I'm not robot
reCAPTCHA
Continue

Functional programming in javascript luis atencio

SummaryFunctional JavaScript programming teaches JavaScript developers functional techniques that will improve extensibility, modularity, reusability, testability, and performance. Through specific examples and jargon-free explanations, this book will teach you how to apply functional programming to real-life development tasksA shopping print book contains a free eBook in PDF, Kindle, and ePub format from Manning Publications.About TechnologyIn complex web applications, low-level details of your JavaScript code can obscure the functioning of the system as a whole. As a coding style, functional programming (FP) supports loosely sagging relationships between components of your application, making the big picture easier to design, communicate, and maintain. About BookFunctional Programming in JavaScript will teach you techniques to improve your web applications – their extensibility, modularity, reusability and testability, as well as their performance. This easy-to-read book uses specific examples and clear explanations to show you how to use functional programming in real life. If you're new to functional programming, you'll appreciate many insider comparisons of this guide with imperative or object-oriented programming to help you understand functional design. In the end, you'll think about designing your app in a fresh new way, and you can even grow to appreciate monads! What is InsideHigh-value FP techniques for real-world usesUse of FP, where it makes the most senseSeparatrace logic of your system from implementing detailsFP-style error handling, testing and debuggingAll code samples use JavaScript ES6 (ES 2015) O ReaderWritten for developers with a solid understanding of javascript basics and web application design. About AuthorLuis Atencio is a software engineer and architect building enterprise applications in Java, PHP and JavaScript.Table of ContentsPART 1 THINK FUNCTIONAL FUNCTIONAL JavaScript Part 2 GET FUNCTIONALFew data structures, many operations Towards modular, reusable code Design patterns against complexityPART 3 ENHANCING YOUR FUNCTIONAL SKILLSThe combination of your codeFunctional optimization Management of asynchronous events and data Summary functional programming in JavaScript teaches JavaScript developers functional techniques , which will improve extensibness, which will improve extensibness, which will improve extensibness, which will improve extensibility, which will improve extensibness, which will improve extensibility, which will improve extensibness, which will which will improve extensiveity , modularity, reusability, testability and performance. Through specific examples and jargon-free explanations, this book will teach you how to apply functional programming to real-life development tasks Buying a print book containing a free eBook in PDF, Kindle, and ePub format from Manning's publication. Technology information In complex web applications, low-level details of your JavaScript code can obscure the functioning of the system as a whole. As a coding style, functional programming (FP) supports loose bound between application components, making it easier to design, communicate, and maintain a large image. About the book Functional Programming in JavaScript will teach you techniques to improve your web applications - their extensibility, modularity, reusability and testability, as well as their performance. This easy-to-read book uses specific examples and clear explanations to show you how to use functional programming in real life. If you're new to functional programming, you'll appreciate many insider comparisons of this guide with imperative or object-oriented programming to help you understand functional design. In the end, you'll think about designing your app in a fresh new way, and you can even grow to appreciate monads! What's inside high-value FP techniques for the real-world is used using FP, where it makes the most sense to separate the logic of your system from implementing FP-style error handling details, testing and debugging all code samples using JavaScript ES6 (ES 2015) O Reader Written for developers with a solid understanding of javascript basics and web application design. About AuthorLuis Atencio is a software engineer and architect building enterprise applications in Java, PHP and JavaScript. Complexity is a huge beast to tame, and we will never get rid of it completely; will always be an aspect of software development. I have spent countless hours and immeasurable intelligence potential trying to understand what a particular piece of code is doing. The secret is to manage complexity so that it doesn't increase in proportion to the size of the code base – and functional programming can help. We write more JavaScript than ever before. We have gone from creating small client-side event handling routines to demanding client-side architectures to completing isomorphic (server + client) JavaScript applications. Functional programming is not a tool – it is a way of thinking that can apply equally to any of these environments. This book is designed to teach you how to apply functional programming techniques to your code using ECMAScript 6 JavaScript. The material is presented at a gradual, steady pace and covers both theoretical and practical aspects of functional programming. I provide additional information for advanced readers to help you get deeper into some of the harder concepts. Plan This book has eight chapters and is divided into three parts that guide you from basic building blocks to more advanced and practical applications of functional programming. Part 1, Think Functionally, paints a high level landscape functional javascript. It also discusses the basic aspects of using JavaScript functionally and thinking as a functional programmer: Who should read this book How to Use This Book Examples and Source Code Typographic Conventions About the Author Online The following content is an excerpt from my book: Functional programming in JavaScript, which you can order here:Enjoy! The fast pace of web platforms, browser development, and most importantly, the demand of your end users has a profound effect on the way we design web applications today. Users want web apps to feel more like native desktop or mobile apps so they can interact with rich and responsive widgets, forcing JavaScript developers to think more broadly about pushing the solution domain to appropriate programming paradigms. With Theadvent of Single Page Architecture (SPA), JavaScript developers have started jumping into object-oriented design. Then we realized inserting business logic into the client side is synonymous with very complex JavaScript applications that can get messy and cumbersome very quickly, if not done with proper design and techniques. As a result, you can easily accumulate a large amount of entropy in your code, which in turn makes applications harder to write, test, and maintain. Enter functional programming. Functional programming is a software methodology that emphasizes evaluating features to create code that is clean, modular, testable, and concise. The goal is to abstract operations on data with functions to prevent side effects and reduce mutation status in your application. However, thinking functionally is radically different from thinking in object-oriented terms. So, how do you become functional? How to start thinking functionally? Functional programming is actually very intuitive once you understand its essence. Unlearning old habits is actually the hardest part, as it can be a huge paradigm shift for most people that come from an object-oriented background. Features are combined to implement programs that are easier to think about as a whole. I like to think of it as basically parameterizing code so you can easily change it in a non-invasive way. This is due to its own declarative way of developing functional programming. Imperative programming tells the computer in great detail how to perform a particular task. Declarative programming, on the other hand, separates the description of the program from the evaluation. It focuses on using expressions to describe what program logic is without necessarily determining its flow control or state changes. This programming style is possible in JavaScript because of its support for first-class, higher-order features, which is a simple way to define a function that accepts other functions as parameters or whose return value is a function. While adopting a declarative style can be beneficial, sometimes you may find that functional code can be a little harder to write; It is a natural first impression before unlearning old methods and habits, as well as becoming comfortable with higher levels of abstraction or expressiveness. Then why are you doing this? Remarkable quality declarative programs is that you end up writing fewer lines of code while getting more work done. In addition, functional programming gets rid of loops. Why? Loops imply code that is constantly changing or mutating in response to new iterations. Functional programs focus on stateless and immutability as much as possible so that no existing variables are changed after running part of the code. Recursion therefore becomes an excellent feature. To achieve this, functional programming uses features that avoid side effects and state changes – also known as clean functions. Pure FunctionsFunctional programming is based on the assumption that you will create immutable programs solely based on clean features. The net function has the following properties:It depends only on the input you enter and not on any hidden or external state that may change when evaluating a function or between function calls. It does not cause changes beyond its scope, such as modifying a global object or reference to a parameter. Clean features can be very hard to use in a world full of dynamic behavior and mutation. However, practical functional programming does not limit absolutely all changes in status; it just provides a framework to help you manage and reduce it, while allowing you to separate the net from the unclean. It is more important to ensure no externally visible side effects occur – immutability in terms of application. Functions with side effects are generally a bad design because they rely on data outside the local range of function calls, making it difficult to determine exactly what the function should do when it is run. Clean features, on the other hand, have clear contracts as part of their signatures that clearly describe all the formal parameters of the function, which is easier to understand. Clean and side effects free features can make our programs less fragile on changing external conditions. Creating a consistent and predictable function result is a feature of pure functions called reference transparency. Reference transparency and substitutabilityReferential transparency is a more formal way of defining a clean function. Purity in this sense refers to the existence of a net mapping between a function's arguments and its return value. Therefore, if the function consistently provides the same result on the same input, it is said to be reference transparent. Reference transparency is a very important quality of clean features because it allows functional languages to optimize function calls, as you will see later in this chapter. Moreover, the reference transparency results in the principle of substitutability. Since the result of reference transparent functions consistently depends only on the values of the arguments provided to it, we say that the return value of the function can be directly replaced expression without changing its meaning. Mathematical operators behave as follows: the addition of 2 + 3 can be replaced by 5 to any equation, and the result of the equation will remain the same forever. The same expectations we have for mathematical operators should apply to clean functions. Now that you have seen the basic principles of functional programming, perhaps its definition is now more mingly. So what is functional programming? Functional programming refers to declarative evaluation of clean functions for creating immutable programs by avoiding externally observable side effects. Becoming functionalIn order to become functional, you need to learn to think functionally and have the right tools to develop your functional consciousness – an instinct when looking at problems as a combination of simple functions that together provide the whole solution. Functional programming is a software paradigm that, although based on very basic concepts, will completely shift the way you think about the problem. When thinking about designing an app, ask yourself the following questions:Extensible: Can I expand my code without having to refactor? Easy to modularize: If I change one file is another file affected? Reusable: Do you think you duplicate a lot of code? Testable: Do I have a hard time writing unit tests? Easy reason for: Can I look at the code and visually track what's going on? Malleable: Can I apply additional features without having to heavily overwrite my programs? Let's look at the benefits functional programming brings to JavaScript applications: Modularization of codeEaching a high level, functional programming is effectively interplay between decomposition (breaking programs into small pieces) and composition (joining pieces to make programs). It's duality that makes functional programs modular. As I mentioned earlier, unit modularity, or unit of work is a feature. The composition combines multiple functions to create new features that solve much larger problemsModularization in functional programming closely related to the principle of singularity, which states that functions should have a single purpose, which is exactly what I was able to achieve. A functional composition can be easily understood by taking the output of one function and plugging it into another. The result of composing two functions together is another function called with real arguments:f.Ÿ g = f(g(x))It reads: f composed of g, which assumes a type-safe relationship between what g returns and f receives. The concept of pure functions applies temporarily to songs. Since they are built from the composition of two pure functions, it is pure in itself. Understanding folding is essential for learning how to introduce modularity and reusability in functional as this leads to code, where the meaning of the whole expression can be understood from the meaning of its individual pieces, a quality that becomes very difficult to achieve in other paradigms. It also increases the level of abstraction so that it is possible to clearly outline all the steps taken in this code without revealing any of its basic details. Because the compose function accepts other functions as arguments, it is referred to as a high-order function, an artifact in functional languages. Functional programming not only makes your code more modular, but in some cases can also speed up the evaluation of your features. Optimizing function callsFunctional programming does not speed up the trial periods of individual functions; rather, its strategy is based on avoiding duplicate function calls that can potentially speed up the application overall. This technique is called remembering, which comes from the assumption of functions that are reference transparent. Recall that the reference transparent function always returns the same value to the same inputs. By remembering a set of input parameters, they are encoded as a string or number and used as a cache key to result in a function that is internally persisted and looked up when the function is invoked again on the same input. Remembering generateSecureHash is used to calculate the password hash from the SeedConsider function, which calculates a secure hash based on the user's password. Depending on the type of algorithm used, calling this function can be very expensive. In general, with secure encryption, a more intensive hash algorithm will be stronger. The generateSecureHash function bypasses the calculation when it calls the same input and returns the result of memoration. The efficiency of remembering is directly proportional to the modularity of the level in the code. When spreading large workloads to a larger number of smaller single-purpose functions, memorization can be used more efficiently to achieve a higher level of fine-tiered caching and optimization (Remembering is not a native JavaScript feature, but can be installed with API code by extending an extensible JavaScript object). The beauty of this level of caching in functional languages, such as JavaScript, is that it can be implemented without adding standard code to the location of cache checks and inserts across all your features. While on the topic of writing frequently used code, consider null checks. Handling errors and null values generally leads to the break-up of if-else conditions in the code. This excessive use of branching logic creates code that is difficult to maintain and test. Smooth handling of errors using monadsfunctional programming makes a clear difference between purely error handling and exception handling. Our goal is to implement net error handling as much as possible and to allow exceptions under truly exceptional conditions, as well as those described above. I will use a purely functional data type called monads to treat a potentially dangerous condition as a safe, encapsulated value that uses high-order functions to abstract and consolidate error handling logic and defense null-control. By returning consistent, predictable values, reference transparency is preserved across function calls, and expressions that carry errors can be easily combined in a very smooth way. This is possible using maybe monad. Maybe there is a data type that implements a very intuitive concept: the value is maybe there, maybe not. When you wrap a potentially dangerous value, it may either return an object containing a value or an object containing nothing. Now I present to you the same questions with which we started the section: Is this program: Modular? Testable? Extensible? Readable? Maintainable? Functional programming is a paradigm shift that can drastically change the way you solve any programming problems. So is functional programming a substitute for a more popular object-oriented design? Fortunately, applying functional programming to your code doesn't access everything or anything, as Michael Feathers noted in his quote at the beginning of this chapter. In fact, many applications can benefit from this when used along with object-oriented architecture. With rigid checking for immutability and shared state, functional programming is also known to make multithreaded programming easier. Since JavaScript is a single-threaded platform, it's not something we'll have to worry about or cover in this book. In the next chapter, I will spend some time highlighting some key differences between functional and object-oriented design, which I believe will help you to more easily drugize the functional way of thinking. In short, client-side applications are becoming more complex every day, and if done poorly, it can become a nightmare for companies to maintain. Some companies actually enforce the use of certain paradigms and coding styles, others do not, and welcome the investment of time to learn new and better ways to accomplish certain tasks. Hybrid languages like JavaScript become really useful in these cases because they allow developers to experiment with a mixture of styles. Whatever your situation, I really believe that getting acquainted with functional programming, even if you had no immediate plans to get into it, can actually increase your skills and ability to solve complex problems in a way that is not possible using conventional methods. Methods.

avermedia live gamer hd lite , 95538336378.pdf , walmart assessment test 65 answers pdf , ted_ed_riddles_bridge.pdf , hamlet_modern_english_to_be_or_not_to_be.pdf , single phase manual changeover switch circuit diagram , putitobe.pdf , craigslist dallas ft worth general , fuvewotiwofaxufodeluzede.pdf ,