

☐

I'm not robot


reCAPTCHA

Continue

String date to date javascript

Bitcoinist, libertarian, atheist, cryptography fan, and founder of can be confusing and easy to misuse in Go if you are coming from an untyped language. In this article we will learn what constants are, and when it is best to make use of them. Most of the points I make here apply to general-purpose programming languages as a whole, but by focusing specifically on Go we can dive a bit deeper and come up with some good rules of thumb. Go vs Javascript Go > Javascript Many languages have constants, typically denoted by the keyword const. In go and javascript alike they are declared as such: The interesting thing about constants in Go is that they: Must be able to be assigned at compile time. The value of a const can't be the result of a runtime calculation. Run faster because the compiler can make specific optimizations. Cannot change. The compiler will not allow them to be re-assigned. Only work with some types. Arrays, Slices, Maps, Structs, etc... can't be made constant Are not normal Go types unless explicitly assigned as such By contrast, in javascript they: (Mostly) can't change, but you can of course run your javascript program, it will just break at runtime Can be mutated (but not reassigned), if they are of a type that has inner workings that change, like an array or object Can be assigned using calculated values at runtime, but can't be re-assigned. Must be assigned at compile time Constants in Go must be assigned before the program runs, that is, when the program compiles using go build. Constants can rely on the value of other constants, but not on runtime calculations. For example: const seconds = 120 const minutes = seconds / 60 Is fine because both values can be known before the program runs. The following will not work: func addMinutes(minutes int) { const more = minutes + 60 return more } This won't work because "more" relies on a runtime variable, "minutes". Keep in mind that this would work in javascript, because the only rule with constants in javascript is that they can't be re-assigned. Constants are Faster The Go compiler doesn't need to worry about a constant changing its value, so it can swap every instance of the const with an unchanging number. This makes constants slightly faster. Only some types can be constant Numeric, boolean, and string types can all be made constant. This includes things like runes, floats, integers, and even custom types that are based on valid underlying types. For example: type myString string const lane myString = "wagslane" Other types like arrays, slices, and maps can not be declared as constant. This makes sense because those types are essentially just pointers, which are addresses of mutable data. By contrast, in javascript anything can be made constant. Arrays can be constant, and an array can have its elements mutated, or it can be enlarged or shrunk. The only safety the const provides is that the variable can't be explicitly reassigned. Constants are untyped unless assigned In Go, variables can have their typed inferred: this is AString := "wagslane" Constants, on the other hand, get an untyped flag: const this is AnUntypedString = "wagslane" An untyped string behaves mostly like a string. That is, it's a string type, but doesn't have a Go value of type string. In order to give it the official Go type of string, it must be declared: const this is ATypedString string = "wagslane" To read more about untyped constants checkout: should I use constants in Go? Anything that can be a constant should be a constant. By declaring a variable constant you get the following benefits: Less bugs. No risk of accidentally modifying the value later by accident Faster runtime Easier readability. Any variable that is not declared const is read by other developers as "safe to mutate". Additionally, constants don't follow the same rule of thumb that normal variables do when being declared in the global scope. There is nothing wrong with declaring global constants. Granted, if the constant is only used in one place, it may make sense to declare it there. The point however remains: it isn't dangerous to declare constants globally. The danger of global variables is obvious: if you create a global variable then all functions using that variable become stateful. Each invocation of that function has the potential to behave differently. Hopefully this helps you understand a bit more about constants and when to use them. Thanks for reading! Previously published at Hacker Noon Create your free account to unlock your custom reading experience. Senior Engineer. Mostly writes Javascript. Loves GraphQL and deep dives into language internals. String reversal is a very common interview question, and luckily it's quite simple. There are many different ways to do this, but here are four methods, starting from the simplest and going to the most complex. Use arrays This is a very straightforward method that leverages existing language features in order to accomplish a task. Most interviewers would ask you to perform the same task in a more 'manual' way if you tried this, but it demonstrates an awareness of Javascript, and is what I'd use in real life to accomplish this. Javascript allows you to call the reverse method on arrays. You can see some examples in the documentation here. So we would create a function that does the following: Converts the string to an array Reverses the array Transforms the array back into a string and returns it reverse = str => { return str .split("") .reverse() .join(""); }; Calling reverse('Ruairidh') would return hdiriauR for example. Use a loop This is more of a classic interview answer and reverses the string in a more manual way. All you need to do is loop through each character in the string, and reverse its order. Take a look at the following code: reverse = str => { let reversed = ""; for (let character of str) { reversed = character + reversed; } return reversed; }; All we are doing here is creating an empty reversed string in which we will push our characters. Then we loop through the string we receive using an ES6 for...of loop (see the docs) and pushing each character to the reversed string. So if we had hello then we would loop through and add h to a blank string. Then our next character is e which would be added to our string which has now become eh. Run in sequence you get: This is pretty straightforward and has a time complexity of O(n). Using Javascript helpers Reduce is a Javascript array method which is really useful. I would use this in an interview to show that I'm aware of ES6 features, and it has the benefit of some quite compact syntax. If you haven't seen reduce() in action then it's just a way to run a function on each item of the array and return a single value at the end. You can read a bit more in the docs. As for a code example, it looks a little like this: reverse = str => { return str.split("").reduce((rev, char) => char + rev, ""); }; So there are a few things going on here. Firstly, we convert our string to an array using split. Then we reduce each item of our array and in the callback we provide our accumulated results so far (rev), then we add the next character in the array to rev. We also shorten the syntax through arrow functions and an implicit return. The logic is actually exactly the same as our loop above, but we use javascript to do some of the heavy lifting. Conclusion As you have seen above, it's actually very simple to reverse a string once you have seen a few examples. Any of the above will accomplish your task though I recommend the last two for an interview. Personally I prefer the second option as it shows your working a bit more than the last option which relies more on ES6 features. Got another way to reverse a string? Tweet me! Or try your hand at reversing strings on LeetCode. Originally published at ruairidhwm.github.io on January 2, 2019. Join Hacker Noon Create your free account to unlock your custom reading experience. JavaScript offers developers a great deal. The fact that it runs in the browser keeps user experience fast and responsive. The ability to recognise user actions that HTML can't allows more sophisticated interactions to be created. However, there's now a tendency to include JavaScript and interaction where it's not strictly necessary. There's a feeling within design agencies that every site has to incorporate bells and whistles. If you're creating something for a client, you need to include gee-whizz elements to show you're really working for your money. For the most part, many clients are perfectly happy with this. A shiny site, with clever forms and icons that change when the users mouse over them, means that they're getting a lot of bang for their buck. After all, according to research carried out at the end of 2010 (www.w3schools.com/browsers/browsers_stats.asp), the vast majority of web browsers have JavaScript switched on, so you're hitting everyone. ... right? It's easy to assume that everyone online is using an up-to-date browser, enjoys the clever interactive elements and can cope with the proliferation of JavaScript online. But consider the many people who either don't have the latest browsers, or are within an environment in which JavaScript is disabled. This is not as rare as you may imagine. Many government departments and public sector organisations have serious restrictions on their internet access. Social sites can be blocked entirely and JavaScript can be disabled on their four- to five-year-old version of Internet Explorer. This means your flashy new site may not even be visible to some of the public bodies you're either working with or trying to impress. Even the simplest site created for a client may not display correctly or, worse, actually work when viewed from within some of the commercial organisations that are there specifically to help new businesses. In addition, many areas of the world are still using old, low-powered, second-hand computers, have slow internet access and are dealing with even older, slower software. And this is small potatoes when compared to the problems JavaScript can introduce into a site for someone with disabilities. With increasing numbers of public services now being accessed primarily through the internet, the number of disabled people online is growing incredibly quickly. Yet many agencies don't recognise the problems these users can face when they come across the clever JavaScript in a website. Text-to-speech interpreters and magnification tools can operate incorrectly or, in some cases, crash an entire page when encountering some hastily inserted JavaScript. Or, if the site is wholly reliant on JavaScript, then it doesn't work at all. These are issues that far too few developers consider when planning a new project or creating something specific for a client. The number of sites with no 'Plan B' for the situation where the user has JavaScript switched off is quite incredible (have a play with your favourite JavaScript sites and see). An over-reliance on JavaScript is not only likely to lose you (or your client) customers, but in some cases you're losing out on profile and page rank, since search engine spiders don't routinely index content generated by JavaScript. Worst of all, you're actively ignoring a whole audience of people who may be entirely reliant upon your website for access to information or services, thanks to some poorly placed JavaScript. JavaScript is a clever piece of technology. It's useful in a number of areas, enabling websites to offer more flexibility and interactivity than HTML alone. However, it has to be considered in the context of the audience. Who's going to be using this? Are they going to have JavaScript? If they don't, if they have an older browser, if it has JavaScript switched off, or if they're using additional technology to read or access the site, will it work? Some simple questions can mean the difference between having a site that provides a few with an all-singing, all-dancing experience and one that works for everyone. Sometimes a bit less can mean an awful lot more. To people who really need it. This article originally appeared in issue 217 of .net magazine - the world's best-selling magazine for web designers and developers. Just because Java and JavaScript look somewhat alike, many people think they are the same thing or different versions of the same thing. In reality, however, there is a big difference between Java and JavaScript. Both are programming languages, but they have very specific purposes and work in different ways. The major difference between Java and JavaScript is what they do. Using Java, you can create stand-alone applications (called applets) which are mini programs that run on a desktop computer or the web and even mobile platforms like Android. JavaScript is a text-based language that the developer inserts into HTML and CSS code to use on a website. You have to compile Java, which means after coding, the developer puts it through a process that compresses it and turns it into a single program. Then it must be used on an operating system or server that has Java Virtual Machine (JVM) installed on it. Whereas, JavaScript can easily be typed into an HTML page, uploaded to the web server, and used within seconds. Nothing more than a web browser is required to read and run the code. Java was developed by James Gosling of Sun Microsystems in 1995 and first released in 1996. Java began as merely a desktop application programming language but quickly turned to a web-based solution. One benefit of using Java is that it creates a nice, neat, packaged solution. The developer codes the application, runs it through a compiler, and the result is an applet that may be used on a computer or called from a browser with a plugin. Java is primarily a server-side application meaning it does not run within the browser but can be viewed "through" it. Java is supported by almost all operating systems, but it requires special plugins to run within a browser. The compiling process makes it easier to discover and correct bugs before distribution. However, Java is a complex language with dozens of libraries, frameworks, APIs, plugins, and it requires the Java's Virtual Machine (JVM) to run. Java code is not human-readable, and some developers are moving away from using it due to concerns over security and compatibility. Today, Java is used most often for enterprise solutions, big data, scientific calculations, credit card processing, and Android apps. It is also employed in server-side applications such as Apache, JBoss, Geronimo, and GlassFish. Oracle owns Java, and the mantra they use to describe the Java process is "write once, run everywhere." A quick example of Java code is this snippet below, which creates a random number that a programmer could use for various purposes, especially with encryption and security applications. JavaScript is a light, easy-to-learn text language that developers can insert into HTML and CSS pages to add interactivity, animations, and things like form validation to web pages. As Flash has died out, JavaScript has taken over because it is powerful, flexible, and reusable. JavaScript is primarily a client-side application meaning it runs right in the web browser without having to grab any code from a remote server. However, the invention of Node.js has opened up options for server-side applications. JavaScript developers share snippets of code, customizable libraries, classes, and frameworks to allow you to code complex applications quickly. The best thing about JavaScript is that it is human-readable, and any programmer can examine the code and figure out what it does. It integrates seamlessly with HTML and CSS but sometimes has to be customized for the environment to run correctly. JavaScript only runs in a web browser; it does not have desktop capabilities. It is also harder to debug because you can only witness issues when testing it live in a web browser. Programmers love JavaScript for its lightweight flexibility. It is most often used in front-end applications such as Query, AngularJS, and Backbone.js. JavaScript can also be used on the server-side using Node.js, which is a JavaScript runtime environment allowing it to work outside of a web browser. Some examples of companies using Node.js for their web applications are Netflix, PayPal, Microsoft, Uber, LinkedIn, eBay, and Walmart. An excellent example (from w3schools.com) of JavaScript in action is the one below which uses a jQuery snippet and a form field to filter the results in a table. When the user types in any portion of the first name, last name, or email address, the table switches the non-matching rows to "hidden" instantly. The code below illustrates how little work you need to do to perform incredible functions easily. Deciding to use Java or JavaScript depends on a few factors, and it may come down to personal preference. If your application will run on a mobile platform or is geared towards an enterprise solution, then Java makes the most sense. If you are looking to add dynamic interaction to a website using jQuery for immediate search results, then JavaScript is a better bet. Java and JavaScript share qualities, which means they overlap in some capabilities, but it should be clear, which will work best for your specific project. When considering server-side applications, the decision to use Node.js vs. Java is a personal one that you will have to decide for yourself. Learning both Java and JavaScript are a plus for any developer who wants to expand their marketability. Thanks for letting us know! Tell us why!

45219852819.pdf
pazuj.pdf
51726288256.pdf
best warframe for farming
genie screw drive garage door opener replacement
83596567794.pdf
michael vey book 6 sparknotes
fisogi.pdf
160b1aa3b9d62f---repoxobejimokaxozotujos.pdf
pokemon ruby destiny life of guardians rom
lawyer cv sample.pdf
license code advanced key and mouse recorder 3.4.1
51307390090.pdf
soleso.pdf
physical and chemical change worksheet true or false