

# PROYECTO WEB CLIENTE RA4 Y RA5: TRIVIADOS

Izan Fernandez Lafite

Larisa Mancebo Morfe

Alba Rojas Espejo

PROYECTO WEB CLIENTE RA4 Y RA5: TRIVIADOS .....	1
PARTICIPACION DE LOS MIEMBROS .....	3
EXPLICACION DE LA APLICACION.....	3
EXPLICACIONES DE CODIGO .....	3
INDEX.HTML.....	4
JUEGO.HTML.....	7
AUTOEVALUACION.HTML.....	11
COOKIE.JS .....	13
JUEGO.JS .....	14
OBJECT.JS .....	19
FORMULARIO.JS .....	21
Ampliación Unidad 06 – DOM y Eventos .....	29
BLOQUE A .....	29
BLOQUE B .....	34
BLOQUE C .....	37
UTILIZACION EXTRA DE JS .....	39
TABLAS DE CONTENIDOS.....	39
FORMULARIO DE AUTOEVALUACION.....	39
JUEGO INTERACTIVO .....	40
REFLEXION FINAL .....	41

## PARTICIPACION DE LOS MIEMBROS

**Izan:** Creación del objeto, validaciones del objeto, creación y manejo de cookies y muestra de resultados.

**Alba:** Creación del formulario del juego, validaciones sobre este formulario y lógica de juego.

**Larisa:** Creación de css de todas las páginas, formulario de autoevaluación y lógica de comprobaciones sobre este formulario.

## EXPLICACION DE LA APLICACION

Hemos decidido crear un juego de formato tipo test con preguntas de diversos temas de cultura general en el que se pone a prueba el conocimiento del usuario.

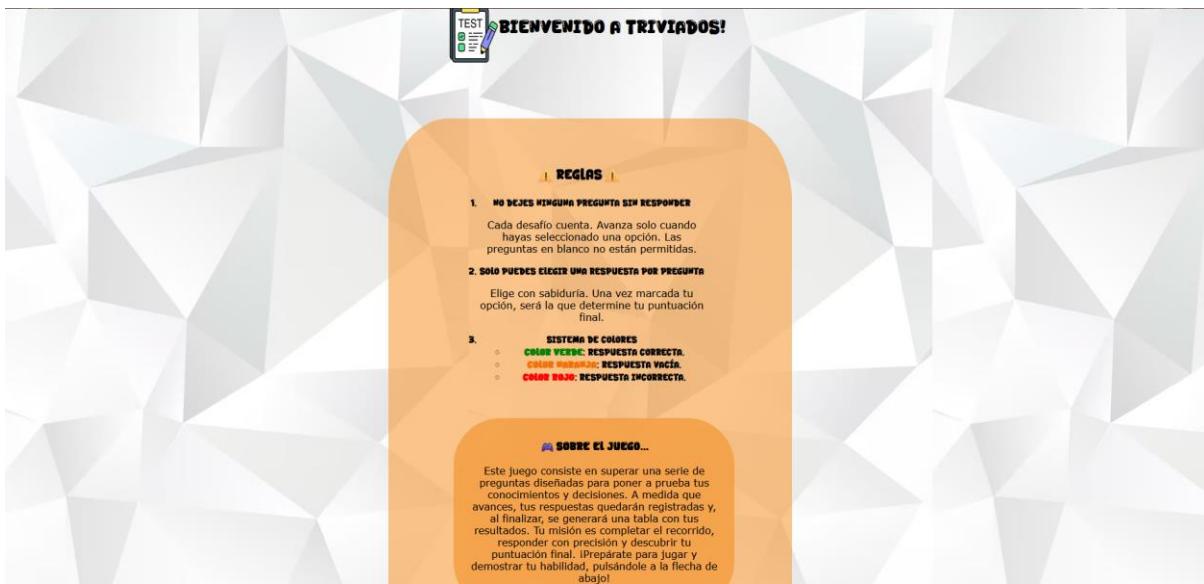
## EXPLICACIONES DE CODIGO

En esta parte nos encargaremos de explicar las distintas partes y funciones de nuestra aplicación. Empezaremos con las explicaciones del apartado de los html.

## INDEX.HTML

```
<body>
  <div class="cabecera">
    
    <h1>Bienvenido a Triviados!</h1>
  </div>
  <div class="rules">
    <h2>⚠ REGLAS ⚠</h2>
    <ol>
      <li>No dejes ninguna pregunta sin responder</li>
        <p>Cada desafío cuenta. Avanza solo cuando hayas seleccionado una opción. Las preguntas en blanco no están permitidas.</p>
      <li>Solo puedes elegir UNA respuesta por pregunta</li>
        <p>Elige con sabiduría. Una vez marcada tu opción, será la que determine tu puntuación final.</p>
      <li>Sistema de colores</li>
        <ul>
          <li><strong style="color: green;">Color Verde</strong>: Respuesta correcta.</li>
          <li><strong style="color: orange;">Color Naranja</strong>: Respuesta vacía.</li>
          <li><strong style="color: red;">Color Rojo</strong>: Respuesta incorrecta.</li>
        </ul>
      </ol>
      <div class="explication">
        <h3>💡 Sobre el juego...</h3>
        <p>Este juego consiste en superar una serie de preguntas diseñadas para poner a prueba tus conocimientos y decisiones. A medida
      </div>
    </div>
    <div class="start">
      <a href="juego.html" class="flechaA"></a><p><a href="juego.html" >START</a></p>
    </div>
  </body>
```

Esta es la vista más fácil que hay en la aplicación siendo sin más una lista de elementos explicativos de como funcionan los distintos elementos visuales referentes a las preguntas y por último una imagen con un enlace para empezar a jugar redirigiéndote al juego.



## ⚠ REGLAS ⚠

**1. NO DEJES NINGUNA PREGUNTA SIN RESPONDER**

Cada desafío cuenta. Avanza solo cuando hayas seleccionado una opción. Las preguntas en blanco no están permitidas.

**2. SOLO PUEDES ELEGIR UNA RESPUESTA POR PREGUNTA**

Elige con sabiduría. Una vez marcada tu opción, será la que determine tu puntuación final.

**3. SISTEMA DE COLORES**

- **COLOR VERDE: RESPUESTA CORRECTA.**
- **COLOR NARANJA: RESPUESTA VACÍA.**
- **COLOR ROJO: RESPUESTA INCORRECTA.**

Aquí se puede apreciar de más de cerca el texto escrito en el html para una mayor claridad.

### SOBRE EL JUEGO...

Este juego consiste en superar una serie de preguntas diseñadas para poner a prueba tus conocimientos y decisiones. A medida que avances, tus respuestas quedarán registradas y, al finalizar, se generará una tabla con tus resultados. Tu misión es completar el recorrido, responder con precisión y descubrir tu puntuación final. ¡Prepárate para jugar y demostrar tu habilidad, pulsándole a la flecha de abajo!



## START

Aquí podemos ver como se ve el html en la web una vez se abre y se empieza a ejecutar.

## JUEGO.HTML

```
<body>
  <div class="cabecera">
    
    <h1>Bienvenido a Triviados!</h1>
  </div>

  <h2 style="text-align: center;">Juego de preguntas tipo Test</h2>

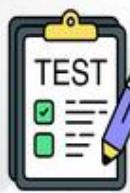
  <form id="formTest">
    <div id="contenedorPregunta"></div>
    <h3 id="error"></h3>
    <button type="submit" id="btn">Siguiente</button>
  </form>

  <h3 id="resultado"></h3>

  <div class="tablasFinales">
    <div id="tablaRes"></div>
    <div id="tablaUser"></div>
  </div>

  <script src="js/juegos.js"></script>
  <script src="js/object.js"></script>
  <script src="js/cookie.js"></script>
</body>
```

En el html del juego podemos comprobar que solo se encuentran los elementos vacíos de html, esto se debe a que se rellenan más adelante en el js con el nombre de juego.js. Los distintos elementos que podemos ver son el formulario con el id de “formTest” que contiene “contenedorPregunta” donde se rellenan las preguntas cuando se inicie el juego, el siguiente elemento del que se hablará es del id “resultado” donde al final del juego se cargará el resultado con la nota final, los dos campos con id que contienen tabla en su nombre son para cargar una tabla de resultados con las respuestas correctas a las preguntas y las respuestas proporcionadas por el usuario.



# BIENVENIDO A TRIVIADOS!

## JUEGO DE PREGUNTAS TIPO TEST

**1. ¿Qué órgano del cuerpo humano consume más energía?**

- Corazón
- Cerebro
- Hígado

Siguiente

Aquí se puede observar cómo se ven las distintas preguntas con sus posibles respuestas.



## BIENVENIDO A TRIVIADOS!

### JUEGO DE PREGUNTAS TIPO TEST

2. ¿Cuál es el metal más abundante en la corteza terrestre?

- Hierro
- Aluminio
- Calcio

[Siguiente](#)



## BIENVENIDO A TRIVIADOS!

### JUEGO DE PREGUNTAS TIPO TEST

3. ¿Qué tipo de energía produce el Sol?

- Nuclear
- Geotérmica
- Cinética

[Siguiente](#)



## BIENVENIDO A TRIVIADOS!

### JUEGO DE PREGUNTAS TIPO TEST

4. ¿Cuál es el país más poblado de África?

- Egipto
- Nigeria
- Etiopía

[Siguiente](#)

Aquí podemos diferenciar entre los 3 distintos estados de una pregunta, no respondida, acertada y errónea y como se le da feedback inmediato al usuario.

**JUEGO DE PREGUNTAS TIPO TEST**

[Jugar de Nuevo](#)

[Autoevaluación](#)

**4 / 15. NOTA DE: 2.6666666666666665**

	A	B	C		Respuestas del usuario
Pregunta 1		Cerebro		Pregunta 1	Corazón
Pregunta 2		Aluminio		Pregunta 2	Hierro
Pregunta 3	Nuclear			Pregunta 3	Nuclear
Pregunta 4		Nigeria		Pregunta 4	Egipto
Pregunta 5	Andes			Pregunta 5	Andes
Pregunta 6	Sudáfrica			Pregunta 6	Sudáfrica
Pregunta 7		Mongol		Pregunta 7	Romano
Pregunta 8		Elcano		Pregunta 8	Magallanes
Pregunta 9		Siglo XVIII		Pregunta 9	Siglo XVI
Pregunta 10		Arthur Conan Doyle		Pregunta 10	Agatha Christie
Pregunta 11		Japón		Pregunta 11	Corea
Pregunta 12		Leonardo da Vinci		Pregunta 12	Miguel Ángel
Pregunta 13		12		Pregunta 13	10
Pregunta 14		180		Pregunta 14	90
Pregunta 15	Siempre par			Pregunta 15	Siempre par

Como último apartado del juego podemos ver las dos tablas que contienen los datos que ha respondido el usuario y las respuestas correctas en formatos distintos con la nota sobre 15 y la nota sobre 10 encima de estas.

## AUTOEVALUACION.HTML

```
<body>
  <div class="cabecera">
    
    <h1>Que te ha parecido?</h1>
  </div>

  <section id="evaluation" class="sectionContainer">
    <h2>Formulario de Autoevaluación</h2>
    <form id="evaluation">
      <div class="formGroup">
        <label for="name">Nombre: </label>
        <input type="text" id="name" class="name">
      </div>
      <div class="formGroup">
        <label for="email">Email: </label>
        <input type="email" id="email" class="email">
      </div>
      <div class="formGroup">
        <label for="genre">Género: </label>
        <select name="choosenGenre" id="genre">
          <option value="">--Seleccione una opción--</option>
          <option value="woman">Mujer</option>
          <option value="man">Hombre</option>
        </select>
      </div>
      <div class="formGroup">
        <label for="satisfaction">Grado de satisfacción con el juego: </label>
        <input type="number" min="1" max="10" id="satisfaction">
      </div>
      <div class="formGroup">
        <label for="comments">Comentarios o Sugerencias: </label>
        <textarea name="comments" id="comments" placeholder="Escríbanos aquí sus sugerencias..." style="resize: none;"></textarea>
      </div>
      <button type="submit" id="sendBtn">Enviar</button>
      <button type="reset" id="resetBtn">Limpiar</button>
    </form>
  </section>
```

```
<div id="results" style="display: none;">
  <h3>Resultados de tu evaluación</h3>
  <div id="resultsContent"></div>
</div>
</section>

<button class="volver" id="btnAE"><a href="juego.html">Volver al Juego</a></button>
```

En este archivo nos encontramos los datos para llenar una vez acabas el juego en el que podemos ver datos básicos para poder identificar al usuario que está realizando el juego, estos campos como el nombre, el email o el género, para poder tener datos sobre el grupo de gente que utilizan nuestro juego. También tenemos otros campos para recibir feedback del usuario como su grado de satisfacción o comentarios para mejorar o informar de errores al realizar el flujo de la aplicación. No solo contiene los campos de texto si no también campos básicos para facilitar la vida al usuario como un botón para limpiar todos los campos u otro para volver a jugar directamente.



**QUE TE HA PARECIDO?**

**FORMULARIO DE AUTOEVALUACIÓN**

**Nombre:**

**Email:**

**Género:**

**Grado de satisfacción con el juego:**

**Comentarios o Sugerencias:**  
Escríbanos aquí sus sugerencias... Caracteres: 0

**Enviar** **Limpiar**

**RESULTADOS DE TU EVALUACIÓN**

**Resumen de tu evaluación:**

**Nombre:** asd  
**Email:** asd@asd.com  
**Género:** Mujer  
**Satisfacción:** 2/10  
**Comentarios:** asdasdasdasdas  
**Nota Global:** 2.666666666666665/10

Aquí podemos apreciar como se ve en el apartado grafico el formulario de autoevaluación y como el borde de los campos al ser marcados de marcan de un color azul. Y a la derecha podemos ver la adición que se mete en el formulario una vez se envía con los datos del participante junto a su nota del juego.

## COOKIE.JS

```
function crearCookie(nota){  
    let fecha = new Date()  
    fecha.setDate(fecha.getDate() + 30)  
    console.log(nota)  
    document.cookie = `notaMedia=${nota};expires=${fecha.toUTCString()}; path=/`  
}
```

Este método se encarga de crear una cookie con el nombre de notaMedia y la fecha del momento actual en el que se crea más 30 segundos, para su fecha de caducidad.

```
function getCookie(cname) {  
    let name = cname + "=";  
    let ca = document.cookie.split(';');  
    for(let i = 0; i < ca.length; i++) {  
        let c = ca[i];  
        while (c.charAt(0) == ' ') {  
            c = c.substring(1);  
        }  
        if (c.indexOf(name) == 0) {  
            return c.substring(name.length, c.length);  
        }  
    }  
    return "";  
}
```

El método que se encarga de recibir el nombre de la cookie que se quiere buscar en caso de que haya más de una y la splittea por el campo de ; para separar los distintos campos de importancia y se encarga de devolver el valor que se encuentre dentro de la cookie o si esta no existe devuelve un campo vacío.

```
function deleteCookie(nombre) {  
    document.cookie = `${nombre}=;expires=Thu, 01 Jan 1970 00:00:00 UTC;path=/`  
}
```

Método encargado de buscar una cookie cuyo nombre se pasa en el código para ser eliminada totalmente.

## JUEGO.JS

```
const preguntas = [
  {
    texto: "1. ¿Qué órgano del cuerpo humano consume más energía?",
    opciones: ["Corazón", "Cerebro", "Hígado"],
    correcta: "Cerebro",
  },
]
```

Lo primero que encontramos en este archivo que se encarga de la lógica del juego es una constante que es un array de objetos que contienen cada una de las preguntas, opciones y respuesta correcta en una estructura de pares clave-valor.

```
let indice = 0; // pregunta actual
let nota = 0; // aciertos totales

const contenedor = document.getElementById("contenedorPregunta");
const resultado = document.getElementById("resultado");
const tablaRes = document.getElementById("tablaRes");
const tablaUser = document.getElementById("tablaUser");
const error = document.getElementById("error");
const res = [];
```

Una vez terminamos con la definición de esta constante encontramos el resto de las variables que se utilizaran tanto para recoger los campos del html como para ciertos campos concretos más adelante en el código como se pueda ver por sus nombres descriptivos.

```
function mostrarPregunta() {
  const p = preguntas[indice];

  contenedor.innerHTML =
    `<div class="pregunta">
      <p>${p.texto}</p>
      ${p.opciones}
        .map(
          (op, i) => `
            <label>
              <input type="radio" name="respuesta" value="${op}">
              ${op}
            </label><br>
            `
        )
        .join("")
    </div>
  `;
}
```

El primer método que nos encontramos es el responsable de llenar las preguntas en un div con sus distintas opciones en radio buttons para que se puedan mostrar en la estructura previamente mostrada en el archivo juego.html. Mas en detalle se encarga de aplicar el método map para que en cada iteración que se contenga en el array de opciones dentro de p se mape con la estructura del radio button.

```
document.getElementById("formTest").addEventListener("submit", function (e) {
  e.preventDefault();

  const marcada = document.querySelector("input[name='respuesta']:checked");

  if (!marcada) {
    error.textContent = "Debes marcar una respuesta";
    contenedor.firstChild.classList.add("vacia");
    return;
  }
  res.push(marcada.value);

  if (marcada.value === preguntas[indice].correcta) {
    $(".pregunta").css("border", "3px solid green");
  } else {
    $(".pregunta").css("border", "3px solid red");
  }

  error.textContent = "";
});
```

La siguiente parte que nos encontraremos es dentro del apartado submit cuando se clique en una de las respuestas y se envie, lo primero de todo se detiene el comportamiento normal del evento submit para que no haya problemas de funcionamiento y se puedan realizar las comprobaciones necesarias siendo estas comprobar si se ha marcado alguna de las respuestas de los radio button para en caso de que no se haya hecho marcarse con un borde naranja y mostrar el mensaje de error pertinente, después de esto se comprueba en tiempo real si la respuesta es correcta o incorrecta para remarcar el borde de color verde en caso de acierto o rojo en caso de fallo, esto se realiza para que el usuario reciba un feedback inmediato a la hora de responder a la preguntas, este feedback está hecho mediante un metodo de jQuery. Después de dar este feedback se limpia el mensaje de error y se pasa a la siguiente parte del código.

```

setTimeout(() => {
    $(".pregunta").css("border", "none");

    indice++;

    // ¿Hemos terminado?
    if (indice >= preguntas.length) {
        let validador = new Validator(res);
        let gridAnswers = validador.answerGridSenderV2();
        let userAnswersGrind = validador.userAnswerGridSender();
        contenedor.innerHTML = "";
        resultado.textContent = `${validador.validateAnswers()}`;
        let cookieDatos = getCookie("notaMedia")
        if(cookieDatos){
            deleteCookie("notaMedia")
        }
        crearCookie(validador.getAccuracy())
        tablaRes.innerHTML =
            `<table style="width:100%; text-align:center;">
                <tr>
                    <th></th><th>A</th><th>B</th><th>C</th>
                </tr>
                ${gridAnswers
                    .map(
                        (fila, i) => `
                            <tr>
                                <td>Pregunta ${i + 1}</td>
                                <td>${fila[0]}</td>
                                <td>${fila[1]}</td>
                                <td>${fila[2]}</td>
                            </tr>
                            `
                    )
                    .join("")}
            </table>
`;
    }
}

```

Esta parte del código se encarga de limpiar el borde para que no deje rastros residuales de los cambios de estilos anteriores, este cambio se realiza dentro de un setTimeout, esta función lo que hace es esperar una cantidad de tiempo en milisegundos, en nuestro caso 250, antes de realizar lo que se encuentra dentro, esto se hace para que el usuario tenga tiempo de ver el resultado a su respuesta sin poder cambiar el radio button que tenía seleccionado. Después de limpiar la entrada comprobara si el índice ha llegado al final de las respuestas y si es así construirá un objeto de tipo Validator que se encargara de revisar los resultados del usuario y de mostrar su nota y las distintas tablas que se mostraran al final del flujo en el apartado de resultados. Aunque no se muestre ambas tablas se realizan de la misma forma cambiando cuantos elementos TH tiene cada una. Pero antes de realizar esta acción de las tablas se comprueba si ya hay una cookie existente con un resultado previo para eliminarla y

volver a crear otra con el nuevo resultado de este test al terminarlo y mostrar los resultados.

```
//BOTONDE REINICIO
const boton = document.getElementById("btn");
boton.textContent = "⟳ Jugar de Nuevo";
boton.type = "button";
boton.onclick = function () {
    location.reload();
};

return;
}

// Mostrar siguiente pregunta
mostrarPregunta();
}, 250);
```

Y por último encontramos un botón que refrescará la página para volver a jugar desde el principio el formulario y como se pasa a la siguiente y el timer del método mencionado en el apartado anterior.

## OBJECT.JS

```
class Validator{
    constructor(userAnswer){
        this.userAnswer = userAnswer,
        this.answers = this.answerFiller(),
        this.score = 0,
        this.accuracy = 0
    }
}
```

Dentro del objeto nos encontramos con un constructor que solo recibe como parámetro de entrada las respuestas del usuario dado que el resto de datos necesarios como la nota, las respuestas correctas, y el accuracy que en este caso hace referencia a la nota media convertida después de hacer la operación para el cálculo sobre 15. El método de answerFiller no se mostrará ya que lo único que hace es devolver un array con las respuestas y su tipo de dato correcto.

```
validateAnswers(){
    for(let i = 0; i < this.userAnswer.length; i++){
        if(this.answers[i] === this.userAnswer[i]){
            this.score += 1
        }else if(typeof this.answers[i] === "number"){
            if(this.answers[i] === Number(this.userAnswer[i])){
                this.score += 1
            }
        }
    }
    this.accuracyCounter()
    let res = `${this.score} / 15. Nota de: ${this.accuracy}`
    return res
}
```

En el método de validateAnswer podemos encontrar la logica principal que se usa para comprobar las respuestas correctas del usuario, recorriendo uno de los arrays dado que ambos siempre tendrán la misma longitud y orden, se comprobara si las respuestas son correctas, si lo son se sumara 1 al score, si es incorrecto no se sumara nada. La segunda comprobación del else if esta para comprobar cuando se están comprobando las preguntas de matemáticas, dado que estas al estar igualadas con un identico no se parsearan automáticamente, pero esta comprobación se hace justo debajo siguiendo la misma lógica antes mencionada. Una vez el bucle termina de recorrer el array, este realiza la nota media con el score para añadirlo al accuracy antes de concatenar ambos resultados y devolver el string para que se pueda mostrar por pantalla.

```

accuracyCounter(){
    this.accuracy = (this.score * 10) / 15
}

```

Este método no es más que una simple regla de tres para calcular la nota media y no mostrar el número de aciertos sobre 15 y que la deba calcular el usuario.

```

userAnswerGridSender(){
    let grid = [
        [` ${this.userAnswer[0]} `],
        [` ${this.userAnswer[1]} `],
        [` ${this.userAnswer[2]} `],
        [` ${this.userAnswer[3]} `],
        [` ${this.userAnswer[4]} `],
        [` ${this.userAnswer[5]} `],
        [` ${this.userAnswer[6]} `],
        [` ${this.userAnswer[7]} `],
        [` ${this.userAnswer[8]} `],
        [` ${this.userAnswer[9]} `],
        [` ${this.userAnswer[10]} `],
        [` ${this.userAnswer[11]} `],
        [` ${this.userAnswer[12]} `],
        [` ${this.userAnswer[13]} `],
        [` ${this.userAnswer[14]} `]
    ]
    return grid
}

```

```

answerGridSenderV2(){
    let grid = [
        [" ",` ${this.answers[0]} `," "],
        [" ",` ${this.answers[1]} `," "],
        [` ${this.answers[2]} `," "," "],
        [` ${this.answers[3]} `," "],
        [` ${this.answers[4]} `," "," "],
        [` ${this.answers[5]} `," "," "],
        [" "," ",` ${this.answers[6]} `],
        [" "," ",` ${this.answers[7]} `],
        [` ${this.answers[8]} `," "],
        [` ${this.answers[9]} `," "],
        [" "," ",` ${this.answers[10]} `],
        [` ${this.answers[11]} `," "],
        [` ${this.answers[12]} `," "],
        [` ${this.answers[13]} `," "],
        [` ${this.answers[14]} `," "," "]
    ]
    return grid
}

```

Como últimos métodos del objeto nos encontramos con dos array multidimensionales que se encargan de montar la estructura para las tablas en los resultados, a la izquierda podemos ver la más simple con solo las respuestas del usuario mostrando el numero de la pregunta y su correspondiente respuesta, y a la derecha podemos observar una estructura un poco más compleja al mostrar los campos vacíos para indicar la letra en la que se encuentran, junto con la respuesta correcta en texto.

## FORMULARIO.JS

```
window.addEventListener('load', function(){
  console.log('Formulario cargado')
  inicializarFormulario()
})
```

Este js empieza con el evento para cargar todas las variables y ser capaz de recoger los datos.

```
function inicializarFormulario(){
  const form = document.getElementById('evaluation')
  const nameInput = document.getElementById('name')
  const emailInput = document.getElementById('email')
  const genreSelect = document.getElementById('genre')
  const satisfactionInput = document.getElementById('satisfaction')
  const commentsTextarea = document.getElementById('comments')

  crearElementosValidacion()
```

Aquí podemos encontrarnos las distintas variables que se recogen para ser validadas más adelante y un metodo que se encargara de crear los elementos para mostrar errores.

```
//validaciones - input y change
nameInput.addEventListener('input', function(event){
    validarNombre(event.target)
})
```

```
genreSelect.addEventListener('change', function(event){
    validarGenero(event.target)
})
```

```
// eventos focus, blur y keydown
nameInput.addEventListener('focus', function(event){
    event.target.style.borderColor = 'blue'
})
```

```
nameInput.addEventListener('blur', function(event){
    event.target.style.borderColor = '#ddd'
    validarNombre(event.target)
})
```

```
document.addEventListener('keydown', function(event){
    console.log('Tecla presionada: ' + event.key)
})
```

En la lista de los distintos eventos que podemos ver en las fotos, vemos las distintas comprobaciones que se realizan a la hora de validar los datos con los distintos campos del formulario como por ejemplo al momento de mandarlos o en el caso del género cuando se cambia dentro del menú desplegable. También podemos observar como se encarga de realizar un foco en los elementos para mostrarlos de un color distinto para que se sepa que elemento se tiene seleccionado y este focus se elimina a la hora de seleccionar el siguiente elemento. Por último el evento de keydown simplemente cuenta cuantas letras el usuario escribe en el apartado del comentario mostrando la cuenta debajo de este.

```
form.addEventListener('submit', function(event){
    event.preventDefault()

    console.log('Formulario enviado - Validando...')

    const nombreValido = validarNombre(nameInput)
    const emailValido = validarEmail(emailInput)
    const generoValido = validarGenero(genreSelect)
    const satisfaccionValido = validarSatisfaccion(satisfactionInput)

    if(nombreValido && emailValido && generoValido && satisfaccionValido){
        procesarForm()
    }else{
        alert('Corrija los errores del formulario.')
    }
})

form.addEventListener('reset', function(event){
    setTimeout(function() {
        limpiarValidaciones()
        document.getElementById('results').style.display = 'none'
    }, 10)
})
```

Aquí podemos observar como se realizan las distintas validaciones dentro del evento submit del formulario, mostrando un alert que parara la ejecución y dará un aviso más visual al usuario de que algo fue mal con sus datos. Aparte de esta comprobacion también podemos ver cómo se limpian los datos y los errores.

```
function validarNombre(campo){
    const valor = campo.value.trim()
    const errorSpan = document.getElementById('nameError')

    if(valor === ''){
        errorSpan.textContent = 'El nombre no puede estar vacío.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    } else{
        errorSpan.style.display = 'none'
        campo.style.borderColor = 'green'
        return true;
    }
}
```

Este método se encarga de validar el nombre para que no se pueda enviar vacío y hacer los cambios para que se note cuando el campo es correcto para el envío o cuando no.

```
function validarEmail(campo){
    const valor = campo.value.trim()
    const errorSpan = document.getElementById('emailError')

    const regexEmail = /^[a-zA-Z0-9.-]+@[a-zA-Z]{2,}\.*/

    if(valor === ''){
        errorSpan.textContent = 'El email no puede estar vacío.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    }else if(!regexEmail.test(valor)){
        errorSpan.textContent = 'Email inválido. Formato: ejemplo@dominio.com.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    } else{
        errorSpan.style.display = 'none'
        campo.style.borderColor = 'green'
        return true;
    }
}
```

Este método valida el email utilizando un regexp para validar que tenga una estructura de correo correcta, aunque no tenga un proveedor valido.

```
function validarGenero(campo){
    const valor = campo.value.trim()
    const errorSpan = document.getElementById('emailError')

    if(valor === ''){
        errorSpan.textContent = 'Seleccione un género.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    } else{
        errorSpan.style.display = 'none'
        campo.style.borderColor = 'green'
        return true;
    }
}
```

Este método se encarga de que no se envié el género en blanco si no que se seleccione una opción de las posibles antes de enviar.

```
function contarCaracteres(campo){
    const valor = campo.value
    const contador = document.getElementById('charCounter')
    const numCaracteres = valor.length

    //actualizar el contador
    contador.textContent = 'Caracteres: ' + numCaracteres
}
```

Este método se encarga simplemente de contar las teclas pulsadas al escribir dentro del apartado de comentario o sugerencia.

```
function validarSatisfaccion(campo){
    const valor = campo.value.trim()
    const errorSpan = document.getElementById('satisfaccionError')

    const numero = parseInt(valor)

    if(valor === ''){
        errorSpan.textContent = 'Debes indicar tu grado de satisfacción.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    }else if(isNaN(numero)){
        errorSpan.textContent = 'Debe ser un número.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    }else if(numero < 1 || numero > 10){
        errorSpan.textContent = 'El número debe estar entre 1 y 10.'
        errorSpan.style.display = 'block'
        campo.style.borderColor = 'red'
        return false;
    } else{
        errorSpan.style.display = 'none'
        campo.style.borderColor = 'green'
        return true;
    }
}
```

Este método se encarga de que el apartado de satisfacción tenga un numero valido entre el 1 y el 10.

```
function crearElementosValidacion(){
    const nameGroup = document.getElementById('name').parentElement
    const nameError = document.createElement('span')
    nameError.id = 'nameError'
    nameError.className = 'error-message'
    nameError.style.display = 'none'
    nameGroup.appendChild(nameError)

    const emailGroup = document.getElementById('email').parentElement
    const emailError = document.createElement('span')
    emailError.id = 'emailError'
    emailError.className = 'error-message'
    emailError.style.display = 'none'
    emailGroup.appendChild(emailError)

    const genreGroup = document.getElementById('genre').parentElement
    const genreError = document.createElement('span')
    genreError.id = 'genreError'
    genreError.className = 'error-message'
    genreError.style.display = 'none'
    genreGroup.appendChild(genreError)

    const satisfactionGroup = document.getElementById('satisfaction').parentElement
    const satisfactionError = document.createElement('span')
    satisfactionError.id = 'satisfactionError'
    satisfactionError.className = 'error-message'
    satisfactionError.style.display = 'none'
    satisfactionGroup.appendChild(satisfactionError)

    const commentsGroup = document.getElementById('comments').parentElement
    const charCounter = document.createElement('span')
    charCounter.id = 'charCounter'
    charCounter.textContent = 'Caracteres: 0'
    charCounter.style.display = 'block'
    charCounter.style.fontSize = '0.9em'
    charCounter.style.color = '#666'
    charCounter.style.marginTop = '5px'
    commentsGroup.appendChild(charCounter)
}
```

Estos son los distintos elementos que se crean para cada uno de los campos al cometer un error y para el contador de caracteres.

```

function procesarForm(){
    const nombre = document.getElementById('name').value.trim();
    const email = document.getElementById('email').value.trim();
    const genero = document.getElementById('genre').value;
    const satisfaccion = document.getElementById('satisfaction').value;
    const comentarios = document.getElementById('comments').value.trim();

    const generoTexto = genero === 'woman' ? 'Mujer' : 'Hombre';

    const resultsDiv = document.getElementById('results');
    const resultsContent = document.getElementById('resultsContent');
    let notaCookie = getCookie('notaMedia')
    console.log("Cookies:", document.cookie);
    console.log(notaCookie)
    resultsContent.innerHTML =
        '<h4>Resumen de tu evaluación: </h4>' +
        '<p><strong>Nombre: </strong>' + nombre + '</p>' +
        '<p><strong>Email: </strong>' + email + '</p>' +
        '<p><strong>Género: </strong>' + generoTexto + '</p>' +
        '<p><strong>Satisfacción: </strong>' + satisfaccion + '/10</p>' +
        (comentarios ? '<p><strong>Comentarios: </strong>' + comentarios + '</p>' : '') +
        '<p><strong>Nota Global: </strong>' + notaCookie + '/10</p>';

    resultsDiv.style.display = 'block';

    alert('Formulario enviado correctamente.');
}

```

Este método se llama una vez que se ha comprobado que el formulario es correcto y cumple todos los requisitos, este mostrara por pantalla de una forma más visual los datos introducidos para dar a entender al usuario que el formulario se ha enviado correctamente. También podemos comprobar cómo se lee la cookie para poder mostrar el resultado de la nota que ha sacado el usuario al terminar el test.

## Ampliación Unidad 06 – DOM y Eventos

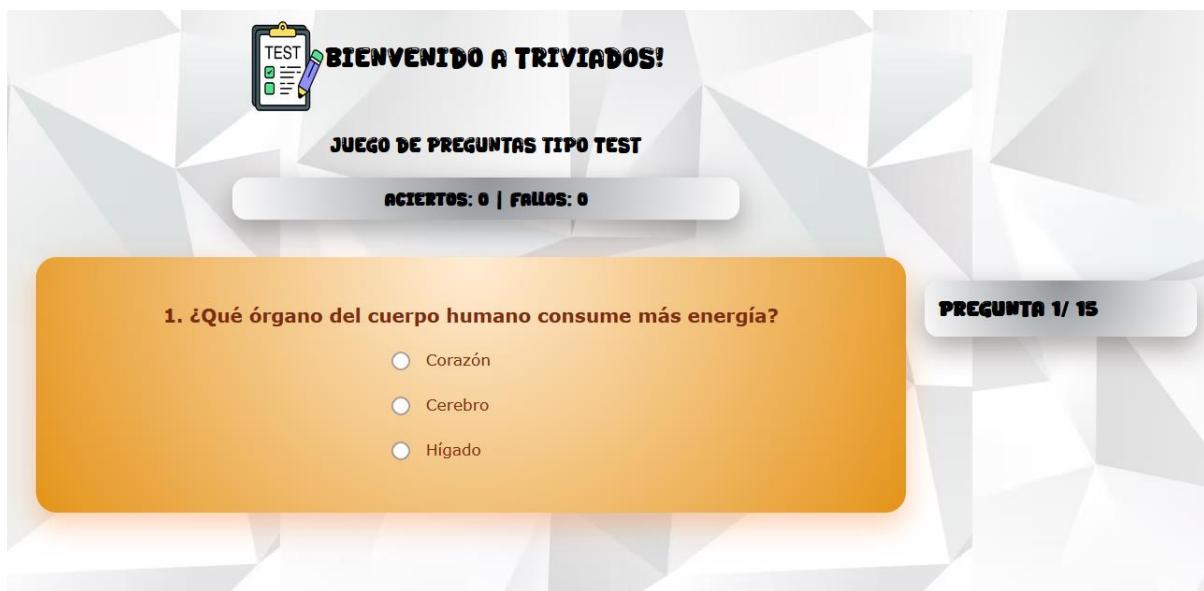
### BLOQUE A

```
const contador = document.createElement("p");
const textoCont = document.createTextNode(` ${preguntas.length}`);
const textAntes = document.createTextNode("");

contador.appendChild(textAntes);
contador.insertBefore(textoCont, null);

textAntes.nodeValue = `Pregunta ${indice + 1}/ `;

contador.setAttribute("class", "contPregunta");
console.log("Atributo de contador: " + contador.getAttribute("class"));
contenedor.appendChild(contador);
```



Este elemento dentro del archivo de juego.js sirve para crear un con un nodo de texto que sirva como contador de preguntas poniendo el numero de pregunta por el que se esta actualmente del total.

```
let bien = 0;
let mal = 0;

const contadorMB = document.createElement("p");
contadorMB.id = "contadorGlobal";
contadorMB.textContent = `Aciertos: ${bien} | Fallos: ${mal}`;

const form = document.getElementById("formTest");
form.parentNode.insertBefore(contadorMB, form);
```

Este nuevo contenedor que tambien crea un p se sirve como contador dinamico del resultado que se lleva cuando se responde una pregunta.

```
const respuestaMarcada = document.getElementsByName("respuesta");
if (!marcada) {
    error.textContent = "Debes marcar una respuesta";
    contenedor.firstChild.classList.add("vacia");
    return;
}
res.push(marcada.value);

//Nuevo RA6 - Bloque A Alba
const labels = contenedor.getElementsByTagName("label");

for (let i = 0; i < respuestaMarcada.length; i++) {
    const labelM = respuestaMarcada[i].parentNode;

    if (respuestaMarcada[i].checked) {
        if (marcada.value === preguntas[indice].correcta) {
            bien++;
            labelM.style.color = "green";
            $(".pregunta").css("border", "3px solid green");
        } else {
            mal++;
            labelM.style.color = "red";
            $(".pregunta").css("border", "3px solid red");
        }
        contadorMB.textContent = `Aciertos: ${bien} | Fallos: ${mal}`;
    } else {
        labelM.style.color = "black";
    }
    labels[i].style.fontWeight = "bold";
}
error.textContent = "";
```

## 2. ¿Cuál es el metal más abundante en la corteza terrestre?

- Hierro**
- Aluminio**
- Calcio**

**Siguiente**

Este metodo pese a no ser nuevo se ha modificado para estar de acuerdo con las nuevas especificaciones, recogiendo las respuestas mediante un getElementsByTagName y las labels estas ultimas solo se hacen para cambiar su estilo al seleccionarlas. Tambien se hace que cambien de color los padres de los radio buttons para dar un mayor feedback a la hora de mostrar si se acierta o no. Tambien se puede observar como se actualiza el texto de los aciertos.

```
if (contador) {  
    console.log("Eliminando contador" + contador);  
    contenedor.removeChild(contador);  
}  
  
contadorMB.parentNode.removeChild(contadorMB);
```

Estos dos partes del código se encargan de eliminar los componentes creados para los contadores de marcar la pregunta por la que vas y la de aciertos y fallos una vez se termina el quiz y se muestran los resultados.

## BLOQUE B

```
const boton = document.getElementById("btn")
boton.textContent = "🎲 Jugar de Nuevo"
boton.type = "button"

function reiniciarJuego(){
    location.reload()
}

//eventos add y remove EventListener
boton.removeEventListener("click", reiniciarJuego)
boton.addEventListener("click", reiniciarJuego)
```

En el comienzo del bloque B podemos ver como recoge el botón y le pone el texto y la función de para reiniciar la página volviendo a empezar el juego, antes de añadir el evento se elimina por si acaso quedara algún rastro del botón o del evento y se le vuelve a añadir para que pueda hacer su función.

```
nameInput.addEventListener('input', function(event){
    validarNombre(event.target);
    console.log("Evento target: " + event.target)
    console.log("Evento type: " + event.type)
});
```

Se modifican todas las comprobaciones que se hacen en los campos para que en vez de utilizar la constante se utilice la variable de event que hace referencia al objeto del que viene para obtener el mismo resultado que teníamos anteriormente al usar event.target para recoger el input

```
satisfactionInput.addEventListener('mouseover', function(event){
    setTimeout(() => {
        event.target.style.borderColor = "#ddd"
    }, 1000);
    event.target.style.borderColor = "lightblue";
});
```

Se modifica también la forma en la que se añaden los seteos usando también la propiedad event y metiéndolos en un setTimeout para que se cambien dinámicamente cuando pasa cierto tiempo.

```

if(nombreValido && emailValido && generoValido && satisfaccionValido){
    procesarForm();
} else{
    // alert('Corrija los errores del formulario.');
}

/* Nuevo RA6 - Bloque 8 - Eventos obligatorios - Larisa */
if(!document.getElementById('emailError').textContent && !document.getElementById('nameError').textContent && !document.getElementById('genreError').textContent && !document.getElementById('satisfactionError').textContent) {
    mensajeCorrecto.textContent = '';
    mensajeCorrecto.textContent += ' Formulario enviado correctamente';
} else{
    mensajeCorrecto.textContent = 'Corrija los errores del formulario.';
}

```

Se pasan de hacer los mensajes de error con un alert a mostrarse mas naturalmente con mensajes dentro de la aplicacion para que no sea algo que rompa el flujo normal si no que queda algo mas organico con el contenido.

```

emailInput.addEventListener('keypress', function(event){
    if (event.key === '*') {
        const errorSpan = document.getElementById('emailError');

        event.preventDefault(); // CLAVE
        event.target.style.borderColor = 'red';

        if (!errorSpan.textContent.includes('SIN asteriscos')) {
            errorSpan.textContent += ' SIN asteriscos';
        }

        errorSpan.style.display = 'block';
    }
});

```

Esta funcion se encarga de comprobar si se pulsa el boton del asterisco \* para parar el evento y sacar un mensaje de error especial para evitar ese tipo de caracteres.

```

emailInput.addEventListener('keydown', function(event){
    event.stopPropagation()
    console.log('Tecla detectada SOLO en email: ' + event.key)
})

```

Se encarga de comprobar que al pulsar una tecla se detenga la propagacion de este evento al resto de componentes el efecto de keydown.

```
document.addEventListener('keydown', function(event){
    console.log('Tecla presionada: ' + event.key);
    console.log('Type : ' + event.type);
    console.log('KeyCode : ' + event.keyCode);

    //Detectar teclas especiales :
    if(event.key === 'Enter'){
        event.target.style.borderColor = 'purple';
    }
    if(event.key === 'Shift'){
        event.target.style.borderColor = 'pink';
    }
    if(event.ctrlKey){
        event.target.style.borderColor = 'black';
    }

});
```

Esta función se encarga de mostrar distintos colores dependiendo de la tecla que se pulsa entre shift, control y enter en el formulario para cambiar el borde del elemento en el que se tenga el foco a la hora de pulsar estas teclas.

## BLOQUE C

```
window.onload = function() {
    console.log('Formulario cargado');

    if (window.addEventListener) {
        inicializarFormulario();           // Navegadores modernos
        /* Nuevo RA6 - Bloque B - Eventos obligatorios - Larisa */
        const h2 = document.getElementById('formH2')
        const textoOriginal = h2.textContent;

        h2.textContent = textoOriginal + ' ✓'
        setTimeout(function(){
            h2.textContent = textoOriginal;
        }, 1000)
    } else if (window.attachEvent) {
        /* Nuevo RA6 - Bloque C - Eventos Cross-Browser - Izan */
        inicializarFormularioIE();         // IE8 o inferior
        var h2 = document.getElementById('formH2');
        var textoOriginal = h2.innerText;

        h2.innerText = textoOriginal + ' OK';

        setTimeout(function () {
            h2.innerText = textoOriginal;
        }, 1000);
    }
}
```

Como se puede ver tanto en el formulario.js como en el juego.js se sigue la misma forma para la navegacion cross-browser usan un evento standar de todos los navegadores como .onLoad, aunque actualmente es un evento deprecated, para que se pueda cambuiar y dentro del if se hace la comprobacion de si addEventListener existe y si existe se manda al flujo normal y si no se manda al flujo normal modificado para no usar addEventlistener si no usar attachEvent.

```

function inicializarFormularioIE() {
    var form = document.getElementById('evaluation');
    var nameInput = document.getElementById('name');
    var emailInput = document.getElementById('email');

    crearElementosValidacion(); // esta función sí funciona en

    // INPUT / KEYUP
    nameInput.attachEvent('onkeyup', function () {
        validarNombre(nameInput);
    });
}

```

Uno de los mayores cambios para que funcione en navegadores con IE8- fue el cambio de todas las variables let y const a var para que fueran legibles por esos navegadores y versiones anteriores de javascript, junto con el cambio de todos los eventos mencionados anteriormente. Este cambio tambien afecta a cada funcion que se usa con event volviendo a la version anterior de como funcionaba el programa modificandose directamente cuando se recoge el elemento.

```

form.attachEvent('onsubmit', function () {
    var okNombre = validarNombre(nameInput);
    var okEmail = validarEmail(emailInput);

    if (!okNombre || !okEmail) {
        alert('Corrija los errores del formulario');
        window.event.returnValue = false; // preventDefault IE
    }
});

// KEYDOWN
emailInput.attachEvent('onkeydown', function () {
    var e = window.event;

    if (e.keyCode === 106) { // *
        e.returnValue = false;
        emailInput.style.borderColor = 'red';
    }

    e.cancelBubble = true;
});

```

Otro de los cambios mas importantes es el hecho de que los eventos de preventDefault y stop propagartion tambien se han cambio a sus versiones compatibles con IE8- aunque estas funciones ya estan deprecadas por su antiguedad, siguen funcionando en dispositivos con estos navegadores.

## UTILIZACION EXTRA DE JS

No en nuestra página no se utiliza js para ningún apartado extra aparte de las funcionalidades previamente mencionadas.

## TABLAS DE CONTENIDOS

### FORMULARIO DE AUTOEVALUACION

ELEMENTO	TIPO DE PREGUNTA	PRESENT E	DECORADO(CS S)	CORREGIDO	VALIDAD O
Caja de texto (text)	Corta	Si	Si	Si	Si
Radio Button	Test única	Si	Si	Si	Si
Combo Box (desplegables)	Test única	Si	Si	Si	Si
Check Button (check box)	Test múltiple	No	No	No	No
Selection Box (sel. múltiple)	Test múltiple	No	No	No	No
Evento de entrada de datos		Si		Si	Si
Evento de ratón		Si		Si	Si
Evento de teclado		Si		Si	Si
Evento de formulario		Si		Si	Si
Evento de foco		Si		Si	Si

Evento de carga		Si		Si	Si
-----------------	--	----	--	----	----

## JUEGO INTERACTIVO

ELEMENTO	PRESENTE	DECORADO CON CSS
<b>Array simple</b>	Si	Si
<b>Array paralelo o multidimensional</b>	Si	Si
<b>Recorrido de arrays en bucle</b>	Si	Si
<b>Métodos de arrays</b>	Si	No
<b>Funciones con parámetros y valores de retorno</b>	Si	Si
<b>Ámbito de variable</b>	Si	No
<b>Función anidada</b>	Si	No
<b>Funciones predefinidas</b>	Si	No
<b>Objeto mediante función constructora</b>	Si	No
<b>Objeto literal</b>	Si	No

## REFLEXION FINAL

Pese a que pueda parecer un proyecto sencillo, cumple la función de ser una web de entretenimiento y su sencillo diseño deja un fácil margen de mejora para un futuro para poder brindar una experiencia aún mejor y una mayor rejugabilidad para el usuario.