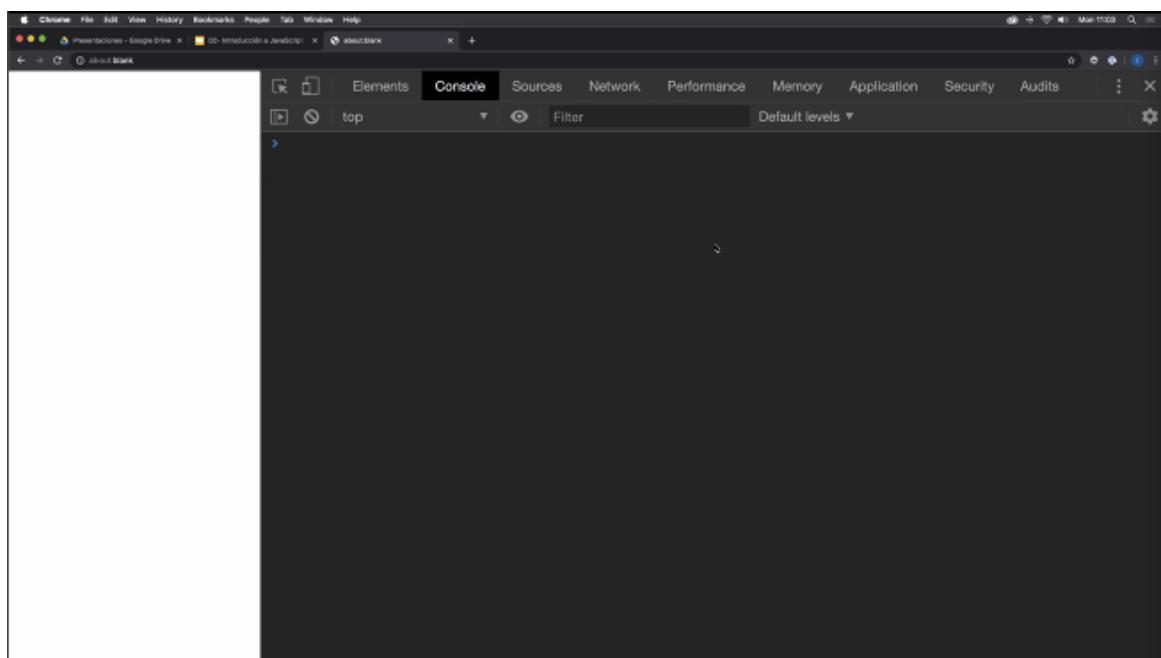


# M1 - 2 - Introducción JavaScript

## Introducción

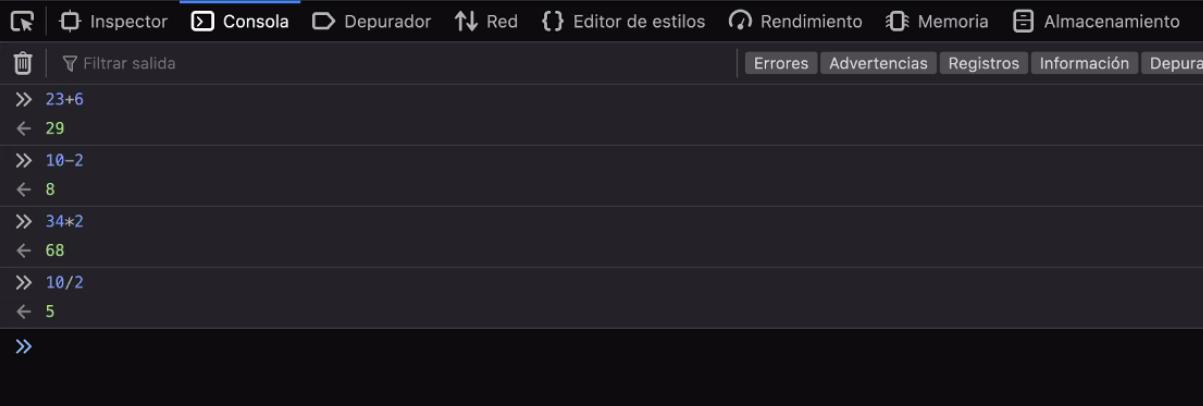
JavaScript es un lenguaje de programación, técnicamente css y html no son lenguajes de programación por que no tiene ninguna lógica por detrás, en cambio un lenguaje de programación si que tiene una lógica. En el caso de javascript vamos a escribir el código en el Visual Studio Code y el propio navegador va a ser el que vaya leyendo el código línea a línea y ejecutandolo, lo que se llama un lenguaje de programación interpretado.

Vamos a abrir el navegador, y en la barra de direcciones introducimos ‘about:blank’, nos dirigimos a una página que vemos que está vacía. Si abrimos el inspector de elementos, vemos que al lado de la pestaña de ‘Elements’, hay otra pestaña que tiene el nombre de ‘Console’ o consola en castellano, vamos a pinchar en la pestaña console y vemos que aparece algo parecido a la foto que tenemos a continuación. El aspecto cambiará un poco según el navegador que estemos utilizando pero viene a ser lo mismo.



En esta consola podemos ejecutar expresiones JavaScript. Empezaremos viendo operaciones simples de matemáticas. Como veis podemos sumar, restar, multiplicar y dividir sin ningún problema y cómo lo haríamos en una calculadora, salvo algunos símbolos que cambian en algunas operaciones. En el caso de la multiplicación, el símbolo que vamos a utilizar es el asterisco ‘\*’, de hecho si probáis a hacer la multiplicación con la ‘x’ veréis que no lo detecta y nos dará un error. También es distinto el símbolo de la división ‘/’.

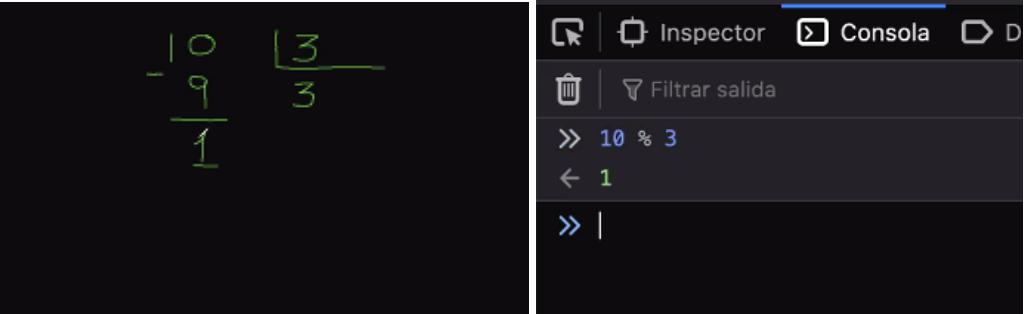
- **suma** → ‘+’
- **multiplicación** → ‘\*’
- **resta** → ‘-’
- **división** → ‘/’



```

    Inspector Consola Depurador Red Editor de estilos Rendimiento Memoria Almacenamiento
    Filtrar salida Errores Advertencias Registros Información Depur
    >> 23+6
    < 29
    >> 10-2
    < 8
    >> 34*2
    < 68
    >> 10/2
    < 5
    >>
  
```

Además de estas operaciones, podemos utilizar otro operador, que llamamos ‘módulo’ y que se corresponde con el símbolo del porcentaje ‘%’. A través de una división con este símbolo lo que vamos a conseguir es obtener el resto de esa división. Por ejemplo, 10 entre 3, tendría un resto de 1. Cómo vemos en el ejemplo de abajo, esta operación utilizando el símbolo del módulo, nos devuelve el resto de la operación.

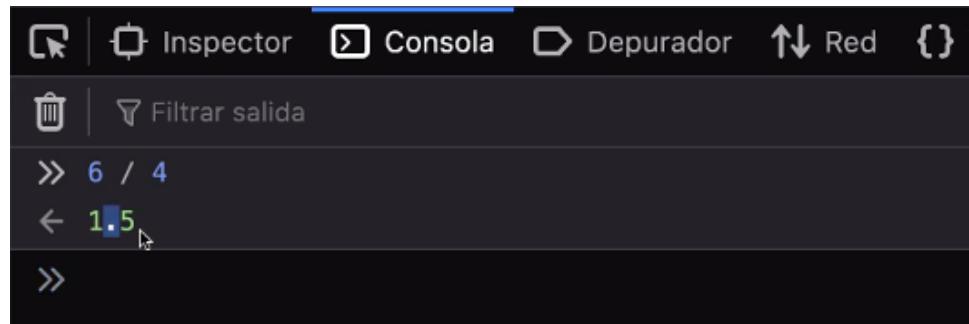


$$\begin{array}{r} 10 \\ - 9 \\ \hline 1 \end{array}$$

```

    Inspector Consola D
    Filtrar salida
    >> 10 % 3
    < 1
    >>
  
```

Si hacemos la división con el símbolo de la división normal, nos devolvería el resultado normal de la división, es decir, si es una división entera nos devolverá un número entero, y si no, nos devolverá un número decimal. Es importante que sepáis que los números decimales utilizan el carácter . (punto) en vez de , (coma) para separar la parte entera y la parte decimal.

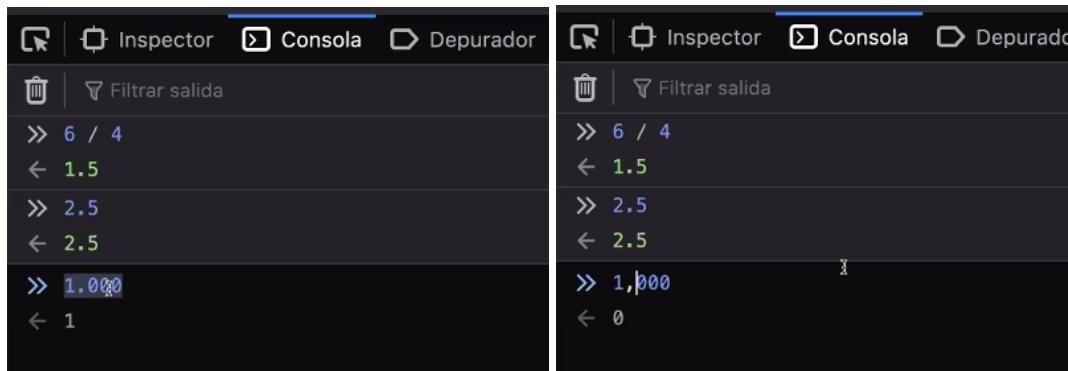


```
< 1.5
```

Por lo tanto, hay que tener mucho cuidado cuando introducimos números decimales o números con 4 o más dígitos.

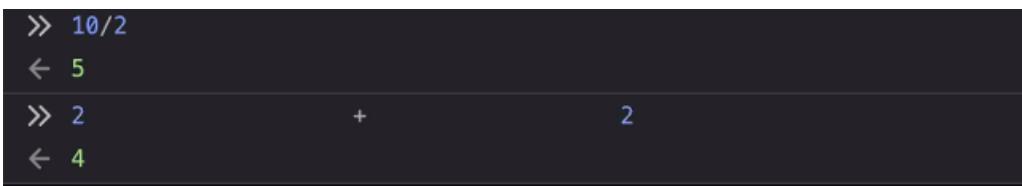
Por ejemplo, cuando queremos introducir números como (mil → 1.000) , si lo escribimos con un punto, lo va a interpretar como un uno, si lo escribimos con una coma, lo intentará interpretar y lo que hará finalmente será quedarse con la parte decimal como si fuera un número entero, es decir, si introducimos 5,67 lo entenderá o traducirá a 67. Con lo que hay que tener dos cosas claras en cuanto a esto:

- Los números decimales los escribiremos con punto
- Los números enteros de 4 o más dígitos los escribiremos sin punto ni coma, por ejemplo → 1000



1000	1,000
1000	1000

Otra de las cosas que tenemos que tener en cuenta son los espacios. ¿Os acordáis que en html estábamos diciendo que los espacios los ignoraban? Aquí también, podemos poner tantos espacios como queráis entre el 2 y el más como en el ejemplo de abajo que no nos va a dar error, aunque no sería recomendable que pongáis tantos espacios, más que nada por la legibilidad del código.



```
10/2
5
2 + 2
4
```

Lo que hay que tener mucho cuidado es donde ponemos los espacios, por que puede pasar que lo que hagamos sea cambiar el sentido en el que se interpretan las instrucciones. Por ejemplo, no es lo mismo → `10 / 2`, que `1 0 / 2`. En esta segunda opción sí que nos saltaría un **error de sintaxis**. Este error nos indica que la manera en la que hemos escrito JavaScript no se ha entendido.

Hay que entender que como cualquier lenguaje de programación, existen unas ciertas normas sobre cómo debe estar escrito y si estas normas no se cumplen nos saldrán estos errores advirtiéndonos de que la sintaxis no es correcta, esto a su vez nos va a ayudar a encontrar fallos en nuestro programa.

```
>> 1 0 / 2
! SyntaxError: unexpected token: numeric literal [Saber más]
>>
```

## Variables

Ahora que vamos a empezar con otro tema, podemos eliminar de la consola las instrucciones anteriores clicando sobre el símbolo de la papelera que encontramos justo encima de la consola a la izquierda.

Las variables en los lenguajes de programación son muy parecidas a las variables que utilizamos en matemáticas (`x = 3`), al final **su función es almacenar** y hacer referencia a otro valor. Podemos hacer una suma directamente sin variables:

```
Filtrar salida
>> 7 * 2
← 14
>>
```

Pero al final, el resultado de esa suma no la hemos almacenado en ningún lugar con lo que si quisiéramos posteriormente utilizarla en otra operación tendríamos que recordarlo o volver a hacer la operación, lo que no es nada práctico.

Aplicándolo a un caso real, supongamos que estamos creando un programa para llevar la gestión de una boda. Se decide que a cada invitado se le van a regalar dos puros, si tenemos 7 invitados, sólo tendríamos que multiplicar los invitados por el número de puros que va a recibir cada uno y ya estaría. Pero qué pasa si al final, se animan 3 invitados más, tendríamos que volver a hacer la operación de nuevo y añadirle los 3 invitados... volviendo a la conclusión de que no es nada práctico.

Deberíamos puntualizar que en el caso de las ecuaciones hay que tener mucho cuidado porque funcionan exactamente igual que en matemáticas, al fin y al cabo son ecuaciones matemáticas.

Entonces recordad que poner paréntesis es super importante, lo que ponemos entre paréntesis es la operación que primero se calcula. Si no ponemos paréntesis, primero se calculan las multiplicaciones y las divisiones de izquierda a derecha y después sumas y restas de izquierda a derecha. Debajo tenemos un ejemplo:

```
>> 7 * 2
← 14
>> 7 + 3 * 2
← 13
>> (7 + 3) * 2
← 20
```

## Crear variables en JavaScript

Para crear una variable tenemos que utilizar la palabra ‘let’, el nombre con el que vamos a identificar a dicha variable seguido de un igual y el valor que le queremos asignar.

Es importante que quede claro que la palabra reservada ‘let’ sólo la debemos utilizar al definir por primera vez la variable. Para utilizar dicha variable en el resto de instrucciones de javascript solo es necesario escribir su nombre. No puede haber dos variables con el mismo nombre.

En cuanto al nombre, hay algunas normas que debemos conocer sobre cómo debe ser el nombre que vayamos a asignar a una variable:

- Sólo puede estar formado por letras, números y los símbolos \$ (dólar) y \_ (guión bajo). Nosotros por el momento, mejor si sólo utilizamos letras y números.
- No podemos añadir espacios, si nuestro nombre consta de más de una palabra lo que haremos será identificar el comienzo de la segunda palabra a través de una mayúscula, es decir, por cada palabra nueva una mayúscula. Por lo demás, en minúscula sería más correcto. (camelCase)
- El primer carácter no puede ser un número.

Por ejemplo:

```
>> let purosPorInvitado
```



## Cómo incluir JavaScript en nuestra página HTML

Vamos a volver al Visual Studio Code y vamos a crear un archivo.html

```
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Introducción a JavaScript</title>
</head>
<body>
  <h1></h1>
  }

  <script>
    |
  </script>
</body>
</html>
```

Si anteriormente habíamos visto cómo podíamos incluir el CSS a través de las etiquetas **<style></style>** dentro del Head, ahora veremos que podemos utilizar las etiquetas **<script></script>** para incluir nuestro código JavaScript. Sólo habrá una diferencia y es que las etiquetas **script** se deben colocar al final, justo antes de la etiqueta de cierre del **body**.

Dentro de las etiquetas script podemos escribir comandos de javascript como hacíamos en la consola.

```
<script>
  2 + 2
  3 - 3
  7 * 2
  let puros = 2
  let invitados = 7 + 3
</script>
</body>
</html>
```

Igual que los dobles espacios, en JavaScript tampoco se interpretan los saltos de línea, lo entiende como un espacio, por lo tanto, cada vez que terminemos un comando pondremos un punto y coma para indicar y separar los comandos.

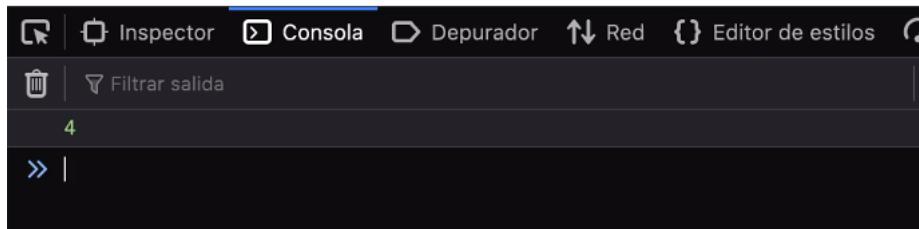
```
<script>
  2 + 2;
  3 - 3;
  7 * 2;
  let puros = 2;
  let invitados = 7 + 3;
</script>
</body>
</html>
```

Si abrimos la página en el navegador y abrimos la consola, vemos que no nos aparece nada. Pero si escribimos 'puros' si que nos devuelve su valor, es decir, que si se está ejecutando el código de javascript, pero no nos está mostrando linea a linea lo que hace.

Cuando queremos mostrar algo en la consola tenemos que utilizar el comando `console.log()` y entre paréntesis lo que queremos mostrar. Por ejemplo:

```
<script>
  console.log( 2+2 );
</script>
```

Y efectivamente en la consola, se muestra el resultado

A screenshot of a browser's developer tools interface, specifically the 'Console' tab. The tab bar above shows 'Inspector', 'Console' (which is highlighted in blue), 'Depurador', 'Red', and 'Editor de estilos'. Below the tab bar, there is a toolbar with icons for search, clear, and filter. The main console area displays the number '4' in green, indicating the result of a previous command. At the bottom, there is an input field starting with '» |'.

## Variables numéricas

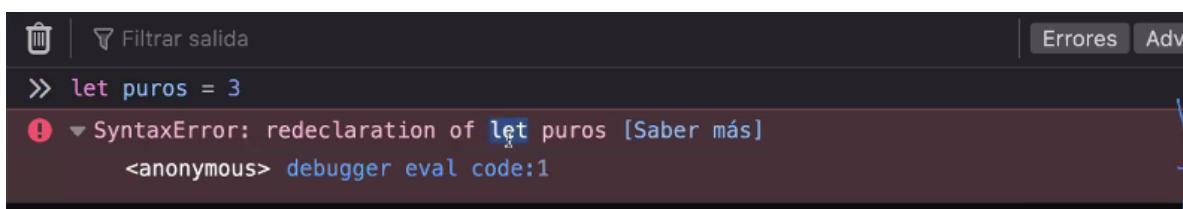
Se utilizan para almacenar valores numéricos, los números pueden ser enteros o decimales y, según eso encontramos dos tipos de variables. Las variables de tipo 'int' → integer (entero en inglés), y variables de tipo 'float' → los decimales.

```
let entero = 24;           // variable tipo int
let decimal = 24.65; // variable tipo float
```

Siguiendo con el ejemplo de la boda, vemos cómo utilizamos 'let' únicamente cuando definimos las variables, pero para referirnos a ellas, para utilizarlas, solamente indicamos el nombre.

```
» let puros = 2
← undefined
» let invitados = 7 + 3
← undefined
» invitados * puros
← 20
»
```

Si ahora que hemos declarado la variable `puros`, intentamos hacer alguna operación con ella o reasignar un valor y añadimos de nuevo 'let', nos va a dar error, porque como hemos comentado anteriormente va a entender que estamos intentando re-declarar la variable y nos va a saltar un error de sintaxis.



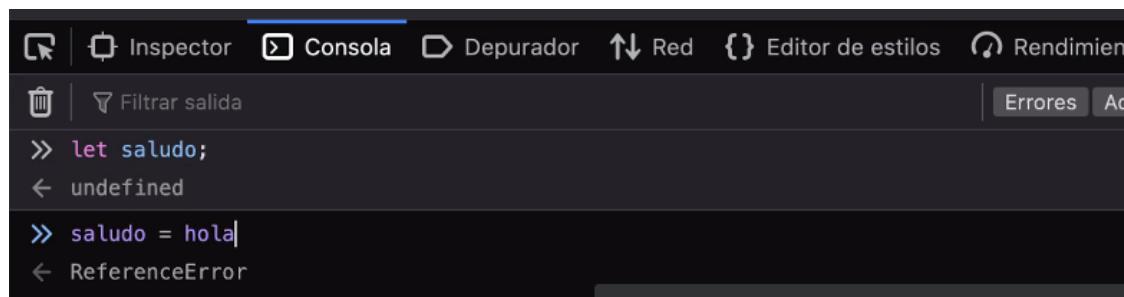
The screenshot shows a browser's developer tools console. At the top, there are buttons for trash, filter, errors, and advanced. Below that, the command `» let puros = 3` is entered. A red exclamation mark icon indicates an error: `! SyntaxError: redeclaration of let puros [Saber más]`. The stack trace shows it's from an anonymous source at debugger eval code:1.

Si quisiéramos sobreescribir una variable, es decir, asignarle otro valor lo tendríamos que hacer de la siguiente manera:

```
>> let dinero = 5;  
< undefined  
>> dinero = 7;  
< 7
```

## Variables de texto

Estas variables almacenan cadenas de texto(palabras, frases, caracteres...), también conocidas como '**string**' en inglés.

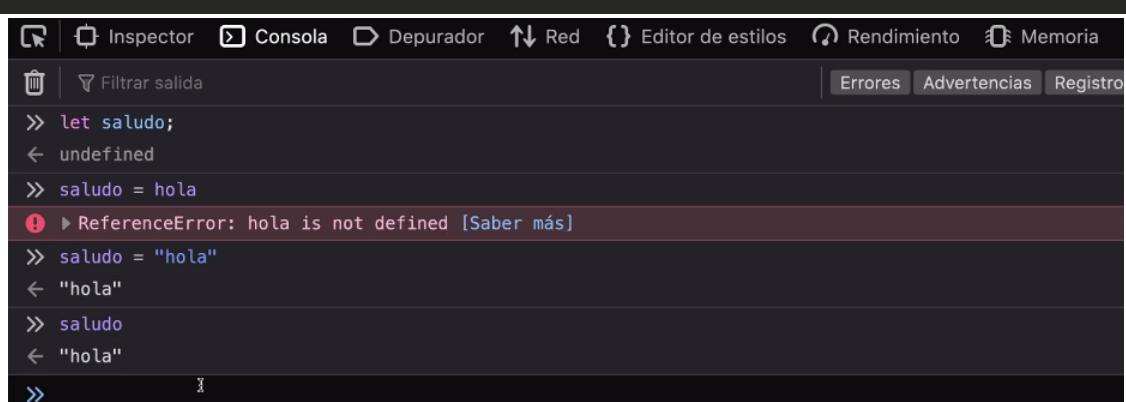


The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The console output is as follows:

```
>> let saludo;  
< undefined  
>> saludo = hola|  
< ReferenceError
```

Si introducimos el texto tal cual, va a pensar que hola es una variable, no entiende que es texto, nos salta un **error de referencia**. Para indicarle que hola es un texto, lo que hacemos es ponerlo entre comillas. Como véis, podemos declarar la variable y asignarle un valor después. Cuando asignamos el valor lo que hacemos es inicializar la variable.

```
let saludo;           // declarar la variable  
saludo = 'hola';   // inicializar la variable
```



The screenshot shows the Chrome DevTools interface with the 'Console' tab selected. The console output is as follows:

```
>> let saludo;  
< undefined  
>> saludo = hola  
✖ ReferenceError: hola is not defined [Saber más]  
>> saludo = "hola"  
< "hola"  
>> saludo  
< "hola"  
>>
```

En cuanto a las comillas podemos utilizar 3 tipos de comillas :

```
"" // comillas dobles
```

```
' ' // comillas simples  
` ` // comillas francesas. Estas comillas son las más extrañas, son las que tenemos al lado de la letra 'p' en el teclado, donde encontramos el icono ^
```

The screenshot shows a browser's developer tools console tab labeled 'Consola'. It displays several string literals being evaluated:

```
"hola"  
"hola"  
'hola'  
"hola"  
`hola`  
"hola"
```

Más adelante veremos la diferencia. Pero por ahora lo que es importante es que cuando vemos las primeras comillas javascript interpreta que vas a introducir texto, hasta que ve las comillas finales por tanto si intentamos entrecollar algo dentro de nuestro string (nuestra cadena de texto) nos va a salir un error. Entenderá que la cadena de texto es (él me ha dicho) pero después (que tal) no es capaz de identificarlo.

```
» "él me ha dicho "qué tal""  
! SyntaxError: unexpected token: identifier [Saber más]
```

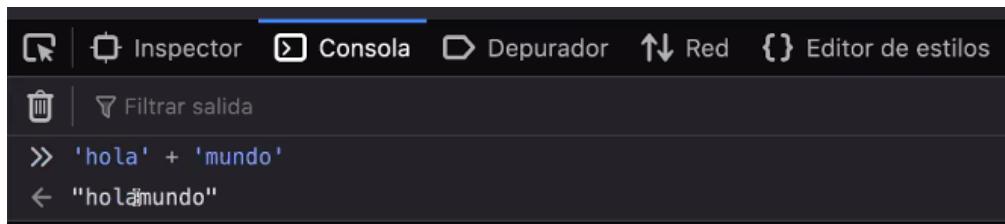
Por lo tanto, como más adelante utilizaremos JavaScript para introducir elementos de HTML con sus atributos y todo, ¿Qué problema vamos a tener? Pues que pasará lo mismo con las comillas que utilizamos en el atributo para darle un valor.

```
» '<p class="importante">hola</p>'  
← "<p class=\"importante\">hola</p>"
```

Por este tipo de problemas es más recomendable en javascript utilizar las comillas simples para que luego cuando estemos mezclando HTML con JavaScript no tengamos problemas. Las últimas comillas ya veremos más adelante para que nos van a servir.

```
>> saludo = saludo + ', mundo'
```

Podemos sumar dos cadenas de texto para concatenarlas, es decir, unirlas de manera que el último carácter del primer string se unirá a el primer carácter del siguiente string, 'sumándose' en uno solo. Para esto es muy importante tener en cuenta los espacios, si por ejemplo, concatenamos 'hola' con 'mundo', la 'a' final de 'hola' se unirá con la 'm' inicial de 'mundo' convirtiéndose en una única cadena de texto 'holamundo' sin espacios.

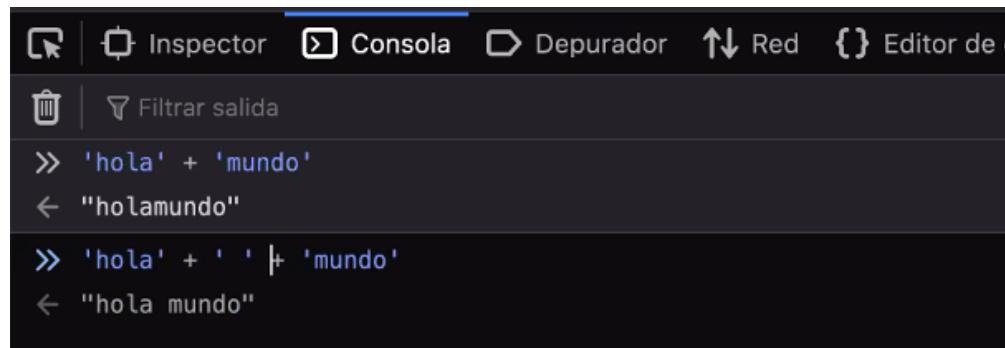


A screenshot of a developer tools interface showing a console tab. The console output shows the following:

```
>> Inspector Consola Depurador Red Editor de estilos
>> Filtrar salida
>> 'hola' + 'mundo'
<- "holamundo"
```

Para crear espacios deberíamos incluir un espacio en alguno de los dos strings, en el caso de incluirlo en 'hola' deberíamos hacerlo al final, y si lo incluimos en 'mundo' debería ser el inicio del string. Esto es porque son los dos lugares por dónde se van a unir ambas cadenas de texto.

Otra de las opciones sería concatenar 3 strings: los dos anteriores y en medio un string con un espacio.



A screenshot of a developer tools interface showing a console tab. The console output shows the following:

```
>> Inspector Consola Depurador Red Editor de estilos
>> Filtrar salida
>> 'hola' + 'mundo'
<- "holamundo"
>> 'hola' + ' ' + 'mundo'
<- "hola mundo"
```

- Problema que nos vamos a encontrar al hacer esto. A la hora de introducir números, cuando concatenamos una cadena con un número, ese número pasa a ser un string, es decir, se ha convertido en texto. Recordad lo de los espacios cuando concatenéis.

 |  Filtrar salida

```
» 'los puros que tenemos que comprar son' + 2
← "los puros que tenemos que comprar son2"
» 'los puros que tenemos que comprar son ' + 2
← "los puros que tenemos que comprar son 2"
```

```
» 'los puros que tenemos que comprar son ' + 10 * 2
← "los puros que tenemos que comprar son 20"
»
```

Bueno pues de momento salvo el problema de los espacios no hemos encontrado ningún otro, ¿No? Hemos añadido un número, hemos hecho una multiplicación y hemos añadido el resultado, ¿Pero qué pasa si en vez de una multiplicación queremos hacer una suma y añadir el resultado?

Qué entenderá el símbolo '+' como una indicación de que queremos concatenar. Por lo tanto, concatenará primero el 10 y seguidamente el 2, es decir, no nos hace la suma.

Esto es por que al concatenar el 10, este se ha convertido a texto y por lo tanto con el 2 ha hecho lo mismo.

```
» 'los puros que tenemos que comprar son ' + 10 * 2
← "los puros que tenemos que comprar son 20"
» 'los puros que tenemos que comprar son ' + 10 + 2
← "los puros que tenemos que comprar son 102"
» |
```

Para evitar esto utilizamos los paréntesis. ¿Os acordáis? Cómo si fuera una ecuación, al poner los paréntesis, primero se calculará la suma y después ya se añadirá el resultado. Por eso hay que tener mucho cuidado cuando concatenamos y hay números por medio.

```
» 'los puros que tenemos que comprar son ' + (10 + 2)
← "los puros que tenemos que comprar son 12"
```

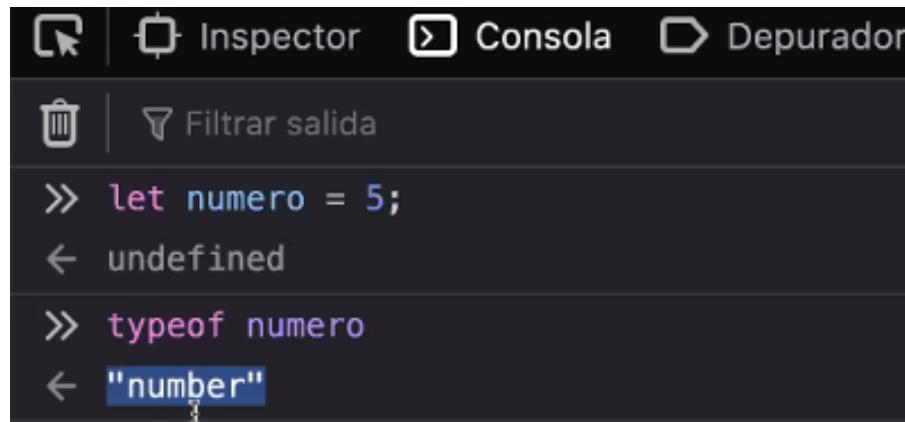
Ya hemos visto cómo al intentar sumar una cadena de texto (por mucho que sea un número) con un número, lo único que vamos a conseguir es concatenarlo. Pero veremos que con el resto de operaciones JavaScript se comporta de una manera muy diferente. Cuando

advierte el símbolo de la multiplicación, intenta convertir este texto en número para poder operar.

```
>> '2'  
< "2"  
>> '2' +3  
< "23"  
>> '2' * 3  
< 6
```

## typeof

Tenemos un operador `typeof`, que nos devuelve el tipo de dato que se encuentra en la variable que indicamos seguidamente



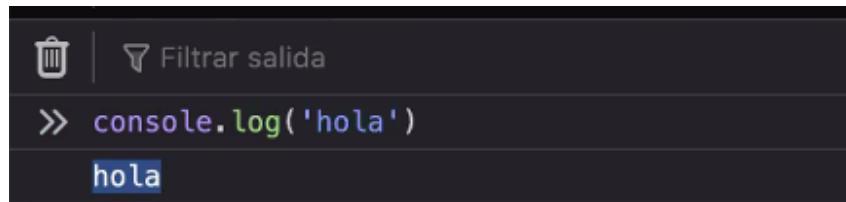
A screenshot of a browser's developer tools console. At the top, there are tabs for 'Inspector', 'Consola' (selected), and 'Depurador'. Below the tabs, there are icons for trash ('Borrar') and filter ('Filtrar salida'). The console output shows:

```
>> let numero = 5;
← undefined
>> typeof numero
← "number"
```

## Debugging

### console.log

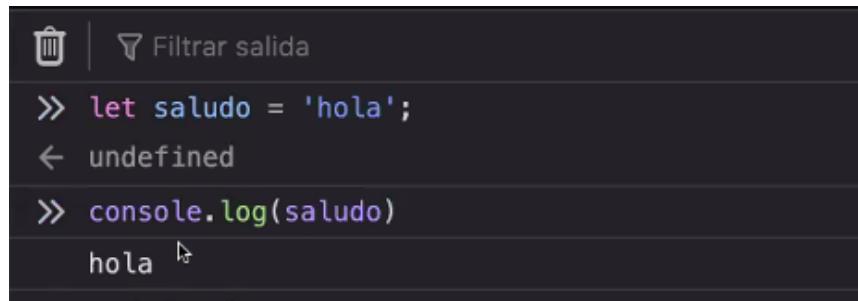
A través de '`console.log()`' podemos mostrar en la consola cualquier cosa, tenemos que indicar entre las paréntesis lo que queremos que se muestre, recordad que si queremos mostrar un texto tenemos que utilizar las comillas. Podemos mostrar un texto:



A screenshot of a browser's developer tools console. At the top, there are icons for trash ('Borrar') and filter ('Filtrar salida'). The console output shows:

```
>> console.log('hola')
hola
```

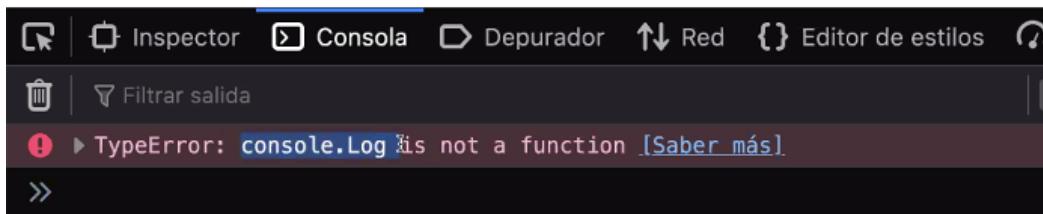
Podemos mostrar el valor de una variable, si lo que queremos mostrar es el valor de una variable, indicamos entre paréntesis el nombre de la variable:



A screenshot of a browser's developer tools console. At the top, there are icons for trash ('Borrar') and filter ('Filtrar salida'). The console output shows:

```
>> let saludo = 'hola';
← undefined
>> console.log(saludo)
hola ↴
```

JavaScript sí que se diferencian las mayúsculas y las minúsculas con lo que si intentamos ponernos el 'log', o el 'console' en mayúsculas nos va a dar error. Los errores nos salen siempre en la consola.



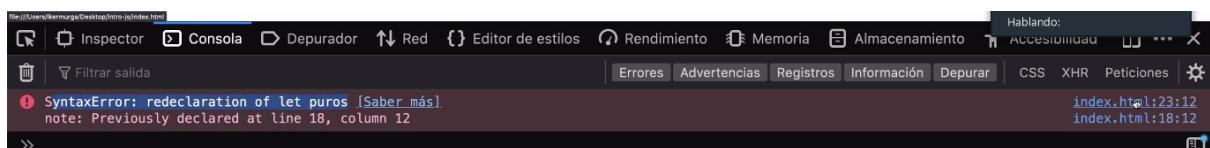
Además de indicarnos el tipo de error, y normalmente darnos una pequeña información sobre el error, como en el ejemplo anterior que ya nos indicaba que el error estaba en 'console.Log' que no lo reconocía, también nos informa en qué línea de nuestro código se encuentra dicho error.

En el siguiente ejemplo, hemos declarado dos veces la variable puros, por lo que nos debería aparecer un error.

```
<script>
    console.log( 2+2 );
    -3;
    75 * 2;
    let puros=2;
    let invitados = 7+3;

    console.log( puros );

    let puros = 8;
</script>
</body>
</html>
```



Cuando abrimos la consola en el navegador, vemos que nos ha saltado un error de sintaxis por redeclarar dos veces la misma variable pero además nos indica en qué línea de nuestro código. Si observamos los enlaces que nos aparecen a la derecha de la consola y clicamos sobre ellos nos muestra nuestro código y subrayado en azul el lugar dónde hemos cometido dicho error.

```
1 <!DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Introducción a JavaScript</title>
7 </head>
8 <body>
9   <div><p>holo</p>
10  <p>adios</p>
11 </div>
12
13
14 <script>
15   console.log( 2+2 );
16   -3;
17   75 * 2;
18   let puros=2;
19   let invitados = 7+3;
20
21   console.log( puros );
22
23   let puros = 8;
24 </script>
25 </body>
26 </html>
```

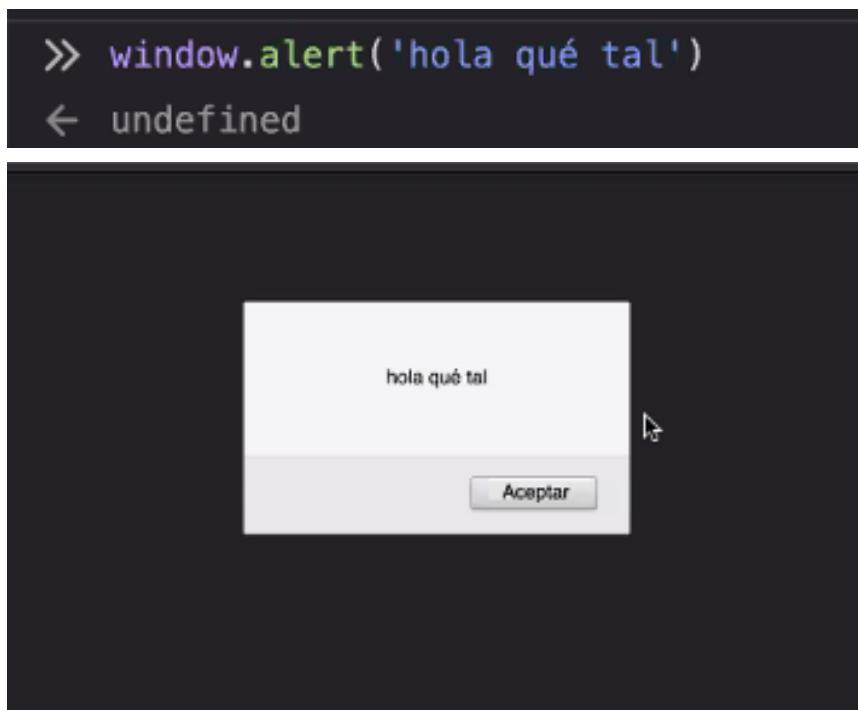
## Debugger

\_>

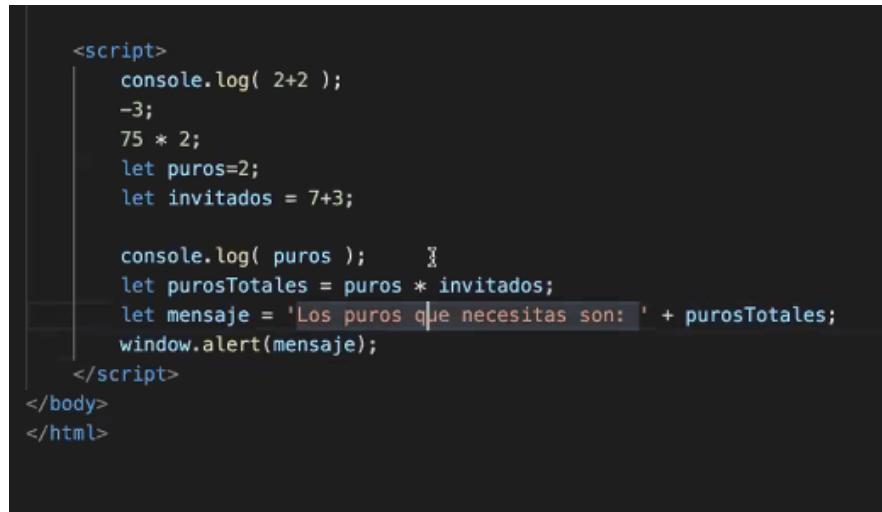
## Input/output

`window.alert()`

Con `window.alert()` podemos mostrar mensajes de alerta en el navegador con lo que podremos dar datos al usuario ( output ), para ello indicamos entre paréntesis el mensaje que queremos que se muestre:



Veamos un ejemplo desde Visual Studio Code



```
<script>
    console.log( 2+2 );
    -3;
    75 * 2;
    let puros=2;
    let invitados = 7+3;

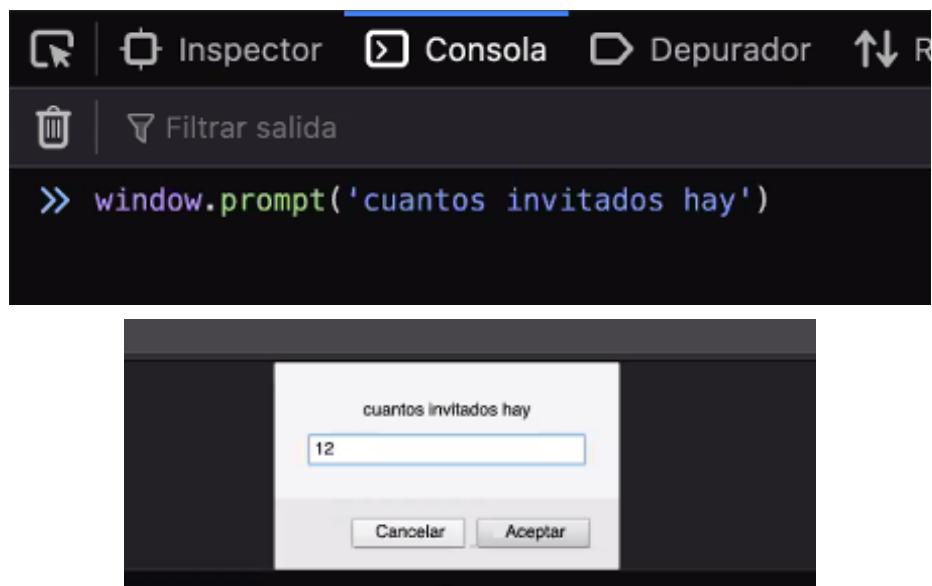
    console.log( puros );
    let purosTotales = puros * invitados;
    let mensaje = 'Los puros que necesitas son: ' + purosTotales;
    window.alert(mensaje);
</script>
</body>
</html>
```



The screenshot shows a Windows-style alert dialog box. The message area contains the text "Los puros que necesitas son: 20". Below the message is an "Aceptar" (Accept) button.

### window.prompt()

Acabamos de ver cómo, desde javascript, podemos mostrar un mensaje en el navegador, pero también podemos recoger desde el navegador información (input). ¿Cómo? a través de 'window.prompt()'. Recordad que JavaScript distingue entre mayúsculas y minúsculas, por lo tanto, tenemos que tener muy en cuenta como se escriben los comandos, si escribes 'Window.Prompt()' os saltará un error



Podemos almacenar, asignar el valor que nos introduce el usuario en una variable para después poder utilizarlo como nosotros queramos, en el ejemplo siguiente hemos almacenado el valor introducido por el usuario en la variable invitados, y si después mostramos el valor de dicha variable, vemos cómo efectivamente la variable almacena el valor que ha introducido el usuario. Ahora bien, si nos fijamos el valor esté entre comillas, ¿Qué significa esto? que este método nos devuelve lo que ha introducido el usuario en formato string, es decir, nos devuelve una cadena texto. Por lo tanto, habrá que tener muy en cuenta esto si después vamos a querer operar con dicho valor por los problemas que nos puedan surgir, como hemos visto anteriormente, por hacer operaciones con un número y una cadena de texto.

```
>> let invitados = window.prompt('cuantos invitados hay');
← undefined
>> invitados
← "12"
>>
```

```
>> let invitados = window.prompt('cuantos invitados hay');
← undefined
>> invitados
← "12" ↵
>> invitados + 2
← "122"
```

Para evitar esto deberíamos convertir este texto en un número, para poder operar sin ningún problema, y ¿Cómo haríamos esto? A través de una función:

## Conversión de tipo

### parseInt() y parseFloat()

¿Os acordáis que cuando hemos visto las variables numéricas veímos dos tipos? Pues podemos convertir o pasar la cadena de texto a cualquiera de ambos tipos, decidiremos cual vamos a utilizar según las operaciones que queramos realizar con ellos. Obviamente si intentamos pasar o convertir una cadena de texto a un número (ya sea float o int) pero el contenido de esa cadena de texto no es numérico, no se va a convertir, pero tampoco nos va a dar un error, nos va a devolver un 'NaN', 'Not a Number'.

```
>> parseInt('5')
← 5 ↵

>> parseInt('5')
← 5
>> parseInt('hola')
← NaN
>> |
```

Podemos indicar directamente entre paréntesis que lo que queremos convertir es lo que nos va a devolver 'window.prompt()':

```
>> parseInt(window.prompt('cuantos invitados hay'))
```

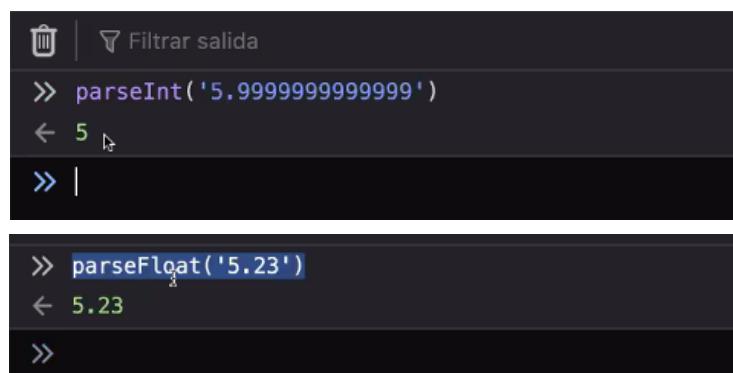
O convertir el valor de una variable:

```
>> let invitados = window.prompt('cuantos invitados hay');
← undefined
>> invitados
← "12"
```

```
» parseInt(invitados)
← 12
```

Tenemos dos tipos de variables numéricas y dos funciones de conversión, una para cada una de ellas, ¿Pero qué diferencia hay entre ellas? Bueno, como habíamos comentado ‘int’ de integer en inglés almacena valores de números enteros y ‘float’, por el contrario, números decimales. Las funciones convierten a un tipo de dato, parseInt() pasa el texto a un número entero y parseFloat() a un número decimal.

Pero debemos tener en cuenta que si tenemos un número decimal y utilizamos el parseInt() lo pasará a un número entero, SIN REDONDEAR, simplemente nos va a devolver la parte entera del número:



```
Filtrar salida
» parseInt('5.999999999999')
← 5
» |
»
» parseFloat('5.23')
← 5.23
»
```

Veamos un ejemplo de cómo utilizaremos esto en nuestro programa:

En este caso tenemos un programa que va a preguntar al usuario cuantos puros son los que desea repartir a cada invitado el día de su boda. Para ello utilizamos ‘window.prompt()’ para mostrar el mensaje que le indicamos entre paréntesis, y recoger la respuesta del usuario. Esta respuesta la vamos a almacenar en una variable ‘respuestaPuros’.

Recordar, que `window.prompt()` nos devuelve una cadena de texto, es decir recoge lo que ha introducido el usuario, su respuesta pero en formato string, cadena de texto. Por lo tanto, y cómo nosotros esperamos un número con el que además vamos a operar, deberemos convertirlo, pasarlo a un número. Lo pasaremos a un int, a un número entero y para eso utilizaremos la función `parseInt()`, y entre paréntesis el nombre de la variable ‘respuestaPuros’ cuyo valor queremos convertir en un int.

Podéis hacer estos dos pasos directamente en uno como veis en el ejemplo de abajo como comentario.

Hacemos la misma operación con los invitados. Una vez tenemos los datos que necesitamos:

- los puros que va a recibir cada invitado
- el número de invitados que van a asistir a la boda

Sólo tenemos que realizar la operación y mostrar el resultado, este lo mostraremos a través de un mensaje de alerta, utilizando window.alert().

Como véis para definir el mensaje hemos concatenado el resultado de la operación con el mensaje que queríamos añadir.

```
<script>
    let respuestaPuros = window.prompt('Cuantos puros por invitado');
    let puros = parseInt(respuestaPuros);

    //let puros = parseInt(window.prompt('Cuantos puros por invitado'))

    let respuestaInvitados = window.prompt('Cuantos invitados hay');
    let invitados = parseInt(respuestaInvitados);
    // let invitados = parseInt(window.prompt('Cuantos invitados hay'));

    let purosTotales = puros * invitados;
    let mensaje = 'Los puros que necesitas son: ' + purosTotales;
    window.alert(mensaje);
</script>
```

Para introducir el valor de la variable en el medio de la cadena de texto deberíamos haberlo concatenado de la siguiente manera:

```
» let puros = 15;
← undefined
» 'Necesitas ${puros} puros'
← "Necesitas 15 puros"
```

Cómo véis resulta un poco tedioso tener que mezclar texto con las variables para mostrar su valor dentro de un texto. Para eso se utiliza el tercer tipo de comillas (comillas francesas ) que habíamos visto anteriormente. A través de estas comillas podemos incluir el valor de variables dentro de las cadenas de texto de una manera mucho más amena.

```
» `Necesitas ${puros} puros`
← "Necesitas 15 puros"
```

Simplemente indicamos a través de las comillas el inicio y el fin de la cadena de texto, y dentro, cuando queramos mostrar el valor de una variable simplemente habrá que indicarlo con la siguiente estructura → símbolo del dolar (\$), y entre llaves el nombre de la variable \${puros} esto lo que va a hacer es calcular el valor de la variable y meterlo dentro del texto en ese momento.

## Operadores

Ya hemos visto algunos operadores:

- Operador de asignación `=`
- Operadores matemáticos `+ - / % *`

Ahora veremos otros operadores que nos facilitarán muchas operaciones.

### Operadores matemáticos con operadores de asignación

Podemos utilizar los operadores matemáticos junto con los de asignación para abreviar las instrucciones:

```
let numero1 = 5;
numero1 += 3; // numero1 = numero1 + 3 (8)
numero1 -= 1; // numero1 = numero1 - 1 (4)
numero1 *= 2; // numero1 = numero1 * 2 (10)
numero1 /= 5; // numero1 = numero1 / 5 (1)
numero1 %= 4; // numero1 = numero1 % 4 (1)
```

```
>> let purosPorInvitado = 5;
← undefined
>> purosPorInvitado = purosPorInvitado + 2;
← 7
>> purosPorInvitado += 2;
← 9
>> purosPorInvitado -= 2;
← 7
>> purosPorInvitado *= 2;
← 14
>> purosPorInvitado /= 2;
← 7
>>
```

## Operadores de incremento y decremento

Estos operadores nos sirven para incrementar o decrementar en una unidad el valor de una variable. Podemos utilizar ambos operandos de dos maneras:

- a través del prefijo `++` o `--`
- a través del sufijo `++` o `--`

La diferencia entre ambos es que a través del prefijo el valor de la variable incrementa en 1 antes de ejecutar la sentencia en la que aparece y, sin embargo, a través del sufijo primero se ejecuta la sentencia y después se le suma uno a la variable. Veamos varios ejemplos:

```
>> let puros = 7;
← undefined
>> console.log(puros++);
    7
← undefined
>> puros
← 8
```

En el ejemplo anterior hemos declarado una variable ‘puros’ a la cual le hemos asignado un valor de 7, después hemos utilizado el operador `++` para incrementar en uno su valor pero lo hemos indicado como sufijo. Al indicarlo como sufijo, primero se ejecuta la sentencia, donde ‘puros’ sigue siendo 7, y posteriormente se le suma 1, con lo que si después volvemos a mostrar su valor, efectivamente es 8. Sin embargo, si indicamos el operador como prefijo, primero le suma 1 y después ejecuta la sentencia.

```
>> let puros = 7;
← undefined
>> console.log(puros++);
    7
← undefined
>> puros
← 8
>> console.log(++puros);
    9
```



## Variables constantes

Las variables constantes son variables como las que hemos visto hasta ahora con la particularidad de que su valor no puede cambiarse.

```
>> const edad = 18;
← undefined
>> edad = 50;
! ▶ TypeError: invalid assignment to const `edad' [Saber más]
<anonymous> debugger eval code:1
```

Si lo intentamos nos salta un error que nos indica que no se puede modificar su valor. Y entonces, ¿Para qué utilizan estas variables? Normalmente, utilizamos este tipo de variables cuando queremos que el valor de algo no se pueda modificar, por ejemplo como es el caso del número de meses que tiene un año, es algo que siempre va a ser así y no vamos a querer modificar.

```
>> const meses = 12;
← undefined
>> meses = 8;
! ▶ TypeError: invalid assignment to const `meses' [Saber más]
```