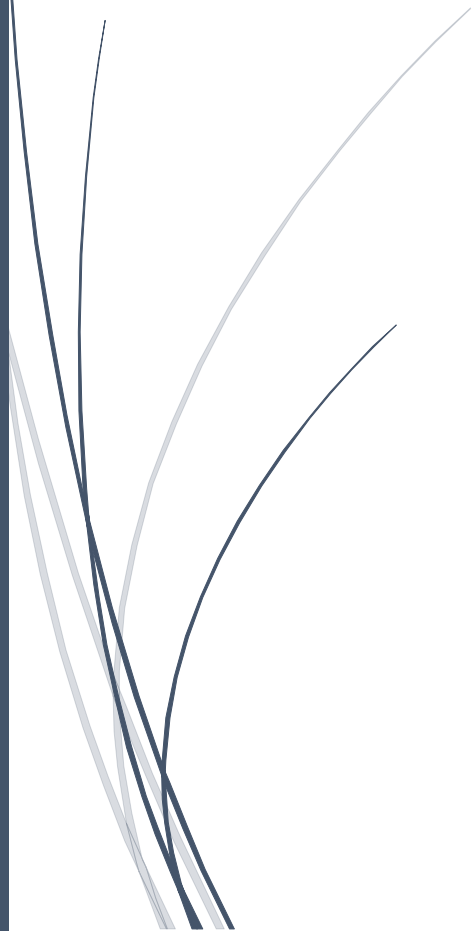
12-2-2024

# Gestión de eventos y formularios en JavaScript

## Tarea 05



ROBERTO RODRÍGUEZ JIMÉNEZ
roberto.rodjim.1@educa.jcyl.es

# Contenido

# Tarea online DWEC05

## 1.- Descripción de la tarea

**Caso práctico**

Antonio, afronta una de las partes más interesantes en su estudio de JavaScript. Se trata de la gestión de los eventos en JavaScript y cómo validar formularios.

Antonio estaba deseando llegar a esta sección, ya que su trabajo de desarrollo en el proyecto se va a centrar en muchas de las cosas que va a ver ahora. Va a tener que validar todos los formularios de la web antigua y tendrá que hacerlo con JavaScript, dando mensajes al usuario, sobre los posibles errores que se vaya encontrando durante la validación. La validación de un formulario es primordial, ya que lo que se busca es que cuando los datos sean enviados al servidor, vayan lo más coherentes posible.

Gracias a la gestión de eventos, Antonio podrá, por ejemplo, capturar acciones del usuario cuando pulse un botón o cuando pase el ratón por zonas del documento, al introducir datos, etc.

Juan está también muy ilusionado con los progresos de Antonio durante todo este tiempo, y le facilita toda la documentación y algunos ejemplos interesantes de validaciones de datos con JavaScript.

### ¿Qué te pedimos que hagas?

Realizar la validación del formulario facilitado en el enunciado, cumpliendo los siguientes requisitos:

1. Programar el código de JavaScript en un fichero independiente. La única modificación que se permite realizar en el fichero .html es la de escribir la referencia al fichero de JavaScript.
2. Almacenar en una cookie el número de intentos de envío del formulario que se van produciendo y mostrar un mensaje en el contenedor "intentos" similar a: "Intento de Envíos del formulario: X". Es decir cada vez que le demos al botón de enviar tendrá que incrementar el valor de la cookie en 1 y mostrar su contenido en el div antes mencionado. Nota: para poder actualizar el contenido de un contenedor o div la propiedad que tenemos que modificar para ese objeto es innerHTML.
3. Cada vez que los campos NOMBRE y APELLIDOS pierdan el foco, el contenido que se haya escrito en esos campos se convertirá a mayúsculas.
4. Realizar una función que valide los campos de texto NOMBRE y APELLIDOS. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en los campos correspondientes.
5. Validar la EDAD que contenga solamente valores numéricos y que esté en el rango de 0 a 105. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo EDAD.
6. Validar el NIF. Utilizar una expresión regular que permita solamente 8 números un guión y una letra. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo NIF. No es necesario validar que la letra sea correcta. Explicar las partes de la expresión regular mediante comentarios.
7. Validar el E-MAIL. Utilizar una expresión regular que nos permita comprobar que el e-mail sigue un formato correcto. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo E-MAIL. Explicar las partes de la expresión regular mediante comentarios.

8. Validar que se haya seleccionado alguna de las PROVINCIAS. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo PROVINCIA.
9. Validar el campo FECHA utilizando una expresión regular. Debe cumplir alguno de los siguientes formatos: dd/mm/aaaa o dd-mm-aaaa. No se pide validar que sea una fecha de calendario correcta. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo FECHA. Explicar las partes de la expresión regular mediante comentarios.
10. Validar el campo TELEFONO utilizando una expresión regular. Debe permitir 9 dígitos obligatorios. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo TELEFONO. Explicar las partes de la expresión regular mediante comentarios.
11. Validar el campo HORA utilizando una expresión regular. Debe seguir el patrón de hh:mm. No es necesario validar que sea una hora correcta. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo HORA. Explicar las partes de la expresión regular mediante comentarios.
12. Pedir confirmación de envío del formulario. Si se confirma el envío realizará el envío de los datos; en otro caso cancelará el envío.

## 2.- Información de interés

### Recursos necesarios

Editor web para teclear el código de la aplicación y un navegador web para ejecutar y probar la aplicación.

Fichero .html con el formulario que hay que validar.

```
<!DOCTYPE html>
<htm xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>DWEC05 - Solución Tarea</title>
        <script type="text/javascript" src="scriptSolucion.js"></script>
        <style type="text/css">
            label{
                width: 150px;
                float:left;
                margin-bottom:5px;
            }
            input,select {
                width:150px;
                float:left;
                margin-bottom:5px;
            }
            fieldset{
                background:#66CCCC;
                width:350px;
                border: thick solid #306;
            }
            legend{
                border-top-width: medium;
                border-right-width: medium;
                border-bottom-width: medium;
                border-left-width: medium;
```

```
            border-top-style: solid;
            border-right-style: solid;
            border-bottom-style: solid;
            border-left-style: solid;
            background-color: #FFF;
        }
        #mensajes{
            float: left;
            background:#33FF33;
            width: 325px;
        }
        #errores{
            float: left;
            background:#FF6633;
            width: 325px;
        }
        #intentos{
            float: left;
            background:#CCFF33;
            width: 325px;
        }
        .error{
            border: solid 2px #FF0000;
        }
    </style>
</head>
<body>
    <fieldset>
        <legend>DWEC05 - Solución Tarea</legend>
        <form name="formulario" id="formulario" action="http://www.google.es" method="get">
            <label for="nombre">Nombre:</label>
            <input type="text" name="nombre" id="nombre" />
            <label for="apellidos">Apellidos:</label>
            <input type="text" name="apellidos" id="apellidos" />
            <label for="edad">Edad:</label>
            <input name="edad" type="text" id="edad" maxlength="3" />
            <label for="nif">NIF:</label>
            <input name="nif" type="text" id="nif" />
            <label for="email">E-mail:</label>
            <input name="email" type="text" id="email" />
            <label for="provincia">Provincia:</label>
            <select name="provincia" id="provincia">
                <option value="0" selected="selected">Seleccione Provincia</option>
                <option value="C">A Coruña</option>
                <option value="LU">Lugo</option>
                <option value="OU">Ourense</option>
                <option value="OU">Pontevedra</option>
            </select>
            <label for="fecha">Fecha Nacimiento:</label>
            <input name="fecha" type="text" id="fecha" />
            <label for="telefono">Teléfono:</label>
            <input name="telefono" type="text" id="telefono" maxlength="9"/>
```

```
                <label for="hora">Hora de visita:</label>
                <input name="hora" type="text" id="hora" />
                <input type="reset" name="limpiar" id="button" value="Limpiar" />
                <input type="submit" name="enviar" id="enviar" value="Enviar" />
            </form>
            <div id="errores"></div>
            <div id="intentos"></div>
        </fieldset>
    </body>
</html>
```

### Consejos y recomendaciones

Se recomienda realizar una función para cada una de las validaciones de tal forma que se pueda llamar a cada una de forma independiente. Las funciones deberían devolver true si la validación ha sido correcta o false (y los mensajes de error solicitados) si la validación ha sido incorrecta.

## 3.- Evaluación de la tarea

### Criterios de evaluación implicados

a) Se han reconocido las posibilidades del lenguaje de marcas relativas a la captura de los eventos producidos.
b) Se han identificado las características del lenguaje de programación relativas a la gestión de los eventos.
c) Se han diferenciado los tipos de eventos que se pueden manejar.
d) Se ha creado un código que capture y utilice eventos.
e) Se han reconocido las capacidades del lenguaje relativas a la gestión de formularios Web.
f) Se han validado formularios web utilizando eventos.
g) Se han utilizado expresiones regulares para facilitar los procedimientos de validación.
h) Se ha probado y documentado el código.

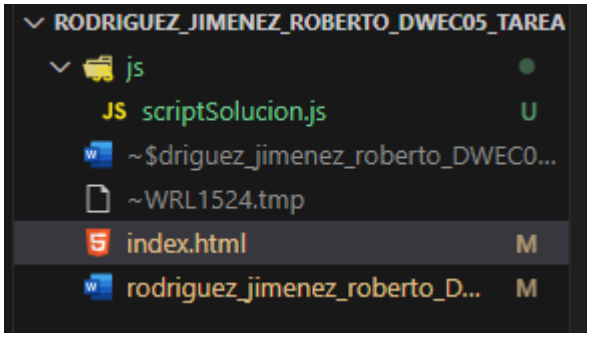
### ¿Cómo valoramos y puntuamos tu tarea?

| **Rúbrica de la tarea** | |
|---|---|
| Apartado 1 | 0,5 puntos |
| Apartado 2 | 1 punto |
| Apartado 3 | 0,75 puntos |
| Apartado 4 | 0,5 puntos |
| Apartado 5 | 0,75 puntos |
| Apartado 6 | 1 punto |
| Apartado 7 | 1,5 puntos |
| Apartado 8 | 0,75 puntos |
| Apartado 9 | 0,75 puntos |
| Apartado 10 | 0,75 puntos |
| Apartado 11 | 0,5 puntos |
| Apartado 12 | 0,75 puntos |
| Claridad y presentación del código, indentación, etc. | 0,5 puntos |

## RESOLUCIÓN

1. Programar el código de JavaScript en un fichero independiente. La única modificación que se permite realizar en el fichero .html es la de escribir la referencia al fichero de JavaScript.

Se mantiene la referencia al fichero JavaScript, pero se coloca en un directorio *js*.
Se modifica, por tanto, el enlace en la llamada al fichero desde *head*.




*Estructura de archivos*

2. Almacenar en una cookie el número de intentos de envío del formulario que se van produciendo y mostrar un mensaje en el contenedor "intentos" similar a: "Intento de Envíos del formulario: X". Es decir, cada vez que le demos al botón de enviar tendrá que incrementar el valor de la cookie en 1 y mostrar su contenido en el div antes mencionado. Nota: para poder actualizar el contenido de un contenedor o div la propiedad que tenemos que modificar para ese objeto es innerHTML.

Para actualizar el valor de la cookie ***intentos*** tenemos que hacerlo en dos pasos: comprobar si existe la cookie para obtener su valor y crearla o actualizarla.

Cargar el botón `enviar` y agregar el oyente, que ejecutará la función `procesarFormulario` pasándole el evento como argumento.

```
window.onload = () => {

    // Declaración de los componentes
    const BTN_ENVIAR = document.getElementById('enviar');

    // Eventos
    BTN_ENVIAR.addEventListener('click', (e) => procesarFormulario(e));


}
```

`procesarFormulario` establece la cookie y la modifica con `setIntento()`.

```
// Procesa el formulario
```

```
const procesarFormulario = () => {

    // Si no hay errores, err no tiene valor, es undefined
    if (err != false) {

        // Solicitamos la confirmación y enviamos el formulario
        if (confirm("¿Enviar el formulario")) {

            // Establece y actualiza la cookie de intentos de envío del formulario
            setIntento();
            formulario.submit();
        }
    }

};
```

*Función setIntento()*

Esta función establece y modifica el valor de la cookie ***intentos***.
Para obtener el valor de la cookie, llama a la función `getIntentos()`, encargada de devolver el valor actual de la cookie.
Después de obtener el valor, lo incrementa (puede ser 0), establece la duración en 1 día y el path en el directorio raíz.

Tanto el nombre como el valor son codificados para escapar los caracteres especiales con `encodeURIComponent`.

```
function setIntento(){

    // obtener los intentos e incrementarlos
    let intentos = getIntentos();
    intentos++;

    // Tiempo de vida de la cookie: 1 día
    let max_time = 24*60*60;

    // Establecemos la cookie, útil durante 5 min.
    document.cookie = `${encodeURIComponent('intentos')}=${encodeURIComponent(intentos)};path=/;max-
age=${max_time};`;

}
```

*Función getIntentos()*

Función que devuelve el valor de cookie ***intentos***.
Las cookies dependen de `document.cookie` (por lo que no pueden ser leídas desde otra ventana o pestaña) y se almacenan como una cadena de texto con el formato nombre1=valor1;nombre2=valor2;…;nombren=valorn;

Para encontrar la cookie ***intentos*** se declara el patrón `/intentos=\d+/` en el que se busca la cadena *intentos=* segida de, al menos, un número.

El primer paso es buscar la cadena en el String que guarda las cookies y, si no aparece, devolvemos 0 sin hacer nada más.

```
function getIntentos(){

    // Patrón buscado
    let pattern = /intentos=\d+/;

    // Obtener las cookies decodificadas
    let cookies = document.cookie;

    // buscar el patrón entre la cadena de cookies
    if(cookies.match(pattern)){
        . . .
    }else{
        return 0;
    }

}
```

En caso de que la búsqueda sea positiva, hay que procesar la cadena de texto con las cookies para obtener, primero, la cookie buscada y, posteriormente, su valor.

```
function getIntentos(){

    . . .

    // buscar el patrón entre la cadena de cookies
    if(cookies.match(pattern)){

        // tenemos las cookies con el formato nombre=valor;nombre=valor...
        let array_cookies = cookies.split(';');

        // Hay que conseguir el registro intentos=\d y para ello usamos la función filter
        // que devuelve un array con las coincidencias
        const result = array_cookies.filter((array_cookies) => array_cookies.match(pattern));

        // Obtenemos un array con las coincidencias, que solo debe ser una
        let intentos = decodeURIComponent(result[0].substring(result[0].indexOf('=') +1));

        return intentos;

    }else{
        return 0;
    }
}
```

*escribirIntentos();*

Los intentos se dibujan al ejecutarse el evento `window.onload`.

```
window.onload = () => {

    . . .

    // Escribe los intentos en la página
    document.getElementById('intentos').innerHTML = 'Intento de envíos del formualario: ' +
getIntentos();


}
```

3. Cada vez que los campos NOMBRE y APELLIDOS pierdan el foco, el contenido que se haya escrito en esos campos se convertirá a mayúsculas.

Para comprobar la pérdida del foco, habrá que estar a la escucha del evento `onblur` en los elementos.

```
    INP_NOMBRE.addEventListener('blur', (e) => {
        toUpper(e);
    });
    INP_APELLIDOS.addEventListener('blur', (e) => {
        toUpper(e);
    });
```

Se llama a la función `toUpper()` que cambia el valor de los inputs el mismo en mayúscuclas. `toUpper(e)` recibe el evento como argumento.

```
const toUpper = (e) => {
    e.target.value = e.target.value.toUpperCase();
}
```

**Nota:** el temario habla de pasar la referencia `this` como parámetro, pero la referencia que se pasa es la de la ventana.

4. Realizar una función que valide los campos de texto NOMBRE y APELLIDOS. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en los campos correspondientes.

```
/**
 * Valida tanto el nombre como el apellido.
 * @returns true si el campo es válido
 */
const validarNombreApellidos = (campo) => {

    // No puede haber ningún carácter ni antes ni después.
    // Se pide un conjunto de entre 2 y 12 que debe aparecer exactamente 1 vez.
    //      /^([a-z]{2-12}){1}$/

    // Opcionalmente puede haber un segundo nombre con las mismas careterísticas,
    // pero precedido de un espacio en blanco.
    //      /^([a-z]{3,12}){1}( [a-z]{2,12})?$/

    // El segundo nombre puede comenzar por una preposición o un artículo.
    // Este conjunto está formado por un espacio en blando y 2 o 3 letras.

    // Entre el primer y segundo grupo puede haber una preposición o un artículo

    // Flags:
    //      g: global
    //      i: No distingue mayúsculas de minúsculas

    // Cadenas válidas probadas
    //  María de la O
    //  Benito
    //  Benito Boniato
    //  Silvestre el Talones

    let pattern = /^([a-záéíóú]{2,12}){1}(((de|la|del|de la|de las|de los|y)){0,1} [a-záéíóú]{1,12})?$/gi;

    // Valor del campo
    let valor = campo.value.trim();

    // Diferenciamos el campo, ya que se usa la misma validación para el nombre que para el apellido
    let etiqueta = (campo.name == "nombre")? 'nombre' : 'apellido';

    // Si no se encuentra coincidencia se llama a la fución imprimirError para que imprima el mensaje de error
    // y establezca el color del texto en rojo.
    if (!valor.match(pattern)) {
        imprimirError(`El ${etiqueta} ${valor} tiene un formato incorrecto`, campo);
        return false;
    }

    // Para evitar problemas con el color del texto, nos aseguramos de que sea negro
    campo.style.color = "#000";
```

```
    return true;
}
```

Los nombres y los apellidos se evalúan de forma independiente.

- ✓ María del Carmen de Dios
- ✓ Juan Po Fernández
- ✓ María de la O Pérez de la Fuente
- x María por el Carmen de un Dios
- ✓ Federico Martín y Martín

## 5. Validar la EDAD que contenga solamente valores numéricos y que esté en el rango de 0 a 105. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo EDAD.

La función `validarEdad(e)` recibe como argumento el evento y de él obtiene los datos.
La función se limita a comprobar la validez de número y que este se mantenga en un rango.

En lugar de usar la función `parseInt()` se usa `isFinite()`.
El motivo es que `parseInt()` devuelve el valor entero de la cadena pasada como argumento hasta que encuentra un carácter no válido.

Number.isInteger() tampoco funciona porque el valor evaluado es de tipo texto, con lo que siempre dará error.

| Valor | parseInt() | isFinite() | Number.isInteger() |
|---|---|---|---|
| “35” | 35 | True | False |
| “35a” | 35 | False | False |
| “a35” | NaN | False | false |

El código es muy simple

```
/**
 * Valida la edad con los requerimientos:
 *  - Sólo valores numéricos.
 *  - Valores entre 0 y 105 años.
 *  - Si hay error, mostrar mensaje y poner el foco en el campo.
 *
 * @param {Event} e valor del campo edad
 * @returns true si el campo es válido
 */
const validarEdad = () => {
    let campo = document.formulario.edad;
    let valor = campo.value;
    let error = null;  // Nos aseguramos de que error tenga un valor no válido

    // Hay dos tipos de error: valor numérico y rango
    let message = [
        `${valor} no es un valor numérico`,
        `${valor} está fuera del rango entre 0 y 105 años`
```

```
    ];

    // Usamos el método isFinite(valor) que comprueba si el argumento es número finito.
    // Convierte el argumento en entero.
    // ¿Por qué no 'parseInt()'? parseInt convierte a entero el argumento hasta que
    // encuentra un carácter no válido.
    // parseInt y isFinite devuelven:
    //      parseInt("a35") NaN   isFinite("a35") false
    //      parseInt("35a")  35   isFinite("35a") false
    if (isFinite(valor)) {
        if (valor < 0 || valor > 105) {
            error = 1;
        }
    } else {
        error = 0;
    }

    if (error != null) {
        imprimirError(message[error], campo);
        return false;
    }
    campo.style.color = "#000";
    return true;

}
```

6. Validar el NIF. Utilizar una expresión regular que permita solamente 8 números un guión y una letra. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo NIF. No es necesario validar que la letra sea correcta. Explicar las partes de la expresión regular mediante comentarios.

```
/**
 * Validar el NIF. Utilizar una expresión regular que permita solamente 8 números
 * un guión y una letra. Si se produce algún error mostrar el mensaje en el contenedor
 * "errores" y poner el foco en el campo NIF. No es necesario validar que la letra
 * sea correcta.
 *
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarNif = () => {

    // Limpiamos los caracteres blancos que pudiera haber al principo y al final
    let campo = document.formulario.nif;
    let valor = campo.value.trim();
```

```
    // Patrón buscado
    // Grupo de números: \d{8} indica que debe contener exactamente 8 dígitos
    // Grupo de letras: [a-z]{1} debe haber exactamente una letra, entre a y z.
    // Antes de los números no puede haber nada: ^
    // Después de la letra no puede haber nada: $
    // Entre los números y las letras debe haber un guión, sin espacios: -
    // El flag i indica que se no se distinguen mayúsculas y minúsculas.
    let pattern = /^\d{8}-[a-z]{1}$/gi;


    if (!valor.match(pattern)) {
        imprimirError(`El correo ${valor} tiene un formato incorrecto`, campo);
        return false;
    }

    campo.style.color = "#000";
    return true;
}
```

7. Validar el E-MAIL. Utilizar una expresión regular que nos permita comprobar que el e-mail sigue un formato correcto. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo E-MAIL. Explicar las partes de la expresión regular mediante comentarios.

```
/**
 * Validar el E-MAIL. Utilizar una expresión regular que nos permita comprobar
 * que el e-mail sigue un formato correcto.
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarEmail = () => {

    // Limpiamos los caracteres blancos que pudiera haber al principo y al final
    let campo = formulario.email;
    let valor = email.value.trim();

    // Se divide el email en cuatro grupos: usuario, @ dominio y extensión de dominio.
    // Grupo 1: usuario
    //      Se compone de letras, mayúsculas o minúsculas, guión alto y bajo y punto.
    //      Debe aparecer, como mínimo, una vez.
    // Grupo 2: @
    //      Es obligatorio que aparezca
    // Grupo 3: Dominio
    //      Grupo de letras, mayúsculas y minúsculas, números y guiones alto y bajo.
    //      Solo aparece una vez.
    // Grupo 4: Niveles de dominio.
    //      Compuesto por un punto seguido por entre 2 y 4 letras munúsculas o mayúsculas.
    //      Este grupo puede aparecer 1 o 2 veces.
```

```
    // Antes y después de la cadena, no puede haber nada: ^ ... $

    // Patrón buscado
    let pattern = /^[a-zA-Z0-9-_.]+@[a-zA-Z0-9-_]+(\.[a-zA-Z]{2,4}){1,2}$/;

    if (!valor.match(pattern)) {
        imprimirError(`El email ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;

}
```

8. Validar que se haya seleccionado alguna de las PROVINCIAS. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo PROVINCIA.

```
/**
 * Validar que se haya seleccionado alguna de las PROVINCIAS.
 * @param {Event} e
 */
const validarProvincia = () => {

    let campo = formulario.provincia;

    // Comprobamos que el valor capturado no es "0"
    if (campo.value == "0") {
        imprimirError("Debes seleccionar una provincia", campo);
        return false;
    }
    campo.style.color = "#000";
    return true;

}
```

9. Validar el campo FECHA utilizando una expresión regular. Debe cumplir alguno de los siguientes formatos: dd/mm/aaaa o dd-mm-aaaa. No se pide validar que sea una fecha de calendario correcta. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo FECHA. Explicar las partes de la expresión regular mediante comentarios.

```
/**
```

```
 * Validar el campo FECHA utilizando una expresión regular.
 * Debe cumplir alguno de los siguientes formatos: dd/mm/aaaa o dd-mm-aaaa.
 * No se pide validar que sea una fecha de calendario correcta.
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarFecha = () => {

    let campo = formulario.fecha;
    let valor = campo.value.trim();

    // Hay que tener en cuenta que se exige un 0 a la izquierda para cumplir la especificación.
    // Grupo día:  ( 0[1-9] | [1-2]\d | 3[0-1] )
    //      Días del 1 al 9: debe comenzar con un 0.
    //      Días del 10 al 29: comienzan con 1 o 2 y siguen con otro número.
    //      Días 30 y 31: comienzan con 3 y uno entre 0 y 1.
    // Grupo mes:  ( 0[0-9] | 1[0-2] )
    //      Meses del 1 al 9: 0 seguido de un número del 1 al 9.
    //      Opciones 2, 3 y 4: los meses 10, 11 y 12.
    // Grupo año:  ( [0-1][0-9]{3} | 20[0-9][0-5] )
    //      Opción 1: admite desde el año 0 hasta 1999
    //      Opción 2: admite desde el año 2000 hasta 2025
    // El separado admite "/" o "-" y para elegirlo se incluye el mes en la opción.

    let pattern = /^(0[1-9]|[1-2]\d|3[0-1])(\/(0[1-9]|1[0-2])\/|-(0[1-9]|1[0-2])-)([0-1]\d{3}|20\d[0-
5])$/;

    if (!valor.match(pattern)) {
        imprimirError(`La fecha ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;
}
```

10. Validar el campo TELEFONO utilizando una expresión regular. Debe permitir 9 dígitos obligatorios. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo TELEFONO. Explicar las partes de la expresión regular mediante comentarios.

```
/**
 * Validar el campo TELEFONO utilizando una expresión regular. Debe permitir 9 dígitos obligatorios.
 * @param {Event} e
 */
const validarTelefono = () => {

    // Valor del campo
    let campo = document.formulario.telefono;
    let valor = campo.value.trim();
```

```
    // Se exigen 9 dígitos: \d{9}
    // No puede haber nada delante (^) ni detrás($).
    let pattern = /^\d{9}$/;

    if (!valor.match(pattern)) {
        imprimirError(`El teléfono ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;
}
```

11. Validar el campo HORA utilizando una expresión regular. Debe seguir el patrón de hh:mm. No es necesario validar que sea una hora correcta. Si se produce algún error mostrar el mensaje en el contenedor "errores" y poner el foco en el campo HORA. Explicar las partes de la expresión regular mediante comentarios.

```
/**
 * Validar el campo HORA utilizando una expresión regular.
 * Debe seguir el patrón de hh:mm. No es necesario validar que sea una hora correcta.
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarHora = () => {

    let campo = formulario.hora;
    let valor = campo.value.trim();

    // Se admite formato 24h, pues no hay selector de modo 12/24h.
    // Las horas van de 00:00 a 23:59
    // Hora: ( [0-1]\d | 2[0-3] )
    //      De 0 a 19: se elige entre 0 y 1 y un dígito (0 a 9).
    //      De 20 a 23: esta opción exige un 2 seguido de un número entre 0 y 3.
    // Minutos: [0-5]\d
    //      El primer dígito está comprendido entre 0 y 5, el segundo, entre 0 y 9.
    let pattern = /^([0-1]\d|2[0-3]):[0-5]\d$/;

    if (!valor.match(pattern)) {
        imprimirError(`La Hora ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;
}
```

12. Pedir confirmación de envío del formulario. Si se confirma el envío realizará el envío de los datos; en otro caso cancelará el envío.

Asociada al evento `onClick` del botón ***enviar*** está la función `procesarFormulario(e)` que recibe el evento como argumento para poder desactivar la acción por defecto del botón, que es envío del formulario.

Para cada validación de un campo del formulario se ha declarado una función que devuelve ***true*** o ***false*** dependiendo del resultado. Este resultado se guarda en un array `error`.

Para saber si el formulario puede seguir adelante, se consulta el array error en busca de un elemento ***false*** con la función `error.find((element) => element == false)`.

La función devuelve los valores encontrados, de forma que, si no hay error, debe devolver `undefined`, es decir, si no devuelve ***false*** es que no hay errores en el formulario.

En caso de no encontrarse errores, se pide la confirmación con un `confirm` que, al devolver ***true***, hace que el formulario sea enviado con la función `submit()`.

```
// Si no hay errores, err no tiene valor, es undefined
if (err != false) {

    // Solicitamos la confirmación y enviamos el formulario
    if (confirm("¿Enviar el formulario")) {

        // Establece y actualiza la cookie de intentos de envío del formualario
        setIntento();
        formulario.submit();
    }
}
```

## Código

### index.html

```
<!DOCTYPE html>
<htm xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
        <title>DWEC05 - Solución Tarea</title>
        <script type="text/javascript" src="js/scriptSolucion.js"></script>
        <style type="text/css">
            label{
                width: 150px;
                float:left;
                margin-bottom:5px;
            }
            input,select {
                width:150px;
                float:left;
                margin-bottom:5px;
            }
            fieldset{
                background:#66CCCC;
                width:350px;
                border: thick solid #306;
            }
            legend{
                border-top-width: medium;
                border-right-width: medium;
                border-bottom-width: medium;
                border-left-width: medium;
                border-top-style: solid;
                border-right-style: solid;
                border-bottom-style: solid;
                border-left-style: solid;
                background-color: #FFF;
            }
            #mensajes{
                float: left;
                background:#33FF33;
                width: 325px;
            }
            #errores{
                float: left;
                background:#FF6633;
                width: 325px;
            }
            #intentos{
                float: left;
                background:#CCFF33;
```

```
                width: 325px;
            }
            .error{
                border: solid 2px #FF0000;
            }
        </style>
    </head>
    <body>
        <fieldset>
            <legend>DWEC05 - Solución Tarea</legend>
            <form name="formulario" id="formulario" action="http://www.google.es"
method="get">
                <label for="nombre">Nombre:</label>
                <input type="text" name="nombre" id="nombre" />
                <label for="apellidos">Apellidos:</label>
                <input type="text" name="apellidos" id="apellidos" />
                <label for="edad">Edad:</label>
                <input name="edad" type="text" id="edad" maxlength="3" />
                <label for="nif">NIF:</label>
                <input name="nif" type="text" id="nif" />
                <label for="email">E-mail:</label>
                <input name="email" type="text" id="email" />
                <label for="provincia">Provincia:</label>
                <select name="provincia" id="provincia">
                    <option value="0" selected="selected">Seleccione Provincia</option>
                    <option value="C">A Coruña</option>
                    <option value="LU">Lugo</option>
                    <option value="OU">Ourense</option>
                    <option value="OU">Pontevedra</option>
                </select>
                <label for="fecha">Fecha Nacimiento:</label>
                <input name="fecha" type="text" id="fecha" />
                <label for="telefono">Teléfono:</label>
                <input name="telefono" type="text" id="telefono" maxlength="9"/>
                <label for="hora">Hora de visita:</label>
                <input name="hora" type="text" id="hora" />
                <input type="reset" name="limpiar" id="button" value="Limpiar" />
                <input type="submit" name="enviar" id="enviar" value="Enviar" />
            </form>
            <div id="errores"></div>
            <div id="intentos"></div>
        </fieldset>
    </body>
</html>
```

scriptSolucion.jswindow.onload = () => {

```
    // Inputs
    const INP_NOMBRE = document.getElementById('nombre');
    const INP_APELLIDOS = document.getElementById('apellidos');

    // Botones
    const BTN_LIMPIAR = document.getElementById('button');
    const BTN_ENVIAR = document.getElementById('enviar');

    // Eventos
    INP_NOMBRE.addEventListener('blur', (e) => toUpper(e));
    INP_APELLIDOS.addEventListener('blur', (e) => toUpper(e));

    BTN_ENVIAR.addEventListener('click', (e) => procesarFormulario(e));

    // Limpiar el formulario
    BTN_LIMPIAR.addEventListener('click', () => limpiar());

    // Escribe los intentos en la página
    document.getElementById('intentos').innerHTML = 'Intento de envíos del formualario: '
+ getIntentos();


}

// Funciones ************************************************

// Procesa el formulario
const procesarFormulario = (e) => {

    e.preventDefault();

    // El formulario aún no ha sido enviado

    // Borramos los avisos de error
    document.getElementById('errores').innerHTML = '';

    // Hay que validar los campos
    const formulario = document.getElementById('formulario');

    // // Obtener los datos del formulario, no captura los botones
    // const data = new FormData(formulario);
    // // Recorrer el formulario
    // console.clear();
    // data.forEach((value, key) => {
    //     console.log(`${key}: ${value}`);
    // });

    // Validar los campos y guardar los errores
    let error = [];
```

```
    error.push(validarNombreApellidos(formulario.nombre));
    error.push(validarNombreApellidos(formulario.apellidos));
    error.push(validarEdad());
    error.push(validarNif());
    error.push(validarEmail());
    error.push(validarProvincia());
    error.push(validarFecha());
    error.push(validarTelefono());
    error.push(validarHora());

    // Buscamos los errores en el array
    const err = error.find((element) => element == false);

    // Si no hay errores, err no tiene valor, es undefined
    if (err != false) {

        // Solicitamos la confirmación y enviamos el formulario
        if (confirm("¿Enviar el formulario")) {

            // Establece y actualiza la cookie de intentos de envío del formualario
            setIntento();
            formulario.submit();
        }
    }

};


/**
 * Devuelve el valor de la cookie 'intentos'
 */
const getIntentos = () => {

    // Patrón buscado
    let pattern = /intentos=\d+/;

    // Obtener las cookies decodificadas
    let cookies = document.cookie;

    // buscar el patrón entre la cadena de cookies
    if (cookies.match(pattern)) {

        // tenemos las cookies con el formato nombre=valor;nombre=valor...
        let array_cookies = cookies.split(';');

        // Hay que conseguir el registro intentos=\d y para ello usamos la función filter
        // que devuelve un array con las coincidencias
        const result = array_cookies.filter((array_cookies) =>
array_cookies.match(pattern));

        // Obtenemos un array con las coincidencias, que solo debe ser una
```

```
        let intentos = decodeURIComponent(result[0].substring(result[0].indexOf('=') +
1));

        return intentos;

    } else {
        return 0;
    }

}

/**
 * Imprime los mensajes de error en la pantalla.
 *
 * @param {string} message
 */
function imprimirError(message, component) {
    component.style.color = '#f00';

    let container = document.getElementById('errores');
    let p = document.createElement('p');
    let text = document.createTextNode(message);
    p.appendChild(text);
    container.appendChild(p);
}

/**
 * Al limpiar el formulario con reset() se mantiene el estilo de los input.
 * Si un campo tenía el color del texto en rojo por ser erróneo, sigue
 * manteniendo el mismo color.
 */
const limpiar = () => {

    // Borramos los avisos de error
    document.getElementById('errores').innerHTML = '';

    // Establecemos el color del texto en negro
    campos = document.querySelectorAll("input, select");
    campos.forEach(campo => {
        campo.style.color = "#000";
    });

}

/**
 * Transforma el texto del input en mayúsclas
 * @param {object} e componente que recibe la acción
 */
const toUpper = (e) => {
    e.target.value = e.target.value.toUpperCase();
}
```

```
// Registrar en su cookie el intento de envio del formulario
function setIntento() {

    // obtener los intentos e incrementarlos
    let intentos = getIntentos();
    intentos++;

    // Tiempo de vida de la cookie: 1 día
    let max_time = 24 * 60 * 60;

    // Establecemos la cookie, útil durante 5 min.
    document.cookie =
`${encodeURIComponent('intentos')}=${encodeURIComponent(intentos)};path=/;max-
age=${max_time};`;

}

/**
 * Valida la edad con los requerimientos:
 *  - Sólo valores numéricos.
 *  - Valores entre 0 y 105 años.
 *  - Si hay error, mostrar mensaje y poner el foco en el campo.
 *
 * @author Roberto Rodríguez <roberto.rodjim.1@educa.jcyl.es>
 * @param {Event} e valor del campo edad
 * @returns true si el campo es válido
 */
const validarEdad = () => {
    let campo = document.formulario.edad;
    let valor = campo.value;
    let error = null;  // Nos aseguramos de que error tenga un valor no válido

    // Hay dos tipos de error: valor numérico y rango
    let message = [
        `${valor} no es un valor numérico`,
        `${valor} está fuera del rango entre 0 y 105 años`
    ];

    // Usamos el método isFinite(valor) que comprueba si el argumento es número finito.
    // Convierte el argumento en entero.
    // ¿Por qué no 'parseInt()'? parseInt convierte a entero el argumento hasta que
    // encuentra un carácter no válido.
    // parseInt y isFinite devuelven:
    //      parseInt("a35") NaN   isFinite("a35") false
    //      parseInt("35a")  35   isFinite("35a") false
    if (isFinite(valor)) {
        if (valor < 0 || valor > 105) {
            error = 1;
        }
    } else {
        error = 0;
    }
```

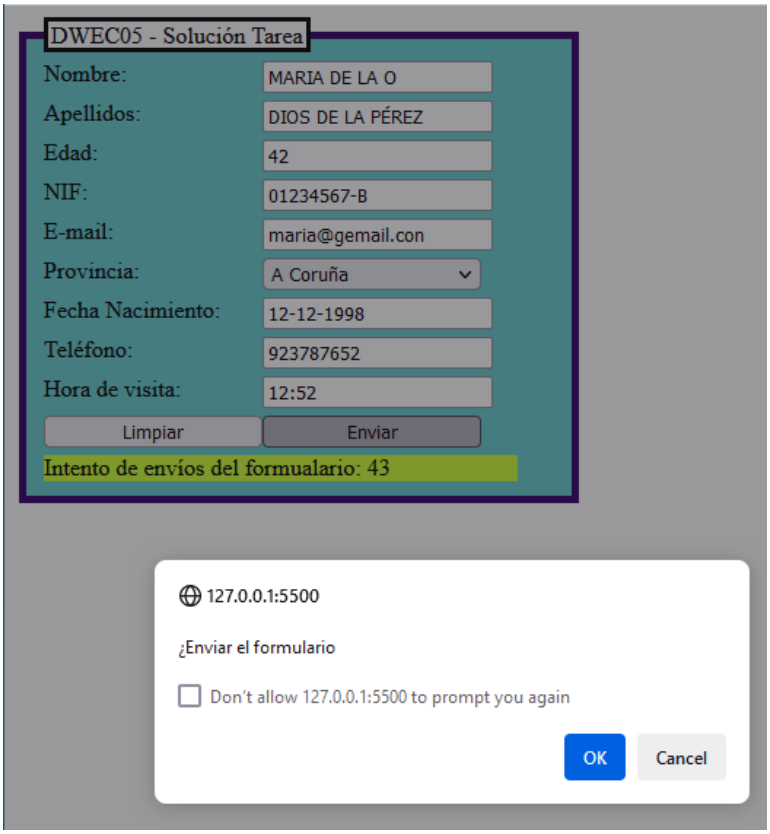
```
    if (error != null) {
        imprimirError(message[error], campo);
        return false;
    }
    campo.style.color = "#000";
    return true;

}

/**
 * Validar el E-MAIL. Utilizar una expresión regular que nos permita comprobar
 * que el e-mail sigue un formato correcto.
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarEmail = () => {

    // Limpiamos los caracteres blancos que pudiera haber al principo y al final
    let campo = formulario.email;
    let valor = email.value.trim();

    // Se divide el email en cuatro grupos: usuario, @ dominio y extensión de dominio.
    // Grupo 1: usuario
    //      Se compone de letras, mayúsculas o minúsculas, guión alto y bajo y punto.
    //      Debe aparecer, como mínimo, una vez.
    // Grupo 2: @
    //      Es obligatorio que aparezca
    // Grupo 3: Dominio
    //      Grupo de letras, mayúsculas y minúsculas, números y guiones alto y bajo.
    //      Solo aparece una vez.
    // Grupo 4: Niveles de dominio.
    //      Compuesto por un punto seguido por entre 2 y 4 letras munúsculas o mayúsculas.
    //      Este grupo puede aparecer 1 o 2 veces.
    // Antes y después de la cadena, no puede haber nada: ^ ... $

    // Patrón buscado
    let pattern = /^[a-zA-Z0-9-_.]+@[a-zA-Z0-9-_]+(\.[a-zA-Z]{2,4}){1,2}$/;

    if (!valor.match(pattern)) {
        imprimirError(`El email ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;

}

/**
 * Validar el campo FECHA utilizando una expresión regular.
 * Debe cumplir alguno de los siguientes formatos: dd/mm/aaaa o dd-mm-aaaa.
 * No se pide validar que sea una fecha de calendario correcta.
```

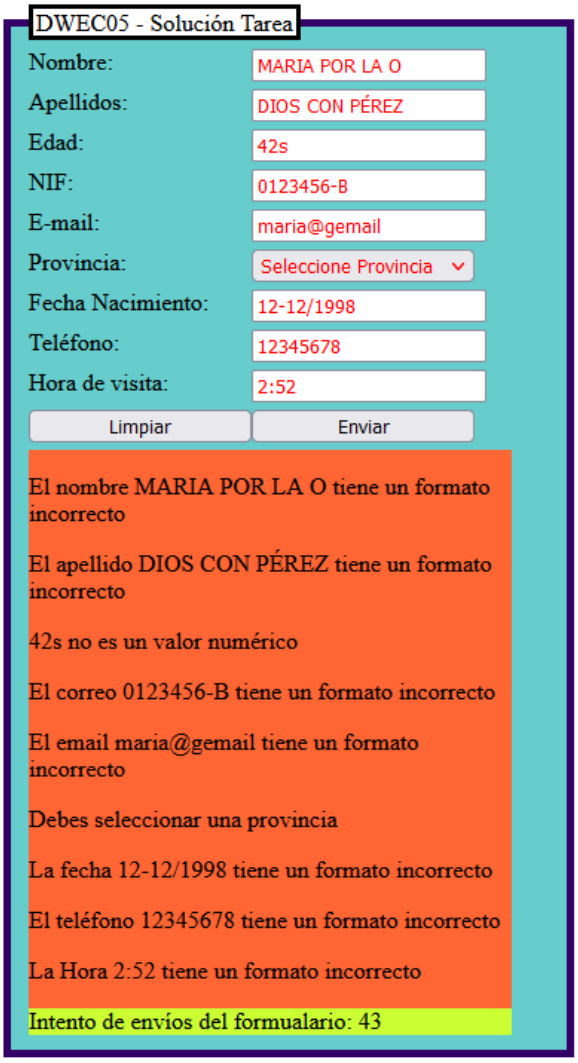
```
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarFecha = () => {

    let campo = formulario.fecha;
    let valor = campo.value.trim();

    // Hay que tener en cuenta que se exige un 0 a la izquierda para cumplir la
especificación.
    // Grupo día:  ( 0[1-9] | [1-2]\d | 3[0-1] )
    //      Días del 1 al 9: debe comenzar con un 0.
    //      Días del 10 al 29: comienzan con 1 o 2 y siguen con otro número.
    //      Días 30 y 31: comienzan con 3 y uno entre 0 y 1.
    // Grupo mes:  ( 0[0-9] | 1[0-2] )
    //      Meses del 1 al 9: 0 seguido de un número del 1 al 9.
    //      Opciones 2, 3 y 4: los meses 10, 11 y 12.
    // Grupo año:  ( [0-1][0-9]{3} | 20[0-9][0-5] )
    //      Opción 1: admite desde el año 0 hasta 1999
    //      Opción 2: admite desde el año 2000 hasta 2025
    // El separado admite "/" o "-" y para elegirlo se incluye el mes en la opción.

    let pattern = /^(0[1-9]|[1-2]\d|3[0-1])(\/(0[1-9]|1[0-2])\/|-(0[1-9]|1[0-2])-)([0-
1]\d{3}|20\d[0-5])$/;

    if (!valor.match(pattern)) {
        imprimirError(`La fecha ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;
}

/**
 * Validar el campo HORA utilizando una expresión regular.
 * Debe seguir el patrón de hh:mm. No es necesario validar que sea una hora correcta.
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarHora = () => {

    let campo = formulario.hora;
    let valor = campo.value.trim();

    // Se admite formato 24h, pues no hay selector de modo 12/24h.
    // Las horas van de 00:00 a 23:59
    // Hora: ( [0-1]\d | 2[0-3] )
    //      De 0 a 19: se elige entre 0 y 1 y un dígito (0 a 9).
    //      De 20 a 23: esta opción exige un 2 seguido de un número entre 0 y 3.
    // Minutos: [0-5]\d
    //      El primer dígito está comprendido entre 0 y 5, el segundo, entre 0 y 9.
    let pattern = /^([0-1]\d|2[0-3]):[0-5]\d$/;
```

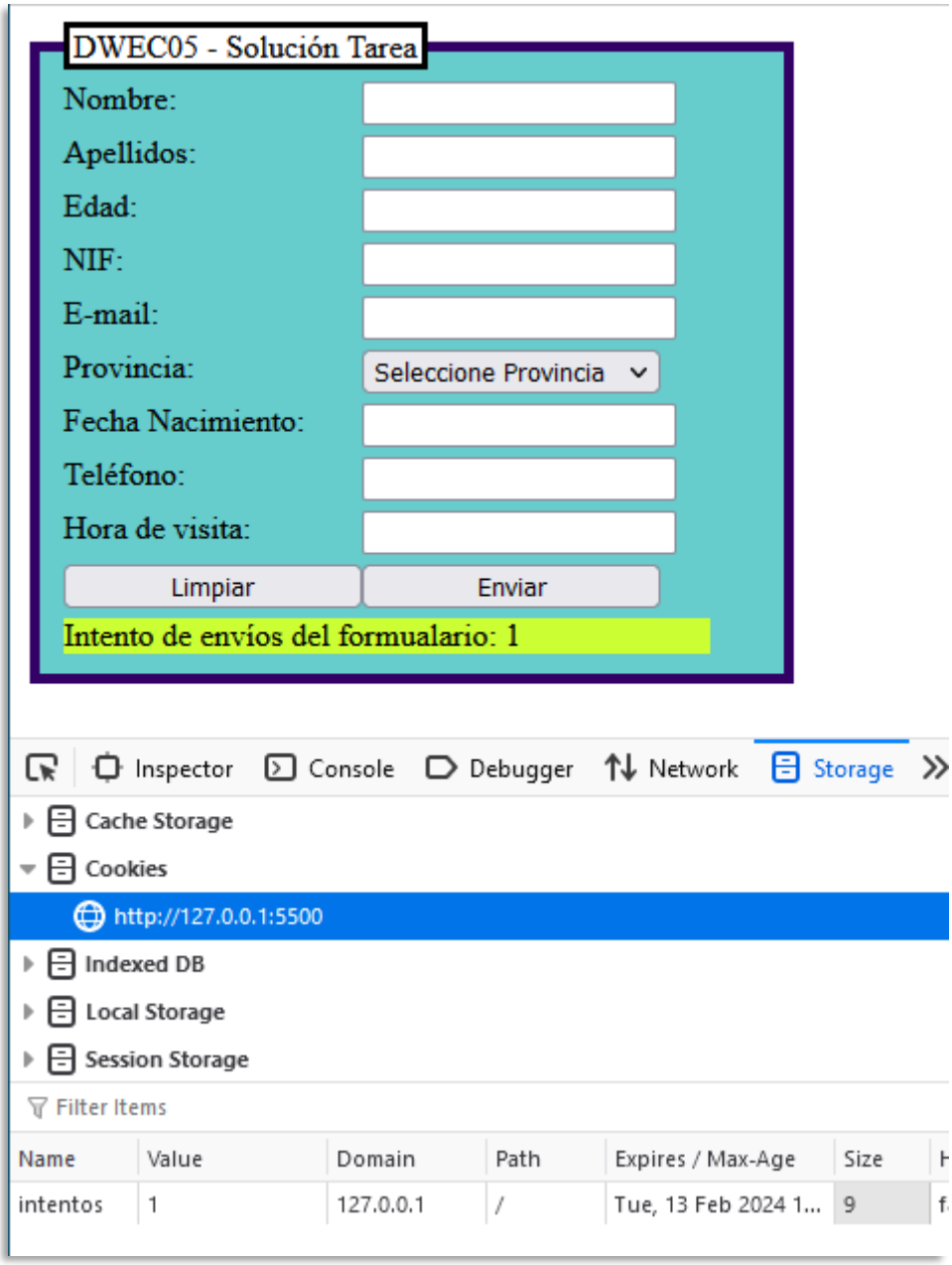
```
    if (!valor.match(pattern)) {
        imprimirError(`La Hora ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;
}

/**
 * Validar el NIF. Utilizar una expresión regular que permita solamente 8 números
 * un guión y una letra. Si se produce algún error mostrar el mensaje en el contenedor
 * "errores" y poner el foco en el campo NIF. No es necesario validar que la letra
 * sea correcta.
 *
 * @param {Event} e
 * @returns true si el campo es válido
 */
const validarNif = () => {

    // Limpiamos los caracteres blancos que pudiera haber al principo y al final
    let campo = document.formulario.nif;
    let valor = campo.value.trim();

    // Patrón buscado
    // Grupo de números: \d{8} indica que debe contener exactamente 8 dígitos
    // Grupo de letras: [a-z]{1} debe haber exactamente una letra, entre a y z.
    // Antes de los números no puede haber nada: ^
    // Después de la letra no puede haber nada: $
    // Entre los números y las letras debe haber un guión, sin espacios: -
    // El flag i indica que se no se distinguen mayúsculas y minúsculas.
    let pattern = /^\d{8}-[a-z]{1}$/gi;


    if (!valor.match(pattern)) {
        imprimirError(`El correo ${valor} tiene un formato incorrecto`, campo);
        return false;
    }

    campo.style.color = "#000";
    return true;
}

/**
 * Valida tanto el nombre como el apellido.
 * @returns true si el campo es válido
 */
const validarNombreApellidos = (campo) => {

    // No puede haber ningún carácter ni antes ni después.
    // Se pide un conjunto de entre 2 y 12 que debe aparecer exactamente 1 vez.
    //      /^([a-z]{2-12}){1}$/
```

```
    // Opcionalmente puede haber un segundo nombre con las mismas careterísticas,
    // pero precedido de un espacio en blanco.
    //      /^([a-z]{3,12}){1}( [a-z]{2,12})?$/

    // El segundo nombre puede comenzar por una preposición o un artículo.
    // Este conjunto está formado por un espacio en blando y 2 o 3 letras.

    // Entre el primer y segundo grupo puede haber una preposición o un artículo

    // Flags:
    //      g: global
    //      i: No distingue mayúsculas de minúsculas

    // Cadenas válidas probadas
    //  María de la O
    //  Benito
    //  Benito Boniato
    //  Silvestre el Talones

    let pattern = /^([a-záéíóú]{2,12}){1}(( (de|la|del|de la|de las|de los|y)){0,1} [a-
záéíóú]{1,12})?$/gi;

    // Valor del campo
    let valor = campo.value.trim();

    // Diferenciamos el campo, ya que se usa la misma validación para el nombre que para
el apellido
    let etiqueta = (campo.name == "nombre")? 'nombre' : 'apellido';

    // Si no se encuentra coincidencia se llama a la fución imprimirError para que imprima
el mensaje de error
    // y establezca el color del texto en rojo.
    if (!valor.match(pattern)) {
        imprimirError(`El ${etiqueta} ${valor} tiene un formato incorrecto`, campo);
        return false;
    }

    // Para evitar problemas con el color del texto, nos aseguramos de que sea negro
    campo.style.color = "#000";
    return true;
}

/**
 * Validar que se haya seleccionado alguna de las PROVINCIAS.
 * @param {Event} e
 */
const validarProvincia = () => {

    let campo = formulario.provincia;

    // Comprobamos que el valor capturado no es "0"
```

```
    if (campo.value == "0") {
        imprimirError("Debes seleccionar una provincia", campo);
        return false;
    }
    campo.style.color = "#000";
    return true;

}

/**
 * Validar el campo TELEFONO utilizando una expresión regular. Debe permitir 9 dígitos
obligatorios.
 * @param {Event} e
 */
const validarTelefono = () => {

    // Valor del campo
    let campo = document.formulario.telefono;
    let valor = campo.value.trim();

    // Se exigen 9 dígitos: \d{9}
    // No puede haber nada delante (^) ni detrás($).
    let pattern = /^\d{9}$/;

    if (!valor.match(pattern)) {
        imprimirError(`El teléfono ${valor} tiene un formato incorrecto`, campo);
        return false;
    }
    campo.style.color = "#000";
    return true;
}
```


DWEC05 - Solución Tarea
Nombre:
MARIA DE LA O
Apellidos:
DIOS DE LA PÉREZ
Edad:
42
NIF:
01234567-B
E-mail:
maria@gemail.con
Provincia:
A Coruña
Fecha Nacimiento:
12-12-1998
Teléfono:
923787652
Hora de visita:
12:52
Limpiar
Enviar
Intento de envíos del formulario: 43
127.0.0.1:5500
¿Enviar el formulario
Don't allow 127.0.0.1:5500 to prompt you again
OK
Cancel


DWEC05 - Solución Tarea
Nombre:
MARIA POR LA O
Apellidos:
DIOS CON PÉREZ
Edad:
42s
NIF:
0123456-B
E-mail:
maria@gemail
Provincia:
Seleccione Provincia
Fecha Nacimiento:
12-12/1998
Teléfono:
12345678
Hora de visita:
2:52
Limpiar
Enviar
El nombre MARIA POR LA O tiene un formato incorrecto
El apellido DIOS CON PÉREZ tiene un formato incorrecto
42s no es un valor numérico
El correo 0123456-B tiene un formato incorrecto
El email maria@gemail tiene un formato incorrecto
Debes seleccionar una provincia
La fecha 12-12/1998 tiene un formato incorrecto
El teléfono 12345678 tiene un formato incorrecto
La Hora 2:52 tiene un formato incorrecto
Intento de envíos del formualario: 43




*Cookie "intentos"*

## Referencias

- Tipos de *input*
  https://www.htmlquick.com/es/reference/tags/input.html

- Propiedades y métodos de los elementos HTML
  https://www.w3schools.com/jsref/dom_obj_all.asp

- Cookies

  Se hace referencia a `encodeURIComponent`
  https://es.javascript.info/cookie

- isFinite()
  https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/isFinite

- Recorrer un formulario con `forEach`
  https://matiashernandez.dev/blog/post/maneja-formulario-con-simple-javascript