

Introducción al lenguaje JavaScript

Introducción a ciclos y funciones

Reconocer las características fundamentales del lenguaje Javascript para el desarrollo web.

Utilizar variables simples y sentencias condicionales para el control del flujo de un algoritmo que resuelve un problema simple acorde al lenguaje Javascript.

{desafío}
latam_

- Unidad 1:
Introducción al lenguaje JavaScript
- Unidad 2:
Funciones y Ciclos
- Unidad 3:
Arrays y Objetos
- Unidad 4:
APIs



Te encuentras aquí



¿Qué aprenderás en esta sesión?

- *Codificar una rutina simple en Javascript a partir de un diagrama de flujo para dar solución a un problema.*
- *Esbozar un diagrama de flujo utilizando la notación requerida para representar un algoritmo condicional.*

De acuerdo a lo aprendido
en la sesión anterior:

¿Si quisiéramos
incrementar en 1 el valor de
una variable a podríamos
escribir `a += 1` y también
`a++`?



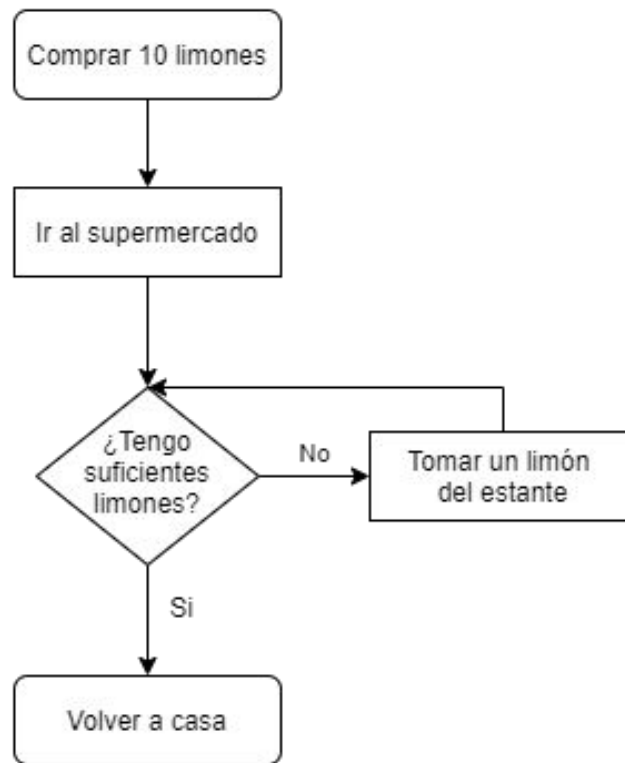
¿El código dentro de un if se ejecutará si y sólo si la expresión a evaluar es verdadera?



/* Diagrama de flujo con ciclos */

Diagrama de flujo con ciclos

- Es posible representar un ciclo utilizando un diagrama de flujo.
- El factor clave es un nuevo uso de los condicionales en esta representación, es decir, mediante el uso de condicionales se repite cierta acción una y otra vez hasta que se cumpla el objetivo o propósito y así pasar al siguiente paso o finalizar el proceso.




/* Estructura de ciclos */

Estructuras de Ciclos

Ciclo for

- El Ciclo for nos permite realizar acciones que se repitan mientras se mantiene una condición. Es una estructura que permite repetir nuestro código cuantas veces sea necesario.
- Su definición formal es de la siguiente manera:

```
for (inicializacion; condicion; actualizacion) {  
    ...  
}
```



Mientras la condición indicada se cumpla, se repetirá la ejecución de las instrucciones definidas dentro de las llaves del for. Además, después de cada repetición, se actualiza el valor de las variables que se utilizan en la condición.

Estructuras de Ciclos

Ciclo for

```
for(var i = 0; i < 5; i++) {  
    alert("Esto es un mensaje dentro del ciclo");  
}
```

`i = 0`

Es donde nosotros vamos a declarar los valores iniciales de las variables que controlan la repetición:

`i < 5`

Es donde establecemos cuantas veces se repetirá el ciclo y donde se decide si continúa o se detiene la repetición.

`i++`

Es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

Demostración - “Ciclo for”



Ejercicio guiado

Ciclo for

Desarrollar y mostrar la tabla de multiplicar para el número 6 utilizando el ciclo for. Para ello, sigamos los siguientes pasos:

- **Paso 1:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un ciclos.js.
- **Paso 2:** En el index.html escribir la estructura básica de un documento HTML.



Ejercicio guiado

Ciclo for

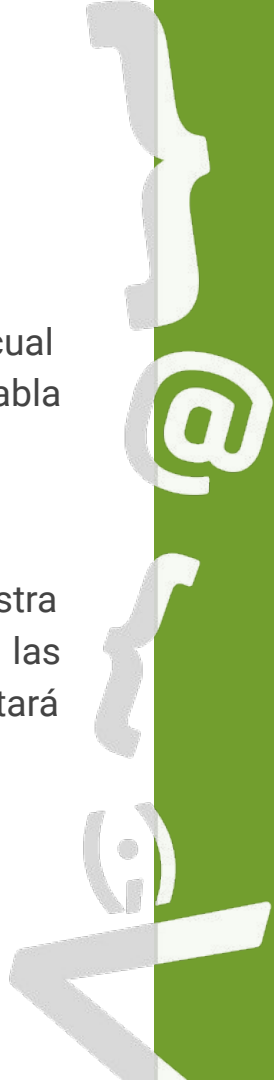
- **Paso 3:** En el archivo script.js se declara una variable que llamamos “número”, la cual almacena la tabla de multiplicar que queremos imprimir en pantalla, en este caso la tabla del 6.

```
var numero = 6;
```

- **Paso 4:** En nuestro ciclo for, iniciamos una variable llamada “i” con un valor de 1, nuestra condición dice que si “i” es menor o igual que 10 automáticamente pasa a las instrucciones que tenemos dentro del for { }. En otras palabras, el ciclo se ejecutará mientras la “i” sea menor o igual a 10.

```
var numero = 6;

for(i = 1; i <=10; i++) {
}
```



Ejercicio guiado

Ciclo for

- **Paso 5:** Dentro del for, necesitamos declarar otra variable con el nombre de “resultado”, esta variable almacena la operación matemática de multiplicación entre el valor de “i” en ese momento por el valor de la variable “número” que vale 6.

```
var numero = 6;

for(i = 1; i <=10; i++) {
    var resultado = numero * i;
    document.write(numero + " " + " x " + " " + i + " = " +
    resultado + "<br>");
}
```

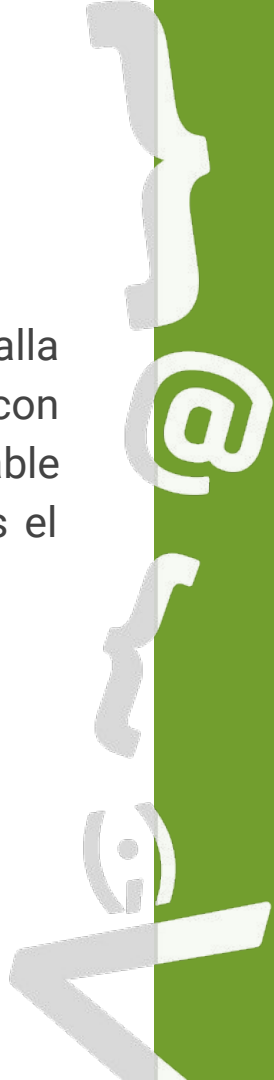


Ejercicio guiado

Ciclo for

- **Paso 6:** Finalmente, con el método “document.write();” imprimimos en pantalla el valor de la variable “número”, concatenamos con un espacio vacío, luego con el signo de multiplicación, seguido por el valor de ese momento de la variable “i”, el signo “=” y posteriormente el valor de la variable “resultado”, que es el resultado de la multiplicación.

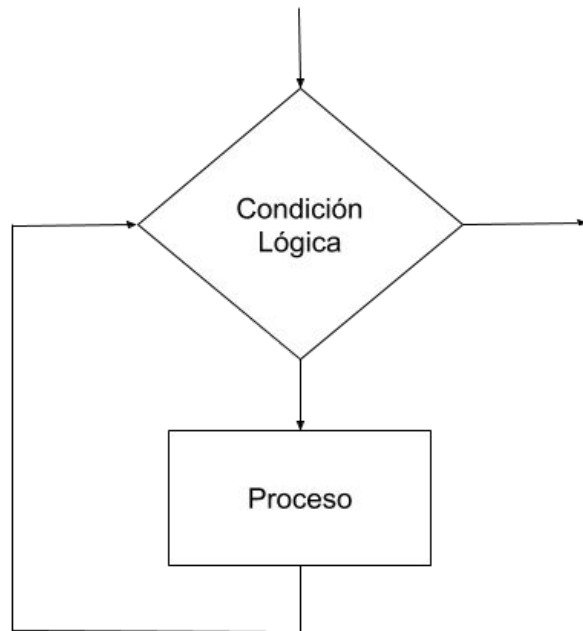
```
var numero = 6;  
  
for(i = 1; i <=10; i++) {  
    var resultado = numero * i;  
    document.write(numero + " " + " x " + " " + i + " = " +  
    resultado + "<br>");  
}
```



Estructura de ciclos

Ciclo while

- Al igual que el ciclo for, el ciclo while son ciclos de repetición de instrucciones. Pero que los podemos usar en distintos casos según estimemos conveniente. Ahora, en el caso de un diagrama de flujo, el ciclo while se dibujaría de la siguiente manera:



Estructura de ciclos

Ciclo *while*

- Partiendo del diagrama de flujo anterior, el bucle o ciclo while escrito en el lenguaje de programación JavaScript, se escribiría de la siguiente manera:

```
while (condición) {  
    ...  
}
```

Mientras se cumpla una condición y sea verdadera se ejecutan las instrucciones que estén dentro del ciclo.

Demostración - “Ciclo while”



Ejercicio guiado

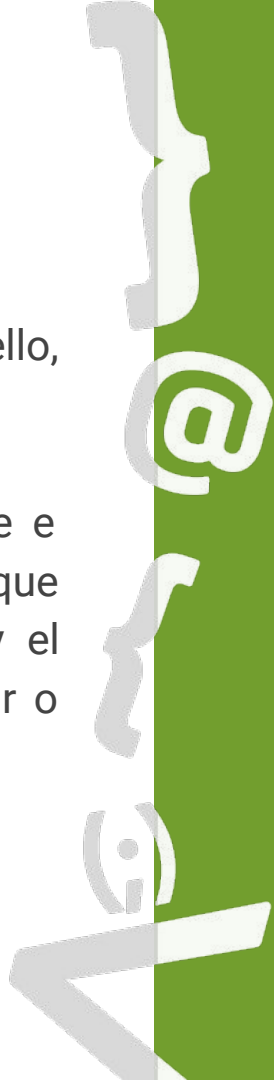
Ciclo while

Imprimir los número pares que correspondan desde el cero (0) hasta el 20, para ello, sigamos los siguientes pasos:

- **Paso 1:** En el archivo ciclos.js, debes armar la estructura del ciclo while e inicializar una variable doble propósito, el primero será el número que deseamos que vaya aumentando a medida que se repitan los ciclos y el segundo será la condición que evaluará el ciclo para saber si debe seguir o salir del ciclo.

```
var num = 0;

while (num <= 20) {
}
```



Ejercicio guiado

Ciclo while

- **Paso 2:** Ya dentro del ciclo, se toma la variable inicializada fuera del ciclo while, se le suma el número 2 y se guarda sobre ella misma, lo que permite ir almacenando y mostrando solamente los números pares. Quedando el código:

```
var num = 0;

while (num <= 20) {
    document.write(num + "<br>");
    num = num + 2;
}
```



/* Diferencias entre un ciclo y un if */

Diferencias entre un ciclo y un if



Ciclos

- Un ciclo es un conjunto de instrucciones que se repiten de manera simultánea mientras cierta condición sea verdadera
- Se utiliza cuando se requiere verificar múltiples veces una condición.



If

- Es la única comprobación de una comparación lógica.
- Se utiliza, por lo tanto cuando se requiere verificar una única vez una condición

**/* Diferencias entre iteradores,
contadores y acumuladores */**

Diferencias entre iteradores, contadores y acumuladores

Iteradores

- Un iterador corresponde a aquella variable utilizada en cada etapa del ciclo para verificar la condición de término.
- Corresponde a una variable que indica en qué etapa del ciclo se encuentra el programa. En el siguiente ejemplo, la variable *i* es el iterador.

```
for(i=0; i<10; i++){  
    // Código del  
    ciclo for  
}
```


Diferencias entre iteradores, contadores y acumuladores

Contadores

- Es una variable que registra el número de ocurrencias de un suceso en particular.
- No interfiere en la condición de término de un ciclo.
- Por ejemplo, si se quisiera saber cuántos números pares hay entre 10 y 100, es necesario utilizar un contador para saber la cantidad precisa de números pares.

```
contador = 0; // Los contadores por lo general se inician en 0
for(i=10; i<=100; i++){
    if(i % 2 === 0 ){ // se divide mediante el módulo del número
        contador++; // es igual que escribir: contador = contador + 1;
    }
}
alert(`Hay ${contador} números pares entre 10 y 100`);
```

Diferencias entre iteradores, contadores y acumuladores

Acumuladores

- Sirven para acumular el valor de varias variables en una sola.
- Son útiles para situaciones en las que no basta con contar ciertos eventos, sino que hay que considerar su contenido.
- Por ejemplo, si se quisiera saber cuál es la suma de todos los números entre el 1 y el 100 (incluyendo ambos), es necesario utilizar un acumulador.

```
acumulador = 0; // Los contadores por lo general se inician en 0
for(i=1; i<=100; i++){
    acumulador += i; // es igual es escribir: acumulador = acumulador + i
}
alert(`La suma total acumulada de los números entre el 1 y 100 es:
${acumulador} `);
```

Demostración - “Iteradores, contadores y acumuladores”



Ejercicio guiado

Iteradores, contadores y acumuladores

- Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números impares comprendidos entre el 0 y el 50 (incluidos ambos) implementando el ciclo for. Es decir, se deben sumar solamente los números impares entre el 1 y el 50. Indicar cuánto fue la suma y a su vez cuántos números se sumaron.


$$1+1=2$$

Ejercicio guiado

Iteradores, contadores y acumuladores

- **Paso 1:** Implementar un contador y un acumulador. Luego de esto se realiza el ciclo for con la configuración necesaria para que el ciclo se repita las 50 veces solicitadas, quedando de la siguiente forma:

```
var contador = 0; // inicio del contador en 0
var acumulador = 0; // inicio del acumular en 0

for(i=1; i<=50; i++){
}
```



Ejercicio guiado

Iteradores, contadores y acumuladores

- **Paso 2:** Activamos la lógica correspondiente dentro de la estructura del for. Como necesitamos saber cuales son los números impares, entonces podemos implementar una estructura condicional como el if y dentro de su condición preguntar si el número es impar mediante la división del módulo (%).

```
var contador = 0; // inicio del contador en 0
var acumulador = 0; // inicio del acumular en 0

for(i=1; i<=50; i++){
    if(i % 2 != 0 ){ // se divide mediante el módulo del número
        contador++; // es igual que escribir: contador = contador + 1;
        acumulador += i; //es igual a escribir: acumulador = acumular + i;
    }
}

alert(`Hay ${contador} números impares entre 1 y 50 y la suma acumulada
de ellos es: ${acumulador}`);
```



/* Funciones en JavaScript */

Funciones

- Son fragmentos de código que podemos reutilizar cuantas veces queramos dándoles dinamismo para que éstas ejecuten o hagan cosas por nosotros.
- Las funciones son como un libro de recetas, tenemos datos de entrada, las instrucciones que serían la función y el resultado esperado serían los datos de salida.



Funciones

- Las funciones son mini programas que cumplen una “función” determinada.
- La principal potencia de esto es que podemos reutilizar este mini programa como y cuando queramos, todas las veces que sea necesario.
- La estructura de una función es la siguiente:

```
function nombre  
(parametros) {  
    // instrucciones  
}
```

**/* Dividir problemas en subprocesos
Identificar la entrada y salida */**

Demostración - “Entrada y salida de la función”

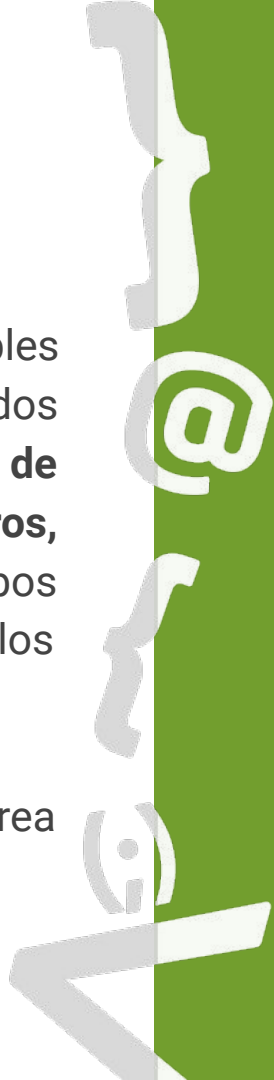


Ejercicio guiado

Entrada y salida de la función

Se le solicita al usuario ingresar dos números enteros y almacenarlos en variables distintas (**valores de inicio**), luego mediante el uso de una función, enviar los dos números ingresados a la función (**argumento, llamado de la función y valores de entrada**), para ser recibidos (**parámetros**) y procesados (**sumando ambos números, procesamiento de valores**) y finalmente retornar el valor final de la suma de ambos números (**retorno de información y valores de salida**). Para ello, sigamos los siguientes pasos:

- **Paso 1:** Crea una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.



Ejercicio guiado

Entrada y salida de la función

- **Paso 2:** En el index.html debes escribir la estructura básica de un documento HTML.
- **Paso 3:** En el archivo script.js debes armar la estructura de la función e inicializar dos variable donde se almacenarán los datos ingresado por el usuario, en este caso los podemos llamar “num1” y “num2”, luego mediante la función “prompt()” solicitamos al usuario ingresar los números. Quedando de la siguiente forma:

```
var num1 = prompt("Ingresa primer número: ");  
var num2 = prompt("Ingresa segundo número: ");  
  
function suma (num1, num2) { // recibimos los dos números como parámetros  
  de la función  
}
```



Ejercicio guiado

Entrada y salida de la función

- **Paso 4:** Activamos la lógica correspondiente dentro de la función, llevando los datos ingresados por el usuario a numéricos mediante la instrucción `parseInt`, luego realizando la suma de ambos números y almacenando el resultado en otra variable, finalmente retornamos el valor de la suma final.

```
var num1 = prompt("Ingresa primer número: ");  
var num2 = prompt("Ingresa segundo número: ");
```

```
function suma (num1, num2) { // recibimos los  
  dos números como parámetros de la función  
    var num1 = parseInt(num1);  
    var num2 = parseInt(num2);  
    var resultadoSuma = num1 + num2;  
    return resultadoSuma; // permite retornar de  
    la función con un valor en específico  
  }  
document.write(suma(num1, num2)); //enviamos  
los dos números ingresados por el usuario como  
argumentos de la función
```

**/* Aplicar funciones para resolver
problemas de carácter repetitivo */**

Demostración - “Funciones en JavaScript”



Ejercicio guiado

Funciones en JavaScript

Ahora realizaremos el mismo **Ejercicio guiado: Entrada y salida de la función**, pero utilizaremos un mini formulario para poder segmentar y dejar la página siempre disponible para que el usuario interactúe con ella sin la necesidad de recargar o refrescar la página, es decir, el usuario deberá ahora ingresar los números en las entradas indicadas del formulario y lo haremos de la siguiente manera.



Ejercicio guiado

Funciones en JavaScript

- **Paso 1:** Escribir este código en el archivo "index.html".

```
<!DOCTYPE html>
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <title>Funciones</title>
</head>
<body>
<form action="">
  <input type="text" id="numero1">
  <input type="text" id="numero2">
  <input type="button" value="Sumar" onclick="alert('El
resultado es: ' + suma() );"> <!-- esta es una forma
antigua de activar funciones con JavaScript desde el
elemento HTML como atributo. -->
</form>

<script type="text/JavaScript" src="script.js"></script>
</body>
</html>
```



Ejercicio guiado

Funciones en JavaScript

- **Paso 2:** Escribir la estructura de la función, la cual, no tendrá parámetros debido a que captamos los valores ingresados por el usuario directo desde el formulario mediante las instrucciones “`getElementById("id").value`” que nos permite obtener la información de un elemento HTML del documento y luego extraer el valor mediante la propiedad “`value`”, y los almacenaremos en una variable:

```
function suma () {  
  var numero1 = parseInt(document.getElementById("numero1").value);  
  var numero2 = parseInt(document.getElementById("numero2").value);  
}
```



Ejercicio guiado

Funciones en JavaScript

- **Paso 3:** Finalmente retornamos el resultado de la suma:

```
function suma () {  
  var numero1 =  
  parseInt(document.getElementById("numero1").value);  
  var numero2 =  
  parseInt(document.getElementById("numero2").value);  
  var resultadoSuma = numero1 + numero2;  
  return resultadoSuma;  
}
```

- Por lo que el resultado al ingresar los valores de 6 y 7 en el formulario y hacer un clic sobre el botón de sumar sería:

El resultado es: 13

Aceptar

Resumen

- Hemos podido profundizar aspectos relacionados a la implementación de algoritmos que resuelven problemas recurrentes, donde se requieren estructuras de control repetitivas que nos permitan manipular ciclos en el código.
- Como te pudiste dar cuenta, vimos lo esencial y la base del lenguaje de programación JavaScript, lo que nos ayudará en nuestra carrera para así lanzarnos con nuestra imaginación y hacer lo que queramos en nuestras páginas web.

¿Existe algún concepto que no
hayas comprendido?

Volvamos a revisar los
conceptos que más te hayan
costado antes de seguir
adelante





Próxima sesión...

- *Revisaremos y realizaremos el material sincrónico que corresponde al **Desafío evaluado**, el cual te permitirá medir los conocimientos aprendidos en estas sesiones.*

{desafío}
latam_

*Academia de
talentos digitales*

