ALURA LATAM

ESTUDIO DE PROGRAMACION DE LO QUE VOY PROGRESANDO

ALURA LATAM

NODE.JS- JavaScript

JavaScript tipos, variables y funciones

24/07/2024

CONTENIDO

# Preparando el ambiente

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84237/next)

JavaScript es conocido como el “lenguaje de la web”. Aunque fue desarrollado en el front-end, hoy en día usamos JavaScript cada vez más en el llamado back-end, o server side (lado del servidor) de las aplicaciones web. Este curso se enfoca en el uso de JavaScript para el backend.

Así, necesitaremos instalar dos herramientas: NodeJS, necesaria para ejecutar código JavaScript sin necesidad del navegador, y Visual Studio Code o VSC, el editor que usamos para escribir código.

## NodoJS

Si ya tomó un curso anterior de JavaScript de back-end, es posible que NodeJS ya esté instalado. Si no estás seguro, puedes consultar con los siguientes pasos:

1) Abra una ventana de terminal en su computadora. Esto puede hacerse de la siguiente manera:

**Windows:** puede acceder a la búsqueda desde el menú de inicio y buscar Símbolo del sistema, o acceder a ella a través de Menú Inicio > Sistema de Windows > Símbolo del sistema.

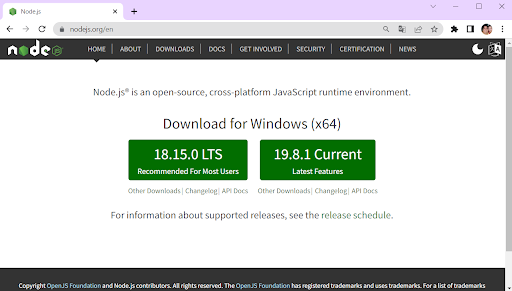
**MacOs:** Terminal está disponible en el menú Aplicaciones, dentro de la carpeta Utilidades.

**Linux (Ubuntu):** La Terminal está disponible en el menú Programas/Aplicaciones. Si no, puede estar dentro de la carpeta Utilidades.

2) En la terminal, escriba node --version o node -v; si devuelve un número de versión, como v16.15.0, NodeJS ya está instalado.

Si necesita instalarlo, siga las instrucciones para cada sistema operativo que se encuentran en la página de inicio de NodeJS. Este curso fue desarrollado utilizando la versión 16.15.0.

El sitio web de NodeJS ofrece dos opciones de descarga, LTS y "actual". Puede elegir la versión LTS (Soporte a largo plazo) y hacer clic en el botón correspondiente para descargar e instalar normalmente como cualquier otro programa.



## Visual Studio Code

Visual Studio Code es el editor que elegimos usar para escribir nuestro código durante este curso. Los enlaces para descargar e instalar según su sistema operativo se encuentran en la página de inicio de VSC.

Hay varios otros editores como Atom, Sublime y Notepad++. Si ya tienes uno de estos instalado o prefieres otro, ¡no hay problema! Solo tenga en cuenta que algunas funciones y herramientas varían de un editor a otro y es posible que no estén disponibles o funcionen de manera diferente.

¡Ya podemos empezar!

# Para saber más: más sobre números

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84239/next)

Los programas funcionan gestionando valores, como el número 3.14 o el texto Juliana y Leonardo. Los tipos de valores que se pueden representar y manipular en un lenguaje de programación se conocen como tipos, y una de las características más fundamentales de un lenguaje de programación es el conjunto de tipos que soporta.

**Variables**

Cuando un programa necesita mantener un valor para uso futuro, asigna el valor a una variable o "almacena" los datos en una variable. Las variables tienen nombres y nos permiten usar esos nombres en nuestros programas para referirnos a valores. La forma en que funcionan las variables es otra característica fundamental de cualquier lenguaje de programación.

Los tipos de datos de JavaScript se pueden dividir en dos categorías: tipos primitivos y tipos de objetos o estructurados. Los tipos primitivos de JavaScript incluyen números, palabras o texto (conocidos como cadenas) y valores booleanos (conocidos como booleanos).

**Tipos numéricos**

Como vimos en clase, podemos almacenar números de diferentes maneras:

**const** edadLeonardo = 39;

**const** pi = 3.14;COPIA EL CÓDIGO

Consejo: En Javascript puedes obtener exacto de pi usando la propiedad Math.PI

El punto flotante puede tener un punto decimal; utilizan la sintaxis tradicional para números reales. Un valor real se representa como la parte entera del número, seguida de un punto decimal y la parte fraccionaria del número.

Los puntos flotantes también se pueden representar mediante notación exponencial: un número real seguido de la letra e (o E), seguido de un signo opcional más (+) o menos (-) y un exponente entero. Esta notación representa el número real multiplicado por 10 elevado a la potencia del exponente.

La división por cero no es un error de JavaScript: simplemente devuelve "Infinito". Sin embargo, hay una excepción: cero dividido por cero no tiene un valor definido y el resultado de esta operación es el valor especial no numérico NaN.

**var** a = 10

**var** b = 0

console.**log**(a/b) // InfinityCOPIA EL CÓDIGO

**var** a = 0

**var** b = 0

console.**log**(a/b) // NaN

# Para saber más: codificación de strings

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84240/next)

Acabamos de ver que usamos el tipo de cadena cada vez que queremos trabajar con datos de texto. Pero si nos detenemos a pensarlo, varios idiomas utilizan distintos caracteres, como acentos e ideogramas. ¿Cómo manejan esto los lenguajes de programación? ¿Y los emojis? ¿Alguna vez visitó un sitio web y notó que los caracteres de los textos no parecían correctos, que en lugar de algunos de ellos aparecían signos de interrogación, cuadrados o guiones?

Todo esto tiene que ver con la codificación de caracteres. En las últimas décadas se han desarrollado varios conjuntos de caracteres especiales, cada uno con sus propios códigos, para que las personas que escriben y leen en idiomas distintos al inglés puedan utilizar computadoras con sus propios idiomas. Y, ¿cómo funciona?

Para que la computadora pueda descifrar un carácter especial, es necesario utilizar un sistema específico que básicamente tiene un código para cada carácter, y que la computadora puede acceder a él para hacer la conversión, una idea similar a la de la criptografía.

Se han desarrollado varios conjuntos de caracteres, desde los específicos del idioma como occidental, latinoamericano, japonés, etc., hasta ASCII (American Standard Code for Information Interchange). y a partir de 2007 se adoptó el formato Unicode. El estándar UTF (Unicode Transformation Format) se utiliza como estándar en la web hasta el día de hoy.

Unicode tiene códigos específicos para "cifrar" y "descifrar" caracteres de más de 150 idiomas antiguos y modernos, así como varios otros conjuntos de caracteres, como símbolos matemáticos e incluso emojis. Wikipedia tiene una lista extensa de todas las tablas con códigos y caracteres Unicode, como las siguientes:

Descripción oficial **del** personaje UTF-16

$ U+0024 SIGNO DE DÓLAR

A U+0041 LETRA A MAYÚSCULA LATINA

✅ U+2705 MARCA DE VERIFICACIÓN

ぁ U+3041 HIRAGANA LETRA A PEQUEÑACOPIA EL CÓDIGO

Podemos probar la transformación/conversión del código Unicode en caracteres usando console.log(). Haz la prueba:

**const** simboloPeso = '\u0024'

**const** aMayusculo = '\u0041'

**const** check = '\u2705'

**const** hiragana = '\u3041'

console.log(simboloPeso)

console.log(aMayusculo)

console.log(check)

console.log(hiragana)COPIA EL CÓDIGO

Los caracteres \u al comienzo del código son caracteres de escape que usamos para señalar a JavaScript que estamos hablando de códigos Unicode y no de las cadenas de texto habituales.

JavaScript usa UTF-16 por defecto. El número 16 se relaciona con los espacios de bits ocupados por cada carácter, 16 en este caso. No profundizaremos en la relación entre los tipos de datos y el espacio de memoria ocupado por cada tipo, puedes investigar más sobre el tema, así como sobre los caracteres de escape, pero por ahora es genial ver cómo funciona Unicode en la práctica.

Las bases de datos pueden aceptar otros tipos de codificación de caracteres, lo que tiene sentido teniendo en cuenta que UTF-16 utiliza una cantidad de espacio de memoria relativamente grande para guardar cada carácter. 16 bits suena pequeño, ¡pero a veces los bancos necesitan guardar grandes cantidades de datos! Sin embargo, con las tecnologías de almacenamiento y tráfico de datos que tenemos hoy, esto ya no es una preocupación tan grande, excepto en casos muy específicos. Ya no es muy común utilizar una codificación diferente a UTF en las bases de datos incluso en el caso de grandes volúmenes de datos, pero siempre dependerá del caso.

Detalles y documentación más precisos sobre Unicode en la página de la [Fundación Unicode](https://home.unicode.org/).

<https://home.unicode.org/>

# Para saber más: trabajando con strings

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84241/next)

JavaScript trae en su biblioteca base varios métodos que utilizamos para manipular cadenas de texto: cambiar de mayúsculas a minúsculas, contar cuántas letras hay en una palabra, eliminar espacios, unir dos cadenas, etc.

Pensemos en algunos ejemplos prácticos para hacer este tipo de cambio. Por ejemplo, para estandarizar una comparación entre cadenas:

**const** ciudad = "buenos aires";

**const** input = "Buenos Aires";

console.log(cidade === input); // falseCOPIA EL CÓDIGO

Nosotros, como personas, pudimos percibir el valor de la ciudad y las variables de entrada como de la misma ciudad, Buenos Aires. Pero para JavaScript, ambos datos son solo cadenas de caracteres, y la comparación fallará, porque como hemos visto, JavaScript distingue entre mayúsculas y minúsculas, tanto en los valores de los datos como en el código que escribimos.

Una de las formas de lidiar con esto es establecer por defecto todas las entradas en el formato de texto que se comparará antes de realizar la comparación. En este caso, pasando todos los caracteres a minúsculas.

**const** ciudad = "buenos aires";

**const** input = "Buenos Aires";

**const** inputMinusculo = input.toLowerCase();

console.log(ciudad === inputMinusculo); // trueCOPIA EL CÓDIGO

Arriba, vemos uno de los métodos de cadena nativos de JavaScript en acción, toLowerCase() que convierte todos los caracteres de la cadena informada (en este caso, input) a letras minúsculas (si son números, no se convierte nada). Puedes consultar más sobre este método en la [MDN](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/toLowerCase).

Otro ejemplo: cualquier entrada de texto que requiera una cantidad mínima de caracteres, como una contraseña o un nombre. La propiedad de longitud se puede utilizar para averiguar cuántos caracteres contiene una cadena:

**const** clave = "miClave123"

console.log(clave.length) // 10 caracteresCOPIA EL CÓDIGO

La propiedad de length se usa mucho en el desarrollo web diario. Puedes saber más sobre ella [aquí](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String/length).

¿Observaste que la longitud no tiene paréntesis al final de la palabra? Hay una diferencia entre los métodos y las propiedades que no cubriremos durante este curso, ¡pero dejaremos el consejo aquí en caso de que tengas curiosidad! ;)

Puedes consultar la lista completa de [métodos de cadenas de MDN](https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/String#m%C3%A9todos) (hay varios), con la descripción de cada uno, y practicar con los ejemplos.

# Cargando datos iniciales

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84263/next)

En JavaScript, como en la mayoría de los lenguajes de programación, es posible almacenar datos en la memoria de la computadora a través de variables.

Estos datos pueden ser palabras, números o valores lógicos como “true” o “false” (llamados tipos booleanos), como se ilustra en la siguiente tabla:

| **nombre de variable** | **valor** |
| --- | --- |
| edad | 2 |
| nombre | "Lara" |
| apellido | "Ayumi" |
| graduado | false |
| diezAnosDespues | "12" |

Sabiendo esto, analice las siguientes afirmaciones y seleccione la verdadera.:

Principio del formulario

* Alternativa correta



Al ejecutar el código const nombreCompleto = nombre + apellido, no se mostraría ningún error y el valor almacenado en la variable sería "LaraAyumi".

Usamos el signo más (+) para unir 2 palabras. Para acercar esta salida al mundo real, podemos agregar una cadena vacía entre las variables, como se muestra en el siguiente código const nombreCompleto = nombre + " " +apellido.

# Para saber más: patrones de nombres de JavaScript

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84242/next)

Un detalle muy importante, del que a veces no nos damos cuenta cuando empezamos a programar, es que cada lenguaje tiene sus propios estándares. Sirven no solo para escribir código que funcione, sino también para crear nombres de variables, estructurar un programa y mucho más.

Lo primero que debemos tener en cuenta es que JavaScript distingue entre mayúsculas y minúsculas, es decir, distingue entre mayúsculas y minúsculas. Esto significa que todo lo que escribimos, ya sean instrucciones específicas del idioma (como console.log) o al nombrar una variable, debe hacerse con el mismo patrón, que incluye las mayúsculas.

Para ilustrar, JavaScript trata los cuatro ejemplos a continuación como variables diferentes y no arrojará ningún error si ejecuta el programa:

**const** miVar = 1;

**const** MiVar = "texto";

**const** mivar = "3";

**const** MIVAR = 2;

console.log(miVar, MiVar, mivar, MIVAR);COPIA EL CÓDIGO

Podemos ver que, en un programa muy grande, la posibilidad de problemas es grande. Entonces, ¿cómo sabemos la forma correcta de nombrarlo? Ahí es donde entra lo que llamamos convenciones, para estandarizar estos aspectos del código.

Hay varias convenciones de nomenclatura, y cada lenguaje de programación tiene una. Estos son algunos de ellos:

**camelCase:** Comienza con una letra minúscula y la primera letra de cada palabra se escribe en mayúscula.

* **Por ejemplo:** miVariable o claveUsuario. Esta es la convención utilizada por JavaScript para variables y funciones.

**snake\_case:** los espacios se reemplazan por el carácter \_ (guion bajo), con todas las palabras en minúsculas.

* **Por ejemplo:** mi\_variable o clave\_usuario.

**kebab-case:** Similar al anterior, pero con espacios reemplazados por guiones.

* Por ejemplo: mi-variable o clave-usuario. Esta convención no se puede utilizar en JavaScript para variables y funciones.

**PascalCase:** Similar a CamelCase, pero en este caso todas las palabras comienzan con una letra mayúscula.

* Por ejemplo: MiVariable o ClaveUsuario.

**Importante:** Nunca use espacios o caracteres especiales, ni comience nombres de variables con números.

Cuando hablamos de convención, hablamos de buenas prácticas y estandarización. Si usa alguno de los patrones anteriores para nombrar variables con JavaScript (con la excepción del patrón kebab-case), su código seguirá funcionando, pero seguir las convenciones es parte del desarrollo de un código legible y bien escrito.

Este es un tema amplio con muchos detalles, y es parte de nuestro trabajo diario como desarrolladores para garantizar que se sigan las llamadas guías de estilo definidas para un producto de código.

Puede aprender los detalles poco a poco a medida que estudia y ver cómo se aplican a los códigos que ve a su alrededor.

Final del formulario

# Para saber más: tipos null y undefined

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84243/next)

En esta clase hablamos de tres tipos primitivos: número, cadena y booleano. Pero hay dos tipos más que no hemos cubierto en profundidad: null e undefined.

null es un tipo especial, ya que se puede traducir como “ausencia de valor” y se puede asignar como el valor de una variable:

**let** input = null;

**if** (input === null) {

console.**log**('No hay información);

} **else** {

console.**log**(input);

}COPIA EL CÓDIGO

Si es así, ¿cuál sería la diferencia entre los dos casos a continuación?

**let** input = **null**;

**let** input2;

console.**log**(input); // null

console.**log**(input2); // undefinedCOPIA EL CÓDIGO

Aquí es donde entra el tipo undefined. Este tipo también representa “ausencia de valor”, pero de otra forma: normalmente, mientras nulo es un valor asignado a una variable que existe y ha sido inicializada, undefined se refiere al valor de una variable que no ha sido inicializada (es decir, no se le ha asignado ningún valor).

undefined es también el valor devuelto por una función que no tiene cláusula de retorno. Veremos más sobre las funciones y volveremos más adelante en el curso.

Es importante tener en cuenta que aunque ambos tipos se utilizan para indicar la ausencia de un valor, los operadores de comparación de JavaScript pueden o no diferenciarlos:

console.**log**(**null** == **undefined**); // true

console.**log**(**null** === **undefined**); // falseCOPIA EL CÓDIGO

En la vida cotidiana es común considerar undefined como una ausencia de valor “inesperada” (provocada por un bug o error en el código) y null como un tipo de dato que también significa ausencia de valor, pero no inesperadamente. Por ejemplo, un campo en una tabla de base de datos que no tiene datos o información solicitada que no es obligatoria y no ha sido completada por el usuario puede tener un valor null.

# Haga lo que hicimos: comportamiento de los tipos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84258/next)

En esta lección, comenzamos a aprender sobre tipos, un concepto inicial y uno de los más importantes en cualquier lenguaje de programación. Estudiamos los tipos primitivos número, cadena y booleano con mayor detalle, pero tenemos que practicar para interiorizar.

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

En el caso de los números, es posible guardar en una variable no sólo el número en sí, como en const num = 5; sino también el resultado de una operación (o más):

**const** suma = 10 + 10;

**const** multiplicacion = 10 \* 10;

**const** operacion = (suma + multiplicacion) + 10;COPIA EL CÓDIGO

Tenga en cuenta que, en el código anterior, usamos paréntesis ( ) para seguir la regla matemática de precedencia en las operaciones.

¿Puedes pensar en más casos con operaciones matemáticas básicas para probar?

# Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84253/next)

Lo que aprendimos en esta aula:

* Que la computadora solo puede almacenar valores en algunos tipos de variables, como texto y números booleanos.
* Cómo declarar una variable numérica, de tipo número, y asociarle un valor, que puede ser un valor entero o de coma flotante.
* Cómo declarar una variable de texto, conocida como cadena, y poner una frase, número, puntuación o una combinación de todos estos dentro de ella.
* Cómo declarar una variable booleana y establecer su valor en verdadero (verdadero) o falso (falso).
* Dónde aparecen más los booleanos, como resultado de comparaciones con === u otros operadores de comparación como >=.

# Variables y la respuesta de todo

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84264/next)

Una persona creó 3 variables diferentes para almacenar datos numéricos importantes. Para la primera variable usó la sentencia var, le dio el nombre respuestaDeTodo y le asignó el valor 42. Para la segunda variable usó la sentencia let, le dio el nombre edad y le asignó el valor 29. Para la tercera variable usó const, lo nombró pi y le asignó el valor 3.14, como se muestra en el siguiente código:

**var** respuestaDeTodo = 42

**let** edad = 29

**const** pi = 3.14COPIA EL CÓDIGO

Para probar su conocimiento de JavaScript, creó un bloque intercambiando los valores de las variables entre ellos e incluyó un archivo console.log dentro y otro fuera del bloque:

{

**var** respuestaDeTodo = 3.14

**let** edad = 42

**const** pi = 29

console.**log**(respuestaDeTodo, edad, pi)

}

console.**log**(respuestaDeTodo, edad, pi)COPIA EL CÓDIGO

Sabiendo esto, revise las declaraciones a continuación y marque la que contiene el código con la salida de consola correcta.

Principio del formulario

* Alternativa correta



3.14 42 29 e 3.14 29 3.14.

¡Muy bien! Las variables declaradas con var o let se pueden reasignar. Sin embargo, una variable declarada con let solo puede usarse en el mismo ámbito en el que está definida. En el caso de la declaración const, significa que la variable siempre tendrá una referencia al mismo objeto o valor primitivo, porque esta referencia no se puede cambiar. La referencia en sí es inmutable, pero el valor que tiene la variable no se vuelve inmutable.

Final del formulario

# Para saber más: conversión o casting de valores

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84244/next)

Ya hemos aprendido los tipos de variables que podemos utilizar, entre las que tenemos números, textos y booleanos. Sin embargo, ¿cómo podemos usar el valor de una variable numérica en un texto, o usar una cadena que contiene solo números para hacer operaciones matemáticas?

Para estos casos, en JavaScript usaremos Number() y String(), que nos permitirán convertir los datos a números o textos.

**String()**

Hagamos algunos ejemplos de conversión de números y booleanos a través de String():

**let** telefono = 12341234;

console.**log**("El teléfono es " + **String**(telefono)); // haremos la conversión del número 12341234 para una cadena de caracteres o string “12341234” y así podemos hacer una concatenación entre stringsCOPIA EL CÓDIGO

Otra opción para transformar un valor en un String es usar toString():

**let** telefono = 12341234;

console.**log**("El teléfono es " +telefono.**toString**()); // toString() es otra forma de realizar esta conversión, está es forma es posible verla también en otros lenguajes de programaciónCOPIA EL CÓDIGO

**let** usuarioConectado = **false**;

console.**log**(**String**(usuarioConectado)); // esto realizará un conversión del tipo de dato booleano para string, va a mostrar un string igual a “false”.

usuarioConectado = **true**;

console.**log**(**String**(usuarioConectado)); // ahora tendremos un string igual a “true”.COPIA EL CÓDIGO

**Number()**

Hagamos algunos ejemplos de conversión de textos y booleanos a través de Number():

// Vamos calcular a área de un rectángulo

**let** ancho = "10";

**let** alto = "5";

console.**log**(**Number**(ancho) \* **Number**(alto)); // De esta forma realizaremos una conversión de String para números, luego es realizada la multiplicaciónCOPIA EL CÓDIGO

Podemos usar el operador más + para convertir textos a números, colocándolos antes de las variables:

**let** ancho = "10";

**let** alto = "5";

console.**log**( + ancho \* + alto); // obtendremos una conversión de String para números usando el signo + antes de las variables.COPIA EL CÓDIGO

**let** miNombre = "Leonardo";

console.**log**(**Number**(miNombre)); // como la variable miNombre no es solo números, en realidad son solo letras, entonces retorna el error NaN (Not a Number, no es número);

console.**log**( + miNombre); // la conversión también devolverá NaNCOPIA EL CÓDIGO

**let** usuarioConectado = **false**;

console.**log**(**Number**(usuarioConectado)); // obtendremos una conversión de dato booleano para número, en ese caso false (falso) retorna el número 0.

usuarioConectado = **true**;

console.**log**(**Number**(usuarioConectado)); // ahora obtendremos la conversión del valor booleano para valor numerico, en ese caso true (verdadero) retorna el número 1.COPIA EL CÓDIGO

Sugerencia de mejores prácticas: aunque JavaScript realiza la mayoría de las conversiones correctamente, pueden surgir problemas, por lo que siempre es una buena idea realizar conversiones explícitamente. No es común usar el operador de suma para convertir a números, pero este uso es posible. Las conversiones booleanas no se usan con frecuencia, pero son posibles.

# Para saber más: palabras reservadas

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84245/next)

Ya aprendimos cómo declarar variables, ya sea let o const, usando la palabra-clave y el nombre que elegimos para la variable. A este nombre lo llamamos precisamente el identificador, e idealmente siempre deben ser lo más explicativos posibles:

let dniUsuario = "8001879";COPIA EL CÓDIGO

Pero, ¿qué sucede si tratamos de identificar una variable con un término que forma parte del lenguaje, como en los casos a continuación?

**let** **var** = 0;

**let** **if** = 0;

**let** **const** = "Alura Latam";COPIA EL CÓDIGO

Haga la prueba para ver que JavaScript no puede reconocer estas palabras-clave como identificadores ni interpretar lo que debe ejecutarse en estas líneas. Esto se debe a que var, if y const son palabras reservadas de JavaScript. Es decir, no podemos usarlos para nombrar (identificar) variables, funciones u otros bloques de código que necesiten identificadores.

Por otro lado, los siguientes ejemplos son aceptables:

let varInicial = 0;

let ifFalse = 0;

let constDeTexto = "Alura Latam";COPIA EL CÓDIGO

En JavaScript, algunas palabras están completamente reservadas (no se pueden usar como identificador en ningún caso), mientras que otras se pueden usar según el contexto, y otras no se pueden considerar completamente reservadas por razones de compatibilidad con versiones anteriores del lenguaje, como es el caso de let (teniendo en cuenta que, hasta ES6, solo era posible declarar variables con var). La palabra let proviene del verbo inglés “allow”, por lo que no es posible dejarlo reservado ya que puede que ya se use como identificador en algún código antiguo.

La mejor práctica, en este caso, es no utilizar ninguno de los términos de la lista a continuación como identificadores, ya sea para variables, funciones, clases o cualquier otro bloque que necesite un nombre. Las únicas excepciones son from, set y target, que son seguras y de uso común.

arguments

**as**

**async**

**await**

**break**

**case**

**catch**

**class**

**const**

**continue**

**debugger**

**default**

**delete**

**do**

**else**

eval

**export**

**extends**

**false**

**finally**

**for**

**from**

**function**

get

**if**

**import**

**in**

**instanceof**

**let**

**of**

**new**

**null**

**return**

set

**static**

super

**switch**

target

this

**throw**

**true**

**try**

**typeof**

**var**

**void**

**while**

**with**

**yield**COPIA EL CÓDIGO

Como los lenguajes están en constante desarrollo, JavaScript también restringe el uso de algunas palabras más que pueden usarse en futuras versiones:

**enum**

implements

**interface**

**package**

**private**

**protected**

**public**COPIA EL CÓDIGO

Consejo de buenas prácticas: trate siempre de nombrar/identificar su código de la forma más semántica posible, pensando en qué datos se guardan en la variable y para qué se usarán. Además de evitar palabras reservadas, hace que el código sea más comprensible y se lea con mayor fluidez.

# Haga lo que hicimos: ¿Variables constantes?

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84259/next)

Puede parecer extraño cuando usamos la palabra variable pero al mismo tiempo JavaScript moderno tiene una opción, llamada const, que precisamente no puede ser variable, no podemos cambiar su valor.

Es decir, la declaración const num = 1; no puede tomar ningún otro valor, y aparentemente este es un comportamiento esperado así como una buena práctica. Después de todo, ¿cuál es el ideal? ¿Cambiar el valor de una variable o no cambiarla nunca? ¿Cómo saber? Comprender cuándo reasignar (es decir, cambiar/reemplazar) el valor de una variable viene con la práctica.

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Al declarar una variable como const, esta debe recibir un valor: const valorComision = 0.25;

Al declarar let, por el contrario, podemos simplemente declarar la variable, sin asignar ningún valor. Además, el valor inicial se puede modificar más adelante:

**let** miLet;

console.**log**(miLet); //undefined

// asignando un valor con Tipo de dato String

miLet = "Soy un texto";

console.**log**(miLet); // "Soy un texto"

// reasignando, esta vez con tipo de datos númerico

miLet = 100;

console.**log**(miLet); //100COPIA EL CÓDIGO

Particularmente no soy partidario de hacer estos cambios en los tipos de datos, pero sintácticamente son válidos.

¡Sigue practicando con otros ejemplos!

# Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84254/next)

Lo que aprendimos en esta aula:

* var tiene un alcance global, por lo que funciona en cualquier parte del código, lo que puede generar problemas potenciales con la lógica de la aplicación.
* let y const tienen un ámbito local, por lo que no se puede acceder a ellos desde otros ámbitos, y const no puede cambiar su valor una vez definido.
* JavaScript considera que algunos valores son falsos durante las comparaciones, como valores null, undefined, 0, NaN y vacías.
* null se usa cuando no queremos dar un valor a una variable, pero tampoco queremos dejarla sin definir.
* null es un objeto en JavaScript, diferente de undefined, que tiene su propio tipo, y esto puede causar problemas si no tenemos cuidado al usarlo.
* JavaScript convierte automáticamente entre tipos de valores, como números a texto y booleanos, pero no es una buena idea dejar que lo haga automáticamente, ya que puede causar errores.
* Podemos forzar un cambio de tipo usando los métodos Number() y String(), donde Number() convierte las variables a números o NaN si hay un error en la conversión, y String() convierte las variables a texto.

# Para saber más: tipos de error

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84246/next)

Mientras trabajamos con la programación, pasamos mucho tiempo lidiando con los errores que aparecen en nuestro código. Esto es completamente normal ya que nos ayudan a resolver problemas. ¿Te imaginas cuánto más difícil sería programar sin un recurso que nos diga dónde nos estamos equivocando y cómo corregirlo?

Como hemos visto, las personas que desarrollan los programas y lenguajes son las responsables del llamado “manejo de errores”, es decir, permitir la comunicación de cuáles fueron los problemas, y esto no siempre ocurre de la mejor manera - aunque ya ha mejorado mucho en las últimas décadas.

Cada lenguaje de programación tiene su propia forma de tratar los errores. JavaScript comienza dividiendo cada tipo de posible error en algunas categorías:

* **RangeError:** cuando el código recibe datos del tipo correcto, pero no dentro del formato aceptable. Por ejemplo, un procesamiento que solo se puede hacer con números enteros mayores o iguales a cero, pero recibe -1.
* **ReferenceError:** generalmente ocurre cuando el código intenta acceder a algo que no existe, como una variable que no se ha definido; a menudo es causado por errores tipográficos o confusión en los nombres utilizados, pero también puede indicar un error en el programa.
* **SyntaxError:** En la mayoría de los casos ocurre cuando hay errores en el programa y JavaScript no puede ejecutarlo. Los errores se pueden escribir incorrectamente o usar métodos o propiedades, por ejemplo, operadores o signos gráficos con elementos faltantes, como olvidarse de cerrar llaves o corchetes.
* **TypeError:** Indica que el código esperaba recibir datos de cierto tipo, como una cadena de texto, pero recibió otro tipo, como un número, booleano o nulo.

NodeJS funciona con otro tipo de errores específicos que no vamos a tratar ahora mismo, pero que siempre puedes consultar en la documentación oficial.

Además del tipo de error, el terminal también proporcionará otra información, como el nombre del archivo y la línea donde se detectó el error. Esto suele ser suficiente para identificarlo y corregirlo, pero también hay casos en los que JavaScript no detecta el error en la línea donde se declara el código, por ejemplo, sino donde se ejecuta. Por eso es importante practicar siempre la lectura de los errores y el stacktrace y nunca omitir este paso.

En el futuro, cuando trabaje en sus aplicaciones, también debería crear sus propios mensajes de error para ayudar a los usuarios.

# Características de JS

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84265/next)

JavaScript fue estandarizado en 1996 por la Asociación Europea de Fabricantes de Computadoras (ECMA), razón por la cual a veces lo escuchas como ECMAScript. Es un lenguaje increíblemente poderoso, presente en la mayoría de los navegadores y en el backend de grandes sistemas.

Sabiendo esto, analiza las alternativas a continuación y marca las verdaderas sobre este lenguaje de programación.

Principio del formulario

* Alternativa incorreta



La división por cero no es un error de JavaScript.

La diferencia de otros lenguajes de programación como Java o Python, la división por cero en JavaScript no genera error. El retorno de esta operación es infinito o infinito negativo (Infinity). Sin embargo, dividir cero por cero da como resultado un NaN (no un número).

* Alternativa incorreta



En JavaScript, los tipos de datos se pueden dividir en dos categorías: tipos primitivos y tipos de objetos.

¡Muy bien! Los tipos primitivos de JavaScript incluyen números, texto (conocidos como cadenas o string) y valores booleanos. El tipo de objeto es una colección de propiedades, cada una de las cuales tiene un nombre y un valor, ya sea un valor primitivo u otro objeto.

Final del formulario

En JavaScript, podemos crear un programa con un paradigma funcional y lógico.

JavaScript es un lenguaje de programación multiparadigma y tiene soporte para los paradigmas funcionales, orientado a objetos o lógico por ejemplo.

# Para saber más: otros métodos de Console API

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84247/next)

Aunque es el más utilizado, .log() es uno de varios métodos que podemos utilizar para mostrar información en la llamada “salida estándar” -la terminal- mientras desarrollamos una aplicación. La palabra log significa algo así como “grabar”, por lo que este método simplemente registra en la terminal lo que le pasamos entre paréntesis, por ejemplo el contenido de una variable o el resultado de una operación.

Entre los otros métodos hay:

* console.error() para mostrar mensajes de error;
* console.table() para ver información tabular de una manera más organizada;
* console.time() y console.timeEnd() para cronometrar la cantidad de tiempo que tarda una operación de código en iniciarse y completarse;
* console.trace() para mostrar el seguimiento de pila de todos los puntos (es decir, archivos llamados) donde pasó el código ejecutado durante la ejecución.

[La documentación oficial de NodeJS](https://nodejs.org/api/console.html) brinda ejemplos sobre cómo usar cada uno de estos métodos y más de la lista. Es una documentación bastante extensa, ¡pero no te preocupes! No tienes que memorizar la lista completa, ya que siempre está disponible como referencia. En la mayoría de los casos de ejemplo, nos quedaremos con console.log().

# Documentación de Node.js v22.5.1

<https://nodejs.org/api/console.html>

# Haga lo que hicimos: usando console.error()

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84260/next)

Si ya probó console.log() usando el navegador, probablemente ya haya visto los métodos console.error() y console.warn() en acción, porque en los navegadores estos métodos se identifican con los colores rojo y amarillo y emojis.

Al trabajar con NodeJS, la “salida estándar” es la terminal y no la consola del navegador, lo que limita un poco el uso de recursos gráficos. Hagamos una prueba con el método console.error().

Cree un archivo .js en su computadora, escriba el siguiente código:

console.**log**("Se generó un error");

console.**error**("Se generó un error");COPIA EL CÓDIGO

Si ejecutamos este código con node script.js (no olvide verificar si está ejecutando el comando dentro de la carpeta/directorio correcto), el resultado es el mismo para ambos comandos:

Se generó un **error**

Se generó un **error**COPIA EL CÓDIGO

Entonces, ¿no hay ninguna diferencia entre usar uno u otro?

Sí, lo hace. Sin embargo, como en cualquier lenguaje de programación, es normal que algunos métodos sólo funcionen de la forma que esperamos si proporcionamos los datos necesarios de forma correcta.

Intentémoslo de nuevo, pasando información ligeramente diferente a console.error():

console.**log**("Se generó un error");

console.**error**(**new** **Error**("Se generó un error"));COPIA EL CÓDIGO

Si ejecutamos este código, el resultado ahora es diferente:

Se generó un **error**

Error: Se generó un **error**

at **Object**.<anonymous> (comparaciones.js:21:15)

at **Module**.\_compile (internal/modules/cjs/loader.js:1076:30)

at **Object**.**Module**.\_extensions..js (internal/modules/cjs/loader.js:1097:10)

at **Module**.load (internal/modules/cjs/loader.js:941:32)

at **Function**.**Module**.\_load (internal/modules/cjs/loader.js:782:14)

at **Function**.executeUserEntryPoint [**as** runMain] (internal/modules/run\_main.js:72:12)

at internal/main/run\_main\_module.js:17:47COPIA EL CÓDIGO

Lo que hemos visto anteriormente, la palabra reservada new seguida de Error en mayúscula, es una muestra de cómo trabajamos con las clases en JavaScript.

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Necesitamos dominar algunos conceptos del lenguaje antes de entrar en este tema, por lo que no se tratará en este curso, pero tenga en cuenta ese término y estructura.

# Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84255/next)

Lo que aprendimos en esta aula:

* Las principales características de JavaScript, como ser un lenguaje interpretado y tipeado dinámicamente.
* Que NodeJS es un intérprete de JavaScript para el backend, es decir, no se ejecuta en el navegador, sino en el servidor.
* Cómo leer los errores que aparecen en la consola de la aplicación, pudiendo utilizar un traductor, como Google Translate, para entender cuál es el error que advierte.
* Qué es stacktrace, también llamado stack, que nos ayuda a saber qué está pasando cuando tenemos un error, y dónde se encuentra.
* ¿Qué es la Console API y cómo podemos usarla usando console.log() y console.error()?
* Donde deberían aparecer los mensajes de la consola dependiendo del intérprete que estemos usando, mientras que para NodeJS los mensajes se muestran en la terminal.

# Para saber más: más sobre operadores

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84248/next)

Hasta ahora hemos practicado con algunos de los operadores de JavaScript, especialmente algunos de los que se usan en operaciones matemáticas:

| **nombre** | **operador** |
| --- | --- |
| operador de suma | x + y |
| operador de resta | x - y |
| operador de multiplicación | x \* y |
| operador de división | x/y |

Empezamos por los operadores más usados, porque siempre que aprendemos un nuevo lenguaje de programación, lo mejor es ir absorbiendo poco a poco más conocimientos sobre cada tema, en lugar de intentar dominarlo todo a la vez.

Sin embargo, la lista de operadores es extensa y hay muchas formas de usarlos. Además de las operaciones matemáticas básicas, el = para asignar un valor a una variable y las comparaciones con == y ===, usamos operadores para varias otras tareas de código, por ejemplo:

* ||: Operador “o”, devuelve verdadero si una condición es válida;
* &&: Operador “y”, devuelve verdadero solo si todas las condiciones son válidas;
* != y !==: Operadores “no iguales” y “estrictamente no iguales”, usados para comparación, de la misma forma que == y === devuelven verdadero o falso.

[La documentación de MDN](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_operators) sobre operadores tiene la lista completa, con varios ejemplos. Tómese el tiempo para familiarizarse con cada uno y trate de pensar en ejemplos del uso de cada uno.

# Expresiones y operadores

<https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_operators>

# Practicando los operadores

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84266/next)

Algo muy común para quienes se dedican al desarrollo de software es la comparación de valores entre variables u objetos para ejecutar un comportamiento u otro. El resultado de estas comparaciones son dos: verdadero o falso (true or false).

Para probar nuestro conocimiento de los operadores de comparación de JavaScript, observe las siguientes variables:

a = []

b = 20

c = **true**

d = ''COPIA EL CÓDIGO

Revise las comparaciones a continuación y marque las que la salida a la consola será verdadera.

 **if** (b == '20') {

console.**log**('Verdade')

} **else** {

console.**log**('Falso')

}

Cuando comparamos 20 de tipo entero con '20' de tipo cadena con el operador == (igual, igual), solo se comparan los valores de las variables, dando como resultado el valor verdadero.

 Alternativa correta



**if** (b === 20 && c != **false** && d == 0) {

console.**log**('Verdadero')

} **else** {

console.**log**('Falso')

}

La variable “b” tiene el mismo valor y el mismo tipo y la primera comparación es verdadera. Dado que el valor almacenado en la variable “c” es verdadero y la comparación dice que “c” no es falso, el resultado también es verdadero. En la última comparación, el valor de “d” que es una cadena vacía se compara con 0 y esto también es cierto. Por lo tanto, la salida de la consola será Verdadero.

 Alternativa correta



**if** (a == 0) {

console.**log**('Verdade')

} **else** {

console.**log**('Falso')

}

El valor de una lista vacía es cero, y cuando se compara con un número, tenemos 0 == 0 dando como resultado verdadero.

# Haga lo que hicimos: utilizando ternarios y template string

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84261/next)

En esta clase vimos contenido diferente sobre JavaScript, pero ¿cómo podemos unir todo lo que se estudió?

Tomando el ejemplo de venta de bebidas para personas mayores y menores de edad, creemos algunas variables para identificar a un cliente, su edad y opciones de bebida.

**const** nombre = "Leonardo";

edad constante = 39;

**const** bebidaMayorEdad= "cerveza";

**const** bebidaMenorEdad = "jugo"; COPIA EL CÓDIGO

Los operadores, entre otras cosas, se usan para comparar condiciones, algo que se usa a menudo en la programación lógica. La notación ${} permite la inserción de valores de variables dentro de una cadena de texto, pero no solo eso; Probemos algo nuevo:

**const** pedido = `${nombre} dice: "Por favor, quiero beber ${edad >= 18 ? bebidaMayorEdad : bebidaMenorEdad}"`

console.log (pedido);COPIA EL CÓDIGO

El resultado que se muestra en la terminal es la oración completa:

Leonardo dice: "Por favor, quiero beber cerveza"COPIA EL CÓDIGO

Esto se debe a que al inicio del código se asignó a la variable edad un valor de 39, es decir, mayor a 18.

Con este ejemplo, usamos tres herramientas de JavaScript vistas durante esta clase:

1) El uso de los operadores: >= como operador de comparación entre el valor de la variable edad y 18 y también el operador ternario como condicional para devolver el valor de la variable bebidaMayorEdad o bebidaMenorEdad según el resultado de la comparación.

2) Templates strings: Usamos la sintaxis de comilla invertida + ${} junto con comillas dobles ””. ¡Prueba también con comillas simples!

3) Operador ternario: Vimos que es posible no solo mostrar el valor de las variables usando ${}, sino también realizar operaciones con JavaScript - por ejemplo, condicionales - e insertar el correspondiente a verdadero o falso en la cadena de texto.

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

¡Practica mucho y piensa en otros ejemplos para probar el uso de estas herramientas!

# Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84256/next)

Lo que aprendimos en esta aula:

* La diferencia entre hacer comparaciones con ==, donde JavaScript convierte entre tipos de variables antes de hacer la comparación, y ===, donde tanto el valor como el tipo de la variable deben ser iguales.
* Como se escribe un operador ternario, con el que hacemos una comparación entre valores tecleando un ?, seguido de la posibilidad verdadero : y la posibilidad falso, es decir, comparación ? true : false.
* El motivo del nombre “operador ternario” se debe a que tenemos 3 operadores juntos en una sola línea para realizar una tarea y devolver un resultado.
* El uso de los templates strings, que facilita la construcción de cadenas que requieren concatenación (la operación en la que se unen 2 cadenas), se introdujo en ES6.
* Cómo usar templates strings, escribiendo el texto entre comillas invertidas (`) y colocando las variables con ${variable} para que su valor permanezca en esa posición.

# Utilizando parámetros

[**Próxima Actividad**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84267/next)

Existen varios lenguajes de programación como Javascript y Python. Cada uno de ellos tiene su propia sintaxis y en ocasiones se comportan de forma diferente.

Cuando ejecutamos una función en Python que espera un parámetro y no se pasa, recibimos un mensaje de error TypeError: conParametro() takes exactly 1 argument (0 given), lo que indica que la función conParametro espera un parámetro. Sin embargo, esto no sucede en JS.

Tenga en cuenta el siguiente código:

function conParametro(param) {

console.log(param)

}

conParametro();

solución:

Dado que no se pasaron parámetros, la salida de la consola será undefined.

¡Bien! En JavaScript, los valores de parámetros de función son iguales a undefined por defecto.

# Para saber más: tipos de funciones

[**Próxima Actividad**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84250/next)

Los parámetros y el retorno de las funciones se utilizan de acuerdo a cada caso específico. Esto significa que no todas las funciones que escribimos siempre necesitarán uno u otro para hacer lo que necesitan. A continuación tenemos más ejemplos para entender mejor algunas situaciones.

**Función sin retorno y sin parámetro:** La siguiente función sólo ejecuta una instrucción, sin necesidad de poner el resultado a disposición del resto del código. En este ejemplo, elegimos usar una cadena fija, por lo que no se necesitan parámetros.

function saludar(){

console.log('¡Hola chicos!')

}

saludar();Copia el código

**Función sin retorno, con parámetro:** similar a la anterior, pero ahora la función recibe, vía parámetro, el nombre de la persona a saludar. De esta manera, es posible reutilizar la función para que funcione de manera similar al nombre de cualquier persona (siempre que esté en formato de datos de cadena).

**function** **saludarPersona**(nombrePersona){

console.**log**(`hola, ${nombrePersona}!`)

}

**saludarPersona**('Helena');Copia el código

**Función con retorno, sin parámetro:** Es posible combinar funciones para que cada una controle solo una parte del código y trabajen juntas.

En el caso siguiente, la función saludar() no necesita recibir ningún parámetro. Pero justo debajo vemos que se está utilizando para ensamblar una cadena en la función saludarPersona(nombrePersona). Esto significa que la cadena ¡Hola chicos! debe estar disponible para otras partes del programa, es decir, debe devolverse utilizando la palabra clave return.

función **saludar**(){

**return** '¡Hola chicos!';

}

**function** **saludarPersona**(nombrePersona) {

console.**log**(`${saludar()} Mi nombre es ${personName}`);

}

**saludarPersona**('Paula') // ¡Hola chicos! Mi nombre es PaulaCopia el código

La función saludarPersona(nombrePersona) recibe una cadena como parámetro donde podemos pasar cualquier nombre en el momento en que ejecutamos (o llamamos) la función. Cuando eso suceda, la función saludar() también se ejecutará y su valor de retorno: la cadena ¡Hola chicos! - tomará el lugar de ${} donde se llama a la función.

**Función con retorno y más de un parámetro:** ¡Recordar que las funciones pueden recibir tantos parámetros como sea necesario, y que JavaScript identifica los parámetros en orden! Es decir, en el ejemplo a continuación, el parámetro numero1 se refiere a 15, el parámetro numero2 se refiere a 30 y el parámetro numero3 se refiere a 45. Nosotros, que estamos desarrollando el código, damos los nombres a los parámetros según los datos que la función espera recibir, en este caso, números.

function operacionMatematica(numero1, numero2, numero3) {

return numero1 + numero2 + numero3;

}

operacionMatematica(15, 30, 45) // 90Copia el código

**Parámetros x argumentos:** En la práctica se refieren al mismo tipo de datos; alguna documentación se refiere a parámetros en el momento en que se define la función (en este caso, numero1, numero2, etc.) y argumentos como los datos que usamos para ejecutar la función (es decir, 30, 45, etc.).

Todavía queda mucho por estudiar en el tema de las funciones, así que practica mucho porque los parámetros y el retorno son conceptos esenciales.

# Para saber más: estructura del if

[**Próxima Actividad**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84251/next)

Durante el curso, hicimos uso de una estructura llamada condicional para definir el flujo de nuestros programas. Las estructuras condicionales son esenciales para la programación, ya que permiten que un programa ejecute tal o cual bloque de código según una condición, por ejemplo: “Si el correo electrónico y la contraseña son correctos, el usuario podrá acceder al sistema. De lo contrario, debería recibir una advertencia de credenciales de acceso incorrectas”. Una condición excluye automáticamente a la otra, ya sea que ambas credenciales (correo electrónico y contraseña) sean correctas o no.

Las estructuras condicionales permiten que el código se bifurque y tome diferentes "caminos" según la condición dada.

A continuación, recordemos la estructura condicional if y sus variaciones:

**if**

La principal estructura condicional es if, que podemos traducir literalmente como “si”:

if(condición) {

// código a ejecutar

}Copia el código

La condición es siempre una expresión que debe evaluarse y devolver true o false (o sus equivalentes "verdadero" o "falso"). El bloque de código dentro del if solo se ejecutará si la expresión devuelve un valor true o "verdadero". Por ejemplo:

**const** num = 10;

**if** (num === **null**) {

console.**log**('número no enviado');

}Copia el código

En el ejemplo anterior, la instrucción console.log('número no enviado'); no se ejecutará, ya que como la variable num se definió con un valor de 10, el resultado de la comparación num === null devolverá false. Es decir, el intérprete como decimos, “no entrará en el if” y no ejecutará el código.

Veamos otro ejemplo:

**const** num = 10;

**if** (num !== **null**) {

console.**log**(`El número es ${num}`);

}Copia el código

En este nuevo ejemplo, el código dentro del bloque if se ejecutará normalmente, porque el resultado de la expresión num !== null (es decir, "el valor de la variable num es diferente de nulo") devuelve verdadero.

La expresión anterior también podría escribirse de la siguiente manera:

**if** (num) {

console.**log**(`El número es ${num}`);

}Copia el código

Con el mismo resultado.

Dado que nulo es un valor que podemos asignar a una variable, también sería posible la siguiente estructura:

**const** num = **null**;

**if** (num === **null**) {

console.**log**('número no enviado');

}Copia el código

El código anterior también se puede escribir de la siguiente manera, ya que nulo se considera un valor falsy:

**if** (!num) {

console.**log**('número no enviado');

}Copia el código

**Múltiples condiciones**

Puede usar los operadores AND (&&) u OR (||) para ejecutar el código en el si una de las condiciones es verdadera, o ambas:

**const** num = 11;

**if** (num > 10 || !num) {

console.**log**('número no válido');

}Copia el código

La consola muestra el mensaje correctamente, ya que una de las condiciones se cumplió y devolvió true (num > 10). Usamos el operador lógico ||, (o en español) para especificar que la condición if se validará si solo una de las condiciones es verdadera, pero no necesariamente todas.

También es posible usar el operador lógico && (y en español) para especificar que la condición if se validará sólo si todas las condiciones devuelven verdadero:

**const** num = 11;

**if** (num > 10 && num < 20) {

console.**log**('número válido');

}Copia el código

La consola anterior funciona con normalidad, ya que el número 11 cumple las dos condiciones especificadas en if: num es mayor que 10 y (&&) es menor que 20.

**if…else**

Otra forma muy utilizada de condicional es agregar la cláusula else para que se ejecute cuando la condición if devuelva false. Podemos leer if…else como “si… de lo contrario”, por ejemplo “si una condición dada es verdadera (true) ejecutar un código determinado; de lo contrario, ejecute otro código”.

**function** **verificaNumero**(numero) {

**if** (numero > 10) {

**return** 'número mayor que 10';

} **else** {

**return** 'número no es mayor que 10';

}

}

console.**log**(**verificaNumero**(9)) //número no es mayor que 10Copia el código

En el ejemplo anterior, el número de comparación > 10 devuelve false, ya que la función se ejecutó con 9 como argumento para numero. Por lo tanto, el bloque de código dentro del if se ignora y el intérprete va directamente al bloque dentro del else.

Tenga en cuenta que else no tiene ninguna condición entre paréntesis, ya que se ejecuta automáticamente si la condición if devuelve un valor false o "falso".

**else if**

El condicional if…else ejecutará un bloque de código entre dos posibilidades, dependiendo del resultado de la expresión que se pasa como condición if. Sin embargo, en algunos casos se necesitan más de dos opciones de flujo para el código; para estos casos existe la cláusula else if:

**const** num = 15;

**if** (num >= 0 && num <= 10) {

console.**log**('número entre cero y diez');

} **else** **if** (num > 10 && num <= 20) {

console.**log**('número entre diez e veinte');

} **else** **if** (num > 20 && num <= 30) {

console.**log**('número entre veinte e treinta');

} **else** {

console.**log**('otro número');

}Copia el código

Tenga en cuenta que la cláusula else if también necesita recibir la condición (o condiciones) como parámetros, a diferencia de else, que debe cerrar la cadena de condicionales y continúa sin recibir ningún parámetro, ya que es el código que se ejecutará si todas las demás condiciones los anteriores fallan.

IMPORTANTE: Cuando trabaje con múltiples condiciones y else if, recuerde siempre que cada condición y cada bloque (if, else if y else) deben representar condiciones mutuamente excluyentes. Es decir, no puede haber ambigüedad entre las condiciones: la misma condición es válida tanto en if como en else if, por ejemplo.

¡Prueba con diferentes valores y verifica los resultados, es muy divertido ver como el flujo cambia de acuerdo a las condiciones 🙂!

# Para saber más: Git y Github

[**Próxima Actividad**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84252/next)

Git es lo que llamamos un sistema de versión (o control) de código. Hay otros, pero se ha convertido en el más utilizado a nivel mundial.

GitHub es una plataforma que utiliza la tecnología Git para que cualquier persona registrada pueda guardar, actualizar y acceder a sus códigos a los códigos de su grupo de trabajo o empresa.

En este curso no vamos a cubrir Git o cómo funciona. Si necesita refrescar su funcionamiento, puede consultar los siguientes materiales de referencia:

* [El curso Alura de GitHub y control de versiones que forma parte de la formación Beginner in Programming](https://www.aluracursos.com/curso-online-git-github-control-version)
* [Git y GitHub para Principiantes en nuestro canal de Alura mas](https://www.youtube.com/watch?v=-LmFK6skG7s)
* [Artículo que escribí para resumir las características y funcionalidades básicas de los sistemas de versionamiento. Uso de git y github](https://ljcl79.medium.com/sistemas-de-versionamiento-qu%C3%A9-son-y-como-nos-ayudan-3fef83f3e33)

# Haga lo que hicimos: separando el código en funciones

[**Próxima Actividad**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84262/next)

Ahora que sabemos cómo declarar funciones con piezas específicas de código y ejecutarlas en el momento adecuado, podemos volver a lo que hemos hecho en clases anteriores y reescribir, o refactorizar, para que cada parte esté separada en su función.

Ver opinión del instructor

### Opinión del instructor

Un ejemplo que podemos aplicar con la suma de variables que creamos al principio del curso:

**const** num1 = 5;

**const** num2 = 5;

**const** resultado = num1 + num2;Copia el código

El código anterior se puede refactorizar como una función, tomando los números como argumentos para que la función se pueda ejecutar con cualquier valor:

**function** **suma**(num1, num2) {

**return** num1 + num2;

}

console.**log**(**suma**(5, 5))Copia el código

¡Practica mucho y, si lo necesitas, estamos disponibles para ayudarte en el foro de Alura Latam!

# Lo que aprendimos

[**Próxima Actividad**](https://app.aluracursos.com/course/javascript-tipos-variables-funciones/task/84257/next)

En esta clase vimos:

* ¿Qué son las funciones? Son pequeñas piezas de código que se pueden ejecutar una o más veces.
* Las ventajas de usar una función: con ellas podemos hacer el código más corto y legible, facilitando el mantenimiento.
* ¿Cómo devolver información desde la función? Usando return, recordando que console.log() solo muestra la información en la terminal y no para otras partes del código.
* La utilidad de los argumentos, ya que con ellos podemos pasar variables para que las funciones puedan usar los valores.
* Que con el hoisting JavaScript analiza todo el código buscando variables declaradas con var y funciones para llevar dichas declaraciones al principio del código.
* Expresiones de función, una forma diferente de construir funciones usando variables de tipo const y llamándolas por su nombre. Recordando que es necesario que el programa pase por la variable antes de que podamos llamarla, ya que no hay soporte para hosting.
* Función de flecha, una función declarada de manera más compacta usando una const. La función de flecha tampoco admite hoisting.

ARTICULO:

# **Strings en JavaScript ¿qué son y cómo manipularlas?**

<https://www.aluracursos.com/blog/strings-con-javascript-que-son-y-como-manipularlas>

<https://www.aluracursos.com/blog/strings-con-javascript-que-son-y-como-manipularlas>