

JAVASCRIPT



JS

Contenido

★ Validación de Formularios

○ Eventos

- .onfocus
- .onblur
- .onchange
- .onsubmit

○ Métodos

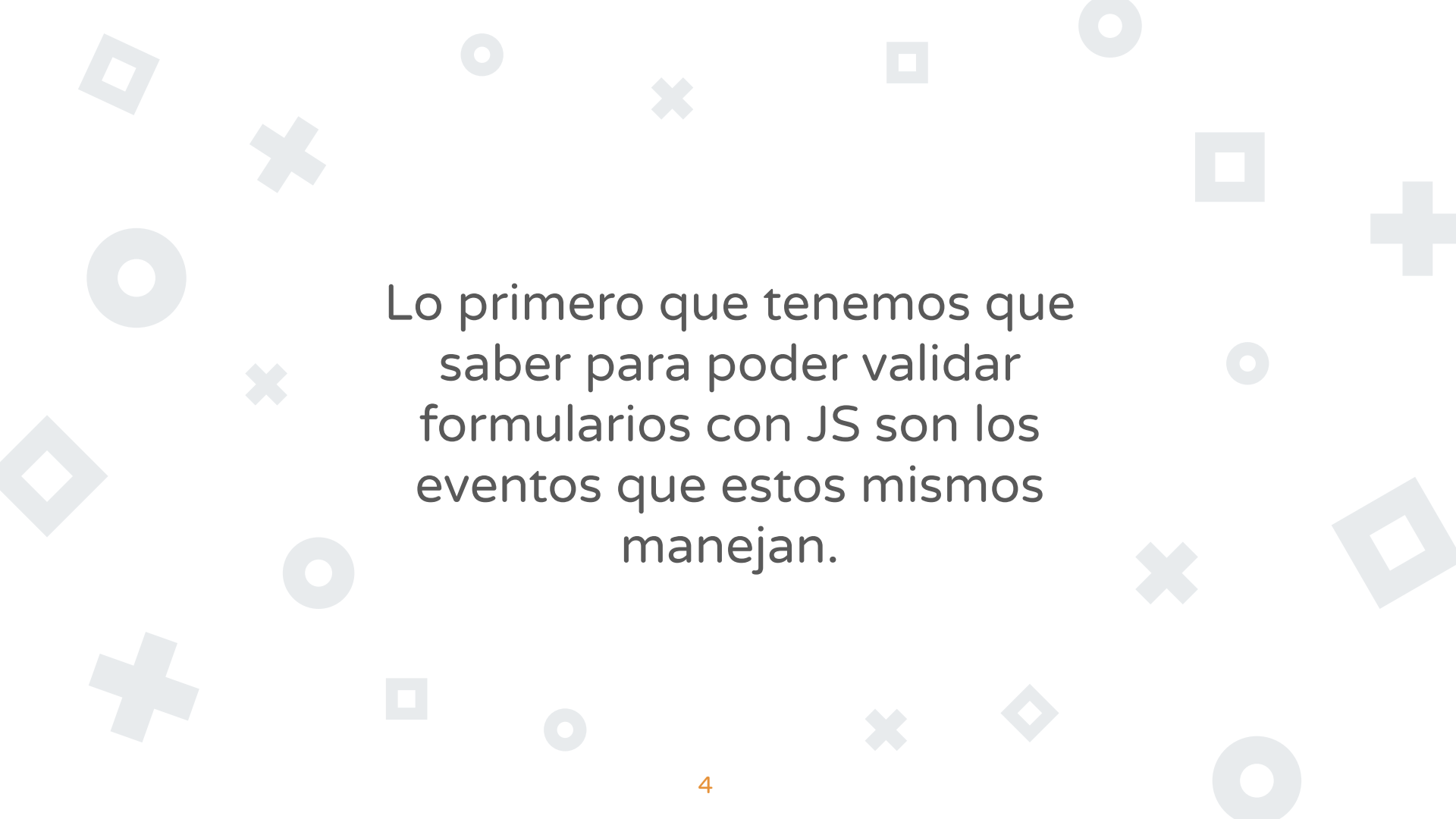
- .elements
- .value
- .selectedIndex
- .options

○ Regex

- Email Type
- Only Numbers

Cuando tenemos un formulario en nuestra UI. Lo mejor que podemos hacer es validar el mismo antes de que la información se envíe al servidor. Así evitamos un request y un response innecesario.

A esta validación se la conoce como validación on-time.

The background is a light gray color with various geometric shapes scattered across it. These shapes include squares, circles, and crosses, some of which are hollow and others are solid. The shapes are distributed in a way that they appear to be floating or scattered around the central text.

Lo primero que tenemos que
saber para poder validar
formularios con JS son los
eventos que estos mismos
manejan.

Eventos de Formularios

.onfocus

Este evento se dispara INMEDIATAMENTE cuando se hace foco sobre un campo de formulario, en otras palabras, cuando se ubica el cursor dentro del campo.

```
var inputName = document.querySelector("[name=user-name]");
inputName.onfocus = function() {
    console.log("You focus the name input!");
}
```

// Recomendado: usar el `addEventListener`.

Eventos de Formularios

.onblur

Este evento se dispara INMEDIATAMENTE cuando el campo ha perdido el foco.

```
var inputName = document.querySelector("[name=user-name]");
inputName.onblur = function() {
    console.log("I lost the focus :/");
}
```

// Recomendado: usar el `addEventListener`.

Eventos de Formularios

.onchange

Este evento se dispara INMEDIATAMENTE cuando el campo ha perdido el foco siempre y cuando el valor del campo haya cambiado.

```
var inputName = document.querySelector("[name=user-name]");
inputName.onchange = function() {
    console.log("I lost the focus and you do some changes");
}
```

// Muy funcional en los campos de tipo <select>

Eventos de Formularios

.onsubmit

Este evento se dispara EXCLUSIVAMENTE sobre el <form> y cuando se ha presionado un botón **submit**.

```
var theForm = document.querySelector(".myForm");
theForm.onsubmit = function(event) {
    if ( hay campos vacios ) {
        event.preventDefault(); // evitamos que se envíe el formulario
    }
}
```


Pero
¿Cómo podemos saber si un
campo está vacío?



Propiedad .value

.value

La propiedad **.value** nos retorna el valor actual de un campo de un formulario.

```
var inputName = document.querySelector("[name=user-name]");
inputName.onblur = function() {
    if( this.value == "" ) {
        console.log("¡Llename el campito por el amor de Jebus!");
    }
}
```

Combinando .value y onsubmit

.value ft .onsubmit

```
var theForm = document.querySelector(".myForm");
var inputName = document.querySelector("input[name='user-name']");
theForm.onsubmit = function(event) {
    if ( inputName.value == "" ) {
        event.preventDefault();
        inputName.classList.add("error");
        inputName.parentElement.querySelector("b").innerText = "Obligatorio";
    }
}
```

Y si tenemos muchos campos en un formulario. ¿Debemos capturar uno por uno?

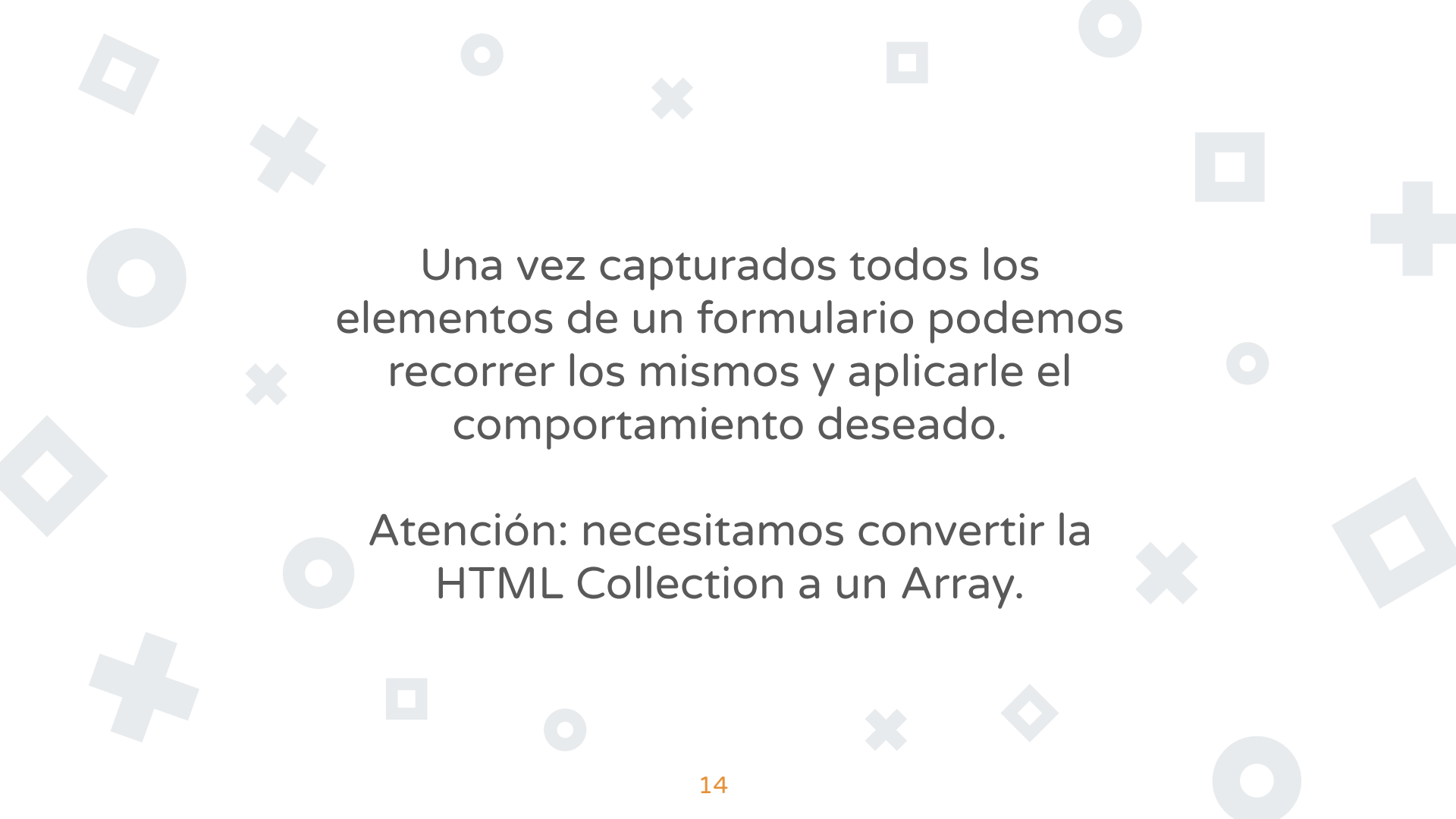


Propiedad .elements

.elements

La propiedad **.elements** (ejecutada sobre un formulario) retorna una colección de elementos HTML pertenecientes al formulario.

```
var theForm = document.querySelector(".myForm");  
var elementsOfForm = theForm.elements;  
console.log(elementsOfForm); // HTML Collection
```



Una vez capturados todos los elementos de un formulario podemos recorrer los mismos y aplicarle el comportamiento deseado.

Atención: necesitamos convertir la HTML Collection a un Array.

```
var theForm = document.querySelector(".myForm");
var elementsOfForm = theForm.elements; // HTML Collection
var elementsinArray = Array.from(elementsOfForm); // Array de elementos HTML
elementsinArray.pop(); // Sacamos el último elemento, seguro es un botón SUBMIT
elementsinArray.forEach( function(element) {
    element.addEventListener("blur", function(){
        if ( this.value == "" ) {
            this.classList.add("error");
        }
    })
});
```

Un campo `<select>` es algo especial,
pues no permite inserción de datos
de parte de usuario si no que ya trae
consigo todas las opciones.

JS nos permite acceder a las mismas
de una manera bastante sencilla.

El <select> y .options

.options

La propiedad **.options**, retorna todos los elementos <option> de un <select>. Al igual que **.elements**, retorna una HTML collection.

```
var theSelect = document.querySelector("select");  
var allTheOptions = theSelect.options;  
console.log(allTheOptions); // HTML Collection
```

// Al retornar una HTML Collection, podemos acceder a sus posiciones como si fuera un Array.

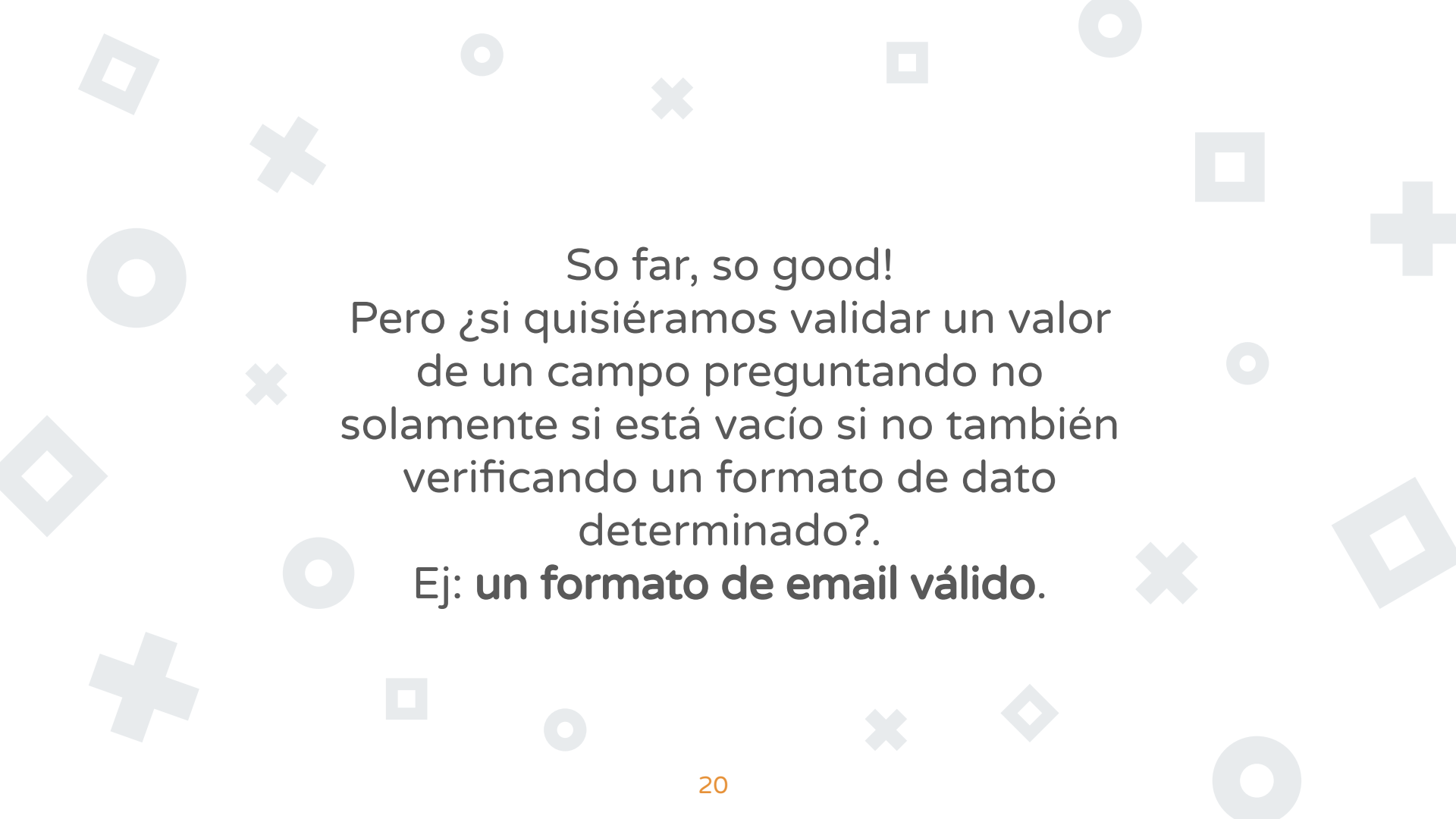
El <select> y .selectedIndex

.selectedIndex

No solo podemos acceder a toda la colección de <options> si no que a su vez podemos acceder a la posición numérica del <option> actualmente seleccionado.

```
var theSelect = document.querySelector("select");  
var allTheOptions = theSelect.options; // Collection of Options  
var optionSelected = theSelect.selectedIndex;  
console.log(optionSelected); // Number del <option> seleccionado  
console.log(allTheOptions[optionSelected]); // ¿?
```

```
var theSelect = document.querySelector("select");
theSelect.addEventListener("change", function(){
    console.log(this.options[this.selectedIndex].value);
    console.log(this.options[this.selectedIndex].innerText);
});
```



So far, so good!
Pero ¿si quisiéramos validar un valor
de un campo preguntando no
solamente si está vacío si no también
verificando un formato de dato
determinado?.

Ej: un formato de email válido.

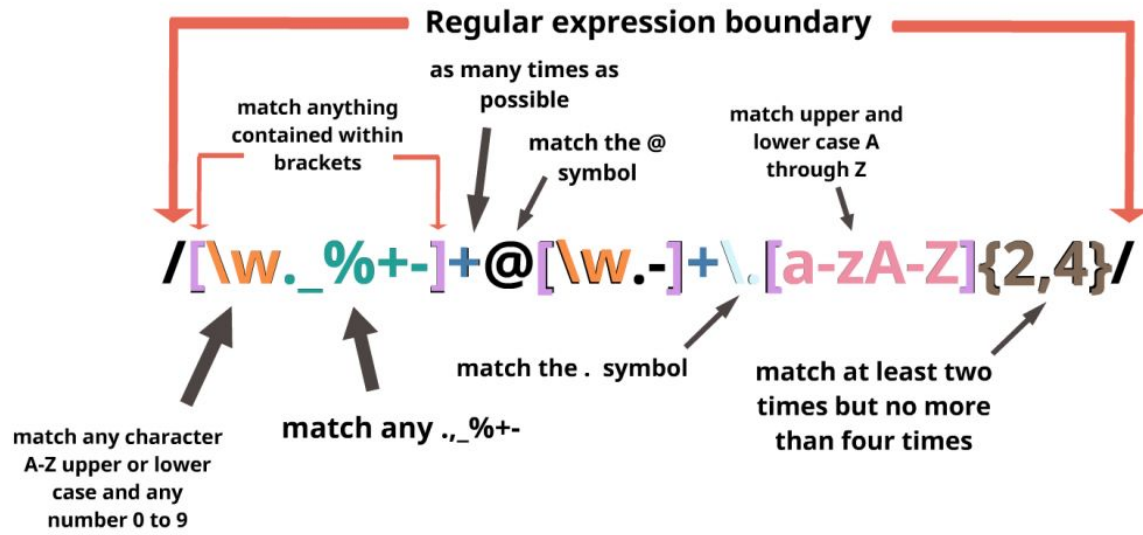


REGEX

Expresiones regulares

The background is a solid yellow color with various geometric shapes scattered across it. These shapes include squares, circles, and crosses, some of which are hollow and others are solid. The shapes are distributed in a non-uniform, random pattern.

*"Una expresión regular es una
secuencia de caracteres que forma
un patrón de búsqueda"*



Regex

Regex - .test()

Lo primero que necesitamos es guardar la **RegEx** en una variable y luego usando el método **.test()**, que recibe como parámetro el valor que deseamos evaluar, testear si el valor suministrado cumple con el patrón, **.test()** retorna un boolean.

```
var regexEmail = /^[a-zA-Z0-9._-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6}$/;  
var validEmail = "javier@digitalhouse.com";  
var notAnEmail = "Peter Griffin";  
  
console.log(regexEmail.test(validEmail)); // true  
console.log(regexEmail.test(notAnEmail)); // false
```


¡A practicar!

Práctica Integradora



```
// To Do  
console.log("Practice Time");
```