****

**STUDENT MANUAL**

**(VERSION 0.1)**

Table of Contents

[Introduction](#_heading=h.gjdgxs) [2](#_heading=h.gjdgxs)

[Chapter 1: Course Overview](#_heading=h.30j0zll) [3](#_heading=h.30j0zll)

[1.1 Curriculum](#_heading=h.1fob9te) [3](#_heading=h.1fob9te)

[1.2 Homework and Preparation](#_heading=h.3znysh7) [5](#_heading=h.3znysh7)

[1.3 Tests](#_heading=h.2et92p0) [5](#_heading=h.2et92p0)

[1.4 Teachers](#_heading=h.tyjcwt) [6](#_heading=h.tyjcwt)

[Chapter 2: How to study programming   
(tips for non-programmers)](#_heading=h.1t3h5sf) [9](#_heading=h.1t3h5sf)

[Chapter 3: Study Strategies](#_heading=h.2s8eyo1) [14](#_heading=h.2s8eyo1)

[1. Planning](#_heading=h.17dp8vu) [14](#_heading=h.17dp8vu)

[2. Don’t just read, code it!](#_heading=h.3rdcrjn) [17](#_heading=h.3rdcrjn)

[3. Go back to the basics](#_heading=h.26in1rg) [17](#_heading=h.26in1rg)

[4. Sleep, Breaks & Exercise](#_heading=h.lnxbz9) [17](#_heading=h.lnxbz9)

[5. Distractions](#_heading=h.1ksv4uv) [18](#_heading=h.1ksv4uv)

[Chapter 4: How to ask Questions](#_heading=h.44sinio) [20](#_heading=h.44sinio)

[Chapter 5: Core Values](#_heading=h.z337ya) [23](#_heading=h.z337ya)

[Final thoughts](#_heading=h.1y810tw) 28

# Introduction

Dear new student of HackYourFuture,

Welcome to our program!  
  
On average, we receive 100-200 applications for every new class. Therefore, to be accepted into HackYourFuture is a big accomplishment.   
  
You are now a member of a large community. Over a 100 students have graduated from HackYourFuture and more than 90% find a job as a professional web developer.  
  
However, our course will be challenging. For the next 7 months there will be moments in which you might want to give up. It is important to remember that you are not the first to go through this. Many of our students, who are now in professional jobs, have at some point thought about quitting our course.   
  
Based on their experiences we compiled this guide to share the lessons learned: How do you make sure to be successful in HackYourFuture?  
  
This guide consists of five chapters. In the first chapter, we provide a short summary of our program. This will answer the question: What will I learn at HackYourFuture? The second chapter discusses the six steps of problem solving, the core skill of what programming is really all about. Chapter three talks about how to study programming most effectively. The fourth chapter details how you can become better at asking questions, which is vital to become a good programmer. Finally, chapter five lays out HackYourFuture’s core values.  
  
Like you, this document is always improving. If you feel something is missing or might be improved, please let us know and we might add your suggestion.  
  
 Good luck on your journey! 

**The HackYourFuture team**

# Chapter 1: Course Overview

During the next half year, you will practice with a variety of technologies and ways of thinking. This will build a solid foundation for you to become a junior web developer. What does this mean?

At the end of the 7 months you will leave HackYourFuture with:

* The skills to build your own full-stack web applications.
* Experience working in a team
* Access to the Alumni Network: a community of HackYourFuture graduates.
* A decent CV and LinkedIn profile coupled with interview skills
* Introductions to companies looking to hire programmers

In order for this to be turned into reality, you will have to work very hard for the next half a year.

In the following section you’ll get an overview of all the subjects you’ll be learning through the course. Keep in mind that

### 1.1 Curriculum

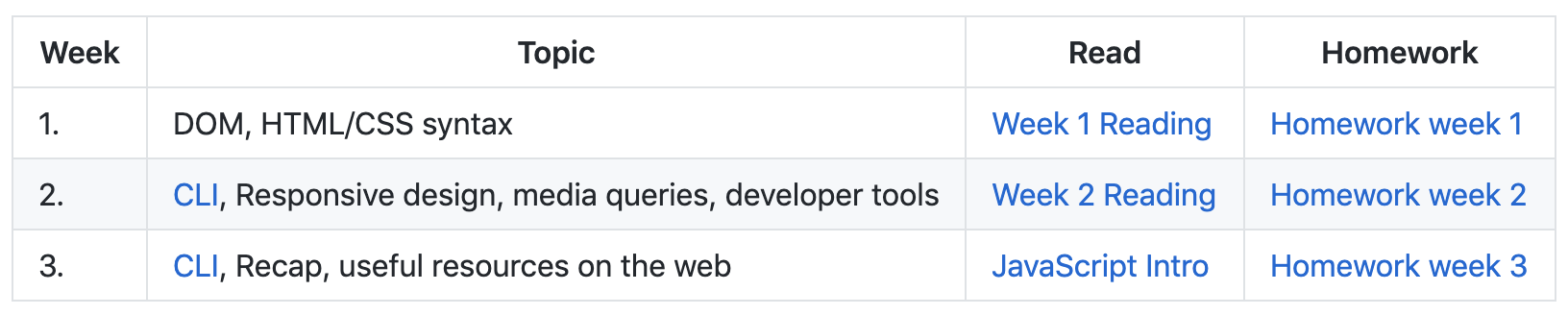
To facilitate your learning, we have developed a curriculum, which you can find here: [www.github.com/hackyourfuture](http://www.github.com/hackyourfuture).   
  
The 7 months are divided into different modules, each building on the one before. Here’s the overview:

1. HTML/CSS. This is the foundation. Every website you know ultimately consists of these two technologies. You will learn how to create simple web pages (HTML) and how to make them look nice (CSS).
2. Command Line*.* This is the direct communication channel between you, the developer, and the computer. While most people use computers using nice icons and beautiful displays, here you will learn how to make things happen by directly executing commands.
3. Version control with GIT. Writing code is a complex, especially if you’re working in multiple files. This is where version control comes in: how to save your workspace at any moment, in case you or someone from your team messes things up. You will learn how to always keep track of the code changes you’re making and undo your work whenever needed.
4. JavaScript. The language of the web. You will learn the basics of programming, how to add interactivity to your web pages and how to make use of other services (GitHub, Google, etc.)
5. Node.js. Software that allows us to use JavaScript outside the browser to create all kinds of applications. You will learn the basics of Node.js in order to understand and build web servers, which will allows us to create complex and dynamic web applications.
6. MySQL. The world’s most popular open-source database. Every application needs a secure place to store sensitive data, so for that we use a database. You will learn the query language SQL in order to store and retrieve your data from your own database.
7. React.js. Currently the most popular way to create fast and dynamic user interfaces. In this module you will learn how to build a more modern and dynamic frontend that changes depending on what the user does.
8. Project. This is where all your knowledge and experience are put to the test. You will be working for a fictional company called ‘Open Source Factory’, and the goal is to build a fully functioning full-stack application while working in a team. See for two examples:  
   a) <http://www.freeweatherapi.com>   
   b) <http://hackyourestate.herokuapp.com/>

  
  
After the Project is finished, and everything went as it should, you will graduate. You will receive a diploma and you’ll become part of the Alumni Network.   
  
However, it is no time to rest yet because the next challenge will be to find an internship (see also section 1.5).

### 1.2 Homework and Preparation

As mentioned, our complete curriculum can be found on [Github](https://github.com/HackYourFuture/curriculum). The modules are ordered sequentially.  
  
The homework can be found in the module repository. For example, if you click on [HTML-CSS](https://github.com/HackYourFuture/HTML-CSS) you’ll be taken to the document that has all the information about that particular subject. If you scroll down you’ll see a table with a column of **Homework**



After the first couple of weeks, this is also where you will publish your homework. In the beginning you might struggle to use Github. But there is a reason we use it: It is the number 1 platform for programmers! To become a good programmer, means you will have to understand how Github works. You will notice that after a few weeks, finding the right homework assignments and reading and pushing your code to Github, will become automatic.

You are expected to **always** come prepared to the class on Sunday. Every week you will have homework assignments and we expect you to finish them *before* our teaching day. The sooner you hand them in, the more time your teacher has to check your homework and give you feedback.

It’s very obvious, but based on the past it seems necessary to mention: **You are not allowed to copy homework from others.** You have to take yourself and HYF sersious. In some occasions copying code is necessary and it will be allowed under three conditions:

1. A source reference is given (in a comment)
2. It must be clear which code is copied.
3. The functionality of the code is explained in a comment.

We are not a secret service that will check everything on fraud and plagiarism, but If we find out it will have consequences.

Not handing in homework without proper communication **will not be tolerated** (see also our values in chapter 5). We have a simple policy on this: 3 strikes and you are out.

### 1.3 Tests

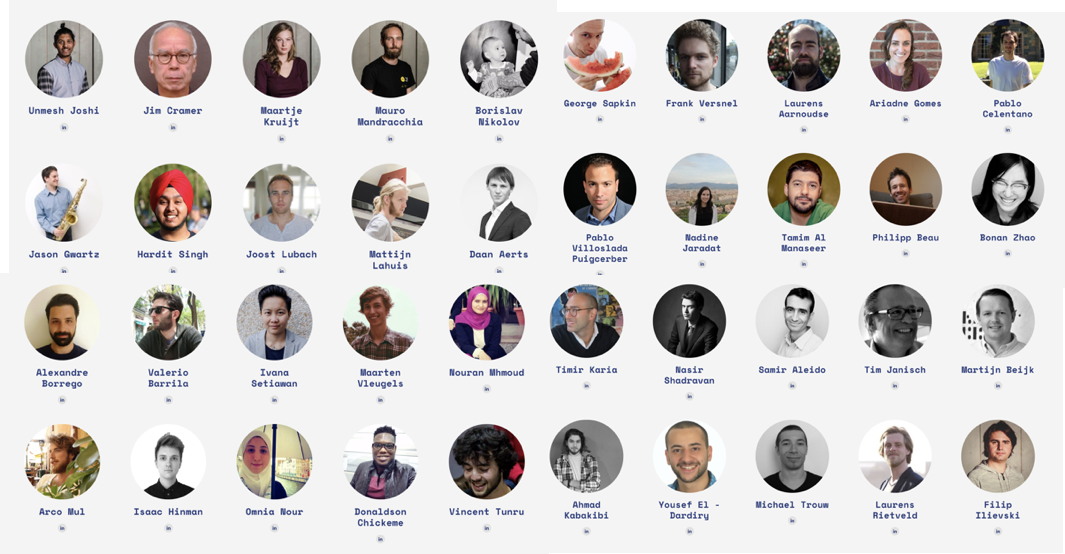
During the course you will have to do tests. This is for two reasons:

1. HackYourFuture wants to make sure you are making enough progress. If it seems like you still have trouble with some concepts we can help you by providing extra support.
2. You want to make sure you master the material, so that you will be ready for an internship once the program finishes.

These tests will not be graded. Instead, you will receive personalized feedback that you are expected to use to make the necessary improvements.

If there are any weaknesses in your understanding of the concepts we have taught you, the test will expose them. This is a good thing: you will get an opportunity to go over them again and master them. Make use of it, because in a company you will not always have such a luxury.

### 1.4 Teachers

HackYourFuture is only possible because of our amazing team of teachers. See for an overview [here](https://www.hackyourfuture.net/about/). Our teachers are medior and senior software developers working at small and big companies. They usually have very busy jobs and they sacrifice their limited free time to *help you* to become a developer.   
  
Respect their time. Value their time. Give them energy and motivate them.  
  
1.5 Start your career  
After graduation, it is time to find an internship. This internship almost always leads to a paid job. HackYourFuture will do its best to find suitable companies for you, as we have an extensive network of partner companies. However, finding work will depend completely on your own efforts.  
  
Finding an internship consists of the following steps:  
  
*1. CV and LinkedIn Support*   
During the project-phase we will contact you and work together to make sure you have a professional CV and LinkedIn profile ready in English.  
  
*2. Career Training (1 session)*  
Right after graduation, we have a one-day career training in which we discuss the basics of finding an internship, how to prepare for interviews, salary negotiations etc.  
  
*3. Practice Interview (1 session)*  
Two interviewers will test your social and technical skills (60-90 minutes). After this, you receive feedback about your performance. This will make you less nervous and prepare you for real interviews with companies.  
  
*4. Improve your technical skills*   
One of the biggest risks is to stop coding after graduation. This will **significantly** decrease your chances of finding an internship. Therefore, we have projects that you can build, with support from teachers, to develop your skills. You can also add these projects to your CV!   
  
*5. Finding internship possibilities*  
We will share a Google Doc list with you. We will train you to find leads for companies that are hiring developers. You will share these contacts with us and we will contact these companies (our experience has taught us this works better than if you contact companies yourself). In addition, we also have a network of partners of our own, which we will use to introduce you to companies.  
  
*6. Education Fee*  
It is in our best interest to find you an internship since we ask companies for a small compensation (*education fee*). We need this education fee to train the next class of students since we are not-for-profit organization.   
  
**IMPORTANT**: Finding an internship opportunity is a two-way street. While we reach out to many different companies, you need to contribute. You can control your chances of getting an internship by working on your technical and communication skills, and provide us with leads to companies.  
  
If you do not contribute, and remain unemployed, this is your responsibility. If the municipality or your contact person contacts us about your progress we will be straightforward and transparent about your performance.

# Chapter 2: How to study programming (tips for non-programmers)

Many people think programming is just about the code, 0’s and 1’s, about frameworks, technical stuff, or complicated words that just confuse us.

However, to be a good programmer, **the most important thing is to be good at problem solving.** We program computers because we have:

1. A problem/need

2. We want to use the computer to help us solve that problem / need

This chapter discusses the six steps of problem solving. This gives you a step-by-step approach on how to analyze and write code.

**1. Understand and define**The key to problem solving, is to really understand the problem. Write down the problem exactly or explain the problem to someone else, until you clearly understand what the problem is you are trying to solve. Many times, software developers start fixing problems without really knowing what the problem is (or what the client wants).

**2.** **Divide**   
Big problems (or challenges) should be divided into smaller problems, so that the problems become easier to solve. What seemed like a big monster problem, now just is a lot of small monsters on top of each other. For instance, if you want to loop through an array and select certain data from it, these are already two steps (making a loop and selecting data). The more you are able to divide big problems into small ones, the better.

**3.** **Finding the right technique**  
Now it is time to think about the techniques you know that could possibly help you to solve this problem (e.g. loops, if statements, filter, map etc.). If you do not find an appropriate technique you will have to find it. In both cases Google will probably be your best friend. Remember, most developers spend hours per day on Google trying to find solutions to their problems!

**4. Experiment**  
Now try out some of the techniques you have found. Often it works well to first try out the technique on an easy example you have made yourself, before you try to solve the problem. Experiment with some examples so that you really understand what the technique is capable of. After that, put it all together and see if your code gives you the expected results.  
  
**5. Debug**Often your code does not do what you want it to do. There is only one reason why this is the case: **Your logic is not sound**.

In the end the computer only executes on instructions that are written in a way it understands. Generally, there can be 2 reasons for your code not doing what you want:

1. The computer throws an error.

2. Your code returns an incorrect value that doesn’t solve your problem

Whenever this happens, you can do the following:

1. In case 1, make sure that you carefully read the error message, as it often gives you a good hint where the problem lies within your code. This should be the first thing to look for when something went wrong.
2. Check for syntax errors. Use JsLint for this (an extension for VSCode that helps you find syntax errors). Most of your mistakes will be simple spelling/syntax mistakes.
3. Use the debugger to understand what steps the computer goes through when interpreting your code, and find where it goes wrong. Try to explain each part of the code to yourself of what it does exactly.
4. Formulate your problem in clear terms to google and search for solutions
5. Ask people in your class/teacher or take a break and look back at your problem with fresh eyes.

**6. Reflect**  
After all this, it is time to reflect on what you have done. Are you happy about how you solved the problem? What went well and what did not?  
  
Most people tend to skip this step. “Why do I need to do this? I have already solved the problem!”. However, you will notice that if you do take some time to reflect on your code and problem-solving approach, you will improve your analytical mindset and become better at memorizing some parts of your code.  
  
Reflection also includes **refactoring**. Refactoring is the process of rewriting your existing solution into better looking, better functional, readable code. Often times you will find that you have solved a problem, but that doesn’t necessarily mean that you have solved it in the best possible way, or that you completely understand all the concepts you have used.   
  
In summary, reflection includes the following steps:

1. how could I rewrite this code for simplicity and better readability (refactoring)
2. what concepts in your code you don’t understand yet 100% (even though it works), and study them.
3. what you can learn from other solutions (by looking at other solutions of the same problem)?
4. what would you do differently next time?

**Example 1: Fruit Vendor**  
  
Let’s use an example to explain these 6 steps in more detail. **Note that the example below is a bit advanced if you are just starting our course.** Imagine a fruit vendor on the market. The fruit vendor has a couple of different fruits he is selling. However, he also has some rotten fruits. His fruits are represented through an array of strings: ["apple","rottenBanana","mango"].  
  
The fruit vendor has hired you to write a program that takes out the rotten fruit and only tells you which fruits are not rotten.  
  
**1. Understand and define**In our example understanding and defining the problem seems pretty straightforward. Indeed, we need to write a program that takes out the rotten fruit and returns the fruits that are not rotten.   
  
However, let’s try to make this a bit more specific. We have to write a program that first takes an array of strings containing fruit names as input. It then filters out the strings that do not contain the word “rotten” and gives them as output.

**2.** **Divide**

We now know what we want to do in a general sense, but can we divide our problems in a few different problems? Yes, we can:  
  
1. Declare a variable named *fruits* with the fruits mentioned in the example   
2. Define an empty function named *TakeOutRottenFruit*  
3. Inside the empty function write instructions that take out the strings containing the word “rotten”.

**3.** **Finding the right technique**  
After going over our notes and the curriculum We discover that the filter() function could be used to filter out elements in an array.   
  
However, the [documentation on Mozilla Development Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter) also tells us that it will return “a new array with all elements that pass the test implemented by the provided function”. That means we still have to instruct our computer to select the string(s) which contain the word ‘rotten’.   
  
After some more googling, we find out that includes() is a function that allows us to check the content of a string. Let us try to combine the two!   
  
**4. Experiment**This leads us to write the following code*const fruits = ["apple", "rottenBanana", "mango"]*

*function takeOutRottenFruit(arrayOfFruits) {*

*const goodFruitsArray = arrayOfFruits.filter(fruit => fruit.include("rotten"));*

*return goodFruitsArray;*

*}*

*takeOutRottenFruit(fruits)*  
**5. Debug**  
When we write the code under step 4 in the Google Chrome Console we get the following error: *Uncaught TypeError fruit.include is not a function*   
  
Probably we have made a small typo. Let’s google the include function. When we do, we notice that we should use fruit.includes (with an -s at the end).  
  
When we run the following code, it does not give any errors anymore.   
  
However, I do notice that only *rottenBanana* is selected and displayed.   
  
Hmmm, what to do? Let’s go back to the basics. I remember that in my JavaScript 1 Module I learned something about *operators* ([here](https://github.com/HackYourFuture/fundamentals/blob/master/fundamentals/operators.md)). Indeed, I find that you can use the *! operator* to return what is not *true* (everything that does **not** contain the word rotten). Let me try to add the *! operator.  
  
const fruits = ["apple", "rottenBanana", "mango"]*

*function takeOutRottenFruit(arrayOfFruits) {*

*const goodFruitsArray = arrayOfFruits.filter(fruit => !fruit.includes("rotten"));*

*return goodFruitsArray;*

*}*

*takeOutRottenFruit(fruits)*

**Yay! It seems to work! Can you think of a different / better solution?**

# Chapter 3: Study Strategies

Most people study inefficiently. This is mostly because they never really learned how to study the right way, even after years of university in some cases! In this chapter we will show you some of the most important strategies to learn in a way that is much more time efficient.   
  
As the HackYourFuture course is very demanding, this is critical to get through the program successfully. In fact, many successful experts that require focus and concentration for their craft (scientists, programmers, writers, artists, craftsmen, etc.) share one thing in common: they are very serious about *how* they learn. This chapter will give you a short introduction into how you can study most effectively.

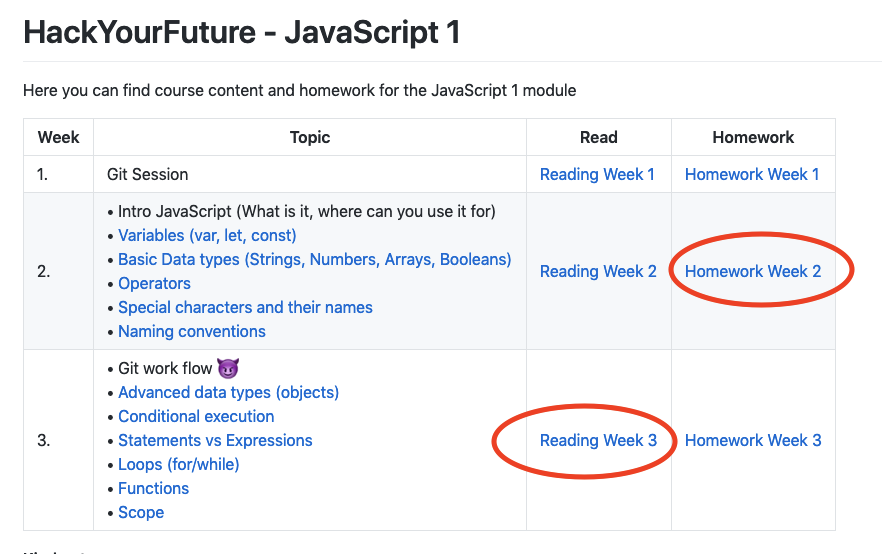
## 1. Planning

The first part of studying should be to make a study plan.   
  
Your homework will consist of various components. You will need to:

* **read/watch** a certain amount of information,
* **apply** this knowledge in the homework exercises.

Every week consists of the homework of the previous week and the reading material for the following week. For example, let’s say you are preparing for the lecture of JavaScript 1 week 3 on Sunday. This means you will have to do:

- the *HOMEWORK* of JavaScript 1 week 2 (The MAKEME.md [here](https://github.com/HackYourFuture/JavaScript1/blob/master/Week2/MAKEME.md))

- the *READING* for JavaScript 1 week 3 (The README.md [here](https://github.com/HackYourFuture/JavaScript1/blob/master/Week3/README.md))  


Before you start doing anything, it’s good to **plan your week ahead.** Divide the large amounts of tasks into small pieces, and think about how much time you expect to spend on each task by having a quick look at it.   
  
Let’s take the example of JavaScript 1 week 3.

**Homework + Recap (19 hours)**

*1. Recapping previous concepts (5 hours)*  
*2. JavaScript exercises (4 hours)*

*3. Freecodecamp challenges (10 hours)*Before starting the homework it is essential to first have a look at the concepts discussed in the previous week(s) before moving forward.  
  
You can do so by going over the Youtube recording of the previous lecture and practice. Your lectures are recorded and uploaded to Youtube ([here](https://www.youtube.com/channel/UCkK246iKcOAvsL0SI_6n3eA/videos)). These clips contain an explanation of the concepts you need to master to do the homework well.   
  
It is advised to watch some parts of the lecture again and practice with the examples/exercises provided in class. Change the values, variables and other aspects of the code to improve your understanding of the practical use of these concepts. In addition to the videos, you can go through the reading material of the previous week. Again, the idea is that you practice the concepts by putting them into code.

**Reading/Watching (6 hours)**

*1. Article arrays (60 min)*

*2. Article Loops (60 min.)*

*3. Article functions (60min)*

*4. Article scope (30min)*

*5. Objects(45min)*

*6. Conditions (45 min)*

*7. Expressions vs statements (25min)*

*8. Handing in homework with git (30 min).*We have a certain number of articles and videos that we share as part of the curriculum. However, it is important to read and watch them effectively. Again, try to follow the examples and code them yourself. Also, you are free to look for your own reading material and videos. **There is a lot of good material on the internet, which you can make use of.** Also, if you find a resource that you think is better than what we offer in the curriculum, please let us know (see textbox below!).

|  |
| --- |
| **How to Improve the Curriculum!**  1. Go to the Github page you want to add a suggestion to 2. Click on Issues.  3. Click new issue and add your suggestions. This can be a URL to an article, a video, graphic or anything else that will improve the curriculum. 4. We will review it and if we find it useful, add it to the curriculum.  We are looking forward to your ideas & suggestions! |

The homework and reading material combined take around 25 hours, however this does not account for getting stuck with exercises. Therefore, you probably want to add another 5 hours at least.

Now that you know how much time you have available and which are the activities you need to study per day, it’s time to make a **weekly schedule**. The key here is not to wait until Wednesday or Thursday, but to start Monday morning already. Learning programming concepts takes time, and if you try to do everything at the end of the week, your brain has very little time to process these concepts. Per day you schedule a number of small tasks, and once you are finished you can feel good about finishing the tasks of the day.   
  
Note that planning is an *art by itself*! If you want to learn how to make better weekly and daily schedules, we recommend you to read the following two posts by Cal Newport ([1](http://calnewport.com/blog/2014/08/08/deep-habits-plan-your-week-in-advance/) and [2](http://calnewport.com/blog/2013/12/21/deep-habits-the-importance-of-planning-every-minute-o)).

## 2. Don’t just read, code it!

As explained in the previous section, reading about code, or listening to your teacher about code, **is not the same as actually coding!** It is incredibly important that you try out the concepts that you have learned, and that you try to apply it in multiple ways in order to become comfortable with it. There is no way around this.   
  
We recommend that when you study something, directly try it out in your code editor by making a simple coding example and see how it works in practice. Only this way you will truly understand it well, and gain the ability to turn ideas into code.  
  
One more tip: Write code **with pen and paper**. Try to write down the basic concepts including some simple examples. Yes, this can also be done on the computer, however [research shows](https://www.theguardian.com/science/2014/dec/16/cognitive-benefits-handwriting-decline-typing) that writing things down with good old-fashioned pen and paper provides cognitive benefits that you cannot achieve when typing on your keyboard. By the way, some companies will ask you to do the same during their interview process – good practice indeed!

## 3. Go back to the basics

The basic concepts presented in the earlier modules are crucial to become a good programmer. Many students in JS2 and JS3 still struggle with arrays, string, objects, array of objects, properties and other basic concepts which are discussed in JavaScript. Many students in the React-module still struggle with callback functions and closures (discussed in JS2). In order to solve more complicated coding challenges, **you need to know these concepts!**   
  
If you really want to master the basic concepts, the best way is to do more exercises with them. You can find a lot on the internet, but you can also easily create your own basic exercises (like the rotten fruit example in chapter 1). For instance, can you come up with some examples on how to use a basic function?

## 4. Sleep, Breaks & Exercise

Sleep is incredibly important for your brain’s memory capability. If you study a lot, but don’t sleep enough, part of all the hours you put in are useless. On average the recommended time of sleep is **~8 hours**. If you go much below this, it will most likely decrease your performance as a student.   
  
We recommend that you do not look at any screen the hour before you go to bed, as [this improves your overall sleep quality.](https://www.sciencenewsforstudents.org/article/evening-screen-time-can-sabotage-sleep) Your brain thinks that the light of your screen is natural light, and this will make you stay awake.

Try to take 10-15 minutes breaks in between studying sessions of 90 minutes, as this re-energizes you and improves your focus. During these breaks, some form of physical exercise is highly recommended. Take a walk outside, stretch, dance...get moving!  
  
More generally, we recommend you eat healthy and do sports. There is a large body of evidence (for instance [this](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5934999/) and [this](https://www.health.harvard.edu/blog/regular-exercise-changes-brain-improve-memory-thinking-skills-201404097110)) that shows exercise helps memory and thinking through both direct and indirect means. Directly through a reduction of insulin resistance, inflammation, and an increase of chemicals in the brain that affect the health of brain cells, the growth of blood vessels in the brain, as well as the survival of new brain cells. Indirectly, exercise improves mood and sleep, and reduces stress and anxiety.

## 5. Distractions

The goal whilst studying should be to reach a state of ‘flow’. Flow can be described as the complete immersion in an activity. By doing so, you lose your sense of space and time. There is quite some research about ‘flow’. This research shows that not only your learning curve increases significantly, also the level of joy increases. In other words, if you are fully concentrated whilst programming you feel way more satisfied as a result.   
  
To reach this level of optimal concentration and create a state of flow, distractions should be shut out as much as possible. The two most common enemies are described below.  
  
**Challenge number 1: Social Media & Phone**

Social media is incredibly bad for your focus and therefore it decreases your ability to learn complex concepts. In order to understand programming concepts your brain needs full focus. Every time it is interrupted by a phone or a message online, your brain has to switch its attention. To go back to your code after that will cost a lot of energy. In fact, studies have shown that [just having a phone in the same room](https://www.journals.uchicago.edu/doi/full/10.1086/691462), reduces your ability to study significantly.   
  
We highly recommend you: when you are studying, put your phone away and use your study time for studying only.

**Challenge number 2: Children and family**  
Many of us are not living by ourselves. You might have children, or family members asking for attention. The challenge here is to try and create a quiet space, where you will not be interrupted. Of course, this is not always possible if you are a parent. But perhaps you can ask your partner or a friend to keep an eye on your children at least for a few hours a day, while you go to a local library, or study in a quiet room at a friend’s house.

|  |
| --- |
| **How to become a pro at learning**  Obviously, this chapter only provides you with a very basic introduction.   If you are interested in learning more about how to learn effectively we recommend you to read more about deliberate learning and deep work. Studying effectively – although NOT easy- is a superpower you can use for the rest of your life!!!  An excellent read about the value of working without distractions (and how to achieve it) is the book *Deep Work* by Cal Newport ([here](http://calnewport.com/books/deep-work/)). |

# Chapter 4: How to ask Questions

Many students find it hard to ask questions during the classes and online (through Slack or otherwise). However, to become a good programmer means you dare to ask questions, lots of questions. In fact, some companies even have a rule on this: If you are stuck, you have *one hour* to solve the problem. If you cannot, you *have to* ask for help.  
  
In HackYourFuture there are several ways to ask for help:

* Ask your classmate or a student from another class
* Ask in Slack (preferably in your classroom channel)
* Approach a teacher during breaks or through slack in a group
* Ask HYF staff to connect you to a graduate or teacher for individual sessions

Nevertheless, we often notice students not asking for help is the prime reason for them dropping out of our program.

Let’s take a classic example we often encounter in HackYourFuture:  
  
During class the teacher explains a concept. At some point the students do not understand what the teacher is trying to explain. One or two students ask questions. However, these students already understand most of the concepts and are therefore comfortable to ask questions to improve their knowledge. Yet, most classmates remain silent and do not ask any questions, even though they can hardly keep up with the teacher. What are the reasons? Below we discuss 3 reasons and some suggestions on how to ask questions anyway.

**Reason 1:**  
You do not want to take the risk of looking stupid in front of the class.  
  
**Solution:**  
  
*Fun story:**A woman, recounting a story about an old man who used to answer all her "stupid questions", explained: "If you ask a question it makes you look stupid for 5 minutes – but if you don't ask – you stay stupid for fifty years, so always ask questions in your life".*  
  
If reason 1 is your reason for not asking, we say: you are wasting your own time. Instead of asking the teacher, you are telling yourself you will understand it by yourself later on. What often happens is that you will only understand 50% of it when you look it up yourself, and this will cost you much more time than just asking the teacher and getting a clear answer directly. Also, in your job you will need to be capable of asking questions in front of people: start practicing now!

**Reason 2:**   
You know you are not completely understanding the concept/explanation but do not know how to transform this into a useful question.  
  
**Solution**  
There are several ways to ask a question when you are not sure how to frame it:  
  
*Recap what you understood, and share where you lost the teacher*

Example: “I Understood how it worked until this point, but after that you lost me. Can you explain part X again?”

*How does X work:*

Example: “How does part x work? I still don’t really understand it.”

*Why do we do X?*

Example: “Why do you use X to solve problem Y?”

*Check your own understanding*

Example: “Do I understand correctly that X does Y and Z?”

*Ask for an example:*  
Example: “Can you give an example of how X works?”

**Reason 3:**  
You lack a fundamental understanding of a basic concept, which prevents you from understanding the new information.

**Solution**  
This situation applies when you have not mastered the underlying basic concepts yet and therefore cannot understand the explanation of the teacher (or reading materials) when discussing more advanced concepts.   
  
For instance, the teacher is explaining *callback functions* with a practical example. However, you do not grasp his/her explanation since you are still not completely comfortable with basic functions, which are explained and used in the JS1/JS2 modules.

If this happens, it is probably a good idea to take some time apart with the teacher, a graduate student or someone else **who is good at explaining the basics**. We can always provide you with some help, for instance by organizing individual video calls or appointments with former students or teachers. Moreover, it is important to revisit the basic concepts that you struggle and *practice, practice, practice* until you dream of Javascript functions. After that, you can try to revisit the more advanced concepts (for instance by re-watching the lecture on YouTube).   
  
If you feel reason 3 applies to you, it is crucial not to wait and hope things get better with time. As the speed of our curriculum is very high, **you have to take initiative and reach out to us for HELP.** Again, good programmers ask for help. Those that do not ask for help, will never become good programmers.

# Chapter 5: Core Values

This chapter lays out our core values. These core values are leading for all of us involved in HackYourFuture: staff, alumni, students and teachers. They are leading in how we learn and teach and interact with each other.   
   
Our core values help us to reach our immediate goal more efficiently, which is to make sure talented junior developers find a job in the tech sector. However, the contribution of core values goes beyond this mere functional argument.  
  
We believe HackYourFuture is more than an education program. HackYourFuture is a community of creative individuals who can do great things now, and in the future. Yet this is only possible if students, teachers and staff all feel they are part of something bigger than a mere training program.

These basic values are an attempt to guide you towards behaviours and attitudes that have proven to be appreciated and effective both in- and outside HackYourFuture. Both in work and in life.  
  
**1. Be Proactive**   
  
HackYourFuture was predicated on the principle of taking control and fate in your own hands. [As explained very eloquently by Neveen Atik (class 15)](https://vimeo.com/273167685): “HYF gives us the opportunity to be out of the labeled group [of refugees] and always needing help. Now you can support yourself”. And you can support others too!   
  
Taking initiative is also in line with the way HackYourFuture was founded. In 2016 we realized a huge demand for talented developers and an influx of talented migrants into the Netherlands. Instead of complaining about migration or the lack of diversity in the tech sector, we set up HackYourFuture. It is that spirit of entrepreneurship we believe makes up the core of HackYourFuture’s DNA.   
  
Some more examples of a proactive approach include:

1. Many students realize 30 hours of studying per week is not enough and take on side-projects to improve their coding skills.
2. While living in his refugee-camp, Sarea Al Kebaly (class 0) built Amsterdam Voorziet, an online platform for underprivileged citizens in need of support and services.
3. Many people complain about the lack of representation of women in tech, but only a few try to actively change anything about it. Therefore, we decided to organize a *women’s coding teaser*, together with female teachers and students, to motivate woman to join HYF. If you want to help out, please contact Aya Al-Abrash (class 14).
4. Many students are already working on their CV during the course and apply for junior developer positions.

**2. Show Commitment**Joining HackYourFuture as a student is a big achievement. You probably worked hard on your application assignment to get in. However, joining HYF also comes with certain expectations. We expect you to be committed to HackYourFuture, as long as you are a student with us.   
  
This includes, amongst others:

* We expect you to do your homework in time and put in all the effort required.
* It might happen that you are not able to make it to class or finish your homework. If this is an exceptional situation, there is no shame in that. However, there *is* shame in not communicating. Please **communicate ahead of time.** A lack of communication (or communicating too late) will lead to you being removed from the HYF program.
* Please respond to the feedback that teachers and other volunteers give you on your homework.
* If you start the HYF course, this means you are with us for 7 months. You have to finish our program before starting an internship or job.

* HackYourFuture will help you find an internship after the program. It is also possible that you find an internship yourself. However, we would still appreciate if you can make sure that we receive an *education fee* from the company you start working for. Why? We are a not-for-profit and rely on these fees to train the next generation of talented developers.[[1]](#footnote-1)
* Please make sure that if you have a client manager (contact person) at the Gemeente, he/she is aware of HackYourFuture and agrees with your participation and starting an internship afterwards. Some municipalities are concerned about internships and prefer for you to start a job immediately. However, to start with an internship is usually the best way to start your career as a programmer. Therefore it is really important to have a good relationship with your client manager and inform them ahead of time. Please reach out to us if you need any support. We can connect them and send a letter to confirm your participation.

Part of being committed is also being able to give and receive feedback. We are committed to improving HackYourFuture as much as possible. Therefore, If you see room for any improvements, we are really happy to hear from you!

**3. Contribute to the Community**  
  
HackYourFuture is largely dependent on the free time and goodwill of volunteers who could easily hang out in the park or spend time with their family and friends on their free Sunday, instead of teaching you for free. As such, contributions by all of us make -or break- HackYourFuture.   
  
There is some overlap with being proactive, the first core value. However, whereas being proactive is mostly about controlling your own fate, to contribute is about the larger community. *Do not ask what HackYourFuture can do for you, ask what you can do for HackYourFuture.*  
  
This is probably one of the most exciting aspects of HackYourFuture: you can make it into what you want it to be. Do you want to set up a study club with some other coders to take on new challenges? Go ahead (hint: you are not the first). Do you feel we need to improve the curriculum? Suggest some changes via pull-requests. Want more socializing? Join us for drinks (or plans drink yourself! We would love to be invited too!).   
  
Some examples:

1. Quite a few students are regularly making git-issues, suggesting improvements to the curriculum.
2. Rabih Alqaraweet (class 3) visits companies together with Wouter to tell the story of HackYourFuture, which helps potential employers to hire our graduated students.
3. Some students of class 16 (now graduated) were meeting on a weekly basis to work on their code.
4. Philipp Beau (teacher) is experimenting with innovative teaching practices to increase interaction in class.
5. Some of our male students made sure some of their female friends or family members attended the women’s coding teaser to increase the participation of women.
6. After graduating and finding a job, Samir Aleido (class 2) has been assisting our lessons for more than two years!
7. When Unmesh Joshi joined HYF, he did not master any of the languages taught in our curriculum. Still, he stuck around and looked for a way to contribute. This resulted in developing our Command Line and Git curriculum which ended up as a key component of our current program.
8. Jason (teacher) moved to Germany, yet is still spending his free time helping to train our graduates for technical interviews through hangouts sessions.
9. Many volunteers organise one-on-one or group hangouts during the week, just because they see there is a need for it.

**4. Be Kind**  
Sometimes the simplest of values are the hardest to uphold. Although working and studying as part of the HackYourFuture program can be stressful at times, it is important to remind ourselves to be kind to each other.   
  
Kindness and empathy are values that make teams and organizations function well. What does this mean on a daily basis?

* We focus on how we can solve problems, not what the problem is and who caused it (hint: if you are typing *git blame* you are doing it wrong). We are considerate about each other’s personal situation.
* We formulate feedback in a constructive way, with a focus on practical solutions rather than things we do not enjoy or like.
* HackYourFuture is **not** a competition. When we see that one of our colleagues is having a hard time, we ask them how they are and if we can help.

Some practical examples include:  
  
1. Some of our more advanced students support their classmates when they notice them falling behind.

2. Jim Cramer (teacher) has spent hours and hours privately tutoring students struggling with the learning material.   
3. Some of our graduates pair up with students to help them with their homework.   
4. Uday Kahki (class 8) bought our former Education Director Maartje a birthday gift ☺

# Final thoughts

As of March 2019, over 100 students graduated from our program. Over 90% are working in the IT sector, within three months after graduation. In other words, HackYourFuture works.   
  
However, throughout you HYF course and you career as a programmer, you will control your own fate.   
  
30 hours of study is just the minimum we suggest. The more you know and the harder you work, the faster you will find yourself in a job you love. The more you participate and become part of the HYF community, the more people are going to be willing to help you out. The more friends you make.

We want to provide you with the knowledge, tools and support you need to build a career as a programmer. To support yourself and your loved ones.

This is not a race against the curriculum or a race against other students. This is a race against yourself, so be sure to come up first every time, every day.

We are glad to have you on board!  
  
**The HYF Team**

1. To be more exact. 70% of our budget is derived from donations and corporate philanthropy (ING Netherlands Fonds is our main funder). 30% of our budget comes from education fees. [↑](#footnote-ref-1)