

INFORMATIVE INFORMATICS

Profiel

- C&M
- E&M
- N&G
- N&T

made by Vincent & Christiaan
Ruijgrok van der Haven
under supervision of Peter Ypma



```
1 from django.db import models
2 from django.contrib.auth.models import User
3 from education.subjects import ComputerScience
4
5 class Groteprogrammeer(models.Model)
6     name = models.CharField(default="Grotepro
7
8     authors = [
9         'Vincent Ruijgrok',
10        'Christiaan van der Haven'
11    ]
12
13    link = "www.groteprogrammeer.nl"
14
15    def str(self):
16        return self.title + "-" + ComputerScience.subtitle
```


Abstract

Computer science is becoming more crucial every day. As processes and productions are digitalised, the demand for software developers and computer scientists has skyrocketed. In the Netherlands, computer science is not an obligatory subject in pre-university education. We anticipate that it will - shortly - be imperative for all students to learn how to program.

We set out to create a course for students in Dutch high school, as fits such a digital subject, we did not create an actual textbook, rather we built a website. This site supports all students in learning informatics. The course starts with an overview of what a computer is and how it works. It transitions to Scratch: a graphical programming language to let students scrutinise programming concepts as functions, loops, and conditions. In the second semester, the students start with actual coding in the language C, which comes close to the hardware, which ensures a deep understanding of the mechanism of a computer.

The entire course will, eventually, consist of eleven semesters. Currently, the first two have been designed, and are close to a finished state. The other nine may follow later.

Contents

Abstract.....	2
Prologue	8
Acknowledgment.....	9
Introduction	10
Chapter 1 – The Front-end	13
1.1 The front end	13
1.2 HTML.....	13
1.2.1 The Markup language.....	13
1.3 CSS	14
1.3.1 The styling language.....	14
1.4 Front-end Frameworks	16
1.5 JavaScript	16
1.5.1 The JavaScript basics	17
1.6 JavaScript frameworks and libraries	18
1.6.1 jQuery.....	19
1.7 CDN and other installation methods.....	19
1.8 JSON and APIs	20
Chapter 2 – Designing a website using Google’s Material Design	21
2.1 The Graphical User Interface	21
2.2 Design Philosophies	21
2.3 Google's Material Design	21
2.3.1 Surfaces, Elevation and Light.....	21
2.3.2 Layout, grid and spacing.....	22
2.3.3 Navigation System	23
2.3.4 Colour System	24
2.3.5 Typography.....	26
2.3.6 Motion and Animations.....	27
Chapter 3 – Building a website using Python and Django	29
3.1 Languages	29
3.2 Python.....	29

3.2.1 Using Python for making a website	29
3.2.2 Flask.....	30
3.2.3 Django	30
3.3 Django – the powerhouse of webdev	30
3.3.1 Webapps	31
3.3.2 URLs.....	32
3.3.3 Views	32
3.3.4 Templates.....	33
3.3.5 Models and data.....	33
3.3.6 The great Django mixtape	35
3.3.7 Forms and input	37
Annotation to 3.3	38
Chapter 4 – Correctness is not everything	39
4.1 The structure	39
4.2 Auto grading	40
4.3 Documentation	44
4.4 GroteProgrammeer.nl.....	45
4.4.1 Hosting	45
4.4.2 Domain registrar.....	45
Chapter 5 – Creating the front end	46
5.1 In the beginning: Researching and learning about the front-end	46
5.1.1 Researching Material Design	46
5.1.2 Vuetify	46
5.1.3 Vue	46
5.2 The departure: Ditching Vue and Vuetify after a month.....	47
5.2.1 Material’s own framework	47
5.2.2 Semantic UI	47
5.2.3 Falling in love with jQuery	47
5.3 The law: Applying material design	48
5.3.1 Applying colour.....	48
5.3.2 Navigation	49

5.3.3 Applying elevation	49
5.4 The numbers: JavaScript and jQuery	49
5.5 Reflection and conclusion	50
Chapter 6 – How do others teach CS?	53
6.1 CS50	53
6.1.1 Structure.....	53
6.1.2 Problem sets.....	55
6.1.3 Languages.....	56
6.1.4 Teaching style.....	57
6.1.5 Online teaching done right.....	58
6.2 CSCircles.....	59
6.2.1 Screenshots	59
Chapter 7 – Teaching, teaching, teaching	64
7.1 Method	64
7.2 David Heerkens, a physics teacher.....	64
7.2.1 Explanations	64
7.2.2 Systematische natuurkunde	65
7.2.3 Videos.....	65
7.3 Lars van Bokhorst, a chemistry teacher	65
7.3.1 Exams	65
7.3.2 Practical exploration.....	65
7.4 Matthijs Alderliesten, a physics teacher	66
7.4.1 The book.....	66
7.4.2 Learning routes	66
7.4.3 Exploration	66
7.5 Kees van Vliet, a mathematics teacher	66
7.5.1 Numbers and Space.....	67
7.5.2 Repetition, repetition, repeti.....	67
7.6 Summary.....	67
Chapter 8 – Ideal vs Reality	68
8.1 The syllabus.....	68

8.1.1 Rijksoverheid	68
8.1.2 Q1: Computers & Scratch	69
8.1.3 Q2: C.....	69
8.1.4 Q3: Python.....	70
8.1.5 Q4: Projects	70
8.1.6 VWO5	71
8.1.7 VWO6	71
8.2 The ideal	72
8.3 The reality	74
8.3.1 The simple things	74
8.3.2 The harder things	75
8.3.3 The things that never made it	75
8.3.4 Things we learned from others	76
Chapter 9 – The education; the website; the product	78
9.1 Separating the website	78
9.1.1 The student environment.....	78
9.1.2 The teacher environment.....	79
9.1.3 The guest environment	80
9.2 Documentation	81
9.2.1 Why use a documentation	81
9.2.2 How made a documentation	81
9.3 Non distracting, yet play and colourful	82
9.3.1 The low-profile navbar	83
9.3.2 Clean and sterile.....	83
9.3.3 Keywords.....	83
9.3.4 Code	84
9.3.5 The questions	84
Conclusion	88
Corresponding questions	88
Discussion.....	90
References	92

Images used	97
Annotation to images used	99
Appendix A – GitHub and website	100
Appendix B – Logbook	101
Vincent’s logbook	101
Christiaan’s logbook.....	104
Appendix C – Interviews	107
Interview with David Heerkens	107
Interview with Lars van Bokhorst.....	116
Interview with Matthijs Alderliesten.....	125
Interview with Kees van Vliet.....	131
Appendix D – Some lessons we made	139

Prologue

You have, in any way, acquired our school thesis "*Informative Informatics*". For this thesis we researched how to create a website by programming and how to teach computer science most effectively. Finally, we realised those thoughts by creating a website on which students can learn computer science.

Since the summer holiday of 2020, we have been programming together. They started with trying to create a terrain generator using a noise map, then got around to implement some artificial intelligence and create command-line games with Python and C.

We saw this thesis as our chance to learn something new, something we had not done before. Normally, while you are a student in school, there is not much time left between studying, doing homework, working, and sporting. Now that we did, in any case, need to do this thesis, why not use this as an excuse to do something you wanted to do for a long time, like programming?

We had always known that web development was a hot topic in the developer world. When we started outlining our ideas, the first thing we knew was that this was going to take a long time, and it has, but every hour of it has been a blast!

Although this thesis may not be like any other thesis you have read or will read, we sincerely hope that you will enjoy our exploration through the jungle that is web development and computer science education. We hope you like it; in any case, we have enjoyed making it!

Sincerely,
Christiaan van der Haven and Vincent Ruijgrok

September 29th, 2021
Gouda

Acknowledgment

We could not have made this thesis without the help of some people. First, we want to thank Peter Ypma, who has helped us comprise all of our wild ideas into one coherent thesis. We want to thank him for all his time, his endless reviews, and infinite critiques, which helped our raise the level of this thesis.

We want to thank David Malan, Brian Yu, and the rest of the CS50 team¹ over at Harvard University in Cambridge, Massachusetts. We could not have created the website and course if it were not for their immensely well-structured set of courses. We want to thank Brian Yu for his ever so useful course on web programming and thank David Malan for his inspired way of teaching in CS50.

We also want to thank Jorijn for recommending us Google's Material Design and helping us with choosing the colour palette for the website.

We want to thank Jistke van Eijk for her review on the grammar and spelling in the lessons and inspiring us in general.

And lastly, but not of least importance, we want to thank David Heerkens, Lars van Bokhorst, Matthijs Alderliesten and Kees van Vliet, who have happily and extensively answered our questions about teaching and textbooks.

¹ <https://cs50.harvard.edu/x/2021/staff/>

Introduction

"You two should try that too!", said Jitske. "What do you mean?", asked Christiaan, who was not all that quick. Meanwhile Vincent had already started thinking, as he often does. Normally this takes about half a minute until he comes to an ingenious idea, but this time it took quite a bit longer...

The three friends went for a walk, as they often did during the pandemic. They had become friends recently and were already getting along really well. Vincent had become friends with Christiaan during the start of the pandemic. They both liked programming, so naturally they started doing small programming projects together; they explored the wide world of programming by trying out a great many things. They did projects like: proving Hilbert's Hotel, a terrain generator, and numerous other small games.

"I think she means that we should also try to create our own business", said Vincent to Christiaan. They had been talking about how Ypma and a few other teachers had made some very successful businesses before changing their career to teaching at the Goudse Waarden. Creating a business is one thing, but making it successful is another thing entirely, and Christiaan knew that. "If we want to create a successful business like them, we should first have an idea; we should create something entirely new, something that has not been done yet, or not done often". Vincent now also realised this, a fun idea for a project is not enough, they would need something more. Vincent had many unfinished projects lying around just like Christiaan and many others like them. Starting and fantasizing about a project is way more fun than actually doing it. It requires lots of time and willpower to push yourself to work on a large project, and after a few weeks the bright glow of fantasizing about the result starts to wear off and new, more exciting ideas start to form; before you know it, you have already started a new project and pushed the other one to the side. This cycle continues and continues like a wheel crushing all the projects beneath it. For Vincent and Christiaan to start a large project they would not only have to have a good idea, but they would need to destroy the wheel.

While Vincent and Christiaan talked on and on about things that have not been invented yet, Jitske was thinking way simpler. "Can't you just combine two of the things you like doing and make something out of that? Just combine programming with helping people", she said. "Like creating an app which helps people with eating a healthier breakfast?", asked Christiaan. "No, more along the lines of teaching people. Teaching people about, for example, programming. For people like me, who don't know much about computers", said Jitske; and with that said she gave the first spark to the great pile of wood drenched in oil that is Vincent and Christiaan. Someone had to give that spark, that idea; a spark which they themselves could not have made, only Jitske could do that.

Preface

Many people have jokingly told us that we made two theses, not only with the time we spent on it, but also with our subject(s). Therefore, we have split our thesis into two parts:

The first part is about creating a website from the ground up. That means we want to program everything, from the server side of the website that will handle things like user accounts and data, to the front of the website; the part which you can see and interact with.

The second part is about creating a computer science course for Dutch students aged 15 - 18. We want to achieve this by first looking at what other people have done like Harvard University's CS50, but also by interviewing the best teachers (in our opinion) at our own school. Then we draw conclusions from that and design our own computer science course.

At this point we have both managed to create a working website and a part of the course. The project is far away from finished, but we have achieved the things we wanted to achieve for this thesis.

For this thesis we did not define a research question, instead we set up a research goal.

*"Creating an online learning platform to assist students
in learning and teachers in instructing computer science
in pre-university education."*

To achieve this goal, we defined four corresponding questions:

- "What are effective ways to instruct unexperienced students; are there any computer science-specific ways?"
- "How does one make a website accessible and understandable for all?"
- "How to create a robust server that handles users' input?"
- "How can one make students excited for the subject computer science?"

Part 1

An online translocation of information.

How to create an educational website.



Figure 1 For an educational website, you need many elements to let students explore a topic. (Best Educational Websites, 2017)

Chapter 1 – The Front-end

Every good website has a Graphical User Interface (GUI), meaning that you can see things on a webpage and interact with them. There are complicated websites, but also very simple ones. There are beautiful GUIs, but also ugly ones. They all have one thing in common: they make use of a front-end to display the interface.

1.1 The front end

To explain it in short, the front-end of a website is the part being rendered by the user's computer. The back-end renders on a server and sends the result to the user in the form of *HTML* which in turn is read and displayed by your browser. So, the 'back'-end is in the back of the website on a server, a place which the user cannot reach (if done correctly). The front-end instead sends the user the website, which is then calculated and rendered on the user's computer. So, the 'front'-end is the front the website, the user can reach, interact, and change its code. So, the 'front'-end is the front the website, the user can reach, interact, and change its code. Making storing user data, like passwords, on the front end a bad idea. However, the front-end is useful for designing all the visible parts of a website because those things do not have to be protected. And that is what the front end is used for mostly. (Pastorino, 2020)

1.2 HTML

HTML, or Hyper Text Markup Language, is the most important coding language for creating a website. HTML was created by Tim Berners-Lee in 1991 for use at CERN, Tim Berners-Lee also founded the World Wide Web. HTML is supported by all modern browsers and has basically become a standard for all websites. It uses very easy markup language so anyone can learn to create websites, but also to keep it extremely flexible. HTML uses tags to indicate what an element should be and how it should look. HTML can become complicated very quickly when using other languages like *CSS* or *JavaScript*. They are popular languages built into browsers so they can interpret them and change the HTML accordingly. (Wikipedia contributors, 2021b)

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>My First Blog Website</title>
5  </head>
6  <body>
7    <h1>Welcome to my blog website!</h1>
8  </body>
9  </html>
10

```

Figure 2 An example of a simple HTML page.

1.2.1 The Markup language

Figure 2 and figure 3 will be used as an example in the following explanation. The explanation will show you how you can create a website which lets users create simple blogs. Throughout this chapter we will build upon this idea.

The first line in every HTML page is the declaration that the following document can be read as an HTML file. This is done by writing `<!DOCTYPE html>`. After that you must declare the root part of the page by writing `<html>`. At line 11 `</html>` can be seen, this



Welcome to my blog website!

Figure 3 The result of the code Figure 2.

tells the computer that everything between line 2 and 11 is the core of the page. The '/' in HTML is used to close these tags. Therefore, HTML is not dependent on indentation and new lines, so a website can in theory be written in one line but using a style is useful to organize your code. The **<head>** tag contains all the meta-data of a page, which for example is the name of the page or imported files. In this case, the **<head>** tag holds the **<title>** tag on line 4, which in turn holds the title of the webpage, which is the little piece of text in the tab in your browser. Again, both the **<head>** tag and the **<title>** tag are closed with the '/'. HTML has a tree-like structure with parent and child tags. In this case the **<head>** tag is the parent, with the **<title>** tag being the child. The **<body>** tag on line 6 is used to contain the HTML page itself. In this example the body only contains a heading, indicated by the **<h1>** tag. The tag is basically a piece of text with a large font size. HTML has many tags which can be used to structure and create a website like images, hyperlinks, buttons and many more. These are the basics of HTML. (w3schools, n.d.)

1.3 CSS

CSS, or Cascading Style Sheets, is the styling language of *HTML*. CSS uses a cascading style to determine the styling of *HTML* objects, meaning that CSS will determine the styling property of an object by looking at its priority. This is explained in 1.3.1 *The styling language*. CSS's main purpose is to increase the usability of a website because *HTML* only is not enough. CSS does this by being able to edit all kinds of styles like the position or colour of an *HTML* element. It can also hold variables to make it easier to, for example, apply the same colour to multiple styles without having to copy-paste colour codes. CSS can also do animations and even be responsive to user input. (Wikipedia contributors, 2021c)

1.3.1 The styling language

Figure 4 and Figure 5 will be used as an example in this paragraph.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>My First Blog Website</title>
5  </head>
6  <body>
7    <h1 style="font-family: sans-serif; font-weight: bold;">Welcome to my blog website!</h1>
8    <p style="font-family: sans-serif; opacity: 0.8;">On this website anyone can write blogs.</p>
9  </body>
10 </html>

```

Figure 4 An example of CSS being used inside HTML tags.

CSS can be used to style inside *HTML* tags. At line 7 an **<h1>** tag can be seen which contains the text: "Welcome to my blog website". Inside the **<h1>** tag it says **style="font-family: sans-serif;"**. This is CSS telling the computer that the font of the text in this **<h1>** tag should be sans-serif instead of the normal serif font. Font is not the only property CSS can alter, there many and many more. Look at line 8 where a **<p>** tag is declared which means paragraph and is used to display normal text. This time it says **opacity: 0.8;** which makes the paragraph appear a bit greyish. But it also,



Figure 5 The result of the code in Figure 4.

again, includes the **font-family** property. Every property must be closed with a semicolon after which another property may be declared. (w3schools, n.d.-a)

1.3.2 Specificity and selectors

Figure 6 and figure 7 will be used as an example in this paragraph.

```

1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>My First Blog Website</title>
5    </head>
6    <style>
7      body {
8        font-family: sans-serif;
9      }
10     h1 {
11       font-weight: bold;
12       font-size: 32px;
13     }
14     p {
15       opacity: 0.8;
16       font-size: 24px;
17     }
18     .blog {
19       font-size: 16px;
20     }
21     #large-header {
22       font-size: 40px;
23     }
24   </style>
25   <body>
26     <h1 id="large-header" style="font-family: fantasy;">Welcome to my blog website!</h1>
27     <p>On this website anyone can write blogs.</p>
28     <h1>Creating a website:</h1>
29     <p class="blog">This blog is about how to create a website using HTML, CSS and JavaScript</p>
30     <p class="blog">First, you need to buy a computer, you will need it! Then you have to install
31   </body>
32 </html>

```

Figure 6 An example of selectors and specificity.

The tool that gives CSS its name is its specificity, in other words: its choosing priority. The language uses selectors to choose which *HTML* objects must be styled. Examples are element selectors, id selectors and class selectors. The order of priority is the following: First comes inline CSS, inline CSS overwrites *everything*. Even though at line 8 it says that all **<body>** elements (or its



Figure 7 The result of Figure 6.

children as there is only one `<body>` element) should have the font *sans-serif*, in figure 6 you can see that the font is something else. That is because the `<h1>` tag at line 26 has an inline style saying it should use the font *fantasy*.

After inline styling comes ids. Ids are declared using a `#`. Also, an example can be seen at line 21. This id is telling the computer that an object with the id **large-header** should have a font-size of 40px.

After ids come classes. Classes work almost the same as ids except that ids are used on only one object while classes are used on many *HTML* objects. The paragraph elements on lines 29 and 30 both have the class **blog**. The **blog** class makes the text appear smaller because it overwrites the **P** selector, which says the font-size should be 24px.

There are three ways to apply styles on an *HTML* document. The first one is of course inline styling which has already been discussed in 1.3.1 *The styling language*. The second one is using the `<style>` tag which can be seen at line 7 and is closed at line 17. This is useful when testing websites but should be avoided in the final product. The last way is using a CSS file and importing it. This can be done with a `<link>` tag seen at line 5 and has the same effect as the `<style>` tag. The link tag has two properties: **rel** and **href**. The **rel** tells the computer that the file imported is a CSS stylesheet. **href** tells the computer where to find the file, which can also be a link to a website (see 1.7 *CDN and other installation methods*). (w3schools, n.d.-a)

1.4 Front-end Frameworks

A front-end framework, or CSS framework, is code which has been written by other people for your own use. The main reason for using a front-end framework is to save time. For example, many people have already written the code for a button or taskbar, so why not use it? A front-end framework saves a lot of hassle with writing code that works, but also code that works across multiple browsers as not all browsers support the same functions. Most front-end frameworks include a grid system so you can easily align objects and keep them aligned across different screen sizes. Most also include built-in colour themes to keep your website consistent and make it easier to say, switch between light and dark mode. There are many front-end frameworks, and most are free as well, so there is more than enough choice. (Bradford, 2019)

1.5 JavaScript

JavaScript together with *HTML* and *CSS* forms the cornerstones of front-end development. JavaScript is the scripting language of the web; it is used to create a responsive website. It is also the most popular coding language on Earth and easy to learn. The most important thing about JavaScript is that it runs on the user's computer instead of on a server. JavaScript can reduce load on the server and generally creates a faster website because the user does not have to wait until rendering is done in the backend. JavaScript can directly interact with *HTML*, being able to add and remove elements. JavaScript also directly interacts with *CSS*, being able to change properties directly or by applying a *CSS* class or id. The downfall of JavaScript is that not all browsers support all functions, which can sometimes make writing JavaScript very tedious. Using a JavaScript framework can help with this (see 1.6 *JavaScript frameworks and libraries*) and make for a faster workflow. But JavaScript on its own is already very powerful. (w3schools, n.d.-c)

1.5.1 The JavaScript basics

Figure 8, Figure 9, and Figure 10 will be used as an example in the following explanation.

```

25 <script>
26   document.addEventListener("DOMContentLoaded", function() {
27
28   });
29
30   function CreateBlog() {
31     var title = document.getElementById('titleinput').value;
32     var text = document.getElementById('textinput').value;
33     document.getElementById('blogbody').innerHTML = '<h1>${title}</h1><br><p>${text}</p>';
34   };
35 </script>
36 <body>
37   <h1 id="large-header" style="font-family: fantasy;">Welcome to my blog website!</h1>
38   <p>On this website anyone can write blogs. Write your title above and the text below.</p>
39   <input id="titleinput"><br>
40   <textarea id="textinput"></textarea><br>
41   <button onclick="CreateBlog()">Press me to insert your blog!</button>
42   <div id="blogbody"></div>
43 </body>
44 </html>

```

Figure 8 An example of JavaScript being used to generate user-made blogs.

In this example we will expand upon the idea of creating a website to post your blogs. The idea is that you can write your blogs in a text field and then post them to make them appear on the page instead of editing the HTML yourself.

JavaScript can be written in a `<script>` tag but, just like CSS, it can also be imported from an external file. The first 3 lines is a JavaScript function that fires when the page has finished loading. That is done by adding an **EventListener** which listens for certain events to happen. And when the event happens, it runs a function declared by the second argument of the **EventListener** function. The first argument is the event the **EventListener** is listening for, which in this case is `'DOMContentLoaded'`, meaning *the page has loaded successfully, you can now edit the page (or DOM (Document Object Model))*. The function it runs is empty in this case because we do not need it. But you can imagine it being useful for things like executing an animation when the page has loaded, or instantly generating HTML based on what the server sends you.



Figure 9 The result of the code in Figure 7.



Figure 10 After the button has been clicked.

There are a few new HTML objects, and a few have been removed from the `<body>` tag.

The two paragraphs and the blog title have been removed as we do not need it anymore. And four new elements have been added:

- An `<input>` tag with the id `"titleinput"`
- A `<textarea>` tag with the id `"textinput"`
- A `<button>` tag with an `onclick="CreateBlog()"` value
- A `<div>` with the id `"blogbody"`

After some of them is a `
` tag, which tells the HTML that there should be a new line, so the objects don't appear next to each other. Now that we know what element's ids are, we can take a look at the JavaScript. The first thing to notice is that the `<button>` tag has an `onclick` value. Which means that when the button is clicked, execute the function given. The function `CreateBlog()` is defined inside the `<script>` tag at line 30. The `CreateBlog()` function does not accept any arguments, so none are given. What is more important is what the function *does*. It first creates two new variables: `title` and `text`. Both must hold the value the user has inputted in the `<input>` element and the `<textfield>` element. That is done by first selecting the element, and then requesting its value. Selecting elements can be done in many ways. You can use a `querySelector` to select an element. Or, by using `getElementById`. Which literally gets you the element with the desired id. Notice that we use this on the `document` variable, which holds the entire page. By putting `.value` at the end, we tell JavaScript that we do not want the entire element. We only want the value of the element, which in this case is the user's input. Now that we have those values, we can start plugging them into the HTML. We use `getElementById` again to get the `<div>` element in which we are putting our two variables. By using `.innerHTML` we tell JavaScript that we want to replace the entire content of `<div>` with what comes after the `'='` sign. Notice that what comes after the `'='` sign is a string but with backticks. When you use backticks in JavaScript you can plug in variables on certain places with the syntax: `` ${VARIABLE} ``. And that is exactly what we want to do. We want to create two new elements: `<h1>` and `<p>` with the variables `title` and `text` as their desired text. The function is then closed and ready to be used.

One last thing to note is that after every function or declaration you must add a semicolon in JavaScript. JavaScript usually also works without adding the semicolons. But sometimes not using them can cause annoying bugs. (w3schools, n.d.-c)

1.6 JavaScript frameworks and libraries

JavaScript frameworks exist to aid the programmer in creating a website faster and more efficiently. This is done by having many built-in functions and tweaks to improve the overall workflow. Just like front-end frameworks they simply exist to save time by using other people's code (see 1.4 *Front-end Frameworks*). Only with JavaScript frameworks it is not just copy-pasting code. A JavaScript framework completely alters the syntax of JavaScript. JavaScript libraries are like frameworks, as they also have a large code library, but their purpose is different. JavaScript libraries are a tool to increase overall productivity while frameworks are purpose-specific and already include an entire framework to build upon. Meaning that a large part of the code of the website already exists. A library simply has many functions you can call to aid you with programming. (trio.dev, n.d.)

1.6.1 jQuery

jQuery is the most popular JavaScript library, and the oldest. jQuery's main purpose is to "write less, do more".

```

27  $(document).ready(()=> {
28      $('button').click(()=> {
29          title = $('#titleinput').val();
30          text = $('#textinput').val();
31          $('#blogbody').html('<h1>${title}</h1><p>${text}</p>');
32      });
33  });

```

And it is great at doing exactly that, hence its popularity. jQuery

uses many easy-to-use selectors to select *HTML* objects. In figure 11 an example can be seen of jQuery in action. The example code does the same as the code in figure 8. But this time the function starts off with an event listener. The event listener `.ready()` is exactly the same as `addEventListener('DOMContentLoaded')`. It's just considerably easier and faster to write down. jQuery also easily selects elements by using `$('#css_selector')` or `$(variable)`. The dollar sign is used to indicate that this is a jQuery function so regular JavaScript knows where to look. `.click()` is also an event listener and `.val()` returns the users input. `.html()` replaces the HTML inside an element, just like `.innerHTML`. Note that in this example `function() {}` has been replaced with `()=> {}`. This is not jQuery specific and can be used in regular JavaScript as well, these are called arrow functions.

jQuery can easily be imported through a CDN (see 1.7 *CDN and other installation methods*) or installed by downloading an incredibly lightweight 30KB .zip file. (JS Foundation - js.foundation, n.d.)

1.7 CDN and other installation methods

There are numerous ways to install plugins and frameworks on the web. One of the easiest ways to do so is via the CDN. A CDN is a Content Delivery Network, meaning that it provides content for your website. The content is of your own choosing, it can be stylesheets, JavaScript files or even fully fledged frameworks. Using a CDN is only recommended for testing purposes

```

<script src="https://code.jquery.com/jquery-3.6.0.js">
<link rel="stylesheet" href="{% static 'home/styles.c
<link rel="stylesheet" href="https://fonts.googleapis
<link rel="stylesheet" href="https://cdn.jsdelivr.net
<link rel="preconnect" href="https://fonts.gstatic.co
<link href="https://fonts.googleapis.com/css2?family=
<script src="https://cdn.jsdelivr.net/npm/semantic-ui

```

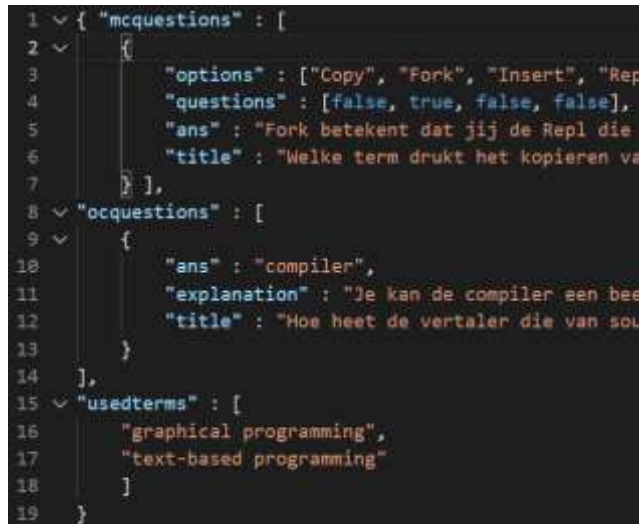
Figure 12 An example of many CDN's being used in an HTML document.

because the content cannot be edited, for example, editing the default theme of front-end framework is not possible. Also, if the hosting of your chosen CDN fails, your entire website might go down with it. This is not a big problem if you are only importing stylesheets, but it can be if suddenly your JavaScript fails. Luckily, there are other ways to install plugins, stylesheets, and other things on your website. One way is by simply downloading the required files, this is not so hard when only installing stylesheets or single JavaScript files. But it can become very tedious with entire frameworks. Another way is via the CLI, or Command Line Interface, which requires you to install an installer after which you can install and download all the required files by typing a few commands in the terminal. NPM is such a CLI and is installed together with NodeJS. NPM makes it very easy to install plugins and packets, for example installing jQuery would only take one line: `npm install jquery`. There are many more NPM-like CLI's, but NPM is definitely the most popular. (What Is a CDN | Cloudflare, n.d.)

1.8 JSON and APIs

JSON, JavaScript Object Notation, is a data format. It is very lightweight and even though it has *JavaScript* in its name, it can be used in more languages like C, C++, Python, JavaScript and more. JSON uses objects and arrays to hold data, they can exist inside each other. An array only holds a few single values or objects separated by a comma encased by square brackets. They can be accessed by looking at their place like this: `options[23]`. This will look inside the array `options` and get the 24th object (an array starts counting from 0). JSON also has objects, which have a key assigned to them, this is done by first giving the key in string format (text) then a colon and after that comes the value. These objects are usually nested, so accessing them is done by using a dot: `mcquestions.options[2]`. This will look for the key `options` inside `mcquestions` and in turn will look for the third object inside `options`. See Figure 15.

JSON is can also be used in an API, an Application Programming Interface, which is used as a way of exchanging data between two programs. JSON is very useful for this, as it can be used in many different programming languages and is very lightweight. In the case of the front-end, the best way to communicate with the backend is to send JSON back and forth. API's can also be used to talk to different websites and web services because of its ease of use. (JSON developers, n.d.) (w3schools, n.d.-d)



```

1 { "mcquestions": [
2   {
3     "options": ["Copy", "Fork", "Insert", "Rep
4     "questions": [false, true, false, false],
5     "ans": "Fork betekent dat jij de Repl die
6     "title": "Welke term drukt het kopiëren va
7   },
8   "ocquestions": [
9     {
10      "ans": "compiler",
11      "explanation": "Je kan de compiler een bee
12      "title": "Hoe heet de vertaler die van sou
13    },
14  ],
15  "usedterms": [
16    "graphical programming",
17    "text-based programming"
18  ]
19 }

```

Figure 13 An example of JSON being used to store data.

Chapter 2 – Designing a website using Google's Material Design

Designing a website or any user interface requires a certain design philosophy. A design philosophy is needed so the user interface is usable and accessible for all people. By using and staying loyal to one philosophy one makes a website which stays the same throughout the entire user interface creating a familiar environment for the user.

2.1 The Graphical User Interface

The GUI, or graphical user interface contains all the visual parts of a website or program. A GUI is not just a user interface, a user interface can also be a command line or a physical button. A graphical user interface, however, is displayed on the screen, usually with buttons, menus, pictures, text and more. In the end both a UI and a GUI want to achieve the same goal: to create a user-friendly interface.

2.2 Design Philosophies

Design philosophies can be seen in all products. They can be seen in buildings, cars, furniture, etc. Artists and designers alike use a design philosophy to create their products. The goal of a design philosophy is that the designer thinks of all aspects of their product like usability, aesthetics, and consistency.

2.3 Google's Material Design

Material is the design philosophy Google uses in all their products to create a consistent experience across all their products. By using material design, one creates a website or program which fits in perfectly with Google's websites and apps, creating an already familiar environment for the user. Material design is simple, obvious, and accessible while also good-looking and flexible, which makes it the perfect design philosophy for a website.

2.3.1 Surfaces, Elevation and Light

One of the biggest principles in material is that it represents the physical properties of objects from the real world which gives us a few simple rules. Material uses surfaces to represent objects and all visual bits. Surfaces are 3-dimensional, giving most surfaces depth. Material works with an axis system: x-axis being left and right, y-axis being up and down, the z-axis moves 'out' of the screen into our 3-dimensional world. Surfaces may not pass through each other, like in the real world. Surfaces are allowed to move, stretch, scale and merge. When in movement they usually move over or under other surfaces. Material surfaces cast shadows on each other, when a surface moves higher up on the z-axis, it casts a larger shadow. Light always comes from the top of the screen casting a larger shadow on the underside of surfaces.



Figure 14 Difference in elevation shown by using shadows. (Material, n.d.)

„Material Design has three-dimensional qualities that are reflected in its use of surfaces, depth, and shadows.” (Google’s Material, n.d.-a)

2.3.2 Layout, grid and spacing

The key principles of material layout are predictability, consistency, and responsiveness. Predictability means that user recognizes objects so it can predict what it means. If all 'retry' buttons would be round with an arrow on them, the user recognizes it and know what it will do. It also creates consistency throughout the layout.

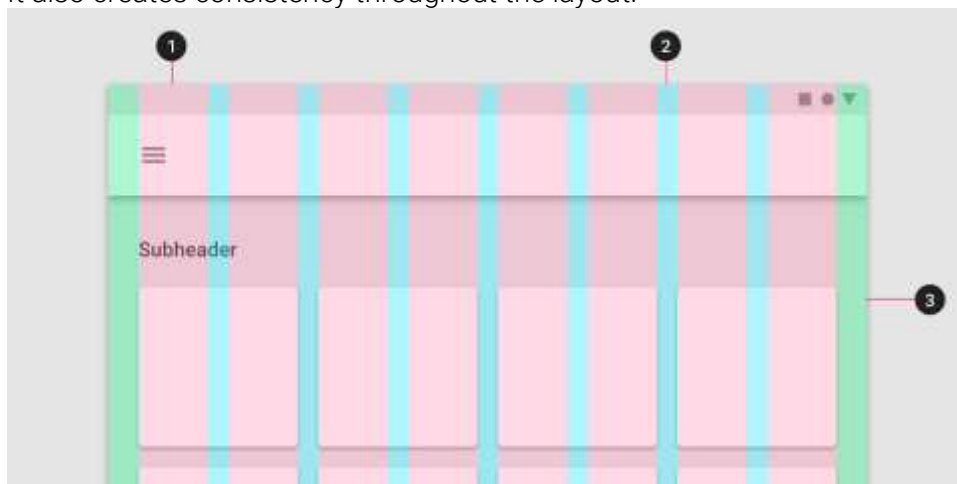


Figure 15 Columns, Gutters and Margins. (Material, n.d.-a)

Consistency is tied to predictability because predictability is created through consistency. A few examples of material consistency are its grid. Material uses a 4dp grid. Dp's, density independent pixels, are a way to measure objects on a screen which is independent from screen density, meaning that buttons will not appear vastly smaller or larger on different screens. The 4dp grid makes object appear more uniform because they will line up instead of being all over the place.

Responsiveness means that material is flexible enough to work on different devices if it is implemented. Material uses columns, gutters, and margins to line up objects and surfaces.

1. Column
2. Gutter
3. Margin

Columns are the space in which content is usually placed, all columns have the same width. Gutters are the space between content, all gutters have the same width. Margins are the 'side gutters', they do not need to have the same size as the gutters and are usually larger, usually no content may be placed on a margin. Content may be placed over gutters, but it must fit between other gutters or margins. See Figure 2.

This grid layout also automatically adapts to multiple screen sizes so it will still look good on say, a phone, even though it was made for desktop use.

„Material Design layouts use uniform elements and spacing to encourage consistency across platforms, environments, and screen sizes.” (Google’s Material, n.d.-c)

2.3.3 Navigation System

Navigation is an important part for every website and app. The user must understand where to find things and how to do certain tasks. A good navigation system takes almost no time to get used to, while also not slowing down the user’s workflow.

Material design’s navigation system uses hierarchy to quicken the user’s workflow while also maintaining simplicity. There are three types of navigation between these hierarchies.

Lateral navigation

The navigation between screens of the same hierarchy is lateral navigation. From these top levels users can go down to lower hierarchies and back up to switch between the top levels. Usually done by using app-bars or similar navigation elements

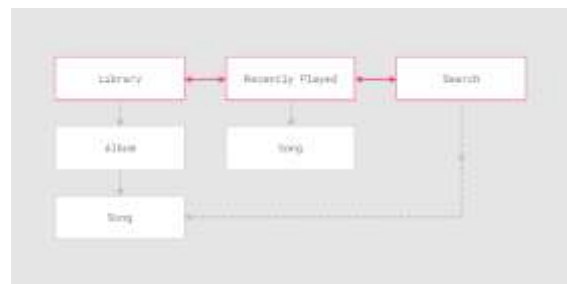


Figure 16 Lateral navigation in a music app. (Material, n.d.-b)

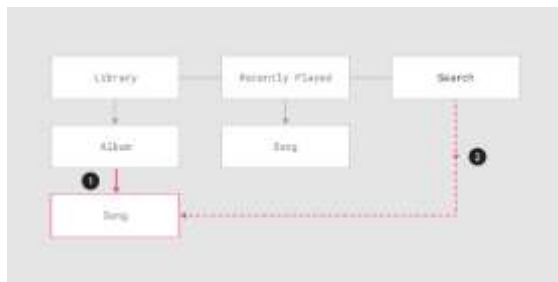


Figure 17 Forward navigation in a music app. (Material, n.d.-b)

Forward navigation

The navigation to lower hierarchies is forward navigation. Usually done using cards, links, or other small UI elements. In Figure 4 moving from *Library* → *Album* → *Song* is normal forward navigation. But you can bypass *Album* by searching for the song in the *Search* screen.

Reverse Navigation

The act of going back up in the hierarchy is reverse navigation. Usually done by closing cards and smaller screens. When going back up the hierarchy, the user goes back the way it came. In Figure 5, if the user searched for the song instead of *Library* → *Album* → *Song*, the user goes back to the *Search* menu.

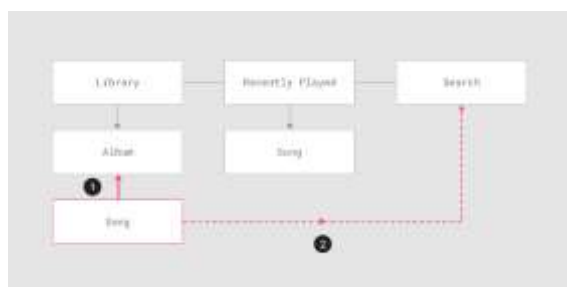


Figure 18 Reverse navigation in a music app. (Material, n.d.-e)

Transitions indicate to the user which type of navigation the user is doing. A forward navigation can be a surface which moves up the z-axis and then expands to fill the screen, so the main screen always stays behind the new surface. Transitions between the same level of hierarchy can be done by pushing the surface out of the screen to display a new one. No overlapping is used to indicate that both surfaces have the same level of hierarchy. Top level transitions can also use a fade-effect to indicate that they are the same level of hierarchy.

„Navigation is the act of moving between screens of an app to complete tasks.” (Google’s Material, n.d.-e)

2.3.4 Colour System

Applying colour to the user interface can be a complex and daunting task because designers can go wrong easily. Luckily, material design has written an extensive tutorial on how to apply colour to the user interface. The main principals of the colour system are:

Hierarchy, making important objects pop out while less important objects fade into the background.

Legibility, text, and icons should meet legibility standards to make it accessible to everyone.

Expression, use your brands colours to strengthen your style.

Material design uses template colours to apply to your user interface. These templates create consistency and make it easier to apply colour. A template exists of primary and secondary colours, with multiple variants of them, mostly lighter and darker variants. It also includes additional user interface colours like background, error, and succession colours.

The primary colour is the colour which is used most in your website or app. You can use variants of the primary colour to distinguish between UI elements.

A secondary colour is optional and should be used scarcely, only to give accent to small UI elements like buttons, highlighted text, or progress bars.



Figure 19 Primary and secondary colour template. (Material, n.d.-a)

Error and succession colours should only be applied in a meaningful and consistent way. Variants of them can indicate hierarchy.

‘On’ colours are template colours that indicate which colour text should have on a template colour. In figure 6 you can see that text should have a white colour for legibility. So, the ‘on’ colour for primary should be white. These ‘on’ colours are useful for consistency.

Legibility is very important when applying colour, it makes for a nicer user experience and better accessibility for visually impaired users. Material design has provided a tool which automatically determines legibility and applies either white or black text.

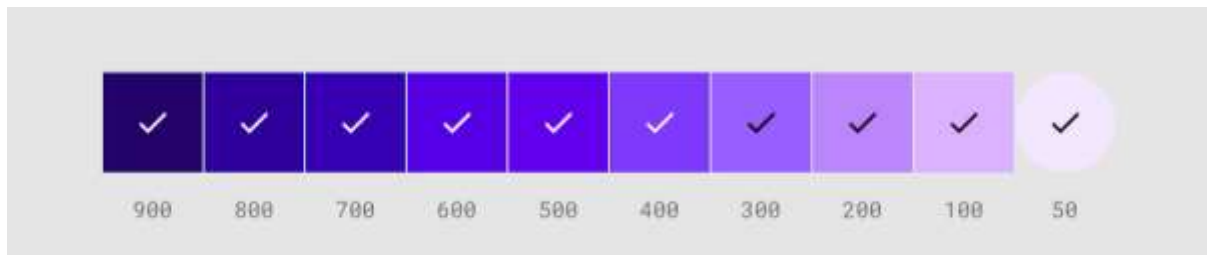


Figure 20 Legibility calculated by Material Design's colour tool. (Material, n.d.-g)

Using other colours than the chosen primary and secondary colours is allowed but should be used with care. One example can be a dark and a light theme, when the user applies the dark theme, colours should change to retain legibility. Another example can be when differentiating between sections of the user interface. Which means that when for example the section where you can see other people's post's the primary colour is red, while on the article section the primary colour is blue indicating a different use for that section. Visualizing data can also be done by using more colours like indicating gains and losses in an investing app.

One of the key rules with applying colour is the 60-30-10 rule (How to Apply a Colour Palette to Your Design – Tutorial, 2020).

Material design never explicitly talks about the rule, but they do imply it by showing lots of examples. The rule basically means that the primary colour



Figure 21 the 60-30-10 rule. (The Futur Academy, 2020)

should be applied to 60% of the screen, the secondary to 30% and all the other colours should only be 10%. In material design's case, the primary colour is most likely to be the background colour, with the primary accent colour being the 30%. The 60-30-10 rule is not only applied in design but also in forms of art like painting and photography.

App bars and other important navigation elements should have the primary colour to indicate importance. Nearby, less important elements can also have the primary colour if they are related. If not, it is better to apply the secondary colour to those elements.

Modal surfaces, surfaces which shortly expand to fill a part of the screen, are usually white but can also be applied the primary or secondary colour.

Cards are also usually white but can also be applied the primary or secondary colour. If a card is selected, changing it to the primary colour is also done often. The baseline colour for normal buttons is the primary colour.

The baseline colour for floating action buttons, important large (circular) buttons, is the secondary colour. This is to distinguish between the normal buttons and make it stand out.

Selection buttons also get the secondary colour.

Colour applied to text is used to either increase legibility or increase its level of importance. Applying the primary or secondary colour to titles or important keywords makes it more important. In Figure 9 the secondary colour is applied to the title and the primary to a link.

Indicating interaction is another important thing in applying colour which can be done by creating strong contrasts. This can also be seen in Figure 9 because the LEARN MORE button has a strong contrast to the background.

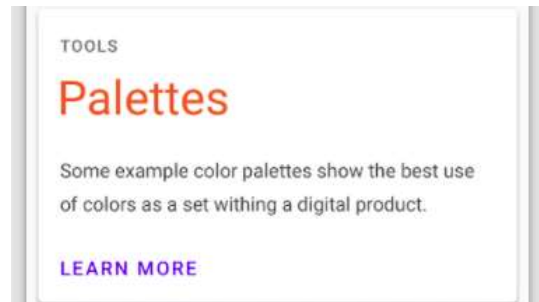


Figure 22 Applying colour to titles and important words. (Material, n.d.-a)

2.3.5 Typography

Just like the colour system, material design also has a template for the type system. It has 13 different scales, weights and letter spacing to apply to different situations. 6 headlines with different scales.

2 subtitle scales.

2 body scales.

A button, caption and overline scale.

Headlines can have very expressive fonts to make them pop out. They will stay legible because of their size.

Subtitles are for short medium-emphasis text, they indicate little importance and can be used for cards. Expressive fonts are better not used for subtitles.

Body text is small and should therefore use very legible text, so no expressive fonts. Body text is used for large pieces of text or entire paragraphs.

Caption and overline should also have legible text, so again, no expressive fonts.

Buttons are usually in all-caps to indicate importance and interaction. Button text should appear distinct from normal reading text.

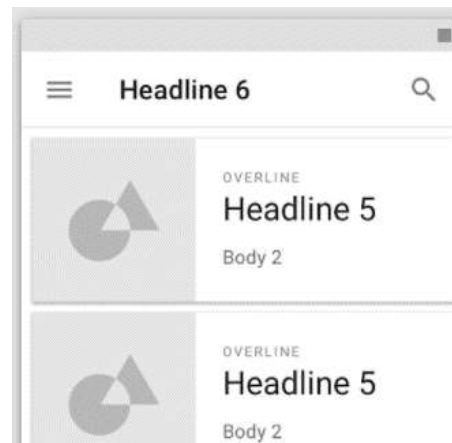


Figure 23 Example of applying material design scales. (Material, n.d.-i)

Just like surfaces, text also aligns to a 4dp grid. The baseline, the invisible line under the letters, is what fits to the 4dp grid. The distance between the baselines of the text should therefore be in increments of 4dp.

Font-weight refers to the thickness of the letters used. So, a bold font has a larger weight than a light weight. Bold fonts can be used to emphasize certain pieces of text like warnings or important words.



Figure 24 Example of a baseline being used (Material, n.d.-j)

There are many fonts created in the world. They have been sorted into types so a designer can easily determine which font to use for their use case.

Serifs have a small shape at the start and the end of a line. Serifs resemble old ink handwriting and are often used in articles and newspapers.

Sans serif literally means *without serifs*. Therefore, sans serifs do not have the small bits at the end of every line. They are often very simplified and used in more modern use cases.

Monospace means *one space*. All characters have the same width. Monospace is often used in computer code and large numbers because it is very easy to count and see how many characters have been written.

Handwriting is often used as a title because it attracts attention, however, it is harder to read.

Display is a miscellaneous excessive font type which is only used as a title.



Figure 25 Example of a serif
(Material, n.d.-k)

„Use typography to present your design and content as clearly and efficiently as possible.”
(Google’s Material, n.d.-b)

2.3.6 Motion and Animations

Hierarchy is an important part in animations. By using the same motion on different objects, you can indicate that those objects are related in a certain way. Good animations can create special moments for the user making them remember your brand. Feedback is also important with animations; say you try to unlock a phone with a passcode. When you fail to give the correct password, the letters will turn red and start shaking indicating that you did not enter the correct password.

Material design uses a motion system to make creating animations easier. The motion system uses four transition patterns.

Container transform

If the container, a surface which contains information, is persistent, meaning that the object stays in existence throughout the animation, a container transform pattern should be used. The container transform should seamlessly transform into a different object. This could be a button which expands to fit the whole screen when pressed displaying new content. When the content inside the container is transitioning, a fade effect can be used to make the animation more seamless.

Shared axis

A shared axis animation is used to indicate a relationship between objects. For example, when pressing a *next* button, the content moves left out of the screen while new content moves from the right side into the screen. The x-axis reinforces horizontal navigation and y-axis vertical navigation. Z-axis, or vertical transitions indicate a change in hierarchy as objects move to the background or the foreground. A z-axis transition can be a picture which expands over all other object to fit the screen when clicked.

Fade through

While a shared axis indicates a strong relationship between objects, a fade through indicates a weak relationship. Fade through basically fades away one object while fading in the new object. Examples could be hitting a refresh button or switching pages on the top hierarchy.

Fade

Fade makes new objects appear on the screen. The objects scale and fade in from a central point. An example could be a warning message which fills a part of the screen. The objects are entirely new and does not replace a different object.

Speed is also important for creating animations. You do not want to keep the user waiting and the animation should also not be too fast so the user cannot follow. Exit and closing animation may be faster because they require less attention from the user. The larger the distance traversed or the larger the animation, the longer should the animation take. Small animations do not require much time while large animations should be slow.

Another important part of animations is easing. Easing makes objects move more fluidly than objects without easing, they accelerate first and then they slow down before coming to a full stop. Objects without easing do not resemble how objects move in the real world. Material design has four types of easing.

Standard easing will accelerate fast and slow down gradually to put emphasis on the end of the animation.

Emphasized easing will put even more emphasis on the end of the animation by accelerating even faster while still slowing down gradually.

Decelerated easing enters the screen at a fast speed, gradually decelerating throughout the animation.

Accelerated easing is the opposite of decelerated easing, accelerating out of the screen instead of moving into the screen.

Lastly, using the same speed, easing or type of animation throughout a user interface can emphasize a brand's style. Users will also become more familiar with the animations so they will recognize and remember them.

„Motion helps make a UI expressive and easy to use.“ (Google's Material, n.d.-d)

Chapter 3 – Building a website using Python and Django

"The Internet is the global system of interconnected computer networks" (Wikipedia contributors, 2021): the internet is nothing more than some computers working together to transfer data and combine values into one coherent output. Since modern programming's start in the 60s, coding has grown from a labour-intensive task into an art that has inspired millions. And with the rise of the Internet in the 90s and early 2000s, programming itself rose also. Forums as Stack Overflow and Quora brought the community together and provided help to those who asked for it, while Massive Online Open Courses from various universities (MIT and Harvard upfront) gained popularity as well. These MOOCs have resulted in an uproar of self-taught computer scientists: according to Stack Overflow's 2016 survey 69% of developers are at least partially self-taught.² The Internet has grown and so has programming itself (*Stack Overflow Developer Survey 2016 Results*, n.d.).

In this chapter we will not explain how to program; we will, however, explain and define the terms for you to understand the rest of the research.

3.1 Languages

Computers only understand zeroes and ones, meaning that you will need to instruct the computer using only binary numbers. Luckily, some very clever people that came before us have made what is called a linker (*C Language Source Code to Exe (Theory)*, 2017). This is a piece of code that translates assembly code (written in characters, so not binary) into an executable file that the computer can thereafter execute. Assembly is a language you can (relatively) easily use to create some code. However, it is still not very easy to use, as you need many steps to get a simple thing done. Therefore, we have invented languages: these are abstractions of assembly that come with a compiler or interpreter that translates the language into machine code (Lithmee, 2018). There are different kinds of languages: some are lower-level and others are higher-level. The former means that the programmer has a lot of power, and therefore needs a lot of steps. The latter means that the programmer is quickly done programming, but often comes at the cost of speed (i.e., they take longer to run) (*Higher-Level and Lower-Level Languages - Computer Science Wiki*, n.d.).

3.2 Python

Python is a higher-level programming language that uses specific indentation, meaning that indentation changes the meaning of the code. It is nowadays a wide used language for all kinds of purposes: data science, creation of artificial intelligence, making games and implementing websites (Kuhlman, n.d.). The language is very popular because it is powerful, yet easy to use.

3.2.1 Using Python for making a website

If you want to use Python to create a website, you will need to get some libraries, these are extensions built onto the basic language. Examples of libraries are `sys` (short for system) and `os` (short for operating system). Each of these provide the programmer with easy-to-use functions that they otherwise needed to implement themselves. To make a website, or rather a web application, you will need to do a lot of regulation. Luckily, there are very smart people in this world that have created accessible and understandable libraries to create a website (*Foreword – Flask Documentation (2.0.x)*, n.d.).

3.2.2 Flask

Flask is a popular so-called micro framework. Frameworks are a level above libraries, often you can use them as if they were a library, but they are very extensive and have a lot of functionality. 'Micro' means that Flask is a framework that does not come that a lot out of the box: you will need to do a lot of configurations yourself, which makes Flask very adaptable and widely applicable (*Quickstart – Flask Documentation (2.0.x)*, n.d.). Flask uses a single python file (in the beginning) which holds different functions that run if the user visits different pages. Using the Flask functions the programmer can make queries to the database. Flask does not choose a database for you, and therefore you can choose which one to use. Every page on the website uses a template: this is an HTML-page that with template language added, enables the programmer to insert variables based on values in the database onto the page and show it to the user. Flask is easy to use, but does not come with that many features, meaning that you will need to create them yourself.

3.2.3 Django

Another more sophisticated framework is Django. This framework is very extensive and comes with a lot of bells and whistles. Your Django website is divided into webapps: these separate some code, to maintain a clean working tree. The core of each webapp is the file called `views.py`, which works like the single python file in Flask: it contains many functions, one for each page or functionality your website holds. Added to Django in comparison to Flask is the file `models.py`. This is a python file that defines models, which are basically SQLite tables. Let us say you have a user and want to store their username and password. (Note: never store a password as a string, but always encrypt it). You will then create a model called `User`, that has two fields: one called `username` that takes a `CharField` (a piece of text) as input and a second `CharField` that is called `password`. You can then create a form in `views.py` that enables users to register and log in. Django also comes with an admin page; according to Django "it's not just scaffolding - it's the whole house" (*Django at a Glance | Django Documentation | Django*, n.d.). This admin page allows the developer to insert some queries into the database without using SQLite statements, it generates the forms for you immediately. This is just the tip of the iceberg of all the handy functions Django has to offer.

3.3 Django – the powerhouse of webdev

"The web framework for perfectionists with deadlines" (*The Web Framework for Perfectionists with Deadlines | Django*, n.d.). It is the early 2000s and internet is on the rise. Two programmers, Simon Willison and Adrian Holovaty were fed up with the complexity of using PHP to maintain their website for their employer. Having both fallen in love with Python, a new language at the time, thanks to Mark Pilgrim's book, they took matters into their own hands. As any programmer should do if current libraries are not fulfilling their tasks; they created their own. After the release of the web development framework named after Adrian's favourite guitarist (*What Is the History of Django - Quora*, n.d.), Django has grown into a first-class, easy-to-use, powerhouse of web development. It is now used by companies as Instagram, National Geographic, Pinterest, and Mozilla. 43% of Python web developers use Django, resulting in first place (Petlova, 2021). The world of webdev has agreed: Django is a powerhouse.

3.3.1 Webapps

Every website made with Django is divided into webapps, these are typically smaller portions of a website that house a certain functionality of your website. If you create a website for your theme park, you will want an app for buying tickets, showing visitors the park and for the contact form. Django does this to keep your working tree clean and cease the computer science principle of files containing thousands of lines of code, making it impossible to find something quickly. Every webapp has a subdirectory according to the webapp's name. It contains a couple of files after you have created it using some simple commands.

Django automatically creates some files for you, as `manage.py` and `settings.py`. From now on you will use the command `python manage.py <command>` to let your app do something. Let us say you want to server to run, simply type `python manage.py runserver`. If you execute that command, even though we have not yet specified what the website should look like, you will see a page where Django tells you the website works.

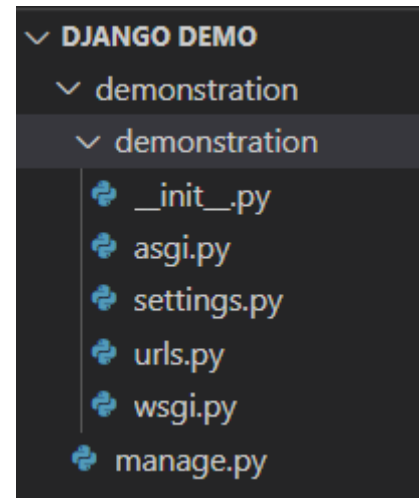


Figure 26 Django's standard file tree.

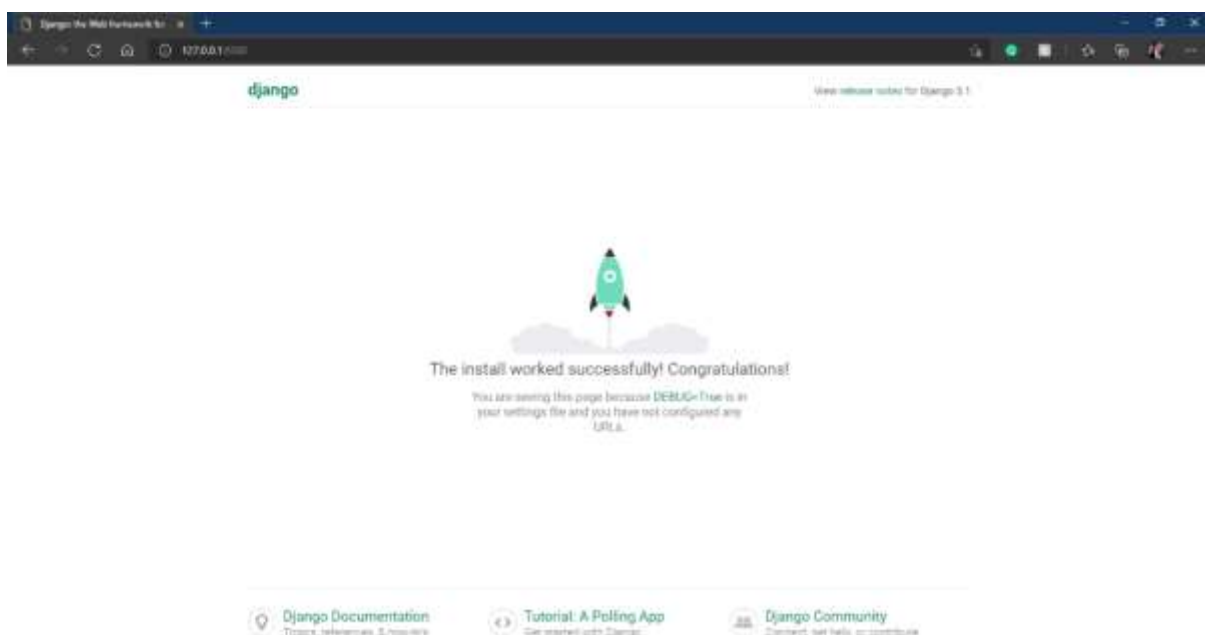


Figure 27 Django shows this page to tell the developer everything works fine, and they can start programming.

Let us create in this chapter, as an example, a portfolio website. On this website we want to create some pages, like an overview of all people on the website and for every person a page that holds their work experience.

So, if we want an app called "portfolio", evidently, use `python manage.py startapp portfolio` to do so. Then, again, Django automatically creates some files for you, such as

views.py (3.3.3 Views), admin.py (3.3.5.3 Admin) and models.py (3.3.5.1 Models). For the webapp to work, add it to the INSTALLED_APPS list in settings.py and everything is ready to roll!

3.3.2 URLs

Now that we have created our first web-app (that works!), we can start with links of all the URLs. "A Uniform Resource Locator (URL), colloquially termed a web address, is a

reference to a web resource that specifies its location on a computer network and a mechanism for retrieving it." An URL is the address of a website, as offices have addresses with street names and numbers, so do online webpages have URLs. In the file urls.py in the demonstration sub-directory, we need to specify at what URL our webapp portfolio "lives". For our demonstration, we would probably want the portfolio app to live at /portfolio. So, let's specify that!

The admin part was already written by Django (3.3.5.3 Admin). The list urlpatterns contains a list of all the URLs of this website (or references thereto). So, we added a new URL to the list called 'portfolio/' and we want it to include another file: in which we specify extra URLs. So, in this file we created the "/portfolio" part, but in portfolio.urls we can then create a URL called "portfolio/peterypma".

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('portfolio/', include("portfolio.urls"))
]
```

Figure 28 The general urls.py that stores the URLs to different webapps.

Let us look at portfolio.urls: we, again, make a list called urlpatterns and add an URL to it. We want the index (homepage) of this webapp to "live" at portfolio/ without anything coming after it. (Recall that because we referenced to it in the general urls.py file, all these URL start with /portfolio following by the path we specify here.) But what should we put on it? We specify the view (how the page looks) of a website in views.py, so after importing it we can say that the index view will specify what to show. We also give this URL a name of index, meaning that we can later refer to it more easily.

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.index, name="index"),
]
```

Figure 29 Adding the index function from views.py to the default path of the hello webapp.

3.3.3 Views

Next, let us look at views.py. This file is the one in which you spend the most time (at least we do), but for now will contain some very easy code. We created a function called index that takes the request variable as input, all functions should do this, so that we can

```
def index(request):
    return render(request, "portfolio/index.html")
```

Figure 30 Defining a basic view to render a template.

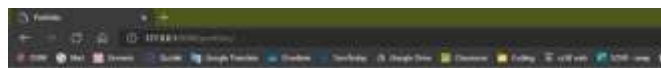
store data from the user, like whether he is logged in or not. Then, everything we do is say: render this template, rendering just means to show to the user; put on their screen.

3.3.4 Templates

Just to recap, if the user visits ourdomain/portfolio the general urls.py file sends the user to the portfolio.urls that says that for the path of nothing (so still /portfolio) we should look for a

```
portfolio > templates > portfolio > <> index.html > ...
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title>Portfolio</title>
5      </head>
6      <body>
7          This is the portfolio homepage
8      </body>
9  </html>
```

Figure 31 The basic HTML page, greeting the user and explaining its goal.



This is the portfolio homepage

Figure 32 The result of the HTML template.

3.3.5 Models and data

We can now show pages to the user, but do we want to manually declare all portfolios of the users on the site in urls.py and create hundreds of templates for all of them? No, right? That is tedious and Django promised us that we could have deadlines (3.3 Django – the powerhouse of webdev) and so there should be an easier option, and there is! We can store some text in the database and show it at different times.

3.3.5.1 Models

First, we must design our *model*: a model is also like a form, but not for text to show, but for data to store. In the models.py file we can define a model. All models in Django are classes (groups of functions and variables), but do not worry about that for now. In the class we can then define variables based on the field in a form. We probably want to store the name as a string (piece of text). Django has different Fields, the CharField

```
demonstration > portfolio > models.py > Portfolio
1  from django.db import models
2
3  # Create your models here.
4  class Portfolio(models.Model):
5      name = models.CharField(max_length=128, unique=True)
6      years_of_experience = models.IntegerField()
7      experience = models.TextField()
```

Figure 33 Creating a model for storing one's portfolio.

is stores a string, like a name. It takes optional arguments (input) such as the `max_length` and whether it should be `unique` (a person can only have one portfolio). The field `years_of_experience` takes an IntegerField and `experience` takes a TextField, this is the same as a CharField but it can store more text. Fantastic, this is now all stored in the database, but what exactly is such a thing?

3.3.5.2 Database

A database is the place where all data gets stored. It is stored in long-term storage on the server. With queries the user can insert data into the database or ask for a set of data. Python and Django use SQLite by standard. SQL the most popular database structure and is used all over the world. SQLite is a simpler, elegant database framework that is open-source (so free as well as safe) and uses a single file. Web development frameworks such as Flask use direct SQLite queries, like this statement.

```
INSERT INTO portfolio (name, years_of_experience, experience) VALUES
("Vincent", 1, "Working in a supermarket")
```

Flask then literally copies the input into the database, that is great low-level accessibility, but not necessary most of the times. Django built its own abstraction with the use of models.

3.3.5.3 Admin

Now that we have created our first model, we want to insert data into the database. The Django development team has created the admin page for that. You might recall that in the general `urls.py` file, there already was one URL specified, `/admin`. So, let us visit `/admin` and see how to use it. Apparently, we need a log in (otherwise all users could access the database). Luckily, we have access to the terminal (or console) on which the website runs, and we can create a superuser there. A superuser is nothing more than an admin that can visit the `/admin` page. By using `python manage.py createsuperuser` and following its instructions it is easily done.

```
demonstration > portfolio > admin.py
1  from django.contrib import admin
2  from .models import Portfolio
3
4  # Register your models here.
5  admin.site.register(Portfolio)
```

Figure 34 Registering a model ensures its visibility on the admin page.

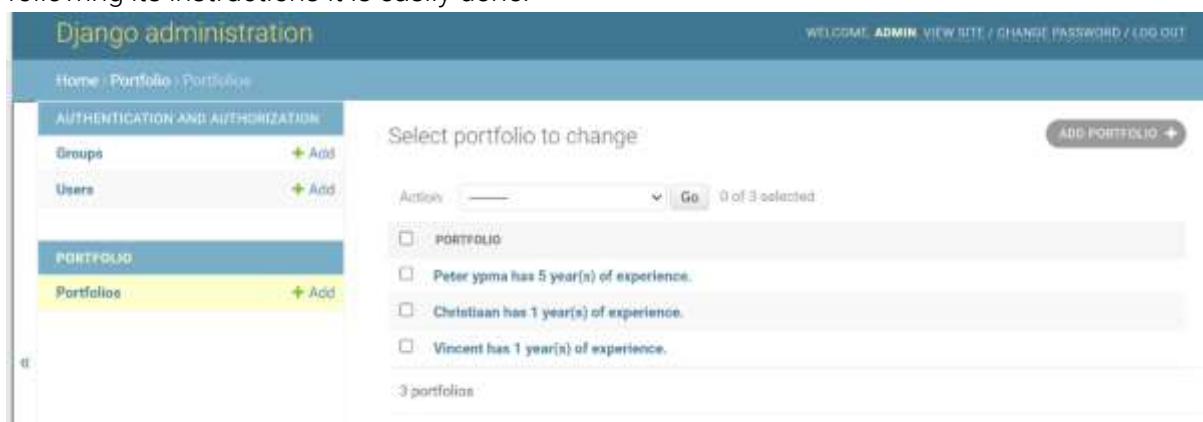


Figure 35 The basic admin view, after logging in with a superuser account and adding some data.

Now that we can visit the admin page: it is empty! It turns out we need to import the portfolio model from `models.py` and then register it into admin. This is because you do not necessarily want all your models to be editable in the admin page and by using this system, you can select which ones you do want. We have registered the model and so we can use it on the admin page.

Figure 36 The form to add or change data in the database using the admin page.

We can click on Portfolios on the left, and it says that there are 0 portfolio (that makes sense, as we did not add one yet). On our screen is now the form based on the model we created in the `models.py` file! Let us add some data so we can later use it.

3.3.6 The great Django mixtape

Now that we have created some loose ends, it is high time to tie them all together. This is the great Django mixtape. So, just a list of what we *currently* have a model that stores users' portfolios an index page that welcomes the users. Now, we need to link this all together by using the `urls.py` and the `views.py` file. First, let us think of a way to render the pages of a portfolio. Look at this URL from the Washington Times

<https://www.washingtontimes.com/news/2021/jun/15/arizona-audit-2020-election-has-become-mecca-gop-c/>

Notice that the 'webapp' news takes some arguments: the year (2021), the month (jun), the day (15), and the title of the article (arizona-audit-2020-election-has-become-mecca-gop-c). Now these are all called *slugs*: pieces of text that provide information for the server. So let us implement this onto our website by saying that `/portfolio/vincent` should render Vincent's portfolio and `/portfolio/christiaan` renders Christiaan's portfolio. For our purposes, let us wave our hands at that and accept that it works, and that portfolio takes an extra argument (input) called `name` that stores the name.

```
def portfolio(request, name):
    p = Portfolio.objects.get(name=name)
    years = p.years_of_experience
    experience = p.experience
    name = name.capitalize()

    return render(request, "portfolio/portfolio.html", {
        'name': name,
        'years': years,
        'experience': experience,
    })
```

Figure 37 The implementation of the portfolio function in `views.py`.

Now let us query for the name in the database so we can get the user's portfolio. We can use the method `objects.get()` to get an object; this function takes filters as arguments, since we want to search by name, we should have something like `objects.get(name=...)`. Now we fill in our variable that, logically, is also called name, resulting in `objects.get(name=name)`. We save this in a temporary variable called `p`. Then it is time to get the values we want from that object. We can save these in variables. The variable `p` stores an object (or dictionary in Python terms) using keys and values. The name of a field we gave it in `models.py` is the key and the value is the data in the database. So, for Peter Ypma the key could be "experience" and to that belongs the value "Teaching mathematics.". In order to access this value, we can use `p.experience`, where `p` is the object storing Peter Ypma's data.

All that is left to do in `views.py` is return these variables we found in the database and give them to our template, so that we can use HTML to create a page that shows all the values nicely. The function `render` can take one extra argument that is a dictionary that holds the values: the keys (first ones) are the names of variables in the template, the values (the last ones) are the names of variables (or literal values) from `views.py`.

3.3.6.1 Variables in templates

Great! We found our values in the database and sent them to our HTML, but how do we use them? Well, luckily for that you can use Django template language, which allows us to insert variables and use simple logic

like if-statements and for-loops. Most syntax in this language starts and ends with two curly braces. So, to insert the variable called `name` into the template, all you need to do is type `{{name}}`. We can now use this knowledge and our HTML-skills to create a simple template. Now navigating to `/portfolio/PeterYpma` results in this page.



Figure 38 The result of using variables in an HTML template.

3.3.6.2 Django template language

Our homepage of the webapp portfolio is a little boring now, is it not? We should put an overview of all people that signed up to our website on it, with links to their pages. First, we should edit the function `index` in `views.py` so it queries all objects (data) from the database, for us to render it onto the user's screen.

```
def index(request):
    p = Portfolio.objects.all()

    return render(request, "portfolio/index.html", {
        "p": p
    })
```

Figure 39 Querying for all the objects from the Portfolio model.

We can use `objects.all()` to get all objects from a model. `p` is now a variable that stores a query set: a set of objects that are the result of a query. In this case, the query set contains all objects from the database. Again: an object is one combination of data: like "Peter Ypma, 5 years of experience, Teaching Mathematics". The beauty of Django now comes in our template file.

```
<body>
  This is the portfolio homepage.
  <br></br>
  All people in our database:
  <ul>
    {% for object in p %}
    <li>
      <a href="/portfolio/{{ object.name }}">{{ object.name }}</a>
    </li>
    {% endfor %}
  </ul>
</body>
```

Figure 40 Using Django template language to loop in an HTML file.

Let me walk you through what happens here. We still have the line of text welcoming users and now there is an extra one that says: "All people in our database". The `
` tags add empty lines. After that comes the beauty: you see, HTML does not natively support for-loops and if-statements, but in combination with Django it does! So, we can loop through each object in our query set `p` and then add a list item `` to our unordered list (list with bullet points, not numbers) ``. Every list item contains a link using the `<a>` tag and that has a value (what the user sees) of the person's name. If the user clicks on the name, it links to an `href` (Hypertext Reference) to `/portfolio/<name>`. If you render this page it looks like this.

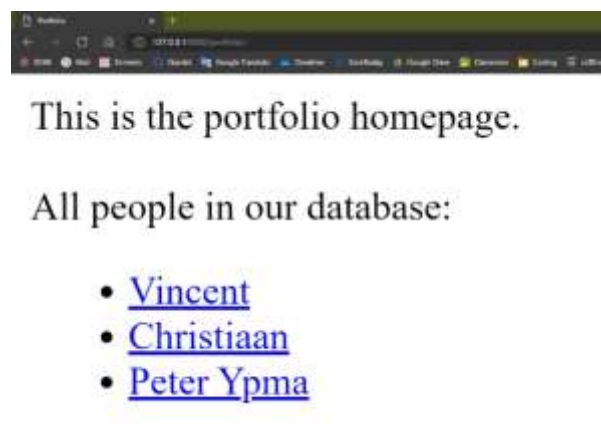


Figure 41 The result of code in 3.3.6.2 Django template language

3.3.7 Forms and input

It is great that we, as a superuser, can use the admin page to add people into the database. But it would be better if there was a form on the website on which people can add data into the website but without access to the admin page. For that we need a form.

```
{% form method="post" %}
  {% csrf_token %}
  {% for field in form %}
  <p>
    {{ field.label_tag }} <br> <span class="field">{{ field }}</span>
    {% for error in field.errors %}
    <br><span style="color: red;">{{ error }}</span>
    {% endfor %}
  </p>
  {% endfor %}
  <button type="submit" id="submit">Add</button>
</form>
```

Figure 42 Using Django's ModelForm and a for-loop to create a form.

We can create a form as a model, by just saying that we want a form for a specific model: in our case for the Portfolio model. Then we let Django generate the form for us (which saves us work). All we need to do is create a form in HTML with a for-loop that combines our page with Django's fields. All Django forms start with the line `csrf_token` that makes sure that user can only enter input they are meant to put in. Then we loop through the form variable and for every field in that form

we generate the basic structure of: label + field (+ error if necessary). This for-loop is quite standard and can be (almost literally) copied into other websites and work immediately. On the left you can see the result of the form. Recall that the Portfolio model had three fields: name, years_of_experience and experience. Notice how these three fields also come back in our form. In our HTML we created a button (in the last line) that can be seen at the bottom of the page as a submit button.

Annotation to 3.3

The demonstrative Django project was created fully and totally developed by Vincent Ruijgrok. The knowledge to be able to create this was acquired in CS50's Web Programming with Python and JavaScript by HarvardX, most of the matter for this demonstration was taught in Lecture 3: [Lecture 3 - CS50's Web Programming with Python and JavaScript \(harvard.edu\)](#).



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:8000/portfolio/add/'. The page title is 'Add'. The form contains three input fields: 'Name:' with the value 'Test Person', 'Years of experience:' with the value '3', and 'Experience:' with the value 'Being used as a test person'. At the bottom of the form is an 'Add' button.

Figure 43 The result of 3.3.7 Forms and input, using ModelForm.

Chapter 4 – Correctness is not everything

February 2021, Vincent Ruijgrok and Christiaan van der Haven are eager to start their thesis in March. Already, they have thought of an idea for their research: creating a website on which students can learn computer science. A good idea, but it struck them quickly that they did not have any of the skills to create a website this complicated: as they wanted to program the entire website themselves. Vincent knew a good course over on EdX: CS50's Web Programming with Python and JavaScript. Almost 25 hours of amazing lectures by Brian Yu later, their minds were filled with ideas for the website and the research. Time to start!

4.1 The structure

Creating and implementing the back end of a website, more commonly called a server, is less of a creative job, rather mathematical. Before we started making any lessons, we created a basic website, with functionalities that we needed most; think of logging in and out, registration, and showing the files of the lessons. As we watched more than 20 hours of lectures on Brian creating websites using Django, it only made sense for us to use that too. We felt that it fit the purposes we had in mind (or rather: the purposes fit Django). Vincent had used Flask before to solve some problem sets from another CS50 course called CS50's Introduction to Computer Science. But he felt that Flask was too basic to fit their needs that realised that, if used effectively, Django was a lot more powerful.

The first set-up of an application using Django is not very difficult, using just a few dozens of lines of code, one can make a registration form, next to a logging in and log out functionality. Django comes out of the box with a User model, containing a piece of text (more commonly called a string) for a username, a second one for one's password (which is stored encrypted) and some others for a name, last login date and time, and more.

The first model we created ourselves is called Filepage, these are models with a path (i.e. a route to find the page), a title and most importantly, a body. This body contains raw HTML to design a webpage. Via the files webapp one can view all files in the database and read them. We chose to not only store text in the body of the Filepage, but HTML. This is because by doing so we can put the styling (like different fonts, colours and text-aligns) directly in the database. If we were only using text, it would look rather boring. Storing raw HTML added complexity but added many options for styling. The main problem we ran into is that Django is very safe, which is of course not a problem, rather a barrier, because Django did not trust HTML from the database. Fortunately, we were able to turn off this "autoescape" feature (for just the files, otherwise it would not be safe) and it worked all fine!

As soon as the most basic version of our website was stable (meaning bugs were fixed), it was time to add a little more subtlety before we started to make lessons. We implemented courses, for which one can sign up. On the user's own page, they can sign up for a course. By doing so, the paragraphs or lessons that belong to this course are loaded onto their user page and they can be easily accessed. At the bottom of all files, we created a button that announced completion. This redirects the user back to their home page and adds a sign of completion to this paragraph in the overview. A user can always sign out of a course and sign in into a new one. The progress is then moved from the Progress model to the OldProgress one, of both the

premise is the same: it contains for every paragraph of a Course (connection of a user to a chapter) whether it is finished or not. In the Progress mode, only relevant Progress is stored for a quicker and easier query. OldProgress can be seen as the log-file: everything everyone has once achieved (or has not!) is stored here. You could imagine that if the number of users increased, this would make searching through and adding into this model go slower and slower, therefore there is the Progress model, again, storing only relevant information. As soon as a user finishes (or leaves) a course the progress is moved to OldProgress. If a user signs up for a new course, the server first checks if the user has ever done something in this course, so that you could work at two courses at once or stop earlier and later continue without losing progress.

4.2 Auto grading

As the basic website-design from the backend came close to a finished state and there was hard work being put in the front-end of the website, it was time to think of how we would create and design lessons. One of the biggest issues we faced was how to submit and check code students have written. In the industry there is a function called auto grading, meaning that the teacher has provided an assignment with some test input and the desired output, a student can execute this code and it returns how many tests have failed or succeeded. One can imagine that this is quite the undertaking and can thus not be made very easily by oneself, therefore we chose to use (free) options out on the internet. As we say: reinventing the wheel is not a good idea, reinventing the applications thereof is. After creating many accounts, reading loads of articles, and watching (literally) hours of videos we concluded there is no ideal option. Just to sketch what we want: an IDE that can be embedded (but also used in a new tab), that runs C code and has a button that can run checks we have provided and returns to the user their score. It is not particularly difficult, yet there are not many people that made this available to us for free. Yes, there are many options, but only for a monthly subscription. So, we chose to settle with a tool from our colleagues at Harvard University's CS50, an introductory course to computer science. They have made `submit.cs50.io`, alongside with their tools `style50`, `check50` and `submit50`. This is not ideal, but it works, that is most important for now. Students can code using a free online IDE called ReplIt, this online tool can be embedded onto our website and used in a page on its own. We will explain the usage with an example. We created a problem 'hello' that should prompt the user for a name and greet them (like "hello, John!").

- For each problem, a student will see this on the lesson's page:
 - By clicking open in repl.it they can "fork" the problem's source code.

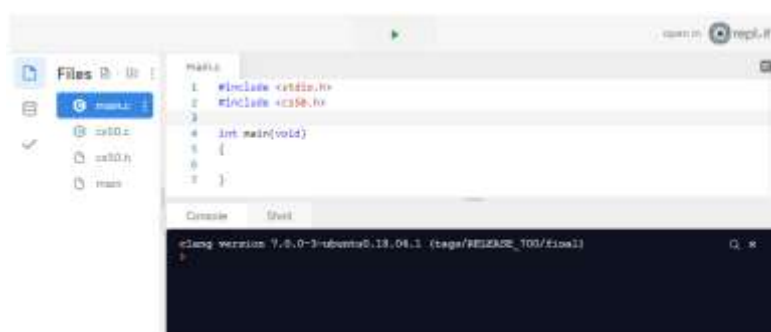


Figure 44 The embed-window on our website.

- Forking the REPL will copy the source code into a new REPL of a student's own, so they can solve the problem.



Figure 45 The view of a forked Repl.it.

- A student can always click the run button and check the code themselves (if the program does not raise any errors). To check the correctness of the code a student should run two commands in the terminal.
- First "**pip install check50**" and later "**check50**" along with a slug: an address where the problem is stored, this slug will be provided to the student, so they only need to copy-paste it.

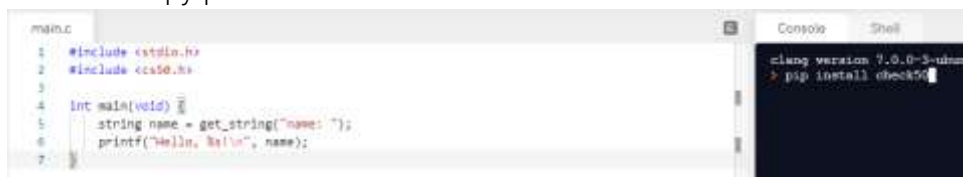


Figure 46 Using pip install check50 in Repl.it.



Figure 47 Using check50 in Repl.it.

- The student may need to log in into GitHub and then it will check their code like this.

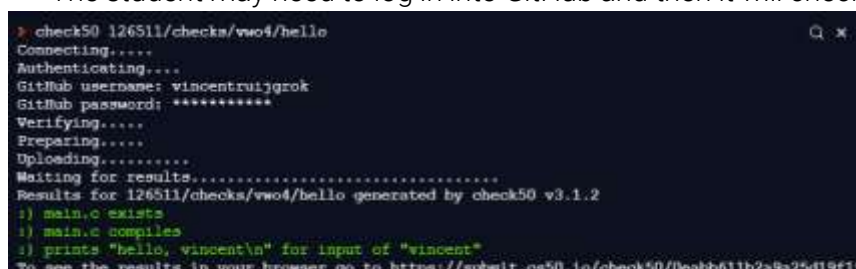


Figure 48 Output of check50 if all is done correctly.

- An error will look like this:

```

> check50 126511/checks/vwo4/hello
Connecting.....
Authenticating....
Verifying.....
Preparing.....
Uploading.....
Waiting for results.....
Results for 126511/checks/vwo4/hello generated by check50 v3.1.2
+) main.c exists
+) main.c compiles
+) prints "hello, vincent\n" for input of "vincent"
   expected "(Hh)ello, vinc...", not "Hey, vincent!\n..."
To see the results in your browser go to https://submit.cs50.io/check50/5e464b73ae455347e0caf4b2839cd16dc16729cb
    
```

Figure 49 Output of check50 if there was unexpected output.

- By opening the provided link, a student can see a more detailed result from check50.

check50
126511/checks/vwo4/hello

```

+) main.c exists
tag
checking that main.c exists...
+

+) main.c compiles
tag
running clang main.c -o main -std=c11 -ggdb -lm -lc50...
+

+) prints "hello, vincent\n" for input of "vincent"
Cases
expected "(Hh)ello, vinc...", not "Hey, vincent!\n..."
tag
running ./main...
loading input vincent...
checking for output "(Hh)ello, vincent!\n"...
+

Expected Output:
(Hh)ello, vincent!

Actual Output:
Hey, vincent!
    
```

Figure 50 Output of check50 in a new tab.

- To submit the code to the teacher, they will need to run two commands again:

```

> pip install submit50
    
```

Figure 51 pip install submit50 in Replit console.

- And submit50 following the same slug as with check50.

```
> submit50 126511/checks/vwo4/hello
Connecting.....
Authenticating....
Verifying.....
Preparing.....
Files that will be submitted:
./main.c
Files that won't be submitted:
./cs50.h
./hahaha.txt
./main
./cs50.c
Keeping in mind the course's policy on academic honesty, are you sure you want to submit these files (yes/no)? yes
Uploading.....
Go to https://submit.cs50.io/users/vincentruijgrok/126511/checks/vwo4/hello to see your results.
```

Figure 52 Using submit50 in Replit console.

Submit50 will tell the student which files will be submitted, and which will not. It will also ask them to confirm their submission. The teacher can see all submissions to all by logging into submit.cs50.io and then under 'My Courses' click the corresponding course (a teacher can have multiple courses). A list will be provided to them, like the following.

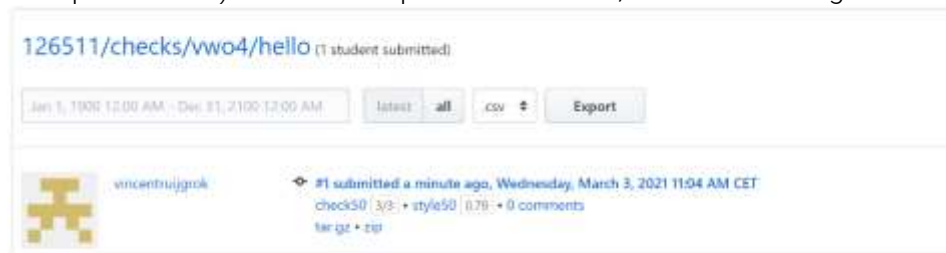


Figure 53 A sneak-peak into submit.cs50.io

You can click many links for a submission, the **tar.gz** and **zip** will let the teacher download the code in the respective file formats. **check50** and **style50** will show the teacher the check50 scores and the feedback of style50, a tool that checks for style (the readability of code for others).

main.c

```
#include <stdio.h>
#include <cs50.h>

int main(void)
{
    string name = get_string("name: ");
    printf("Hello, %s!\n", name);
}
```

And consider adding more comments!

Figure 54 Output of style50.

For style50 the red means delete, and the green means add, so in this case the curly brace should not be on the same line as `int main(void)` but should be on a new line.

For a student to get in a class, a teacher can create their own course (with all the problems we have made) and share a link via which students can join the class. Teachers can also make their own assignments.

4.3 Documentation

One extra thing we wanted for our website was something called documentation. In the programming world documentation is the manual for a language or framework. It contains all the functions and explains how to use them and what parameters they take in. We wanted to do so as well, as it would help students (we reckoned) if they could click on a term to get an explanation in a sentence or two what that term meant, or function did. For that we created a new webapp called docs. It uses a webdev principle called “single page applications”: these are applications that never redirect you to a new page. It uses JavaScript to renew and change the contents of the page based on user input. We did this so that the documentation page can be imported into the lesson directly (for those familiar, using fetch) and to browse the docs you would not need to refresh the page. It works like this: aside from the page running the docs app, we also created a multi-page app using links and database queries to populate the pages. If you would then visit the single page app, it would simply “go” to that page, copy-paste everything in it onto the current page. By doing so, a page refresh was not necessary and if a button was clicked: JavaScript went to that page and copied everything. By doing so, the docs webapp worked imported into the file’s app but also on its own.

4.4 GroteProgrammeer.nl

Of course, it is fun to have a website you can run on your own machine. However, that means that you can only run it on your localhost, the web server that runs on your computer. So, others cannot access it. If you want to deploy (i.e., share with the rest of the world) your website, you need to do a lot more than type `python manage.py runserver`.

4.4.1 Hosting

Your website will, at first, not have that many users, and so your computer could handle the load just fine. But running the website (accessible to the internet) on your own machine, is, no matter the number of users, a bad idea. If you turn off your computer, the website will shut down. If you lose connection to the internet, the website will be down. That is why you want a hosting service to take care of that. You give the code and instructions to run the website to the service and they will run it on their machines, mostly located in a datacentre. These computers in the datacentre do not turn off (or: are not supposed to) and are always connected to the internet. If they are to shut off, for maintenance, for example, they will migrate, or move, your website to a different computer that will not shut off. This ensures that your website is always accessible to anyone with an internet connection. We chose Heroku to host for us. Heroku is a famous hosting service that is used by many web developers. You can host the website for free, but you will probably want to upgrade to Hobby tier, so that you can have your own database, that costs a few USD per month. If you have committed (via Git) your code to Heroku, the server will start automatically. By visiting `www.<app-name>.herokuapp.com` the website is visible for everyone.



Figure 55 Heroku, the hosting service we chose. (Heroku Image, n.d.)

4.4.2 Domain registrar

Now that the app was visible on `groteprogrammeer.herokuapp.com` (and it still is!), it was time to make sure that we had our own web address, we chose for www.groteprogrammeer.nl. To make use of that domain, you need to claim it. You can do that via a domain registrar, we chose for Strato, because we knew it was cheap. For a dollar a year, we can now use that domain. Linking the server, hosted on Heroku to our domain, claimed on Strato, was rather difficult. Not because the process is difficult, but because we did not understand how it works. Luckily, we do now: this is how it works in short.

Using OpenSSL, a Linux program that you can use on the server, one can create a .cert and .key file, that have all the details about your website. Uploading these to Heroku results in an SSL certificate, so that you are allowed to host your site on https, the safe version of the http-protocol. Now you can copy the DNS (Domain Name System) from Heroku to Strato, so `groteprogrammeer.nl` “listens” to our app on Heroku, so it has the same result as going to `groteprogrammeer.herokuapp.com`.

Chapter 5 – Creating the front end

Nowadays, creating websites is accessible to almost all people with an internet connection. One barely needs any knowledge of coding languages or design principles. You can just go to Google Sites, Wix or if you want something a bit more complicated: WordPress. But where is the fun in that? It is like calling yourself an artist because you traced the Mona Lisa with a pencil and see-through paper. No, we wanted to create our own design, our own layout, our own website. Absolute freedom.

5.1 In the beginning: Researching and learning about the front-end

All of our research started off with 25 hours of lectures from CS50's Web Programming with Python and JavaScript given by Brian Yu. The course taught us many things about programming the front-end using HTML, CSS, and JavaScript. We were very excited to start programming, so we quickly jumped into the massive world of web programming and started our journey. Our simple CS50 bubble burst immediately, with CS50's lectures alone we could not make it very far, we realised. There was way more to learn.

5.1.1 Researching Material Design

While researching about website layouts Jorijn (a friend of ours) told us to look up Google's Material Design. Google made an entire website dedicated to explaining how their design philosophy works. The reason we chose to go with material design is because we liked the clean look of it. Material design is also very well explained so anyone can understand it easily, it does not require too much knowledge about colour theory or other artistic terms as their website explains it all. We also think consistency is important for our websites so students can instantly recognize relations (for example all important terms are bold and coloured in our secondary colour). It is also important to make the website not too distracting for the students to help them concentrate, material design is excellent at adding details that do not distract to keep the website simple and clean. The research consisted of reading their website and taking notes where needed.

5.1.2 Vuetify

After researching Material Design, we wanted to go with a front-end framework. We did not really like the look of Bootstrap, so Bootstrap was quickly shoved off the table even though bootstrap is a great front-end framework. The reason we did not choose bootstrap was because it is not very faithful to Material's design principles and lagging behind compared to other frameworks. It has changed very recently, and the new version looks way more like Material design (Otto, 2021). But at that time, it did not, so Material recommended us Vuetify. Vuetify is built for Material and therefore is the perfect front-end framework. It looked the way we wanted it to look and seemed fairly easy to use and could be included via a CDN, so no install was required. We did however overlook the 'Vue' in Vuetify.

5.1.3 Vue

Vuetify required Vue, as it turns out. Vue is a new JavaScript framework released in 2014 by Evan You. Vue is based on Angular as Evan You said: "I figured, what if I could just extract the part that I really liked about Angular and build something really lightweight." (Cromwell, 2017).

And so, he did, Vue is only 20Kb in .zip format and is blazingly fast. It lets you create templates which are pieces of HTML attached to JavaScript which can be reused repeatedly. Those templates then get their own HTML tags. Reactivity means that the code updates itself in real time. We tried to install Vue, but it required Node. And Node does not like working together with Django (our backend) because Node is already a server on its own and running two servers simultaneously required lots of knowledge we did not have. It can be done however, so we tried. And failed. Installing Vue was not an option anymore, luckily Vue can also be included through a CDN. Including it via a CDN can be done very easily: copy-paste a few lines of code and you can use most of the framework. And that worked. But not well, because including it via a CDN is meant for testing purposes so the framework lacked many critical functions. We had, at that moment, put countless hours into researching Vue and getting it to work through a CDN. Until one moment where a comment on *Stack Overflow* (no source) said to the person asking a question that he should first learn to use JavaScript and then move on to frameworks and libraries like Vue and Node. We did not have any prior experience with JavaScript so using Vue was, now obviously, almost impossible.

5.2 The departure: Ditching Vue and Vuetify after a month

All progress on the front-end was deleted with only the idea of how the layout should look remaining. Ditching Vue was a hard decision as everything had to be done anew, new research had to be done and most of all: we still would not have anything remotely presentable. We went through with it.

5.2.1 Material's own framework

We decided that we *had* to have a front-end framework because not using one meant creating every element of the website by hand, which would cost us much time. So, after some researching, we saw that Material had its own (not so complete) framework. It immediately became apparent that it was nowhere near complete. The documentation was not clear, and it missed a lot of components we needed. So, the search went on. (Material, n.d.)

5.2.2 Semantic UI

Semantic UI was big contender. We did not like Bootstrap for how it looked, but Semantic did look like Material and it was also complete. The documentation is very extensive, and they have many components which can be used on the fly without requiring much tweaking. We did not do a full install of Semantic UI because that required Node. We managed to import it through a CDN because we did not succeed in doing a full install, but the CDN worked fine. There was one downside to importing it through the CDN: you could not change the template colours. So that had to be done manually with CSS and it broke all kinds of things like the colour of a button not changing when the user hovered above it. We ended up using more CSS than Semantic UI. Semantic also has many components that require JavaScript, or more specifically jQuery. (Semantic, n.d.)

5.2.3 Falling in love with jQuery

Semantic required jQuery to work properly, so that also had to be included through a CDN. When searching for errors and bugs about JavaScript we found that half of the answers would be given using jQuery. Everyone seemed to use it. At first it seems weird but once you start to use

it, it quickly starts to feel familiar and makes writing JavaScript a hundred times faster. In a matter of days, we switched from using only JavaScript to using only jQuery.

5.3 The law: Applying material design

Through all this we tried to stay faithful to the rules given to us by Material design. This paragraph will explain how we managed to apply those rules.

5.3.1 Applying colour

The first thing you do when designing the layout of a website is choosing a colour palette. We went with an intense yellow as our primary colour. We also chose two shades of that yellow so we could use the colours on top of each other, or to indicate hierarchy between elements. We chose yellow for a very simple reason: we just liked the way it looked. Our secondary colour became a bright pink because it formed good contrast to the yellow. We do not use that colour often because it is the secondary colour. We do use it to highlight important words and pieces of code (see *Figure 58*). We chose to make it bright because of our bright primary colour and to make it stand out more because we used it scarcely. This is to indicate importance and separate it from the other elements which use the primary colour. Our primary background-colour is white because we chose to make a website with a light theme. We have decided that we do not want to create a dark mode as that would require a lot of time, which we do not have.

We also use error and succession colours to indicate errors, but also to indicate whether a student has answered correctly or not. Our primary colour is our neutral colour. When a message comes in that is neither bad nor good, or if the student has requested an explanation of the question, then the primary colour is used. By using the same colours often, a user will not even have to read the message because they already know whether it is bad or good. The user will recognise many of these patterns across our website so they get used to it quickly.

```
--accent1: #fbb947;
--accent1-lighten: #fcd7d;
--accent1-darken: #fba918;
--accent2: #fb4789;
--white-darken: #f0f0f0;
--greytext: #616161;
--success: rgb(37, 199, 86);
--error: rgb(255, 72, 0);
```

Figure 57 our CSS colour variables

nen gebruiken om onze problemen c
des: **graphical programming**. Dat is
I zegt het al, je gebruikt hier geen bl

Figure 58 Applying the secondary colour.

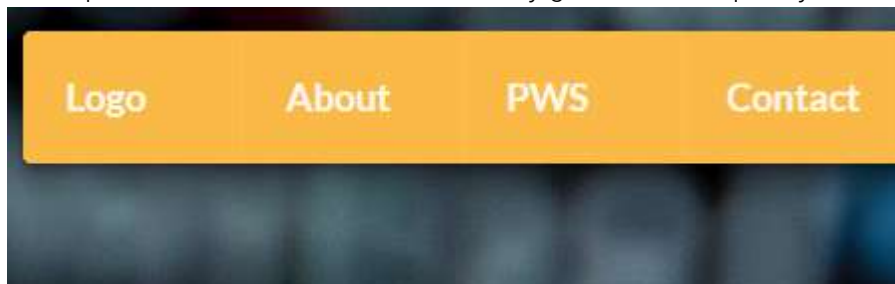


Figure 59 A yellow coloured navbar.

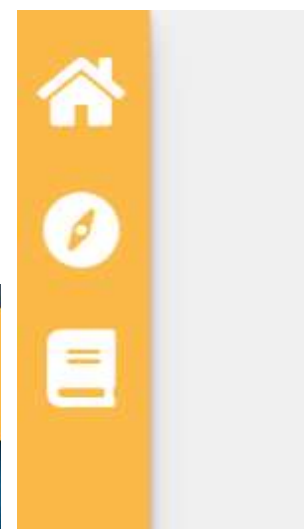


Figure 56 Another yellow, vertical navbar.

5.3.2 Navigation

Material uses three main principles for the layout namely: lateral, forward, and reverse navigation. See 2.3.3 *Navigation System*. Our main navbar is our top hierarchy, you can use it to move between pages like *Home*, *About*, *PWS*, *Contact*, *Register* and *Login*. You can move down through *Login* and *Register* to go to the student's page. If you log out, you move back up the hierarchy into the home page. The student's page has its own navbar to navigate laterally between pages like *documentation*, *lessons*, and the *student home page*. We decided that logging in moves you to an entirely new Django app. It is completely separated from non-logged in pages to indicate that logging in gives you special features.

5.3.3 Applying elevation

We used CSS variables for applying shadows to our elements. Those shadows have been specifically made for a material website. Currently, we have 3 different types of shadows for different elevations. The largest one we only use for pop-up messages and other elements which do not have a relation with the content. The smaller shadows are used to separate content from each other. Both navbars also float to let content move under them and to indicate a higher level of importance. The same is for code snippets and question cards. Both have a small shadow applied to them to show importance and to separate them from the content. Important words do not have a shadow because they are part of the text.

```
--shadow-1: 0 1px 1px 0 rgba(0,0,0,0.12),
            0 2px 2px 0 rgba(0,0,0,0.12),
            0 4px 4px 0 rgba(0,0,0,0.12),
            0 8px 8px 0 rgba(0,0,0,0.12);
--shadow-2: 0 1px 1px 0 rgba(0,0,0,0.12),
            0 2px 2px 0 rgba(0,0,0,0.12),
            0 4px 4px 0 rgba(0,0,0,0.12),
            0 8px 8px 0 rgba(0,0,0,0.12),
            0 16px 16px 0 rgba(0,0,0,0.12);
--shadow-3: 0 1px 1px 0 rgba(0,0,0,0.12),
            0 2px 2px 0 rgba(0,0,0,0.12),
            0 4px 4px 0 rgba(0,0,0,0.12),
            0 8px 8px 0 rgba(0,0,0,0.12),
            0 16px 16px 0 rgba(0,0,0,0.12),
            0 32px 32px 0 rgba(0,0,0,0.12);
```

via de Run-button. Dan compilet

5.4 The numbers: JavaScript and jQuery

HTML and CSS would not get us very far with making a website that allows for user input and reacts to that. As has been explained in 5.2 *The departure: Ditching Vue and Vuetify after a month* we stopped using Vue and moved to JavaScript and jQuery. Making this switch has made our lives way easier as we finally understood what we were

gratis. Klik op open in Replit (rechtsboven) of je 'Log in' en 'Sign up'. Als je al een account hebt, klik op open in Replit. Klik op Google, Github en Facebook. Wij gaan nu naar de vraagkaart om advies. Klik op het zwart-witte logo

Figure 61 The question card has a shadow, but the important word has not.

```
if (onscroll.scrollTop >= 1168 && onscroll.scrollTop <= 1708) {
  $(this).css('background-color', 'white')
} else {
  $(this).css('background-color', 'var(--accent1)')
}
```

Figure 62 An example of changing the colour of the navbar at certain thresholds.

doing. We first started off by making a few simple pieces of code like when I scroll this far: change the colour of the navbar to white.

Once we started to get more experienced, we moved on to harder things like generating question cards from server data. The question card code has gone through many revisions and started off as something you had to remake every time you made a lesson. Which we found not to be very practical, it had to become way easier to add questions to your lessons. So, we came up with a JSON API. The questions are now written in a different place from the lessons, but they are linked together. When the user's computer receives the question data, it begins to parse it into a readable form for JavaScript. The function then looks inside the lesson to find the places where it can put the questions. It then generates them accordingly.

```
// Builds the question card from JSON
function buildmcquestion(qnum) {
  $('.multiplechoice').eq(qnum).append(`<h1 class="questiontitle">${
    qobj.mcquestions[qnum].options.forEach(element => {
      $('.multiplechoice').eq(qnum).append(`<div class="ui radio ch
    });
  $('.multiplechoice').eq(qnum).append(`
    <button class="ui button check" id="checkanswer">Check</butto
    <button class="ui button retry" id="retryanswer" style="displ
    <button class="ui button show" id="showanswer" style="display
  `);
  $('.multiplechoice').eq(qnum).append(`<div class="correctanswer">
```

Figure 63 Code that generates multiple choice questions.

Another complex feature we made is the automatic recognition of keywords. The code loops through the lesson and adds HTML around the keyword to make it appear different from the rest of the text. This is also very useful for making the lessons as you only must choose which words to highlight instead of writing extra HTML for every keyword you used.

5.5 Reflection and conclusion

To conclude: making the front-end has not been easy. We've had to do everything at least four times, sometimes even more. Not all problems were big and complicated ones, but there were a few. One of them was obviously using Vue. Choosing to use Vue was a big mistake, we did not have enough knowledge about JavaScript to understand what we were doing. Also, putting countless hours into researching frameworks which brought us nowhere was a mistake. We should have kept it simple by using just CSS as in the end it would have saved us time. We thought using frameworks would save us time but instead it made things way too complicated and slowed us down. However, we should have used jQuery earlier because it was one of the only frameworks that have helped us creating the website faster. Furthermore, we are very happy how the website looks. Using material design was a great choice as it guided us to create a responsive and good-looking website.

Intermezzo

So, you have finally reached it: the middle, the centre, the intermezzo. You seem to have already made it thus far. If you do not like this 'breaking of the 4th wall', then you should skip this already and start part 2.

What? Still here? Well, I guess you are sick of all this gibberish about programming, colour theory, and all that. You want something else, something not so ... mathematical? In the Netherlands we call this 'exact subjects' or 'beta subjects', those are the more scientific subjects like physics, chemistry, and biology.

You want me to tell you a story? One like at the start of this thesis?

Fine, I will give you a story; but you will have to continue reading part 2 after the story, OK?

Once upon a time there was a Vincent and a Christiaan, Vincent wa-

You say you know this story already? No you do not! It is not about programming, nor this thesis! So, let me continue.

Vincent was a king, not just a king, he was king of seven kingdoms. Christiaan was the governor of just one of these kingdoms, the most northern one. Even though Christiaan was Vincent's servant, they were still great friends. One day, Vincent's most trusted advisor died unexpectedly. Vincent needed a new advisor, and who better to choose than Christiaan! They had been friends for a long time and they had fought together to remove the mad king and claim the throne for Vincent. To honour Christiaan's new position, Vincent went on a journey to the north with his entire court. For this, Christiaan had to prepare a great fea-

You say this is based on Game of Thrones? OK, maybe. But I will make sure season 8 will be better. No? You want a different story? Fine.

Once upon a time there was a small creature, a halfling, he looked the same as any human but smaller. His name was Christiaan and he lived a peaceful life in a hole in the ground. One day, the wizard Vincent knocked on his door. Christiaan opened the door and invited Vincent in for a cup of tea, he was curious what the wizard had to say. The wizard told him about gold, dragons and dwar-

You know this story as well? You do not seem to like my stories, do you? Only because I did not make them up? Let me try one more time.

Once upon a time there was a school, not just a school, a school for wizards. In this scho-
I give up. You know all the stories there are! How am I supposed to satisfy you!?

What? You say I should make up my own story? I am not a writer! This thesis might have 50000 words, but that does not make me a writer.

We do not have the creativity to write a story like that, we will never be that good! So you say that we should at least try? That we should try things that are out of our comfort zone? You must be crazy! That will just result in a great fiasco, we will just embarrass ourselves; no, we would rather keep to the things we know and understand. The world is too big for us, we are but a drop in the ocean.

You say the ocean is made up of drops...?

We are DONE here! Continue to part 2, please!

Part 2

CS in the Class

How to teach computer science.



Figure 64 On our website, students learn programming. (Study Computer Science, 2017)

Chapter 6 – How do others teach CS?

Ever since the invention of the computer, people need to be taught how to use such a device. Therefore, computer science studies have risen. And now that computer science has grown into what it is today, the study is at many universities and colleges one of the largest courses they teach. So, of course, we also took some inspiration from our colleagues over at some schools we ought to be teaching CS right.

6.1 CS50

"Harvard University's introduction to the intellectual enterprises of computer science and the art of programming". That is how CS50, a Harvard course for freshmen or sophomores alike, who take an interest in computer science, introduces itself. This course is not only the start for a computer science major, but it also acts as an introduction to computer science in general, so that students with a different major can create programs with useful applications in the academic or scientific field they major in. Taught by David J. Malan since 2007, the course has seen a leap in students: in 2006 only 132 students enrolled in the course; 282 the year later; that number has grown to 735 in 2019. That is more than a 5x increase over the number in 2006, the last year without Malan's lectures. (C., 2018)



Figure 65 CS50 is Harvard University's largest course. (Amazon.Com: CS50: Apps & Games, n.d.)

6.1.1 Structure

Since Malan took over CS50 in 2007, the course has seen many reinventions of itself, but the structure, for the last years, has been the same. The course has 10 to 12 weeks with its students to show them the "intellectual enterprises of computer science". This is how they chose to fill the lectures, based on the structure of CS50x 2021 and CS50 Fall 2020 (the same course). (CS50, n.d.)

Week 0

In Week 0, the course starts with the abstraction of computers. How do they work? What are bytes and bits? How can those be used to show the programs, videos and games on your Macs or PCs? What are pixels and frames per second? How does one represent numbers in binary? Then, the focus shifts onto the first programming "language" of the course: Scratch. A graphical programming language made by people from MIT. Scratch is a perfect language to introduce students to programming concepts, such as variables, functions, conditions, and loops.

Week 1

In the second week, although, confusingly, called Week 1, of CS50, students are taken away from the puzzle pieces and colourful blocks of Week 0's Scratch and dropped into the language C. In Week 1, the basics of C are explained and covered. Students are asked to create programs for which you need print-statements, if-conditions, and for-loops.

Week 2

In Week 2, Malan talks students through the boiler-plate code (code that is the same for each program) of C: what does it mean and why is it there? Then, he talks about programming tricks: like rubber duck debugging, where you use a rubber duck to talk your code through to, so you (oftentimes) see the mistake yourself. Then, the course dives deeper into C, specifically, arrays and strings.

Week 3

In this week, Malan takes the students into a new world, the one of algorithms. How do you sort a list efficiently? How to find a number in an unsorted list? And in a sorted list? How do you judge the "speed", or performance, of an algorithm? What happens to its performance when you increase the size of the list to a trillion items?

Week 4

In the fifth week of CS50, students are introduced to computer memory. Not from a hardware perspective, but from a software view. The course introduces students to the idea of pointers and the functions like malloc (memory allocation) and free (free memory). Also, Malan zooms in onto the computer's Random-Access Memory or RAM and explains how a program uses different parts of that RAM to its advantage.

Week 5

This week is the last week the course spends in C. This time: data structures. Now that students (are supposed to) know how to work with pointers, the ideas of a linked list, where each element has a pointer to the next element; and hash tables, wherein you split the workload by using multiple linked lists; are introduced to students.

For years, the first five (or four) weeks of the course were spent in C, as described above. All the subjects below are also part of the syllabus for many years now. However, the order in which they are taught to students differs each year. In 2018, they first discussed HTML and CSS before diving into Python and then Flask and SQL, different from 2020.

Week 6

Having spent five weeks in C, it is high time to switch to a more modern language: Python. The course uses this week to translate the knowledge from the past five weeks into simple, syntax-light scripts. Then, it focuses on the advantages and disadvantages of Python. It may be simple to write, but it is slow to run. So, programmers should choose their language wisely.

Week 7

In this week, students dive into the weeds of databases. How can you use SQL (Structured Query Language) statements, or queries, to get data from the database? How do you save databases? Students are shown that you can use other's database, for example from IMDb, to create your own programs!

Week 8

In this week of CS50, students are taught how to create web pages using HTML (HyperText Markup Language). How to style them using CSS (Cascading Style Sheets) and how to make them pop and vivid, or rather lively, with the use of another popular programming language JavaScript.

Week 9

In the second to last week of this course, students are shown how to use the knowledge from the past three weeks, Python, SQL and HTML, CSS, and JavaScript to create their own websites and web applications. For this, the course uses the micro-framework Flask.

Week 10

The last week of CS50 is, traditionally, spent by talking about the use case of CS50 (2018's lecture was called "Life after 50") and about security. How can you defend yourself against hackers and what techniques do hackers use to hack your program, website, or app?

6.1.2 Problem sets

Every week, after the lectures, students are to solve problems in a problem set. Each problem set has approximately 3 problems the students have to solve. The very first problem, in pset0, asks the students to create any program in Scratch (as long as it holds up to some standards). In pset1, the students' main problem is called Mario. They are supposed to use some loops to create a Mario-like pyramid, like shown below.

```
#  
##  
###
```

In week 2, the major exercise is called Caesar, where students use Caesar's encryption method and character arithmetic to encrypt messages.

The next week, students are supposed to create command-line programs that simulate elections, either via the runoff or Tiedeman convention.

In the fifth week, the problem set involves the problem Recover, wherein students get a raw (i.e., only 0s and 1s) file, that is, supposedly, recovered from a broken camera. Students are asked to find the JPEGs in the file, via the unique header.

Then, students use hash tables, or trees (or tries) to create their own spell checker. By importing 143,091 words from a file and checking all of Shakespeare's works, students face to problem that comes with sloppy code: a lack in performance.

When the students transitioned to Python, they are asked to translate some of the programs described above from C into Python.

Pset7 has a fun problem, that of Fifty Ville (a new problem in 2021), that requires students to find a thief based on ATM transactions, license plate detectors and interview. All of those are stores in a database, so students need to use their SQL skills to unravel the mystery.

When students learned how to create web pages with HTML, CSS, and JavaScript, they need to create their own 'homepage', which are at least four HTML pages, styled with CSS that do whatever the student wants.

The last problem is a very cool one: students are asked to create a web application on which users can buy and sell stocks, based on real-time prices.

After doing all these, and more, exercises and solving all these problems; students now must turn to their own creativity: it's time for the Final Project. Students are allowed to create anything with any means, as long as you program! See <https://cs50.harvard.edu/x/2021/gallery/> to see what students create.

6.1.3 Languages

In computer science education, language choice is always a tricky subject to start discussing. Many people view it totally different, which will lead to a lot of disagreements. In 2018, we took the CS50 course ourselves, as an introduction to programming. So, we can judge whether the language choice of CS50 was spot on or miserably mistaken, at least for us (and estimate how this will work for other students).

CS50 starts with a graphical programming language called Scratch. By dragging and dropping puzzle pieces one can code their own games, animations, or interactive programs in the (web based) Scratch environment. It serves a good introduction to programming concepts, like functions, variables, and loops. It is not, however, very representable of 'what's out there', as all programming languages use a very different type of syntax: text.

After the first week of CS50, it transitions to C: a language developed in the 1970s, which makes it ancient. Albeit old, C is a language that is relatively small, so that CS50 can show the language almost in its entirety, having left out only some parts, as functions pointers and unions. Also, C is very low-level, having only Assembly Language that comes closer to the hardware, which makes it very real, without any abstractions. Lecturer Malan summarised it as "In C, there's no magic. If you want something to be somewhere in memory, you have to put it there yourself. If you want a hash table, you have to implement it yourself." (*Why Does CS50 Use C?*, n.d.) On the other side, C is a very arcane language, meaning that you must know a considerable amount about hardware, before you can use, for example, a complicated data structure. CS50 sees this a strength, because in one week, wherein they transition from C to Python, those many lines of code to create a linked list can be comprised into only one: `myList = []`. (*Why Does CS50 Use C?*, n.d.)

After spending the first half of the course in C (and Scratch), it moves on to Python, a more modern, popular language that is universally applicable. Whether you want to create a website, encrypt a password, hack your aunt's bank account, or create an artificial neural network that can recognise monkeys by region, it can all be done by Python. And the beauty is, Python is popular (in contrast to C), this means that for many things, someone else has already created a library (a piece of code you can import and use). C is quite a verbose language, meaning that you will need a lot of characters and boiler-plate code (code that you write in every program) to do something simple. That is not the case, however, with Python. The language is very English-like, which makes it suitable for beginners, as the syntax will not form a barrier.

```

1  #include <stdio.h>
2
3  int main(void)
4  {
5      int array[5] = {'A', 'D', 'F', '!', 'r'};
6
7      for (int i = 0; i < 5; i++)
8      {
9          printf("%c\n", array[i]);
10     }
11 }

```

Figure 66 A program to print each element in an array in the language C.

```

1  list = ['A', 'D', 'F', '!', 'r']
2
3  for character in list:
4      print(character)

```

Figure 67 A program to print each element in a list in the language Python.

After translating all the programming knowledge from C into Python, CS50 transfers to web development. Adding a whopping four new languages: SQL, JavaScript, HTML and CSS. The language choice for web development, or rather front-end development is rather trivial: HTML, CSS, and JavaScript; there are simply no competitors. For the back end of web applications, CS50 uses the Python micro-framework Flask, that itself uses SQL queries for database interactions. So, in total CS50 uses many different languages, and therefore, quoting from Malan, they hope that “at term's end a student will not say “I learned C” but, rather, “I learned how to program.””.

6.1.4 Teaching style

CS50 is not, in our eyes, the mainstream computer science course. Many people would imagine a course to be taught by an unenthusiastic, old, and boring professor that does not really want to teach. CS50 is different; Malan is different. Many of the first-time viewers of a CS50 lecture are (positively) surprised by his enthusiasm and energy. For more than two hours, Malan spends energetically almost running around the stage, while talking, like a kid about his painting, how cool computer science is and why it is so beautiful. It is, simply said, inspiring.

The course does not, in contrast to many others, fall back on a textbook. It uses solely Malan's voice and the introduction (in written form) of every problem set to communicate with online students. Every lecture, in contrast, is supplied with a set of notes: a summary of all the things Malan said and the programs he made (which can be downloaded as well).

That is, what by us at least, is used as a book. It is the reference for things you have forgotten how they work.



Figure 68 A demonstration to show you need to be very precise when coding. (CS50 2018 - Lecture 1 - C, 2018)

Another famous property of the CS50 lectures is its demonstrativeness. Computational ideas are oftentimes accompanied by a demonstration. In the year we took – 2018 – the course focused heavily on being precise and used a demonstration like this. A student was called onto the stage and given everything to make a sandwich. Everyone, naturally, knew how to use those tools and ingredients to create a lovely lunch. But Malan, who did the demonstration, as well as the student called forward, was stubborn. The audience was asked to give instructions on how to create that sandwich from its parts. He did everything according to the instructions given by the students, but always managed to find a corner-case, one where he did manage to follow the instructions, but did not reach the intended result. For example, when he was asked to smear the peanut butter onto the bread, he put it on the wrong side, making a mess. This was an important moment in the show (while watching you forget it is a lecture): he thought students to be precise when they write code, to think of those corner cases; not by saying it bluntly and boringly; no, he used a fun demonstration that will stick with the students for the rest of their carrier.

In the fall of 2020, during the COVID pandemic, CS50 moved from Sander's theatre (their normal lecture hall) to a real theatre, with a prop shop. Together with the team from the theatre, they used wooden boxes and arrows to demonstrate a linked list. It was, in our opinion, a highlight in the 13-year history of the course under Malan.

6.1.5 Online teaching done right

CS50 is not only one of the largest course on Harvard University, it is better known by another figure: 2 million. That is the number of students that have taken another version of the course, just like us: CS50x. Harvard's counterpart on the course website EdX. To this course more than two million people have enrolled. The best part is this: it is (said to be) no different from the on-campus course. All the materials, lectures and problems are freely accessible on their website: <https://www.cs50.harvard.edu/x/2021>. There is one part of the course in Massachusetts that

has, as of yet, not been translated to the X-version: sections. In sections, the students get to work on their exercises, while being helped by TAs (teaching assistants) or TFs (teaching fellows). For now, you will need to seek help on other forums, like Reddit, Stack Overflow, Ed, or Discord. (edX, n.d.)

6.2 CSCircles

The University of Waterloo has created another course we took in a math class a couple of years ago. This course is called CSCircles, it is considerably less detailed and not of the same scope as CS50 but it has shown us how to teach computer science on a website (and not in a lecture). In some simple to understand text it explains what you need to know and then uses some (mostly mathematical based) programming exercise for you to figure out the rest. The content, however, is not what we are interested in. Look at this web page: <https://cscircles.cemc.uwaterloo.ca/4-types/>. If you are unable to open the link, see the screenshots below.

The page has some elements:

- Theory: simple text that tells you what to know.
- Examples: a (working) code example where you see what the code, described in the text above does.
- Questions: a question that makes you think about the way a function works a little deeper.
- Coding exercise: an exercise to make sure you grasp the subject.

This is a basic structure of how any course (and specifically a computer science one) progresses from unknown to comfortable with a subject. (*Using This Website / Computer Science Circles*, n.d.)

6.2.1 Screenshots

These are taken on the 21st of September 2021, from Lesson 4: Types.

4: Types

From this lesson onwards, examples and code input boxes have buttons labelled **Open in console** and **Visualize**. Use them to help debug and explore the code.

In the [Hello, World!](#) program we saw that Python was able to repeat a sentence back to us. We have also seen several examples of arithmetic with numbers. Numbers and sentences are fundamentally different objects, and it causes a Python error when you try to mix them in the wrong way.

Example

Trying to compare a string and a number.

```
x = "Hello, World!"  
y = 35  
z = max(x, y)
```

Run programOpen in consoleVisualize

As you can see, we get an error saying that the two arguments to `max` are of different types. The error is a good introduction to the rest of the lesson:

- "Hello, World!" is a **string** value, which is shown as `str` in Python. A string is any sequence of numbers, letters, and punctuation; we will learn more about them in lesson 7A.
- 35 is an **integer** value, which is shown as `int` in Python. An integer just means a *whole number*; for example, 42, -12, and 0 are integers.

The lesson starts with the foreknowledge ("In the (...) program we saw that". It continues the lesson by introducing the student to a problem in Python (in this case the problem of datatypes). It immediately outlines what causes that problem. This is an example of explaining very directly. There are no stories: they say how it is, and why that is the case.

Continues on the next page →.

Using an object of a bad type is a very common cause of errors in programs. It is like trying to drink a sandwich: you can't do it because you can only drink things of liquid type, and a sandwich is of solid type.

You can determine the type of an object by calling the `type` function on it.

Example

Some examples of types:

```
print(type("Hello, World!"))
print(type(34))
print(type(1.234))
```

Run program
Open in console
Visualize

(The meaning of `class` is similar to `type`) The above example demonstrates that numbers are further divided into two different types, `int` which we mentioned above, and `float`, which is used for storing decimal numbers. You should think of floats as inexact or approximate values (we will explain more in lesson 7B). You can usually mix float values with `int` values, and the result will be another float.

Example

Mixing an `int` and a float:

```
x = 1.2
y = 2
z = x * y
print(x, y, z)
print(type(x), type(y), type(z))
```

Run program
Open in console
Visualize

Now that the problem is established, they first compare the problem with something everyone understands: trying to drink a sandwich. That, of course, is impossible and so is combining two variables of different types, it simply does not work.

While explaining types, they realise there was one students did not know, the float. So it is high time to introduce them to it. Using another example (directly using the new knowledge about types), they explain that a float and a integer make a float.

Continues on the next page →.

In fact, what Python really does when you mix a float with an int is that it converts the int to a float, and then works with the two floats.

Multiple Choice Exercise: Floating ()

If we change 1.2 to 1.5 in the above program, what is first line of output?

Your choice: Select one

Check answer

It is often necessary to change data from one type to another type. Just as you can convert a sandwich from solid to liquid form by using a blender, you can change data from one type to another type using a **typecast function**. You write the name of the desired new type in the same way as a function call, for example

```
x = float("3.4")
print(x-1)
```

changes the string "3.4" to the float 3.4, and then prints out 2.4. Without the typecast, the program would crash, since it cannot subtract a number from a string.



Sometimes, Python does let you combine strings and numbers using arithmetic operators. The statement `print("hots" * 2)` prints hotshots. Python's rule is that multiplying a string *s* by an integer *n* means to put *n* copies of the string one after another. We'll see later that "addition of two strings" is also well-defined in Python.

Various typecasts behave differently:

- converting a float to an int loses the information after the decimal point, e.g. `int(1.234)` gives 1, and `int(-34.7)` gives -34.
- converting a str to an int causes an error if the string is not formatted exactly like an integer, e.g. `int("1.234")` causes an error.
- converting a str to a float causes an error if the string is not a number, e.g. `float("sandwich")` causes an error.

Using the example that was used to explain the difference between integers and floats, they continue with datatypes. The students are now able to establish the type of a variable, but how to change it? Using a simple example, they show how you can typecast in Python. Then, comes in a yellow box with a danger symbol (so this is something to take notice of) an exception of what has been explained previously. This makes the course simple to understand but also complete, as it does not skip over things that make it difficult.

A common use of typecasting that we will see soon is to convert user input, which is always a string, to numerical form. Here is a quick illustration:

Example

Example of typecasting:

```
inputStr = "12"                # a piece of user input
print("The input type is", type(inputStr))
x = int(inputStr)
print(x, "is of type", type(x), "and its square is", x*x)
print(inputStr * inputStr)    # this line should cause an error
```

Run program
Open in console
Visualize

Here is one more exercise to finish the lesson.

Because there are now lots of editor commands, some have been moved into the menu labelled **More actions...**

Coding Exercise: Tasty Typecasting

Write a program to help you feed your friends at a party by doing some math about *square pizzas*. Assume the grader defines a string `inputStr` for you, which is a decimal string meaning the side length L of the pizza in cm. The area of the pizza should be computed using the formula $A = L * L$. Then, assuming that each person needs to eat 100 cm² of pizza, compute the number of people it can feed, *rounded down to the nearest integer*. [Hint](#)

Example: if `inputStr` is "17.5", the area will be 306.25 cm², so 3 is the correct output.

delete this comment and enter your code here

Run program
Open in console
Visualize
History

More actions...

Once you are done, head over to the next lesson.

3: Comments and Quotes
4: Types
Next 5: Input
6: If

Lastly, they explain how the knowledge the students have now acquired is mostly used (in this case with converting user input) and finish the lesson with an exercise. In this exercise students have to do some math with a fun context: dividing a square pizza among people.

You can see that the lesson is mostly examples and descriptions of those examples. CSCircles has created a very short and easy to understand course that does go deep into Python.

Chapter 7 – Teaching, teaching, teaching

Famous science writer and author Roger Lewin has understood teaching completely:

"Too often we give children answers to remember rather than problems to solve" - Roger Lewin

It is this quote we kept in mind while thinking of a way to teach and how to structure our website. How do you teach a class? What are effective ways to instruct?

7.1 Method

To answer these questions, we interviewed four of our teachers, as we are no teacher ourselves, we did this to try to understand the art that is teaching and its ways to clarify - sometimes daunting - matters. We came up with some questions in advance:

- How do you structure your lessons?
 - Are there parts of a lesson that come back every time?
- How do you construct an explanation?
- How do you shape the lessons over a chapter?
- How do you prepare for your lessons? How much time does that take?
- How do you keep the attention of your students?
- How to motivate students for your subject?
- What do you miss of the book you teach with?

Described below are the notable answers that each of the teachers gave. The full transcriptions of the interviews can be read in the appendices.

7.2 David Heerkens, a physics teacher

The first interviewee was David Heerkens, a young and enthusiastic physics teacher.

7.2.1 Explanations

He explained that for him, an explanation consists of, fundamentally, these three things:

- what do we know?
- how does this new thing relate to that?
- how does this new thing work?

Oftentimes, the first question is answered by asking the students some questions, either very theoretical or practical, or giving them an exercise they should be able to do by now. The students have now activated their foreknowledge: what do they know? Then, they are introduced to a problem or situation they cannot solve or figure out right now, that is where the new matter comes into place. Now that the students know how the new matter works, or what it is, they are ready to process the new knowledge. They do this with exercises, where they need to think a little harder on what has been told.

7.2.2 Systematische natuurkunde

With physics, the book *Systematische Natuurkunde* (which loosely translates to *Systematic Physics*) is used. According to Heerkens, advantages of this book are that it is direct, so without an entire story on why something works, it clearly and concisely tells how the physics works. SN has an online platform, where students can view the book (in case they have forgotten to bring it) and do extra exercises. It means that there are a lot of options for students to familiarise themselves with old material or understand new matter better, which is always a good thing. Another thing he does not really like is the choice of what is put in the book, and what is online. The exercises of the same level as the test or exam, the "practice tests" are not in the book itself, but only available in the online environment. This simply means that less people do the test, and so that they practice less. Heerkens would have liked it if they had been in the book itself.

7.2.3 Videos

Heerkens has added onto SN his own videos because he felt that an on-demand explanation would be something that the students want. And rightfully so, every year days before exam, the views of all his videos peaks, as students repeat the matter in their own time, with the help of his videos. He emphasises as often as he can that every explanation that uses another way to explain the subject, can be seen as a new explanation: every time students learn something new.

7.3 Lars van Bokhorst, a chemistry teacher

With van Bokhorst, the conversation was very interesting. He showed how, even teachers, do not always know exactly what they want. He promptly said "Just right" to a question about the number of exercises in *It's all Chemistry*. It turned out however, that he missed some easy assignments, the ones you need to do a dozen of, to let students explore the concept of the matter. He always uses extra exercises on sheets of paper, instead of those from the book, because he feels they do not fit the need of students well.

7.3.1 Exams

Another interesting topic we discussed are the examinations. How do you test how – in this case – chemically fluent a student is? Well, not like CITO does. (CITO is the Dutch organisation that makes all general exams, also those for chemistry.) He explained how CITO uses so much text, that it defeats the purpose. On some HAVO exams, you can answer most of the questions, without having read the text. You do need them, however, in VWO exams. He outlined how the chemistry tests should not be about who can read, but who can do chemistry.

7.3.2 Practical exploration

Another element of education that van Bokhorst thought was very important was doing thing practically, in the form of an experiment. He feels that students have difficulty with understanding the bare chemistry but will get a better feel for the subject after they have seen something burn, explode, dissolve, or change colour. He explained that he introduced qualitative analysis for his classes. In this experiment you get a substance, and you need to determine what it is, using all your tricks and knowledge. This lets students repeat what they know about chemistry, but also lets them explore and wonder about the mists of molecules.

7.4 Matthijs Alderliesten, a physics teacher

When we asked Alderliesten about his style of explaining physics, he (repeatedly) came back to the same principle: repetition. He argued that the more structured your lessons are, the better they become, as students know what to expect. He always uses the same structure for a lesson: discuss exercises from last lesson, explain a new topic and let students do some exercises. This makes every lesson structured and expectable, so that students can fully focus on the new matter.

7.4.1 The book

With physics, we use Systematische Natuurkunde (SN), about which Alderliesten is quite positive. He likes the fact that it is direct. Other books use techniques like: you see this in the world, how does it translate to physics? Something that is, in his opinion, too long-winded. He does note, however, that the book could do better in terms of exercises. Because, they are at exam level, which makes them too difficult for first-time process. He would like to see some easier exercises, so that students can more fluently transfer from explanation to exercises.

The online environment, where students can practice extra with the topics, is a good addition to the book. However, it is rather difficult to work with, which undermines the purpose of “easily available extra exercises”. Specifically, adding students to classes and seeing their progress is, supposedly, difficult to do.

7.4.2 Learning routes

Another interesting thing he mentions is different routes of learning, meaning that some will take the route in which you learn more and some in which you learn just enough. This works well, as the route fits the students better, however he noticed that after some time, the students who were in the group wherein they did not have to see Alderliesten do an entire exercise, were also listening to his explanation, because it is good to see an experienced physicist do an exercise.

7.4.3 Exploration

Another way of teaching, the, what we call, “explore how it works” method is, according to Alderliesten, a bad thing. Because it may be positive that students have had a hands-on assignment with physics, it often leads to misunderstandings: students are going to confuse several terms, which, if misunderstood, are very difficult to get right. The only thing you can do to repair these misunderstandings is repeating the correct way.

7.5 Kees van Vliet, a mathematics teacher

In the conversation with van Vliet, we explored how you can alternatively teach mathematics. He has a lot of experience using lectures. He would lecture during the first lesson of the week, explaining all the subjects of that week. In the other lessons, he would let the students do exercises and practice with the material, while he was (or his colleagues were) around to help the students with questions about the exercises or the topic. He explained to us that, although it did work very well for him, it is very tricky to get it right. If you are not your class’s lecturer, you do not get to know your students very well. Also, you might not know some analogy or

demonstration the other teacher used. However, if you do teach the class you lecture, van Vliet outlined that it would work great. As students get to practice more in class and, in the meantime, get used to the way of teaching at the university.

7.5.1 Numbers and Space

The book we use for mathematics, Numbers and Space, is a great book to use, according to van Vliet. Because students can use it to read from and understand the maths or just to practice from. It is multi-applicable, which makes it great. However, he misses some easy assignments, as those that are in the book get too difficult too soon, so that students often do not know how to solve a question, because they are not yet equipped to do such an exercise.



Figure 69 Numbers and space.
(Getal En Ruimte VWOBO Deel 1, n.d.)

7.5.2 Repetition, repetition, repeti...

One of the most important things in education, a thing van Vliet could not stress enough, was the importance of repetition. He does this by asking at the start of each lesson some questions that students can answer with the knowledge of last lesson. Also, he tries to spread out subjects as much as possible, so that students do exercises on that topic as much as possible. He explained that doing five exercises is much more powerful if you do them on five different (concurrent) days, rather than doing them concurrently.

7.6 Summary

To summarise everything the teachers have taught us in these interviews in just a few bullets

- Repetition is key: repeat the same matter as much as possible, as students will remember it much better.
- Use easy exercises: often books have too few easy exercises to let students explore the topic.
- Teach in as many ways as possible. Try using all the students' senses. Use videos, text, examples, experiments, animations, demonstrations and more.

Chapter 8 – Ideal vs Reality

How nice would it be to live in an ideal world? Living in a world like that would simply be utopian. Sadly, we do not live in such a world therefore not everything is ideal. This chapter is about the sad reality that not every expectation can be lived up to. We will dive deeply into the thoughts and ideas behind our course and how these have translated into reality; and how some did not translate at all.

8.1 The syllabus

“A syllabus is a document that outlines all the essential information about a college course. It lists the topics you will study, as well as the due dates of any coursework including tests, quizzes, or exams.” (Shorelight team, 2021)

Not only colleges and universities use a syllabus, but many high school courses also follow a syllabus and that is why we also wanted to make one. Also, because it gives a great overview of all the subjects in the course. We have split our syllabus into quarters, each quarter has either 16 or 24 lessons as we expect the computer science course to be a 2.5-hour subject, meaning that the first half of the school year the students have two lessons each week and in the second half, they will have three lessons per week. Each quarter is eight weeks long. The school year is longer than 32 (4 times 8) weeks, but there are exams and other activities that will decrease the total amount of lessons. Our experience (at De Goudse Waarden) is that each quarter has eight weeks to lessons, on average.

8.1.1 Rijksoverheid

In the Netherlands the Ministry of Education creates demands for schools, so they know what to expect on the central exams, but not all courses have a central exam, like IT. However, the Ministry still creates a syllabus, that is because getting a diploma in a certain course should still be fair for everyone (i.e., of the same difficulty in all schools). The students are tested through school exams only, these are exams made by the school. The syllabus tells the schools on what subjects they must test their students. To check whether schools comply, occasionally they can expect an inspection by the someone from the Ministry.

The syllabus (Examenprogramma Informatica Havo/Vwo, 2019) consists of 6 obligatory subjects:

- Domain A consists mostly of the skills learned in the other 5 domains and is generally tested through projects.
- Domain B is all about understanding algorithms and being able to solve problems through algorithms.
- Domain C is all about data and how to store and review it. Students will also learn to use databases.
- Domain D is being able to write programs in any programming language.
- Domain E is about hardware, software, and security.
- Domain F is about design, not only for usability but also security and privacy and the effect on the society.

The government has also made numerous optional domains from which VWO (the highest level in pre-university education) must choose 4 and HAVO (the second highest level) only 2. For now, we have only concerned ourselves with the obligatory domains, as they are most important according to us (and we simply do not have the time to cover all domains for now).

8.1.2 Q1: Computers & Scratch

The first quarter of our course consists of 16 lessons divided into two subjects. The first six lessons will be about the basics of computing. Questions that will come up include: what is a computer? How does a computer calculate? And how can something so simple create complex things? The second part, consisting of ten lessons will introduce programming using scratch. Throughout the second part there will be a number of small projects challenging the students to put their newly acquired knowledge to use.

We chose to start with the basics of computers because there will be students who do not know much about them. So, we made sure that students do not need to have any foreknowledge about the computer and will learn about the basics like ones and zeroes, processors, and monitors. However, some may already know all this, therefore we have added a few extra things to think about like Turing Completeness and instruction sets. This means that after about 3 weeks all the students will be ready to start programming as they will now know how a computer works.

We chose to use Scratch as an introduction mainly because it introduces the students to programming in a non-intimidating way. The language is not text-based, like almost all other programming languages; Scratch uses colourful blocks of code that work the same way as text-based code but are easier to understand: the language is graphical. By dragging and dropping puzzle pieces students can create their own programs, like game, animations, interactive programs and more. Scratch is useful for short and simple programs but not for large complex ones as the language is limited, which makes Scratch perfect to start with.

We have also chosen to include a few projects to put the knowledge of the students to practice. These projects will start small as they will not have learned enough for some larger projects. However, towards the end, they will have mastered Scratch and will be able to make a video game. The last project will take a few lessons and the teacher will help the students with errors and questions they have.

8.1.3 Q2: C

The second quarter also consists of 16 lessons, but this time it will be about only one subject: programming in C. In this quarter the students will learn the programming language C, an enormous difference from Scratch. C is a language that is “unmagical”, meaning that it does not do something without the student telling it to do so. This may make the programming process more difficult than in other languages, like Python. However, it also has a considerable didactic advantage: students understand what is happening, nothing happens without them asking for it. And, because the students will already have learned the logic behind programming through Scratch, they will know how an *else if* statement works, so that they can focus on the things that are different in C. They will, also learn a few new things like arrays and linked lists.

C can be seen as the Latin of computer languages: once you learn Latin, one can learn languages like Italian, French, and Spanish very easily as they are closely related. Just like most real-world languages are based on Latin, so are most programming languages based on C. Once you learn C, learning C++, Python or JavaScript is incredibly easy.

In this semester there will also be many projects to work on after (or during) the lesson. The explanations themselves are kept short, so there is more than enough time to put the new knowledge into practice. We think that these assignments are very important because programming is all about writing code yourself and solving your own problems and not about looking at how other people solved problems.

8.1.4 Q3: Python

The third quarter will consist of 3 parts: learning python, a project and domain B (algorithms). It consists of 24 lessons, all parts will be equally divided into 8 lessons. Students will start by translating C into Python, another, more modern and easy language. Python is similar to C, but comes with many features in advance, which saves a lot of time. The project will be done in groups to improve teamwork. It will be done in Python and will start with a brainstorm and planning the project. After that the teacher will help the students to make sure that the students will have made something working at the end. Throughout the project the students will not stop learning new things, during the project they will finish domain B: the domain about algorithms.

We chose Python, because it is more modern language than C, it is also easier to learn. Because C is time-consuming to use, students cannot create large projects with it, as that will take too much time. Switching to Python, a language with more features, will allow students to create larger and cooler projects. Also, Python is very popular these days, so it is a useful language to learn. And, that means that there are libraries for Python that will result in cool projects, like creating a voice-recognition program with a dozen lines of code.

We also chose to finish domain B in this quarter. Up until now, the students will have learned a lot about algorithms. They started in the first quarter with learning about computers, and they put many of these problem-solving skills to use in projects. There will be 8 lessons to prepare the students for the school exam about domain B.

8.1.5 Q4: Projects

The fourth quarter will consist of 24 lessons as well. 8 lessons will be about domain E, which is about software, hardware, and security. These lessons will focus mainly on software and hardware and how computers interact with each other through the internet, security will be more apparent the year after. The main part of this quarter will be a larger project than before. Students will again work in groups to improve their teamwork. The students will be completely free to create whatever they want as long as it has something to do with programming. Just like the last project, the first few lessons will be about brainstorming and planning. When the teacher approves the project, then the students may start programming. We will provide this project with many ideas and useful extra lessons such as an introduction to pygame (a Python library to create games).

As we have stated earlier, we think projects are the key to learning how to program. The students will face many problems and will learn to solve these problems themselves. Projects will also prepare them for real-world professional work and for university. If we were to provide only small assignments students would not learn to solve their own problems as they could just look up the answer. In the real world there might not be an answer, you would have to find the answer yourself or create a workaround to solve the problem. We have decided that these projects must be done in groups because we think teamwork is very important, because in a professional environment you constantly have to work with other developers, you have to understand their code and they have to understand yours. The final assignment will count as a school exam and will mainly be assessed on teamwork, design of their code and usability. Presenting the project to their classmates will also be a part of the mark, to practice summarising what you have done and presenting in general.

8.1.6 VWO5

The main focus of VWO5 will be building websites and finishing the other domains. Therefore, the students will start with finishing domain E which they started in last quarter of last year. Domain E is also about security, which ties in nicely with the next subject: websites and databases. The students will learn to create fully-fledged websites using HTML, CSS, and JavaScript. The students will use Django as a backend framework and learn about databases with SQL. Together with a lesson series about the rest of domain C will be enough to finish domain C with a school exam. After they know everything about security threats and usability, they can start creating their own website. This time they will create it themselves as it is going to be a portfolio website. The website must include all the projects they made throughout the course while also being an interesting website and applying all the things learned from domain C and E. The students will then have a lesson series on domain F: design, usability, security, and the effect on society. They will have school exam on this subject.

8.1.7 VWO6

In VWO4 and VWO5 the students will have finished domain B until F. In the first part of V6, they will work on the domains of choice. There are many optional domains and teachers will be able to choose them themselves. We will provide all the optional domains with lessons, assignments, and tests. After the optional domains are finished, the students will do another large project in groups or pairs. For this we will provide a few lessons about git and GitHub to improve teamwork. This project will be the last and largest projects of them all. Students will get complete freedom in what the project is going to be, so it can be a website, a video game or software. We will provide ideas for this project. The students will not only have to make something, but they will also have to explain their choices in a paper. Therefore, they will be assessed on what they created, their teamwork, security, usability, and code design. Their paper will be assessed separately, and they will also have to pitch their product/project to their classmates. The project, the paper, and the presentation will finish domain A.

The reason we have this large project at the end is because we want the students to be able to explain their choices and reflect on the process in the form of a paper to ready them for university. The presentation is important because presenting your products to businesses is

not only good skill, but also important for developers as they are often hired by businesses for programming projects.

8.2 The ideal

We started this project by writing down what our ideal website would look like and what features it would have, then scoring those features based on how hard we think it would be to create them. The scores go from 1-10 with 1 being very easy, 5 being doable and 10 being probably possible but not worth doing.

Titles at the top (1)

We wanted to have titles of lessons at the top of the page so students could always look up if they were in the correct lesson.

Short questions (3)

We wanted to have short questions throughout our lessons so students would also put their knowledge into practice to help them remember and learn. There would be multiple choice questions and open questions which you could check immediately and see the answer.

Practical questions (6-7)

We wanted to have more practical questions where you would write your own code and be able to execute it inside the website. Another thing we wanted with this is that you could auto grade it, meaning that the website itself would check if your code works. These larger practical questions are important, as most of these questions will combine multiple things the students have learned into one assignment.

Video discussion (4)

Each larger assignment will have a discussion video discussing the assignment by explaining the way of thinking. So, the steps you need to take to get to the solution. We chose not to show the answer, because students should still be able to try and write the assignment themselves. We will not have videos discussing small assignments because they can be explained with just text as well.

Example answers (2)

Students should also be able to find examples of solutions to the assignments. If the student is not able to do the assignment even after the video discussion, they can then look up an example answer.

Clickable keywords (1)

Keywords should be marked and should also be able to be clicked on which would find you the definition of the keyword in the documentation. This makes it easier and quicker for students to search the definitions of certain keywords.

Keyword list (1)

Students should be able to open the list of keywords used in a lesson so they can look them up more quickly and so they have an overview of all the keywords used.

Animations (3)

We wanted to create animations for a few harder subjects so students would understand them better. Some subjects like neural networks and sorting algorithms are better visualized using an animation.

Example code (3)

Example code is very important in our lessons because we use many examples to explain subjects. One of the main requirements for this code was that it should have syntax highlighting, meaning that certain pieces of code like built-in functions should get a certain colour. This way students get a better overview of the code, and it will also look similar to Repl and other IDE's.

Scores to teachers (8)

All the scores from the open and multiple-choice questions and also the coding assignments should be sent to the teacher environment. This is important so the teacher can review the assignments their students made so they can help them better. Having everything in one place is also very useful so the teacher will not have to walk around the class to collect homework, they can instead instantly see what their students have made.

From the interviews we learned that repetition is very important therefore we wanted to keep repeating certain important subjects by, for example creating exercises where subjects from previous chapters and paragraphs will be asked. We also found from our own experience that being direct and straight to the point can really aid the learning experience as no unnecessary things are discussed. This direct method of teaching can be found in CSCircles as well, where they explain how something works simply and directly. This does stand in contrast to CS50 where many subjects are explained using sometimes farfetched demonstrations, which in turn will excite students more. We needed to find a balance between being direct, while also keeping the attention of students.

Another thing we learned from the interviews is that students should practice a lot because that is the best way to remember certain things. Therefore, we want to include many exercises where students will apply their newly acquired knowledge. We want to do this in the form of projects where students will get a sort of 'field experience' of programming.

We want to start our chapters and paragraphs with the number zero instead of one. CS50 does this as well, so students remember that computers start counting at zero and not at one. Also, because it is a more artistic choice to represent computer science in the naming scheme. We really liked that reference.

8.3 The reality

The reality is often very different from the ideal because the world is simply not ideal. We wrote down many things at the start, a few of those things we did not expect to finish at all; some of those things we never even started, but a few did end up making it into the website.

8.3.1 The simple things

Many of the simple ideas like title on top made it into the website instantly as they were a matter of minutes to implement correctly. We also managed to create the clickable keywords and the keyword list, we first thought they would be very easy to implement, but in the end it was quite a bit harder. At this point both the list and the clickable feature works but is not in a finished state as things like styling and animations still have to be added. The questions were also supposed to be one of the easier things; and they were easy to make, however we still ended up redoing the questions three times as we were not entirely happy with how they worked and wanted to expand on a few features and future proof them.

We also managed to implement example code with syntax highlighting using a plugin called highlightJS. The implementing part of this was very easy but finding a plugin which did exactly what we wanted was quite a bit harder. The current state of the highlighting is not entirely finished as there is still some work to do on the styling, but it does work.

Les 0 - Programmeren in C

In hoofdstuk 0 hebben we gekeken naar werken en hoe we ze kunnen gebruiken problemen op te lossen. Ook is er geïnt programmeert, dit hebben we toen gedaan van blokjes en het invullen van wat waa werken met ons toetsenbord, dit doen w de programmeertaal C. Klik bijvoorbeeld i

```
char c = (char) x;

bool b = (bool) x;

float f = (float) x;

// etc.
```

Figure 71 Syntax highlighting on the website.

8.3.2 The harder things



The screenshot shows a Replit IDE window. At the top, there's a tab labeled 'main.c'. The code in the editor is:

```
1 int main(void)
2 {
3     printf("Hallo!");
4 }
```

Below the code editor, there's a 'Console' tab. The output shows a warning from the compiler:

```
main.c:3:5: warning: implicitly declaring library function 'printf' with type
'int (const char *, ...)' [-Wimplicit-function-declaration]
printf("Hallo!");
^
main.c:3:5:      include the header <stdio.h> or explicitly provide a
declaration for 'printf'
1 warning generated.
> ./main
Hallo!>
```

Figure 72 Embedded code execution on GroteProgrammeer using Replit.

When we made the list with ideas, we instantly knew that creating auto grading would become very hard to do; yet we still managed to implement it in a way through Replit and CS50's open-source auto grading. We are currently not entirely satisfied with the result because it has many limitations. The ideal was that we would make our own code execution and grading, we found out that it is possible to do it yourself, but it would take hundreds of hours, which we did not have. Therefore, we used what other people had already made before us, mashed it together and made something out of that, which worked.

8.3.3 The things that never made it

One of the things which we instantly decided that would not make it into the product for now, was the videos. These videos cost a lot of time to make and were not a priority. We will, however, definitely make these videos in the future because we think that including these videos could help students a lot.

Another thing which did not make it was the example answers on the website, we did make example answers to our exercises but at this point we have not yet implemented them into the website because we simply did not have the time.

The last thing is that the questions and exercise results should be sent to the teacher. We ended up deciding that having the questions that are placed throughout the lessons should not be sent

to the teacher. We decided this because those questions are of little importance and even less important to the teachers as they are mostly multiple-choice questions from which the teacher cannot understand why or how the student has answered in a certain way; the teacher can, however, ask the students their reasoning behind their answer during the lessons. Sending the results of the coding exercises to the teacher proved to be incredibly difficult as we did not make an auto grading system ourselves but used CS50's auto grading system instead. We also did not manage to create a teacher environment, so sending it to the teacher was already not doable with the time we had.

8.3.4 Things we learned from others

Rekenen

In de taal C hebben we 5 verschillende manieren om iets uit te rekenen:

- optellen: dit doe je met +

```
int x = 4 + 2;
```

- aftrekken: dit doe je met -

```
int c = a - b;
```

- vermenigvuldigen: dit doe je met * (SHIFT + 8)

```
int s = 4 * 3;
```

- delen: dit doe je met /

```
int m = k / 1;
```

Naast deze vier is er nog één: de modulus. Je weet misschien nog van de basisschool dat als het delen niet goed uitkwam je te maken had met een rest. $10 / 3 = 3$ rest 1, omdat $3 \times 3 + 1 = 10$. Soms is het handig om alleen die rest te berekenen, dit doe je met de modulus. $10 \% 3$ betekent dus deel 10 door 3, wat is dan de rest?

- modulus: dit doe je met %

```
int d = 19 \% 5;
```

Figure 73 An example of direct language usage while also explaining in a more interesting way.

Implementing the things, we learned from the interviews proved to be not so hard. We tried to make our language as direct as possible while also throwing in some interesting facts. *Figure 4* is an example of this, we used direct language like "In C we have 5 different ways to calculate" "addition: you do this with a +". But we also added a more interesting story about how the students have already learned to use the modulo function on the primary school.

In the lesson seen in *Figure 4* operators are discussed. Operators are also discussed in the previous chapter, the chapter about scratch. We purposefully re-explain operators because we think it is important that students remember them so they can use them better in their own code.

Lastly, we start all our chapters and paragraphs from zero, because we liked the reference to computer science it captures.

Chapter 9 – The education; the website; the product

Transferring an idea to reality is the most beautiful thing about design. The product you created takes a physical form rather than staying an image in one's mind or a sketch on a piece of paper. This project also started out as an idea with no shape or form, yet now it has started to take shape. One of the first things we wrote down was what our ideal website would look like; now we can reflect on those ideas as many have already made it into the website. This chapter shows those ideas and explains the choices we made regarding the physical aspect of our website and lessons.

9.1 Separating the website

For our educational website we found it important to separate the websites into multiple sections. That is because we think students should not be distracted by other functions of the website and, because we think that students should, obviously, not be able to reach all the teacher functions like creating new classes and assignments. The same applies to guests, as they should only be able to reach the front page of our website. This paragraph will describe these three sections and provide proof in either screenshots or concept art as at the time of writing the website has not yet been completed. The screenshots should also be taken with a grain of salt, many of the pages shown will still be a work-in-progress.

9.1.1 The student environment

One of the main ways in which we separate the student section from the guest section is by having an entirely different navigation bar. The navbar, for short, the yellow bar on the left, is oriented vertically instead of horizontally indicating a completely different section of the website. The navbar stays the same throughout the entire student page.



Figure 74 Screenshot of groteprogrammeer.nl's student section at the time of writing.

9.1.2 The teacher environment

At first glance the teacher section looks very much like the student section. That is because both sections imply a sort of 'premium' look because they are meant for customers of groteprogrammeer. But both are different; the teacher's navbar has different functions as teachers need to be able to reach students to, for example, check if a student handed in their homework.



Figure 75 The teacher's page. (concept)

The home page is also different for teachers. Do note that the background colour of the student section is a placeholder, in the future it will become the background colour used in *Figure 2*. Both teacher and student sections have rectangular white boxes in which content is displayed. But the content of both sections is different. Teachers can generate classes instantaneously with a code and students will be able to join these classes in the future. Teachers can also see each individual class and send messages to them or start a lesson with them. They are also able to create assignments and view those assignments once they are handed in.



Figure 76 A rectangular section of the teacher's section. (concept)

9.1.3 The guest environment

As mentioned earlier, the guest section is entirely different from the student and teacher sections. The section has a horizontal navbar instead of a vertical one. The guest page is meant to be for people that do not have an account and are just exploring the website. That is why we will inform the user on the front page about what the website is meant to be. The guest will also be able to visit a few different pages, about us or our thesis. They will also be able to seek contact with us. But the main function of this section is registering and logging in.



Figure 77 A screenshot of the guest section.

One might have noticed that the colours stay the same throughout all the sections. That is, to form cohesion. Cohesion is important because even though the user should notice that they went to a different section, they should still think that they are on the same website. This is also done by using the same font and shapes. We mostly use Sans-serif and Roboto for text and we only use rounded corners on all sections.

9.2 Documentation

A documentation is used to describe, explain, or give information about a subject. They often contain a list of keywords with explanations and follow a tree-like structure which starts with a single keyword and then expands into multiple.

9.2.1 Why use a documentation

We chose to create a documentation on our website for our users because of multiple reasons. First, we found from experience that having a list of keywords can really help with learning. Oftentimes, books from school have an appendix with a list of words but with no explanation. Another reason is because learning to use a documentation can be very helpful in the future with programming. Many coding languages and programs have a documentation, learning to use these documentations can therefore be very helpful. The tree-like structure also helps with linking keywords to each other, which cannot be done in a book.

9.2.2 How we made the documentation

The documentation can be found in two places: on its own page and next to a lesson. We wanted to make the documentation in such a way that it could be instantly accessed by the students without opening a new window. So, we created a slider system where the documentation would slide out with the press of a button. Clicking on keywords will also open the documentation of that keyword as we will discuss in the next paragraph.



Figure 78 Lesson 0 with documentation on the right (work-in-progress).

At the time of writing the documentation has not been sorted out yet and there is also no animation to slide it in. But the definitions of the keywords can be loaded in from the database. There is also an *open in page* button which opens the documentation page.

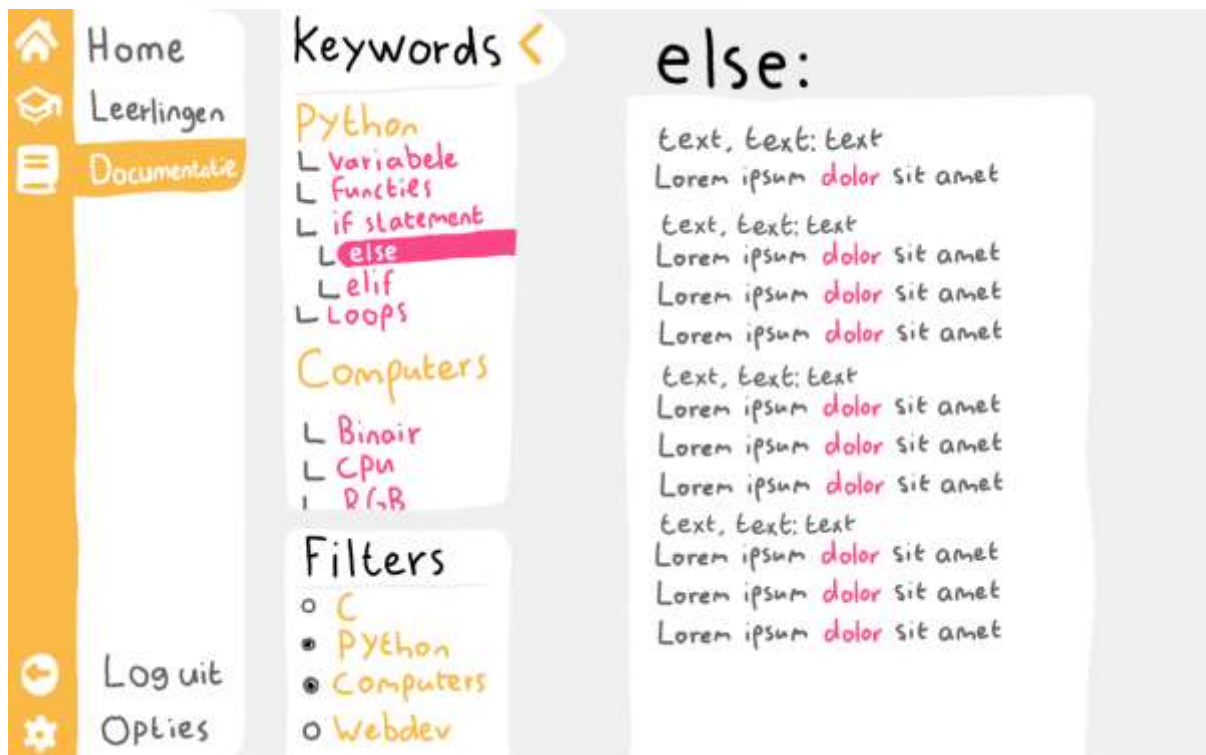


Figure 79 The documentation page. (concept)

The documentation page does not exist at the time of writing so we created concept art for it, which can be seen in Figure 6. The page contains a list of keywords which can be selected. From here one can visit all the keywords which have been added. There will also be a sorting system which will sort the keywords, so the list does not get too long. Some keywords will open more keywords when clicked, that is because of the tree-like structure of the documentation. An example: we have two keywords: fruits and vegetables. If one clicks on fruits, it gives a definition of fruit, but it will also show a list of fruits under it. With indentation, it can be indicated that there is a link between parent keyword and its children. In Figure 6 one can see that *else* and *elif* are related to *if statement*. The definition of the keyword can also link to other keywords in the documentation.

9.3 Non distracting, yet play and colourful

One of our main goals with the overall look of the website was that it should be catchy, but also not distracting. So, we chose very vibrant and catchy colours but made many of the website's features hidden until the user wanted to use them.

9.3.1 The low-profile navbar

The low-profile navbar is one of those 'hidden until needed' features. The navbar resides at the side of the website instead of the usual top. That is, because a navbar on the side takes up less space as most computer screens are shorter than they are wide. When the user hovers on the navbar with their mouse, it folds out revealing several buttons to go to other pages.



Figure 80 Vertical navbar.



Figure 81 Vertical navbar (folded out).

9.3.2 Clean and sterile

The colour which has been used most on the website is white, or shades of it. Most of the pages have a white background colour with black or slightly grey text. This creates a very bright and sterile environment; the website looks very clean. The use of vibrant colours also helps making this effect more evident. We think that a clean environment will help students concentrate as they will not be distracted by too many colours.

9.3.3 Keywords

As discussed in the previous paragraph about the documentation, we have marked important keywords. We do this, so students can instantly recognize important words so they can link it to a lesson or to another keyword. The keywords are made bold and pink. Only making the words bold was not enough to make them stand out, so they are also pink. The keywords can be clicked on and will open the documentation with the definition.



Figure 82 Screenshot of an early test version of the keyword system.

9.3.4 Code

A computer science website needs a way to display code differently from normal text as it would otherwise get confusing quickly. So, we decided to mark pieces of code which are referenced to inside the text. These pieces of code we mark inversely to the keywords. They are marked in pink, and the text is white. The font we use inside these code pieces is also different. We use a monospace font because displaying code in mono is useful as things like indents or large numbers are more easily recognized. And because it creates cohesion with the rest of the code. Clicking on the pieces of code copies it to your clipboard, which can be useful if you do not want to select it yourself.



Figure 83 Marked code at the time of writing.

9.3.5 The questions

For the lessons we wanted the questions to stand out most as they are very important. We think asking students questions throughout and after the lesson is important for helping them understand the subject better. Question cards have the primary colour applied to them with white text. Questions have three forms: regular questions with no checking, multiple choice questions which can be checked and explain why the correct answer is correct and there are open questions in which you can type and check your answer just like the multiple-choice questions.

Opdracht 1: Druk op de groene pijl bovenin, als het goed is zie je onderin het donkerblauwe venster "Hallo!" verschijnen.

Figure 84 Regular question card. Translation: Press the green arrow atop, if all is well, you will see "Hello!" in the dark blue windows below.

Welke term drukt het kopiëren van een stuk code uit zodat jij het voor jezelf kan aanpassen?

- ☐ Copy
- ☐ Fork
- ☐ Insert
- ☐ Repl

Check

Figure 85 Multiple choice question card. Translation: Which keyword expresses copying code so you can change it for yourself?

Hoe heet de vertaler die van source code machine code maakt?

Antwoord

Check

Figure 86 Open question card. Translation: What is the name of the program that translates source code into machine code?

Examples of these question cards can be seen at Figure 84, 85 and 86. Once a question card is checked you will either have answered correctly or incorrectly. When answered correctly, the

card will turn green, and the correct answer will be displayed in bold. The explanation of the correct answer will also be displayed.



Figure 87 multiple choice question card answered correctly. Translation: Forking means copying our Repl so you can change it.

When answered incorrectly, the card will turn red and prompt you with a choice: you can either try again or display the correct answer. You can only use the 'display answer' button when you have tried at least once. We think it is important that students try at least once, even if the student just checks a random checkbox. On a test no answer is always 0 points while a guess will give you a small chance of getting a point. If you do hit the 'display answer' button, the card will turn a more neutral, yellow colour and, of course, display the correct answer with its explanation. The same applies to the open question card.



Welke term drukt het kopiëren van een stuk code uit zodat jij het voor jezelf kan aanpassen?

- ☐ Copy
- ☐ Fork
- ☒ Insert
- ☐ Repl

Figure 88 Multiple choice question answered incorrectly.



Welke term drukt het kopiëren van een stuk code uit zodat jij het voor jezelf kan aanpassen?

- ☐ Copy
- ☒ **Fork**
- ☐ Insert
- ☐ Repl

Fork betekent dat jij de Repl die wij voor je hebben gemaakt kopieert, zodat je hem zelf kan aanpassen.

Figure 89 Multiple choice question after answer has been shown.

Conclusion

In our introduction, we established the following goal:

"Creating an online learning platform to assist students in learning and teachers in instructing computer science in pre-university education."

This goal has been (mostly) achieved, but how far did we manage to finish the website?

Let us start with the things that we did manage to create or design. We have created a website on which users can create an account and log in. If you are not logged in, you can view the guest environment, with details about ourselves, this thesis and more. When you log in, you can enrol in a course (chapter) and see all the lessons that belong to that chapter. There is a page where you can view these lessons with highlighted code, clickable words and interactive coding environments using Repl. The first two chapters have been finished, each containing 16 lessons. Each lesson in Chapter 1 has been equipped with one or more exercises to process everything the students have learned this lesson. They can also use an automated checker to find out how well they wrote their code (i.e., how well it works). If a student has forgotten the meaning of a certain word in computer science, they can click that word in the lesson and the documentation window will slide in from the side, wherein students can view the definition of the clicked word, as well as links to the rest of the words. Although it works, it has not been styled yet, so it may hurt your eyes a little.

We have not, however, finished everything. Not all of the lessons we created are of the same quality and some lack exercises and questions. The website, currently, does not support teachers' accounts and relies on a third-party service to check student's progress, something we would like to host internally in the future. Also, only 2 out of the 11 chapters have been finished. 11 being the bare minimum, because we would probably do a chapter on every domain of choice, so teacher (or students) can choose themselves. That brings up the total to 20 chapters.

We can conclude that the goal has been achieved, as students can now start to learn computer science. However, it has not been completely finished, as there is still a lot of work to do.

Corresponding questions

When we established our goal, we also added some corresponding, guiding questions that needed answering before we could create a website.

"What are effective ways to instruct unexperienced students; are there any computer science-specific ways?"

To ensure that all students understand what you are teaching, you must repeat – and keep repeating – what you are saying, so students will never forget it: repetition is key. Also, you must not use incredibly long stories and explanations to explain something, be as concise as possible, without sacrificing any important details. And lastly, use as many of the students' senses in the explanation, so make sure you tell them something, they read, they watch, they see, they touch, etc.

“How does one make a website accessible and understandable for all?”

For creating a website that users will understand, we used Google's Material Design, which is a system that helps you create a simple, obvious, and accessible while also good-looking website. It also helps with creating a consistent look and feel of the website, so students can use it intuitively.

“How to create a robust server that handles users' input?”

To create a back end of a website (a server), we used Django, a Python framework that allows you to make views to render pages, models to store data and has a lot of features out of the box. For deploying our website from the localhost (your own machine) to the internet, we used Heroku to host our website (it even has a free tier!).

“How can one make students excited for the subject computer science?”

Students will be distracted if you are talking too long, without any purpose. Be concise in your explanations and use as many examples as you can. Also, take a look at CS50 (a Harvard course on computer science) to be inspired by the diversity of demonstrations and analogies about the computer's mechanism.

Discussion

As we have touched on many times now, not everything has been finished for this thesis. This is not a surprise, in the first meeting with Peter Ypma, he asked us to decrease the scope of the thesis, because it would be too much if we were to elaborate all of our ideas. So, we did not expect to finish everything, and – evidently – we did not. This is a list of all the things that have not yet been implemented onto the website or the lessons

- The documentation window needs styling.
- Teachers need to be able to have a different account than students, with which they can create classes and view progress of those students, send messages, create custom chapters and custom lessons.
- We want to leave the CS50 auto-grading framework and create our own that is embedded into the website.
- The website needs to be responsive to viewport size (whether you are on your phone, tablet, computer, or television), so that the website looks equally beautiful on all devices.
- We need to add between nine and eighteen chapters and their lessons on numerous subjects.
- All lessons need to have videos wherein we summarise the matter.
- All exercises need to have videos wherein we look at how to solve that exercise.
- Some lessons need animations to make some ideas come to life (like inserting into a linked list).
- There should be documentation on the usage of the website, both for students as well as for teachers (and administrators).

This website was the very first (major) project in web development we have done. The development of the website went overall unexpectedly well: we ought that it would take dozens of hours to create something presentable. In reality, it took a lot of time, but not as much as we had expected it to. Working together went very well, while we were thinking of what needed to do, we quickly got to a satisfying division of the tasks. Something that was tricky in the beginning, however, was using Git and Github. This is a program and website respectively, with which we can store our code online and combine our versions into one website. This is a difficult tool to use as a beginner, so databases have been wiped and files have been lost. Luckily, we were able to restore (most of) those files.

In our naïve dreams at the start of this thesis, we thought that writing lessons was no more work than writing down what you know. This (as Peter Ypma warned us) took a considerable larger amount of time. A lot of planning and rewriting was involved to get the lessons good enough. We did enjoy making the lessons, so it was not a problem having to spend more time writing the lessons.

Another thing we did not get to do in this thesis, simply because there was too little time, was checking how well our idea works. So, we could do that in another thesis with the research question:

“Does GroteProgrameer work in practice?”

All the negativity aside, we have had a blast making this thesis. Although not all the processes were necessarily our favourite (think of creating the list of references), we are very proud of this thesis. We are also proud to be able to say that we have made our own website that is now globally accessible via www.groteprogrameer.nl. Do not hesitate to take a look.

We hope you have enjoyed reading our thesis as much as we have writing it!

Sincerely,
Christiaan van der Haven
Vincent Ruijgrok

September 29th, 2021
Gouda

References

- 9 Harvard computer science classes you can take online for free — including an intro course that's already enrolled 2 million people. (2020, July 10). Business Insider Nederland. <https://www.businessinsider.nl/harvard-cs50-online-computer-science-classes?international=true&r=US>
- Bradford, L. B. (2019, November 15). *What to Know About Front-End Framework in Web Development?* The Balance Careers. <https://www.thebalancecareers.com/what-is-a-front-end-framework-and-why-use-one-2071948>
- C. (2018, May 19). *CS50's Changing Demographics - CS50*. Medium. <https://cs50.medium.com/cs50s-changing-demographics-d00fb7369d6>.
- C Language Source Code to Exe | Build Process | Compilation, PreProcessor (Theory)*. (2017, December 11). YouTube. <https://www.youtube.com/watch?v=gSackZtqlUI>
- Cromwell, V. (2017, June 2). *Evan You*. Between the Wires. <https://web.archive.org/web/20170603052649/https://betweenthewires.org/2016/11/03/evan-you/>
- CS50. (n.d.). CS50. Retrieved 21 September 2021, from <https://cs50.harvard.edu/college/2021/fall/>
- Django at a glance | Django documentation | Django*. (n.d.). Django. Retrieved 8 July 2021, from <https://docs.djangoproject.com/en/3.2/intro/overview/>
- edX. (n.d.). *CS50's Introduction to Computer Science*. Retrieved 21 September 2021, from <https://www.edx.org/course/introduction-computer-science-harvardx-cs50x>
- Examenprogramma informatica havo/vwo*. (2019). Examenblad. https://www.examenblad.nl/examenstof/informatica-havo-en-vwo-3/2022/vwo/f=/examenprogramma_Informatica_havo-vwo.pdf

Foreword — Flask Documentation (2.0.x). (n.d.). FlaskProject. Retrieved 8 July 2021, from

<https://flask.palletsprojects.com/en/2.0.x/foreword/>

Google's Material. (N.d.-a). *Surfaces*. MaterialDesign. Retrieved 8 July 2021, from

<https://material.io/design/environment/surfaces.html#material-environment>

Google's Material. (N.d.-b). *The type of system*. Material. Retrieved 8 July 2021, from

<https://material.io/design/typography/the-type-system.html#type-scale>

Google's Material. (N.d.-c). *Understanding layout*. Material. Retrieved 8 July 2021, from

<https://material.io/design/layout/understanding-layout.html#principles>

Google's Material. (N.d.-d). *Understanding motion*. Material. Retrieved 8 July 2021, from

<https://material.io/design/motion/understanding-motion.html#principles>

Google's Material. (N.d. -e). *Understanding navigation*. Material. Retrieved 8 July 2021,

from <https://material.io/design/navigation/understanding-navigation.html#lateral-navigation>

Higher level and lower-level languages - Computer Science Wiki. (n.d.).

ComputerScienceWiki. Retrieved 8 July 2021, from

https://computersciencewiki.org/index.php/Higher_level_and_lower_level_languages

How to Apply a Color Palette to Your Design – Tutorial. (2020, May 26). YouTube.

<https://www.youtube.com/watch?v=eXcKOqviLEo>

JS Foundation - js.foundation. (n.d.). *jQuery*. JQuery. Retrieved 9 July 2021, from

<https://jquery.com/>

JSON developers. (n.d.). *JSON*. Json. Retrieved 9 July 2021, from

<https://www.json.org/json-en.html>

Kuhlman, D. (n.d.). *A Python Book: Beginning Python, Advanced Python, and Python*

Exercises. WebArchive. Retrieved 8 July 2021, from

https://web.archive.org/web/20120623165941/http://cutter.rexx.com/%7Edkuhlman/python_book_01.html#important-features-of-python

Lithmee, B. (2018, June 27). *Difference Between Compiler Interpreter and Assembler*.

Pediaa.Com. <https://pediaa.com/difference-between-compiler-interpreter-and-assembler/#:%7E:text=The%20difference%20between%20compiler%20interpreter,languange%20programs%20to%20machine%20language>

Material. (n.d.). *Material web components*. Retrieved 9 July 2021, from

<https://material.io/components?platform=web>

Otto, M. (2021, May 5). *Bootstrap 5*. Bootstrap Blog.

<https://blog.getbootstrap.com/2021/05/05/bootstrap-5/#more-component-updates>

Pastorino, M. (2020, August 13). *Frontend vs Backend: What's The Difference?* Pluralsight.

<https://www.pluralsight.com/blog/software-development/front-end-vs-back-end>

Petlovana, Y. (2021, June 19). *Top 13 Python Web Frameworks to Learn in 2020*. Steelkiwi.

<https://steelkiwi.com/blog/top-10-python-web-frameworks-to-learn/>

Quickstart — Flask Documentation (2.0.x). (n.d.). FlaskProject. Retrieved 8 July 2021, from

<https://flask.palletsprojects.com/en/2.0.x/quickstart/>

Semantic. (n.d.). *Semantic UI*. Semantic UI. Retrieved 9 July 2021, from <https://semantic-ui.com/>

Shorelight team. (2021, August 30). *What Is a Syllabus and Why Is It Important?*

Shorelight. <https://shorelight.com/student-stories/what-is-a-syllabus-and-why-is-it-important/>

Stack Overflow Developer Survey 2016 Results. (n.d.). Stack Overflow. Retrieved 8 July

2021, from <https://insights.stackoverflow.com/survey/2016#developer-profile-education>

- trio.dev. (n.d.). *JavaScript Frameworks: What Are They and How Do They Work?* Trio Developers. Retrieved 9 July 2021, from <https://trio.dev/blog/javascript-framework>
- Using This Website | Computer Science Circles.* (n.d.). CSCircles. Retrieved 21 September 2021, from <https://cscircles.cemc.uwaterloo.ca/using-this-website/>
- w3schools. (n.d.-a). *CSS Tutorial.* Retrieved 9 July 2021, from <https://www.w3schools.com/css/default.asp>
- w3schools. (n.d.-b). *HTML Tutorial.* Retrieved 9 July 2021, from <https://www.w3schools.com/html/default.asp>
- w3schools. (n.d.-c). *JavaScript Tutorial.* Retrieved 9 July 2021, from <https://www.w3schools.com/js/default.asp>
- w3schools. (n.d.-d). *What is JSON.* Retrieved 9 July 2021, from https://www.w3schools.com/whatis/whatis_json.asp
- The Web framework for perfectionists with deadlines | Django.* (n.d.). Django. Retrieved 8 July 2021, from <https://www.djangoproject.com/>
- What is a CDN | Cloudflare.* (n.d.). Cloudflare. Retrieved 9 July 2021, from <https://www.cloudflare.com/nl-nl/learning/cdn/what-is-a-cdn/>
- What is the history of the Django web framework? Why has it been described as ‘developed in a newsroom’?* - Quora. (n.d.). Quora. Retrieved 8 July 2021, from <https://www.quora.com/What-is-the-history-of-the-Django-web-framework-Why-has-it-been-described-as-developed-in-a-newsroom>
- Why does CS50 use C?* (n.d.). Quora. Retrieved 8 October 2021, from <https://www.quora.com/Why-does-CS50-at-Harvard-use-C-as-its-primary-language>
- Wikipedia contributors. (2021a, June 18). *Internet.* Wikipedia. <https://en.wikipedia.org/wiki/Internet>

Wikipedia contributors. (2021b, June 29). *HTML*. Wikipedia.

https://en.wikipedia.org/wiki/HTML#HTML_versions_timeline

Wikipedia contributors. (2021c, July 7). *CSS*. Wikipedia. <https://en.wikipedia.org/wiki/CSS>

Images used

Amazon.com: CS50: Apps & Games. (n.d.). Amazon. Retrieved 21 September 2021, from

<https://www.amazon.com/CS50/dp/B01A7U8VM6>

Best educational websites. (2017, September). [Illustration].

<https://i1.wp.com/www.edutechpost.com/wp-content/uploads/2017/09/best-educational-websites.png?resize=768%2C768&ssl=1>

CS50 2018 - Lecture 1 - C. (2018). [Picture].

<https://i.ytimg.com/vi/wEdvGqxafq8/maxresdefault.jpg>

Getal en Ruimte VWOB Deel 1. (n.d.). [Picture].

https://www.bookmatch.nl/omslag/9789001842321-Getal-en-ruimte-11e-ed-wiskunde-b-vwo-deel-1_large_nw.jpg

Heroku image. (n.d.). [Logo].

https://miro.medium.com/max/1400/1*kg8aAUNxIo55OhNhTo4srQ.png

Material. (n.d.-a). *Colour hierarchy* [Image]. Applying Colour to UI.

<https://material.io/design/color/applying-color-to-ui.html#typography-and-iconography>

Material. (n.d.-b). *Colour template* [Image]. The Colour System.

<https://material.io/design/color/the-color-system.html#color-theme-creation>

Material. (n.d.-c). *columns, gutters, and margins* [Image]. Columns, Gutters and Margins.

<https://material.io/design/layout/responsive-layout-grid.html#columns-gutters-and-margins>

Material. (n.d.-d). *Forward navigation* [Image]. Forward Navigation.

<https://material.io/design/navigation/understanding-navigation.html#types-of-navigation>

Material. (n.d.-e). *Lateral navigation* [Image]. Lateral Navigation.

<https://material.io/design/navigation/understanding-navigation.html#types-of-navigation>

Material. (n.d.-f). *Material elevation* [Image]. Material Elevation.

<https://material.io/design/environment/elevation.html#elevation-in-material-design>

Material. (n.d.-g). *Reverse navigation* [Image]. Navigation.

<https://material.io/design/navigation/understanding-navigation.html#types-of-navigation>

Material. (n.d.-h). *Text on colour* [Image]. The Colour System.

<https://material.io/design/color/the-color-system.html#color-theme-creation>

Material. (n.d.-i). *Typography* [Image]. Typography.

<https://material.io/design/typography/the-type-system.html#type-scale>

Material. (n.d.-j). *Typography* [Image]. Typography.

<https://material.io/design/typography/understanding-typography.html#type-properties>

Material. (n.d.-k). *Typography* [Image]. Typography.

<https://material.io/design/typography/understanding-typography.html#type-classification>

Study computer science. (2017, January). [Picture]. <https://miuc.org/wp-content/uploads/2017/01/Study-Computer-Science.jpg>

The Futur Academy. (2020, May 26). *How to apply a colour palette* [Image from video].
How to Apply Colour to a Colour Palette.

<https://www.youtube.com/watch?v=eXcKOqviLEo>

Annotation to images used

Note that all images that do not have an in-text reference in their caption are screenshots made by us. They are our own and are therefore not mentioned in the Images Used section.

Appendix A – GitHub and website

If you want to take a look at the code we used for creating GroteProgrammeer.nl, it is visible until 31st of December 2021 via this link: <https://github.com/126511/groteprogrammeer>.

The result of the website can be found at www.groteprogrammeer.nl. This is not necessarily how we left it at the end of this paper, as we will continue to change how it works and add onto it in the future.

Appendix B – Logbook

In our logbooks, we kept track of the time we put in the thesis.

Vincent's logbook

Date (2021)	Summary of activity	Time (in min)
January and February	Completed the cs50 webdev course	1800
from 21-2 to 27-2	Creating a basic webserver, with functionalities as logging in and out. Creating register forms and a user's personal homepage	420
from 28-2 to 6-3	Creating the courses functionality of our website: we can create courses that have a unique chapterpath, with a list of all paths belonging to it in the database. Students can enrol into courses and see its corresponding material.	360
from 7-3 to 1-4	During our PWS-dag we realised we wanted the exercises to be auto graded, I researched this and read a lot of documentation of online IDEs and other options, ultimately settling with submit.cs50.io	300
on 7-3	PWS-dag: we worked on our thesis the entire day and got started with our research. We came up with the research questions and thought of how we would answer those. Conversation with Peter Ypma was also included	600
from 1-4 to 3-4	Writing up everything for the first phase: our literature study and what we had done thus far.	180
from 4-4 to 10-4	Creating the framework for the documentation. Using a custom API to import the HTML using JS's fetch option.	360
from 4-4 to 10-4	Creating the framework for the questions on the file pages. This mainly serves as an example for Christiaan what I meant.	240
from 11-4 to 24-4	I used this time to research how others teach computer science using C. Mostly I watched some lectures and read some online courses.	300

from 2-5 to 22-5	Creating the first 4 lessons of Chapter 1. I made Lesson 0: introduction, Lesson 1: Printing & variables, Lesson 2: Hello, student and Lesson 4: Calculations	900
on 1-6	Conversation with Peter Ypma on what to do for the second phase. What needed to be done and what could be postponed?	60
from 2-6 to 8-6	Writing down details on the experience of creating the website's structure. Writing down details of what's out there in terms of back-end frameworks and teaching computer science. Reading some papers on teaching cs.	360
on 8-6	Interviewing David Heerkens, a physics teacher on how he approaches teaching and how he'd approach making a course. Time also includes me typing down everything that had been said during the recording	180
on 10-6	Interviewing Lars van Bokhorst, a chemistry teacher on how he approaches teaching and how he'd approach making a course.	60
on 11-6	Interviewing Matthijs Alderliesten, a physics teacher on how he approaches teaching and how he'd approach making a course	30
on 11-6	Interviewing Kees van Vliet, a maths teacher on how he approaches teaching and how he'd approach making a course.	60
on 9-6	Working on writing the chapter on Building a website using Python and Django for the concept paper in July	240
on 14-6	Discussed interviews with christiaan	60
from 13-6 to 22-06	Transcribing the interviews of David Heerkens and Lars van Bokhorst	300
on 2-7	Discussing current progress with Ypma	120
from 6-7 to 9-7	Finishing the current chapters and writing chapter 5. Creating the final form of phase 2	540
from 25-7 to 7-8	Redesigning structure for lessons of chapter 1, writing lessons 0 through 7	1200
on 8-8	Redesigning back end partly because it was bad	240
on 9-8	Reimplementing file page showing and making; HTML-ified lessons 0-3	240

from 1-9 to 7-9	Creating lessons 8-11	300
on 2-9	PWS-dag: talking with Ypma on the progress and working with Christiaan on a timetable for the homestretch	240
on 3-9	Deploying the website to www.groteprogrammeer.nl	180
from 4-9 to 9-9	Processing feedback on chapters 3 and 5	180
from 10-9 to 18-9	Starting on chapter 7	180
on 18-9	Transcribing the interview of Alderliesten	180
on 19-9	Continuing ch6	60
from 20-9 to 24-9	Finishing chapter 6	180
from 25-9 to 28-9	Writing chapter 7 and finishing transcribing interviews with van Vliet and Bokhorst	300
on 29-9	Finishing everything, doing sources, checking spelling and making sure everything looks good.	240
Total time		11190
233.125% (total 4800 minutes)		
186 hours and 30 minutes		

Christiaan's logbook

Date (2021)	Summary of activity	Time (in min)
January and February	Completed the cs50 webdev course	1800
from 21-2 to 27-2	Learning and reading about material design	360
from 28-2 to 6-3	Learning about Vue and Vuetify. Watching multiple tutorials. Starting to work on the homepage design in Vue and Vuetify using my new knowledge on material design	900
on 7-3	PWS-dag: we brainstormed about research questions and discussed what the goal of our project would be. We also talked with Peter Ypma, our supervisor for the project.	300
from 7-3 to 21-3	Making very slow progress on the front page of the website because of Vue. Still watching tutorials on Vue.	420
from 28-3 to 3-4	Working on phase 1 and completing it. Phase 1 includes the research goal and questions associated with it. It also includes how we want to achieve that goal.	180
from 4-4 to 10-4	Finally making the switch from Vue back to normal Javascript. Had to delete the entire front page and start over. Way more progress than with Vue.	240
from 11-4 to 17-4	Making even more progress on the front page, finishing the complete layout of it.	240
from 18-4 to 24-4	Designing the userlayout. Which includes making a user-page, a sidebar and more.	240
from 9-5 to 15-5	Working on Vincent's questions, redesigning it using JQuery. Questions worked, but did not have all intended features, needed a redesign.	180
from 16-5 to 22-5	Redesigned the javascript questions so they can now be generated through a JSON API. They have all intended features and can be expanded upon later	180

from 23-5 to 29-5	Completed the design and layout of the lessons. Keywords are now automatically assigned a button which will lead to the correct documentation page. This is all generated to improve the lesson-writing workflow.	300
on 1-6	Talked with Peter Ypma about our progress on the website, also discussed what is needed to complete phase 2.	60
from 1-6 to 5-6	Started writing the material design theory and planned the front end theory. Also, wrote down the interview questions	240
on 8-6	Interview with David Heerkens on teaching physics, a subject closely related to computer science.	60
on 10-6	Interview with Lars van Bokhorst on teaching chemistry, another subject we found closely related to computer science. Also discussed the details of phase 2 with Vincent.	120
on 11-6	Interview with Mathijs Alderliesten, a physics teacher. And Kees van Vliet, a maths teacher. Also got the logbook up to date for myself.	180
on 12-6	Continued writing material design theory.	210
on 14-6	Discussed the interviews with Vincent	60
on 17-6	Basically finished chapter 2, started working on chapter 1	180
on 19-6	Continued working on chapter 1, finished it until 1.5.1	180
on 20-6	Finished Chapter 1 well enough for phase 2.	330
on 2-7	discussing the progress of phase 2 with Ypma	120
from 2-7 to 9-7	Redid Chapter 1 and made finished chapter 5	420
from 25-7 to 2-9	Created lessons 0.0 - 0.4	600
on 2-9	PWS-dag: talked with Ypma about our progress	240
from 2-9 to 11-9	Small edits to lessons 0.0 - 0.4 and finished first 2 scratch lessons	300
on 12-9	Created two more scratch lessons	210
on 16-9	Fixed chapter 5 using Ypma's notes	60
on 17-9	Started working on chapter 9. Also creating concept art for teacher page.	180
on 18-9	Worked on chapter 9	90
on 19-9	Finished Chapter 9 and worked on the front of the PWS	300

from 20-9 to 24-9	redid the front of the PWS and worked on the introduction/prologue	180
on 26-9	worked on chapter 8 and discussed progress with Vincent	300
on 27-9	Finished chapter 8 and discussed the remaining things to do.	240
on 29-9	Finished the thesis.	300
	Total time	10500
218.75% (total 4800 min)		
175 hours		

Appendix C – Interviews

These are the transcriptions of the interviews we conducted with David Heerkens, Lars van Bokhorst, Matthijs Alderliesten and Kees van Vliet. Other teachers that were named in the interviews are either removed (i.e., “other teachers”) or just their first letter, to secure anonymity.

V: Vincent Ruijgrok
C: Christiaan van der Haven
H: David Heerkens
B: Lars van Bokhorst
A: Matthijs Alderliesten
K: Kees van Vliet

The interviews were held in Dutch, and so their transcriptions are also in Dutch. Copy-paste them into DeepL (<https://www.deepl.com/nl/translator>) if you want to translate them to your favourite language.

Interview with David Heerkens

V: Als eerste hadden we de vraag: op welke manier geeft u structuur aan uw les?

H: Dat ligt een beetje aan je vak, mijn vak is heel erg georiënteerd op: we doen een stukje uitleg, we laten wat zien en je maakt sommetjes. Dat klinkt flauw, maar dat is de manier waarop zelfs professoren natuurkunde leren, door dingen uit te pluizen. En door het uitpluizen vind je zelf de verbanden tussen het een en het ander. Dat is bij mijn vakgebied de methode, dus wat ik meestal doe. Ik pak het boek en kijk waar we zijn gebleven; ik bedenk welk stuk van het curriculum ik vandaag uitleg en daar verzin ik ter plekke een voorbeeld bij of ik pak iets van mijn materiaal. En uiteindelijk doe ik daar actieve verwerking bij. Ik had net bijvoorbeeld een oefenstencil en daar hang ik dan vervolgens alles aan op. Met HAVO4 en VWO4 zijn we nu bezig met elektrische schakelingen, dus jongens we hebben die wetten behandeld: hier heb je een puzzel, ga maar puzzelen. En dan zie je dat de leerlingen die het moeilijk vinden daar vragen over gaan stellen en de leerlingen die het kunnen gaan heel hard. Mijn doel is dat leerlingen heel veel oefenen.

V: Dus vooral heel veel oefenen?

H: Ja, alleen het moet wel stapsgewijs. Je kan niet zeggen: oké jongens dit is de stof, ga maar puzzelen. Dat kan je een keer doen, alleen dan haken heel veel leerlingen af. Dus, je hebt er ook methodes voor, zoals het **directe instructiemodel**. Moet je maar eens op Googelen. Dan zie je dat je eerst voorkennis gaat activeren, dan nieuwe informatie aanbieden. Waarom is die voorkennis zo belangrijk? Als je net van Economie komt zit je na te denken over Euro's en daarmee kan je iedereen hoofd in de natuurkunde stand zetten. Bij natuurkunde moet je het opeens hebben over Volts of Joules. Je moet in je brein de juiste lade opentrekken: daarin zit een soort kapstok waaraan je nieuwe informatie kan hangen. Dat doe je door een demonstratie of een voorbeeld, door iets te laten zien. Je kan ook laten zien waarom iets niet werkt, en dat is bij programmeren misschien wel heel belangrijk.

C: Ja, dat is geen slechte, nee.

H: Dat je laat zien: als je het zo doet gaat het goed, want. Maar als je het zo doet kan het snel fout gaan. Daarna doe je iets samen met leerlingen: we doen samen opgave 34 en daarna laat je ze los. Aan het einde van de les herhaal je de leerdoelen. Doe je controlevragen, formatieve toets of rondkijken in de schriften.

V: Dus het eigenlijk een beetje oppervlakkige uitleg en dan door oefenen of demonstreren ga je de diepte in waardoor het praktisch wordt.

H: Ja voor mijn vakgebied wel. Dat is een beetje wat ik doe. Jullie hebben allebei les gehad van mij. Soms kies ik ervoor om leerlingen te laten zwemmen. De route van A naar B is niet altijd standaard, je moet vaak even nadenken hoe ik mijn bootje ga varen. Want uiteindelijk, hoe verder je komt, dichterbij je eindexamen: hoe complexer worden de vragen, hoe meer inzicht. Dus dat is een beetje de globale lesopzet.

V: Dan even de methode, we gebruiken bij natuurkunde Systematische Natuurkunde: wat vindt u heel erg fijn aan SN?

H: Wat ik heel fijn vind, is dat voor leerlingen, het is een goed boek, het sluit aan op het eindexamen. Bij het maken van schoolexamen pak ik oude examens en zie ik dat een vraag daaruit ook in het boek staat. Dus het is op examenniveau. Sommige vragen vind ik vergezocht. Ik vind het wel heel fijn dat de uitleg en voorbeelden in het boek heel erg "to the point" zijn: dus geen hele verhalen, nee gewoon dit. Ze zijn ook heel helder: welke formules moet je kennen. Je ziet de formules in 1 oogopslag. De digitale methode is ook heel fijn, voor leerlingen zijn alle boeken online beschikbaar. Dus als je VWO6 zit, hebben ze alle keuzekaternen en boeken beschikbaar gesteld, zodat als je in je eindexamen zit je dingen kan terugzoeken. En dat is uiteindelijk voor leerlingen een belangrijke factor. Je moet ze stimuleren om hun spullen bij zich te hebben, maar als er een keer iets misgaat, we zijn allemaal mensen (ik vergeet ook weleens wat, hoe drukker je het hebt hoe meer je vergeet), dan kan je alsnog aan het werk. Je schoolse carrière is veel nadenken maar ook veel doen. Dus dat is wat ik heel prettig vind. Ehm, sommige keuzes van het boek ben ik het niet mee eens. In de vorm van hoe bepaalde dingen worden aangeboden. Sommige dingen doe ik in een andere volgorde, maar je moet als methodeschrijver keuzes maken, en er zijn tachtigduizend meningen over hoe het zou moeten, net als dat we zeventien miljoen thuiscoaches hebben als Nederland speelt. Zij maken daar een keuze in, maar ik kan zeggen: ik doe eerst paragraaf 5.1, dan 5.3 en dan pas 5.2 want dat vind ik logischer. En als je straks zelf een methode maakt, moet je ook gaan nadenken: wat vind ik logisch? En het kan zomaar zijn dat een leerling die daar gebruik van maakt denkt van: maar dit is niet logisch en doe eerst stap 1 dan 3 dan 2 en de volgende doet gewoon 1 2 3.

C: Is het dan handig om de methode zo op te bouwen dat leerlingen hun eigen volgorde kunnen kiezen?

H: Ja, kijk als jij een website maakt met informatie van hoe moet je programmeren met voorbeelden en zo. Dan moet je kijken naar W3school.com als het gaat om web-development

skills. Dan heb je eerst allemaal programmeertalen, dat is de eerste stap, dus het is eigenlijk een soort trechter. Je klikt op HTML, vervolgens zie je HTML-basics, HTML gevorderd en HTML-expert. Basics gaat over hoe maak je een simpele pagina, bij gevorderd doe je tabellen en submit formuleren en bij expert ga je dieper in HTML5 kijken zoiets. Dus als je een methode maakt die bedoeld is om leerlingen te leren programmeren moet je ook zo een "top down" ding hebben. Dat kan van boven naar beneden dat kan van links naar rechts. Je kunt een menubalk maken met onderwerpen en als je eroverheen gaat met hover krijg je weer meer opties, dat moet je helemaal zelf weten. Als het maar gestructureerd is. Zodat iemand die heel snel door heeft hoe het zit, sommige mensen hebben daar echt aanleg voor, die missen net even een stukje informatie, dat moeten ze snel kunnen vinden. Net als dat je achter in het boek in een register kijkt, dan zoek je arbeid op en staat er blz. 17, dat is op een website net zo.

V: Zijn er dan ook dingen van SN die u mist, waarvan u denkt ik zou het fijn vinden dat ze dat aan de leerlingen geven of ik mis dat?

H: Er zijn heel veel methodes die ook practica beschrijven in het boek. SN doet dat niet en stelt ook geen materiaal daarvoor beschikbaar. Waarschijnlijk is hun keuze, sommige onderwerpen kan je geen practicum overdoen: zoals medische beeldvorming in VWO6, dan moet je met radioactieve stofjes gaan werken. Ja, dat kan wel, maar dan moet je een licentie voor hebben en geschoold personeel; wij hebben dat overigens niet. Dus zij kiezen ervoor om de practica eruit te laten, wat er wel als vervanging is: oké die en die doet dit proefje met deze resultaten en hoe moet je dat verwerken? Dus de context-concept benadering zit er wel gedeeltelijk in. Wat mij niet bevalt aan SN is dat er in het boek geen oefentoets. Die zit dan wel weer in de digitale methode, maar er zijn altijd leerlingen die niet in de digitale methode kunnen, die hebben geen toegang tot de oefentoets, dat is onhandig. Ik had hem liever in het boek gezien.

V: Ja, zodat je alles op een plek hebt: dus als je een ding hebt, het boek van Iddink of je kan bij de website dan heb je alles. Dat het niet verspreid is?

H: Ja, want nu, kijk V4 is best wel dik, V5 wat dunner en V6 nog dunner. We gaan van 7 naar 4 naar 3 hoofdstukken. Uiteindelijk was er in V5 en V6 boek ruimte voor oefentoetsen. Want we hebben aan het eind van het hoofdstuk wel de afsluiting, dat zijn wel inzicht vragen, maar de oefentoets staat dan weer online. Ja dat is een keuze, dat had ik liever in het boek gezien. Want in het boek mag je schrijven, maar soms moet je bij de oefentoets nog dingen printen: dat ik denk, ja stop het dan in het boek, dan is het al geprint.

V: Ja, nee dat is een beetje raar. Op welke manier bereidt u uw lessen voor?

H: [gelach] Korte antwoord: niet. Nee, ik heb een hele duidelijk lijn, van wat ik belangrijk vind en we hebben op school een programma met wat we allemaal doen. In de natuurkunde is ook wel een opbouw. Bij scheikunde bouw je een huis: je moet eerst de fundering leggen, met atomen en reacties en zo; daarna kan je door naar redox, zuur/base. Bij natuurkunde komt de mechanica altijd terug: dus iets met snelheid, krachten en energie. Dus uiteindelijk zie je dat eerste de mechanica komt, dat zit daarom in VWO4 en energie, dat is toch wat abstract zit in VWO5. En die abstractie nemen we mee naar VWO6 naar de energie van deeltjes en zo. Als ik mijn lessen voorbereid heb ik eigenlijk heel eerlijk door de tijd heen materiaal ontwikkeld. Ik heb een

PowerPoint met essentiële informatie uit de paragraaf en wat voorbeelden tussendoor. Ik gebruik die PowerPoint eigenlijk als leidraad. In dat opzicht: ik ben meer een verhalenverteller, bij het ene onderwerp kan ik heel makkelijk een verhaal vertellen en bij het andere onderwerp is dat wat lastiger. Oprecht, ik kan je colleges geven over radioactiviteit, maar over licht zijn we na een college klaar, ook vanwege mijn interesse. Ik heb een aantal kapstokken. Eigenlijk ben ik een docent die best wel ad hoc dingen doet, dus als ik een verhaal vertel en ik heb opeens een lumineus idee dan ren ik naar de kast en pak ik daar iets uit en laat ik wat zien. Dus ik dat opzicht ben ik iets minder, of nou ja, ik ben anders georganiseerd, laat ik het zo zeggen. Ik heb tijdens de coronaperiode mijn vakantie afgezegd, je kon toch nergens heen. Toen heb ik alle PowerPoints afgemaakt tot en met VWO6, dus het hele programma VWO4, 5 en 6 staat er. Oefenmateriaal heb ik ontwikkeld. Ik heb een YouTube kanaal met video's erop en voor V4 tot en met 6 is elk paragraaf beschikbaar, behalve voor VWO4, daar zitten wat gaatjes.

V: Hoezo had u het gevoel dat video's een nuttige toevoeging waren? Want u heeft daarvoor jarenlang zonder gedaan?

H: Nou die video's heb ik altijd al gehad, alleen in de coronaperiode werd het heel duidelijk, dat er ook leerlingen zijn, zeker omdat je ook les moest geven via een scherm. Nou dat is weinig inspirerend, en supersaai. En daarnaast als jij 9 uur per dag naar een scherm staart ben je echt een zombie. Dus ik zei, jongens we doen het een beetje anders. Ik heb 50 minuten van jullie tijd, ik ga die niet volkletsen. Mijn doel ik dat je een opdracht afrond, en als je dat heel snel doet door eerst de video te kijken en dan snel die opdracht te maken ben binnen 25 minuten klaar. Dan kan je die andere 25 minuten chillen: dan kan je koffie halen, of in de zon zitten. Of wat uitgebreider lunchen als het daar tijd voor is. En uiteindelijk wilde ik het ook als naslagwerk doen, er zijn veel leerlingen die weleens een les missen en dan wil ik niet dat ik nog heel veel tijd kwijt ben aan dingen nog een keer uitleggen. En dat ik kan zeggen: kijk even die video, dan ben je binnen 15 minuten klaar. Dat kan gewoon in je bed of op de bank, dat moet je zelf weten, daar zit alle informatie in. Stel je voor je breekt je been, dan moet je thuis ook nog door kunnen.

V: Maar normaal gebruiken leraren daar het boek voor.

H: Ja, alleen bij natuurkunde dat sommige echt een verbale toelichten behoeven. Met een voorbeeld of iets wat je aanstreept, iets wat je visueel maakt. Dat is het voordeel van video's, ik heb dan een tekenprogramma en een tekentablet (overigens al voor alle andere docenten) waarmee ik het duidelijk maak. Dat wilde ik gewoon doen, het hele flipping-the-classroom-principe geloof ik niet in: als in we verplaatsen de uitleg naar thuis en doen in de les opdrachten. Want wat gaat er gebeuren: leerlingen zien de video niet en wat doen docenten uit de goedheid van hun hart? Die gaan het opnieuw uitleg, dat stimuleert leerlingen niet. Dus het flipping-the-classroom ben ik niet principieel voor, alleen in VWO5 doen we het nu, vanwege het tijttekort. Wat je wel ziet, als je het gebruikt als naslagwerk: ik zag een enorme piek in het aantal weergaven vlak voor het eindexamen, van alle VWO5 en 6 onderwerpen. Dan weet ik gewoon: mijn leerlingen zitten dit te kijken omdat dingen nog niet helder zijn. En dat is bij jullie stukje wel heel belangrijk: want er bestaan talloze video's over hoe je moet programmeren, maar ze doen vaak alleen een voorbeeld. Dan doen we zo van [typt heel snel op het toetsenbord] en F5 en het is goed. Terwijl je wilt dat leerlingen het uiteindelijk begrijpen: waarom werkt iets zo. Stel je programmeert iets in Python, ik maak weleens modellen over vallen met wrijving ofzo. Dan ga

je een hele hoop variabelen definiëren, en je moet een keer vertellen dat je halverwege je code kan je variabelen herdefiniëren. Alleen als leerlingen zien: $v = 10$ meter per seconde en verderop $v = 20$ meter per seconde dan snappen ze het niet meer. Ja maar, we hadden het toch al gedefinieerd? Dan moet je duidelijk maken dat je halverwege dat je waardes kan vervangen. Of je kan iets heel vaak laten uitrekenen, dat hebben we met modelleren gedaan. Gewoon iedere keer een heel klein stapje. Hoe zit het met het gebruik van dubbele punten, aanhalingstekens, haakjes? Wat voor slashes gebruiken en waarom gebruik je deze slash en niet de andere slash? Waarom moet hier een dubbele quote boven? Hoe geef je commentaar? Kan je meerdere lijnen commentaar geven als je dat prettig vindt, en hoe doe je dat dan, en wat als je maar een lijntje doet? Zeg maar in Python kan je met een hashtag een lijn commentaar geven, maar met 3 aanhalingstekens kan je een heel blok aan tekst toevoegen. Dat ligt er maar net aan wat je aan het maken bent. Stel je bent een website aan het maken, dan is het wel handig dat als je een ingewikkelde functie hebt, dat je erbij schrijft wat het doet. En als dat uit meerdere lijntjes gebruikt moet je dus drie aanhalingstekens gebruiken in plaats van "functie voor het verwerken van de inloggegevens". Ik noem maar wat. Dat zijn wel de dingen waarover je moet nadenken, dat ik ook de reden waarom in mijn video's ook een kop en een staart zit. Dus ik begin met: wat ga ik in deze video bespreken? Daarna behandel ik het aan de hand van voorbeeld, plaatjes, applets. Daarna werk ik een voorbeeld uit: hoe pas je dat dan toe, hoe schrijf ik dat dan op? Wat is impliciet, welke eenheid moet erbij, hoe rond je af? En uiteindelijk kom je terug, wat hebben we besproken, met een korte samenvatting en dan is de cirkel rond.

V: Dus u heeft best wel veel tijd gestoken in het maken van video's, was het dan prettig geweest als de methode die erbij zou leveren?

H: Nee, vanwege het feit dat ik het principieel oneens ben met iemand anders. Dat is mijn eigen arrogantie. Wij hebben op school een bepaalde visie, een bepaald programma. Wij doen het op een bepaalde manier. Ik maak mijn video's wel op basis van het boek, want ik gebruik plaatjes en figuren en zo. Na Engels zijn er de meeste Natuurkunde methodes, op de tweede plek. Omdat natuurkundigen het oneens zijn over wat moet je eerst behandelen zodat het ander logisch uitkomt.

V: Maar kinetische energie werkt op elke school hetzelfde, dus je zou zeggen dat als je losse uitlegvideo's hebt, kan je ze zelf in een bepaalde volgorde aan je leerlingen geven.

H: Zeker, alleen het nadeel is dat iedereen een bepaalde manier van uitleggen heeft. En dat sluit aan bij wat leerlingen fijn vinden of juist totaal niet. Jullie hebben les gehad van verschillende docenten, bij sommige klikte het wel bij andere niet. De ene maakte het alleen maar vager en de ander maakte het alleen maar duidelijker. Ik weet niet of jullie les gehad hebben van meneer S.? Nou die maakte het alleen maar vager. Kijk als je les hebt gehad van meneer L. heb je een hele andere manier van lesgeven gehad dan van meneer Alderliesten of van mij. Want meneer Alderliesten en ik lijken best wel veel op elkaar qua uitleg. Alleen ik ben iets amicaler en meneer Alderliesten is iets beheerster en iets meer gestructureerd en ik doe gewoon: oh, ik heb nu hier zin in en doe heel veel uit de losse pols. Terwijl meneer Alderliesten meer aan zijn PowerPoint vasthoudt als leidraad, daar wijken we amper vanaf. Ik ben meer, ik zou niet willen zeggen, flexibeler; maar ik zit anders in elkaar. Als ik vind dat het nodig is om af te wijken van de PowerPoint, dan wijken we zeker af.

C: Maar dat ligt eraan bij leerling wat die daarvan vindt. Sommige zullen uw lessen fijner vinden, en anderen totaal niet.

H: Dat mag, je hebt nooit een volledige dekking. Als je een uitleg maakt zal je ongetwijfeld wat vergeten of als je het later terugleest dat je denkt: heb ik dit opgeschreven, dit klinkt helemaal niet logisch. Want hetgeen wat ik door de tijd heen gebruikte om te leggen gebruik ik niet meer, omdat het het alleen maar onduidelijker maakte. Dan gebruik ik anderen vergelijkingen of enzovoorts. Dus om terug te komen op de vraag: ik heb heel veel video's gezien van collega's uit de landen. Ik vind de video's van meneer Wietsma, ik heb samen met heb gestudeerd, alleen ik vind zijn manier van uitleggen prima, maar hij praat veel te snel. Dus ik zet zijn video's op 0,75 keer de snelheid. Maar er zijn ook collega's, [praat mompelend en langzaam] dan moet je de snelheid aanpassen naar twee keer, [weer normaal] omdat het anders gewoon niet te doen is. Kijk, om mijn video's zal ongetwijfeld iets aan te merken zijn, alleen omdat ik het principieel niet eens was met iedereen maakte ik het zelf. Als mijn leerlingen komen met: meneer, ik heb de video van Wietsma gezien en nu snap ik het, helemaal top. Uiteindelijk is mijn doel dat de leerlingen het snappen en of je dat nou bereikt met gedeeltelijk hulp van mij, gedeeltelijk hulp van een ander, maar je bereikt je doel: dikke prima. En als je je doel volledig door mij bereikt, ook helemaal prima. Maar als je je doel niet bereikt door mij en volledig door iemand anders, maar je hebt er niets over gezegd, dan word ik boos. Want ik wil het graag weten; zijn er dingen die anders moeten. Dat is denk ik best wel belangrijk aan lesgeven, dat je openstaat voor feedback. Als leerlingen achter je rug om gaan roddelen, dat is mijn eer te na. Ik wil juist dat ze het tegen mij zeggen, dat ik er rekening mee kan houden. Want als ik het via-via moet vernemen... Hoe meer schakels er in de keten zitten, hoe meer het verhaal verdraaid is. Je kent dat spelletje wel dat als je in een kring staat met tien man en ik vertel de eerste een geheim, en dat moet bij de tiende uitkomen, dan klopt er niets meer van.

V: Ja ik heb eigenlijk nog een vraag: zijn er nog dingen die u geleerd heeft in de afgelopen tijd lesgeven? Waarvan u eerst dacht dat het een heel goed idee was, en dat uiteindelijk toch niet zo een goed idee bleek?

H: Ik ben erachter gekomen dat als je zelf boven de stof staat, en dat mag ik van mezelf echt wel vinden, dat je niet op elk niveau zomaar op de bonnefooi les kan geven. Dus, waar ik dit jaar in VWO6, het was het eerste jaar dat ik dit deed, echt achter ben gekomen, dat je jezelf echt goed moet voorbereiden. In VWO6 hebben we onderwerpen die we behandelen, alleen ik beheers die onderwerpen wel, maar je moet eigenlijk net verder denken, want er zitten ook leerlingen bij die zelf lezen en video's kijken en dan zeggen: ja maar meneer, dat klopt niet. En dan zeg ik: ja dat klopt, dat komt later pas. Dat komt omdat ik erover na heb gedacht, maar als ik dat niet heb gedaan, sta je met een mond vol tanden. Je moet een balans zoeken, tussen wat is het echte verhaal en wat is het verhaal dat we vertellen. Hetgeen wat we jullie vertellen over schakelingen klopt, omdat ze allemaal heel simpel en met gelijkstroom zijn. Alleen als we het kijken naar de werkelijkheid... hebben jullie al integeren gehad met wiskunde B?

V & C: Ja.

H: Nou bij integeren ga je kijken naar een heel klein tijdstapje en bij differentiëren doe je dat ook. De natuurkunde zit vol met ingewikkelde wiskunde, alleen als je wilt kijken naar wat is er aan de hand in de echte wereld met bijvoorbeeld wisselstroom, dan kom je op eens in de differentiaalvergelijkingen. Ga die maar eens oplossen, dat is echt lastig. Dat doen we niet met jullie, we zeggen we doen een versimpeling en met gelijkstroom gelden deze regels en reken het maar uit. Maar in de werkelijk is het helemaal niet zo simpel. Ik moest iets over kwantummechanica gaan onderwijzen, ik vind dat heel interessant maar snap het niet. Zelfs de professoren, die er heel diep over nagedacht hebben zeggen tot op de dag van vandaag: ik snap er niets van. Je moet accepteren dat het zo is. En met programmeren is er uiteindelijk iemand die de vertaalslag gemaakt heeft van de programmeertaal naar de machinetaal, en die heeft een library gemaakt. Waarom die library zo in elkaar zit moet je je niet afvragen, geen idee. Dat moet je je niet afvragen, die is er gewoon. Een hele slimme kerel of vrouw heeft dat heel goed over nagedacht, dat botst natuurlijk ook met mensen, die zeggen: dat is niet logisch, niet goed over nagedacht. Maar daar ga je straks achter komen, sommige dingen moet je expres weglaten en sommige dingen moet je heel veel aandacht besteden. Dat is wat ik geleerd heb, ik was voorheen, ik ben best wel wiskundig ingesteld, en ik deed al die wiskundige afleidingen. Maar degene die wel goed zijn in wiskunde, die kunnen dat prima volgen, maar leerlingen met wiskunde A die zien de cijfers en letters dansen en zeggen: geen idee waar hij het over heeft. Terwijl als je het aanschouwelijk maakt, kan iedereen het volgen en de wiskunde komt later wel. Dat is misschien wel het grootste inzicht van de afgelopen jaren. Want de natuurkunde is heel wiskundig, alleen het voordeel is dat natuurkunde ook een realistische component heeft. Bij wiskunde deed je in de onderbouw iets met een taxi met een startbedrag en zoveel euro per kilometer, en daar hingen ze een lineair verband aan op. Hartstikke leuk, en dat klopt ook wel met de echte wereld. Maar naar mate je verder komt in de wiskundewereld worden de voorbeelden minder sprekend. Als je het gaat hebben over differentiëren en integeren, dan zeggen we: we maken het tijdstapje super klein, ja wat is superklein; kan ik dat meten op mijn horloge of mijn stopwatch? Op een gegeven moment verlies je de verbinding met de realiteit. En dan is het de taak om de boel om te draaien en de connectie te leggen met de realiteit en vervolgens koppelen we daar de wiskunde aan. Dan zeggen we: oké, we nemen de integraal van 0 tot r . Je kunt de formule van de gravitatie-energie wiskundig aantonen vanuit een integraal, je kunt ook zeggen: de afstand van de aarde neemt af of toe en daarom wordt de aantrekkingskracht groter of kleiner, dan ga je in pijltjes denken. Je kan zeggen dat zijn vectoren, dan kan je er wiskundig over nadenken. Je kan ook zeggen: als ik verder wegga van iets, trekt het minder hard. Als iets minder hard trekt, hoef je er ook minder energie in te stoppen. Allemaal van dat soort concepten. Dus uiteindelijk is dat de grootste realisatie die ik heb gehad, dat je niet alles wiskundig moet doen, wat voor jou logisch klinkt. Ik heb heel veel opgaven gemaakt, ik heb heel veel wiskunde gedaan, maar dan krijg je het kaliber: ja maar dit snap je toch? En dan zie je leerlingen kijken van: waar heeft hij het over. Terwijl als je zegt, iets met pijltjes. Met elektromagnetisme zul je het zien, jullie hebben het waarschijnlijk al gezien: als een stroom van je afgaat zie je dat als een pijl als je aan de achterkant naar die pijl kijkt zie je een kruis vanwege die veertjes en als een pijl naar je toekomt zie je een puntje en moet je rennen. Dat is wel aanschouwelijk, waarom is dat dan zo? Ja, dat heeft iets te maken met het uitproduct en iets met vectoren; meneer, wat is een uitproduct. Het moet tot de verbeelding spreken. En bij programmeren, het is redelijk wiskundige, moet je nadenken over hoe je de vertaalslag van de echte wereld kan maken naar dit? Stel je wilt ze een model laten maken over iets wat aan het veren is. Je maakt een tekening van de situatie en dat ding blijft stil, de positie blijft onveranderd.

Dus als ik mijn stopwatch start en blijf ernaar kijken gebeurt er niets, dus horizontale lijn. Vervolgens duw ik hem in, en we weten van een veer, dat als ik hem loslaat dat hij terugschiet. Wat gaat hij dan doen? Dan gaat hij heen en weer bewegen, dan kan je het hebben over positieve en negatieve richting. Maar uiteindelijk kan je zeggen: dat is een sinus of een cosinus en dan kan je iets vertellen, maar als ik met een video kijk waar dat ding is krijg ik heel veel beeldjes, en elk beeldje komt overeen met plekje. Dan krijg je een hele lange lijst met data, dan kan je het concept van een array uitleggen. Zo ga je van de werkelijke wereld naar iets wiskundigs. Vervolgens ga je kijken: dus als we de positie kunnen berekenen, dan definieer ik wat variabelen en dan laat je dat wegschrijven in een lijst. En die lijst laat je printen op je scherm en zie je een lijn. En dat is iets, ik ben ook niet heilig, maar dat is wel iets wat tot de verbeelding spreekt. In plaats van: we gaan kijken naar een blokje wat oscilleert en iets met een sinus-vorm. De formule voor de sinus. Wat is een sinus? En vervolgens als mensen dit zien, iets wat heen en weer gaan en je hebt de volgende analogie. Je hebt iets wat op en neer gaat en dat maak je vast aan een treintje, dat doet dat dit [maakt de vorm van een sinus in de lucht] en daar kan je het concept aanhangen. Dat ding gaat alleen maar op en neer, maar je kijkt in de tijd, iets gaat van links naar rechts, dan kijk ik waar was het iedere keer, en daar kan je een hele hoop dingen aan ophangen en spreekt veel meer tot de verbeelding dan als je zegt: jullie hebben het over de sinus gehad. Wat is de sinus? Dat is denk ik de grootste realisatie, en uiteindelijk hoe dat dan werkt. Voor mijn studie heb ik een opdracht voor TNO gedaan, daarvoor moest ik een computermodel maken, maar ik ben geen programmeur en moest op Stack Overflow en W3Schools allemaal dingen opzoeken over Python. En dan zit ik te kijken en denk ik: wat is een tuple? Geen idee, ik weet het nog steeds niet. Kijk, bij een integer kan ik me iets voorstellen, dat is een geheel getal. En dan zit je ernaar te kijken en krijg je een foutmelding en zit je te kijken: ja maar waarom dan? Dus dat is ook iets waar je aandacht aan moet besteden, dat doe je in de les ook, als een leerling zijn vinger opsteekt en zegt: meneer, waarom komt u daarop uit, ik kom op iets anders uit. Dan moet je niet zeggen, ja dat van jou is fout. Maar je moet het hele proces doorlopen en kijken waar de kink in de kabel zit. Want dat is bij natuurkunde wel zo, het vak is heel gevoelig voor misconcepten. Als jij het gevoel hebt, ik kan dit zo intikken en vervolgens doet het niet: ja maar waarom dan? Dat zijn wel van die dingen die vervelend zijn. En uiteindelijk ontstaan de misconcepten uit beelden die wij al hebben. Heel veel mensen denken dat als de resulterende kracht de andere kant op is dan de beweging dat je spontaan de andere kant op beweegt, maar dat is niet zo: je komt eerst tot stilstand en dan pas ga je de andere kant op. Maar als je dat misconcept in je hoofd hebt, ga je dat wel structureel inzetten voor van alles en nog wat en dat zorgt ervoor dat je niet goed uitkomt. Dus de kunst van het lesgeven is de laatjes in hoofden opentrekken, vervolgens nieuwe informatie aanbieden zonder misconcepten, maar leerlingen vormen daar altijd hun eigen beeld bij, dat heet constructivisme: jullie construeren jullie eigen kennis en informatie in je hoofd, dat is ook de reden waarom je niet kant-en-klaar een USB-stick in je hoofd kan stoppen en kan kopiëren plakken, je moet dingen actief verwerken. Daarom zijn aantekeningen maken, dingen opschrijven belangrijk, zo onthoud je dingen. Als ik tegen je zeg: hier zijn tien getallen, lees ze even en daarna ga je wat anders doen, en daarna moet je vertellen welke getallen het waren. Ik denk dat je er drie goed hebt. We kiezen dan wel 171 en 63 ofzo. Als ik je zou vragen om ze over te schrijven, weet ik zeker dat je er zes hebt onthouden. Omdat je dat motorisch doet, met je hersenen, daarom onthoud je dingen beter. Programmeren is een kwestie van bekijken, lezen en nadoen. Dat is wel een beetje trail-and-error. Als het trial stuk goed gaat is dat heel belonend, maar het error stuk is heel frustrerend. Dat is wel het lastige aan programmeren: mensen

moeten weten wat we moeten doen, als ze niet meer weten wat ze moeten doen. En dat is het moeilijkste stuk, dus dat zijn wel dingen om rekening mee te houden.

V: Oké. [Tegen C] Ik weet niet of jij nog vragen hebt?

C: Nee, niet echt.

V: Nou dat was hem dan.

H: Nou dit was mijn spreekbeurt [gelach]

Interview with Lars van Bokhorst

V: Op welke manier geeft u structuur aan uw les?

B: Door van tevoren leerdoelen op te stellen, zodat leerlingen weten wat we vandaag gaan doen. Dat herken je ook wel van mijn PowerPoint, ligt er een beetje aan welke les we hebben, maar het helpt mij als docent wat we gaan doen. En als je het helemaal goed wilt doen, doe je dit ook nog even op het einde.

V: En op welke manier stelt u die doelen dan op? Door het boek door te bladeren?

B: Ja, door het boek bladeren, door van tevoren nadenken wat ik de leerlingen leren. Ik wil ze misschien een oplosreactie van een zuur leren, dat ze dat aan het einde van de les kunnen. Soms heeft de methode de leerdoelen, en tegelijkertijd, dat is ook wel iets wat ik volgend jaar op mijn nieuwe school ga doen, en dat geldt ook voor jullie, want jullie hebben nog geen methode, die zijn jullie aan het schrijven, dan kijk ik naar de leerdoelen van het CITO. Want het CITO heeft leerdoelen opgesteld; wat een leerling moet kunnen voor het eindexamen. Die zijn voor het vak informatica zeker opgesteld, dat is wel mijn advies wat ik jullie mee wil geven. Het CvTS, dat zijn de mensen die examens maken, zij maken ook die leerdoelen. Google daar eens op. Maar die leerdoelen kun je vinden, en als je weet: dit zijn de leerdoelen, ga je nadenken wat past bij het begin. En je begint natuurlijk niet met zuur-base, als leerling nog geen reactievergelijking kloppend kunnen maken. Dus maak ik de keuze om de reactievergelijkingen eerder in het jaar te doen.

V: En, zijn er verschillende soorten lessen over een heel hoofdstuk gezien? Wat zijn dingen die terugkomen elk hoofdstuk?

B: Ja, er zijn wel dingen die ik probeer te bedenken van tevoren, dat ik het hele hoofdstuk in 1 keer uitleg. Dat wil ik nog wel eens beslissen, anders heb ik elke keer een blokje, waarvan leerlingen denken: huh? Terwijl als ik 1 keer het hele plaatje laat zien: dit is waarvoor we het doen, en daarna knip ik dat in brokjes. Het kan ook zijn dat ik denk: dat is niet nodig, dan knip in het sowieso in brokjes. Ik denk dat je dat ook wel herkent in mijn lessen.

V: Wanneer kiest u ervoor om het hele hoofdstuk in 1 keer uit te leggen en wanneer niet?

B: Dat is afhankelijk van het hoofdstuk, soms weet ik gewoon dat dit logischer wordt als je het hele plaatje kent. Bij redox en zuur-base doe ik dat. Bij redox is het relaxed dat je weet dat we naar die half reacties toe gaan, maar de eerste les hoeft je dat niet te kunnen. De eerste les hebben we het over of een elektron voor of na de pijl komt. Dat is fijner als je al het eind-stukje hebt gezien. Maar als we gaan kijken naar Lewis structuren, vind ik het niet nodig om te beginnen met het einde, maar vind ik het juist fijn om te beginnen met wat een Lewis structuur is en wat die puntjes inhouden. De les daarop gaan we pas Lewis structuren opbouwen, dus het is heel erg afhankelijk van het onderwerp. En ik denk dat bij een vak zoals informatica, bij het programmeren, dan maak je de keuze: er is een taal en we beginnen bij de basics van die taal.

Je kiest ervoor om niet te beginnen met een website maken met CSS, maar eerst leg je uit hoe de taal werkt en welke variabelen je nodig hebt.

V: Dus u probeert eerst een overzicht te maken, zodat we uiteindelijk op elk stukje kunnen inzoomen, terwijl het duidelijk blijft wat we aan het doen zijn.

B: Ja, soms doe ik dat ook wel in de 3e, als ze het moeilijk vinden, dan zet ik het rechtsonder in de PowerPoint. Dan zet ik het recht onderin, het woord polymeren vonden ze lastig, dus ik schreef op "om plastic te maken".

V: Ja, dus ook de realiteit erbij betrekken?

B: Precies, een context erbij te betrekken, dat is heel erg belangrijk. Dat zeg je heel goed.

V: En hoe doet u dat, door bijvoorbeeld te kijken: dit zijn zuur-base reacties, hoe worden die gebruikt?

B: Ja, precies. Wat ik zelf uit mijn eigen ervaring heb meegenomen. Van de spiegelisomeren, dan geef ik context. Dan hebben we het over softenon. De helft van het medicijn zorgde voor mismaakte kinderen en de andere helft werkte. En dan geef ik de context eraan, dat het molecuul het licht linksom of rechtsom kan breken. Dan geef je er context aan, zodat het ook leuker wordt. Je kan het aan een kapstokje hangen. Als leerlingen denken: o ja, ze hadden medicijnen gemaakt, maar ze hadden bepaalde kennis niet en die kennis gaan wij nu leren. Zodat als jij geneeskunde wilt doen, dat jij daar een koppeling aan kan leggen. En over zuren, ja, er zijn natuurlijk zwemmers. Ik vraag wel eens wie er zwemmen en die zeggen dan dat ze met staafjes in het zwembad moeten en kijken of het zwembad is. Nou ja, dan maak ik een koppeling aan indicatoren. Vakken zoals scheikunde zijn opgebouwd uit context-concept benadering, natuurkunde volgen mij ook. Misschien hoorde je dat ook al van mijn collega's. Concept is dan een reactievergelijking maken en de context is dan: we hebben de boel in de fik gestoken en er komt rook bij vrij.

V: En als u aan het voorbereiden bent? Hoe doet u dat, u gebruikt PowerPoints, hergebruikt u die?

B: Ja, weet je, het eerste jaar maak je die PowerPoints, dan komen er fouten in. Dat is nou eenmaal zo, een mens maakt fouten. Het mooiste is eigenlijk een dubbele klas: twee keer V5 of twee keer V4. Dan denk je de eerste keer: oh, er zit een foutje in. Dan pas je dat snel aan en kan je vaak dezelfde week die les nog een keer geven. En het jaar erop pas je jouw mening weer aan en verander je wat dingen. Op een gegeven moment staan die PowerPoints er gewoon. Ook een voordeel van lang lesgeven is dat je soms kan denken, ik heb even geen zin in die PowerPoint, ik doe het even op het bord. En daarbij komt ook nog wel eens, als ik in de klas zie dat wat er in de PowerPoint staat niet werkt, nou... dan ga ik iets anders doen. Maar goed, het is inlezen, PowerPoint is de structuur en daar kan ik tijdens de les van afwijken, door te kijken naar wat de leerlingen fijn vinden.

V: Ja, precies, dus u kijkt vooral naar wat de leerlingen fijn vinden en daar past u dan uw uitleg op aan.

C: Dus ook gewoon veel proberen?

B: Ja, dat is wel een advies, je moet niet denken, ik probeer het de eerste keer en het werkt niet, terwijl het de tweede keer wel werkt. En het kan ook zijn dat het bij de ene klas niet werkt en bij de andere wel. Dan is het allebei V4.

V: En hoe komt dat?

B: Dat verschilt per niveau, of de interesse van de groep. Ik neem het verhaal van de zwembaden van net. Ja, als er niemand zwemt in de klas wordt er niet echt gereageerd. Of er zit een figuur in de klas en die zegt dan: "Ja dat moet ik altijd doen van mijn zwemleraar!" en heeft daar opeens een hele mening over en gaat er vragen over stellen. Ja, dan wordt je invulling anders.

V: Als u een uitleg geeft, op welke manier bouwt u die op? We beginnen met de context, maar de uitleg van het concept, zijn er standaard stappen die u maakt tijdens de uitleg?

B: Ik probeer altijd, dat vinden leerlingen volgens mij wel fijn, dat moeten jullie maar zeggen zo, maar ik probeer uit te leggen waar het vandaan komt: waarom is het zo? De grote vraag waarom? Het valt me op dat ik het bij havo minder doe dan bij vwo. Bij havo is het: dit is een zuur, dit gebeurt er. Ja, ik heb vandaag zuren uitgelegd, daarom heb ik die in mijn hoofd. Maar ja, weet je, dan is het gewoon zo. Terwijl die vwo-leerling zegt: "Ja maar meneer, als ik weet waar het vandaan komt snap ik het beter en kan ik beter mijn kennis erop toepassen. Daarom probeer ik erover na te denken: waarom is iets zo? Waarom doen we het zo? Die probeer ik uit te leggen, totdat we alle 'waaroms' hebben gehad.

V: Denkt u dat dit alleen bij scheikunde van toepassing is of bij meerdere vakken gebruikt kan worden?

B: Ik denk dat dat bij natuurkunde ook zo is, net als bij wiskunde.

V: Want vaak is waar het vandaan komt ook onderdeel van de stof.

B: Nee, niet altijd. Dat valt mij bij scheikunde op. Soms zegt het boek: dit is hoe het is. Soms zeg ik in de les: tot hier en niet verder, anders ben ik drie lessen verder. Neem bijvoorbeeld waarom elektronen in paren voorkomen, terwijl min en min afstoot? Dan probeer ik toch iets te zeggen over een op- een neerwaartse draai zodat er aantrekkingskracht ontstaat. Maar dat is een hele minimale uitleg. Want dan gaan we de kwantumchemie in en gaan we elektronen als golven benaderen, nou weet je, dan ben ik 80% van de klas kwijt.

V: U versimpelt het en doet alsof het daar vandaan komt, terwijl je eigenlijk nog dieper kan.

B: Ja, precies, en het kan dan zijn dat ik een rondje maak, en ze zeggen: meneer, hoe zit het nou precies? Dan wil ik nog wel wat dieper gaan. Maar tijdens de klassikale uitleg is er een grote valkuil, want als je alles gaat vertellen zijn er misschien vijf leerlingen die denken "meer, meer, meer". Terwijl driekwart van de klas zoiets heeft van: "oh my god, wat is dit?" Nu snap ik het niet meer terwijl ik het eerst wel snapte. Dat is iets waarvoor je moet opletten. Dus als je een variabele gaat uitleggen aan een groep, dan moet je gaan zeggen: dit is dit en daar gaan we mee oefenen.

V: Ja, je moet een lijn trekken en ervoor zorgen dat de hele klas het onder de knie krijgt voordat we verder gaan.

B: Volgens mij is het met wiskunde ook zo. Je leert in de inhoud van een bol. Als je kunt differentiëren en integreren in de bovenbouw kan je die formule gaan herleiden, maar dat doe je niet in het begin.

V: Ja, meneer Heerkens zei ook dat hij in het begin de wiskundige afleiding van een formule deed; waarom een formule zo was, maar daar is hij vanaf gestapt. Het is dan wel leuk dat drie mensen het snappen, maar de rest niet meer. Dus dat is ook iets, u wilt de uitleg zo simpel mogelijk houden omdat het anders niet aankomt.

B: Ja, vooral niet de eerste keer, sommige onderwerpen komen na de vierde ook in de vijfde en zesde terug en dat heb je het steeds maar herhaald. Dat is ook een advies: herhaling is key.

V: Ja, en dan over de methode, we gebruiken bij scheikunde Chemie Overal, wat vindt u fijn aan CO?

B: Op zich vind ik de structuur fijn, de indeling van de paragrafen en de hoeveelheid tekst. Er staat niet superveel tekst in, maar ook weer geen tekst. Ikzelf had op de middelbare school Curie, waaruit ik les kreeg en daar stond geen tekst in, er waren alleen opgaven en wolken met stapjes. Dat vond ik zelf te weinig, dan word je te afhankelijk van je docent. CO biedt daar genoeg aan, maar wat ik mis zijn de contexten. Je ziet dat ze dachten: we moeten nu een context hebben, ook al heeft de context er niets mee te maken, we bouwen die context zo om dat hij er wel mee heeft te maken. Er zijn ook heel veel modules van de TU Delft, misschien moet je daar ook wel naar kijken. Ze hebben modules geschreven voor middelbare scholen om les uit te geven. Ik moet me er nog in verdiepen, ik heb de ervaring nog niet opgebouwd, maar die gaan zuur-base uitleggen met eerst een hoofdstuk over tandheelkunde en daar maken ze de koppeling naar zuur-base met tandglazuur. De TU Delft doet dat net wat beter dan CO.

V: Dus u zou willen dat de context een nog grotere rol speelt in de methode dan dat het nu doet?

B: Ja, want dan weet je waarom je het doet. Als jij een lift wilt bouwen, omdat je een bouwmodel aan het maken bent en je wilt er een liftje in omdat je in de eerste zit en dat lijkt je vet. Als je dan de wet van Ohm nodig hebt van natuurkunde klikt hij veel sneller, omdat jij het wilt. Je hebt dan die kennis nodig en dan wil je het leren.

V: Kan de context ook een valkuil zijn?

B: Jazeker, als er te veel rook ontstaat waardoor je het niet meer in de simpele context kan zien. Soms moet je een reactievergelijking opstellen, dan moet je er gewoon tien of twintig van kloppend maken en op een gegeven moment kan je het. Als jij elke keer een reactievergelijking moet opstellen om het te leren, maar je krijgt elke keer een hele lap tekst wat erboven staat, dan wordt je gek. Je wilt gewoon het concept leren van reactievergelijkingen kloppend maken. Je moet niet elke keer er dertig contexten boven gaan zetten.

V: Is de hoeveelheid context en concept goed verdeeld?

B: Ja, dat mis ik dus weleens, als je naar de eindexamens kijkt. Voor scheikunde lijkt het wel een boekwerk, er is gewoon heel veel tekst. En Chemie Overal doet heel erg zijn best om te doen wat ik net zei: korte opgaven met een concept. Vervolgens heb je wel contexten die dan weer heel kort; jullie hebben nooit een context van een halve bladzijde gehad dat je dan vervolgens een sommetje gaat maken. Dat is iets wat jullie niet herkennen aan Chemie Overal. Als je naar de eindexamens gaat kijken is dat wel meer, en dat zorgt er soms voor dat uit verhouding is. Maar tegelijkertijd, en dat maakt het complex, is een vierdeklasser nog niet gereed om zo een context aangeboden te krijgen. Dus dat wil je niet in de vierde klas, maar juist in de zesde.

V: Dus Chemie Overal is daar gewoon tussenin gaan zitten, terwijl een opbouwende lijn beter zou zijn?

B: Precies.

V: Want, u maakt ook de toetsen, probeert u dat daar wel in? Dat een V4 toets veel minder tekst heeft terwijl een V5 en V6 toets steeds meer tekst gaan maken.

B: Als ik kijk naar mijn V4 toets van evenwichten, die is één kantje. Er zit dan wel veel rekenwerk achter, de toets duurt 50 minuten, en dan komen ze nog tijd tekort. Maar op het moment dat we verder komen gaan we naar de examenstof en moet er meer tekst komen. En daardoor komt die begrijpend lezen vaardigheid. En daardoor ontstaat het verhaal: "ja maar meneer, de toets leek niet op de opgaven in het boek". Als je er een context aan vastplakt heeft een leerling het gevoel dat het anders is. Maar we zijn eigenlijk nog steeds een redoxvergelijking aan het opstellen, alleen met een ander verhaaltje.

V: Vindt u persoonlijk die enorme verhalen op het examen nuttig? Of als u zelf het examen zou herzien, zou u zeggen, laat die context maar zitten?

B: Dit is een hele interessante vraag, waar ik niets over te zeggen heb. Want het eindexamen is gewoon heel veel lezen geworden door de jaren heen en daar baal ik weleens van. Ik had graag gezien dat het korter was, als ik naar een scheikunde examen van twintig jaar geleden ga kijken, zijn ze veel korter en veel chemischer. Nu wordt er getoetst: kan jij goed lezen. Neem het examen van 2019, dat leek gewoon een IQ test! Ik denk dan: ik heb ze vier jaar scheikunde geleerd, allemaal concepten, dat is dan wel zonde. Maar ja, ik heb daar eigenlijk niks over te zeggen,

want als ik nu al die contexten uit het schoolexamen ga halen, heb ik de leerlingen niet goed voorbereid op het eindexamen.

V: En waarom denkt u dan dat CITO (de organisatie die de eindexamens maakt) toch die contexten in het examen stopt?

B: Volgens mij vindt het CITO leesvaardigheid gewoon heel erg belangrijk van een vwo-leerling.

V: Verschilt dat enorm tussen HAVO en VWO, ik weet bijvoorbeeld dat HAVO Wiskunde A 19 pagina's was?

B: Voor scheikunde scheelt dat wel. Als ik over dit jaar spreek (2021) waren er heel veel vragen die je gewoon kon maken zonder de tekst te hebben gelezen. Dat viel me heel erg op bij HAVO. Maar bij het vwo-examen had je geen idee gehad wat je moest doen zonder de tekst erboven. Maar het zou wel wat minder mogen, wat meer scheikunde. Ik heb eerder wel gezegd dat ik die context heel erg belangrijk vind in mijn lessen, maar als je op een lab werkt, is je context ook niet altijd heel erg lang.

V: Dus de focus van het eindexamen is doorgeslagen? De focus ligt te veel op lezen en niet op de chemie?

B: Ja, dat vind ik op dit moment wel.

V: En is dat een rare mening als u om zich heen kijkt naar collega's?

B: Nee, menig collega vindt dit ook.

C: Maar dat geldt niet alleen voor scheikunde, denk ik? Ook voor natuur- en wiskunde?

B: Ja, ze vinden dat leesvaardigheid zo belangrijk, dat komt continu terug. Als je ook naar het basisonderwijs gaat kijken, dat vraagt zich nu ook af: leer ik ze te veel trucjes? En werkt dat wel?

V: Bij informatica is er geen eindexamen, alleen maar schoolexamens. Dus wat dat betreft heb je als docent een wat vrijere keuze, omdat je niet moet afsluiten met een algemeen examen. Wat zijn dan dingen die u wel zou willen van een eindexamen, wat u nu niet heeft in een examen?

B: Ja, de teksten korter, dat is het grootste. Ja het hoeft niet helemaal to the point, ik vind die contexten wel heel erg belangrijk, maar dat rookgordijn vind ik echt onzin. Er zat een vraag in het examen waar ze "geen" in hadden verstopt, dus als je x0,3 moest doen, moest je eigenlijk x0,7 doen. Dat is gewoon leesvaardigheid. Iedereen snapt de chemie, maar leest over geen heen.

V: Wat mist u van Chemie Overal?

B: Hetgeen waar ze nu naartoe aan het werken zijn, en dat vind ik wel erg cool is dit. Stel je bent slecht in het metriek stelsel, van kubieke meter naar kubieke centimeter. Ik vind niet dat ze het op dit moment goed doen, maar goed. Als jij dat de hele tijd fout doet tijdens het maken van het huiswerk, moet er een melding komen om extra te oefenen, dat is het idee. Stel je maakt reactievergelijkingen kloppend en je doet het gelijk helemaal goed, dan kan je verder. Maar als iemand anders het lastig vindt, krijgt hij meer om te oefenen. Ik mis het levendige. De onlinemethode op dit moment is een online boek. Je kan dan wel leerroutes kiezen, maar de leerlingen die slecht zijn in scheikunde maken de ster-opgaven niet, terwijl die juist goed zijn voor die zwakkere leerling. Het lastige is dat scheikunde lastig is digitaal, je moet uitleggen enz., de computer kan dat niet nakijken. Maar met informatica is dat een stuk makkelijker, en dan kan je gaan kijken: hé, jij hebt dit goed gedaan, ga maar verder. Zo maak je het geheel adaptief. Nu zeg ik vaak: maak opgave 20 tot en met 30, maar bij 22 denk jij dan, ja ik weet het nu wel. Als ik dan van tevoren voor 30 leerlingen moet gaan bedenken welke sommetjes zij moeten maken zodat ze het goed genoeg leren, ja dat is niet te doen. Maar juist met iets als informatica, is daar meer ruimte voor.

V: Verder nog dingen die u mist? Ik heb natuurlijk geen ervaring met de docentenhandleiding...

B: Ja, die is prima. Ik kijk daar eigenlijk nooit naar. Ik heb gewoon een berg experimenten, dat is echt wel fijn. Daaruit kiezen we degene die wij leuk vinden. Wat is verder nog slecht aan de methode? Hmm. Nee, ik ben tevreden over de hoeveelheid tekst, het aantal dikgedrukte woorden.

V: Is het aantal opgaven genoeg? Is het te veel? Te weinig? Is het ooit te veel?

B: De leerlingen klagen er wel over als ik zeg dat ze een hele paragraaf moeten maken, dat het te veel is. Het is niet te weinig, ik denk dat het wel aardig is. Wat vinden jullie zelf?

C: Het lijkt altijd heel veel, maar als ik eenmaal bezig ben valt het wel mee.

V: Nou, paragrafen zijn best wel groot, er zitten veel verschillende dingen in. En er zit altijd 1 ding tussen dat écht lastig is, en daarvan zitten nooit genoeg opgaven in. De andere opgaven gaan dan weer over iets anders wat ik wel snap.

B: Ja, dat is wat ik bedoelde met dat adaptief leren, dat als je iets fout doet, dat er meer opgaven komen. Soms mis ik wel wat in Chemie Overal, en dan maak ik dat zelf. Zoals een boekje met oefenopgaven voor een schoolexamen, dat heb ik dan zelf, omdat ik weet dat leerlingen dat heel erg fijn vinden. Echte opgaven op examenniveau. Ja, dat is iets wat ik mis van de methode: conceptenstencils! Lijstjes met conceptvragen, zoals een lijstje met zuur-basereacties. Bijvoorbeeld ook met mol-rekenen, dan had ik ook gewoon de uitleg gekopieerd uit het boek en heel veel opgaven en dat werkte gewoon goed.

V: Dus er zijn wél te weinig opgaven in het boek?

B: Van bepaalde concepten. Ja, eigenlijk wel. De kale concepten missen, omdat je die nodig hebt om uiteindelijk een contextopgave te maken.

V: En met kwalitatieve analyse? Daarover staat ook niets in het boek.

B: Nee dat klopt, dat is iets wat wij hebben gemaakt. Ik miste een lessenserie waarin leerlingen zelf bezig zijn waarin alle concepten voorbijkomen. Alles komt voorbij: redox, zuur-base, organisch, alles.

V: Want is dit standaard in het examenprogramma?

B: Nee, alleen hier op school. Daarom knallen we de hoofdstukken er aan het begin van het jaar doorheen, zodat we aan het eind van het jaar alles kunnen herhalen met die practica. Dus ja, wat mis ik? Een stuk herhaling. Maar dit hoeft niet per se in het boek te zitten, want alle stof zit wel in het boek. Alleen ik maak deze herhaling zelf nog extra erbij. Volgend jaar gaan jullie titreren, en dan ga je met practica alles herhalen, zuur-base, redox, organisch, weer alles komt dan voorbij. En dan komt het wel in een context, maar dan weer net anders. Maar het stofjesonderzoek komt niet met een verhaal, maar wel met een context, dus anders dan in het eindexamen. Het is een continue zoektocht tussen contexten en concepten. Dat is als docent zijnde een innerlijke strijd. Ik wil jullie de concepten leren met vette contexten, maar ik moet jullie ook de examentrucjes leren: hoe lees je het nou? Dat valt samen met Nederlands. Als jullie een goede Nederlands docent hebben, gaat mijn gemiddelde omhoog, zonder dat jullie beter scheikunde kunnen.

V: Dat is dan eigenlijk een beetje raar. En dan: wat heeft u de afgelopen jaren tijdens het lesgeven geleerd? Waarvan dacht u eerst, dit is een goed plan, maar bleek dat helemaal niet zo te zijn?

B: Mijn valkuil is te veel diepgang geven. Toen ik mijn master deed zat ik heel diep in alle stof en dan ging ik gewoon te diep op de stof in. Vijf leerlingen snapten het dan nog en wilden meer weten, maar twintig andere snapten het niet meer. Een andere valkuil kan zijn: te weinig practica. Want in zo een practica komt alles samen: de context, je doet het, je ziet het, je ervaart het.

V: Dat geldt dus ook voor informatica. De leerlingen moeten gewoon zelf iets gaan maken.

B: Precies, jullie ervaren nu dat je de scheikundekennis kan gebruiken om uit te zoeken welke stof je in je potje hebt. We hebben het dan wat makkelijker gemaakt door maar vier keuzes te geven, maar op het lab krijgen ze iets en dan is de vraag: wat is het?

V: Ja dus, te weinig practica betekent dat de leuke concepten uit de lessen niet landen, omdat leerlingen het niet gezien hebben.

B: Juist, en daar heb ik dit jaar gewoon last van gehad. Maar dat komt door corona.

V: Ziet u ook dat de hoofdstukken met minder practica slechter gemaakt worden?

B: Nee, dat hoeft niet, het ligt aan het hoofdstuk. Waar ik het wel aan merk is dat de analyse (in H10) altijd minder goed wordt begrepen omdat we daar geen practica mee doen. We hebben gewoon die apparatuur niet om die practica te doen. Dan merk ik direct dat het slechter gemaakt wordt.

V: Dan mis je het praktische.

B: Ja, leerlingen moeten het echt doen. Leerlingen moeten het zelf gaan doen, dan leren ze echt pas.

V: [tegen C] Ik weet niet of jij nog vragen hebt?

C: Nee.

V: Nou dat was hem dan.

B: Top.

V: Dank u wel!

Interview with Matthijs Alderliesten

V: Als eerste, ja, dus het zijn een paar vraagjes. Als eerste hadden wij de vraag, op welke manier geeft u structuur aan uw les?

A: Goeie vraag, ik probeer heel veel hetzelfde te doen. Het begint eigenlijk te zeggen wat we gaan doen, heel kort een minuut.: wat hebben we vorige keer gedaan? Altijd opgaven op vergelijkbare manier aan te pakken, dus wat weten we al? Wat hebben we nodig en hoe gaan we daarnaartoe? En ik probeer ook altijd even aan het einde van de les even af te sluiten: dit hebben gedaan, dit is het huiswerk voor de volgende keer. Lukt niet altijd even goed, maar ik probeer het wel. Ik heb altijd een PowerPoint als ondersteuning voor mijn verhaal, zowel uitleg als antwoorden voor de opgaven en dat is soms uitgebreider, soms minder uitgebreid, Maar dat geeft de mogelijkheid om en te luisteren en te kijken en dat kost wel tijd in voorbereiding, maar ik denk ook wel dat dat rust geeft voor mensen die dat willen.

V: En hergebruikt u die PowerPoint of maakt u die elk jaar opnieuw?

A: Die hergebruik ik, maar de methode verandert natuurlijk ook elke keer en soms heb ik niet al opgaven uitgewerkt en moet ik die het jaar erna bijwerken. Soms denk ik, dit ga ik ook andere manier uitwerken, of ik ga een keer wat nieuws bedenken om het toe te lichten. Maar soms is er wel wat te recyclen.

V: Precies en als we dan kijken naar lessen over het hele hoofdstuk, zijn er dingen die elke keer terugkomen? Dus ja, dat is natuurlijk de toets bespreken, Maar de losse dingen die per hoofdstuk terugkomen, en die je niet per se elke les terugkomen, maar wel in elke periode weer.

A: Oké, dus we moeten een keer toets bespreken en een laatste les voor de toets met vragen. Ja, die zit er vaak wel in. Heel praktisch, het is vaak: we hebben nog 1 keer huiswerk, dat bespreken we en de 20 minuten die over zijn, zijn voor vragen. Voor het bespreken van de toets pak ik meestal een heel lesuur, wat ik meestal op dezelfde manier doe. Ik loop de vragen door, maar je hebt ook een uitwerking voor je neus, zodat je kan luisteren en kijken.

V: Zijn er dan nog andere onderdelen die elke periode terugkomen?

A: Nee

V: Dus alle andere lessen zijn: opgaven bespreken, uitleg, opgaven maken?

A: Ja, en soms een demo. Maar dat doen we niet zoveel.

V: En doet u veel practica bij natuurkunde, vindt u?

A: In VWO5 en 6 heel erg weinig. Nu in het nieuwe gebouw en met de lockdown is dat echt wel minder. In het oude gebouw deed ik dat wel meer; ik liep het kabinet in en pakte iets voor een

demo. Nu probeer je dat op het scherm te doen. Het zou wel meer mogen, maar elke les met een practicum moet ik ook weer bespreken, vind ik. En dan heb ik liever twee lessen meer voor opgaven dan voor een practicum, want uiteindelijk hebben jullie meer aan die opgaven.

V: Hoe bereidt u uw lessen voor? U maakt een PowerPoint dan, maar zijn er andere dingen die u doet voor een les?

A: De eerste keer als er een nieuwe druk is, ga ik die opgaven maken. Dan weet je wat er nieuw is en erin zit. Dan werk je ze uit op de PowerPoint en soms vind ik de uitleg in het boek niet zo goed, en kijk ik vooral: wat moet je weten voor de toets of het examen? Ik doe dus soms dingen anders dan in het boek, omdat ik eigenwijs ben of omdat ik de illusie heb wat jullie lastig vinden en wat vaak op het examen komt. Soms doe ik dat niet, en houd ik me trouw aan het boek. Jullie hebben handvatten aan het boek, dus ik moet weer iets nieuws maken als ik me niet aan het boek houd. Voor de PowerPoint kijk ik of er demonstratiemateriaal is en ik zorg dat ik weet wat ik wil gaan vertellen. Toen ik voor het eerst kwantummechanica moet uitleggen in V6, dan verdiep ik me erin. Dan pak ik mijn studieboeken erbij. Meestal heb je aan de uitleg genoeg, en vertel je iets meer dan je moet weten, maar als er een moeilijke vraag komt moet ik die ook kunnen beantwoorden, vind ik. Dus dat doe je dan extra. Er komt echt wel eens een vraag die ik niet weet, maar die zoek ik dan op, maar dat gebeurt niet zo vaak.

V: Wat zijn goede keuzes die Systematische Natuurkunde maakt?

A: Leuke vragen, jongens! Er zijn veel methodes natuurkunde, SN is een beetje vanuit wiskundige grondslag gebouwd, SN is heel erg recht: dit is de theorie, dit is een voorbeeld, hier is een opgave.

C: Heel erg straight to the point.

A: Ja, je hebt ook methodes die zeggen: dit zie je in de natuur gebeuren, hoe zou dat kunnen? Ga dat maar verklaren, de conclusie is dan: dit is de theorie. Dat is niet mijn stijl, ik ben er wel mee opgegroeid, maar is niet mijn stijl. Ik wil eerst: wat is de theorie? Daarna gaan we oefenen. Niet zelf uitzoeken wat er gebeurt, dan kan je er ook komen, maar het heeft niet mijn voorkeur. Je moet gewoon veel sommen maken. 8 jaar geleden zijn ze naar minder opgaven gegaan in het boek, daar is toen best wat commentaar op geweest en in jullie editie zit een redelijke hoeveelheid sommen. Maar ze hebben best wel wat geschrapt, maar dat hadden ze niet moeten doen.

V: Vindt u dat er te weinig opgaven in het boek zitten?

A: Ik vind eigenlijk dat er te weinig makkelijke opgaven in zitten, instapopgaven. Sommigen vinden dat niet zo heel erg, sommigen hebben dat wel nodig: gewoon vaardigheid opbouwen met drie makkelijke sommetjes en dan door. Sommige opgaven zijn echt wel moeilijk en dan moeten jullie lekker aan de bak. Is ook wel goed voor jullie, lekker oefenen met examensommetjes. Online zijn ze verbeterd met de toets A, toets B en zelftoets. Dat zou ik structureler kunnen gebruiken, maar is best wel omslachtig, je kan niet makkelijk koppelen aan een klas en zo. Maar iedereen die het wilt gebruiken kan dat doen. Dus dat vind ik fijn, en ook dat er zelf feedback wordt geleverd, dat scheelt mij weer tijd en jullie kunnen wel gewoon weer met tien extra sommetjes oefenen voor het hoofdstuk.

V: Wat zijn dingen die u mist van het boek. Meneer Heerkens miste de practica, wat mist u?

A: Ja dat is er wel eentje die je mist. Daardoor hebben jullie ook minder practica, dat is misschien een luie reden, maar je moet zelf het kabinet in of gaan googelen voor practica en onze tijd is niet zo ruim. Online kan het best wel versterken, iedereen maakt een zelftest en slaat dan misschien wel en misschien niet de eerste paar opgaven over. Dat gepersonaliseerd leren is in opkomst, dus over twintig jaar krijg je een maatwerk diploma, met vwo-natuurkunde en HAVO Engels. Dat wordt veel normaler, met veel meer individuele leerroutes, daar zie je de onderwijswereld langzaam naar toe gaan. Om dat goed in de les te krijgen, dat vind ik lastig. 1 les voorbereiden is makkelijker dan 30, of 3. Ik heb wel eens hele hoofdstukken de klas in blokken verdeeld, dat vindt de klas wel leuk, maar is ook heel intensief. Ik heb liever 10 opgaven extra in het boek die ik kan schrappen dan dat ik extra blaadjes moet geven, je neemt dit boek mee, ook al zitten er 30 bladzijdes meer in.

V: Dus, eigenlijk is het te algemeen?

A: Ja, ik wil gewoon meer makkelijke opgaven met de eenheden en verbanden. Je kan zeggen, dat is flauw, en bij HAVO zou ik het wel meer willen zien, maar ik denk dat het handig is. Ik denk dat er online best wel wat te winnen valt, maar ik denk niet dat het klassikaal handig is. Ik kan nu zeggen tegen leerlingen: kijk maar online, daar kan je oefenen, dat is al heel fijn en een paar doen dat, de meeste niet, prima. Als jij een acht hebt voor je toets, maakt het me echt niet uit hoeveel je geoefend hebt. Ik denk wel dat er in de komende jaren meer te winnen is.

V: En vindt u dat het boek qua opgaven goed voorbereidt op het eindexamen?

A: Ja, zeker op HAVO. Het VWO heeft de laatste jaren heeft een omslag gemaakt, dit jaar ook weer naar andere vragen. Met "leg uit", en "licht toe" en "wat gebeurt er als". SN is wel wiskundig, en je moet dus veel rekenen. Ik zeur in de klas best veel op "leg uit"; oefenen dat is, probeer het goed op te schrijven, dat moet ik ook wel doen, want anders dan leren jullie het niet. Voor het eindexamen moet ik daar altijd extra op hameren, dat is een vaardigheid waarop ik extra focus, omdat ik weet dat het nodig is. Qua stof, zit alles erin, sommige methodes hebben een hoofdstuk examenvoorbereiding, dat heeft het boek niet.

V: Ja, want examens zijn op een bepaalde manier gemaakt en dan is het fijn als je, daar ook de structuur van kent.

A: Precies, daarom doe ik examenopgaven in de toets, of het SE of in de les, omdat je het moet leren.

V: Wat is iets wat u de afgelopen jaren geleerd heeft tijdens het lesgeven, zoals bij het gedifferentieerd lesgeven?

A: Ja, ik had drie groepen: team blauw, groen en oranje. Die hadden de normale hoeveelheid instructie, meer of minder en het hele verhaal met meer opgaven en minder et cetera. Als je dat gedaan hebt, is het ook voor twee jaar vastgelegd, dan kan je dat opnieuw doen. Wat ik merk is

dat er, ook als je een goede leerling bent, het goed is om te zien hoe een leraar een opdracht uitwerkt. Uiteindelijk zat iedereen die al door mocht werken, toch stiekem naar de uitleg te luisteren. Helemaal prima, schuif dan een team op. Maar dat wilden ze niet. Wat ik niet goed vind, is als je zegt: ik heb cum laude, dan hoef je niet te komen, dat mag je doen, maar dan mis je vaak welke toelichting je moet geven. We merkten dat die leerlingen lager gingen scoren op hun laatste SE en CE. Leerlingen hebben bij natuurkunde veel lessen nodig. Ik heb ook wel eens gedaan: ga maar ontdekken wat er gebeurt. Ja, dat heb ik gedaan, en ook maar 1 keer. Daarna ga je namelijk alsnog vertellen hoe het zit. De leeropbrengst is niet zo groot, ja, je hebt een keer een snoetje in je handen gehad. Dat is winst, maar verder niet. Meestal, krijg je daardoor misconcepten, als het verkeerd zit, krijg je het niet meer goed. Als het eenmaal verkeerd in je hoofd zit, krijg je het niet meer recht. Terwijl, als je nog niets in je hoofd hebt zitten en je vertelt het in 1 keer goed, dan gaat het bij 90% goed. Maar als je zegt: zoek het zelf uit, dan zit het nog maar bij 60% goed, en ben je dus 40% kwijt. Daarom heb ik de illusie dat ik weet wat ik moet vertellen, dat is echt niet altijd goed. Maar uiteindelijk leren jullie het wel, en daarom ben ik hier.

V: U geeft geen onderbouw, toch?

A: Nee, dit jaar niet eens een V4.

V: Nee, dus u krijg altijd leerlingen die al de basis van natuurkunde hebben gezien. Wat doet u dan tegen die misconcepten?

A: Ja, heel hard hameren. Die misconcepten zitten vooral bij elektriciteit, een beetje bij krachten en afstand. Heel vaak zeggen dat het niet goed is en hoe het wél moet. Ik weet nu ook wat jullie moeilijk vinden, dan kan ik daarop extra focussen. Ik stamp het erin, ik noem wat jullie fout doen, ik probeer dat elke les te zeggen en zo hoop ik op de toets dat bij niemand het fout doen. Het mag dan wel een misconcept uit de derde klas zijn; dat is gewoon hameren.

V: Dus dat is ook weer zo een plek waar makkelijke opgaven helpen?

A: Ja, dat helpt heel erg. Als je vraagt dit is de stroomrichting, waar gaan de elektronen heen? Driekwart denkt dan: ja, de andere kant. En een kwart denk: wat een rare vraag, dat is toch dezelfde richting. Die worden gelijk gecorrigeerd door de mensen om hun heen. Wanneer ze het doen in een toepassingsvraag: dan had je het eerder moeten oefenen. Ik hamer dus.

V: Wat vindt u van projecten, waarin je langere tijd bezig bent in een groepje met stof, maar dan op een andere manier?

A: Dat is heel goed. We zijn in V4 gestart met elke vrijdagmiddag 2 uur practicum van biologie, scheikunde of natuurkunde. Dat vind ik heel fijn en dat ontbreekt aan tijd in het lesprogramma, om het niet voor te koken. Normaal heb je driekwartier om het hele practicum te doen, opstelling, meten, verwerken, rekenen, opruimen, naar buiten. Ik weet wel wat eruit komt, ik hoef geen verslag te zien. In de onderbouw moet je een keer een verslag maken. Maar als je een keer twee uur de tijd hebt, kan je ergens induiken en zeggen: onderzoek het maar, kijk maar waar je op uitkomt. En dan ga je nadenken met elkaar, dat is super goed. Dat kan je ook met de stof doen, met opgaven, met practica of met duurzaamheid ofzo. Daar hebben we gewoon geen tijd voor.

Als we dat wel hadden, had ik het gedaan. Wij zitten bij ons op school, krap in de tijd, dus ik richt me op wat jullie moeten weten en dat gaat goed. De bredere onderzoek vaardigheden ontbreken: jullie kunnen het prima, je doet een PWS, je hebt dit voorbereidt, maar echt een natuurkundig onderzoek opzetten, dat ontbreekt.

V: Ja, en dat is gewoon jammer.

A: Ja, maar als ik het wel doe, moet ik de hoofdstukken sneller doen, en voor sommige is dit tempo al hoog. Dus ik zou er liever meer tijd in steken, dus als er tijd extra is staat dat soort dingen op het lijstje. Maar nu niet.

V: Dus alleen in V4.

A: Ja, ik wil het doortrekken naar V5 en daarna PWS, dat je daar vrijdagmiddag aan kan werken. Dat de TOAs er zijn, en dat je daaraan kan werken, maar dat kost ook allemaal tijd en geld.

C: Bij scheikunde hebben ze het wel voor elkaar gekregen.

A: Ja, bij scheikunde doen ze wel practica in de lessen, dat doen ze best wel goed. Zij hebben het weten om te draaien, om die manier de stof uit te leggen. Dat kan voor sommige delen van natuurkunde ook, maar dan verlies je je structuur. Ik denk dat als je natuurkunde wilt begrijpen, je sommetjes moet maken. Maar bij scheikunde heb je onderzoek vaardigheden die je moet kunnen. Bij natuurkunde heb je het iets theoretischer. Ze hebben het goed gedaan, een goed voorbeeld, maar bij natuurkunde kan het niet helemaal en dan verlies je je structuur.

V: Op welke manier zorgt u ervoor dat alle leerlingen erbij blijven tijdens de les?

A: Haha, goede vraag. Ja, niet eigenlijk. Ik probeer het door iets nuttigs te vertellen als ik aan het woord ben. Gestructureerd een verhaal aanbieden, ik vind dat als jij als docent een goed verhaal hebt, dat er dan ook naar je geluisterd wordt. Je hebt de oude docenten truc, maar de basis is een goed verhaal hebben. Als jij eindelijk iedereen stil hebt en je gaan een slecht verhaal ophangen, daar prikt iedereen zo doorheen. Maar als jij een verhaal vertelt, moet het zinvol zijn. Daar begint het bij en dan mag ik ook zeggen: nu ga ik iets zeggen, jij houdt je mond. Niet iedereen gaat daarna keihard opgaven maken, maar ik probeer de vragen van mensen goed te beantwoorden, zodat je verder kan en dat je er iets van leert. Ik vind mezelf niet een strenge docent. Als je wilt leren, moet je bij mij terecht kunnen, dus als je aan de slag gaat heb je iets aan mij. Dan is het zinvol om in de les te zijn, dat is belangrijk. Want sommigen kan ik een boek geven en halen anderhalf jaar later een acht voor hun examen, dat kan, dan heb ik geen toegevoegde waarden. Maar ik ga de les in met de gedachte dat ik wél van toegevoegde waarde ben. Hoe houd ik iedereen erbij? Zorgen dat ik een verhaal heb, ik probeer zoveel mogelijk te spreken. Ik wil voorkomen dat er een leerling is die aan het eind van de dag met geen enkele docent heeft gepraat. Dan doe je als school iets verkeerd. Ik ben ook niet zo een prater, maar ik kan me voorstellen, dat als niemand iets zegt, dat dat raar is. Ik probeer geïnteresseerd te zijn in hoe het gaat en hoe je leert. Ik hoef niet te weten wat je op dinsdagavond doet met wie en waarom en wat je eet, maar als het niet goed gaat, wil ik voor jullie klaar staan.

V: Dat was het dan. Dank u wel.

A: Helemaal goed.

Interview with Kees van Vliet

V: Als eerste hadden we de vraag, op welke manier geeft de structuur aan uw les?

K: Nou, wat ik eigenlijk altijd doe, dat weet je misschien wel van vorig jaar is dat ik een soort PowerPoint hebt en dan heb ik aan de linkerkant heb ik daar eigenlijk altijd het programma staan, dus de linkerkant van mijn PowerPoint geeft het programma aan. En ik denk dat dat voor leerlingen fijn is dat ze een beetje zien wat er komt, maar het is gewoon voor mezelf ook al fijn. Dan vergeet ik tenminste niets. Dus het helpt mij zelf ook om de structuur erin te houden. En het helpt me ook een beetje bij het voorbereiden, want daar denk ik dus altijd wel heel erg na over wat zijn dan echt de onderwerpen van deze les? Wat zijn de dingen waarvan ik wil dat het echt blijft hangen, zeg maar, want dat zijn dus de kopjes die ik dan gebruik. Dat is eigenlijk een beetje de basis van de structuur.

V: En wat zijn vaste onderdelen die elke les terugkomen?

K: In principe de start ik eigenlijk altijd, maar daar ben ik vorig jaar volgens mij halverwege mee begonnen maar met een, soort terugblik. Dus, wat hebben we vorige les gezien en een paar vragen daarover die ik dan wat willekeurig stel. En ja, dat is natuurlijk altijd, maar dat heeft heel erg met het vak te maken, er zit een stukje uitleg in. En dan het bespreken van vragen of het oefenen of nou ja, zelf aan de slag gaan, zeg maar, maar vooral ook het zelf aan de slag gaan is juist ook bij programmeren denk ik, heel belangrijk en dat is het bij wiskunde ook.

V: Ja, u ruimt daar ook expres tijd voor in de les.

K: Ja, ik probeer dat eigenlijk, maar zeker in een bovenbouw lukt dat niet altijd, omdat daar soms als je vragen gaat bespreken, en soms kost dat gewoon meer tijd. Maar ik probeer eigenlijk dat je altijd wel de helft van de les zelf kan werken, dus dat je wel echt gewoon de helft van de tijd zelf aan de slag kan. En dan nog meer vragen daarover kunt stellen, want je kunt ook wel dingen als huiswerk opgeven. Maar zo weet je nooit of het gedaan wordt het niet. En als je thuis met je huiswerk vastloopt, kun je geen vragen stellen, terwijl juist in de les, als je er als docent bij bent, dan kun je natuurlijk iemand weer verder helpen.

V: Ja dus u introduceert eerst een beetje het onderwerp en hoe zullen we uiteindelijk mee aan de slag gaan? Dat gebeurt echt in het maken van de opgaven. Als er dan vragen zijn, dan kunt u daarbij helpen. Kiest ervoor om juist heel veel opgaven te maken, zodat leerlingen veel oefenen om een beetje het minimum te hebben, zodat ze het in ieder geval een paar keer gezien hebben, maar zodat ze misschien wel het huiswerk in ieder geval maken.

K: Dat is natuurlijk altijd een beetje een balans zoeken. En in de laatste jaren doe ik eigenlijk allebei, dus dit moet je in elk geval wel maken, maar als je wilt oefenen, kun je deze vraag nog extra maken en wat ik de laatste jaren ook een beetje probeer om te doen is om eigenlijk. Nou ja, vroeger deed ik altijd gewoon les 1 in de week een onderwerpje en vragen maken, les 2 een onderwerpje, vragen over maken; les 3: onderwerpje, vragen over maken. Nu probeer ik vaak om

al in de eerste les eigenlijk al die onderwerpen even te doen en dan van elk onderwerp een vraagje en dan in les twee weer van elk onderwerp een vraag. En in les 3 weer elk onderwerp een vraag, zodat je het eigenlijk al 3 keer gedaan hebt, want anders heb je het ene onderwerp 1 keer geoefend, het andere onderwerp 1 keer geoefend. Qua hoeveelheid vragen maakt het dus niet uit, maar je bent er vaker mee bezig geweest en dan blijf het gewoon beter hangen.

V: Ja, je bent er steeds iets korter mee bezig, waardoor uiteindelijk beter blijft hangen. Ja en u bent met dat steeds stukjes en dat dan weer samenvoegen bent ook bezig geweest met een collegezaal, dat hadden we op een gegeven moment op de Heemskerk, wat was daar een beetje het plan van en hoe is dat gegaan?

K: Ja eigenlijk wel een beetje hetzelfde. En, ik vond dat dus wel heel fijn werken, want zo een college zorgt ervoor dat ik dat echt heel goed voor moest bereiden, omdat je dan echt heel goed na moest denken van hoe komt dit over en hoe kan ik dat doen? Dus dat was voor mij een goede stok achter de deur. Maar ik in dat college gaf ik dus de uitleg voor de hele week. Het huiswerk na dat college was heel weinig, maar gewoon van elk onderwerp één vraagje, gewoon even om het te verwerken, zeg maar van wat we nu gedaan hebben. En dan daarna in de eerste les na het college kregen ze gewoon van elk onderwerp een paar vragen en in het tweede les weer van elk onderwerp een paar vragen en voor de keuze les stond dan nog weer van elk onderwerp een paar vragen.

C: Werkte dat goed?

K: Het wisselde een beetje, wat ik heel erg merkt, tenminste. Ik had vorig jaar bijvoorbeeld twee vwo 5 wiskunde A groepen, maar die had ik allebei en die gaf ik dus ook college en dat werkte heel goed. Maar we hebben ook wel eens gehad dat ik bijvoorbeeld, toen nog met meneer B., dan hadden wij allebei een groep en dan gaf hij bijvoorbeeld eerst half jaar college, een ik het tweede half jaar, maar dan was het lastiger omdat je dan zelf niet zo goed met het onderwerp bezig was als je geen college gaf en dan in de les kwamen leerlingen en dan weet je eigenlijk niet zo goed waar je precies mee bezig bent. De band met de leerlingen was dan ook wat anders. Dus het werkte wel heel goed, maar eigenlijk alleen als je het helemaal zelf in de hand had.

V: Ja, want het college was dan aan iedereen die dat vak had, terwijl de lessen waarin je vragen kon stellen, die was met je eigen docent, dus het kon zijn dat je dan elke week uitleg kreeg van een andere docent? Ja, dus dat is natuurlijk ook als docent als je vragen moet gaan beantwoorden een beetje vervelend? En dan ben je heel lang bezig met de leerlingen leren kennen.

K: Dat was inderdaad het moeilijkste dat ik dan dacht: ik weet eigenlijk niets eens hoe jij heet. En normaal weet ik dat na twee weken. Maar ik vond het dus wel heel goed werken als ik het zelf deed. Leerlingen waren ook tevreden over, ook omdat ze gewoon wel een duidelijke weektaak hadden en dus omdat de onderwerpen gewoon een paar keer behandeld werden. Het hoor college is natuurlijk heel rigide, daar kon ik niet echt inspelen op als jij al wat meer snapt dan, dat was gewoon echt 50 minuten luisteren. Ja, Ik deed wel opdrachtjes tussendoor, maar de andere lessen konden in principe iedereen gewoon zelfstandig aan de slag was dus ook veel tijd om vragen te stellen.

V: En doen jullie het hoorcolleges nog steeds?

K: Nou nu, dit jaar mocht het niet vanwege corona, omdat de kans bestond en dat is uiteindelijk ook gebeurd natuurlijk, dat er minder leerlingen per lokaal mochten. En dan waren die groepen gewoon te groot dus dit jaar kon het niet. En volgend jaar denk ik dat het ook niet zal zijn, omdat hier op school zeg maar in dit gebouw is niet zo'n grote ruimte. Ja, op de mediatheek of zo, maar dan moet dat elke keer weer apart ingericht worden.

V: Dus het zijn eigenlijk vooral de praktische redenen die jullie daar nu van behoeden?

K: Ja klopt. Ik ben er zelf verder op zich wel voorstander van, maar wel ook alleen in VWO 5 en 6. Als je naar de universiteit gaat, krijg je dat ook op die manier les, ik vind dat je daar best wel aan moet kunnen wennen. Ik denk ook dat leerlingen in VWO5 en 6 dat wel kunnen, maar je kan dat bijvoorbeeld niet in een HAVO 2 doen. 50 minuten zo intensief bezig zijn is dan gewoon te veel.

V: En, is dat iets wat eigenlijk vooral bij de wiskunde past op de middelbare school?

K: Nou, geschiedenis en Engels hebben we ook al van die colleges gegeven, volgens mij natuurkunde ook dacht ik. Maar dat was vooral ook het eerste jaar, toen kon je gewoon aangeven dat je daar wel mee wilde experimenteren. En bij geschiedenis bijvoorbeeld, deden ze het vooral ook omdat geschiedenis zo breed is en dan had meer bijvoorbeeld de ene docent wat meer kennis van de Koude Oorlog en de andere wat meer kennis van iets anders en dan verdeelde ze het dus een beetje wie welk onderwerp uit zou leggen. En bij Engels deed het vooral bij literatuur, omdat het dan toch in één keer heel veel is en dan konden ze gewoon in minder lessen de uitleg geven. Maar het wisselt dus een beetje. Ik denk dat het voor elk vak wel nuttig zou kunnen zijn, maar het hangt wel een beetje van het vak en onderwerp af. Bij wiskunde wilde ik er mee bereiken dat ik gewoon de stof vaker langs kon laten komen. Terwijl geschiedenis het om een andere reden deed en Engels weer om een andere reden. Ik denk ook dat als elk vak het zou doen, zou je als leerling er helemaal gek worden. Want je wilt eigenlijk die colleges in het begin van de week hebben zodat je de rest van de week kunt oefenen. Maar je kunt ook moeilijk op maandag en dinsdag gewoon achter elkaar alleen maar colleges hebben, daar word je ook niet blij van.

V: En dan zijn woensdag donderdag, vrijdag, eigenlijk de dagen dat je net zo goed gewoon thuis je opgave kan gaan maken en je daar de hele dag een beetje zit voor...

K: Ja, je ziet, er zitten wel wat haken en ogen aan, maar het zou voor elk vak wel ergens moeten kunnen denk ik.

V: En als u dan een uitleg geeft, op welke manier bouwt u dan op? Dus u begint met een terugblik en dan?

K: Ja, want ook dat ik wel een beetje van het onderwerp af. Als het een heel nieuw onderwerp is dan begin natuurlijk gewoon eigenlijk vanaf het begin, maar meestal zeker bij wiskunde nu in de

bovenbouw. Haakt het bijna altijd wel ergens op aan. Ja die vectoren, waar jullie nu mee bezig zijn, dat is dan wel weer iets echt nieuws. Maar haakt toch ook wel weer ergens een beetje bij aan. Maar dan probeer je toch vooral even te kijken: welke voorkennis is er? Want er is natuurlijk een bepaalde voorkennis nodig, maar is die er ook bij iedereen? En daar borduur je dan een beetje op voort van: dit zijn de dingen die je al weet. O ja, en heel vaak is het zo van o dit weet je van het ene, dit weet je van het andere en als dat samenvoegen, dan hebben we weer een nieuw iets.

V: Ja, en probeert u het steeds heel theoretisch te houden. Of ook dat u het kort theoretisch uitlegt en vervolgens doet u een opdracht voor zodat iedereen ziet hoe het zit.

K: Ja, dat laatste doe ik ook wel eens, maar ook dit verschilt weer. Bij wiskunde B houd ik het theoretischer, omdat ik vind dat je als leerling van theorie naar praktijk moet kunnen gaan, zonder dat het voorgekauwd is, maar ik heb ook twee HAVO 2 klassen, waar ik een opdracht voor doe en dan doen ze dezelfde opgave met andere getallen, zodat ze het kunnen kopiëren, maar dat is voor VWO 5 Wiskunde B doe ik dat niet.

V: Dus de vorm van de uitleg hangt af van de klas die u lesgeeft?

K: Ja, en ook van het vak. Wiskunde B is vereist meer inzicht dan Wiskunde A, ook in de uitleg.

V: Hoe bereidt u uw lessen voor?

K: Dat hangt af van hoe druk ik het heb. Voor een periode maak ik een planning: welke hoofdstukken komen er aan bod, welke vragen maken we. En dan weet ik voor een les welke opgaven leerlingen krijgen als huiswerk en dan ga ik kijken welke onderwerpen daarbij horen en dan ga ik denken hoe ik dat kan uitleggen en dat verwerk ik in een PowerPoint. Het belangrijkste voor mezelf is dat de onderwerpjes duidelijk in de PowerPoint staan en dan ga ik kijken: wil ik er plaatjes bij, of vergelijkingen of tekst etc. En dan maak ik de PowerPoint.

V: En maakt u die elk jaar opnieuw?

K: Dat wisselt, ik pas het soms ook gewoon aan. Toen ik met die presentaties begon, vier jaar geleden. Het jaar daarna gebruikte ik voor hetzelfde college dezelfde PowerPoint, alleen dan was ik vergeten waarom ik het zo had gedaan. Er was dan wel een goed idee, alleen ik was in mijn uitleg even vergeten hoe en wat. Sindsdien maak ik de PowerPoint opnieuw, soms met veel kopiëren hoor, maar dan herhaal ik voor mezelf wat er allemaal in zit.

V: Ja, dat je niet zelf verstoelt staat van je goede ideeën van twee jaar geleden?

K: Nee, precies.

V: Dan even over het boek: wat vindt u fijn aan Getal en Ruimte?

K: Ik denk dat het fijn is dat ze heel veel uitleg geven met uitgebreide uitleg. Dat zorgt ervoor dat leerlingen zelfstandig ermee aan de slag kunnen gaan. Een goede leerlingen kan zonder docent

het boek uitwerken. Dat vind ik er goed aan, maar dat is ook gelijk het grootste nadeel. Ze leggen soms te uitgebreid uit en ze kauwen heel veel voor terwijl je zelf dingen moet gaan proberen, dat is soms ook goed.

V: Ja want ik hielp wel eens iemand van een andere school en daar gebruikte ze de Wageningse methode, wat me opviel is dat ze heel veel tekst gebruiken. Bij G&R is het: dit is hoe het zit, bam, bam voorbeeld, gaan. Als je dan naar een stuk uitleg kijkt dat het alleen maar wiskunde is, en dan af en toe het woordje “dus” ertussen. Dat als ik het zit te lezen ik ook denk, ja hallo, zeg het even in een zin tegen me.

K: Ze doen bij G&R vaak een abstracte introductie en dan een voorbeeld en dat is het. Precies.

V: Het is zo dat ze bij G&R heel goed nadenken hoe je een uitleg een beetje zelf ontdekt met die oriëntatie-opgaven, maar die maken we nooit.

C: Zijn dat die opgaven voor een theorie blok?

V: Ja. Er staat dan ook in de uitleg: “in opdracht 9 heb je bewezen dat” en dan denk ik: ja leuk, maar ik niet. Is dat overslaan om tijd te besparen?

K: Ja, en het zijn vaak een beetje vage vragen, dus leerlingen lopen er vaak op vast. Het zijn geen vragen die je ooit op een toets zou zetten. Ik gebruik zelf vaak die O-opgaven in mijn uitleg, dat zie je misschien niet, maar de opbouw in zo een opgave, gebruik ik dan weer. Met die vectoren nu, had je dat je de hoek tussen vectoren moest bewijzen met de cosinus. Daar had meneer Ypma een mooi ding gemaakt met het omschrijven van wat dingen en dan kon je het bewijzen.

V: Ja, dat deden we ook inderdaad, en ik zat daar echt 45 minuten van: wat gebeurt hier?

K: Ja, ik zag dat wat hij gedaan had ook, maar vond het wel mooi, maar te ingewikkeld. Nou, dat blijkt dus. Dat is bij die O-opgaven ook vaak, maar ze helpen me wel om de uitleg vorm te geven.

V: Ik denk dat G&R goed op zichzelf staat. Het is daarvoor gemaakt, in tegenstelling tot de Wageningse methode.

K: Ja, bij Moderne Wiskunde staat er geen uitleg in het boek, dat kan ook nog. Je hebt er eigenlijk drie in Nederland. G&R is het grootst, daarna Moderne Wiskunde, daarna de Wageningse methode. Andere scholen hebben dan weer eigen methodes.

V: Alleen wat ik merk bij G&R is dat als je docenten er net een ander plan langstrekken, dan heb je niets aan de theorie, terwijl ik de opgaven wel kan.

K: Ja, maar dat is op zich ook niet erg.

V: Behalve als je het boek als naslagwerk gebruikt. Wat mist u van G&R?

K: Ik denk dat ik dat overal wel een beetje mis, is dat het soms de basis een beetje mist. Ze hebben heel weinig makkelijke opgaven. Daar moet je er eigenlijk wat meer van hebben, zodat je daarna door kan naar die moeilijke vragen. Ik mis gewoon basisvragen. Bijvoorbeeld bij raaklijnen aan cirkels met de afstandsformule. In het boek zit 1 vraag waarin je snapt hoe het moet. Bij de andere vragen moet je eerst zelf drie stappen zetten voordat je kan gaan beginnen aan de vraag.

V: Heeft het boek verder genoeg opgaven?

K: Ja, wel genoeg, maar van een te hoog niveau. Die denk-opgaven zijn soms ondoenlijk, terwijl er te weinig opgaven zijn om jezelf te trainen zodat je weet dat je het kan.

C: Dus meer makkelijke opgaven?

K: Ja, eigenlijk wel.

V: Ik denk ook dat het, naar mate je langer op school zit, het minder wordt. Ik zit nu even aan de eerste te denken, daar ging vraag 3 tot en met q of zo en moest je die allemaal maken.

K: Ja, alleen in bovenbouw VWO heeft niet iedereen het nodig, dus het zou wel fijn zijn als je dat aan materiaal had, dan kan je het geven aan leerlingen die het lastiger vinden. Nu heeft meneer Ypma het zelf gemaakt. Want bijvoorbeeld 10.4 met dat pad met vectoren. Die opgaven in het boek zijn echt heel lastig, dus als je dan eerst de opgaven uit zijn boekje maakt, die zijn wat basic, dan kan je daarna door naar het boek.

V: Wat me ook opvalt, is dat de examenopgaven helemaal niet lijken opgaven uit het boek. Behalve dat je dezelfde wiskunde gebruikt is de manier van vragenstellen heel erg anders. Heel vaak is het in het boek “Los op.”, terwijl dat bij een examenopgave niet zo is. De wiskunde is dan even ingewikkeld, maar je moet daar gaan bekijken wat je moet doen. Dat is in het boek niet zo.

K: Ik denk dat, hoe verder je komt vanaf nu, je ziet dat de vragen er meer op gaan lijken. Alleen wat je ziet bij wiskunde is dat je in het begin veel technieken leert, maar op het eindexamen wordt nooit los een techniek gevraagd. Als je naar Hoofdstuk 9 kijkt over de logaritmen, dan krijg je rijtjes vergelijkingen, dat zit niet in het examen. Maar dat vind ik dus wel goed, want dan laten ze je ermee oefenen. Vervolgens komen die lastigere vragen met een verhaaltje.

V: Dus zijn er wel genoeg lastige examenopgaven in het boek?

K: Ja, dat wordt gewoon steeds meer. Dat zal je nog wel merken. In deel 4 zitten veel meer van dat soort vragen. Hoofdstuk 16 is zelfs helemaal examenopgaven. Maar het eindexamen is een eindpunt en om dat te kunnen moet je eerst technieken beheersen.

V: Ja, wiskunde is natuurlijk anders dan vakken die bestaan uit losse onderwerpen zoals natuurkunde of biologie.

K: Ja, wiskunde is meer een huis, waar je op stapelt. Als de onderkant niet zo goed is, dan stort het huis in elkaar. Bijvoorbeeld met sinussen en cosinussen, daar gaan we nu mee verder met bewegingsvergelijkingen, dan moet je wel de basis beheersen. Maar op je examen komt nooit: los deze vergelijking op, maar wel bereken het snijpunt van dit en dit. Dan moet je zelf bedenken dat je een vergelijking moet oplossen om daar te komen.

V: De volgorde van de hoofdstukken, want we doen het helemaal niet op volgorde. Waarom?

K: Toen jullie in de vierde zaten waren we net begonnen met 100 minuten toetsen, met 1 toets per periode die je mocht herkansen. En dan wilden we per periode 1 onderwerp doen. En dat kon soms zijn dat we hoofdstukken van hetzelfde onderwerp bij elkaar pakten, omdat je meer dan één hoofdstuk per periode doet. Dat heeft voor- en nadelen: het is goed dat je het in zijn geheel ziet, maar je hebt het dan ook maar één keer gezien en daarna een jaar lang niet meer. Terwijl als je ze losdoet, komt het een paar keer terug. We proberen toch wel te groeperen, dus Hoofdstuk 9 en 11 borduren op elkaar voort, terwijl het wel verschillende dingen zijn. Bij 9 ben je aan het differentiëren, maar bij 11 aan het integreren. Maar wel allebei met e-machten en logaritmen. En bij schoolexamens proberen we ook onderwerpen bij elkaar te doen, dat je niet vijf verschillende onderwerpen aan het doen bent op 1 SE.

V: Dus wat dat betreft heeft het boek een andere keuze gemaakt.

K: Ja, ik snap de keuze van het boek wel, maar als je dan bezig bent denk ik: o, dit hoort er nog bij.

V: Ja want we gaan nu ook opeens naar 12.4.

K: Ja en Hoofdstuk 14 lijkt daar ook weer heel erg op. 10, 12, 14 horen bij elkaar, en dan is het wel aardig om daaruit wat dingen te pakken.

V: Zijn er nog andere dingen die u mist van G&R?

K: Ja, in de nieuwe methode hebben ze programma's. Als je goed bent, kun je deze opgaven overslaan en als je er moeite mee hebt kun je hiermee extra oefenen. Alleen ik heb daar even naar gekeken, die programma's schelen maar een paar vragen, omdat ze niet zoveel oefenmateriaal hebben.

V: Ja, alleen je moet daar ook wel mee opletten. Want het kan ook zo zijn dat de leerlingen die het lastig vinden dan niet de moeilijke opgaven maken en dan hebben ze niet het goede niveau bereikt.

K: Ja, daar moet je op letten inderdaad. Nee, met die denk-opgaven bijvoorbeeld, als je het lastig vindt, moet je dan ook die vragen maken die het verschil maken tussen een negen en een tien?

V: Op welke manier houdt u de aandacht van leerlingen erbij tijdens een les?

K: Goede vraag, haha. Door af te wisselen. De wiskundelessen zijn dan niet erg opwindend, met allerlei werkvormen en zo, maar ik merkte gewoon dat leerlingen dat niet fijn vinden. Een beetje afwisseling. Dat maakt het wat meer behapbaar.

V: Wat is in de afgelopen jaren iets wat u geleerd hebt tijdens het lesgeven?

K: Ik denk dat ik dingen toe ben gaan voegen. Nou ja, oké. Vroeger leerde ik leerlingen echt zelf een onderwerp ontdekken, dat doe ik niet meer. Omdat ik merkte dat leerlingen helemaal de verkeerde kant op gaan, en als ze dan iets verkeerd hebben geleerd, is het lastig om dat weer goed te krijgen. Dat doe ik dus niet meer. Ik leg eerst alles uit, en dus niet meer: ga maar aan de slag. Wat ik de laatste paar jaar geleerd heb, is de kracht van herhaling. Daarom deel het nu dus anders in. En ook die terugblikken, om dat maar terug te laten komen. In de 4^e bij het hoofdstuk over differentiëren, ik geloof dat ik tien keer de vraag had bij de terugblik: wat zegt de afgeleide? En dan vroeg je het in les 9, dan was het al 8 keer aan bod geweest, en dan wisten ze het nog niet. Vroeger had ik dat niet door, dan dacht ik: het is allemaal helder, het komt goed. Maar nu weet ik dus dat het niet altijd landt, of anders landt dan je verwacht.

V: [tegen C] Ik weet niet of jij nog vragen hebt?

C: Nee, denk het niet.

V: Dat was hem dan, tenzij u nog iets wilt toevoegen.

K: Nee, denk het niet. Ik ben wel benieuwd hoe jullie dit gaan verwerken, want jullie hebben er wel goed over nagedacht hoor en verdiept in het boek en zo, leuk!

V: Dank u wel voor uw tijd.

Appendix D – Some lessons we made

You can see all the lessons we have made on our website (www.groteprogrammeer.nl), here are two sample lessons from Chapter 1 – An introduction with C. They are in Dutch, as this website is meant for Dutch students.

Les 2 - Hallo, leerling

In de vorige les hebben we gezien hoe je met `printf` stukjes tekst kan printen in de terminal. In deze les gaan we kijken hoe we input kunnen vragen aan de gebruiker, hiervoor gebruiken we een bibliotheek met functies.

De uitvoer van functies

Functies kunnen twee soorten output hebben:

- Een neveneffect: iets wat je kan zien of horen, zoals iets printen in de terminal of een geluidje afspelen.
- Een retourwaarde: een waarde die je kan opslaan om er vervolgens verder iets mee te doen.

In de vorige les hebben we een voorbeeld van een functie met een neveneffect gezien: `printf`. Maar een functie kan dus ook een retourwaarde als uitvoer hebben. Denk weer even aan de functie `optellen` die twee waardes optelt. Om het resultaat (de som van de twee argumenten) op te slaan, gebruik je de volgende regel code.

```
int resultaat = optellen(3, 5);
```

`int` staat voor integer (een geheel getal, zoals 1 of 2). `resultaat` is de naam van de output van de functie en het `=`-teken betekent dat ik de retourwaarde van `optellen(3, 5)` wil opslaan in de variabele `resultaat`. In de volgende les leer je meer over variabelen, voor nu is dit genoeg.

Gebruikersinvoer

Een programma dat de gebruiker groet is leuk, maar het is nog veel leuker als een programma reageert op de gebruiker. Hiervoor hebben we gebruikersinvoer nodig. Zo kunnen we bijvoorbeeld de gebruiker vragen om hun naam in te voeren. Dit doe je met de volgende regel code.

```
string naam = get_string("Voer hier je naam in: ");
```

Merk op dat deze regel best wel lijkt op de regel code van **Uitvoer van functies**. We hebben alleen geen `int` (geheel getal) maar een `string`, een stukje tekst; we kunnen immers iemands naam niet als getal opslaan! De naam van de variabele is niet meer resultaat maar `naam` en we roepen een andere functie op, namelijk `get_string`. Dit is een functie die 1 argument nodig heeft: de vraag die je wilt stellen aan de gebruiker. De gebruiker kan vervolgens daarachter typen en bevestigen door op Enter te drukken. De functie leest deze invoer en retourneert hem in de variabele `naam`. We hadden de variabele ook `a` of `b` kunnen noemen, maar `naam` is iets duidelijker.

Opdracht 1: Voer het programma hieronder uit. Vul je naam in als het programma er om vraagt en druk op de Entertoets. Het programma zou je nu moeten begroeten.


```

main.c
1  #include <stdio.h>
2  #include <cs50.h>
3
4  int main(void)
5  {
6      string naam = get_string("Voer hier je naam in: ");
7      printf("hallo, %s\n", naam);
8  }

Console Shell
clang version 7.0.0-3-ubuntu0.18.04.1 (tags/RELEASE_700/final)

```

Bibliotheken

De functies `get_string` en `printf` zijn geen gewone functies: ze komen uit een bibliotheek. Zoals je in een echte bleb een hele lijst met boeken hebt, zo heb je in onze bibliotheken een hele lijst met functies. Je mag deze functies gewoon gebruiken, mits je bovenin je bestand vertelt dat je ze gebruikt. Dit doe je voor `printf` door deze regel.

```
#include <stdio.h>
```

Voor `get_string` doe je het met deze regel code.

```
#include <cs50.h>
```

De `#include`-regels staan altijd als eerste in je programma. Standard Input/Output (stdio) is een populaire bibliotheek voor functies voor de in- en uitvoer van data, hierin is de functie `printf` gedefinieerd. De cs50 library is gemaakt door mensen van Harvard University en bevat wat invoer-functies die gemakkelijk zijn om te gebruiken, zoals `get_string`. De `#include`-regels eindigen NIET op een puntkomma!

Opdracht 2

Fork de Repl hieronder. Op regel 10 staat een manier van printen die je nog niet kent, het is dus niet gek dat je die regel niet snapt. Hierover leer je in de volgende les. Voer de volgende deelopdrachten uit.

- Importeer de bibliotheek `les2` (Gebruik dubbele aanhalingstekens in plaats van driehoekige haakjes).
- In het bestand staat drie keer een `0`. Deze moet je vervangen voor een functie-oproep (zoals `printf("Hallo!\n")`).

Opdracht 2

Fork de Repl hieronder. Op regel 10 staat een manier van printen die je nog niet kent, het is dus niet gek dat je die regel niet snapt. Hierover leer je in de volgende les. Voer de volgende deelopdrachten uit.

- Importeer de bibliotheek `les2` (Gebruik dubbele aanhalingstekens in plaats van driehoekige haakjes).
- In het bestand staat drie keer een `0`. Deze moet je vervangen voor een functie-oproep (zoals `printf("Hallo!\n")`).
- Zorg dat `x` en `y` allebei een waarde hebben van input van de gebruiker, gebruik hiervoor de functie `get_int`, die verder hetzelfde werkt als `get_string`, behalve dat hij de gebruiker om een getal en niet om een stuk tekst vraagt.
- Zorg dat `z` de waarde heeft van het optellen van `x` en `y`. Omdat je `les2` hebt geïmporteerd mag je de functie optellen gebruiken (met de syntaxis `optellen(3, 5)` voor de som van 3 en 5).

Deze opdracht is best lastig voor mensen die nog nooit geprogrammeerd hebben, dat is niet gek. Je hebt in de afgelopen drie lessen een hele hoop nieuwe dingen geleerd die allemaal (net) niet logisch lijken. Je gaat ervan vanzelf de logica inzien. Lukt een opdracht niet? Bekijk dan het filmpje waarin de opdrachten worden besproken, zo leer je er wat van.

open in [replit](#)

```

main.c
6   int x = 0;
7   int y = 0;
8
9   int z = 0;
10
11   printf("De som is %i\n", z);
12 }
  
```

Console Shell

```

clang version 7.0.0-3-ubuntu0.18.04.1 (tags/RELEASE_700/f1)
  
```

