Introducción

Lenguaje interpretado. No necesita ser compilado.

Se lee línea por línea, script por script.

El interprete es el navegador.

Hay algunos frameworks de Js, como angular o react que si deben ser compilados.

Inicialmente se creó para ser utilizada en el navegador. Últimamente adquirió nuevas funcionalidades, con node.js y NPM (Node Package Manager).

ECMAScript es una entidad reguladora, que estandariza JavaScript para los distintos navegadores. ES5 y ES6 son los mas usados. ES6 prima, que es el mas nuevo (2015).

EL IDE mas utilizado es Visual Studio Code.

Plugins:

Live Server.

JavaScript (ES6) Code Snipets. Atajos para la escritura.

Prettier. Formateador de código. Shift+Alt+F.

Bracket pair colorizer. Colorea.

Indent-Rainbow. Colorea las indentaciones.

Quokka.Js. Marcaría “errores”.

Primer Proyecto con NodeJs

Alt+Windows entras… + CMD entras al símbolo de sistema. Ahí con Mkdir creas carpetas.

Una vez en la carpeta indicada, pones code . para abrir el VSCode.

Inicializa usando la consola y poniendo “nmp init”, esto crea un archivo: package. Json El cual va a tener toda la info del proyecto.

Te hace unas preguntas. Test Command y entry point en ppio se las deja en blanco.

{

  "name": "hola-mundo",

  "version": "1.0.0",

  "description": "este es mi primer proyecto en Open Bootcamp",

  "main": "index.js",

  "scripts": {

    "test": "echo \"Error: no test specified\" && exit 1"

  },

  "keywords": [

    "holamundo",

    "first",

    "project"

  ],

  "author": "Pablo Krojzl",

  "license": "ISC"

}

Se pone eso en la consola para ejecutar el código, desde node:

PS C:\Users\Pcablo\Curso-Js\PrimerProyecto\_NodeJs> node holaMundo.js



Scripts:

 "scripts": {

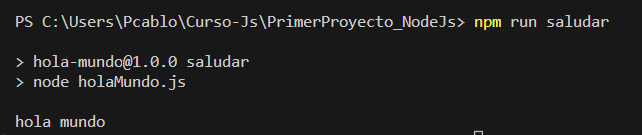
    "saludar": "node holaMundo.js",

    "test": "echo \"Error: no test specified\" && exit 1"

  },

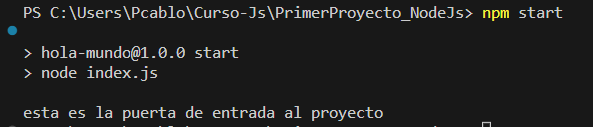
Agregamos el script saludar.

Y por consola lo llamamos:



Esto es ilustrativo y no se usa. Lo que si se hace es llamar start, que linkee con index.js. No hay necesidad de poner run, solo npm start.

"start": "node index.js",



Tipado inferido. Se definen de forma genérica las variables. Ventajas: Da mas libertad. Desventajas:

2 grupos de tipos: primitivos y objetos.

Primitivos: number, string, boolean, null y undefined.

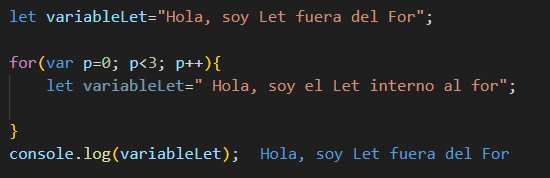
3 formas de definer “variables”.

Var, let o const.

Const es la que mas se utiliza en angular y react. Son constantes: no cambian.

Actualmente se usa mas el let que el var.

El var es mas abarcativo que el let: por ejemplo si se declara una variable con un let, dentro de un for, va a estar declarada también para afuera del for.



Si variableLet dentro del for fuese var, daría error.

Operador typeof:



Al final de cada línea pone ; para delimitar el final de una línea. En Js, a día de hoy no es algo obligatorio.

El . se utiliza para acceder a los atributos internos de un objeto.

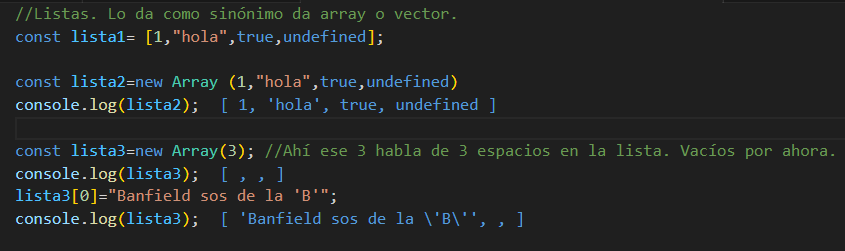
{} Se utilizan para objetos, funciones y estructuras de control.

[] Se utiliza para listas o arreglos. Para definirlos o para acceder a algún valor determinado.



Paréntesis () se utiliza para funciones.

Listas



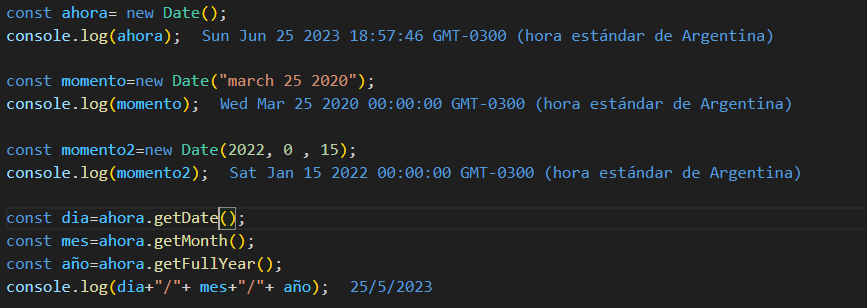
Objetos

Por mas que se lo defina como constante, sus atributos internos pueden cambiar.



Fechas

Dice que es feo, aunque hay librerías que lo facilitan: Moment.js



If else y switch

Idéntico, o casi, a C#.

Estructuras de control.

El código va allí donde ciertas condiciones le indican.

Bifurcaciones. Vas por un camino u otro.

Comparaciones

== Igualdad débil.

=== Igualdad fuerte.

!= y !== < > >= <=

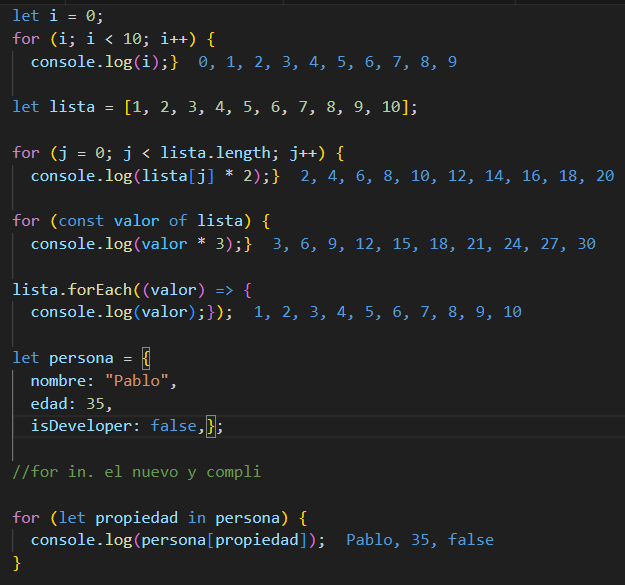
For y while

For

ForOf. Igual al foreach de C#

ForEach. Expresión lambda.

ForIn.

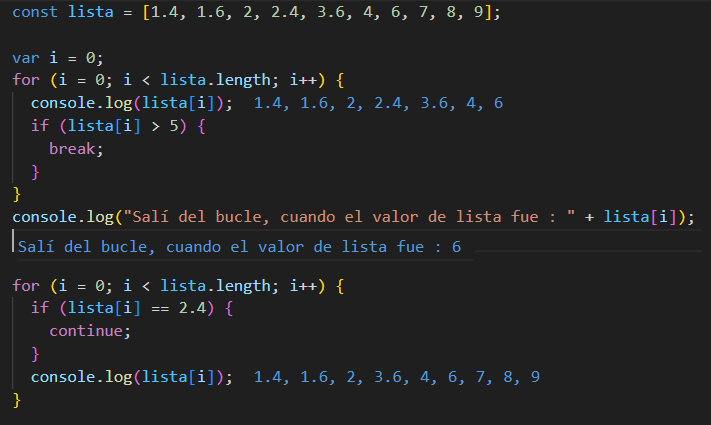


Casi siempre se usa el while y no el do while.

Se usan igual que en C#.

Continue y break

Son cosas “muy específicas”.



Con el break, cuando se da cierta condición sale del bucle.

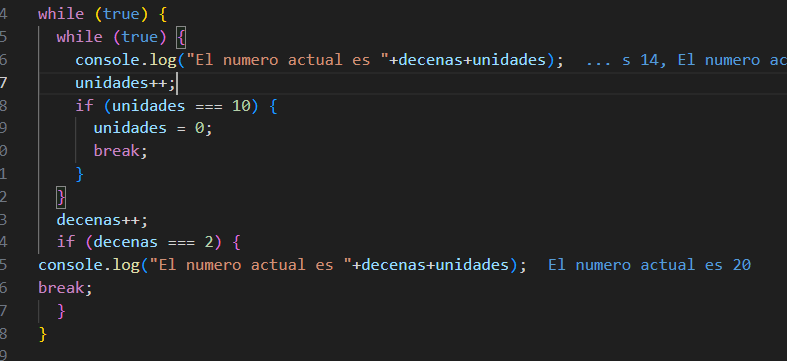
Con el continue, cuando se da cierta condición, pasa al siguiente índice del bucle, ignorando el código siguiente; vuelve arriba.

Afirma que hay que utilizarlo en casos muy excepcionales. Dice que si está todo bien hecho, no hace falta usarlo. Sería una mala práctica.

Etiquetas en los bucles

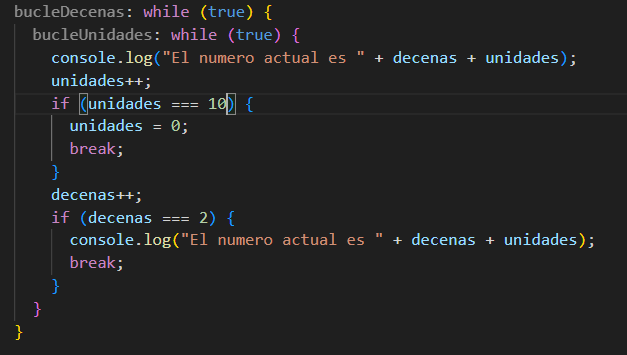
Labels permiten controlar bucles.

Si se hace uso de las labels, dentro de un mismo bucle, se puede afectar a bucles externos.



Esta es la forma sin los labels. Es mas confusa y se debe anidar un while dentro de otro.

Así quedaría:



Reitera que esta tampoco es una de las mejores prácticas. Pero al igual que lo del continue/Break, te puede sacar de apuros.

let lista=[1,2,3,4,5,6,7,8,9,10]

let numero=1;

for(i=1; i<lista[length];i++)

{

numero=numero\*lista[i];

}

console.log(numero);

Strings

Es buena práctica manejarse siempre, o bien con comillas simples o con comillas dobles, pero son lo mismo.

let stresp="Eres un \"gallina\" McFly";

console.log(stresp);

let stresp2='Eres un "papanatas" Mcfly';

console.log(stresp2);

let stresp3="Eres un 'gallina' McFly";

console.log(stresp3);

Comillas invertidas o backticks

ALT + 96 o ALT gr + }.

Posibilita dentro de las mismas hacer esto:

let strloco= `Y por eso dijo: ${stresp3}`;

console.log(strloco);

A la vez, posibilitan escribir mas de una línea.

A la vez, facilitan la escritura de código HTML.

let plantilla = `

<html>

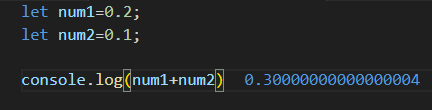
    <h1> Título </h1>

    <p> este es un párrafo</p>

</html>

`;

Números



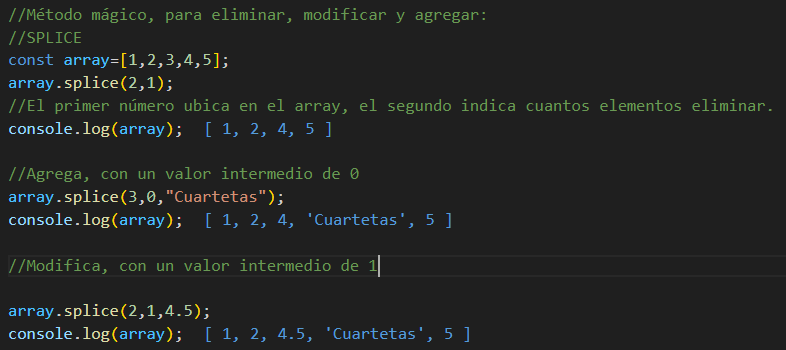
Afirma que es común que las cuentas fallen. Llama a esto pequeño truco:

console.log(Math.round((num1+num2)\*100)/100);

\*\* para elevar al cuadrado

Listas

Método mágico SPLICE:



Obtener lista a partir de otra.

//Obtener lista a partir de otra con slice

//Se le indica inicio y final

let fragmentoLista= lista4.slice(0,5);

console.log(fragmentoLista);

Iteraciones

onst array = [1, 2, 3, false, "hola", true, "chau", false, true];

//Forma tradicional. Dice que quedó medio vieja.

for (let index = 0; index < array.length; index++) {

  console.log(array[index]);

}

//Metodo forOf, mi ForEach

for (const iterator of array) {

  console.log(array[iterator]);

}

//Reivindica el método foreach este, propio de Js.

array.forEach((valor) => {

  console.log(valor);

});

Búsquedas

const variable = array.find((valor) => {

  if (valor == true) {

    return true;

  }

})

Búsqueda dentro de objetos:

const listaObjetos = [

  { nombre: "Pablo", edad: 35 },

  { nombre: "Lucía", edad: 33 },

  { nombre: "Paco", edad: 27 },

  { nombre: "Arantxa", edad: 11 },

];

const objeto = listaObjetos.find(o => {

  if (o.nombre === "Pablo") {

    return true;

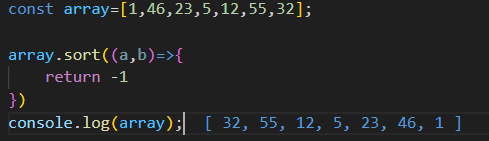
  }

});

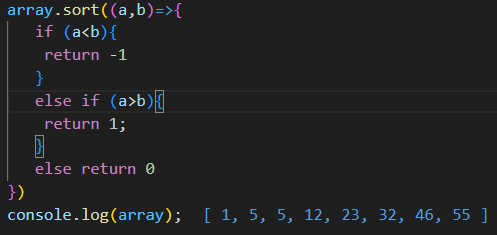
console.log(objeto.edad);

Sort

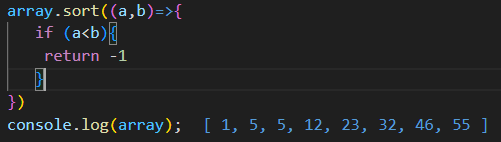
Con el return -1 invierte la lista.



Esa es la forma posta de ordenar la lista.

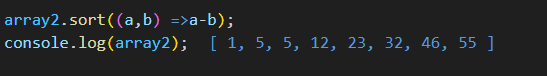


Aunque esta, mas cortita, funciona.



Ese número que se retorna no importa cuál sea, lo que importa es el signo que tiene.

Otra forma, que solo funcionaría con arrays numéricos.



Ahí es al revés.

array2.sort((a,b)=>b-a);

console.log(array2);

Ahí ordena objetos, según la edad:

const listaObjetos = [

  { nombre: "Pablo", edad: 35 },

  { nombre: "Lucía", edad: 33 },

  { nombre: "Paco", edad: 27 },

  { nombre: "Arantxa", edad: 11 },

];

listaObjetos.sort((a, b) => {

  if (a.edad < b.edad) return -1;

  else if (a.edad > b.edad) return 1;

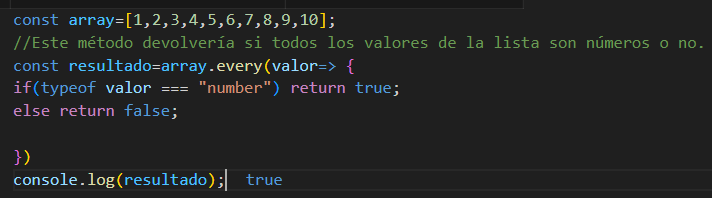
  else return 0;

});

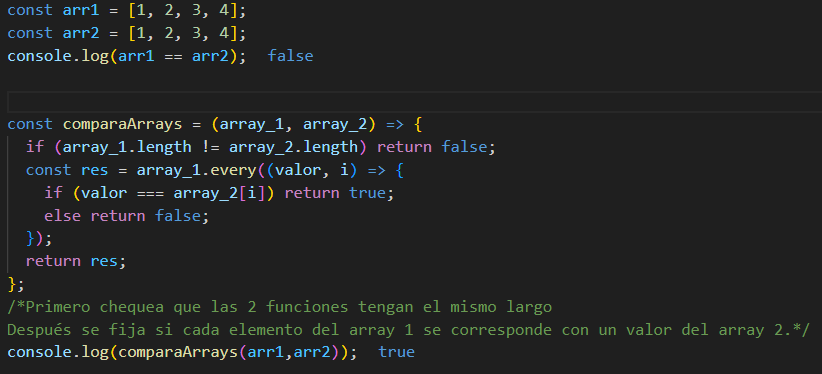
Forma cortita:

listaObjetos.sort((a, b) =>a.edad-b.edad);

Every

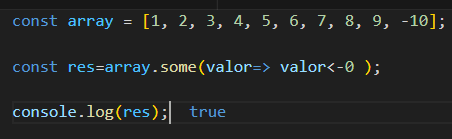


Comparar listas:



Some

Visibiliza si algún elemento cumple alguna condición. En eso difiere del every.



Acá chequea si algún elemento es menor que 0.

Obtener lista a partir de objeto iterable.

Dice que no se utiliza mucho.

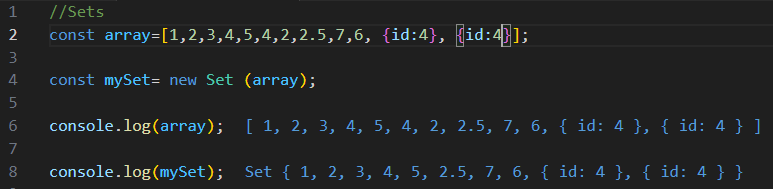


Sets (conjuntos)

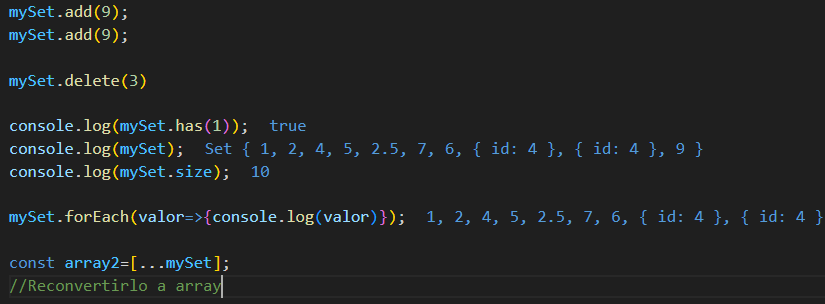
Son una suerte de lista.

Es un conjunto no ordenado de elementos únicos. No permite almacenar elementos repetidos.

Como se ve, pueden meterse objetos que parezcan iguales, porque no serían totalmente iguales.

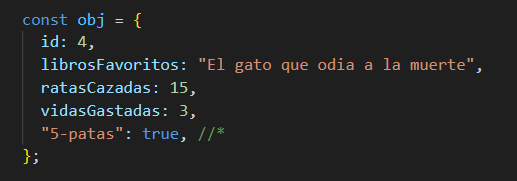


.add .delete .clear .has (para ver si tiene o no algún valor). .size (propiedad) .forEach para iterar

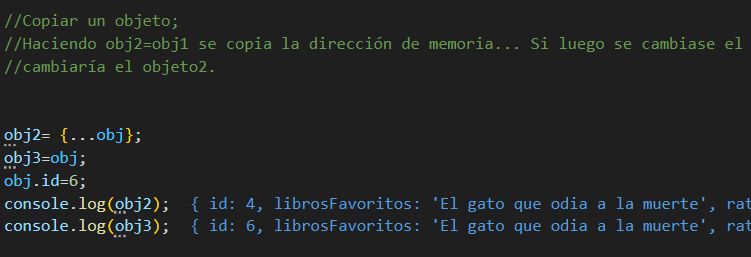


Objetos

\*ver cómo se ponen nombre “un poco raros”.



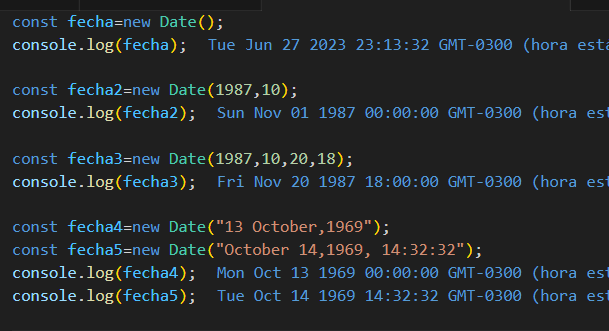
Clonar un objeto



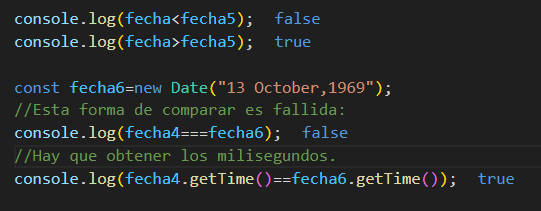
Ordenar objetos según algún valor.



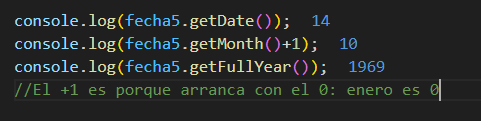
Fechas

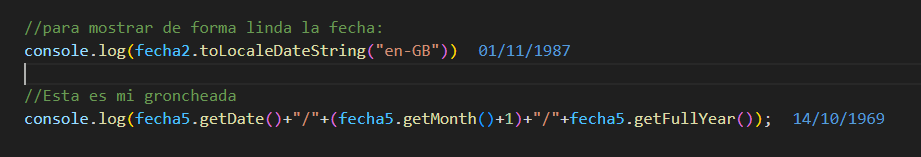


Comparar fechas



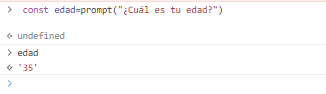
.getTime para obtener los milisegundos de las fechas.





Consola

Abre el navegador y en la barra de búsquedas pone about:blank. Esto abre hoja en blanco. Una vez ahí pone F12 (herramientas de desarrollador).

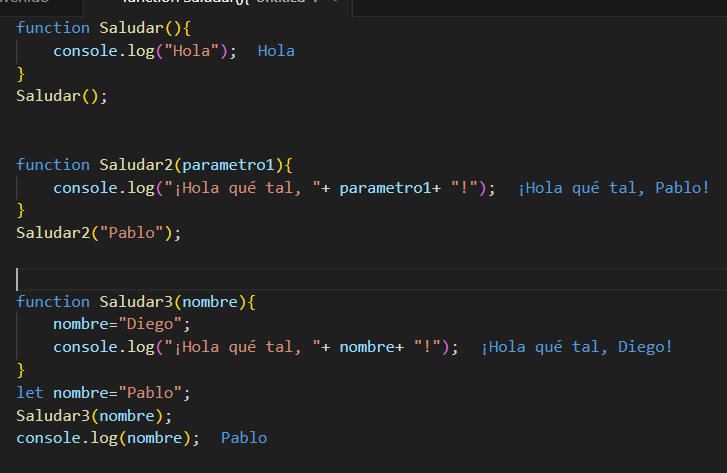


Con el prompt se dispara la pregunta al usuario.

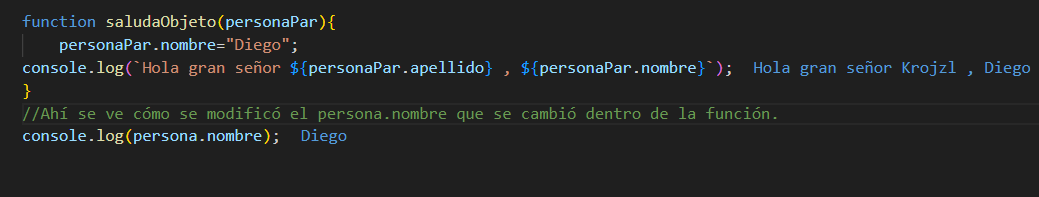
Luego con el const edad= se guarda la respuesta del mismo.

Devolverá un string. Se lo puede parsear o castear al string para hacerlo numérico.

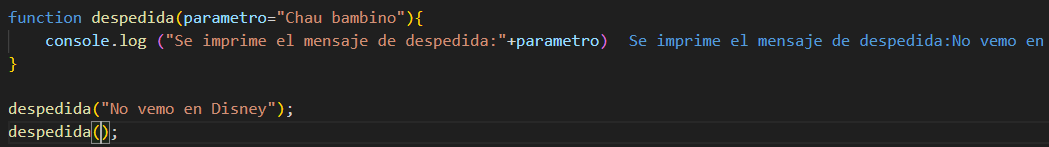
Funciones



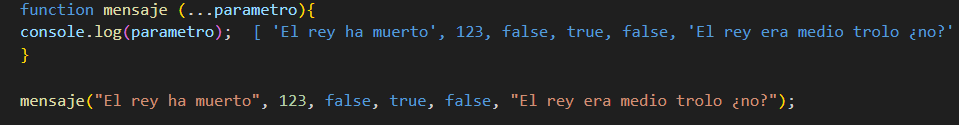
Parámetros se pueden pasar por valor o por referencia. Los objetos por default se pasan por referencia.



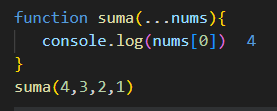
Parámetro por defecto



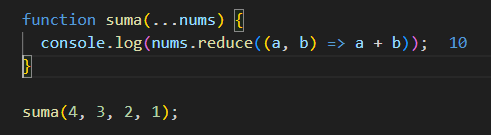
Parámetros con propagación



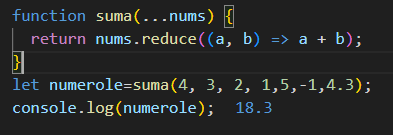
Acá con los números lo que se ve es que lo que se recibe por propagación es un array en este caso en el índice 0 de nums está el 4.



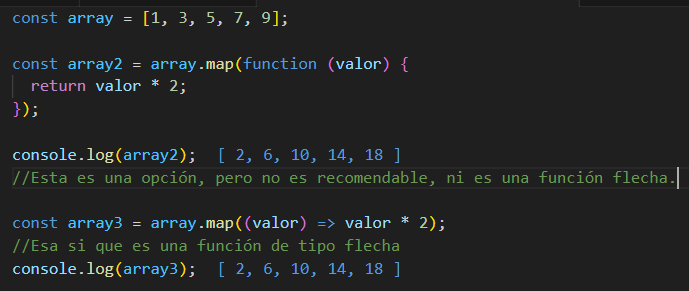
Y ahí se hace la suma



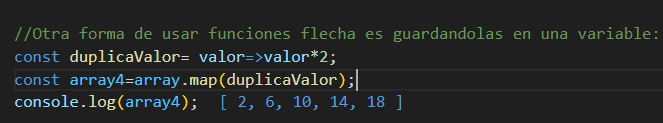
Acá con return:



Funciones de tipo flecha



En la segunda forma, se le asigna un nombre a la variable (duplicaValor), por lo que no es anónima, en el caso de arriba si que lo es.



En el orden inverso no se podría, primer hay que inicializar la variable.



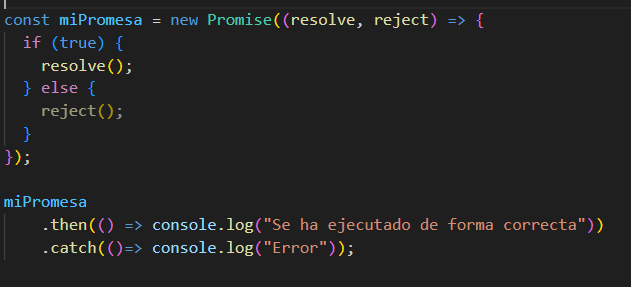
Funciones asíncronas y promesas

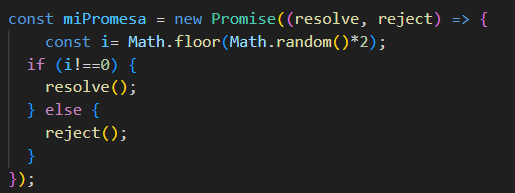
Lo menciona como algo avanzado en Js.

Asíncronas. P ej, al llamar a una BBDD. Puede demorar el llamado algo de tiempo.

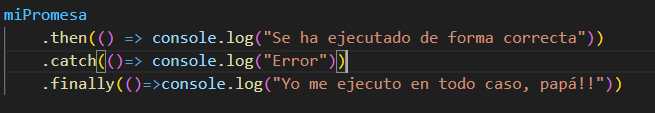
Acá hay una promesa. Si es verdadero se ejecuta resolve, si no, se rechaza la promesa. Con el .then y el .catch se va a actuar después, en función de que sea resolve o reject.

En este caso, siempre va a ser true la promesa, por eso, luego la modifica, generando un número aleatorio 0 o 1 para que el resultado cambie y también pueda ser false.





Luego agrega el bloque finally, que se ejecuta siempre, sea false o true la funcion.



Funciones generadoras

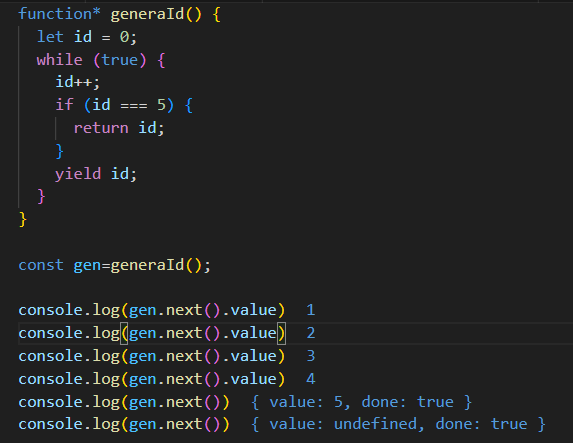
Generan, p ej, un Id incremental.

El asterisco\* hace que cada vez que se la llama te devuelve un número. Se complementa con el yield: se queda esperando en esa parte del código, a la espera de volver a ser llamada. En este caso lo que se queda esperando es el Id, que ya no será 0.

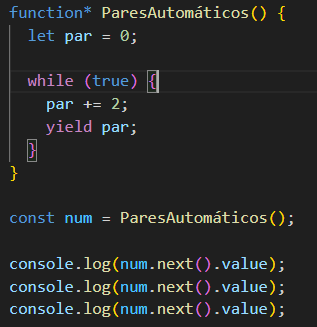
La guarda en una constante.

Hay objetos de tipo generator.

Con .next genera el nuevo valor. Devuelve un objeto, para obtener su valor se le pone .value



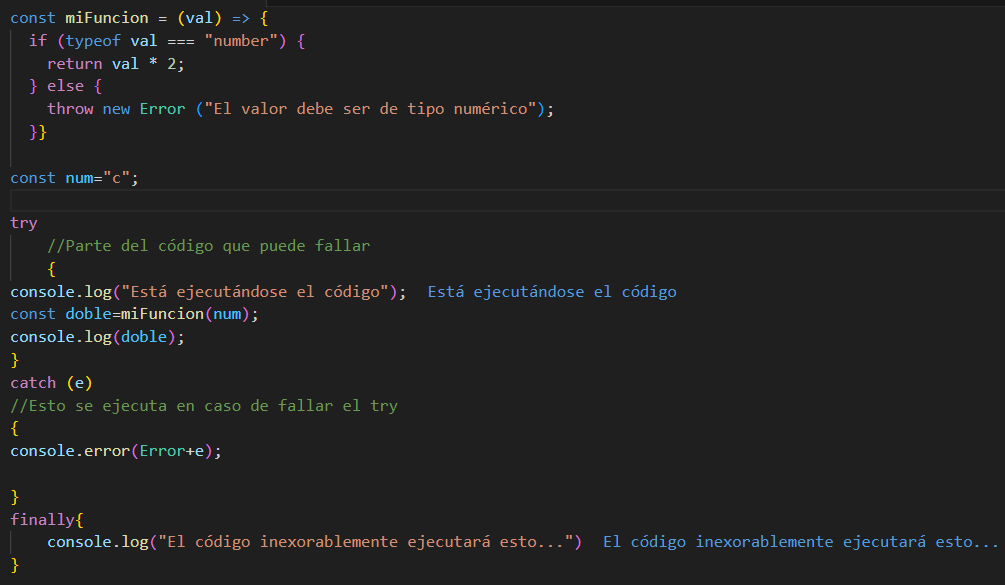
Esta la hice yo, se ve como el yield funciona como retorno.



Manejo de errores

Hay un “else implícito”. Si no es numérico el valor pasado =>throw new error: convierte al resultado en error y se detendría el programa.

El programa no se detiene por la operación de los try catch.



Logs

Se crea vía node. Con Terminal: npm init -y

Npm Winston es una librería que permite manejar los logs: npm install Winston dentro de la carpeta del proyecto

Descarga allí este Código y lo pega en el index.js

const winston = require('winston');

const logger = winston.createLogger({

level: 'info',

format: winston.format.json(),

defaultMeta: { service: 'user-service' },

transports: [

//

// - Write all logs with importance level of `error` or less to `error.log`

// - Write all logs with importance level of `info` or less to `combined.log`

//

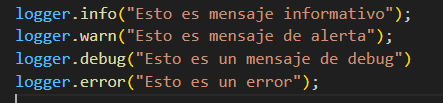
new winston.transports.File({ filename: 'error.log', level: 'error' }),

new winston.transports.File({ filename: 'combined.log' }),

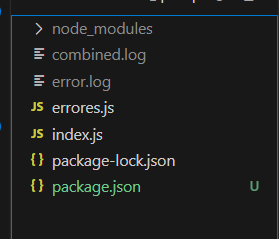
],

});

Luego, se pueden escribir distintos tipos de msjs, como los que siguen, usando logger. Tal cosa



Y al tocar npm start se crean esos archivos de ahí.

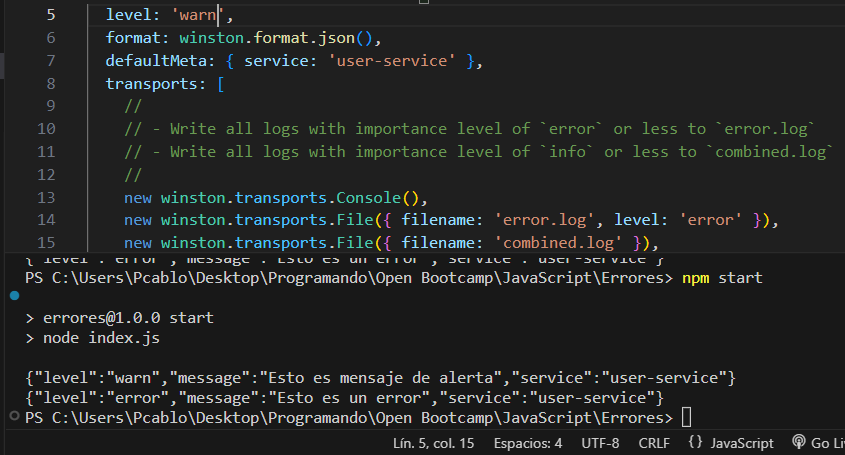




Esos son los niveles de logger. Si se quiere ampliar el detalle se puede modificar en la parte del código importada debug o silly allí donde por defecto dice info.

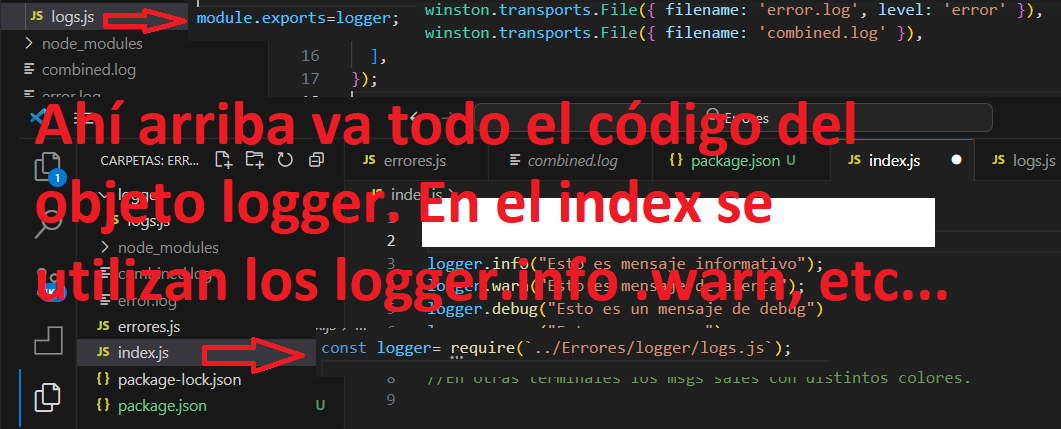
new winston.transports.Console(),

Agrega esa línea en la parte de los transports para que además, los logs aparezcan por consola.



Aparecen 2 msjs de los 4 que escribí, porque en level establecí “warn”.

Dice que es buena práctica poner todo el objeto logger en una carpeta aparte. Luego hacer module.exports=logger; para que se pueda utilizar en todo el proyecto.



Módulos

Parte del código que se puede utilizar en los archivos de un proyecto.

Ejecutar sin quokka ni json

Basta con escribir por terminal node index.js

Todos los archivos tienen por defecto un objeto llamado module.

2 formas para exportar un módulo

Hasta el 2016 solo se podía esta:

1. Common Js – require

Archivo cargado de funciones matemáticas:

1. function FactorialFunc(num) {
2. let factorial = 1;
3. for (let i = 1; i <= num; i++) {
4. factorial \*= i;
5. }
6. return factorial;
7. }
8. function Suma(num1, num2) {
9. return num1 + num2;
10. }
11. module.exports={
12. FactorialFunc,
13. Suma
14. }

Archivo que utiliza las funciones matemáticas:

const moduloMath=require("./Modulos/matematicas.js");

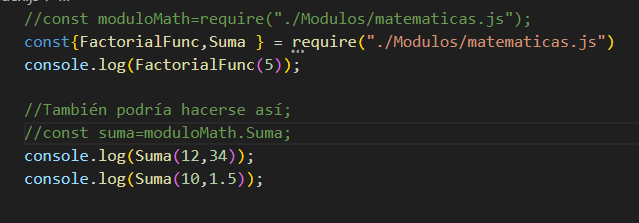
console.log(moduloMath.FactorialFunc(5));

//También podría hacerse así;

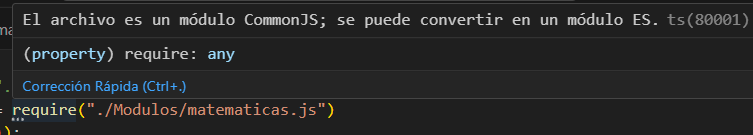
const suma=moduloMath.Suma;

console.log(suma(12,34));

1. Import ES6 - import



Es muy similar. El código queda un poquito mas pelado.

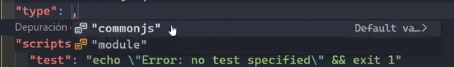


Señala los 3 puntitos del require.

Esta que sigue es la forma postiña…

Crea otra carpeta donde inicializa json.

En ese archivo crea un atributo type, y ahí se ve esto:



Selecciona la opción module.

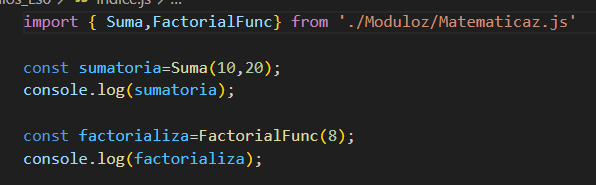
Luego en el archivo con los módulos a exportar, se borra el module.exports y se les pone a las funciones que se van a exportar, el export:

 export function Suma(num1, num2) {

    return num1 + num2;

  }

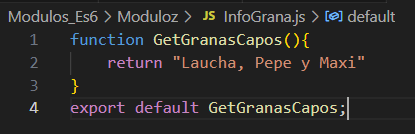
Así se lo importa:

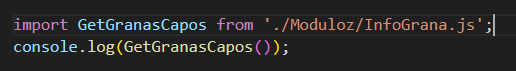


Con ‘Import \* as ModuloMatematicas (nombre de fantasí) from … ‘ se importa todo lo que haya exportado en el archivo.

Export default

Aclara que cada archivo solo puede tener un export default.





Librerías

Axios es una librería http que permite hacer llamadas, a diferentes servidores, etc.

Usar esto en el terminal Npm install axios

Lo usa para llamar al servicio externo PokeApi

<https://pokeapi.co/api/v2/> pokemon/ditto

Eso de arriba lo usará luego. Se obtiene en la pagina de PokeApi

import axios from "axios";

// Make a request for a user with a given ID

axios.get('https://pokeapi.co/api/v2/pokemon/ditto')

  .then(function (response) {

    // handle success

    console.log(response);

  })

  .catch(function (error) {

    // handle error

    console.log(error);

  })

  .finally(function () {

    // always executed

  });

El resto del código que se ve es copypasteado de la página de axios. Solo se modifió el console.log(response.data).

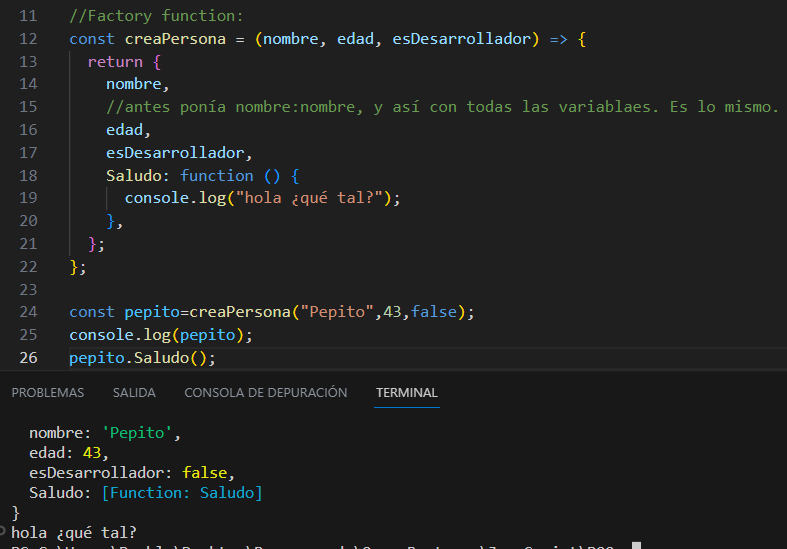
JQuery permite interactuar con HTML. También puede hacerse de forma nativa, pero de esta forma es mas sencilla.

React. Librería piola.

Chalk. La uso en ejercicio. Cambia los coloretes en consola.

POO

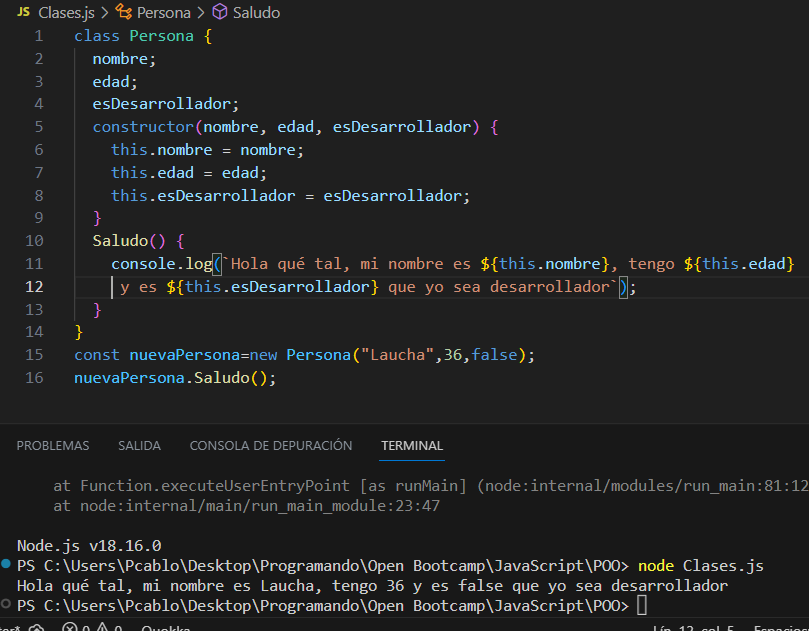
Crea una función function factory, para crear distintos objetos con una misma forma.



Esta es una forma, pero la posta, los trolos de JavaScript se la chorearon de C#, ¡¡papá!!

Clases

Muy parecido, con su constructor y sus .this.



InstanceOf para ver si tal objeto es instancia de tal clase o no. Devuelve true o false.

console.log(nuevaPersona instanceof Persona);

Modificadores de acceso

Private => #

Protected=> \_

class Persona {

    #nombre;

    #edad;

    \_esDesarrollador;

  constructor(nombre, edad, esDesarrollador) {

    this.#nombre = nombre;

    this.#edad = edad;

    this.\_esDesarrollador = esDesarrollador;

  }

  Saludo() {

    console.log(`Hola qué tal, mi nombre es ${this.#nombre}, tengo ${this.#edad}

     y es ${this.\_esDesarrollador} que yo sea desarrollador`);

  }

  ObtenNombre(){

    return this.#nombre;

  }

}

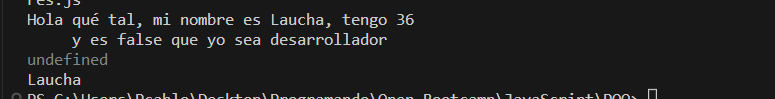
const nuevaPersona=new Persona("Laucha",36,false);

nuevaPersona.Saludo();

console.log(nuevaPersona.edad);

console.log(nuevaPersona.ObtenNombre());

En el undefined se ve que se intentó acceder a una propiedad privada desde afuera, sin utilizar un método público que te la entregue.



Getters – Setters Idem C#.

Herencia (inheritance)

Se utiliza extends y cambia mucho la forma en que el constructor hijo manda los parámetros al constructor padre (super)).

class Persona {

  constructor(nombre, edad) {

    this.nombre = nombre;

    this.edad = edad;

  }

  Saludo() {

    console.log(`Hola qué tal, mi nombre es ${this.nombre}, tengo ${this.edad}

     `);

  }

}

class Desarrollador extends Persona {

  constructor(nombre, edad, lenguaje) {

    super(nombre, edad);

    this.lenguaje = lenguaje;

  }

  //Así se hace el override del método padre Saludo()... De esta forma se genera polimorfismo

  Saludo() {

    super.Saludo()

    console.log(" y soy desarrollador de "+this.lenguaje)

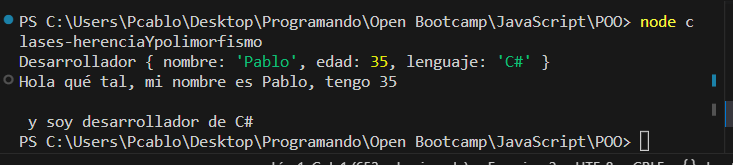
  }

}

const nuevoDev = new Desarrollador("Pablo", 35, "C#");

console.log(nuevoDev);

nuevoDev.Saludo();



¡¡Interfaces!!

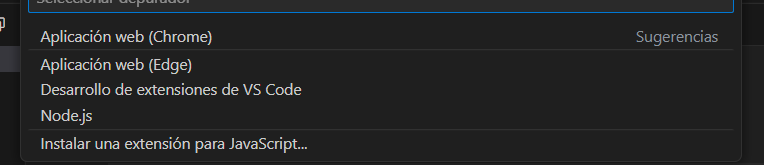
No existen (a priori, aclara).

En TypeScript si existen.

Debug

Ir ejecutando el código de forma controlada para ver si funciona como se pretende.

Punto rojo===Break Points o puntos de ruptura.



Ese panel se abre con F5.

Utiliza Node.js

Linting y EsLint

Linting: metodología para mantener el código homogéneo.

Manejarse con comillas simples a veces, y dobles otras. Poner o no el ; no están ni bien ni mal. Pero no hay que ser inconsistente.

Instalar extensión eslint.

//Archivo de configuración de reglas de ESLint

Crea un archivo .json.

Va al sitio de eslint al apartado configuration files y copia ese pedazo de código:

{

"env": {

"browser": true

},

"rules": {

// Override our default settings just for this directory

"eqeqeq": "warn",

"strict": "off"

}

}

Recomienda otra extensión: error lens. Potente. Te marca los errores línea por línea.

Aquellas cosas que se consideran como reglas en el json, en caso de no cumplirse, aparecerán como errores.

forma posta de configurar ESLint

ESLint=> GetStarted. Lo primero es instalar eslint.

1.Hace un npm init -y

2. npm i -D eslint. Esto es lo que crea:

  "devDependencies": {

    "eslint": "^8.44.0"

  }

3. npm init @eslint/config. Te tira una serie de preguntas. Solución:

To check syntax, find problems, and enforce code style

CommonJS (require/exports) (En este caso)

Hay guías de estilo públicas que se pueden utilizar. En nuestro caso es personalizado.

.json.

Space/Double/Windows/No/

Finalizado el cuestionario, te genera un archivo .eslintrc.json así:

{

    "env": {

        "browser": true,

        "es2021": true

    },

    "extends": "eslint:recommended",

    "parserOptions": {

        "ecmaVersion": "latest"

    },

    "rules": {

        "indent": [

            "error",

            4

        ],

        "linebreak-style": [

            "error",

            "windows"

        ],

        "quotes": [

            "error",

            "double"

        ],

        "semi": [

            "error",

            "never"

        ]

    }

}

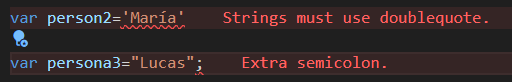
El cual podría hacerse manualmente o modificarse ahora.



Yo no definí esta regla, sin embargo, por defecto te tira error una vez que se crea el archivo. Se puede sobreescribir (override)la regla:

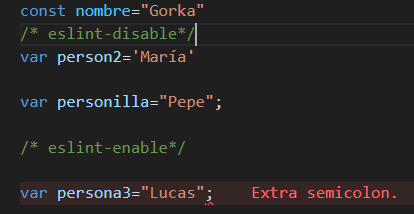
   "no-unused-vars":"off"

Así, el error desaparece.



Ahí gedea con 2 cosas que yo si configuré.

Disable / Enable



var persona3="Lucas"; //eslint-disable-line

Así se hace en una sola línea. A continuación, next-line:



En eslint-config hay las configuraciones estandarizadas. Él recomienda la standard.

Crear un script para alguna de estas opciones:

Se hace desde el package.json

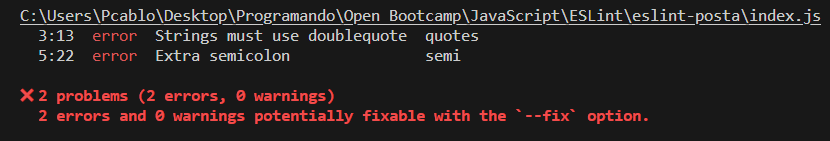
Opción 1: que nos muestre todos los errores

"scripts": {

    "lint": "eslint ."

  },

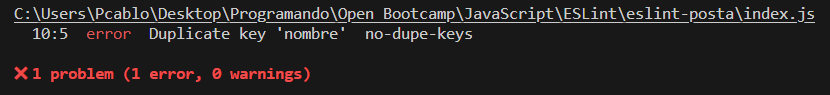
Luego se lo hace correr: npm run lint



Opción 2: que los corrija automáticamente.



Comando: npm run lint-fix



Corrigió los problemas que habían, pero uno persistió, porque no lo había configurado yo.

HTML

   <script src="index.js"></script>

Hace nmp init -y

Pero además hace:

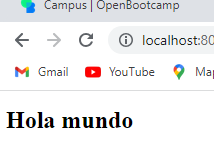
Npm install --global http-server

Este global server sería visto en el .json y por eso sería importante.

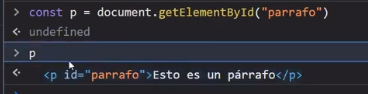
Toca ej .json agregando un script start

  "start": "http-server ."

Luego, entrando a localhost:8080 se visualiza esto:



Ubica esa línea en el head del html. En el index hice una multiplicación.



Habla de la importancia de dónde se inserta un script dentro del HTML.

TypeIt

Cómo se importa este archivo a nuestro HTML. En la parte previa al cierre de body lo pone.

<body>

    <h1>Hola mundo</h1>

    <span id="myElement"></span>

    <script src="index.js"></script>

    <script src="https://unpkg.com/typeit@8.7.1/dist/index.umd.js"></script>

    <script src="animacion.js"></script>

</body>

Eso se pone en el html.

Index.js no se ve reflejado en la página. El segundo script importa la librería TypeIt y el tercer script vincula con un Js en el que copié código de la página de TypeIt:

new TypeIt("#myElement", {

  strings: "Este es un texto personalizado?"

}).go();

Esa es la animación que genera:



Eventos

Bind() (vincula) se ejecuta en funciones.

Hay eventos por defecto. Vienen con un comportamiento por defecto.

Cuando uno haga click en:

 <input type="submit" value="Enviar">

Se ejecutará el evento onsubmit por defecto.

const pk=document.getElementById("formulario")

Ahí se asocial la variable con el elemento HTML con Id formulario, que es este:

 <form action="" method="" id="formulario">

        <input type="text" name="nombre" placeholder="Ingrese aquí su nombre">

        <input type="submit" value="Enviar">

    </form>

Acá primero se pasa por consola y luego se evita que se ejecute el comportamiento por defecto.

pk.addEventListener("submit", evento=>{

console.log(evento);

evento.preventDefault();

})

Eventos customizados

AddEventListener tiene:

String que hace referencia al nombre del evento y una función callback

Se crea una función que crea el evento, y posteriormente la llama con el dispatchEvent.

El evento se crea con new custom evento. Ahí se utiliza el mismo string del evento que se puso antes. Y luego en detail, se crea un objeto con los detalles del evento.

<body>

    <h1 id="h-texto">Hola, qué tal?</h1>

    <script src="index.js"></script>

</body>

const hTexto = document.getElementById("h-texto");

hTexto.addEventListener("cambioTexto", evento => {

    hTexto.innerText=evento.detail.texto

    hTexto.style.color=evento.detail.color

});

function cambiarTexto(nuevoTexto, color) {

  const evento = new CustomEvent("cambioTexto", {

    detail: {

      texto: nuevoTexto,

      color

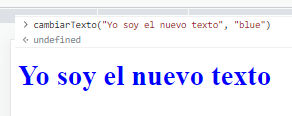
    }

  })

  hTexto.dispatchEvent(evento)

}

Finalmente escribo por consola:



Y se ejecuta la función que llama al evento y sucede la magia.

No se por què lo tuve que hacer con Live Server, de la otra forma no me funcaba la función.

JQuery

Permite interactuar con los elementos del html.

Descarga la versión comprimida. Lo hace así:

  <script src="https://code.jquery.com/jquery-3.7.0.min.js"></script>

Se utiliza con el símbolo $ mas (). Adentro del parentise va algún identificador.

$("li").hide();

Eso oculta los li.

Dice que como medida de seguridad se escribe de otra forma, para evitar que se ejecute el js antes de que se termine de cargar el html, porque tiraría errores. La forma es:

$(document0).ready(() =>{

$("h1").hide()

})

//o en su forma simplificada:

$(()=>{

$("h1").hide()

})

<body>

    <h1>Trabajando con Jquery</h1>

    <ul>

        <li id="el-1">Elemento 1</li>

        <li id="el-2">Elemento 2</li>

        <li id="el-3">Elemento 3</li>

            </ul>

            <button class="hide-btn">Hide</button>

            <button class="show-btn">Show</button>

</body>

<script src="index.js"></script>

//Evento de eventos. Disparado al cargarse el documento html

$(() => {

//Evento disparado al tocar botón Hide

  $(".hide-btn").click(() => {

    console.log("Ocultando");

    $("h1").hide();

  });

//Evento disparado al tocar botón Show

  $(".show-btn").click(() => {

    console.log("Mostrando");

    $("h1").show();

  });

});

Podrían cambiarse el .hide y .show por .fadeout y .fadeIn.

Ahora, al tocar en un elemento de la lista, cambiará el color del h1.

  $("li").click(()=>{

    $("h1").css({color:"red"})

  })

Ahí toca elementos .css

También está el dblclick (doble click).



Ahí hace click en un botón, y se agrega un elemento con texto “New Element” a una lista, usando el .append , con el .prepend lo agrega al principio.

$("li").mouseenter(()=>{

    console.log("El mouse ha entrado (ha hecho hover)")

})

Ahí, cada vez que hago hover, aparece ese msj por consola.

$("li").mouseenter((elem)=>{

    elem.target.style.color="violet"

})

$("li").mouseleave((elem)=>{

    elem.target.style.color="black"

})

Con esos de ahí, al hacer hover en un botón se colorea de violeta, al quitar el mouse afuera, retorna a su color negro.

Diálogos

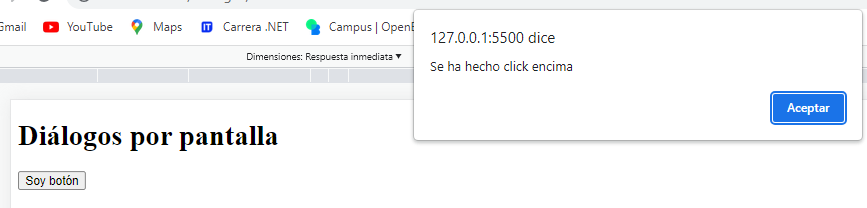
const boton = document.querySelector("#btn");

boton.addEventListener("click", ()=>{

 alert("Se ha hecho click encima")

})

Primero se captura en la variable botón a un botón de html. Cuando se hace click ahí, se dispara el evento alert, que va a mandar el msj detallado.



boton.addEventListener("click", ()=>{

 confirm("¿Está Ud. De acuerdo?") && console.log("OK")

})

Si se responde positivamente a la pregunta se imprimirá por consola OK.

boton.addEventListener("click", ()=>{

 confirm("¿Está Ud. De acuerdo?")

 ?  console.log("OK")

 : console.log("NO!!!!")

})

Lo llama operador ternario. ? si se acepta marcha OK, y si no, marcha NO!!!!

Forma sencilla y larga de verlo:



Dialogo en el que se introduce info por parte del usuario:

const botonInfo=document.querySelector("#info");

botonInfo.addEventListener("click", ()=>{

const nombre=prompt("¿Cuál es tu nombre secreto?")

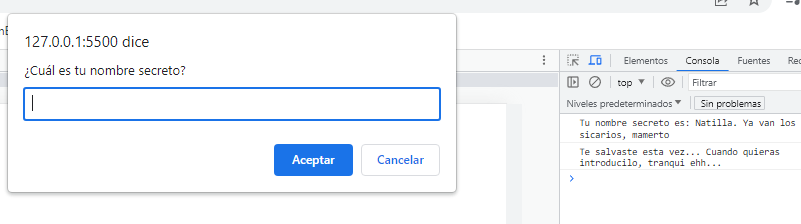
if (nombre){

    console.log("Tu nombre secreto es: "+nombre +". Ya van los sicarios, mamerto")

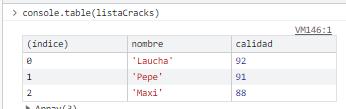
}

else console.log ("Te salvaste esta vez... Cuando quieras introducilo, tranqui ehh...")

})



Para visualizar info por consola de una lista, también puede hacerse el console.table:

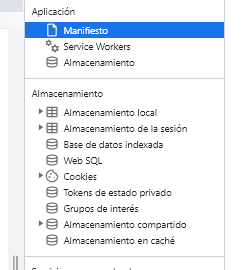


Persistencia de los datos en el navegador



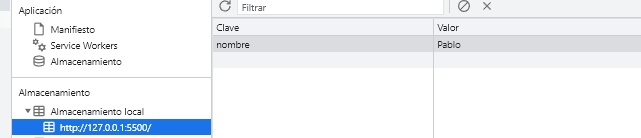
3 formas de guardar los datos.

Herramientas de desarrollador =>Application



localStorage.setItem("nombre","Pablo")

Escribo eso y aparece así:



Esa línea podría borrarse y aún así seguir vivo ese dato.

console.log(localStorage.getItem("nombre"))



Para guardar objetos es algo diferente y raro:

localStorage.setItem("Persona", JSON.stringify({nombre:"Lautaro", edad:36}))



Así se lo recupera:

console.log(JSON.parse(localStorage.getItem("Persona")))

JSON.stringify convierte un objeto/array en string

JSON.parse convierte un objeto stringifycado en un objeto de JavaScript.

Así se elimina:

localStorage.removeItem("Persona")

con las sessionStorage se trabaja igual. Con las cookies es algo diferente.

document.cookie="nombreCookie=PabloCookie"

console.log(document.cookie)

Por aplicación=> Cookie



Por consola



Así se le establece la caducidad:

document.cookie="nombreCaducidad=Nombre;Expires="

+ new Date (2023,0,1).toUTCString()

Eso no me funcó. Esto si:

document.cookie = "hacerAlgoUnaSolaVez="+persona.apellido+"; expires=Fri, 31 Dec 9999 23:59:59 GMT";

document.cookie= "ObjetoCookie="+JSON.stringify({Nombre:persona.nombre, Apellido:persona.apellido})+ ";max-age=120"

Drag & Drop

Arrastrar y soltar elementos en página web.

Hay una etiqueta específica de html para hacer un elemento arrastrable:

draggable="true"

Antes de tocar nada de Js. Ya tiene este html que se ve como se muestra:

<body>

    <div class="container">

        <div class="seccion">

            <p class="parrafo" draggable="true">1. Primero</p>

            <p class="parrafo" draggable="true">2. Segundo</p>

        </div>

        <div class="seccion">

            <p class="parrafo" draggable="true">3. Tercero</p>

            <p class="parrafo" draggable="true">4. Cuarto</p>

        </div>

    </div>

</body>



Con esto que se viene, por cada click hecho en un párrafo se printea “click hecho”.

const parrafos=document.querySelectorAll(".parrafo")

parrafos.forEach(parrafo=>{

parrafo.addEventListener("click", ()=>{

    console.log("Click hecho")

})

})

dragstart

Eso significa inicio de arrastre

 console.log("Estot arrastrando el párrafo "+parrafo.innerText)

Con eso te printea el texto del p.

parrafo.addEventListener("dragstart", ()=>{

    console.log("Estoy arrastrando el párrafo "+parrafo.innerText)

    parrafo.classList.add("dragging")

})

Ahí cuando está arrastrando algo le crea una clase (de html) nueva.

Dragover

El event está deprecated. Para poder utilizarlo, tuve que quitarle la “etiqueta” de indeseable.

secciones.forEach((seccion) => {

  seccion.addEventListener("dragover", () => {

    event.preventDefault();

    console.log("Dragover");

  });

  seccion.addEventListener("drop",()=>{

    console.log("Drop")

  })

});

Eso hace que al estar arrastrando se incremente el dragover, hasta que te vas del área del párrafo. Si soltás el mouse, imprime drop.

DataTransfer se usa bastante y tiene bastantes variaciones.

    event.dataTransfer.dropEffect="move"

Ese cambia un poquito el cursor cuando se pincha un elemento.

Papelera de reciclaje, papá!!

Geolocalización

<https://developers.google.com/maps/documentation/javascript/adding-a-google-map?hl=es-419#maps_add_map-html>

Básicamente copypasteo todo lo de ahí.

A través de la Api de Google maps.

<script>(g=>{var h,a,k,p="The Google Maps JavaScript API",c="google",l="importLibrary",q="\_\_ib\_\_",m=document,b=window;b=b[c]||(b[c]={});var d=b.maps||(b.maps={}),r=new Set,e=new URLSearchParams,u=()=>h||(h=new Promise(async(f,n)=>{await (a=m.createElement("script"));e.set("libraries",[...r]+"");for(k in g)e.set(k.replace(/[A-Z]/g,t=>"\_"+t[0].toLowerCase()),g[k]);e.set("callback",c+".maps."+q);a.src=`https://maps.${c}apis.com/maps/api/js?`+e;d[q]=f;a.onerror=()=>h=n(Error(p+" could not load."));a.nonce=m.querySelector("script[nonce]")?.nonce||"";m.head.append(a)}));d[l]?console.warn(p+" only loads once. Ignoring:",g):d[l]=(f,...n)=>r.add(f)&&u().then(()=>d[l](f,...n))})  
        ({key: "AIzaSyB41DRUbKWJHPxaFjMAwdrzWzbVKartNGg", v: "beta"});</script>

Se importa ese script.

Hace falta una api key, para eso hay que ir a Google cloud => crear proyecto.

key: "AIzaSyBJxd6RRb2-0F711hcR0och1XPwkzd1zX8"

Esa descargué yo.

Hay que tocar el css. Se puede importar también.

Agregar markers

  // The marker, positioned at Uluru

  const marker = new AdvancedMarkerView({

    map: map,

    position: position,

    title: "Uluru",

  });

Obtener geolocalización y luego hacer un marker de ello.

Crea función geoposición.

Podría el navegador no admitir geolocalización, por eso el if con el alert.

  if (email!=null && nombre!=null && mensaje!=null){

    console.log("Los datos han sido ingresados correctamente")

  }

  else {

    alert(email.value + " " + nombre.value + " " + mensaje.value);

  }