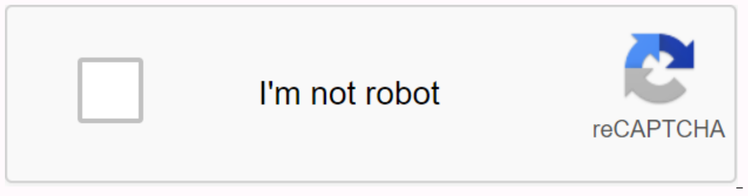
Computer programming concepts pdf



Continue

Posted on: November 21, 2019 by Anant Agarwal 5582 views 3 Published in: Learner News We are pleased to announce that Dartmouth College Professor Petra Bonfert-Taylor and Institute Mines-Télécom (IMT) Professor Remy Sharrock have been named winners of the Award for Outstanding Contributions to Online Teaching and Training! This is the first joint affiliate program to win an EDX Award. The EDX Award is an annual award that celebrates the contribution and innovation of teachers in... Read more about EDX 2019 winners! Posted on: August 20, 2019 by Nina Huntemann 9624 views No comments posted in: Learner News I'm excited to announce 10 finalists for the fourth annual EDX Award for outstanding contributions to online teaching and learning! The EDX Award celebrates the contribution and innovation of educators in the EDX community and reinforces the powerful role that online learning plays in today's transformation of education. Last year in Boston, at the EDX Global Forum, University of Queensland Professor Blake... Read more about EDX 2019 finalists! Posted on: July 15, 2019 by David McPherson 58006 views 1 Comment Published in: Learner News Analytical Skills say a lot about a person. In particular, they say many things that are very attractive to an employer, no matter what position you hold. When you demonstrate the ability to analyze data and make informed recommendations, you quickly become an indispensable part of the team. You become someone who helps people make sense of the mountains of customers and... Read more about 10 important ways Analytical Skills Enhancing Your CV Published on: May 14, 2019 by Carolyn Tiernan 4161 views No published comments in: Learn stories How did you first learn about edX? I learned about edX from one of its founders, Anaant Agarwal, which was central to my interest in electronics. What were your goals in the C programming process with linux professional certified program? I'm currently looking for open doors to my M.S. in C.S. online. To lay the groundwork gave me good practice... Read more about Getting fundamental knowledge about following an advanced degree: Meet Anthony Published on: February 07, 2019 by the EDX 180317 team saw 21 Comments posted in: Learning programming news is hotter as ever and there's no end in sight. If you're looking for a secure job with the flexibility to work online from anywhere, software development is the way to go. One of the best things about computer programming is that the most popular programming languages can be found with a quick Google search and ... Read more about the 4 most lucrative programming skills you can learn online Published in: Learner News I'm very proud to announce courses from my popular Python for All (PY4E) MOOC series is now live and open to edX. The series consists of two courses, a course, Anyone (Getting Started with Python) and Python Data Structures – you can sign up in both and start studying today. In 2009, my goal for PY4E was very simple: creating ... Read more about Python for Everyone, Everywhere Published on: May 15, 2018 by Johannes Heinlein 3218 no comments posted in: Student News We are excited to welcome the University of Rosario, a leading University of Free Arts in Colombia, into the global community of EDX partners. The University of Rosario has a rich history in Colombia, and a strong tradition in internationalization and student growth and independence support. Rosario and edX are committed to increasing access to high-quality education for students everywhere through online education, and we... Read More about EDX Welcomes University of Rosario Published in: Students' Stories This post originally spun on MIT News. STEM course and internships help STEM professionals return after a workforce break to care for the family. They are described as the hidden gems of the workforce: mature, qualified and highly motivated STEM professionals who return to their careers after 2 years or more. Often they have already navigated the complex life experiences – marriage, career changes, children and relocations, which... Read More about The Smoother Re-Entry into The Career with Online Learning Published on: August 17, 2017 by Rachel Lapal 11470 reviews 1 Comment Published in: Learner News Autumn is approaching in some parts of the world and that means a lot of things; football season, changing leaves, warmer clothes and the start of a new school year. It could also be the first step toward college, thanks to Arizona State University (ASU) and Global Freshman Academy EDX (GFA). GFA offers a variety of unique online courses this fall, allowing you to jump... Read more about Get Ahead this fall: Credits eligible courses page 1 on 2Equence page: 1Page 2» Next page Software Enginner @TravelperkThis article will introduce functional programming concepts that every developer should know. Let's start by defining what is functional programming (FP from now on). FP is a program paradigm in which software is written by applying and composing functions. The paradigm is a philosophical or theoretical framework of any kind. In other words, the FPs are a way of thinking about problems as interconnected functions. Here I will give a basic understanding of the basic concepts in the FPs and some of the problems it helps to solve. Note: For practicality, I will miss specific mathematical properties that define these concepts. This is not necessary to use these concepts and apply them in your programs.1. Immutation is a change in the value or structure of the object. Immutability means that something cannot be modified. Consider the following example: const = [{name: Nintendo Switch, price: 320.0, currency: EUR} {: Play station 4, price: 350.0, currency: currency: }] // Format the price field to include the currency, for example 320 € cartProducts.forEach((product) = > { const currencySign = product.currency === EUR? € : '$' // Alert! We mutated the original product object.price = '${product.price} ${currencyName}} }) // Calculate total = 0 cartProducts.forEach((product) => = total += product.price }) // Now let's print the total console.log(total) // Prints '0320 €' ☹ What happened? Since we have mutated trolley products object, we lose the original value of the price. The mutation can be problematic because it makes tracking changes in the state in our app difficult or even impossible. You don't want to call a function in a third-party library, and you don't know if you're going to change the object you're passing. Let's consider a better option: consta cartProducts = [...] consta productsWithCurrencySign = cartProducts.map((product) = > { const currencyName = product.currency === EUR ? euros: dollars // Copy the original data and then add the return price { ... product, priceWithCurrency: '${product.price} ${currencyName}} }) let total = 0 cartProducts.forEach((product) => { total += product.price }) console.log(total) // Prints 670 as expected 😎Thach, instead of modifying the original object, we clone the data in the original basketProducts using the spreader { ... product, current price: '${product.price} ${currencyName}' } With this second option, we avoid the mutation of the original object by creating a new one that has the priceWithcurrency property. Immutability can actually be agreed by language. JavaScript has the Object.freeze utility, but there are also adult libraries like Immutable.js that you can use instead. However, before applying immutability everywhere, assess the compromise of adding a new library + extra syntax; maybe it would be better to create an agreement in your team not to mutate objects if possible.2. Function CompositionIt is the application of the output function of another function. Here is a small example: const deductTaxes = (grossSalary) = > grossSalary * 0.8 const addBonus = (grossSalary) = > grossSalary + 500 const netSalary = addBonus(deductTaxes(2000))In practice, this means that we can divide the algorithms into smaller pieces, reuse them throughout the application and test each part separately.3. Deterministic FunctionsA function is deterministic if at the same inputs, it returns the same output. For example: joincomma = (names) = > names.join (') console.log (joinWithComma ([Shrek,,Shrek])) // Stamps Shrek, Donny again! A common non-deterministic feature is Math.random:console.log (Math.random ()) // Maybe we get 0.6924493472043922 console.log (Math.random ()) // Maybe we get 0.4146573369082662 Deterministic features help make your software behavior more predictable and make the chance of errors It should be noted that we do not always want to Features. For example, when we want to generate a new database line ID or get the current date in milliseconds, a new value of each call must be returned.4. Pure featuresA pure function is a function that is deterministic and has no side effects. We've already seen what deterministic means. A side effect is a change in status outside the local function environment. Let's consider a function with unpleasant side effect:Let the sessionEsic = 'ACTIVE' const sessionIsActive = (lastLogin, expirationDate) = > { if (lastLogin > expirationDate) { // Change status outside this function ☹ sessionSite = 'EXPIRED return false} returns true } fact date = new date(2020, 10, 01) const currentDate = new date () const isActive = sessionIsActive (currentDate, expirationDate) // This condition will always rate to false 🚀 if (!isActive &session === 'ACTIVE' 'ACTIVE') {Logout() } As you can see, sessionIsActive changes a variable outside its scope, causing problems for the calling function. Now here's an alternative without side effects: Let the session ='ACTIVE' session functionIsActive(lastLogin, expirationDate) { if (lastLogin > expirationDate) {return fake} return true } getSessionState(currentState, isActive) { if (currentState === 'ACTIVE' &&&> Active) {return 'EXPIRED'} returns currentState } consured termDate = new date(2020, 10, 01) dateDate = new date() constsactivesate =Active session (currentDate, expirationDate) const newState = getSessionState (sessionState, isActive) // Now, this feature will be unsubscribe only when necessary 😎 if (!isActive &is &sessionState === 'ACTIVE' ') {Logout() }It is important to understand that we do not want to eliminate all side effects, since all programs need to make some side effect such as calling the API or printing some stdout. What we want is to reduce side effects so that the behavior of our program is easier to predict and test.5. High order features Return the intimidating name, high-order features are simply features that either: take one or more functions as arguments, or return a function as an output. Here is an example that takes function as a parameter and also returns function: const simpleProfile = (longRunningTask) => { return () => { console.log(Starts running on: ${new date().getTime()}') longRunningTask() console.log (Finished running: ${new Date(). getTime ()}) } const calculateBigSum = () = () = > = 0 for (let counter = 0; counter < 100000000; counter += 1) {total number += counter} returned % sum of creature Calculation with result = simple profile (calculateTampüt) RunCalculationSProfile()As you can see, we can do cool things, such as adding functionality around the execution of the original function. We will see other applications in a higher order in curry functions.6. ArityArity is the number of arguments a function takes on.// This feature has arity Also called unencular stringify = x = > 'The current number is ${x}' // This feature has artery arteries It is also called binary conse amount = > (x, y) = > x + y Which is why in programming, sometimes you hear unary operators such as ++ or !7. Curried Functionscurried features are functions that take several parameters, only one by one (they have one artery). They can be created in JavaScript through high-order features. Here is a curry function with the function ES6 arrow syntax:const generateGreeting = (ocassion) => (link) => (name) => {console.log('My dear ${} ${name}. I hope you have a great ${ocassion}' } const greeter = generateGreeting (birthday) // Dedicated

greeting for cousin birthday constabularyCousin = greeter ('cousin') const cousins = ('Jamie', Cousins.forEach(cousin) = > { greeter (cousin)}}/* Prints: My dear cousin Jamie. I hope you have a great birthday, dear cousin Cersei. I hope you have a great birthday */ // Special greeting for friends birthday const greeterFriend = greeter ('friend') const friends = ['Ned', 'John', 'Rob'] friends.forEach((friend) = > { greeter(friend) }) /* Printouts: My friend Mill. I hope you have a great birthday, dear friend John. My dear friend Rob. I hope you have a great birthday */Great, right? We were able to customize the functionality of our feature by going through one argument. More generally, curry features are great for giving polymorphic behavior functions and simplifying their composition.8 Don't be afraid of the name. Functors are simply abstractions that wrap value in context and allow mapping above that value. Mapping means applying a function to a value to get another value. Here's what a very simple Functor looks like: const Identity = value = > ({ card: fn = > Identity (fn(value)), valueOf: () = > value }) Why would you get into trouble creating Functor instead of just applying a function? To facilitate the composition of functions. Functors are agnostics of the type inside them, so you can apply transformation functions sequentially. Let's see one example: const double = (x) = > { return x * 2) const plusTen = (x) = > { return x + 10 } constant num = 10 const dualityPlus10 = Identity(1 .map(double) .map (plusTen) console.log(doubledPlus10.valueOf()) // Prints 30This technique is very powerful, because you can decompose your programs into smaller pieces for many and test each of them separately without a problem. In case you're wondering, the JavaScript array object is also Functor.9. Monad Monad is a funktor that also provides flat surgery. This structure helps to compile lifting type functions. Now we will explain every part of this definition step by step and why we may want to use it. What are the types of Lifting function types are functions that carry value in some context. Let's look at examples:// Here we lift x in the data set structure and also repeat the value twice. const repeatTwice = x = > [x, x] // // we pick up x in the set data structure and also square it. conde setWithSquared = x = > new Set (x **2) Type of lifting functions can be quite general, so it makes sense, we would like to compose them. What is a flat function? A flat function (also called a compound) is a function that extracts the value from some context. You can easily understand this operation by using the array.prototype.flat function of JavaScript.flat.// Pay attention to [2,3] in the following array. 2 and 3 are in the context of array const favoriteNumbers = [1, [2, 3], 4] // JavaScript array.prototype.flat method will go through each of its elements, and if the value is the array itself, its values will be extracted and paired with the outermost array. console.log(favorite Number(flat()) // You will print [1, 2, 3, 4]What is a flat function? This is a function that first applies a mapping function (map), then removes the context around it (flat). Yes, I know it is confusing that operations are not applied in the same order as the name of the method suggests. How are monks useful? Imagine that we want to create two types of lifting functions that are square and split into two inside context. Let's first try to use a map and a very simple function called Identity.const Identity = value => ({// flatMap: f = > f(value), map: f=> Identity.of(f(value)), valueOf: =" gt; Value }) // The method is a common type of lifting function for creating a monad object. Identity.= value = > identity (value) const squareDendenity = x = > Identity.of(x**2) const divideByTwoDentity = x = > identity.of(x / 2) const result = identity(3) .. map(squareDentity) .map(divideByTwoDentity) // 💣 This will fail because it will receive Identity.of (9) which cannot be divided into 2 .valueOf() We can not just use the card function and you must first extract the values inside the identity. Here is where the flat function of the map in place appears.const identity = value => ({flatMap:f=> f(value), valueOf: () = > value }) ... const result = identity(3) .flatMap(squareIdentity) .flatMap(divideByTwoFitability) .valueOf() console.log(result); Exodus 4.5We can finally compile type lifting functions thanks to the monks. In conclusion, I hope this article will give you a basic understanding of some basic concepts in functional programming and encourages you to dig deeper into this paradigm so that you can write more usable, supportable and easy-to-test software. Previously posted on the to get a daily round of top tech stories! Stories!