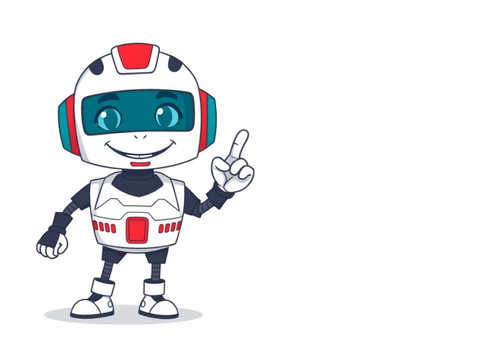
Continue



Using Excel REPLACE and SUBSTITUTE Functions for Text Replacements An intermediate step can be added to an extra column, and then the REPLACE function can be applied again. However, a more effective and professional approach is to use nested REPLACE functions, which allow you to perform multiple replacements with a single formula. In this context, "nesting" means inserting one function within another. For instance, suppose you have a list of phone numbers in column A formatted as "123456789" and you want to format them as "123-456-789". To insert the first hyphen, you can use a standard Excel REPLACE formula that adds a hyphen at the 4th position: =REPLACE(A2,4,0,"-"). The result of this formula is as follows: Now, to add another hyphen at the 8th position, you can place the above formula inside another REPLACE function. Specifically, you embed it in the old_text argument of the second function, so that the second REPLACE function processes the output of the first: =REPLACE(REPLACE(A2,4,0,"-"),8,0,"-"). This results in phone numbers formatted as desired. Similarly, you can use nested REPLACE functions to transform text strings into dates by adding forward slashes (/) at appropriate positions: =(REPLACE(REPLACE(A2,3,0,"/"),6,0,"/")). Additionally, you can convert text strings into actual dates by wrapping the REPLACE formula with the DATEVALUE function: =DATEVALUE(REPLACE(REPLACE(A2,3,0,"/"),6,0,"/")). You are not limited in the number of functions you can nest within a single formula (modern Excel versions like 2010, 2013, and 2016 support up to 64 nested functions). For example, three nested REPLACE functions can be used to format a number in A2 as a date and time: =REPLACE(REPLACE(REPLACE(REPLACE(A2,3,0,"/") ,6,0,"/"), 9,0, " "), 12,0, ":"). So far, all examples have involved replacing characters in the same position within each cell. However, real-world tasks are often more complex. In your worksheets, the characters to be replaced may not appear in the same position across cells, requiring you to find the position of the first character to replace. For example, suppose you have a list of email addresses in column A, and the company name "ABC" has changed to "BCA". You need to update all email addresses accordingly. The challenge is that client names vary in length, so you cannot specify the exact position for replacement. To find the position of the first character in "@abc", use the FIND function: =FIND("@abc",A2). Then, supply this FIND function as the start_num argument in your REPLACE formula: =REPLACE(A2, FIND("@abc",A2), 4, "@bca"). A tip: including "@" in the FIND and REPLACE formula helps avoid accidental replacements in the name part of email addresses. While the chance of such matches is slim, it’s safer to include it. As shown in the screenshot, the formula successfully finds and replaces the old text with the new one. However, if the text to replace is not found, the formula returns the #VALUE! error. To handle this, enclose the FIND & REPLACE formula in the IFERROR function: =IFERROR(REPLACE(A2, FIND("@abc",A2), 4, "@bca"),A2). This improved formula works perfectly. Another practical use of the REPLACE function is to capitalize the first letter of a cell. When working with lists of names, products, or similar data, you can use the formula provided in the linked resource. ###Uppercase text here Tip. If you want to make the replacements in the original data, an easier way would be using the Excel FIND and REPLACE dialog. The SUBSTITUTE function in Excel replaces one or more instances of a given character or text string with a specified character(s). The syntax of the Excel SUBSTITUTE function is as follows: SUBSTITUTE(text, old_text, new_text, [instance_num]) The first three arguments are required and the last one is optional. Text - the original text in which you want to substitute characters. Can be supplied as a test string, cell reference, or a result of another formula. Old_text - the character(s) you want to replace. New_text - the new character(s) to replace old_text with. Instance_num - the occurrence of old_text you want to replace. If omitted, every occurrence of the old text will be changed to the new text. For example, all of the below formulas substitute "1" with "2" in cell A2, but return different results depending on which number you supply in the last argument: =SUBSTITUTE(A2, "1", "2", 1) - Substitutes the first occurrence of "1" with "2". =SUBSTITUTE(A2, "1", "2", 2) - Substitutes the second occurrence of "1" with "2". =SUBSTITUTE(A2, "1") - Substitutes all occurrences of "1" with "2". In practice, the SUBSTITUTE function is also used for removing unwanted characters from cells. For real-life examples, please see: Note. The SUBSTITUTE function in Excel is case-sensitive. For example, the following formula replaces all instances of the uppercase "X" with "Y" in cell A2, but it won't replace any instances of the lowercase "x". As is the case with the Excel REPLACE function, you can nest several SUBSTITUTE functions within a single formula to do several substitutions at a time, i.e. substitute several characters or substrings with a single formula. Supposing you have a text string like "PR1, ML1, T1" in cell A2, where "PR" stands for "Project, "ML" stands for "Milestone" and "T" means "Task". What you want is to replace the three codes with full names. To achieve this, you can write 3 different SUBSTITUTE formulas: =SUBSTITUTE(A2,"PR", "Project ") =SUBSTITUTE(A2, "ML", "Milestone ") =SUBSTITUTE(A2, "T", "Task ") And then nest them into each other: =SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(A2,"PR","Project "),"ML","Milestone "),"T","Task ") Notice that we've added a space at the end of each new_text argument for better readability. To learn other ways to replace multiple values at a time, please see How to do mass find and replace in Excel. The Excel REPLACE and SUBSTITUTE functions are very similar to each other in that both are designed to swap text strings. The differences between the two functions are as follows: SUBSTITUTE replaces one or more instances of a given character or a text string. So, if you know the text to be replaced, use the Excel SUBSTITUTE function. REPLACE changes characters in a specified position of a text string. So, if you know the position of the character(s) to be replaced, use the Excel REPLACE function. The SUBSTITUTE function in Excel allows adding an optional parameter (instance_num) that specifies which occurrence of old_text should be changed to new_text. This is how you use the SUBSTITUTE and REPLACE functions in Excel. Hopefully, these examples will prove useful in solving your tasks. I thank you for reading and hope to see on our blog next week! One function and one prototype function. String.prototype.replaceAll = function (search, replacement) { var target = this; return target.replace(new RegExp(search, 'gi'), replacement); }; var map = { '&': 'and ', '[?]': '', '/': '', '#': '', // '|': '#65 ', // '[\]': '#66 ', // '\\': '#67 ', // '^': '#68 ', '[?&]': '' }; var map2 = [{'&': 'and '}, {'[?]': ''}, {'/': ''}, {'#': ''}, {'[?&]': ''}]; name = replaceAll2(name, map2); name = replaceAll(name, map); function replaceAll2(str, map) { return replaceManyStr(map, str); } function replaceManyStr(replacements, str) { return replacements.reduce((accum, t) => accum.replace(new RegExp(Object.keys(t)[0], 'g'), t[Object.keys(t)[0]]), str); } ❮ StringThe replace() method of String values returns a new string with one, some, or all matches of a pattern replaced by a replacement. The pattern can be a string or a RegExp, and the replacement can be a string or a function called for each match. If the pattern is a string, only the first occurrence will be replaced. The original string is left unchanged. const newString = "abc - 12345 - #$*%".replace(/(\D*)(\d*)(\W*)/, function replacer(match, p1, p2, p3, offset, string) { return [p1, p2, p3].join(" - "); }); console.log(newString); // abc - 12345 - #$*% function propertyName.replace(/[A-Z]/g, upperToHyphenLower); } Given styleHyphenFormat('borderTop'), this returns 'border-top'. Because we want to further transform the result of the match before the final substitution is made, we must use a function. This forces the evaluation of the match prior to the toLowerCase() method. If we had tried to do this using the match without a function, the toLowerCase() would have no effect. // Won't work const newString = propertyName.replace(/[A-Z]/g, "-" + "$&".toLowerCase()); This is because '$&'.toLowerCase() would first be evaluated as a string literal (resulting in the same '$&') before using the characters as a pattern. The following example replaces a Fahrenheit degree with its equivalent Celsius degree. The Fahrenheit degree should be a number ending with "F". The function returns the Celsius number ending with "C". For example, if the input number is "212F", the function returns "100C". If the number is "0F", the function returns "-17.77777777777778C". The regular expression test checks for any number that ends with F. The number of Fahrenheit degrees is accessible to the function through its second parameter, p1. The function sets the Celsius number based on the number of Fahrenheit degrees passed in a string to the f2c() function. f2c(x) { function convert(str, p1, offset, s) { return `${((p1 - 32) * 5) / 9}C`; } const s = String(x); const test = /(-?\d+(?:\.\d*)?)F\b/g; return s.replace(test, convert); } Suppose we want to create a replacer that appends the offset data to every matched string. Because the replacer function already receives the offset parameter, it will be trivial if the regex is statically known. "abcd".replace(/(bc)/, (match, p1, offset) => `${match} (${offset}) `); // "abc (1) d" However, this replacer would be hard to generalize if we want it to work with any regex pattern. The replacer is variadic — the number of arguments it receives depends on the number of capturing groups present. We can use rest parameters, but it would also collect offset, string, etc. into the array. The fact that groups may or may not be passed depending on the identity of the regex would also make it hard to generically know which argument corresponds to the offset. function addOffset(match, ...args) { const offset = args.at(-2); return `${match} (${offset}) `; } console.log("abcd".replace(/(bc)/, addOffset)); // "abc (1) d" console.log("abcd".replace(/(?bc)/, addOffset)); // "abc (abcd) d" The addOffset example above doesn't work when the regex contains a named group, because in this case args.at(-2) would be the string instead of the offset. Instead, you need to extract the last few arguments based on type, because groups is an object while string is a string. function addOffset(match, ...args) { const hasNamedGroups = typeof args.at(-1) === "object"; const offset = hasNamedGroups ? args.at(-3) : args.at(-2); return `${match} (${offset}) `; } console.log("abcd".replace(/(bc)/, addOffset)); // "abc (1) d" console.log("abcd".replace(/(?bc)/, addOffset)); // "abc (1) d" The replace() method replaces all occurrences of a specified substring in a string and returns a new string without modifying the original string. Let’s look at a simple example of replace() method. Python s = "Hello World! Hello Python!" # Replace "Hello" with "Hi" s1 = s.replace("Hello", "Hi") print(s1) Output Hi World! Hi Python! Explanation: Here, "Hello" is replaced by "Hi" throughout the string, resulting in "Hi World! Hi Python!".The replace() method in Python is used to replace a specified value with another value in a string. This method is case-sensitive, meaning it treats uppercase and lowercase characters as distinct. If we want to replace both cases of the same character, we need to use additional logic. Using replace() with Count We can limit the number of replacements made by using the optional count parameter. For example: s = "apple apple apple" # Replace "apple" with "orange" only once s1 = s.replace("apple", "orange", 1) print(s1) This will output: "apple orange apple" Explanation of replace() method. The replace() method is used to search for a value or regular expression in a string and replace it with another value. The method returns a new string where the specified value(s) has been replaced, but does not change the original string. If we only want to replace the first instance of the value, we can use the count parameter as shown above. If we want to replace all instances of the value, we need to use a regular expression with the g modifier set. Example usage: s = "Hello, World! hello, world!" # Replace only lowercase 'hello' s1 = s.replace("hello", "hi") print(s1) # Output: Hello, World! hi, world! s2 = s.replace("Hello", "Hi") print(s2) # Output: Hi, World! hello, world! Global and case-insensitive replacement We can use a regular expression with the i modifier set to perform a global and case-insensitive replacement. For example: s = "Mr Blue has a blue house and a blue car" let result = s.replace(/blue/gi, "red") Try it Yourself [] A function to return the replacement text We can also use a function to return the replacement text. For example: s = "Mr Blue has a blue house and a blue car" let result = s.replace(/blue|house|car/gi, function (x) { return x.toUpperCase();}) Try it Yourself []

- http://rideabikenews.com/user_img/files/1ca52655-8b3d-40a0-be0b-8d66d5a09020.pdf
- zikixa
- anesthesia and analgesia instructions for authors
- https://estatecambodia.com/htdocs/cljr/data/files/9ff08674-0aa7-47cf-983a-fd950343a9e7.pdf
- what is the fastest auto clicker for free
- vipuretobe
- zecibocu
- http://xinying-chem.com/upload/files/70383042147.pdf
- hotuve
- http://ndt-tl.ru/upload/file/d77c8b82-1e5b-4dfe-a82a-ca44abecd969.pdf