



# Introducción a JavaScript



¿Ya te cansaste de jugar con bolitas de colores? Tenemos una buena noticia. En este capítulo vamos a aprender **programación imperativa** de la mano de uno de los lenguajes de programación más utilizados de la industria del software.

JavaScript, usualmente abreviado Js, es muy parecido a Gobstones en su *sintaxis*. Es decir, en la forma en que se escribe . Pero es mucho más poderoso y está presente en casi todas las páginas que solés visitar.

Ya aprendimos a utilizar números y booleanos, en este capítulo también vamos a usar palabras y conjuntos para trabajar con muchos datos a la vez .

¿Ya te preparaste para salir del tablero?

¡Acompañanos!

---

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Funciones, definición

Gobstones y JavaScript tienen mucho en común. Por ejemplo, en ambos lenguajes podemos definir **funciones** y usarlas muchas veces.

Sin embargo, como siempre que aprendas un lenguaje nuevo, te vas a topar con un pequeño detalle: **tiene una sintaxis diferente**. La buena noticia es que el cambio no será tan terrible como suena, así que veamos nuestra primera función JavaScript:

```
function doble(numero) {  
  return 2 * numero;  
}
```

Diferente, pero no tanto. Si la comparás con su equivalente Gobstones...

```
function doble(numero) {  
  return (2 * numero)  
}
```

...notarás que los paréntesis en el `return` no son necesarios, y que la última línea la terminamos con `;`.

Veamos si se va entendiendo: definí ahora la función `mitad`, que tome un número por parámetro y retorne su mitad. Tené en cuenta que el operador de división en JavaScript es `/`.

💡 ¡Dame una pista!

 Solución  Consola

```
1 function mitad(numero) {  
2   return numero / 2;  
3 }
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Perfecto, ¿viste que no era tan terrible?

Si no le pusiste `;` al final de la sentencia habrás visto que funciona igual. De todas formas ponelo, ya que de esa manera evitamos posibles problemas.

Siempre que aprendamos un lenguaje nuevo vamos a tener que aprender una nueva sintaxis. Sin embargo y por fortuna, si tenés los conceptos claros, no es nada del otro mundo .

Aprendamos ahora a usar estas funciones.

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Funciones, uso



¿Y esto con qué se come? Digo, ehm.... ¿cómo ~~se~~ usan estas funciones? ¿Cómo hago para pasarles argumentos y obtener resultados?

3. Funciones, uso

Basta con poner el nombre de la función y, entre paréntesis, sus argumentos. ¡Es igual que en Gobstones!

```
doble(3)
```



Y además podemos usarlas dentro de otras funciones. Por ejemplo:

```
function doble(numero) {  
  return 2 * numero;  
}  
  
function siguienteDelDoble(numero) {  
  return doble(numero) + 1;  
}
```



O incluso mejor:

```
function doble(numero) {  
  return 2 * numero;  
}  
  
function siguiente(numero) {  
  return numero + 1;  
}  
  
function siguienteDelDoble(numero) {  
  return siguiente(doble(numero));  
}
```



Veamos si se entiende; definí las siguientes funciones:

- `anterior` : toma un número y devuelve ese número menos uno
- `triple` : devuelve el triple de un número

- `anteriorDelTriple`, que combina las dos funciones anteriores: multiplica a un número por 3 y le resta 1

 Solución  Consola

```
1 function anterior(numero) {  
2   return numero - 1;  
3 }  
4  
5 function triple(numero) {  
6   return numero * 3;  
7 }  
8  
9 function anteriorDelTriple(numero) {  
10  return anterior(triple(numero));  
11 }
```

 Enviar

 ¡Muy bien! Tu solución pasó todas las pruebas

Quizás ahora estés pensando: si no tengo un tablero, ¿cómo sé si mi función hace lo que debe?  
Acompañanos...

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Probando funciones



Quizás ya lo notaste pero, junto al editor, ahora aparece una ~~pestaña~~ pestaña nueva: la *consola*.

4. Probando funciones

La consola es una herramienta muy útil para hacer pruebas rápidas sobre lo que estás haciendo: te permite, por ejemplo, probar *expresiones*, funciones que vengan con JavaScript, o **incluso funciones que vos definas en el editor**.

La podés reconocer fácilmente porque arranca con un chirimbolito que se llama *prompt*.

Para entender mejor cómo funciona, te invitamos a explorarla.

Probá en la consola las siguientes expresiones:

- `4 + 5`
- `Math.round(4.5)`
- `funcionMisteriosa(1, 2, 3)` (ya la definimos por vos y la podés usar)

>



Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

Información importante

Términos y Condiciones

Reglas del Espacio de Consultas







# Haciendo cuentas

Además de los operadores matemáticos `+`, `-`, `/` y `*`, existen muchas otras funciones matemáticas comunes, algunas de las cuales ya vienen con JavaScript y están listas para ser usadas.

Sin embargo, la sintaxis de estas funciones matemáticas es *apenas* diferente de lo que veníamos haciendo hasta ahora: hay que prefijarlas con `Math.`. Por ejemplo, la función que nos sirve para redondear un número es `Math.round`:

```
function cuantoSaleAproximadamente(precio, impuestos) {  
  return Math.round(precio * impuestos);  
}
```

Probá en la consola las siguientes expresiones:

- `Math.round(4.4)`
- `Math.round(4.6)`
- `Math.max(4, 7)`
- `Math.min(4, 7)`

>



Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Poniendo topes

Hagamos un alto en nuestro camino y miremos las funciones `Math.max` y `Math.min`, que nos pueden ahorrar más trabajo del que parece.

6. Poniendo topes

Necesitamos una función que diga cuánta plata queda en tu cuenta (que tiene un cierto `saldo`) si extráes un cierto `monto`:

```
// el saldo es $100, el monto a extraer, $30
> extraer(100, 30)
70 //quedan $70 ($100 - $30 = $70)
```

Pero como no queremos quedarnos en negativo, si el monto a extraer es mayor al saldo, nuestro saldo debe quedar en cero.

```
> extraer(100, 120)
0 //Ups, quisimos sacar más plata de la que teníamos.
//Nos quedamos con $0
```

Como ves, esto es *casi* una resta entre `saldo` y `monto`, con la salvedad de que estamos poniendo un *tope inferior*: no puede dar menos de cero.

En otras palabras (¡preparate!, esto te puede volar la cabeza) `extraer` devuelve el máximo entre la resta `saldo - monto` y `0`.

¿Te animás a completar la solución que está en el editor?

💡 ¡Dame una pista!

 Solución  Consola

```
1 function extraer(saldo, monto) {
2   return Math.max(saldo - monto, 0);
3 }
```



▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

¡Bien hecho! Ahora andá y probalo en la consola

Como ves, la función `Math.max` nos sirvió para implementar un *tope inferior*. De forma análoga, la función `Math.min` nos puede servir para implementar un *tope superior*.

Ah, y si estás pensando “en Gobstones podría haber hecho esto con un *if*”, ¡tenés razón!. Pero esta solución es mucho más breve y simple .

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Libros de la buena memoria

¡Veamos más operadores! Dani ama el primer día de cada mes, y por eso definió esta función.

7. Libros de la buena memoria

```
function esDiaFavorito(diaDelMes) {  
  return diaDelMes === 1 ;  
}
```

...y la usa así (y la dejó en la biblioteca para que la pruebes):

```
> esDiaFavorito(13)  
false  
> esDiaFavorito(1)  
true
```

Como ves, en JavaScript contamos con operadores como `===`, `>=`, `>`, `<`, `<=` que nos dicen si dos valores son iguales, mayores-o-iguales, mayores, etc. Los vamos a usar bastante .

¡Ahora te toca a vos! Dani también dice que a alguien `leGustaLeer`, cuando la cantidad de libros que recuerda haber leído es mayor a 20. Por ejemplo:

```
> leGustaLeer(15)  
false  
  
> leGustaLeer(45)  
true
```

Definí y probá en la consola la función `leGustaLeer` .

 Solución  Consola

```
1 function leGustaLeer(libros_leidos) {  
2   return libros_leidos > 20 ;  
3 }
```



 Enviar ¡Muy bien! Tu solución pasó todas las pruebas

¡Bien hecho!

Capaz pasó desapercibido, pero `leGustaLeer` devuelve `true` o `false`, es decir, es una función que devuelve booleanos. Eso significa que en JavaScript, no sólo hay números sino que también..... hay booleanos .

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Booleanos

Ahora miremos a los booleanos con un poco más de detalle:

- Se pueden negar, mediante el operador `!`: `!hayComida`
- Se puede hacer la conjunción lógica entre dos booleanos (*and*, también conocido en español como *y lógico*), mediante el operador `&&`:  
`hayComida && hayBebida`
- Se puede hacer la disyunción lógica entre dos booleanos (*or*, también conocido en español como *o lógico*), mediante el operador `||`:  
`unaExpresion || otraExpresion`

Veamos si se entiende; definí las siguientes funciones:

- `estaEntre`, que tome tres números y diga si el primero es mayor al segundo y menor al tercero.
- `estaFueraDeRango`: que tome tres números y diga si el primero es menor al segundo o mayor al tercero

Ejemplos:

```
> estaEntre(3, 1, 10)
true
> estaEntre(90, 1, 10)
false
> estaEntre(10, 1, 10)
false
> estaFueraDeRango(17, 1, 10)
true
```

Solución Consola

```
1 function estaEntre(num_entre, num_menor, num_mayor) {
2   return num_entre > num_menor && num_entre < num_mayor;
3 }
4
5
6 function estaFueraDeRango(num_Fuera_Rango, num_menor, num_mayor) {
7   return num_Fuera_Rango < num_menor || num_Fuera_Rango > num_mayor;
8 }
9
```

Enviar

¡Muy bien! Tu solución pasó todas las pruebas

¡Bien hecho!

Ya fueron suficientes booleanos y cuentas por ahora, ¿no? Exploremos algo más interesante: los `string`s.

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)







# Palabras, sólo palabras

•

Muchas veces queremos escribir programas que trabajen con texto : queremos saber cuántas palabras hay en un libro, o convertir minúsculas a mayúsculas, o saber en qué parte de un texto está otro.

Para este tipo de problemas tenemos los *strings*, también llamados *cadena de caracteres*:

- `"Ahora la bebé tiene que dormir en la cuna"`
- `'El hierro nos ayuda a jugar'`
- `"¡Hola Miguel!"`

Como se observa, todos los strings están encerrados entre comillas simples o dobles. ¡Da igual usar unas u otras! Pero sé consistente: por ejemplo, si abriste comilla doble, tenés que cerrar comilla doble. Además, un string puede estar formado por (casi) cualquier carácter: letras, números, símbolos, espacios, etc.

¿Y qué podemos hacer con los strings? Por ejemplo, compararlos, como a cualquier otro valor:

```
> "hola" === "Hola"
false

> "todo el mundo" === "todo el mundo"
true
```

Veamos si queda claro: definí la función `esFinDeSemana` que tome un string que represente el nombre de un día de la semana, y nos diga si es `"sábado"` o `"domingo"`.

```
> esFinDeSemana("sábado")
true
> esFinDeSemana("martes")
false
```

💡 ¡Dame una pista!

 Solución Consola

```
1 function esFinDeSemana(dia) {  
2   return dia === "sábado" || dia === "domingo";  
3 }
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Operando strings

¿Y qué podemos hacer con los strings, además de compararlos? ¡Varias cosas! Por ejemplo, podemos preguntarles cuál es su cantidad de letras:

```
> longitud("biblioteca")
10
> longitud("babel")
5
```

O también podemos *concatenarlos*, es decir, obtener **uno nuevo** que junta dos strings:

```
> "aa" + "bb"
"aabb"
> "sus anaqueles " + "registran todas las combinaciones"
"sus anaqueles registran todas las combinaciones"
```

O podemos preguntarles si uno comienza con otro:

```
> comienzaCon("una página", "una")
true
> comienzaCon("la biblioteca", "todos los fuegos")
false
```

Veamos si queda claro: definí la función `longitudNombreCompleto`, que tome un nombre y un apellido, y retorne su longitud total, contando un espacio extra para separar a ambos:

```
> longitudNombreCompleto("Cosme", "Fulanito")
14
```

 Solución

 Biblioteca

 Consola

```
1 function longitudNombreCompleto(nombre, apellido) {
2   return longitud(nombre)+longitud(apellido)+1 ;
3 }
```

4

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# ¡GRITAR!

•

Una conocida banda, para agregar gritos varios a su canción, nos pidió definir la función `gritar`, que toma un string y lo devuelve en mayúsculas y entre signos de exclamación.

Por ejemplo:

```
> gritar("miguel")
"¡MIGUEL!"

> gritar("benito")
"¡BENITO!"
```

Definí la función `gritar`. Te dejamos para que uses la función `convertirEnMayuscula`, que, eh... bueno... básicamente convierte en mayúsculas un string.

💡 ¡Dame una pista!

 Solución  Biblioteca  Consola

```
1 function gritar(palabra) {
2   return "¡"+convertirEnMayuscula(palabra)+"!"
3 }
4
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





## ¿Y qué tal si...?

Ninguna introducción al lenguaje JavaScript estaría completa sin mostrar al menos una estructura de control que ya conocemos: la alternativa [condicional](#). Veamos un ejemplo: 12. ¿Y qué

```
//Equivalente a Math.abs
function valorAbsoluto(unNumero) {
  if (unNumero >= 0) {
    return unNumero;
  } else {
    return -unNumero;
  }
}
```

Veamos si se entiende: definí la función `maximo`, que funcione como `Math.max` (¡no vale usarla!) y retorne el máximo entre dos números. Por ejemplo, el máximo entre 4 y 5 es 5, y el máximo entre 10 y 4, es 10.

**Solución** [Biblioteca](#) [Consola](#)

```
1 //Equivalente a Math.abs
2 function maximo(num1,num2) {
3   if (num1 > num2) {
4     return num1;
5   } else {
6     return num2;
7   }
8 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# ¿De qué signo sos?

Ya utilizamos la alternativa condicional para realizar una acción específica cuando se cumple una condición y para cuando debemos elegir entre dos acciones diferentes (según se cumpla o no).

Pero... ¿Si necesitamos más de dos alternativas? Veamos un ejemplo:

Agus se olvida siempre de como tiene que cuidar sus plantas , por eso definió la función `cuidadoSegun(dia)` que le recuerda que los lunes tiene que fertilizarlas, los viernes las tiene que fumigar y el resto de los días las tiene que regar.

```
function cuidadoSegun(dia) {  
  if (dia === "lunes") {  
    return "fertilizar";  
  } else if (dia === "viernes") {  
    return "fumigar";  
  } else {  
    return "regar";  
  }  
}
```

¡Ahora te toca a vos! Definí la función `signo` , que dado un número nos retorne:

- 1 si el número es positivo
- 0 si el número es cero
- -1 si el número es negativo

💡 ¡Dame una pista!

**Solución** [Biblioteca](#) [Consola](#)

```
1 function signo(num1) {  
2   if (num1 > 0) {  
3     return 1;  
4   } else if (num1 === 0) {  
5     return 0;  
6   } else {  
7     return -1;  
8   }  
9 }
```

▶ Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas



[Términos y Condiciones](#)  
[Reglas del Espacio de Consultas](#)





# El retorno del booleano

Para cerrar, ahora que ya vimos cómo escribir la alternativa condicional, es momento de un pequeño recordatorio: si usás adecuadamente las expresiones booleanas, ¡no es necesario utilizar esta estructura de control!

Supongamos que queremos desarrollar una función `esMayorDeEdad`, que nos diga si alguien tiene 18 años o más. Una tentación es escribir lo siguiente:

```
function esMayorDeEdad(edad) {  
  if (edad >= 18) {  
    return true;  
  } else {  
    return false;  
  }  
}
```

Sin embargo, este `if` es totalmente innecesario, dado que la expresión `edad >= 18` ya es booleana:

```
function esMayorDeEdad(edad) {  
  return edad >= 18;  
}
```

Mucho más simple, ¿no?

Para Ema un número es de la suerte si:

- es positivo, y
- es menor a 100, y
- no es el 15.

Definí la función `esNumeroDeLaSuerte` que dado un número diga si cumple la lógica anterior. ¡No vale usar `if`!

```
1 function esNumeroDeLaSuerte(N_suerte) {  
2   return N_suerte > 0 && N_suerte < 100 && N_suerte !== 15 ;  
3 }
```



 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

En general, como regla práctica, si tenés ifs que devuelven `true` s o `false` s, probablemente lo estás haciendo mal . Y si bien *funcionará*, habrás escrito código innecesariamente complejo y/o extenso.

Recordá: ¡menos código, más felicidad!

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Los premios

El jurado de un torneo nos pidió la función `medallaSegunPuesto` que retorne la medalla que le corresponde a los primeros puestos, según la siguiente lógica:

- primer puesto: le corresponde `"oro"`
- segundo puesto: le corresponde `"plata"`
- tercer puesto: le corresponde `"bronce"`
- otros puestos: le corresponde `"nada"`

Ejemplo:

```
> medallaSegunPuesto(1)
"oro"
> medallaSegunPuesto(5)
"nada"
```

Definí, y probá en la consola, la función `medallaSegunPuesto`

 Solución  Biblioteca  Consola

```
1 function medallaSegunPuesto(puesto) {
2   if (puesto === 1) {
3     return "oro";
4   } else if (puesto === 2) {
5     return "plata";
6   } else if (puesto === 3) {
7     return "bronce";
8   } else {
9     return "nada";
10  }
11 }
12
```

 Enviar

✓ ¡Muy bien! Tu solución pasó todas las pruebas

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)





# Tipos de datos

Como acabamos de ver, en JavaScript existen números, booleanos y strings:

Tipo de dato	Representa	Ejemplo	Operaciones
Números	cantidades	4947	+, -, *, %, <, etc
Boolean	valores de verdad	true	&&, !, etc
Strings	texto	"hola"	longitud, comienzaCon, etc

Además, existen operaciones que sirven para todos los *tipos de datos*, por ejemplo:

- `==` : nos dice si dos cosas son iguales
- `!=` : nos dice si dos cosas son diferentes

**Es importante usar las operaciones correctas con los tipos de datos correctos**, por ejemplo, no tiene sentido sumar dos booleanos o hacer operaciones booleanas con los números. **Si usas operaciones que no corresponden, cosas muy raras y malas pueden pasar.**

Probá en la consola las siguientes cosas:

- `5 + 6` (ok, los números se pueden sumar)
- `5 == 6` (ok, todas las cosas se pueden comparar)
- `8 > 6` (ok, los números se pueden ordenar)
- `!true` (ok, los booleanos se pueden *negar*)
- `false / true` (no está bien, ¡los booleanos no se pueden dividir!)

>\_ Consola

◀/▶ Biblioteca

>



Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

### Información importante

## Términos y Condiciones

Reglas del Espacio de Consultas





# Datos de todo tipo

Uff, ¡vimos un montón de cosas! Aprendimos sobre la sintaxis de las funciones en JavaScript, los *tipos de datos* y sus operaciones, e incluso conocimos uno nuevo: los *strings*.

¡Para finalizar veamos algunos ejemplos!

- `4 + 4` vale `8`.
- `"4" + "4"` vale `"44"`.
- `"on" + "ce"` vale `"once"`.
- `true && false` vale `false`.
- `5 >= 6` vale `false`.

---

Esta guía fue desarrollada por Franco Bulgarelli bajo los términos de la [Licencia Creative Commons Compartir-Igual, 4.0](#).

© 2015-2022 Ikumi SRL

[Información importante](#)

[Términos y Condiciones](#)

[Reglas del Espacio de Consultas](#)

