

Introducción al lenguaje JavaScript (Parte I)

¿Por qué aprender JavaScript?

Competencia

- Distinguir las características del lenguaje JavaScript para comprender sus particularidades versus otros lenguajes de programación.

Introducción

JavaScript surge ante la necesidad de mayor interactividad con el usuario y una creciente importancia de mejorar la experiencia del sitio a través de herramientas que le agregan dinamismo. Actualmente, JavaScript es un requerimiento indispensable para un programador y junto con HTML y CSS, forman los 3 pilares del desarrollo web.

En este capítulo, conoceremos en detalle las características de JavaScript, para contextualizar la importancia de este lenguaje, sus principales usos y ventajas, a través de la documentación oficial y una comparativa frente a otros lenguajes de programación. Esto te permitirá tener nociones básicas de este lenguaje, para comprender mejor los contenidos de los siguientes capítulos.

¿Por qué utilizar JavaScript?

JavaScript se ha transformado, en el último tiempo, en un lenguaje muy popular entre la comunidad global de desarrolladores. No porque este lenguaje sea la moda simplemente, sino porque durante el tiempo ha demostrado ser un lenguaje sólido, liviano, simple y muy funcional.

La programación es la manipulación de datos y JavaScript lo hace perfecto, ya que nos permitirá manipular los elementos de una página web, obtener datos de sus elementos, modificarlos, animarlos, condicionarlos y procesarlos, además de guardarlos para ocuparlos más tarde, mandarlos o pedirlos al servidor. En definitiva, casi todo lo que se nos ocurra podemos hacerlo con JavaScript.

Características Generales de JavaScript

JavaScript es un lenguaje de programación multipropósito y multiparadigma (es posible programar en JavaScript siguiendo los paradigmas Procedimental, Orientado a Objetos y Funcional), interpretado y débilmente tipado (el lenguaje que infiere el tipo de dato asignado de acuerdo a su contenido), que puede ser ejecutado sin problemas en todos los navegadores modernos.

En la siguiente imagen, se puede apreciar cuáles son las principales características de JavaScript:

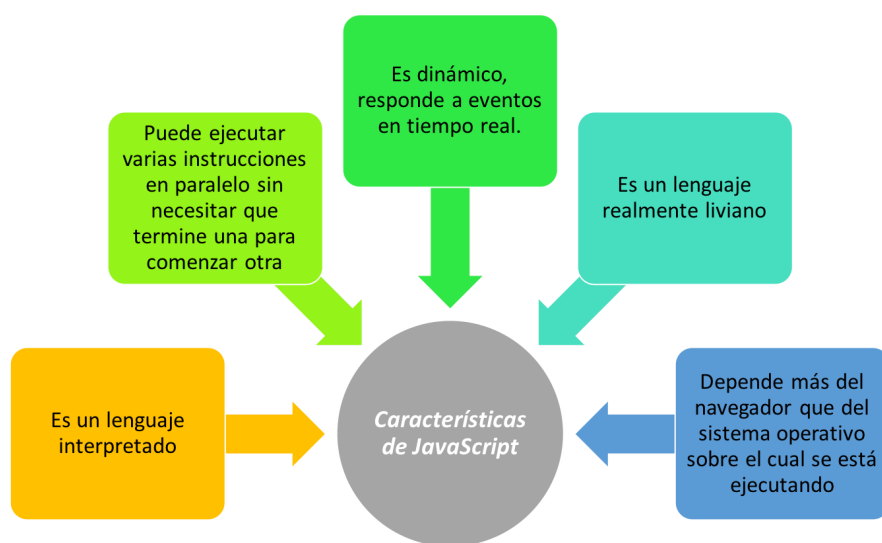


Imagen 1. Características de JavaScript.
Fuente: Desafío Latam

Usos de JavaScript

El hecho de que JavaScript sea multipropósito implica que puede ser utilizado para diferentes cosas dentro del ámbito de la programación. Su primer uso, y para el cual fue diseñado en un principio, fue para darle interactividad y efectos visuales a las páginas web. Pero hoy en día, su implementación abarca desde integración de sistemas hasta aplicaciones móviles, como se muestra en la siguiente imagen:

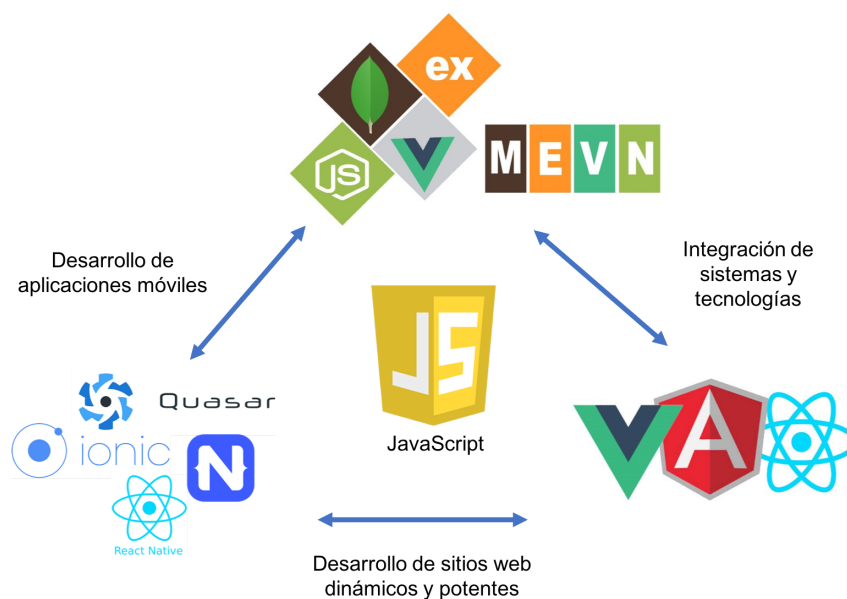


Imagen 2. Uso de JavaScript en la actualidad.

Fuente: Desafío Latam

Es una buena idea saber JavaScript

Toda persona dedicada al desarrollo de aplicaciones web tendrá que usar JavaScript en algún momento de su vida, aunque use un lenguaje diferente en el lado del servidor. Esto, porque JavaScript es parte fundamental de toda aplicación web moderna, dado que entrega la capacidad de agregar interactividad a una página web para hacerla más atractiva de usar. Lo anterior convierte a JavaScript en un factor clave en la experiencia de usuario.

Esta preponderancia en el ámbito de la experiencia de usuario, lo convierte en un imprescindible de toda persona dedicada al desarrollo web.

JavaScript Vs Otros lenguajes de programación

En la siguiente tabla podrás conocer las características de JavaScript versus otro lenguajes de programación:





| Comparador Lenguaje | ¿Qué es? | Ventajas | Desventajas | Usos |
|--|---|---|---|---|
| JavaScript  | Es un lenguaje de programación que puede ser utilizado para crear programas que luego son acoplados a una página web o dentro de programas más grandes. | Velocidad Simplicidad Versatilidad | Es posible desactivar el JavaScript en el navegador web Requiere de otras aplicaciones Como es un programa que se ejecuta en el lado del cliente, sus códigos pueden ser leídos por otros usuarios. | Puede insertarse en cualquier página independientemente de la extensión del fichero. JavaScript puede también ser usado dentro de scripts escritos en otros lenguajes como Perl y PHP. |
| PHP  | Es un lenguaje de programación interpretado, diseñado originalmente para la creación de páginas web dinámicas. | Orientado al desarrollo de aplicaciones web dinámicas. Multiplataforma. | Solo se ejecuta en un servidor y se necesita un servidor web para que funcione. | Se usa principalmente para la interpretación del lado del servidor, páginas web y CMS Se usa en todos los sistemas operativos |
| Python  | Es un lenguaje de programación interpretado de tipado dinámico cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. | Simplificado y rápido Elegante y flexible Portable | La "curva de aprendizaje cuando ya estás en la parte web no es tan sencilla". La mayoría de los servidores no tienen soporte a Python, y si lo soportan, la configuración es un poco difícil. | El uso de Python está muy extendido en el análisis datos y la extracción de información útil para empresas. Se ocupa de los datos tabulares, matriciales y estadísticos |
| C++  | Es un lenguaje de programación orientado a objetos que toma la base del lenguaje C. | Es potente en cuanto a lo que se refiere a creación de sistemas complejos un lenguaje muy robusto | No es atractivo visualmente No soporta para creación de páginas web | Sirve para todos los sistemas operativos pero cada uno con su respectiva versión para dicho sistema |

Imagen 3. Comparación de JavaScript
Fuente: Desafío Latam

Esto es sólo el inicio

Muchos de los conceptos mencionados hasta el momento son avanzados y pueden ser abrumadores para alguien que recién se inicia en el mundo web. Pero este es sólo el inicio del camino, y una vez que vayas adquiriendo más experiencia, vas a dominar todas y cada una de estas nuevas palabras y raros conceptos.

Este es el inicio del camino. ¡Vamos con todo!

Diagramas de flujo y pseudocódigo

Competencias

- Identificar los componentes de los diagramas de flujo y pseudocódigos para resolver de manera secuencial un problema real.
- Construir diagramas de flujo y pseudocódigos para representar algoritmos de baja complejidad.

Introducción

Una de las primeras consideraciones que debemos tener al enfrentarnos al mundo de la programación, es la lógica antes que cualquier lenguaje en particular, es nuestra mejor herramienta para resolver los desafíos del día a día.

En este capítulo abordaremos una forma de representar gráficamente esta lógica, como son los diagramas de flujo. Estos nos permiten visualizar los procesos y la secuencia en que se requieren realizar ciertas operaciones para solucionar un problema dado. Además, aprenderemos a escribir nuestros programas en pseudocódigo, que es una descripción de alto nivel de un algoritmo.

Estas herramientas son fundamentales para tener claridad sobre los flujos de información antes de comenzar cualquier desarrollo, facilitando el ciclo de pensar, descomponer un problema, plantear una solución y escribir la solución en forma de código.

Algoritmos

Un algoritmo es una serie de pasos consecutivos que se realizan con un objetivo específico. Según la RAE, un algoritmo es un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema ([referencia](#)).

Un buen ejemplo de algoritmo es el origami: en esta técnica el objetivo es hacer formas con un trozo de papel sólo con dobleces sucesivos. Si quisiéramos lograr una grulla de origami, hay que ejecutar los siguientes pasos:

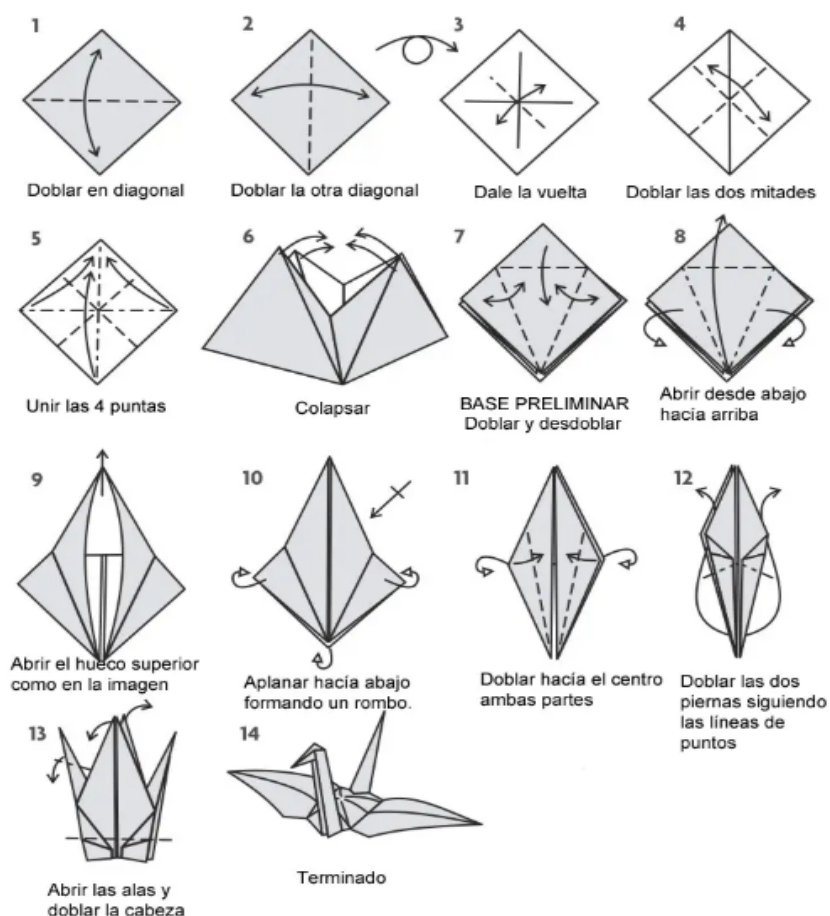


Imagen 4. Pasos para ejecutar el algoritmo de una grulla de origami

Fuente: comohacerorigami.net

Para finalmente llegar al resultado:

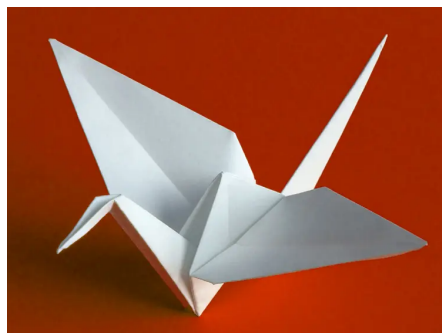


Imagen 5. El origami es un ejemplo perfecto de algoritmo.

Fuente: comohacerorigami.net

Por consiguiente, para dar una adecuada solución a todo problema, éste se debe estructurar en tres partes fundamentales, como son:

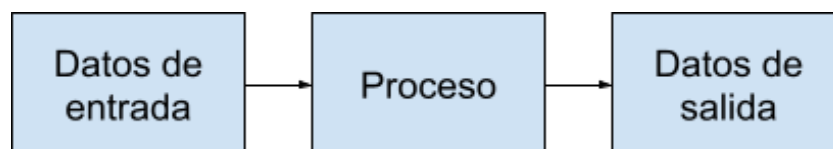


Imagen 6. Estructura para la solución de un problema.

Fuente: Desafío Latam

En otras palabras, es importante desglosar el problema, entender que sucede y que se pretende solucionar para poder generar una respuesta viable y por último, podríamos iniciar el proceso de escritura de nuestro código.

Para profundizar sobre cómo realizar un buen análisis de los datos de entrada, puedes consultar el documento **Material Apoyo Lectura - Análisis de un problema** ubicado en "Material Complementario". En este documento podrás revisar en detalle cuáles son los pasos para dar solución a un problema con un ejemplo práctico.

Formas de escribir un algoritmo

Así como un algoritmo puede ser representado como una serie de pasos, como en una receta o en un origami, también puede ser representado de otras formas, tales como:

- Diagramas de flujo.
- Pseudocódigo.
- Implementando los pasos directamente en un lenguaje de programación.

Los dos primeros elementos son esfuerzos por esquematizar y estructurar la representación de los algoritmos, con el objetivo de establecer un marco de referencia común para toda persona que quiera comprender un algoritmo. La escritura en algún lenguaje de programación, por su parte, es la realización tecnológica con la que se resuelve el problema a través del algoritmo.

Diagramas de flujo

Un diagrama de flujo es una representación visual de un algoritmo y se usa principalmente para comunicar procesos que suelen ser complejos, permitiendo una comprensión de manera más fácil. Se les llama "de flujo" ya que tienen un inicio, una representación visual de los pasos a seguir y un final. Dado que es una esquematización, contiene una simbología bien definida:

- Inicio y fin
- Línea de flujo
- Datos de entrada y salida
- Procesos
- Decisiones






| Símbolo | Nombre | Función |
|---|------------------|--|
|  | Inicio / Final | Representa el inicio y el final de un proceso |
|  | Línea de Flujo | Indica el orden de la ejecución de las operaciones. La flecha indica la siguiente instrucción. |
|  | Entrada / Salida | Representa la lectura de datos en la entrada y la impresión de datos en la salida |
|  | Proceso | Representa cualquier tipo de operación |
|  | Decisión | Nos permite analizar una situación, con base en los valores verdadero y falso |

Imagen 7. Símbolos de un diagrama de flujo.

Fuente: [smartdraw](https://smartdraw.com)

Estos símbolos se unen con flechas unidireccionales, con tal de representar el flujo y la secuencialidad de los pasos, formando una estructura bien definida que puede ser interpretada y posteriormente puesta en práctica como se muestra a continuación:



Imagen 8. Estructura secuencial de un diagrama de flujo.
Fuente: Desafío Latam

Ahora bien, supongamos que una de las lámparas de nuestro hogar dejó de funcionar. A continuación se muestra un diagrama de flujo con una posible solución al problema de la lámpara que no funciona:

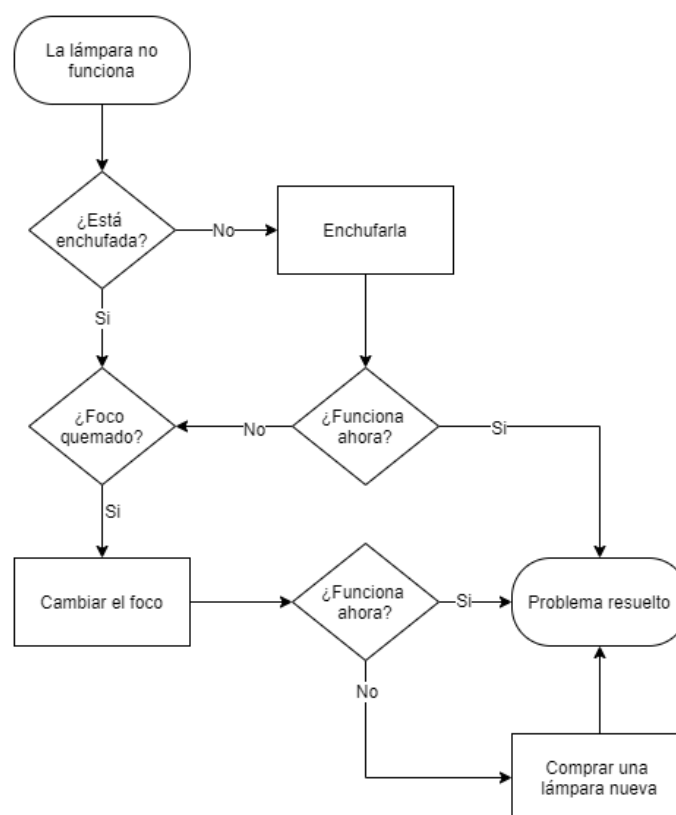


Imagen 9. Ejemplo de diagrama de flujo.
Fuente: Desafío Latam

Los diagramas de flujo son importantes porque significan una representación visual de una solución. Esto favorece la comprensión y la comunicación de la solución propuesta, tanto a nivel personal como con otras personas. En el mundo de la programación es clave la colaboración entre profesionales, ya sea del mismo rubro como de otros, por lo que poder representar visualmente nuestra solución es una poderosa herramienta.

Ejercicio guiado: Diagramas de flujo

Desarrollar un diagrama de flujo que realice la suma de dos números enteros ingresados por el usuario, mostrando el resultado de la suma.

- **Paso 1:** Realizar una tabla para desglosar el problema en tres partes: datos de entrada, proceso y datos de salida.

| Datos de Entrada | Proceso | Datos de Salida |
|---|--|---|
| Números ingresados por el usuario numero1 numero2 | Realizar la suma de los dos números numero1 + numero2 | Mostrar el resultado de la suma de ambos números Sumatoria Final |

Tabla 1. Estructura en partes del problema.

Fuente: Desafío Latam

- **Paso 2:** Realizar el diagrama de flujo pertinente. Para ello, lo primero es utilizar los diagramas presentados en la imagen N° 7, donde se muestra que el inicio de todo diagrama de flujo debe ir encerrado dentro de un óvalo, posteriormente se realizaria la lectura de los datos mediante el paralelogramo, quedando de la siguiente manera:

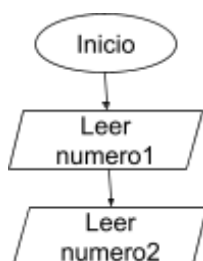


Imagen 10. Paso 2 - Inicio y lectura de los datos.

Fuente: Desafío Latam

- **Paso 3:** Realizar los procedimientos necesarios para poder obtener los resultados que deseamos o que requiere el problema planteado. En este caso, sería realizar la sumatoria de los dos términos ingresados por el usuario dentro un rectángulo, siendo $\text{numero1} + \text{numero2}$, y el resultado será mostrado como solución final en un paralelogramo, como se muestra a continuación:

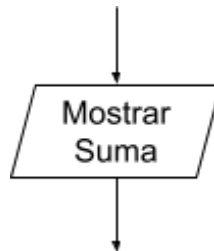


Imagen 11. Paso 3 - Procesamiento de datos, muestra y final.

Fuente: Desafío Latam

- **Paso 4:** Para finalizar el diagrama del flujo, se encierra en un óvalo la palabra Fin y así terminar con la solución al problema planteado, como se muestra en la siguiente imagen:

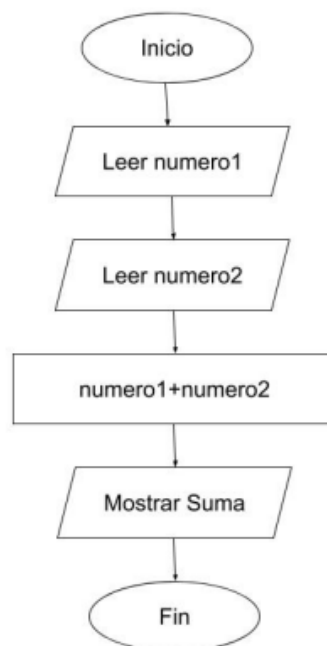


Imagen 12. Paso 4 - Final del diagrama de flujo.

Fuente: Desafío Latam

Ejercicio guiado: Área de un rectángulo

Desarrollar un diagrama de flujo que permita calcular el área de un rectángulo, partiendo de los datos ingresados por el usuario como la base y altura.

- **Paso 1:** Crear una tabla para identificar las tres partes fundamentales del problema:

| Datos de Entrada | Proceso | Datos de Salida |
|---|---|--|
| Valores ingresados por el usuario base altura | Realizar el cálculo del área del rectángulo mediante la fórmula: $\text{área} = \text{base} * \text{altura}$ | Mostrar el resultado del área del rectángulo Área Final |

Tabla 2. Estructura en partes del problema.

Fuente: Desafío Latam

- **Paso 2:** Realizar el diagrama de flujo, partiendo por el inicio de todo diagrama de flujo, el cual, debe ir encerrado dentro de un óvalo, posteriormente se realizaría la lectura de los datos (base y altura) mediante el paralelogramo, quedando de la siguiente manera:

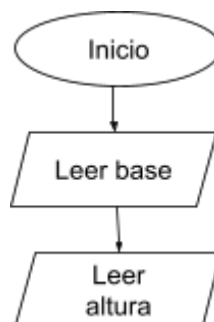


Imagen 13. Paso 2 - Inicio y lectura de los datos.

Fuente: Desafío Latam

- **Paso 3:** Realizar los procedimientos necesarios para obtener los resultados que requiere el problema. En este caso, sería realizar la multiplicación de la base por la altura ($\text{base} \times \text{altura}$) y posteriormente mostrar el resultado en un paralelogramo, mientras que el final del diagrama de flujo se encierra en un óvalo, como se muestra en la siguiente imagen:

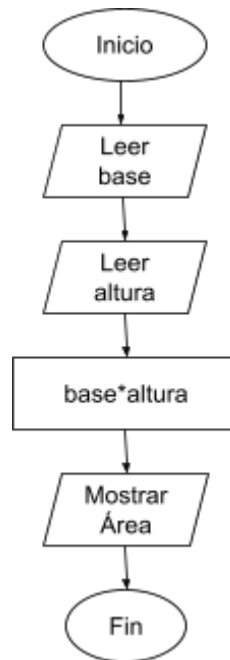


Imagen 14. Paso 3 - Procesamiento de datos, muestra y final.
Fuente: Desafío Latam

Pseudocódigo

Corresponde a otra forma de representar un algoritmo. Es una estructuración de los pasos a seguir de un algoritmo escrito de una manera secuencial, en la que se detallan todos los pasos a seguir. Se le llama pseudocódigo ya que el objetivo es escribir el algoritmo como si se estuviera escribiendo en un lenguaje de programación, sin que sea un lenguaje específico. Es decir, escribimos con nuestras palabras las acciones que correspondan, como: Leer Valor, Mostrar resultado, entre otras. Por ejemplo, si queremos realizar el pseudocódigo para un programa que sume dos números ingresados por el usuario y muestre el resultado, sería algo como lo siguiente:

| Datos de Entrada | Proceso | Datos de Salida |
|---|--|---|
| Números ingresados por el usuario numero1 numero2 | Realizar la suma de los dos números numero1 + numero2 | Mostrar el resultado de la suma de ambos números Sumatoria Final |

Tabla 3. Estructura en partes del problema.
Fuente: Desafío Latam

Esto, representado en pseudocódigo, se grafica de la siguiente manera:

```
Inicio
  Leer numero1
  Leer numero2
  numero1 + numero2
  Mostrar numero1 + numero2
Fin
```

Como se pudo observar en el ejemplo anterior, en el pseudocódigo se utiliza la instrucción **leer** para especificar que el usuario tiene que ingresar un valor y **mostrar** para imprimir el valor en pantalla, también declarando el **inicio** y el **fin** del algoritmo.

Ventajas de usar pseudocódigo

La gracia de escribir el pseudocódigo de un algoritmo es que permite centrarse en la estructura misma del algoritmo más que en las condiciones propias de un lenguaje de programación. Una vez listo el pseudocódigo, la implementación de este en un lenguaje de programación suele ser más sencilla.

El pseudocódigo, al igual que el diagrama de flujo, nos permite pensar en términos independientes al lenguaje de programación y concentrarnos en describir lo que estamos tratando de hacer y los pasos necesarios, en lugar de cómo lograrlo. En programación, es muy importante poder comunicar al computador lo que tiene que hacer paso a paso. Por consiguiente, escribir pseudocódigo y diagramas de flujo ayuda reforzar esta habilidad.

Ejercicio guiado: Pseudocódigo

Crear un algoritmo en pseudocódigo que calcule el área de un rectángulo, partiendo de los datos ingresados por el usuario como lo son la base y la altura.

- **Paso 1:** Crear una tabla para identificar las tres partes fundamentales del problema:

| Datos de Entrada | Proceso | Datos de Salida |
|---|---|--|
| Valores ingresados por el usuario base altura | Realizar el cálculo del área del rectángulo mediante la fórmula: $\text{área} = \text{base} * \text{altura}$ | Mostrar el resultado del área del rectángulo Área Final |

Tabla 4. Estructura en partes del problema.

Fuente: Desafío Latam

- **Paso 2.** Dar inicio a la escritura de nuestro código en pseudolenguaje, indicando los procesos y procedimientos, siguiendo la secuencia correspondiente al enunciado. Por lo que primeramente debemos iniciar el pseudocódigo y luego leer los valores de entrada:

```
Inicio
  Leer base
  Leer altura
```

- **Paso 3:** Realizar la operación requerida en el proceso al inicio del ejemplo, lo cual, sería realizar el cálculo del área mediante la multiplicación de la base por la altura:

```
base * altura
```

- **Paso 4:** Una vez multiplicado los datos ingresados por el usuario, se procede a terminar el pseudocódigo mostrando el resultado del área del rectángulo e indicando el fin del algoritmo, quedando todo el pseudocódigo de la siguiente manera:

```
Inicio
  Leer base
  Leer altura
  base * altura
  Mostrar base * altura
Fin
```


Ejercicio Propuesto (1)

Realizar un diagrama de flujo que permita determinar el área de un triángulo rectángulo si el usuario ingresa los valores como la base y la altura del triángulo. Fórmula: $\text{área} = (\text{base} * \text{altura}) / 2$.

Ejercicio Propuesto (2)

Desarrollar un diagrama de flujo que permita calcular el área de una circunferencia partiendo de la fórmula: $\text{área} = \pi * \text{radio}^2$. (La constante pi tiene un valor aproximado de 3,14).

Ejercicio Propuesto (3)

Diseñar un diagrama de flujo para calcular el área de un rombo, en donde el área es se calcula mediante la fórmula: $\text{área} = (\text{Diagonal mayor} * \text{diagonal menor}) / 2$

Ejercicio Propuesto (4)

Realizar un algoritmo en pseudocódigo que permita determinar el área de un triángulo rectángulo si el usuario ingresa los valores como la base y la altura del triángulo. Fórmula: $\text{área} = (\text{base} * \text{altura}) / 2$

Ejercicio Propuesto (5)

Desarrollar un pseudocódigo que permita calcular el área de una circunferencia partiendo de la fórmula: $\text{área} = \pi * \text{radio}^2$. (La constante pi tiene un valor aproximado de 3,14).

Ejercicio Propuesto (6)

Diseñar un pseudocódigo para calcular el área de un rombo, en donde el área es se calcula mediante la fórmula: $\text{área} = (\text{Diagonal mayor} * \text{diagonal menor}) / 2$.

Introducción a JavaScript

Competencias

- Reconocer el entorno de desarrollo y las herramientas de trabajo recomendadas, para interactuar con una página web y ejecutar comandos de JavaScript.
- Codificar un programa en Javascript utilizando el inspector de elementos y diagramas de flujo para la implementación de un algoritmo.

Introducción

JavaScript es un lenguaje de programación que se utiliza principalmente para agregar dinamismo e interactividad a una página o aplicación web. Así, gracias al uso de JavaScript, es posible crear aplicaciones web con movimiento e interacción con el usuario.

Técnicamente JavaScript es un lenguaje de programación del tipo interpretado. Esto quiere decir, que el computador lee línea a línea el código escrito por el programador y ejecuta las acciones allí contenidas, sin preocuparse del resto del código fuente. Es un proceso similar al de un intérprete de lenguaje de señas, donde este gestiona las señas a medida que otra persona habla.

Para entender de qué manera funciona JavaScript hay que ver con qué otros actores se encuentran relacionados. Esencialmente dentro del desarrollo web Front-End, JavaScript se encuentra estrechamente relacionado con HTML y CSS.

Entender estos conceptos te permitirá identificar los elementos que interactúan en una página web y comprender el alcance de JavaScript en el desarrollo. Además, al finalizar el capítulo tendrás nociones básicas de la sintaxis de este lenguaje, para ir poco a poco adquiriendo herramientas para realizar programas cada vez más complejos.

Entendiendo las herramientas para Desarrolladores en los navegadores

La gran mayoría de los navegadores modernos (prácticamente todos) traen incorporadas una serie de herramientas diseñadas para facilitar la vida de los desarrolladores web. La consola de desarrollo de JavaScript es una herramienta maravillosa a la hora de analizar nuestro código cuando estamos programando. Esta nos muestra mensajes de información, error o alerta. Además, incluye un inspector de elementos para ver el código HTML de una página web y verdaderos depuradores de código que nos permiten identificar errores más fácilmente. También nos permite interactuar con la página, ejecutando expresiones o comandos de JavaScript.

Para acceder a la consola desde Google Chrome y/o Mozilla Firefox, presiona las teclas **Control + Mayus + J**, desde Mac **cmd + alt + J** y desde Internet Explorer la tecla **f12**. La barra de herramientas para desarrolladores, como podemos ver en la siguiente imagen, muestra varias pestañas en las que se pueden encontrar información de las peticiones HTTP (los elementos de la página que se cargan de internet), los errores, análisis del CSS, JavaScript, los Logos (mensajes), errores y advertencias de seguridad.

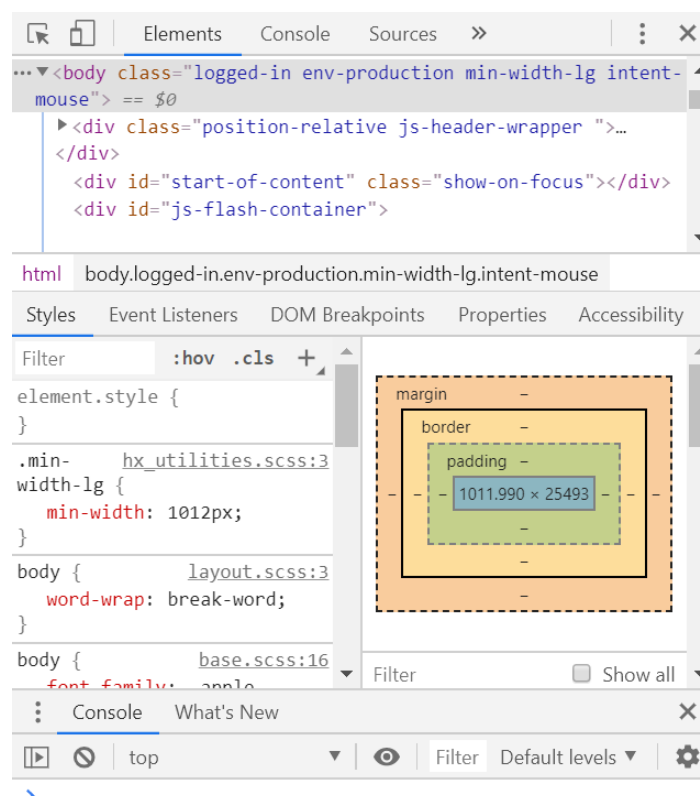


Imagen 15. Las herramientas de desarrollador de Google Chrome.

Fuente: Desafío Latam

Sintaxis del Lenguaje

Antes de explicar la sintaxis del lenguaje, es preciso ver algunos ejemplos de JavaScript:

- Sumar dos números:

```
1 + 2;
```

- Mostrar un mensaje de alerta:

```
alert('Este es un mensaje de alerta');
```

- Mostrar un dato en la consola de desarrolladores:

```
console.log('Mensaje para el desarrollador');
```

La sintaxis de un lenguaje de programación se refiere al conjunto de reglas que se deben seguir al momento de escribir el código. Esto quiere decir que hay formas de escribir un lenguaje de programación y no es llegar y escribir cualquier cosa.

La sintaxis de JavaScript está inspirada en otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- **No se toman en cuenta los espacios en blanco y las nuevas líneas:** El intérprete de JavaScript ignora cualquier espacio en blanco que sobre, por lo que el código se puede ordenar de forma adecuada para entenderlo mejor (tabulando las líneas, añadiendo espacios, creando nuevas líneas, etc.)
- **Distinguen las mayúsculas y minúsculas:** En JavaScript si se intercambian mayúsculas y minúsculas en el código el script no funciona. Puedo tener dos variables llamadas: número y otra llamada Número, ambas son distintas. A esta característica se le conoce como case sensitive.
- **No se define el tipo de las variables:** Al crear una variable, no es necesario indicar el tipo de dato que almacenará. De esta forma, una misma variable puede almacenar diferentes tipos de datos durante la ejecución del script. Una variable puede guardar primero un número y luego un texto sin problemas.
- **No es necesario terminar cada sentencia con punto y coma (;):** Aunque JavaScript no obliga a hacerlo, es conveniente y se recomienda terminar cada sentencia con el carácter del punto y coma (;).

- **Se pueden incluir comentarios:** Los comentarios se utilizan para añadir información en el código. Aunque el contenido de los comentarios no se visualiza por pantalla, si que se envía al navegador del usuario junto con el resto del script. Debido a lo anterior, es necesario extremar las precauciones sobre la información incluida en los comentarios.
- **Se deben respetar las palabras reservadas:** al igual que la mayoría de los lenguajes de programación, JavaScript tiene una serie de [palabras reservadas](#) que son propias del lenguaje de programación, las cuales identifican acciones, sentencias, procesos, métodos y/o funciones de JavaScript, como el caso de var, let, for, if, function, export, in, entre otras, por lo tanto, no pueden ser utilizadas para otro propósito distinto al creado e implementado por el lenguaje de programación.

¿Cómo incluir JavaScript en un archivo HTML?

La integración de JavaScript y HTML es muy flexible, ya que existen al menos tres formas para incluir código JavaScript en las páginas web:

1. Entre las etiquetas script dentro del código HTML, ejemplo: `<script>alert("Soy JavaScript")</script>`.
2. Como un atributo de un elemento HTML, como por ejemplo usando el atributo onclick, lo cual resultaría en `<article onclick="miFuncion()"></article>`. Esta es la opción menos recomendada y no se emplea actualmente.
3. Entre las etiquetas **script**, pero esta vez usando el atributo **src** del propio elemento para darle la ruta del archivo con extensión .js que se utilizará. Esto implica que el código JavaScript y el código HTML estarán en archivos separados. Esta es la opción más recomendada y más implementada en la actualidad.

Uno de los puntos más importantes que todo programador debe tener claro, es la creación de un código limpio, ordenado y simple de entender. En base a esto, la tercera forma de incluir JavaScript a HTML es la más recomendada dado que mantiene separados los diferentes aspectos de la página o aplicación web, facilitando su mantención y evolución futura.

Para comprender cómo utilizar esta forma recomendada, puedes consultar el documento **Material Apoyo Lectura - Incluyendo JavaScript en un archivo HTML** ubicado en "Material Complementario". Aquí podrás ver paso a paso cómo incluir un archivo JavaScript como un recurso referenciado en el HTML.

Ejercicio guiado: Mi primer programa en JavaScript

Realizar un ejemplo que permitirá interactuar con JavaScript mediante nuestro navegador, para que podamos ver cómo la consola ejecuta el código JavaScript de manera automática.

Realizaremos algunas instrucciones breves, como mostrar mensajes de texto en ventanas emergentes, sumas y restas.

- **Paso 1:** Abrir la consola del navegador web con **Control + Mayus + J** desde PC, o **cmd + alt + J** desde Mac, o haciendo un clic con el botón derecho de nuestro ratón sobre el navegador de preferencia, posteriormente un clic con el botón izquierdo sobre la palabra desplegada en el menú denominada: “inspeccionar elemento” (en Firefox) o “inspeccionar” (en Chrome) y se nos abrirá de la siguiente manera:

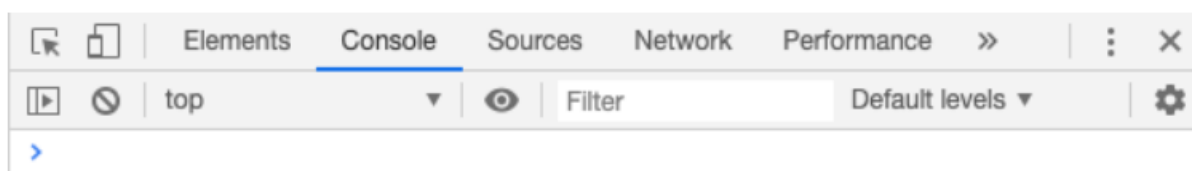


Imagen 16. Consola del navegador Chrome.

Fuente: Desafío Latam

Lo primero que haremos es una suma matemática, para que puedas observar que la consola trabaja automáticamente bien.

- **Paso 2:** Una vez abierta la barra de herramientas para desarrolladores y habiendo seleccionado la pestaña consola:
 - Ingresar $10 + 23$
 - Apretar Enter

Hecho lo anterior, el resultado será el siguiente:

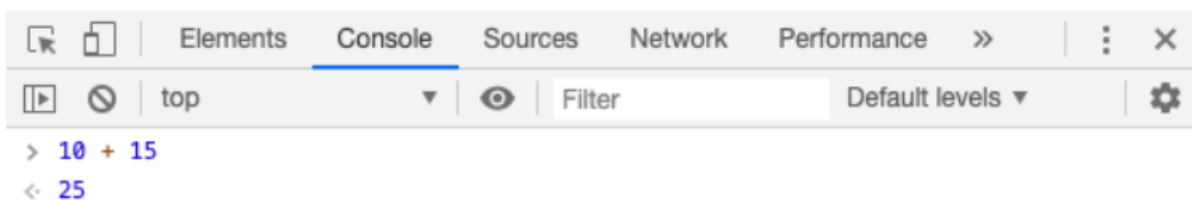


Imagen 17. Suma en la consola.

Fuente: Desafío Latam

Ahora, esta misma operación la haremos utilizando un poco de JavaScript. Si no entiendes nada de esto, tranquilo/a más adelante veremos cada detalle de este ejemplo.

IMPORTANTE. Es posible agregar un salto de línea sin ejecutar el código usando la combinación de teclas Shift + Enter

- **Paso 3:** Se declaran dos variables, num1 y num2, cada variable almacenará un valor y luego las sumamos.

```
var num1 = 10;  
var num2 = 15;
```

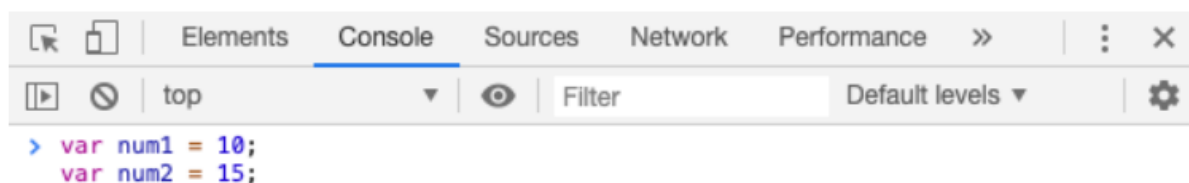


Imagen 18. Suma usando variables.
Fuente: Desafío Latam

- **Paso 4:** Ahora, agregamos la siguiente línea para obtener el resultado (para hacer un salto de línea, sin ejecutar, lo hacemos presionando las teclas **Shift + Enter**).

```
console.log(num1 + num2);
```

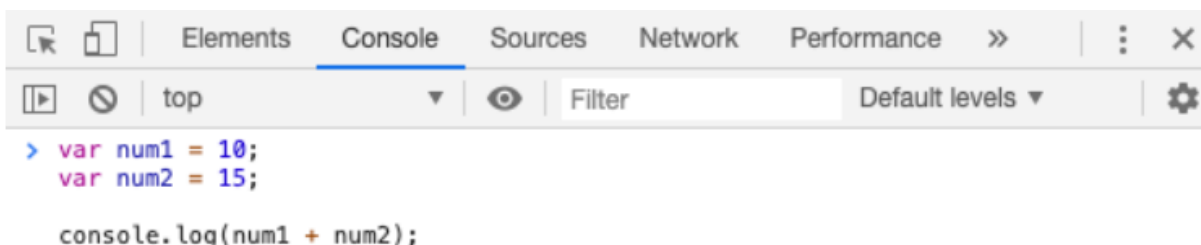


Imagen 19. Suma usando variables.
Fuente: Desafío Latam

Quedando como resultado:

```
> var num1 = 10;  
var num2 = 15;  
  
console.log(num1 + num2 );  
25
```

Imagen 20. Suma usando variables.
Fuente: Desafío Latam

- **Paso 5:** Por último, agregaremos el siguiente código:

```
alert ("Hola! esto es JavaScript en {desafío} latam_");
```

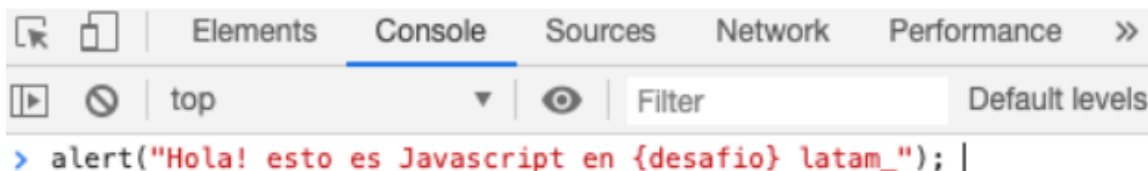


Imagen 21. Mostrando el resultado en una alerta.

Fuente: Desafío Latam

Damos Enter y obtendremos lo siguiente:

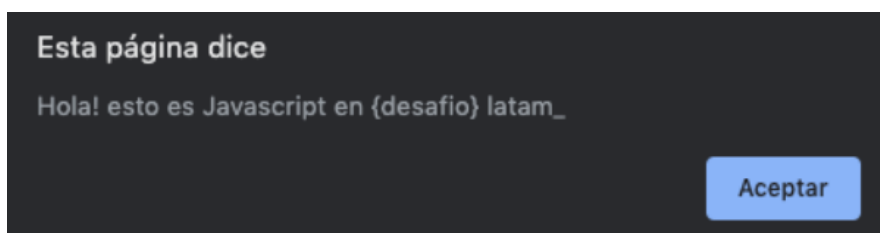


Imagen 22. Resultado de la alerta en el navegador.

Fuente: Desafío Latam

Continuando con nuestro primer programa en JavaScript desde la consola del navegador web, vamos a realizar la operación básica de resta de dos números y mostramos en una alerta el resultado de la operación.

- **Paso 6:** Ahora procedamos a crear dos variables con la palabra reservada "var" y les damos el nombre de valor1 y valor2, quedando de la siguiente manera:

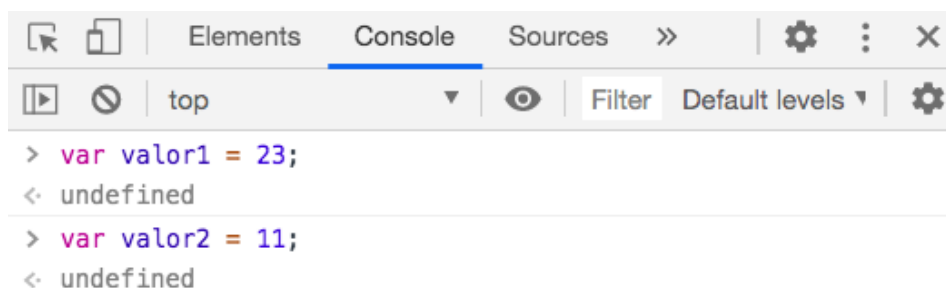


Imagen 23. Ingresando variables en la consola del navegador.

Fuente: Desafío Latam

- **Paso 7:** Una vez ingresados los valores y almacenados, procedemos a realizar la resta de ambas variables (los valores ingresados), como se observa a continuación:

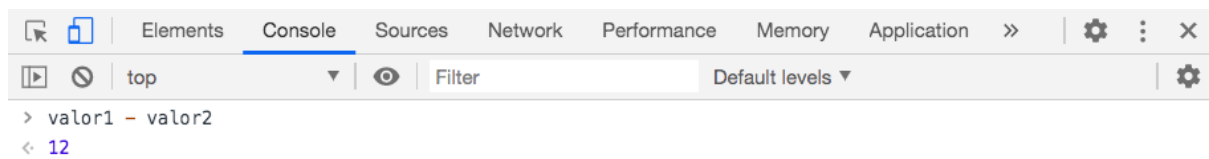


Imagen 24. Resta de variables en la consola del navegador.
Fuente: Desafío Latam

- **Paso 8:** Luego, nos queda mostrar el resultado mediante la instrucción `alert` en la consola del navegador, quedando de la siguiente manera:

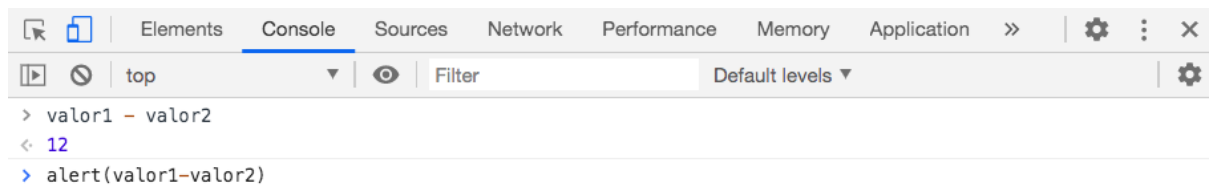


Imagen 25. Agregando la instrucción `alert()` en la consola del navegador.
Fuente: Desafío Latam

- **Paso 9:** Finalmente, al escribir la instrucción anterior y presionar la tecla enter, se mostrará una ventana emergente con el resultado directo de la resta, como se muestra a continuación:



Imagen 26. Ventana emergente con el resultado de la resta.
Fuente: Desafío Latam

Diagrama de flujo a código JavaScript

En el capítulo anterior se presentaron los diagramas de flujo como herramientas para representar gráficamente un algoritmo. Ahora que ya se conoce lo básico de JavaScript, estamos en condiciones de convertir un diagrama de flujo en un código para JavaScript.

Para esto, es posible tomar como referencia el siguiente diagrama de flujo:

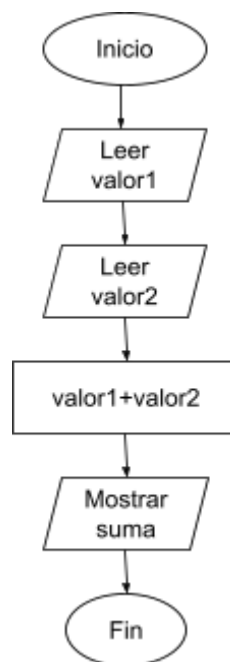


Imagen 27. Diagrama de flujo a transformar en código.
Fuente: Desafío Latam

Según este diagrama (que describe una suma de dos números), es necesario:

- Leer el valor de un primer número.
- Guardar este número en una variable llamada valor1.
- Leer el valor de un segundo número.
- Guardar este nuevo número en una variable llamada valor2.
- Mostrar el resultado de sumar los dos números ingresados.

Para obtener valores a partir del usuario, es posible usar la función **prompt**. Esta función abre una ventana en el navegador con una entrada de texto y un mensaje definido por el programador, para que allí el usuario indique el valor solicitado.

```
prompt("Ingresa un número");
```

El valor que ingrese el usuario puede ser guardado en una variable, para ser utilizado posteriormente:

```
var num = prompt("Ingresa un número");
```

Hay que tener cuidado cuando se usa la función **prompt**, ya que el resultado que entrega siempre es un texto o cadena de caracteres, aunque escribamos un número. Así, si el usuario ingresa el número uno, **prompt** entregará el valor "1" (que es un texto) y no el número 1. Para solucionar eso, es posible utilizar otra función llamada **parseInt**, la cual convierte un texto en el número correspondiente, es decir, transforma el tipo de dato carácter a numérico, siempre que esto sea posible.

```
var numero = parseInt("1");
```

Para mostrar el contenido hay dos opciones. La primera es usar la función `console.log`, que muestra el resultado en la consola del navegador.

```
console.log('Valor a mostrar en la consola');
```

La segunda opción es usar la función `alert`. Esta función abre una ventana similar a la de `prompt`, pero esta vez sólo para mostrar información y no para ingresar datos.

```
alert('Valor a mostrar en la ventana emergente')
```

En base a lo anterior, el código del diagrama de flujo que se muestra en la imagen 27, se expresa de la siguiente manera:

```
var n1 = prompt("Ingrese un primer número entero: ");  
var n1 = parseInt(n1);  
var n2 = prompt("Ingrese un segundo número entero: ");  
var n2 = parseInt(n2);  
alert(n1+n2);
```

Ejercicio guiado: De diagrama de flujos a código

Llevar el siguiente diagrama de flujos a código mediante el lenguaje de programación JavaScript, para ello, se solicita restar dos números ingresados por el usuario y mostrar el resultado. Al desarrollar este código, quedaría de la siguiente manera:

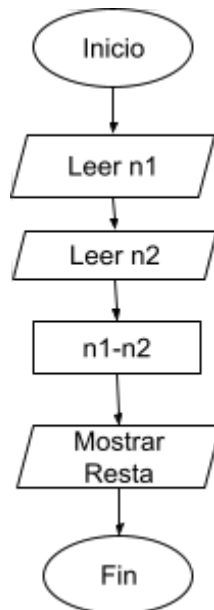


Imagen 28. Diagrama de flujo Resta de dos números.

Fuente: Desafío Latam

- **Paso 1:** Para llevar el diagrama de flujo anterior a JavaScript, primero se debe identificar la o las variables de entrada, en este caso: "n1 y n2", donde se utilizará la función `prompt()` para solicitarlos y enviarlos a las variables respectivas:

```
var n1 = prompt("Ingrese un primer número entero: ");  
var n1 = parseInt(n1);  
var n2 = prompt("Ingrese un segundo número entero: ");  
var n2 = parseInt(n2);
```

- **Paso 2:** Definir el proceso de resta con esas dos variables (n1-n2) y se muestra el resultado final. Implementar una función propia de JavaScript que es el alert():

```
var n1 = prompt("Ingrese un primer número entero: ");  
var n1 = parseInt(n1);  
var n2 = prompt("Ingrese un segundo número entero: ");  
var n2 = parseInt(n2);  
alert(n1-n2);
```

El resultado obtenido para el código anterior si el usuario ingresa los números 5 y 9 respectivamente, sería:



Imagen 29. Resultado de la resta en ventana emergente.

Fuente: Desafío Latam

Ejercicio guiado: De pseudocódigo a código

Realizar un pseudocódigo que solicite al usuario el ingreso de dos números enteros y calcule la multiplicación de ambos, mostrando por pantalla el resultado.

- **Paso 1:** Implementar un pseudocódigo con los procedimientos que estudiamos anteriormente, el resultado sería:

```
Inicio
  Leer num1
  Leer num2
  num1 * num2
  Mostrar num1 * num2
Fin
```

- **Paso 2:** Si llevamos el pseudocódigo anterior a JavaScript, lo primero que podemos observar es que se están leyendo dos valores de entrada y estos deben ser ingresados por el usuario, lo que quiere decir que se debe implementar la instrucción `prompt()` para solicitar al usuario los datos y almacenarlos en una variable para poder realizar la suma y mostrarlos en una ventana emergente. Quedando el código en JavaScript de la siguiente manera:

```
var num1 = prompt("Ingrese un primer número entero: ");
var num1 = parseInt(num1);
var num2 = prompt("Ingrese un segundo número entero: ");
var num2 = parseInt(num2);
alert(num1*num2);
```

Siendo el resultado de la multiplicación si se ingresa el número 3 y el número 5:

15

Ejercicio Propuesto (7)

En base al siguiente diagrama de flujo, construye el código en JavaScript que muestra el promedio de dos números:

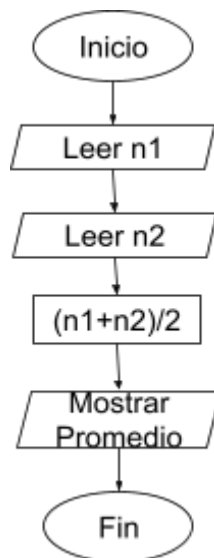


Imagen 30. Diagrama de flujo promedio de dos números
Fuente: Desafío Latam

Ejercicio Propuesto (8)

En base al siguiente pseudocódigo, construye el código en JavaScript que muestra la conversión de pulgadas a metros, de acuerdo al siguiente pseudocódigo:

```
Inicio
  Leer pulgada
  pulgada * 0.254
  Mostrar pulgada * 0.254
Fin
```

Ejercicio Propuesto (9)

Realizar un programa con JavaScript que al ingresar dos valores (valor1 = 20 y valor2 = 7) calcule la suma, la resta, la multiplicación y la división de esos dos números en la consola del navegador web de tu preferencia, posteriormente muestra en una ventana emergente (alerta) un mensaje que indique solamente el resultado de la multiplicación.

Variables en JavaScript

Competencias

- Distinguir variables y constantes como recurso para almacenar valores en JavaScript.
- Codificar una rutina JavaScript utilizando las variables y sus distintos tipos de datos para resolver un problema planteado.

Introducción

Así como en matemáticas entendemos una variable como un elemento que permite hacer referencia a algún valor, tal como en las ecuaciones y fórmulas, en programación nos referimos a ellas como un espacio de memoria donde se almacena un dato.

Esto es esencial para cualquier programa de mediana complejidad, pues permite que el programa mantenga su funcionamiento, independientemente de los valores que ingresen a él.

En este capítulo abordaremos este concepto, sus ventajas y desventajas. Además, tendremos un primer acercamiento a lo que son los tipos de datos. De esta manera, iremos adquiriendo poco a poco herramientas que nos permiten agregarle dinamismo a la lógica de los programas que desarrollemos, adquiriendo la experiencia necesaria para resolver desafíos cada vez más complejos.

Variables

En programación las variables nos permiten almacenar datos en pequeñas cajas. Por ejemplo, si tenemos que guardar el nombre de un usuario que ingresó a la plataforma de desafío latam, para luego imprimirlo por pantalla y así el usuario pueda ver su nombre, lo podemos guardar en una variable, en alguna “cajita”.

Las variables en JavaScript se declaran de la siguiente manera:

```
var usuario = 'Alexis';  
var edad = 30;
```

Se declaran con la palabra var, esta palabra reservada se encarga de decirle a JavaScript o al computador que lo que viene es una variable. Luego, seguido de var viene el nombre que queremos colocarle a la variable, a continuación después del signo = viene el dato que estamos almacenando.

Para poder utilizar una variable hay miles de formas, todo va a depender de la manera que nosotros creemos que es más conveniente manipularla.

Como su nombre lo dice, en una variable su valor puede variar. Vamos a hacer un ejemplo para poder entenderlo. En el siguiente código declaramos una variable “nombre” y almacenamos el dato de tipo texto (string) “Alexis”:

```
var nombre = "Alexis";  
alert(nombre);
```

Si bien el código anterior podría haber sido escrito de la siguiente forma, esta no es la más recomendable.

```
alert('Alexis');
```

Esta forma no es correcta ya que siempre tendrá el mismo resultado. Es decir, siempre mostrará el mensaje Alexis en la alerta. La gracia de usar variables es que permiten conocer a simple vista qué es lo que se está guardando en ella.

Para saber a simple vista qué es lo que se está almacenando en una variable es importante nombrarlas adecuadamente. En el ejemplo a continuación, con sólo leer el nombre de la variable es posible saber qué es lo que está almacenando.

```
alert(nombre);
```

El siguiente código también es válido, pero no se puede saber con solo leerlo cuál va a ser su resultado.

```
alert(a);
```

Esto dado que la variable “a” podría guardar un nombre, la edad de una persona, la cantidad de pasos que dio en la última hora o cualquier cosa. Debido a lo anterior, es importantísimo nombrar adecuadamente las variables, es una buena práctica que marca diferencias entre un buen y un mal desarrollador.

Junto con lo anterior, la importancia de las variables es que se pueden utilizar en más de una ocasión. Si se considera el siguiente código:

```
alert('Alexi');  
alert('Alexi');  
alert('Alexi');  
alert('Alexi');  
alert('Alexi');
```

Es posible notar que el nombre está mal escrito, ya que dice Alexi y debería ser Alexis. Para solucionar este error de escritura, habría que cambiar todos los Alexi por Alexis, es decir, realizar ese cambio 5 veces. ¿Qué pasaría si el código fuese mucho más grande y hubiera que realizar ese cambio 100 o 1.000 veces? Sería una situación muy poco práctica. Si para lo anterior se hubieran utilizado variables, habría sido necesario realizar un solo cambio.

```
var nombre = 'Alexi';  
  
alert(nombre);  
alert(nombre);  
alert(nombre);  
alert(nombre);  
alert(nombre);
```

En el código anterior, basta con corregir el error en una única línea (la primera) y el programa funcionará tal y como se espera.

Continuando con el ejemplo anterior, ahora borramos y declaramos la misma variable pero no le asignamos ningún valor (creamos nuestra caja, la dejamos abierta pero no colocamos nada dentro de ella por el momento)

```
var nombre;
```

Agregamos también otra variable, la cual almacenará una edad:

```
var edad;
```

No le asignamos a ninguna de las dos variables un valor, ya que en el ejemplo queremos ver el por qué del nombre "variable". El valor se lo asignaremos nosotros cuando carguemos la página .html en el navegador.

Para hacer esto, agregamos la siguiente línea de código:

```
nombre = prompt("Ingrese su nombre");  
edad = prompt("Ingrese su edad");  
  
// Le asignamos el valor a cada variable desde el valor que ingresamos  
en el método prompt.
```

Lo que el método `prompt()` nos permite hacer aquí, es solicitar que el usuario ingrese un dato, luego ese dato ingresado lo almacenamos en cada una de las variables que declaramos.

Luego, utilizando el comando `document.write()` imprimimos en la página .html los valores de cada variable de la siguiente manera:

```
var nombre;  
var edad;  
  
nombre = prompt("Ingrese su nombre");  
edad = prompt("Ingrese su edad");  
  
document.write(nombre);  
document.write(" ");  
document.write(edad);
```

```
document.write(nombre);  
document.write(" ");  
document.write(edad);
```

El ejemplo mostraría como resultado:

```
Alexis 30
```

En el comando `document.write()`; lo que hicimos fue mostrar los valores ingresados y almacenados en cada variable. El operador `+` nos sirve para concatenar o unir dos valores distintos.

En este ejemplo pudimos apreciar básicamente la función de las variables, que el valor cambia cada vez que nosotros cargamos la página `.html` e ingresamos datos.

Constantes

Las constantes son parecidas a las variables ya que almacenan datos, con la diferencia que no pueden cambiar porque son “constantes”. Una variable puede tener un valor, pero luego puede tener otro y así sucesivamente, pero cuando estemos programando vamos querer en muchos casos almacenar datos los cuales no queremos que cambien y para esto utilizamos las constantes.

Cuando yo declaro una constante esta ya no es modificable, sino que solamente tiene carácter de lectura.

La forma en que nosotros podemos declarar una constante es de la siguiente manera:

```
const añoNacimiento = 1988;
```

En vez de “var” como en las variables, acá utilizamos la palabra reservada “**const**” seguido del nombre que le pondremos a la constante y posterior su valor.

En la versión anterior a ES6 existía solamente una manera de declarar variables, que era con la palabra “var”. Ahora con ES6 (ECMAScript 6) tenemos dos opciones más: “const” y una manera que no vimos pero que brevemente explicaremos aquí, que es “let”.

“var”, en resumen tiene ámbito de función y la podemos llamar dentro y hacia una función. Pero con “let” podemos crear variables en ámbitos de bloques. Es decir, por ejemplo, si declaramos una variable con la palabra reservada “let” denominada “i” en un ciclo repetitivo o en una estructura condicional, esta variable solamente la podremos ocupar dentro del mismo ciclo repetitivo o estructura condicional. Al igual que “const”, que también tiene ámbito de bloques, pero a diferencia de “var” y “let” no puede ser reasignado su valor.

Debido a que este es un primer acercamiento al mundo de la programación, las ventajas y desventajas entre var, let y const se abordarán en las siguientes unidades a lo largo de la carrera, al igual que las estructuras condicionales y los ciclos repetitivos en el código.

Ejercicio guiado: Trabajando con variables y constantes

Crear un programa con JavaScript desde un archivo externo que solicite al usuario dos números enteros mediante el uso del método “prompt()”, almacenando esos dos números en variables separadas y realizando las cuatro operaciones matemáticas básicas (suma, resta, multiplicación y división). Luego mostrar el resultado para cada operación en el mismo documento .html mediante el comando `document.write()`.

- **Paso 1:** Crear una carpeta en tu computador en tu sitio de trabajo, con el nombre que desees, luego crea dos archivos dentro de ella, el primero será el index.html y el segundo será el script.js.

| Nombre | Fecha de modificación | Tamaño | Clase |
|------------|-----------------------|-----------|---------------|
| index.html | hoy 19:47 | 291 bytes | HTML document |
| script.js | hoy 19:48 | 493 bytes | JavaScript |

Imagen 31. Carpeta contenedora con los dos archivos creados.

Fuente: Desafío Latam

- **Paso 2:** En el documento index.html, crear la estructura básica de un documento HTML y enlace mediante la etiqueta script en el atributo “src” el archivo externo de JavaScript para que lo puedas implementar.
- **Paso 3:** Ahora es momento de trabajar en nuestros script.js, para ello, pedir al usuario los datos y almacenarlos en una variable por separado. En este caso implementaremos el método “prompt()” para solicitar al usuario los datos y pasarlos a una variable, quedando de la siguiente manera:

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

- **Paso 4:** Realizar las operaciones matemáticas e indicar mediante el `document.write()` cuál es la operación y cuál sería el resultado. En este caso, se debe tener cuidado con el valor de las variables, debido a que el método `prompt` las recibe como cadena de caracteres y no números, por lo que se deben convertir a número empleando la función `parseInt()`, quien se encargará de la transformación de cadena de caracteres a números enteros. Esa función se estará utilizando nuevamente en los siguientes capítulos donde será explicada más en detalle.

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

document.write("La suma es: ");
document.write(parseInt(num1)+parseInt(num2));
document.write(" ");
document.write("La resta es: ");
document.write(parseInt(num1)-parseInt(num2));
document.write(" ");
document.write("La multiplicación es: ");
document.write(parseInt(num1)*parseInt(num2));
document.write(" ");
document.write("La división es: ");
document.write(parseInt(num1)/parseInt(num2));
```

Luego, al abrir el archivo `index.html` en el navegador web, e ingresar 12 y 34 respectivamente, encontrarás el siguiente resultado:

Este es un documento HTML con JavaScript

La suma es: 46 La resta es: -22 La multiplicación es: 408 La división es: 0.35294117647058826

Imagen 32. Resultado en el documento `index.html`.

Fuente: Desafío Latam

Ahora veremos cómo utilizar una constante y la diferencia entre una variable. Haremos una operación matemática con un valor constante que nunca cambia y con un dato variable.

- **Paso 5:** Reemplazar el contenido del archivo “.js” con el siguiente código:

```
const pi = 3.14;  
var num;  
  
num = prompt("Ingrese número para multiplicar por 'pi': ");  
alert(pi*num);
```

- **Paso 6:** El resultado a continuación corresponde al ingreso del número 4:



Imagen 33. Resultado del código anterior en ventana emergente.

Fuente: Desafío Latam

¿Qué hicimos? Simple, declaramos una constante llamada “pi” la cual matemáticamente tiene un valor que nunca cambia, ya que siempre “pi” va a tener un valor fijo. También declaramos una variable “num”, la cual obtendrá su valor de acuerdo a lo que nosotros ingresemos en pantalla con el método “prompt”.

Ejercicio propuesto (10)

Realizar un programa que permita determinar el área de un triángulo rectángulo si el usuario ingresa los valores como la base y la altura del triángulo. Fórmula: $\text{área} = (\text{base} \times \text{altura}) / 2$. Mostrar el resultado final en una alert.

Ejercicio Propuesto (11)

Desarrollar un programa en JavaScript que permita calcular el área de una circunferencia, partiendo de la fórmula: $\text{área} = \pi \times \text{radio}^2$. (La constante pi tiene un valor aproximado de 3,1416 y debe ser declarado en una variable del tipo constante).

Ejercicio Propuesto (12)

Diseñar un programa para calcular el área de un rombo, en donde el área es se calcula mediante la fórmula: $\text{área} = (\text{Diagonal mayor} \times \text{diagonal menor}) / 2$.

Tipos de datos en JavaScript

Competencias

- Identificar los tipos de datos primitivos en un programa JavaScript, para aplicarlos correctamente de acuerdo a la sintaxis del lenguaje.
- Codificar un programa utilizando operaciones de los tipos de datos para resolver un problema.

Introducción

Como lo mencionamos en el capítulo anterior, la programación es la manipulación de datos. Frente a esto, es de suma importancia conocer los tipos de datos que estaremos manipulando con este lenguaje de programación llamado JavaScript.

Por ejemplo, cuando estamos contratando un plan de teléfono, el ejecutivo nos solicita nuestros datos personales: Rut, Edad, Fecha de Nacimiento, Dirección, Teléfono y Correo electrónico. Aquí tenemos diferentes tipos de datos, números, fecha, mail, texto, etc. con los que el ejecutivo puede manipular como él quiera para elaborar un contrato. De esta forma nosotros como programadores tenemos que conocer el tipo de dato que estaremos manipulando para ver de qué manera lo podemos hacer y trabajar de una manera más óptima.

JavaScript tiene definido los tipos de datos que podemos manipular y los llama como tipos de datos primitivos, esto hace referencia a un tipo de dato que representa un solo valor y es inmutable (que no se puede modificar).

En este capítulo veremos en detalle los tipos de datos primitivos, comprenderemos a qué nos referimos cuando decimos que JavaScript es un lenguaje débilmente tipado junto a sus ventajas y desventajas. Manejar estos conceptos nos permitirá hacer operaciones con variables y controlar el comportamiento de nuestros programas en relación a los tipos de datos que intervienen en los procesos.

Tipos de Datos Primitivos

En JavaScript tenemos cinco tipos de datos primitivos:

- **Boolean:** Representa una entidad lógica y puede tener dos valores; true o false.
- **Number:** Un valor Numérico.
- **String:** Una cadena de texto.
- **Undefined:** Una variable a la cual no se le haya asignado valor tiene entonces el valor undefined.
- **Null:** Una variable que se le asigna y representa un valor nulo o vacío.

Con el siguiente código podemos identificar a qué tipo pertenece el dato que hemos almacenado en la variable "verdadero", utilizando el operador "typeof" que devuelve un texto "string" con dicha información:

```
var verdadero = true;  
// declaramos la variable "verdadero" con un valor Boolean.  
  
alert(typeof verdadero);  
// Este mensaje de alerta nos menciona el tipo de dato que es
```

(Cuando queremos comentar en JavaScript lo hacemos con // y luego el comentario)

Al ejecutar el código, indicará que se trata de un dato de tipo "boolean":

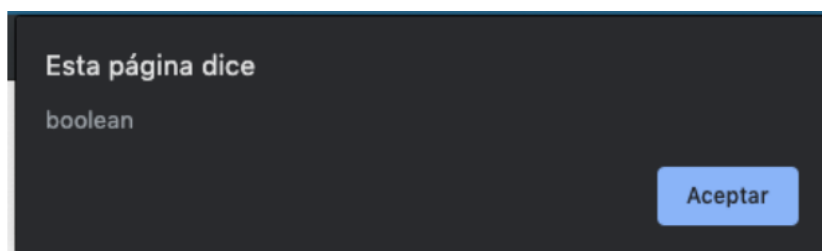


Imagen 34. Resultado ejecución de código utilizando el tipo de dato "boolean".

Fuente: Desafío Latam

En nuestro trabajo como desarrolladores vamos a ocupar tipos de datos "**string**" para manejar textos en nuestros elementos, ya sea valores, nombres de clases, etc. la mayoría de las cosas que sean formadas por palabras las podremos manipular usando strings.

Por consiguiente, los datos tipo **números** nos servirán para realizar tanto lógica como para manipular valores o atributos de los elementos de una página web. Vamos a poder construir

lógica para realizar acciones dependiendo del resultado de los cálculos de estos números, por ejemplo, contar la cantidad de letras en una entrada de texto, o realizar cambios en un elemento cuando el ancho de la página sea menor que cierto número.

Mientras que los **Booleanos** serán usados también en lógica verificando, por ejemplo, si una operación es verdadera o no, si un elemento existe o no dentro de la página web, junto con una infinidad de operaciones nos pueden dar un resultado final VERDADERO o FALSO, SÍ o NO, EXISTE o NO EXISTE.

Es por esto, lo importante que es conocer los tipos de datos en JavaScript, para así saber que tipo de datos estamos manipulando al programar.

JavaScript es débilmente tipado

En los diferentes lenguajes de programación existen dos formas de manejar los tipos de datos que se pueden almacenar en una variable. Una opción es que sea débilmente tipado y la otra es que sea fuertemente tipado.

Que un lenguaje de programación sea débilmente tipado significa que no es necesario que el programador indique explícitamente el tipo de dato que guardará una variable. Esto, dado que el intérprete o compilador infiere el tipo de dato según el valor asignado a la variable. Lo anterior implica, en la práctica, que sus variables pueden cambiar de un tipo a otro sin problemas durante la ejecución de un programa. Es decir, a una variable que en un principio fue del tipo Number se le puede asignar un dato del tipo String sin problemas, provocando que ahora la variable sea de este último tipo.

En los lenguajes de programación fuertemente tipados, en cambio, el tipo de dato debe ser declarado junto con la definición de la variable. Esto implica que esa variable sólo aceptará datos del tipo definido durante todo el programa. En este caso, si a una variable del tipo Number se le quisiera asignar un dato del tipo String, el programa arrojaría un error y no podría continuar con su ejecución.

Como ya lo mencionamos, JavaScript es un lenguaje débilmente tipado por eso no requiere de la declaración del tipo de dato al crear una variable, basta con sólo con indicarlo, como muestra el siguiente código:

```
var nombre;  
var edad;
```

Si JavaScript fuera fuertemente tipado, se tendría que definir el tipo de dato junto con la creación de la variable, como se muestra en el siguiente ejemplo (a modo ilustrativo dado que no es JavaScript):

```
String nombre;  
Number edad;
```

Puedes profundizar más en cómo influye la forma de manejar los tipos de datos en JavaScript en la forma en que programamos, puedes consultar el **Material Apoyo Lectura - Aplicar correctamente los tipos de datos**, ubicado en “Material Complementario”. En este artículo podrás conocer las ventajas y desventajas del tipado débil de JavaScript.

Rutinas con variables y tipos de datos.

En JavaScript, es importante siempre tener en cuenta el tipo de datos que estamos manejando en una variable, esto con el propósito de realizar las operaciones adecuadas dependiendo del dato disponible, debido a que si sumamos dos números que sean del tipo de dato string, el resultado obtenido no será precisamente la suma de esos dos números.

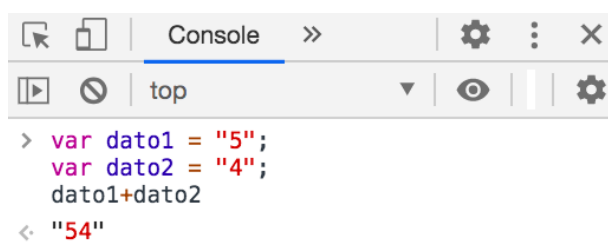


Imagen 35. Rutinas con variables y tipos de datos string.

Fuente: Desafío Latam

Como se puede observar en la imagen anterior, el resultado de la suma que debería ser el número 9, no es el mostrado, en su caso el resultado mostrado es “54”, esto se debe a que los dos valores que estamos intentando sumar (dato1 y dato2) están recibiendo valores del tipo string, mostrando otro resultado debido a otro proceso denominado concatenación, el cual, explicaremos más adelante.

Ahora, realicemos la prueba para visualizar qué tipos de datos son recibidos y guardados en una variable cuando se utiliza el método “prompt()”. Para ello, crea una carpeta en tu lugar de trabajo y luego dentro de ella crea dos archivos, un index.html y un script.js. Dentro del index.html debe ir lo siguiente:

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width,
```

```
initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Este es un documento HTML con JavaScript</h1>
  <script src="script.js"></script>
</body>
</html>
```

Luego en el archivo script.js, se debe incluir el código que permita solicitar un valor, almacenarlo en una variable y luego mostrarlo con un `document.write()`, indicando el tipo de dato mediante el operador "typeof". Como se muestra a continuación:

```
var dato = prompt("Ingrese un dato: ");
document.write(typeof dato);
```

En el caso de ingresar un número, por ejemplo el número tres (3), el resultado en el documento del navegador web sería:

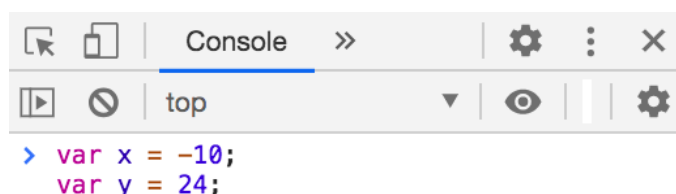
Este es un documento HTML con JavaScript

string

Imagen 36. Resultado de la ejecución del código.
Fuente: Desafío Latam

Como se puede observar, para el método "prompt()" siempre regresa un valor del tipo string. En este caso, en el uso de la función "parseInt()" es importante pasar los datos a tipo "Number" y poder realizar las operaciones matemáticas correspondiente con el tipo de datos "Number".

Realicemos ahora el siguiente ejemplo, en donde procesamos la suma de dos variables numéricas y dos variables como cadena de caracteres por la consola del navegador web. Veamos que sucede:



The image shows a web browser's developer console. The console tab is active, and the 'top' section is selected. The code entered in the console is: `> var x = -10;` and `var y = 24;`. The console interface includes standard icons for back, forward, and search, as well as a settings gear icon.

```
> var x = -10;
var y = 24;
```

Imagen 37. Iniciando variables con valores en la consola del navegador.
Fuente: Desafío Latam

Si realizamos una operación matemática básica con esos dos términos, el resultado sería:

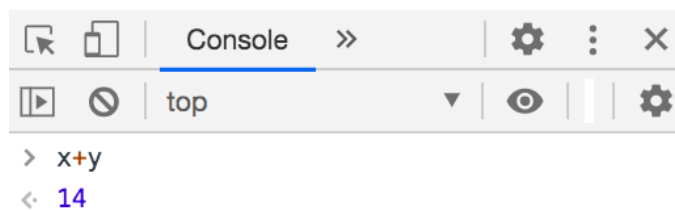


Imagen 38. Resultado de la ejecución del código anterior.
Fuente: Desafío Latam

Pero en el caso de tener dos variables con cadena de caracteres, como se muestra a continuación:

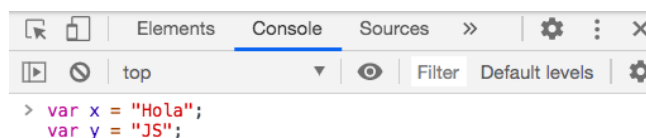


Imagen 39. Iniciando variables con valores en la consola del navegador.
Fuente: Desafío Latam

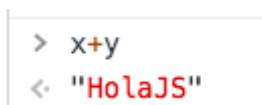


Imagen 40. Resultado de la ejecución del código anterior.
Fuente: Desafío Latam

Se puede observar que el resultado no es una suma como se hace en los procesos matemáticos básicos con números, sino, que une ambas palabras en una sola. A esto se le denomina **concatenación**.

Concatenación de variables

En un ejemplo anterior se mostró el valor de dos variables diferentes (nombre y edad) en una página web a través de JavaScript. Para ello, fue utilizado el siguiente código:

```
var nombre;  
var edad;  
  
nombre = prompt("Ingrese su nombre");  
edad = prompt("Ingrese su edad");  
  
document.write(nombre);  
document.write(" ");
```

```
document.write(edad);
```

Concatenar dos variables implica juntar su contenido en una única variable o resultado, para ello se puede utilizar el operador +, si bien este operador sirve para sumar valores numéricos, cuando se aplica sobre strings su resultado es una concatenación.

```
1 + 2; // Da como resultado el número 3

"hola" + 1; // Da como resultado el string "hola1"

"hola"+" "+"desafío latam"; // Da como resultado "hola desafío latam"
```

¡Prueba estos ejemplos por tu cuenta en la consola del navegador!

Operaciones matemáticas con variables

En las variables también podemos realizar operaciones matemáticas entre ellas, por ejemplo:

```
var num1 = 10;
var num2 = 20;

var resultado = num1 + num2;

document.write("La suma de los números" + " " + num1 + " " + " y " +
num2 + " " + "es:" + " " + resultado);
```

En las variables también podemos almacenar operaciones matemáticas, por ejemplo:

```
var multiplica = 5 * 9;

document.write("Resultado de multiplicación: " + multiplica);
```

Las variables son de mucha ayuda y una gran herramienta al momento de programar y manipular datos. Por el contrario, cuando no se requiere que estas variables se modifiquen, se utiliza el recurso de las **“constantes”**.

Interpolación de variables

Con las últimas versiones de JavaScript se agregó una nueva forma de concatenar strings en JavaScript: la interpolación. Esta operación permite concatenar varias variables, con un resultado idéntico al del operador +, pero mucho más cómodo de usar para quien escribe el código, tanto la interpolación como la concatenación de variables se utiliza principalmente, para generar textos de ayuda o información para el usuario.

En la interpolación, todas las variables involucradas se colocan dentro de dos operadores backtick (`).

```
`Esto es un texto interpolado`
```

Para mostrar el valor de una variable dentro de una interpolación y también para diferenciar el nombre de la variable de un texto cualquiera, se utilizan los llamados placeholders, correspondientes a marcadores que indican la ubicación de una variable. Un placeholder comienza con \${ y termina con }. Todo lo que esté dentro de un placeholder será considerado una variable.

```
var nombre = "Alexis";  
  
alert(`¡Hola ${nombre}!`)
```

Dentro de un placeholder también se puede incluir cualquier instrucción JavaScript.

```
alert(`¡Hola ${prompt('Ingresa tu nombre:')}!`)
```

Para finalizar este punto, vamos a modificar el siguiente código escrito en JavaScript pero realizado con concatenación para ahora usar interpolación y notar las diferencias entre ambas opciones, aunque la respuesta sea la exactamente igual.

```
var texto_a = "Saludos";  
var texto_b = "a todos";  
var num1 = 20;  
var num2 = 30;  
  
console.log("La suma de: "+num1+", más: "+num2+", es: "+num1+num2+".");  
console.log(texto_a+" "+texto_b+".");
```

Como se puede apreciar en el código anterior, se muestran dos mensajes utilizando la modalidad de concatenación para poder unir texto con el valor de las variables que deseamos. Ahora, si transformamos ese mismo código a una notación con interpolación, quedaría de la siguiente forma:

```
var texto_a = "Saludos";  
var texto_b = "a todos";  
var num1 = 20;  
var num2 = 30;  
  
console.log(`La suma de: ${num1}, mas: ${num2}, es: ${num1+num2}.`);  
console.log(`${texto_a} ${texto_b}.`);
```

Al ejecutar ambos programas, obtendremos el mismo resultado, el cual sería:

```
La suma de: 20, mas: 30, es: 50.  
Saludos a todos.
```

En este caso, lo que se hizo fue modificar la estructura del mensaje para dejar de concatenar mediante sumas y pasar a interpolar mediante los operadores backtrick (`), el símbolo \$ y las llaves {}. Permitiendo ser más práctica y rápida la inserción de variables en conjunto con cadenas de caracteres o texto.

Ejercicio guiado: Resolver un problema usando variables y tipos de datos

Una vez manejado el uso de variables, es posible utilizarlas para resolver diferentes tipos de problemas. Se solicita codificar lo siguiente usando JavaScript:

1. Pide al usuario que ingrese su nombre.
 2. Pide al usuario que ingrese un número.
 3. Pide al usuario que ingrese un segundo número.
 4. Muestra la suma, la resta, la división y la multiplicación entre los dos números ingresados.
 5. Debe ser un único mensaje por cada operación (4 en total).
 6. El mensaje debe seguir el siguiente formato: {Nombre del usuario}, el resultado de {nombre de la operación} {primer número} y {segundo número} es {resultado de la operación}. Por ejemplo: "Alexis, el resultado de sumar 1 y 2 es 3".
 7. Para mostrar el mensaje pueden implementar document.write();
- **Paso 1:** Crear una carpeta en nuestro lugar de trabajo y crear dos archivos dentro de ella, como lo es index.html y script.js.
 - **Paso 2:** En el archivo index.html, crear la estructura básica de un documento HTML.

- **Paso 3:** Ahora dentro del script.js, se procede a solicitar los datos al usuario mediante el método "prompt()" y los almacenamos cada uno de ellos en una variable distinta:

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

- **Paso 4:** Realizar los procedimientos matemáticos solicitados y almacenarlos en una nueva variable.

```
var nombre = prompt("Ingresa tu nombre");  
var num1 = prompt("Ingresa el primer número");  
var num2 = prompt("Ingresa el segundo número");  
  
var suma = parseInt(num1)+parseInt(num2);  
var resta = parseInt(num1)-parseInt(num2);  
var multiplica = parseInt(num1)*parseInt(num2);  
var division = parseInt(num1)/parseInt(num2);
```

- **Paso 5:** Dentro de una misma estructura de mensaje con el `document.write()` mostrar el resultado final para cada operación siguiendo la estructura indicada en el enunciado, quedando el ejercicio final de la siguiente manera:

```
var nombre = prompt("Ingresa tu nombre");  
var num1 = prompt("Ingresa el primer número");  
var num2 = prompt("Ingresa el segundo número");  
  
var suma = parseInt(num1)+parseInt(num2);  
var resta = parseInt(num1)-parseInt(num2);  
var multiplica = parseInt(num1)*parseInt(num2);  
var divide = parseInt(num1)/parseInt(num2);  
  
document.write(nombre+" el resultado de sumar "+num1+" + "+num2+" es  
"+suma+". ");  
document.write(nombre+" el resultado de restar "+num1+" - "+num2+" es  
"+resta+". ");  
document.write(nombre+" el resultado de multiplicar "+num1+" * "+num2+"  
es "+multiplica+". ");  
document.write(nombre+" el resultado de dividir "+num1+" / "+num2+" es  
"+divide+". ");
```

Por lo tanto, el resultado final en el navegador web sería:

Este es un documento HTML con JavaScript

Juan el resultado de sumar $3 + 6$ es 9. Juan el resultado de restar $3 - 6$ es -3. Juan el resultado de multiplicar $3 * 6$ es 18. Juan el resultado de dividir $3 / 6$ es 0.5.

Imagen 41. Resultado de la ejecución del código.

Fuente: Desafío Latam

Ejercicio propuesto (13)

Para seguir practicando lo aprendido, plantea una posible solución al siguiente problema implementando todo lo aprendido hasta el momento, por lo tanto, debes desarrollar el diagrama de flujo, el pseudocódigo y por último llevarlo a JavaScript:

- Realizar un programa con JavaScript que solicite al usuario el primer nombre, el primer apellido, la edad y el número de identificación único, para mostrar en un solo mensaje mediante la instrucción `document.write()` y con interpolación de variables lo siguiente: Hola {nombre}, tu edad es {edad} y tu número de identificación es {id}. Ejemplo: "Hola Juan, tu edad es 34 y tu número de identificación es 12345678-9"

Resumen

JavaScript es un lenguaje que nos permite añadir dinamismo e interactividad a una página web. A lo largo de esta lectura hemos ido conociendo herramientas para resolver problemas de lógica, formas de diagramar flujos y de escribir código de alto nivel, como lo es el pseudocódigo.

Además, tuvimos un primer acercamiento con la sintaxis y los principales elementos de JavaScript, siendo capaces de resolver operaciones matemáticas a través de este lenguaje.

Este es sólo el comienzo, ya que cuanto más desarrollemos nuestro pensamiento lógico, más rápido obtendremos soluciones a nuestros problemas cotidianos en programación y nuestros programas harán más a menudo lo que esperamos que realicen con un menor esfuerzo. El desarrollo del pensamiento lógico y nuestra capacidad de abstracción son habilidades que se irán desarrollando con práctica y constancia.

Solución de los ejercicios propuestos

1. Realizar un diagrama de flujo que permita determinar el área de un triángulo rectángulo si el usuario ingresa los valores como la base y la altura del triángulo. Fórmula: $\text{área} = (\text{base} * \text{altura}) / 2$.

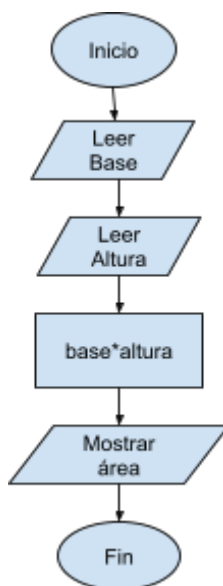


Imagen 42. Solución diagrama de flujo triángulo
Fuente: Desafío Latam

2. Desarrollar un diagrama de flujo que permita calcular el área de una circunferencia partiendo de la fórmula: $\text{área} = \pi * \text{radio}^2$.

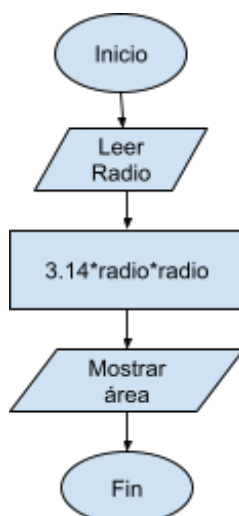


Imagen 43. Solución diagrama de flujo circunferencia
Fuente: Desafío Latam

3. Diseñar un diagrama de flujo para calcular el área de un rombo, en donde el área es se calcula mediante la fórmula: $\text{área} = (\text{Diagonal mayor} * \text{diagonal menor})/2$

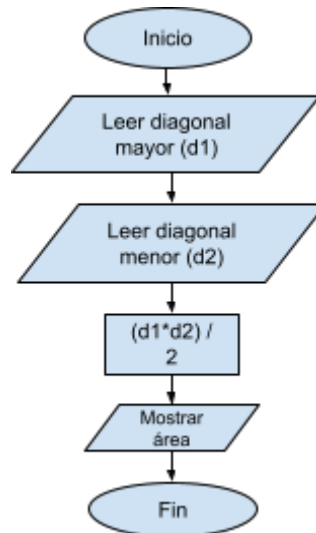


Imagen 44. Solución diagrama de flujo rombo
Fuente: Desafío Latam

4. Realizar un algoritmo en pseudocódigo que permita determinar el área de un triángulo rectángulo si el usuario ingresa los valores como la base y la altura del triángulo. Fórmula: $\text{área} = (\text{base} * \text{altura}) / 2$

```
Inicio
  Leer base
  Leer altura
  (base * altura) / 2
  Mostrar (base * altura) / 2
Fin
```

5. Desarrollar un pseudocódigo que permita calcular el área de una circunferencia partiendo de la fórmula: $\text{área} = \pi * \text{radio}^2$. (La constante pi tiene un valor aproximado de 3,14).

```
Inicio
  Leer radio
  3.14 * radio * radio
  Mostrar 3.14 * radio * radio
Fin
```

6. Diseñar un pseudocódigo para calcular el área de un rombo, en donde el área es se calcula mediante la fórmula: $\text{área} = (\text{Diagonal mayor} * \text{diagonal menor})/2$

```
Inicio
  Leer diagonal mayor
  Leer diagonal menor
  (diagonal mayor * diagonal menor) / 2
  Mostrar (diagonal mayor * diagonal menor) / 2
Fin
```

7. En base al siguiente diagrama de flujo, construye el código en JavaScript que muestra el promedio de dos números:

```
var n1 = prompt("Ingrese un primer número entero: ");
var n2 = prompt("Ingrese un segundo número entero: ");
alert((n1+n2)/2);
```

8. En base al siguiente pseudocódigo, construye el código en JavaScript que muestra la conversión de pulgadas a metros, de acuerdo al siguiente pseudocódigo:

```
var pulgada = prompt("Ingrese un pulgadas: ");
alert(pulgada*0.254);
```

9. Realizar un programa con JavaScript que al ingresar dos valores (valor1 = 20 y valor2 = 7) calcule la suma, la resta, la multiplicación y la división de esos dos números en la consola del navegador web de tu preferencia, posteriormente muestra en una ventana emergente (alerta) un mensaje que indique el resultado de la multiplicación.

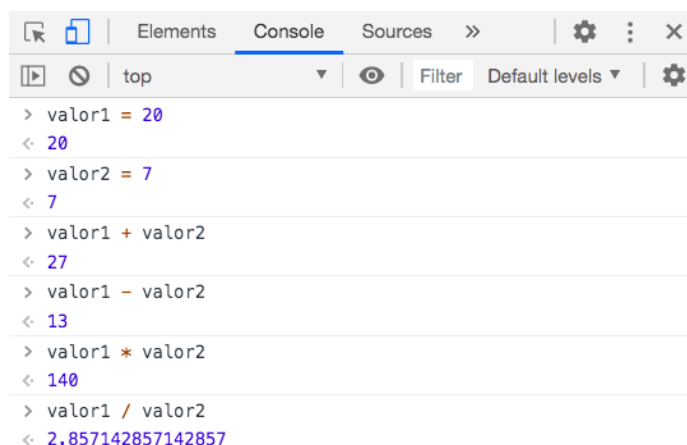


Imagen 45. Solución cálculos con JavaScript
Fuente: Desafío Latam



Imagen 46. Solución cálculos con JavaScript
Fuente: Desafío Latam

10. Realizar un programa que permita determinar el área de un triángulo rectángulo si el usuario ingresa los valores como la base y la altura del triángulo. Fórmula: $\text{área} = (\text{base} \times \text{altura}) / 2$. Mostrar el resultado final en una alert.

```
var base = prompt("Ingrese la base del triángulo");
var altura = prompt("Ingrese la altura del triángulo");

document.write("El área del triángulo es: ");
document.write((parseInt(base)*parseInt(altura))/2);
```

11. Desarrollar un programa en JavaScript que permita calcular el área de una circunferencia, partiendo de la fórmula: $\text{área} = \pi \times \text{radio}^2$. (La constante pi tiene un valor aproximado de 3,1416 y debe ser declarado en una variable del tipo constante).

```
const PI = 3.1416;
var radio = prompt("Ingrese el radio de la circunferencia");

document.write("El área de la circunferencia es: ");
document.write((PI*parseInt(radio)*parseInt(radio)));
```

12. Diseñar un programa para calcular el área de un rombo, en donde el área es se calcula mediante la fórmula: $\text{área} = (\text{Diagonal mayor} \times \text{diagonal menor}) / 2$

```
var diagoMayor = prompt("Ingrese el valor de la diagonal mayor");
var diagoMenor = prompt("Ingrese el valor de la diagonal menor");

document.write("El área del rombo es: ");
document.write((parseInt(diagoMayor)*parseInt(diagoMenor))/2);
```

13. Realizar un programa con JavaScript que solicite al usuario el primer nombre, el primer apellido, la edad y el número de identificación único, para mostrar en un solo mensaje mediante la instrucción `document.write()` y con interpolación de variables lo siguiente: `Hola {nombre}, tu edad es {edad} y tu número de identificación es {id}`. Ejemplo: "Hola Juan, tu edad es 34 y tu número de identificación es 12345678-9"

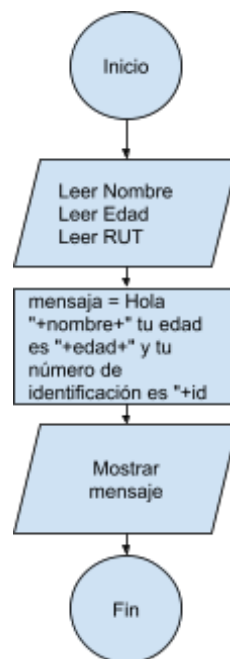


Imagen 47. Solución diagrama de flujo datos del usuario
Fuente: Desafío Latam

Código en Pseudocódigo

```
Inicio
  Leer nombre
  Leer edad
  Leer RUT
  mensaje = "Hola "+nombre+" tu edad es "+edad+" y tu número de
  identificación es "+RUT
  Mostrar mensaje
Fin
```

Código en JavaScript.

```
var nombre = prompt("Ingresa tu nombre");
var edad = prompt("Ingresa tu edad");
var id = prompt("Ingresa tu número de identificación");
var mensaje = "Hola "+nombre+" tu edad es "+edad+" y tu número de
  identificación es "+RUT
document.write(mensaje);
```