* 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();  });  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"));  });  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 |