed combineert kun je een willekeurig programmeerprobleem oplossen.

Basisstructuren:

* **Sequentie** (opeenvolging)
* **Iteratie** (herhaling)
* **Selectie** (keuze)

**Sequentie**

Sequentie wordt van boven naar beneden één voor één uitgevoerd:

|  |
| --- |
| opdracht 1 |
| opdracht 2 |
| opdracht 3 |
| opdracht 4, enzovoort |

Een voorbeeld bij het bakken van een ei:

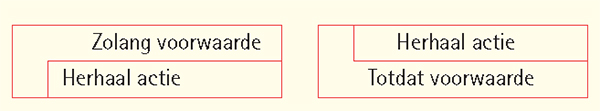
|  |
| --- |
| vuur aansteken |
| pan op het voor zetten |
| een halve eetlepel boter in de pan doen |
| boter laten smelten |
| ei breken en in de pan doen |
| ei gedurende drie minuten bakken |
| vuur uitdoen |

Dit noemen we dus ook een alogritme.

**Iteratie**

2 soorten iteraties:

* **Iteratie met controle vooraf**
* **Iteratie met controle achteraf**



Bij controle achteraf:

Bij controle vooraf:

|  |
| --- |
| vuur aansteken |
| pan op het vuur zetten |
| aantal gebakken eieren is 0 |
| *zolang* aantal gebakken eieren ongelijk aan 3   |  | | --- | | een halve eetlepel boter in de pan doen | | boter laten smelten | | ei breken en in de pan doen | | ei gedurende drie minuten bakken | | ei op bord leggen | | aantal gebakken eieren plus één | |
| vuur uitdoen |

|  |  |
| --- | --- |
| vuur aansteken |  |
| pan op het vuur zetten |  |
| aantal gebakken eieren is 0 |  |
| |  | | --- | | een halve eetlepel boter in de pan doen | | boter laten smelten | | ei breken en in de pan doen | | ei gedurende drie minuten bakken | | ei op bord leggen | | aantal gebakken eieren plus één |   *totdat* aantal gebakken eieren gelijk aan 3 |  |
| vuur uitdoen |  |

Verschil controle vooraf en controle achteraf:

Bij controle achteraf wordt de herhaling minimaal 1 keer uitgevoerd.

Bij controle achteraf wordt herhaald tot de voorwaarde waar is.

Bij controle vooraf wordt er herhaald tot de voorwaarde niet waar is.

**Selectie**

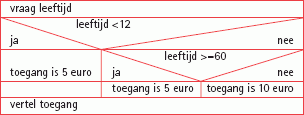
Selectie houdt in dat in het programmaverloop, je afhankelijk bent van een bepaalde keuzenmogelijkheid.

Welke kant het programma opgaat hangt van jou antwoord af.

In een PSD geef je dit weer in een driehoek en twee kolommen.

Voorbeelden:

https://www.instruct-online.nl/afbeeldingen/37036.gif



In het gedeelte dat herhaald wordt, mag een keuzediagram voorkomen

De vakken van een keuzediagram mogen nieuwe keuzes bevatten.

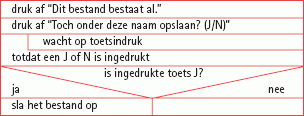
Enzovoort.

**Het nalopen van een PSD**

Een controle wordt ook wel een ooggetuigenverslag genoemd.

Het doel is tweeledig:

* Als de werking van het PSD bekend is controleer je of het PSD juist is.
* Als de werking van de PSD onbekend is onderzoek je wat het PSD doet.



Je ziet eerst twee sequentie-opdrachten. Het blijkt dat het hierbij gaat om iets wat op het scherm afgedrukt moet worden:

* een waarschuwing dat een bestand al onder een bepaalde naam op schijf aanwezig is
* de vraag of deze naam desondanks gehandhaafd moet blijven.

**3.1 Stapsgewijs programmeren**

**Imperatief programmeren** is het stapsgewijs in code omschrijven wat een programma moet doen, net als een algoritme of een PSD.

**Pseudocode** is alleen bedoeld om dingen makkelijker uit te leggen.

Vaak wordt bij pseudocode := gebruikt om aan te geven dat een variabele een waarde krijgt.

**If en Else**

**Niet elke If heeft een Else, maar wel elke Else een If.**

Voorbeeld van een If en Else:

ALS leeftijd < 12

DAN

Toegang := 5 euro

ANDERS

ALS leeftijd >= 60

DAN

Toegang := 5 euro

ANDERS

Toegang := 10 euro

EINDE-ALS

EINDE-ALS

**Lus**

Wanneer een programma een iteratie (herhaling) bevat, is er sprake van een lus (loop in Engels)

Bekendste lus = while-lus

Voorbeeld van een lus:

getal := 0

ZOLANG getal < 3

Getal := getal + 1

EINDE-ZOLANG

Er wordt een getal net zolang verhoogd totdat het niet meer kleiner is dan 3.

Oneindige lus = als de voorwaarde van een while-lus altijd waar blijft.

**Booleaanse expressies**

|  |  |  |
| --- | --- | --- |
| **Symbool** | **Naam** | **Gebruik** |
|  | groter-dan | x > y is true als x groter is dan y |
| >= | groter-dan-of-gelijk-aan | x >= y is true als x groter is dan y of net zo groot |
| < | kleiner-dan | x < y is true als x kleiner is dan y |
| <= | kleiner-dan-of-gelijk-aan | x <= y is true als x kleiner is dan y of net zo groot |
| not | not | not x is true als x false is en andersom; de Booleaanse waarde van x wordt geïnverteerd |
| and | and | x and y is true als x en y beide true zijn |
| or | or | x or y is true als x of y true is (of als ze allebei true zijn) |
| xor | exclusive or | x xor y is true als x of y true is (en niet als ze allebei true zijn) |

ALS leeftijd < 12 OF leeftijd >= 60

DAN

toegang := 5 euro

ANDERS

toegang := 10 euro

EINDE-ALS

**Syntax**

De syntaxis is de grammatica van een programmeertaal en beschrijft hoe de code er in een bepaalde programmeertaal uit moet komen te zien.

**Variabelen**

Soorten variabelen:

* Geheel getal (integer, ook wel int)
* Kommagetal (double of float)
* Karakter (character of char, bijvoorbeeld $)
* Boolean (bool, kan alleen maar true of false zijn oftewel 0 en 1)
* String (reeks karakters, bijvoorbeeld: ‘tijger’ of ‘fietswiel123’.

**Arrays**

**Array**s = lijst

Array = een geordende rij van waardes van hetzelfde type.

**Het volgnummer** = de index

Indexering begint met 0, niet met 1.

Voorbeeld van code:  
rij\_van\_getallen := [32, 59, 825, 9042]

**De indexering:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **rij\_van\_getallen** | | | | |
| **index:** | 0 | 1 | 2 | 3 |
| **waarde:** | 32 | 59 | 825 | 9042 |

**Functies**

Functies gebruik je als je een bepaald stukje code vaker wilt gebruiken zonder dat je de code opnieuw hoeft te schrijven.

**Het aanroepen van de functie** = als in het programma de naam van een functie staat, wordt daar het **subprogrammaatje** dat erbij hoort, uitgevoerd.

Van een functie moeten de volgende zaken worden bepaald:

* **De naam** (unieke naam is nodig. Sommige programmeertalen laten toe dat namen vaker voorkomen, zolang de parameters verschillen. Dit heet overloading)
* **De parameters** (speciale variabelen die een waarde krijgen bij het aanroepen van de functie)
* **Het returntype** (waarde-type dat de functie na uitvoeren oplevert)
* **De body** (code die uitgevoerd wordt als de functie wordt aangeroepen)

Voorbeeld:

som(int getal1, int getal2) : int

int optelling := getal1 + getal2

return optelling

De functie 'som' telt twee getallen van het type 'int' bij elkaar op en geeft het resultaat (wat hier de naam 'optelling' heeft gekregen) als int terug. De termen 'getal1' en 'getal2' zijn de parameters en worden gescheiden door een komma. We kunnen nu naar deze functie refereren, of anders gezegd: we kunnen deze functie aanroepen en het resultaat in een andere variabele stoppen. Bijvoorbeeld:

int getalA := som(10, 20)

int getalB := som(getalA, 70)

int getalC := som(som(50, 50), getalB)

Als bovenstaande code uitgevoerd zou worden zou na afloop getalA de waarde 30 hebben, getalB 100 en getalC 200.

**Parameters en argumenten**

Bij het aanroepen worden parameters ingevuld door argumenten.

**Argument** = een waarde die meegegeven wordt bij een functieaanroep.

Een functie hoeft niet per se een returnwaarde te hebben. Dit geef je aan met **void**.

Void is eigenlijk geen variabele-type, maar de afwezigheid van een waarde.

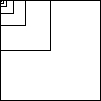
zegHallo() : void

print(“hallo”)

Deze functie heeft geen parameters en geen returnwaarde.

**Recursie**

Wanneer een patroon zich binnen zichzelf herhaalt is er spraken van **recursie**.



Voorbeeld:

MijnFunctie() : void mijnFunctie()

**Overflow** = als een recursieve functie zichzelf oneindig herhaalt en het crasht.

**Stopconditie** = punt waarmee het recursieve proces tot stoppen gebracht wordt.

faculteit(int getal) : int

if(getal = 1) return 1

return getal \* faculteit(getal - 1)

Voorbeelden van **functionele programmeertalen** zijn Lisp, Haskell en Clojure.

**4. Versiebeheersystemen**

**Versiebeheersysteem** = Version Control System (VCS)

Versiebeheersysteem = systeem dat het makkelijker maakt om software te ontwikkelen.

Met een VCS kunnen programmeurs makkelijker tegelijk aan een project werken.

**Archief** = repository (opslagplaats)

**Archief** = een plaats waar ontwikkelaar hun toevoegingen naartoe sturen

Repository heeft altijd een versienummer

**Working copy** = een kopie van het systeem dat je nodig hebt om er zelf aan te werken

**Voordelen van versiebeheersystemen:**

* Meerdere mensen kunnen makkelijker aan hetzelfde project werken.
* Het archief is een back-up van het project.
* Van elk apart bestand binnen het project worden eerdere versies bewaard.
* Je kunt zien wat er veranderd is na een toevoeging.
* Je kunt zien wie wanneer iets heeft veranderd.
* Je kunt zien wat ieders bijdrage is aan het project.

**Client-server principe** = 1 centrale opslagplaats

Veelgebruikte versiebeheersystemen:

* Concurrent Versions Systems (CVS)
* Bazaar
* Mercurial
* Revision Control System (RCS)
* Git
* Subversion (SVN)

Je kunt via **commandline** (cmd in windows) werken met versiebeheersystemen.

Nadeel hiervan: je moet de commando’s weten om hiermee te kunnen werken.

**Belangrijke termen:**

* **Checkout**
* **Commit**
* **Update**
* **Conflict**
* **Revert**

**Checkout** = voor de eerste keer alle bestanden uit de repository kopieert naar jouw working copy.

**Commit** = toevoegingen wanneer een ontwikkelaar een aanpassing doet.

**Update** = wanneer iemand bestanden in de repository naar zijn of haar working copy download.

**Conflict** = wanneer iemand bijvoorbeeld een commit doet op een bestand, terwijl een andere persoon ook wijzigen heeft aangebracht in dit bestand.

Dan zijn er 2 uitkomsten:

* De wijzigen worden samengevoegd
* Er treedt een conflict op. Dan kunnen ze niet worden samengevoegd.

**Revert** (terugzetten) = een of meerdere bestanden worden teruggezet naar de versie van de repository.

Met een revert los je makkelijk een conflict op.

**5. Testen**

**Trial-and-error manier** = code schrijven, uitvoeren, foutjes tegenkomen en verbeteren. Dit herhalen totdat de code doet wat het moet doen.

**Software testen** = doelgericht zoeken naar fouten en afwijkingen van de eisen van de software.

Mindset tester: zo veel mogelijk fouten vinden

Mindset programmeur: zo min mogelijk fouten maken.

**3 testelementen om te kunnen testen:**

* **Het testobject**: het te testen systeem.
* **De testbasis**: de eisen aan het systeem.
* **De testmethode**: de manier waarop het getest wordt.

Voorbeeld van testen:

Je mag een naam van maximaal 20 karakters gebruiken, de tester voert dan een naam in van 20 karakters en 21 karakters.

**Statisch testen**: het testobject hoeft niet te draaien

**Dynamisch testen**: het testobject moet draaien

Bij dynamisch testen wordt getest of het gedrag van het testobject overeenkomt met de testbasis.

**Blackbox**: de tester heeft geen kennis van hoe het systeem is gebouwd

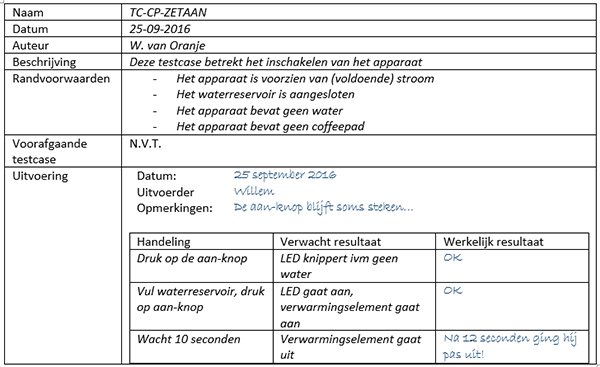
**Whitebox**: de tester heeft wel kennis van hoe het systeem is gebouwd

**Handmatig testen**: een tester neemt een systeem en gaat hier actief mee werken.

Bij een goede test wordt elke functionaliteit aangeraakt en wordt gecontroleerd of het testobject aan de systeemeisen voldoet. Er wordt ook echt gezocht naar fouten.

**Testcase** = nauwkeurige beschrijving van de testmethode van een functionaliteit of veelvoud van functionaliteiten.

Een testcase beschrijft deze onderdelen:

* Unieke naam of ID
* Datum waarop testcase is gemaakt
* Auteur van testcase
* Korte beschrijving van testcase
* Randvoorwaarden en condities (middelen die beschikbaar moeten zijn of dingen die gedaan moeten worden voor de testcase)
* Voorafgaande stappen of testcases
* De stappen van de testcase
* De verwachte uitkomst van elke stap

Wanneer een testcase uitgevoerd wordt, worden deze zaken ingevuld:

* De datum waarop uitgevoerd
* De persoon die het uitvoert
* Het resultaat van elke stap
* Eventuele opmerkingen of conclusies

**Systeemtest** = hele systeem wordt getest op de opgestelde eisen.

Systeemtest = blackboxtest, omdat de tester geen kennis behoort te hebben.

Systeemtest = dynamische test, omdat het hele systeem wordt getest en dus in werking moet zijn.

**Acceptatietest** = test die wordt gedaan door de opdrachtgever, het is bedoeld om te controleren of het voldoet aan de eisen.

Acceptatietest = dynamisch en blackbox

**Automatische tests** = soort geprogrammeerde testcases.

Er wordt een script doorlopen waarbij een deel van het testobject aangeraakt wordt, daarna wordt gecontroleerd of de uitvoer overeenkomt met de verwachtingen.

De kracht van automatische tests: voortdurend uitgevoerd zonder dat het extra tijd kost.