

## Programación inicial

### **Introducción a la Programación utilizando el lenguaje Javascript.**

***Programación inicial.  
Tecnicatura Superior en Programación.  
UTN-FRA***

**Autor:** *Esp. Ing. Ernesto Gigliotti*

**Revisores:** *Mg. Mauricio Dávila*

*Versión : 5*



Esta obra está bajo una [Licencia Creative Commons Atribución-CompartirIgual 4.0 Internacional](http://creativecommons.org/licenses/by-sa/4.0/).

**Índice de temas****Introducción a la Programación.**

- Computadora.
- Programa.
- Lenguaje de programación.
- Compilación.
- Proceso de programación.
- Sistema Operativo.
- Lenguajes de programación interpretados.

**El Lenguaje Javascript**

- Programando Javascript con Chrome.
- Variables.
- Operadores aritméticos básicos.
- Salida de datos: Consola y Alert.
- Entrada de datos: Prompt.
- Ejercicios.

**Operaciones lógicas y control de flujo.**

- El tipo de dato boolean.
- Operadores relacionales.
- Más comparaciones.
- La condición "Y" y la condición "O".
- Sentencias de control de flujo.
  - Sentencia if-else.
  - Sentencia if-else-if.
  - Sentencia switch.
- Ejercicios.

**Bloques iterativos.**

- Sentencia while.
- Sentencia for.
- Sentencia break.
- Ejercicios.

**Funciones y operadores.**

- Funciones.
- Operadores Incremento y Decremento.
- Operadores de asignación.
- Ejercicios.

**Arrays.**

- Arrays de números.
- Arrays de Strings.
- Ejercicios.

**Bibliografía.**

## Programación inicial

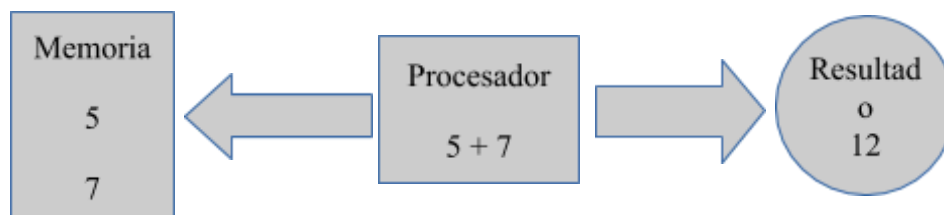
### Introducción a la Programación

Comenzaremos por definir qué es una computadora, y cómo podemos utilizarla para que realice tareas por nosotros.

### Computadora

Una computadora, es más que un dispositivo que nos permite navegar en Internet para acceder a redes sociales, escribir un texto o enviar e-mails. Para un programador, una computadora es un dispositivo capaz de ejecutar órdenes a una gran velocidad. ¿Qué son estas órdenes? No son más que cálculos entre ciertos valores. Sin entrar en grandes detalles, podemos dividir una computadora en dos partes: **una memoria de datos y un procesador**. En la memoria de datos se guardarán los datos con los que el procesador realizará cálculos.

Por ejemplo: Tenemos dos valores almacenados en la memoria : 5 y 7, el procesador tiene la capacidad de "leer" esos dos valores y realizar un cálculo con ellos, por ejemplo, sumarlos.



La pregunta es ¿cómo llega una computadora que solo puede sumar valores, a permitirme navegar por Internet, o ver una película o escuchar un mp3? La respuesta es que, obviamente, la única operación que realiza un procesador no es la suma, y que tampoco realiza una sola operación, sino **un conjunto ordenado de ellas**, y este conjunto es lo que se conoce como **un programa**.

### Programa.

*"Un programa informático es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora. Sin programas, estas máquinas no pueden funcionar."*

De modo que nuestra tarea como programadores, valga la redundancia, es escribir programas. Estos programas realizarán tareas, al comienzo más simples (desde sumar dos números hasta una simple calculadora), y luego más complejas.

Ejemplo de nuestro primer programa:

**ValorA = 5**  
**ValorB = 7**  
**Resultado = ValorA + ValorB**

Para explicar este pequeño programa, introduciremos el concepto de "Variable". Una variable, es una porción de memoria de nuestra computadora, en donde podremos almacenar un valor.

## Programación inicial

En nuestro programa, representamos estas porciones de memoria con un nombre definido por el programador. En este caso, tenemos 3 variables:

**ValorA**  
**ValorB**  
**Resultado**

En la primera se almacenará el valor 5, en la segunda el valor 7 y en la tercera, la operación "suma" entre las dos primeras.

Cabe destacar que en un programa, cada línea se ejecuta por el procesador en **orden descendente**, es decir:

1. **ValorA = 5**
2. **ValorB = 7**
3. **Resultado = ValorA + ValorB**

Al ejecutarse la primera línea, el procesador almacena el valor 5 en la porción de memoria llamada "ValorA":

Memoria de datos	
ValorA	5
ValorB	-
Resultado	-
...	...

Al ejecutarse la segunda línea, el procesador almacena el valor 7 en la porción de memoria llamada "ValorB":

Memoria de datos	
ValorA	5
ValorB	7
Resultado	-
...	...

Por último, al ejecutarse la tercer línea, el procesador realiza la operación suma entre los valores almacenados en las porciones de memoria "ValorA" y "ValorB" y almacena el resultado en la porción de memoria llamada "Resultado":

Memoria de datos	
ValorA	5
ValorB	7
Resultado	12
...	...

## Programación inicial

### Lenguaje de programación

Una vez comprendido qué es un programa, nos resta preguntarnos ¿cómo se escribe un programa? Es decir, no podemos pretender que una computadora nos entienda cuando le decimos:

```
ValorA = 5
ValorB = 7
Resultado = ValorA + ValorB
```

Porque del mismo modo que nosotros escribimos este programa sin ninguna consideración, otra persona podría haber escrito:

```
ValorA vale 5
ValorB vale 7
Resultado es igual a (ValorA mas ValorB)
```

Como podemos observar, necesitamos definir **ciertas reglas** de cómo escribir un programa, para que cualquiera que lo escriba, lo haga de la misma forma, y a su vez, esta forma tiene que poder ser interpretada por la computadora. Estas reglas de escritura, conforman un **lenguaje de programación**.

No existe sólo un lenguaje de programación, sino que a lo largo de la historia de la computación, se fueron desarrollando una gran cantidad de ellos (C/C++, Java, Python, Javascript, etc.) cada uno posee sus reglas de escritura, y el programador que escriba en cada lenguaje deberá respetarlas para que el programa funcione.

### Ejemplos de nuestro pequeño programa en algunos lenguajes de programación:

#### Javascript:

```
let valorA;
let valorB;
let resultado;

valorA=5;
valorB=7;
resultado=valorA + valorB;
```

#### Python:

```
valor_a = 5
valor_b = 7
resultado = valor_a + valor_b
```

#### PHP:

```
<?php
$valor_a = 5;
$valor_b = 7;
$resultado = $valor_a + $valor_b;
?>
```

## Programación inicial

C:

```
int main(int argc, char ** argv) {

    int valorA;
    int valorB;
    int resultado;

    valorA=5;
    valorB=7;
    resultado=valorA+valorB;

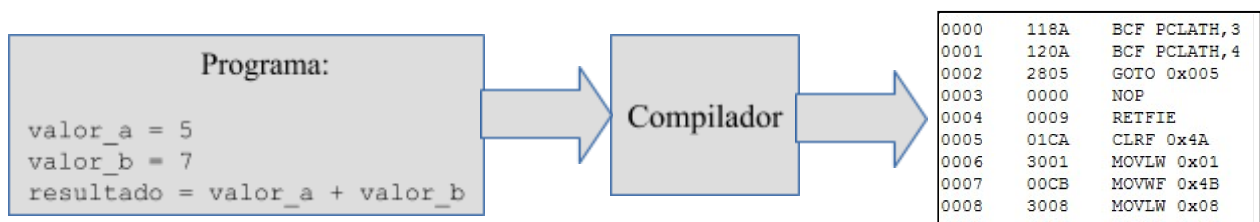
}
```

Cada lenguaje tiene su sintaxis y pueden diferir ampliamente, durante el curso se estudiará el lenguaje **JavaScript**, debido a su simplicidad.

### Compilación

Debido a la gran diferencia de sintaxis que poseen los diferentes lenguajes, nos preguntamos ¿cómo una computadora puede comprender las sentencias escritas? La respuesta es que no lo hace, el conjunto de órdenes que nosotros escribimos al programar, deben ser traducidos a un conjunto de órdenes que el procesador comprende. Este conjunto de órdenes es conocido como el **set de instrucciones del procesador**. Estas instrucciones no son más que números que el procesador sabe interpretar para saber qué operación debe realizar.

El “traductor” de nuestro lenguaje al set de instrucciones del procesador (también conocido como **lenguaje de máquina**) es un programa que se llama “Compilador”.



El compilador, transformará lo que el programador escribe en un conjunto de números que el procesador puede interpretar y ejecutar. También será el encargado de verificar que lo que el programador escribió cumple con las reglas del lenguaje, es decir, existe un compilador para cada lenguaje.

### Proceso de programación

Haremos un resumen de los pasos necesarios para obtener un programa que realice una tarea:

- 1) Elegir un lenguaje de programación para escribir nuestro programa.
- 2) Escribir el programa respetando las reglas del lenguaje elegido.
- 3) “Compilar” nuestro programa o “código” para obtener un archivo ejecutable que el procesador puede interpretar.

Este archivo que nos devuelve el compilador, será nuestro programa.

El programa obtenido, será diferente para cada tipo de procesador (X86, ARM, PowerPC, etc.) y también será diferente para cada **sistema operativo** (Windows, Linux, MAC OS)

## Programación inicial

### Sistema Operativo

*"Un sistema operativo (SO, frecuentemente OS del inglés Operating System) es un programa o conjunto de programas que en un sistema informático gestiona los recursos de hardware y provee servicios a los programas de aplicación\*, ejecutándose en modo privilegiado respecto de los restantes."*

Esto significa que un sistema operativo, es un gran programa base que nos permite ejecutar programas, dándoles a éstos acceso a las "partes" de una computadora, como por ejemplo la pantalla por la cual mostramos información, los discos rígidos en los que almacenamos información, la placa de red mediante la cual tenemos acceso a Internet, o el puerto USB mediante el cual podemos conectar un Pendrive y tener acceso a su información. Estas "partes" es lo que se conoce como **hardware**.



Nuestros programas, estarán al nivel de "Aplicación" mientras que en el nivel de "Sistema Operativo" se encontrará Windows, Linux, Android, o el sistema que estemos usando en nuestra computadora, y el "Hardware" no es más que todas las partes físicas de nuestra computadora a las cuales los programas tienen acceso para realizar tareas.

La segunda parte de la definición, dice:

*"provee servicios a los programas de aplicación\*, ejecutándose en modo privilegiado respecto de los restantes."*

Esto significa, que el Sistema Operativo administrará los diferentes programas que se ejecutan al mismo tiempo en una computadora, para que éstos no interfieran entre sí y todos puedan funcionar correctamente como si fueran el único programa que se está ejecutando.

\* Se llama programa de aplicación a los programas que interaccionan con el usuario.

## Programación inicial

### Lenguajes de programación interpretados

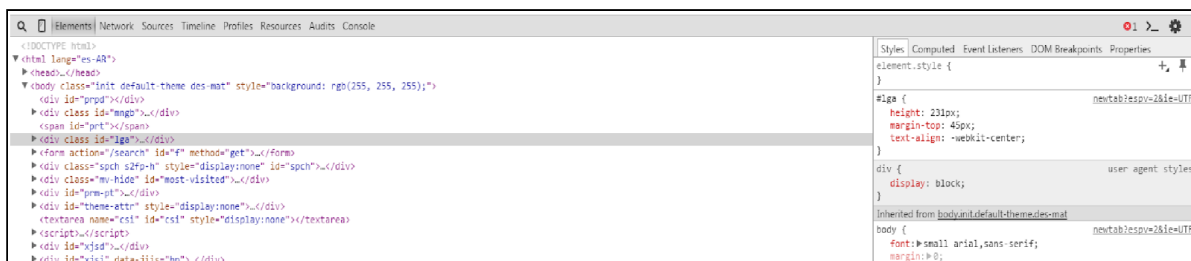
Previamente hablamos de un proceso llamado compilación, mediante el cual nuestro programa, escrito en un lenguaje en particular, se transforma en las instrucciones que el procesador es capaz de ejecutar. Existen una gran variedad de lenguajes que no se compilan, sino que son interpretados por un segundo programa, el cual sí es compilado. Este segundo programa se conoce como "Intérprete".

Si hablamos del lenguaje **Javascript**, podemos afirmar sin entrar en detalles, que el intérprete de este lenguaje es el navegador web (por ejemplo Chrome). Esto quiere decir que cuando nosotros escribimos un programa en Javascript, éste es interpretado y ejecutado por el navegador web, el cual es otro programa que se está ejecutando en el sistema operativo que posee instalado la computadora.



### Programando Javascript con Chrome

Simplemente abrimos el navegador, con una página en blanco, hacemos click derecho sobre la misma y en el menú que aparece elegimos "Inspeccionar elemento". Nos parecerá una ventana debajo o al costado de la página, la cual posee muchas lengüetas:



Seleccionamos la última, llamada "Console". Allí podremos escribir nuestro programa. Para probar su funcionamiento, por ejemplo escribimos:

```
console.log("Hola mundo");
```

Al presionar ENTER, se ejecutará la línea que escribimos y aparecerá por pantalla el mensaje. Más adelante explicaremos qué significa esta línea de programa.

A partir de aquí todos los ejemplos de programas que se mencionan, estarán escritos en lenguaje Javascript.



## Programación inicial

### Variables

Como se mencionó con anterioridad, una variable es una porción de memoria donde podemos almacenar valores. En Javascript, al definir una variable para poder utilizarla, es necesario utilizar la palabra reservada **var**:

```
let a;
```

### Tipos de valores

Números enteros.  
Números con coma.  
Texto.

La sintaxis para definir una variable es la siguiente:

**let nombre;**

Ejemplo:

```
let a;  
let valor;  
let numeroDecimal;
```

Después de definir las variables, podremos utilizarlas cargándole valores mediante el signo "="

```
a = 5;  
valor=7;  
numeroDecimal = 3.14;
```

Cabe destacar que la coma de los números con decimales se reemplaza por el punto y que todas las líneas terminan con ";"

También podemos crear la variable con un valor asignado en la misma línea:

```
let a = 5;
```

### Operadores aritméticos básicos

De la misma forma que sumamos dos valores y almacenamos el resultado en una variable, podemos realizar otras operaciones:

**+** : Suma  
**-** : Resta  
**\*** : Multiplicación  
**/** : División  
**%** : Resto

## Programación inicial

Las operaciones pueden combinarse como en una expresión matemática, utilizando paréntesis.

Ejemplo:

```
let a;  
let b;  
let resultado;  
  
a=5.5;  
b=2.5;  
  
resultado = (2*a + b) / 3 ;
```

Como puede observarse en el ejemplo, también pueden combinarse el uso de variables y literales (números constantes) es decir, poner  $2*a$  sabiendo que "a" vale 5.5, es lo mismo que poner  $2*5.5$  o en su defecto 11.

La ventaja de utilizar variables, es que el valor que queda almacenado en ella no es fijo, sino que puede cambiar, de modo que podemos utilizar una variable que tiene un valor que no conocemos, en un cálculo posterior:

Ejemplo:

```
a=5.5;  
b=2.5;  
resultado = (2*a + b) / 3 ;  
resultado = resultado + 2;
```

Al ejecutarse la tercera línea, la variable resultado se carga con el valor 4.5, al ejecutarse la cuarta línea se suma el contenido de "resultado" con 2 y se guarda el resultado de la operación en la propia variable "resultado" que era un operando de la suma realizada. De este modo el valor final de la variable resultado será 6.5

### Salida de datos: Consola y Alert

Si escribimos nuestro programa que suma dos números, y lo ejecutamos:

```
let a;  
let b;  
let resultado;  
  
a=5;  
b=7;  
resultado = a + b;
```

Podremos observar que no tenemos forma de darnos cuenta qué valor quedó almacenado en "resultado" ya que luego de almacenar el valor, el programa termina, sin indicarnos absolutamente nada.

Para mostrar mensajes por pantalla, por ejemplo el valor de la variable "resultado", para comprobar que la suma se haya realizado correctamente, recurriremos a llamar a una **función** la cual se encarga de realizar el trabajo de imprimir por pantalla por nosotros.

Debido a que el uso de **funciones**\* no es parte de esta introducción, les pedimos que por el momento sólo se utilice teniendo la idea general que imprimirá por pantalla variables que nosotros le indiquemos.

\*ver capítulo "Funciones y operadores"

## Programación inicial

Para imprimir un texto por pantalla:

```
console.log("Introduccion a Javascript");
```

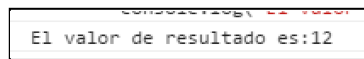
Para imprimir una variable, concatenamos al final del texto la variable con el símbolo "+":

```
console.log("El valor de resultado es:"+resultado);
```

De esta forma agregamos esta línea al final de nuestro programa, para saber qué valor contiene la variable "resultado":

```
let a;
let b;
let resultado;
a=5;
b=7;
resultado = a + b;
console.log("El valor de resultado es:"+resultado);
```

Al ejecutar el programa, aparecerá en la consola nuestro mensaje:



Si cambiamos los valores de las variables "a" o "b" veremos que el resultado es diferente:

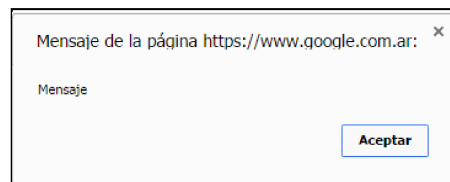
```
a=5;
b=10;
```



```
resultado = a + b;
```

También podemos utilizar un diálogo que nos muestra un mensaje, esperando que el usuario presione "Aceptar" para continuar con la ejecución de nuestro programa, para ello escribimos:

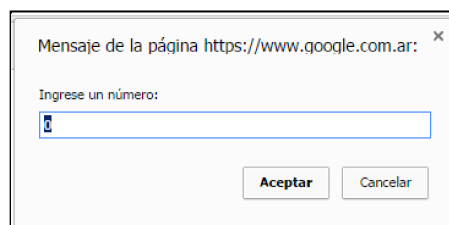
```
alert("Mensaje");
```



### Entrada de datos: Prompt

Podemos disparar un diálogo para que el usuario ingrese un texto o número utilizando la siguiente función:

```
let respuesta = prompt("Mensaje", "Texto ingresado por defecto");
```



## Programación inicial

Si por ejemplo escribimos el siguiente código, veremos el diálogo que indica la figura:

```
let a;  
a = prompt("Ingrese un número:", "0");  
console.log("El usuario ingreso:"+a);
```

Es importante destacar, que el valor devuelto (el que se almacena en "a") es un texto y no un número, por lo que no podemos por ejemplo sumarle otro número a la variable "a" sin antes convertirla en numérica, para ello utilizamos otra función:

```
a = parseInt(a); // Convertimos "a" a numérica
```

De esta forma si queremos imprimir el número que ingresó el usuario incrementado en 5 unidades (le sumamos 5) el programa sería el siguiente:

```
let a;  
a = prompt("Ingrese un número:", "0");  
a = parseInt(a); // Convertimos "a" a numérica  
a = a + 5;  
console.log("El valor incrementado es:"+a);
```

## Ejercicios

1. Crear un programa que defina una variable llamada "variable". Cargar la variable con el valor 5. Imprimir la variable por pantalla.
2. Encontrar los errores en el siguiente programa:

```
console.log("Hola mundo);
```

3. Crear un programa que defina tres variables llamadas "variable\_a", "variable\_b" y "variable\_resultado". Cargar las primeras dos con los valores numéricos 33 y 77. Sumar ambas variables y guardar el resultado en la variable "variable\_resultado". Imprimir el resultado por pantalla.
4. Crear un programa que defina 5 variables llamadas "a", "b"... "e" y una sexta variable llamada "promedio". Cargar las 5 variables con valores y calcular el promedio de los mismos. Imprimir el resultado por pantalla.

### Operaciones lógicas y control de flujo

#### El tipo de dato boolean

Imaginemos que existe una variable numérica pero en la cual el número más pequeño que podemos almacenar es "0" y el número más grande es "1", llamamos a este tipo de dato **"boolean"**. Este tipo sólo puede contener 2 valores.

Nos podemos preguntar para qué puede servir un tipo de dato de esta característica, y la respuesta es que existe todo un álgebra desarrollada para poder realizar operaciones con estos tipos de datos y es muy útil. A estas operaciones se las conoce como **operaciones lógicas** y al álgebra, **álgebra booleana**.

El tipo de dato boolean almacena dos valores, si nos abstraemos del lenguaje, podemos pensar en una variable booleana como algo en donde se almacenan dos estados bien diferenciados:

Blanco o Negro.  
Prendido o Apagado.  
Verdadero o Falso.  
Si o No.

En el lenguaje Javascript, estos dos valores no son números sino que se representan con las palabras reservadas **"true"** y **"false"**.

Si a una variable le asignamos los valores **"true"** o **"false"**, esta variable es una variable booleana.

```
let variableBoolean;
variableBoolean = true ;
```

Tampoco podemos utilizar las operaciones aritméticas que usábamos con los valores numéricos, es decir, no podemos ni sumar ni restar booleans sino que existen, como ya mencionamos antes, operaciones lógicas:

**Intersección** : Se realiza mediante el símbolo "&&"

**Unión** : Se realiza mediante el símbolo "||"

operación intersección o "and":

Variable 1		Variable 2		Resultado
false	&&	false	=	false
false	&&	true	=	false
true	&&	false	=	false
true	&&	true	=	true

operación unión o "or":

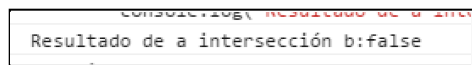
Variable 1		Variable 2		Resultado
false		false	=	false
false		true	=	true
true		false	=	true
true		true	=	true

## Programación inicial

Escribamos un programa que calcule la intersección entre dos variables booleanas:

```
let a;  
let b;  
let resultado;  
  
a = true ;  
b = false ;  
  
resultado = a && b ;  
  
console.log("Resultado de a intersección b:"+resultado);
```

Al ejecutar el programa, obtenemos como resultado "false", ya que la tabla mencionada anteriormente define que la operación intersección entre un valor "true" y uno "false", da como resultado "false":



## Operadores relacionales

Existen ciertos símbolos que nos permiten hacer comparaciones, estas operaciones dan como resultado valores del tipo boolean.

Si queremos saber si el valor almacenado en una variable es mayor que 5, utilizamos el operador relacional ">":

```
let variable;  
let resultado;  
  
variable = 8;  
  
resultado = variable > 5;  
  
console.log("Resultado de la comparación:"+resultado);
```

Si ejecutamos el programa, obtenemos que el resultado de la comparación de la variable (que contiene el valor 8) y el literal "5", da "true" (verdadero).

Si cambiamos la línea "variable=8;" por "variable=3;" y ejecutamos de nuevo el programa, obtenemos como resultado "false" (falso)

Notemos que el resultado de la operación "variable > 5" se asigna a una variable del tipo "boolean" ya que el resultado de una comparación, solo tiene dos valores posibles: o se cumple lo que estamos comparando, o no se cumple. En el caso de cumplirse, el resultado es **true**, y en el caso de no cumplirse, el resultado es **false**.

### Más comparaciones

¿Qué otras cosas podemos comparar? Podemos comparar si una variable es:

igual  
mayor  
menor  
distinto

que otra variable o literal. **Estas 4 comparaciones dan como resultado un valor del tipo boolean.**

¿Cómo comparamos por menor? Simplemente utilizando el símbolo en sentido contrario:

Comparación por mayor : ">"  
Comparación por menor : "<"

Ejemplo:

```
let variableA;
let variableB;
let resultado;
variableA = 3;
variableB = 5;
resultado = variableA < variableB;
console.log("Resultado de la comparación:"+resultado);
```

En este ejemplo, comparamos si "variableA" es menor que "variableB" de ser así, la variable "resultado" almacenará un "true", de lo contrario, almacenará un "false".

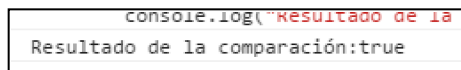
¿Cómo comparamos por igual? Mediante el símbolo "=="

¿Cómo comparamos por distinto? Mediante el símbolo "!="

Ejemplo:

```
let variableA;
let variableB;
let resultado;
variableA = 5;
variableB = 5;
resultado = variableA == variableB;
console.log("Resultado de la comparación:"+resultado);
```

Al ejecutar el código, da como resultado "true":



Otras operaciones relacionales:

Mayor igual : ">="

Menor igual : "<="

## Programación inicial

### La condición "Y" y la condición "O"

Imaginemos que tenemos 2 variables, cada una contiene la nota de un alumno:

```
let notaJuan;  
let notaPedro;  
notaJuan=8;  
notaPedro=8;
```

y queremos que nuestro programa calcule **si los 2 alumnos aprobaron** ( 7 o más ), para ello comparamos cada variable con la nota, con el literal "7", para saber si es mayor o igual a 7:

```
let notaJuan;  
let notaPedro;  
let resultadoJuan;  
let resultadoPedro;  
notaJuan=8;  
notaPedro=8;  
  
resultadoJuan = notaJuan >= 7;  
resultadoPedro = notaPedro >= 7;
```

Hasta aquí tenemos 2 variables donde cada una contiene el resultado de la comparación de la nota de cada alumno. Pero nosotros queremos saber **si los 2 aprobaron**, es decir, si para los 2 resultados, la comparación dio "true".

En otras palabras, necesitamos saber si "resultadoJuan" es igual a "true" **Y** "resultadoPedro" es igual a "true". Para ello utilizamos el **operador intersección**, el cual dará como resultado "true" si **todos** los miembros que computemos son true:

```
let notaJuan;  
let notaPedro;  
let resultadoJuan;  
let resultadoPedro;  
  
let resultado;  
  
notaJuan=8;  
notaPedro=9;  
  
resultadoJuan = notaJuan >= 7;  
resultadoPedro = notaPedro >= 7;  
  
resultado = resultadoJuan && resultadoPedro;  
  
console.log("Resultado de la comparación:"+resultado);
```

La variable "resultado" se cargará con "true" si y sólo si "resultadoJuan" es "true" y "resultadoPedro" es "true". Si alguna de las dos notas es menor a 7 ( y como consecuencia alguna de las dos variables resultado es "false" ) entonces "resultado" será "false".



## Programación inicial

**Siempre que nuestra condición implique que se cumpla una condición "Y" otra, utilizamos el operador intersección "&&"**

Ahora supongamos que queremos saber si alguno de los dos aprobó, en otras palabras: si "resultadoJuan" es "true" **O** "resultadoPedro" es "true". Para ello utilizamos el **operador unión**, el cual dará como resultado "true" si **alguno** de los miembros que computemos es "true":

```
resultado = resultadoJuan || resultadoPedro;
```

La variable "resultado" se cargará con "true" si "resultadoJuan" es "true" **O** "resultadoPedro" es "true". Si las dos notas son menores a 7 ( y como consecuencia las dos variables resultado son "false" ) entonces "resultado" será "false".

**Siempre que nuestra condición implique que se cumpla una condición o la otra, utilizamos el operador unión "||"**

### Modo abreviado

No es necesario definir una variable para guardar cada resultado de comparación y después hacer intersecciones o uniones entre sólo 2 operandos, sino que puede escribirse todo en menos líneas:

```
let notaJuan;
let notaPedro;
let resultado;

notaJuan=8;
notaPedro=9;

resultado = (notaJuan>=7) && (notaPedro>=7);

console.log("Resultado de la comparación:"+resultado);
```

Ahora el código parece más legible : **Si** "notaJuan" es mayor o igual a 7 **Y** "notaPedro" es mayor o igual a 7, **entonces** "resultado" será **true**.

### Sentencias de control de flujo

Si sólo pudiéramos escribir programas en donde se ejecutan una línea debajo de la otra sin control, no podríamos hacer que el programa se comporte de manera diferente según algunas condiciones que nosotros queramos, es por ello que existen sentencias de control de flujo, las cuales nos permitirán ejecutar o no, cierto bloque de programa según alguna condición.

#### Sentencia if-else

La sentencia "if" nos permite ejecutar un bloque de líneas de programa solo si se cumple cierta condición, de lo contrario, el bloque de líneas no se ejecutará.

Sintaxis:

## Programación inicial

**if** (CONDICIÓN)

```
{  
    bloque de líneas de código que se ejecutan si se cumple la condición.  
    ...  
}  
else  
{  
    bloque de líneas de código que se ejecutan si no se cumple la condición.  
    ...  
}
```

## Programación inicial

Las dos palabras reservadas que utilizamos son "if" y "else" y podemos distinguir dos bloques encerrados entre llaves, el primer bloque, se ejecutará si "CONDICIÓN" es igual a "true". Si es igual a "false", el bloque que se ejecutará será el segundo.

Ejemplo:

```
let nota;
let aprobo;

nota = 8;
aprobo = nota >= 7;

if(aprobo)
{
    console.log("El alumno está aprobado");
}
else
{
    console.log("El alumno está desaprobado");
}
```

Diagram illustrating the execution flow:

- 1er bloque** (1st block) is executed when the condition is true.
- 2do bloque** (2nd block) is executed when the condition is false.

En este ejemplo, realizamos la comparación de una variable que contiene el valor de la nota de un alumno, y la comparamos con el valor literal "7" en el caso de que la nota sea mayor o igual a "7" la variable boolean "aprobo" se cargará con "true", de lo contrario, con "false".

Como se explicó anteriormente, si la condición de la sentencia "if" es "true", como en este caso, (porque nota vale 8 y entonces "aprobo" vale "true") entonces se ejecutará el primer bloque encerrado entre llaves, y se saltará el segundo.

Si ejecutamos el programa, podemos observar que la línea que imprime por consola "El alumno está aprobado" se ejecuta (primer bloque) pero la línea que está dentro del segundo bloque, no.

En el caso de cambiar el valor de la nota por un número menor a 7, la variable "aprobo" se cargará con "false", y el primer bloque de nuestro if-else no se ejecutará, pero el segundo sí.

Con nota=5 , el resultado de la ejecución del programa es "El alumno está desaprobado"

### Evitando la variable boolean

Puede escribirse el mismo programa de una manera más reducida, sin utilizar la variable boolean, escribiendo dentro de los paréntesis del "if" directamente la comparación que da como resultado un valor boolean:

```
let nota;
nota = 8;

if(nota >= 7)
{
    console.log("El alumno está aprobado");
}
else
{
    console.log("El alumno está desaprobado");
}
```

## Programación inicial

El programa queda mucho más claro y se lee de la siguiente forma: **Si** nota es mayor o igual a 7, se imprime "El alumno está aprobado" **sino**, se imprime "El alumno está desaprobado".

### Sentencia if-else-if

Dentro de un bloque ( el de "if" o el de "else" ), o en ambos, puede volver a escribirse otro if-else.

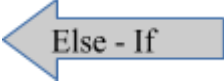
Supongamos que en caso de no aprobar, ( bloque "else" ) queremos indicar si el alumno puede recuperar ( si saco más de 4 ) o debe recursar la materia ( nota menor a 4 ). Escribimos el siguiente programa:

```
let nota;
nota = 5;

if(nota >= 7)
{
    console.log("El alumno está aprobado");
}
else
{
    if(nota >= 4)
    {
        console.log("El alumno está desaprobado y puede recuperar");
    }
    else
    {
        console.log("El alumno está desaprobado y debe recursar");
    }
}
```

Existe una forma más compacta de escribir un "if" que esta dentro de un "else":

```
let nota;
nota = 3;

if(nota >= 7)
{
    console.log("El alumno está aprobado");
}
else if(nota >= 4) 
{
    console.log("El alumno está desaprobado y puede recuperar");
}
else
{
    console.log("El alumno está desaprobado y debe recursar");
}
```

El else-if nos evita poner tantas llaves y el programa queda más compacto, sin dejar de comprenderse fácilmente.

Pueden colocarse tantos "else-if" como se deseen.

## Programación inicial

### Reglas a tener en cuenta:

Siempre que haya un "if" no es necesario que exista un "else"

Cuando el bloque de programa dentro de las llaves tiene una sola línea, las llaves pueden omitirse.

Si se usa un "else-if" tampoco es necesario escribir un último "else" como se escribió en el ejemplo.


Es importante aclarar que una vez que alguna de las condiciones de alguno de los "if" se cumple, y se ejecuta el bloque de código que le corresponde, **no se ejecutan el resto de los bloques**, y el programa continuará después de todos los bloques "if-else".

### Sentencia switch

La sentencia "switch" evalúa una variable numérica entera, y nos permite definir un bloque de líneas de programa a ejecutar, para cada valor que definamos, ejemplo:

Si tenemos una variable que indica el día de la semana, y queremos imprimir el nombre del día según el valor numérico, podemos hacer 7 "if" o utilizar el siguiente programa:

```
let dia;
dia = 2;
switch(dia)
{
    case 1 :
    {
        console.log("Domingo");
        break;
    }
    case 2 :
    {
        console.log("Lunes");
        break;
    }
    case 3 :
    {
        console.log("Martes");
        break;
    }
    case 4 :
    {
        console.log("Miércoles");
        break;
    }
    case 5 :
    {
        console.log("Jueves");
        break;
    }
    case 6 :
    {
        console.log("Viernes");
        break;
    }
    case 7 :
    {
        console.log("Sábado");
    }
}
```



En este ejemplo se ejecuta este bloque.

## Programación inicial

```
        break;  
    }  
}
```

El valor de la variable que coloquemos entre los paréntesis del switch, se comparará con los valores literales que coloquemos en las sentencias "case", si se cumple la igualdad, se ejecutará el bloque de código encerrado entre llaves que corresponde a ese "case". La sentencia "break" va al final de cada bloque "case" y evita que al terminar el bloque en cuestión, se siga ejecutando el que está debajo.

Al ejecutar este programa, la salida por consola será "Lunes"

### Ejercicios

1. Crear un programa que defina las variables "nota1" y "nota2". Asignarle valores a ambas. Comparar si "nota1" es mayor a "nota2" e imprimir el resultado de la comparación.
2. Crear un programa que defina las variables "nota1" y "nota2". Asignarle valores a ambas. Comparar si "nota1" es mayor a "nota2" imprimir el mensaje "Nota1 es mayor" o "Nota2 es mayor" según el resultado de la comparación.
3. Las puntuaciones de las películas se clasifican en:
  - 0 : mala
  - 1 : regular
  - 2 : buena
  - 3 : muy buena
  - 4 : excelente

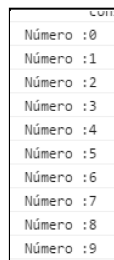
Definir una variable llamada "puntuacion" y cargarla con un valor de 0 a 4. Imprimir la clasificación de la película según el valor numérico.

### Bloques iterativos

Supongamos que queremos imprimir todos los números entre 0 y 9. Con los conocimientos adquiridos hasta el momento, podemos hacer lo siguiente:

```
console.log("Número :0");
console.log("Número :1");
console.log("Número :2");
console.log("Número :3");
console.log("Número :4");
console.log("Número :5");
console.log("Número :6");
console.log("Número :7");
console.log("Número :8");
console.log("Número :9");
```

Al ejecutar el programa, vemos por la consola:



```
Número :0
Número :1
Número :2
Número :3
Número :4
Número :5
Número :6
Número :7
Número :8
Número :9
```

También podemos escribir el programa utilizando una variable numérica para llevar el valor de la cuenta, e ir sumándole 1 cada vez que imprimimos una línea:

```
let contador;
contador=0;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
contador=contador+1;
console.log("Número :"+contador);
```

## Programación inicial

Al ejecutar el programa, vemos por la consola la misma información que la primera versión que escribimos.

Como se puede notar en nuestro segundo programa, simplemente escribimos dos líneas de programa y las repetimos 10 veces.

```
let contador;  
contador=0;  
console.log("Número :"+contador);  
contador=contador+1;  
console.log("Número :"+contador);  
contador=contador+1;  
console.log("Número :"+contador);  
contador=contador+1;
```

A medida que se ejecuta ese par de líneas una y otra vez, el valor contenido en la variable se va incrementando en 1, de modo que al imprimir el valor, vemos la cuenta ascendente deseada.

¿Qué ocurre cuando queremos imprimir hasta 100, o hasta 1000, o más? Sería impracticable repetir ese par de líneas de código 1000 veces. La solución consiste en utilizar una palabra reservada del lenguaje llamada **"while"** la cual nos permite ejecutar en forma repetida un bloque de líneas de programa.

### Sentencia while

Para utilizar un "bucle while" debemos escribir la palabra while, indicar entre paréntesis la condición (mientras sea true, se ejecutará el bloque de líneas de programa) y por último indicar entre llaves el bloque de líneas de código al cual le queremos repetir su ejecución mientras se cumpla la condición.

Ejemplo:

```
let contador;  
contador=0;  
while(contador<10)  
{  
    console.log("Número :"+contador);  
    contador=contador+1;  
}
```



## Programación inicial

En el ejemplo, utilizamos la palabra `while` y como hicimos con el `if`, entre paréntesis pusimos el cálculo de una comparación.

```
while(contador<10)
```

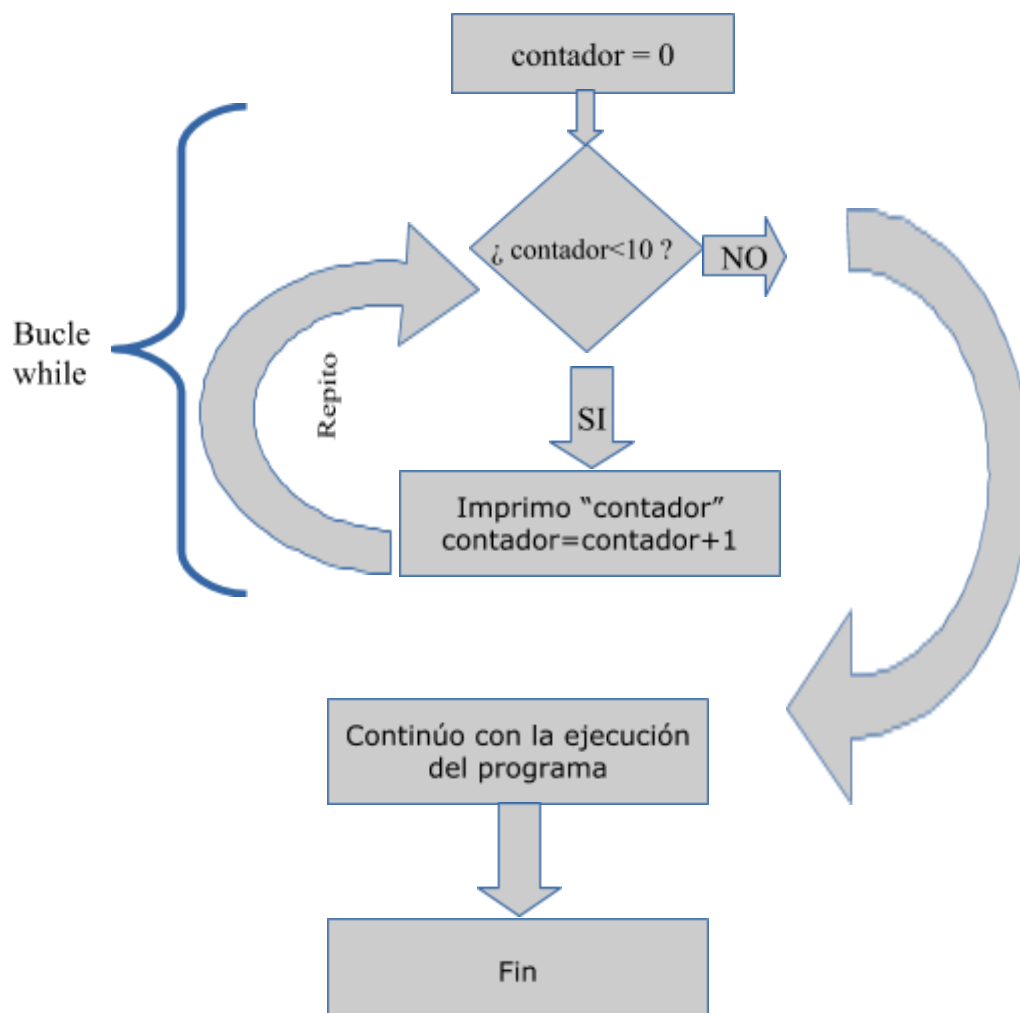
Si la comparación da como resultado `"true"` (verdadero) entonces se ejecutarán una vez, las líneas de programa que están entre las llaves:

```
{
    console.log("Número :"+contador);
    contador=contador+1;
}
```

Luego, se ejecutará la comparación nuevamente, teniendo en cuenta que la variable `"contador"` se incrementó de 0 a 1:

```
while(contador<10)
```

Como la comparación se sigue cumpliendo, se volverá a ejecutar el bloque, incrementando nuevamente la variable `contador`, de 1 a 2. Este ciclo se repetirá hasta que la variable `contador` alcance el valor 10, en donde la comparación `"contador<10"` dará como resultado `"false"` y no se ejecutará el bloque entre llaves, sino que la ejecución del programa continuará con la línea que está después del bloque `while` ( en este ejemplo no hay mas líneas de programa y éste termina ).



### Sentencia for

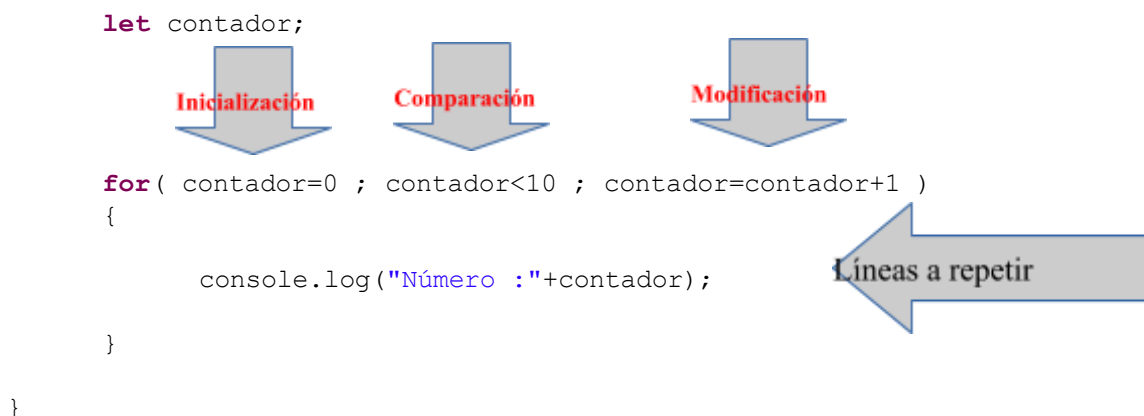
A pesar de que con el bucle "while" es suficiente para realizar cualquier tipo de algoritmo que requiera iteraciones y debido a que es muy común el tipo de bucle que vimos en el ejemplo, el cual contiene un contador que comienza en un valor el cual se va incrementando/decrementando con cada iteración, se define en el lenguaje otro tipo de bucle llamado "for".

En el bucle "for" la operación de inicializar el contador, la comparación y la modificación del contador queda definido entre los paréntesis:

#### Ejemplo:

```
for ( INICIALIZACIÓN ; COMPARACIÓN ; MODIFICACIÓN )
{
    Líneas de programa a repetir
}
```

Si reescribimos nuestro programa utilizando un bucle "for" en vez de un bucle "while" quedaría de la siguiente manera:



```
let contador;
for( contador=0 ; contador<10 ; contador=contador+1 )
{
    console.log("Número :"+contador);
}
}
```

La **inicialización** se ejecuta solo una vez.

La **comparación** es evaluada muchas veces ( una antes de cada iteración )

La **modificación** es ejecutada muchas veces ( una después de cada iteración )

## Programación inicial

### Sentencia break

Si en algún momento, en alguna iteración, queremos "romper" el bucle iterativo, como si se hubiera cumplido la condición de salida, utilizaremos la sentencia "**break**".

Esta sentencia puede utilizarse tanto en el bucle "while" como en el "for".

Ejemplo:

```
let contador;

for( contador=0 ; contador<10 ; contador=contador+1 )
{
    console.log("Número :"+contador);

    if(contador == 5)
    {
        break;
    }
}
```

Como se mencionó anteriormente, se realizarán 10 iteraciones en las cuales la variable "contador" irá incrementando su valor desde 0 hasta 10.

Al colocar una sentencia "if" preguntando si contador vale 5, y en el caso de cumplirse ejecutar la sentencia "break", el bucle de 10 iteraciones se verá interrumpido cuando "contador" valga 5 ejecutándose solo 5 iteraciones.

### Ejercicios

1. Crear un programa que imprima una cuenta regresiva de 59 a 0.
2. Crear un programa que imprima los números pares de 0 a 100.
3. Se escribió un programa para imprimir todos los números de 100 a 0 inclusive, en forma descendiente, pero el código contiene errores, encontrar los errores del programa y corregirlos:

```
let numero;

for (numero=100 , numero>0 , numero=numero+1)
{
    console.log("numero:"+numero);
}
```

4. Crear un programa que imprima los números de 0 a 100 indicando en cada uno de ellos si el número es divisible por 3 o no.
5. Indicar la diferencia en los siguientes programas. ¿Se comportan de igual forma? En el caso de no ser así, indicar la diferencia y la forma de corregirlos para que ambos se comporten de forma equivalente.

## Programación inicial

### Programa 1

```
let numero;

numero = 5;

while (numero > 0)
{
    console.log("numero:"+numero);
    numero--;
}
```

### Programa 2

```
let numero;

numero = 5;

while (true)
{
    console.log("numero:"+numero);
    numero--;
    if(numero < 0)
    {
        break;
    }
}
```

## Programación inicial

### Funciones

Recurramos al ejemplo del bucle que imprime los números del 0 al 10:

```
let contador;

for( contador=0 ; contador<10 ; contador=contador+1 )
{
    console.log("Número :"+contador);
}
```

Supongamos que a cada número generado le queremos calcular su valor elevado al cuadrado, es decir,  $y=x^2$ . Por ejemplo, si el número es dos, el resultado es  $2^2 = 4$ , si el número es 3, el resultado es  $3^2 = 9$ .

Podemos modificar nuestro programa, agregando una nueva variable que contenga el resultado de esta operación.

```
let contador;
let potencia;

for( contador=0 ; contador<10 ; contador=contador+1 )
{
    potencia = contador*contador;
    console.log("Número al cuadrado:"+potencia);
}
```

Supongamos ahora que nuestro cálculo no es sólo elelet un número al cuadrado sino que es mucho más complejo, como se ve en el siguiente ejemplo:

```
let contador;
let calculo;

console.log("Cálculos del 0 al 10");
for( contador=0 ; contador<10 ; contador=contador+1 )
{
    calculo = contador*contador;
    calculo = calculo + 5;
    calculo = ( calculo * 7 ) /2;

    console.log("Cálculo:"+calculo);
}
```

Cálculo  
complejo

## Programación inicial

Imaginemos ahora que tenemos que repetir este cálculo también para el número 33, de modo que deberemos escribir las líneas de programa para imprimir el resultado para este valor:

```
let contador;
let calculo;

console.log("Cálculos del 0 al 10");
for( contador=0 ; contador<10 ; contador=contador+1 )
{
    calculo = contador*contador;
    calculo = calculo + 5;
    calculo = ( calculo * 7 ) /2;

    console.log("Cálculo:"+calculo);
}
contador = 33;

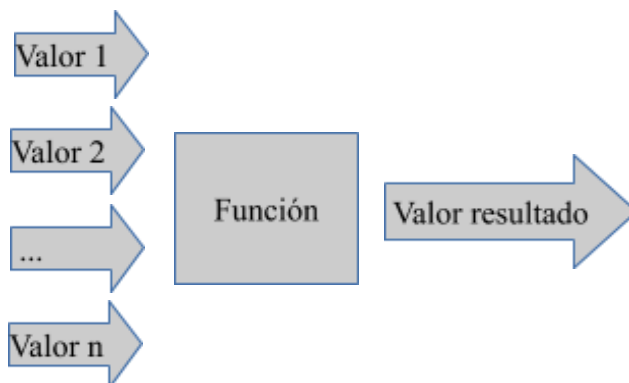
calculo = contador*contador;
calculo = calculo + 5;
calculo = ( calculo * 7 ) /2;

console.log("Cálculo para valor 33:"+calculo);
```

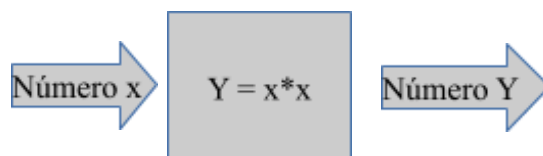
Como podrán observar, el programa puede volverse repetitivo y desprolijo, haciéndolo difícil de entender y modificar en futuras oportunidades.

Para evitar este problema, y poder reutilizar porciones de programa, se utilizan **funciones**.

Pensamos en una función como una caja negra en donde se ingresan ciertos valores, se procesan, y se devuelve otro valor.



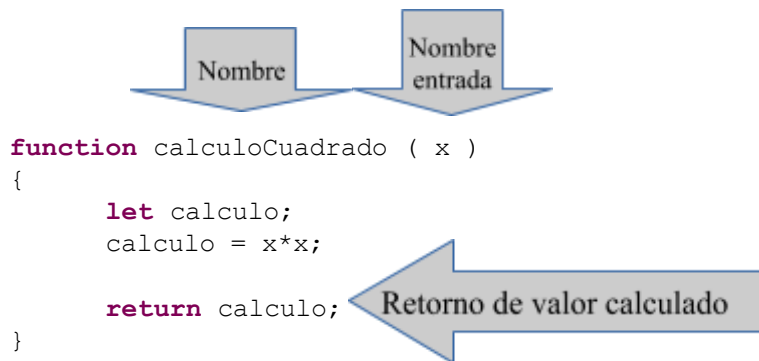
Si queremos crear una función que calcule el cuadrado de un número, el diagrama sería el siguiente:



Nuestra función recibirá el valor de entrada, el cual llamamos "x", calculará internamente el valor de  $x*x$ , y devolverá el resultado de la operación.

## Programación inicial

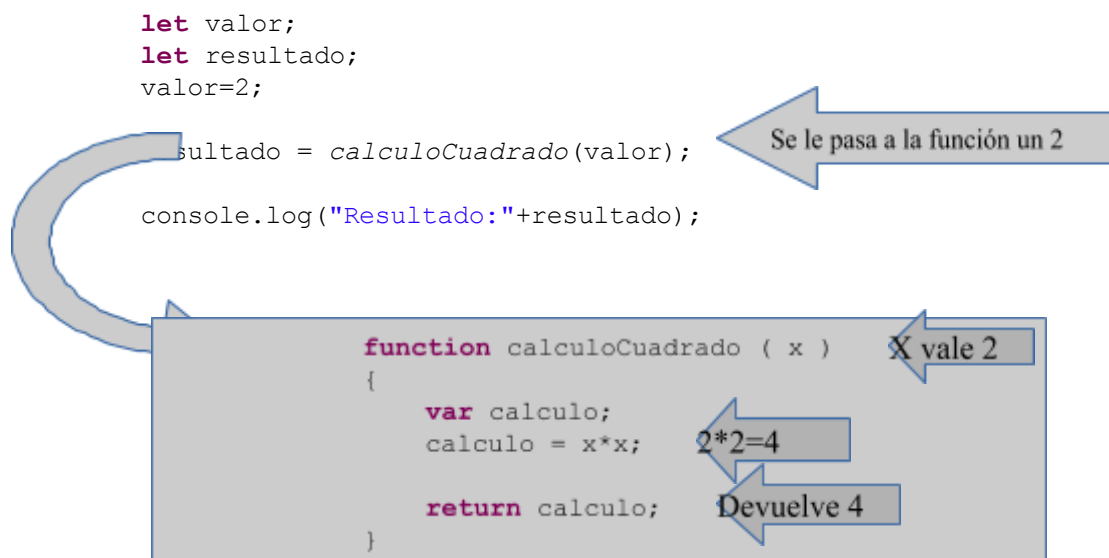
Creemos nuestra función en lenguaje Javascript.



Como puede observarse, los valores de entrada de la función se definen a la derecha, entre paréntesis, si hay más de uno, se separan con coma. Estas variables de entrada se llaman **argumentos de la función**.

Imaginemos a los argumentos como variables que definimos para usar, con la diferencia que ya van a estar cargadas con un valor, que es el valor de entrada que nos cargó quien ejecutó la función. Cuando llamamos a la misma, estamos obligados a pasarle valores, uno por cada argumento que ésta recibe.

Supongamos que queremos llamar a la función para que calcule el cuadrado de el número 2.



Se puede observar, que para llamar a la función y que se ejecute, colocamos el nombre de la misma, entre paréntesis colocamos las variables **cuyos valores se copiarán a los argumentos de la función** ( en este caso el valor 2 contenido en "valor" se copia en "x" ) y realizamos una asignación a la izquierda, en donde la variable "resultado" se cargará con el valor que retorne nuestra función ( en este caso, 4 ).

De este modo logramos encapsular un bloque de líneas de código dentro de una función, podremos llamar a la función tantas veces como queramos, sin tener que repetir ni una sola vez, las líneas de programa que colocamos dentro de ella. Esto produce que nuestro programa sea más ordenado, fácil de leer, mantener y modificar.

## Programación inicial

Miremos el ejemplo sobre el programa del bucle, para demostrar en lugar en el programa Javascript donde debe escribirse la función, y notar como el programa queda más legible.

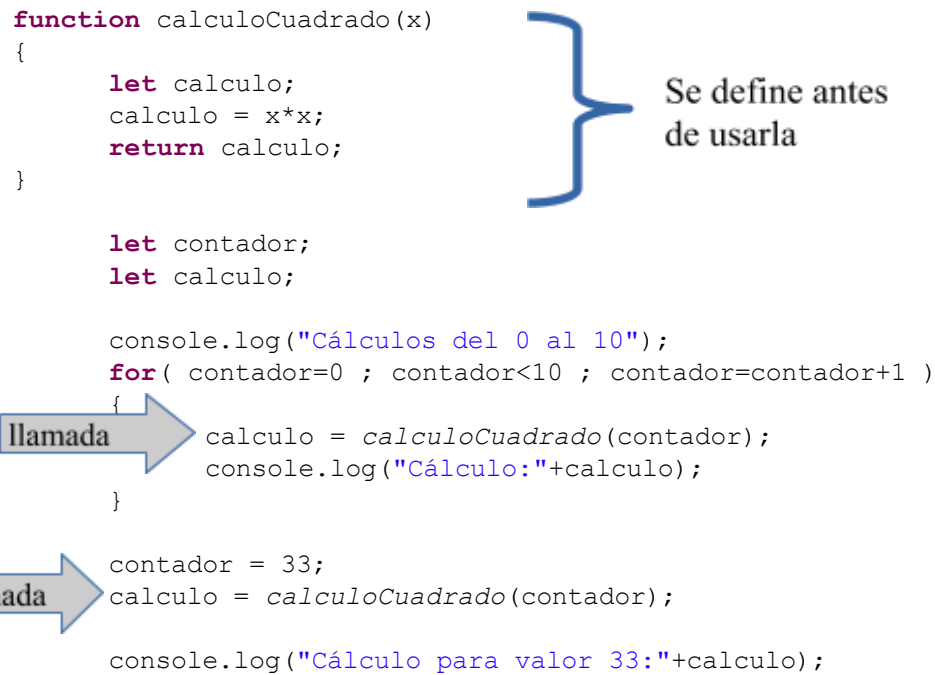
```
function calculoCuadrado(x)
{
    let calculo;
    calculo = x*x;
    return calculo;
}

let contador;
let calculo;

console.log("Cálculos del 0 al 10");
for( contador=0 ; contador<10 ; contador=contador+1 )
{
    calculo = calculoCuadrado(contador);
    console.log("Cálculo:"+calculo);
}

contador = 33;
calculo = calculoCuadrado(contador);

console.log("Cálculo para valor 33:"+calculo);
```



Si nos fijamos en la forma en que imprimimos mensajes por la consola, podemos darnos cuenta que estuvimos utilizando una función para hacerlo: **"log"** la cual recibe como argumento, una variable de texto.

Por ahora también ignoraremos que la función "log()" se llama con la palabra "console" separada por un punto.



## Programación inicial

### Operadores Incremento y Decremento

Además de los operadores suma y resta, existen 2 operadores más: Incremento y decremento, estos operadores nos permiten incrementar una unidad el valor de una variable.

#### Ejemplo:

Este programa:

```
let i;  
i=5;  
  
i=i+1;  
  
console.log("i vale:"+i);
```

es equivalente a este otro:

```
let i;  
i=5;  
  
i++;  
  
console.log("i vale:"+i);
```

Para ambos programas, el valor de "i" termina siendo 6, es decir que la línea **i=i+1** es equivalente a **i++**.

Esto nos introduce al operador incremento "++" el cual incrementa una unidad el valor de una variable.

De la misma manera, el operador decremento se escribe "--" y decrementa una unidad el valor de una variable:

```
let i;  
i=5;  
  
i--;  
console.log("i vale:"+i);
```

En este caso el valor de "i" termina siendo 4.

Recurramos al ejemplo del bucle for, para mostrar un uso práctico del operador incremento, en donde teníamos una variable "contador" que incrementábamos en uno por cada bucle:

```
let contador;  
for( contador=0 ; contador<10 ; contador=contador+1 )  
{  
    console.log("Número :"+contador);  
}
```

Utilizando el operador incremento, podría escribirse de la siguiente manera:

```
let contador;  
for( contador=0 ; contador<10 ; contador++ )  
{  
    console.log("Número :"+contador);  
}
```

### Resumen

Expresión:	Equivalente a:
i++	i=i+1
i--	i=i-1

### Operadores de asignación

Anteriormente, mencionamos un operador de asignación : el igual "=", pero éste no es el único operador de asignación que posee Javascript, ya que tenemos ciertos operadores que realizan una operación aritmética y luego una asignación en la misma línea, ejemplos:

Operador	Expresión:	Equivalente a:
+=	i += a	i = i + a
-=	i -= a	i = i - a
*=	i *= a	i = i * a
/=	i /= a	i = i / a
%=	i %= a	i = i % a

### Ejercicios

1. Crear un programa que tenga una función que calcule el perímetro de una circunferencia llamada "calcularPerimetro" y reciba el radio como argumento. Probar la función con todos los valores de radios enteros entre 1 y 10 e imprimir los resultados por pantalla.
2. Crear un programa que tenga una función que calcule el máximo entre 3 números enteros pasados como argumento y devuelva el mayor.
3. Crear un programa que tenga una función que calcule el área tanto de rectángulos como de triángulos, para ello, la función debe recibir 3 argumentos:
  - "flagTriangulo" : si recibe "true" se indica que es un triangulo, si recibe "false" se indica que es un rectángulo.
  - "base" : valor de la base del triangulo o rectángulo.
  - "altura" : valor de la altura del triangulo o rectángulo.

## Programación inicial

### Arrays de números

Supongamos que necesitamos almacenar los valores de la cantidad de horas extra que un empleado realiza por día. Deberemos crear una variable para cada día, en donde almacenaremos las horas extras que realiza el empleado para cada día:

```
let horasDomingo;
let horasLunes;
let horasMartes;
let horasMiercoles;
let horasJueves;
let horasViernes;
let horasSabado;
```

Como puede observarse, esto es engorroso de manejar, y trae como complicación adicional, por ejemplo, si queremos sumar las horas de todos los días, para calcular las horas extras semanales, deberemos sumar las 7 variables:

```
let horasSemanales;

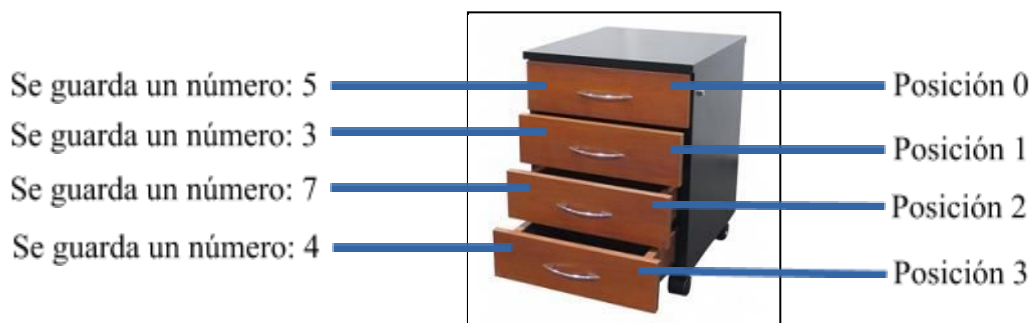
horasSemanales = horasDomingo + horasLunes + horasMartes +
                 horasMiercoles + horasJueves + horasViernes + horasSabado;
```

Existe una manera más práctica de resolver este problema, y es utilizando **arrays**.

Puede pensarse en un array como un mueble con cajones uno debajo del otro.



Cada cajón, almacenará un número o texto.  
Cada cajón posee una posición, llamada "índice" o "index".  
Las posiciones se numeran desde 0.  
La cantidad de cajones, será el tamaño del array.



## Programación inicial

Nuestra "cajonera", es equivalente al siguiente array:

Valor	Posición
5	0
3	1
7	2
4	3

En donde en la posición 0, tenemos guardado el valor 5, en la 1 el valor 3, en la 2 el valor 7 y en la 3 el valor 4. Este array tiene un tamaño de 4.

Para utilizar un array en Javascript, deberemos definirlo, al igual que una variable, y no es necesario declarar cuantos espacios va a tener el array ( el tamaño ).

Ejemplo, indicamos la variable y luego con corchetes "[]" indicamos que será un array.



```
let array = []; // definimos la variable array
```

Ahora ya tenemos nuestro array creado, pero vacío ¿Cómo accedemos al contenido de cada posición, ya sea para leer o cargar ésta con un valor?

Una vez definido el array, nos referiremos al contenido de una posición en particular, mediante la sintaxis:

```
array[posicion]
```

Esto quiere decir que para cargar un valor 5 en la posición 0 del array, escribimos:

```
array[0]=5;
```

Y para leer el contenido de la posición 3 y cargarlo en una variable "contenido", escribimos:

```
let contenido;
contenido = array[3];
```

Puede observarse que es bastante intuitivo de utilizar, donde antes teníamos una variable, ahora tenemos "array[posicion]".

Para definir el array de 4 posiciones del ejemplo, escribimos el siguiente código:

```
let array = []; // definimos la variable array
array[0]=5;
array[1]=3;
array[2]=7;
array[3]=4;
```

Si leemos un valor de una posición en donde nunca se cargó un valor:

```
let v = array[100];
console.log(v);
```

se imprimirá "undefined" por la consola.

## Programación inicial

Volvamos al problema principal, en donde teníamos que definir 7 variables, una para cada día de la semana, en donde teníamos que guardar las horas extra de un empleado.

En vez de hacer esto:

```
let horasDomingo;  
let horasLunes;  
let horasMartes;  
let horasMiercoles;  
let horasJueves;  
let horasViernes;  
let horasSabado;
```

Haremos esto:

```
let horasPorDia = []; // definimos la variable horasPorDia
```

En donde cada posición del array "horasPorDia" va a corresponder a un día:

Posición 0 : Aquí se guardan las horas extras para el día Domingo.  
Posición 1 : Aquí se guardan las horas extras para el día Lunes.  
Posición 2 : Aquí se guardan las horas extras para el día Martes.  
Posición 3 : Aquí se guardan las horas extras para el día Miércoles.  
Posición 4 : Aquí se guardan las horas extras para el día Jueves.  
Posición 5 : Aquí se guardan las horas extras para el día Viernes.  
Posición 6 : Aquí se guardan las horas extras para el día Sábado.

Si queremos cargar 6 horas extra en el día Martes, escribiremos:

```
horasPorDia[2] = 6 ; // 6 horas para el martes
```

Si queremos obtener las horas extras cargadas para el día Jueves, escribiremos:

```
let horasJueves;  
horasJueves = horasPorDia[4]; // leo las horas del jueves
```

¿Cómo obtenemos las horas extra semanales? Utilizando un bucle para ir leyendo los valores del array y sumarlos.

```
int posicionDia;  
int horasSemanales;  
  
horasSemanales=0;  
  
for(posicionDia=0 ; posicionDia<7 ; posicionDia=posicionDia+1)  
{  
    horasSemanales = horasSemanales + horasPorDia[posicionDia];  
}  
console.log("Las horas extra semanales son:"+horasSemanales);
```

En el ejemplo, utilizamos un bucle "for" para iterar 7 veces una línea de programa, la cual suma la variable "horasSemanales" con el valor de horas tomado del array **para cada día**, debido a que la variable entre corchetes "posicionDia", toma los valores de 0 a 6. De esta manera podemos leer del array los valores de cada posición.

**NOTA:** Debemos asegurarnos de que las 7 posiciones tengan valores cargados previamente.

**Arrays de Strings**

De la misma forma que definimos un array de números, definiremos un array de Strings

Ejemplo:

```
let dias = [];  
  
dias[0] = "Domingo";  
dias[1] = "Lunes";  
dias[2] = "Martes";  
dias[3] = "Miércoles";  
dias[4] = "Jueves";  
dias[5] = "Viernes";  
dias[6] = "Sábado";  
  
let i;  
for(i=0 ; i<7 ; i=i+1)  
{  
    console.log("Día: "+dias[i]);  
}
```

En este ejemplo, cargamos el array de Strings con los textos de los nombres de las días, luego al iterar con un bucle "for", leemos los textos del array y los imprimimos por la consola.

**Ejercicios**

1. Crear un programa en donde el usuario ingresa números y se guardan en un array. Al ingresar "-1" se imprimirán los números ingresado junto con sus posiciones dentro del array.
2. Crear un programa en donde el usuario ingresa números hasta que ingresa "-1", en dicho caso el programa termina e imprime el promedio de los números ingresados.
3. Crear un programa en donde el usuario ingresa 5 números. Al finalizar, el programa imprime los números ordenados de mayor a menor.
4. Crear un programa en donde el usuario ingrese el nombre y el precio de 5 productos, al finalizar, se listarán en pantalla el nombre y el precio de los productos, indicando con un "\*" el que posee el precio menor.
5. Crear un programa que contenga una función capaz de recibir un array de números como argumento, y devuelva un array invertido. Por ejemplo, si el array de entrada es [3,7,2] el array de salida es [2,7,3].

## **Programación inicial**

### **Bibliografía**

Tanenbaum, A. (1992) Modern Operating Systems, Englewood Cliffs: Prentice-Hall

JavaScript Arrays. (s.f.). Descargado de [http://www.w3schools.com/js/js\\_arrays.asp](http://www.w3schools.com/js/js_arrays.asp)

JavaScript Popup Boxes. (s.f.). Descargado de [http://www.w3schools.com/js/js\\_popup.asp](http://www.w3schools.com/js/js_popup.asp)