* Prototipos

|  |
| --- |
| // Prototipo object  Let objeto = {  “propiedad”: ”datos”  };  Console.log(variable)  // Prototipo String y object  Let variable = “string”  Console.log(variable.\_proto\_)  // Prototipo number y object  Let variable = number  Console.log(variable.\_proto\_)  //Creando prototipo tipo función y tipo objecto  Let variable = function(){};  Console.log(variable.prototype) |

Todos los objetois tienen minimamente 1 prototipo de 2, el prototipo identificador y el objecto. Los prototipos que creamos como las funciones son mutables es decir se pueden modificar. Los prototipos son objetos. Estan encadenados el prototipo identificador contiene el prototipo objecto, prototipos definidos para ver el prototipo objecto (\*.\_proto\_.\_proto\_) o el creado (\*.prototype.\_proto\_). Heredan propiedades de su prototipo.

|  |
| --- |
| // \_proto\_(dunder proto)  Let variable = [1,2,3,];  Console.log(variable); //nos muestra todas las propiedades de su prototipo heredado  // \_proto\_: Array  // \_proto\_(dunder proto)  Let variable = “string”;  Console.log(variable); //nos muestra todas las propiedades de su prototipo heredado  // \_proto\_: String |

Tanto los Arrays y los Strings heredan de prototipos de los mismos nombres los cuales contienen todos los métodos que podemos usar con ellos.

Cuando se crean los prototipos estos se guardan:

|  |
| --- |
| Class Objeto{  Constructor(){};  Hablar(){  Console.log(“hola”);  }  }  Const objeto = new Objeto();  Console.log(objeto)  //modificando un prototipo  Objeto.\_proto\_hablar=()=>{  Console.log(“modificado afueara”)}  //agregando prototipo a otro  Let arr = [];  arr.\_proto\_=objeto;  arr.hablar(); |

Todos los Objetos heredan el prototipo Object.

* Modo Estricto “use strict”

1. Convierte errores de Javascript en excepciones
2. Mejorando la optimización de los errores y consigue mejores tiempos de ejecución.
3. Evita sintaxis usadas en posteriores a ES6
4. No permite que el programador realice una “mala sintaxis”

|  |
| --- |
| // Error que no se puede hacer en el modo estricto, las variables hay que declararlas  Nombre = “daniel” |

El modo estricto es una instrucción para el interprete.

* Usando modo estricto

|  |
| --- |
| “use strict”; //puedre estar en el scope global o dentro de una función como primera linea  // variables declaradas  Let nombre = “daniel”  Console.log(nombre)  //Modificar las propiedades (defineProperty() y writeable)  Object.defineProperty(obj, “nombre”, {value: “pedro”, writeable: false});  //obj.nombre=”roberto” //Error por que es solo de lectura  Console.log(obj.nombre)  //Agregar propiedades  Const obj ={nombre: “daniel”};  Object.preventExtensions(obj);// Evita que agrega propiedades  Obj.apellido = “dan”  Console.log(obj.apellido)  // no se puede agregar propiedades aun string  Const str = “dnaiel”  Str.canal = “da”  Cnosole.log(str.canal) //error  //No existen variables del mismo nombre  Function hablar(texto, texto){ //normalmente toma el ultimo cuando hay parámetros repetidos pero use strict no lo permite  Console.log(texto)  }  // Nose puede eliminar objetos con delete solo propiedades, normalmente no nos muestra nada pero igual no lo permite  Const obj = {  Nombre: “Daniel”  }  Delete obj:// error debería ser “delete obj.nombre;”  Console.log(obj.nombre)  // Las palabras reservadas no pueden usarse como variables, pero si permite await  Let package = “pedro”;  Console.log(package)  // no existe with y los números octales llevan un “o” después del primer 0.  Conole.log(0o25); // números en base 8 |

* Funciones flecha

1. Si solo hay una expresión, la retornan (solo en una línea y sin corchetes)
2. Paréntesis opcionales ante un solo parámetro (sin parámetros igual lleva parantesis)
3. No son adecuadas para ser usadas como métodos y no pueden ser usadas como constructores
4. This contextual (no tienen propio this, sino que toman el de la función que los envuelven)
5. Las reglas de modo stricto aplicadas a this son ignoradas
6. Función flecha invocada a través de los métodos call, apply y bind
7. No tienen objeto Arguments
8. No tienen propiedad del prototipo prototype
9. No se puede usar Yield (por ende no se puden usar como funciones generadoras)
10. No puede contener saltos de línea entre sus parámetros y su flecha
11. Orden de parseo
12. Retornan literales si su cuerpo esta entre ()

* Funciones recursivas

"strict mode";

// Ejemplo de funciones recursivas

const validarEdad = (err)=>{

let edad;

try {

if (err) edad = prompt(err);

else edad = prompt("introduce tu edad");

edad = parseInt(edad);

if (isNaN(edad)) throw "introduce un numero para tu edad";

if (edad > 18) console.log("sos mayor de edad");

else console.log("eres menor de edad");

} catch (e) {

validarEdad(e);

}

}

validarEdad();

* Clausuras o cierres

"strict mode";

// Las clausulas son funciones que retornan otras funciones

const saludar = (nombre)=>{

return function(){

console.log("hola" + nombre);

}

}

// Agregar la funcion a una variable

let saludo = saludar("daniel");

// Sabemos que en los eventos se recibe en los

// parametros una funcion

addEventListener("load", saludo);

* Parametro por defecto

"strict mode";

// Variables por defecto, en caso que faltasen parametros

const suma = (a,b)=>{ // Podemos poner valores por defeco en caso que faltasen

// (a, b=0), (a=0, b=0) en la misma entrada de los parametros

// si el valor b es undefined seteamos a 0

b = typeof b !== 'undefined' ? b : 0;

// b = b || 0; //o esta linea

console.log(a+b);

}

suma(15);

-Parametro rest

"strict mode";

// Recibir n parametros de entrada

// Si en los parametros queremos otras entradas ademas

// de la sintaxis rest, para que funcionen rest debe de

//ser el ultimo parametro (a,b,...num)

const suma = (...num)=>{ // Podemos poner valores por defeco en caso que faltasen

let resultado = 0;

// los parametros llegan en forma de arrays

for (let i = 0; i < num.length; i++) {

resultado += num[i];

}

console.log(resultado);

}

Suma(15);

- Operador ternario

"strict mode";

// Las condicionales pueden tratarse en una linea

// gastan menos recursos que los if

// Si hay mas de una linea de operacion van (b1,b2,...,bn)

let edad = 30;

(edad > 18) ?

// Bloque por verdadero

(

console.log('AFIRMATIVO'),

console.log('Es mayor de edad')

):

// Bloque por falso

console.log('NEGATIVO');

|  |
| --- |
| -Parámetro Spread  "strict mode";  let valor1 = "valor 1";  let valor2 = "valor 2";  let valor3 = "valor 3";  let arr = ["valor 1", "valor 2", "valor 3"];  // Esto solo nos devuelve el array  console.log(arr);  // Si quisieramos mostrar sus valores podriamos hacer arr[0],...  // Podemos usar el parámetro spread, de la siguiente manera  console.log(valor1, valor2, valor3);  console.log(...arr); // Parámetro spread, lo divide en parámetros cada valor  let arr2 = ["valor 4", "valor 5"];  // Se puede utilizar para parametros de funciones  arr.push(...arr2); // Es como decir: arr.push(arr2[0], arr2[1]);  let arr3 = [...arr, ...arr2]; // Otra manera de concatenar arrays |

|  |
| --- |
| -Operadores bit a bit (investigar)  - AND &  - OR |  - XOR ^  - NOT ~ |

|  |
| --- |
| -OBJETO DATE  "strict mode";  // El constructor por defecto  const fechaNoObjeto = Date();  console.log(fechaNoObjeto);  // Instanciamos un objeto al cual podemos utilizar sus diferentes metodos  const fecha = new Date();  // Dias de la semana D=0,L=1,M=2,Mi=3,J=4,V=5,S=6  console.log(fecha.getDay());  // Meses del año, que es el indice de un arreglo de los 12 meses, empieza por 0 termina en 11  console.log(fecha.getMonth());  // Fecha del mes  console.log(fecha.getDate());  // Año  console.log(fecha.getFullYear());  // Hora  console.log(fecha.getHours());  // Minutos  console.log(fecha.getMinutes());  // Segundos  console.log(fecha.getSeconds());  // Milisegundos  console.log(fecha.getMilliseconds());  // Diferencia Horaria en minutos  console.log(fecha.getTimezoneOffset()); |

|  |
| --- |
| -LOCALSTORAGE Y SESSIONSTORAGE  "strict mode";  // Estas APIs tienen una diferencia y es que cuando se actualice  // una página o se la cierre LocalStorage mantiene datos guardados  // en cambio SessionStorage no.  // Objeto Storage  console.log(localStorage);  console.log(sessionStorage);  // Agregando datos en formato clave-valor  localStorage.setItem("nombre","daniel");  sessionStorage.setItem("apellido", "danish");  // Obteniendo datos  localStorage.getItem("nombre");  sessionStorage.getItem("apellido");  // Eliminar datos  localStorage.removeItem("nombre");  sessionStorage.removeItem("apellido"); |

|  |
| --- |
| * DRAG AND DROP   "strict mode";  const objDeArrastre = document.querySelector(".claseObjArrastre");  const objContenedor = document.querySelector(".claseobjContenedor");  objDeArrastre.addEventListener("dragstart", e=>{      console.log("Se empezó a agarrar el objeto.");      // dataTransfer identificador(format) y valor      // con e.target.className obtenemos el nombre de clase del objeto agarrado      e.dataTransfer.setData("clase", e.target.className);  });  objDeArrastre.addEventListener("drag", ()=>{      console.log("El objeto no se suelta y esta en movimiento.");  });  objDeArrastre.addEventListener("dragend", ()=>{      console.log("El objeto se ha soltado.");  });  objContenedor.addEventListener("dragenter", ()=>{      console.log("El objeto a entrado en el área.");  });  objContenedor.addEventListener("dragover", e=>{      console.log("El objeto esta encima del área.");      // Permitimos que se suelte el objeto en el área      e.preventDefault();  // Podemos cambiar el estilo de la zona  // changeStyle(e.srcElement, “#888”);  });  objContenedor.addEventListener("drop", e=>{      console.log("El objeto se ha soltado en el área.");      // Mostramos el dato transferido, en este caso el nombre de      // de clase del objeto arrastrado con dataTransfer.      console.log(e.dataTransfer.getData("clase"));  // dataTransfer también admite archivos  // e.dataTransfer.files[0]  // Yesto podemos leerlo con FileReader  });  objContenedor.addEventListener("dragleave", ()=>{      console.log("El objeto se ha salido del área.");  }); |
|  |

|  |
| --- |
| -GEOLOCALIZACIÓN  "strict mode";  // Accedemos a funcionalidades del navegador con navigator  const geolocation  = navigator.geolocation;  // Funcion que nos devolvera la posicion, latitud y longitud  const posicion = (pos)=>{      // Objeto GeolocationPosition      console.log(pos);      // Latitud      console.log(pos.coords.latitude);      // Longitud      console.log(pos.coords.longitude);  }  // Funcion de manejo de errores  const err = ()=> console.log(e);  // Configuracion de opciones  const opciones = {      maximumAge: 0,      timeout: 3000,      enableHighAccuracy: true  }  // Metodo getCurrentPosition  geolocation.getCurrentPosition(posicion, err, opciones);  // Para estar vigilando constatemente el movimiento en  // tiempo real se utiliza el método watchPosition  //geolocation.watchPosition() INVESTIGAR COMO FUNCIONA |

|  |
| --- |
| * HISTORIAL   "strict mode";  // Vas atrás de una página  history.back();  // Vas adelante de una página  history.forward();  // Tamaño del historial  history.length  // Vas a sitios -1=atras, 0 o nada=refresca, 1=adelante  history.go(-1);  history.go(0);  history.go();  history.go(1);  // Modifica la URL y conserva la info (state,title,url)  // guarda entradas en el historial  history.pushState({nombre: "daniel"}, "", "?modificacion");  // Utilizamos el evento popstate para apenas volvamos a  // las rutas que modificamos nos retorne el estado de dicha  // ruta modificada.  window.addEventListener("popstate", (e)=>{      // Mostramos lo que hay en estado      console.log(e.state);  });  // Modifica la URL  // No guarda entradas en el historial  history.replaceState({nombre: "daniel"}, "", "?modificacion"); |

|  |
| --- |
| -FILEREADER  // Supongamos que el archivo elegido contiene:  // {"nombre" : "daniel"}  const archivo = document.querySelector("input");  const leerArchivo = ar => {      const reader = new FileReader();      // utilizamos readAsText pero se puede utilizar otros metodos de lectura      reader.readAsText(ar);      // el evento load de reader referencia la carga del archivo      reader.addEventListener("load", (e)=>{          // con e.currentTarget.result mostramos el contenido del archivo en formato string          console.log(JSON.parse(e.currentTarget.result));      })  }  // Escuchamos cambios realizados en el input con el evento change  archivo.addEventListener("change", (e)=>{      leerArchivo(archivo.files[0]);  });  // podriamos iterar si es que se selecciona mas de una archivo  // el elemento input deberia tener el atributo multiple habilitado  // archivo.files es un array con todos los archivos, solo es recorrerlo.  // Si qusieramos cargar un video  const cargarVideo = ar => {      const reader = new FileReader();      // Cargamos el video en un buffer de Array      reader.readAsArrayBuffer(ar);      // Si queremos mostrar el progreso de carga con el evento progress      reader.addEventListener("progress", e=>{          // Calculamos el progreso, e.loaded nos devuelve cuanto peso del archivo esta siendo cargado          // ar.size nos muestra el peso total del archivo          let carga = Math.round(e.loaded / ar.size \* 100);          console.log(carga);      });      // Escuchamos la carga con el evento load      reader.addEventListener("load", e=>{          // Capturamos el video en un objeto Blob          let video = new Blob([new Uint8Array(e.currentTarget.result)], {type: "video/mp4"});          // Creamos un URL del objeto Blob          let url = URL.createObjectURL(video);          // Creamos un elemento video de HTML          let elementoVideo = document.createElement("VIDEO");          // Seteamos la ruta en el elemento          elementoVideo.setAttribute("src", url);          // Agregamos el elemento video a algun contenedor de la pagina HTML          document.querySelector(".contenedor").appendChild(elementoVideo);          // Al finalizar la carga lo reproducimos, si no funciona puede ser con onplay()          elementoVideo.play();      });  } |

|  |
| --- |
| * INDEXEDDB * La base de datos se asocia a la URL de la pagina para que funcione bajo esa direccion   "strict mode";  // En esta varible IDBRequest alojamos la solucitud de apertura  // indexedDB.open("nameDatabase", 1) abrimos la db asignando el nombre y su version  // En caso de no encontrar la base de datos la creará  const idbRequest = indexedDB.open("nameDatabase", 1);  // Esto detecta si no se ha creado la db y necesita crearse de nuevo  // y lo hace, al hacerlo podemos crear los almacenes de objetos  idbRequest.addEventListener("upgradeneeded", ()=>{      console.log("Se actualizó la base de datos");      // Obtenenmos la base de datos      const db = idbRequest.result;      // Creamos el almacen de objetos para que se vaya llenando      // automaticamente con una clave autoincrementable      // esta puede ser otra con keypath(investigar)      db.createObjectStore("nombres",{          autoIncrement: true      });  });  // verifica si la apertura fue satisfactoria al abrir la base de datos  idbRequest.addEventListener("success", ()=>{      console.log("No hubo problemas al abrir la base de datos");  });  // Verfica errores en la apertura  idbRequest.addEventListener("error", ()=>{      console.log("Hubo un error al abrir la base de datos");  });  // Ejemplo de con que haríamos un CRUD  const agregarObjeto = objeto => {      const db = idbRequest.result;      // Necesitamos permisos de transaccion en el almacen de objetos      // introducimos nombre del almacen y el tipo de transaccion      const idbTransaction = db.transaction("nombres", "readwrite");      // Identificamos el almacen de objetos con los permisos de transaccion      const almacenObjetos = idbTransaction.objectStore("nombres");      // Agregamos un objeto      almacenObjetos.add(objeto);      // Evento para saber si se ha completado la transaccion      idbTransaction.addEventListener("complete", ()=>{          console.log("Objeto agregado correctamente.");      })  }  const leerObjetos = ()=> {      const db = idbRequest.result;      const idbTransaction = db.transaction("nombres", "readonly");      const almacenObjetos = idbTransaction.objectStore("nombres");      // Habilitamos un puntero en el almacen de objetos      const puntero = almacenObjetos.openCursor();      // Si no hay problemas con el puntero el evento success lo detecta      puntero.addEventListener("success", ()=>{          // hasta que el puntero sea nulo iterara gracias al metodo continue y          // el evento success          if (puntero.result) {              console.log(puntero.result.value);              // El metodo continue permite con la lectura de los              // demas elementos moviendo el cursor al siguiente              puntero.result.continue();          } else console.log("Se terminó de leer los datos.");      })  }  const modificarObjeto = (objeto, clave) => {      const db = idbRequest.result;      const idbTransaction = db.transaction("nombres", "readwrite");      const almacenObjetos = idbTransaction.objectStore("nombres");      // En caso que put no encuentre el objeto lo crea      almacenObjetos.put(objeto, clave);      idbTransaction.addEventListener("complete", ()=>{          console.log("Objeto modificado/agregado correctamente.");      })  }  const eliminarObjeto = clave => {      const db = idbRequest.result;      const idbTransaction = db.transaction("nombres", "readwrite");      const almacenObjetos = idbTransaction.objectStore("nombres");      almacenObjetos.delete(clave);      idbTransaction.addEventListener("complete", ()=>{          console.log("Objeto eliminado correctamente.");      })  } |

|  |
| --- |
| * MATCHMEDIA   "strict mode";  // matchMedia funciona como los media querys  // El valor de retorno es un booleano  const mq = matchMedia("(max-width: 500px)");  // Evento change en mq para escuchar los cambios de resoluciones  mq.addEventListener("change", ()=>{      console.log("Resolucion cambiada.");      // mq.matches nos devuelve un booleano depende el valor de      // captura en la variable mq      if (mq.matches) {          // Si la resolución es menor o igual a 500px      } else {          // Si la resolucion es mayor a 500px      }  }); |

|  |
| --- |
| * INTERSECTION OBSERVER   "strict mode";  // INTERSECTION OBSERVER nos detecta si un elemento es visible  // o no en el Viewport  // Seleccionamos diversos elementos para esta prueba  const elementos = document.querySelectorAll(".classNameElements");  // Funcion que se llamara a si misma en el observador  const verificacionVisibilidad = (entradas) => {      for (const entrada of entradas) {          if (entrada.isIntersecting) {              console.log("Se esta viendo el elemento.");          }      }  }  // Opciones del observador, modificación del área de referencia  const opciones = {      root: "viewport", // Si no se pone nada por defecto es viewport, pero es la referencia que se toma para observar cualquier elemento      rootMargin: "30px 30px 30px 30px", // margin del root en este caso viewport, modifica el área de referencia del obnservador      treshold: 0.5 // Cuanto debería ocupar el área de observación (viewport), 0=ni bien toque el borde, 1=hasta que se muestre por conmpleto  }  // Observador en este caso esta observando diversos elementos  const observador = new IntersectionObserver(verificacionVisibilidad, opciones);  for (const elemento of elementos) {     observador.observe(elemento);  } |

|  |
| --- |
| -NOTIFICATION  "strict mode";  // Veficamos si las notificaciones son admitidas por el navegador  if (!('Notification' in window)) {  console.log("Las notificaciones no estan disponiles en tu navegador.");  }  // Preguntamos si queremos activar las notificaciones al usuario  Notification.requestPermission(()=>{  if (Notification.permission == "granted") {  console.log("Las notificaciones han sido activadas.");  let nota = new Notification("Notificandonte un hola :D");  console.log(nota);  }  if (Notification.permission == "denied") {  console.log("Las notificaciones han sido denegadas.");  }  if (Notification.permission == "default") {  console.log("Las notificaciones estan configuradas por defecto.");  }  }) |

|  |
| --- |
| - WEB WORKERS  "strict mode";  // Tipo de Web Workers: Dedicated Worker, Shared Worker, Service Worker y Abstract Worker  // Este es un Dedicated Worker, y en el constructor se puede introducir un nombre y un objeto de opciones  // El objeto de opciones puede llevar valores: type, credentials y name  // Método postMessage() para enviar mensajes entre el Web Worker y el Script Principal  // Evento onMessage para recibir mensajes entre el Web Worker y el Script Principal  // Metodo terminate() Finaliza la ejecución del web Worker, usado en el Script principal  // Política de origen cruzado Same-Origin, esto basicamente significa que los web workers responden a un solo entorno de protocolo/host:puerto (el mismo dominio o origen)  // Básicamente un Web Worker funciona para correr operaciones en segundo plano  // Como el de cargar imagenes, datos, elementos multimedia, etc.  // Un Web Worker no es lo mismo que un Script principal porque no tiene conexión directa con window  // es decir sus operaciones se limitan a operaciones que pueden interrumpir el hilo principal  // En el parámetro del worker tambien puede ir una URL que especifique el archivo del worker  const worker = new Worker("archivoWorker.js");  // Mostramos los detalles del objeto  console.log(worker);  // La escucha de un evento en el worker para escuchar mensajes en el Script Principal  worker.addEventListener("message", e=>{      // mostramos el mensaje o el objeto enviado      console.log(e.data);      // Cerramos el worker      worker.terminate();  });  worker.postMessage("Mensaje al Worker.");  /////////////////////////////////////////////////////////////////  // En el archivo archivoWorker.js:  // Cuando estamos en un archivo Worker no es necesario nombrarlo para mostrar los metodos  addEventListener("message", e=>{      console.log(e.data);      postMessage("Mensaje al Script Principal.");  }); |

|  |
| --- |
| -OBJETO NAVIGATOR  "strict mode";  // OBJETO NAVIGATOR  console.log("AppCodeName: ",navigator.appCodeName);  console.log("AppName: ",navigator.appName);  console.log("AppVersion: ",navigator.appVersion);  console.log("Connection: ",navigator.connection);  console.log("Geolocation: ",navigator.geolocation);  console.log("HardwareConcurrency: ",navigator.hardwareConcurrency);  console.log("Language: ",navigator.language);  console.log("Languages: ",navigator.languages);  // Archivos permitidos, informacion parecida al de la cabecera  console.log("MimeTypes: ",navigator.mimeTypes);  console.log("OnLine: ",navigator.onLine);  // informacion del navegador  console.log("UserAgent: ",navigator.userAgent);  // verifica si las cookies estan activadas  console.log("CookieEnabled: ",navigator.cookieEnabled);  console.log("Permissions: ",navigator.permissions);  // Sistema operativo  console.log("Plataform: ",navigator.platform);  // Plugins en un array  console.log("Plugins: ",navigator.plugins);  // Gecko  console.log("Product: ",navigator.product);  // Es uun Wworker que trabaja en paraleleo distinto al dedicated worker que funcionan para un sitio y no comparten informacion  // a diferencia del Service Worker  console.log("ServiceWorker: ",navigator.serviceWorker);  // MÉTODOS  // Para el siguiente método se recomienda usar Media String, Media Recorder o Media Device  navigator.getUserMedia(); // Permite el uso del audio y video de nuestro dispositivo  navigator.registerContentHandler(); // Permite a los web sites registrarse como posibles controladores de un tipo MIME determinado  navigator.registerProtocolHandler(); // Permite a los sitios webs registrarse como posibles controladores de un protocolo determinado  navigator.requestMediaKeySystemAccess(); // Devuelve un objeto Promise para un objeto MediaKeySystemAccess  navigator.sendBeacon(); // Usado para transferir de forma asincrona conjuntos pequeños de datos HTTP del agente usuario al servidor  navigator.javaEnabled(); // Devuelve un valor boolean que indica si el navegador tiene o no habilitado java  navigator.vibrate(); // Causa vibración en el dispositivo que lo soporte |

|  |
| --- |
| -MEMOIZER  "strict mode";  // MEMOIZER  // Es una tecnica de programación para guardar resultados de operaciones de funciones que son iguales para que su consulta siguiente no demore el tiempo de la operación de otra vez.  // FUNCION DE ESTRES  const funcionDeLargaDuracion = num => {      const a = 20;      const b = 12;      let c = 0;      for (let i = num - 1; i >= 0; i--) {          for (let j = i - 1; j >= 0; j--) {              c+= a\*b;          };      };      return c;  };  //  RESULTADOS SIN MEMOIZACION  const date1 = new Date();  funcionDeLargaDuracion(90000);  console.log(new Date() - date1);  const date2 = new Date();  funcionDeLargaDuracion(90000);  console.log(new Date() - date2);  const date3 = new Date();  funcionDeLargaDuracion(90000);  console.log(new Date() - date3);  const date4 = new Date();  funcionDeLargaDuracion(90000);  console.log(new Date() - date4);  // IMPLEMENTACION DE UNA FUNCION MEMOIZER  var cache = []; // Array que sirve de memoria  // Funcion memoizer que recibe como parametro una funcion y retorna otra funcion la que se debe introducir un parametro, esto se debe porque la funcion de estres tambien recibe un parametro entonces ese valor resultante se almacenera en el arreglo de cache internamente, el indice es el mismo que el valor de de entrada de la funcion de estres, para identificarla rapidamente.  const memoizer = func =>{      return e => {          const index = e.toString();          if (cache[index] == undefined) {              cache[index] = func(e);          };          return cache[index];      };  };  //  RESULTADOS CON MEMOIZACION  // Esto nos devuelve una funcion que internamente es la de estres, al que se llama memo ahora y ahora podemos introducir el parametro que en la funcion memoizer es 'e'  const memo = memoizer(funcionDeLargaDuracion);  const date5 = new Date();  memo(90000);  console.log(new Date() - date5);  const date6 = new Date();  memo(90000);  console.log(new Date() - date6);  const date7 = new Date();  memo(90000);  console.log(new Date() - date7);  const date8 = new Date();  memo(90000);  console.log(new Date() - date8); |

|  |
| --- |
| -CACHE  "strict mode";  // Funciona casi similar como IndexedDB  // Al abrirlo nos devuelve una promesa que tiene varios metodos, parecidos a IndexedDB, como: add, addAll, delete, keys, match, matchAll, put.  caches.open("nombre-de-cache").then(cache => {      console.log(cache);      // cache.add(request), toma una URL, la recupera y agrega el objeto de respuesta resultante a la cache dada. Esto es funcionalmente equivalente a llamar fetch() y luego usar put() para agregar los resultados a la cache.      cache.add("./archivo.js"); // agregamos un archivo al cache      // cache.addAll(request), toma una matriz de URL, las recupera y agrega los objetos de respuesta resultante a la cache dada.      cache.add(["archivo1.html", "archivo2.css", "archivo3.js"]);      // cache.match(request, options), Devuelve un Promise que se resuelve(resolve) con la respuesta asociada con la primera solicitud coincidente en el objeto almacenado      cache.match("archivoRepetido.html").then(res => {          console.log(res);      });      //Nos devuelve un arreglo de todas las respuestas coincidentes.      cache.matchAll("archivoRepetido.html").then(res => {          console.log(res);      }); // nos devuelve todos los archivos llamados archivoRepetido.html.      // cache.put(request, response), toma tanto una solicitud como su respuesta y la agrega a la caché dada.      // Este metodo put sirve para modificaciones pero en si es mas aconsejable que se utilice add() para modificaciones.      fetch("archivo.html").then(res=>{          cache.put("archivo.html", res)      })      // cache.delete(request, options), encuentra la entrada del objeto cuya clave es la solicitud, devolviendo un Promise que resuelve true si el objeto se encuentra y se elimina una entrada coincidente. Si no se encuentra ninguna entrrada, la promesa se resuelve en false.      cache.delete("archivo.html");      // cache.keys(), devuelve un Promise que se resuelve en una matriz de keys de los objetos almacenados. Trae toda la informacion.      cache.keys().then(res=>{          console.log(res);      })  }); |

|  |
| --- |
| -SERVICE WORKERS  "strict mode";  // Script Principal, navegador  // Verificamos si el navegador admite Service Worker  if (navigator.serviceWorker) {      // Registramos el archivo que nos servirá de Service Worker      navigator.serviceWorker.register("nombreServiceWorker.js");  };  // Para enviarle un mensaje al Service Worker necesitamos saber si esta listo con el metodo ready el cual devuelve una promesa, el cual tiene una propiedad llamada active que contiene el metodo postMessage.  navigator.serviceWorker.ready.then(      res => res.active.postMessage("Saludando al Service Worker.")  );  // Para que el navegador escuche los mensajes del Service Worker  navigator.serviceWorker.addEventListener("message", e=>{      console.log("Mensaje recibido del Service Worker:");      console.log(e.data);  });  ////////////////////////////////////////////////////////  // Service Worker (nombreServiceWorker.js)  // Para señalar el Service Worker se utiliza self algo como el this  self.addEventListener("install", e=>{      console.log("Service Worker instalado.");  });  self.addEventListener("activate", ()=>{      console.log("Service Worker está activo.");  });  self.addEventListener("fetch", ()=>{      console.log("Service Worker interceptando peticion.");  });  self.addEventListener("message", e=>{      console.log("Mensaje recibido del navegador:");      console.log(e.data);      // Para enviar un mensaje desde el Service Worker, accedemos al metodo source      e.source.postMessage("Saludando al navegador.")  }); |

|  |
| --- |
| -EJEMPLO DE GUARDAR UNA PAGINA EN EL CACHE CON SERVICE WORKER |