# ICA 10 - Code Validator 1 (Working with strings)

FUN FACT: The data you collect using an HTML form’s text box will \*always\* be a string, even if the data is all digits.

## Background

We are going to explore how JavaScript string functions can be used to validate patterns in strings. This might seem like an abstract exercise, but it really is very practical. Here are some real-world scenarios where pattern validation would be appropriate.

* **Visa credit card** numbers are typically **16 characters**, **all digits**, and always **start with a 4**. There are additional constraints as well, but the common, practical checks are length, digit-only content, and the leading "4".
* The digits and structure of a **U.S. Social Security Number (SSN)** have historical and geographic significance. Originally, an SSN consisted of nine digits formatted as AAA-GG-SSSS. The first three digits (area number) originally indicated the state or region where the number was issued; the middle two digits (group number) were assigned in a specific pattern to help manage the numbering system; and the last four digits (serial number) uniquely identify the individual within that group.

Since 2011, the Social Security Administration began “randomization” to protect privacy and prevent fraud — meaning the area numbers no longer correspond to specific states. However, the format still provides predictable structure, which can be validated to catch typos or invalid combinations (such as all zeros in any segment).

* **FOAPAL** stands for **Fund, Organization, Account, Program, Activity, and Location** — a structured coding system used in higher education and nonprofit accounting.
	+ Fund – Identifies the source of money (e.g., state appropriations, grants, student fees).
	+ Organization – Indicates the department or unit responsible (e.g., MIS Department).
	+ Account – Classifies the type of transaction (e.g., salary, travel, supplies).
	+ Program – Connects the transaction to a functional area or purpose (e.g., instruction, research).
	+ Activity (optional) – Tracks specific projects or short-term initiatives.
	+ Location (optional) – Used to record physical or reporting locations.

For example, a valid FOAPAL might look like:

110012-530000-7100-200100-12345

* **Password strength policy**: require a minimum length, check for at least one uppercase letter, one lowercase letter, one digit, and one special character.
* A **model number or product code** is a unique identifier assigned by a manufacturer to distinguish one product version or configuration from another. It’s often a combination of letters, digits, and sometimes dashes or periods that encode information such as product line, size, color, or region.

## Scenario

We have been asked to write a small JavaScript application to help our team verify product codes. A valid code must be **15 characters long**, begin with **909-**, and include **exactly three occurrences of the letter A**, regardless of whether those letters are uppercase or lowercase.

With the exception of the **“-“** after **909** all the characters must be digits (0 through 9) or alphabet letters (a through z or A through Z).

That means that a valid string will have **15** characters. **One** of those characters will be a **“-“** and the other **14** characters will be **alphanumeric**.

If any of these requirements are not met, the code should be considered invalid. In this activity you will build a function to test a code and return either "Valid code" or "Invalid code" based on those rules.

## Step‑by‑Step Instructions

1. **Sign in** to the class server using Bitvise. Use the username and password your instructor provided. The hostname for the server is **misdemo.temple.edu**.
2. **Change to your wwwroot directory**. At the prompt, type:
* cd wwwroot
1. In your terminal session type the following command to retrieve the zip file:
* wget https://misdemo.temple.edu/classexamples/ica10validator1.zip
1. **Unzip the file.** This will create a folder named ica10 containing validator1.html.
* unzip ica10validator1.zip
* cd ica10
* nano validator1.html
1. **Open the validator1.html file in your web browser**.

https://misdemo.temple.edu/<<username>>/ica10/validator1.html

1. **Open the validator1.html file in nano.** You should first complete the countDigits() function. This function, when completed, should tell you how many digits are in an input string. There is already a similar function called countAlphas() that has been written for you. Using countAlphas() as a reference, complete countDigits().
2. **Test your work.** Be sure to save your changes (Control-O) and refresh the page. After the page loads, open the console and test countDigits() and countAlphas() with commands such as:

countDigits("X7ABC"); // returns 1
countDigits("12344321"); // returns 8
countAlphas("X7ABC"); // returns 4
countAlphas("12344321"); // returns 0

1. Now edit the validateCode() function in validator1.html. There are comments in the JavaScript code that will help you. You need to read them. Be sure to save your changes (Control-O) and refresh the page after each test.
2. **Test your function using the browser console.** After the page loads, open the console and run commands such as:
* validateCode('909-AA1A23X4567'); // Valid
validateCode('909-A1A1X1A1234'); // Valid
validateCode('123-A1A1A1A1234'); // Invalid
* Verify that valid codes return "Valid code" and invalid codes return "Invalid code" as expected.
1. **Repeat until your function works** for several different examples. Some sample screenshots are at the end of this document. When you are satisfied, continue to the next step.
2. **Submit the URL to your work using the corresponding Canvas assignment.**

## Facts & Formulas

* The data you collect using an HTML form’s text box will \*always\* be a string, even if the data is all digits.
* Use code.trim() to remove any leading or trailing spaces before you begin testing.
* A string’s length is found using its .length property. For example, code.length returns the number of characters in code.
* The .slice(start, end) method returns a portion of a string. To get the first four characters of a code you can write code.slice(0, 4).
* Convert the entire string to uppercase with .toUpperCase() before counting the letter A so that you do not need to worry about case.
* Use a for loop to examine each character in the code. Remember that you can access individual characters with square brackets, like code[i].

## Hints & Reminders

* To count occurrences of a specific letter, start a counter at 0 and increment it each time the character matches your target letter.
* If you follow the hints in the start file, you will see that strategy here is to count the characters that are digits, and the characters that are alphabet characters. The sum of those two numbers needs to be **14.**
* When one of the rules is violated, you can return early with "Invalid code" instead of continuing through the rest of the function.
* Do **not** use console.log() as your final output. However, using a console.log() aftert each rule is passed can help you debug your code.

**SAMPLE OUTPUT**

 