Introducción al lenguaje JavaScript (Parte II)

Control de flujos

Competencias

- Desarrollar rutinas utilizando operadores y control de flujo para resolver un problema.
- Refactorizar flujos de múltiples if y else utilizando operadores lógicos para facilitar la legibilidad del código.

Introducción

Es posible realizar un programa completo utilizando solamente variables, constantes y operadores en una sucesión lineal de instrucciones básicas, pero esto puede traer complicaciones si es necesario repetir un proceso múltiples veces. Por ejemplo, para sumar los 100 primeros números ingresados por un usuario con las herramientas que conocemos, podemos crear variables y realizar la pregunta “Ingrese número” 100 veces, sin embargo, el trabajo se hace repetitivo y conduce a errores. ¿Qué sucede si además es necesario seguir una ruta diferente dependiendo de alguna condición?

Es por ello que existen las estructuras de control de flujo y ciclos. Si utilizamos estas estructuras, 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. El manejar estos conceptos nos permitirá construir programas que respondan de mejor manera a los desafíos que se nos planteen. Aprender cómo manejar adecuadamente el control de flujo, es clave para desarrollar lógicas cada vez más complejas y cercanas a la realidad.

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**: =

El operador "=" se ocupa para asignar. Primero se calcula el valor del lado derecho del operador "=" (valor fuente) y luego la variable al lado izquierdo del operador (la variable objetivo).

```
var numero = 2;
```

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

Operadores que se ocupan de 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 operación matemática en un solo operador.

```
var numero += 2 // es lo mismo que numero = numero + 2
var numero -= 2 // es lo mismo que numero = numero - 2
var numero *= 2 // es lo mismo que numero = numero * 2
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. Representan una versión simplificada de la asignación compuesta.

```
var numero = 1;
```

```
numero++;
console.log(numero) // nos devuelve 2
numero--;
console.log(numero) // nos devuelve 1
```

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

JavaScript tiene la particularidad que es bastante permisivo con la forma en que 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 === 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 positivo 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 compararlos con operadores de igualdad.

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

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

- **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, pero ahora ocuparemos estos operadores comparativos para efectuar lógica basada en valores numéricos.

```
2 > 3; // false
2 < 3; // true
```

- **Lógico:**
  - && (y)
  - || (ó)
  - ! (negación)

Estos operadores se utilizan bastante cuando ocupamos declaraciones condicionales, donde se necesita 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 su izquierda tanto como el de su derecha, por ejemplo:

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

nombre === 'pedro' && edad === 22; // true
```

En la última línea del ejemplo anterior se verifica si la variable nombre es estrictamente igual a pedro y si su edad es estrictamente igual al número 22.

Dado que ambas condiciones son verdaderas, cada una de las condiciones a ambos lados del "&&" nos devuelve True.

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 que toda la sentencia sea verdadera.

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

nombre === 'pedro' || edad === 10; // true
```

El operador negación (!), por su parte, requiere de un único valor para funcionar, lo que hace este operador es invertir (o negar) el valor booleano de una variable o de una sentencia condicional. Así, si se niega un valor true el resultado obtenido es false y si se niega un valor false el resultado es true.

```
!(nombre === 'pedro'); // false
!(edad === 22); // true
```

Ejercicio guiado: Operadores y comparadores

Realizar ahora el siguiente ejercicio, sabiendo que "x" es igual a 20 y "z" es igual a 13, ¿Cuál será el resultado para la siguiente operación (x-z) > (z-x), utilizando operadores y comparadores?

Lo primero que podemos realizar es la operación de forma manual y después probarla en la consola del navegador para verificar lo realizado. Por lo tanto, sigamos los siguientes pasos:

- ***Paso 1:*** Sustituir los valores indicados dentro de la comparación, para x=20 y z=13, quedando: (20-13) > (13-20).

- ***Paso 2*:** Realizar la operación matemática: (7) > (-7).

- ***Paso 3:*** Analizar los resultados y ver si se cumple la condición de comparación, en este caso es mayor que ">", por lo tanto se podría leer: ¿siete es mayor que -7? La respuesta a esto sería verdadera, es decir, "true". Por lo tanto esa sería la salida en la consola del navegador al comprobar la condición.

- ***Paso 4:*** Implementar la operación completa en la consola, el uso de variables y el operador de asignación para guardar los valores indicados en las variables correspondientes:

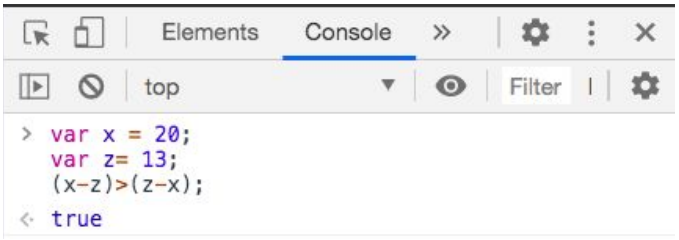

Imagen 1. Resultado mediante consola.
Fuente: Desafío Latam

Ejercicio propuesto (1)

Considerando las siguientes variables: `a = 10 y b = 5`, ¿cuál es el resultado de las siguientes operaciones lógicas?

```
a > 5 || b < 10
```

```
a == 10 && b == 5
```

```
(a > 5 || a < 15) && b > 0
```

```
!(a > 5) && b < 10
```

```
(a - 5) >= (a*b) - (b/a)
```

La estructura If

La estructura más utilizada en JavaScript y en la mayoría de los 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 con valor verdadero (true). La condición está 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 este bloque condicional.

Ejercicio guiado: La estructura if

Solicitar al usuario que ingrese dos números enteros y comparar si el primer número es menor o igual que el segundo. Luego, comprobar si el segundo número es mayor o igual que cero. Finalmente, comprobar si el primer número es menor que cero o distinto de cero.

- ***Paso 1:*** Crear un archivo .html llamado "index.html" y uno .js, el cual llamaremos "script.js" dentro de una carpeta en nuestro lugar de trabajo preferido. No olvidar incluir en el .html el archivo externo del .js como se ha hecho en los capítulos anteriores.

- ***Paso 2:*** De igual forma a continuación está la instrucción que se debe agregar al final de la etiqueta body:

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

- ***Paso 3:*** Ahora, en el archivo script.js, implementar las estructuras if necesarias para poder realizar las comparaciones indicadas en el enunciado, en este caso, serán tres estructuras if, una para comprobar si el número 1 es menor o igual que el número 2, otra para comprobar si el número 2 es mayor o igual a cero y la última para comprobar si el número 1 es distinto de cero o menor que cero. Si la respuesta es correcta o verdadera (true), utilizamos un alert para mostrar el mensaje correspondiente. En el caso que alguna de estas condiciones no se cumpla, no pasará por la instrucción dentro de las llaves de los if. Por lo tanto, al codificar lo anterior en el archivo script.js, quedaría:

```
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");
}
```

Ejercicio propuesto (2)

Realiza un programa en JavaScript implementando la estructura de control if, que solicite al usuario ingresar dos números y que compruebe si el número 1 es positivo, si el número dos es negativo y finalmente si el número 2 es mayor o igual que el número 1, indicando si se cumple cada condición mediante un alert.

Estructura If - else

En muchas ocasiones, lo que debemos realizar no son lógicas del tipo "si se cumple la condición, hagámoslo; si no se cumple, no hagamos nada". Normalmente, 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 la siguiente:

```
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 {desafío} 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). Por 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 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 el nombre era igual a “Gary” este mostraba un mensaje de saludo a Gary. Pero también si el 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?”

Ejercicio guiado: Estructura if-else

Solicitar al usuario que responda a una pregunta en específico, en este caso: “¿Quiere usted aprender a programar con JavaScript?”, si el usuario responde que “Si” o “SI” o “si”, enviar el mensaje: “Felicitaciones, ya eres parte de Desafío Latam”, de lo contrario: “Que lastima!!!… te esperamos”. Por lo tanto, sigamos los siguientes pasos:

- ***Paso 1:*** En el archivo script.js, se inicializa una variable para almacenar el dato ingresado por el usuario:.

```
var respuesta = prompt("¿Quiere usted aprender a programar con
JavaScript?: Si o No");
```

- ***Paso 2:*** La variable se utilizará para preguntar dentro del if si cumple o no con la condición y generar la respuesta adecuada dependiendo si el resultado de la condición es “true” o “false”

```
var respuesta = prompt("¿Quiere usted aprender a programar con
JavaScript?: Si o No");

if (respuesta == "Si" || respuesta == "si" || respuesta == "SI") {
      console.log("Felicitaciones, ya eres parte de Desafío Latam");
} else {
      console.log("Que lastima!!!... te esperamos");
}
```

Observando en detalle el ejercicio anterior, se puede apreciar como dentro una estructura if-else se realizan distintas comparaciones utilizando los operadores lógicos y de comparación en una sola línea de instrucción o código. Si vemos el resultado cuando el usuario ingrese "Si";

```
Felicitaciones, ya eres parte de Desafío Latam
```

En el caso de ingresar "No":

```
Que lastima!!!... te esperamos
```

Ejercicio propuesto (3)

Realiza un programa en JavaScript implementando la estructura de control if-else, que solicite al usuario indicar si desea aprender a programar con NodeJS, si el usuario indica que "Si" o "SI" o "si" indicar mediante un alert "Es hora de estudiar NodeJS", de lo contrario indicar mediante un alert "No quieres aprender NodeJS, sigue estudiando JS".

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 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, realizaremos 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 se termina con un corte del proceso y esto se hace mediante la palabra “break;”. Así mismo, cuando ya no colocamos más casos ingresamos la propiedad “default” que se ejecutará si es que ningún caso se ha cumplido.

Por legibilidad de código, a veces preferimos utilizar switch antes que múltiples if-else; si bien, el resultado es el mismo, visualmente es más sencillo de interpretar de esta manera. Puedes ver un ejemplo de cómo realizar esto en el documento **Material Apoyo Lectura - Transformar múltiples ciclos if-else en switch**, ubicado en “Material Complementario”.

Ejercicio guiado: Estructura switch

Se solicita a un usuario que ingrese un número del 1 al 7 y se indique mediante un mensaje el dia de la semana a cual pertenece ese número, para ello, sigamos los siguientes pasos:

- ***Paso 1*:** En el archivo script.js, armar la estructura del Switch e inicializar una variable donde se almacenará el dato ingresado por el usuario, en este caso la podemos llamar “num” y mediante la función “prompt()” solicitamos al usuario ingresar el número. Quedando de la siguiente forma:

```
var num = prompt("Escribe un Número del 1 al 7");

switch () {
  case "1":
      break;
  case "2":
      break;
  case "3":
      break
  case "4":
      break
  case "5":
      break
  case "6":
      break
  case "7":
      break
  default:
      break;
}
```

- ***Paso 2:*** Activamos la lógica correspondiente dentro de la estructura del Switch, colocando la variable dentro de los paréntesis del switch para que la estructura se encargue de evaluar la condición de la variable y comparar con los valores preestablecidos dentro de los "case":

```
var num = prompt("Escribe un Número del 1 al 7");
switch (num) {
  case "1":
      alert("El día de la semana es lunes");
      break;
  case "2":
      alert("El día de la semana es martes");
      break;
  case "3":
      alert("El día de la semana es miércoles");
      break
  case "4":
      alert("El día de la semana es jueves");
      break
  case "5":
      alert("El día de la semana es viernes");
      break
  case "6":
      alert("El día de la semana es sábado");
      break
  case "7":
      alert("El día de la semana es domingo");
      break
  default:
      alert("El número ingresado no corresponde a un día de la semana");
      break;
}
```

Generando el resultado en el navegador:

Imagen 2. Resultado en ventana emergente.
Fuente: Desafío Latam

Ejercicio propuesto (4)

Realiza un programa en JavaScript implementando la estructura de control switch, donde se solicita a un usuario que ingrese un número del 1 al 12 e indicar mediante un mensaje el mes del año al cual pertenece ese número. Si no pertenece a ningún mes indicar que el número no está permitido.

Uso de sentencias condicionales dentro de un bloque if

Existirán ocasiones en las que sea necesario verificar varias condiciones a la vez, donde todas tendrán que ser verdaderas para ejecutar cierto código. Supongamos que se ha creado una nueva política pública que beneficia a toda persona mayor de 18 años, pero menor de 40 años, que viva en las regiones 12 o 15 del país. Así, para verificar si una persona es beneficiaria, se puede utilizar el siguiente código:

```
if(edad >= 18) {
   if(edad <= 40) {
       if(region == 12 || region == 15) {
           // Persona apta para recibir el beneficio
       }
  }
}
```

Si bien este código funciona, no se recomienda anidar tantos if de manera consecutiva, ya que aumenta la complejidad del código y disminuye su mantenibilidad. Es por lo anterior que hay que modificar el código para mejorarlo, lo cual es posible de lograr usando comparaciones lógicas. Con esto, el código anterior puede ser reducido a:

```
if(edad >= 18 && edad <= 40 && (region == 12 || region == 15)) {
   // Persona apta para recibir el beneficio
}
```

Para aprender más sobre cómo codificar un programa en JavaScript utilizando estilos y convenciones de programación tales como indentación y estructura de código, puedes consultar el documento **Material Apoyo Lectura - La importancia de las buenas prácticas,** ubicado en “Material complementario”.

Ejercicio guiado: Uso de sentencias condicionales

Se le solicita al usuario ingresar la edad y responder a la pregunta, **"¿Quieres estudiar en Desafío Latam?":**

- ❏ Evaluar si la edad es mayor o igual a 18 y la respuesta es "si" o "Si" o "SI", responder con un mensaje de bienvenida, de lo contrario, con el mensaje: "Que bueno, pero debes estar acompañado por tu representante"
- ❏ Si su respuesta es "no" o "No" o "NO", responder con el mensaje: "Que lastima!!!... te esperamos pronto"
- ❏ En el caso de no ser ninguno de los datos mencionados anteriormente, indicar un mensaje: "Datos errados".

- ***Paso 1*:** En el archivo script.js debes armar la estructura del if e inicializar dos variables donde se almacenarán los datos ingresados por el usuario, en este caso las podemos llamar "num" y "respuesta" y mediante la función "prompt()" solicitamos al usuario ingresar los datos. Quedando de la siguiente forma:

```
var num = prompt("Ingresa tu edad actual");
var respuesta = prompt("¿Quieres estudiar en Desafío Latam?, Sí o No");
```

- ***Paso 2*:** Activamos la lógica correspondiente dentro de la estructura if-else-if, colocando las variables dentro de los paréntesis del "if" y del "else if" para que la estructura se encargue de evaluar las condiciones de las variables y comparar:

```
var edad = prompt("Ingresa tu edad actual");
var respuesta = prompt("¿Quieres estudiar en Desafío Latam?, Sí o No");

if (edad >= 18 && respuesta == "Si" || respuesta == "SI" || respuesta ==
"si") {
      alert("Excelente... Bienvenido");
} else if(edad < 18 && respuesta == "Si" || respuesta == "SI" ||
respuesta == "si") {
      alert("Que bueno, pero debes estar acompañado por tu
representante");
} else if(respuesta == "NO" || respuesta == "No" || respuesta == "no") {
      alert("Que lastima!!!... te esperamos pronto");
}
```

Generando el resultado en el navegador si el usuario ingresa 23 de edad y Si:

Imagen 3. Resultado en ventana emergente.
Fuente: Desafío Latam

Ejercicio propuesto (5)

Desarrolla un programa en JavaScript donde se solicite al usuario ingresar su edad y responder a la pregunta: **"¿Estás estudiando en la universidad u otra institución educativa?"** con un "Si" o "SI" o "si" o "NO" o "no" o "No".

Para luego, evaluar si la edad es mayor o igual a 18 y la respuesta es "si" o "Si" o "SI", responder con el mensaje: "Es correcto, deberías estar estudiando", de lo contrario, responder con el mensaje: "Que bueno, debes estar en el colegio aun, sigue así".

Al indicar que "no" o "No" o "NO" está estudiando, responder con el mensaje: "Que mal!!… deberías estar estudiando", y en el caso de no ser ninguno de los datos mencionado anteriormente, indicar un mensaje: "Datos errados"

Ciclos

Competencias

- Codificar un programa en Javascript utilizando diagramas de flujo para la implementación de un algoritmo.
- Elaborar rutinas utilizando estructuras de control repetitivas, diferenciando iteradores, contadores y acumuladores para elaborar algoritmos que resuelvan un problema dado.

Introducción

Los ciclos son estructuras de control repetitivas que toman el control de la ejecución del programa mientras se cumpla cierta condición. Una vez que esta condición se deje de cumplir, el programa continúa su ejecución luego de la declaración del ciclo. Por ello, se dice que estas estructuras son repetitivas y fundamentales para la programación, ya que permiten ejecutar un mismo código en reiteradas ocasiones sin necesitar escribir este código múltiples veces. Es decir, un código que se escribe una única vez se utiliza múltiples veces.

Existen dos tipos fundamentales de ciclos: for y while, los cuales son básicos e imprescindibles para toda persona que se dedique a la programación. Es importante destacar que el ciclo while tiene una variación en su estructura para crear otro ciclo repetitivo, como lo es do-while.

Comprender cómo manejar adecuadamente estas estructuras, nos permitirá implementar lógicas complejas en nuestros desarrollos. El manejo de ciclos es clave para cualquier programador, no sólo bajo la lógica de JavaScript, sino para tener las competencias y resolver adecuadamente desafíos que requieran cierta recurrencia.

Diagrama de flujo con ciclos

Es posible representar un ciclo utilizando un diagrama de flujo. Para ello, el factor clave es un nuevo uso de los condicionales en esta representación, es decir, mediante el uso de condicionales se repite cierta acción una y otra vez hasta que se cumpla el objetivo o propósito (condición de comparación) y así pasar al siguiente paso o finalizar el proceso. En el siguiente ejemplo, el algoritmo representado es comprar 10 limones en el supermercado y cuando termine de tomar los limones suficientes, puede regresar a la casa.

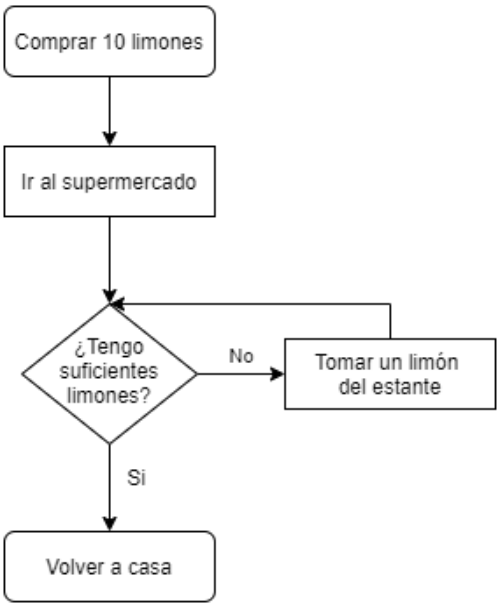

Imagen 4. Ejemplo de diagrama de flujo con ciclos.
Fuente: Desafío Latam

Como se puede apreciar en el diagrama de flujo anterior, la condición encerrada en el rombo es quien define la siguiente acción, en otras palabras, si se cumple o satisface la condición se vuelve a casa, de lo contrario se sigue tomando limones del estante.

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 (i++ es igual a decir i = i +1), 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. Eso es porque i tiene un valor inicial de 0 (La inicialización). Como la condición es que i sea menor que 5, la condición entonces ya no se cumple y las instrucciones del ciclo for no se ejecutan una sexta o séptima vez, por consiguiente, se detiene y sale del ciclo repetitivo para ejecutar la siguiente instrucción o dar fin al programa.

Ejemplo de ciclo for

Veamos un sencillo ejemplo para ver en ejecución un ciclo for, el cual, tiene como propósito mostrar 10 veces en el documento web el mensaje: "`{desafio} latam_`".

La forma de abordar este requerimiento mediante un ciclo for, es la siguiente:

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

Observemos que en muy pocas líneas de código hemos logrado implementar el enunciado:

{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_
{desafio} latam_

Imagen 5. Ejemplo de Ciclos.
Fuente: Desafío Latam

¿Qué hicimos aquí?

Simplemente se imprimió en pantalla 10 veces un texto.

Para poder hacer eso hicimos lo siguiente:

1. Declaramos nuestra variable i con un valor inicial de 0.
2. Luego dijimos en la condición que el ciclo se ejecutará hasta que nuestra variable "i" sea menor que 10.
3. 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:
, 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 en 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.

Para conocer cómo trabajar en código a partir de un diagrama de flujo, te invitamos a consultar el documento **Material Apoyo Lectura - De diagrama de flujo a código JavaScript,** ubicado en "Material Complementario", donde encontrarás un ejercicio que te permitirá aclarar algunas dudas respecto a esta tarea.

Ejercicio guiado: Ciclo for

Desarrollar y mostrar la tabla de multiplicar para el número 6 utilizando el ciclo for. Para ello, sigamos los siguientes pasos:

- ***Paso 1*:** Crear una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un ciclos.js.

- ***Paso 2*:** En el index.html escribir la estructura básica de un documento HTML.

- ***Paso 3*:** En el archivo script.js se declara una variable que llamamos "número", la cual almacena la tabla de multiplicar que queremos imprimir en pantalla, en este caso la tabla del 6.

```
var numero = 6;
```

- ***Paso 4:*** 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 automáticamente pasa a las instrucciones que tenemos dentro del for { }. En otras palabras, el ciclo se ejecutará mientras la "i" sea menor o igual a 10.

```
var numero = 6;

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

- ***Paso 5:*** 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 "número" que vale 6.

```
var numero = 6;

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

- ***Paso 6:*** Finalmente, con el método "document.write();" imprimimos en pantalla el valor de la variable "número" 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.

```
var numero = 6;

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

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 6. Ejemplo de condición.
Fuente: Desafío Latam

Ejercicio propuesto (6)

Realizar un programa en JavaScript mediante el uso del ciclo for que permita recibir un número entero cualquiera y muestre la tabla de multiplicar del 1 al 10 para ese número ingresado. Además, el mensaje a mostrar debe tener la estructura: 1 X 10 = 10, para cada uno de los resultados mostrados, haciendo uso de interpolación en vez de concatenación.

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. Ahora, en el caso de un diagrama de flujo, el ciclo while se dibujaría de la siguiente manera:

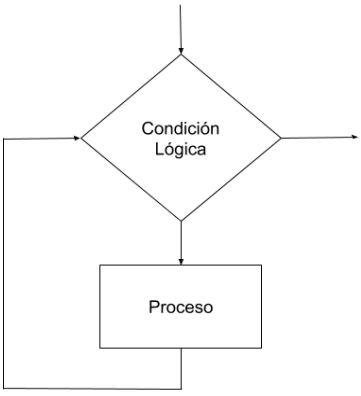

Imagen 7. Estructura de un ciclo While en diagrama de flujo.
Fuente: Desafío Latam

Partiendo del diagrama de flujo anterior, el bucle o ciclo while escrito en el lenguaje de programación JavaScript, se escribiría de la siguiente manera:

```
while (condición) {
  ...
}
```

Este ciclo funciona de la siguiente forma: Mientras se cumpla una condición y sea verdadera se ejecutan las instrucciones que estén dentro del ciclo. En el siguiente diagrama de flujo, se muestra como resultado en pantalla el listado de números desde el 0 hasta el 10, con una variable de inicio igual a 0 que servirá para contar y verificar la condición en el ciclo.

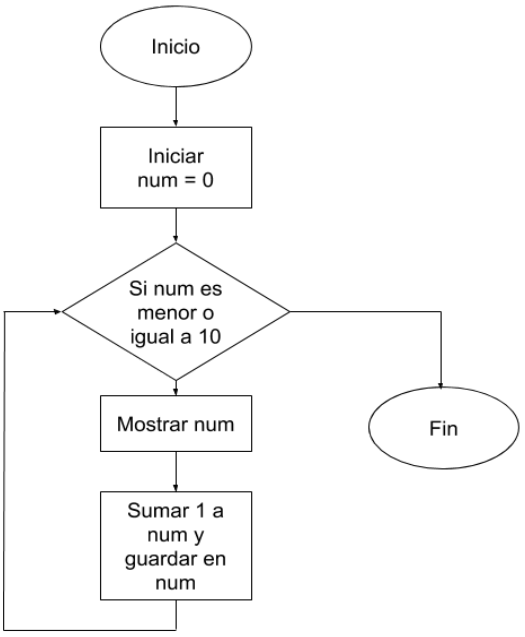

Imagen 8. Diagrama de flujo con ciclo while.
Fuente: Desafío Latam

El diagrama anterior en código, se escribe de la siguiente manera:

```
var num = 0; // inicio de la variable doble propósito

while (num <= 10) { // condición a evaluar antes de entrar al ciclo
  document.write(num + "<br>"); // imprimimos el valor de la variable
  num++; // incrementamos la variable en 1 para antes de iniciar el
ciclo nuevamente y pasar por la condición de evaluación
}
```

Generando el resultado en el navegador:

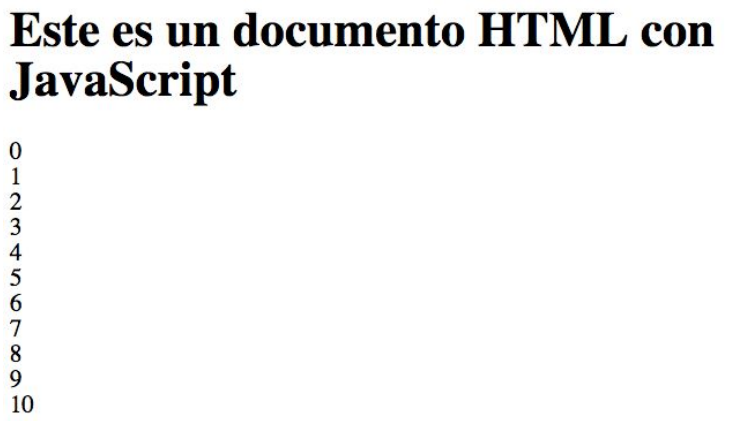

Imagen 9. Resultado del ciclo while en el navegador web.
Fuente: Desafío Latam

Ejercicio guiado: Ciclo while

Imprimir los número pares que correspondan desde el cero (0) hasta el 20, para ello, sigamos los siguientes pasos:

- ***Paso 1:*** En el archivo ciclos.js, debes armar la estructura del ciclo while e inicializar una variable doble propósito, el primero será el número que deseamos que vaya aumentando a medida que se repitan los ciclos y el segundo será la condición que evaluará el ciclo para saber si debe seguir o salir del ciclo.

```
var num = 0;

while (num <= 20) {
}
```

- ***Paso 2:*** Ya dentro del ciclo, se toma la variable inicializada fuera del ciclo while, se le suma el número 2 y se guarda sobre ella misma, lo que permite ir almacenando y mostrando solamente los números pares. Quedando el código:

```
var num = 0;

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

Lo cual, generaría como resultado en nuestro navegador web:

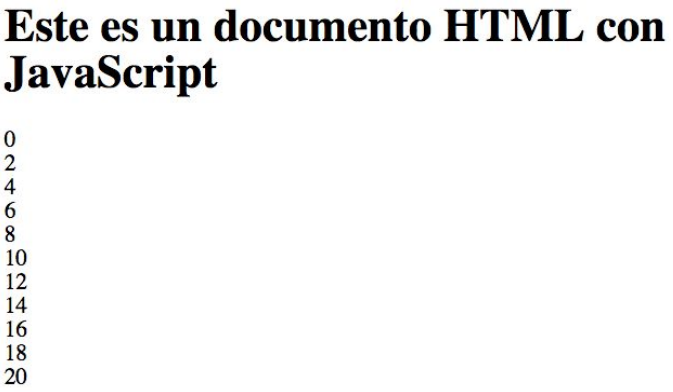

Imagen 10. Resultado del ciclo while en el navegador web.
Fuente: Desafío Latam

Ejercicio propuesto (7)

Implementar el diagrama de flujo del inicio del capítulo pero esta vez usando el ciclo while.

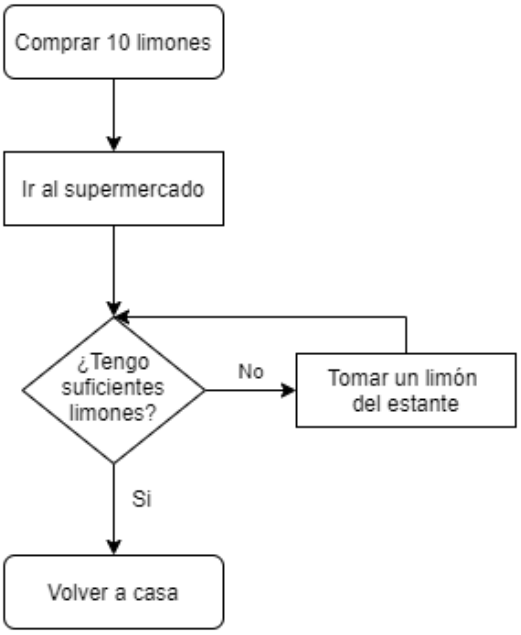

Imagen 11. Diagrama de flujo para implementar con ciclo while.
Fuente: Desafío Latam

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, 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 es falsa sale inmediatamente del bucle.

Hagamos un ejemplo que solicite al usuario ingresar su nombre en cada iteración del ciclo y mientras el usuario no ingrese ningún tipo de dato, el ciclo se seguirá ejecutando solicitando el nombre del usuario.

```
do {
  var nombre = prompt("Ingresa tu 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.

Ejercicio guiado: Ciclo do-while

Imprimir los números impares que correspondan desde el cero (0) hasta el 20 utilizando do-while. Sigamos los siguientes pasos:

- ***Paso 1*:** En el archivo ciclos.js, crear toda la estructura del ciclo do-while e inicializar una variable doble propósito, el primero será el número que deseamos que vaya aumentando a medida que se repitan los ciclos y el segundo será la condición que evaluará el ciclo para saber si debe seguir o salir del ciclo.

```
var num = 1;

do{
}while (num <= 20)
```

- ***Paso 2*:** Se agrega el código que queremos que se ejecute dentro del ciclo:

```
var num = 1;

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

Como se puede apreciar, es muy parecida la estructura y el procedimiento al ciclo while, como se hizo anteriormente. La diferencia con el ciclo do-while, es que **primero se hace el proceso dentro del ciclo y al final es cuando se realiza la comparación o condición lógica del ciclo para ver si continúa o no**. Por lo tanto, el resultado del código anterior en nuestro navegador web, sería:

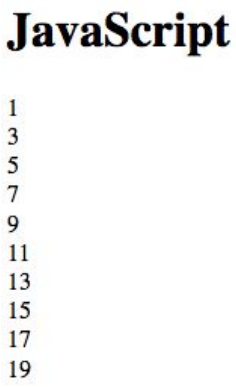

Imagen 12. Resultado del ciclo while en el navegador web.
Fuente: Desafío Latam

Ejercicio propuesto (8)

Implementa el diagrama de flujo del inicio del capítulo pero esta vez usando el ciclo do-while.

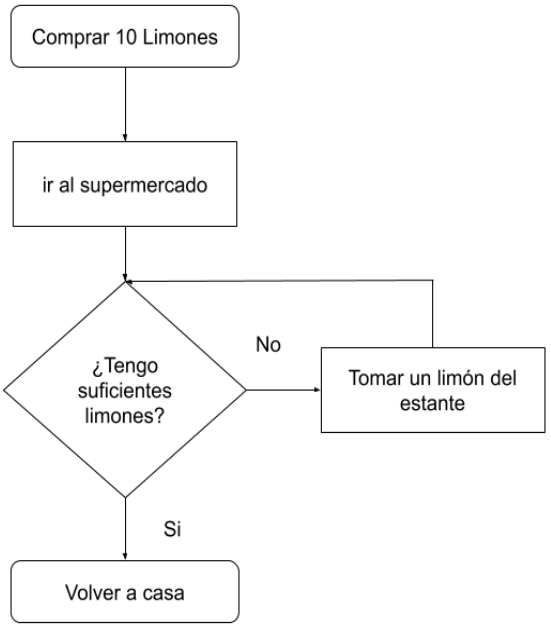

Imagen 13. Ejemplo de diagrama de flujo con ciclo do while.
Fuente: Desafío Latam

Diferencias entre un ciclo y un if

Un ciclo es un conjunto de instrucciones que se repiten de manera mientras cierta condición sea verdadera. Una sentencia if, en cambio, es la única comprobación de una comparación lógica. Así, si se requiere ejecutar una acción de manera repetitiva mientras se cumpla una condición, se deben usar ciclos (for o while). En cambio, si se requiere verificar una única vez alguna condición, corresponde usar if.

En resumen:

- Si se requiere verificar una única vez una condición, debes usar if.
- Si se requiere verificar múltiples veces una condición, entonces se debe usar ciclos repetitivos, como for o while.

Diferencias entre iteradores, contadores y acumuladores

Los ciclos pueden ser utilizados para solucionar diferentes situaciones y problemas, por lo que es necesario conocer las diferentes formas en las que se puede utilizar esta estructura de control.

Iteradores

Un iterador corresponde a aquella variable utilizada en cada etapa del ciclo para verificar la condición de término, corresponde a una variable que indica en qué etapa del ciclo se encuentra el programa. En el siguiente ejemplo, la variable i es el iterador.

```
for(i=0; i<10; i++){
   // Código del ciclo for
}
```

Contadores

Un contador es una variable que registra el número de ocurrencias de un suceso en particular, es diferente de un iterador ya que un contador no interfiere en la condición de término de un ciclo. Por ejemplo, si se quisiera saber cuántos números pares hay entre 10 y 100, es necesario utilizar un contador para saber la cantidad precisa de números pares. A continuación se muestra el código que ejecuta lo anterior:

```
contador = 0; // Los contadores por lo general se inician en 0
for(i=10; i<=100; i++){
   if(i % 2 === 0 ){ // se divide mediante el módulo del número
       contador++; // es igual que escribir: contador = contador + 1;
  }
}
alert(`Hay ${contador} números pares entre 10 y 100`);
```

Acumuladores

Los acumuladores, tal y como su nombre lo indica, sirven para acumular el valor de varias variables en una sola, son útiles para situaciones en las que no basta con contar ciertos eventos, sino que hay que considerar su contenido. Por ejemplo, si se quisiera saber cuál es la suma de todos los números entre el 1 y el 100 (incluyendo ambos), es necesario utilizar un acumulador. A continuación se muestra el código que ejecuta lo anterior:

```
acumulador = 0; // Los contadores por lo general se inician en 0
for(i=1; i<=100; i++){
   acumulador += i; // es igual es escribir: acumulador = acumulador + i
}
alert(`La suma total acumulada de los números entre el 1 y 100 es:
${acumulador} `);
```

Ejercicio guiado: Iteradores, contadores y acumuladores

Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números impares comprendidos entre el 0 y el 50 (incluidos ambos) implementando el ciclo for. Es decir, se deben sumar solamente los números impares entre el 1 y el 50. Indicar cuánto fue la suma y a su vez cuántos números se sumaron. para ello, sigamos los siguientes pasos:

- ***Paso 1*:** Implementar un contador, para saber la cantidad (conteo) de números impares y un acumulador que permite ir almacenando la suma de esos números, entonces se debe crear ambas variables e inicializarlas en cero. Luego de esto se realiza el ciclo for con la configuración necesaria para que el ciclo se repita las 50 veces solicitadas, quedando de la siguiente forma:

```
var contador = 0; // inicio del contador en 0
var acumulador = 0; // inicio del acumular en 0

for(i=1; i<=50; i++){
}
```

- **Paso 2:** Activamos la lógica correspondiente dentro de la estructura del for, en este caso, como necesitamos saber cuales son los números impares, entonces podemos implementar una estructura condicional como el if y dentro de su condición preguntar si el número es impar mediante la división del módulo (%), si es así, se debe contar esa iteración y acumular al mismo tiempo la sumatoria de los números que vayan resultado ser impar, luego de terminar el ciclo se muestra el mensaje final por una alert, indicando la cantidad de números y el total acumulado:

```
var contador = 0; // inicio del contador en 0
var acumulador = 0; // inicio del acumular en 0

for(i=1; i<=50; i++){
   if(i % 2 != 0 ){ // se divide mediante el módulo del número
      contador++; // es igual que escribir: contador = contador + 1;
      acumulador += i; //es igual a escribir: acumulador = acumular + i;
  }
}
alert(`Hay ${contador} números impares entre 1 y 50 y la suma acumulada
de ellos es: ${acumulador}`);
```

Generando el resultado en el navegador:

Imagen 14. Resultado en ventana emergente del ejemplo.
Fuente: Desafío Latam

Ejercicio propuesto (9)

Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números pares comprendidos entre el 0 y el 100 (incluidos ambos) implementando el ciclo for. Es decir, se deben sumar solamente los números pares entre el 0 y el 100. Indicar cuánto fue la suma y a su vez indicar cuántos números se sumaron. (Ayuda: debes implementar el ciclo for, contadores y acumuladores).

Ejercicio propuesto (10)

Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números pares comprendidos entre el 0 y el 100 (incluidos ambos) implementando el ciclo while. Es decir, se deben sumar solamente los números pares entre el 0 y el 100, indica cuándo fue la suma y a su vez indicar cuántos números se sumaron. (Ayuda: debes implementar el ciclo while, contadores y acumuladores).

Ejercicio propuesto (11)

Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números impares comprendidos entre el 0 y el 100 (incluidos ambos) implementando el ciclo do-while. Es decir, se deben sumar solamente los números impares entre el 0 y el 100, indicar cuándo fue la suma y a su vez indicar cuántos números se sumaron. (Ayuda: debes implementar el ciclo do-while, contadores y acumuladores).

Funciones en JavaScript

Competencias

- Aplicar funciones para resolver problemas de carácter repetitivo o que sea necesario segmentar.
- Codificar funciones utilizando paso de parámetros y retorno para resolver un problema.

Introducción

A lo largo de los capítulos anteriores hemos aprendido desde lo más sencillo y básico del lenguaje de programación JavaScript. Hemos manipulado datos, almacenado en variables o constantes, utilizando condicionales y ciclos para mostrarlos.

Cuando desarrollamos problemas de mediana complejidad, es natural utilizar frecuentemente las mismas instrucciones. Una función, de forma general, nos permite agrupar dichas lógicas para realizar una tarea concreta bajo un nombre que nos facilite identificar un comportamiento específico del código.

En este capítulo veremos que a través de las funciones podremos optimizar nuestro código, darle mejor legibilidad y consistencia para hacerlo mantenible en el tiempo.

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 queramos dándoles dinamismo para que éstas 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 15. Ejemplo de Función.
Fuente: Desafío Latam

Un libro de recetas guardado en su estante, es equivalente a una función declarada, cuando sacamos este libro y seguimos los pasos de la receta, es equivalente a ejecutar la función.

Es así como las funciones son mini programas que cumplen una "función" determinada. La principal potencia de esto es que podemos reutilizar este mini programa como y cuando queramos, todas las veces que sea necesario.

La estructura de una función es la siguiente:

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

Con el siguiente ejemplo muy básico veremos de qué manera 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, después llamamos a la función por su nombre para que se ejecute.

Ahora bien, qué pasa si por ejemplo a la función anterior de saludo queremos que muestre el nombre ingresado por un usuario en nuestra página web y saludarlo en conjunto con el nombre ingresado. Para esto tenemos que ingresar unos parámetros a la función, ¿esto qué significa?, que a las funciones podemos pasarles argumentos y recibir parámetros, que pueden ser variables, constantes, entre otros, que podemos enviar para que puedan ser recibidas y trabajadas en la función. Entonces 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 (parámetro) 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?

Ejercicio guiado: Aplicando funciones

Realizar un ejercicio donde se solicita al usuario que ingrese la edad y dentro de una función se muestre la edad del usuario con un saludo desde el documento web. Para ello, sigamos los siguientes pasos:

- ***Paso 1*:** Crea una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un funciones.js.

- ***Paso 2*:** En el index.html debes escribir la estructura básica de un documento HTML.

- ***Paso 3*:** En el archivo funciones.js debes armar la estructura de la función e inicializar una variable donde se almacenará el dato ingresado por el usuario, en este caso la podemos llamar "edad" y mediante la función "prompt()" solicitamos al usuario ingresar la edad actual. Quedando de la siguiente forma:

```
var nombre = prompt("Ingrese su edad: ");

function saludo (edad) { // tomamos el parámetro de la variable.
  document.write("¿Hola, tienes " + edad + " años");
}
```

- **Paso 4:** Realizar el llamado a la función pasando el argumento en el llamado, en este caso es la edad ingresada por el usuario, para poder mostrar el mensaje dentro de la función con el dato recibido (parámetro) :

```
var edad = prompt("Ingrese su edad: ");

function saludo (edad) { // tomamos el parámetro de la variable.
  document.write("Hola, tienes " + edad + " años!!");
}

saludo(edad) // enviamos la edad como argumento de la función
```

Generando el resultado en el navegador:

Imagen 16. Resultado del ejercicio en el navegador web.
Fuente: Desafío Latam

Ejercicio propuesto (12)

Desarrollar un programa en JavaScript que, solicitando al usuario el nombre, apellido y edad, mostrando así un saludo en el documento con el nombre, apellido y la edad, ejemplo: "Hola Juan Lopez, tienes 34 años".

Dividir problemas en subprocesos e identificar la entrada y salida

En este ejemplo 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". Es decir, ahora debemos implementar diversos conceptos nuevos, como el caso de parámetros, argumentos y retorno de una función, lo que conlleva a dividir el problema en partes, como: Valores de inicio, llamado de la función y valores de entrada, procesamiento de los valores dentro de la función, retorno de información y valores de salida.

Ejercicio guiado: Entrada y salida de la función.

Se le solicita al usuario ingresar dos números enteros y almacenarlos en variables distintas (**valores de inicio**), luego mediante el uso de una función, enviar los dos números ingresados a la función (argumento**, llamado de la función y valores de entrada**), para ser recibidos (**parámetros**) y procesados (sumando ambos números**, procesamiento de valores**) y finalmente retornar el valor final de la suma de ambos números (**retorno de información y valores de salida**). Para ello, sigamos los siguientes pasos:

- ***Paso 1*:** Crea una carpeta en tu lugar de trabajo favorito y dentro de ella crea dos archivos, un index.html y un script.js.

- ***Paso 2*:** En el index.html debes escribir la estructura básica de un documento HTML.

- ***Paso 3*:** En el archivo script.js debes armar la estructura de la función e inicializar dos variable donde se almacenarán los datos ingresado por el usuario, en este caso los

podemos llamar “num1” y “num2”, luego mediante la función “prompt()” solicitamos al usuario ingresar los números. Quedando de la siguiente forma:

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

function suma (num1, num2) { // recibimos los dos números como
parámetros de la función
}
```

- ***Paso 4*:** Activamos la lógica correspondiente dentro de la función, llevando los datos ingresados por el usuario a numéricos mediante la instrucción parseInt, luego realizando la suma de ambos números y almacenando el resultado en otra variable, finalmente retornamos el valor de la suma final. Ya fuera de la función, debe ir el llamado a la misma donde se pasan los argumentos y se muestra en el resultado directamente en el documento:

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

function suma (num1, num2) { // recibimos los dos números como
parámetros de la función
  var num1 = parseInt(num1);
  var num2 = parseInt(num2);
  var resultadoSuma = num1 + num2;
  return resultadoSuma; // permite retornar de la función con un valor
en específico
}
document.write(suma(num1, num2)); //enviamos los dos números ingresados
por el usuario como argumentos de la función
```

Generando el siguiente resultado en el navegador para los valores de 5 y 7:

Imagen 17. Resultado del ejemplo en el navegador web.
Fuente: Desafío Latam

Ejercicio propuesto (13)

Desarrollar un programa en JavaScript que, solicitando al usuario tres números enteros, realice la suma de los tres números y retorne el valor desde la función. Por lo tanto, se debe imprimir en pantalla el resultado de la función y enviar como argumentos los tres números ingresados por el usuario.

Aplicar funciones para resolver problemas de carácter repetitivo

En el caso que un problema se debe repetir la cantidad de veces que solicite el usuario sin la necesidad de refrescar o recargar nuestra página web se puede implementar el uso de funciones que realicen el proceso repetitivo una y otra vez con las mismas instrucciones de código, sin la necesidad de crear más estructuras de código para que cumplan esa función.

Ejercicio guiado: Funciones en JavaScript

Ahora realizaremos el mismo **Ejercicio guiado: Entrada y salida de la función**, pero utilizaremos un mini formulario para poder segmentar y dejar la página siempre disponible para que el usuario interactúe con ella sin la necesidad de recargar o refrescar la página, es decir, el usuario deberá ahora ingresar los números en las entradas indicadas del formulario y lo haremos de la siguiente manera.

- ***Paso 1*:** Escribir este código en el 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() );"> <!-- esta es una forma antigua de activar funciones con
JavaScript desde el elemento HTML como atributo. -->
</form>
```

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

- ***Paso 2:*** Escribir la estructura de la función, la cual, no tendrá parámetros debido a que captamos los valores ingresados por el usuario directo desde el formulario mediante las instrucciones "getElementById("id").value" que nos permite obtener la información de un elemento HTML del documento y luego extraer el valor mediante la propiedad "value", y los almacenaremos en una variable:

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

- ***Paso 3:*** Finalmente retornamos el resultado de la suma:

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

Por lo que el resultado al ingresar los valores de 6 y 7 en el formulario y hacer un clic sobre el botón de sumar sería:

Imagen 18. Resultado del ejemplo en el navegador web.
Fuente: Desafío Latam

Ejercicio propuesto (14)

Desarrollar un programa en JavaScript, que solicite un solo número al usuario y verifique mediante una función, si dicho número que ingresó el usuario es positivo, negativo o nulo. Por lo tanto, se debe indicar los siguientes mensajes según corresponda:

- "El número es positivo".
- "El número es negativo".
- "El número es nulo"

(Ayuda: utiliza parseInt para convertir de string a number)

Resumen

En esta lectura hemos podido profundizar aspectos relacionados a la implementación de algoritmos que resuelven problemas recurrentes, donde se requieren estructuras de control repetitivas que nos permitan manipular ciclos en el código.

Comprender estos conceptos y utilizarlos correctamente es clave para optimizar el código que escribimos, hacerlo legible y mantenible en respuesta a problemas de mediana complejidad.

El uso de funciones en JavaScript, es un concepto que aún debemos profundizar, pero es importante que comprendas su estructura básica y continúes practicando, para que te familiarices con su uso y ventajas.

Como te pudiste dar cuenta, vimos lo esencial y la base del lenguaje de programación JavaScript, lo que nos ayudará en nuestra carrera 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.

Solución de los ejercicios propuestos

1. Considerando las siguientes variables: `a = 10 y b = 5`, ¿cuál es el resultado de las siguientes operaciones lógicas?

```
a > 5 || b < 10 // regresa true
```

```
a == 10 && b == 5 // regresa true
```

```
(a > 5 || a < 15) && b > 0 // regresa true
```

```
!(a > 5) && b < 10 // regresa false
```

```
(a - 5) >= (a*b) - (b/a) // regresa false
```

2. Realiza un programa en JavaScript implementando la estructura de control if, que solicite al usuario ingresar dos números y que compruebe si el número 1 es positivo, si el número dos es negativo y finalmente si el número 2 es mayor o igual que el número 1, indicando si se cumple cada condición mediante un alert.

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

if(num1 > 0) {
   alert("El primer número ingresado es positivo");
}
if(num2 < 0) {
   alert("El segundo número que ingresaste es negativo");
}
if(num2 >= num2) {
   alert("El segundo número que ingresaste es mayor o igual que el
primer número");
}
```

3. Realiza un programa en JavaScript implementando la estructura de control if-else, que solicite al usuario indicar si desea aprender a programar con NodeJS, si el usuario indica que “Si” o “SI” o “si” indicar mediante un alert “Es hora de estudiar NodeJS”, de lo contrario indicar mediante un alert “No quieres aprender NodeJS, sigue estudiando JS”.

```
var respuesta = prompt("¿Quiere aprender NodeJS?: Si o No");

if (respuesta == "Si" || respuesta == "si" || respuesta == "SI") {
      alert("Es hora de estudiar NodeJS");
} else {
      alert("No quieres aprender NodeJS, sigue estudiando JS");
}
```

4. Realiza un programa en JavaScript implementando la estructura de control switch, donde se solicita a un usuario que ingrese un número del 1 al 12 e indicar mediante un mensaje el mes del año al cual pertenece ese número. Si no pertenece a ningún mes indicar que el número no está permitido.

```
var num = prompt("Ingrese el número entre el 1 y el 12");

switch (num) {
      case '1':
           alert("El mes es Enero");
      break;
      case '2':
           alert("El mes es Febrero");
      break;
      case '3':
           alert("El mes es Marzo");
      break;
      case '4':
           alert("El mes es Abril");
      break;
      case '5':
           alert("El mes es Mayo");
      break;
      case '6':
           alert("El mes es Junio");
      break;
      case '7':
           alert("El mes es Julio");
```

```
        break;
        case '8':
             alert("El mes es Agosto");
        break;
        case '9':
             alert("El mes es Septiembre");
        break;
        case '10':
             alert("El mes es Octubre");
        break;
        case '11':
             alert("El mes es Noviembre");
        break;
        case '12':
             alert("El mes es Diciembre");
        break;
        default:
             alert("El número no pertenece a un mes calendario");
        break;
}
```

5. Desarrolla un programa en JavaScript donde se solicite al usuario ingresar su edad y responder a la pregunta: "¿Estás estudiando en la universidad u otra institución educativa?" con un "Si" o "SI" o "si" o "NO" o "no" o "No".

   Para luego, evaluar si la edad es mayor o igual a 18 y la respuesta es "si" o "Si" o "SI", responder con el mensaje: "Es correcto, deberías estar estudiando", de lo contrario, responder con el mensaje: "Que bueno, debes estar en el colegio aun, sigue así".

   Al indicar que "no" o "No" o "NO" está estudiando, responder con el mensaje: "Que mal!!... deberías estar estudiando", y en el caso de no ser ninguno de los datos mencionado anteriormente, indicar un mensaje: "Datos errados"

```
var edad = prompt("Ingresa tu edad actual");
var respuesta = prompt("¿Estás estudiando en la universidad u otra
institución educativa?");

if (edad >= 18 && respuesta == "Si" || respuesta == "SI" || respuesta ==
"si") {
      alert("Es correcto, deberías estar estudiando");
} else if(edad < 18 && respuesta == "Si" || respuesta == "SI" ||
```

```
respuesta == "si") {
      alert("Que bueno, debes estar en el colegio aun, sigue así");
} else if(respuesta == "NO" || respuesta == "No" || respuesta == "no") {
      alert("Que mal!!... deberías estar estudiando");
}
else {
      alert("Datos errados");
```

6. Realizar un programa en JavaScript mediante el uso del ciclo for que permita recibir un número entero cualquiera y muestre la tabla de multiplicar del 1 al 10 para ese número ingresado. Además, el mensaje a mostrar debe tener la estructura: 1 X 10 = 10, para cada uno de los resultados mostrados, haciendo uso de interpolación en vez de concatenación.

```
var num = prompt("Ingrese el número para la tabla de multiplicar: ");

for (let i = 1; i <= 10; i++) {
      var resultado = num * i;
      document.write(`${num} X ${i} = ${resultado} <br>`);
}
```

7. Implementar el diagrama de flujo del inicio del capítulo pero esta vez usando el ciclo while.

```
var limon = 1;

while (limon <= 10) {
   document.write("Tomar un limón del estante <br>");
   limon = limon + 1;
}
```

8. Implementa el diagrama de flujo del inicio del capítulo pero esta vez usando el ciclo do-while.

```
var limon = 1;

do {
   document.write("Tomar un limón del estante <br>");
```

```
   limon = limon + 1;
}while (limon <= 10);
```

9. Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números pares comprendidos entre el 0 y el 100 (incluidos ambos) implementando el ciclo for. Es decir, se deben sumar solamente los números pares entre el 0 y el 100. Indicar cuánto fue la suma y a su vez indicar cuántos números se sumaron. (Ayuda: debes implementar el ciclo for, contadores y acumuladores)

```
var contador = 0; // inicio del contador en 0
var acumulador = 0; // inicio del acumular en 0

for(i=0; i<=100; i++){
   if(i % 2 === 0 ){ // se divide mediante el módulo del número
      contador++; // es igual que escribir: contador = contador + 1;
      acumulador += i; //es igual a escribir: acumulador = acumular + i;
  }
}
alert(`Hay ${contador} números pares entre 1 y 100 y la suma acumulada
de ellos es: ${acumulador}`);
```

10. Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números pares comprendidos entre el 0 y el 100 (incluidos ambos) implementando el ciclo while. Es decir, se deben sumar solamente los números pares entre el 0 y el 100, indica cuándo fue la suma y a su vez indicar cuántos números se sumaron. (Ayuda: debes implementar el ciclo while, contadores y acumuladores).

```
var contador = 0; // inicio del contador para números pares
var acumulador = 0; // inicio del acumular para suma de números pares
var i = 0; // inicio de contador para iteración

while(i <= 100){
   if(i % 2 === 0 ){ // se divide mediante el módulo del número
      contador++; // es igual que escribir: contador = contador + 1;
      acumulador += i; //es igual a escribir: acumulador = acumular + i;
  }
  i++;
};

alert(`Hay ${contador} números pares entre 1 y 100 y la suma acumulada
de ellos es: ${acumulador}`);
```

11. Elaborar un programa con JavaScript que permita sumar e indicar la cantidad de números impares comprendidos entre el 0 y el 100 (incluidos ambos) implementando el ciclo do-while. Es decir, se deben sumar solamente los números impares entre el 0 y el 100, indicar cuándo fue la suma y a su vez indicar cuántos números se sumaron. (Ayuda: debes implementar el ciclo do-while, contadores y acumuladores)

```
var contador = 0; // inicio del contador para números impares
var acumulador = 0; // inicio del acumular para suma de números impares
var i = 0; // inicio de contador para iteración

do {
   if(i % 2 != 0 ){ // se divide mediante el módulo del número
      contador++; // es igual que escribir: contador = contador + 1;
      acumulador += i; //es igual a escribir: acumulador = acumular + i;
  }
  i++;
}while(i <= 100);

alert(`Hay ${contador} números impares entre 1 y 100 y la suma acumulada
de ellos es: ${acumulador}`);
```

12. Desarrollar un programa en JavaScript que, solicitando al usuario el nombre, apellido y edad, mostrando así un saludo en el documento con el nombre, apellido y la edad, ejemplo: “Hola Juan Lopez, tienes 34 años”.

```
var nombre = prompt("Ingrese nombre: ");
var apellido = prompt("Ingrese tu apellido: ");
var edad = prompt("Ingrese su edad: ");

function saludo (nombre, apellido, edad) {
  document.write("Hola "+ nombre +" "+apellido", tienes " + edad + "
años!!");
}

saludo(nombre,apellido,edad)
```

13. Desarrollar un programa en JavaScript que, solicitando al usuario tres números enteros, realice la suma de los tres números y retorne el valor desde la función. Por lo tanto, se debe imprimir en pantalla el resultado de la función y enviar como argumentos los tres números ingresados por el usuario.

```
var num1 = prompt("Ingresa un primer número: ");
var num2 = prompt("Ingresa un segundo número: ");
var num3 = prompt("Ingresa un tercer número: ");

function suma (num1, num2, num3) {
  var num1 = parseInt(num1);
  var num2 = parseInt(num2);
  var num3 = parseInt(num3);
  var resultadoSuma = num1 + num2 + num3;
  return resultadoSuma;
}
document.write(suma(num1, num2, num3));
```

14. Desarrollar un programa en JavaScript, que solicite un solo número al usuario y verifique mediante una función, si dicho número que ingresó el usuario es positivo, negativo o nulo. Por lo tanto, se debe indicar los siguientes mensajes según corresponda:
    - "El número es positivo".
    - "El número es negativo".
    - "El número es nulo"

    (Ayuda: utiliza parseInt para convertir de string a number)

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

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))}`);
```