Flujo (Parte II)

Control de Flujo y Ciclos

Competencias

- Entender qué es el control de flujo en programación.
- Memorizar qué son los Ciclos y para qué sirven.
- Aplicar Ciclos a ejemplos prácticos.

Introducción

Es posible realizar un programa completo utilizando solamente variables, constantes y operadores en una sucesión lineal de instrucciones básicas, sin embargo, esto puede traer complicaciones si es necesario repetir un proceso múltiples veces, por ejemplo, sumar los 100 primeros números ingresados por un usuario, con las herramientas que conocemos podemos crear un par de 2 variables y realizar la pregunta “Ingrese número” cien veces, sin embargo, el trabajo se hace un tanto repetitivo ¿Escribir las mismas 3 líneas de código 100 veces? Qué sucede si además es necesario seguir una ruta diferente dependiendo de una variable de entrada como la edad o si tiene una sesión activa, etc.

Es por ello que existen las estructuras de control de flujo y ciclos, si utilizamos estructuras de control de flujo, los programas dejan de ser solamente una sucesión lineal de instrucciones para convertirse en programas *inteligentes* que pueden tomar decisiones en función del valor de las variables.

Operadores y Comparadores

Los operadores son la forma con la que efectuamos acciones u operaciones con las variables y los valores.

Los operadores más comunes dentro de Javascript son los siguientes:

- **Asignación:** =

```
var numero = 2;
```

El operador de “=” se ocupa para *asignar*, primero se calcula el valor del lado *derecho* del operador “=” (valor fuente) y luego se pone dentro de la variable al lado *izquierdo* del operador (la variable objetivo).

- **Matemática:** + (adición), - (sustracción), * (multiplicación), / (división)

Operadores que se ocupan para realizar cálculos matemáticos.

```
var numero = 2;
numero + 2;
numero * 4;

console.log(numero); // nos devuelve 2
```

- **Asignación compuesta:** +=, -=, *=, /=

Estos son operadores que combinan asignación y una operación matemática en un solo operador.

```
var numero += 2 // es lo mismo que numero = numero + 2
```

- **Incrementar/Decrementar:** ++, --

Estos operadores se ocupan bastante cuando queremos subir o bajar en 1 sola unidad alguna variable. Especialmente útiles cuando estamos dentro de un *loop*.

```
var numero++;  // es lo mismo que numero = numero + 1

for (let i = 0; i <= 9; i++) {
  console.log( i );
}
// en este loop de ejemplo, en la consola veremos como resultado 0 1 2 3 4 5 6
7 8 9
```

- **Igualdad:** == (sueltamente iguales), === (estrictamente iguales), != (sueltamente desiguales), !== (estrictamente desiguales).

Javascript tiene la particularidad de que es bastante permisivo con la forma en ocupamos los operadores entre variables y sus diferentes tipos, lo que nos permite realizar operaciones como esta:

```
// mismo valor, diferente tipo de dato (Number vs String)
console.log(22 == '22'); // true
console.log(22 === '22'); // false
```

La principal diferencia entre el operador == y ===, es que este último compara no solo el valor de dos variables, sino también verifica su tipo de dato. Para el caso del ejemplo, ambos valores son 22, pero el tipo de dato difiere: 22 es Number, mientras que '22' es un String. Luego, para la comparación con ==, este solo pone atención al valor, por ende el resultado es true, mientras que ===, también se fija en el tipo de dato, al ser diferente Number y String, el resultado es false.

A tener en cuenta que para el proceso de comparación de == JavaScript realiza una 'coerción de tipo', es decir, antes de la comparación intenta convertir ambas variables a un tipo común (como se vio en el ejemplo anterior), esto da como resultado postivo para comparaciones como la que vemos a continuación.

```
console.log(false == 0); // true
console.log(false === 0); // false
```

Debido a esto, tenemos que tener mucho cuidado con la manera en que manejamos los *tipos de datos* de nuestros *valores*, ya que Javascript nos va a permitir sumar *números* con *strings* y nos va a permitir compararlos con operadores de igualdad.

Por esta misma razón, herramientas más avanzadas como los frameworks sugieren hacer uso de === para evitar este tipo de problemas.

Nota: es necesario tener especial cuidado en el uso de ciertos operadores como la suma para tipos de datos diferentes, debido a que genera comportamientos no deseados como por ejemplo:

```
console.log(10 + 2); // 12
console.log(10 + '2'); // '102'
```

- **Comparación:** < (menor que), > (mayor que), <= (menor o igual a que), >= (mayor o igual a que).

Es lo mismo que vimos en matemáticas en el colegio, ocuparemos estos operadores comparativos para efectuar lógica basada en valores numéricos, como por ejemplo en esta *declaración condicional if:*

```
const primerValor = 22;

if ( primerValor > 21 ) {
  console.log('mi variable es mayor a 21');
}
else {
  console.log('mi variable menor o igual que 21');
}
// nos va a devolver 'mi variable es mayor a 21'
```

- **Lógico:** && (y), || (ó)

Estos operadores se ocupan bastante cuando ocupamos *declaraciones condicionales*, donde necesito que se cumplan ciertas condiciones para ejecutar un código en particular.

En el caso de “&&” (que se lee como *and ó doble ampersand)* este es un operador que va a preguntar si se está cumpliendo el caso de si izquierda tanto como el de su derecha, por ejemplo:

```
const nombre = 'pedro'
const edad = 22

if (nombre === 'pedro' && edad === 22){
  console.log('el nombre es pedro y su edad es 22')
}
// las condiciones se cumplen y nos devuelve 'el nombre es pedro y su edad es
22'
```

Acá estamos ocupando una *declaración condicional* para preguntar si la variable nombre es *estrictamente igual* a pedro Y si su edad es *estrictamente igual* a el *número* 22.

Dado ambas condiciones son *verdaderas*, cada una de las condiciones a ambos lados del “&&” nos devuelve *True* y se ejecuta el bloque de la *declaración condicional.*

De forma similar, el operador “||” (que se lee como “ó” u OR) también va a preguntar a su izquierda y a su derecha, con la diferencia que sólo va a necesitar que uno de estos 2 sea *True* para ejecutar el bloque de la *declaración condicional:*

```
const nombre = 'pedro'
  const edad = 22

if (nombre === 'pedro' || edad === 10){
  console.log('una de las 2 condiciones son verdaderas')
}
// las condiciones se cumplen y nos devuelve 'una de las 2 condiciones son
verdaderas'
```

La estructura If

La estructura más utilizada en JavaScript y en la mayoría de lenguajes de programación es la estructura if. Esta estructura se emplea para tomar decisiones en función de una condición. Su definición es la siguiente:

```
if(condicion) {
   ...
 }
```

Si la condición se cumple, se ejecutan todas las instrucciones que estén dentro de {...}. Si la condición no se cumple, no se ejecuta la instrucción contenida dentro de las llaves {...} y la aplicación continua ejecutando el resto de instrucciones del código.

Por ejemplo:

```
var mostrarMensaje = true;

if(mostrarMensaje = true) {
 alert("Es verdadero");
}
```

En este ejemplo, declaramos una variable de tipo Boolean. Ya que contiene un valor verdadero (true). La condición lo que está haciendo es comparando si el valor de la variable “mostrarMensaje” es igual o no a true. Como los dos valores coinciden, la igualdad se cumple y por tanto la condición es cierta, su valor es true y se ejecutan las instrucciones contenidas en ese bloque del if.

Ahora haremos un ejercicio muy entretenido para que practiquemos la estructura de flujo If.

Nos creamos un archivo .html llamado “index.html”, también creamos nuestro archivo .js, el cual en este caso lo llamaremos “javascript.js”. No olvides incluir en el .html el archivo externo del .js como lo hemos hecho en los capítulos anteriores. De igual forma te dejamos la instrucción que debes ingresar al final de la etiqueta para que la recuerdes:

```
<script type="text/javascript" src="javascript.js"></script>
```

En ejemplo es el siguiente, ingresa este código en tu archivo .js :

```
var num1;
var num2;

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

if(num1 <= num2) {
 alert("El primer número ingresado NO ES MAYOR que el segundo número");
}
if(num2 >= 0) {
 alert("El segundo número que ingresaste es positivo, Mayor o igual que 0");
}
if(num1 != 0 || num1 < 0) {
 alert("El primer número que ingresaste es distinto de 0 o es negativo");
}
```

¿Qué hicimos?, Para comenzar declaramos dos variables, a las que nuevamente al igual que en ejemplos de capítulos anteriores le dimos y almacenamos un valor desde el método “prompt();”.

Luego hicimos tres condiciones. En la primera, si el primer número que ingresamos es menor o igual al segundo número ingresado, pasará a la instrucción dentro de las llaves y nos mostrará el mensaje.

La segunda condición, si el segundo número ingresado es mayor o igual a cero, este nos mostrará el mensaje de alerta que está dentro de las llaves de ese if.

Y en la tercera condición, si el primer número ingresado es distinto de cero o es menor que cero nos mostrará el mensaje de alerta que está dentro de las llaves.

En el caso que alguna de estas condiciones no se cumpla, no pasara por la instrucción dentro de las llaves de los if.

Estructura If … else

En gran parte de las ocasiones, las decisiones que debemos realizar no son del tipo "si se cumple la condición, hagámoslo; si no se cumple, no hagamos nada". Normalmente las condiciones suelen ser "si se cumple esta condición, hagámoslo; si no se cumple, hagamos esto otro".

Para esto, existe una variante de la estructura if llamada if...else. Su definición formal es de la siguiente manera:

```
if(condicion) {
   ...
 }
 else {
   ...
 }
```

Si la condición que tenemos declarada se cumple, se ejecutan todas las instrucciones que se encuentran dentro de las llaves del if. Pero si la condición es el caso contrario y no se cumple, se ejecutan todas las instrucciones contenidas en las llaves del else { }.

Por ejemplo:

```
var edad;

edad = prompt("Ingresa tu edad: ")

if(edad >= 18) {
 alert("Eres mayor de edad");
}
else {
 alert("Todavía eres menor de edad");
}
```

En este ejemplo que vimos, si el valor de la variable, el cual ingresamos mediante el método “prompt();” es mayor o igual que el valor numérico 18, la condición del if { } se cumple y por tanto nos mostrará el mensaje que nos dice que “Somos mayores de edad”. Sin embargo, si el valor de la variable “edad” es menor que 18, automáticamente se ejecutará la instrucción que está dentro de las llaves de else { }. En este caso, se mostraría el mensaje "Todavía eres menor de edad".

Pero no solamente esta estructura puede trabajar con tipos de datos numéricos sino que también con cadenas de texto.

Por ejemplo:

```
var nombre = prompt("Ingresa tu nombre completo: ")

if(nombre == "") {
 alert("Aún no nos has dicho tu nombre");
}
else {
 alert("Gracias! Ya hemos guardado tu nombre en {deafio} latam_");
}
```

La condición del if { } que vimos en este ejemplo se realiza mediante el operador ==, que es el que generalmente se emplea para comparar dos valores (no confundir con el operador = que se utiliza para asignar valores). En el ejemplo, si la cadena de texto almacenada en la variable “nombre”, la cual le mandamos el valor mediante el método “prompt ();” es vacía (es decir, es igual a "") se muestra el mensaje que definimos como instrucción dentro del if { }. De ser de otra forma, se muestra el mensaje definido en el bloque else { }.

La estructura if...else también se puede encadenar para realizar varias comprobaciones seguidas, por ejemplo puedes comprobar este código:

```
var nombre = "Gary";

if (nombre == "Gary") {
   alert("Hola Gary!");
}

else if (nombre == "Claudio"){
   alert("Hola Claudio!");
}

else {
   alert("¿Quién eres?");
}
```

En este ejemplo si te diste cuenta agregamos un else if { }. Que es como decir “pero si es este caso” . Se utiliza para realizar una cadena de comprobaciones seguidas.

Determinamos la variable “nombre” que almacenaba la cadena de texto “Gary”. Con el If teníamos la condición que si nombre era igual a “Gary” este mostraba un mensaje de saludo a Gary. Pero también si nombre era “Claudio”, también se ejecuta la instrucción de saludo. Pero sino era ninguna de estas condiciones pasaba automáticamente al else { } y mostraba un mensaje “¿Quién eres?.

Estructura Switch

Al momento de utilizar estructuras de condición, la estructura Switch en ciertas ocasiones puede ser más eficiente, ya que está especialmente diseñada para manejar de forma sencilla múltiples condiciones sobre la misma variable.

Su definición formal puede parecer un tanto compleja, pero la verdad es que su uso es muy sencillo:

```
switch (variable) {
   case valor_1:
           // instrucciones
       break;
   case valor_2:
           // instrucciones
       break;
   case valor_3:
           // instrucciones
       break

   default:
           // instrucciones
       break;
}
```

Ahora bien, haremos el siguiente ejemplo para ver cómo funciona esta estructura. Escribimos el siguiente código en nuestro archivo .js :

```
var respuesta = prompt("Escribe un Número del 1 al 4");

switch (respuesta) {
   case "1":
           alert("Escribiste el número uno");
       break;
   case "2":
           alert("Escribiste el número dos");
       break;
   case "3":
           alert("Escribiste el número tres");
       break
   case "4":
           alert("Escribiste el número cuatro");
       break
   default:
           alert("No es un número entre 1 y 4");
       break;
}
```

En este ejemplo, declaramos una variable con el nombre de “respuesta” a la cual le asignamos un valor que le preguntamos al usuario, en este caso un número entre el 1 y el 4. Luego con la estructura Switch lo que hacemos es evaluar la variable “respuesta” con los distintos casos. En el caso cuando el usuario escriba “1” se ejecutará un mensaje de alerta donde se señalará el número ingresado, y así

con cada caso.

Cada caso ser termina con un corte del proceso, y esto se hace mediante la palabra “break;”.

Cuando ya no colocamos más casos ingresamos la propiedad “default” que se ejecutará si es que ningún caso se haya cumplido.

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) {
   ...
}
```

La idea principal del funcionamiento de un bucle, ciclo o loop for es la siguiente:

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.

Analicemos esta explicación detalladamente con código, para que así entendamos a detalle este tan importante ciclo.

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

La **inicialización:** es donde nosotros vamos a declarar los valores iniciales de las variables que controlan la repetición:

```
var i = 0;
```

Por tanto, en primer lugar lo que tenemos que hacer es declarar una variable i a la cual le asignamos el valor de 0. Es decir, al iniciar el ciclo la variable i su valor será de 0.

La **condición:** es donde establecemos cuantas veces se repetirá el ciclo y donde se decide si continúa o se detiene la repetición.

```
i < 5;
```

En este caso, mientras la variable i su valor sea menor de 5 el bucle o ciclo se ejecuta indefinidamente. Como la variable i se ha inicializado en un valor de 0 y la condición para salir del bucle es que i sea menor que 5, si no se modifica el valor de i de alguna forma, el ciclo se repetiría indefinidamente.

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

```
i++
```

Ahora bien en este caso, el valor de la variable i se incrementa en una unidad después de cada repetición. Esta parte de actualización donde se incrementa la variable i se ejecuta después de las ejecución de las instrucciones que incluye el ciclo for.

Así, en este caso durante la ejecución de la quinta repetición el valor de i será 4. Después de esta quinta ejecución, se actualiza el valor de la variable i, que ahora valdrá 5.

Como la condición es que i sea menor que 5, la condición entonces ya no se cumple y las instrucciones del for no se ejecutan una sexta o séptima vez.

Veamos un sencillo ejemplo para ver en ejecución un ciclo for.

Creamos al igual que en los capítulos anteriores nuestros dos archivos para trabajar en los ejemplos prácticos, nuestro “index.html” y nuestro archivo javascript “ciclos.js”.

Agregamos el siguiente código a nuestro archivo “ciclos.js”

```
for(var i = 0; i <10; i++) {
   document.write("{desafio} latam_" + "<br>");
 }
```

Guardamos y luego ejecutamos nuestro “index.html” y nos aparecerá algo como esto:

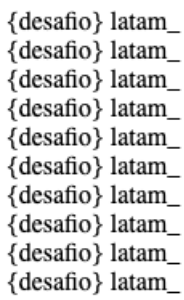

Imagen 1. Ejemplo de Ciclos.

¿Que hicimos aquí?

Imprimimos en pantalla 10 veces un texto.

Para poder hacer eso hicimos lo siguiente:

Declaramos nuestra variable i con un valor inicial de 0. Luego dijimos en la condición que el ciclo se ejecutará hasta que nuestra variable i sea menor que 10. Mientras eso se cumpla se ejecutará a continuación la instrucción que tenemos dentro de las llaves de nuestro ciclo for.

```
document.write("{desafio} latam_" + "<br>");
```

Lo que este código hace es simplemente escribir en nuestro documento html el texto que nosotros indiquemos entre los ( ). Si te das cuenta concatenamos el texto “{desafio} latam_” con la etiqueta: `<br>`

Básicamente para generar así un salto de línea. Ya que “document.write();” nos imprime texto de forma lineal.

Luego de entrar a esta instrucción, comienza nuevamente el ciclo, y toma la actualización con el incremento en uno de la variable i entra en acción por primera vez “i++”.

Entonces ahora la variable que un inicio valía 0 ahora vale 1. Pasamos nuevamente por la condición y esta es verdadera ya que la variable i sigue siendo menor que 10, y así sucesivamente seguirá haciendo el mismo proceso y al final terminará imprimiendo 10 veces el texto de la instrucción dentro del ciclo for.

Ahora para que notes la diferencia, este mismo ejemplo sin una estructura de ciclo for sería de esta forma:

```
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
document.write("{desafio} latam_" + "<br>");
```

Te puedes dar cuenta que si bien es cierto el resultado es el mismo, pero… ¿Te imaginas si queremos imprimir un texto 100 o 200 veces? sería una locura y nuestro código quedaría muy saturado.

Con nuestro ciclo for simplificamos bastante nuestro trabajo, prueba cambiando la condición y reemplaza el 10 por un 100. Sólo haciendo eso imprimimos 100 veces un texto con solo dos líneas de código.

```
for(var i = 0; i <100; i++) {
   document.write("{desafio} latam_" + "<br>");
 }
```

Ahora haremos un ejemplo muy interesante y divertido, haremos la tabla de multiplicar del número 6.

Borramos todo lo que habíamos escrito en nuestro archivo .js o si lo quieres puedes crear otro archivo y luego solamente lo incluyes al archivo .html

Escribimos el siguiente código JavaScript:

```
var numero = 6;

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

En este ejemplo lo que hicimos en un principio fue declarar una variable la cual llamamos “numero” que es la que almacena la tabla de multiplicar que queremos imprimir en pantalla, en este caso la tabla del 6.

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 automaticamente pasa a las instrucciones que tenemos dentro del for { }.

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 “numero” que vale 6.

Seguido de esto con el método “document.write( );” imprimimos en pantalla el valor de la variable “numero” que es 6, y lo concatenamos con un espacio vacío “ ” y luego con el signo de multiplicación “x”, seguido después también por el valor de ese momento de la variable “i”, concatenado de n espacio en blanco, el signo “=” y posteriormente el valor de la variable “resultado”, que es el resultado de la multiplicación.

Y así seguirá imprimiendo hasta que se cumpla la condición que en este caso es:

```
i <=10;
```

El resultado que tendremos en pantalla es el siguiente:

6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
6 x 10 = 60

Imagen 2. Ejemplo de condición.

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.

El bucle o ciclo While se escribe de la siguiente manera:

```
while (condición) {
   ...
}
```

Este ciclo funciona de la siguiente forma: Mientras se cumpla una condición y sea verdadera se ejecutan ciertas instrucciones.

En este ejemplo imprimimos en pantalla el listado de números del 0 al 10:

```
var num = 0;

while (num <= 10) { // condición
   document.write(num + "<br>"); // imprimimos el valor de la variable
   num++; // incrementamos la variable en 1
}
```

Otro ejemplo que podemos hacer con el ciclo while, imprimir los número pares hasta el 20:

```
var num = 0;

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

Ciclo Do While

También tenemos otra opción al momento de usar bucles. Tenemos a Do While, una variación del While.

Básicamente podemos hacer lo mismo, pero lo que diferencia el Do While del While es que este bucle siempre va ejecutar una instrucción al menos una vez. A diferencia del While, que primero se analiza la condición y si esta es falsa sale del bucle. Do While por su parte ejecuta la instrucción, luego ve la condición y si esta falsa sale inmediatamente del bucle.

Hagamos un ejemplo, escribamos este código y obtendremos:

```
do {
   var nombre = prompt("Ingresa tú nombre");
 }
while (!nombre);
document.write(nombre);
```

Este bucle nos obligará a introducir un nombre. Preguntará una y otra vez hasta que obtenga algo que no sea una cadena vacía. Aplicando el operador ! en la variable “nombre” lo convierte un valor a Booleano negándolo y todas las cadenas excepto " " se convierten a true. Esto significa que el bucle continúa corriendo hasta que demos un nombre que no sea una cadena vacía.

Funciones en JavaScript

Competencias

- Entender que son y para que se usan las funciones.
- Se capaz de dividir problemas en subprocesos.
- Aplicar funciones para resolver problemas de carácter repetitivo o que sea necesario segmentar.
- Identificar entradas y salidas de una función.

Introducción

A lo largo de los capítulos anteriores hemos aprendido desde lo más sencillo y básico del lenguaje de programación JavaScript hasta lo que ahora comenzaremos a ver. Hasta acá solamente hemos venido viendo datos, manipulandolos de cierta manera, almacenarlos en variables o constantes, utilizando condicionales y ciclos para mostrar ciertos datos.

Funciones

Comenzaremos a dar más vida a nuestro código con las funciones, ¿Pero qué son las funciones? Las funciones son fragmentos de código que podemos reutilizar cuantas veces nosotros queramos dándoles dinamismo para que estas ejecuten o hagan cosas por nosotros.

Para que entendamos de una manera muy simple y práctica, las funciones son como un libro de recetas. Por ejemplo, si nosotros queremos cocinar un arroz con pollo, teniendo en consideración que nunca hemos preparado en nuestra vida un arroz con pollo. ¿Qué es lo que hacemos? buscamos una receta que nos ayude e indique cómo preparar nuestro arroz con pollo .

Y esta receta tiene: ingredientes, pasos a seguir y al final hay un resultado esperado.

Ahora pensemos y vamos comparando este ejemplo, los ingredientes de la receta vendrían siendo los datos de entrada. Las instrucciones y los pasos que hay que seguir es la función. Y el resultado esperado serían los datos de salida.

Veamos el siguiente esquema para que nos quede mucho más claro este ejemplo:

Imagen 3. Ejemplo de Proceso.

El libro de recetas es igual cuando **declaramos una función**. Cocinar, seguir los pasos de la receta es **Ejecutar la Función.** Entonces las funciones son mini programas que cumplen una "función" determinada y que podemos reutilizar como y cuando queramos.

Una función su estructura es la siguiente:

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

Con el siguiente ejemplo muy básico veremos de qué manera estas se ejecutan:

```
function saludo () {
   document.write("¿Hola cómo estás?");
}     // declaramos la función

saludo();   // llamamos o ejecutamos la función
```

Declaramos una función con el nombre "saludo" y dentro de ella una instrucción para que nos imprima en pantalla un saludo. Luego después llamamos a la función por su nombre para que esta se ejecute.

Ahora bien, qué pasa por ejemplo si esta misma función de saludo queremos que muestre el nombre real del usuario que está ingresando a la página web. Para esto tenemos que ingresar unos parámetros a la función, ¿esto qué significa?, que a las funciones podemos pasarles parámetros, que digamos pueden ser variables, constantes, etc. que podemos enviar para que puedan ser trabajadas en la función.

Hagamos este ejemplo:

```
var nombre = prompt("Ingrese su nombre: "); // declaramos una variable para
que almacene el nombre del usuario.

function saludo (nombre) { // tomamos el parámetro de la variable.
   document.write("¿Hola " + nombre + " cómo estás?"); // concatenamos el
saludo.
}

saludo(nombre); // llamamos la función y le enviamos el valor de la variable
nombre.
```

Guarda y ejecuta tu archivo .html y verás como en el saludo aparece el nombre que ingresaste. ¿Genial verdad?

En estos ejemplos vimos de manera básica cómo declarar y llamar una función. Ahora veremos algo un poco más avanzado, donde haremos que la función devuelva un valor. Y para esto utilizaremos la sentencia “return”.

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

function suma (num1, num2) {
   var num1 = parseInt(num1);
   var num2 = parseInt(num2);
   var resultadoSuma = num1 + num2;
   return resultadoSuma;
   }
document.write(suma(num1, num2));
```

Lo que hicimos en este ejemplo fue declarar dos variables “num1” y “num2” las cuales almacenamos los valores que el usuario ingresa mediante el método prompt( );

Luego creamos nuestra función de suma, la que llamamos “suma” donde enviamos los parámetros “num1” y “num2”, ya dentro de nuestra función tuvimos que declarar unas variables para poder obtener los valores que ingresamos, ahí utilizamos el parámetro parseInt( ); lo que hace esto es convertir cadenas de texto en un valor entero. Ya que sin esto puedes probar quitando este parámetro parseInt y te darás cuenta que lo que se mostrará en pantalla serán los valores ingresados concatenados.

Ahora haremos el mismo ejemplo pero lo utilizaremos en un mini formulario, y lo haremos de la siguiente manera. Escribimos este código en nuestro 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(numero1, numero2) );">
</form>

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

Ahora agregamos el siguiente código en nuestro archivo .js :

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

En este ejemplo logramos declarar nuestra función “suma” y sus instrucciones correspondientes.

Utilizamos variables para almacenar dentro de la función obteniendo estos valores desde el formulario realizado en nuestro archivo “.html” . Los obtuvimos con “document.getElementById( );” y convertimos de ese valor ingresado como texto a un número entero con el mismo parámetro que usamos en el ejemplo anterior, parseInt( );

En nuestra función terminamos retornando un valor con return y el resultado de la suma.

Para finalizar el proceso, llamamos desde el archivo html a la función suma y mostramos el resultado en un mensaje de alerta.

Como te pudiste dar cuenta, vimos lo esencial y la base del lenguaje de programación JavaScript, lo que nos ayudará en nuestra carrera de Front-End para así lanzarnos con nuestra imaginación y hacer lo que queramos en nuestras páginas web. Con esta base y con tus ganas de seguir aprendiendo y practicando llegarás muy lejos a caminos de nuevas tecnologías y frameworks que quizás jamás lograste dimensionar que podrías llegar a dominar como desarrollador.