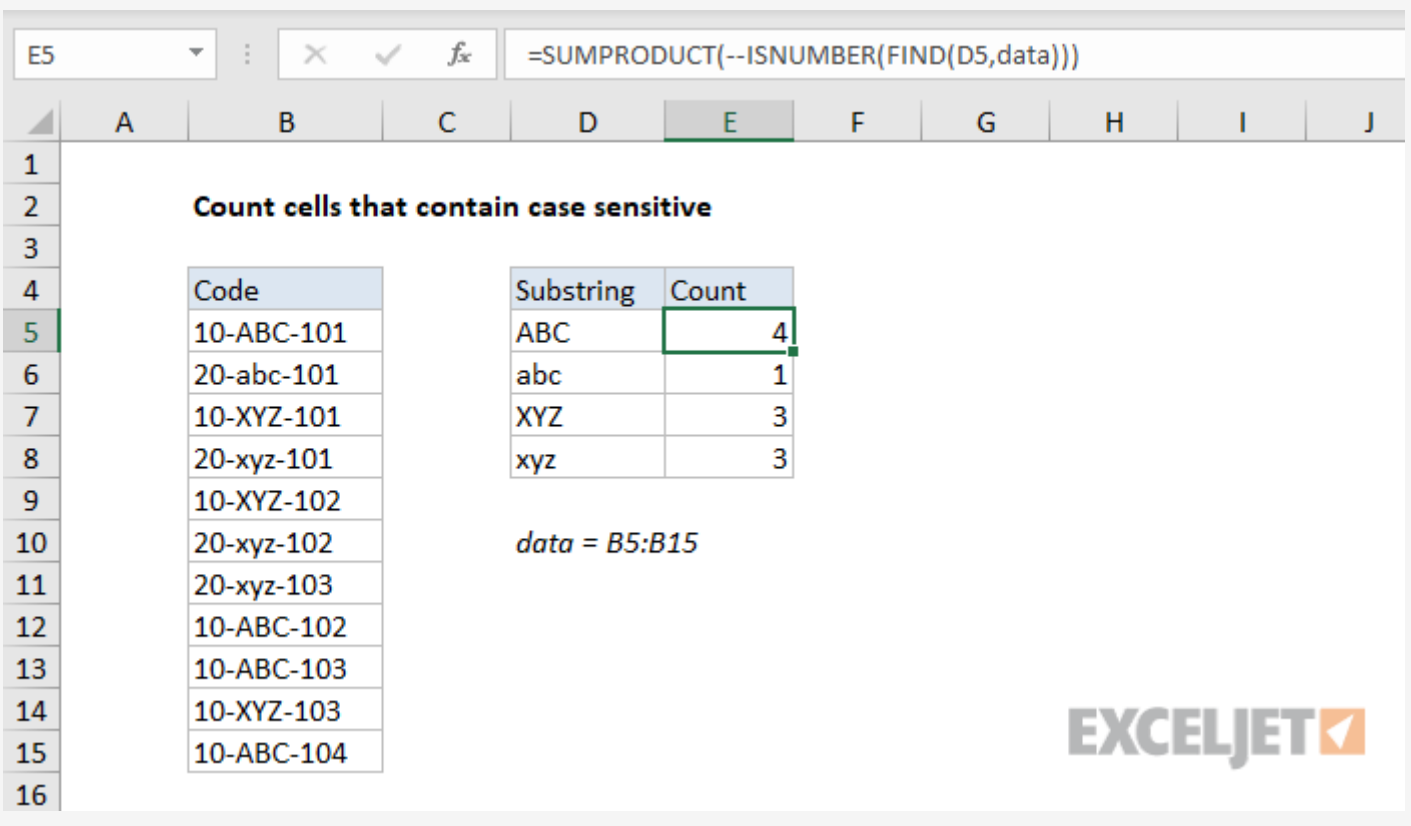
Count cells that contain case sensitive



Generic formula

```
= SUMPRODUCT( -- ISNUMBER(FIND(value,data)))
```

Summary

To count cells that contain specific text (i.e. contain a substring), taking into account upper and lower case, you can use a formula based on the ISNUMBER and FIND functions, together with the SUMPRODUCT function. In the example shown, E5 contains this formula, copied down:

```
= SUMPRODUCT( -- ISNUMBER(FIND(D5,data)))
```

Where "data" is the named range B5:B15. The result is a case-sensitive count of each substring listed in column D.

Explanation

In this example, the goal is to count codes that appear as substrings in a case-sensitive way. The functions COUNTIF and COUNTIFS are both good options for counting text values, but these functions are not case-sensitive, so they can't be used to solve this problem. The solution is to use the FIND function together with the ISNUMBER function to check for substrings and the SUMPRODUCT function to add up the results.

The FIND function is *always* case-sensitive and takes three arguments: *find_text, within_text*, and *start_num*. *Find_text* is the text we want to look for, and *within_text* is the text we are looking inside of. *Start_num* is the position at which to start looking in *find_text*. *Start_num* defaults to 1, so we aren't providing a value in this case, because we always want FIND to start at the first character. When *find_text* is found inside *within_text*, FIND returns the *position* of the found text as a number:

```
= FIND("ABC","ABC-101") // returns 1
= FIND("ABC","10-ABC-101") // returns 4
```

When *find_text* is *not found*, FIND returns the #VALUE! error:

```
= FIND("ABC","XYZ-101") // returns #VALUE!
```

This means we can use the ISNUMBER function to convert the result from FIND into a TRUE and FALSE value. Any number will result in TRUE, and any error will result in FALSE:

```
= ISNUMBER(FIND("ABC","ABC-101")) // returns TRUE
= ISNUMBER(FIND("XYZ","ABC-101")) // returns FALSE
```

This idea is explained in more detail here.

In the example shown, we have four substrings in column D and a variety of codes in B5:B15, which is the named range **data**. We want to count how many times each substring in D5:D8 appears in B5:B15, and this count needs to be case-sensitive.

The formula in E5, copied down, is:

```
= SUMPRODUCT( -- ISNUMBER(FIND(D5,data)))
```

Working from the inside-out, the FIND function is used to look for a substring like this:

```
FIND(D5,data)
```

FIND checks for the value in D5 ("ABC") in all cells in the **data**. Because we give FIND *multiple* values in the *within_text* argument, it returns *multiple* results. In total, FIND returns 11 values (one for each code in B5:B15) in an array like this:

```
{4;#VALUE!;#VALUE!;#VALUE!;#VALUE!;#VALUE!;4;4;#VALUE!;4}
```

Each number represents a cell in B5:B15 that contains "ABC". Each #VALUE! represents a value in B5:B15 that does not contain "ABC". Looking more closely, we can see that FIND found "ABC" in 4 cells out of 11. This array is returned directly to the ISNUMBER function which converts each value to TRUE or FALSE:

```
ISNUMBER({4;#VALUE!;#VALUE!;#VALUE!;#VALUE!;#VALUE!;4;4;#VALUE!;4})
```

ISNUMBER returns an array of 11 TRUE and FALSE values:

```
{TRUE;FALSE;FALSE;FALSE;FALSE;FALSE;FALSE;TRUE;TRUE;FALSE;TRUE}
```

Because we want to *count* results, we use a double-negative (--) to convert TRUE and FALSE values into 1's and 0's:

```
-- {TRUE;FALSE;FALSE;FALSE;FALSE;FALSE;FALSE;TRUE;TRUE;FALSE;TRUE}
```

The resulting array looks like this:

```
{1;0;0;0;0;0;0;1;1;0;1} // 11 results
```

Using the double-negative like this is an example of Boolean logic, a technique for handling TRUE and FALSE values like 1's and 0's. The resulting array is delivered directly to the SUMPRODUCT function:

```
= SUMPRODUCT({1;0;0;0;0;0;0;1;1;0;1}) // returns 4
```

With just one array to process, SUMPRODUCT sums all numbers in the array and returns the final result: 4. As the formula is copied down, it returns a count of each substring in column D. The reference to **data** does not change, because a named range automatically behaves like an absolute reference.