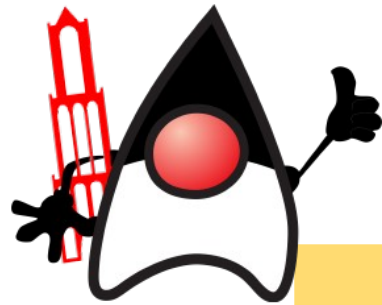
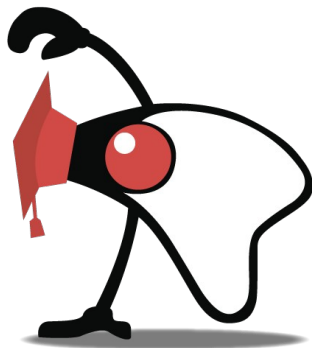
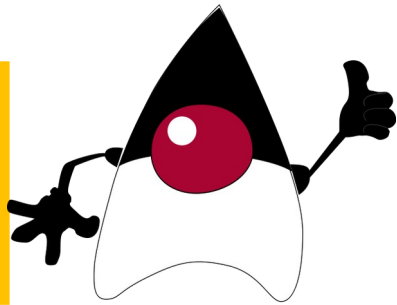




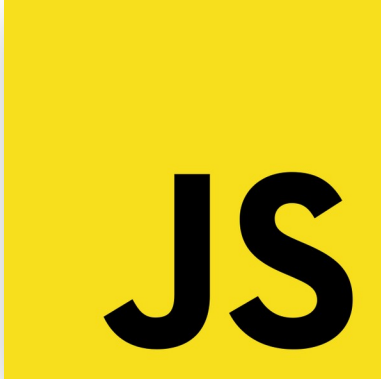
Curso FullStack Python

Codo a Codo 4.0



Javascript

Parte 2

A large yellow square with a subtle drop shadow, containing the letters 'JS' in a bold, black, sans-serif font.

JS

Operadores prefijo y posfijo

Los **afijos** se anteponen o se posponen en un nombre de una variable. Cuando hablamos de **prefijo** nos referimos a que se antepone a la variable y el **posfijo** se pospone. Se utilizan para realizar operaciones aritméticas, tanto para incrementar como para decrementar el valor de una variable.

Operador	Descripción	Ejemplo
i++	incremento posfijo	a=i++ primero a=i y después i=i +1
++ i	incremento prefijo	a=++i primero i=i +1 y después a=i
i--	decremento posfijo	a=i-- primero a=i y después i=i - 1
-- i	decremento prefijo	a=--i primero i=i - 1 y después a=i

[Ver ejemplo operadores-prefijo-posfijo.html](#)

Operadores de asignación

No solamente el = (igual) es un operador de asignación. Tenemos otras variantes:

Operador	Descripción	Equivale a
=	x = 3	x = 3
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

[Ver operadores-asignacion.html](#)

Operadores de comparación

Operador	Descripción
==	igual a
===	igual valor y tipo
!=	no igual a
!==	igual valor no tipo
>	mayor que
<	menor que
>=	mayor o igual que
<=	menor o igual que
?	operador ternario

[Ver operadores-comparacion.html](#)

Operadores lógicos

Operador	Descripción
&&	Y lógico (Conjunción)
 	O lógico (Disyunción)
!	NO lógico (Negación)

Negación !

Prop A	Resultado o
!True	False
!False	True

Conjunción &&

Prop A	Prop B	Resultado o
True	True	True
True	False	False
False	True	False
False	False	False

Disyunción ||

Prop A	Prop B	Resultado o
True	True	True
True	False	True
False	True	True
False	False	False

[Ver operadores-lógicos.html](#)

Operadores bit a bit

Los operadores de bits funcionan con números de 32 bits. Cualquier operando numérico de la operación se convierte en un número de 32 bits. El resultado se convierte de nuevo a un número de JavaScript.

Nro Decimal	Numero Binario
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111

Operador	Descripción	Ejemplo decimal	Ejemplo binario	Resultado binario	Resultado decimal
&	AND	5 & 1	0101 & 0001	0001	1
	OR	5 1	0101 0001	0101	5
~	NOT	~15	~00000000. ..00001111	11111111...1111 0000	-16
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	Desplazamiento a la izquierda de llenado cero	5 << 1	0101 << 1	1010	10
>>	Desplazamiento a la derecha	5 >> 1	0101 >> 1	0010	2

Estructuras de control | Condicionales

Cuando escribimos código Javascript, por defecto, el navegador leerá el script **de forma secuencial**, es decir, una línea detrás de otra, desde arriba hacia abajo. Por lo tanto, una acción que realicemos en la línea 5 nunca ocurrirá antes que una que aparece en la línea 3.

Al hacer un programa necesitaremos establecer **condiciones o decisiones**, donde busquemos que el navegador realice una **acción A** si se **cumple** una condición o una **acción B** si **no se cumple**. Este es el primer tipo de estructuras de control que encontraremos. Para ello existen varias estructuras de control:

Estructura de control	Descripción
If	Condición simple: Si ocurre algo, haz lo siguiente...
If/else	Condición con alternativa: Si ocurre algo, haz esto, sino, haz lo esto otro...
?:	Operador ternario: Equivalente a If/else , método abreviado.
Switch	Estructura para casos específicos: Similar a varios If/else anidados.

Para los próximos temas ver ejemplo condicionales (.html y.js)

Estructuras de control | If

El más conocido de estos mecanismos de estructura de control es el **if** (*si condicional*). Con él podemos indicar en el programa que se tome un camino sólo si se cumple la condición que establezcamos. Si no la cumple no se ejecutará nada y el programa seguirá su curso:

```
var nota = 7;  
console.log("Nota: ", nota);  
// Condición (si nota es mayor o igual a  
5)  
if (nota >= 5) {  
    console.log("¡Estoy aprobado!");  
}
```

JS

En este caso, como el valor de nota es superior a 5, nos aparecerá en la consola el mensaje «¡Estoy aprobado!». Sin embargo, si modificamos en la primera línea el valor de nota a un valor inferior a 5, no nos aparecerá ese mensaje.

Estructuras de control | If / else

Se puede dar el caso que queramos establecer una **alternativa** a una condición. Para eso utilizamos el **if** seguido de un **else**. Con esto podemos establecer una acción **A** *si se cumple la condición*, y una acción **B** *si no se cumple*.

Modificaremos el ejemplo anterior para mostrar también un mensaje cuando estamos suspendidos, pero en este caso, en lugar de mostrar el mensaje directamente con un `console.log` vamos a guardar ese texto en una nueva variable **calificación**:

```
JS
var nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");
// Condición
if (nota < 5) {
  // Acción A (nota es menor que 5)
  calificacion = "suspendido";
} else {
  // Acción B: Cualquier otro caso a A (nota es mayor o igual que 5)
  calificacion = "aprobado";
}
```

Estructuras de control | Condicionales

Operador ternario

El **operador ternario** es una alternativa de condicional **if/else** de una forma mucho más corta y, en muchos casos, más legible. Vamos a reescribir el ejemplo anterior utilizando este operador:

```
// Operador ternario: (condición ? verdadero : falso) JS
var calificacion = nota < 5 ? "suspendido" : "aprobado";
console.log("Estoy", calificacion);
```

Este ejemplo hace exactamente lo mismo que el ejemplo anterior. La idea del operador ternario es que podemos condensar mucho código y tener un if en una sola línea. Obviamente, es una opción que sólo se recomienda utilizar cuando son if muy pequeños.

```
if (nota < 5) { JS
  calificacion = "suspendido";
} else {
  calificacion = "aprobado";
}
```

*Esta es una copia del ejemplo anterior. En el operador ternario el **=** reemplazaría al **if**, mientras que el **?** actuaría como “entonces” acompañado de las acciones si se cumple la condición, y el **:** actuaría como el “si no” acompañado de las acciones si **no** se cumple la condición*

Estructuras de control | Condicionales

Condicional If múltiple

Es posible que necesitemos crear un condicional múltiple con más de 2 condiciones, por ejemplo, para establecer la calificación específica. Para ello, podemos anidar varios **if/else** uno dentro de otro, de la siguiente forma:

```
var nota = 7;  
console.log("He realizado mi examen.");  
// Condición  
if (nota < 5) {  
  calificacion = "Insuficiente";  
} else if (nota < 6) {  
  calificación = "Suficiente";  
} else if (nota < 8) {  
  calificacion = "Bien";  
} else if (nota <= 9) {  
  calificacion = "Notable";  
} else {  
  calificacion = "Sobresaliente";  
}  
console.log("He obtenido un", calificacion)  
;
```

JS

Las combinaciones son infinitas, por ejemplo un **if** dentro de otro, o un **if** dentro de un **else**, dependiendo la lógica de programación que quiero utilizar.

*Sin embargo, anidar de esta forma varios **if** suele ser muy poco legible y produce un código algo feo. En **algunos casos** se podría utilizar otra estructura de control llamada **switch**, que puede ser útil.*

Estructuras de control | Switch

```
var nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");
switch (nota) {
  case 10:
    calificacion = "Sobresaliente";
    break;
  case 9:
  case 8:
    calificacion = "Notable";
    break;
  case 7:
  case 6:
    calificacion = "Bien";
    break;
  case 5:
    calificacion = "Suficiente";
    break;
  case 4:
  case 3:
  case 2:
  case 1:
  case 0:
    calificacion = "Insuficiente";
    break;
  default:
    // Cualquier otro caso
    calificacion = "Nota errónea";
    break;
}
console.log("He obtenido un", calificacion);
```

JS

La estructura de control **switch** permite definir casos específicos a realizar en el caso de que la variable expuesta como condición sea igual a los valores que se especifican a continuación mediante los **case**.

- Este ejemplo no es exactamente equivalente al anterior, ya que funcionaría si sólo permitimos notas que sean **números enteros** (del 0 al 10, sin decimales). En el caso de que nota tuviera por ejemplo, el valor 7.5, mostraría Nota errónea.

Con los if múltiples podemos controlar casos de números decimales (comparamos rangos). Esto con switch *no se puede hacer* ya que está indicado para utilizar sólo con casos con valores concretos y específicos.

- Al final de cada caso es necesario indicar un **break** para salir del switch. En el caso que no se haga, el programa saltará al siguiente caso, aunque no se cumpla la condición específica.

Estructuras de control | If con && - If con ||

Podemos combinar el **if** con los operadores lógicos **&&** (*AND*) y **||** (*OR*) para lograr programas más potentes:

```
var altura = 0;
var edad = 0;
altura = parseFloat(prompt("Ingrese la altura"));
;
edad = parseInt(prompt("Ingrese la edad"));
if (altura > 1.30 && edad > 14) {
    console.log("Cumple con los requisitos");
} else{
    console.log("No cumple con los requisitos");
}
```

JS

IF COMBINADO CON && (AND):

Deben cumplirse **todas** las condiciones para que ocurra la parte *verdadera*. En el resto de los casos será *falsa*.

```
var color;
color = prompt("Ingrese el color del auto");
if (color == "Rojo" || color == "Verde") {
    console.log("El auto pertenece a la categoría A");
} else{
    console.log("El auto pertenece a la categoría B");
}
```

JS

IF COMBINADO CON || (OR):

Deben cumplirse **alguna** de las condiciones para que ocurra la parte *verdadera*. De no cumplirse ninguna será *falsa*.

Estructuras de control | Resumen

- **If:** Condición simple: Si ocurre algo, haz lo siguiente...

```
if (condicion) {  
    //bloque de codigo que se ejecuta si la condicion es verdadera  
}
```

JS

- **If/else:** Condición con alternativa: Si ocurre algo, haz esto, sino, haz esto otro...

```
if (condicion) {  
    //bloque de codigo que se ejecuta si la condicion es verdadera  
} else {  
    //bloque de codigo que se ejecuta si la condicion es falsa  
}
```

JS

- **Operador ternario:** Alternativa de condicional if/else en una sola línea.

```
var variable = condicion ? verdadero : falso;  
var calificacion = nota < 5 ? "suspendido" : "aprobado";
```

JS

Estructuras de control | Resumen

- **If múltiple:** Contempla la declaración else if para especificar una nueva condición si la primera condición es falsa.

```
if (condicion1) {  
    //bloque de codigo que se ejecuta si la condicion es verdadera  
} else if (condicion2) {  
    //  
    bloque de codigo que se ejecuta si la condicion1 es falsa y la condicion2 es verdadera  
} else {  
    //bloque de codigo que se ejecuta si la condicion1 es falsa y la condicion2 es falsa  
}
```

JS

- **Switch:** Contempla distintos valores que puede tomar una variable.

```
switch (var) {  
    case 1:  
        //instrucciones en caso de cumplirse el case 1  
        break;  
    case 2:  
    case 3:  
        //  
        instrucciones en caso de cumplirse el case 2 o 3  
        break;  
    default:  
        // Cualquier otro caso  
        break;  
}
```

JS

Estructuras de control | Resumen

- **If combinado con AND / OR:** Permite condiciones que deban darse a la vez (&&) o alternativamente (||).

o AND

```
if (condicion1 && condicion2) {  
    //  
    bloque de codigo que se ejecuta si todas las condiciones son verdaderas  
} else{  
    //bloque de codigo que se ejecuta si alguna de las condiciones es falsa  
}
```

JS

o OR

```
if (condicion1 || condicion2) {  
    //  
    bloque de codigo que se ejecuta si alguna de las condiciones es verdadera  
} else{  
    //bloque de codigo que se ejecuta todas las condiciones son falsas  
}
```

JS

Estructuras de control | Bucles e Iteraciones

Una de las principales ventajas de la programación es la posibilidad de crear bucles y repeticiones para tareas específicas, y que no tengamos que realizarlas varias veces de forma manual. Existen muchas formas de realizar bucles, vamos a ver los más básicos, similares en otros lenguajes de programación:

Tipo de bucle	Descripción
<code>while</code>	Bucles simples.
<code>for</code>	Bucles clásicos por excelencia.
<code>do..while</code>	Bucles simples que se realizan siempre como mínimo una vez.

Estructuras de control | Bucles e Iteraciones

Conceptos básicos sobre bucles

- **Condición:** Al igual que en los *if*, en los bucles se va a **evaluar una condición** para saber si se debe repetir el bucle o finalizarlo. Generalmente, si la condición es **verdadera**, se repite. Si es **falsa**, se finaliza.
- **Iteración:** Se llama así a cada **repetición** de un bucle. Por ejemplo, si un bucle repite una acción 10 veces, se dice que tiene 10 iteraciones.
- **Contador:** Muchas veces, los bucles tienen una variable que se denomina **contador**, porque *cuenta el número de repeticiones que ha hecho*, hasta llegar a un número concreto y finalizar. Dicha variable hay que inicializarla (crearla y darle un valor) antes de comenzar el bucle.
- **Incremento:** Cada vez que terminemos un bucle se suele realizar el incremento (o decremento) de una variable, generalmente de la denominada variable **contador**.
- **Bucle infinito:** Es lo que ocurre si en un bucle se nos olvida *incrementar la variable* contador o escribimos una *condición que nunca se puede dar*. El bucle se queda eternamente repitiéndose y el programa se queda «colgado».

Estructuras de control | While (mientras)

El bucle **while** se usa cuando el fin de la repetición depende de una condición. Vamos a repasar el siguiente ejemplo y todas sus partes, para luego repasar que ocurre en cada iteración del bucle:

```
i = 0; // Inicialización de la variable contador
// Condición: Mientras la variable contador sea menor de 5
while (i < 5) {
    console.log("Valor de i:", i);
    i = i + 1; // Incrementamos el valor de i
}
```

JS

Valor de i: 0
Valor de i: 1
Valor de i: 2
Valor de i: 3
Valor de i: 4

- Antes de entrar en el bucle **while**, se inicializa la variable **i** a **0**.
- Antes de realizar la primera **iteración** del bucle, comprobamos la **condición**.
- Si la condición es **verdadera**, hacemos lo que está dentro del bucle.
- Mostramos por pantalla el valor de **i** y luego incrementamos el valor actual de **i** en **1**.
- Volvemos al inicio del bucle para hacer una **nueva iteración**. Comprobamos de nuevo la **condición** del bucle.
- Cuando la condición sea **falsa**, salimos del bucle y continuamos el programa.

Es **muy importante** que esa condición en un momento pase de ser verdadera a falsa, sino tengo un loop infinito que en programación es un error grave.

Estructuras de control | While (mientras)

Una muestra paso a paso de las iteraciones de este primer ejemplo:

Iteración del bucle	Valor de i	Descripción	Incremento
Antes del bucle	i = undefined	Antes de comenzar el programa.	
Iteración #1	i = 0	¿(0 < 5)? Verdadero. Mostramos 0 por pantalla.	i = 0 + 1
Iteración #2	i = 1	¿(1 < 5)? Verdadero. Mostramos 1 por pantalla.	i = 1 + 1
Iteración #3	i = 2	¿(2 < 5)? Verdadero. Mostramos 2 por pantalla.	i = 2 + 1
Iteración #4	i = 3	¿(3 < 5)? Verdadero. Mostramos 3 por pantalla.	i = 3 + 1
Iteración #5	i = 4	¿(4 < 5)? Verdadero. Mostramos 4 por pantalla.	i = 4 + 1
Iteración #6	i = 5	¿(5 < 5)? Falso. Salimos del bucle.	

El bucle **while** es muy simple, pero requiere no olvidarse accidentalmente de la inicialización y el incremento (además de la condición).

```
i = 0;
while (i < 5) {
    console.log("Valor de i:", i)
    ;
    i = i + 1;
}
```

JS

Ver ejemplo while (.html y.js)

Estructuras de control | For (para)

El bucle **for** es quizás uno de los más utilizados en el mundo de la programación. En Javascript se utiliza exactamente igual que en otros lenguajes como Java o C/C++. Veamos el ejemplo anterior utilizando un bucle for:

```
// for (inicialización; condición; incremento) JS  
for (i = 0; i < 5; i++) {  
    console.log("Valor de i:", i);  
}
```

*En programación es muy habitual empezar a contar desde **cero**. Mientras que en la vida real se contaría desde **1 hasta 10**, en programación se contaría desde **0 hasta 9**.*

Valor de i: 0
Valor de i: 1
Valor de i: 2
Valor de i: 3
Valor de i: 4

Como vemos, la sintaxis de un **bucle for** es mucho más compacta y rápida de escribir que la de un **bucle while**. La primera vez puede parecer algo confusa, pero es mucho más práctica porque te obliga a escribir la **inicialización**, la **condición** y el **incremento** antes del propio bucle, y eso hace que no te olvides de estos tres puntos fundamentales.

Estructuras de control | For (para)

El bucle **for** se suele usar cuando **se conoce de antemano** cuantas repeticiones se tienen que hacer.

Ejemplo: Mostrar por pantalla los números enteros del 1 a 10.

```
for (var i=1; i<=10; i++) {  
    console.log(i);  
}
```

JS

1

2

3

4

5

6

7

8

9

10

Ejemplo: Mostrar por pantalla los múltiplos de 2 hasta 100.

```
for (var i=2; i<=100; i+=2)  
    console.log(i);  
}
```

JS

2

4

6

8

...

94

96

98

100

Ver ejemplos for (.html y.js)

Estructuras condicionales | Ejercicios

1. Leer la hora por pantalla (número entero) y saludar en la consola según el horario: buenos días o buenas noches. Considerar que el día dura hasta 19 hs.
2. Modificar el ejercicio anterior incorporando que salude: buenos días (14 hs), buenas tardes (19 hs) o buenas noches. Además el saludo debe aparecer en el `<body>`.
3. Crear un programa que, según el número de día de la semana solicitado, diga “buen lunes”, “buen martes”, etc. en el `<body>`
4. Crear un programa que permita definir la estación del año de acuerdo a un día y un mes dados.

Estructuras repetitivas | Ejercicios

1. Escribir la tabla del 2 en el body. Por ejemplo:
 - $2 \times 1 = 2$
 - $2 \times 2 = 4$
 - $2 \times 3 = 6$
 - $2 \times 4 = 8$
2. Leer números enteros y mostrar la tabla de multiplicar de ese número en la consola, hasta que ingrese un 0.

Material complementario y ejercicios

Curso JS: https://www.youtube.com/playlist?list=PLhSj3UTs2_yVC0iaCGf16glrrfXuiSd0G
(lista de reproducción)

Los videos que recomiendo luego de las primeras dos clases de JS son:

- 1. Introducción
- 2. Variables
- 3. Tipos de Datos
- 6. Condicionales
- 7. For
- 8. While

Bucle FOR: https://www.w3schools.com/js/js_loop_for.asp

Bucle WHILE: https://www.w3schools.com/js/js_loop_while.asp

Ejercicios

- Del archivo **“Actividad Práctica - JavaScript Unidad 1”** están en condiciones de hacer los ejercicios: 6 a 8 y 13 a 25. Los ejercicios **NO** son obligatorios.