Esta clase va a ser

- grabada

Clase 04. JAVASCRIPT

Programación con funciones

Objetivos de la clase

- Conceptualizar **función** en programación y comprender sus ventajas.
- Identificar **parámetros** de entrada y salida de una función.
- Comprender qué es el **Scope** y conocer las **variables globales** y **variables locales** en JavaScript.
- Definir y diferenciar **función anónima** y **función flecha**.

CLASE N°3

Glosario

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.

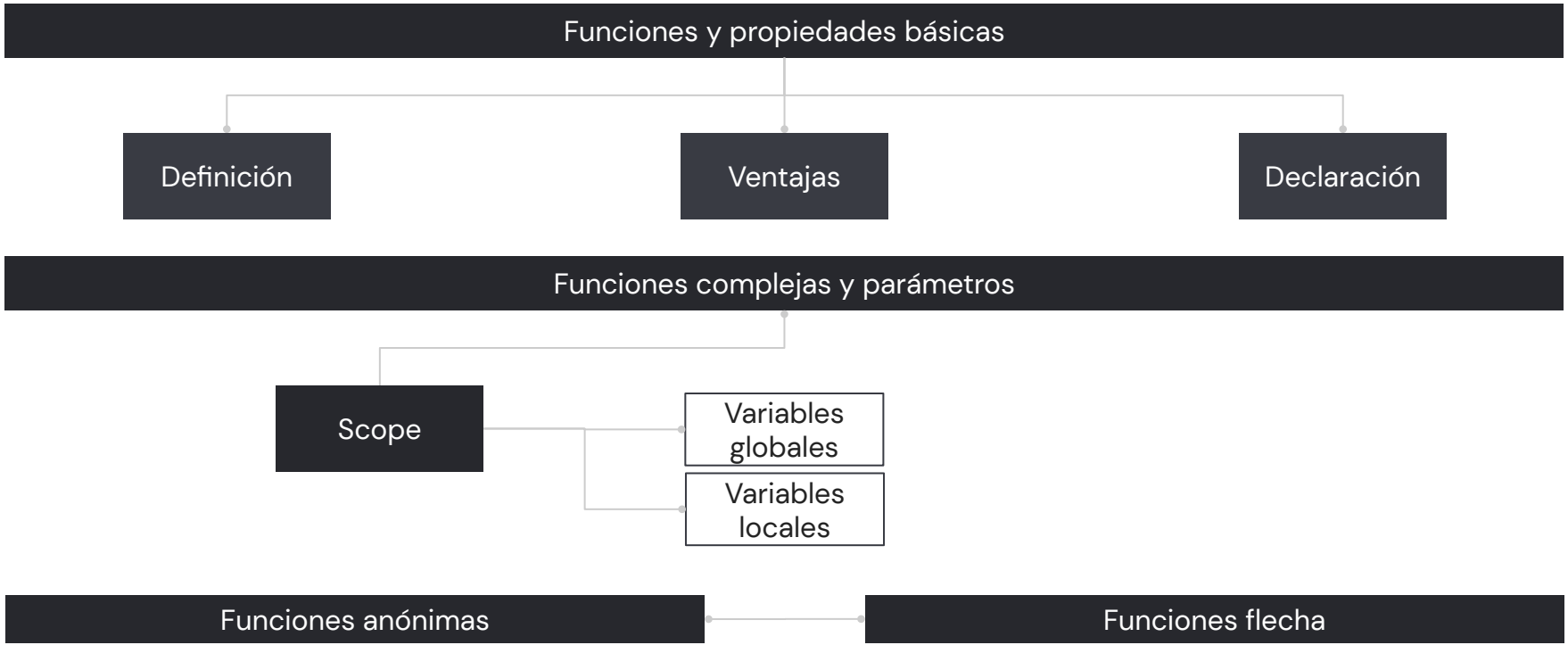
Sentencia break: A veces, cuando escribimos una estructura for, necesitamos que bajo cierta condición el ciclo se interrumpa. Para eso se utiliza esta sentencia.

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.

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

MAPA DE CONCEPTOS CLASE 4



Temario

03

Ciclos e iteraciones

- ✓ Ciclos
- ✓ Operador Switch

04

Programación con funciones

- ✓ Funciones y propiedades básicas
- ✓ Scope
- ✓ Funciones anónimas y funciones flecha

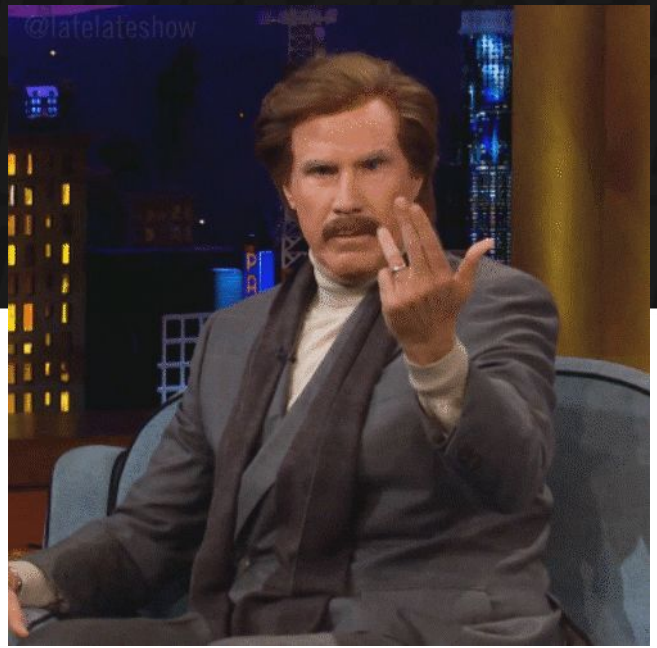
05

Objetos

- ✓ Constructores
- ✓ Métodos y operaciones
- ✓ Clases

CODERHOUSE

Empezamos...



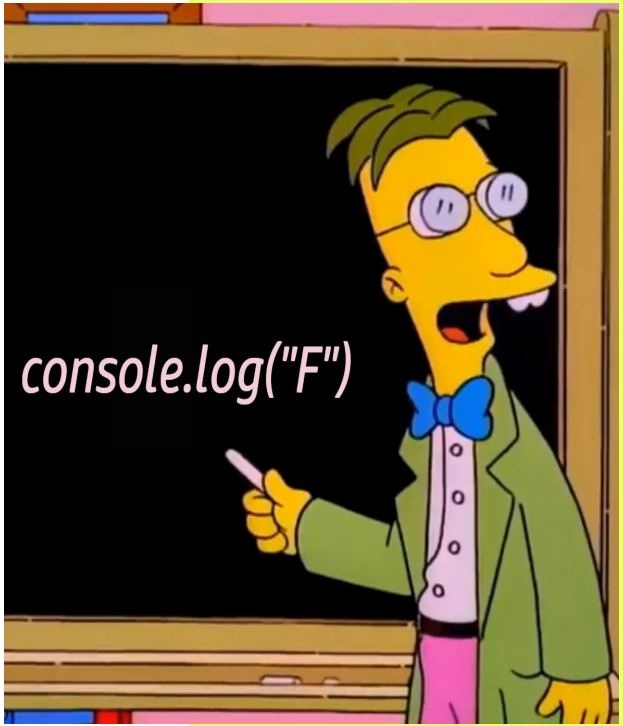
CODERHOUSE

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 puede reutilizar a lo largo de diferentes instancias del código.



CODERHOUSE

¿Y qué ventajas me dan las funciones?

Las principales ventajas del uso de funciones son:

- ✓ Evita instrucciones duplicadas (Principio DRY).
- ✓ Soluciona un problema complejo usando tareas sencillas (Principio KISS).
- ✓ Focaliza tareas prioritarias para el programa (Principio YAGNI).
- ✓ Aporta ordenamiento y entendimiento al código.
- ✓ Aporta facilidad y rapidez para hacer modificaciones.

Declaración

Se declara a través de la palabra reservada **function**. Deben tener un nombre en minúscula y sin espacios seguidos de los característicos paréntesis (). 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!");
}
```

Llamado

Una vez que declaramos la función, podemos usarla en cualquier otra parte del código todas 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 llamado a la función.

```
saludar();
```

Donde escribamos el llamado, se interpretarán las instrucciones definidas en esa función.

Ejemplo práctico

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

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

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

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

Usando una función

Podríamos entonces crear una función que se llame **solicitarNombre()** para que se le solicite al usuario la cantidad de veces que necesitemos

```
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

CODERHOUSE

Parámetros

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

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

Cuando enviamos a la función uno o más valores para 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

Los parámetros son variables que se declaran dentro de la función, entre sus paréntesis. Los valores de éstos se definen luego en el llamado.

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

Así, podemos armar funciones dinámicas que, siguiendo la lógica que queramos, pueden generar distintos resultados al recibir diferentes valores.

Parámetros

El valor que toman estos parámetros se definen en el *llamado*.
Cuando llamamos a la función, los valores que pasamos a la función entre paréntesis se asignan **posicionalmente** a los parámetros correspondientes, generando posibles resultados diferentes:

```
conParametros("Hola", "Coder"); // -> “Hola Coder”
conParametros("Cursando", "JS"); // -> “Cursando JS”
```

En este caso, el primer string que pasamos se asigna en parametro1, y el segundo string en parametro2; armando las salidas según la lógica definida.

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

En 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);
```

Resultado de una función

La función puede comportarse como una operación que genera valores (como en las operaciones matemáticas y lógicas previas).
En el espacio donde se llama a la función se genera un nuevo valor:
este valor es el definido por el **return** de la misma.

```
let resultado = sumar(5, 8);


console.log(resultado)  // ⇒ 13
```

CODERHOUSE

```
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

CODERHOUSE

Ejemplo en vivo

¡VAMOS A PRACTICAR LO VISTO!

Break

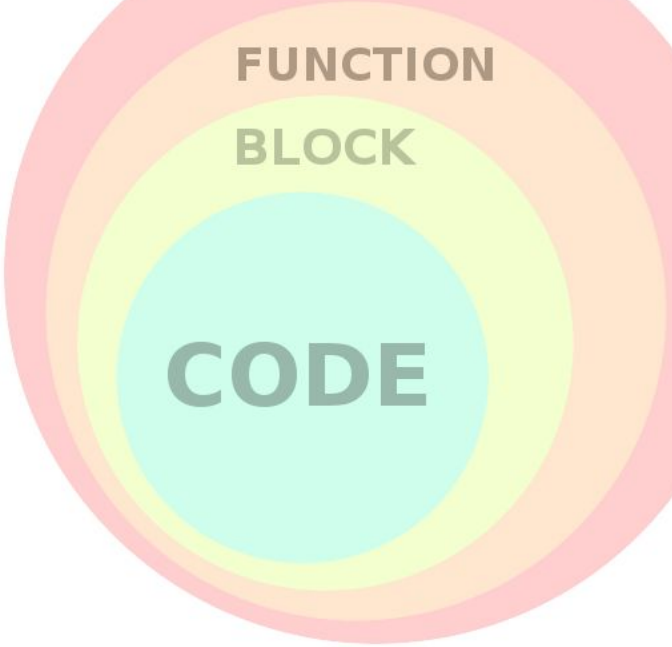
¡5/10 minutos y volvemos!

Scope

Scope

El scope o **ámbito** de una variable es la zona del programa en la cual se define, el contexto al que pertenece la misma dentro de un algoritmo, restringiendo su uso y alcance.

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



Variables globales y locales

CODERHOUSE

Variables globales

Si una variable se declara fuera de cualquier función o bloque, automáticamente se transforma en variable global.

```
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);
```

Puede ser referenciada desde cualquier punto del programa.

Variables locales

Cuando definimos una variable dentro de una función o bloque es una variable local, y será accesible sólo dentro de ese espacio. **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
```

CODERHOUSE

Variables locales y globales

```
let nombre = "John Doe" // variable global

function saludar() {
    let nombre = "Juan Coder" // variable local
    console.log(nombre)
}
//Accede a nombre global
console.log(nombre)    // → "John Doe"

//Accede a nombre local
saludar() // → "Juan Coder"
```

Hay que entender que las variables **globales** y **locales** se identifican como diferentes entre sí, y pueden existir en el programa bajo el mismo nombre sin conflicto.

CODERHOUSE

Scope

Entender que cada scope local es un espacio cerrado nos permite crear bloques de trabajo bien diferenciados e independientes, sin preocuparnos por repetir nombres de variables, sabiendo que se entienden como diferentes según donde las llamemos.

```
function sumar(num1, num2) {
    let resultado = num1 + num2
    return resultado
}

function restar(num1, num2) {
    let resultado = num1 - num2
    return resultado
}
```

Funciones anónimas y funciones flecha

Funciones anónimas

Una función anónima es una función que se define **sin nombre** y se utiliza para ser pasada como parámetro o asignada a una 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 usa => (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 descuento = 50
//Calculo el precioProducto + IVA - descuento
let nuevoPrecio = resta(suma(precioProducto, iva(precioProducto)), descuento)
console.log(nuevoPrecio)
```

Ejemplo en vivo

¡VAMOS A PRACTICAR LO VISTO!

Primera entrega de tu Proyecto final

Empieza a armar la estructura inicial de tu proyecto integrador.

CODERHOUSE

Compuesta por...

- ✓ Crear un algoritmo con un condicional.
- ✓ Crear un algoritmo utilizando un ciclo.
- ✓ **Armar un simulador interactivo, la estructura final de tu proyecto integrador.**
- ✓ Recuerden que **tendrán hasta 7 días para resolver la entrega y subirla**.

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 créditos? ¿un simulador personalizado?
- ✓ Armar la estructura HTML del proyecto.
- ✓ **Incorporar lo ejercitado en las clases anteriores, algoritmo condicional y algoritmo con ciclo.**
- ✓ Utilizar funciones para realizar esas operaciones.

Formato

- ✓ Página HTML y código fuente en JavaScript. Debe identificar el apellido del alumno/a en el nombre de archivo comprimido por ““PreEntrega1+Apellido””.

Simulador interactivo

Aspectos a incluir

- ✓ 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 con la cantidad de turnos registrados.
- ✓ Calcular edad promedio de personas registradas.
- ✓ Calcular nota final de alumnos ingresados.

Sugerencias

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.

¿Preguntas?

Encuesta
sobre esta clase

Por encuestas de Zoom

¡Terminamos el módulo 1: Conceptos básicos!

Cuéntanos qué temas te resultaron más complejos de entender. **Puedes elegir más de uno**. Vamos a retomar aquellos temas que resultaron de mayor dificultad en el próximo AfterClass.

CODERHOUSE

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.

CODERHOUSE

MATERIAL AMPLIADO

Recursos

Scope

- ✓ Te lo explico con gatitos

Documentación

- ✓ Documentación LET
- ✓ Documentación CONST

Disponible en nuestro repositorio.

Resumen
de la clase hoy

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

Muchas gracias.

Opina y valora esta clase

#DemocratizandoLaEducación