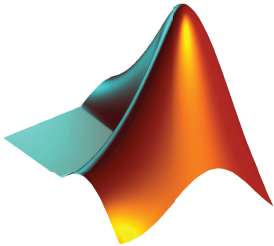


Programación en MATLAB



EMat

Escuela de
Matemática

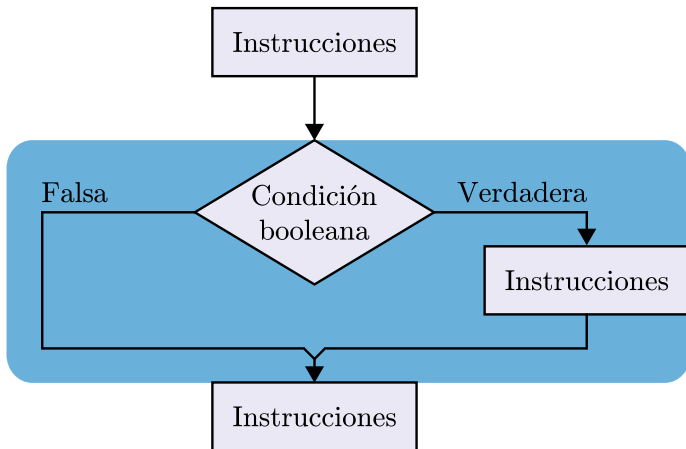
Filánder Sequeira Chavarría

Última actualización: 8 de agosto de 2020

Organización de la presentación

- 1 Estructuras de selección
- 2 Estructuras de repetición
- 3 Funciones definidas por el usuario

La instrucción “si” (**if**)



Instrucción **if**

La instrucción **if** tiene dos partes:

- La condición que se está evaluando. Esta puede ir o no entre paréntesis `()`.
- La o las instrucciones que van a ejecutarse si la condición es verdadera. Estas van entre el comando **if** y el comando **end**. La expresión **end**, indica el cierre del **if**.

```
if condición booleana
  instrucción 1
  instrucción 2
  .
  .
  .
  instrucción  $n$ 
end
```

Operadores de relaciones e igualdad

Operador	Significado
$a == b$	Igualdad ($=$)
$a \neq b$	Diferente (\neq)
$a < b$	a menor que b
$a > b$	a mayor que b
$a \leq b$	a menor o igual que b
$a \geq b$	a mayor o igual que b

Operadores lógicos

MATLAB proporciona operadores lógicos que se usan para simular los conceptos Y, O y NO del álgebra booleanda. Estos se conocen con el nombre de conjunción (\wedge), disyunción (\vee) y negación (\neg), respectivamente. Los operadores correspondientes son: $\&\&$, $||$ y \sim .

Operadores lógicos

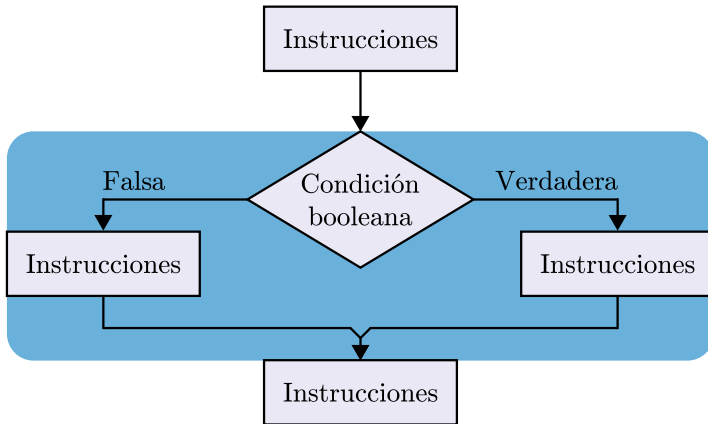
MATLAB proporciona operadores lógicos que se usan para simular los conceptos Y, O y NO del álgebra booleana. Estos se conocen con el nombre de conjunción (\wedge), disyunción (\vee) y negación (\neg), respectivamente. Los operadores correspondientes son: `&&`, `||` y `~`.

P	Q	P && Q	P Q	~P
true	true	true	true	false
true	false	false	true	false
false	true	false	true	true
false	false	false	false	true

Ejemplo

```
a = -3;  
  
if a > 0  
    disp('Número positivo');  
end  
  
if a < 0  
    disp('Número negativo');  
end
```

La instrucción “si-sino” (**if-else**)



Instrucción **if-else**

Esta instrucción tiene tres partes:

- La condición booleana que se está evaluando, la cual puede ir entre paréntesis `()`.
- La o las instrucciones que van a ejecutarse si la condición es verdadera. Entre el **if** y la instrucción **else**.
- La o las instrucciones que van a ejecutarse si la condición es falsa. Entre el **else** y el **end**.

```
if condición booleana
    instrucción 1
    .
    .
    .
    instrucción  $m$ 
else
    instrucción 1
    .
    .
    .
    instrucción  $n$ 
end
```

Ejemplo

```
a = 0;

if a > 0
    disp('Número positivo');
else
    if a < 0
        disp('Número negativo');
    else
        disp('Número es cero');
    end
end
```

Ejemplo

```
a = 12121212121212;  
  
if mod(a, 3) == 0  
    disp('Es múltiplo de 3');  
else  
    disp('No es múltiplo de 3');  
end
```

Tabulaciones

- Las tabulaciones permiten tener un código más ordenado. Lo cual permite que se pueda entender este de una mejor manera, dado que con ellos se ve claramente, por ejemplo, dónde cierra cada instrucción **if**.
- Un buen programador, está acostumbrado al uso de las tabulaciones y de los comentarios.

La instrucción **if-else** anidados

Es cuando se tienen varias instrucciones **if** juntas, tal y como se muestra a continuación:

```
a = 0;

if a > 0
    disp('Número positivo');
else
    if a < 0                                % Esto
        disp('Número negativo');          % es un
    else                                    % if
        disp('Número es cero');           % anidado
    end                                    %
end
```


La instrucción **if-else** anidados

Se pueden acomodar estos **if** poniendo juntos el **if** y el **else**:

```
a = 0;

if a > 0
    disp('Número positivo');
elseif a < 0
    disp('Número negativo');
else
    disp('Número es cero');
end
```

```
if condición booleana 1
    instrucciones
elseif condición booleana 2
    instrucciones
elseif condición booleana 3
    instrucciones
else % Cuando no se cumplen 1, 2 y 3
    instrucciones
end
```

Ejercicio

Ejercicio

Escriba un programa que tenga un número x entero, y muestre un mensaje con la siguiente información sobre x :

```
'El número es positivo y par.'  
'El número es positivo e impar.'  
'El número es negativo y par.'  
'El número es negativo e impar.'  
'El número es cero.'
```

Observación

¿Cómo se pregunta en MATLAB si el número x pertenece al intervalo $]a, b[$?

$a < x < b$	$a < x \ \&\& \ x < b$
-------------	------------------------

Puede verificarse esto, ejecutando en MATLAB la instrucción:

$1 < -3 < 2$

Observación

¿Cómo se pregunta en MATLAB si el número x pertenece al intervalo $]a, b[$?

$a < x < b$	$a < x \ \&\& \ x < b$
-------------	------------------------



Puede verificarse esto, ejecutando en MATLAB la instrucción:

`1 < -3 < 2`

Observación

¿Cómo se pregunta en MATLAB si el número x pertenece al intervalo $]a, b[$?

$a < x < b$	$a < x \ \&\& \ x < b$
-------------	------------------------



 

Puede verificarse esto, ejecutando en MATLAB la instrucción:

`1 < -3 < 2`

Observación

¿Cómo se pregunta en MATLAB si el número x pertenece al intervalo $]a, b[$?

$a < x < b$	$a < x \ \&\& \ x < b$
	

Puede verificarse esto, ejecutando en MATLAB la instrucción:

`1 < -3 < 2`

Ejercicio

Ejercicio (para la casa)

Sean $a, b \in \mathbb{R}$, tales que $a < b$, y $f(x) := \sin\left(e^{\frac{x}{2}}\right)$. Si se cumple que $f(a)f(b) \leq 0$ (como es el caso particular de $a = 0$ y $b = 3$), entonces al dividir el intervalo $[a, b]$ en tres partes iguales, tal y como se muestra:



con $p := \frac{2a+b}{3}$ y $q := \frac{a+2b}{3}$, se debe cumplir que $f(a)f(p) \leq 0$, $f(p)f(q) \leq 0$, o bien, $f(q)f(b) \leq 0$.

Escriba una función en MATLAB que reciba a y b , y que, luego de verificar que en efecto $f(a)f(b) \leq 0$, determine cuál de los tres subintervalos mantiene esta desigualdad.

Organización de la presentación

- 1 Estructuras de selección
- 2 Estructuras de repetición
- 3 Funciones definidas por el usuario

Repetición

- Los seres humanos realizan tareas repetitivas como los son: comer, dormir y trabajar. Análogamente, las computadoras también llevan a cabo ciertas tareas repetitivas, por ejemplo, buscar en archivos cierta información deseada.
- En programación, una repetición se conoce como ciclo.
- MATLAB cuenta con dos instrucciones para realice repeticiones: **while**, **for**.

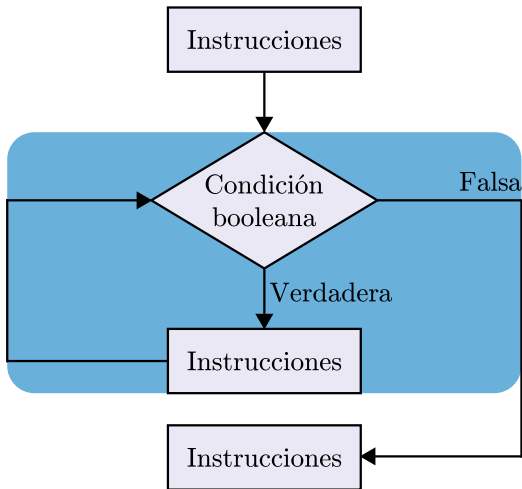
Repetición

- Los seres humanos realizan tareas repetitivas como los son: comer, dormir y trabajar. Análogamente, las computadoras también llevan a cabo ciertas tareas repetitivas, por ejemplo, buscar en archivos cierta información deseada.
- En programación, una repetición se conoce como ciclo.
- MATLAB cuenta con dos instrucciones para realice repeticiones: **while**, **for**.

Repetición

- Los seres humanos realizan tareas repetitivas como los son: comer, dormir y trabajar. Análogamente, las computadoras también llevan a cabo ciertas tareas repetitivas, por ejemplo, buscar en archivos cierta información deseada.
- En programación, una repetición se conoce como ciclo.
- MATLAB cuenta con dos instrucciones para realice repeticiones: **while**, **for**.

La instrucción “mientras que” (**while**)



```
while condición booleana
    % instrucciones a repetir
    % mientras que la condición
    % booleana sea verdadera
end
```

La instrucción `while`

- Se repiten las instrucciones dentro del `while`, mientras que la condición booleana sea verdadera, en caso contrario, se detiene el ciclo.
- Se debe garantizar que las instrucciones de repetición paren. Es decir, es indispensable que en algún momento la condición booleana cambie su valor de `true` a `false`.

La instrucción `while`

- Se repiten las instrucciones dentro del `while`, mientras que la condición booleana sea verdadera, en caso contrario, se detiene el ciclo.
- Se debe garantizar que las instrucciones de repetición paren. Es decir, es indispensable que en algún momento la condición booleana cambie su valor de `true` a `false`.

Ejemplo

```
while true
    % Este ciclo no para
    disp('Hola');
end
```

Este es un ciclo infinito.

Ejemplo

```
while true
    % Este ciclo no para
    disp('Hola');
end
```

Este es un ciclo infinito.
El proceso se “mata” oprimiendo: Ctrl+C.

Ejemplo

Escribir un programa, que calcule el factorial de un número natural n , denotado $n!$, donde:

$$n! = n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 3 \cdot 2 \cdot 1.$$

Es decir, se debe obtener que:

$$4! = 24$$

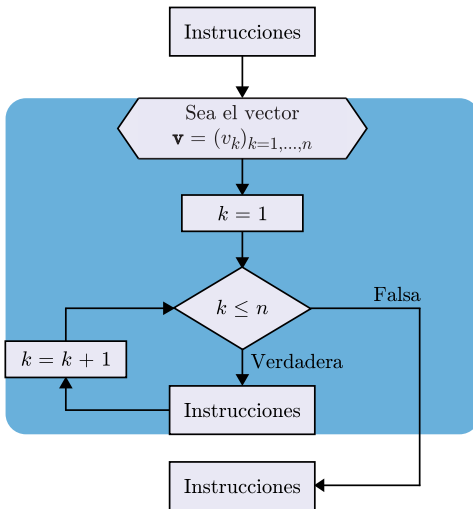
$$5! = 120$$

$$10! = 3628800$$

Solución

```
n = 10;  
% Cálculo del factorial de n:  
prod = 1; % Almacena los productos efectuados  
cont = 1; % Controla los valores a multiplicar  
while cont <= n  
    prod = prod * cont;  
    cont = cont + 1;  
end  
prod
```

La instrucción “para” (**for**)



La instrucción `for`

- Recorre un vector dado, componente por componente. Para cada una de las entradas repite un conjunto de instrucciones.
- Un ciclo `for` puede ser implementado a través de un ciclo `while`.
- Nótese que en un ciclo `for` la condición de parada (o bien, la cantidad de veces que se repiten las instrucciones) es dada por el tamaño del vector.

La instrucción **for**

- Recorre un vector dado, componente por componente. Para cada una de las entradas repite un conjunto de instrucciones.
- Un ciclo **for** puede ser implementado a través de un ciclo **while**.
- Nótese que en un ciclo **for** la condición de parada (o bien, la cantidad de veces que se repiten las instrucciones) es dada por el tamaño del vector.

La instrucción **for**

- Recorre un vector dado, componente por componente. Para cada una de las entradas repite un conjunto de instrucciones.
- Un ciclo **for** puede ser implementado a través de un ciclo **while**.
- Nótese que en un ciclo **for** la condición de parada (o bien, la cantidad de veces que se repiten las instrucciones) es dada por el tamaño del vector.


```
for k = vectorDeValores  
    % instrucciones a repetir  
end
```

o bien, en particular:

```
for k = 1 : n  
    % instrucciones a repetir  
end
```

Además de los ejemplos anteriores, un caso más general para la variable del ciclo, es el siguiente:

```
for k = valorInicial : incremento : valorFinal  
    % instrucciones  
end
```

Ejemplo

```
for k = 5 : -0.1 : -5  
    k  
end
```

produce el resultado:

```
k =  
    5  
k =  
  4.9000  
k =  
  4.8000  
.  
.  
.  
k =  
   -5
```

Ejemplo del factorial

```
n = 10;

prod = 1;
for cont = 1 : n
    prod = prod * cont;
end
disp(['El factorial de ' num2str(n)...
      ' es: ' num2str(prod)])
```

Ejercicio

Ejercicio

Programar la sucesión de Fibonacci:

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2} \end{cases}$$



que permita obtener el término n de la sucesión.

Sumas y productos

$$S = \sum_{i=m}^n a_i \Big\} \equiv \left\{ \begin{array}{l} S = 0; \\ \textbf{for } k = m : n \\ \quad S = S + a(k); \\ \textbf{end} \end{array} \right.$$

$$P = \prod_{i=m}^n a_i \Big\} \equiv \left\{ \begin{array}{l} P = 1; \\ \textbf{for } k = m : n \\ \quad P = P * a(k); \\ \textbf{end} \end{array} \right.$$

Ejercicio

Ejercicio

Similar a lo anterior, complete:

$$S = \left\{ \sum_{i=m_1}^{n_1} \sum_{j=m_2}^{n_2} a_{ij} \right\} \equiv \left\{ \right.$$

Ejercicio

Ejercicio (para la casa)

Dada una matriz $\mathbf{A} \in \mathbb{R}^{m \times n}$ fija, escriba instrucciones en MATLAB que permitan calcular el siguiente valor:

$$S(\mathbf{A}) := \sum_{i=1}^m \sum_{\substack{j=1 \\ j \neq i}}^n a_{ij}.$$

Organización de la presentación

- 1 Estructuras de selección
- 2 Estructuras de repetición
- 3 Funciones definidas por el usuario

Definición de funciones

En MATLAB para declarar una función, la estructura básica consiste de:

```
function [variables a retornar] = nombre(  
    parámetros)  
    .  
    .  
    .  
end
```

Definición de funciones

- **nombre:** consiste del nombre de la función, y este debe ser significativo y no debe coincidir con funciones previamente definidas. Además, recuerde que MATLAB es sensible a mayúsculas y minúsculas.
- **variables a retornar:** van entre corchetes `[]` y representan los valores que la función retorna en consecuencia a su cálculo.
- **parámetros:** son los argumentos necesarios para que la función pueda llevarse a cabo.

Definición de funciones

- **nombre:** consiste del nombre de la función, y este debe ser significativo y no debe coincidir con funciones previamente definidas. Además, recuerde que MATLAB es sensible a mayúsculas y minúsculas.
- **variables a retornar:** van entre corchetes `[]` y representan los valores que la función retorna en consecuencia a su cálculo.
- **parámetros:** son los argumentos necesarios para que la función pueda llevarse a cabo.

Definición de funciones

- `nombre`: consiste del nombre de la función, y este debe ser significativo y no debe coincidir con funciones previamente definidas. Además, recuerde que MATLAB es sensible a mayúsculas y minúsculas.
- `variables a retornar`: van entre corchetes `[]` y representan los valores que la función retorna en consecuencia a su cálculo.
- `parámetros`: son los argumentos necesarios para que la función pueda llevarse a cabo.

Invocación de una función

- Primero se debe guardar la función en un archivo `*.m`, con el mismo nombre que se le puso a la función. Es decir, si la función se llama `factorial`, el archivo se debe llamar `factorial.m`
- Para llamar o invocar una función, basta con su nombre seguido (entre paréntesis) por los parámetros. **No con F5.**
- La función puede llamarse desde otro programa, o bien directamente de la ventana de comandos.
- Los valores que se envíen (parámetros) no serán modificados por la función. Para cambiarlo, se utiliza la instrucción de retorno.

Invocación de una función

- Primero se debe guardar la función en un archivo `*.m`, con el mismo nombre que se le puso a la función. Es decir, si la función se llama `factorial`, el archivo se debe llamar `factorial.m`
- Para llamar o invocar una función, basta con su nombre seguido (entre paréntesis) por los parámetros. **No con F5.**
- La función puede llamarse desde otro programa, o bien directamente de la ventana de comandos.
- Los valores que se envíen (parámetros) no serán modificados por la función. Para cambiarlo, se utiliza la instrucción de retorno.

Invocación de una función

- Primero se debe guardar la función en un archivo `*.m`, con el mismo nombre que se le puso a la función. Es decir, si la función se llama `factorial`, el archivo se debe llamar `factorial.m`
- Para llamar o invocar una función, basta con su nombre seguido (entre paréntesis) por los parámetros. **No con F5.**
- La función puede llamarse desde otro programa, o bien directamente de la ventana de comandos.
- Los valores que se envíen (parámetros) no serán modificados por la función. Para cambiarlo, se utiliza la instrucción de retorno.

Invocación de una función

- Primero se debe guardar la función en un archivo `*.m`, con el mismo nombre que se le puso a la función. Es decir, si la función se llama `factorial`, el archivo se debe llamar `factorial.m`
- Para llamar o invocar una función, basta con su nombre seguido (entre paréntesis) por los parámetros. **No con F5.**
- La función puede llamarse desde otro programa, o bien directamente de la ventana de comandos.
- Los valores que se envíen (parámetros) no serán modificados por la función. Para cambiarlo, se utiliza la instrucción de retorno.

Ejemplo

```
function [prod] = factorial( n )  
    prod = 1;  % valor de retorno  
    for k = 1:n  
        prod = prod * k;  
    end  
end
```

Para llamarlo se escribe: `p = factorial(5);`

Ejemplo (Modificación)

```
function [prod] = factorial( n )  
    prod = 1;    % valor de retorno  
    for k = 1:n  
        prod = prod * k;  
    end  
    n = 0;    % Reinicia n en cero  
end
```

Ejemplo (Modificación)

Al llamar la función de la forma:

```
n = 5;  
p = factorial(n);  
p, n
```

El resultado, es:

```
p =  
    120  
n =  
     5
```

La ventaja es que ahora, si necesitamos calcular muchos factoriales, no es necesario escribir el código:

```
prod = 1;  
for k = 1:n  
    prod = prod * k;  
end
```

muchas veces, basta con llamar a la función muchas veces.

Ejemplo

```
n = 10;

for k = 1 : n
    msj = ['El factorial de ', num2str(k), ...
          ' es ', num2str(factorial(k))];
    disp(msj);
end
```

Ejemplo (Resultado)

```
El factorial de 1 es 1
El factorial de 2 es 2
El factorial de 3 es 6
El factorial de 4 es 24
El factorial de 5 es 120
El factorial de 6 es 720
El factorial de 7 es 5040
El factorial de 8 es 40320
El factorial de 9 es 362880
El factorial de 10 es 3628800
```

Declaración de funciones

- Las variables definidas dentro de una función son variables locales, pues sólo son accesibles dentro de la función, fuera de ella, no existen. Así, se pueden modificar estas y usar nombres que ya se usaron en otros archivos.
- Dentro de las funciones, los valores de retorno deben ser calculados en algún momento.
- Dentro de una función se puede escribir el comando **return** para devolver el control al programa principal que hizo la llamada a la función.

Declaración de funciones

- Las variables definidas dentro de una función son variables locales, pues sólo son accesibles dentro de la función, fuera de ella, no existen. Así, se pueden modificar estas y usar nombres que ya se usaron en otros archivos.
- Dentro de las funciones, los valores de retorno deben ser calculados en algún momento.
- Dentro de una función se puede escribir el comando **return** para devolver el control al programa principal que hizo la llamada a la función.

Declaración de funciones

- Las variables definidas dentro de una función son variables locales, pues sólo son accesibles dentro de la función, fuera de ella, no existen. Así, se pueden modificar estas y usar nombres que ya se usaron en otros archivos.
- Dentro de las funciones, los valores de retorno deben ser calculados en algún momento.
- Dentro de una función se puede escribir el comando **return** para devolver el control al programa principal que hizo la llamada a la función.

Sobrecarga de funciones

- Se puede llamar a más de una función con el mismo nombre, pero debe tener distintos parámetros para que haya una distinción.
- Por ejemplo,

```
function [y] = f( x )
```

```
    y = x^2 + 2*x + 1;
```

```
return
```

```
function [z] = f( x, y )
```

```
    z = x^2 + y^2 - x*y + 2*x - y + 1;
```

```
return
```

Ejercicio

Ejercicio

Programa una función que reciba tres números reales: a , b , c y retorne tres números que corresponden a ordenar los anteriores de manera creciente, es decir, del menor al mayor.

¿Cómo sería la función que ordena de manera decreciente?

Funciones predefinidas

- MATLAB cuenta con varias funciones previamente implementadas. En particular algunas funciones matemáticas importantes.
- Por ejemplo, la función `factorial(n)` de MATLAB determina el factorial de un número natural n .

Funciones matemáticas

Instrucción	Descripción
$\sin(x)$	seno, en radianes.
$\cos(x)$	coseno, en radianes.
$\tan(x)$	tangente, en radianes.
$\operatorname{asin}(x)$	arco-seno, en radianes.
$\operatorname{acos}(x)$	arco-coseno, en radianes.
$\operatorname{atan}(x)$	arco-tangente, en radianes.
$\log(x)$	logaritmo natural.
$\log_{10}(x)$	logaritmo en base 10.
$\exp(x)$	función exponencial de base e .
$\operatorname{sqrt}(x)$	raíz cuadrada.
$\operatorname{abs}(x)$	valor absoluto.

Ejercicio

Ejercicio

Escriba una función en MATLAB que, dado un vector $\mathbf{x} \in \mathbb{R}^n$ retorne el vector $\mathbf{F} \in \mathbb{R}^n$ tal que $F_i := f(x_i)$, donde:

$$f(t) := \begin{cases} \frac{1 - \cos(t)}{t^2} & \text{si } t \neq 0 \\ \frac{1}{2} & \text{si } t = 0 \end{cases}$$

Ejercicio

Ejercicio (para la casa)

Ahora, considerando:

$$g(t) := \begin{cases} \frac{2 \operatorname{sen}^2\left(\frac{t}{2}\right)}{t^2} & \text{si } t \neq 0 \\ \frac{1}{2} & \text{si } t = 0 \end{cases}$$

donde es claro que $f(t) = g(t)$. Evalúe ambas en:

```
x = [0.188e-7, 0.189e-7, 0.19e-7];
```

¿Son realmente iguales estas funciones?