

Documentación

Antes de iniciar, es bueno configurar linterns para buenas practicas:

En la raíz del proyecto instalamos ESLintern:

```
>npm install eslint --save-dev
```

```
>npx eslint --init
```

En el archivo eslint.config.js:

```
import js from "@eslint/js";
import globals from "globals";
import pluginReact from "eslint-plugin-react";
import { defineConfig } from "eslint/config";

export default defineConfig([
  {
    files: ["**/*.js,mjs,cjs,jsx"],
    plugins: { js },
    languageOptions: {
      globals: {
        ...globals.browser,
        ...globals.node,
      },
    },
    rules: {
      semi: ["error", "always"],
      "no-unused-vars": "error",
      "no-extra-semi": "error",
      eqeqeq: "error",
      camelcase: "error"
    },
    extends: ["js/recommended"],
  },
  pluginReact.configs.flat.recommended,
]);
```

Para complementarlo podemos usar Prettier para espaciados, sangría, etc.

Podemos subir el código a git poniendo en la terminal del código los comandos:

```
> git init
```

```
> git remote add origin https://github.com/tuusuario/nombre-repositorio.git
```

```
> git add .
```

```
> git commit -m "Mi primer commit: proyecto inicial"
```

```
> git push -u origin main
```

Podemos escribir código JavaScript, en la opción:
inspeccionar
console

```
>alert("Test") //nos muestra un modal con el texto Test
```

Lo que se hace en la consola es para hacer pruebas. Podemos ver cosas en la consola del navegador usando `>console.log()` en nuestro programa main

En código, dentro del `<body> </body>`, ponemos `<script src="main.js"></script>` para añadir código de JS. Lo mas aconsejable es ponerlo al final antes del `</body>`.

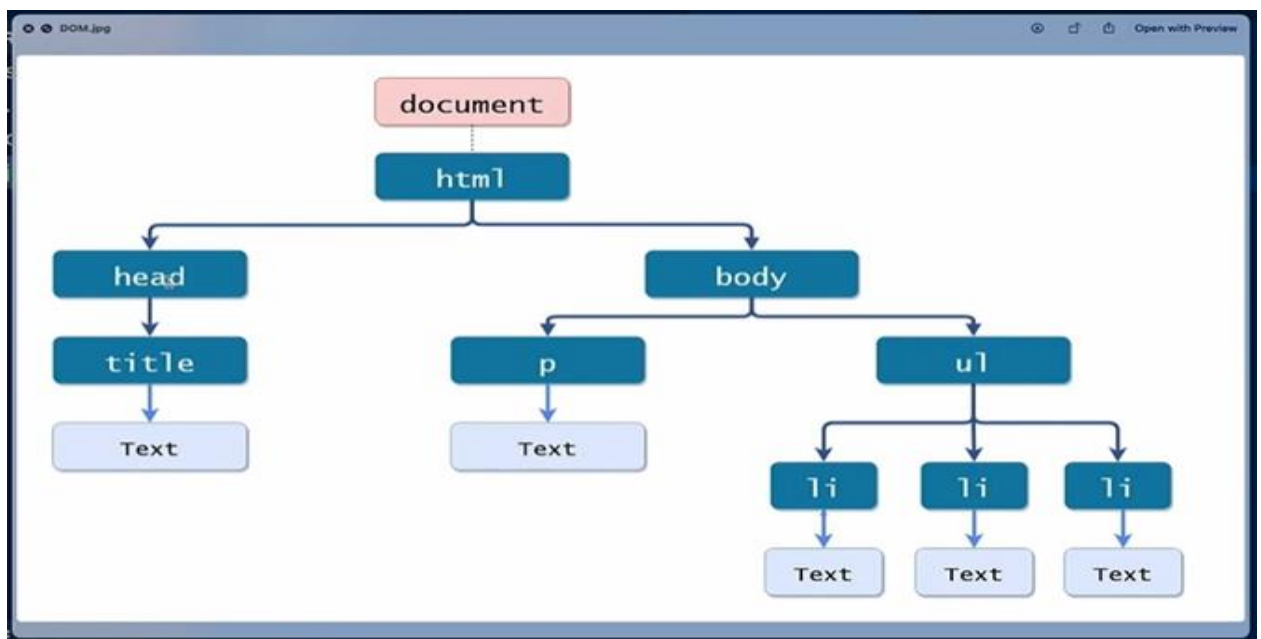
Función estática: no le pasamos argumentos.

```
function testAlert(){  
    alert("Test desde mi local que es medio malo")  
    alert("Test desde mi local 1 prueba hola")  
}  
  
testAlert()
```

Función dinámica: le pasamos argumentos.

```
function testAlert(message){  
    alert(message)  
}  
  
testAlert("Texto 123") // Salida : Modal con texto 123  
testAlert(4+5) // Salida : Modal con texto 9
```

DOM (Document Object Model)



Cuando se carga la página, se carga un documento de JS llamado "document".

```

document //Resultado: Todo el código HTML

document.getElementById('greeting') //Resultado: El elemento que tiene el
ID = greeting

document.getElementById('greeting').innerHTML = 'Hello World'
//Resultado: Cambia el contenido de ese elemento
//Se puede filtrar elementos por Id, class, tag, etc.

document.querySelector('greeting').innerHTML = 'Hello World' //Resultado:
El elemento con ese selector css

document.querySelector("p#weather").style.color = 'red' //Resultado: El
elemento con ese selector cambia de color a rojo

document.querySelector("p#weather").classList.add('nombreClase')
//Resultado: El elemento con ese selector se le añade una clase definida
en el CSS

document.querySelector("p#weather").classList.toggle('nombreClase')
//Resultado: El elemento con ese selector se le intercambia entre añadir
y borrar una clase definida en el CSS , según su estado actual.

```

Eventos:

```

document.querySelector("p#weather").addEventListener("click",
function(){alert("p element clicked")}) //Resultado: Cada vez que hago
click en el elemento p#weather saldrá un modal con el texto "p element
clicked"

document.querySelector("#open-nav-menu").addEventListener("click",
function(){
    alert("Hola bro");
});
// #open-nav-menu: Id que se busca
// "click": Evento que sucede
// function(): Lo que queremos que hacer, se puede poner nombre a la
función

//Ejemplo práctico: Añadir o quitar la clase "nav-open", segun se haga
click en un botón, para abrir o cerrar un menú.

document.querySelector("#open-nav-menu").addEventListener("click",
function(){
    document.querySelector("header nav .wrapper").classList.add("nav-
open");
});

```

```
document.querySelector("#close-nav-menu").addEventListener("click",
function(){
    document.querySelector("header nav .wrapper").classList.remove("nav-
open");
});
```

Variables:

Let y var son similares, pero let es para un entorno local y var para global. Const es para valores constantes.

Arrays son objetos y se pueden hacer operaciones en los mismos (ver ws3 para ver ejemplos de operaciones con JS).

Objetos:

student =["John", 1980, "Italy"] este array tiene esos valores, los cuales se acceden a sus posiciones con "student[0]".

Una mejor forma es: student ={"name": "John", "yearOfBirth": 1980, "country": "Italy"}, donde el objeto "student" tiene 3 propiedades (name, yearOfBirth, country) y accedemos usando la notación "student.name"

Para añadir una propiedad usamos student.id="12bc" y nuestro objeto tendrá ahora: student ={"name": "John", "yearOfBirth": 1980, "country": "Italy", "id": "12bc", }

Para añadir cambiar el valor de una propiedad usamos student.name="Myke" y nuestro objeto tendrá ahora: student ={"name": "Myke", "yearOfBirth": 1980, "country": "Italy", "id": "12bc", }

Eventos:

```
document.querySelector(".weather-group").addEventListener("click",
function(e){
    console.log(e)
}); //weather-group es class por eso lleva "." para buscarlo. "#" es para
id
//"e" es para saber que pasa como evento
```

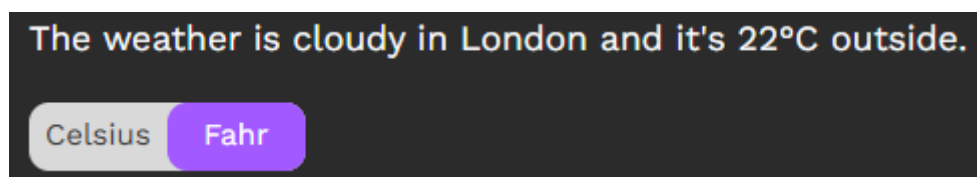
[more](#)

```
main.js:18
  PointerEvent {isTrusted: true, pointerI
▶ d: 1, width: 1, height: 1, pressure:
  0, ...}

main.js:18
  PointerEvent {isTrusted: true, pointerI
▶ d: -1, width: 1, height: 1, pressure:
  0, ...}

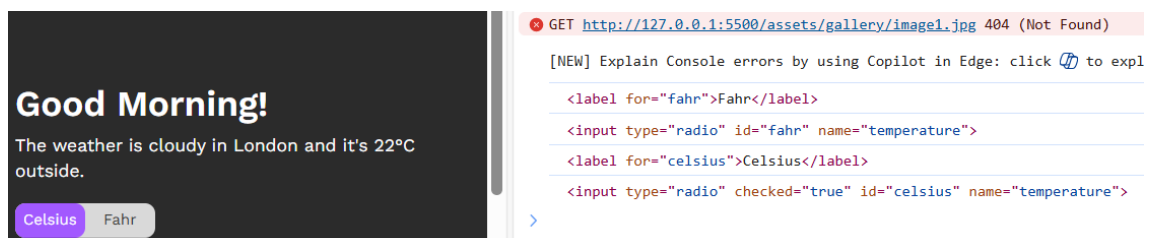
returnValue: true
screenX: 854
screenY: 304
shiftKey: false
▶ sourceCapabilities: InputDeviceCapabilities {
▶ srcElement: input#fahr
  tangentialPressure: 0
▶ target: input#fahr
  tiltX: 0
```

Para nuestro ejemplo de cambiar el texto según se seleccione el botón de Celsius Fahrenheit necesitamos el evento “target: input#fahr” que cada vez que se hace click ahí cambia su estado “target: input#celsius”



```
document.querySelector(".weather-group").addEventListener("click",
function(e){
  console.log(e.target)
});
```

Me muestra:



Donde nos interesa saber su id, ya que este cambia según que input este seleccionado (no confundir el checked es solo una propiedad para establecer la selección por default).

Código que cambia el texto (establecido por defecto en “celsiusText”):

```

let greetingText = "Good Afternoon"
let weatherCondition = "sunny"
let userLocation = "Rio de Janeiro"
let temperature = 26
let celsiusText = `The weather is ${weatherCondition} in ${userLocation}
and it's ${temperature.toFixed(1)}°C outside.`
let fahrText = `The weather is ${weatherCondition} in ${userLocation} and
it's ${celsiusToFahr(temperature).toFixed(1)}°F outside.`
document.querySelector("#greeting").innerHTML = greetingText
document.querySelector("p#weather").innerHTML = celsiusText
document.querySelector(".weather-group").addEventListener("click",
function(e){
  if(e.target.id == "celsius"){
    document.querySelector("p#weather").innerHTML = celsiusText
  }
  else{
    document.querySelector("p#weather").innerHTML = fahrText
  }
});

```

Hora y fecha en JS:

```

> new Date()
< Wed Jul 30 2025 12:24:39 GMT-0400 (hora estándar del Amazonas) {}

> new Date().getHours()
< 12

> new Date().getMinutes()
< 15

> new Date().getSeconds()
< 7

```

Cambio automático de hora:

Local time:

08 : 25 : 13

Hours Mins Secs

<div class="local-time"> flex

<div class="time-holder"> flex

08 == \$0

Hours

:

<div class="time-holder"> flex

25

mins

```

let localtime = new Date()
// elemento que tenga un atributo específico: span[data-time=hours]
document.querySelector("span[data-time=hours]").textContent =
localtime.getHours()
document.querySelector("span[data-time=minutes]").textContent =
localtime.getMinutes()

```

```
document.querySelector("span[data-time=seconds]").textContent =  
localtime.getSeconds()
```

Para que funcione todo el tiempo y no solo cuando se actualice la página:

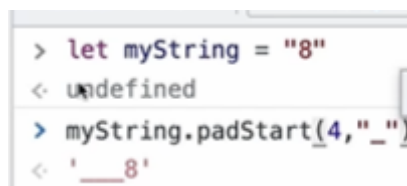
- Para retrasar algo:

```
setTimeout(function(){  
    console.log("Hola después de 3 s")  
}, 3000) // funcion que queremos retrasar 3 segundos, despues de 3  
segundos todo lo que este en la función se va a ejecutar
```

- Para que se repita algo:

```
setInterval(function(){  
    console.log("Hola después de 3 s")  
}, 3000) // funcion que queremos repetir cada 3 segundos
```

Función padStart:



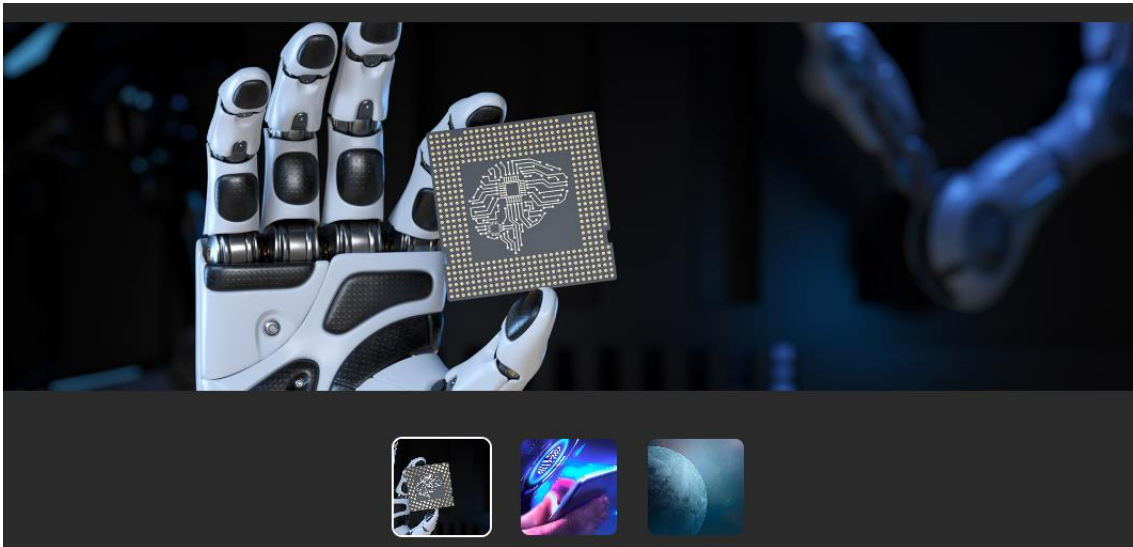
```
> let myString = "8"  
< undefined  
> myString.padStart(4, "_")  
< '___8'
```

```
document.querySelector("span[data-time=hours]").textContent =  
localtime.getHours().toString().padStart(2, "0")
```

Bucles:

```
for(let a=0; a<10; a++){  
    console.log(a)  
}  
  
let animals= ["dog", "cat", "lion", "zebra"]  
for(let a in animals){  
    console.log(animals[a])  
} //recorre el array tantas veces cuando haya de elementos, mostrando el  
valor de cada posición  
  
let animal= {name: "dog", color: "white"}  
for(let a in animal){  
    console.log(a + ": " + animal[a])  
} //recorre el objeto, mostrando la propiedad y su valor
```

Para cambiar entre las imágenes seleccionadas y que se muestre como principal:



```
let galleryImages= [  
  {src: "../assets/gallery/image1.jpg", alt: "image 1"},  
  {src: "../assets/gallery/image2.jpg", alt: "image 2"},  
  {src: "../assets/gallery/image3.jpg", alt: "image 3"},  
];  
for(let image in galleryImages){  
  console.log(galleryImages[image])  
}
```

▶ {src: '../assets/gallery/image1.jpg', alt: 'image 1'}	main.js:47
▶ {src: '../assets/gallery/image2.jpg', alt: 'image 2'}	main.js:47
▶ {src: '../assets/gallery/image3.jpg', alt: 'image 3'}	main.js:47

Otra opción es usar `forEach` que funciona con arrays:

```
galleryImages.forEach(function(image, index){  
  console.log(index)  
})
```

0	main.js:51
1	main.js:51
2	main.js:51

Como `galleryImages` es un **array** donde sus elementos son objetos, podemos usar

```
galleryImages.forEach(function(image, index){  
  console.log(image)  
}) // Si la funcion solo se usa ahi no es necesario nombrarla
```

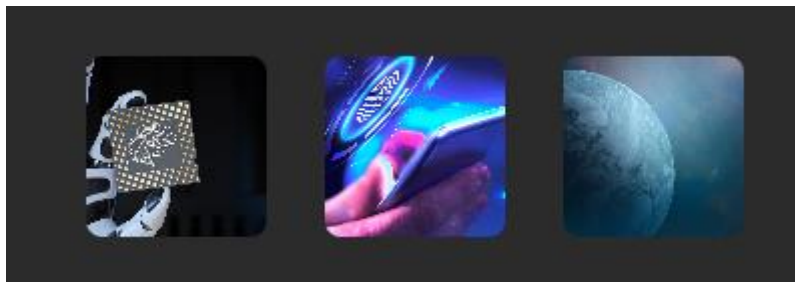

▶ {src: './assets/gallery/image1.jpg', alt: 'image 1'}	main.js:51
▶ {src: './assets/gallery/image2.jpg', alt: 'image 2'}	main.js:51
▶ {src: './assets/gallery/image3.jpg', alt: 'image 3'}	main.js:51

Seleccionamos la sección de imagen principal que tiene id="gallery" (padre) y lo que nos interesa es el hijo "img".

```
<section id="gallery" class="gallery-section">
  
```

```
let mainImage = document.querySelector("#gallery > img")
mainImage.src = galleryImages[0].src
mainImage.alt = "image 1"
```

Para las miniaturas en lugar de poner las imágenes en el HTML lo llevamos a JS:



Borramos

```
<div class="thumbnails">
  
  
  
</div>
```

Para agregarlas mediante JS, ponemos:

```
galleryImages.forEach(function(image, index){
  let thumb = document.createElement("img"); //crea una etiqueta
  thumb.src = image.src;
  thumb.alt = image.alt;
  thumb.dataset.arrayIndex = index; //establece la propiedad data-
array-index
  thumb.dataset.selected = false;
  console.log(thumb);
})
```

```

main.js:85


main.js:85


main.js:85


```

Código final para las miniaturas:

```

let galleryImages= [
  {src: "./assets/gallery/image1.jpg", alt: "image 1"},
  {src: "./assets/gallery/image2.jpg", alt: "image 2"},
  {src: "./assets/gallery/image3.jpg", alt: "image 3"},
];
let mainImage = document.querySelector("#gallery > img") // padre
id=gallery, hijo elemento img
let thumbnails = document.querySelector("#gallery .thumbnails") // padre
id=gallery, hijo clase thumbnails
mainImage.src = galleryImages[0].src //añado un valor
(galleryImages[0].src) a la propiedad src del mainImage
mainImage.alt = galleryImages[0].alt
galleryImages.forEach(function(image, index){
  let thumb = document.createElement("img"); //crea una etiqueta
  thumb.src = image.src;
  thumb.alt = image.alt;
  thumb.dataset.arrayIndex = index //establece la propiedad data-array-
index
  thumb.dataset.selected = false;
  thumbnails.appendChild(thumb)
})

```

Con esto si añadimos una nueva imagen solo la ponemos en el array “galleryImages” para que se cargue en el HTML.

Para añadir la funcionalidad de escoger y mostrar en la pantalla principal siendo marcada la miniatura como seleccionada:

```

let galleryImages= [
  {src: "./assets/gallery/image1.jpg", alt: "image 1"},
  {src: "./assets/gallery/image2.jpg", alt: "image 2"},
  {src: "./assets/gallery/image3.jpg", alt: "image 3"},
];
let mainImage = document.querySelector("#gallery > img") // padre
id=gallery, hijo elemento img

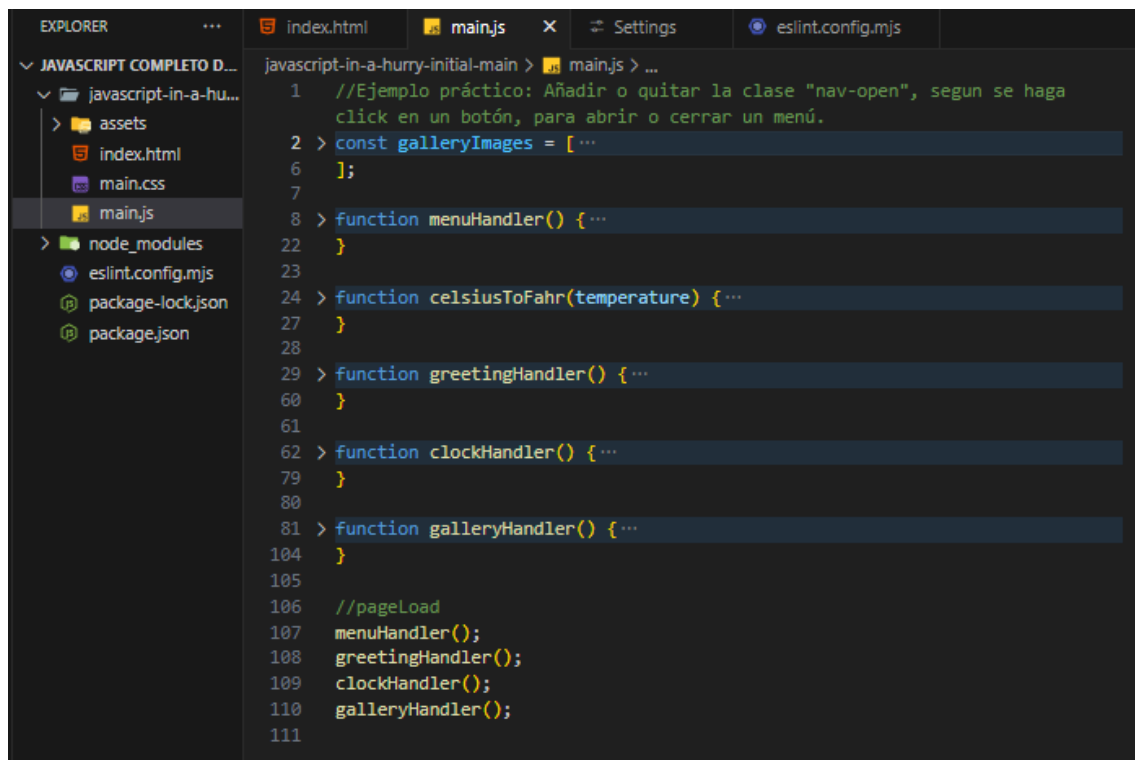
```

```

let galleryImages= [
  {src: "./assets/gallery/image1.jpg", alt: "image 1"},
  {src: "./assets/gallery/image2.jpg", alt: "image 2"},
  {src: "./assets/gallery/image3.jpg", alt: "image 3"},
];
let mainImage = document.querySelector("#gallery > img") // padre
id=gallery, hijo elemento img
let thumbnails = document.querySelector("#gallery .thumbnails") // padre
id=gallery, hijo clase thumbnails
mainImage.src = galleryImages[0].src //añado un valor
(galleryImages[0].src) a la propiedad src del mainImage
mainImage.alt = galleryImages[0].alt
galleryImages.forEach(function(image, index){
  let thumb = document.createElement("img"); //crea una etiqueta
  thumb.src = image.src; // a la etiqueta creada se la añade la
propiedad src
  thumb.alt = image.alt; // a la etiqueta creada se la añade la
propiedad alt
  thumb.dataset.arrayIndex = index //establece la propiedad data-array-
index con el valor "index" (elemento 0,1,2 del array galleryImages)
  thumb.dataset.selected = index == 0 ? true : false; //establece la
propiedad data-selected con el elemento "0" marcado por defecto
  thumb.addEventListener("click", function(e){
    let selectedIndex = e.target.dataset.arrayIndex; // escuchamos el
evento y extraemos el valor de la propiedad data-array-index
    let selectedImage = galleryImages[selectedIndex] // seleccionamos
un objeto del galleryImages
    mainImage.src = selectedImage.src // La imagen elegida la ponemos
como mainImage
    mainImage.alt = selectedImage.alt
    thumbnails.querySelectorAll("img").forEach(function(img){
      img.dataset.selected = false
    }) // del padre id=gallery con hijo clase thumbnails obtenmos
todas las imágenes y las ponemos en falso su propiedad data-selected
    e.target.dataset.selected = true // ponemos en true el evento
seleccionado
  }) //cuando se hace click escuchamos cual imagen se hizo click, se
selecciona y se pone como mainImage
  thumbnails.appendChild(thumb)
})

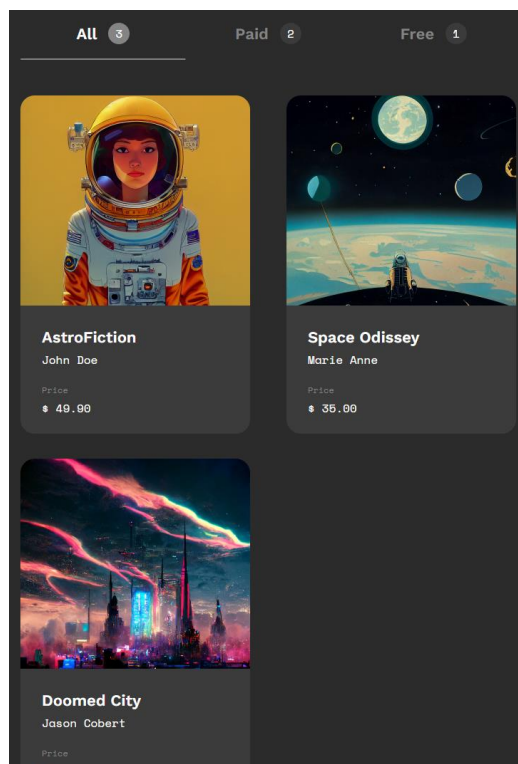
```

Hasta acá todas las líneas de código funcionan correctamente, pero son muchas líneas y se debe tener buenas prácticas. Se crearán manejadores (handlers) para cada sección.



```
1 //Ejemplo práctico: Añadir o quitar la clase "nav-open", segun se haga
2 > const galleryImages = [ ...
6 ];
7
8 > function menuHandler() { ...
22 }
23
24 > function celsiusToFahr(temperature) { ...
27 }
28
29 > function greetingHandler() { ...
60 }
61
62 > function clockHandler() { ...
79 }
80
81 > function galleryHandler() { ...
104 }
105
106 //pageLoad
107 menuHandler();
108 greetingHandler();
109 clockHandler();
110 galleryHandler();
111
```

Para la selección de productos:



Ahora esta en HTML, pero lo borramos ya que ahora será dinámico.

```

196 |         </div>
197 |
198 |         <div class="products-area">
199 | > .....<div class="product-item">...
207 | .....</div>
208 | > .....<div class="product-item">...
216 | .....</div>
217 | > .....<div class="product-item">...
225 | .....</div>
226 |

```

Creamos el array “products” para hacer lo mismo que con las imágenes y replicar lo que había en HTML.

```

//Esto había en HTML
//<div class="product-item">
//      
//      <div class="product-details">
//          <h3 class="product-title">AstroFiction</h3>
//          <p class="product-author">John Doe</p>
//          <p class="price-title">Price</p>
//          <p class="product-price">$ 49.90</p>
//</div>
//</div>

const products = [
  {
    title: "AstroFiction",
    author: "John Doe",
    price: 49.9,
    image: "./assets/products/img6.png",
  },
  {
    title: "Space Odissey",
    author: "Marie Anne",
    price: 35,
    image: "./assets/products/img1.png",
  },
  {
    title: "Doomed City",
    author: "Jason Cobert",
    price: 0,
    image: "./assets/products/img2.png",
  },
  {
    title: "Black Dog",
    author: "John Doe",
    price: 85.35,
    image: "./assets/products/img3.png",
  },
]

```

```

{
  title: "My Little Robot",
  author: "Pedro Paulo",
  price: 0,
  image: "./assets/products/img5.png",
},
{
  title: "Garden Girl",
  author: "Ankit Patel",
  price: 45,
  image: "./assets/products/img4.png",
},
];

```

En el código final es necesario entender cómo funciona el **append**, esta función nos ayuda a añadir un elemento dentro de otro (un hijo a un padre), los elementos se crean con “**document.createElement(“nombre_elemento”)**”, las clases se añaden con “**.classList.add**”, el texto dentro el elemento creado se añade con “**.textContent = “texto_a_añadir”**”. La cantidad de decimales la definimos con “**.toFixed(#decimales)**”. Todos los append se hace al final de adentro hacia afuera.

```

function productsHandler() {
  let productsSection = document.querySelector(".products-area");
  products.forEach(function (product, index) {
    let productItem = document.createElement("div");
    productItem.classList.add("product-item");

    let productImage = document.createElement("img");
    productImage.src = product.image;
    productImage.alt = product.title;

    let productDetails = document.createElement("div");
    productDetails.classList.add("product-details");

    let productTitle = document.createElement("h3");
    productTitle.classList.add("product-title");
    productTitle.textContent = product.title;

    let productAuthor = document.createElement("p");
    productAuthor.classList.add("product-author");
    productAuthor.textContent = product.author;

    let priceTitle = document.createElement("p");
    priceTitle.classList.add("price-title");
    priceTitle.textContent = product.title;

    let productPrice = document.createElement("p");

```

```

productPrice.classList.add("product-price");
productPrice.innerHTML =
  product.price === 0 ? "Free" : "$" + product.price.toFixed(2);

productDetails.append(productTitle);
productDetails.append(productAuthor);
productDetails.append(priceTitle);
productDetails.append(productPrice);

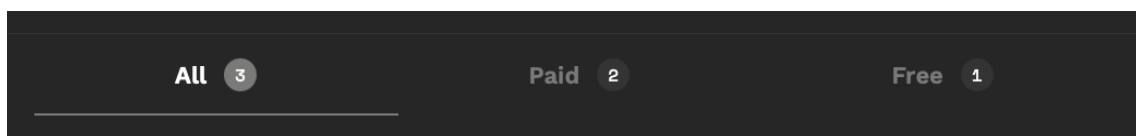
productItem.append(productImage);
productItem.append(productDetails);

productsSection.append(productItem);
});
}

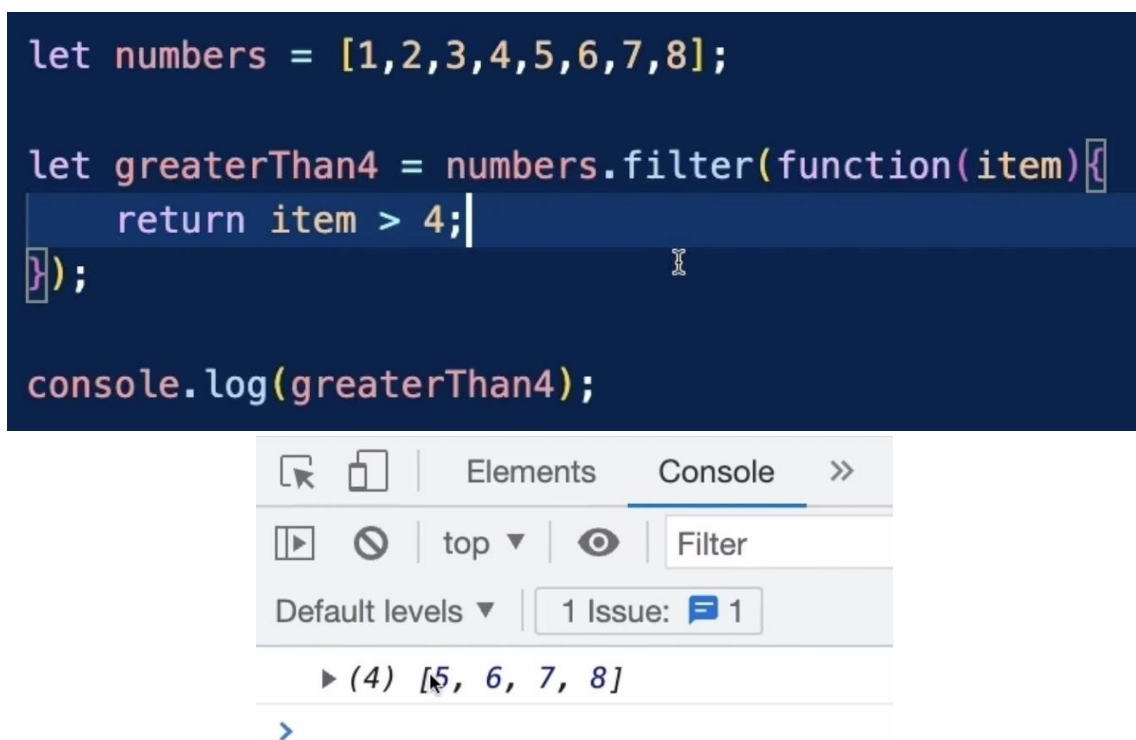
productsHandler();

```

Para el filtrado:



Usaremos el `array.Filter`, que nos ayuda a obtener los datos que cumplan con el requerimiento. Por ejemplo:



Para nuestro caso tenemos el array products y lo vamos a filtrar mediante el precio.

Primero para la cantidad total de productos:

```
// <section id="products" class="container products-section">
// <div class="products-filter">
// <input type="radio" checked="true" id="all" name="products" />
// <label for="all">
// All
// <span class="product-amount">3</span>
// </label>

let totalProducts = products.length;
document.querySelector(
  ".products-filter label[for=all] span.product-amount"
).textContent = totalProducts;
```

La sección comentada es del HTML, y se reemplaza utilizando JS. Este código va en la función productsHandler.

Para la sección de Paid y Free el análisis es similar, pero utilizando el .filter:

```
let totalPaidProducts = products.filter(function (product) {
  return product.price > 0;
});
document.querySelector(
  ".products-filter label[for=paid] span.product-amount"
).textContent = totalPaidProducts.length;

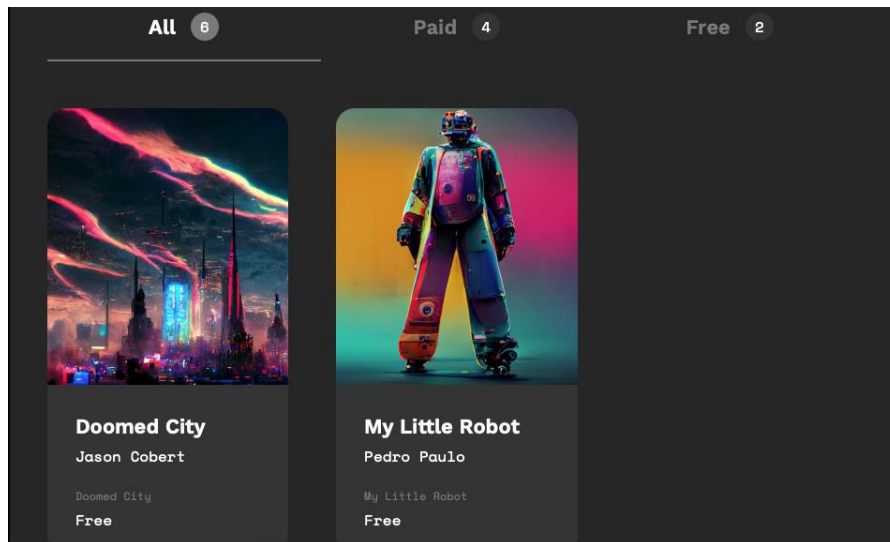
let totalFreeProducts = products.filter(function (product) {
  return product.price <= 0 || !product.price;
});
document.querySelector(
  ".products-filter label[for=free] span.product-amount"
).textContent = totalFreeProducts.length;
```

Cuando usamos el filtro podemos mostrar todos los productos o solo unos cuantos. Para poder alternar entre esto es mejor separar esa sección que muestra las cards. Entonces creamos otra función donde le pasaremos el parámetro de que cards queremos mostrar (pasaremos el array). Por ejemplo:

```
function selectedProducts(productsToShow) {
  let productsSection = document.querySelector(".products-area");
```



```
productsToShow.forEach(function (product) {  
  let productItem = document.createElement("div");  
  productItem.classList.add("product-item");  
  
  let productImage = document.createElement("img");  
  productImage.src = product.image;  
  productImage.alt = product.title;  
  
  let productDetails = document.createElement("div");  
  productDetails.classList.add("product-details");  
  
  let productTitle = document.createElement("h3");  
  productTitle.classList.add("product-title");  
  productTitle.textContent = product.title;  
  
  let productAuthor = document.createElement("p");  
  productAuthor.classList.add("product-author");  
  productAuthor.textContent = product.author;  
  
  let priceTitle = document.createElement("p");  
  priceTitle.classList.add("price-title");  
  priceTitle.textContent = product.title;  
  
  let productPrice = document.createElement("p");  
  productPrice.classList.add("product-price");  
  productPrice.textContent =  
    product.price === 0 ? "Free" : "$" + product.price.toFixed(2);  
  
  productDetails.append(productTitle);  
  productDetails.append(productAuthor);  
  productDetails.append(priceTitle);  
  productDetails.append(productPrice);  
  
  productItem.append(productImage);  
  productItem.append(productDetails);  
  
  productsSection.append(productItem);  
});  
}  
  
selectedProducts(totalFreeProducts);
```



Para seleccionar entre lo que se muestra usamos la función que muestra las cards en la sección limpiándolo cada vez que se lo llama:

```
function selectedProducts(productsToShow) {
  /*codigo..
  productsSection.textContent = ""; //elimina el contenido
  ...codigo*/
}
```

Y con este código lo llenaremos de acuerdo con el array que queramos.

```
function productsHandler() {
  /*codigo..
  selectedProducts(products);
  ... codigo ...
  let productsFilter = document.querySelector(".products-filter");
  productsFilter.addEventListener("click", function (e) {
    let productsToShow =
      e.target.id === "all"
      ? products
      : e.target.id === "paid"
      ? totalPaidProducts
      : totalFreeProducts;
    selectedProducts(productsToShow);
    ...codigo*/
  })
}
```

JavaScript Avanzado:

En esta sección aprenderemos como conectarnos con datos en internet.

El objeto navigator nos permite acceder a propiedades del dispositivo, como nivel de batería, o acceso a la ubicación del usuario (latitud y altitud).

```
navigator.geolocation.getCurrentPosition((position) => {  
  console.log(position);  
});
```

```
main.js:247  
▼ GeolocationPosition {coords: GeolocationCoordinates, t  
  timestamp: 1675781936138} ⓘ  
  ▼ coords: GeolocationCoordinates  
    accuracy: 25.976  
    altitude: null  
    altitudeAccuracy: null  
    heading: null  
    latitude: 41.38388888888889  
    longitude: -8.488333333333333  
    speed: null  
    ▶ [[Prototype]]: GeolocationCoordinates  
  timestamp: 1675781936138  
  ▶ [[Prototype]]: GeolocationPosition
```

Con arrow function podemos reducir líneas de código, por ejemplo:

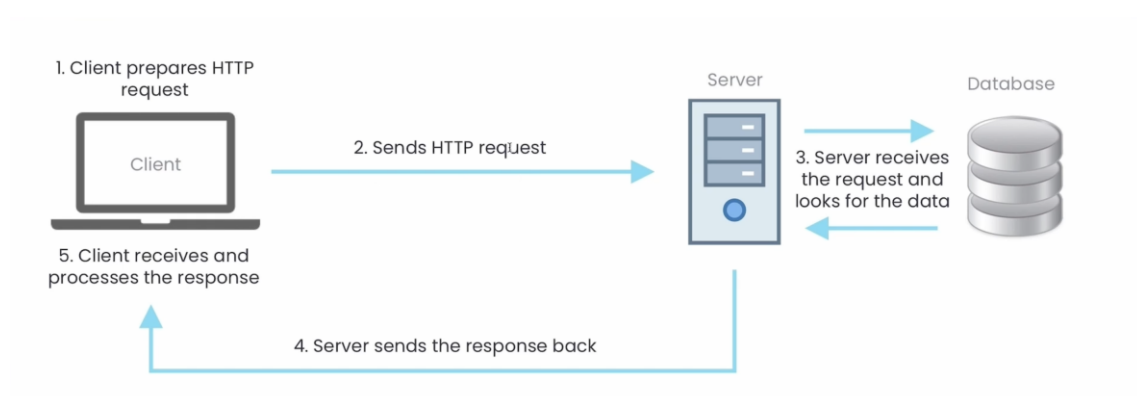
```
let totalFreeProducts = products.filter(function (product) {  
  return product.price <= 0 || !product.price;  
});
```

Se convierte en:

```
let totalFreeProducts = products.filter(  
  (product) => product.price <= 0 || !product.price  
);
```

Si hay más de una línea debemos utilizar los {} para el retorno, lo mismo con el parámetro, si solo es uno no es necesario ponerlo entre ().

¿Como funciona la solicitud HTTP?



Usando fetch podemos hacer solicitudes HTTP usando JS.

```

navigator.geolocation.getCurrentPosition( position => {
  fetch("https://opentdb.com/api.php?amount=1")
    .then(response => console.log(response));
});

```

main.js:24/

```

Response {type: 'cors', url: 'https://opentdb.com/api.
▼ php?amount=1', redirected: false, status: 200, ok: tru
e, ...} ⓘ
  body: (...)
  bodyUsed: false
  ▶ headers: Headers {}
  ok: true
  redirected: false
  status: 200
  statusText: ""
  type: "cors"
  url: "https://opentdb.com/api.php?amount=1"
  ▶ [[Prototype]]: Response

```

Fetch() se conecta con la API (para este ejemplo es una API de preguntas aleatorias) y espera una respuesta (async y wait), cuando ya hay respuesta, .then maneja la respuesta para que podamos hacer algo con ella, en ese ejemplo es mostrarla en consola.

Pero nuestro resultado es un array, a nosotros nos interesa el objeto que entrega la API, para llegar ahí necesitamos convertirlo a json y con esa respuesta podemos manejar lo que nos interesa.

```

navigator.geolocation.getCurrentPosition( position => {
  fetch("https://opentdb.com/api.php?amount=1")
    .then(response => response.json())
    .then(data => console.log(data));
});

```

```

▼ {response_code: 0, results: Array(1)} ⓘ
  response_code: 0
  ▼ results: Array(1)
    ▼ 0:
      category: "Geography"
      correct_answer: "Phoenix"
      difficulty: "easy"
      ► incorrect_answers: (3) ['Montgomery', 'Tallahassee']
      question: "What is the capital of the American state of Arizona?"
      type: "multiple"
      ► [[Prototype]]: Object
      length: 1
      ► [[Prototype]]: Array(0)
      ► [[Prototype]]: Object

```

Para nuestra API meteorológica vamos a 'OpenWeathermap.org', nos logeamos para conseguir la apiKey y la apiUrl.

```

const weatherAPIKey = "1bb443a2743f41a221cecdf806188466";
const weatherAPIURL =
`https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={lon}&appid=${weatherAPIKey}`;

navigator.geolocation.getCurrentPosition((position) => {
  let latitude = position.coords.latitude;
  let longitude = position.coords.longitude;
  let url = weatherAPIURL
    .replace("{lat}", latitude)
    .replace("{lon}", longitude);
  fetch(url)
    .then((response) => response.json())
    .then((data) => console.log(data));
});

```

Con esto veo en consola mis datos en la API del clima

```

function weatherHandler() {
  navigator.geolocation.getCurrentPosition((position) => {
    let latitude = position.coords.latitude;
    let longitude = position.coords.longitude;
    let url = weatherAPIURL
      .replace("{lat}", latitude)
      .replace("{lon}", longitude);
    fetch(url)
      .then((response) => response.json())
      .then((data) => {

```

```

const weatherCondition = data.weather[0].description;
const userLocation = data.name;
const temperature = data.main.temp;

let celsiusText = `The weather is ${weatherCondition} in ${userLocation} and it's
${temperature.toFixed(
1
)}°C outside.`;
let fahrText = `The weather is ${weatherCondition} in ${userLocation} and it's
${celsiusToFahr(
temperature
).toFixed(1)}°F outside.`;

document.querySelector("p#weather").innerHTML = celsiusText;
document
.querySelector(".weather-group")
.addEventListener("click", function (e) {
if (e.target.id === "celsius") {
document.querySelector("p#weather").innerHTML = celsiusText;
} else {
document.querySelector("p#weather").innerHTML = fahrText;
}
});
});
});
}

```

Con esta función ya obtenemos los datos y los mostramos finalmente en la página web (notar que una parte del texto fue cambiado de la función greetingHandler a esta nueva función)

Manejo de errores usando try & catch, esto nos sirve para que, si hay un error en una parte del código, el resto no se rompa.

```

try{
//todo el codigo que funciona bien
} catch (error){
//que hacer si algo dio error
console.log(error); //ideal mostrar en la consola al dev cual fue el error
}

```

También funciona con el fetch:

```
function weatherHandler() {
  navigator.geolocation.getCurrentPosition((position) => {
    let latitude = position.coords.latitude;
    let longitude = position.coords.longitude;
    let url = weatherAPIURL
      .replace("{lat}", latitude)
      .replace("{lon}", longitude);
    fetch(url)
      .then((response) => response.json())
      .then((data) => { ...
    }).catch((error) => {
      console.log(error);
      document.querySelector("p#weather").innerHTML =
        "Unable to get the weather info. Try again later";
    });
  });
}
```

Entonces, por ejemplo, si la temperatura diera error desde la API:

```
const temperature = undefined;
```

