

Elementos y Estructuras de Control

1ºDAM IoT

Programas y algoritmos

Un **programa** es una secuencia de instrucciones que son ejecutadas por la CPU para realizar una tarea determinada.

Esta secuencia de instrucciones se escribe en un lenguaje de programación siguiendo una determinada estructura y orden.

La forma en la que se organizan las instrucciones del programa responde a una estrategia o algoritmo orientada a resolver una determinada tarea.

Un **algoritmo** es una descripción ordenada y sistemática de los pasos necesarios para realizar una tarea.

Algoritmos y programas

Los algoritmos que resuelven un problema no suelen ser únicos. Los siguientes dos algoritmos resuelven el problema de ordenar una serie de números de menor a mayor.

OrdenarBurbuja(lista):

 n := longitud(lista)

 // Iterar sobre la lista

 para i desde 0 hasta n - 1:

 // Últimos i elementos ya están en su lugar correcto

 para j desde 0 hasta n - i - 1:

 // Si el elemento actual es mayor que el siguiente,

 //intercambiarlos

 si lista[j] > lista[j + 1]:

 intercambiar(lista[j], lista[j + 1])

Ordenar(lista):

 para i desde 0 hasta longitud(lista) - 1:

 // Suponemos que el elemento actual es el mínimo

 minimo := i

 para j desde i + 1 hasta longitud(lista):

 // Si encontramos un elemento más pequeño

 // actualizamos minimo

 si lista[j] < lista[minimo]:

 minimo := j

 // Intercambiamos el elemento actual

 // con el mínimo encontrado

 si i != minimo:

 intercambiar(lista[i], lista[minimo])

Algoritmos y programas

Para implementar un programa, es necesario crear, en primer lugar, un algoritmo o estrategia.

Esto permitirá especificar claramente cuál será la tarea a realizar y el modo de realizarla.

Una vez que el algoritmo (la estrategia) está planteado, lo concretamos en una secuencia de instrucciones de un lenguaje de programación.

Algoritmos y programas

Ejemplo en C# de uno de los algoritmos de ordenación anteriores.

```
static void OrdenarBurbuja(int[] lista)
{
    int n = lista.Length;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (lista[j] > lista[j + 1])
            {
                // Intercambiar los elementos si están en el orden incorrecto
                int temp = lista[j];
                lista[j] = lista[j + 1];
                lista[j + 1] = temp;
            }
        }
    }
}
```

Algoritmos y programas

Ejemplo en PHP de uno de los algoritmos de ordenación anteriores.

```
function ordenarBurbuja($arr) {  
    $n = count($arr);  
    for ($i = 0; $i < $n - 1; $i++) {  
        for ($j = 0; $j < $n - $i - 1; $j++) {  
            if ($arr[$j] > $arr[$j + 1]) {  
                // Intercambiar los elementos si están en el orden incorrecto  
                $temp = $arr[$j];  
                $arr[$j] = $arr[$j + 1];  
                $arr[$j + 1] = $temp;  
            }  
        }  
    }  
    return $arr;  
}
```

Algoritmos y programas

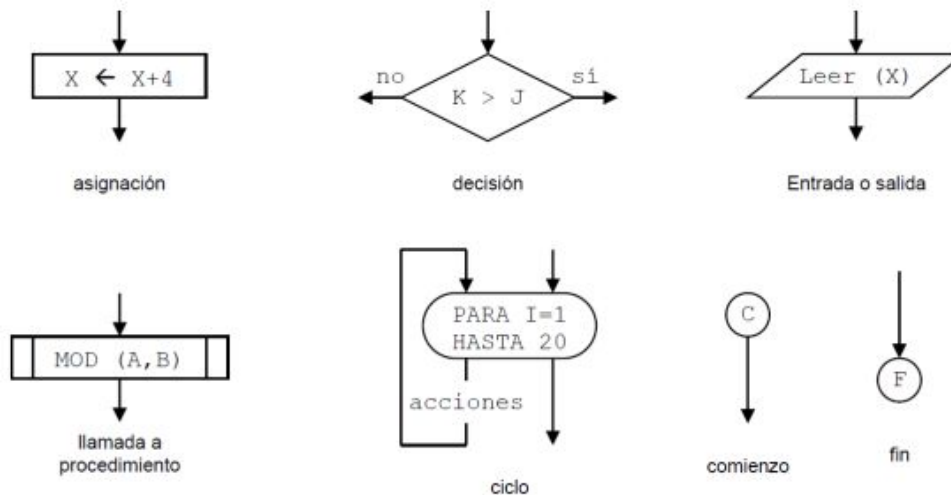
Un algoritmo se puede representar mediante algún tipo de representación gráfica o bien mediante una notación especial denominada pseudocódigo, como veremos en los apartados siguientes.

Las ventajas de representarlo bajo cualquiera de estas formas son:

- Permite tener una guía de la secuencia de pasos a implementar, que puede ser consultada durante el proceso de implementación. La implementación de un programa no suele ser una tarea trivial.
- Obtenemos una representación universal, que es independiente del lenguaje de programación y puede ser traducida, por tanto, a cualquier lenguaje.

Algoritmos y programas

La representación gráfica de algoritmos se suele realizar mediante diagramas de flujo, en los que se combinan los siguientes elementos para detallar la secuencia de pasos del algoritmo.



Diseño de algoritmos

1. Identificar tareas principales.
2. Ordenación de las tareas.
3. Especificar y concretar subtareas.
4. Implementar (codificar) el algoritmo.
5. Ejecución.
6. Depuración.

Elementos dentro del código de un programa

- Comentarios
- Identificadores
- Palabras reservadas
- Variables
- Tipos de datos
- Constantes
- Literales
- Operadores
- Expresiones
- Estructuras de control en programación estructurada
 - Secuencia de instrucciones.
 - Estructuras condicionales.
 - Estructuras repetitivas (bucles).

Comentarios

Permiten realizar anotaciones sobre el código fuente.

Parte fundamental de la documentación técnica.

Todo programador debe incluirlos en los programas que desarrolle.

Finalidad:

- Aportar información sobre el código que se está desarrollando.
- Ayudar a otros programadores o a uno mismo.

Dos tipos:

- De línea: Suele usarse “//”.
- De bloque: /* y */ en C# y Java.

Comentarios

Ejemplo de comentario en línea en C#.

```
// My comments about the class name could go here...  
class Program  
{  
    .....
```

Ejemplo de comentario en bloque en C#.

```
/*  
My comments about the class name could go here...  
Add as many lines of comments as you want  
    ...and use indentation, if you want to!  
*/  
class Program  
{  
    .....
```

Identificadores

Son los “nombres” que el programador asigna a: variables, subprogramas o incluso el propio programa.

La mayoría de lenguajes imponen unas restricciones:

- Deben comenzar por una letra.
- Pueden continuar con más letras, números o el carácter “_”.
- Longitud mínima de un carácter.
- No valen espacios ni caracteres especiales @, #, !, etc.

En la mayoría de lenguajes, el empleo de tildes o la letra ‘ñ’ no se permite, pero aunque se permita no es recomendable.

Debemos procurar que:

- Tengan una longitud no demasiado larga, pues podría ser engorroso su uso.
- Sean representativos de su contenido o uso (no llamar “color” a una variable que almacenará una temperatura, por ejemplo).

Ejemplos no válidos → 2x, altura max, altura-max

Ejemplos válidos → prog1, area, base, altura

Palabras reservadas

Nombres que NO pueden utilizarse como identificadores.

El lenguaje de programación los necesita para instrucciones o elementos del lenguaje.

Cada lenguaje tiene su propio conjunto de palabras reservadas:

- Ejemplos en Pascal → program, var, begin, end...
- Ejemplos en C# → for, while, int, string...

Si usamos alguna palabra reservada el IDE (programa que usamos para programar) nos la marcará cómo no válida.

También nos dará un error al compilar el programa.

Variables

Cualquier programa maneja internamente datos sobre los que realiza operaciones.

Para poder manejar esos datos en el código de un programa, éstos deben almacenarse temporalmente en **variables**.

Una variable de un programa actúa del mismo modo que una variable matemática en una ecuación: almacena un dato perteneciente a un tipo determinado.

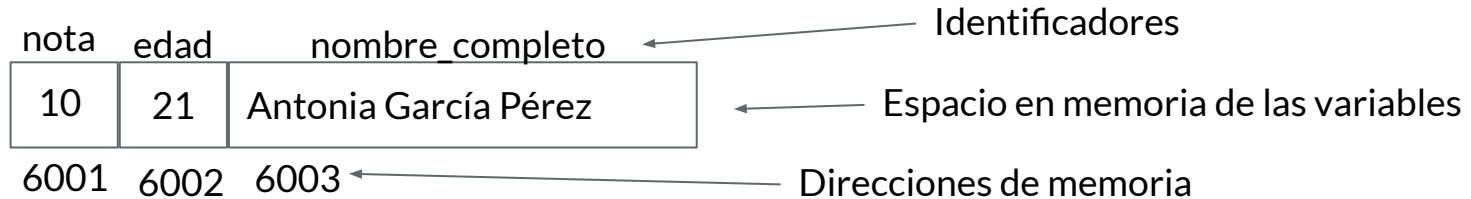
Las variables son nombradas con un identificador.

El programa inicializa una variable, cambia su valor, opera con ella, etc.

Variables

Una variable no es más que una ubicación en la memoria principal del ordenador, en la que se almacena un dato.

Esa ubicación de la memoria tiene una dirección asignada, pero, dado que trabajar con direcciones de memoria resulta complejo y puede conducir a errores, los lenguajes actuales permiten sustituir el empleo de direcciones de memoria por el empleo de identificadores o nombres de variable, que hacen referencia a esa dirección de memoria, pero utilizando un nombre, que es más fácil de manejar en el código.



Constantes

Son variables cuyo valor no puede ser alterado.

Se les da valor al definirlas y dicho valor es el mismo durante toda la ejecución del programa .

Pueden usarse para imponer restricciones y para aquellos valores que son siempre fijos (símbolo π de matemáticas siempre tiene un valor de 3,14, el IVA...).

Ventajas: Al indicar su valor en un único punto del programa es sencillo cambiarla para modificar el comportamiento del programa.

Normalmente se escriben en mayúsculas para diferenciarlas de las variables.

No todos los lenguajes de programación soportan el uso de constantes.

Tipos de datos

Un tipo de datos define la naturaleza y limitaciones de los valores que puede representar una variable.

- Naturaleza → número entero, número en coma flotante, un texto, un carácter, un valor booleano (verdadero o falso), etc.
- Limitaciones → longitud máxima de un texto, valor mínimo o máximo de un número, etc.
- Operaciones → qué operaciones pueden realizarse sobre él, por ejemplo si es un tipo de datos numérico tendremos los operadores de la suma, resta, división, etc.

Dependiendo del lenguaje, puede ser necesario “presentar” y decir de qué “tipo es” una variable antes de poder usarla.

- **Declaración de la variable:** `PSelnt` → *Definir num1 Como Entero;*

En algunos lenguajes (C, C#, Java, etc.) se puede dar un valor inicial a la variable en su declaración. → **Inicialización**

Cuando declaro e inicializo a la vez estamos haciendo una **definición**.

Tipos de datos

Según la tipificación nos encontramos otra clasificación de los lenguajes de programación:

- **Lenguajes tipados:** Al declarar una variable hay que indicar de qué tipo es, quedando así restringido el conjunto de valores posible y las operaciones que es posible realizar con dicha variable.
 - Ejemplo: Pascal, Java, C, C#, etc.
- **Lenguajes no tipados:** Prácticamente no hay que declarar las variables ni decir de qué tipo son. La primera aparición que hagan en el código deberá ser una asignación y, a partir de ese momento, quedarán declaradas y serán del tipo correspondiente al valor que reciben. Lo que hace el traductor o compilador es averiguar de que tipo es la variable dependiendo del valor asignado.
 - Ejemplo: Javascript, PHP, etc.

Variables

Ejemplos de variables en C#:

```
int num1 = 5; //Declaración con inicialización  
int num2; //Declaración  
num1 = num1 * 2; //Multiplico por 2  
num2 = num1+6; //Inicializo num2
```

Variables

Ejemplo de variable en PHP:

```
<?php
// Declarar una variable llamada "miVariable" y asignarle el valor 42
$miVariable = 42;

// Imprimir el valor de la variable
echo "El valor de miVariable es: " . $miVariable;
?>
```

Literales

Caso particular de constantes que no requieren ser definidas:

- Se usan directamente sobre el código.
- Son cualquier valor que pongamos directamente en el código.

Los **literales numéricos** se indican tal cual, sin separador de miles y usando el punto como separador decimal.

Los **literales para textos** se encierran entre comillas, pueden ser simples y/o dobles dependiendo del programa.

- En Pascal → 'esto es un literal'.
- En C, Java o C# → "esto es un literal".
- En PHP o Javascript se pueden usar ' ' o " ".

Operadores

Son signos especiales que permiten realizar cálculos sobre operandos que intervienen.

Los operandos pueden ser: literales, variables, constantes, funciones o cualquier otra cosa capaz de formar una expresión.

Cada tipo de dato soporta un conjunto especial de operadores.

Operadores

- **Operadores aritméticos:** operan sobre dos datos de cualquier tipo numérico y permiten realizar las operaciones aritméticas fundamentales sobre datos de tipo numérico: suma (+), resta (-), producto (*), división entera o decimal (/), y módulo o resto (%).
- **Operadores relacionales:** permiten establecer comparaciones entre dos datos. Los operadores relacionales más importantes son: igualdad (==, =), distinto (!=, <>), menor (<), mayor (>), menor o igual (<=) o mayor o igual (>=).
- **Operadores lógicos:** permiten realizar operaciones sobre datos de tipo booleano. Los operadores lógicos son: AND (&&), OR (||), NOT (!) y XOR (^). AND y OR son operadores duales (requieren dos datos) pero NOT y XOR son unarios (se aplican a un único dato).

Operadores

- **Operadores de asignación y reasignación:** permiten establecer o reasignar el valor de una variable. Son los siguientes: asignación (=, :=) y pre/post incremento/decremento (los estudiaremos más adelante).
- **Operador de concatenación de cadenas:** se utiliza para unir dos cadenas, formando una nueva cadena. Opera sobre dos datos de tipo string (texto, cadena). Es el operador (+).

Existen más operadores que iremos viendo conforme avance el módulo, y definirlos ahora no tiene sentido sin unos conocimientos más avanzados.

Operadores relacionales

Los operadores relacionales al procesarlos devuelven si es verdadero(true) o falso(false).

Por ejemplo,

- $5 > 6 \rightarrow$ Verdadero, true.
- $2 > 10 \rightarrow$ Falso, false.
- $3 == 2 \rightarrow$ Falso, false.

Operadores relacionales

También podremos evaluar expresiones que contienen variables.

Por ejemplo, tenemos dos variables num1 que es igual 5, num2 que es igual 3 y num3 que es igual a 10.

Tendremos algo así declarado:

```
num1=5;  
num2=3;  
num3=10;
```

¿Qué resultado obtendremos?

num1>num2 →

num2>=num3 →

num3<>num2 o num3!=num2 →

Operadores relacionales

También podremos evaluar expresiones que contienen variables.

Por ejemplo, tenemos dos variables num1 que es igual 5, num2 que es igual 3 y num3 que es igual a 10.

Tendremos algo así declarado:

```
num1=5;  
num2=3;  
num3=10;
```

¿Qué resultado obtendremos?

num1>num2 → **True**

num2>=num3 → **False**

num3<>num2 o num3!=num2 → **True**

Operadores lógicos

AND, &&

Operando 1	Operador AND	Operando 2	Resultado
True	AND (&&)	True	True
True	AND (&&)	False	False
False	AND (&&)	True	False
False	AND (&&)	False	False

Operadores lógicos

OR, II

Operando 1	Operando OR ()	Operando 2	Resultado
True	OR ()	True	True
True	OR ()	False	True
False	OR ()	True	True
False	OR ()	False	False

Operadores lógicos

not, !

Operando	Resultado
True	False
False	True

Operadores lógicos

Ejercicio:

true AND true =

false OR true =

not true =

Operadores lógicos

Ejercicio:

true AND true = false

false OR true = true

not true = false

Precedencia de operadores

El orden de evaluación de una expresión que contenga más de un operador es de izquierda a derecha.

Primero se tendrán en cuenta los paréntesis. ()

Operadores relacionales.

Operadores lógicos.

Por ejemplo,

$3 > 4 \text{ and } 1 == 1 = \text{false and true} = \text{false}$

$\text{not}(a < 2)$ donde a es igual 2 $\rightarrow \text{not}(\text{false}) \rightarrow \text{true}$

Precedencia de operadores

Ejercicio: sabiendo que el valor de a, b, c y d es el siguiente evalúa si las expresiones devuelven verdadero (true) o falso (false).

a=2, b=3, c=10, d=3

a==2 and (c>=3 or c<=10) =

a>b or (c>0 and d>0) =

not(a<b and c>d) =

Precedencia de operadores

Ejercicio: sabiendo que el valor de a, b, c y d es el siguiente evalúa si las expresiones devuelven verdadero (true) o falso (false).

a=2, b=3, c=10, d=3

$a==2 \text{ and } (c>=3 \text{ or } c<=10) = \text{true and } (\text{true or true}) = \text{true and true} = \text{true}$

$a>b \text{ or } (c>0 \text{ and } d>0) = \text{false or } (\text{true and true}) = \text{false or true} = \text{true}$

$\text{not}(a<b \text{ and } c>d) = \text{not } (\text{true and true}) = \text{not}(\text{true}) = \text{false}$

Conversiones de tipo

Algunos tipos de datos pueden tener cierta “compatibilidad”

- Cualquier dato numérico podría convertirse a un texto.
- Un número entero puede convertirse a un número en coma flotante, etc.

Los lenguajes de programación suelen proporcionar una o dos de estas técnicas.

- Casting: consiste en agregar un texto a una expresión para forzar que el tipo resultante se convierta a uno concreto.
 - `(string) 3+2` → se interpreta como “5” no como 5
 - `((string)3+2)+”1”` → da como resultado “51” y no 6
- Métodos de conversión → subprogramas que aceptan una expresión como entrada y devuelven el resultado convertido a un tipo dado.
 - `Convert.ToInt32(“123”)` → da como resultado 123, convertido a entero

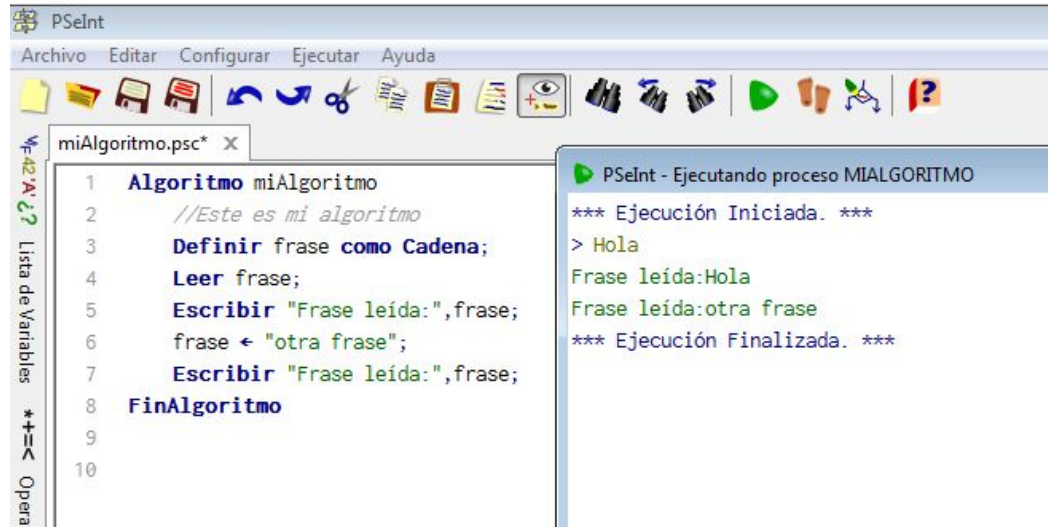
Algunas conversiones no se pueden realizar

- “hola” no tiene un valor interpretable como número.
- Un número real no se puede convertir a entero sin perder precisión

PSeInt

Los primeros conceptos de programación que estamos viendo los vamos a probar en PSeInt.

Todos odo estricto: Menú → Configurar → Opciones del Lenguaje (Perfiles) → Estricto → Aceptar



The screenshot shows the PSeInt IDE interface. The main window displays a Pascal script named 'miAlgoritmo.psc' with the following code:

```
1  Algoritmo miAlgoritmo
2      //Este es mi algoritmo
3      Definir frase como Cadena;
4      Leer frase;
5      Escribir "Frase leída:",frase;
6      frase ← "otra frase";
7      Escribir "Frase leída:",frase;
8  FinAlgoritmo
9
10
```

On the left side, there is a vertical toolbar with icons for file operations, editing, and execution. Below the code editor, there is a 'Lista de Variables' (List of Variables) panel.

On the right side, a console window titled 'PSeInt - Ejecutando proceso MIALGORITMO' shows the execution output:

```
*** Ejecución Iniciada. ***
> Hola
Frase leída:Hola
Frase leída:otra frase
*** Ejecución Finalizada. ***
```

PSeInt

Para definir las variables: **DEFINIR** *nombreVariable* **COMO ENTERO**;

Asignar un valor a variable: *nombreVariable* **<-** valor;

Leer la entrada de teclado de usuario: **LEER** *nombreVariable*;

Escribir por pantalla:

ESCRIBIR "Un texto solo", *nombreVariable*;

ESCRIBIR "Un texto", *nombreVariable*;

ESCRIBIR *nombreVariable*;

Operaciones relacionales

Algoritmo sin_titulo

Definir numero1 Como Entero;

Definir numero2 Como Entero;

Definir resultado Como Entero;;

numero1 ← 10;

numero2 ← 5;

resultado ← numero1 + numero2;

La variable resultado guarda la suma de numero1 y numero2.

Escribir numero1;

Escribir numero2;

¿Qué valores se imprimen?

Escribir resultado;

resultado ← (resultado - 2) + numero1;

El valor de la variable resultado cambia tras la operación

Escribir numero1;

Escribir numero2;

Escribir resultado;

¿Qué valores se imprimen?

FinAlgoritmo

Tarea - 1

Estructuras de control en programación estructurada

Cualquier programa puede escribirse a base de hacer un uso combinado de las tres estructuras de control de la programación estructurada:

- Secuencia.
- Condicionales.
- Repetición.

Secuencia de instrucciones

Se basa en establecer las instrucciones de un programa formando una secuencia, una tras otra de forma que se ejecutan en un determinado orden.

Características

- No se producen pausas entre instrucciones.
- La ejecución de una secuencia no empieza hasta que acaba la anterior.
- Una vez finalizada la última instrucción, la secuencia se da por terminada.

En un programa podemos encontrar una o varias secuencias de instrucciones. Algunas pueden estar compuestas por una única instrucción.

Cada instrucción en una línea por lo general y terminada en “;”.

Delimitar las secuencias de instrucciones

- Pascal → “begin” y “end”
- C, Java, C#, pHp, Javascript → “{” y “}”

Condicionales

Determinadas partes del código de un programa (a las que podemos llamar bloques, pero que no dejan de ser secuencias) tienen la posibilidad de ejecutarse o no según el resultado de evaluar una expresión de tipo booleano, esto es, que da como resultado “verdadero” o “falso” tras ser evaluada.

La mayoría de lenguajes implementan esta estructura usando como palabras reservadas IF, THEN, ELSE (SI, ENTONCES, SINO).

Ejemplo:

“SI el valor de la variable “temperatura” es mayor de 100, ENTONCES advertir de que el agua se va a evaporar, SI NO, SI el valor de la variable “temperatura” es mayor de 50, ENTONCES advertir de que el agua está muy caliente.”

Condicionales

Tipos de condicionales:

- Condicional Simple
- Condicional Doble
- Condicional Múltiple

Condicional simple

Aquella en la que se evalúa la condición al inicio de un bloque.

Si la evaluación devuelve “verdadero”, el bloque se ejecuta.

Si devuelve “falso”, no se ejecuta.

```
...  
SENTENCIA1  
SENTENCIA2  
SI (expresión lógica) ENTONCES  
    SENTENCIA3  
    SENTENCIA4  
FINSI  
SENTENCIA5  
SENTENCIA6  
...
```

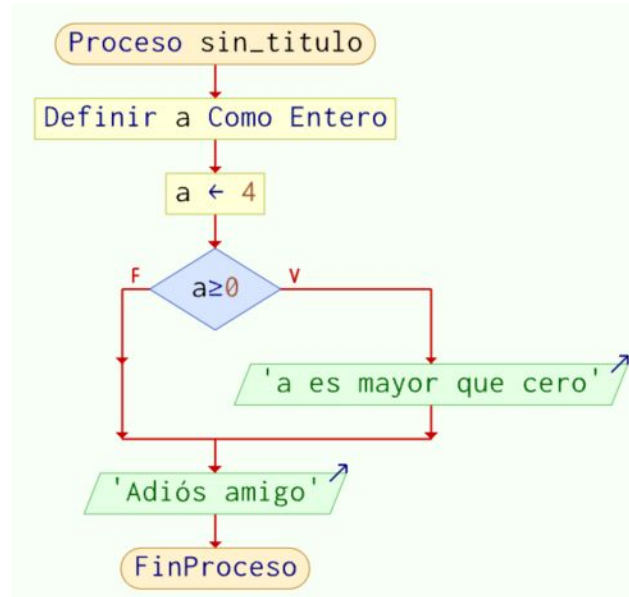
Condicional simple

Ejemplo: Declara una variable de tipo de entero. Inicializala, puedes asignarle el valor tú mismo o solicitarlo al usuario. Si el número introducido es mayor que cero tienes que informar al usuario con una frase. Al terminar el programa despídete del usuario.

Condicional simple

Ejemplo: Declara una variable de tipo de entero. Inicializala, puedes asignarle el valor tú mismo o solicitarlo al usuario. Si el número introducido es mayor que cero tienes que informar al usuario con una frase. Al terminar el programa despídete del usuario.

```
Proceso sin_titulo
    Definir a Como Entero;
    a ← 4;
    Si a >= 0 Entonces
        Escribir "a es mayor que cero";
    FinSi
    Escribir "Adiós amigo";
FinProceso
```



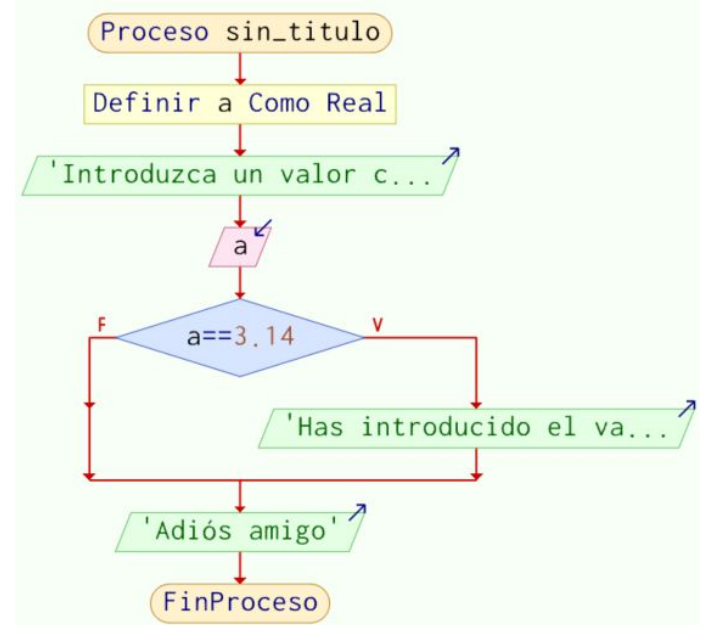
Condicional simple

Ejercicio: Solicita al usuario un número con decimales. Comprueba si se trata del número PI. En caso de que sea igual dile que ha introducido el número PI. Despídete al salir del programa.

Condicional simple

Ejercicio: Solicita al usuario un número con decimales. Comprueba si se trata del número PI. En caso de que sea igual dile que ha introducido el número PI. Despídete al salir del programa.

```
Proceso sin_titulo
    Definir a Como Real;
    Escribir "Introduzca un valor con decimales por favor";
    Leer a;
    Si a == 3.14 Entonces
        Escribir "Has introducido el valor de PI";
    FinSi
    Escribir "Adiós amigo";
FinProceso
```



Condicional doble

Se diferencia de la anterior porque permite habilitar otro bloque de sentencias que se ejecutará en el caso (y sólo en el caso) de que la expresión lógica sea evaluada como “falsa”.

```
...  
SENTENCIA1  
SENTENCIA2  
SI (expresión lógica) ENTONCES  
    SENTENCIA3  
    SENTENCIA4  
SINO  
    SENTENCIA5  
    SENTENCIA6  
FINSI  
SENTENCIA7  
SENTENCIA8  
...
```

Si “expresión_lógica” es verdadera → sentencia3,
sentencia4

Si “expresión_lógica” es falsa → sentencia5,
sentencia6

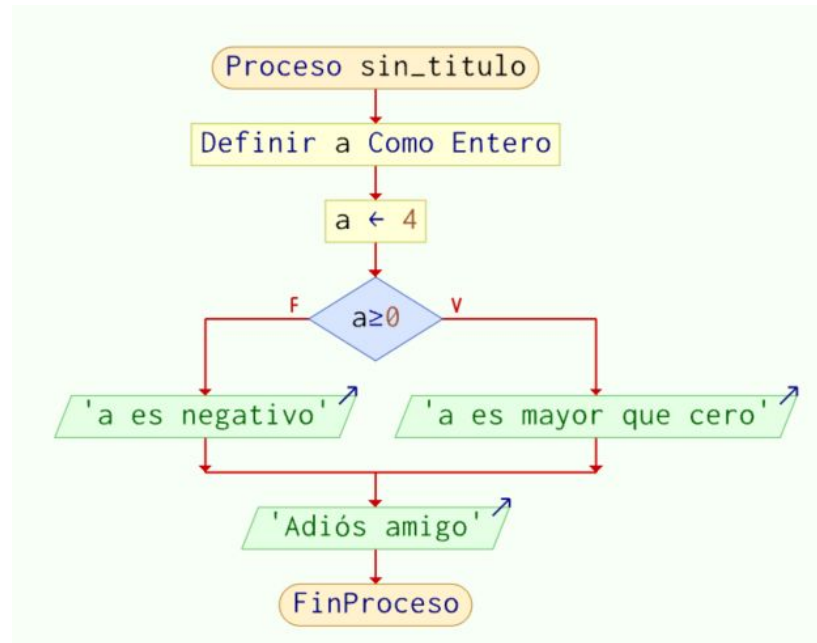
Condicional doble

Ejemplo: Modifica el ejercicio en el que comprobabas si el número es mayor o igual que cero. Si no lo es informa de que es negativo.

Condicional doble

Ejemplo: Modifica el ejercicio en el que comprobabas si el número es mayor o igual que cero. Si no lo es informa de que es negativo.

```
Proceso sin_titulo
  Definir a Como Entero;
  a ← 4;
  Si a ≥ 0 Entonces
    Escribir "a es mayor que cero";
  sino
    Escribir "a es negativo";
  FinSi
  Escribir "Adiós amigo";
FinProceso
```



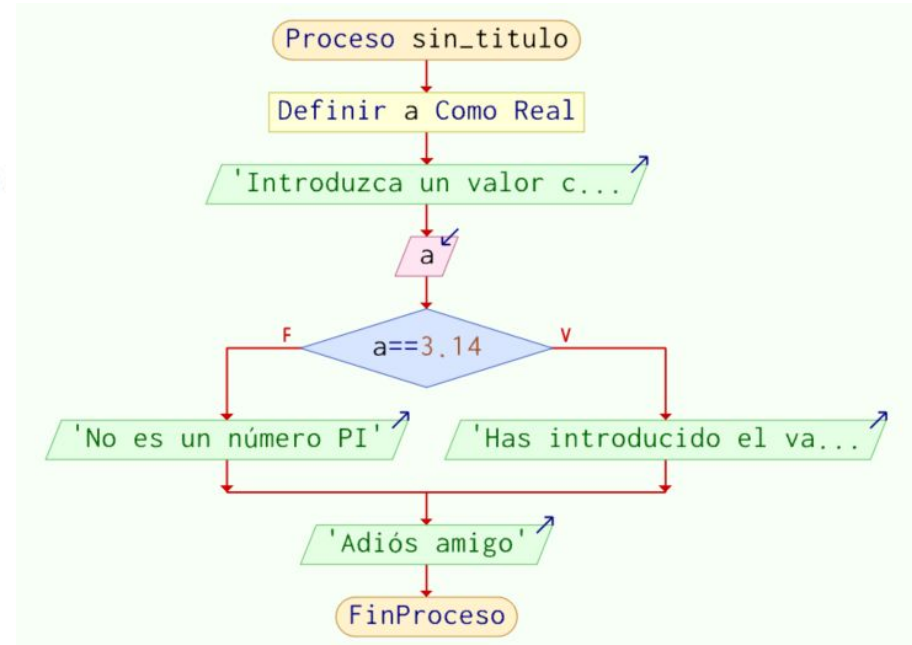
Condicional doble

Ejercicio: Modifica el ejercicio en el que comprobabas si el número introducido es PI. Ahora sí no es PI informa al usuario también de ello.

Condicional doble

Ejercicio: Modifica el ejercicio en el que comprobabas si el número introducido es PI. Ahora sí no es PI informa al usuario también de ello.

```
Proceso sin_titulo
    Definir a Como Real;
    Escribir "Introduzca un valor con decimales por favor";
    Leer a;
    Si a == 3.14 Entonces
        Escribir "Has introducido el valor de PI";
    SiNo
        Escribir "No es un número PI";
    FinSi
    Escribir "Adiós amigo";
FinProceso
```



Condicional múltiple

Complementa al anterior a base de evaluar otras condiciones si no se cumplen las anteriores.

El número de expresiones que se pueden “enlazar” es, a priori, ilimitado.

```
...
SENTENCIA1
SENTENCIA2
SI (expresión_lógica_1) ENTONCES
    SENTENCIA3
    SENTENCIA4
SINOSI (expresión_lógica_2) ENTONCES
    SENTENCIA5
    SENTENCIA6
SINOSI (expresión_lógica_3) ENTONCES
    SENTENCIA7
    SENTENCIA8
SINO
    SENTENCIA9
    SENTENCIA10
FINSI
SENTENCIA11
SENTENCIA12
...
```


PSeInt

No incluye la condición múltiple. Deberías dividir en dobles y simples.

```
Si numero>10 Entonces
    Escribir "Mayor que 10";
SiNo
    Escribir "Menor o igual que 10";
FinSi
```

¿Cómo tengo en cuenta que número puede ser 10?

Condicional múltiple

Proceso sin_titulo

Definir numero Como Entero;

a ← -1;

Si numero > 10 Entonces

 Escribir "Mayor que 10";

SiNo

 si numero < 10 Entonces

 Escribir "Menor que 10";

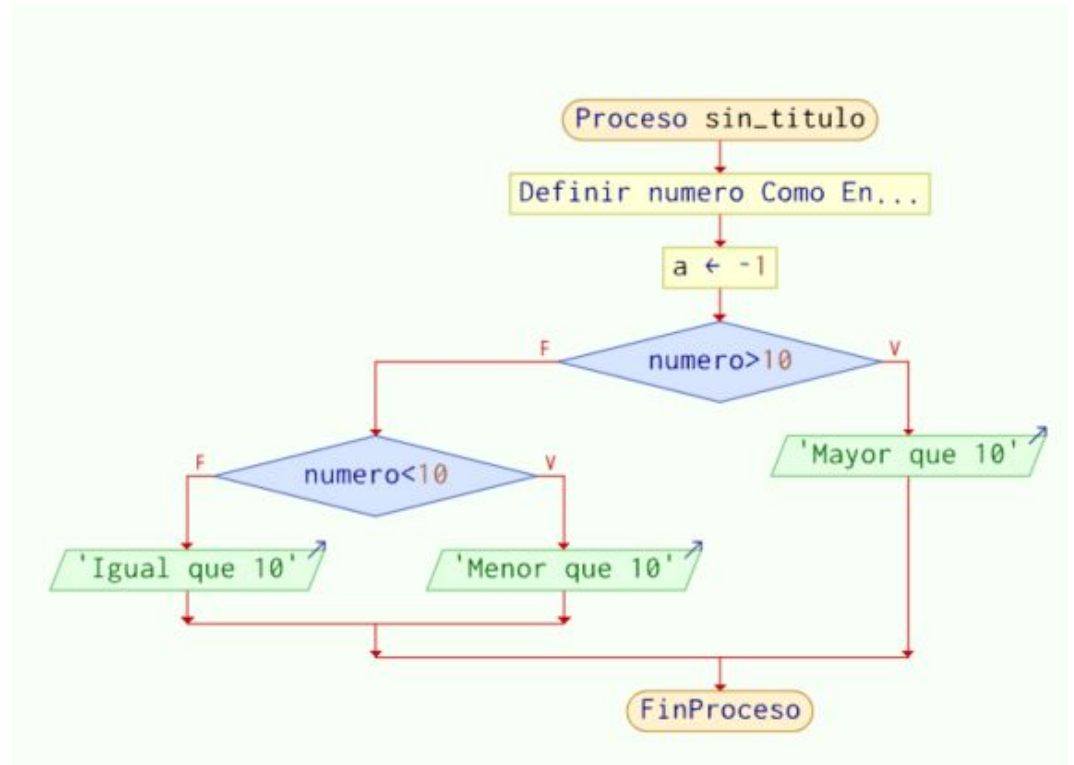
 SiNo

 Escribir "Igual que 10";

 FinSi

FinSi

FinProceso



Consejo para las condicionales múltiples

Hacer una reflexión previa de cuáles son los rangos de valores más probables a cumplir la condición.

Imagina que solicitas la edad al usuario y queremos comprobar si la edad introducida es válida. Podemos pensar que una edad válida va de los 0 años a los 110 años.

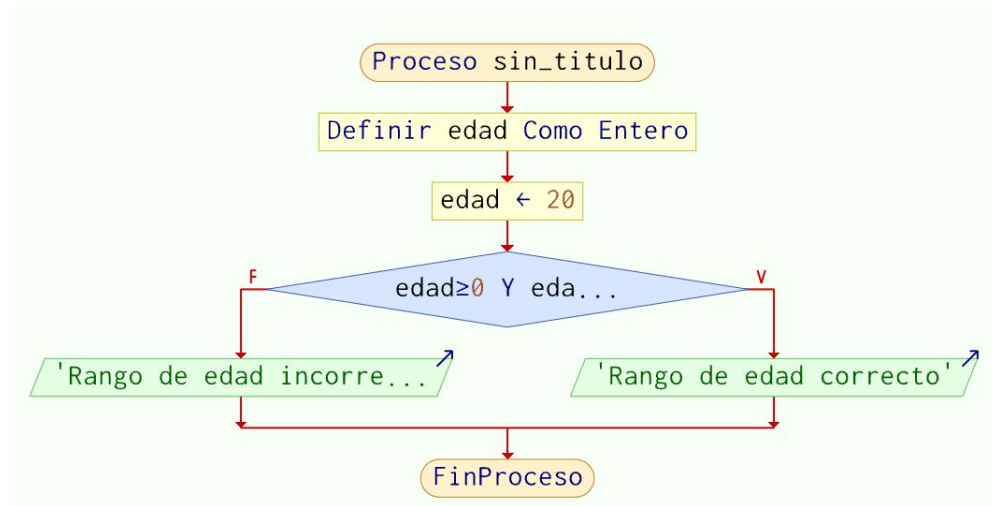
Existen más probabilidades de que el usuario no se equivoque, por lo que primero evaluaríamos la condición que es más probable que devuelva true.

¿Cómo lo escribirías?

Los operadores lógicos en PSeInt son: **y** (and), **o** (or).

Consejo para las condicionales múltiples

```
Algoritmo sin_titulo  
  Definir edad Como Entero;  
  edad ← 20;  
  Si edad ≥ 0 y edad ≤ 110 Entonces  
    Escribir "Rango de edad correcto";  
  SiNo  
    Escribir "Rango de edad incorrecto";  
  FinSi  
FinAlgoritmo
```



Tarea - 2

Condicional múltiple - SEGUN

Cuando las condiciones son igualdades con valores literales, se puede utilizar un tipo especial de condicional múltiple (SEGUN).

Esto es tenemos una variable que dependiendo del valor que tome realizaremos unas acciones u otras. Podemos escribir un SI detrás de otro pero queda más claro el código con un SEGUN.

No todos los lenguajes de programación incorporan esta estructura (switch).

```
según expresión hacer
inicio
  valor1: acciones-1
  valor2: acciones-2
  valor3: acciones-3
  ...
  valor4: acciones-N
  si_no: acciones-si_no
fin
```

Condicional múltiple - SEGUN

Ejemplo: Imagina que dado un número entero debes informar al usuario si el número introducido es 0, 1, 2 y en cualquier otro caso informarle de que el número no es válido. Podríamos escribir algo como esto utilizando la condicional SI.

Proceso sin_titulo

Definir numero Como Entero;

numero ← 2;

Si numero ==0 Entonces

 Escribir "Es un 0";

SiNo

 Si numero ==1 Entonces

 Escribir "Es un 1";

 SiNo

 Si numero ==2 Entonces

 Escribir "Es un 2";

 SiNo

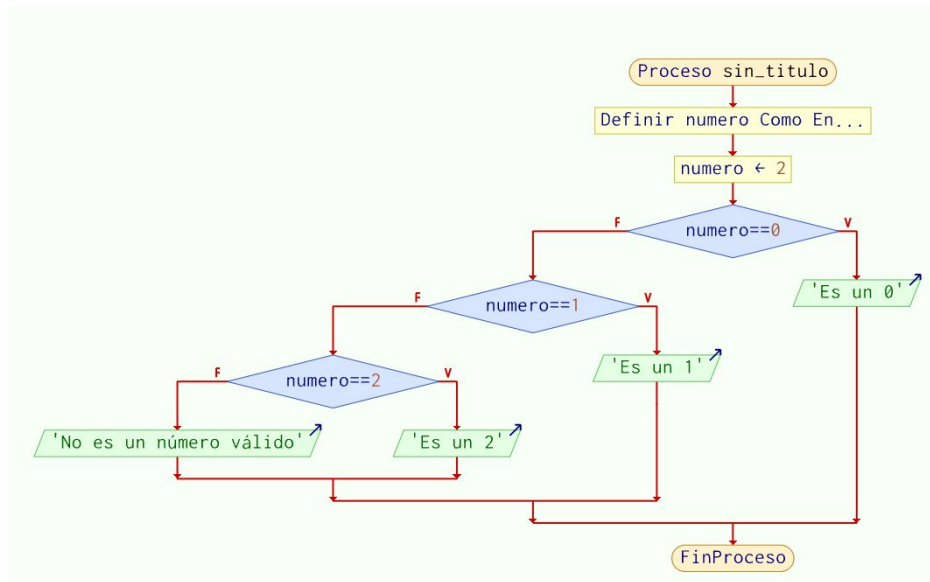
 Escribir "No es un número válido";

 FinSi

 FinSi

FinSi

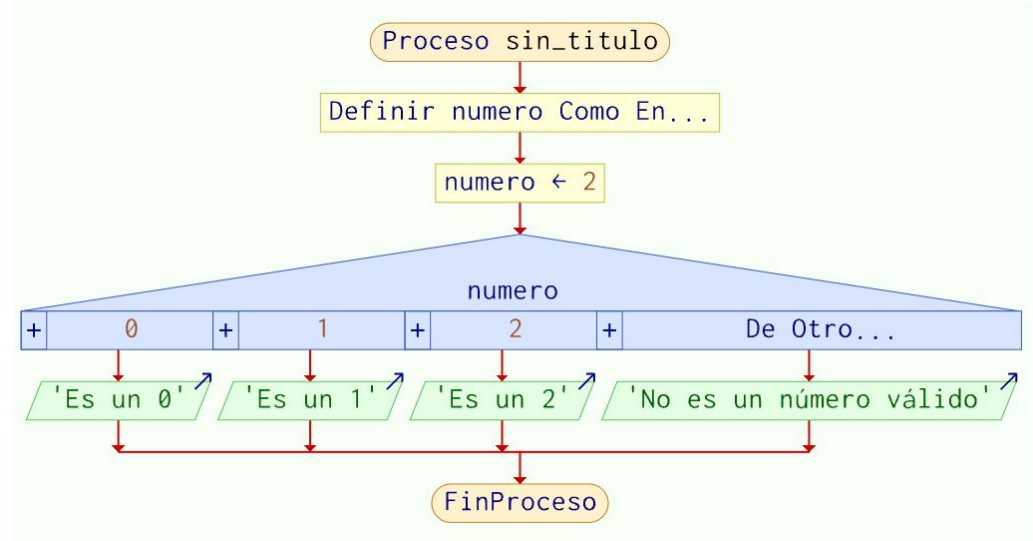
FinProceso



Condicional múltiple - SEGUN

Ejemplo: Reescribimos el código con la condicional SEGUN.

```
Proceso sin_titulo
  Definir numero Como Entero;
  numero ← 2;
  Segun numero Hacer
    0:
      Escribir "Es un 0";
    1:
      Escribir "Es un 1";
    2:
      Escribir "Es un 2";
  De Otro Modo:
      Escribir "No es un número válido";
FinSegun
FinProceso
```



Condicional múltiple - SEGUN

Ejercicio: Solicita un número entero al usuario. Si es un 1 escribe 1, si es 2, multiplica el valor introducido por 2 y lo imprimes por pantalla, si es un 3 dile que te escriba otro número y lo multiplicas por 3 y lo imprimes por pantalla. Si introduce cualquier otro número díselo. Despídete al finalizar el programa.

Condicional múltiple - SEGUN

Ejercicio: Solicita un número entero al usuario. Si es un 1 escribe 1, si es 2, multiplica el valor introducido por 2 y lo imprimes por pantalla, si es un 3 dile que te escriba otro número y lo multiplicas por 3 y lo imprimes por pantalla. Si introduce cualquier otro número díselo. Despídete al finalizar el programa.

Algoritmo sin_titulo

Definir numero1 Como Entero;

Definir numero2 Como Entero;

Leer numero1;

Segun numero Hacer

1:

Escribir "1";

2:

numero2←numero1*2;

Escribir numero2;

3:

Escribir "Otro número";

Leer numero2;

numero2←numero2*3;

Escribir numero2;

De Otro Modo:

Escribir "El número no es válido";

FinSegun

FinAlgoritmo

Tarea - 3