Funciones y Ciclos

Funciones

{desafío} latam_

Codificar un programa utilizando funciones para la reutilización de código acorde al lenguaje Javascript.

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

Te encuentras aquí

{desafío} latam_

¿Qué aprenderás en esta sesión?

- *Identificar las características y utilidad de programar funciones en un programa.*
- *Distinguir el alcance de una variable dentro y fuera de una función acorde al lenguaje Javascript.*
- *Codificar un programa invocando funciones personalizadas para resolver un problema acorde al lenguaje Javascript.*

{desafío} latam_

Según lo aprendido hasta ahora, ¿qué es una función?

Según el siguiente código, ¿cuál es el parámetro de la función?

```
function saludo (nombre) {
   document.write("¿Hola " + nombre + " cómo estás?");
}
```

{desafío} latam_

/* Funciones */

Funciones

Tipos de funciones

- Existen dos tipos de funciones, las que vienen incluídas en JavaScript (también llamadas built-in functions), como por ejemplo console y Math:

```
console.debug(Object);
console.info(Object);
console.error(Object);
console.log(Object);

Math.random();
```

- Y las otras, son las que podemos definir nosotros:

```
function hola(nombre){
     console.log(`Hola ${nombre}.`)
}

hola('Javier');    // => Hola Javier
```

Funciones anónimas

Otras formas de desarrollar funciones

- No todas las funciones poseen un nombre, sino que también existen las funciones anónimas, las cuales son asignadas (de ser necesario) a una variable que otorgará el nombre para poder ser invocada posteriormente:

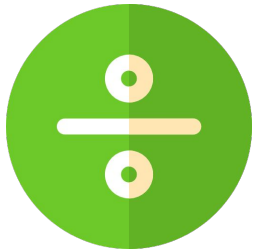
```
const nombre = function (parametro1, parametro2, ...) {
    // ...
}
nombre();
```

Demostración - “Funciones anónimas”

Ejercicio guiado

Funciones anónimas

- Solicitar al usuario que ingrese dos números enteros, y dentro de una función anónima se realice la división de ambos números, retornando y mostrando, en el mismo llamado de la función, el resultado como una variable. Para ello, sigamos los pasos:



Ejercicio guiado

Funciones anónimas

- **Paso 1:** En el archivo funciones.js debes armar la estructura de la función anónima, es decir, sobre una variable llamada “divide” declaramos la función.

- Luego inicializar dos variables donde se almacenarán los datos ingresados por el usuario y serán pasados como argumento en el llamado a la función, en este caso, podemos llamar a las variables “num1” y “num2” y mediante la función “prompt()” solicitamos al usuario ingresar los datos.

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

let divide = function (num1,num2) {
     //proceso
}
```

Ejercicio guiado
Funciones anónimas

- **Paso 2:** Realizar el llamado a la función pasando los argumentos, en este caso, los dos números ingresados por el usuario y agregamos la función dentro de la estructura del mensaje con un document.write():

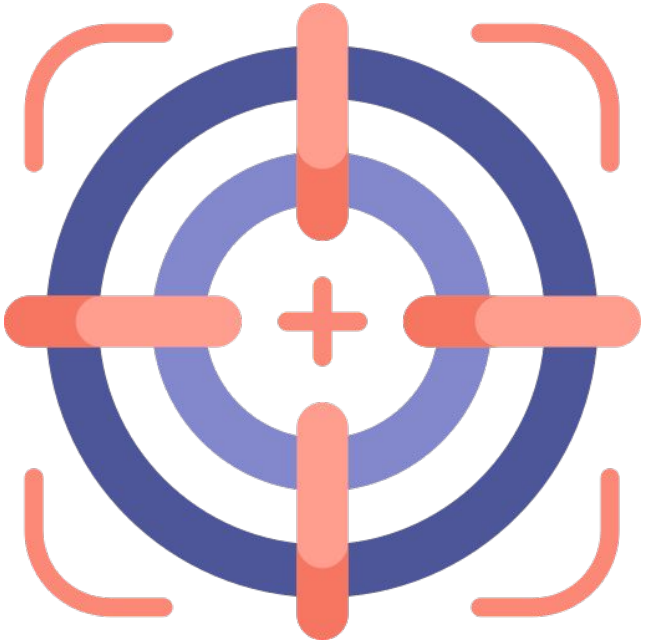
```
var num1 = prompt("Ingrese el primer número: ");
var num2 = prompt("Ingrese el segundo número: ");

let divide = function (a,b) {
    let resultado = parseInt(a) / parseInt(b);
    return resultado;
};

document.write("Resultado de la división: "+divide(num1,num2));
```

/* Alcance en una función */

Alcance en una Función

- El scope o alcance de una variable en JavaScript, se define como el espacio o segmentos de código donde esa función es conocida.

- El scope decide a qué variables de tu programa tienes acceso directo. Sin embargo, la misma función delimita hasta donde conocen sus propias variables y hacia dónde hacen referencia estas mismas.



{desafío} latam_

Demostración - “Alcance en una función”

Ejercicio guiado

Alcance en una función

Demostrar cómo opera el alcance en las funciones, para esto: crear dos variables con el mismo nombre ubicadas en distintos espacios de código (dentro y fuera de la función) y mostraremos su valor.

{desafío} latam_

Ejercicio guiado

Alcance en una función

- **Paso 1:** En el archivo funciones.js debemos agregar el código necesario para inicializar la variable “miVariable” con el número 10, luego crear una función llamada “miFuncion” que no recibirá ni retorna ningún tipo de valor, pero sí tendrá el inicio de la variable “miVariable” esta vez con el número 5 como valor inicial, luego dentro de ella mostramos la variable recién creada dentro de la función con ayuda de un console.log().

```
var miVariable = 10; // variable global

function miFuncion () {
    var miVariable = 5 // se declara una "nueva" variable local al usar
'var'
    console.log(miVariable); // se muestra la variable en la consola
};
```

Ejercicio guiado

Alcance en una función

- **Paso 2:** Fuera de la función realizaremos un console.log() de “miVariable” y el llamado a la función “miFuncion”. Por último, hacemos nuevamente el llamado mediante un console.log() de “miVariable”.

```
var miVariable = 10;  // variable global

function miFuncion () {
    var miVariable = 5 // se declara una "nueva" variable local al usar 'var'
    console.log(miVariable); // se muestra la variable en la consola
}

console.log(miVariable);
miFuncion(); // se hace el llamado de la función
console.log(miVariable);
```

Ejercicio guiado

Alcance en una función

- Por ende, el resultado de la ejecución del código anterior sería:

```
10
5
10
```

- Del resultado anterior podemos entender que el primer `10` corresponde a la declaración de `var miVariable = 10` que se encuentra al comienzo de nuestro código, el valor `5` corresponde al `console.log` de la función `miFuncion`, la cual en su interior declara otra variable con valor `5` y el último valor de `10` corresponde al valor de `miVariable` declarado al principio igual que el primer 10.

Ejercicio guiado

Alcance en una función

- **Paso 3:** ¿Qué pasaría si en el código anterior, realizamos un pequeño cambio dentro de la función y no inicializamos la variable con la palabra reservada var y solamente le cargamos el número 5?, como se muestra en el siguiente código:

```
var miVariable = 10; // variable global

function miFuncion () {
    miVariable = 5  // Se hace uso de la variable "global" 'miVariable'
    console.log(miVariable);
}

console.log(miVariable);
miFuncion();
console.log(miVariable);
```

/* Alcance de una Función dentro de una Función */

Alcance de una Función dentro de una Función

- Existe un concepto en informática llamado recursividad, que en palabras simples consta de hacer referencia o definir un elemento con base en sí mismo.

- En las funciones, consiste en llamar nuevamente a la función desde ella misma de forma recursiva, por lo tanto, se pueden hacer ciclos repetitivos mediante la recursividad de funciones y con la ayuda de condicionales, para ir repitiendo una secuencia hasta que cierto valor llegue a un punto.

Demostración: “Suma de términos”

Ejercicio guiado

Suma de términos

Calcular la suma de todos los términos almacenados en una variable con los datos del tipo arreglo del 1 al 9.

- **Paso 1:** En el archivo funciones.js debes armar la estructura de la función e inicializar una variable con datos del tipo arreglo, en este caso la podemos llamar “nums” e inicializarla con los valores del 0 al 9 dentro de un array: [0,1,2,3,4,5,6,7,8,9].

```
const nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

function suma (nums) {
}

console.log(sum(nums))
```

{desafío} latam_

Ejercicio guiado

Suma de términos

- **Paso 2:** Dentro de la función, se debe crear una estructura de control de flujo con “if”, para comprobar si ya no existen números dentro del arreglo para sumar, en el caso de no existir se debe retornar cero y detener la recursión, esto se puede lograr midiendo la longitud de la variable que contiene el arreglo con la instrucción “length”.

```
const nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

function suma (nums) {
    // La Recursión se detiene cuando un array está vacío
    if (nums.length < 1) { // con el length se mide la cantidad de
espacios que tiene la variable nums
        return 0;
    }
```

Ejercicio guiado

Suma de términos

- **Paso 3:** Para poder finalizar el proceso en la función, se debe ir extrayendo y eliminando a la vez los números de la variable “nums” que contiene el arreglo de números. Esto se puede lograr mediante el método “shift()”.

```
const nums = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9];

function suma (nums) {
    // La Recursión se detiene cuando un array está vacío
    if (nums.length < 1) { // con el length se mide la
cantidad de espacios que tiene la variable nums
        return 0;
    }
    // El método shift() remueve el primer elemento del array
   // y retorna ese valor. Este método cambia la longitud del
array
    var valor = nums.shift();

    // retornando en cada pasada la suma de los primeros
valores que se van removiendo
    return valor + suma(nums);
}
console.log(suma(nums))
```

/* Alcance de variables declaradas let y var */

Alcance de variables declaradas let y var

- Antes de ES6 sólo se podían declarar variable usando `var` las cuales al momento de ser usadas dejaban a nuestras variables disponibles fuera del contexto o bloque donde son declaradas, debido a que `var` tiene ambiente de función, esto significa que cuando se ejecuta el código y se detecta una variable `var` , JavaScript la "eleva" (hoisting)

- Con ES6 nace la definición de variables con la palabra protegida **let** que permite definir variables que solo existen dentro del contexto o bloque donde se declaró.

- Usar **let** o **var** depende de cada situación y problema a resolver, ya que en algunos casos no importará si la variable se mantiene fuera del bloque, pero hay otros en que sí y en ese caso debemos usar **let** para definir.

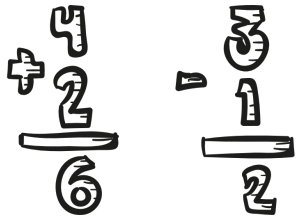
Demostración - “Alcance de variables”

Ejercicio guiado

Alcance de variables

Desarrollar un código en JavaScript que permita calcular la suma y la resta de dos números por funciones separadas, pero manteniendo el mismo nombre de las variables en ambas operaciones.

Además, los datos ingresados por el usuario se deben guardar en variables separadas dentro de cada función con el mismo nombre para las variables.



{desafío} latam_

Ejercicio guiado

Alcance de variables

- **Paso 1:** En el archivo funciones.js, se debe armar la estructura de las funciones, una para la suma y otra para la resta, ambas funciones no van a recibir ningún tipo de datos, por lo tanto, en la función para la suma se crearán dos variables con “let” para guardar los números solicitados al usuario, así mismo se realizará la suma de ambos números y se retornará con el resultado. La variable para almacenar el resultado de la suma y resta debe ser inicializada fuera de las funciones con var.

```
function suma() {
      let num1 = prompt("Ingrese un primer número para al suma");
      let num2 = prompt("Ingrese un segundo número para al suma");
      resultado = parseInt(num1) + parseInt(num2);
      return resultado;
};
```

Ejercicio guiado
Alcance de variables

- **Paso 2:** Realizada la función de la suma, pasemos a realizar la función de la resta la cual es muy parecida, lo único que se debe considerar es que esta vez se restan los dos valores ingresados por el usuario, como se muestra a continuación:

```
function resta() {
      let num1 = prompt("Ingrese un primer número para al resta");
      let num2 = prompt("Ingrese un segundo número para al resta");
      resultado = parseInt(num1) - parseInt(num2);
      return resultado;
};
```

Ejercicio guiado

Alcance de variables

- **Paso 3:** Para finalizar el ejercicio, solo se debe crear la variable para el resultado de manera global y hacer el respectivo llamado a ambas funciones:

- Al ejecutar el código anterior, el resultado obtenido dependerá del valor ingresado por el usuario.

```
function suma() {
      let num1 = prompt("Ingrese un primer número
para al suma");
      let num2 = prompt("Ingrese un segundo número
para al suma");
      resultado = parseInt(num1) + parseInt(num2);
      return resultado;
};

function resta() {
      let num1 = prompt("Ingrese un primer número
para al resta");
      let num2 = prompt("Ingrese un segundo número
para al resta");
      resultado = parseInt(num1) - parseInt(num2);
      return resultado;
};
var resultado;
document.write(`El resultado de la suma es:
${suma()} <br>`);
document.write(`El resultado de la resta es:
${resta()}`);
```

/* Función Arrow */

Función Arrow

- Con ES6 también nacen las arrow functions que es una forma simplificada de escribir o definir una función, incluso el poder tener una función en una sola línea de código.

- Ahora con ES6 podemos escribir funciones eliminando la palabra function y usando =>.

```
const nombre = (parametro1,
parametro2, ...) => {
    // ...
}
nombre();
```

Demostración - “Arrow Functions”

Ejercicio guiado

Arrow Functions

- En el siguiente código, se solicita al usuario ingresar un número entero para enviarlo como argumento en el llamado a la función y recibirlo como parámetro para poder comparar el valor mediante una estructura condicional.

- Realizar la transformación a funciones flecha o arrow de ES6:

```
var num = prompt("Ingrese un numero entero");
var resultado = 0;

function verificar(numero) {
     if (numero > 0) {
          resultado = "positivo";
     } else if (numero < 0) {
          resultado = "negativo";
     } else if (numero === 0) {
          resultado = "nulo";
     } else {
          resultado = "no es un número";
     }
     return resultado
}

alert(`El numero ingresado es:
${verificar(parseInt(num))}`);
```

Ejercicio guiado

Arrow Functions

- **Paso 1:** Lo primero que debemos observar es la estructura actual, en este caso, la función tiene parámetros y distintos procesos dentro de ella, por lo tanto, se deben utilizar los paréntesis “(parametro1...)” y las llaves “{proceso}” para separar los procesos dentro de la función original, agregando la flecha después del parámetro:

```
var num = prompt("Ingrese un numero entero");
var resultado = 0;

let verificar = (numero) => {  // cuerpo de la función donde debe
ir todo el proceso
}
```

Ejercicio guiado

Arrow Functions

- **Paso 2:** Agregamos toda la estructura condicional if-else-if dentro de la función, almacenando el mensaje correspondiente en la variable global y retornando el resultado dentro de esa variable.

```
var num = prompt("Ingrese un numero
entero");
var resultado = 0;

let verificar = (numero) => {
     if (numero > 0) {
          resultado = "positivo";
     } else if (numero < 0) {
          resultado = "negativo";
     } else if (numero === 0) {
          resultado = "nulo";
     } else {
          resultado = "no es un número";
     }
     return resultado
}

alert(`El numero ingresado es:
${verificar(parseInt(num))}`);
```

/* Comparativa entre ES5 y ES6 */

Comparativa entre ES5 y ES6

Característica	ES5	ES6

Retornar un número	`function retornaNum()`
	`{`
	`  return 3`
	`}`
		`let retonarNum = () =>`
		`3;`

Retornar un número de una función con parámetros	`function sumarDos(num)`
	`{`
	`  return num + 2;`
	`}`
		`let sumarDos = (num) =>`
		`num + 2;`

Alcance de bloque (let)	No existe.	`let numero = 1;`
Template string	`let miNombre = "Jon Doe"` `let cadena = "Hola " +` `miNombre;`	`let miNombre = "Jon Doe"` `let cadena = `Hola` `${miNombre}`;`

¿Existe algún concepto que no hayas comprendido?

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

Próxima sesión...

- *Codificar una rutina JavaScript, aplicando las diferentes formas de declarar funciones para resolver el problema planteado.*
- *Desarrollar algoritmos utilizando ciclos de instrucciones if/else y ciclos anidados para resolver un problema de baja complejidad.*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales