

## TEMA II

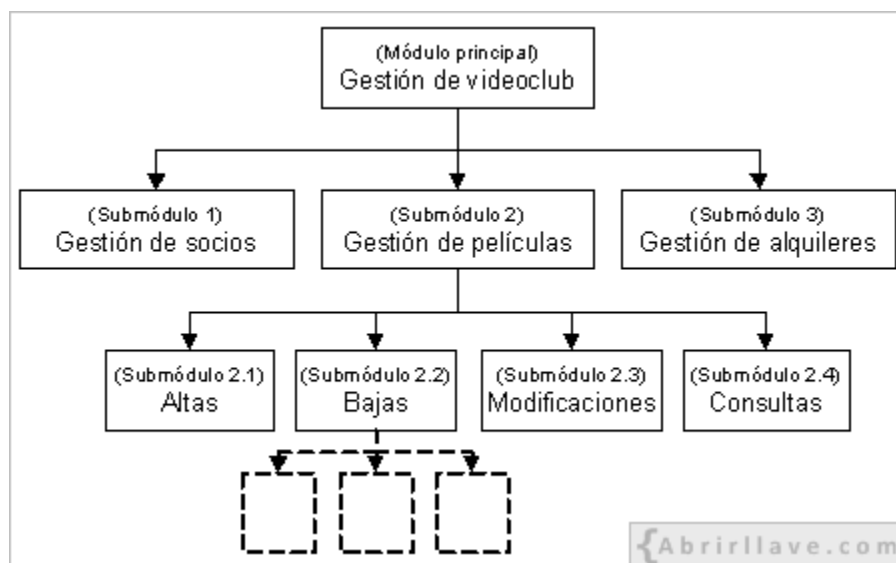
### DISEÑO DE UN PROGRAMA

Una vez que establecemos los requisitos de un programa, pasar a la fase de diseño. En esta etapa se tiene que encontrar una solución informática al problema planteado. Dicha solución determinará cómo se va a resolver el problema.

Por norma general, no suele ser fácil encontrar una solución óptima y, menos aún, cuando el problema tiene cierta envergadura. Para encontrar dicha solución, el desarrollador puede hacer uso del diseño modular.

#### Diseño modular

Dado un problema a resolver, en primer lugar, hay que estudiar si existe la posibilidad de dividirlo en otros más pequeños, llamados subproblemas (este método es conocido como "divide y vencerás"). Cada uno de ellos puede tratarse de manera aislada; por tanto, la complejidad global del problema disminuirá. Del mismo modo, si los subproblemas obtenidos siguen siendo demasiado complicados pueden dividirse de nuevo. Y así sucesivamente, hasta llegar a subproblemas sencillos.



En resumen, la solución a un problema suele venir dada por un programa representado por un módulo principal, el cual se descompone en subprogramas (submódulos), los cuales, a su vez, también se pueden fraccionar, y así sucesivamente, es decir, el problema se resuelve de arriba hacia abajo. A este método se le denomina diseño modular o descendente (top-down).

## Qué es un algoritmo

En la fase de codificación, todos los módulos definidos mediante el diseño modular se convertirán en un programa, es decir, la aplicación final estará compuesta por todos los programas que se diseñen. Pero antes, hay que determinar cuáles son las instrucciones o acciones de cada uno de dichos programas. Para ello, se debe hacer uso de algoritmos.

Un algoritmo es la secuencia de pasos o acciones que resuelve un determinado problema. Los algoritmos constituyen la documentación principal que se necesita para poder iniciar la fase de codificación y, para representarlos, se utiliza, fundamentalmente, dos tipos de notación: **pseudocódigo y diagramas de flujo**. El diseño de un algoritmo es independiente del lenguaje que después se vaya a utilizar para codificarlo.

Además del pseudocódigo y los diagramas, tenemos otras herramientas que nos puede ayudar a optimizar nuestro código: tablas y árboles de decisión.

### TABLAS DE DECISIÓN

Son una técnica utilizada para expresar el comportamiento de procesos elementales, cuando presentan un comportamiento complejo con muchas alternativas y condiciones difíciles de expresar en lenguaje natural.

Recogen las condiciones en que se generan unos y otros resultados de salida en función de los valores que toman los datos de entrada.

Por un lado, se ponen en columnas las condiciones que afectan a los valores de los datos de entrada y en las columnas finales las acciones que constituyen los datos de salida. De esta forma, cada fila de la tabla codifica una combinación concreta de los valores de entrada y la

acción de salida correspondiente. Tienen que reflejarse todos los casos posibles. Los casos posibles son siempre 2 elevado al número de factores.

**Ejemplo1.** Supongamos que existe un ordenador que controla un motor. Se quiere mostrar un mensaje si se calienta el motor o si se dispara el consumo, y además si se calienta el motor se quiere hacer sonar una alarma. Hay dos factores:

- que se caliente el motor [0,1]
- que se dispare el consumo [0,1]

Tabla de decisión.

Hay  $2^2$  líneas

Calentar motor	Disparar consumo	Mensaje motor caliente	Mensaje consumo excesivo	Alarma
0	0	0	0	0
0	1	0	1	0
1	0	1	0	1
1	1	1	1	1

**Ejemplo2.** Renovación de un seguro de automóvil.

Si el usuario no ha dado ningún parte el último año se rebaja el seguro, si ha dado algún parte el último año y es menor de 25 años se sube a no ser que tenga más de 5 años de antigüedad que se deja igual. En el resto de los casos se queda igual.

Se tienen en cuenta tres factores:

- más de 5 años de antigüedad [0,1]
- si se ha dado algún parte el último año [0,1]
- usuario tiene menos de 25 años [0,1]

Tabla de decisión.

Hay  $2^3$  líneas.

antigüedad	<25	parte	Subir	Bajar	igual
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	0	0	1

### PROCESO DE TRANSFORMACIÓN DE UNA TABLA DE DECISIÓN

1. Comprobar que la tabla está completa y no es contradictoria.
2. Simplificar la tabla. Dos reglas de decisión son simplificables si son iguales en todos sus valores (incluida la acción que se especifica), salvo en el de una de sus condiciones. En este caso, ambas reglas de decisión pueden sustituirse por una sólo con todos los valores iguales de ambos y una invariante -, en el de la condición discrepante. Deben simplificarse sucesivamente todas las parejas de reglas que se pueda. Repito este proceso todas las veces que sea posible
3. Ordenar la tabla. Se trata de dejar arriba y a la izquierda las más genéricas, es decir las que tengan mayor número de invariantes.

Simplificamos la tabla estableciendo las parejas de reglas

antigüedad	<25	parte	Subir	Bajar	igual
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	0	1	0
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	0	0	1

antigüedad	<25	parte	Subir	Bajar	igual
0	-	0	0	1	0
-	0	1	0	0	1
1	-	0	0	1	0
0	1	1	1	0	0
1	1	1	0	0	1

Ordenamos la tabla

antigüedad	<25	parte	Subir	Bajar	igual
-	-	0	0	1	0
-	0	1	0	0	1
0	1	1	1	0	0
1	1	1	0	0	1

## ÁRBOL DE DECISIÓN

Los árboles de decisión son un algoritmo de aprendizaje automático que se utiliza en la ciencia de datos para procesar grandes volúmenes de datos y solventar problemas. Son algoritmos y técnicas de machine learning que nos permiten la construcción de modelos predictivos basados en su clasificación según ciertas características o propiedades.

En los modelos de clasificación queremos predecir el valor de una variable mediante la clasificación de la información en función de otras variables (tipo, pertenencia a un grupo...).

Por ejemplo, queremos pronosticar qué personas comprarán un determinado producto, clasificando entre clientes y no clientes, o qué marcas de portátiles comprará cada persona mediante la clasificación entre las distintas marcas. Los valores para predecir son predefinidos, es decir, los resultados están definidos en un conjunto de posibles valores.

Se compone de los siguientes elementos:

1. Ramas o estados intermedios: se representan por medio de rectángulos y en su interior se representan las condiciones.
2. Hojas o estados finales: Se representan por medio de círculos y en su interior se sitúan las acciones.
3. Conexiones: líneas que enlazan las ramas y hojas.

Hay que tener en cuenta al construir un árbol:

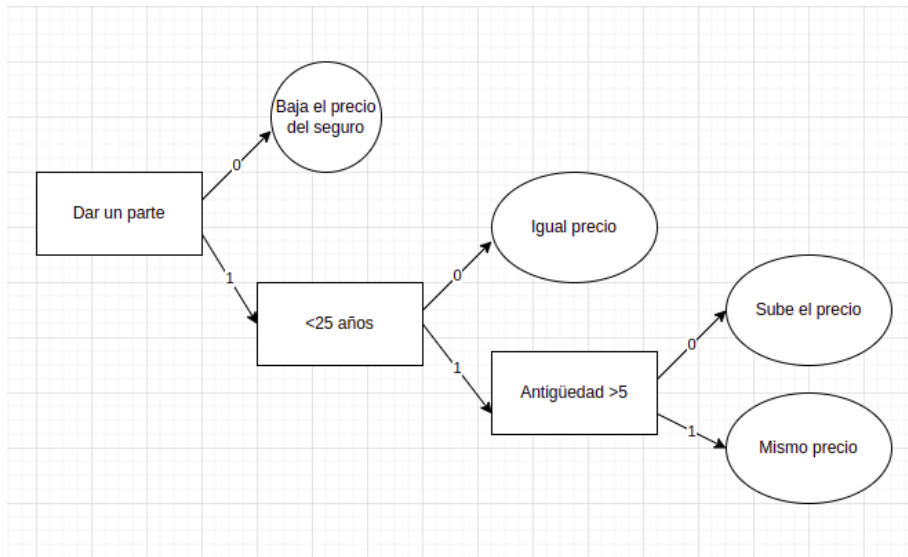
- Se dibujan de izda a derecha.
- En cada nivel hay una rama de la que salen tantas conexiones como posibles valores pueda tomar la condición.
- En el primer nivel se sitúa la condición más genérica.
- En el último nivel se sitúan las hojas.

**Ejemplo.** Renovación de un seguro de automóvil.

Si el usuario no ha dado ningún parte el último año se rebaja el seguro, si ha dado algún parte el último año y es menor de 25 años se sube a no ser que tenga más de 5 años de antigüedad que se deja igual. En el resto de los casos se queda igual.

Se tienen en cuenta tres factores:

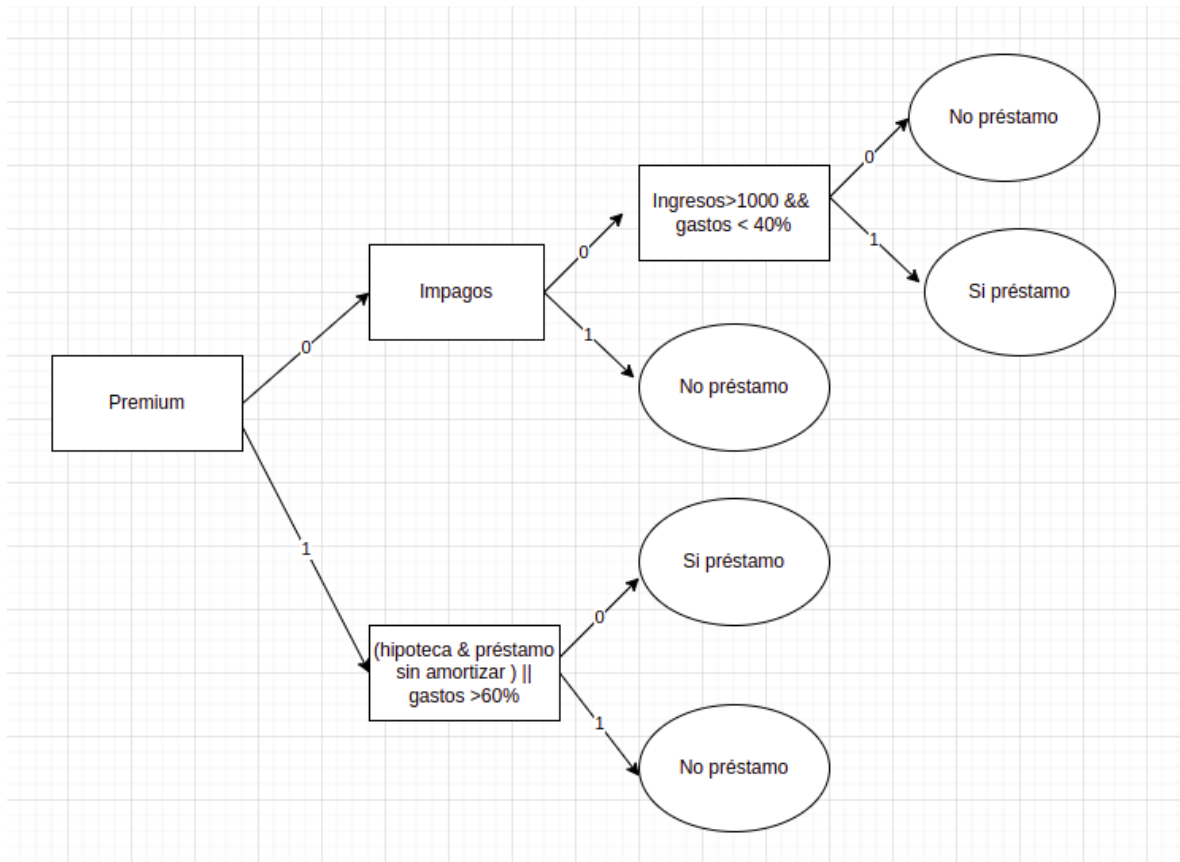
- más de 5 años de antigüedad [0,1]
- si se ha dado algún parte el último año [0,1]
- usuario tiene menos de 25 años [0,1]



**Ejemplo.** Para realizar la predicción de si un banco debe ofrecer préstamos de hasta 10.000 euros a sus clientes en una campaña de marketing. Tiene clientes premium y normales.

Los clientes premium son aquellos que tienen depósitos superiores a los 100.000 euros y/o ingresos superiores a los 2.000 euros al mes en el banco y los clientes normales el resto.

1. En el caso de los clientes premium si tienen hipotecas o préstamos sin amortizar superiores al 50% de sus depósitos y/o gastos mensuales superiores al 60% de sus ingresos no se le ofrecerá el préstamo, en caso inverso sí.
2. Si el cliente es del tipo normal, consideraremos dos factores distintos:
  - Si el cliente tiene algún impago no se le ofrecerá el préstamo.
  - Si el cliente está al día en sus pagos, sus ingresos son superiores a 1.000 euros al mes y sus gastos inferiores al 40% se le ofrecerá el préstamo, en caso contrario no.






## 2. ORDINOGRAMAS

Un ordinograma es una representación gráfica de todas las instrucciones de un programa reflejando la secuencia lógica de las operaciones necesarios para la resolución del problema. Muestra gráficamente el algoritmo del programa.

Características:

- Debe tener un inicio y un fin.
- Se diseña de arriba abajo y de izquierda a derecha.
- Se debe evitar el cruce de líneas ya que crean confusión.
- Utiliza expresiones independientes del lenguaje de programación.
- Se usan conectores  para unir líneas de flujo.

### Instrucciones primitivas

- Asignación de un valor a una variable.

numero = 5

numero = numero +1

numero = resultado

- Entrada de datos por teclado.

LEER numero

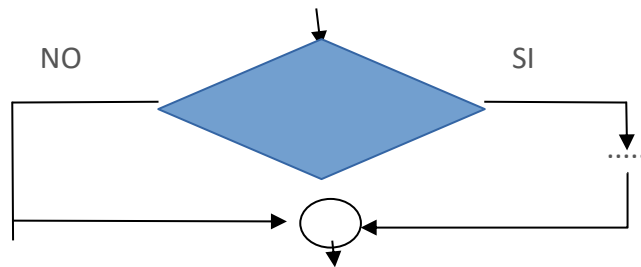
- Salida de datos por pantalla.

MOSTRAR numero

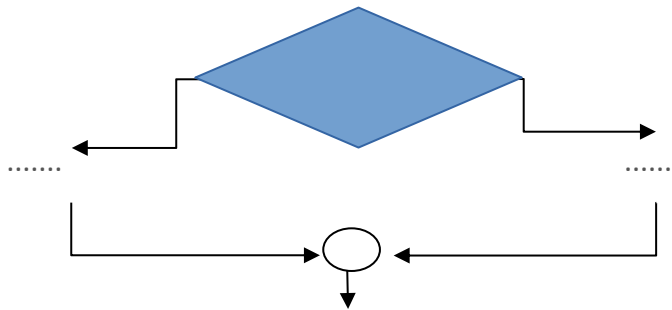
MOSTRAR "Introduzca nombre"

Instrucciones de control no repetitivas

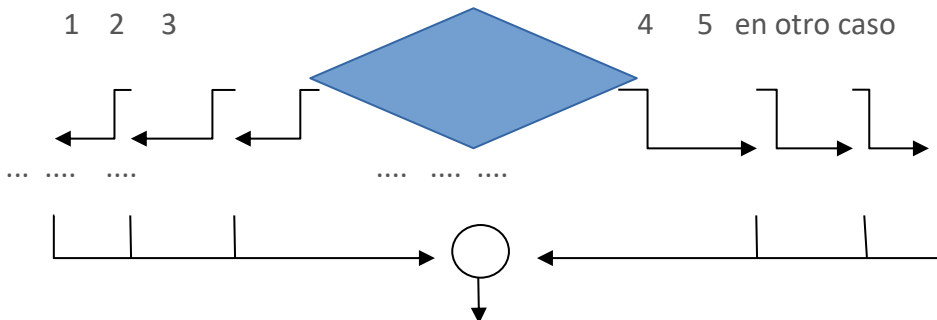
- Simple (if): Se evalúa la condición, y dependiendo de si es verdadera o no se continúa por una de las dos ramas. En la rama del NO no hay instrucciones.



- Compuesta (if else): Se evalúa la condición, y dependiendo de si es verdadera o no se continúa por una de las dos ramas. Hay instrucciones en ambas ramas.

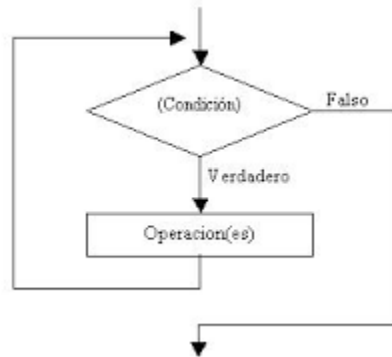


- Switch: Dependiendo del valor del dato encerrado en el rombo se sigue por un camino u otro.



### Instrucciones de control repetitivas

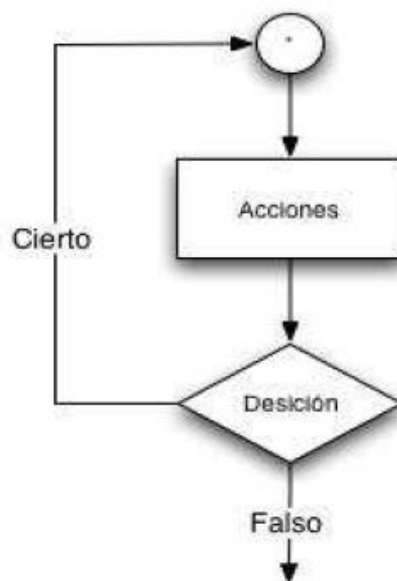
- Mientras (while)



Se evalúa la condición, si es falsa no ejecutamos nada, si es verdadera ejecutamos las instrucciones del SI, una vez ejecutadas volvemos a evaluar la condición.

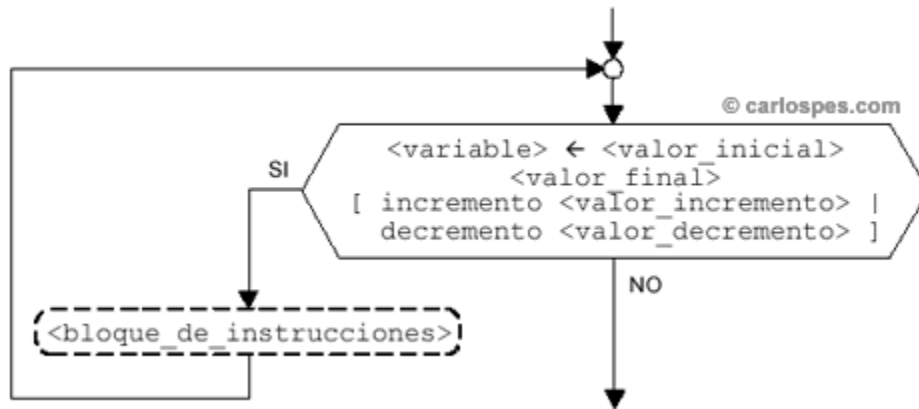
- Mientras con al menos una ejecución (do while)

Primero se ejecuta el bloque de instrucciones, después se evalúa la condición, si es verdadera se vuelve a ejecutar el bloque de instrucciones, si es falsa se continúa con el resto del programa.



- Para (for)

Se realizan las inicializaciones y se evalúa la condición, si es verdadera se ejecutan las instrucciones del for y después se realiza el incremento del rombo y se vuelve a evaluar la condición. Si la condición no se cumple no se ejecutan las instrucciones.



## Ejemplos

1. Programa que lea un número de teclado y me diga si es positivo o negativo
2. Realizar programa que inicialice un saldo, lea un depósito, agregue el depósito al saldo, lea un retiro, verifique que hay suficiente saldo. Si hay suficiente saldo, realiza la operación.
3. Hacer un programa que realice descuentos en base a lo siguiente:
  - Si la compra es menor a 500 no hay descuento
  - Si es entre 500 y 1000, 10% de descuento
  - Si es entre 1000 y 2000, 20% de descuento
  - Si es mayor a 2000, 30% de descuento
4. Programa que lee una nota y valida que sea correcta
5. Programa que muestra por pantalla los 10 primeros números naturales (con while y for)

### 3. Pseudocódigo

El pseudocódigo es un lenguaje de programación algorítmico, intermedio entre el lenguaje natural y cualquier lenguaje de programación específico,

No existe una notación formal o estándar de pseudocódigo, sino que, cada programador puede utilizar la suya propia. Se pretende facilitar la codificación posterior de los algoritmos.

Ejemplos: Pseudocódigo de algún ejemplo anterior