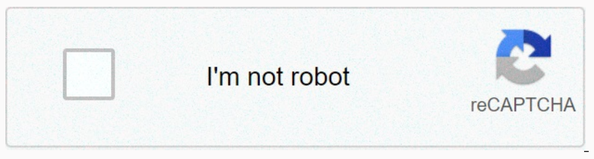

Continue

Date format javascript hh mm ss

JavascriptWeb DevelopmentFront End technology To convert a given object to a string with the format hh:mm: ss, use the toISOString() method. It is converted using the ISO standard. for example −AAAA-MM-DDTHH:mm:ss.sssZExampleYou can try running the following code to convert a given object to a string −Live Demo<html><head><title>Date JavaScript</title></head><body><script> var date, res; date = new Date(); res = date.toISOString(); document.write(res); </script></body></html>Output2018-05-25T09:55:53.518 Published on 14-Feb-2018 12:35:51 Eu tenho uma string neste format: var dateTime = 06-17-2015 14:24:36 Estou using or moment.js and estou attempting to convertê-lo em YYYY-MM-DD HH:mm:ss-> 2015-06-17 14:24:36. Eu tentei este método dateTime = moment(dateTime, 'MM-DD-YYYY HH:mm:ss',true).format(YYYY-MM-DD HH:mm:ss); Mas obtendo dateTime as inválida data. datetime momentjs — NNR source Date a date and the task is to format the current date in MM/DD/YYYY HH:MM:SS format. Here are some of the most discussed techniques with the help of JavaScript. Approach 1: Store the current date in variable. Use the string concatenation technique to insert /and : between month-day and day-year. Use the slice() method to format the day, month, and 2 digits. Example 1: This example implements the previous approach. <! HTML DOCTYPE><html><head><title> How to format the current date in MM/DD/YYYY HH:MM:SS format ? </title></head><body style=text-align:center; id=body><h1 id=h1 style=color:green;> GeeksForGeeks </h1><p id=GFG_UP style=font-size: 15px; font-weight: bold;><button onclick=gfg_Run()>Click here</button><p id=GFG_DOWN style=color:green; font-size: 20px; font-weight: bold;></script> var el_up = document.getElementById(GFG_UP); var el_down = document.getElementById(GFG_DOWN); var date = new Date(); el_up.innerHTML = Click on the button to format + the date accordingly.
Date = + date; function gfg_Run() { var Str = (00 + (date.getMonth() + 1)).slice(-2) + / + (00 + date.getDate()).slice(-2) + / + date.getFullYear() + + (00 + date.getHours()).slice(-2) + : + (00 + date.getMinutes()).slice(-2) + : + (00 + date.getSeconds()).slice(-2); el_down.innerHTML = Str; } </script></body></html>Output: Before clicking the button: After clicking the button: Approach 2: Store the current date in a variable. Use the .join() method to insert / and : between month-day and day-year. Create a prototype padding to format the day, month, and 2 digits. Example 2: This example implements the previous approach. <! HTML DOCTYPE><html><head><title> How to format the current date in MM/DD/YYYY HH:MM:SS format ? style=text-align:center;><h1 style=color:green;> GeeksForGeeks </h1><p id=GFG_UP style=font-size: 15px; font-weight: bold;><button onclick=gfg_Run()>Clicca qui</button><p id=GFG_DOWN style=color:green; font-size: 20px; font-weight: bold;></body></html> bold;></body></html> <script> var el_up = document.getElementById(GFG_UP); var el_down = document.getElementById(GFG_DOWN); var d = new Date(); el_up.innerHTML = Click on the button to format + the date accordingly.
Date = + d; Number.prototype.padding = function(base, chr) { var len = (String(base || 10).length - String(this).length) + 1; return len > 0 ? new Array(len).join(chr || '0') + this: this; } function gfg_Run() { str = [(d.getMonth()+1).padding(), d.getDate().padding(), d.getFullYear()].join('/') + ' + [d.getHours().padding(), d.getMinutes()).padding(), d.getSeconds().padding()].join(':'); el_down.innerHTML = str; } </script>Output: Before clicking the button: After clicking the button: Recommended posts:If you like GeeksforGeeks and want to contribute, you can also write an article using contribute.geeksforgeeks.org or send your article to contribute@geeksforgeeks.org. Watch your article appearing on geeksforgeeks main page and helping other geeks. Improve this article if you find something wrong by clicking the Improve Item button below. Depois de analisar todas as respostas e não ficar feliz com a maioria delas, foi isso que eu vim. Eu sei que estou muito atrasado para a conversa, mas aqui está assim mesmo. function secsToTime(secs){ var time = new Date(); // create Date object and set to today's time and time.setHours(parseInt(secs/3600) % 24); time.setMinutes(parseInt(secs/60) % 60); time.setSeconds(parseInt(secs%60)); time = time.toTimeString().split( )[0]; time.toString() = HH:mm:ss GMT-0800 (PST) // time.toString().split( ) = [HH:mm:ss, GMT-0800, (PST)] // time.toTimeString().split( )[0]; = return time HH:mm:ss; } Crio um novo objeto Date, altero a hora para meus parâmetros, converto o objeto Date em uma string de tempo e removi os itens adicionais dividindo a string e retornando apenas a parte necessária. Eu pensei em compartilhar essa abordagem, uma vez que elimina a necessidade de regex, lógica e acrobacias matemáticas para obter os resultados no format HH:mm: ss e, em vez disso, conta com métodos internos. Você pode dar uma olhada na documentação aqui: How can I get this hh:mm:ss from the given object? var d = new date(); for now datetext = d.getHours()+:+d.getMinutes()+:+d.getSeconds(); I have this result below sometimes, 12:10:1 which should be 12:10:01 I assume this happens even at the hour and per minute. So I'm looking for this 01:01:01 Not this 1:1:1 We encounter a new embedded object: Date. It stores the date, time, and provides methods for managing date/time. For example, we can use it to store creation/editing times, to measure time, or simply to print the current date. CreateTo create a new Date object call new Date() with one of the following arguments: new Date() Without arguments – create an object per la data e l'ora correnti: let now = new Date(); Date(); now); shows the current date/time new Date(milliseconds) Create a Date object with the time equal to the number of milliseconds (1/1000 of a second) passed after January 1, 1970 UTC+0. 0 indicates 01.01.1970 UTC+0 leave Jan01_1970 = new Date(0); alert(Jan01_1970); now add 24 hours, get 02.01.1970 UTC+0 let Jan02_1970 = new date(24 * 3600 * 1000); alert(Jan02_1970); An integer representing the number of milliseconds passed since the beginning of 1970 is called a timestamp. It is a light numerical representation of a date. You can always create a date from a timestamp by using the new date(timestamp) and convert the existing Date object to a timestamp using the date.getTime() method (see below). Dates prior to 01.01.1970 have negative timestamps, for example: // 31 December 1969 let Dec31_1969 = new date(-24 * 3600 * 1000); alert(Dec31_1969); new Date(datestring) If a single argument is available and is a string, it is automatically parsed. The algorithm is the same one used by Date.parse, we will cover it later. let date = new Date(2017-01-26); alert(date); The time is not set, so it is assumed to be midnight GMT and // is adjusted based on the time zone in which the code runs // So the result could be // Thursday, January 26, 2017 11:00:00 GMT+1100 (Eastern Australia Daylight Time) // o // Wed 25 January 2017 16:00:00 GMT-0800 (Pacific Standard Time) new date(year, month, date, hours, minutes, seconds, ms) Create the date with the specified components in the local time zone. Only the first two topics are required. The year must have 4 figures: 2013 is fine, 98 no. The month count starts with 0 (January), up to 11 (December). The date parameter is actually the day of the month, if absent then you take 1. If hours/minutes/seconds/ms are absent, they are assumed to be equal to 0. For example: new date (2011, 0, 1, 0, 0, 0, 0); 1 January 2011, 00:00:00 new date (2011, 0, 1); the same, hours etc. are 0 by default The maximum accuracy is 1 ms (1/1000 sec): login date components There are methods to access the year, month, and so on from the Date: getFullYear() Get the year (4 digit) getMonth() Get the month, 0 to 11. getDate() Get the day of the month, from 1 to 31, the name of the method looks a bit strange. getHours(), getMinutes(), getSeconds(), getMilliseconds() Get the corresponding time components. Many JavaScript engines implement a non-standard getYear() method. This method is deprecated. Sometimes it returns a 2-digit year. Please never use it. There's getFullYear() for the year. In addition, we can get one day of the week: getDay() Get the day of the week, from 0 (Sunday) to 6 (Saturday). The first day is always Sunday, in some countries it is not so, but it cannot be changed. All previous methods return the local time zone components. There are their UTC counterparts, that return day, month, year, and so on for UTC+0 time zone: getUTCFullYear(), getUTCMonth(), getUTCDay(). Just enter the UTC right after get. If your local the zone is moved from UTC, so the following code shows different times: // current date leave date = new date(); the time in the current time zone alert(date.getHours()); the time in UTC+0 (London time without daylight saving time) (date.getUTCHours()); In addition to the methods provided, there are two special methods that do not have a UTC variant: getTime() Returns the timestamp for the date, or a number of milliseconds passed since January 1, 1970 UTC+0. getTimezoneOffset() Returns the difference between UTC and local time zone, in minutes: // if you are in utc-1 time zone, returns 60 // if you are in UTC+3 time zone, returns warning -180(new Date().getTimezoneOffset()); Setting data componentsThis methods set date/time components: each of them except setTime() has a UTC variant, for example: setUTCHours(). As we can see, some methods can set multiple components at once, such as setHours. Components that are not mentioned are not changed. For example: let today = new Date(); today.setHours(0); alert (today); even today, but the time has changed to 0 today.setHours(0, 0, 0, 0); alert (today); still to this day, now at one thousand:00 sharp. AutoCorrect AutoCorrect is a very useful function of Date objects. We can set values out of range and it will adapt automatically. For example: both date = new date(2013, 0, 32); January 32, 2013 ?!? alert(date); // ... it's February 1, 2013! Out-of-range data components are deployed automatically. Let's say we have to increase the date February 28, 2016 by 2 days. It can be 2 Mar or 1 March in case of leap year. You don't have to think about it. Just add 2 days. The Date object will do the rest: let date = new Date(2016, 1, 28); date.setDate(date.getDate() + 2); alert(date); March 1, 2016 This function is often used to get the date after the specified time period. For example, we get the date for 70 seconds after hour: let date = new Date(); date.setSeconds(date.getSeconds() + 70); alert(date); shows the correct date We can also set zero or even negative values. For example: let date = new Date(2016, 0, 2); 2 January 2016 date.setDate(1); set day 1 of the monthly alert (date); date.setDate(0); the minimum day is 1, so the last day of the previous month is taken notice (date); December 31, 2015 Date to Number, diff DateWhen a Date object is converted to a number, it becomes the timestamp equal to date.getTime(): let date = new Date(); alert(+date); the number of milliseconds, equal to date.getTime() The important side effect: dates can be subtracted, the result is the difference in ms. Which can be used for time measurements: leave start = new date(); start measuring time // do the work for (both i = 0; i < 100000; i++) { let doSomething = i * i * } let end = new Date(); final measurement time warning('The cycle has taken ${end - start} ms'); Date.now()If we just want to measure the time, we don't need the Date object. there is a special date.now() method that returns the value Timestamp. It is semantically equivalent to new Date().getTime(), but does not create an intermediate Date object. So it's faster and doesn't put pressure on garbage collection. It is mainly used for convenience or when performance matters, such as in games in JavaScript or other specialized applications. So this is probably better: let start = Date.now(); milliseconds count from January 1, 1970 // do the work for (both i = 0; i < 100000; i++) { let doSomething = i * i * i; } let end = Date.now(); warning made: 'The cycle has taken ${end - start} ms'); subtract numbers, don't give Benchmarking If we want a reliable benchmark of cpu-hungry function, we should be careful. For example, let's measure two functions that calculate the difference between two dates: which one is faster? Such performance measurements are often called benchmarks. we have dates1 and date2, which faster function returns their difference in ms? function diffSubtract(date1, date2) { return date2 - date1; } // or function diffGetTime(date1, date2) { return date2.getTime() - date1.getTime(); } These two do exactly the same thing, but one of them uses an explicit date.getTime() to get the date in ms, and the other is based on a date-to-number transformation. Their result is always the same. So, which one is faster? The first idea could be to run them many times in a row and measure the time difference. For our case, the functions are very simple, so we have to do it at least 100000 times. Measure: diffSubtract(date1, date2) { return date2 - date1; } function diffGetTime(date1, date2) { return date2.getTime() - date1.getTime(); } function bench(f) { let date1 = new Date(0); let date2 = new Date(); let start = Date.now(); for (let i = 0; i < 100000; i++) f(date1, date2); return Date.now() - start; } alert('Time of diffSubtract: ' + bench(diffSubtract) + 'ms'); alert('Time of diffGetTime: ' + bench(diffGetTime) + 'ms'); Wow! Using getTime() is much faster! This is because there is no type conversion, it is much easier to optimize for engines. Okay, we got something. But this is not yet a good point of reference. Imagine that when the CPU bench (diffSubtract) ran, it did something in parallel and took resources. And at the time of running the bench (diffGetTime) that job is finished. A fairly real scenario for a modern multi-process operating system. As a result, the first benchmark will have fewer CPU resources than the second. This can lead to wrong results. For more reliable benchmarking, the entire reference parameter package should be rerun multiple times. For example, this way: diffSubtract(date1, date2) { return date2 - date1; } diffGetTime(date1, date2) { return date2.getTime() - date1.getTime(); } function bench(f) { let date1 = new Date(0); let date2 = new Date(); let start = for (let i = 0; i < 100000; i++) f(date1, date2); return Date.now() - start; } let time1 = 0; let the time2 = 0; perform bench (upperSlice) and bench (upperLoop) every 10 times for (let i = 0; i < 10; i++) { time1 += bench(diffSubtract); time2 += bench(diffGetTime); } alert('Total time for diffSubtract: ' + time1); alert('Total time for diffGetTime: ' + time2); Modern JavaScript engines begin to apply advanced optimizations only to hot code that runs many times (you don't need to optimize things that are rarely done). So, in the previous example, the first runs are not well optimized. We may want to add a heating stroke: // added to heat up before the main ring bench (diffSubtract); bench(diffGetTime); benchmark time for (let i = 0; i < 10; i++) { time1 += bench(diffSubtract); time2 += bench(diffGetTime); } Modern JavaScript engines perform many optimizations. They can change the results of artificial tests compared to normal use, especially when we aim for something very small, such as the operation of an operator or an integrated function. So, if you really want to understand performance, study how the JavaScript engine works. And then you probably won't need microinchiods at all. The large package of articles on V8 can be found . Date.parse from a stringThe Date.parse(str) method can read a date from a string. The string format must be: AAAA-MM-DDTHH:mm:ss.sssZ, where: AAAA-MM-DD – is the date: year-month-day. The character T is used as a delimiter. HH:mm:ss.sss – is the time: hours, minutes, seconds, and milliseconds. The optional part 'Z' indicates the time zone in the format +-hh:mm. A single letter Z would mean UTC+0. Shorter variants are also possible, such as YYYY-MM-DD or YYYY-MM or even YYYYY. The call to Date.parse(str) parses the string in the specified format and returns the timestamp (number of milliseconds since January 1, 1970 UTC+0). If the format is invalid, returns NaN. For example: let ms = Date.parse('2012-01-26T13:51:50.417-07:00'); alert(ms); 1327611110417 (timestamp) We can instantly create a new given object from the timestamp: let date = new Date(Date.parse('2012-01-26T13:51:50.417-07:00'); alert(date); The summary date and time in JavaScript are represented with the Date object. We can't just create date or time: Data objects always carry both. Months are counted from scratch (yes, January is a zero month). The days of the week in getDay() are also counted from scratch (i.e. Sunday). The date automatically corrects when out-of-range components are set. Good for adding/subtracting days/months/hours. Dates can be subtracted, giving their difference in milliseconds. This is because a date becomes the timestamp when it is converted to a number. Use Date.now() to quickly get the current timestamp. Note that unlike many other systems, timestamps in JavaScript are in milliseconds, not seconds. Sometimes we need to more precise time. JavaScript itself doesn't have a way to measure time in microseconds (1 millionth of a second), but most environments provide it. For example, the browser has performance.now() which provides the number of milliseconds from the beginning of loading the page with precision (3 digits after the point): warning('Load started ${performance.now()}ms ago'); Something like: Loading started 34731.26000000001ms ago // .26 is microseconds (260 microseconds) // more than 3 digits after the decimal point is precision errors, only the first 3 are correct node.js has microtime module and other ways. Technically, almost all devices and 800 devices allow you to achieve greater accuracy, but it is not in Date. Date.