Juan Felipe Quintero Gutierrez

Primeros puntos:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>App</title>
    <!-- bootstrap -->
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
      crossorigin="anonymous"
    />
    <!-- icons -->
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-
icons.css"
    />
  </head>
  <body>
    <div class="container col-xl-10 col-xxl-8 px-4 py-5">
      <div class="row align-items-center g-lg-5 py-5">
        <div class="col-lg-7 text-center text-lg-start">
```

```
    <h1 class="display-6 fw-bold lh-1 mb-3" id="title">Funciones</h1>
    <p class="lead" id="cambio">
      Una función es un conjunto de instrucciones que se agrupan para
      realizar una tarea concreta y que se pueden reutilizar fácilmente.
    </p>
    <button class="btn btn-primary w-50 my-5">Read more</button>
  </div>
  <div class="col-md-10 mx-auto col-lg-5 text-center">
    <div class="p-4 border rounded-3 bg-light">
      <h2 class="fw-bold">Informacion personal</h2>
      <hr class="my-4" />
      <ul class="list-group list-group-flush">
        <li class="list-group-item">Juan Felipe Quintero Gutierrez</li>
        <li class="list-group-item">
          <i class="bi bi-envelope"></i> jfquinterogu@cesde.net
        </li>
        <li class="list-group-item">
          <i class="bi bi-whatsapp"></i> 3128283889
        </li>
        <li class="list-group-item">
          <i class="bi bi-snapchat"></i> @q_guti
        </li>
        <li class="list-group-item">
          <i class="bi bi-facebook"></i> @q.guti
        </li>
      </ul>
    </div>
  </div>
</div>
```

```
    </div>
    <!-- funcion para redireccionar -->
    <script>
      // constantes para seleccionar los elementos
      const btn = document.querySelector("button");
      const cambio = document.querySelector("#cambio");
      const title = document.querySelector("#title");


      // Boton para redigirir + funcion flecha
      btn.addEventListener("click", () => {
        setTimeout(() => {
          window.location.href = "./funciones.html";
        }, 5000);
        cambio.textContent = "En 5 segundos vas a cambiar de pagina. ¡Gracias!";
        title.textContent = "Redireccionando...";
        title.style.color = "red";
      });
    </script>


    <!-- bootstrap -->
    <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
      integrity="sha384-
IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"
      crossorigin="anonymous"
    ></script>
    <script
```

```
      src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
      integrity="sha384-
cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"
      crossorigin="anonymous"
    ></script>
  </body>
</html>



<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Funciones</title>
    <!-- bootstrap -->
    <link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
      rel="stylesheet"
      integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
      crossorigin="anonymous"
    />
    <!-- icons -->
    <link
      rel="stylesheet"
      href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.9.1/font/bootstrap-
icons.css"
    />
```

```
</head>
<body>
  <main class="container">
    <section class="row">
      <div class="my-5 text-center">
        <h1 class="display-5 fw-bold">Funciones</h1>
        <div class="col-sm-12 col-md-9 mx-auto">
          <p class="lead m-5">
            Las funciones son uno de los bloques de construcción fundamentales
            en JavaScript. Una función en JavaScript es similar a un
            procedimiento — un conjunto de instrucciones que realiza una tarea
            o calcula un valor, pero para que un procedimiento califique como
            función, debe tomar alguna entrada y devolver una salida donde hay
            alguna relación obvia entre la entrada y la salida. Para usar una
            función, debes definirla en algún lugar del ámbito desde el que
            deseas llamarla.
          </p>
        </div>
      </div>
    </section>
    <section class="row mt-5" id="ocultar">
      <div class="col-sm-12 col-md-6 text-center">
        <h2 class="display-6 text-center">Funcion Tradicional</h2>
        <p class="lead m-5" id="txt">
          Las funciones de toda la vida de JavaScript, aquellas a las que
          llamamos las normales o las estándar, se incluyen en JavaScript
          desde su aparición. Se crean mediante la declaración funcion,
          indicando después el nombre de la función y un paréntesis de
          apertura y otro de cierre, entre los cuales podemos definir los
```

```
      posibles parámetros que recibe la función. He aquí un ejemplo:
    </p>
    <pre id="ejemplos">function iniciar() {// Código}</pre>
    <button class="btn btn-success my-1" id="btnCambiar">mas</button>
  </div>
  <div class="col-sm-12 col-md-6 text-center">
    <h2 class="display-6 text-center">Funcion Flecha</h2>
    <p class="lead m-5" id="textFlecha">
      Las funciones flecha de JavaScript fueron introducidas en la versión
      ES6 de JavaScript, que apareció en el año 2015. Son algo así como
      una versión adicional con respecto a las funciones normales. Al
      igual que las funciones anónimas, carecen de nombre. Podrás
      asignarlas a una variable, pero el nombre de la variable será eso
      mismo, el nombre de la variable, no el de la función. Para declarar
      estas funciones se declara primero una variable y luego, tras los
      posibles parámetros, se escribe el conjunto de símbolos =>, que son
      la representación de una flecha. El nombre de las funciones flecha
      viene precisamente de este símbolo. Tras la flecha, se coloca el
      cuerpo de la función entre llaves. He aquí un ejemplo:
    </p>
    <pre id="ejemFlecha">const iniciar = () => {// Código}iniciar();</pre>
    <button class="btn btn-success my-1" id="btnFlecha">mas</button>
  </div>
</section>
<section class="container my-5">
  <button class="btn btn-warning w-100 my-5" id="comparar">
    Comparar
  </button>
  <p class="lead m-2" id="txtuno"></p>
```

```
        <p class="lead m-2" id="txtdos"></p>
        <pre id="ejemplocomparacion"></pre>
        <p class="lead m-2" id="txttres"></p>
        <ul id="lista" style="list-style-type: none;  display: none;"
class="list-group border-0">
          <li id="itemuno" class="list-group-item"></li>
          <li id="itemdos" class="list-group-item"></li>
        </ul>
        <p class="lead m-2" id="txtcuatro"></p>
      </section>
    </main>


    <script>
      const btnTradicional = document.querySelector("#btnCambiar");
      const ejemplos = document.querySelector("#ejemplos");
      const txt = document.querySelector("#txt");
      const btnFlecha = document.querySelector("#btnFlecha");
      const text = document.querySelector("#textFlecha");
      const ejemplo = document.querySelector("#ejemFlecha");


      btnTradicional.addEventListener("click", () => {
        if (
          btnTradicional.innerHTML == "mas" ||
          btnTradicional.innerHTML == "volver"
        ) {
          txt.innerHTML =
            "Si quisieras invocar una de estas funciones, como la de nuestro
ejemplo, tendrías que llamarla así:";
```

```
          ejemplos.innerHTML = "iniciar();";
          btnTradicional.innerHTML = "variables";
        } else if (btnTradicional.innerHTML == "variables") {
          txt.textContent =
            "Además también es posible asignar cualquier función a una variable, tal y como hacemos en este ejemplo:";
          ejemplos.textContent =
            "const iniciar = function iniciar() {// Código} iniciar();";
          btnTradicional.innerHTML = "anonimas";
        } else if (btnTradicional.innerHTML == "anonimas") {
          txt.innerHTML =
            "Si así lo prefieres, también puedes declarar estas funciones como anónimas, de modo que carezcan de nombre. La única diferencia entre una función anónima y una con nombre, es que en caso de que ocurra un error, no verás el nombre de la función en la traza que se muestre. Aquí un ejemplo de una función anónima:";
          ejemplos.innerHTML =
            "const iniciar = function () {// Código} iniciar();";
          btnTradicional.innerHTML = "parametros";
        } else if (btnTradicional.innerHTML == "parametros") {
          txt.innerHTML =
            "También podemos pasar parámetros a la función independientemente de cómo la declaremos:";
          ejemplos.innerHTML =
            "function iniciar(param1, param2) {// Código} iniciar('Soy', 'JuaneFe!');";
          btnTradicional.innerHTML = "finish";
        } else {
          btnTradicional.innerHTML = "volver";
        }
      });
```

```
      btnFlecha.addEventListener("click", () => {
        if (btnFlecha.innerHTML == "mas" || btnFlecha.innerHTML == "volver") {
          text.innerHTML =
            "En caso de que tengamos un solo parámetro, podemos obviar los
paréntesis:";
          ejemplo.innerHTML = "const iniciar = param => {// Código} iniciar();";
          btnFlecha.innerHTML = "continuar";
        } else if (btnFlecha.innerHTML == "continuar") {
          text.innerHTML =
            "En caso contrario, como cuando tenemos dos o más parámetros o cuando
no tenemos ninguno, los paréntesis son obligatorios. Por otro lado, en caso de
que la función solamente incluya una sentencia, es posible obviar las llaves de
la función. En este caso, se sobreentiende que hay una sentencia return, por lo
que el resultado de la sentencia es devuelto por la función:";
          ejemplo.innerHTML =
            "const iniciar = param => 'Soy ' + param; iniciar('JuaneFe');";
          btnFlecha.innerHTML = "seguir";
        } else if (btnFlecha.innerHTML == "seguir") {
          text.innerHTML =
            "Si así lo prefieres, también puedes declarar estas funciones como
anónimas, de modo que carezcan de nombre. La única diferencia entre una función
anónima y una con nombre, es que en caso de que ocurra un error, no verás el
nombre de la función en la traza que se muestre. Aquí un ejemplo de una función
anónima:";
          ejemplo.innerHTML =
            "const iniciar = function () {// Código} iniciar();";
          btnFlecha.innerHTML = "finish";
        } else {
          btnFlecha.innerHTML = "volver";
        }
      });
```

```
const btnComparar = document.querySelector("#comparar");
const uno = document.querySelector("#txtuno");
const dos = document.querySelector("#txtdos");
const tres = document.querySelector("#txttres");
const lista = document.querySelector("#lista");
const liuno = document.querySelector("#itemuno")
const lidos = document.querySelector("#itemdos")
const cuatro = document.querySelector("#txtcuatro");
const ejemploComparacion = document.querySelector("#ejemplocomparacion");
const ocultar = document.querySelector("#ocultar");
btnComparar.addEventListener("click", ()=>{
  ocultar.style.display = "none";
  lista.style.display ="block";
  uno.textContent = "Tal y como hemos visto, las diferencias en lo que concierne a la sintaxis son bastante grandes. Sin embargo, por ahora no son muy diferentes en cuanto a su funcionalidad. Además, tanto las funciones normales como las funciones flecha pueden ser usadas como métodos de cualquier clase usando la misma sintaxis.";
  dos.textContent = "Sin embargo existe una diferencia bastante grande entre los dos tipos de función. Hablamos del trato que cada tipo de función hace del elemento this en el cuerpo de la función. Vamos a proponer el siguiente ejemplo:";
  ejemploComparacion.textContent ="const coche = {marca: 'Ford',modelo: 'Mustang',arrancar: function() {console.log(`Arrancando el coche ${this.marca} ${this.modelo}`);},parar: () => {console.log(`Parando el coche ${this.marca} ${this.modelo}`);}}";
  tres.textContent = "La función arrancar() es una función normal o estándar, mientras que la función parar() es una función flecha. Vamos a ver el trato que se le da this en cada caso:";
  liuno.textContent ="En la función arrancar(), la palabra reservada this hace referencia al objeto coche, ya que this hace referencia al propio ámbito en el que se define la función.";
```

```
        lidos.textContent ="Sin embargo, en la función frenar(), la palabra this no hace referencia al objeto coche, sino que hace referencia al ámbito en el que está definida la constante coche. Es decir, al contexto padre que está por encima del objeto.";

        cuatro.textContent = "En las funciones flecha, this no hace referencia a la instancia del objeto en el que se define, sino que hace referencia al ámbito al que this hace referencia externamente. Esto significa que las funciones flecha no son la mejor opción a la hora definir un método de un objeto, ya que habitualmente siempre querrás tener acceso al objeto dentro de al función. En cualquier otro caso, el uso de las funciones flecha es lo que se recomienda.";

      })

    </script>



    <!-- bootstrap -->

    <script

src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"

      integrity="sha384-
IQsoLXl5PILFhosVNubq5LC7Qb9DXgDA9i+tQ8Zj3iwWAwPtgFTxbJ8NT4GN1R8p"

      crossorigin="anonymous"

    ></script>

    <script

      src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"

      integrity="sha384-
cVKIPhGWiC2Al4u+LWgxfKTRIcfu0JTxR+EQDz/bgldoEyl4H0zUF0QKbrJ0EcQF"

      crossorigin="anonymous"

    ></script>

  </body>

</html>
```

PROMESAS:

```
// Promesas
```

```
// resolve y reject -> Ejecutar funciones


//Se cumplio la promesa...
const promesaCumplida = new Promise((resolve, reject)=>{
    setTimeout(()=>{
        const result = 2+2;
        if(result === 4){
            resolve('Se realizo la suma y el resultado es '+result);
        } else{
            reject('No dio lo que se esperaba');
        }
    }, 3000);
});


//Ejecucion cuando se cumple (Ese va a dar la info ahora mismo)
promesaCumplida.then((mensaje)=>{
    console.log(mensaje);
});


//Ejecucion cuando no se cumple
promesaCumplida.catch((mensaje)=>{
    console.log(mensaje);
});


//No se cumple la promesa
const promesaRechazada = new Promise((resolve, reject)=>{
```

```
    //Accion a ejecutar
    setTimeout(()=>{
        const result = 2+3;
        if(result === 4){
            resolve('Se realizo la suma y el resultado es '+result);
        } else{
            reject('No dio lo que se esperaba');
        }
    }, 3000);
});


//Ejecucion cuando se cumple
promesaRechazada.then((mensaje)=>{
    console.log(mensaje);
});


//Ejecucion cuando no se cumple (Ese va a dar la info ahora mismo)
promesaRechazada.catch((mensaje)=>{
    console.log(mensaje);
});
```