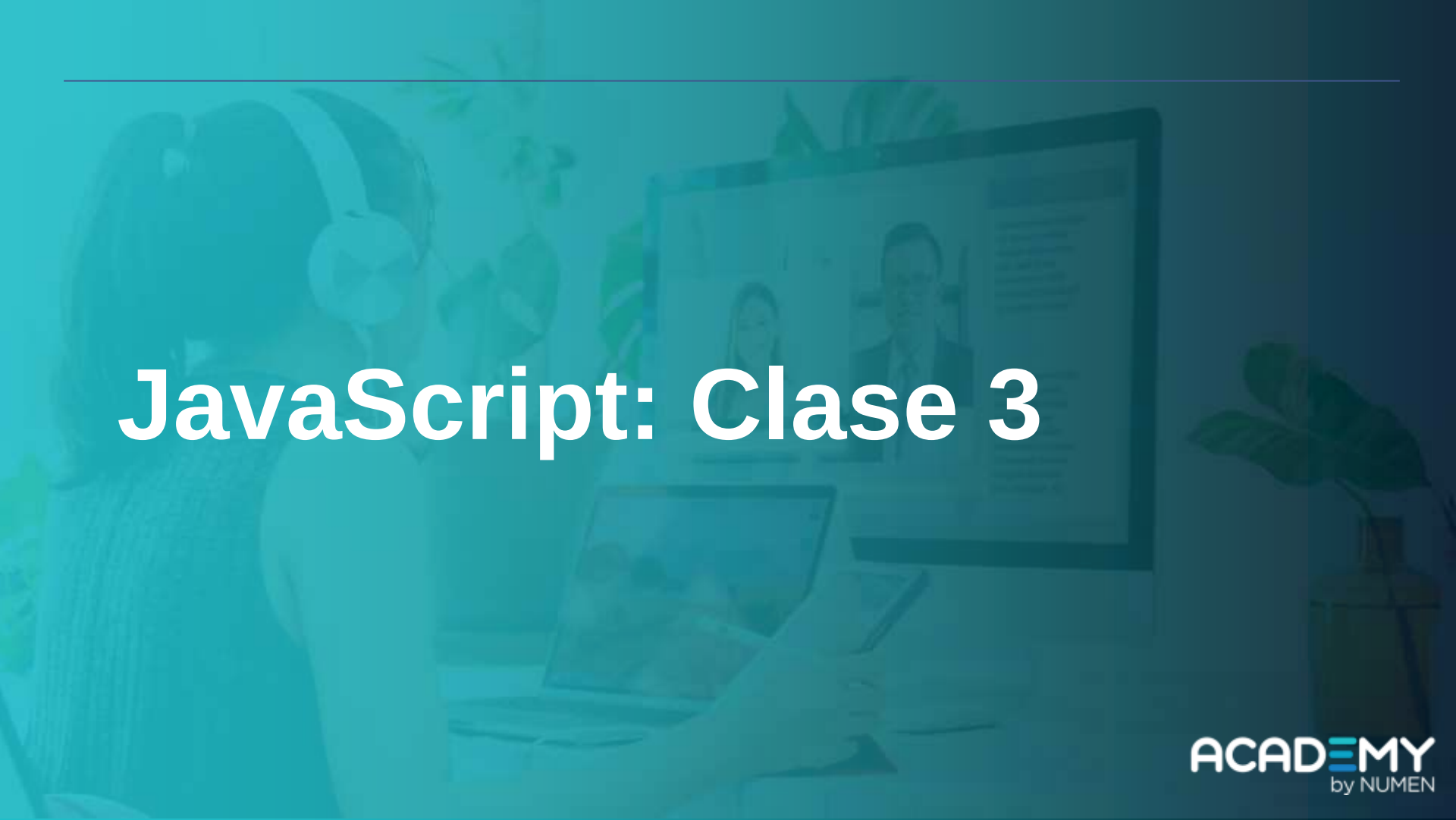
ACADEMY by NUMEN

FULL STACK

Comenzamos en unos minutos


ACADEMY
by NUMEN

JavaScript: Clase 3



ACADEMY by NUMEN

Funciones

ACADEMY by NUMEN

“Divide y vencerás”

En programación, una función es un conjunto de instrucciones que llevan a cabo un objetivo en particular. Esto hará que podamos dividir un programa largo en subprogramas, reduciendo repeticiones, asociando nombres a subprogramas y aislándolos entre sí.

Cada subproblema se puede resolver independientemente de los demás y las soluciones de estos subproblemas pueden combinarse para resolver el problema original.

A partir de la descomposición funcional de un problema, se logra dividir un problema en un conjunto de tareas cada vez más sencillas de resolver.

DECLARACIÓN

Una función se identifica mediante un **nombre**, respetando el formato **camelCase** y por lo general escritas utilizando verbos en infinitivo, dejando en claro que realizan una acción.

Hay casos en los que las funciones no devuelven ningún valor. Por ejemplo, aquellas que se limitan a imprimir un título. En este caso, es algo que funciona por sí solo y no necesitan ser almacenadas en una variable para ser utilizadas.

```
function imprimir() {
    console.log("Hola Mundo!")
}
```

Una vez definida la función, es necesario llamarla para ser ejecutada.

```
function imprimir() {
    console.log("Hola Mundo!")
}

imprimir()
```

Argumentos y parámetros

ACADEMY by NUMEN

Una función está compuesta de **instrucciones** y **datos**.

Los datos que son propios de esa función serán llamados **datos locales** y aquellos datos que deba recibir o enviar a otras funciones serán llamados **datos de comunicación**.

Los datos conocidos por todos los módulos del sistema se llaman **datos globales**.

Los datos son los valores de información de los que se necesita **disponer**, y en ocasiones **transformar**, para ejecutar la función del programa.

Algunas funciones requieren de datos externos para poder operar. Para ello recibimos esos datos por medio de **parámetros**, los cuales son pasados dentro de los paréntesis de la misma.

Se denominan **parámetros** a la serie de datos con los que se comunican los módulos. Deben definirse en el encabezado de la función, y al momento de invocar.

Los **argumentos** son aquellos datos que recibe la función al ser invocada.

Los **parámetros** y los **argumentos** se corresponden por posición.

```
// Definición de la función
function saludar(nombre, apellido) {
    console.log("Hola " + nombre + " " + apellido
+ "!");
}

// Invocación
saludar("Laura", "Gimenez")
```

Return

ACADEMY by NUMEN

Hay situaciones en las que necesitaremos disponer del resultado que nos devuelva una función. Para ello contamos con la instrucción **return**;

```
function sumar(a, b) {
    let resultado = a + b
    return resultado;
}

// La suma tomara como valor el resultado de la funcion
var suma = sumar(a, b);
```

Scope

ACADEMY by NUMEN

Por scope entendemos el alcance o ámbito de referencia de las variables. El alcance determina la zona del programa donde la variable es definida y conocida. De acuerdo a este alcance, las variables utilizadas en los programas y/o funciones se clasifican en **locales** y **globales.**

VARIABLES LOCALES

Son aquellas que están declaradas y definidas dentro de un bloque de código o función, pudiendo acceder al valor de la misma solo en ese ámbito.

```
function sumar() {
    // Cuando la variable figura con un color mas opaco
    // es por que no esta siendo utilizada en ninguna parte
    // del codigo
    var resultado = a + b;

    // Aqui dentro podemos acceder al valor resultado de la suma

    console.log(resultado);
}

// Error de consola
// No se puede acceder a la variable resultado
// Sera posible acceder a su valor dentro del bloque de la funcion
console.log(resultado)
```

VARIABLES GLOBALES

Son aquellas cuyo valor puede ser obtenido y alterado desde cualquier parte del código.

```
var resultado = 3;

function sumar(a, b) {
    resultado = a + b;
}

console.log(resultado);

sumar(2, 3);

// Al estar 'resultado' declarado por fuera de la función,
// al invocar sumar() estamos modificando directamente
// a esa variable reasignando su valor
console.log(resultado);
```

var, let y const

ES6 nos trae 2 nuevos tipos de variables que vienen a solucionar algunos inconvenientes que presentaba la variable **var**. En esta demostración jugaremos un poco con el **console.log** viendo que imprime dependiendo de donde lo coloquemos. Esto irá arrojando varias diferencias que los sorprenderán si no están aún interiorizados con esto.

¡A JUGAR!

```
var musica = "Pop"
console.log(musica)

{
  var musica = "Rock"
}

var musica = "Pop"
```

```
let musica = "Pop"
console.log(musica)

{
  let musica = "Rock"
}

let musica = "Pop"
```

```
const musica = "Pop"
console.log(musica)

{
  const musica = "Rock"
}

const musica = "Pop"
```

Tipos de funciones

ACADEMY by NUMEN

FUNCIONES ANÓNIMAS

Son aquellas que no tienen un nombre asociado.

En este caso, las usaremos en lo llamado **function declaration (declaración de función)**, donde asignaremos como valor la **función anónima** en una constante. Por convención se declaran con **const** para asegurarnos de que su valor no será re-asignado durante la ejecución del código con otro tipo de dato, perdiendo su propiedad de función.

Se invocan igual que las funciones vistas anteriormente.

Pueden recibir o no parámetros.

```
const saludar = function (nombre) {
    console.log("Hola " +   nombre)
}

saludar("Rocio");
```

Tipos de funciones

Funciones **declaradas**:

Una función declarada puede invocarse en cualquier parte de nuestro código, incluso antes de que la función sea declarada. Una función declarada tiene la siguiente pinta:

```
function funcionDeclarada() {
    console.log("Hola Mundo")
}
```

Funciones **expresadas**:

Una función expresada es un tipo de función que se le ha asignado como valor a una variable y que no puede invocarse antes de su definición.

Una función expresada sería algo así:

```
var funcionExpresada = function () {
    console.log("Adios Mundo")
}
```

Arrow Functions

Las funciones flecha son bastante parecidas a las funciones clásicas solo que con algunas cuantas ventajas.
Algo a remarcar de este tipo de funciones es que solo pueden usarse como funciones expresadas o anónimas, pero no como funciones declaradas.

```
// Función flecha clásica
const funcionFlecha1 = (a, b) => {
  return a + b
}
```

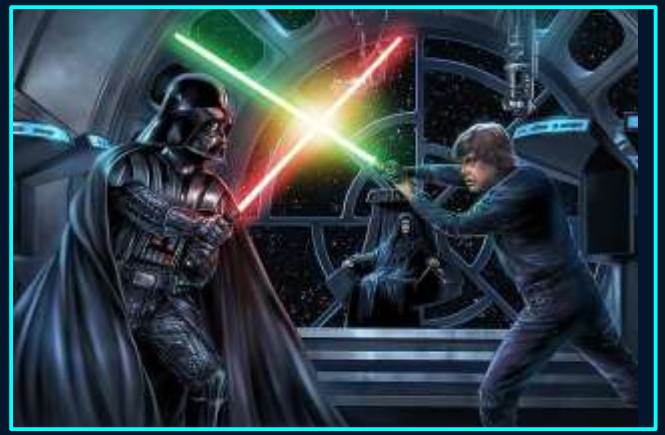
```
// Función flecha de único parámetro y única instrucción
const funcionFlecha2 = numero => numero * numero
```

Estructuras de control dentro de funciones

ACADEMY by NUMEN

if - else dentro de una función

Cuando colocamos estructuras de control dentro de una función, cada caso requerirá retornar un valor. Veamos cómo se vería una estructura if - else.



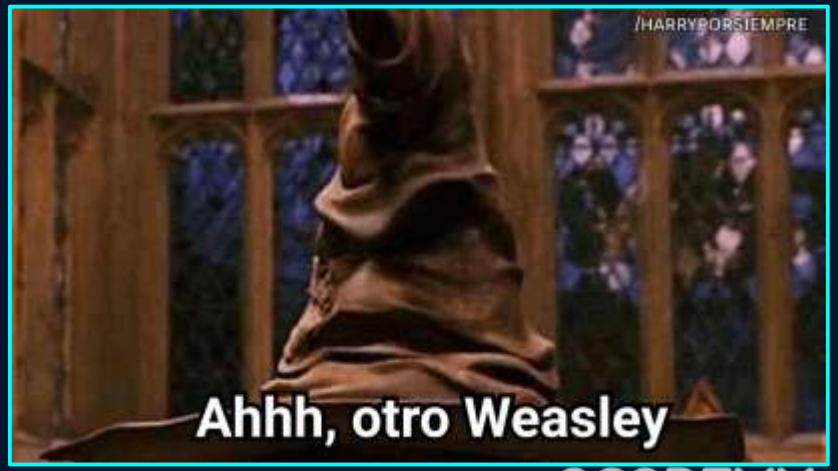
```
const ladoDeLaFuerza = (elegir) => {
    if (elegir === "oscuro") {
        return "Eres un Sith"
    } else if(elegir === "luminoso") {
        return "Eres un Jedi"
    }
    return "Eres un personaje irrelevante"
}
```

Veamos el switch-case en una función

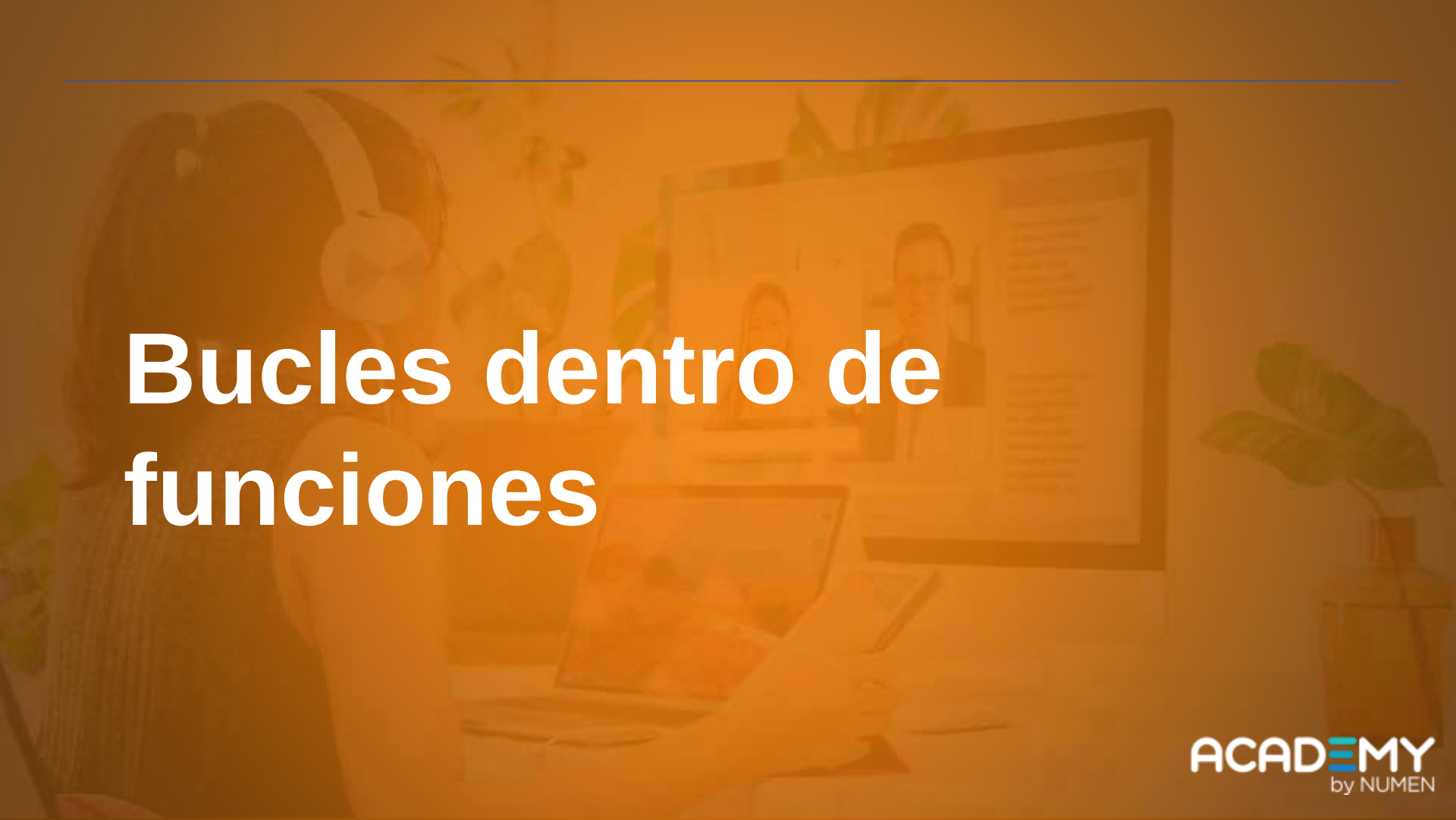
```
const sombreroSeleccionador = (test) => {
    switch (test) {
        case valienteYAtrevido:
            return "Griffindor";
        case justicieroYAmable:
            return "Hufflepuff";
        case argumentativoYAnalitico:
            return "Ravenclaw";
        case astutoEInescrupuloso:
            return "Slytherin";
        default:
            return "Azkaban"
    }
}
console.log(sombreroSeleccionador(numeroAleatorio)
)
```

```
const valienteYAtrevido = 1,
    justicieroYAmable = 2,
    argumentativoYAnalitico = 3,
    astutoEInescrupuloso = 4;

const numeroAleatorio = Math.ceil(Math.random()
* 5)
```



Bucles dentro de funciones

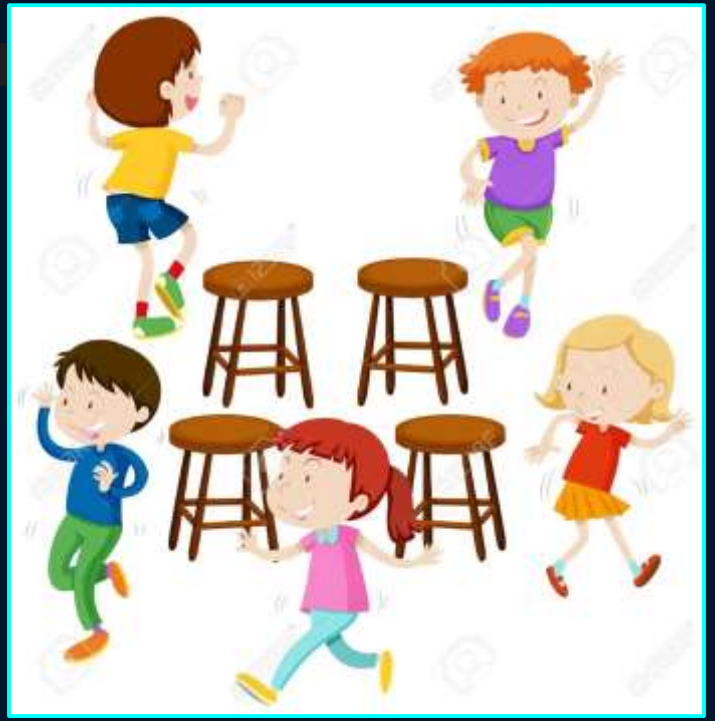


ACADEMY by NUMEN

veamos el bucle for en una función

```
const juegoDeLasSillas = (alumnos, sillas) => {
    for (let i = sillas; i > 1; i--) {
        console.log("Comienza ronda")
        alumnos = alumnos - 1
        if (alumnos === 2) {
            alumnos = alumnos - 1
        }
    }

    return alumnos
}
```



```
const juegoDeLasSillas = (alumnos, sillas) => {
    // Si hay igual cantidad de sillas que de alumnos
    if(sillas === alumnos) {
        sillas = sillas - 1
    }

    for (let i = sillas; i > 1; i--) {
        console.log("Comienza ronda")

        // Si los alumnos son muchos más que las sillas
        if (alumnos > sillas + 1) {
            alumnos = sillas
            alumnos = alumnos + 1
        }

        alumnos = alumnos - 1

        if (alumnos === 2) {
            alumnos = alumnos - 1
        }
    }
    return alumnos
}
```

Mejoremos un poco nuestra programación agregando algunas validaciones.

ACADEMY
by NUMEN