

Javascript

Lenguaje Interpretado

Var nombre = " "

(Las variables son dinámicas), se puede almacenar y reutilizar y la reutilización es de cualquier tipo de dato

`typeof` → retorna el tipo de variable que se almacena

No se define el tipo de variable

Tipos de Datos

- int
- float
- string
- boolean
- function
- Symbol
- Clase → función
- undefined
- object

null = ausencia de valor

Array → tipo Object

var autos = [] ; console.log(" ") ;

Concatenación

Var = var1 + var2; → concatenación de cadenas

Javascript evalúa de izquierda a derecha

Var var1 = " ";

let var1 = " " → Buena práctica

const var1 = " " → No se modifican después

En Javascript se pueden declarar variables sin necesidad de poner var, let o const

Javascript es sensible a Mayúsculas

var1 ≠ Var1 ← son variables diferentes

Javascript no toma como válido variables que inician con un número

Operadores

+ → suma

- → resta

* → multiplicación

/ → división

** → exponente

% → residuo

Incremento y Decremento

Preincremento

$a++$ → se incrementa y luego se asigna

Postincremento

$a + +$ → se asigna y luego se incrementa

Predecremento

$--a$ → se decrementa y luego se asigna

Postdecremento

$a--$ → se asigna y luego se decrementa

Precedencia de Operadores

→ ()	++ --	*	+ -	>>	<< =
[]	+ -	/		<<	>> =
.	! ~	%	L to R	L to R	L to R
new	delete				
L to R	typeof	L to R			
	void				
R to L					

== → se revisa el valor sin importar el tipo de dato
!= → revisa el valor y el tipo de dato
L to R

Operador Ternario

Condición? Expr1 : Expr2
↑ true ↑ false

let resultado = (3 > 2)? "verdadero": "falso"

Convertir de String a Number

Number(variable);

NaN

Not a Number → "18x"

isNaN(var) → función que verifica
si una variable es o
no un número

Estructura if-else

```
if(condicion){  
} else {  
}
```

Estructura switch

```
switch(var){  
    case 1: break;  
    case 2: break;  
    case 3: break;  
    default:}
```

Estructura While

while (condicion){

{ var ++;

Estructura Do-While

do{

{while (condicion);

Estructura For

for (inicializacion; condicion; incremento){

y

Break

Se utiliza para romper ciclos

Continue

Ya no deja que siga el programa o se ejecuten las líneas inmediatas y sigue a la siguiente iteración

Labels

Hace que se vaya a una parte de tu programa donde tenga esa etiqueta

inicio:

continue inicio;

Tipo de programación
go to

Arreglos

let array = new Array(); ← no se usa

const array = [];
↑ ^ apunta a la referencia de
no modifica memoria, accedemos a la
la referencia ubicación del array para poder
de memoria modificarlo

array[0] → devuelve el elemento en esa posición

array.length → retorna el tamaño del array

array.push(' '); → agrega el elemento
al array

Array.isArray(array) → función para saber
si es un arreglo
array instanceof Array ↑

Funciones

```
function nombre(a,b){  
  ↗ parámetros  
}
```

```
nombre(5,6);
```

```
function nombre(a,b){  
  return a+b;  
}
```

```
let resultado = nombre(5,6);
```

Funciones de Tipo Expresión

```
let x=function(a,b){ return a+b};
```

```
let var = x(1,2);
```

Funciones de tipo Self Invoking

Solo se manda a llamar a sí misma y solo se utiliza una vez

```
(function(){})(function(a,b){  
})(5,3);
```

Funciones como Objetos

arguments.length → retorna el número de argumentos de una función

Las funciones son objetos en JS

Tienen propiedades y métodos

Funciones Tipo Flecha

const var = (a, b) => a + b
 ^ argumentos

let var2 = var(2, 3);

Parámetros y Argumentos

Parámetros → lista de variables que están definidas en una función → (a, b)

Argumentos → Valores que reciben los parámetros → (5, 3)

Paso por valor

Se trabaja con tipos primitivos, se envían copias a la función, por lo que la variable original no cambia, solo pasa el valor a la función, pero no su variable.

```
function cambiar(a){  
    a = 20  
}
```

Paso por Referencia

Se usa en objetos, se pasa el objeto, la variable apunta al objeto, el argumento pasa la referencia del objeto, por lo que es posible modificarlo, ya que no cambia la variable, cambia lo que apunta la variable.

Objetos

Contiene propiedades y métodos

```
let p = {
```

nombre: " ",

p.nombre

Accedemos

ap: " ",

p.ap

a los datos

edad: ;

p.edad

}

```
let p = {
    nombre: " ",
    ap = " "
}
edad = ;
metodo: function() {
    return this.nombre + '' + this.ap
}
```

↑ Acceso a los métodos

Declaración de métodos

```
let p2 = new Object(); ← crear objeto
p['ap'] → acceder a las propiedades de un objeto
```

For in

```
propiedad del objeto
↓
for (propiedad in p) {
    p[propiedad]; → accedemos al valor
                    de la propiedad del
                    objeto
}
```

Agregar y Eliminar Propiedades de un Objeto

`p.propiedad = 'j'`

↑ Agregar propiedades

`delete p.propiedad` → eliminar propiedades

Imprimir Object

Concatenando → `(p.var1 + ',' + p.var2);`

For in → `for (propiedad in p) {`

`persona[propiedad]`

`Object.values` → `let array = Object.values(p);`

↑
Convierte objeto en arreglo
`console.log (array);`

`JSON` → `let json = JSON.stringify(p);`

↑
Convierte objeto en String
`console.log (json);`

Método GET

```
get metodo() {  
    return this.var1.toUpperCase();  
}  
p.metodo  
    ↑  
    | Convierte la  
    | String en mayúsculas
```

Método SET

```
set atr(var) {  
    this.atr = atr  
}  
p.atr = 'en';
```

Constructor

Función especial para crear objetos

new → reserva espacio en memoria para el objeto que vamos a crear

```
function Persona(nombre, ap){  
    this.nombre = nombre;  
    this.ap = ap;  
}
```

```
let p1 = new Persona('nombre', 'apellido');  
let Objeto = {};
```

← crear un objeto

Prototype

Se usa para agregar propiedades a todos los objetos ya creados y agregar un valor por default

```
Clase.prototype.attr = ' ';
```

Call

No permite llamar un método definido en un objeto, desde otro objeto

```
objeto.met.call(objeto2);
```

Apply

objeto.metodo.apply(objeto2);

Igual que call, solo que en apply para pasar argumentos, se debe de pasar un array con los valores, call se mandan los parámetros uno por uno

.apply(obj, array);

.call(obj, ' ', ' ', ...);

Clases

Es una plantilla, posee atributos y métodos,
un objeto es una instancia de una clase,
cada objeto contiene sus propios valores

```
class Clase {
```

```
    constructor(atr, atr2) {
```

```
        this.atr = atr;
```

```
        this.atr2 = atr2;
```

```
}
```

```
let obj1 = new Clase(' ', '');
```

Se manda a
llamar al
constructor

Método GET y SET

Métodos usados para obtener o cambiar el valor de un atributo de un objeto

```
get atr() {  
    return this._atr;  
}
```

```
set atr(atr) {  
    this._atr = atr  
}
```

obj.atr;

obj.atr = ' ';

Se mandan a llamar dependiendo de las variables

No son necesarios pero es buena práctica agregar estos métodos

Herencia

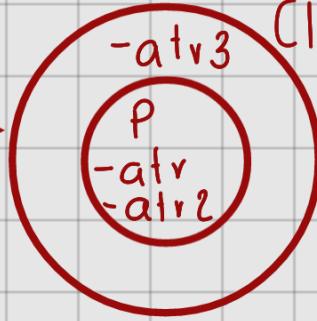
Capacidad de heredar atributos y métodos de la clase padre a la clase hija

```
class Clase extends ClaseHija {  
    constructor (atr1, atr2, atr3) {  
        super (atr1, atr2);  
        this._atr3 = atr3;  
    }  
}
```

llama al
constructor de la
clase padre

Para crear métodos en clases solo basta con poner el nombre del método, sin la palabra function

```
metodo () { → obj. metodo();  
    return '';  
}
```



← Objeto
creado, uno
solo

Sobreescritura

Modificar el comportamiento de un método de la clase padre, debe de tener el mismo nombre y parámetros

```
metodo(){  
    super.metodo() + ' ';
```

Clase Object

Clase padre de todas las clases

Método `toString`

Se hereda de manera automática, debido a que es un método de la clase Object, por lo tanto, lo hereda cualquier clase. Se debe de sobreescribir para que funcione y de la información del objeto

Palabra Static

Hace que los métodos definidos por esta palabra ya no dependan de los objetos, si no de la clase

```
static metodo () {
```

}

Atributos Estáticos

```
static contador = 0;
```

↑

pertenece a la clase y no a los objetos, se accede por la clase

las clases hijas también heredan los métodos y atributos estáticos

Constantes Estáticas

```
static get MAX_DBJ () {
```

```
    return 5;
```

y

For Each

array.ForEach((posicion) => {

 ↑
 arreglo

 función
 ↓

 ↑
 variable que
 sirve para recover
 el array

array.ForEach((recorrimiento, indice, arr))

 ↑ ↑ ↑
 variable posición array

Los arreglos soportan cualquier tipo de dato,
entero/floatante/string/función y objetos en
el mismo array

Template String

toString() {

 return ` \$ {this.var} \$ {this.var1}`

Respeta los enteros y valores que se va
dejando

Modo Strict

Se usa para que una variable sea declarada forzadamente o función

"use strict"

Se puede utilizar al inicio del programa
o al inicio de una función

Polimorfismo

Múltiples formas en que se puede llamar
un método, dependiendo su tipo de dato al
cual apuntan

InstanceOf

Ayuda a saber si la variable es una instancia
de una clase específica

if (var instanceof Clase) { }

Manejo de Errores

Se usa para no terminar el programa abruptamente, si no que manejará el error y seguirá el programa

```
try {  
    catch(){  
        }  
    finally{  
        }
```

← líneas que pueden contener el error
↑ cacha el error y devuelve el error y el mensaje del mismo
← este bloque siempre se ejecuta independiente de los errores si hay o no

Throws

Arroja nuestros propios errores

```
try {  
    throw ; , ;  
} catch(){}
```

↑ bloq que arroja un error propio, puedes configurar el mensaje

Funciones Flecha

const ff = (a, b) => {}
{} ↑ ← contenido de la
parametros función
(No se aplica el hosting, se debe de definir
y luego llamar / no al revés)

const ff = () => console.log(' ') ;
() ↑ Solo si hay solo una
línea de código

const ff = () => ({ var: ' ' })
() ↑ regresa un objeto / se
pone entre paréntesis
para evitar confusiones

const ff = parámetro => ' ' ;
parámetro ↑ se omiten paréntesis si solo
hay un parámetro

FuncionesCallBack

Función que se manda de parámetro otra función y dentro de esa función se puede mandar a llamar tal función que fue un parámetro, una función que manda a llamar una función dentro de ella

```
function suma(op1,op2,funcionCallback){  
    let res = op1 + op2;  
    funcionCallback(res);  
}
```

```
suma(5,3,funcion);
```

↑ Solo con el nombre es suficiente

Función setTimeout

Se usa para poner un timing de ejecución para que se ejecute una función callback esto en ms

`setTimeout(funcion, 3000);`
3 milisegundos

Ayuda a tener varios procesos para que se ejecuten en paralelo y manden después del timeout establecido

`setTimeout(()=>{ ' ' }, 4000);`

Esto hace que ya no sea secuencial la ejecución y que se ejecutan las funciones dependiendo su timing.

Función setInterval

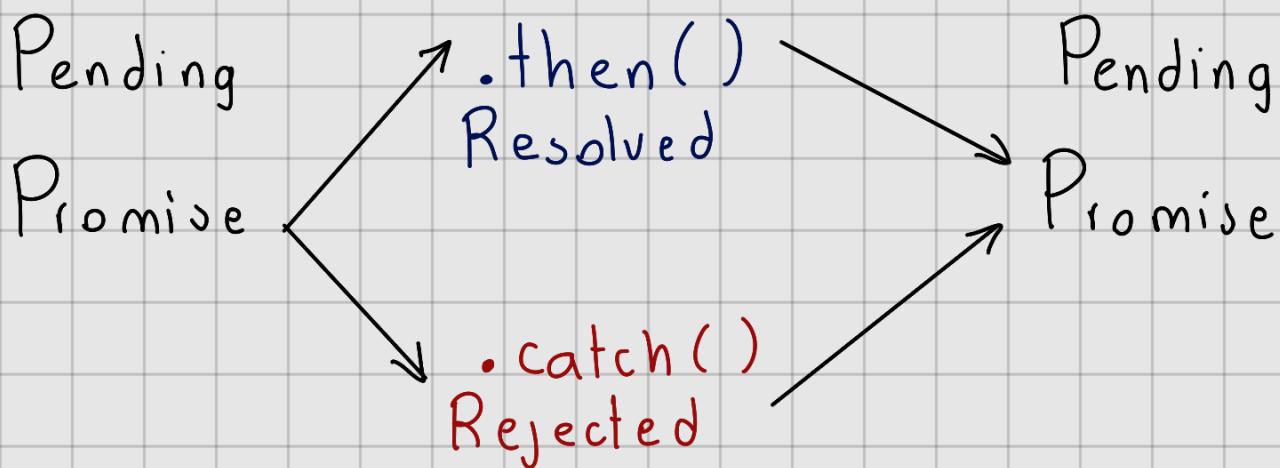
Manda a llamar a la función callback
muchas veces dependiendo su tiempo de
intervalo de llamada establecido

```
const función = () => {  
    ↑  
    ← callback
```

```
setInterval (función, 1000);  
    ↑  
    Función  
    ↑  
    callback  
    ↑  
    Intervalo de  
    llamada
```

Promesas

Una promesa es código que tiene varios estados, se encuentra en estado pendiente, una vez ejecutado el código pasa a estado resuelto o rechazado



```
let p = new Promise((resolved, rejected)  
  => { let e = true;  
    if (e) resolver('');  
    else rechazar('');  
  });
```

```
p.then(  
    valor => console.log(valor),  
    error => console.log(error));  
);  
  
p.then(valor => console.log(valor))  
.catch(error => console.log(error));
```

```
let p = new Promise((resolver) => {  
    setTimeout(() => resolver(``), 3000);  
})  
  
p.then(valor => console.log(valor));
```

Async

Al poner la palabra `async` antes de la definición de un método, significa que ese método está obligado a regresar una promesa

```
async function f () {  
    return ' ';
```

{

```
f().then(valor => console.log(valor));
```

Await

Espera el resultado de una promesa y trabaja con el resultado

```
async function f () {  
    let p = new Promise(resolver => {  
        resolver(' ');  
    });  
}
```

```
console.log(await p);
```

```
async function f() {  
    let p = new Promise(resolver => {  
        setTimeout(() => resolver(''),  
            3000);  
    });  
    console.log(await p);  
}
```

DOM HTML

Document Object Model

Documento HTML, las etiquetas de HTML de cualquier página HTML son el DOM, cada elemento HTML son objetos en javascript, document es el objeto que hace que javascript interactúe con los elementos HTML

<script></script> → etiqueta que agrega código js a un documento HTML

A los elementos que se quieren manipular se les debe de poner un id

let c = document.getElementById(' ').innerHTML;

↑ Accedemos al elemento
↑ Retorna un objeto
↑ Accedemos al contenido

1 C.innerHTML = '';

↑ Se modifican los elementos HTML

let p = document.getElementsByTagName('p');

↑ puede retornar
un arreglo de
elementos

↑ Obtiene los elementos
mediante el nombre
de la etiqueta

let c = document.getElementsByClassName('');

↑ puede retornar
un arreglo de
elementos

↑ Obtiene los elementos mediante el nombre de la clase
el nombre de la
clase

let q = document.querySelectorAll('');

↑ Obtiene todos los elementos
del tipo de elemento que
contiene cierta clase
('p.azul')

onclick='funcion()'

↑ Manda a llamar a una función de javascript para
manejo de eventos de tipo click en HTML

```
let f = document.forms['']  
↑           ↑           ↑  
regresa    recuperamos   id del formulario  
el formulario información de  
en forma    un formulario  
de arreglo
```

- `.value` → accedemos a el valor del contenido del elemento form de HTML

`document.write(' ');` → escribe información
sobre el documento HTML
↑
sobreescibe si se utiliza
después de que se haya
cargado el documento
HTML

```
document.getElementById(' ').src=' ';
```

↑ Obtenemos la imagen ↑ Accedemos ↑ Asignamos
por el id al atributo de nuevo valor
src

```
document.getElementById(' ').style.color='';  
          ↑           ↑  
Accedemos al atributo   Elemento a  
de estilo      modificar
```

`document.getElementById(' ').onclick = funcion`
evento de click

onload = ' ' → Evento cuando termina de cargarse la página web

cookies: Archivos que guardan información cuando navegamos por páginas web

navigator: Objeto de javascript

`navigator.cookieEnabled`

↳ Sabes si las cookies están habilitadas

onchange = ' ' → Se usa para modificar los valores de un form, se llama cuando se modifican los campos

onmouseover = ' ' → Se usa para el cambio de un elemento pero cuando pones el mouse encima de tal elemento.

onmouseout = ' ' → Se usa para modificar un elemento cuando quitas el mouse de dicho elemento

onmousedown = ' ' → Cuando damos click sobre un elemento

onmouseup = ' ' → Cuando ya se dio click, se suelta el elemento

onfocus = ' ' → Cambia el elemento cuando el foco está en el, es decir, cuando escribes dentro de una caja de texto

onblur = ' ' → Cambia el elemento cuando pierde el foco, es decir, cuando ya se terminó de escribir o utilizar la caja de texto

addEventListener(' ', funcion) ← callback

↑ Método para agregar 1 o más eventos que quieras programar

La función asociada solo es la referencia Al argumento se pone el nombre pero eliminando el 'on'

```
function función(evento){  
    evento.target  
    ↗ Opcional  
    ↑  
    regresa el elemento que lanzó este evento  
  
Las funciones flecha se pueden usar, sin  
embargo no se podría reutilizar código  
document.getElementById(' ').addEventListener('focus', (evento) => {  
    evento.target.style.background = ' ';});  
↑  
función flecha ¡No se puede reutilizar!  
forma.addEventListener(' ', function, true);  
↑  
Use Capture o propaga los eventos a  
Delegación de → elementos internos  
Eventos contenidos en el elemento  
afectado
```