Tema 5

Programación en Matlab

Fundamentos de Informática

Grado en Ing. Química

ugr | Universidad de Granada

Jesús Alcalá y David Pelta

DECSAI

Universidad de Granada

Índice

Objetivos

- Conocer las distintas estructuras de control que existen y ser capaces de generar un script que contenga distintas estructuras.
- Generar script que permitan resolver problemas más avanzados.
- Ser capaces de dividir un script en módulos.
- Diseñar nuestras propias funciones.

Bibliografía

- J. Garcia Molina, F. Montoya Dato, et al., Una introducción a la Programación. Un enfoque algorítmico, Thompson, 2005
- Pérez López, César, MATLAB y sus aplicaciones en las Ciencias y la Ingeniería. Madrid : Pearson Educación, 2002
- Gilat, Amos Matlab : una introducción con ejemplos prácticos. Barcelona : Reverté, 2006
- García de Jalón J., Rodríguez J. , Vidal J.. Aprenda Matlab 7.0 como si estuviera en primero
- Stephen J. MATLAB programming for engineers. Thomson, 2008.
- ***Numerosos libros en la Biblioteca de Ciencias y el Politécnico***

Operadores Relacionales

Son los operadores habituales de comparación de números.
El resultado es 1 (true) o 0 (false). Recíprocamente, cualquier valor distinto de 1 es considerado *true*, y el 0 equivale a *false*.

$(4 < 5) \Rightarrow true$
$(4 > 5) \Rightarrow false$

Operadores:

- < menor que
- > mayor que
- <= menor o igual que
- >= mayor o igual que
- == igual que
- ~= distinto que

Operadores Relacionales

También se aplican sobre 2 **matrices o vectores**. La comparación se realiza **elemento a elemento** y el resultado es otra matriz de unos y ceros del mismo tamaño

```
>> A=[1 2;0 3]; B=[4 2;1 5];
>> A==B
ans =
     0     1
     0     0
>> A~=B
ans =
     1     0
     1     1
```

```
¿ Que hace esto ?
>> a = [1 2 3 4 5];
>> b = [1 2 6 7 8];
>> a == b
>> sum(a == b)
>> sum(a ~= b)
```

Operadores Lógicos

- Son operadores binarios
- Se aplican sobre datos (y expresiones) de tipo booleanas:
  - &: and (función equivalente: and(A,B)). Se evalúan ambos operandos
  - &&: and breve: si el primer operador es *false* no se evalúa el segundo
  - |: or (función equivalente: or(A,B)). Se evalúan ambos operandos
  - ||: or breve: si el primer operador es *true* no se evalúa el segundo
  - ~: not (función equivalente: not(A)). Este es un operador unario
  - xor(A,B): realiza un "or exclusivo"
- Devuelven true o false

NOT	True	False
False	True	

XOR	True	False
True	False	True
False	True	False

AND	True	False
True	True	False
False	False	False

OR	True	False
True	True	True
False	True	False

Ejercicio

Dadas las variables *count = 0, limit = 10, x=2, y=7,* calcule el valor de las siguientes expresiones booleanas

(count == 0) && (limit < 20)

(limit > 20) || (count < 5)

~(count == 12)

(count == 1) && (x < y)

~(((count < 10) || (x < y)) && (count >= 0))

((count>5) && (y==7)) || ((count<=0) && (limit==5*x))

~((limit ~= 10) && (z > y))

Estructuras

Las estructuras de control de un lenguaje de programación se refieren al orden en que las instrucciones de un algoritmo se ejecutarán. El orden de ejecución de las sentencias o instrucciones determinán el flujo de control.

Las tres estructuras básicas de control son:

- ***secuencia***
- ***selección***
- ***repetición***

Estructuras

*Un programa propio puede ser escrito utilizando las tres estructuras de control básicas (*Bôhm y Jacopin (1996)).

Un programa se define como *propio* si cumple lo siguiente:

- Posee un sólo punto de entrada y salida o fin para control del programa.
- Existen caminos desde la entrada hasta la salida que se pueden seguir y que pasan por todas las partes del programa.
- Todas las instrucciones son ejecutadas y no existen lazos o bucles infinitos.

Estructura Secuencial

Las sentencias se ejecutan sucesivamente, en el orden en que aparecen. No existen “saltos” o bifurcaciones.

```
a = input('Introduce coeficiente de grado 2: ');
b = input('Introduce coeficiente de grado 1: ');
c = input('Introduce coeficiente independiente: ');
disp ('Las raíces son: ');
(-b + sqrt((b^2) - (4 * a * c))) / (2 * a)
(-b - sqrt((b^2) - (4 * a * c))) / (2 * a)
```

Estructura Secuencial

Si bien el programa anterior es correcto, no es capaz de manejar situaciones excepcionales.

Por ejemplo, ¿qué pasa en la sentencia:

```
( -b + sqrt( b^2 - 4*a*c ) ) / (2*a);
```

si `a == 0` **o** `(b^2 - 4*a*c) < 0`**?**

En el primer caso, obtendríamos un error de ejecución por intentar hacer una división por cero.

Solución: primero comprobar (mediante **estructuras condicionales**), y no efectuar el cálculo si no es posible

Estructura Condicional

También recibe el nombre de *"estructura de selección"*

Permite elegir entre diferentes cursos de acción en función de condiciones.

`Si la nota del examen es mayor o igual que 5 mostrar "Aprobado"`

Si la condición es *verdadera*, entonces se ejecuta la sentencia ***mostrar,*** y luego el programa continuaría en la sentencia siguiente al ***Si***

Si la condición es *falsa*, la sentencia **mostrar** se ignora y el programa continúa

Estructura Condicional Simple

La instrucción:

Si la nota del examen es mayor o igual que 5 mostrar *"Aprobado"*

Se traduce a Matlab mediante una sentencia condicional simple:

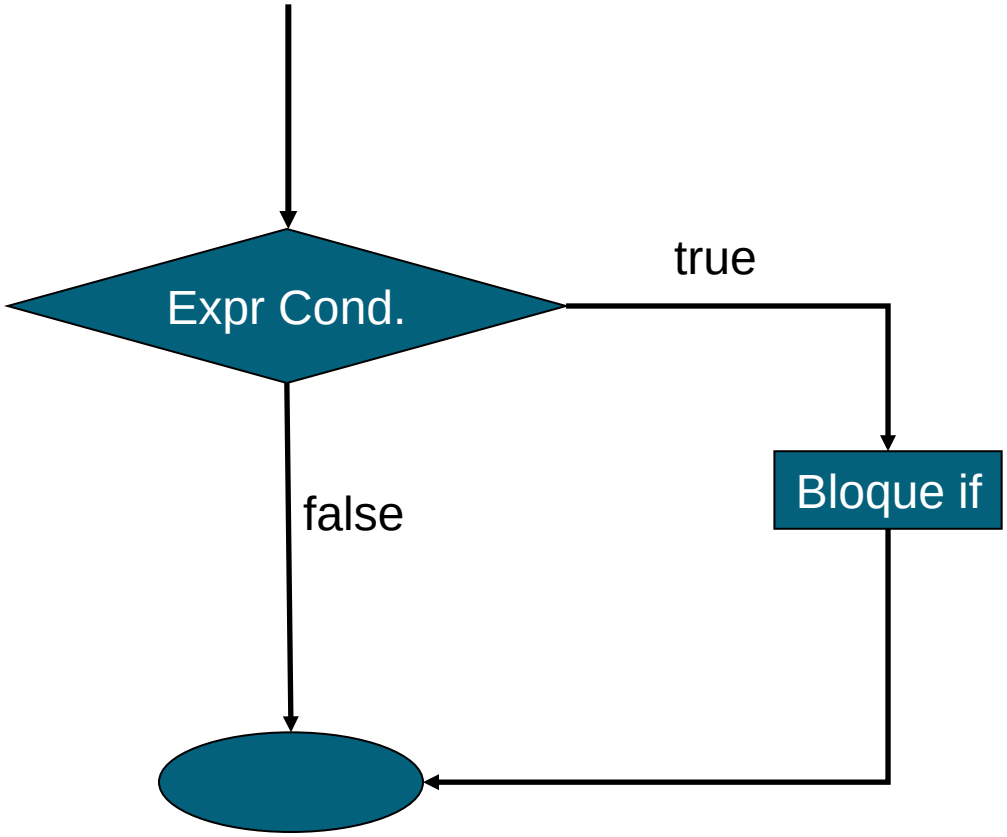
```
if nota >= 5
   disp ('Aprobado')
end
```

La forma general es:

```
if <condicion>
   <bloque if>
end
```

Estructura Condicional Simple

El diagrama asociado es:



Estructura Condicional Simple

La condición del *if* puede ser una condición matricial, del tipo A==B, donde A y B son matrices del mismo tamaño.

Para considerar que la condición se cumpla, es necesario que sean iguales dos a dos todos los elementos de las matrices A y B

Si A==B exige que todos los elementos sean iguales dos a dos

Si A~=B exige que todos los elementos sean diferentes dos a dos

Cálculo de las Raíces

```
a = input('Introduce coeficiente de grado 2: ');
b = input('Introduce coeficiente de grado 1: ');
c = input('Introduce coeficiente independiente: ');
if a~=0
  disp ('Las raíces son: ');
  (-b + sqrt((b^2) - (4 * a * c))) / (2 * a)
  (-b - sqrt((b^2) - (4 * a * c))) / (2 * a)
end
```

Cálculo de las Raíces (2)

Esta aproximación no calcula raíces si ***a==0***.

¿Cómo hacer que también calcule la solución si a==0 (ecuación de primer grado)?

```
if a==0
  disp ('La raíz es: ');
  -c/b
end
```

Algo a considerar: las condiciones son excluyentes (**"*a*" vale 0** ó **"*a*" vale distinto de 0**)

Ejercicios

1. Implementar un script que pida un valor al usuario y que escriba por pantalla “Par” si el valor introducido es un número par.

2. Implementar un script que pida un valor al usuario y diga por pantalla si es multiplo de 5

Estructura Condicional Doble

Permite elegir entre 2 cursos de acción diferentes en función del valor de verdad de una expresión lógica.

Si* la nota del examen es *mayor o igual* que *5
mostrar *"Aprobado"*
Si no* mostrar *"Suspenso"

En Matlab:

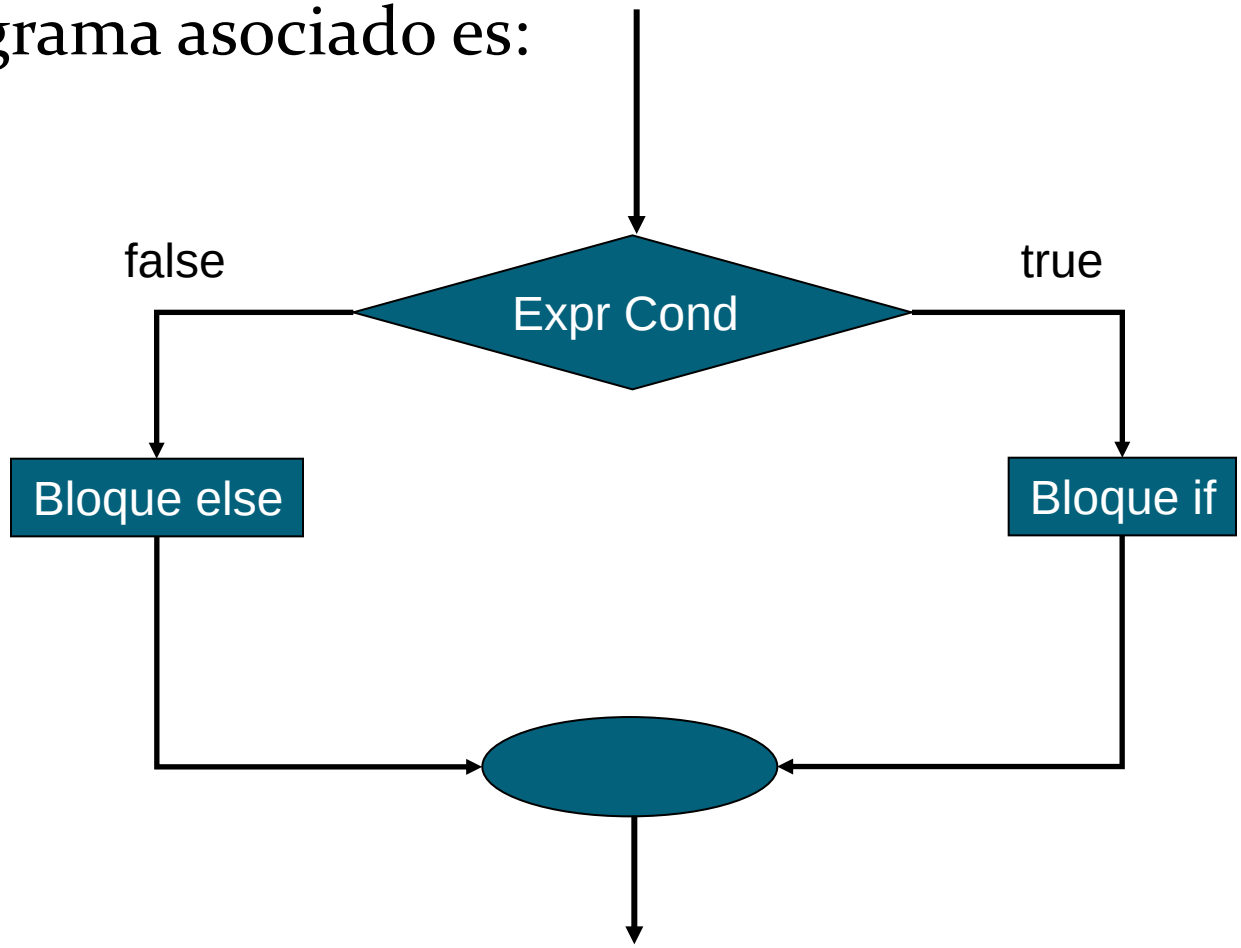
```
if nota >= 5
  disp ('Aprobado')
else
  disp ('Suspenso')
end
```

La forma general es:

```
if <condicion>
  < bloque if >
else
 < bloque else >
end
```

Estructura Condicional Doble

El diagrama asociado es:



Cálculo de Raíces

En la ecuación de segundo grado, cuando **`a == 0`**, entonces la raíz es única y vale ***`-c/b`***

```
if a~=0
  disp ('Las raíces son: ');
  (-b + sqrt((b^2) - (4 * a * c))) / (2 * a)
  (-b - sqrt((b^2) - (4 * a * c))) / (2 * a)
else
  disp ('La raíz es: ');
  -c/b
end
```

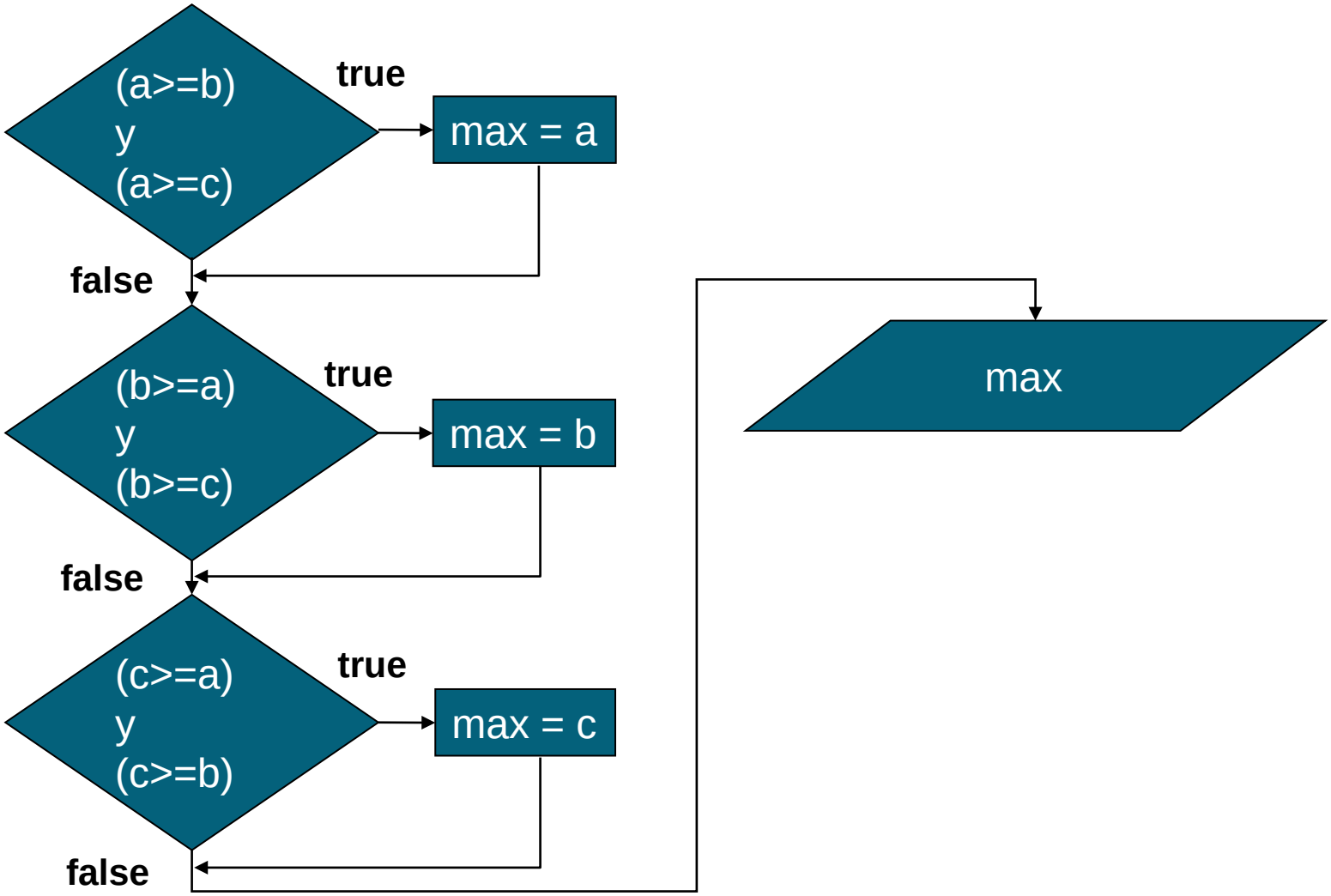
Ejemplo: El mayor de 3

- ¿Cómo hacer para calcular el máximo de tres números *a*, *b* y *c* y almacenar el resultado en una variable *max* cuyo valor se mostrará por pantalla al final?

Ejemplo: El mayor de 3

```
a = input('Introduce el valor a: ');
b = input('Introduce el valor b: ');
c = input('Introduce el valor c: ');
if a>=b && a>=c
    max = a;
end
if b>=a && b>=c
    max = b;
end
if c>=a && c>=b
    max = c;
end
disp('El valor mayor es: ')
max
```

Ejemplo: El mayor de 3



Anidamiento de estructuras condicionales

Permite elegir entre varios cursos de acción diferentes en función del valor de verdad de una expresión lógica.

Forma general:

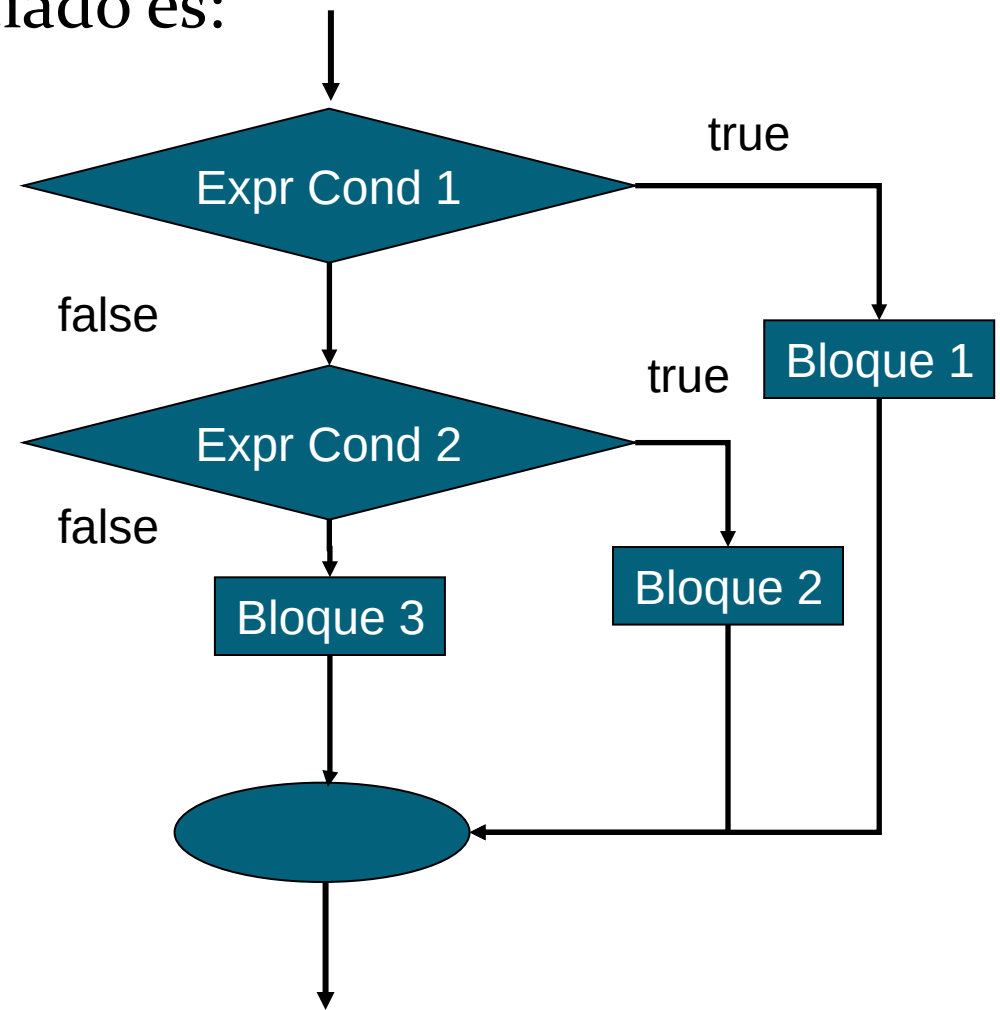
```
if <condición>
  <bloque if>
elseif <condición>
  <bloque elseif>
...
else
  <bloque else>
end
```

Pueden haber tantos *elseif* como queramos

No es obligatorio añadir la parte *else* al final

Anidamiento de estructuras condicionales

El diagrama asociado es:



Ejercicios

1. Implementar un script que pida una nota al usuario y que escriba por pantalla: Sobresaliente si es > 9, Notable si está entre 7 y 9, Aprobado si está entre 5 y 7, y Suspenso en cualquier otro caso.

2. Implementar un script que pida un valor y devuelva si el valor es par o impar

Anidamiento de estructuras condicionales (2)

```
Si la nota es mayor o igual
que 9
   imprimir "Sobresaliente”
else
  Si la nota es mayor o
  igual que 7
    imprimir "Notable"
  else
    Si la nota es mayor o
  igual
    que 5
       imprimir "Aprobado"
    else
       imprimir "Suspenso"
```

```
if nota>=9.0
    disp ('Sobresaliente')
elseif nota>=7.0
    disp ('Notable')
elseif nota>=5.0
    disp ('Aprobado')
else
    disp ('Suspenso')
end
```

¿Cuándo se ejecuta cada instrucción?

```
if condic_1
  inst_1;
  if condic_2
    inst_2;
  else
    inst_3;
  end
  inst_4;
else
  inst_5;
end
```

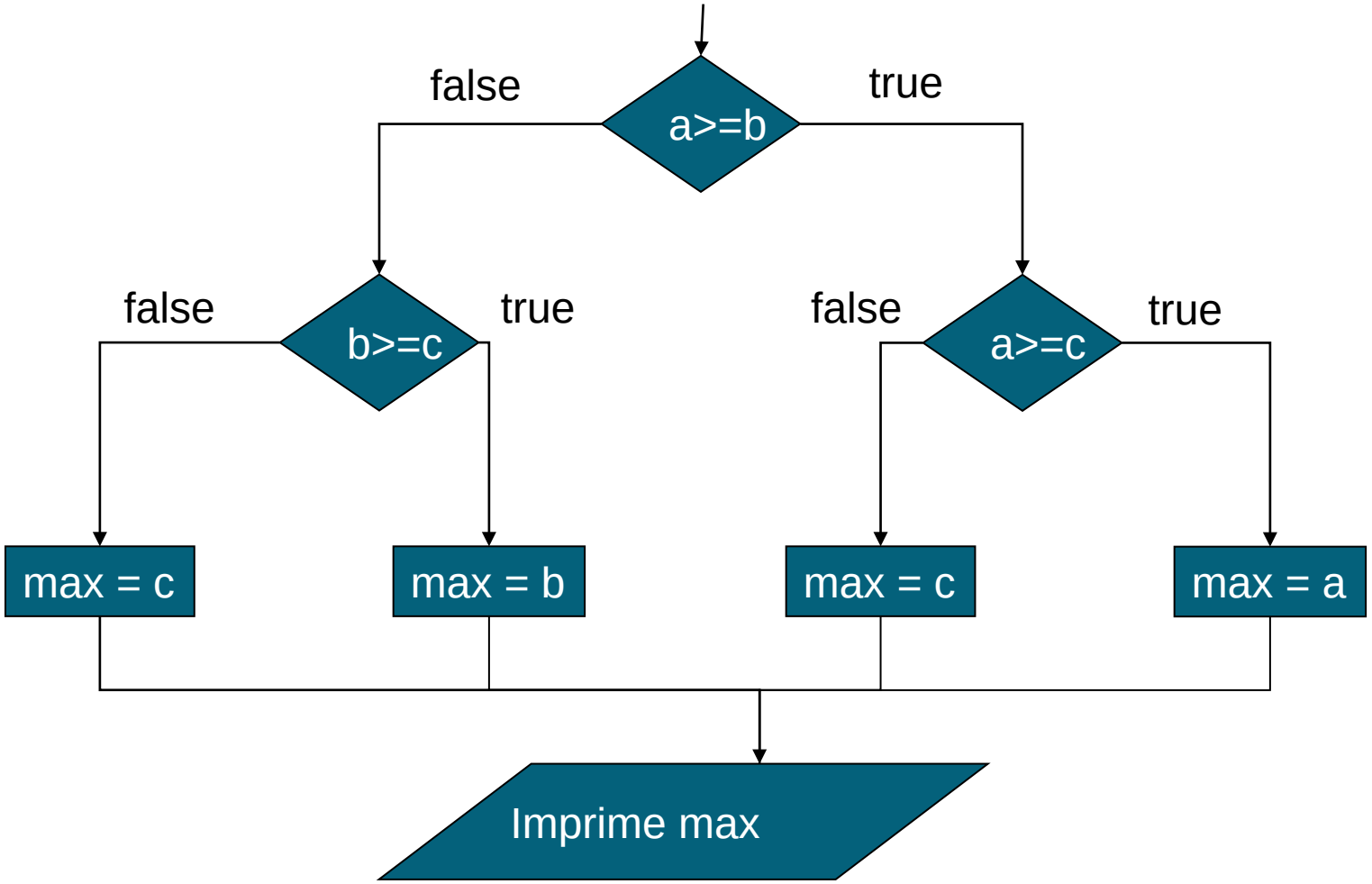
	condic_1	***condic_2***
inst_1	*true*	*independiente*
inst_2	*true*	*true*
inst_3	*true*	*false*
inst_4	*true*	*independiente*
inst_5	*false*	*independiente*

Ejemplo: El mayor de 3

```
if a>=b
    if a>=c
        max = a;
    else
        max = c;
    end
else
    if b>=c
        max = b;
    else
        max = c;
    end
end
disp ('El resultado es: ')
max
```

Opción 2

Ejemplo: El mayor de 3

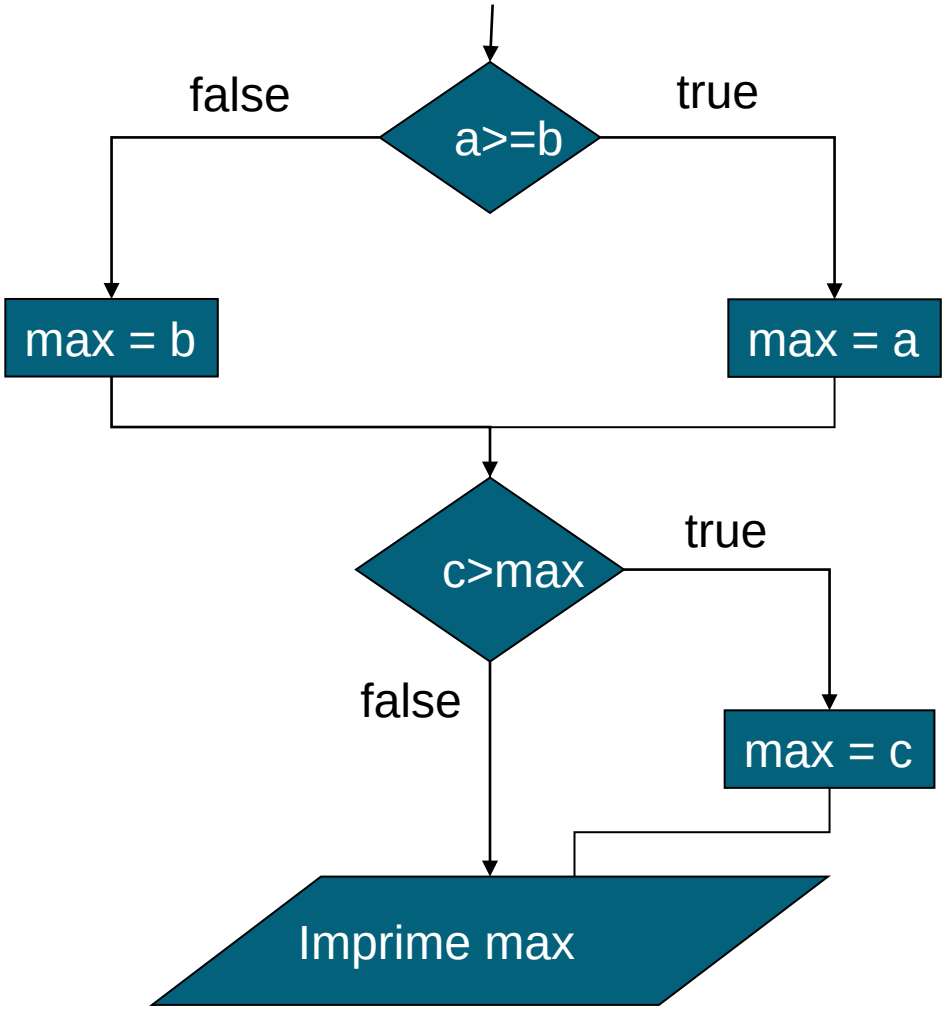


Ejemplo: El mayor de 3

```
if a>=b
    max = a;
else
    max = b;
end
if c>max
    max = c;
end
disp ('El resultado es: ')
max
```

Opción 3

Ejemplo: El mayor de 3



Ejemplo: El mayor de 3

	Condiciones	Asignaciones
Opción 1	3	1
Opción 2	2	1
Opción 3	2	1 ó 2

Buscaremos un término medio entre Eficiencia y Elegancia

Ejercicios

- Completar el script de la ecuación de segundo grado, teniendo en cuenta los valores del discriminante también.
- Implementa un script que calcule el mayor de cuatro números.

Ejemplo

```
dato1 = input('Introduce el primer operando: ');
dato2 = input('Introduce el segundo operando: ');
opcion = input('Selecciona (1) sumar, (2) restar: ');
if opcion == 1
disp ('Suma = ')
dato1 + dato2
end
if opcion == 2
disp ('Resta = ')
dato1 - dato2
end
if opcion ~= 1 && opcion ~= 2
disp ('Ninguna operacion')
end
```

En este caso, las condiciones no son excluyentes

Ejemplo

```
dato1 = input('Introduce el primer operando: ');
dato2 = input('Introduce el segundo operando: ');
opcion = input('Selecciona (1) sumar, (2) restar: ');
if opcion == 1
   disp ('Suma = ')
   dato1 + dato2
elseif opcion == 2
   disp ('Resta = ')
   dato1 - dato2
else
   disp ('Ninguna operacion')
end
```

Más eficiente

Estructura Condicional Múltiple

Permite definir en una sola estructura, una serie de pares (condición, acción), con condiciones mutuamente excluyentes.

Muy utilizada en la construcción de menús.

```
switch <expresión>
case <constante1>,
  bloque 1
case {<constante2>,<constante3>,…}
  bloque 2
…
otherwise,   (opcional)
  bloque 2
end
```

Estructura Condicional Múltiple

- **<expresión>** cuyo resultado debe ser un número escalar o una cadena de caracteres.
- **<constante>** es un valor con el que se compara la expresión.
- **switch** sólo comprueba la igualdad.
- Se pueden **agrupar varias condiciones dentro de unas llaves**. Basta la igualdad con cualquier elemento para que se ejecute el bloque.
- El identificador especial **otherwise** permite incluir un caso por defecto, que se ejecutará si no se cumple ningún otro. Se suele colocar como el último de los casos.
- En las estructuras condicionales múltiples también se permite el anidamiento.

Ejemplo

```
dato1 = input('Introduce el primer operando: ');
dato2 = input('Introduce el segundo operando: ');
opcion = input('Selecciona (1) sumar, (2) restar: ');
switch opcion
    case 1,
        disp ('Suma = ')
        dato1 + dato2
    case 2,
        disp ('Resta = ')
        dato1 - dato2
    otherwise,
        disp ('Ninguna operacion')
end
```

Ejemplo

```
dato1 = input('Introduce el primer operando: ');
dato2 = input('Introduce el segundo operando: ');
opcion = input('Selecciona (S) sumar, (R) restar: ', 's');
switch opcion
    case {'S','s'},
        disp ('Suma = ')
        dato1 + dato2
    case {'R','r'},
        disp ('Resta = ')
        dato1 - dato2
    otherwise,
        disp ('Ninguna operacion')
end
```

Ejercicios

- Implementa un script que pida al usuario 2 valores y una de las siguientes opciones: Sumar (S o s), Restar (R o r), Multiplicar (M o m) o Dividir (D, d). Mostrar por pantalla el resultado de la operación realizada según la opción introducida.

Estructuras Repetitivas

Estructuras Repetitivas

Las estructuras repetitivas son también conocidas como *bucles, ciclos o lazos.*

Una estructura repetitiva permite la ejecución de un conjunto de sentencias:

- mientras se satisfaga una condición determinada (controladas por condición)
- un número determinado de veces (controladas por contador)

Bucles Controlados por Condición

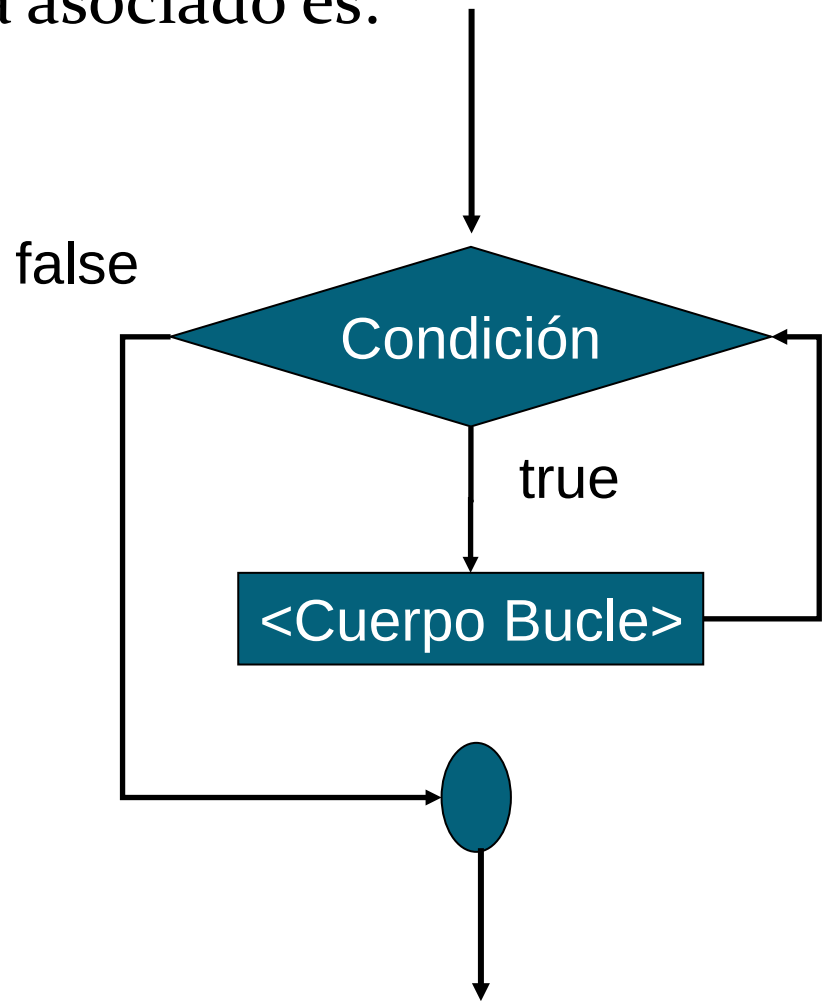
Se ejecuta el conjunto de sentencias de interés mientras la condición sea verdadera.

La forma general es:

while **<condición>**
<cuerpo del bucle>
end

Bucles Controlados por Condición

El diagrama asociado es:



Ejemplo

Escribir 10 líneas con 4 estrellas cada una

```
cont = 1;

while cont <= 10
 disp ('****')
 cont = cont + 1;
end
```

```
cont = 0;

while cont < 10
 disp ('****')
 cont = cont + 1;
end
```

Ejemplo

Escribir *tope* líneas con 4 estrellas cada una

```
tope = input ('Indique el número de líneas: ')
cont = 0;

while cont < tope
 disp ('****')
 cont = cont + 1;
end
```

Ejemplo

Escribir los números pares entre [0..*tope*]

```
tope = input('Intro..: ');
i = 0;
while i <= tope
  disp (num2str(i))
  i = i + 2;
end
```

¿Qué ocurre si el código anterior se cambia por :

```
tope = input('Intro..: ');
i = 0;
while i <= tope
  i = i + 2;
  disp (num2str(i))
end
```

El **0** no se imprime e imprime un número par de más

Ejercicios

Implementa un script que calcule la siguiente sumatoria:

$$\sum_{i=1}^{10} i$$

Implementa un script que calcule la siguiente producto:

$$\prod_{i=1}^{10} i$$

Ejemplo: Control por centinela

Sumar valores que se leen por teclado hasta que se lee -1.
(el valor utilizado para detener el procesamiento se conoce como centinela)

Lectura anticipada: se usa un bucle y se comprueba el primer valor.

```
suma = 0;
valor = input('Intro…: ');
while valor  ~= -1
  suma = suma + valor;
  valor = input('Intro…: ');
end
disp ('La suma es: ')
suma
```

Ejemplo de Lectura Anticipada

```
% Programa que va leyendo números enteros hasta que
% se introduzca el cero. Imprimir el número de
% pares e impares introducidos
ContPar = 0;
ContImpar = 0;
valor = input('Introduce valor: ');
while valor ~= 0
    if mod(valor,2) == 0
        ContPar = ContPar + 1;
    else
        ContImpar = ContImpar + 1;
    end
    valor = input('Introduce valor: ');
end
disp('Fueron ');
disp(num2str(ContPar));
disp(' pares y ');
disp(num2str(ContImpar));
disp(' impares');
```

Problema para Resolver

Tenemos que realizar un programa que calcule la nota promedio de un conjunto de notas. La cantidad de notas puede variar en cada ejecución del programa.

- Utilizaremos un valor "*centinela*" para indicar el fin de las notas.
- El proceso terminará cuando se ingrese dicho valor
- El valor concreto del centinela debe ser diferente al de cualquier posible entrada válida (en este caso podría ser -1)

Bucles como Filtros en la Entrada de Datos

Los ciclos son útiles para forzar qu
entrada pertenezcan a un rango o c
opciones predeterminados.

Sigue pidiendo el valor mientras sea negativo (hasta que sea positivo)

```
valor = input('Introduzca un valor > 0: ');
while valor <= 0
  valor = input('Introduzca un valor > 0: ');
end
cos(valor)
```

Ejercicios

- Completar el script que pide al usuario 2 valores y una de las siguientes opciones: Sumar (S o s), Restar (R o s), Multiplicar (M o m) o Dividir (D, d), para que se asegure de que el usuario introduce una opción correcta.
- Modificar el script de la nota para asegurarse de que el usuario introduce un valor entre 0 y 10.

Bucle Sin Fin

Bucle al que la evaluación de la condición nunca le permite dejar de ejecutarse.

```
contador = 2;
while contador < 3
        contador = contador - 1;
        disp (num2str(contador));
end
```

```
contador = 1;
while  contador ~= 10
        contador = contador + 2;
end
```

Bucles Controlados por Contador

Se utilizan para repetir un conjunto de sentencias un número fijo de veces. Se necesita una variable controladora, un valor inicial, un valor final y un incremento.

Ejemplo

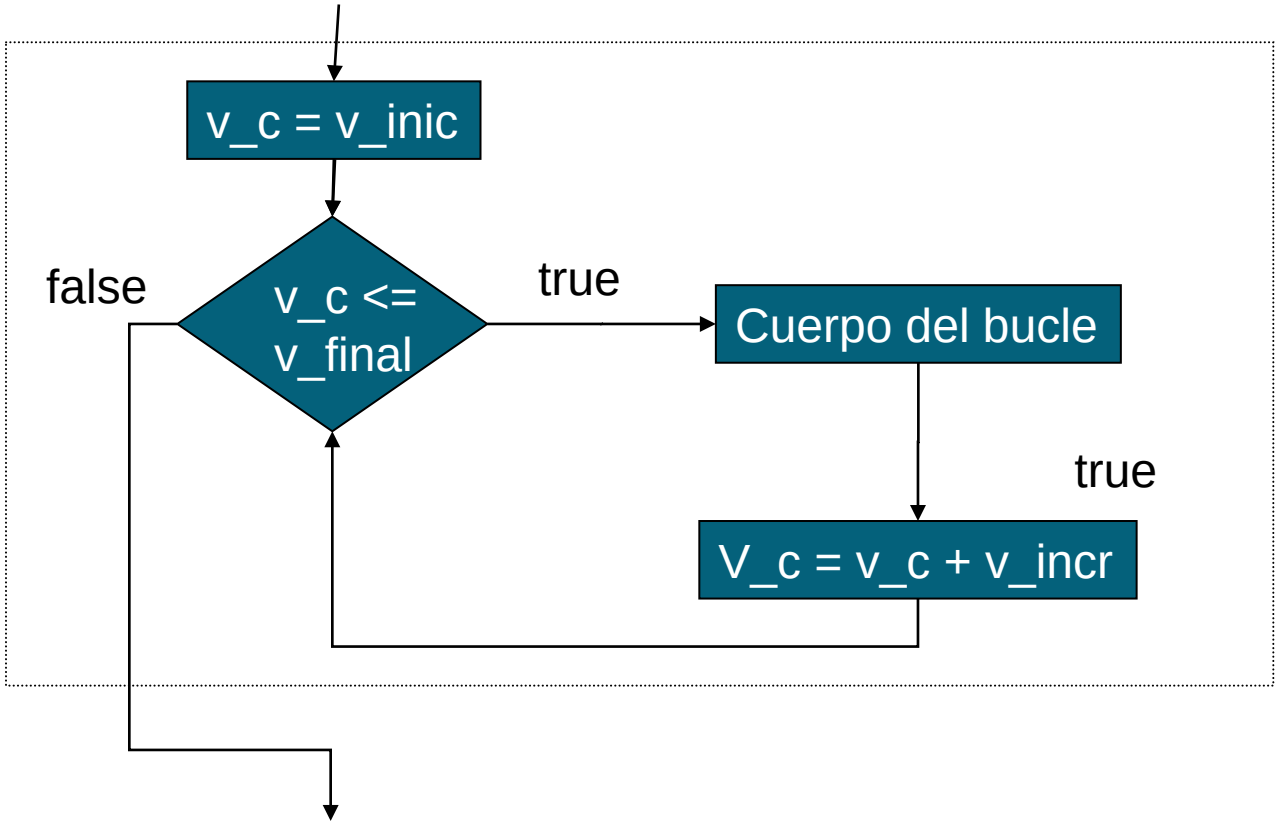
```
% Hallar la media de 5 números enteros.
% Entradas: 5 números enteros (se leen en valor).
% Sal                    e los 5 números.
suma
i = 0;
while i <= 5
    valor = input('Introduce v
    suma = suma + valor;
    i = i + 1;
end
suma / 5
```

Valor inicial del contador

Prueba de límites del valor del contador

Incremento del valor del contador

Ejemplo: Diagrama



Existe un tipo específico de bucles con esta utilidad

Ejemplo

Determinar el promedio de las notas de un examen

```
suma = 0;
contador = 0;

nota = input ('Ingrese la nota (-1 para fin)');
while nota ~= -1
    suma = suma + nota;
    contador = contador + 1;
    nota = input ('Ingrese la nota (-1 para fin)');
end
if contador > 0
    disp ('El promedio es: ');
    suma / contador
else
    disp ('No se ingresaron notas');
end
```

Bucles Controlados por Contador: *for*

Si conocemos exactamente la cantidad de veces que necesitamos repetir un conjunto de sentencias, entonces podemos usar un bucle *for*.

En general, los bucles controlados por contador requieren

- una variable de control o contador
- un valor inicial para el contador
- un valor final para el contador
- una condición para verificar si la variable de control alcanzó su valor final.
- un valor de incremento (o decremento) con el cual se modifica la variable de control en cada bucle

Bucles Controlados por Contador: *for*

La forma general del bucle *for* es:

```
for i=1:n
      < conjunto de sentencias>
end
```

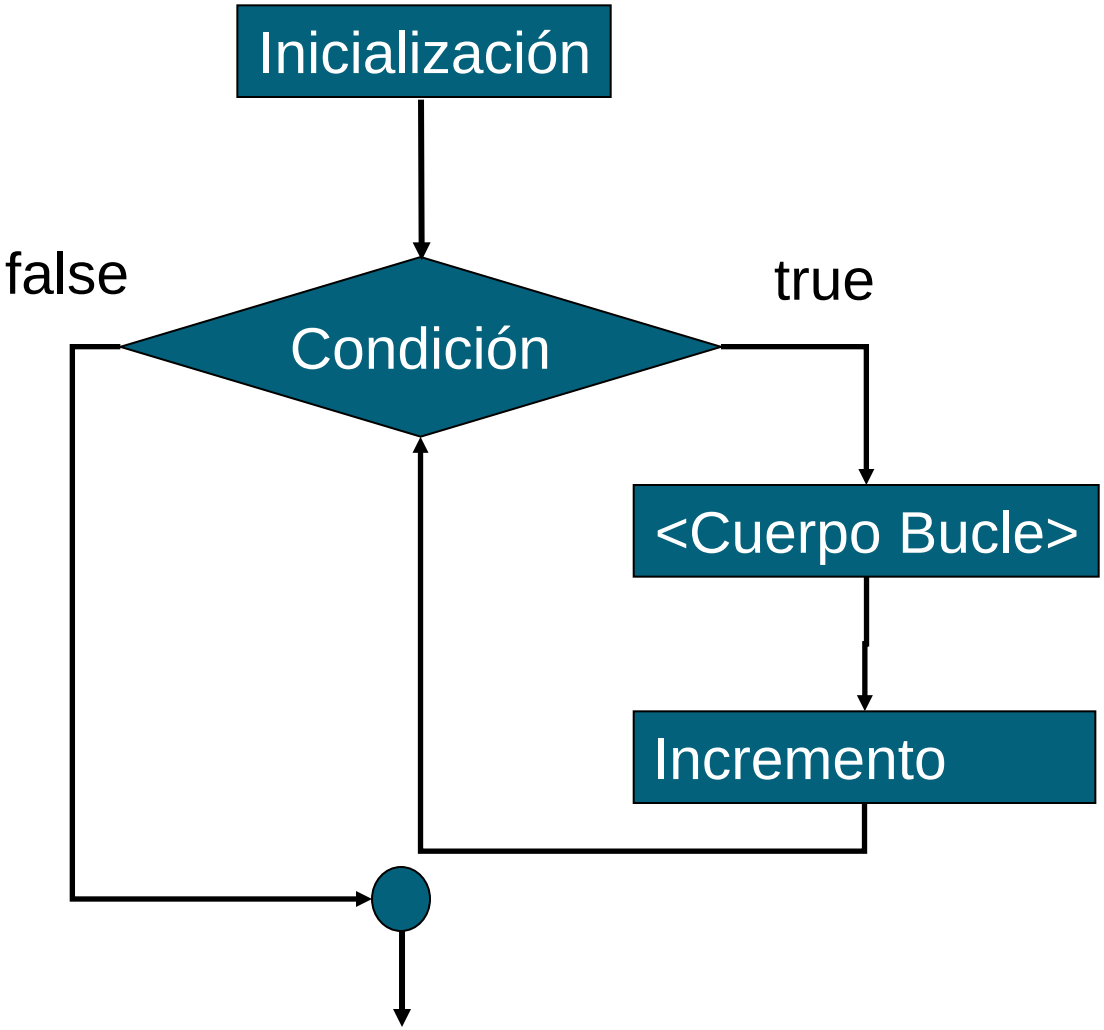
ó

```
for i=vectorValores
      < conjunto de sentencias>
end
```

Ejemplo: imprimir los valores enteros entre 1 y 10

```
for i=1:10
  i
end
```

Diagrama


Inicialización
false
Condición
true
<Cuerpo Bucle>
Incremento

Bucles Controlados por Contador: *for*

Bucles anidados (para cada valor i, j toma todos sus posibles valores):

```
for i=1:n
  for j=1:n
     < conjunto de sentencias>
  end
end
```

La variable contador también puede tomar valores de una matriz. La variable i es un vector que va tomando en cada iteración el valor de una de las columnas de A.

```
for i=A
  < conjunto de sentencias>
end
```

Ejercicios

Implementar un script que devuelva la suma de todos los valores de una matriz utilizando sentencias for.

Implementar un script que devuelva la media de todos los valores de una matriz utilizando sentencias for.

Implementar un script que pida un valor al usuario y compruebe si es un número primo.

Relación entre *while* y *for*

Se debe notar que los bucles *for* se pueden reescribir como bucles *while*

```
for inicialización:incremento:condición
   < conjunto de sentencias>
end
```

```
inicialización
while condición
   < conjunto de sentencias>
   incremento
end
```

Ejemplos

```
% tabla de multiplicar de un nº dado
nro = input('Valor: ');
for i=1:10
  nro * i;
end
```

```
% sumar los nros pares entre 2 y 200

suma = 0;
for i=2:2:200
  suma = suma + i;
end
disp ('La suma es: ')
suma
```

Ejemplos

```
% tabla de multiplicar de un nro dado
nro = input('Valor: ');
i = 1;
while i <= 10
  nro * i;
  i = i+1;
end
```

```
% sumar los nros pares entre 2 y 200

suma = 0;
i = 2;
while i <= 200
  suma = suma + i;
  i = i+2;
end
disp ('La suma es: ')
suma
```

$\sum_{i=1}^{10} i$

Ejercicios

Modificar el script que calcula la siguiente sumatoria utilizando un for:

$$\sum_{i=1}^{10} i$$

Modificar el script que calcula la siguiente producto utilizando un for :

$$\prod_{i=1}^{10} i$$

Sentencias Especiales

break: Hace que termine la ejecución del bucle *for* y/o *while* mas interno de los que comprenden a dicha sentencia.

```
for i=1:5
  for j=1:10
    if j == i
       break;
    else
       j
    end
  end
end
```

Estudiar las sentencias: *Continue* y *Try...Catch...End*

Funciones

Motivación (I)

- Una forma natural de atacar problemas grandes es dividirlo en sub-problemas que se puedan resolver de forma "independiente" y luego combinarse.
- En programación, esta técnica se refleja en el uso de sub-programas: conjunto de instrucciones que realizan una tarea específica.
- En Matlab los sub-programas se denominan funciones.
- Recibe valores de entrada (parámetros) y proporciona un valor de salida (valor de retorno). La función se *llama* o *invoca* cuando deseamos aplicarla.
- Analogía: un jefe (él que llama la función) solicita a un empleado (la función), que realice una tarea y devuelva los resultados una vez que finalice.

Motivación (II)

La utilización de subprogramas permite:

- *Reducir la complejidad del programa ("divide y vencerás").*
- *Eliminar código duplicado.*
- *Limitar los efectos de los cambios (aislar aspectos concretos).*
- *Ocultar detalles de implementación (p.ej. script complejos).*
- *Promover la reutilización de código*
- *Mejorar la legibilidad del código.*
- *Facilitar la portabilidad del código.*

Funciones

Una función puede tener varios argumentos, aunque el ***resultado*** o ***valor de la función*** es único.

Ejemplo:

$f_1(x,y) = x * y$

$f_2(x,y,z) = x^2 + y^2 + z^2$

Los lenguajes proveen una serie de funciones predefinidas que facilitan la tarea al programador. Por ejemplo, las incluídas en la librería matemática.

Para utilizarlas, debemos escribir *nombre de la función(argumentos)*

sqrt(900)

la función *sqrt*, toma un argumento de tipo *double* y devuelve como resultado un valor de tipo *double*

Funciones

- Durante el curso, ya hemos utilizado funciones "provistas por el lenguaje", especialmente, las de operaciones matemáticas

- Ahora tenemos la posibilidad de definir nuestras propias funciones.

Definición de Funciones

La primera fila del fichero <*nombre.m*> que defina la función tiene la forma:

function [<lista de valores de retorno>] = nombre(<lista de parámetros>)

<Lista de valores de retorno>: Los valores de retorno que van entre corchetes y separados por comas (siempre que haya mas de uno). Si no hay se eliminan los [] y el =.

nombre: El nombre de la función (debe coincidir con el nombre del fichero .m).

<lista de parámetros>: Argumentos de la función que van entre paréntesis y separados por comas (siempre que haya mas de uno). Si no hay se eliminan los ().

Definición de Funciones. Cont.

A continuación se añaden todas las instrucciones que se aplican para calcular el valor que se debe devolver.

Las variables creadas dentro de la función **NO SON ACCESIBLES** desde otras partes, por lo que **NO interfieren** con variables creadas en otras funciones o en el WorkSpace. Ademá, a diferencia que con los script, las variables **NO SE QUEDAN CREADAS** en el **WorkSpace**.

Definición de Funciones: Argumentos

Una función **NO PUEDE** modificar nunca los argumentos que recibe.

Los valores que se le pasan como argumentos no se copian en locales a la función si no son modificados por dicha función (**paso por referencia**). Sin embargo, si dentro de la función se realizan cambios sobre los argumentos, antes se sacan copias de dichos argumentos a variables locales y se modifican las copias (**paso por valor**).

Ejemplos

```
function [resultado]  = promedio (v1, v2)
    resultado = (v1+v2)/2;

function [resultado] = esPar (nro)
    if mod(nro,2) == 0
            resultado = 1;
    else
            resultado = 0;
    end

function [resultado] = esMayor (a, b)
    if a > b
            resultado = 1;
    else
            resultado = 0;
    end
```

Ejecutar

Para llamar a las funciones que creamos solo tenemos que ponerlas en el directorio de trabajo actual (o añadirlas al path el directorio en el que se encuentran los ficheros creados), poner en el intérprete el nombre de la función y pasarle sus argumentos.

Ejemplos:

```
>> x = promedio(4.5, 9.6)
>> esPar(5)
>> esMayor(9, 5)
```

Ejercicios

Implementar una función que calcule el factorial de un número.

Implementar una función que calcule el número combinatorio de dos número (n y m).

Impementar una funcion que reciba un valor (v) y una matriz (A) y devuelva si dicho valor se encuentra en la matriz.

Ejecución

Qué ocurre cuando hacemos:

```
x = promedio(5.3, 8 );
```

?

- Se copia el valor de los argumentos.
- Se transfiere el flujo de ejecución a la función invocada.
- Cuando la función termina su ejecución, devuelve el control a la función o al interprete desde el que fué llamada.

Ejemplos

```
function [resultado] = sumatoria(inicio, fin)
  resultado = 0;
  for i=inicio:fin
    resultado = resultado + i;
  end

function [resultado] = potencia (n, k)
  resultado = 1;
  for i=1:k
    resultado = resultado * n;
  end
```

Llamadas entre Funciones

Dentro de una función podemos llamar a otras funciones proporcionadas por Matlab o creadas por nosotros mismos.

Ejemplo:

```
>> k = 3;
>> cuadrado(k)
>> cubo(k)
```

```
function [resultado] = cuadrado (n)
  resultado = n*n;
```

```
function [resultado] = cubo (n)
  resultado = n*cuadrado(n);
```

Funciones que No Devuelven Valores

- En la primera fila del fichero ponemos:

  function nombre_func (<lista de parámetros>)

- Útiles para mostrar información:

  mostrarValores(v1, v2)

Ejemplo: imprimir k asteriscos

imprimir k símbolos

Sub-Funciones

- Solo a partir de la versión Matlab 5.0.
- Son funciones adicionales definidas en el mismo fichero .m con nombre diferente al del fichero.
- Solo pueden llamadas por las funciones contenidas en ese fichero (invisibles para otras funciones externas).

Ejemplo: fichero *mi_fun.m*

```
function y = mi_fun (a, b)
  y = subfun1(a, b);

function x = subfun1 (y, z)
  x = subfun2(y, z);

function x = subfun2 (y, z)
  x = y + z + 2;
```

Help para las Funciones

- Podemos utilizar help con las funciones que creamos.
- Mostrará las primeras líneas del fichero que comienzan con el carácter %, es decir, son comentarios.
- Especialmente importante es la primera línea de comentarios (llamada línea H1). En ella tiene que estar la información más relevante sobre la función.
- La función *lookfor* busca una determinada palabra en cada primera línea de comentarios de todas las funciones .m

Reglas de Alcance

El buen uso de la programación modular requiere que las funciones sean independientes.

Esto se consigue intentando satisfacer dos condiciones:

- Cada módulo se diseña sin conocimiento del diseño de otros módulos
- La ejecución de un subprograma particular no tiene por que afectar a los valores de las variables de otros subprogramas.

Dado que se permite el anidamiento en la llamada a funciones, es necesario establecer mecanismos para evitar problemas con los identificadores definidos en varias partes del código.

Conceptos de **variables local**

variable global

Variables Locales

Son aquellas que se crean en el cuerpo de la función. Solo son "visibles" o "usables" dentro de la función donde se han creado.

Dos funciones diferentes, pueden utilizar los mismos nombres de variables sin "interferencias" ya que se refieren a posiciones diferentes de memoria.

```
>> k = 3;
>> cuadrado(k)
```

```
function [resultado] = cuadrado(n)
  k = n*n;
  resultado = k;
```

La variable *k* sigue valiendo 3 después de llamar a *cuadrado(k)*

Variables Globales

Son visibles en todas las funciones (y en el interprete) en las que se declaran como tales.

Estas variables se declaran precedidas por la palabra *global* y separadas por espacios en blanco:

global variable1 variable2

Estas variables SOLO son visibles en aquellas funciones (o espacios de trabajo) donde son definidas como tales.

Se les suele dar nombres largos y en mayusculas para distingüirlas fácilmente.

Ejemplo

```
>> global VAR;
>> VAR = 3;
>> ejemplo (4)
ans =
   12
```

```
function [resultado] = ejemplo (n)
   global VAR
   resultado = n * VAR;
```

Recomendaciones

- Escriba funciones como si fueran "cajas negras": el usuario debe saber QUÉ hace la función y no CÓMO lo hace.
- La primera línea de la función y un comentario adecuado deben ser suficientes para saber como usarla.
- Conceptos Relacionados: abstracción procedural, ocultamiento de información.

Ejercicios

- Defina una función que calcule el máximo de dos valores.
- Muestre como se puede utilizar la anterior para calcular el máximo de tres valores
- Defina una función que calcule la distancia entre dos puntos en el plano.
- Utilice la función anterior para calcular el perímetro de un cuadrilátero (definido mediante los 4 vértices)
- Implemente una función que permita dibujar cuadrados