TEMA 1

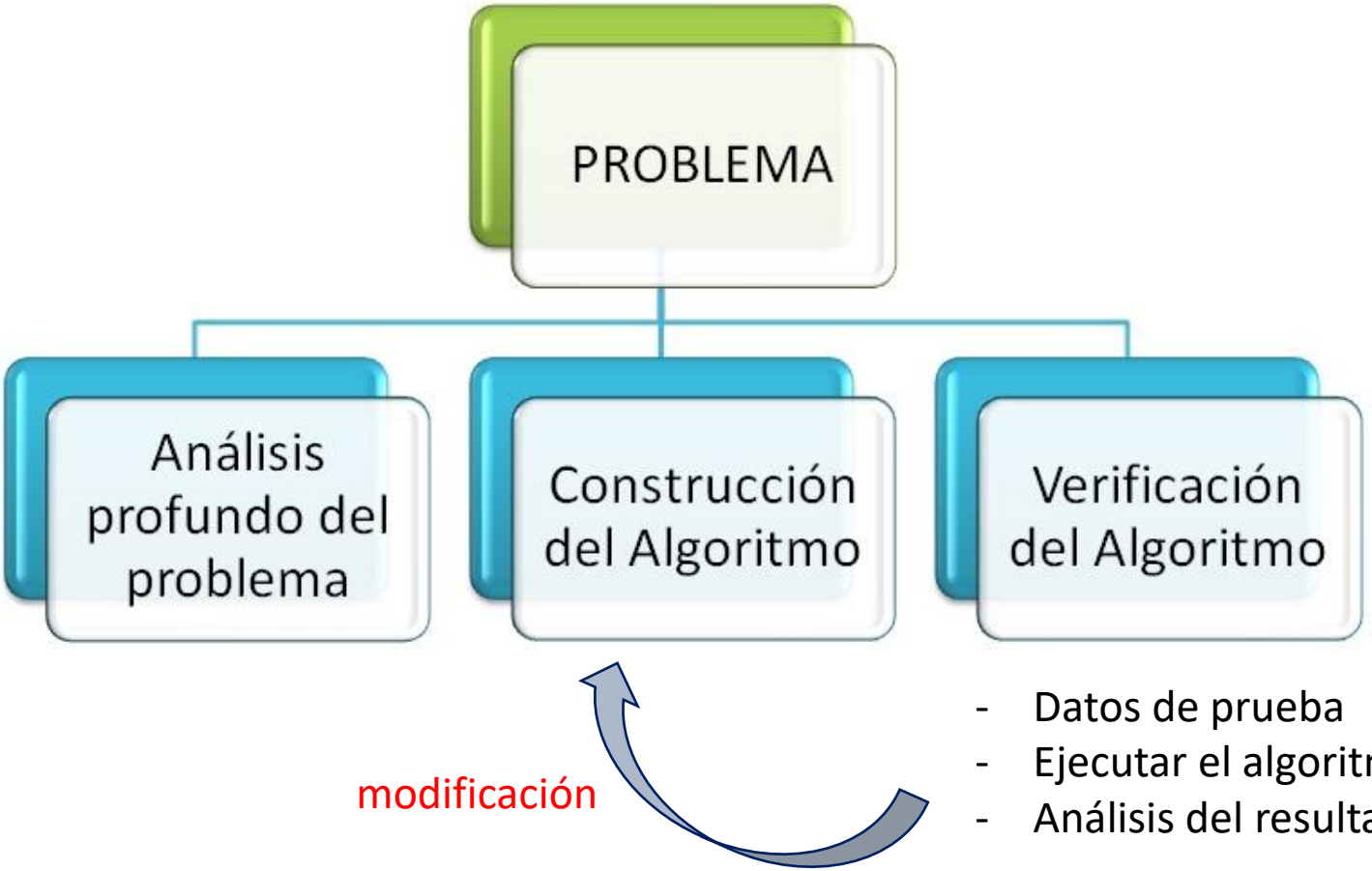
INTRODUCCIÓN Y REVISIÓN CONCEPTOS PREVIOS

INTRODUCCIÓN Y DESCRIPCIÓN GENERAL

- ÍNDICE
  - INTRODUCCIÓN
  - EMPLEO DEL SOFTWARE PSEINT
  - ASIGNACIÓN Y ENTRADA/SALIDA
  - ELEMENTOS BÁSICOS
  - ESTRUCTURAS CONDICIONALES
  - ESTRUCTURAS REPETITIVAS

Según Niklaus Wirth un **programa** está formado por **algoritmos** y **estructura de datos.**

PASOS PARA RESOLVER UN PROBLEMA



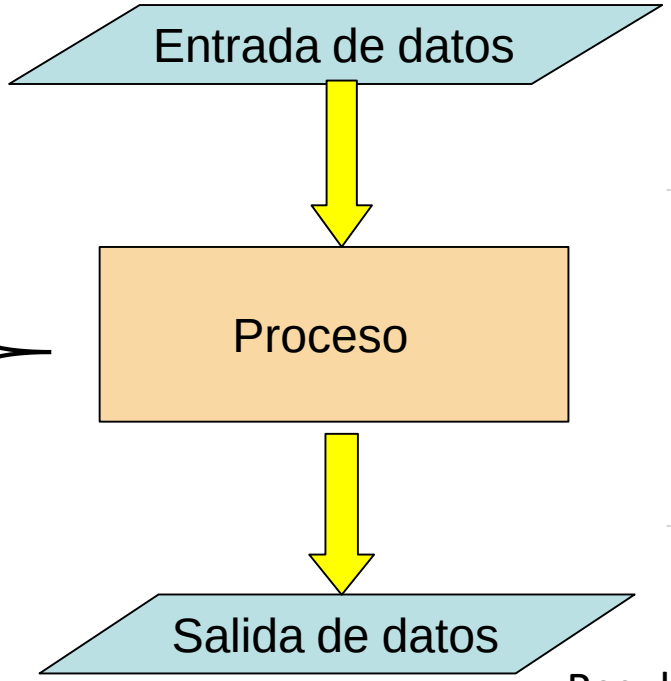
La solución de un problema se puede expresar mediante un algoritmo. No tiene por qué ser único.

PARTES DE UN PROGRAMA

La definición de un algoritmo debe describir tres partes: ENTRADA, PROCESO y SALIDAS.

ALGORITMO (PSEUDOCÓDIGO)

CÓDIGO (LENGUAJE)



- Tipos de datos
- Cantidad

- Transformación de datos
- Pasos a seguir
- Tener en cuenta restricciones (EDAD >0 Y EDAD<100).

Resultados que se quieren obtener (tipo de datos y cantidad)

EJEMPLO SENCILLO DE ALGORITMO

Preparar una taza de té

```
Entrada: tetera, taza, bolsa de té
Salida: taza de té

Inicio
   Tomar la tetera
   Llenarla de agua
   Encender el fuego
   Poner la tetera en el fuego
   Esperar a que hierva el agua
   Tomar la bolsa de té
   Introducirla en la tetera
   Esperar 1 minuto
   Echar el té en la taza
Fin
```

DIFERENCIA ALGORITMO – CÓDIGO FUENTE

ALGORITMO

```
Proceso Adivina_Numero

    intentos<-10
    num_secreto <- azar(100)+1

    Escribir "Adivine el numero (de 1 a 100):"
    Leer num_ingresado
    Mientras num_secreto<>num_ingresado Y intentos>1 Hacer
        Si num_secreto>num_ingresado Entonces
            Escribir "Muy bajo"
        Sino
            Escribir "Muy alto"
        FinSi
        intentos <- intentos-1
        Escribir "Le quedan ",intentos," intentos:"
        Leer num_ingresado
    FinMientras
```

- Pasos: **acciones**
- Lenguaje: **Pseudocódigo**
- Se realiza en la fase de **DISEÑO**

CÓDIGO FUENTE (PROGRAMA)

```
25
26 import java.util.*;
27 import java.lang.*;
28 import java.util.Random;
29
30 class Rextester
31 {
32   public static void main(String args[]) {
33     // l: lanzamientos; m: fr. muestreo
34     int l = 10000000, m = 5000;
35     // f: favorables; b=0: C; b=1; X
36     int f = 0, b = 0, r, n = 0;
37     double fr; // fr: frec. rel
38
39     Random g = new Random();
40     for(int i=0;i<(l/m);i++) {
41       for(int j=0;j<m;j++) {
42         r = g.nextInt(2);
43         if(r == b) f++;
```

- Pasos: **sentencias o instrucción**
- Lenguaje: **Lenguaje de programación**
- Se realiza en la fase de **CONSTRUCCIÓN**

De un mismo algoritmo podemos obtener diversos códigos en diferentes lenguajes de programación (Java, Python, PASCAL, …)

CARACTERÍSTICAS DE LOS ALGORITMOS

- **PRECISO:** Indica el orden de realización de cada paso. Pasos no ambiguos.

- **DEFINIDO:** Si se sigue dos veces o más, se debe obtener el mismo resultado.

- **FINITO:** Número finito de pasos. Se debe terminar en algún momento.

Recordar ejemplo de algoritmo para preparar una taza de té.

VENTAJAS ALGORITMOS

- **VENTAJAS:**

- Se puede centrar uno en aspectos lógicos (cómo resolver el problema), sin conocer lenguajes de programación (aspecto más técnico).

- Ayuda a resolver más fácil y rápido los problemas.

- Permiten redactar en orden, paso a paso, lo que hay que hacer.

- Disminuye sensiblemente el riesgo de errores.

- Es sencillo pasar de pseudocódigo a un lenguaje de programación.

- Si hacemos ingeniería inversa partiendo de código fuente, puede ser muy complicado o laborioso traducir a pseudocódigo (conocer bien el lenguaje de programación, analizar y “desmenuzar” las librerías empleadas, etc.)

INCONVENIENTES ALGORITMOS

- **INCONVENIENTES:**

- Se invierte tiempo en diseñar el algoritmo y después traducir a un lenguaje de programación. Es más rápido escribir directamente el código sin hacer algoritmos.

- Para la resolución de la mayoría de los problemas existen varios algoritmos diferentes. Hay que analizar el algoritmo que conduce a la mejor implementación (**recursos** y **tiempo**).

- La elección del mejor algoritmo puede ser muy complicado y conllevar un análisis matemático sofisticado.

- Para pseudocódigo no hay normas definidas, NO SE RIGE POR UN ESTÁNDAR, pudiendo hacer que la lógica resulte complicada de ver por el programador que implemente el algoritmo.

- Puede ser difícil de entender si es muy extenso.

PSEUDOCÓDIGO

- Es la **representación narrativa** de los pasos que debe seguir un algoritmo para dar solución a un problema determinado.

- Es una **técnica NO GRÁFICA**: Utiliza palabras que indican el proceso a realizar.

- Se considera un **“primer borrador”**, dado que el pseudocódigo **tiene que traducirse** posteriormente a un lenguaje de programación.

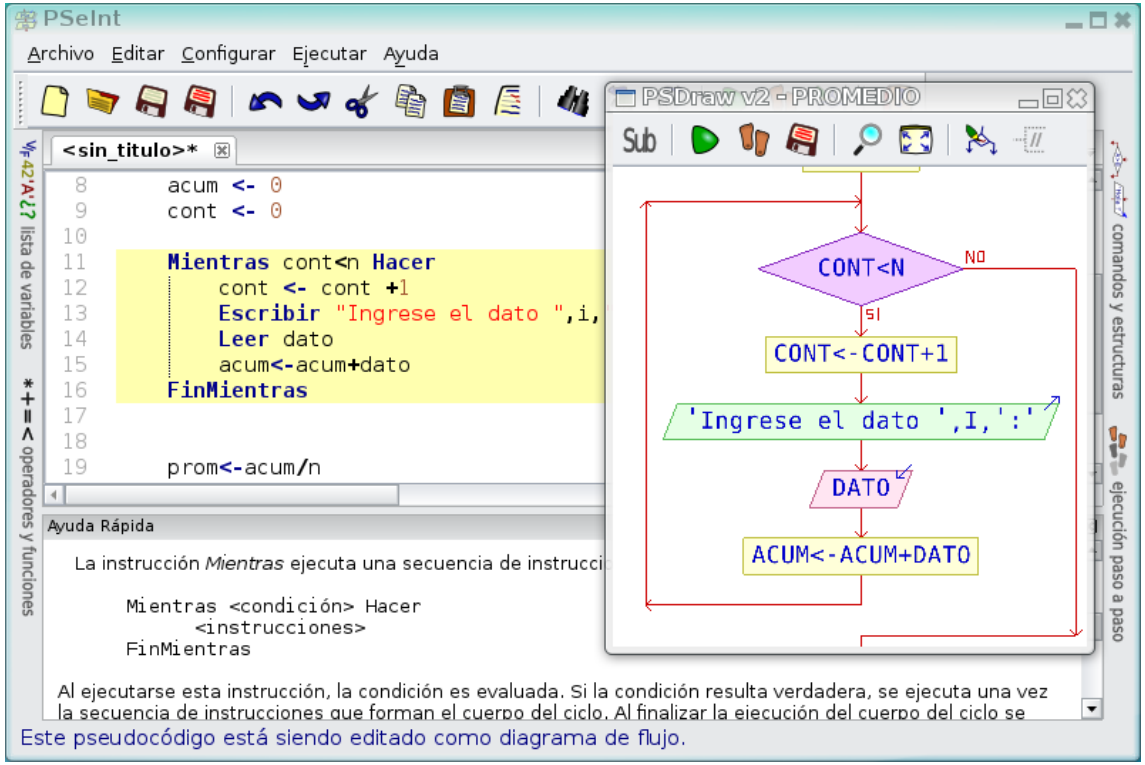
- El pseudocódigo **no puede ser ejecutado** por una computadora. No hay compiladores/intérpretes para ello. Algún software, como PseInt, puede ejecutarlo, pero con fines didácticos.

- Emplea una serie de **palabras clave** o palabras especiales, igual que los lenguajes de programación.

DESCARGA Y UTILIZACIÓN

EMPLEO DE PSEINT

PSeInt (PSeudocode Intérprete): Es una herramienta para asistir a un estudiante en sus primeros pasos en programación.



Permite escribir algoritmos en pseudocódigo en español

http://pseint.sourceforge.net/

EMPLEO DE PSEINT

VENTAJAS:

- Totalmente libre y gratuito (licencia GPLv2)
- Es multiplataforma.
- Posee **previsualización y exportación** a varios lenguajes de programación.
- Permite la exportación a HTML.
- Permite **personalización del pseudocódigo** (punto y coma final, exigencia de declaración de variables, etc.) mediante PERFILES (CONFIGURACIONES). Nosotros emplearemos el **perfil ESTRICTO** (Ver documento personalización).
- Permite la **ejecución paso a paso**, para analizar con detalle el algoritmo, inspeccionando variables y expresiones.
- Permite **probar el algoritmo para evitar un posible error lógico**, para lo cual se hace una **prueba de escritorio**, lo cual significa dar valores simulados a las variables y revisar los resultados.

EMPLEO DE PSEINT

- **VENTAJAS:**

- El pseudocódigo se guarda en un archivo (.psc) para poder emplearlo/modificarlo posteriormente.

- Es como un Entorno de Desarrollo Integrado (IDE) y proporciona **ayudas**: Resalta las palabras clave, nos indica errores, etc.

¿Por qué PSEINT?

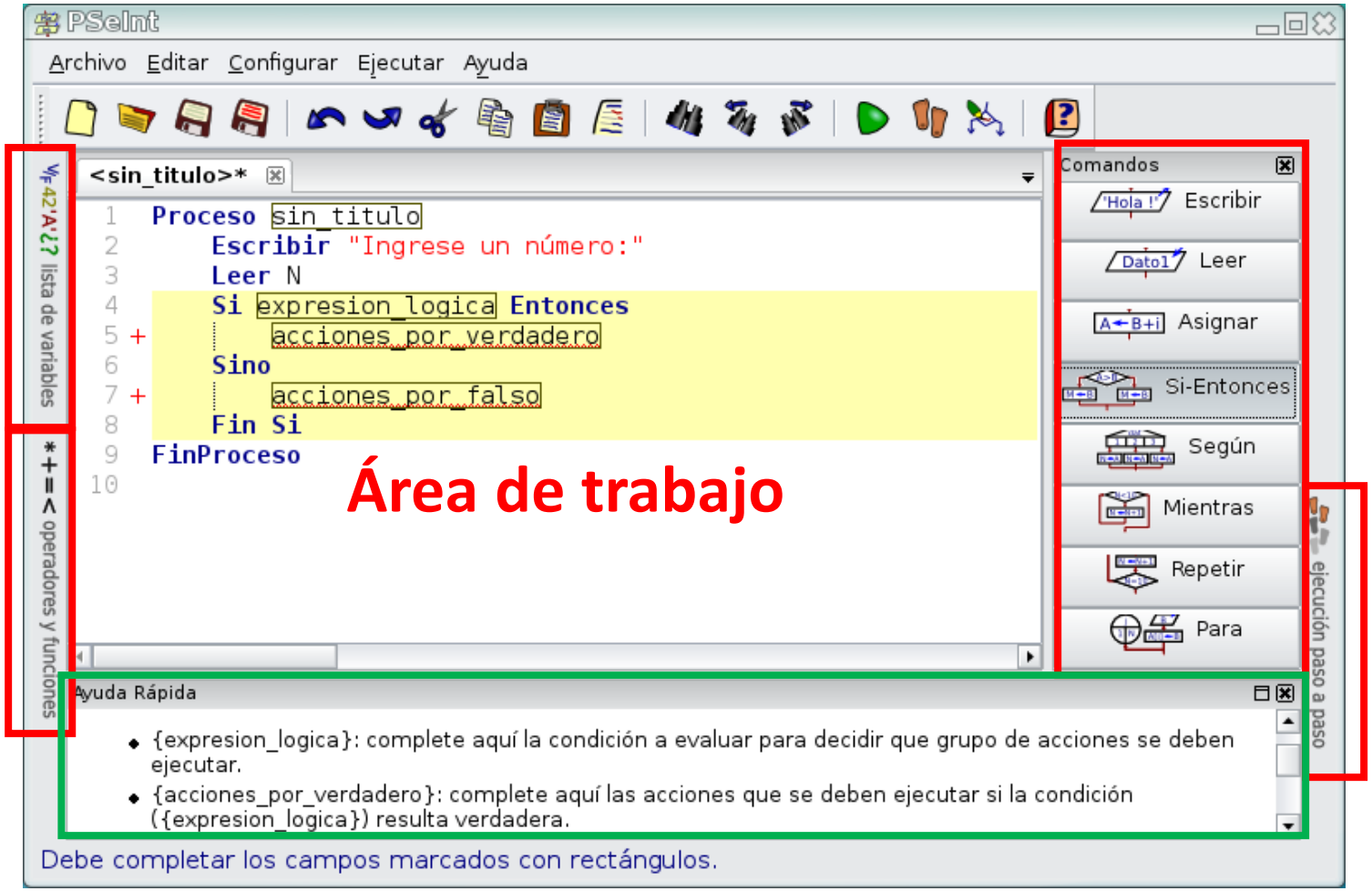
- Lo vamos a emplear como si se escribiera el **algoritmo en papel**. Para ello, durante la asignatura, no vamos a emplear las ayudas que proporciona este software.
- Se puede ejecutar con **datos de prueba** y ver el resultado. Así verificamos si es correcto o no.
- En definitiva, es mejor que escribir el pseudocódigo en un papel.

Además…

- Introduciremos **buenas prácticas** desde el principio, antes de emplear un lenguaje de programación.
- Estas buenas prácticas son independientes del software o lenguaje de programación. Y son comunes.

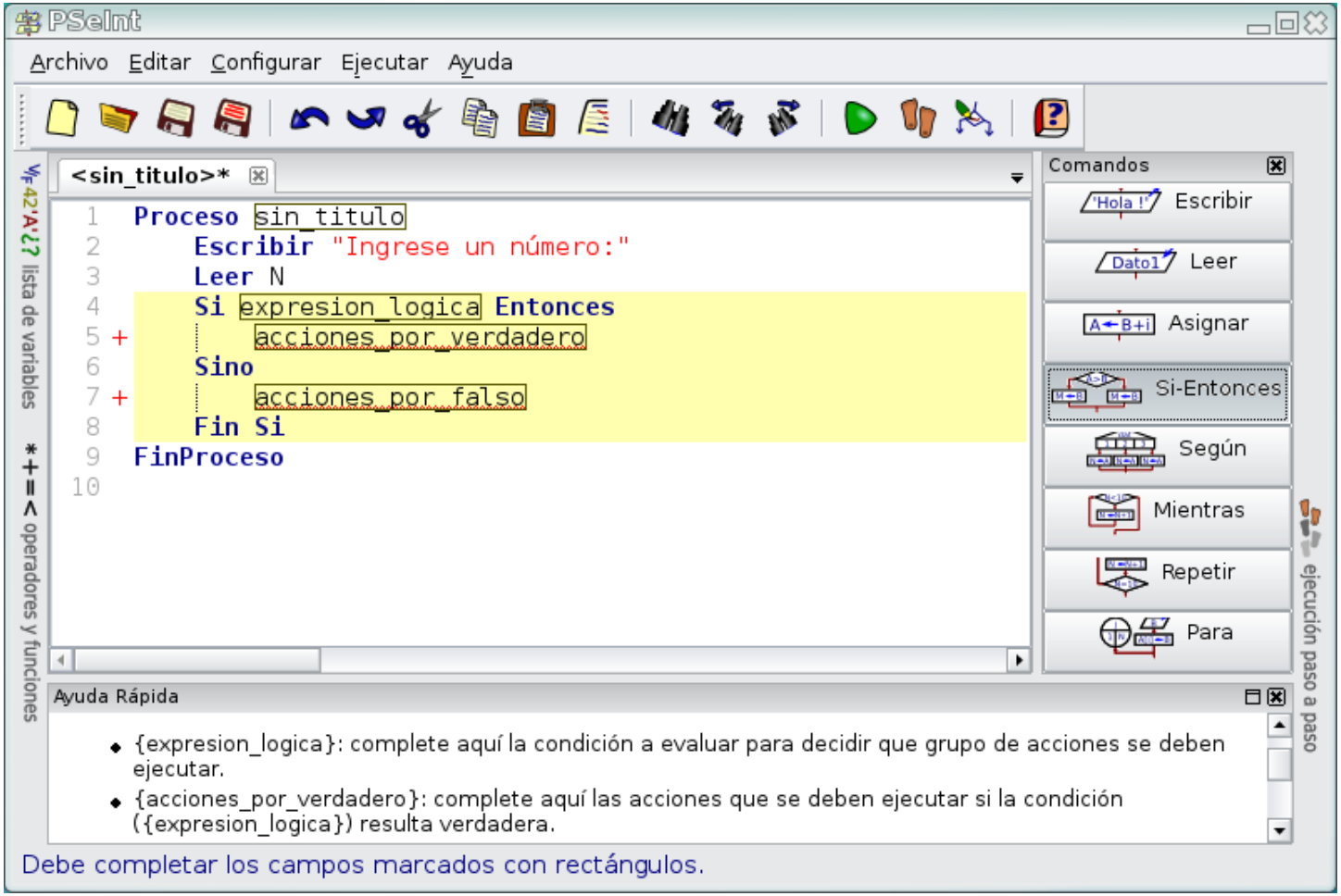
ENTORNO DE PSEINT

PARTES QUE COMPONEN LA INTERFAZ



ENTORNO DE PSEINT

CONCENTRARSE EN LO IMPORTANTE



No vamos a emplear ayudas ni plantillas de comandos.

CONSTRUCCIÓN DE ALGORITMOS CON PSEINT

- ESQUEMA A SEGUIR SIEMPRE:

```
// Descripción breve del algoritmo
// Autor
// Fecha
Proceso <<identificador>> //Cabecera

// sección de declaraciones (constantes y variables)
Definir <<lista de variables>> Como <<tipo de dato>>

// sección de acciones
        acción 1
        acción 2
            …
 FinProceso
```

TIPOS DE ACCIONES

- **DE ASIGNACIÓN:**

  variable ← expresión

  – Interpretación: Se evalúa la expresión de la parte derecha y se almacena el resultado en la variable de la parte izquierda

  – La variable ha de ser del mismo tipo que la expresión.

  - Ejemplos:

A←12	A=12;
B←A	B=12;
C←B+A	C=12+12=24;
C←C+1	C=24+1=25
A← D-1	A= ?-1= ?

TIPOS DE ACCIONES

- **ENTRADA o LECTURA DE DATOS**
  - Ejemplo:

```
Leer empleados, puestos, turnos;
```

Si el usuario teclea 50 y pulsa INTRO, 25 y pulsa INTRO, 3 y pulsa INTRO, la variable *empleados* será igual a 50, la variable *puestos* valdrá 25 y la variable *turno* 3.

Equivale a:

```
empleados <- 50
puestos <- 25
Turnos <- 3
```

TIPOS DE ACCIONES

- **SALIDA o ESCRITURA DE DATOS**

  Escribir <<lista de argumentos separados por comas>>;

  – **Interpretación**: el ordenador escribe de manera ordenada los argumentos en un dispositivo de salida (habitualmente el monitor)

  - Ejemplo: Supongamos que `anio <- 2006`

```
Escribir anio;
```

Muestra por pantalla: 2006

```
Escribir "Estamos en el año ", anio-1;
```

Muestra por pantalla: Estamos en el año 2005

Otro ejemplo:

```
Escribir "2*6";
Escribir 2*6;
```

ELEMENTOS BÁSICOS

- **Palabras reservadas**: aquellas que tienen un significado predefinido en el lenguaje de programación (o en pseudocódigo). Ej: BEGIN, END, IF, DEFINIR, …

- **Identificadores:** nombres asociados a diferentes elementos de un lenguaje (tipos, constantes, variables…)
  - Predefinidos: vienen definidos en el lenguaje. Ejemplo: int, float, void…
  - Definidos por el usuario: son elegidos por el usuario para nombrar variables, constantes, etc.

ELEMENTOS BÁSICOS

- **Identificadores:** Para nombrar a los diferentes elementos se han de seguir unas reglas generales:
  - Los identificadores empiezan por una letra, a la que continuan opcionalmente más letras y números. Ej: `miCoche1, el1declase`.
  - No suelen permitirse caracteres especiales en el nombre Ej: `mi*coche` es erróneo.
  - Utilizar nombres claros: `nombre, apellidos, telefono, mesCosecha`, etc.
  - No emplear `tabla`, `variable` o `lista` como identificador.
  - No emplear `ñ` ni acentos.

ELEMENTOS BÁSICOS

- **Caracteres especiales**: Representan alguna operación o acción especial en el lenguaje. Ejemplo: `+ - * / := . , ; = < > <=`

  `>= <> ( ) [] (* *) {}`

- **Comentarios**: Sirven para documentar el programa. Dependiendo del lenguaje de programación van delimitados por caracteres especiales.

  Ejemplo en C: `/* Esto es un comentario */`

  En Pascal: `{Esto es un comentario}`

  En PSeInt: `// Esto es un comentario`

ELEMENTOS BÁSICOS

- **DATOS SIMPLES:**
  - Números enteros (int).
    - Ejemplos de números enteros: 9, -34, 0
  - Números reales (float, double).
    - Ejemplos de números reales: 9.0, -34.4, 0.000032
  - Lógicos (boolean).
    - cierto o verdadero (true).
    - falso (false).
  - Carácter (char).
    - Ejemplos de caracteres alfabéticos: ‘a’, ‘A’, ‘v’
    - Ejemplos de caracteres numéricos: ‘0’, ‘1’, ‘9’
    - Ejemplos de caracteres especiales: ‘+’, ‘-’, ‘<’

ELEMENTOS BÁSICOS

- **DATOS SIMPLES:**
  - Estos tipos de datos los emplea PSeInt
  - Una variable debe declarase antes de utilizarse por primera vez (depende del lenguaje).
  - En Pseint las declaramos SIEMPRE.

En PseInt:

```
Definir <var1> , <var2> , ... , <varN>  Como
[REAL/ENTERO/LOGICO/CARACTER];
```

- **Ejemplos:**

```
Definir letraPulsada Como Caracter;
Definir cantidadTotal, gasto Como Real;
Definir apertura Como Logico;
```

CONSTRUCCIÓN DE ALGORITMOS CON PSEINT

Nuestro primer algoritmo con PSeInt
empleando buenas prácticas y
aplicando recomendaciones.

ELEMENTOS BÁSICOS

- **EXPRESIONES**
  - Son combinaciones adecuadas de constantes, variables, símbolos de operación, paréntesis y funciones.
  - Está compuesta de operandos y operadores.
  - Toda **expresión** posee un **tipo**, que se calcula operando y hallando el resultado.
  - El resultado indica el tipo de dato de la expresión.
  - Pueden ser: **aritméticas, relacionales, lógicas y de carácter.**
  - Ej: (2+3*5.2/5)^2 Operandos: 2, 3, 5, 5.2. Operadores: ^ + * /

ELEMENTOS BÁSICOS

- **OPERACIONES ARITMÉTICAS** (En orden de prioridad)

OJO!!! No todos los operadores existen en todos los lenguajes de programación

Operador	Significado	Operandos	Resultado
^	Potencia	Entero o real	Entero o real
*	Multiplicación	Entero o real	Entero o real
/	División	Real	Real
div	Div. Entera	Entero	Entero
mod o %	Modulo (resto)	Entero	Entero
+	Suma	Entero o real	Entero o real
-	Resta	Entero o real	Entero o real

ELEMENTOS BÁSICOS

- **OPERACIONES ARITMÉTICAS**

OJO!!! No todos los operadores existen en todos los lenguajes de programación o se representan de la misma forma

- En PSeInt no existe el operador **DIV (División entera)**. En su lugar:

```
Escribir TRUNC(19/3);  //  Presenta en pantalla: 6

Escribir 19/3;         // Presenta en pantalla: 6,3333
```

- En PSeInt el operador potencia es: ↑ ó ALT + 24

- En Javascript y Python operador potencia: **

ELEMENTOS BÁSICOS

- **OPERADORES RELACIONALES**

  Permiten comparar **caracteres** (código ASCII) o **números**.

Operador	Significado	Operandos	Resultado
<	Menor que	Del mismo tipo	Booleano
>	Mayor que	Del mismo tipo	Booleano
= ==	Igual que	Del mismo tipo	Booleano
<=	Menor o igual	Del mismo tipo	Booleano
>=	Mayor o igual	Del mismo tipo	Booleano
<> !=	Distinto de	Del mismo tipo	Booleano

ELEMENTOS BÁSICOS

- **OPERADORES LÓGICOS**
  - Son operadores que actúan sobre valores booleanos y devuelven un booleano.
  - Existen dos constantes lógicas: *verdadero* (*true*) y *falso* (*false*). Las variables lógicas pueden tomar sólo estos dos valores.

A	B	***not*** A	A ***and*** B	A ***or*** B
false	false	true	false	false
false	true	true	false	true
true	false	false	false	true
true	true	false	true	true

Prioridad →

\+ -

ELEMENTOS BÁSICOS

- **PRECEDENCIA DE OPERADORES**

Operadores	*Categoría*	*Nivel de Precedencia*
( )	Paréntesis	Máximo: 1
–, ++, - -, NOT (unitarios)	Negación, unitarios	2
***, /, DIV, MOD, AND (binarios)**	Multiplicativos	3
+, –, OR (binarios)	Aditivos	4
=, < >, <, <=, >, >= (binarios)	Relacionales	Mínimo: 5

Igual precedencia: asociatividad Izda->Dcha

ELEMENTOS BÁSICOS

- **FUNCIONES INTERNAS con PSeInt:**

Operador	Significado	Operandos	Resultado
abs(x)	Valor absoluto	Entero o real	Entero o real
atan(x)	Arco tangente	Entero o real	Real
cos(x)	Coseno	Entero o real	Real
exp(x)	Exponencial (e)	Entero o real	Real
ln(x)	Log natural	Entero o real	Real
azar(x)	Num. aleatorio	Entero o real	Entero aleatorio en el rango [0;x-1]

ELEMENTOS BÁSICOS

- **FUNCIONES INTERNAS**

Operador	Significado	Operandos	Resultado
redon(x) (round)	Redondeo	Real	Entero más cercano a x
seno(x) (sin)	Seno	Entero o real	Real
raiz(x) (sqrt)	Raiz cuadrada	Entero o real	Real
trunc(x)	Truncamiento	Real	Entero

ELEMENTOS BÁSICOS

- **Constantes**: Son datos que NO van a cambiar durante toda la ejecución del programa.

```
Definir ESTRELLA Como Caracter;
ESTRELLA <- "*";

Definir PI Como Real;
PI <- 3.141592;
```

- **Variables**: Son datos que pueden cambiar.

```
Definir saldo, reintegro Como Real;
Definir nota, hora, dedos Como Entero;
```

FLUJO DE CONTROL DE UN ALGORITMO

– **Tres estructuras de control de flujo:**

Programación estructurada

Secuencial

Selección: ***`Si`*** *(If)* , ***`Según`*** *…* ***`Hacer`*** *(switch)*

Repetitiva o iterativa: ***`Desde`*** *(For)*, ***`Mientras`*** *…* ***Hacer*** *(while)*, ***`Repetir / Hasta Que`***.

*El **flujo de control es secuencial** salvo que empleemos selección o repetitivas/iterativas.*

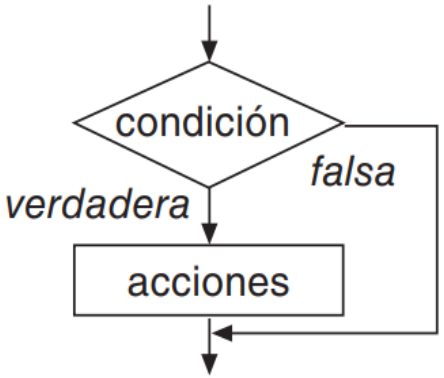
Bifurcación incondicional*:* ***`ir-a`*** *(goto). NO EMPLEAR!! Cualquier algoritmo se puede reescribir para no utilizar esa acción.*

ESTRUCTURAS SELECTIVAS

- **Si … Entonces** o alternativa **simple**
  - Pseudocódigo:

```
Si <condición> Entonces
    <acciónA>  //Puede haber más de una
FinSi
```

  - Diagrama de flujo

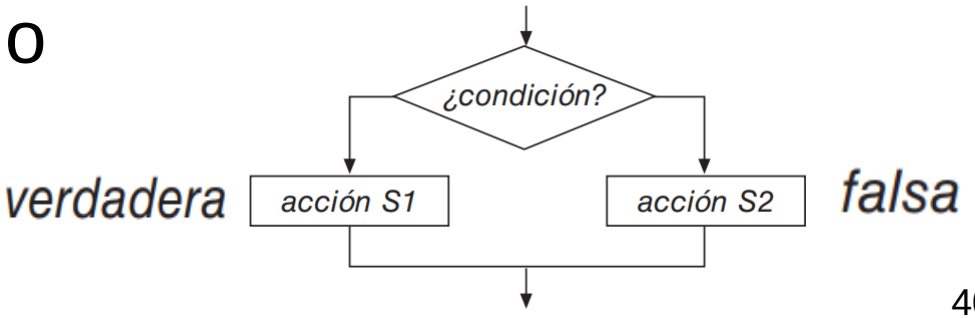


ESTRUCTURAS SELECTIVAS

- **Si … Entonces… / SiNo …** o alternativa **doble**
  - Pseudocódigo

```
Si <condición> Entonces
    <acciónA>   //Puede haber más de una
SiNo
    <acciónB>   //Puede haber más de una
FinSi
```

  - Diagrama de flujo



Ejercicio nº 1

ESTRUCTURAS SELECTIVAS

- Alternativa **múltiple**

```
Si <condición1> Entonces
    <acción A>              //Puede haber más de una
SiNo
    Si <condición2> Entonces
        <acción B>      //Puede haber más
    SiNo
        <acción C>      //Puede haber más
    FinSi
FinSi
```

Ejercicio nº 2

ESTRUCTURAS SELECTIVAS

- **Según … Hacer** o alternativa **múltiple**

  Mucho más claro si tenemos más de dos elecciones posibles.

```
Segun <expresión> Hacer
   e1, e2: <acciónA>  //Puede haber más
                …
    eN: <acciónX> //Puede haber más
    De Otro Modo:
       <acciónY>
FinSegun
```

```
Leer(a);
Segun (a*2) Hacer
  2: Escribir "a=1";
  4: Escribir "a=2";
De Otro Modo:
  Escribir "a es mayor a 2";
FinSegun
```

ESTRUCTURAS SELECTIVAS

- **Según …. Hacer:**
  - Expresión selectora: Tiene que ser de tipo ordinal, solo puede tomar valores fínitos:
    - enteros, caracter, booleano.

    Muchos lenguajes solo permiten enteros. **Con PSeInt: perfil estricto.**

  - Las etiquetas (e1,e2…) tienen que tener valor/es constante/s:
    - constantes, literales, listas de constantes o subintervalos

  - Las acciones: Una acción o bloque de acciones

Ejercicio nº 3

ESTRUCTURAS REPETITIVAS

Bucle: Bloque de acciones que se repiten *n* veces. ¿Qué contiene?

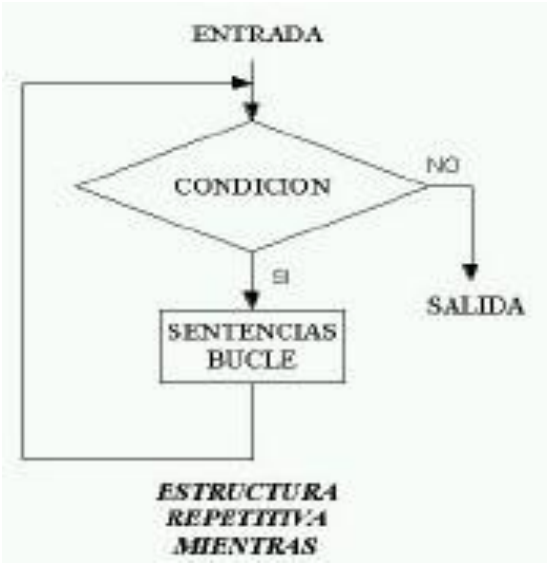
Iteración: Es cada repetición. ¿Cuántas veces?

- **Mientras …. Hacer (while)**.

  Itera por verdadero (True).

  Condición de salida del bucle:

  Al principio.




```
Mientras <condición_continuación> Hacer
    <cuerpo del bucle>
FinMientras
```

ESTRUCTURAS REPETITIVAS

- **Repetir / Hasta que:**

  Itera por falso (False).

  Condición de salida del bucle: Al final

  El bucle se ejecuta al menos una vez.

```
Repetir
   <cuerpo del bucle>
Hasta Que <condición_finalización>
```

– Diagrama de flujo:

Ejercicio nº 4

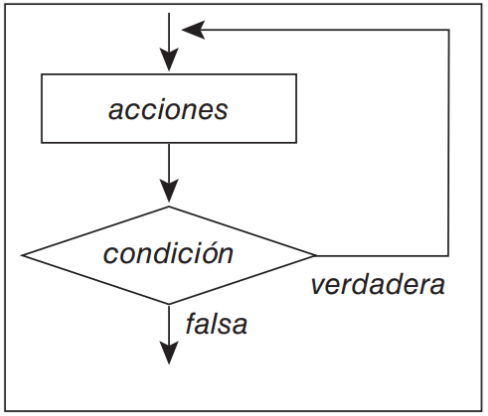
ESTRUCTURAS REPETITIVAS

- **hacer…mientras (do-while)**.

  Itera por verdadero (True).

  Condición de salida del bucle: Al final.

  El bucle se ejecuta al menos una vez.



– Pseudocódigo:

```
Hacer
    <cuerpo del bucle>
Mientras <condición_terminación>
```

En PSeInt o se emplea 'Repetir / Mientras Que' o 'Repetir / Hasta Que.'

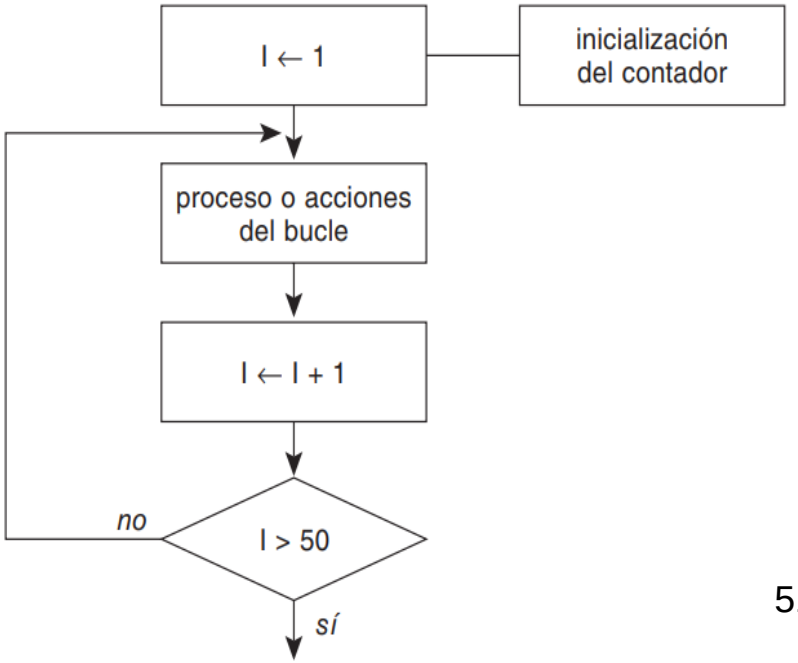
ESTRUCTURAS REPETITIVAS

- **Para (for)**.

  Cuando se conoce el número de iteraciones = Es fijo.

```
Para i ← vi Hasta vf Con Paso [incremento] Hacer
    <proceso>
FinPara
```

i = índice o contador



ESTRUCTURAS REPETITIVAS

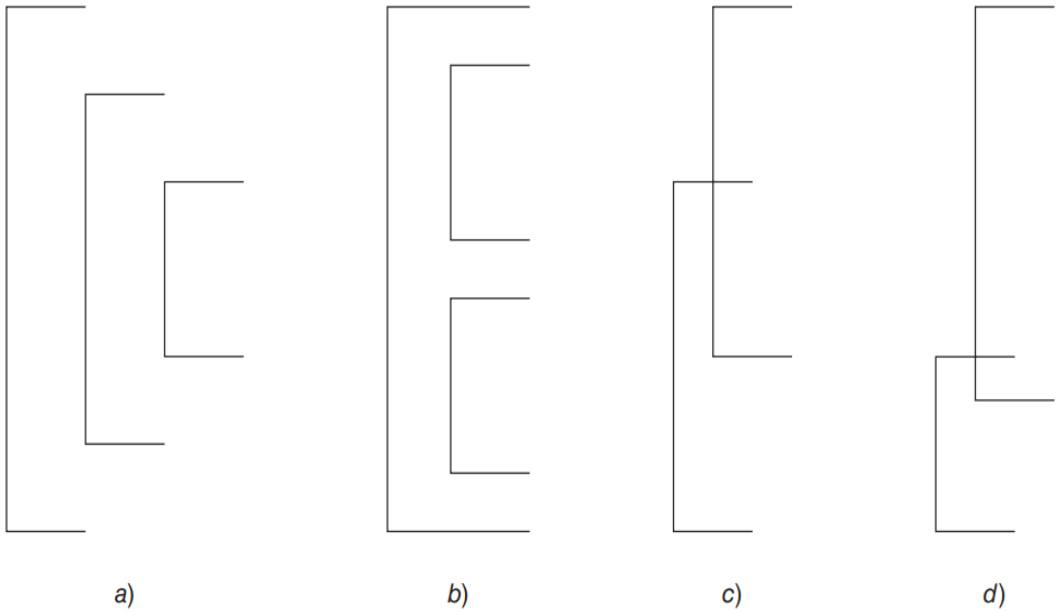
- **Para (for)**.
  - Las expresiones inicial **`vi`** y final **`vf`** cumplen:
    - son tipos compatibles con el tipo del índice.
    - se evalúan sólo una vez.
    - dependiendo de sus valores y del incremento (paso) puede no ejecutarse el cuerpo del bucle.
  - **Recomendación importante:**
    - no modificar el valor de **`vi`** y **`vf`** dentro del ciclo

ESTRUCTURAS REPETITIVAS

- **Para (for)**.
  - La variable índice o contador "**`i`**" debe ser de un tipo ordinal.
  - **Restricciones para índice `i`**:
    - No modificarla dentro del cuerpo (Leer, asignación,..)
    - No usar en `Para` anidado. **Emplear otro índice!!**
    - Al salir del ciclo `Para`, se supone que su valor está sin definir.

ESTRUCTURAS REPETITIVAS

- **ESTRUCTURAS ANIDADAS.**



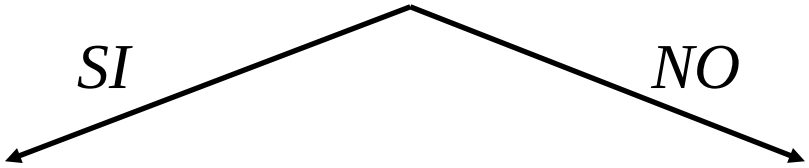
a) y b) correctas // c) y d) incorrectas (error)

La estructura interna debe estar incluida totalmente dentro de la externa y no puede existir solapamiento.

ESTRUCTURAS REPETITIVAS

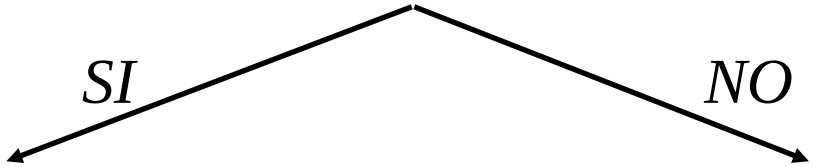
Recomendaciones técnicas:

¿ Se sabe de antemano *cuántas veces* el cuerpo del bucle ha de ejecutarse?



Usar ciclo PARA

¿ Se sabe de antemano que el cuerpo del bucle ha de ejecutarse *al menos una vez*?



Usar ciclo REPETIR / HASTA QUE

Usar ciclo MIENTRAS … HACER

CARACTERÍSTICAS DE UN BUEN ALGORITMO

- **Corrección:**
  - El algoritmo debe funcionar. Puede ser difícil de asegurar en algoritmos complejos.
- **Eficiencia:**
  - El algoritmo no debe desaprovechar recursos (memoria y tiempo de ejecución).
- **Claridad:**
  - El algoritmo debe estar bien documentado.

Ejercicios nº 5 y 6

(para hacer en casa)