

Left

styles.css

Right

styles.min.css

FastGulp

FastGulp

dist

assets

CSS

styles.css

styles.min.css

node_modules

src

CSS

bootstrap.css

index.css

fonts

styles.css

```

6639     }
6640   }
6641   @media (min-width: 1200px) {
6642     .visible-lg-block {
6643       display: block !important;
6644     }
6645   }
6646   @media (min-width: 1200px) {
6647     .visible-lg-inline {
6648       display: inline !important;
6649     }
6650   }
6651   @media (min-width: 1200px) {
6652     .visible-lg-inline-block {
6653       display: inline-block !important;
6654     }
6655   }
6656   @media (max-width: 767px) {
6657     .hidden-xs {
6658       display: none !important;
6659     }
6660   }
6661   @media (min-width: 768px) and (max-width: 991px) {
6662     .hidden-sm {
6663       display: none !important;
6664     }
6665   }
6666   @media (min-width: 992px) and (max-width: 1199px) {

```

styles.min.css

```

1  /*!
2   * Bootstrap v3.3.5 (http://getbootstrap.com)
3   * Copyright 2011-2015 Twitter, Inc.
4   * Licensed under MIT
5   (https://github.com/twbs/bootstrap/blob/master/LICENSE)
6   */
7  /*! normalize.css v3.0.3 | MIT License |
8   github.com/necolas/normalize.css */
9  .label,sub,sup{vertical-align:baseline}.btn,.btn-group,.btn-group-vertical,.caret,.checkbox-inline,.radio-inline,img{vertical-align:middle}hr,img{border:0}body,figure{margin:0}.navbar-fixed-bottom.navbar-collapse,.navbar-fixed-top.navbar-collapse,.pre-scrollable{max-height:340px}.btn-group>.btn-group,.btn-toolbar.btn,.btn-toolbar .btn-group,.btn-toolbar .input-group,.col-xs-1,.col-xs-10,.col-xs-11,.col-xs-12,.col-xs-2,.col-xs-3,.col-xs-4,.col-xs-5,.col-xs-6,.col-xs-7,.col-xs-8,.col-xs-9,.dropdown-menu{float:left}.remarked,html{webkit-text-size-adjust:100%}html{font-family:sans-serif;-ms-text-size-adjust:100%}article,aside,details,figcaption,figure,footer,header,hgroup,main,menu,nav,section,summary{display:block}audio,canvas,progress,video{display:inline-block;vertical-align:baseline}audio:not([controls]){display:none;height:0}[hidden],template{display:none}a{background-color:transparent}a:active,a:hover{outline:0}b,optgroup,strong{font-weight:700}dfn{font-style:italic}h1{margin:.67em 0}mark{color:#000;background:#ff0}sub,sup{position:relative;font-size:75%;line-height:0}sup{top:-.5em}sub{bottom:-.25em}svg:not(:root){overflow:hidden}hr{height:0;box-sizing:content-

```

Javascript

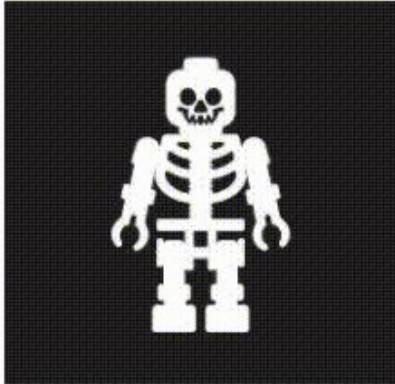
Mood de esta semana



Recordemos para qué sirve Javascript

...

HTML
structure



CSS
presentation/appearance



JavaScript
dynamism/action

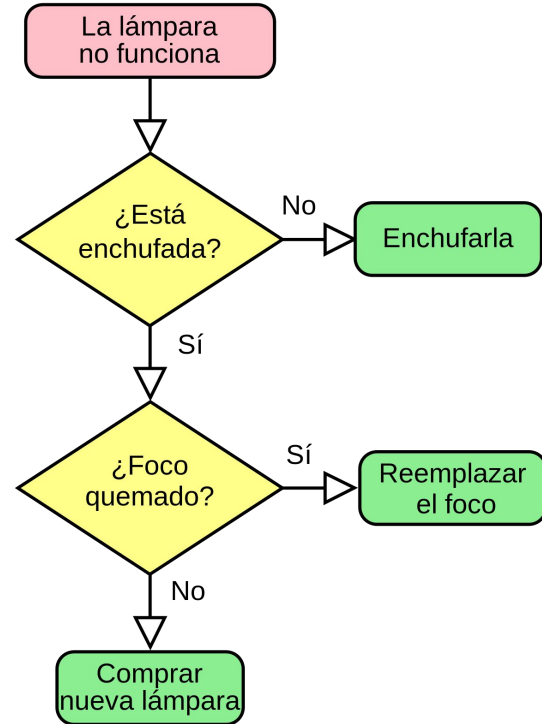


Por qué decimos que Javascript da la funcionalidad?

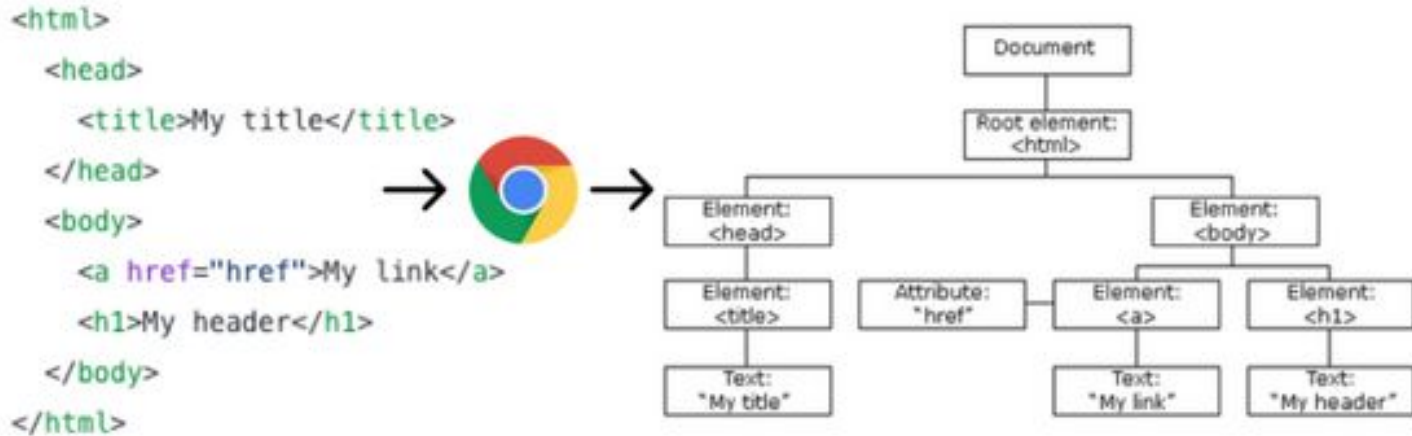
1. Javascript es un lenguaje de programación.

Un lenguaje de programación nos va a permitir resolver problemas.

Ej.: Podemos tomar datos ingresados por el usuario, procesar esos datos y dar una respuesta.



2. Javascript puede acceder al DOM



En resumen...

Modal Dialog

Auto Extra Small Small Medium Large

Modal Title

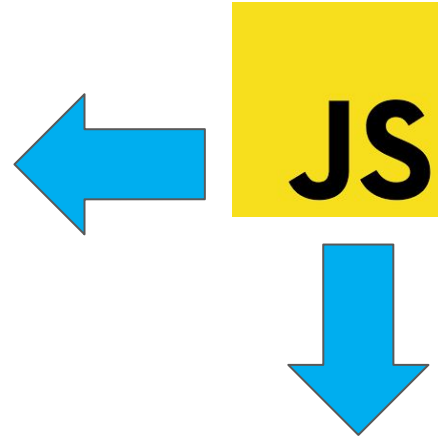
Email address

Email

Password

Password

Save Cancel



Comencemos!

Implementando Javascript

```
<html>
<head>
</head>
<body>
.
.
.
Los scripts suele ir al final,
justo antes de terminar el body
<script src="./script.js"></script>
<script src="./script2js"></script>
<!--Se puede agregar aqui mismo el
script entre <script></script>
pero no es algo común -->
<script>
//codigo de javascript
console.log("Hola")
</script>
</body>
</html>
```

referenciamos el
archivo

ó

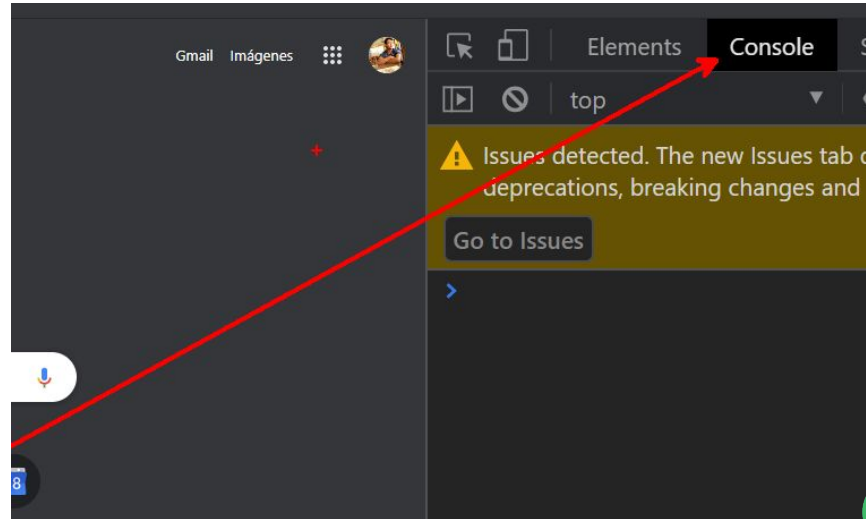
lo incluimos en el
mismo documento
HTML



console.log();

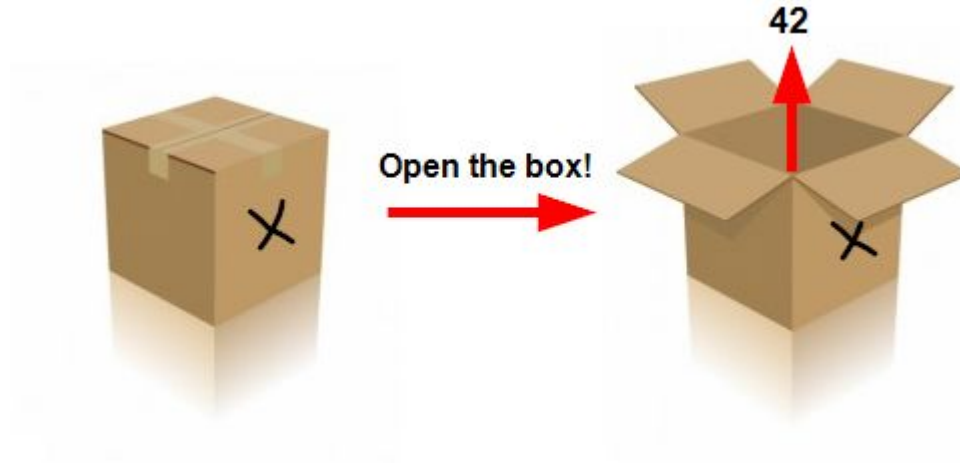
Para ver los resultados de Javascript podemos usar la Consola de Javascript, la encontramos en las herramientas de desarrollo (F12) en la pestaña Console

Si queremos mostrar algo ahí usaremos el:



Variables

Cualquier lenguaje de programación va a necesitar un lugar donde guardar información y una manera de cómo referenciar esa info.



Variables y tipos de datos

Los nombres de variables no pueden contener espacios y son sensibles a mayúsculas.

variable identifier
start with
assignment operator
value
End of the statement

`var name = 'James Bond';`

```
// Datos numericos
var edad = 30;
var promedio = 14.5;
// Cadenas de texto
var nombre = "Osmar Montesinos";
var direccion = 'Av Antartica 5002';
var numero = "20";
// Booleanos
var aprobado = false;
var error = true;
// Podemos declarar variables sin inicializar
var x;
```



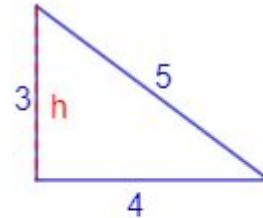
Operadores Aritméticos

Operator	Description	Example
+	Addition	$25 + 5 = 30$
-	Subtraction	$25 - 5 = 20$
*	Multiplication	$10 * 20 = 200$
/	Division	$20 / 2 = 10$
%	Modulus	$56 \% 3 = 2$
++	Increment	var a = 10; a++; Now a = 11
--	Decrement	var a = 10; a--; Now a = 9



Ejercicio









Programa un algoritmo que calcule el área de un triángulo, considera las variables necesarias y muestre el resultado en la consola.



$$\begin{aligned} A &= \frac{b \cdot h}{2} = \\ &= \frac{4 \cdot 3}{2} = \\ &= \frac{12}{2} = 6 \end{aligned}$$

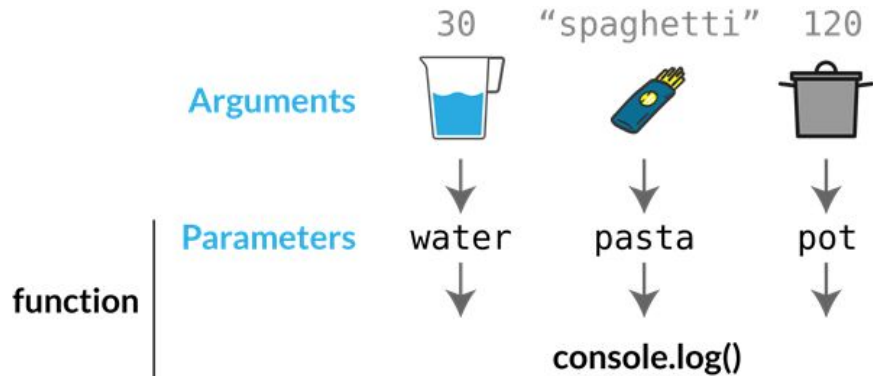
Funciones

Una tarea que puedo llamar varias veces para que haga la misma cosa.

```
function makeSandwich(, , ) {  
  let sandwich =  =  +  + ;  
  return  ;  
}
```

Funciones

Una función puede tener unos parámetros (nombre) ya conocidos y esos para parámetros tendrán un argumento (valor).



```
function hacerPasta(agua,pasta,olla){  
  //makepasta :D  
  return pasta; //opcional  
}  
  
hacerPasta(1000, "spaghetti", "olla de 2L");
```


Ámbito (Scope)

Dependiendo de donde se declare una variable solo podremos acceder a ella en ese ámbito que está declarado y no fuera de ese ámbito.

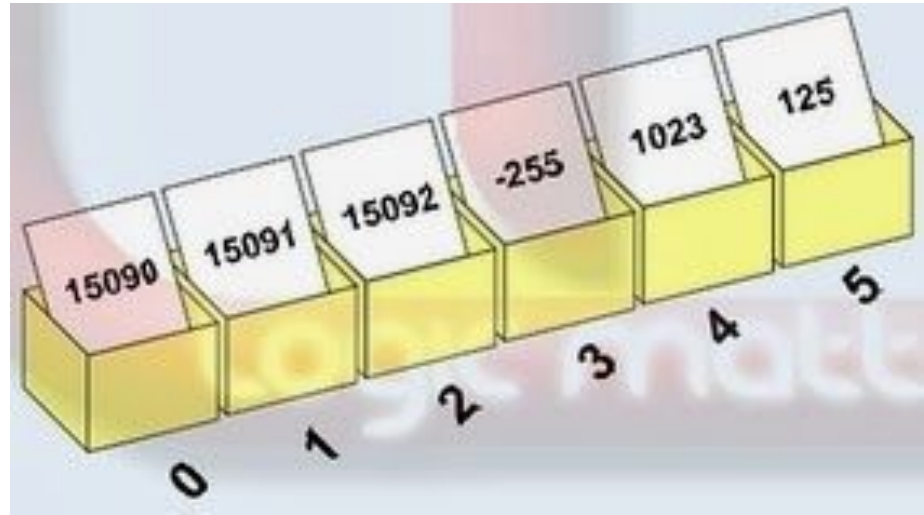
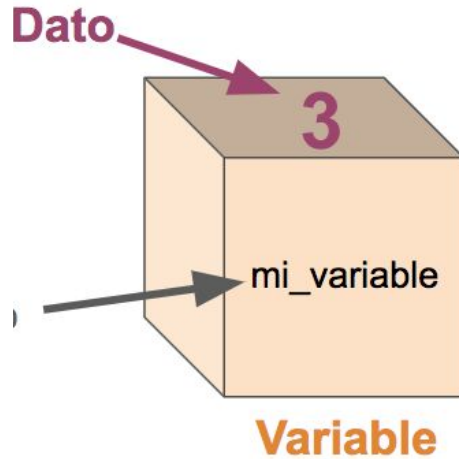
```
var a = 4;  
  
function foo(x) {  
  var b = a * 4;  
  
  function bar(y) {  
    var c = y * b;  
    return c;  
  }  
  
  return bar(b);  
}  
  
console.log(foo(a));  
// 256
```

The diagram illustrates variable scope with three nested levels, each represented by a colored box and a numbered label:

- 1** (Global Scope): `global: a, foo`. This scope contains the global variable `a` and the `foo` function.
- 2** (foo Scope): `foo: x, b, bar`. This scope is created when `foo` is called and contains the parameter `x`, the local variable `b`, and the `bar` function.
- 3** (bar Scope): `bar: y, c`. This scope is created when `bar` is called from within `foo` and contains the parameter `y` and the local variable `c`.

Arreglos

Si tengo muchos datos en vez de crear muchas variables, puedo crear un arreglo, los arreglos me permiten guardar información en una sola variable pero con el añadido de una posición



Arreglos



```
var nombres = ['Osmar', 'Freddy', 'Lesly', 'Elon', 'Majo'];  
var edad = [18, 25, 30, 27, 29, 24]
```

```
console.log(nombres[0]) //Osmar  
console.log(nombres[4]) //Majo
```

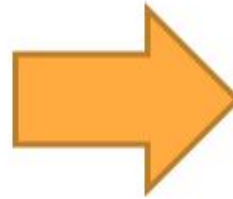
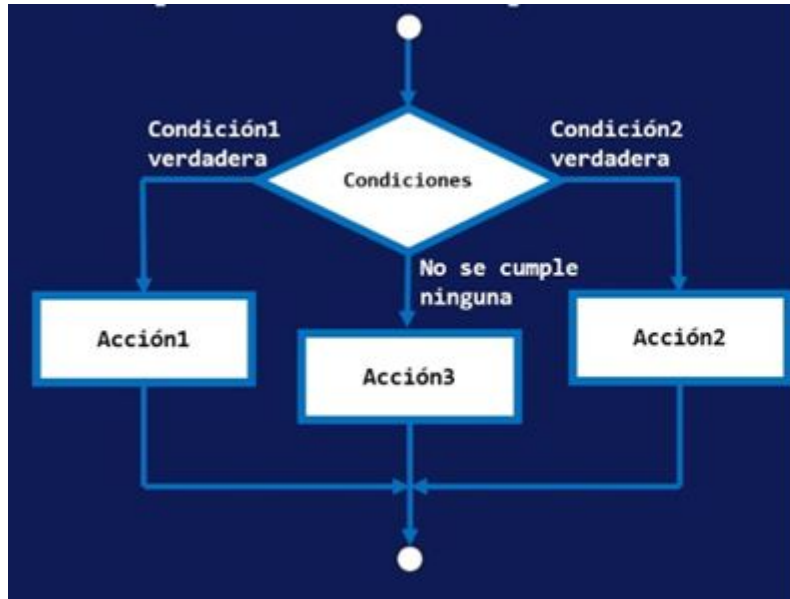
```
console.log(edad[1]) //25
```

```
//La longitud de el arreglo, cuantos elementos tiene  
console.log(edad.length) //6
```

Estructuras Condicionales

If Else

Si queremos evaluar situaciones y ejecutar algo si esa situación se cumple tenemos if else, donde podemos evaluar una situación o varias



```
>> if (condición1)
    { Acción1; }
else if (condición2)
    { Acción2; }
else
    { Acción3; }
```

Operadores Lógicos

Operadores condicionales

- == Igual
- != Diferente
- > mayor
- < menor
- <= menor igual
- >= mayor igual

Operadores lógicos

Operador	Significado
	OR lógico (ó)
&&	AND lógico (y)
!	NOT lógico (no)

Operadores Lógicos

NOT

x	x'
0	1
1	0

AND

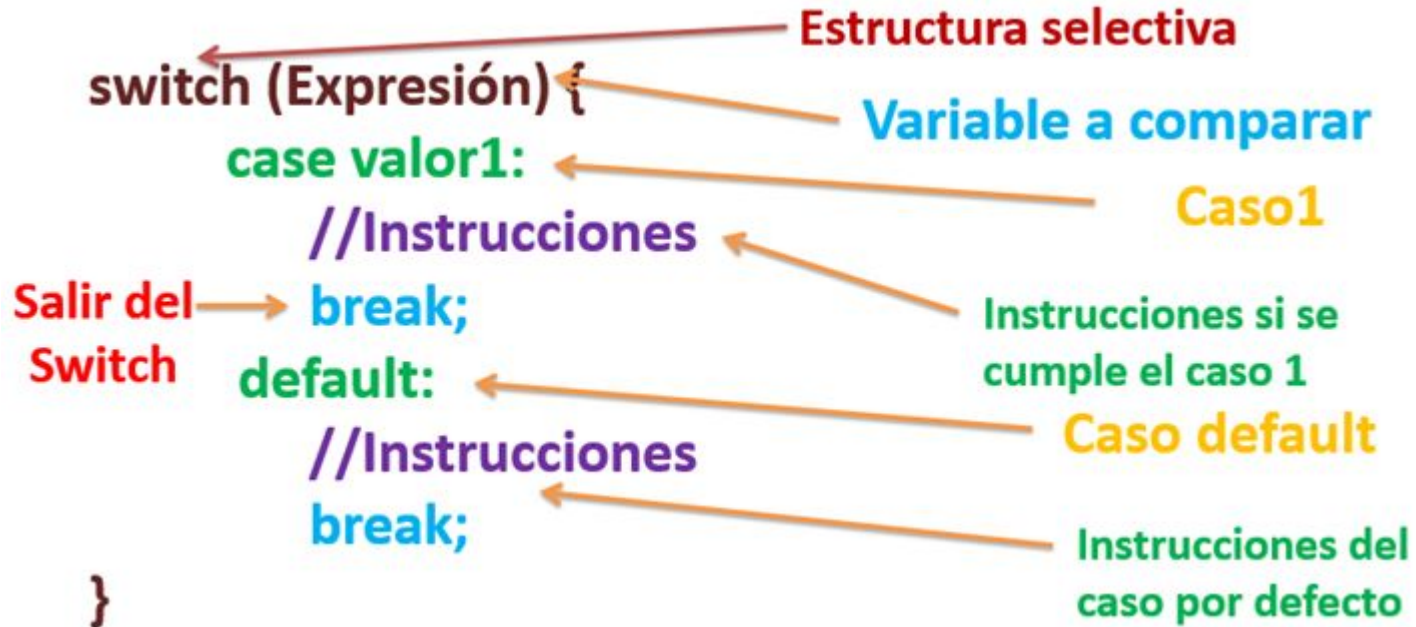
x	y	xy
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x+y$
0	0	0
0	1	1
1	0	1
1	1	1

Switch

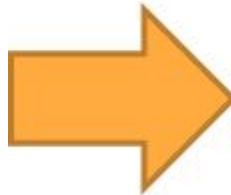
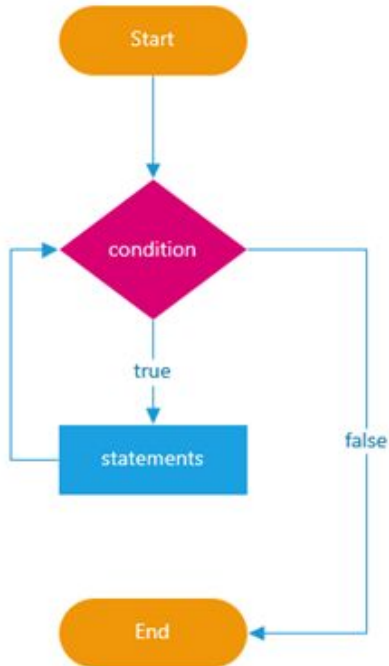
Para evaluar diferentes condiciones hasta que alguna se cumpla, si ninguna se cumple ejecutará el default.



Y si queremos hacer tareas que son repetitivas?

While

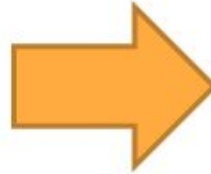
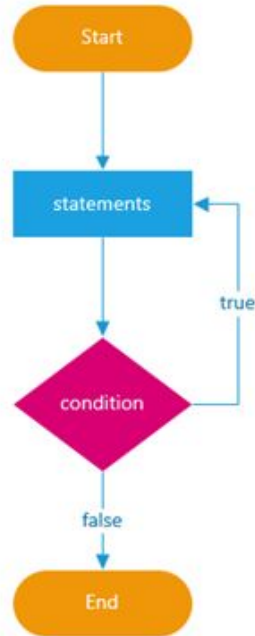
Si se cumple una condición la ejecutará hasta que esa condición no se cumpla



```
let count = 1;
while (count < 10) {
  console.log(count);
  count +=2;
}
```

Do While

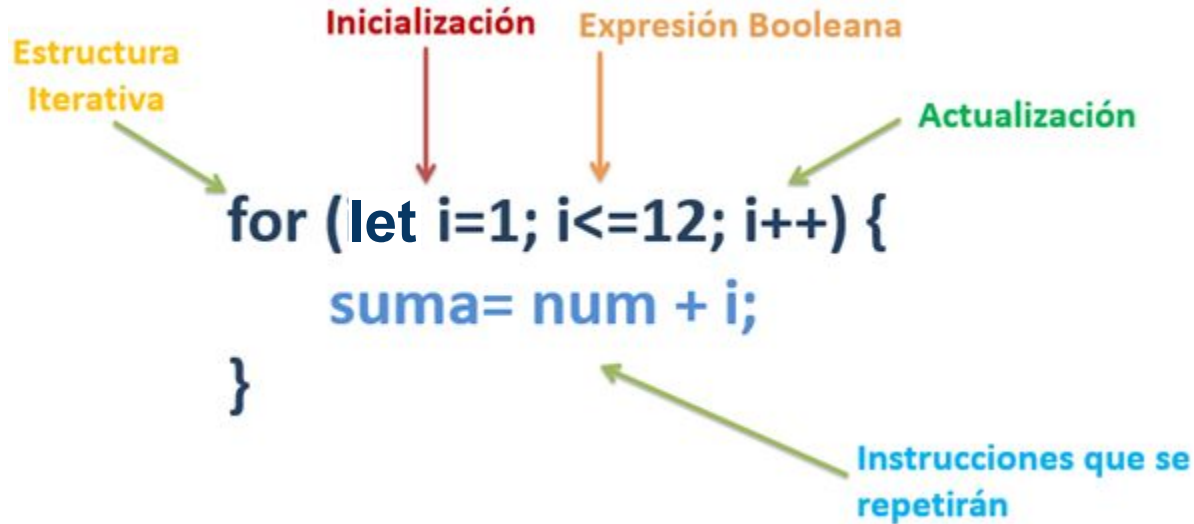
Ejecutará una instrucción al menos una vez y de ahí verificará la condición



```
let count = 0;  
do {  
  count++;  
  console.log('count is:' + count);  
} while (count < 10);
```

FOR

Las instrucciones se cumplirán de forma repetitiva hasta que la expresión booleana (la condición que servirá para limitar el funcionamiento de mi bucle) no se cumpla.



arreglos con FOR

Las sentencias for son muy útiles cuando se trata de “recorrer” arreglos ya que con el iterador podemos recorrer cada posición que tiene un arreglo.

```
var miVector3 = new Array("Juan", "Pons", 45);
```

0	1	2
Juan	Pons	45

Representación gráfica de un array de 3 posiciones

```
let miVector3 = ["Juan", "Pons", 45]

for (let i = 0; i < miVector3.length; i++) {
  console.log(miVector3[i]);
}

/**
=>
Juan
Pons
45|
*/
```

Break y Continue

Me sirven para controlar el flujo de un bloque de código

```
for (let i = 0; i < 10; i++) {  
    //salta a 4 o nos detenemos en 3  
    if (i == 3) {  
        //continue; break;  
    }  
}
```



Break



Continue

Objetos

Solemos tener objetos en todas partes, entendamoslo como cualquier cosa de la que podamos describir sus características y capacidades



Objetos

Los objetos me permiten guardar datos de forma estructurada, relacionándolos como propiedad: valor



```
let jugador = {  
  nombre: "Dybala",  
  nacionalidad: "Argentina",  
  disparo: 89,  
  pase: 85,  
  hobbies: ['entrenar', 'tomar mate']  
  
  patear: function(fuerza){  
    //patea  
  }  
}
```


Resumen Operadores

Sirvan para hacer operaciones matemáticas, operaciones lógicas o comparaciones

```
5 == 5    //true  
5 == '5'  //true  
5 === 5   //true  
5 === '5' //false
```

```
5 != 5    //false  
5 != '5'  //false  
5 !== 5   //false  
5 !== '5' //true
```

```
2 + 2  // 4  
4 - 2  // 2  
2 * 3  // 6  
4 / 2  // 2
```

```
let a = 10; log(++a, a); // 11 11  
let b = 10; log(b++, b); // 10 11  
let c = 10; log(--c, c); // 9 9  
let d = 10; log(d--, d); // 10 9
```

JAVASCRIPT OPERATOR CHEATSHEET

```
let a = 20; a += 5; // a = 25  
let b = 20; b -= 5; // b = 15  
let c = 20; c *= 5; // c = 100  
let d = 20; d /= 5; // d = 4  
let e = 20; e %= 5; // e = 0
```

```
(5 === 5) ? 'a' : 'b' // a  
  
3 > 2 && 1 < 2        // true  
3 > 2 || 5 < 2        // true  
!true                 // false
```

```
3 > 3 // false  
3 < 3 // false  
3 >= 3 // true  
3 <= 3 // true  
5 % 2 // 1
```

Cómo afrontamos la tarea de desarrollar un algoritmo?

Es como explicar algo

Pero a un niño, tienes que explicarle los pasos, las reglas.

- Hazlo en papel (que datos, variables necesitas, anotalo!)
- Divide todo el proceso en pequeñas partes.

!Extra

Ejercicios resueltos JS:

<https://www.youtube.com/watch?v=0m4e5C-n8H0>

Libro con algoritmos resueltos

<https://drive.google.com/file/d/13ch-qwrpUZDyTI9E5YX>

https://drive.google.com/file/d/1KnGp-Mm1A_jO/view?usp=sharing



Extra 2

Apuntes By Majo

https://drive.google.com/file/d/1qgVVMV_EeF5qwcREp38EzlyUbCKh2VO5/view?usp=sharing

Lessons JS

<https://sabe.io/classes/javascript>

JS Principiantes

<https://www.youtube.com/watch?v=RqQ1d1qEWIE>

