

Programación

# Estructuras repetitivas

Unidad 3

Jesús Alberto Martínez  
versión 0.2





Reconocimiento – NoComercial – CompartirIgual (by-nc-sa): No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.  
Basado en los apuntes del CEEDCV




# Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 Importante

 Atención

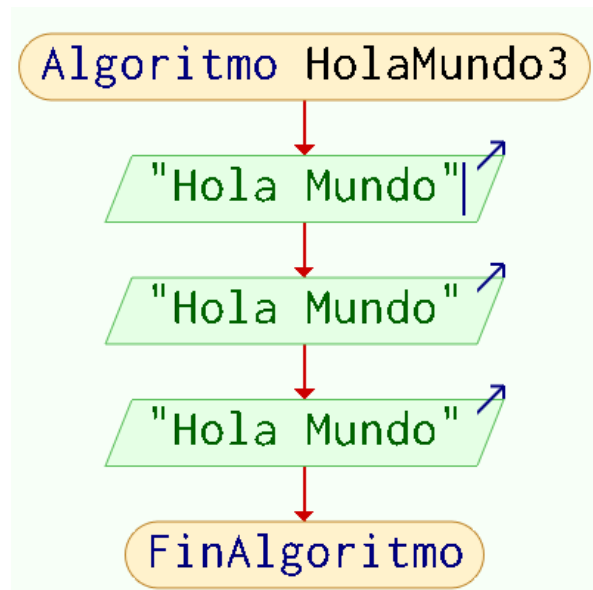
 Interesante

## Unidad 3. Estructuras repetitivas

1	Introducción.....	3
2	Estructura Mientras (WHILE).....	4
3	Estructura Repetir - Hasta (DO-WHILE).....	6
4	Estructura Para (FOR).....	8
5	Formas de acabar un bucle.....	9
6	Elementos auxiliares.....	12
6.1	Contadores.....	12
6.2	Acumuladores.....	12
6.3	Interruptores.....	13
7	Ejemplos.....	14
7.1	Ejemplo 1.....	14
	Ejemplo 2.....	15
	Ejemplo 3.....	16

# 1 Introducción

Imaginemos el siguiente problema, Muestra por pantalla 3 veces el texto "Hola Mundo". Con lo que sabemos hasta ahora, la solución sería:



Perfecto!! Ahora bien, ¿que pasaría si nos piden que mostremos el mensaje, por ejemplo, 100 veces? Para ayudarnos en estos problemas tenemos las estructuras repetitivas.



Las **instrucciones repetitivas (o bucles)** son aquellas que **permiten variar o alterar la secuencia normal de ejecución de un programa** haciendo posible que un grupo de operaciones (acciones) se **repita** un número determinado o indeterminado de veces, dependiendo del cumplimiento de una condición.

Veremos tres tipos: Bucle Mientras (WHILE), Bucle Hacer-Hasta (DO-WHILE) y Bucle Para (FOR)

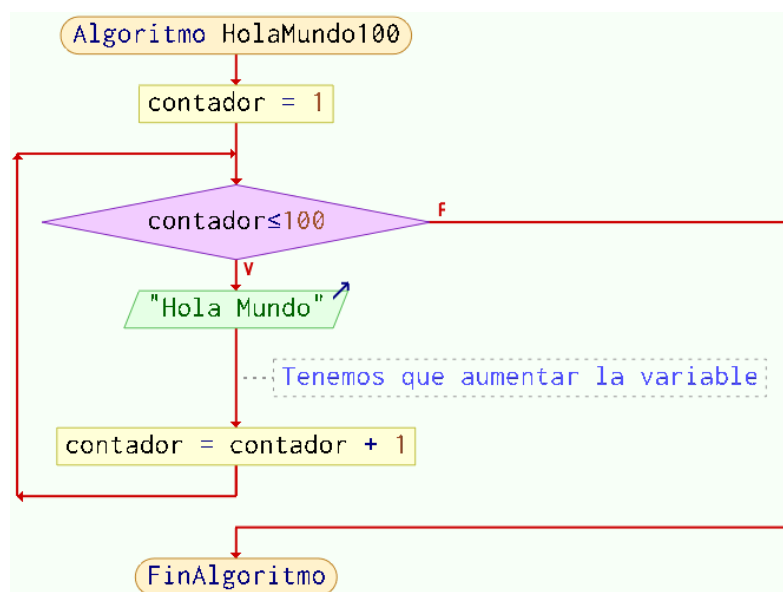
## 2 Estructura Mientras (WHILE)



En la estructura Mientras o “WHILE” el bloque de acciones **se repite mientras la condición del bucle sea cierta**, evaluándose siempre la condición antes de entrar en el bucle, por ello es posible que las acciones no se ejecuten nunca.

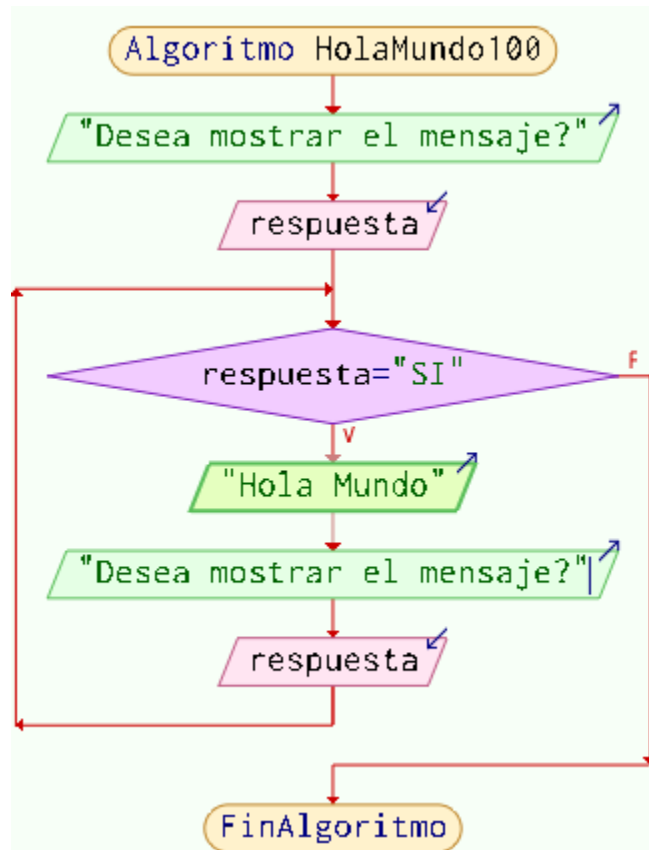
Pseudocódigo	Ordinograma
Mientras Condición Hacer Instrucción 1 Instrucción 2 ... Instrucción N FinMientras.	<pre> graph TD     Start(( )) --&gt; Cond{condición}     Cond -- V --&gt; Acc1[acción 1]     Acc1 --&gt; Acc2[acción 2]     Acc2 --&gt; Accn[acción n]     Accn --&gt; Cond     Cond -- F --&gt; Exit(( )) </pre>

Para solucionar nuestro pequeño problema, necesitamos contar las veces que queremos mostrar el texto “Hola Mundo”, necesitamos una variable, esa variable será la que establezca la condición para repetir o no mostrar el texto.



Ahora supongamos otro pequeño problema, supongamos que queremos mostrar el mensaje mientras que el usuario quiera, le preguntaremos si quiere mostrar el mensaje y, por ejemplo, si contesta "SI", mostraremos el mensaje, y volveremos a preguntarle, y así mientras que conteste "SI".

En este caso, no tenemos un número determinado de repeticiones, la condición para repetir en este caso es que el valor de una variable sea "SI" o sea otro valor. Tendremos que preguntar dentro del bucle si queremos continuar o no, pero también tendremos que preguntarlo antes de la primera repetición.



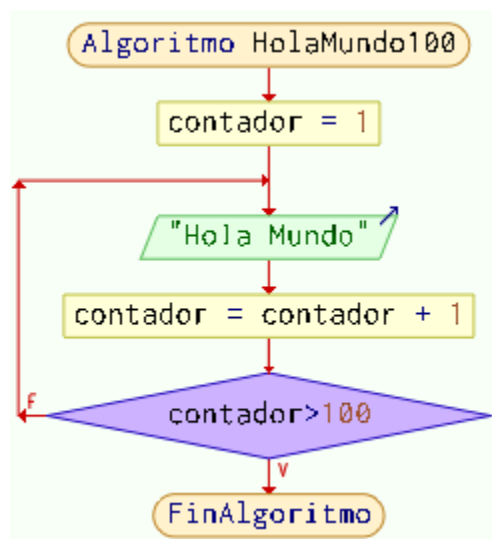
### 3 Estructura Repetir - Hasta (DO-WHILE)



En la estructura repetir - hasta o **"DO-WHILE"**, el bloque de instrucciones **se repite mientras que la condición sea cierta**, y la condición se evalúa al final del bloque por lo que siempre se ejecutarán al menos una vez el bloque de instrucciones.

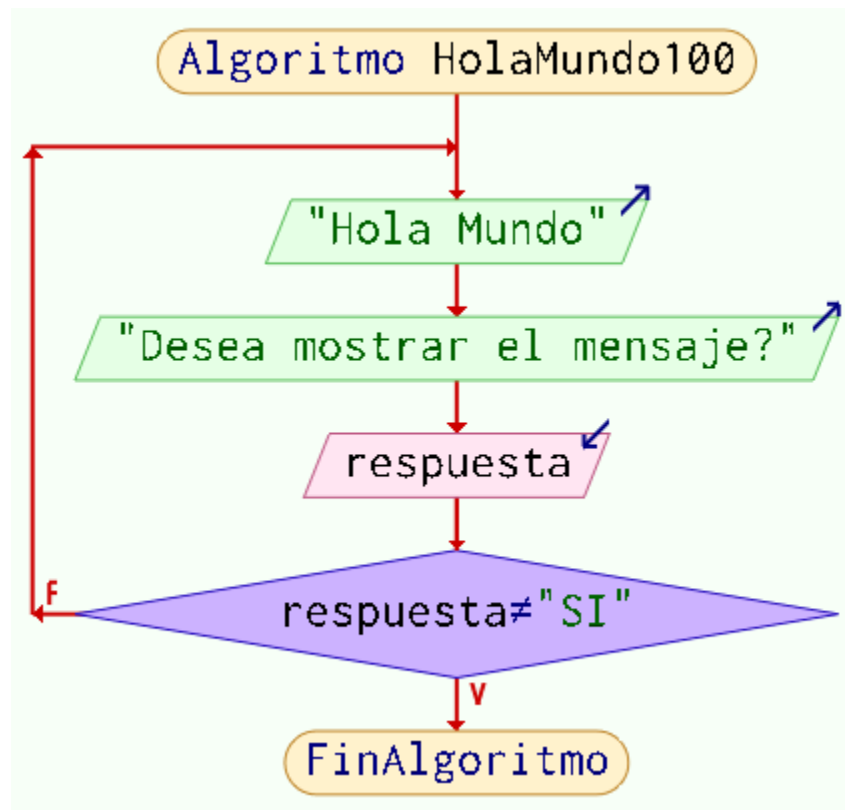
Pseudocódigo	Ordinograma
Repetir Instrucción 1 Instrucción 2 ... Instrucción N Mientras Condición.	

Vamos a ver nuestros dos ejemplos usando la estructura repetir-hasta



Podemos apreciar como la condición de finalización cambia respecto al bucle mientras.

En cuanto al segundo problema podemos apreciar otras diferencias.



En este caso, siempre se mostraré el mensaje "Hola Mundo" al menos una vez.

## 4 Estructura Para (FOR)

En la estructura Para o "FOR" se conoce de antemano el número de veces que se ejecutará el bloque de instrucciones.



El bloque de acciones **se repite mientras que la condición sea cierta, evaluándose siempre la condición antes de entrar en el bucle**, por ello es posible que las acciones no se ejecuten nunca.

Esta explicación es idéntica a la del bucle WHILE, pero un bucle FOR debe cumplir las siguientes características:

1. La variable contador se inicializa con un valor inicial.
2. La condición siempre debe ser:  $\text{variable\_contador} \leq \text{valor\_final}$
3. En cada interacción, la variable contador se incrementa en un determinado valor.

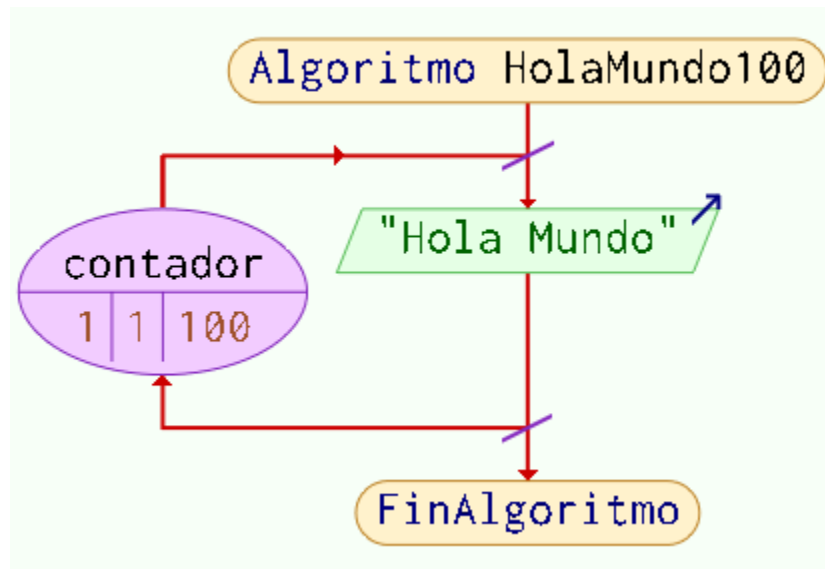
Pseudocódigo	Ordinograma
<pre> Para Var_Cont de ValorInicial a ValorFinal con Incremento = n   Instrucción 1   Instrucción 2   ...   Instrucción N FinPara           </pre>	

Es equivalente a

Pseudocódigo	Ordinograma
<pre> variable=ValorInicial Mientras variable&lt;=ValorFinal   Instrucciones   variable=variable+Incremento FinMientras           </pre>	



Volviendo a nuestros pequeños ejemplos.



¿Cuál de las 3 versiones os gusta más?

En cuanto al segundo problema, la estructura Para no sería la adecuada, y dependiendo del lenguaje de programación seríamos capaces o no de utilizarla.

## 5 Formas de acabar un bucle

Uno de los peligros que se corren cuando se escribe un bucle es que éste no acabe nunca, lo que se denomina **bucle infinito**, para no cometer este error grave debemos recordar que las condiciones de los bucles deben poder cambiar dentro del bucle, es decir que si por ejemplo utilizamos una variable comparada con una constante, dicha variable debe poder cambiar de valor dentro del bucle.

⚡ Las estructuras repetitivas deben incluir un mecanismo para que éstas se acaben.

Algunos de los métodos más usados para esa labor son los siguientes:

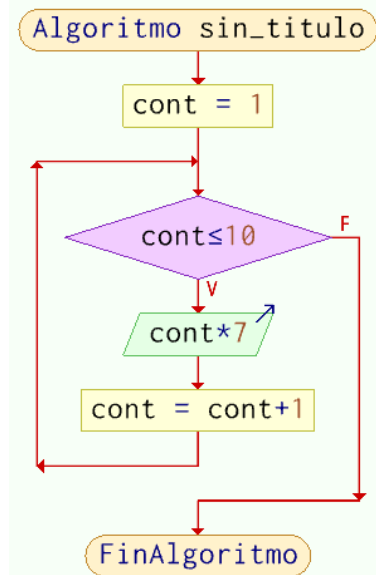
1. **Cuando sabemos el número de veces que se va a repetir la estructura, utilizaremos un contador.**

Por ejemplo, en un enunciado del tipo: "imprimir la tabla del 7", sabemos que el proceso va desde 1 a 10, por lo tanto, usaremos un contador.

```

cont = 1
Mientras cont <=10
    Escribir cont * 7
    cont = cont + 1
FinMientras

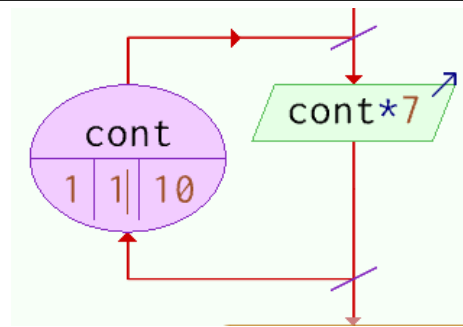
```



```

Para cont de 1 a 10 Hacer
    con incremento = 1
    Escribir cont*7
FinPara

```



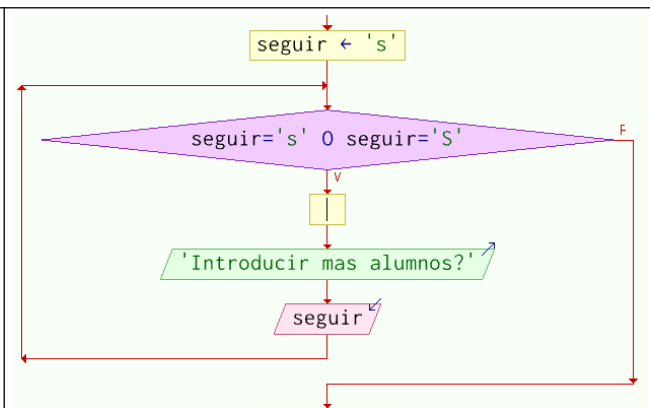
## 2. Preguntando si queremos seguir en el bucle.

Por ejemplo, en un ejercicio del tipo “introducir N alumnos y hallar su media”, debemos preguntar si queremos introducir más alumnos:

```

...
seguir="s"
Mientras ((seguir="s") o (seguir="S"))
    ...
    Escribir "Introducir más alumnos?"
    Leer seguir
FinMientras
...

```



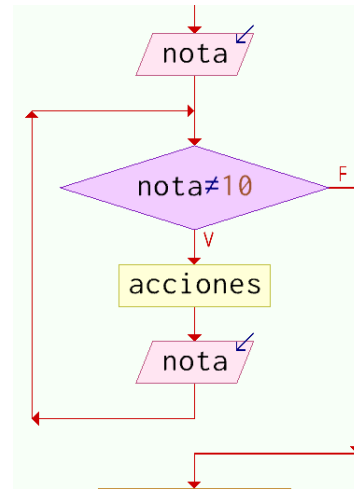
### 3. Usando un valor centinela.

Así, en el caso del siguiente problema: "Introducir N notas hasta introducir un 10":

```

. . .
Leer nota
Mientras (nota <> 10)
    . . .
    Leer nota
FinMientras
. . .

```



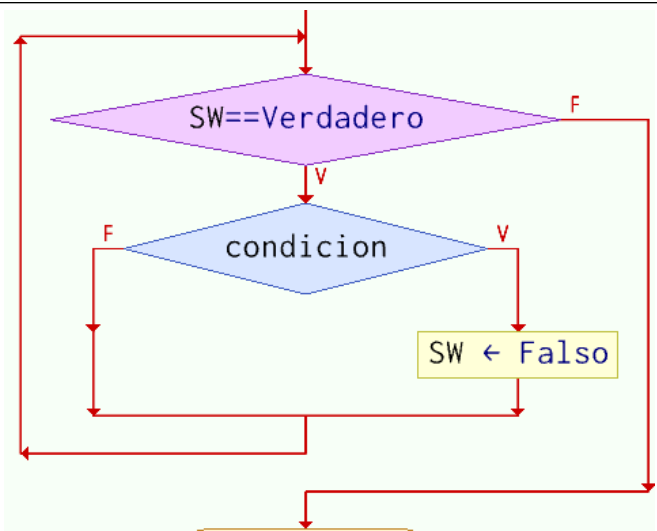
### 4. Usando un interruptor que tomará el valor lógico true o false.

En un ejemplo como "Repetir ciertas instrucciones mientras la condición sea cierta":

```

. . .
Mientras (SW = Verdadero)
    . . .
    si (condicion) SW ← Falso
FinMientras
. . .

```



## 6 Elementos auxiliares

---

Los **elementos auxiliares** son **variables que realizan funciones específicas dentro de un programa**, y por su gran utilidad, frecuencia de uso y peculiaridades, conviene hacer un estudio separado de las mismas.

### 6.1 Contadores

---

Si vamos a repetir una acción un número determinado de veces y esa variable se va a incrementar siempre en una cantidad constante, se denomina **contador**. Sería útil llamarla algo así como CONT, CONTA, CONTADOR... Si tuviéramos varios contadores dentro de un programa podríamos llamarlos CONT1, CONT2...

Se utilizan en los siguientes casos:

- Para contabilizar el número de veces que es necesario repetir una acción (variable de control de un bucle).
- Para contar un suceso particular solicitado por el enunciado del problema. Un contador debe inicializarse a un valor inicial (normalmente a cero) e incrementarse cada vez que ocurra un suceso.

### 6.2 Acumuladores

---

Si por el contrario, dicho objeto se va **incrementando de forma variable** se denomina **acumulador**. Deberemos llamarla ACU, ACUM, ACUMULA, ACUMULADOR, SUMA, ... u otra palabra significativa.

Se utiliza en aquellos casos en que se desea obtener el total acumulado de un conjunto de cantidades, siendo inicializado con un valor cero.

También en ocasiones hay que obtener el total acumulado como producto de distintas cantidades, en este caso se inicializará a uno.

Por ejemplo: imprimir la suma de N edades.

## 6.3 Interruptores

---

Por último, tenemos ciertas variables que pueden **tomar dos valores: cierto o falso**. Se les denomina **interruptores o switches** y su función es que ciertas instrucciones se ejecuten mientras tenga un valor determinado. A las variables de este tipo las denominaremos SW.

Se utiliza para:

- Recordar que un determinado suceso a ocurrido o no en un punto determinado del programa, y poder así realizar las decisiones oportunas.
- Hacer que dos acciones diferentes se ejecuten alternativamente dentro de un bucle.

Por ejemplo: introducir N edades y acabar al introducir un 99.

## 7 Ejemplos

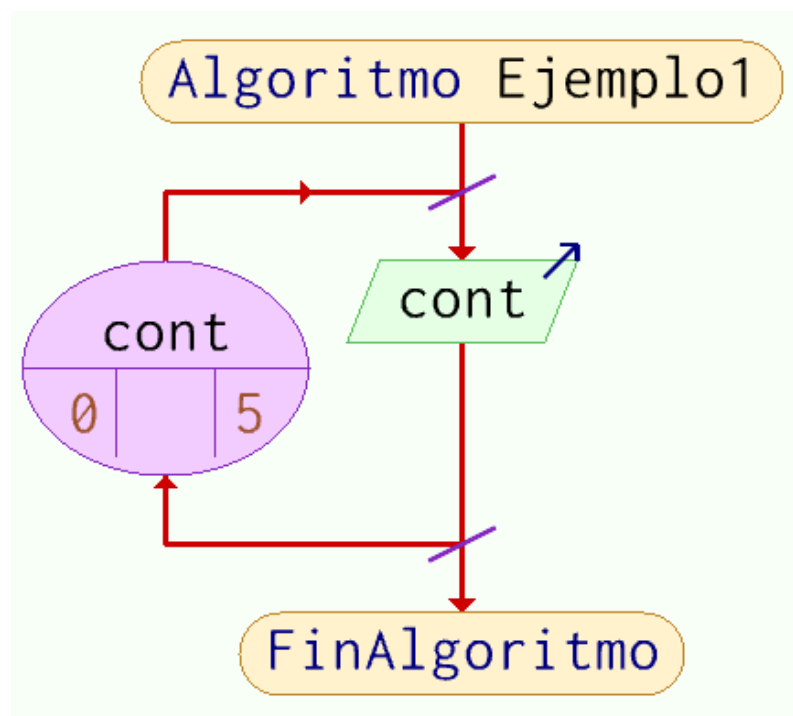
### 7.1 Ejemplo 1

En el primer ejemplo vamos a ver como crear un programa que muestre por pantalla desde el número 0 hasta el 5.

Para un caso como éste, es interesante utilizar la estructura PARA (FOR) ya que conocemos el número de veces que se va a ejecutar el bloque de instrucciones. En este caso 6 porque vamos desde el 0 hasta el 5.

Tenemos que tener en cuenta que:

- El bloque se repite mientras la condición sea cierta.
- La condición se evalúa siempre antes de entrar en el bucle.
- La variable contador se inicializa con el valor inicial. En este caso 0
- La condición es:  $\text{cont} \leq \text{valor\_final}$
- En cada iteración la variable contador se incrementa en un determinado valor. En este caso 1

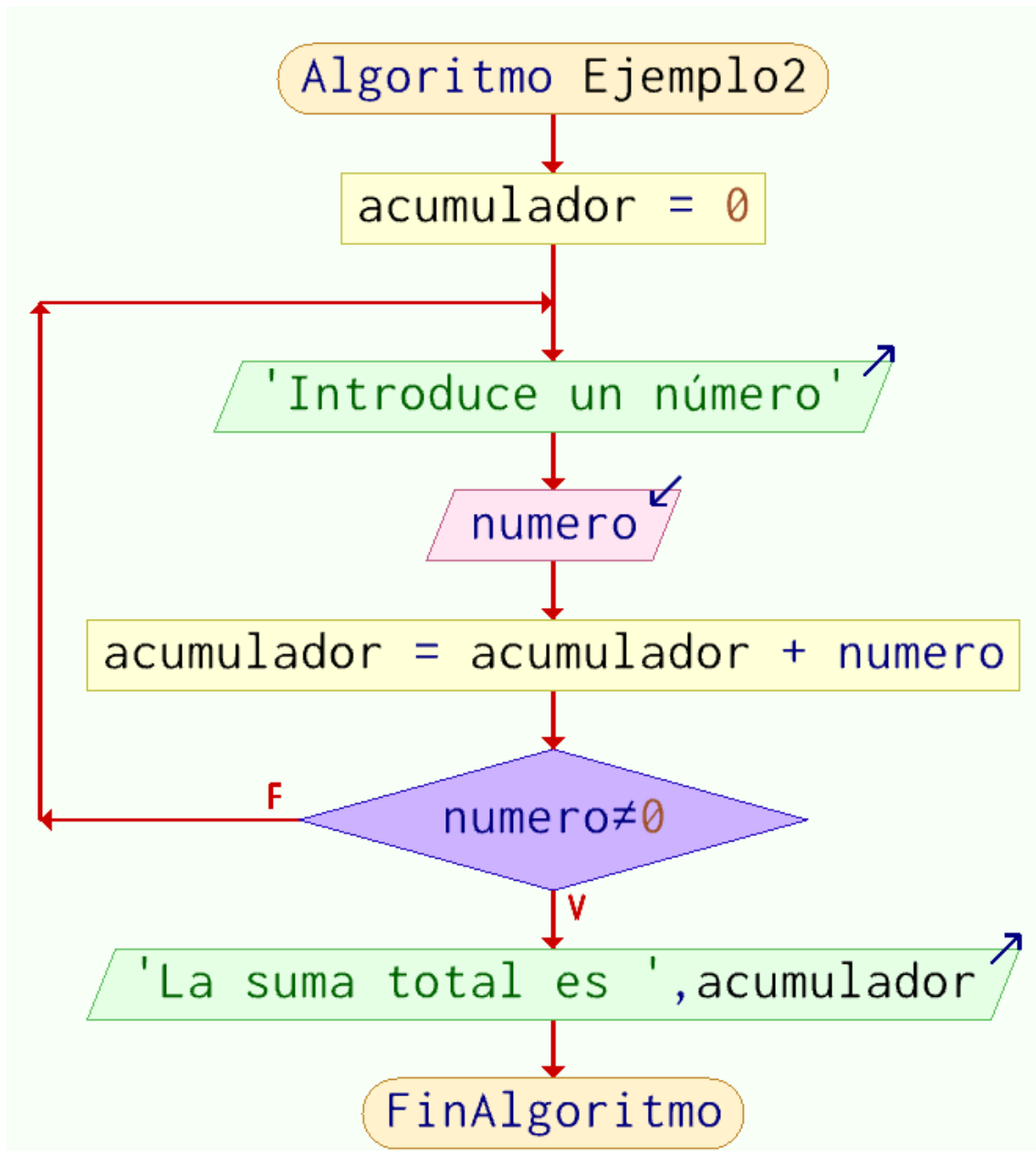


## Ejemplo 2

En el segundo ejemplo vamos a crear un sencillo programa que pida por teclado al usuario números hasta que éste introduzca un 0 y al final nos muestre la suma de los números introducidos.

Para este caso se utilizaría una estructura HASTA (DO-WHILE) ya que primero le pedimos al usuario el número y después comprobamos si es distinto de 0.

Y usaremos una variable de tipo acumulador para obtener la suma final.



## Ejemplo 3

En el tercer ejemplo vamos a crear un sencillo programa que calcule la suma de los tres primeros números (del 1 al 3).

Utilizamos un acumulador para obtener el resultado del sumatorio.

