

***RECUERDA PONER A GRABAR LA
CLASE***





¿DUDAS DEL ON-BOARDING?

MIRALO AQUI



Clase 03. JAVASCRIPT

CICLOS/ITERACIONES



OBJETIVOS DE LA CLASE

Entender:

- ¿Qué es un ciclo o bucle y cómo nos permite repetir operaciones similares fácilmente?
- ¿Qué tipos de ciclos podemos emplear y cuáles son sus diferencias ?
- ¿Cómo combinar operadores lógicos, ciclos y funciones para resolver cada problema?

CRONOGRAMA DEL CURSO

Clase 2



Control de flujos



EJEMPLOS EN VIVO



CREAR UN ALGORITMO CON
UN CONDICIONAL

Clase 3



Ciclos/Iteraciones



EJEMPLOS EN VIVO



CREAR UN ALGORITMO
UTILIZANDO UN CICLO

Clase 4



Funciones



EJEMPLO EN VIVO

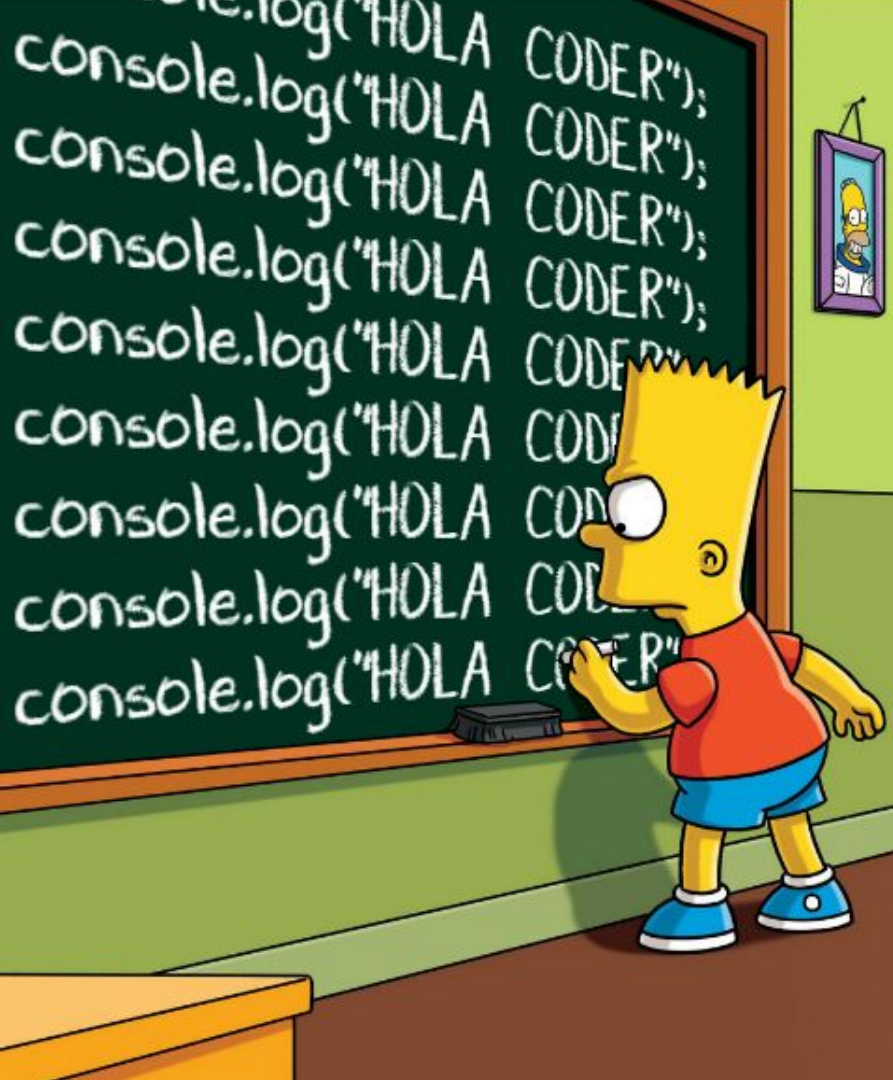


CREAR UN ALGORITMO
UTILIZANDO FUNCIÓN



SIMULADOR INTERACTIVO

CICLOS



CICLOS EN JAVASCRIPT

Los ciclos, también conocidos como bucles o iteraciones son un medio rápido y sencillo para hacer algo repetidamente.

Si tenemos que hacer alguna operación más de una vez en el programa, de forma consecutiva, usaremos las estructuras de bucles de JavaScript: `for`, `while` o `do..while`.

TIPOS DE BUCLES

CICLOS POR CONTEO

Repiten un bloque de código un número de veces específica.

Estructura **for**.

CICLOS CONDICIONALES

Repiten un bloque de código mientras la condición evaluada es verdadera. Estructuras **while** y **do...while**

ESTRUCTURA FOR

```
for(desde; hasta; actualización) {  
    ... //lo que se escriba acá se ejecutará mientras dure el  
    ciclo  
}
```

El "**desde**" es la zona en la que se establecen los valores iniciales de las variables que controlan el ciclo.

El "**hasta**" es el único elemento que decide si se repite o se detiene el ciclo.

La "**actualización**" es el nuevo valor que se asigna después de cada repetición a las variables que controlan la repetición.

EJEMPLO PRÁCTICO

En el siguiente ejemplo utilizamos un for para contar de 0 a 9.

```
for (let i = 0; i < 10; i++) {  
    alert(i);  
}
```

Ahora usamos for para contar de 1 a 10.

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

EJEMPLO APLICADO FOR (1): TABLAS

Algoritmo para calcular la tabla de multiplicar de un número.

```
// Solicitamos un valor al usuario
let ingresarNumero = parseInt(prompt("Ingresar Numero"));
// En cada repetición, calculamos el número ingresado x el número de repetición (i)
for (let i = 1; i <= 10; i++) {
    let resultado = ingresarNumero * i ;
    alert(ingresarNumero + " X " + i + " = " + resultado);
}
```

EJEMPLO APLICADO FOR (2): TURNOS

Algoritmo para dar turno del 1 al 20 a los nombres ingresados

```
for (let i = 1; i <= 20; i++) {  
    // En cada repetición solicitamos un nombre.  
    let ingresarNombre = prompt("Ingresar nombre");  
    // Informamos el turno asignado usando el número de repetición (i).  
    alert(" Turno  N° "+i+" Nombre: "+ingresarNombre);  
}
```

SENTENCIA BREAK

A veces, cuando escribimos una estructura for, necesitamos que bajo cierta condición el ciclo se interrumpa. Para eso se utiliza la sentencia break; Al escribir esa línea dentro de un ciclo for, el mismo se interrumpirá como si hubiera finalizado.

```
for (let i = 1; i <= 10; i++) {  
  //Si la variable i es igual 5 interrumpo el for.  
  if(i == 5){  
    break;  
  }  
  alert(i);  
}
```

SENTENCIA CONTINUE

A veces, cuando escribimos una estructura for, necesitamos que bajo cierta condición, el ciclo saltee esa repetición y siga con la próxima. Para eso se utiliza la sentencia continue;

```
for (let i = 1; i <= 10; i++) {  
    //Si la variable i es 5, no se interpreta la repetición  
    if(i == 5){  
        continue;  
    }  
    alert(i);  
}
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

WHILE

WHILE

La estructura *while* permite crear bucles que se ejecutan **ninguna o más veces**, dependiendo de la condición indicada.

El funcionamiento del bucle *while* se resume en: ***mientras se cumpla la condición indicada, repite indefinidamente las instrucciones incluidas dentro del bucle.***

WHILE

Ejemplo de repetición infinita:

```
let repetir = true;  
while(repetir){  
    console.log("Al infinito y...¡Más allá!");  
}
```

7818 Al infinito y...¡Más allá!

EJEMPLO APLICADO WHILE: ESC

Algoritmo que solicita una entrada al usuario hasta que ingresa "ESC"

```
let entrada = prompt("Ingresar un dato");  
//Repetimos con While hasta que el usuario ingresa "ESC"  
while(entrada != "ESC" ){  
    alert("El usuario ingresó "+ entrada);  
    //Volvemos a solicitar un dato. En la próxima iteración se evalúa si no es ESC.  
    entrada = prompt("Ingresar otro dato");  
}
```

DO...WHILE

La estructura `do..while` permite crear bucles que se ejecutan una o más veces, dependiendo de la condición indicada.

A diferencia de `while`, garantiza que el bloque de código se interpreta al menos una vez, porque la condición se evalúa al final.

```
let repetir = false;
do{
    console.log("¡Solo una vez!");
}while(repetir)
```

EJEMPLO APLICADO DO...WHILE: N°

Algoritmo que solicita una entrada y se detiene cuando NO es un número

```
let numero = 0;
do{
    //Repetimos con do...while mientras el usuario ingresa un n°
    numero = prompt("Ingresar Número");
    console.log(numero);
    //Si el parseo no resulta un número se interrumpe el bucle.
}while(parseInt(numero));
```

SWITCH

SWITCH

La estructura `switch` está especialmente diseñada para manejar de forma sencilla **múltiples condiciones sobre la misma variable** (técnicamente se podría resolver con un `if`, pero el uso de `switch` es más ordenado). Su definición formal puede parecer confusa, pero veamos un ejemplo para entender su simpleza.

SWITCH

```
switch(numero) {  
  case 5:  
    ...  
    break;  
  case 8:  
    ...  
    break;  
  case 20:  
    ...  
    break;  
  default:  
    ...  
    break;  
}
```

Cada condición se evalúa y si se cumple, se ejecuta lo que esté indicado adentro.

Normalmente, después de las instrucciones de cada case se incluye la sentencia `break` para terminar la ejecución del `switch`, aunque no es obligatorio.

¿Qué sucede si ningún valor de la variable del `switch` coincide con los valores definidos en los `case`?

En este caso, se utiliza el valor `default` para indicar las instrucciones que se ejecutan cuando ninguna condición anterior se cumplió.

```
let entrada = prompt("Ingresar un nombre");  
//Repetimos hasta que se ingresa "ESC"  
while(entrada != "ESC" ){  
    switch (entrada) {  
        case "ANA":  
            alert("HOLA ANA");  
            break;  
        case "JUAN":  
            alert("HOLA JUAN");  
            break;  
        default:  
            alert("¿QUIÉN SOS?")  
            break;  
    }  
    entrada = prompt("Ingresar un nombre");  
}
```

EJEMPLO APLICADO WHILE Y SWITCH: ENTRADAS

Algoritmo que hace operación según la entrada. Pero ignorando la ejecución de bloque si la entrada en “ESC”

¡LO MÁS IMPORTANTE!

Todas los temas que vimos (y los que vamos a ver), se pueden (y deben) combinar entre sí. De forma que en una función haya un condicional, con un for adentro, y dentro de ese un while, y así la combinación es infinita.

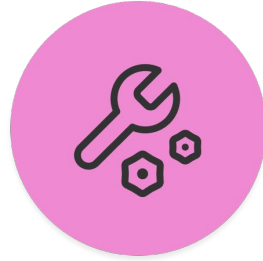
¡Ahí es cuando la programación JavaScript empieza a volverse interesante!

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



CREAR UN ALGORITMO UTILIZANDO UN CICLO

Tomando como base los ejemplos anteriores de la estructura *for* *while* y *do..while*, crear un algoritmo que repita un bloque de instrucciones.

CREAR UN ALGORITMO UTILIZANDO UN CICLO

Formato: Página HTML y código fuente en JavaScript.

Sugerencia: Usamos la instrucción for para repetir un número fijo de veces. Mientras que usamos while cuando queremos repetir algo hasta que se deje de cumplir una condición.

Desafío
generico



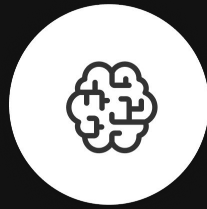
>> Consigna: Tomando como base los ejemplos anteriores de la estructura for y while, crear un algoritmo que repita un bloque de instrucciones. En cada repetición es necesario efectuar una operación o comparación para obtener una salida por alerta o consola.

>>Ejemplo:

- Pedir número mediante prompt y sumarle otro número en cada repetición, realizando una salida por cada resultado
- Pedir un texto mediante prompt, concatenar un valor en cada repetición, realizando una salida por cada resultado, hasta que se ingresa "ESC".
- Pedir un número por prompt, repetir la salida del mensaje "Hola" la cantidad de veces ingresada.

¿PREGUNTAS?





¡PARA PENSAR!

¿Te gustaría comprobar tus conocimientos de la clase?

Te compartimos a través del chat de zoom
el enlace a un breve quiz de tarea.

Para el profesor:

- *Acceder a la carpeta "Quizzes" de la camada*
 - *Ingresar al formulario de la clase*
 - *Pulsar el botón "Invitar"*
 - *Copiar el enlace*
- *Compartir el enlace a los alumnos a través del chat*



RECURSOS:

Material
ampliado



- Bucles |

Los apuntes de Majo (Página 17 a 19).

Te lo explico con gatitos. Bucle FOR.

Te lo explico con gatitos. Bucle WHILE.

- Funciones |

Los apuntes de Majo (Página 20).

Te lo explico con gatitos. Parte 1.

Te lo explico con gatitos. Parte 2.

- Documentación |

Documentación FOR.

Documentación WHILE.

Disponible en [nuestro repositorio.](#)

CODER HOUSE

GLOSARIO:

Clase 2

Operadores lógicos: permiten agrupar expresiones lógicas. Las expresiones lógicas son todas aquellas expresiones que obtienen como resultado verdadero o falso. Los operadores lógicos son aquellos que hacen de nexo de este tipo de expresiones.

Condicionales: cuando en programación hablamos de condicionales, hablamos de una estructura sintáctica que sirve para tomar una decisión a partir de una condición.

Anidar: en programación, se refiere a escribir una sentencia junto a una subsiguiente dentro de la misma estructura sintáctica. Es decir, que no hay un salto de línea en el medio.

Estructura IF: es la más utilizada en la mayoría de los lenguajes. Si la condición se cumple (es decir, si su valor es true) se ejecutan todas las instrucciones que se encuentran dentro de {...}. Si la condición no se cumple (es decir, si su valor es false) no se ejecuta ninguna instrucción contenida en {...} y el programa continúa ejecutando el resto de instrucciones del script.

IF...ELSE: en ocasiones, las decisiones que se deben realizar no son del tipo "si se cumple la condición, hazlo; si no se cumple, no hagas nada". Normalmente las condiciones suelen ser del tipo "si se cumple esta condición, hazlo; si no se cumple, haz esto otro".

GLOSARIO:

Clase 3

Declarar función: se dice declarar cuando uno define una función en el código.

Ciclo: en programación, se refiere a un conjunto de indicaciones que se repiten bajo ciertas condiciones.



Clase 04. JAVASCRIPT

PROGRAMACIÓN CON FUNCIONES



OBJETIVOS DE LA CLASE

Entender:

- ¿Qué es una función y cómo nos ayuda a escribir menos código?
- ¿Qué son los parámetros de entrada y salida de una función?
- ¿Qué es el Scope global y el Scope local?
- ¿Qué es una función anónima y una función flecha?

GLOSARIO:

Clase 3

Ciclos en JS: en programación, ciclo se refiere a un conjunto de indicaciones que se repiten bajo ciertas condiciones. Las estructuras de ciclos o cíclicas son las que debemos utilizar cuando necesitamos repetir ciertas operaciones de la misma manera durante N cantidad de veces.

Sentencia break: a veces, cuando escribimos una estructura for, necesitamos que bajo cierta condición el ciclo se interrumpa. Para eso se utiliza esta sentencia.

Sentencia continue: a veces, cuando escribimos una estructura for, necesitamos que bajo cierta condición, el ciclo saltee esa repetición y siga con la próxima. Para eso se utiliza esta sentencia.

Estructura while: permite crear bucles que se ejecutan ninguna o más veces, dependiendo de la condición indicada.

CRONOGRAMA DEL CURSO

Clase 3



Ciclos/Iteraciones



EJEMPLOS EN VIVO



CREAR UN ALGORITMO
UTILIZANDO UN CICLO

Clase 4



Funciones



EJEMPLO EN VIVO



CREAR UN ALGORITMO
UTILIZANDO FUNCIÓN



SIMULADOR INTERACTIVO

Clase 5



Objetos



EJEMPLOS EN VIVO

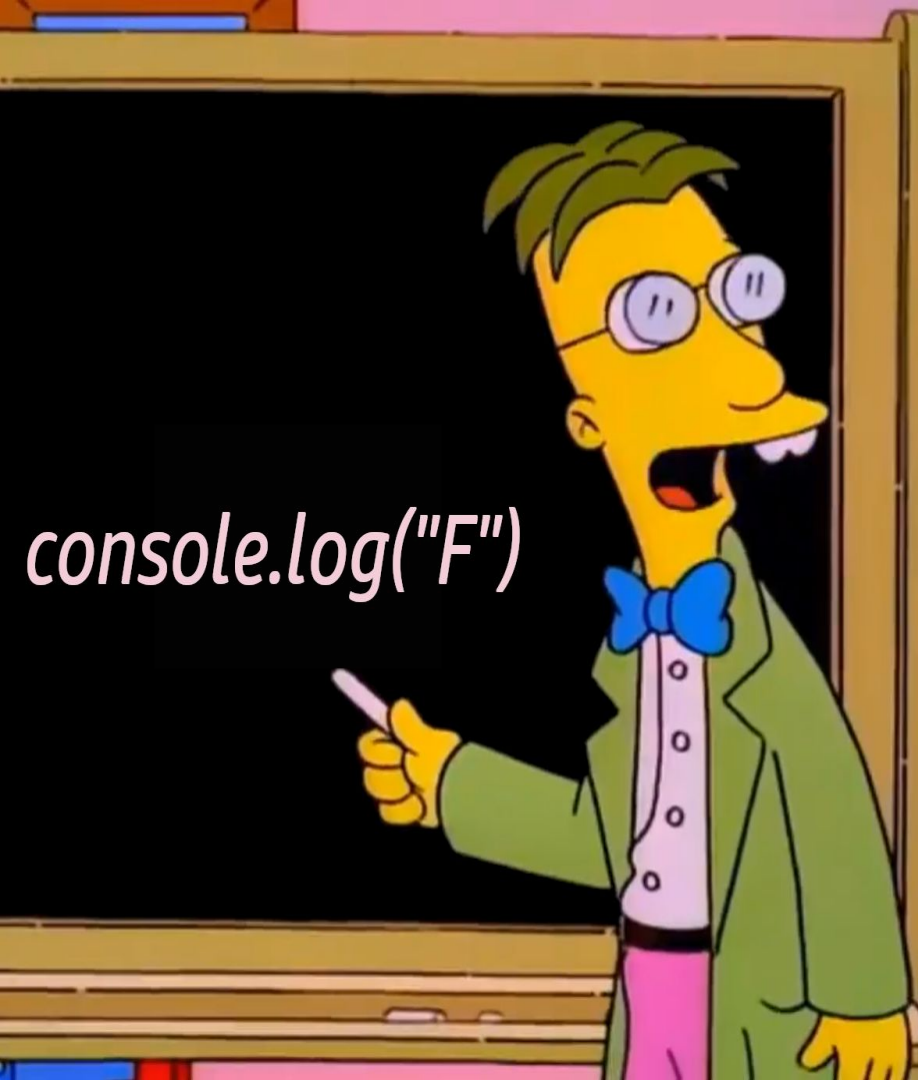


CREAR UN OBJETO Y
UTILIZARLO



INCORPORAR OBJETOS

FUNCIONES Y PROPIEDADES BÁSICAS



FUNCIONES

Cuando se desarrolla una aplicación o sitio web, es muy habitual utilizar una y otra vez las mismas instrucciones.

En programación, una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta, que luego se pueden reutilizar a lo largo de diferentes instancias del código.

¿Y QUÉ VENTAJAS ME DAN LAS FUNCIONES?

Las principales ventajas del uso de funciones son:

- Evitar instrucciones duplicadas ([Principio DRY](#))
- Solucionar un problema complejo usando tareas sencillas ([Principio KISS](#))
- Focalizarse en tareas prioritarias para el programa ([Principio YAGNI](#))
- Aporta ordenamiento y entendimiento al código
- Aporta facilidad y rapidez para hacer modificaciones

¿CÓMO ESCRIBIRLAS?

Todas las funciones se escriben igual. Deben tener un nombre en minúscula y sin espacios. Deben abrirse y cerrarse con llaves. El contenido de la función se escribe entre las llaves. El nombre de la función no se puede repetir en otra.

```
function saludar() {  
    console.log("¡Hola estudiantes!");  
}
```

¿Y AHORA?

Una vez que declaramos la función podemos usarla en cualquier otra parte del código las veces que queramos. Para ejecutar una función sólo hay que escribir su nombre y finalizar la sentencia con (). A esto se lo conoce como *llamada de la función*

```
saludar();
```

EJEMPLO PRÁCTICO

Si debemos solicitar un nombre al usuario mostrarlo en un alert, normalmente podríamos hacer esto:

```
var nombreIngresado = prompt("Ingresar nombre");  
alert("El nombre ingresado es " + nombreIngresado);
```

Si queremos repetir esto 2 veces más , podemos copiar y pegar el código.

```
var nombreIngresado = prompt("Ingresar nombre");  
alert("El nombre ingresado es " + nombreIngresado);  
var nombreIngresado = prompt("Ingresar nombre");  
alert("El nombre ingresado es " + nombreIngresado);
```

USANDO UNA FUNCIÓN

Podríamos entonces crear una función que se llame solicitarNombre() para solicitar al usuario la cantidad de veces que necesitamos

```
function solicitarNombre() {  
    let nombreIngresado = prompt("Ingresar nombre");  
    alert("El nombre ingresado es " + nombreIngresado);  
}
```

Para llamar a la función, la invocamos en otra parte del código:

```
solicitarNombre();  
solicitarNombre();  
solicitarNombre();
```

FUNCIONES: PARÁMETROS

PARÁMETROS

Una función simple, puede no necesitar ninguna dato para funcionar.

Pero cuando empezamos a codificar funciones más complejas, nos encontramos con la necesidad de recibir cierta información para funcionar.

Cuando enviamos a la función uno o más valores para que ser empleados en sus operaciones, estamos hablando de los **parámetros de la función**.

Los parámetros se envían a la función mediante variables y se colocan entre los paréntesis posteriores al nombre de la función.

PARÁMETROS

```
function conParametros(parametro1, parametro2) {  
    console.log(parametro1 + " " + parametro2);  
}
```

Dentro de la función, el valor de la variable parametro1 tomará al primer valor que se le pase a la función, y el valor de la variable parametro2 tomará el segundo valor que se le pasa.

EJEMPLO APLICADO: SUMAR Y MOSTRAR

```
//Declaración de variable para guardar el resultado de la suma
let resultado = 0;
//Función que suma dos números y asigna a resultado
function sumar(primerNumero, segundoNumero) {
    resultado = primerNumero + segundoNumero;
}
//Función que muestra resultado por consola
function mostrar(mensaje) {
    console.log(mensaje);
}
//Llamamos primero a sumar y luego a mostrar
sumar(6, 3);
mostrar(resultado);
```

RESULTADO DE UNA FUNCIÓN

El ejemplo anterior sumamos dos números a una variable declarada anteriormente. Pero las funciones pueden generar un valor de retorno usando la palabra `return`, obteniendo el valor cuando la función es llamada

```
function sumar(primerNumero, segundoNumero) {  
    return primerNumero + segundoNumero;  
}  
  
let resultado = sumar(5, 8);
```

```
function calculadora(primerNumero, segundoNumero, operacion) {  
  switch (operacion) {  
    case "+":  
      return primerNumero + segundoNumero;  
      break;  
    case "-":  
      return primerNumero - segundoNumero;  
      break;  
    case "*":  
      return primerNumero * segundoNumero;  
      break;  
    case "/":  
      return primerNumero / segundoNumero;  
      break;  
    default:  
      return 0;  
      break;  
  }  
}  
  
console.log(calculadora(10, 5, "*"));
```

EJEMPLO APLICADO: CALCULADORA

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



BREAK

¡5/10 MINUTOS Y VOLVEMOS!

VARIABLES LOCALES Y GLOBALES

DEFINICIÓN

El ámbito de una variable (llamado "scope" en inglés), es la zona del programa en la que se define la variable, el contexto al que pertenece la misma dentro de un algoritmo.

JavaScript define dos ámbitos para las variables: global y local.

VARIABLES LOCALES

Cuando definimos una variable dentro de una función o bloque es una **variable local**, la misma existirá sólo durante la ejecución de esa sección. Si queremos utilizarla por fuera, la variable no existirá para JS.

```
function sumar(primerNumero, segundoNumero) {  
    let resultado = primerNumero + segundoNumero;  
}  
  
//No se puede acceder a la variable resultado fuera del bloque  
console.log(resultado);
```

✖ ▶ Uncaught ReferenceError: resultado is not defined

VARIABLES GLOBALES

Si una variable se declara fuera de cualquier función o bloque, automáticamente se transforma en variable global, independientemente de si se define utilizando la palabra reservada var, o no.

```
let resultado = 0

function sumar(primerNumero, segundoNumero) {
    resultado = primerNumero + segundoNumero;
}

sumar(5, 6);

//Se puede acceder a la variable resultado porque es global
console.log(resultado);
```

FUNCIONES ANÓNIMAS Y FUNCIONES FLECHA

FUNCIONES ANÓNIMAS

Una función anónima es una función que se define sin nombre y se utiliza para ser pasadas como parámetros o asignada a variable. En el caso de asignarla a una variable, pueden llamar usando el identificador de la variable declarada

```
//Generalmente, las funciones anónimas se asignan a variables declaradas como constantes  
const suma = function (a, b) { return a + b };  
const resta = function (a, b) { return a - b };  
console.log(suma(15,20));  
console.log(resta(15,5));
```

FUNCIONES FLECHA

Identificamos a las funciones flechas como funciones anónimas de sintaxis simplificada. Están disponibles desde la versión ES6 de JavaScript, no usan la palabra `function` pero usamos `=>` (flecha) entre los parámetros y el bloque

```
const suma = (a, b) => { return a + b };  
//Si es una función de una sola línea con retorno podemos evitar escribir el cuerpo.  
const resta = (a, b) => a - b ;  
console.log(suma(15,20));  
console.log(resta(20,5));
```

EJEMPLO APLICADO: CALCULAR PRECIO

```
const suma = (a,b) => a + b;
const resta = (a,b) => a - b;
//Si una función es una sola línea con retorno y un parámetro puede evitar escribir los ()
const iva = x => x * 0.21;
let precioProducto = 500;
let precioDescuento = 50;
//Calculo el precioProducto + IVA - precioDescuento
let nuevoPrecio = resta(suma(precioProducto, iva(precioProducto)), precioDescuento);
console.log(nuevoPrecio);
```

Ejemplo
en vivo



¡VAMOS A PRACTICAR LO VISTO!

CODER HOUSE



SIMULADOR INTERACTIVO

Empieza a armar la estructura inicial de tu proyecto integrador.

SIMULADOR INTERACTIVO

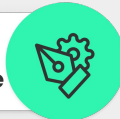
Formato: Página HTML y código fuente en JavaScript. Debe identificar el apellido del alumno/a en el nombre de archivo comprimido por “claseApellido”.

Sugerencia: Algunos criterios a tener en cuenta para seleccionar un proceso a simular por primera vez son:

“ELEGIR UN PROCESO BIEN CONOCIDO” : Si conozco una situación que implique adquirir cierta información y estoy bien familiarizado en “cómo se hace” es más fácil traducir la solución a un lenguaje de programación.

“ELEGIR UN PROCESO QUE ME RESULTE INTERESANTE” : Si me siento motivado sobre el tema, es más llevadero enfrentar los retos de desarrollo e interpretación: Antes de programar existe la etapa de relevamiento y análisis que me permite identificar cómo solucionar el proceso.

Desafío
entregable

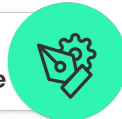


SIMULADOR INTERACTIVO

>> Consigna: Con los conocimientos vistos hasta el momento, empezarás a armar la estructura inicial de tu proyecto integrador. A partir de los ejemplos mostrados la primera clase, deberás:

- Pensar el alcance de tu proyecto: ¿usarás un cotizador de seguros? ¿un simulador de simulador personalizado?
- Armar la estructura HTML del proyecto.
- Incorporar al menos un prompt para pedir un dato y luego mostrarlo mediante alert realizando alguna operación.
- Utilizar funciones para realizar esas operaciones.

Desafío
entregable



>>Aspectos a incluir en el entregable:

Archivo HTML y Archivo JS, referenciado en el HTML por etiqueta `<script src="js/miarchivo.js"></script>`, que incluya la definición de un algoritmo en JavaScript que emplee funciones para resolver el procesamiento principal del simulador

>>Ejemplo:

Calcular costo total de productos y/o servicios seleccionados por el usuario.

Calcular pagos en cuotas sobre un monto determinado.

Calcular valor final de un producto seleccionado en función de impuestos y descuentos.

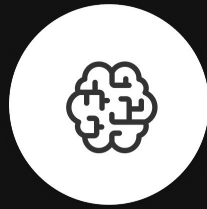
Calcular tiempo de espera promedio en relación a la cantidad de turnos registrados.

Calcular edad promedio de personas registradas.

Calcular nota final de alumnos ingresados.

¿PREGUNTAS?





¡PARA PENSAR!

¿Te gustaría comprobar tus conocimientos de la clase?

Te compartimos a través del chat de zoom
el enlace a un breve quiz de tarea.

Para el profesor:

- *Acceder a la carpeta "Quizzes" de la camada*
 - *Ingresar al formulario de la clase*
 - *Pulsar el botón "Invitar"*
 - *Copiar el enlace*
- *Compartir el enlace a los alumnos a través del chat*



RECURSOS:

Material
ampliado



- Scope |
Te lo explico con gatitos.
- Documentación |
Documentación LET
Documentación CONST

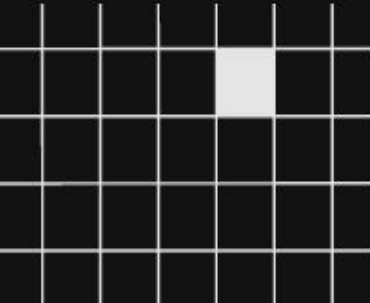
Disponible en [nuestro repositorio](#).

CODER HOUSE



¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

- Parámetros y resultado de una función.
 - Variables locales y globales.
 - Funciones anónimas y flecha
- 



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDO LA EDUCACIÓN

CODER HOUSE

CODER HOUSE

***¡GRACIAS POR ESTUDIAR CON
NOSOTROS!***
