# JAVASCRIPT THE HARD WAY pdf

## 1: Learn JavaScript The Hard Way Has Begun â€“ Learn Code The Hard Way

*Learn JavaScript The Hard Way is an introduction to JavaScript using the successful exercise based method that has taught millions of people how to code. It will teach you web front-end development in a unique way that hasn't been done before.*

More If Statements Exercise  Otherwise If Statements with Else Exercise  If Statements with Strings Exercise  More Chains of Ifs and Else Exercise  Nested If Statements Exercise  Making Decisions with a Big Switch Exercise  More String Comparisons Exercise  Choosing Numbers Randomly Exercise  Repeating Yourself using a While Loop Exercise  A Number-Guessing Game Exercise  Infinite Loops Exercise  Using Loops for Error-Checking Exercise  Do-While Loops Exercise  Adding Values One at a Time Exercise  Adding Values for a Dice Game Exercise  Calling a Function Exercise  Calling Functions to Draw a Flag Exercise  Displaying Dice with Functions Exercise  Returning a Value from a Function Exercise  Areas of Shapes Exercise  Thirty Days Revisited with Javadoc Exercise  Importing Standard Libraries Exercise  Programs that Write to Files Exercise  Getting Data from a File Exercise  Saving a High Score Exercise  Counting with a For Loop Exercise  Nested For Loops Exercise  Generating and Filtering Values Exercise  Finding Things in an Array Exercise  Arrays Without Foreach Loops Exercise  Lowest Temperature Exercise  Mailing Addresses Records Exercise  Records from a File Exercise  An Array of Records Exercise  A Deck of Playing Cards Exercise  You should take as long as it takes to get through it, but focus on doing work every day. Some people take about 3 months, others 6 months, and some only a week. What kind of computer do I need?

## 2: Learn Javascript the Hard Way

*In Defense of JavaScript (the Hard Way) Learning to code the "hard way" means to learn the fundamentals and to be aware of the noise that obscures the underlying reality.*

The JavaScript book is now in full development mode and exercises will start landing this weekend. You can see the outline for the book at the end of this article. I said that the that the first exercises would start coming down in April, but there were a few issues with the latest release of JavaScript that derailed my original plans for the book. I wanted to wait until I could figure out exactly how to do the book and make it resistant to possible future changes to JavaScript. The other issue I had to solve was exactly what platform to support. I originally wanted to support browsers, but those are so diverse in what they allow, and so that I had to abandon that plan. I realized there are plenty of books that teach you how to use JavaScript in the browser, but not many that focus on the basic computer science concepts and fundamentals. So I changed the format and focused just on Node. Node is a little behind on some of the features, but it has enough for people to learn basic JavaScript and complete a bunch of projects. With the platform solved I then had to figure out a way to update and modernize the style of book that I write. I originally was going to do a book similar to my others, but then I realized I could most likely combine the beginner book such as Learn Python the Hard Way with my advanced book Learn More Python the Hard Way. Combining the two structures made for a much more complete book that takes someone from complete beginner to actually finishing many projects in JavaScript. The final improvement that I worked on was improving the quality of the videos for the JavaScript book. With the JavaScript book I want to show people my actual development environment as I work on the projects. I normally use a really large screen, even larger than p. I found that beginners develop an unrealistic idea of how programmers actually is done because they see me using a different set up than I actually use when I code. I actually use a screen that is much larger than that and sometimes even used two screens. For the JavaScript book I wanted to use a large enough video size that people could watch me code as close to how I really code as possible. On a large monitor, with multiple windows, and without switching windows. To accomplish this I had to do several tests of different video encoding software, recording systems, audio equipment, and also did quite a lot of work in live coding sessions with other students. The end result is that I can now produce p videos with higher quality sound a lot faster and cheaper than I could before. The Planned Features Given all that the features of the JavaScript book are going to be the following: UHD quality video at P resolution, but with smaller files for people with limited download Internet. A combined format that starts from a beginner level and goes all the way to completing many projects of increasing degrees of difficulty, effectively being two books. Focusing on more systems level programming with Node and using the most modern JavaScript I can get away with that Node supports. A total of 62 exercises. This first release will be a draft, and should include most of the videos for those exercises. I will then be able to post probably three or six videos a week depending on the difficulty of the exercises.

## 3: Six Simple Mind Tricks to Help You Learn JavaScript Faster â€” SitePoint

*And the lpthw equivalent in popularity, the almighty and the hard (If you don't understand railroad way) 'JavaScript The Good Part'. The combination of lpthw and freecodecamp may satisfy you. Also, you can make a great tutorial for yourself too that can be a equivalent - however, this is more than the hard way.*

Fortunately, these challenges can be recognized and ultimately conquered. For example, to become a front-end developer, your road map might look like this: Allow me to explain. When you read something and it makes sense, it can be tempting to move on to the next thing immediately. You give the previous concepts a quick glance to refresh your memory and then move on again. Limit the amount of stuff you learn at one time. Practice for real â€” actually write code. When you learn a new concept, make sure to try it out, play with it, get comfortable with it, and even combine it with other concepts. I learned this the hard way on several occasions. Try this mindset shift: How would you feel? It would be like a kid getting a new toy and not being allowed to play with it. Do something cool with your new skills. Whether or not this describes you, there are still lessons to be learned here. How does that happen? Getting started on something is the hardest part, so by keeping the initial commitment small, I find it much easier to jump in. The good news is that you can use this same psychology to your advantage when learning to code. A friend of mine was once confused about a certain feature of JavaScript. I asked him to walk me through what he knew and then explain which part was confusing. As he went through the piece of code, I noticed that he was rushing. I stopped him again. Try again, but this time, I want you to literally go through each line of this and tell me what exactly is happening in each line. The key was that he had taken the time to step through each piece of it instead of trying to understand all of it at once. In cases like this, thinking slower actually makes you learn faster. Write Complex Code in Plain Language First If a piece of code is going to be complicated or unfamiliar, write it out in plain language first. That way, you can figure out what you want the code to do before you actually have to write it. Here are two benefits to this approach: Stop worrying about future decisions and dive in. Make practice fun by treating new skills like toys. Find time to code by only making tiny commitments the way you would with sites like Facebook, YouTube, or Wikipedia. So how do you approach learning? Meet the author Yaphi Berhanu is a web developer who loves helping people boost their coding skills. He writes tips and tricks at http: In his completely unbiased opinion, he suggests checking it out.

## 4: What are your thoughts on Learn Python the Hard Way? : learnprogramming

*Learn JavaScript - The Hard way codylindley April 20, 14 k. Learn JavaScript - The Hard way. I've been asked for live links. Go them here: https://docs.*

Learn Design the Hard Way is a complete course on visual design taught in Sketch, no prior design experience required. Sign up for the beta Sign up to get notified when the full design course is released. You wanted feedback on the flow, the interactions, and the underlying concept. What you got was nitpicky advice on the shade of blue you picked, the placement of your form labels, and your icon choice. Since visual design is the stuff you can see, people will always notice and respond to it first, sometimes at the exclusion of all else. We all judge books by their covers â€" this is called salience bias , and all humans are susceptible to it. How do you accommodate this bias while keeping people focused on solving the right problems? There is a step by step process you can follow that will allow you to produce high quality design, avoid rookie mistakes, and imbue your work with credibility. The findings prove the importance of visual design: Yet, the study shows a clear link between solid design and site credibility. They found that aesthetically pleasing interfaces are not only perceived to work better, but they do literally work better. The full course will cover: Affordances Understand how visual design contributes to and detracts from the usability of an interface. Typography With a few constraints and simple math, learn to consistently achieve gorgeous typography. Layout Using common rules and grid systems, design layouts that direct the flow of traffic. Color Use beautiful, consistent color palettes to engage users and evoke the right response. Master the fundamentals Learn Design the Hard Way is not for everyone. It is an ideal course for anyone working on digital products: Design is a learn-by-doing sort of craft and the course is structured around practical exercises. This course is designed for people who want to learn the design process from the ground up. The full course includes High Quality Ebook Learn the theory behind each design principle and technique and how it fits into the process. Video Walkthroughs Watch the author walk through each exercise to compare against what you have done. Sign up below to get a discount when it launches! Zac went beyond anything I anticipated and thought through the design, content, marketing, usability, and entire structure for my company. When I saw what he could do and that he was teaching design as well I had to get him to create a course teaching his methods to non-designers like me. At Tradecraft, we teach while designers work on real projects for real companies. I am also the cofounder of Foliotwist , a service for visual artists.

## 5: Learn JavaScript - The Hard way - Speaker Deck

*Learn JavaScript The Hard Way Has Begun TLDR: The JavaScript book is now in full development mode and exercises will start landing this weekend. I've worked out a way to make this book be two books in one and will produce p videos for it.*

To a number of you, what you are about to read will appear to be very obvious and just the sensible thing to do. Take the advice below to heart and keep it in a part of your brain that has a quick access route so you can apply it without thinking about it. I am sure you will find things to disagree with, and that is a good thing - you should question what you read, and strive to find better solutions. However, I have found that following these principles has made me a more effective developer and allowed other developers to build upon my work more easily. None of these make much sense â€” good variable and function names should be easy to understand and tell you what is going on â€” not more and not less. One trap to avoid is marrying values and functionality in names. A function called isLegalDrinkingAge makes more sense than isOverEighteen as the legal drinking age varies from country to country, and there are other things than drinking to consider that are limited by age. Hungarian notation is a good variable naming scheme to adopt there are other naming schemes to consider , the advantage being that you know what something is supposed to be and not just what it is. It is very informative for some, but seems like extra overhead to others â€” it is really up to you whether you use it or not. Keeping to English is a good idea, too. Programming languages are in English, so why not keep this as a logical step for the rest of your code. Having spent some time debugging Korean and Slovenian code, I can assure you it is not much fun for a non-native speaker. See your code as a narrative. If you can read line by line and understand what is going on, well done. If you need to use a sketchpad to keep up with the flow of logic, then your code needs some work. Try reading Dostojewski if you want a comparison to the real world â€” I got lost on a page with 14 Russian names, 4 of which were pseudonyms. Avoid globals Global variables and function names are an incredibly bad idea. The reason is that every JavaScript file included in the page runs in the same scope. Say you have three functions and a variable like this: This can get annoying and repetitive. It is easier to wrap the whole thing in an anonymous function and protect the scope that way. This trick is called the Module Pattern: None of these are available from the outside at all any more. If you want to make them available you need to wrap the things you want to make public in a return statement: This makes it easy to call functions and access variables from other places without having to go through the myNameSpace name. It also means that you can have a public alias for a function in case you want to give it a longer, descriptive name for internal linking but a shorter one for the outside: Stick to a strict coding style Browsers are very forgiving when it comes to JavaScript syntax. This should not however be a reason for you to write sloppy code that relies on browsers to make it work. The easiest way to check the syntactical quality of your code is to run it through JSLint â€” a JavaScript validation tool that gives you a detailed report about the syntax warnings and their meaning. People have been writing extensions for editors for example the JS Tools for TextMate that automatically lint your code when you save it. JSLint can be a bit touchy about the results it returns and â€” as its developer Douglas Crockford says â€” it can hurt your feelings. Clean and valid code means less confusing bugs to fix, easier handover to other developers and better code security. When you rely on hacks to get your code to work it is likely that there is also a security exploit that uses the same hacks. In addition, as hacks get fixed in browsers, your code will cease to work in the next version of the browser. Valid code also means that it can be converted by scripts to other formats â€” hacky code will need a human to do that. Comment as much as needed but not more Comments are your messages to other developers and yourself, if you come back to your code after several months working on something else. What I see as a flaw in this argument is that explanations are a very subjective thing â€” you cannot expect every developer to understand what some code is doing from exactly the same explanation. Again the trick is moderation. If you comment out parts of your code to be used at a later stage or to debug code there is a pretty sweet trick you can do: Removing the slash will comment it out again. For larger applications comment documentation in JavaDoc style makes a lot of sense â€” you are seeding the overall documentation of your product by writing code. Avoid mixing with other technologies Whilst it is possible to create everything you need in a document using JavaScript and the DOM it is not necessarily the most effective way of doing so. Furthermore, not every JavaScript developer is proficient or interested in CSS, which means there will be a lot of back and forth until the outcome is reached. Say for example you want to hide all DIVs with a certain class in a document. You could loop through all the DIVs, check their classes and then change their style collection. In newer browsers you could use a CSS selector engine and then alter the style collection. The easiest way however is to use JavaScript to set a class on a parent element and use syntax along the lines of element. Use shortcut notation when it makes sense Shortcut notation is a tricky subject: Objects are probably the most versatile thing you have in JavaScript. The old-school way of writing them is doing something like this: Instead it makes much more sense to have the following construct, also called an object literal: This is a misnomer as arrays with named properties rather than an index are actually objects and should be defined as such. For example, the following construct defines a variable as 1 or -1, depending on the value of another variable: Another common situation in JavaScript is providing a preset value for a variable if it is not defined, like so: Modularize â€” one function per task This is a general programming best practice â€" making sure that you create functions that fulfill one job at a time makes it easy for other developers to debug and change your code without having to scan through all the code to work out what code block performs what function. This also applies to creating helper functions for common tasks. If you find yourself doing the same thing in several different functions then it is a good idea to create a more generic helper function instead, and reuse that functionality where it is needed. Also, one way in and one way out makes more sense than forking the code in the function itself. Say you wanted to write a helper function to create new links. You could do it like this: A cleaner way is to return the link and cover the extra cases in the main functions that need them. This turns addLink into the more generic createLink: That way you can easily change the application and remove functionality without having to scan the rest of the document for dependencies. Enhance progressively Progressive Enhancement as a development practice is discussed in detail in the Graceful degradation versus progressive enhancement. In essence what you should do is write code that works regardless of available technology. It is amazing how many times you will build a massively convoluted JavaScript solution for a problem that can be solved easily without it. One example I encountered was a search box on a page that allowed you to search different data: In the original version the different data options were links that would re-write the action attribute of the form to point to different scripts on the back end to perform the searches. The problem was that if JavaScript was turned off the links would still show up but every search would return standard web results as the action of the form never got changed. The solution was very simple: This not only made the search work correctly for everybody, it also made it easy to track how many users chose which option. By using the correct HTML construct we managed to get rid of both the JavaScript to switch the form action and the click tracking scripts and made it work for every user out there â€" regardless of environment. Allow for configuration and translation One of the most successful tips to keep your code maintainable and clean is to create a configuration object that contains all the things that are likely to change over time. These include any text used in elements you create including button values and alternative text for images , CSS class and ID names and general parameters of the interface you build. For example the Easy YouTube player has the following configuration object: The script will get access to the different elements of the player with these IDs, so if you change them in the HTML below, make sure to also change the name here! This changed during development of this, so I thought it useful to make it a parameter. It is of utmost importance to keep code maintenance simple, avoiding the need for future maintainers having to read all your code and find where they need to change things. Avoid heavy nesting Nesting code explains its logic and makes it much easier to read, however nesting it too far can also make it hard to follow what you are trying to do. The other problem of nesting is variable names and loops. This can become messy quite quickly: If the list nesting were to go even deeper I would need more variable names, and so on and so on. It makes more sense to put the task of creating nested lists for each member in its own function and call this with the right data. This also prevents us from having a loop inside a loop. The addMemberData function is pretty generic and is very likely to come in handy at another time. Taking these thoughts on board, I would rewrite the code as follows: One of the most common mistake is to read the length attribute of an array at every iteration: You can avoid that by storing the length value in a different variable: This includes regular expressions and â€” more importantly â€” DOM manipulation. You can create the DOM nodes in the loop but avoid inserting them into the document. You can see this when running complex web applications when your computer is already maxed out with other work â€” changes take longer or get shown half way through and so on. Instead of constantly creating and applying elements, have a tool function that turns a string into DOM elements and call this function at the end of your generation process to disturb the browser rendering once rather than continually. This is wasted time and effort â€” we should build code based on agreed standards as outlined in this course of articles, not for one browser. The web is for everybody, not an elite group of users with a state-of-the-art configuration. As the browser market moves quickly you will have to go back to your code and keep fixing it. This is neither effective nor fun. If something amazing works in one browser only and you really have to use it, put that code in its own script document and name it with browser and version. This means that you can find and remove this functionality more easily, should this browser become obsolete. This is not only about evil people wanting to hack your systems; it starts with plain usability.

## 6: Learn Design the Hard Way

*Learn Java the Hard Way Really learn the basics of programming with Java "Learn Java the Hard Way" is a book with tutorial videos that teaches you how to code the same way a lot of us learned as children in the s: typing in short, simple programs and getting them to work.*

## 7: XMLHttpRequest the hard way | www.enganchecubano.com

*Category Comedy; Song E.T. - The Extra-Terrestrial - Flying; Artist The City of Prague Orchestra; Album Film Music of John Williams.*

## 8: Preloading images using javascript, the right way and without frameworks | frag (frÄƒg)

*In this article, I'm going to present six mind tricks which will help you learn JavaScript faster and become a happier, more productive coder. I learned this the hard way on several.*

## 9: Become a Programmer, Motherfucker

*Learn Java the Hard Way. Second Edition! Table of Contents. Preface: Learning by Doing; "Learn Java the Hard Way" is Â© Graham Mitchell.*

# JAVASCRIPT THE HARD WAY pdf

*Kikkoman Oriental Cooking In the studio with Simon Michael Rachmaninov orchestral music Making Lesbians Visible in the Substance Use Field (Journal of Lesbian and Gay Studies (Journal of Lesbia Colonialism and the African experience Octavia butler parable of the sower Asymptotic Statistics (Cambridge Series in Statistical and Probabilistic Mathematics) Crucified with Christ Susan Sorensen Spanish for spanish speakers 1 textbook The development of language 9th edition Balancing karma id locke The book of assistance How do i make a editable in word Study and Solutions Guide for Algebra and Trigonometry Leisure education IV Jongs Community Dental Health (Community Dental Health Jongs)) Search for the Word of God Marko kloos points of impact Recess: new old story news Authority in the Church Changing definition of masculinity Berserk Raids Ibm Laplace and fourier transforms goyal and gupta History of the Arkansas Teachers Association Beijings experimental theatre by Meng Jinghui Frogments from the Frag Pool Throne of glass 4 Indian country: cultural views of the Spokanes A travellers history of Russia and the USSR Dark Power Collection (Forbidden Doors) Personality disorders borderline personality disorder Once upon an Eskimo time Worlds Wonders (Raintree Fusion: Social Studies) Intravascular ultrasound imaging in coronary artery disease Steroids and peptides Aristotelian/scholastic hylomorphism and the rise of mechanism Lady vivian defies a duke The naturalization act of 1870 july 14th Premarital cohabitation : cautions and concerns Lucky Day (Care Bears)*