Eerste probeersel

Fontys ICT sem1

2023-09-20

Table of contents

Preface **4**

1 Introduction **5**

2 softwarematerial - Overzicht **6**
2.1 Inleiding 6
2.2 Structuur van dit materiaal 6
2.3 Eerste Hulp Bij Vasthangen 6
2.4 Personas / Volgorde 7
2.5 Onderdelen 7
2.5.1 1. Computational Thinking 7
2.5.2 2. Toolbox Basis: Basisconcepten van programmeren (zonder objecten) 7
2.5.3 3. Toolbox Verdiepend: Objects en Classes 8
2.6 Diversen 8
2.6.1 Software Engineering 8
2.6.2 Het Software Profiel bij FHICT 8
2.6.3 verder… 8
2.6.4 Waar vind ik dit materiaal 8

3 Computational thinking **9**
3.1 Wat is Computational Thinking? 9
3.2 Training 9
3.2.1 Binair 1 9
3.2.2 Binair 2 10
3.2.3 Getal raden: hoger lager 10
3.2.4 Project Euler Challenge 79 10
3.2.5 Maken van een Algoritme / Flow Chart 10
3.2.6 Renteberekening 12
3.2.7 Deelbaarheid 12
3.2.8 Priem 1 12
3.2.9 Priem 2 12
3.2.10 Flash card 12
3.2.11 Stamboom 13
3.2.12 Tafeltjes 13
3.2.13 Rekenspelletje 13

3.2.14 Anagram 1 13
3.2.15 Anagram 2 13
3.2.16 Koffieautomaat 14
3.2.17 De niet-koffieautomaat 14
3.2.18 Eigen opdracht verzinnen 14
3.3 Bronnen 14

4 Summary **15**

References **16**

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

2 softwarematerial - Overzicht

2.1 Inleiding

Dit is een verzameling (open) software-materiaal van het startsemester FHICT (Fontys Hogeschool voor ICT). Het is openbaar toegankelijk voor iedereen en vrij van copyright. Het is in de loop der jaren gemaakt en verzameld door docenten, met hier en daar toevoegingen door studenten! FHICT verwijst naar dit materiaal vanuit een canvas-course, daar staat de FHICT-specifieke info, terwijl deze github-pages redelijk FHICT-onafhankelijk zijn.

2.2 Structuur van dit materiaal

In het theorie-meteriaal onderscheiden we: + `Tutorial`: practical steps, learning oriented. + `How-to guide`: practical steps, problem oriented. + `Discussion`: theoretical, understanding oriented. + `Reference`: theoretical, information oriented. Accurate and complete.

De opdrachten zijn te verdelen in: + `Training` (voorheen exercises): 'vingeroefeningen', horen bij een stukje theorie om te oefenen, maar zijn als bewijsmateriaal om je docent te overtuigen niet te gebruiken. + `Wedstrijd`: Om je docent te overtuigen dat je het niet alleen snapt maar ook doet ga je zelf opdrachten verzinnen, variëren en uitbreiden op bestaande opdrachten!

Zie meer uitleg over de uitgangspunten.

2.3 Eerste Hulp Bij Vasthangen

- How-to: Technisch-inhoudelijke vragen vind je in de How-to. Staat je antwoord er niet bij? Laat het horen! (ook als je het antwoord zelf al hebt gevonden. Vast dank!)
- how to Look At A Tutorial

- Frequently Asked Questions: Als je vragen hebt over de cursus bij FHICT: kijk eerst eens in de FAQ of anderen dezelfde vraag ook al hadden.
- some more really powerful help can be found here

2.4 Personas / Volgorde

Vergelijk jezelf met de personas en lees de tips: Heb je al programmeerervaring? Of juist nog helemaal niet?

intro

2.5 Onderdelen

2.5.1 1. Computational Thinking

De manier van denken, goed om iets van te weten… Oorspronkelijk gebruikt als materiaal om kennis met elkaar en met de docent te maken. Werkt het best al discussiërend met stift bij het whiteboard.

- Computational Thinking

2.5.2 2. Toolbox Basis: Basisconcepten van programmeren (zonder objecten)

Een introductie in programmeren (nog zonder objecten).

- Toolbox basis

2.5.3 3. Toolbox Verdiepend: Objects en Classes

Verdieping, programmeren met objecten

- Toolbox Verdiepend: Objects en Classes

2.6 Diversen

2.6.1 Software Engineering

Dit betreft een aantal niet-programmeer aspecten die een beginnend software engineer moet kennen en kunnen.

- Software Engineering

2.6.2 Het Software Profiel bij FHICT

- Het Software Profiel

2.6.3 verder...

- Motivatie, Concentratie, thuiswerken: Niet onbelangrijk tijdens een gemiddelde corona-crisis...
- Workshops

2.6.4 Waar vind ik dit materiaal

- op Github pages (hier het meest leesbaar) of in een
- Github-repository (pull requests met verbeteringen welkom)

3 Computational thinking

3.1 Wat is Computational Thinking?

We bekijken enkele voorbeelden:

Als je een trap wil stofzuigen kun je het beste van boven naar beneden werken. Als je een vloer schildert (zie Donald Duck), zorg dan dat je bij een deur eindigt. Denk voor je iets doet na over de volgorde en gevolgen van acties.

Stel ik plan een trip met vrienden met het OV naar een event in een andere stad. Ik kan dat opsplitsen in: - Waar zien we elkaar? - Hoe kom ik bij het station?
- Hoe laat vertrekt de trein, hoe lang duurt het, hoe betaal ik? - Hoe kom ik vanuit het station bij het event? - en je kunt vast nog wel een aantal onderdelen verzinnen…

Om een groot 'probleem' op te lossen is het handig het probleem in kleinere problemen op te lossen.

Je ziet dat computational thinking ook 'buiten de computer' speelt.

3.2 Training

Hieronder een aantal opdrachten waarmee je zogenaamde 'computational thinking' wordt uitgedaagd.

3.2.1 Binair 1

De digitale wereld is opgebouwd uit nullen en enen, '0' en '1' is alles wat een computer kent. Oh ja? Ja.
Deze opdracht introduceert het binaire tellen:

- Binary01

3.2.2 Binair 2

Kun je tot 31 tellen op 1 hand? Typ **'count to 31 on one hand'** in je favoriete zoekmachine. Tot hoever zou je dan kunnen tellen als je twéé gebruikt?

3.2.3 Getal raden: hoger lager

Hoger lager.

3.2.4 Project Euler Challenge 79

De uitdagingen van Project Euler zijn vaak problemen die het best met een computer op te lossen zijn. Euler79 is echter ook zonder hulp van een laptop op te lossen: op whiteboard bijvoorbeeld!

- ProjectEuler 79

Misschien vind je nog wel andere Euler-challenges waar je bijvoorbeeld een flow chart voor een mogelijke oplossing voor kunt verzinnen.

3.2.5 Maken van een Algoritme / Flow Chart

Als je software bouwt ontwerp je vaak een algoritme dat een probleem oplost.
Dit algoritme kun je op verschillende manieren beschrijven. Wij willen voor de oriëntatie dat je flow charts maakt. Schrijf deze op white board (eventueel op papier) en lever een foto daarvan in.

Een goede oefening in het maken van een algoritme is de opdracht 'algo01'. In het document wordt eerst iets uitgelegd over 'flow charts', daarna maak je een algoritme voor Kruisje-Rondje:

- Algo01

Voor mensen die nooit hebben geprogrammeerd is Kruisje Rondje een prima keuze, maar als je al ervaring hebt, of heel goed bent: 4 op een rij, dammen, schaken, rubik's cube zijn mogelijke alternatieven! (als je 1 van deze kiest zul je die misschien niet helemaal af krijgen) Misschien kun je zelf nog iets anders verzinnen wat jou meer uitdaging geeft: Overleg dan met je docent.

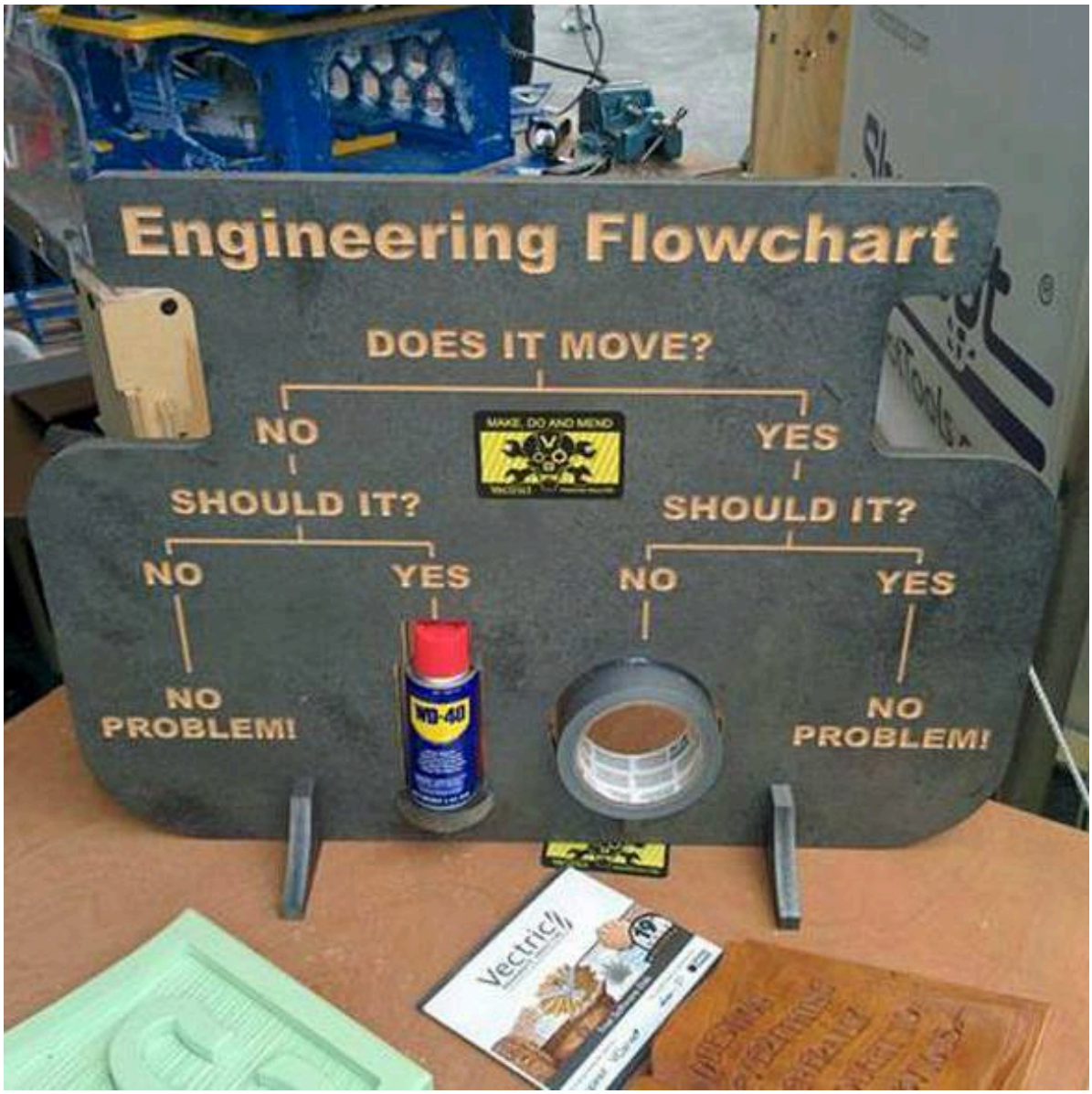

Figure 3.1: fig:flowchart

3.2.6 Renteberekening

Gegeven een startsaldo (bijvoorbeeld 234 euro), een rentepercentage en een aantal (zeg n) jaren. Maak een flow chart voor een programma dat berekent hoeveel het saldo na 'n' jaar is.

3.2.7 Deelbaarheid

Maak een algoritme dat het kleinste getal bepaalt dat groter is dan 1000 en dat deelbaar is door 37.

Algemener: Maak een algoritme dat het kleinste getal bepaalt dat groter is dan een gegeven waarde (de 'threshhold') en dat deelbaar is door een te geven 'deler'.

3.2.8 Priem 1

Maak een flow chart voor een algoritme dat van een getal bepaalt of het een priemgetal is.

3.2.9 Priem 2

Maak een flow chart voor een programma dat alle priemgetallen tussen 1000 en 2000 opsomt.

3.2.10 Flash card

Een flash card programma is een programma dat jou helpt bij het leren van (bijvoorbeeld) Franse woordjes.

Flash Cards waren van oudsher op papier of karton gemaakt. Op de voorkant kan dan bijvoorbeeld het Nederlandse woord staan en op de achterkant het Franse woord.

Simpele vorm: neem een willekeurig kaartje, laat dit zien, laat gebruiker een woord invoeren, antwoord of dat de goede vertaling is, en dat een aantal malen herhalen!

Het wordt wat uitdagender als jouw algoritme bijhoudt hoeveel vragen er al zijn gesteld, hoeveel daarvan er goed beantwoord waren en hoeveel procent dat is.

Als je hiermee bezig bent kom je vragen tegen. beantwoord deze zoals je zelf denkt dat goed is. Praat erover met anderen om je heen waar dat nodig is om tot goede beslissingen te komen.

3.2.11 Stamboom

Gegeven een stukje stamboom. Van de personen in de stamboom is gegeven of ze mannelijk danwel vrouwelijk zijn en verder wie hun (biologische) vader en moeder zijn.

- Verzin een manier om dit in een schema op te schrijven. Laat dit zien door op papier een schema te maken voor een familie van tenminste 15 personen (neem bijvoorbeeld een stuk van je eigen familie).
- Hoe kun je in het schema zien van een persoon wat het geslacht is en wie de vader/moeder is.
- Van een persoon wil ik op kunnen vragen wie de (directe) kinderen zijn.
- Hetzelfde voor alle nazaten, daarmee bedoelen we de directe en indirecte kinderen (kleinkinderen, kleinkleinkinderen,…).

3.2.12 Tafeltjes

Een standaardoefening voor programmeurs is het tonen van de tafeltjes, dus iets als:

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
en zo voort.
```

Maak hier een flow chart voor.

3.2.13 Rekenspelletje

Programma verzint 2 willekeurige getallen, gebruiker moet het product bereken en invoeren. Programma geeft aan of dit goed is of fout. Maak hier een flow chart voor.

3.2.14 Anagram 1

Bepaal of 2 strings anagrammen van elkaar zijn. Maak hier een flow chart voor.

3.2.15 Anagram 2

Genereer alle anagrammen van een gegeven string. Maak hier een flow chart voor. Als je niet weet wat een anagram is: je weet vast wel een manier om dat uit te vinden?

3.2.16 Koffieautomaat

Denk eens na over wat een koffieautomaat doet. Hij wacht op muntjes, registreert je keuze, maakt dan wat jij gekozen hebt. Maak hier een flow chart voor.

3.2.17 De niet-koffieautomaat

In het dagelijks leven heb je nog met meer apparaten te maken. Kun je er een verzinnen met een interessant 'gedrag'? Maak er een flow chart voor. Bijvoorbeeld: printer, scanner, copier.

3.2.18 Eigen opdracht verzinnen

Misschien kun je zelf iets verzinnen met **and / or**

3.3 Bronnen

- binair rekenen
- Computational Thinking
- brilliant.org

4 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.