## Mail Validation

var mailformat = /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/; - This line defines a regular expression pattern (mailformat) that represents a valid email address. It checks for the general structure of an email address, including the presence of an "at" symbol (@) and a domain extension of two to three characters.

## Password Validation

## To check a password between 8 to 15 characters which contain at least one lowercase letter, one uppercase letter, one numeric digit, and one special character

To validate the said format we use the regular expression ^(?=.\*\d)(?=.\*[a-z])(?=.\*[A-Z])(?=.\*[^a-zA-Z0-9])(?!.\*\s).{8,15}$.

**Alphabets Validation**

**/^[A-Za-z]+$/**

**Contact Validation**

**/^92(\d{3})?(\d{3})?(\d{4})$/ This validation ensures 92 as a code in the beginning of number**

**CNIC Validation**

**/^([0-9]{5})[-]([0-9]{7})[-]([0-9]{1})+/**

**URL Validation**

**/(?:https?):\/\/(\w+:?\w\*)?(\S+)(:\d+)?(\/|\/([\w#!:.?+=&%!\-\/]))?/**

****Important Code Snippet Explanation****

* ****let requiredFields = [inputEmail3, inputPassword3, inputConfirm, inputName, inputDate, inputNumber, inputCnic, inputUrl];****
* ****let filledFields = requiredFields.filter(field => field.value !== "");****

**These lines involve the use of arrays and the `filter()` method in JavaScript.**

**1. `let requiredFields = [inputEmail3, inputPassword3, inputConfirm, inputName, inputDate, inputNumber, inputCnic, inputUrl];`**

**This line creates an array called `requiredFields` and initializes it with a list of variables: `inputEmail3`, `inputPassword3`, `inputConfirm`, `inputName`, `inputDate`, `inputNumber`, `inputCnic`, and `inputUrl`. These variables likely represent form input fields in an HTML document. The purpose of this array is to store references to the required fields that need to be filled in the form.**

**2. `let filledFields = requiredFields.filter(field => field.value !== "");`**

**This line creates a new array called `filledFields` using the `filter()` method. The `filter()` method is called on the `requiredFields` array and takes a callback function as an argument.**

**The callback function is defined using an arrow function (`field => field.value !== ""`). This function is applied to each element (`field`) in the `requiredFields` array.**

**Inside the callback function, `field.value !== ""` is used as the filtering condition. It checks if the `value` property of the `field` (input field) is not an empty string. In other words, it checks if the field is filled or not.**

**The `filter()` method creates a new array (`filledFields`) containing only the elements from `requiredFields` that satisfy the filtering condition. In this case, `filledFields` will contain only the fields that are filled (i.e., their `value` is not an empty string).**

**After executing these lines, the `requiredFields` array stores references to the required form fields, and the `filledFields` array contains only the fields that have been filled by the user.**

## DOM object

In JavaScript, the `guessField` variable in your code likely refers to an HTML input element that was selected using `document.querySelector(".guessField")`. When you select an HTML element using `querySelector()`, you obtain a reference to the actual DOM (Document Object Model) element in the web page.

The DOM element represents the actual HTML element in the web page, and it comes with various properties and methods that you can access and use. Here's why `guessField` can use properties like `.value` and methods like `.focus()`:

1. `.value`: The `value` property is a common property of HTML input elements. It allows you to get or set the current value of the input element. Since `guessField` refers to an HTML input element, you can access its `value` property to retrieve or modify the value entered by the user.

2. `.focus()`: The `focus()` method is a built-in method of DOM elements, including HTML input elements. Calling `guessField.focus()` sets the focus on the `guessField` input element, making it active for user interaction. This method is useful for automatically bringing attention and input focus to a specific element, such as an input field, when certain conditions are met.

By having a reference to the actual DOM element with `guessField`, you can access and utilize its properties and methods as defined by the HTML specification for that specific element type.

## [Events](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/A_first_splash" \l "events)

At this point, we have a nicely implemented checkGuess() function, but it won't do anything because we haven't called it yet. Ideally, we want to call it when the "Submit guess" button is pressed, and to do this we need to use an event. Events are things that happen in the browser — a button being clicked, a page loading, a video playing, etc. — in response to which we can run blocks of code. Event listeners observe specific events and call event handlers, which are blocks of code that run in response to an event firing.

Add the following line below your checkGuess() function:

guessSubmit.addEventListener("click", checkGuess);

Copy to Clipboard

Here we are adding an event listener to the guessSubmit button. This is a method that takes two input values (called arguments) — the type of event we are listening out for (in this case click) as a string, and the code we want to run when the event occurs (in this case the checkGuess() function). Note that we don't need to specify the parentheses when writing it inside [addEventListener()](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener" \o "addEventListener())