Programación Web Básica.

UNQ

Prof. Jorge Nicolás Terradillos
Prof. Fernando Blanco

En la clase de hoy...

-Variables y ámbitos

-Valores y operadores lógicos

-Condicionales

-Bucles y contadores

-Objetos

01

Variables y ámbitos

Variables VAR

Además de let, hay otras formas de definir variables en JavaScript, por ejemplo **VAR**

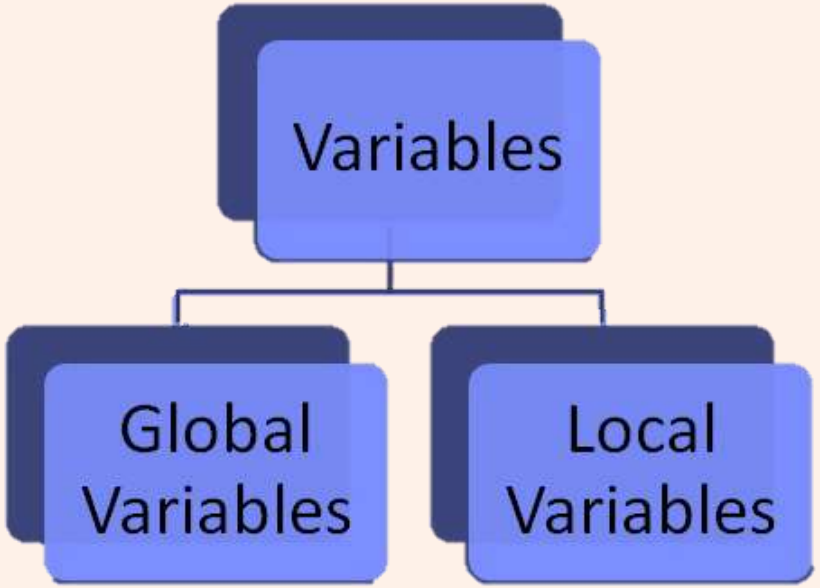
```
var x = 5;
var y = 6;
console.log(x+y); //nos da 11
```

La diferencia entre las variables **VAR** y **LET** tiene que ver con el ámbito

Ambito

El ámbito se refiere a en qué parte del código la variable es valida y se puede utilizar

Las variables **VAR** se pueden usar y redefinir en cualquier parte ya que su ámbito es global, en cambio con **LET** dependiendo de dónde se escribió



Ambito global

Puedo acceder a su valor desde cualquier lugar del programa:

```
var miVariableGlobal = 15;
console.log(miVariableGlobal);
function miFuncion() {
        console.log(miVariableGlobal);
}
miFuncion();
```

Ambito local

Puedo acceder a su valor solo desde el bloque de código dónde se escribió:

```
function mostrarMiNombre() {
          let miNombre = “Jorge”;
          console.log(miNombre);
}
mostrarMiNombre(); // devuelve la palabra Jorge
console.log(miNombre); //arroja un error porque miNombre
                       //se usa solo dentro de la función
```

02 Valores lógicos

Valores lógicos

Los valores booleanos son valores que representa en la lógica el verdadero (true) y el falso (false). Por defecto, estos son los dos únicos tipos de valores booleanos posibles

Otros valores como null o undefined son interpretados como false

Sintaxis de booleanos

```
console.log(True); //error
console.log(true);
console.log(false);


/*true y false para que JS los tome tienen que estar escritos completamente en minúsculas*/
```

Comparación

En JS, la comparación es una operación que nos devuelve valores verdaderos o falsos, por ejemplo decir que 2==1 (dos equivale a uno) nos devuelve el valor falso

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

Comparación

En JS, la comparación es una operación que nos devuelve valores verdaderos o falsos, por ejemplo decir que 2==1 (dos equivale a uno) nos devuelve el valor falso

OPERADORES LÓGICOS Y RELACIONALES	DESCRIPCIÓN	EJEMPLO
==	Es igual	a == b
===	Es estrictamente igual	a === b
!=	Es distinto	a != b
!==	Es estrictamente distinto	a !== b
<, <=, >, >=	Menor, menor o igual, mayor, mayor o igual	a <=b
&&	Operador and (y)	a && b
||	Operador or (o)	a || b
!	Operador not (no)	!a

```
console.log(5 == 5); //true
console.log("hola" == "hola"); //true
console.log("hola" == "Hola"); //false
console.log("hola" == "Javascript"); //false

console.log([1,2,3] == [1,2,3]); /*false porque NO
SE PUEDEN COMPARAR ARRAYS de forma directa*/
```

Igualdad estricta

Si quiero probar igualdad estricta (que sean exactamente iguales):

Voy a usar ‘===’

```
console.log(9 == ‘9’);  //true
console.log(9 === ‘9’); // false . porque no es el mismo tipo de dato
// el operador == transforma los datos en un mismo tipo de dato.
```

Desigualdad

El operador != comprueba si dos elementos NO son iguales

```
console.log(5!=8); //true
console.log(5!=5); //false
console.log("Hola"!="Hola"); //false
console.log(5!="hola"); //true
```

Mayor y menor

Los operadores matemáticos de mayor y menor también dan resultados lógicos

```
console.log(6 > 5); //true
console.log(8 < 10); //true
console.log(9 < 5); //false
console.log("B" > "A"); //true
```

También se pueden usar menor o igual (<=) y mayor o igual (>=)

Actividad

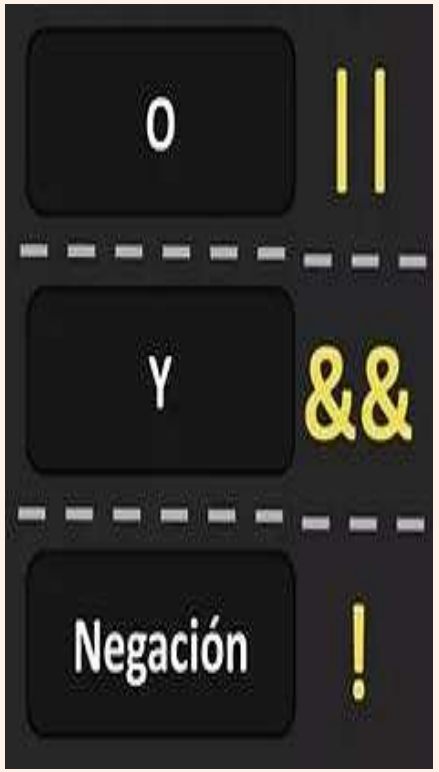
-Punto 1: Comparando

Tomando el siguiente array:

```
let ejercicio[5,6,6,7,8,9,"5","6","7"];
```

¿Qué resulta de comparar el primer y el séptimo elemento? Probar con igualdad estricta.
¿Qué resulta de preguntar si el tercer elemento es mas grande que el segundo elemento?
¿Es el último elemento menor al cuarto elemento?
¿Qué sucede si se quiere comparar más de un elemento a la vez?

Operadores lógicos



Los operadores lógicos comparan varios valores booleanos y devuelven respuestas booleanas

Para entender los resultados hay que tener en cuenta todos las posibles combinaciones de valores lógicos

Operadores lógicos

Los operadores lógicos comparan varios valores booleanos y devuelven respuestas booleanas

Para entender los resultados hay que tener en cuenta todos las posibles combinaciones de valores lógicos, asi que supongamos que “x” e “y” son dos posibles valores lógicos

x	y
true	true
true	false
false	true
false	false

Negación

Operador NOT (!): negación, invierte el valor booleano

x	!x
true	false
false	true

Conjunción

x	y	X && y
true	true	true
true	false	false
false	true	false
false	false	false

Operador de conjunción (**&&**), también llamado **AND** da verdadero solo cuando ambos valores son verdaderos

Disyunción

Operador de disyunción (||), también llamado **OR** da verdadero cuando al menos un valor es verdadero

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

Disyunción

Operador de disyunción (||), también llamado **OR** da verdadero cuando al menos un valor es verdadero

x	y	x || y
true	true	true
true	false	true
false	true	true
false	false	false

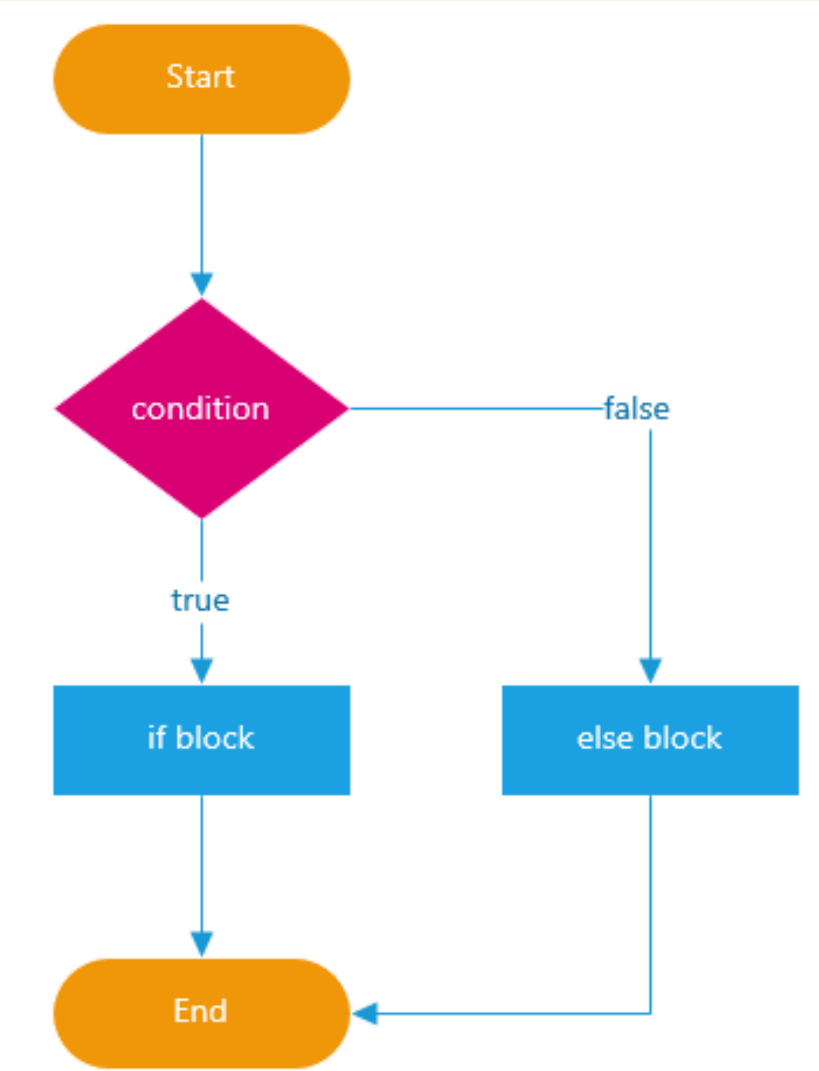
```
let a = 20;
console.log((a > 5) && (a <10)); //true
console.log((a > 30) || (a <10));//false
a = 5;
console.log((a > 5) && (a <10)); //false
a = 1;
console.log((a > 5) && (a <10)); //false
console.log((a > 5) || (a != 3)); //true
```

03

Condicionales

Condicionales

Como en todo lenguaje de programación, JavaScript permite crear código que pueda tener más de una resolución, según una **condición** (que será un valor booleano)
Esto se hace con la palabra clave “**if**” seguida del bloque para el caso true y la palabra clave “**else**” para el caso false



//los condicionales permiten decidir si un bloque de código se ejecuta dependiendo de una condición.

```
if (condicion) {
    //si la condicion es verdadera se ejecuta
    console.log('la condicion es verdadera');
}
```

Sintáxis

```
//siempre dentro del paréntesis se evalua algo que puede ser verdadero o falso.
let x = 5;

lf (x > 2) {
    //si la condicion es verdadera se ejecuta
    console.log('la condicion es verdadera');
}
/*si la condicion NO es verdadera, el código NO se ejecuta*/
```

¿Y si la condición es falsa? No se ejecuta el bloque del “if”, pero puedo hacer que se ejecute un bloque diferente con “else”

```
let estacion = “verano”;

lf (estacion == “invierno”) {
    //si la condicion es verdadera se ejecuta
    console.log(‘la condicion es verdadera’);
}
else {
    console.log(‘la condicion es falsa’);
}
```

Clausula elseif

Ejemplo: queremos saber si un número es igual, mayor o distinto a 100 ¿Qué hacemos con el if? Lo más fácil es anidar

```
function mayorACien(numero){
  if (numero>100){
      console.log(“el número es mayor a 100”);
   }
   else if (numero==100){
     console.log(“el numero es igual a 100”)}
   else{
     console.log(“el número es menor a 100”);
}
```

Otro ejemplo: ¿qué hace este código?

```
function clasificarValor(valor) {
    if (valor % 2 == 0){
        console.log('valor divisible por 2');
    } else if  (valor % 3== 0){
        console.log('valor divisible por 3');
    }
      else {
  console.log('no es divisible por  2 ni 3');
}
```

Actividad

-Punto 1: ¿Es par o impar?

Crear una función llamada es esPar, a la que le pasamos un número por parametro y nos devuelve true si el número es par o false si no es par

-Punto 2: ¿Vacio?

Crear una función llamada vacio, a la que le pasamos un array y nos dice true en caso de que el array esté vacio. BONUS: si le pasamos un valor que no sea número, la función deberá responder «esto no es un arreglo».

04

Bucles lógicos

Repetición

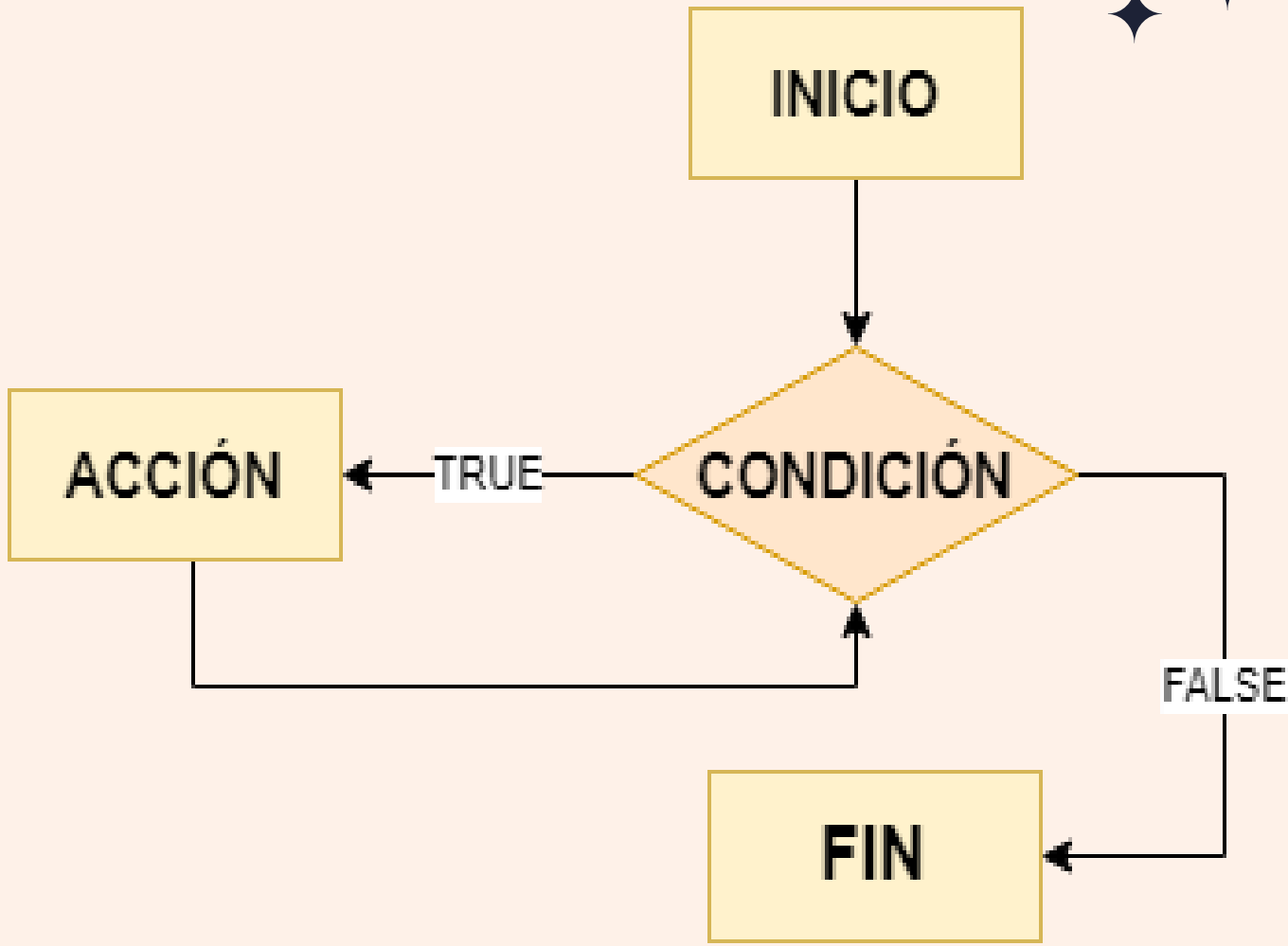
¿Cómo podríamos hacer una función que imprimiera varias veces un mismo valor?

```
function imprimirMiValor(miValor){
  console.log(miValor);
  console.log(miValor);
  console.log(miValor);
...
}
```

Tenemos que diseñar un **bucle** que se repita tantas veces como lo necesitamos

Bucles

En JavaScript tenemos varias estructuras de bucles, entre ellas el “while”, donde dada una condición booleana (como con el if), se ejecuta un bloque de código mientras esa condición siga siendo verdadera



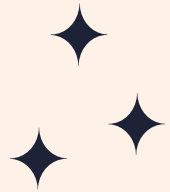
```
function imprimirMiValor(miValor){
  while (true){
    console.log(miValor);
  }
}
```

Aquí entraremos a un bucle infinito ya que la condición true siempre será true

Contadores

Para detener el bloque iniciamos un contador de las veces que se repite y cada vez que se repite, este contador aumenta en 1

```
function imprimirMiValor(miValor){
    let contador = 0;
    while (contador<5){   //hace que se detenga cuando llega a 5
        console.log(miValor);
        contador++;
    }
}
```

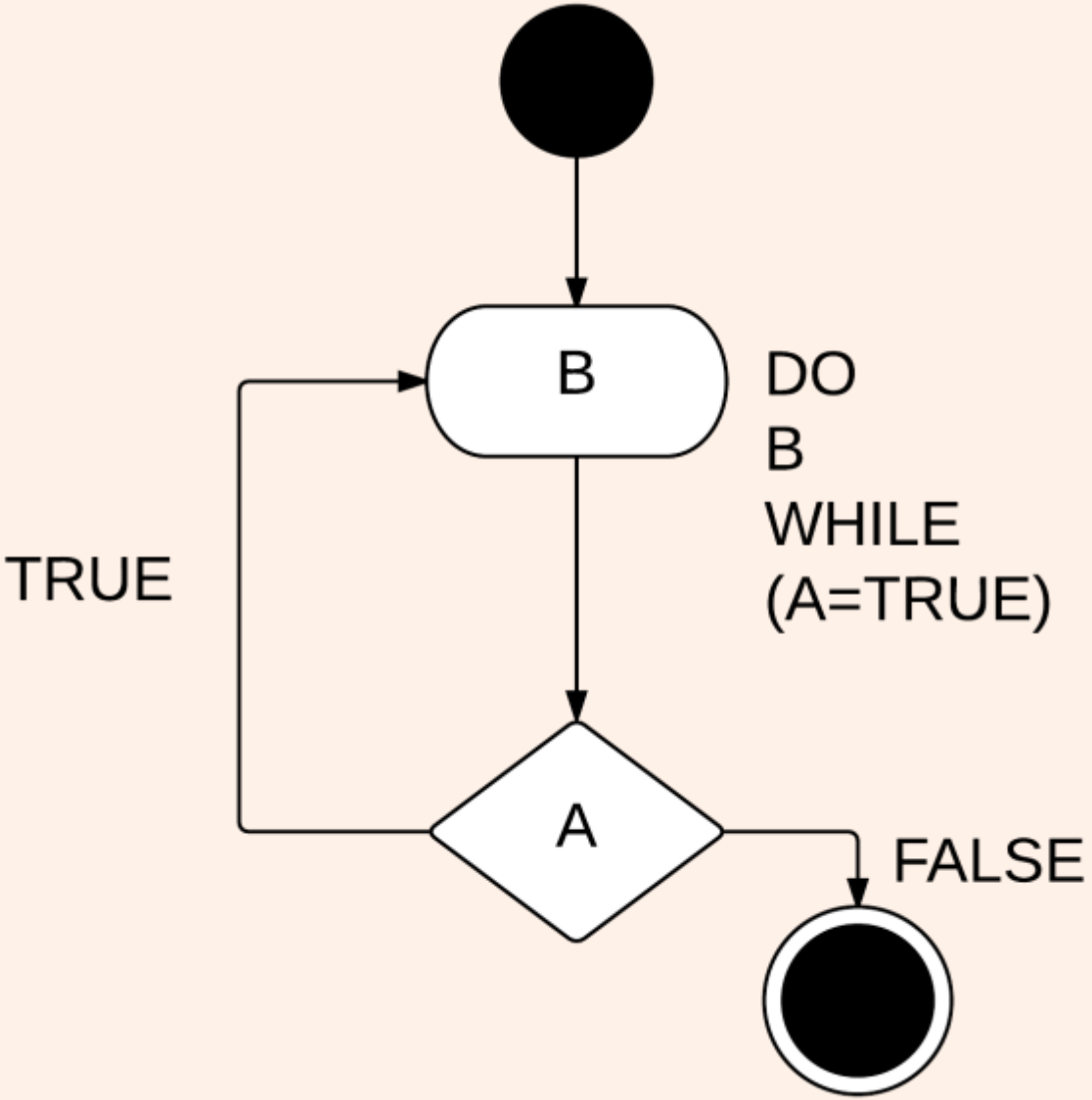


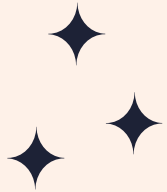
```
function imprimirXVeces(miValor,cantidad){
  let contador = 0;
  while (contador<cantidad){
//podemos pasar la cantidad de repeticiones por parametro
      console.log(miValor);
      contador++;
    }
}
```

Do While

También se puede usar la sintaxis “do {bloque de código} while (condición)”

La diferencia es que así primero ejecuta el código y después revisa si cumple la condición





```
let result = “”;
let i = 0;
do {
  i = i + 1;
  result = result + i;
}
while (i < 5);
console.log(result); // resultado: "12345"
```

Actividad

-Punto 1

Crear una función “imprimirArray” que, tomando un array, imprima todos los elementos que contiene.

-Punto 2

Crear una función “caracteresTotales” que, tomando un array de strings, nos sume que tan largo son los strings que contiene en total

-Punto 3

Crear una función “limpiarDeImpares” que, tomando un array de números, lo retorne sin los números impares (pueden usar una función auxiliar que verifique si un numero es impar)

Resumiendo

Variables	Booleanos	Operadores	If	Bucles
La variable LET que venimos usando es variable local, VAR es variable global	Los valores lógicos true y false nos permiten analizar y comparar otros valores	También podemos combinar enunciados lógicos usando operadores como NOT, AND y OR	La declaración condicional IF y ELSE permite que el código responda según valoraciones lógicas	Con enunciados lógicos podemos crear código que se repita según condiciones

Eso es todo

Se escuchan dudas, quejas y propuestas

Fin de la clase