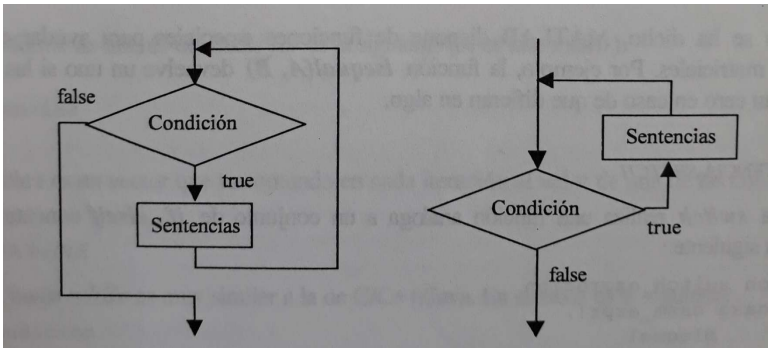
Programación en Scilab

Bifurcaciones y Bucles



Los **Bucles** permiten repetir una instrucción (u otra análoga) sobre diferentes datos siempre que se cumpla una dada condición **(control)**

Sentencia *While*

```
while   (boolean expression)
    instructions
end
```

➢ Tarea- solicitar al usuario que ingrese un valor x y escribir un progama que divida a "x" sobre 2 mientras x sea mayor que 1.

```
x = 16;
while ( x > 1 )
    x = x/2;
end
```

Clase 4

Sentencia *While*: Uso

Crear un script para averiguar si un número es menor a un valor dado

```
function menor_a()
    x=input("ingrese un numero real x=")
    x_max=input("ingrese el extremo mayor
x_max=")

    while x<x_max
        disp(x)
        x=x+1;
    end
endfunction
```

Sentencia *for*

```
for   variable = start: step: end

    instructions

end
```

```
n = 10;
for k = 1:n
    y(k) = exp(k);
end
```

```
// "traditional" for loops
n=5;
for i = 1:n
    for j = 1:n
        a(i,j) = 1/(i+j-1);
    end;
end
for j = 2:n-1
    a(j,j) = j;
end;
a
for j= 4:-1:1
    disp(j);
end // decreasing loop
```

Expresiones lógicas
operadores lógicos o Booleanos

Expresiones lógicas

operador	descripción
`&`	Y
`|`	O
`~`	No
`<`	Menor
`>`	Mayor
`<=`	Menor o igual
`>=`	Mayor o igual
`==`	Igual (el símbolo simple `=` se usa para asignar valor a variables)
`~=`	Distinto
`or(a<b)`	algún elemento de `a` es menor que el correspondiente de `b`
`and(a<b)`	todos los elementos de `a` son menores que los correspondientes de `b`

Clase 4

Expresiones lógicas
operadores lógicos o Booleanos

a & b *"Y (and) lógico"*
a | b *"O (or) lógico"*
~ a *negación (NOT) lógica*
a == b *"verdadero si las dos expresiones son iguales"*
a ~= b ó a <> b *"verdadero si las dos expresiones son distintas"*
a < b *"verdadero si a es menor que b"*
a > b *"verdadero si a es mayor que b"*
a <= b *"verdadero si a es menor o es igual que b"*
a >= b *"verdadero si a es mayor o es igual que b"*

Clase 4

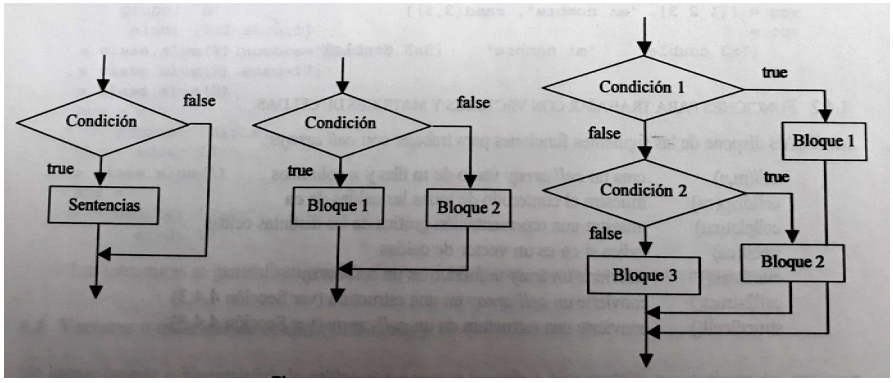
Variable Booleana

las variables booleanas pueden almacenar solo dos valores: *%t (%T)* (verdadero) o *%f (%F)* (falso)

```
-->a=%T
 a    =
  T
-->b  =  (  0  ==  1  )
 b    =
  F
-->a&b
 ans    =
  F
```

```
-->A  =  [1  2  7
-->6  9  8]
 A    =
     1.      2.      7.
     6.      9.      8.
-->A>3
 ans    =
  F  F  T
  T  T  T
```

Bifurcaciones y Bucles



Las **bifurcaciones** permiten una u otra operación según se **Cumpla** o **No se cumpla** una dada **condición.**

Declaración condicional if

```
if  expresión lógica  then
        instrucción 1
else
        instrucción 2
end
```

```
if (x>=0) then
    disp("x is positive");
else
    disp("x is negative");
end
```

Declaración condicional if

```
if  rand(1,1) > 0.5
then
        disp(“verdadero”)
;
else
        disp(“falso”);
end
```

Ejemplo:

```
i=2
for j = 1:3,
  if i == j then
    a(i,j) = 2;
  elseif abs(i-j) == 1 then
    a(i,j) = -1;
  else a(i,j) = 0;
  end,
end
```

```
if rand(1,1)>0.5 then disp(“verdadero”);  else disp(“falso”);  end
```

Declaración condicional *case*

la sentencia case provee forma estructurada para seleccionar entre diferentes opciones (case es el valor variable que se selecciona entre un conjunto de valores)

case

se usa para indicar "selección"

Descripción

esta declaración se usa en la instrucción:
select ..... case .... end

```
select expr0
    case expr1 then
        instructions 1
    case expr2 then
        instructions 2
      ...
    case exprn then
        instructions n
    [else instructions]
end
```

Declaración condicional *case*

Ejemplo con la estructura "case" ... "select"... end.

```
1
2  A = 10
3  select A
4  case 1 then
5      disp('hola')
6  case 2 then
7      disp('chau')
8  else
9      disp(':)')
10 end
```

- El comando "**select**" captura el valor de "A"
- La sentencia "**case**" compara el valor de "A" con el asignado a cada caso (case 1, case 2, etc)
- Se ejecuta la instrucción correspondiente al case cuyo valor coincide con el valor de A capturado por "**select**"

Declaración condicional *case*

Cree y ejecute el siguiente script:

Case_function.sce

```
1  A = 'a'
2  select A
3  case 'a' then
4      disp('hola')
5  case 'b' then
6      disp('chau')
7  else
8      disp(':)')
9  end
10
```

Scilab 6.0.1 Console

Clase 4

Funciones útiles en cálculo con matrices

`sum(A)`	suma de las componentes de la matriz **A**
`sum(A,1), sum(a;'r')`	es un vector fila (row) conteniendo la suma de los elementos de cada columna de **A**
`sum(A,2), sum(a;'c')`	es un vector columna conteniendo la suma de los elementos de cada fila de **A**
`trace(A)`	traza de A : **sum(diag(A))**
`prod(A)`	producto de las componentes de la matriz A
`prod(A,1), prod(A,'r')`	es un vector fila (row) conteniendo el producto de los elementos de cada columna de **A**
`prod(A,2), prod(A,'c')`	es un vector columa conteniendo el producto de los elementos de cada fila de **A**
`max(A)` `max(A,'r'), max(A,'c')`	máximo de las componentes de la matriz **A** máximos de columnas y filas resp. (como antes)
`mean(A)` `mean(A,'r'), mean(A,'c')`	media de las componentes de la matriz **A** medias de columnas y filas resp. (como antes)
`norm(v)` `norm(v,2)`	norma euclídea del **vector v**
`norm(v,p)`	norma-**p** del **vector v**: **sum(abs(v).^p)^(1/p)**
`norm(v,'inf')` `norm(v,%inf)`	norma infinito del **vector v:** **max(abs(v))**
`norm(A)` `norm(A,2)`	máximo autovalor de la **matriz A**
`norm(A,1)`	norma-**1** del **matriz A**: máximo entre las sumas de sus columnas: **max(sum(abs(A),'r'))**
`norm(A,'inf')` `norm(A,%inf)`	norma infinito de la **matriz A**: máximo entre las sumas de sus filas: **max(sum(abs(A),'c'))**
`size(A)`	devuelve, en un vector fila, las dimensionbes de la matriz A
`size(A,'r')` `size(A,'c')`	número de filas/columnas de la matriz A
`length(A)`	devuelve un escalar con el número de elementos de la matriz A: si A es una vector, length(A) es su longitud; si A es una matriz length(A) es el producto de sus dimensiones

Clase 4

Función "sum"

sum: suma los elementos de un vector o matriz.

Sintaxis:

y=sum(x); y=sum(x,"*") estas dos expresiones son idénticas, suma todos los elementos del vector o matriz

y=sum(x,*orientación*).

Orientación: puede ser "", "r", "c", o un número. Se usa para indicar la dimensión que se desea sumar en la matriz*

Ejemplo:

```
x=[1:100];
y=sum(x,"*");
disp(y)
```

```
sum(1:2:10)
```

Función modulo / pmodulo

modulo o **pmodulo:** calcula el resto de dividir dos números. retorna un "entero" (int)

Sintaxis:

i=modulo(n,m); i = n - m .* int (n ./ m) *(la respuesta puede ser positiva o negativa si n o m son negativos)*

i=pmodulo(n,m); i = n - |m| .* floor (n ./ |m|), la respuesta es positiva or cero.

```
mod(1,1)=modulo(-3, 9); mod(1,2)=modulo(10, -4)
mod(1,3)=pmodulo(-3, 9); mod(1,4)=pmodulo(10, -6)
mod(1,5)=pmodulo(-10, -6);
--> mod
 mod  =
  -3.   2.   6.   4.   2.
```

Función *int*

Descripción:

int(x) : devuelve la parte entera de la matriz real x, esto es, la parte entera de cada elemento de la matriz. Funciona igual que **fix**

Ejemplo -1:

```
int([1.3 1.5 1.7 2.5 3.7])
    ans  =
      1.   1.   1.   2.   3.
```

Ejemplo -2: Observe los resultados para x negativo

```
int([-1.3 -1.5 -1.7 -2.5 -3.7])
    ans  =
     -1.  -1.  -1.  -2.  -3.
```

Función *floor*

Descripción:

floor(x) : devuelve una matriz conformada por enteros resultantes de redondear al valor entero más pequeño

Ejemplo -1:

```
floor([1.3 1.5 1.7 2.5 3.7])
    ans  =
      1.   1.   1.   2.   3.
```

Ejemplo -2: Observe los resultados para x negativo

```
floor([-1.3 -1.5 -1.7 -2.5 -3.7])
    ans  =
     -2.  -2.  -2.  -3.  -4.
```