The background features a central server tower with a large padlock in front of it. Four laptops are arranged around the server. Red binary code (0s and 1s) is scattered across the scene, creating a digital atmosphere.

MÓDULO I

SISTEMAS OPERATIVOS Y APLICACIONES

INFORMÁTICAS

Definiciones

- **Algoritmo**

Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

- **Programa (RAE)**

Conjunto unitario de instrucciones que permite a una computadora realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos, etc.

- **Programa (Otra definición)**

Secuencia de instrucciones que han sido escritas en un lenguaje de programación concreto, entendibles por un ordenador, y que al ejecutarla permiten realizar un trabajo o resolver un problema

Pasos para programar

1. El ordenador carece de sentido común.
2. Debemos pensar / diseñar el algoritmo en lenguaje natural.
3. Debemos cubrir todos los posibles casos que puedan suceder.
4. Generar un diagrama de flujo si es necesario.
5. Desarrollar el programa en pseudo-código.
6. Pasar el pseudo-código a lenguaje de programación.

Ejemplo – Arrancar un coche

1. Abrir el coche.
2. Abrir la puerta.
3. Introducir la llave en el mecanismo de arranque.
4. Girar la llave de encendido.
5. Soltar la llave una vez el coche arranque.

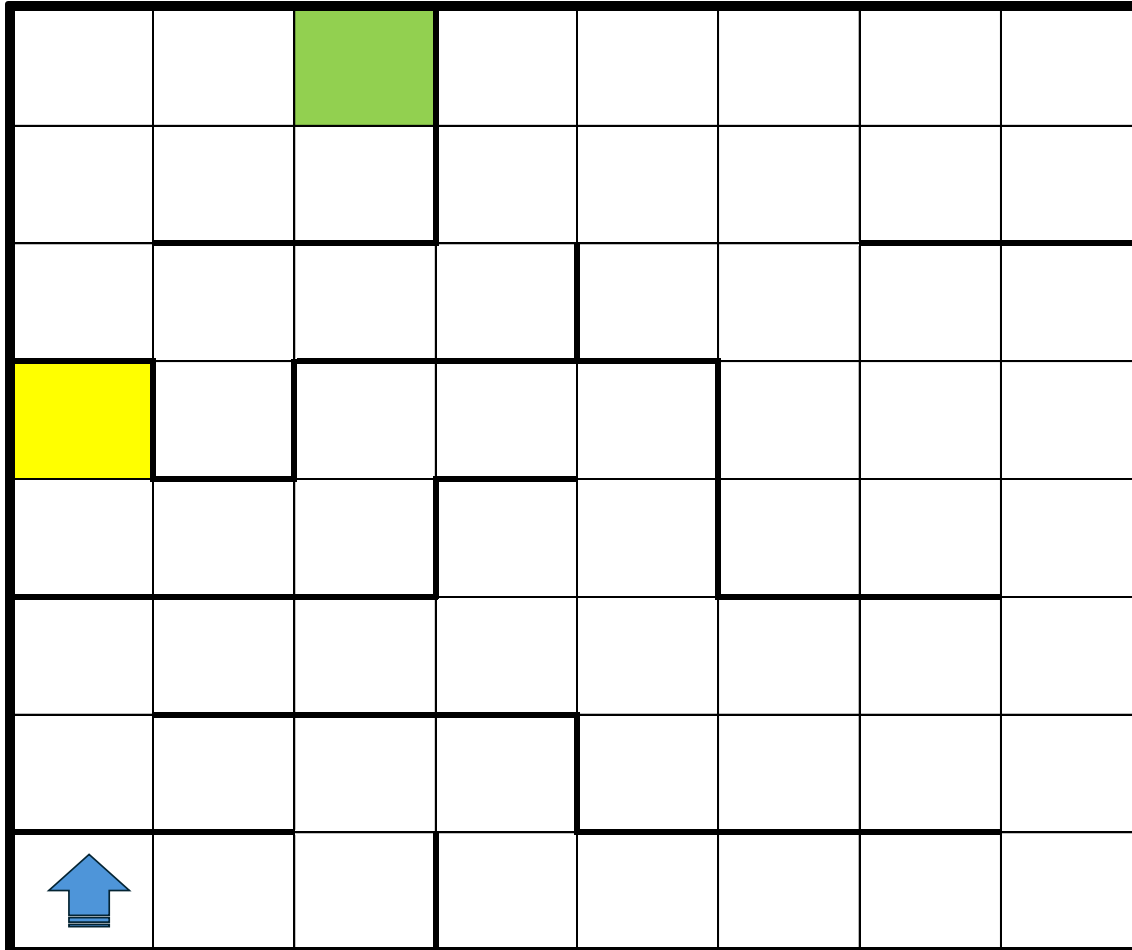
Ejemplo – Arrancar un coche

- No todas las situaciones están cubiertas:
 - El coche puede estar abierto.
 - La puerta del conductor puede estar abierta.
 - El coche puede estar arrancado.
 - Controlar si no hay puesta una marcha, etc.
- Una persona tiene la capacidad para detectar de forma casi automática esas situaciones, pero la máquina no.

Pasos para programar

- Necesitamos estructuras de decisión, por ejemplo:
 - Si el coche está cerrado, lo abro.
 - Si la puerta está cerrada, la abro.
 - Entro al coche
 - Etc, etc.

Desarrollar un algoritmo

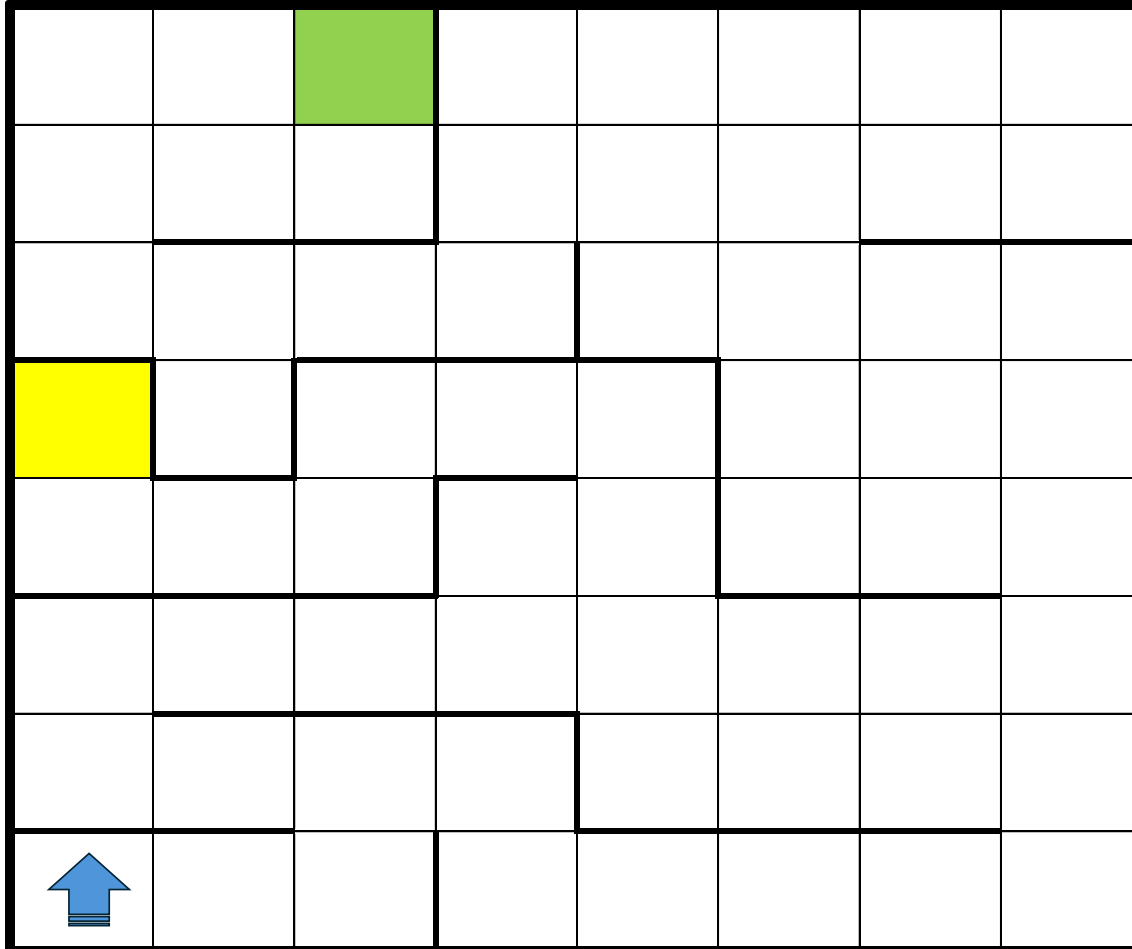


- Instrucciones posibles
 - Girar D
 - Girar I
 - Avanzar casilla

Objetivo

Llegar a la casilla F (FIN)
pasando por la B (BONUS)

Desarrollar un algoritmo



- Instrucciones posibles

- Girar D
- Girar I
- Avanzar casilla
- Repetir n veces

Objetivo

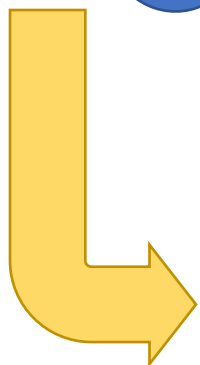
Llegar a la casilla F (FIN)
pasando por la B (BONUS)

SUMAR DOS NÚMEROS

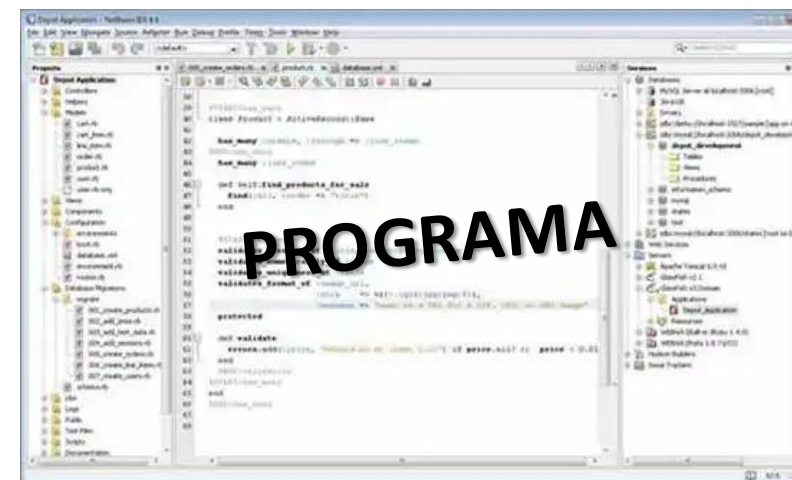
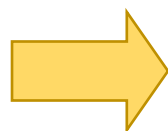
¿Cómo lo resolvemos nosotros?

1. Preguntar por el primer número y anotarlo en una casilla, por ej. en la casilla a1
2. Preguntar por el segundo número y anotarlo en una casilla, por ej. en la casilla a2
3. Sumar al número de la casilla a1 el de la casilla a2 y el resultado anotarlo en la casilla a3
4. Decir el número que está en la casilla a3

Problema >>>> programa solución



COMPRENSIÓN



Lenguaje de Programación

- **Léxico**

Vocabulario, conjunto de las palabras de un idioma, o de las que pertenecen al uso de una región, a una actividad determinada, etc.

- **Sintaxis**

Gram. Parte de la gramática que estudia el modo en que se combinan las palabras y los grupos que estas forman para expresar significados, así como las relaciones que se establecen entre todas esas unidades.

Inform. Conjunto de reglas que definen las secuencias correctas de los elementos de un lenguaje de programación.

- **Semántica**

Asocia un significado (la acción que debe llevarse a cabo) a cada posible construcción del lenguaje.

Lenguajes de programación

- Árbol, Arbol, Árvol, Arvol, Hárbol, ...
- El, árbol: y").hojas ;...
- El árbol tiene sus hojas de color rojo, todas ellas verdes claras como el carbón.
- for, if, guail, du
- for int if;
- //Para sumar 2 números
- `resu = num1 * num2;`

Lenguajes de programación por cercanía al lenguaje humano

- **Lenguaje máquina (programa objeto)**

Instrucciones directamente entendidas por el ordenador (sin traducción para la CPU). Son por lo tanto dependientes de la máquina.

- **Lenguaje de bajo nivel (ensamblador)**

Las instrucciones se escriben en códigos alfabéticos (mnemotécnicos). Es dependiente de la máquina.

- **Lenguaje de alto nivel (programa fuente)**

Instrucciones escritas con palabras similares a las empleadas en los lenguajes “humanos”.

Son independientes de la máquina y para su ejecución y necesitan ser traducidos a instrucciones en lenguaje máquina.

Compilador

- Traduce un programa fuente, escrito en un lenguaje de alto nivel, a un programa objeto, en lenguaje máquina.
- El programa fuente puede estar contenido en uno o varios ficheros.
- El programa objeto puede almacenarse como un archivo independiente para su posterior procesamiento, sin tener que repetir de nuevo el proceso de traducción.
- Una vez traducido el programa, su ejecución es independiente del compilador.

Intérprete

- Hace que un programa fuente escrito en un lenguaje de alto nivel vaya, sentencia a sentencia, traduciéndose y ejecutándose.
- Toma una sentencia fuente, la analiza e interpreta, dando lugar a su ejecución inmediata, sin crear un fichero objeto.
- La ejecución del programa está supervisada por el intérprete.
- Cada vez que se ejecuta el programa, se tiene que volver a analizar.
- Si localiza un error en tiempo de ejecución, puede corregirse el error y continuar a partir de la instrucción errónea.

Lenguajes de programación

- **Programación Imperativa**

Se deben detallar los pasos necesarios para realizar una tarea. Se describe el “cómo” hacerla.



- **Programación declarativa**

Se describe el resultado a obtener y los mecanismos disponibles sin detallar los pasos necesarios para obtenerlos. Se describe el “qué” hacer.



Paradigmas de Programación



Los lenguajes más utilizados

PYPL

Tiobe Index



SUMAR DOS NÚMEROS

- **Entrada:** Dos números enteros (**num1 y num2**)
- **Salida:** Un número entero (**resu**)
- **Restricciones:** ninguna

PROBLEMA COMPRENSIÓN

1. Preguntar por el primer número y anotarlo en una casilla, por ej. en la casilla num1
2. Preguntar por el segundo número y anotarlo en una casilla, por ej. en la casilla num2
3. Sumar al número de la casilla num1 el de la casilla num2 y el resultado anotarlo en la casilla resu
4. Decir el número que está en la casilla resu

ALGORITMO

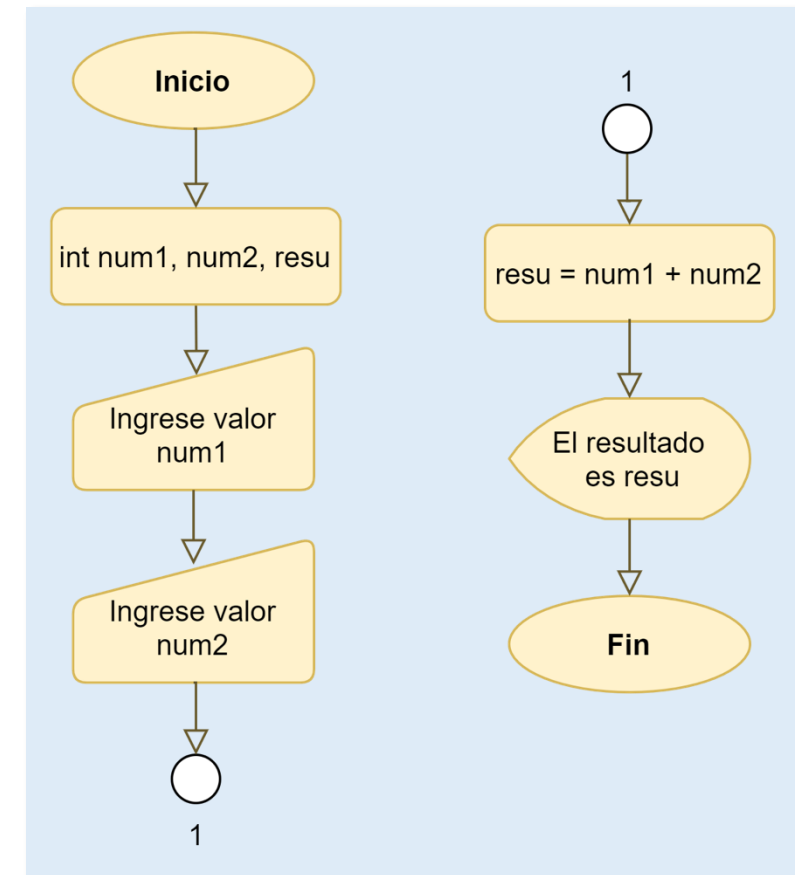
```
Proceso suma_numeros
  Definir num1, num2, resu Como Entero;
  Escribir "Ingrese primer valor";
  Leer num1;
  Escribir "Ingrese segundo valor";
  Leer num2;
  resu = num1 + num2;
  Escribir "El resultado es: ", resu;
FinProceso
```

El Algoritmo

Pseudo-código

```
Proceso suma_numeros
  Definir num1, num2, resu Como Entero;
  Escribir "Ingrese primer valor";
  Leer num1;
  Escribir "Ingrese segundo valor";
  Leer num2;
  resu = num1 + num2;
  Escribir "El resultado es: ", resu;
FinProceso
```

Diagrama de Flujo



Variable

- Una variable asocia un nombre único (identificador) a una dirección de memoria donde puede almacenarse un valor determinado durante la ejecución de un programa. El valor puede cambiar durante la ejecución.
- Una variable, una vez creada, tiene asociado no sólo un valor sino también un tipo de dato.
- Tanto los caracteres que se pueden utilizar en un identificador como los tipos de datos con los que se pueden definir las variables, depende del lenguaje de programación concreto.
- En cualquier momento de la ejecución del programa se puede consultar el valor de la variable o bien cambiarlo.

Variables

Tipos de datos más frecuentes

- Número entero (int) entero
- Número real (double) real
- Carácter (char) caracter
- Lógico (boolean) lógico
- Cadena (string) cadena

Declaración

```
Definir num1 Como Entero;  
Definir x, y, resultado Como Real;
```





Inicialización y asignación (l-value)




```
num1 = 56;  
x = 3.14;
```

Uso del valor (r-value)

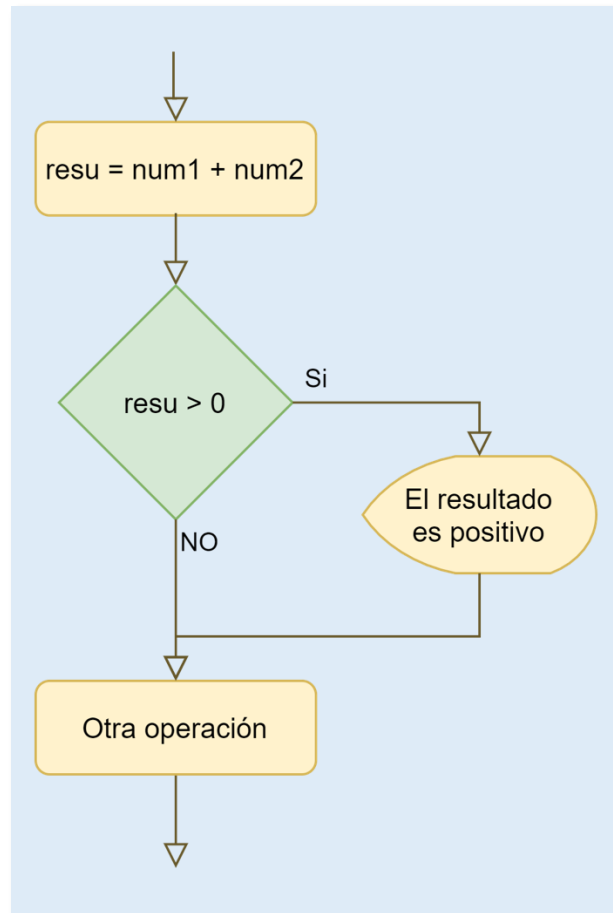
```
resultado = x * y;  
Escribir resultado;
```

Diagramas de Flujo

	Inicio / Fin
	Proceso
	Entrada
	Salida

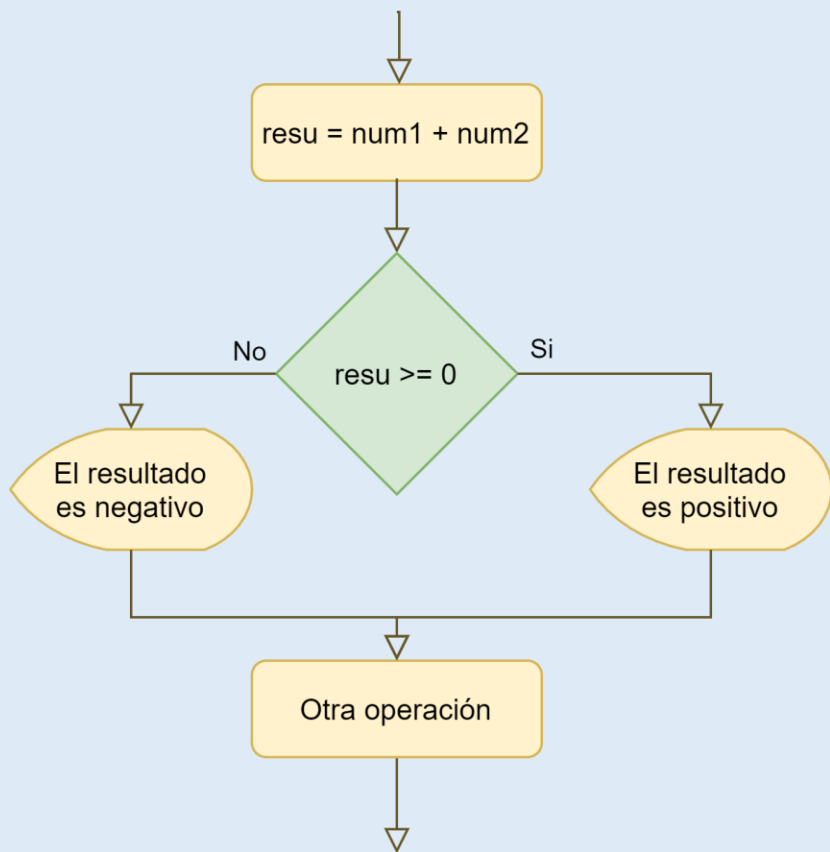
	Condicional
	Iteración
	Conector

Estructura de control Condicional Simple



```
...  
resu = num1 + num2;  
Si resu > 0 Entonces  
    Escribir "El resultado es positivo";  
FinSi  
//Otra operación  
...
```


Estructura de control Condicional Doble



...

```
resu = num1 + num2;
```

```
Si resu >= 0 Entonces
```

```
    Escribir "El resultado es positivo";
```

```
SiNo
```

```
    Escribir "El resultado es negativo";
```

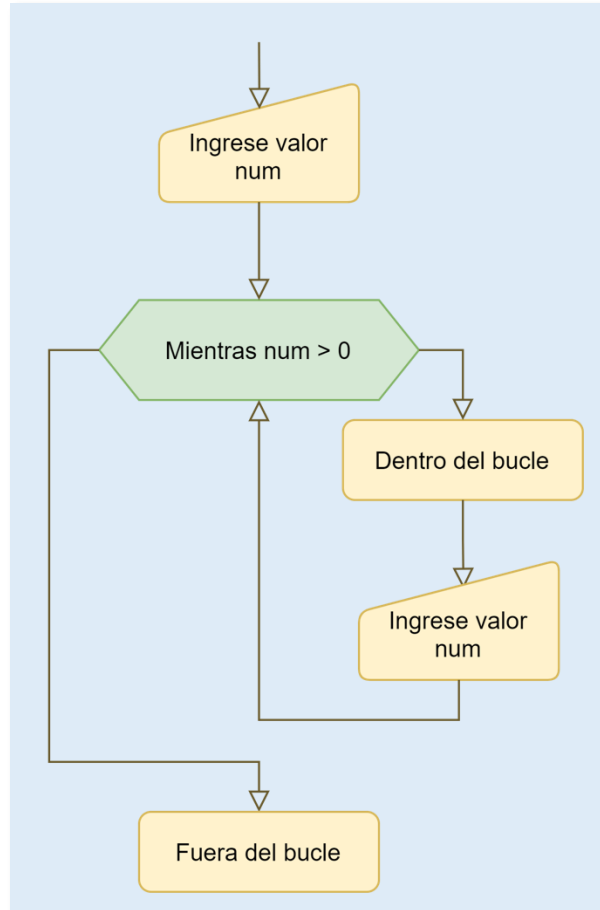
```
FinSi
```

```
//Otra operación
```

...

Estructura de control

Iteraciones - Mientras (while)



...

Escribir "Ingrese primer valor";

Leer num;

Mientras num > 0 **Hacer**

// Dentro del bucle

Escribir "Ingrese segundo valor";

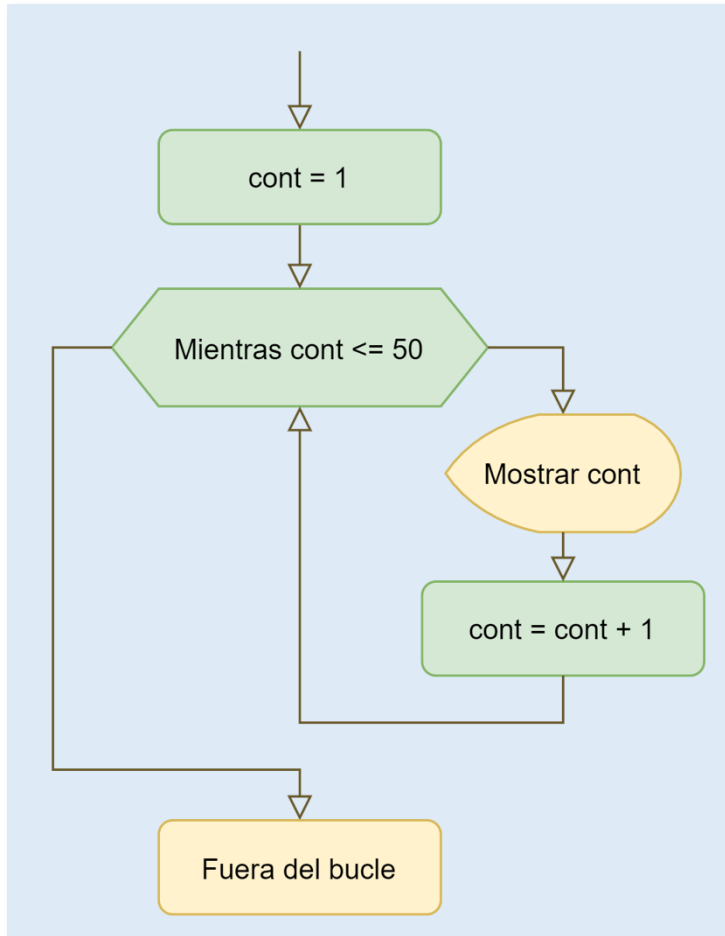
Leer num2;

FinMientras

//Fuera del bucle

...

Implementación - contador



...

cont = 1;

Mientras cont <= 50 Hacer

Escribir cont;

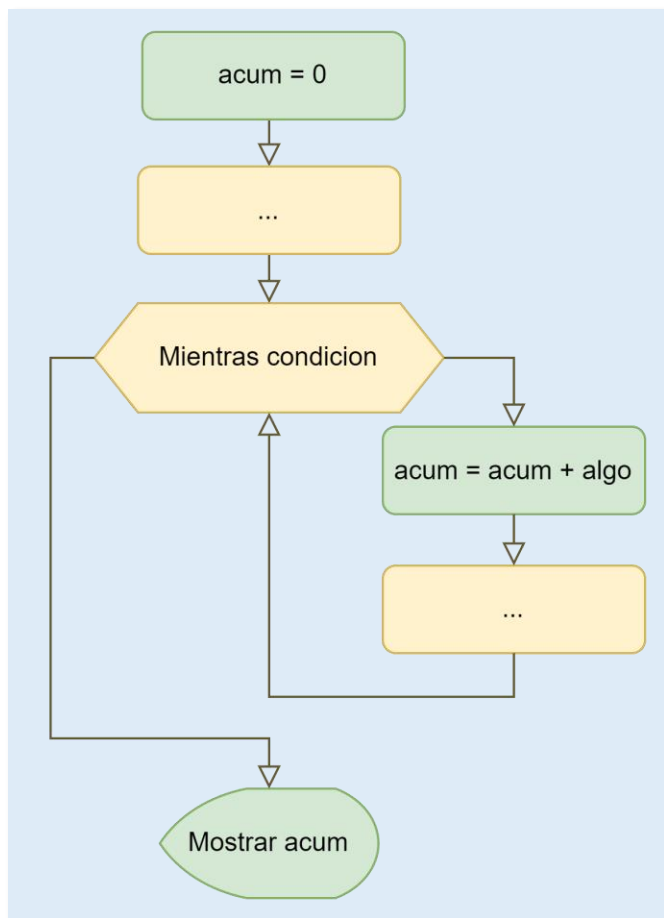
cont = cont + 1;

FinMientras

//Fuera del bucle

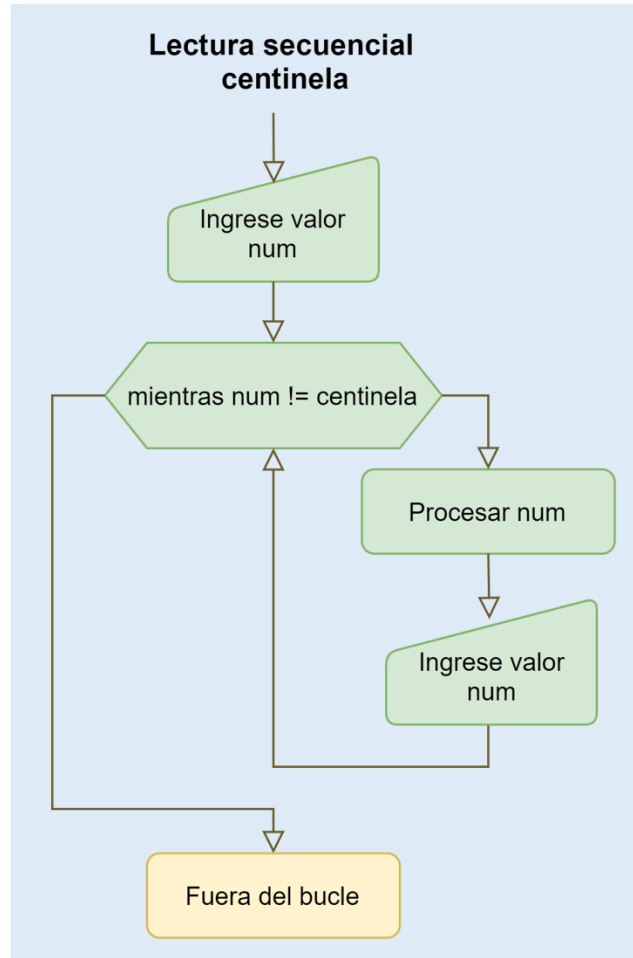
...

Implementación - acumulador



```
...  
acum = 0;  
...  
Mientras <condición> Hacer  
    acum = acum + algo;  
    ...  
FinMientras  
Escribir acum;  
...
```

Implementación – lectura secuencial fin de lectura con valor centinela



...

Leer num;

Mientras num != <valor_centinela> Hacer

//Procesar num

Leer num;

FinMientras

//Continua fuera del bucle

...

Ej.: Resoluci

er grado

■ Problema:

Resolver una ecuación de primer grado. Calcular el valor de x en
 $a.x + b = 0$

■ Entrada:

Dos números reales (a , b)

■ Salida:

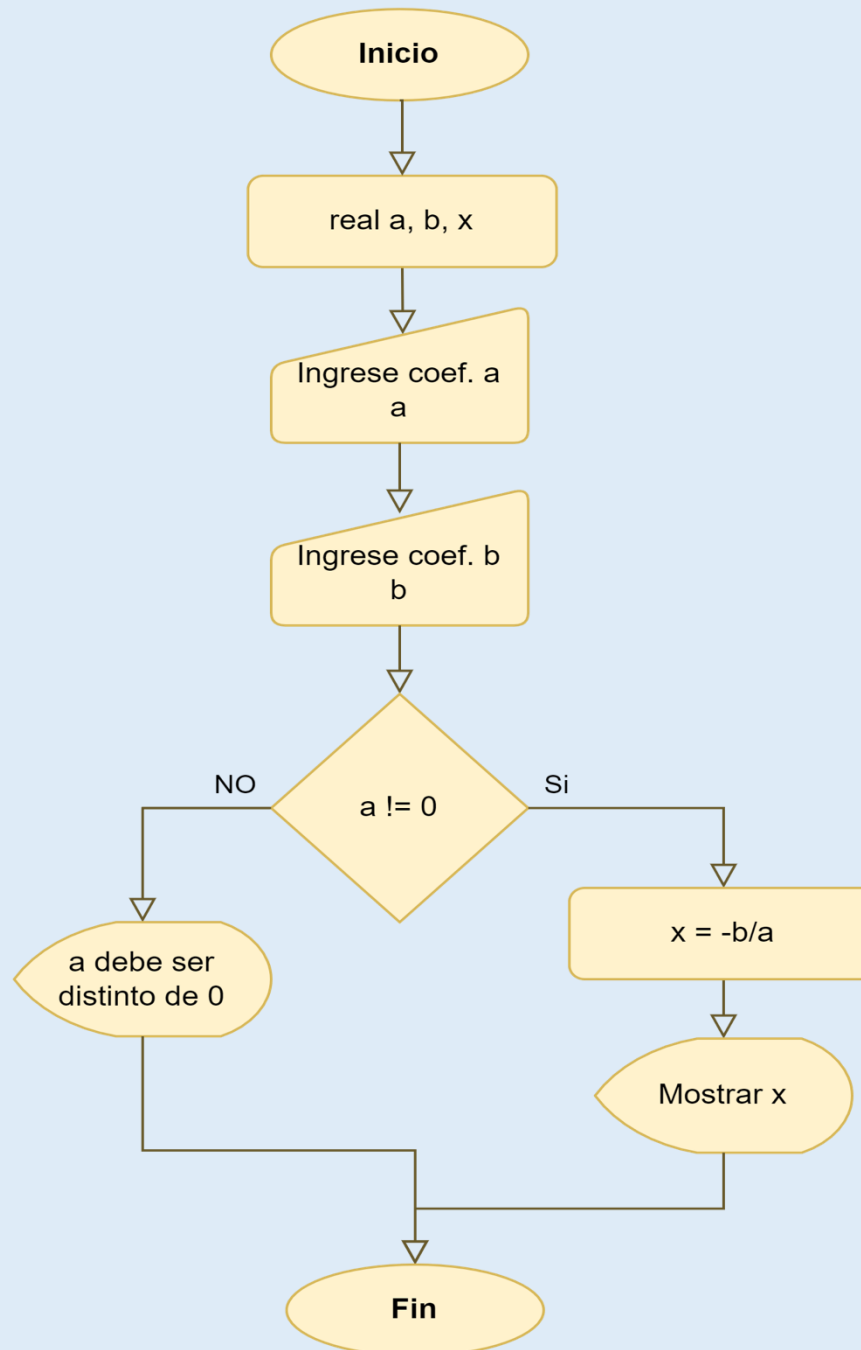
Un número real (x)

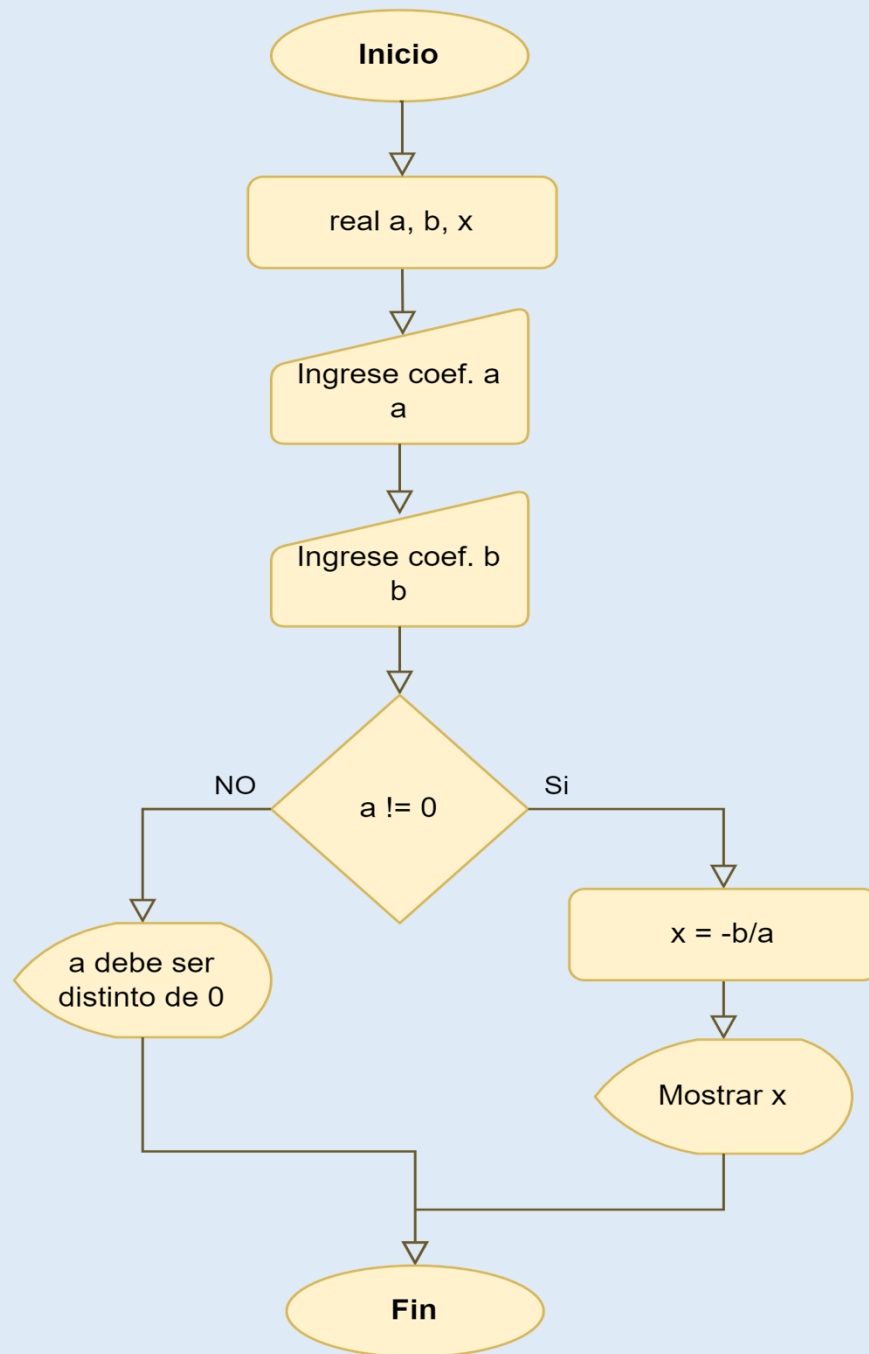
■ Cálculo:

$$x = -b / a$$

■ Restricciones:

$$a \neq 0$$





Proceso ec_primer_grado

Definir a, b, x Como Real;

Escribir "Ingrese coef. a";

Leer a;

Escribir "Ingrese coef. b";

Leer b;

Si $a \neq 0$ Entonces

$x = -b / a$;

Escribir x;

SiNo

Escribir a, "no puede ser 0";

FinSi

FinProceso

Ej.: Resolución ecuación de primer grado

■ Problema:

Resolver una ecuación de primer grado. Calcular el valor de x en
 $a.x + b = 0$

■ Entrada:

Dos números reales (a , b)

■ Salida:

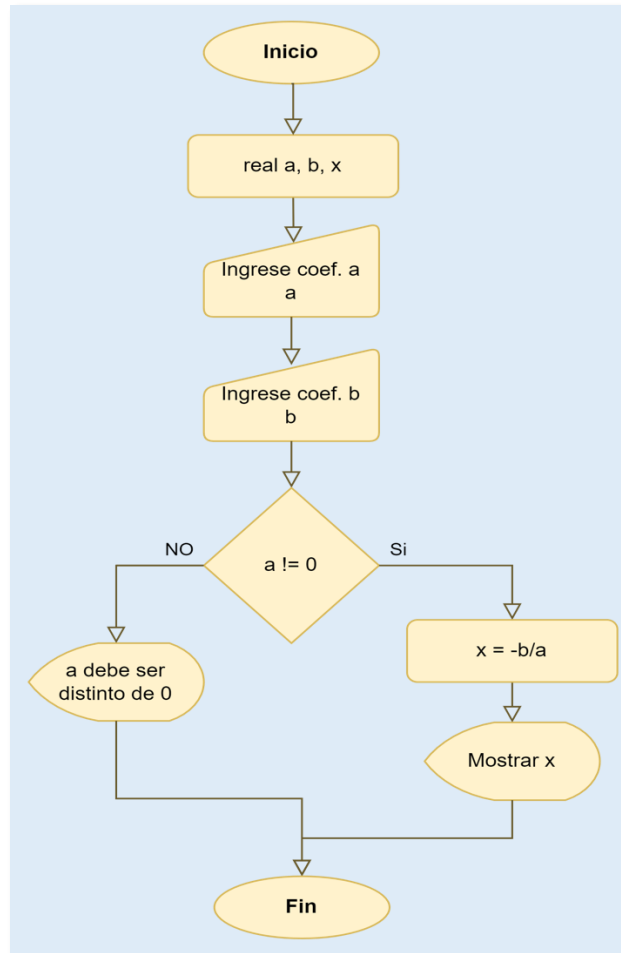
Un número real (x)

■ Cálculo:

$$x = -b / a$$

■ Restricciones:

$$a \neq 0$$



Proceso ec_primer_grado

Definir a , b , x Como Real;

Escribir "Ingrese coef. a ";

Leer a ;

Escribir "Ingrese coef. b ";

Leer b ;

Si $a \neq 0$ Entonces

$$x = -b / a;$$

Escribir x ;

SiNo

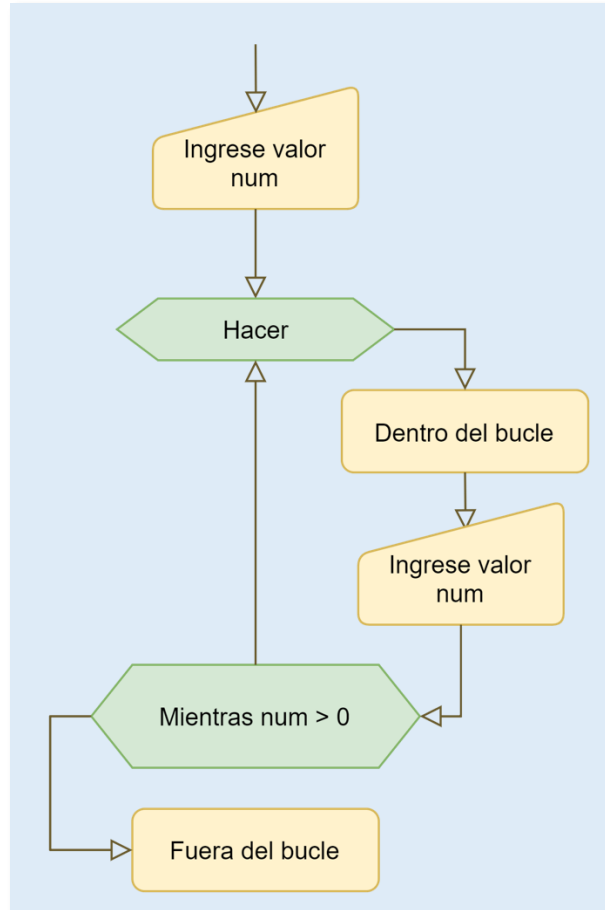
Escribir a , "no puede ser 0";

FinSi

FinProceso

Estructura de control

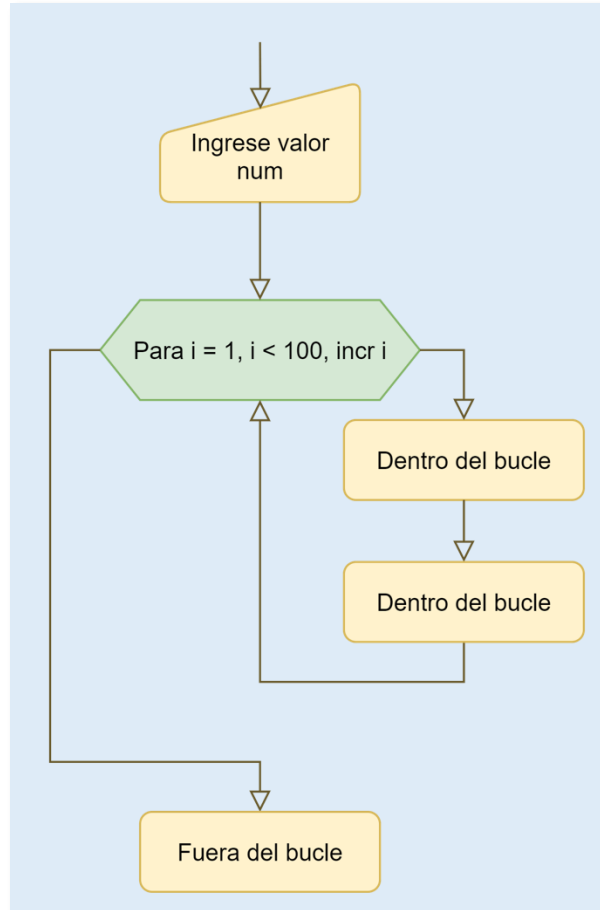
Iteraciones – Repetir Mientras (do –while)



```
...
Escribir "Ingrese primer valor";
Leer num;
Repetir
    //Dentro del bucle
    Escribir "Ingrese valor";
    Leer num;
    Mientras Que num > 0;
//Fuera del bucle
...
```

Estructura de control

Iteraciones – Para (for)



...

Escribir "Ingrese primer valor";

Leer num;

Para i = 1 **Hasta** 100 **Con Paso** 1 **Hacer**

//Dentro del bucle

//Dentro del bucle

FinPara

//Fuera del bucle

...

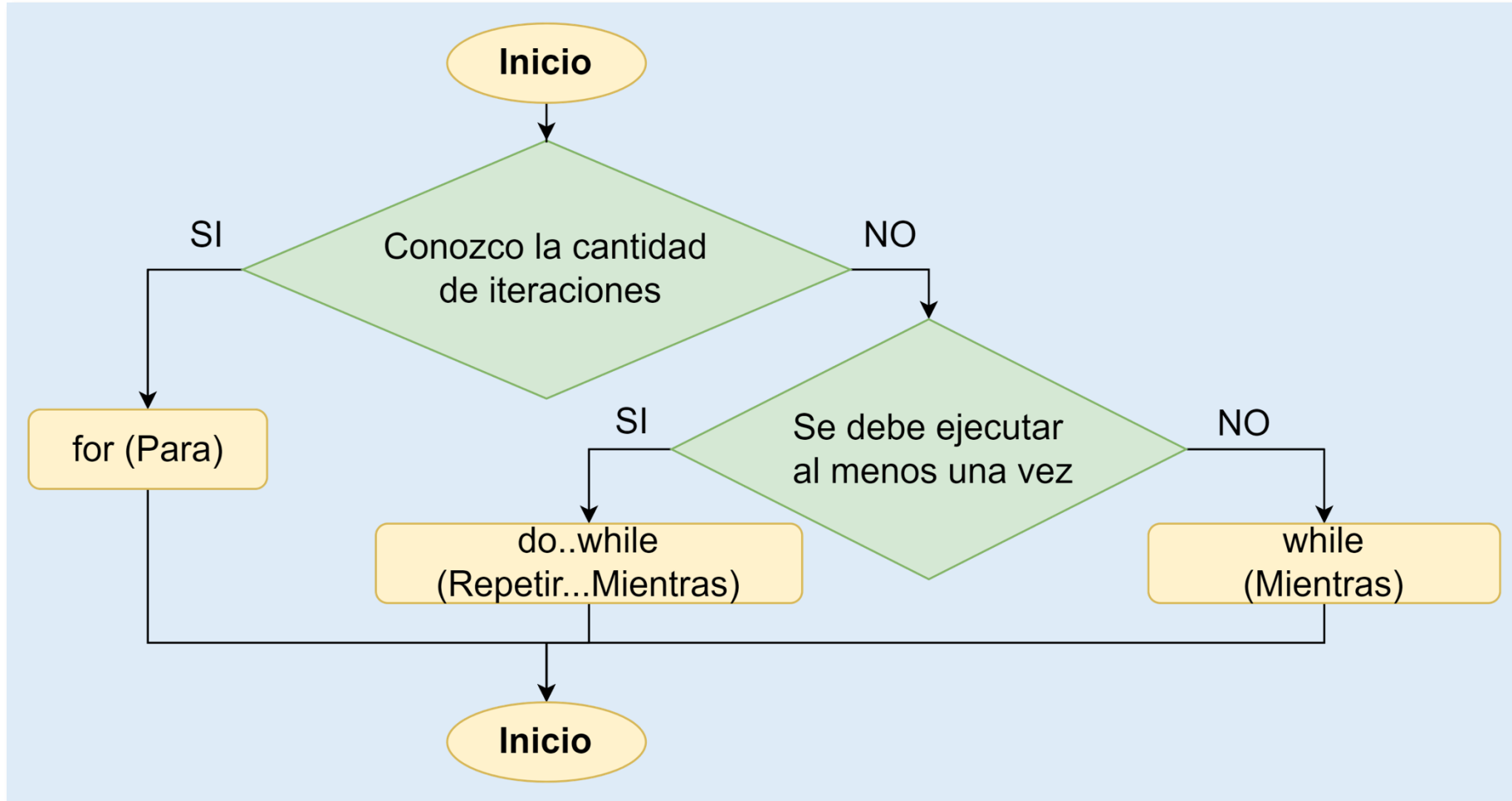
Estructuras iterativas

```
...  
Mientras <exp_lógica> Hacer  
    //Dentro del bucle  
FinMientras  
//Continua fuera del bucle  
...
```

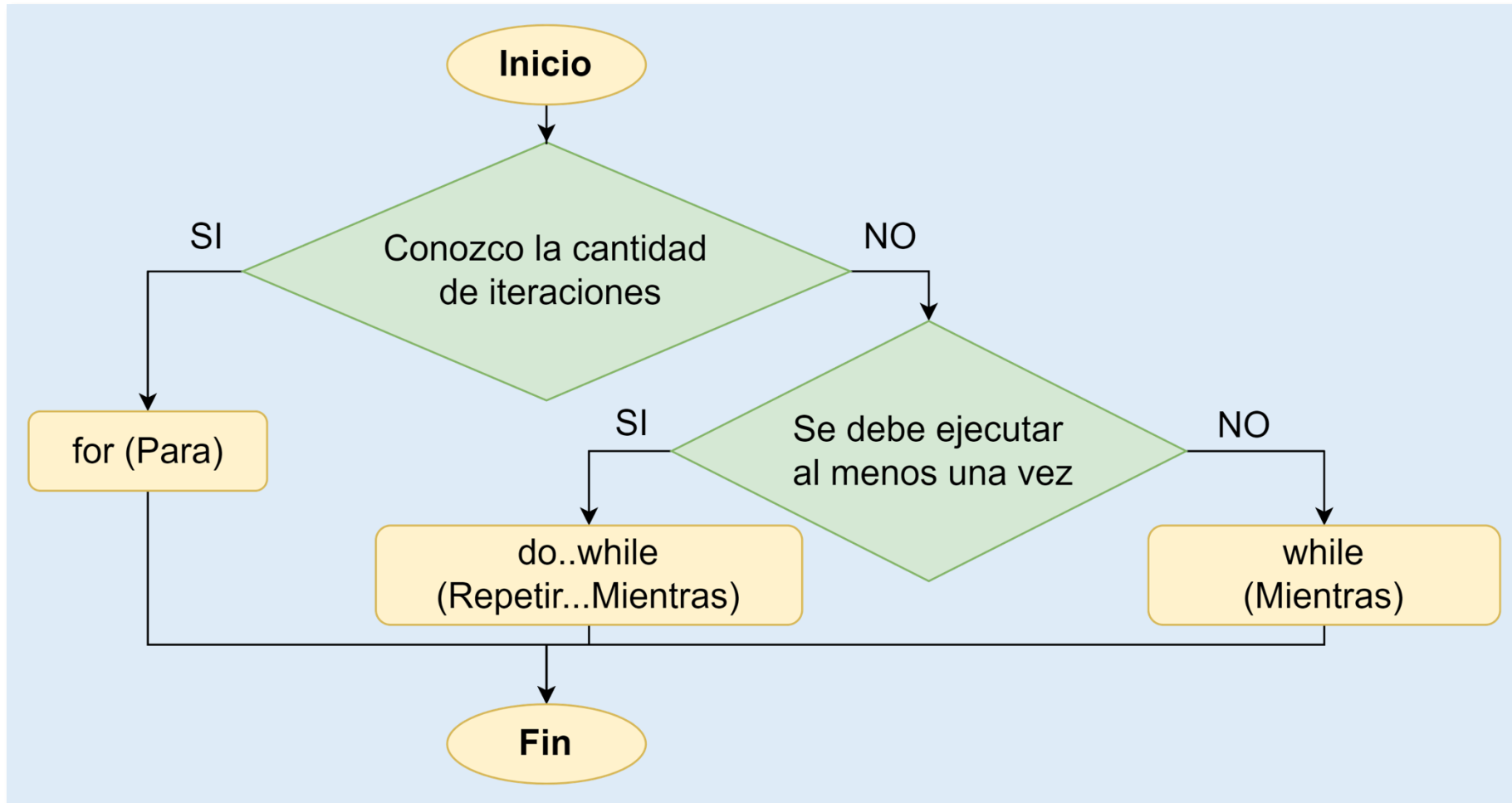
```
...  
Repetir  
    //Dentro del bucle  
Mientras Que <exp_lógica>  
//Continua fuera del bucle  
...
```

```
...  
Para <var=valor> Hasta <valor_final> Con Paso <valor_paso> Hacer  
    //Dentro del bucle  
FinMientras  
//Continua fuera del bucle  
...
```

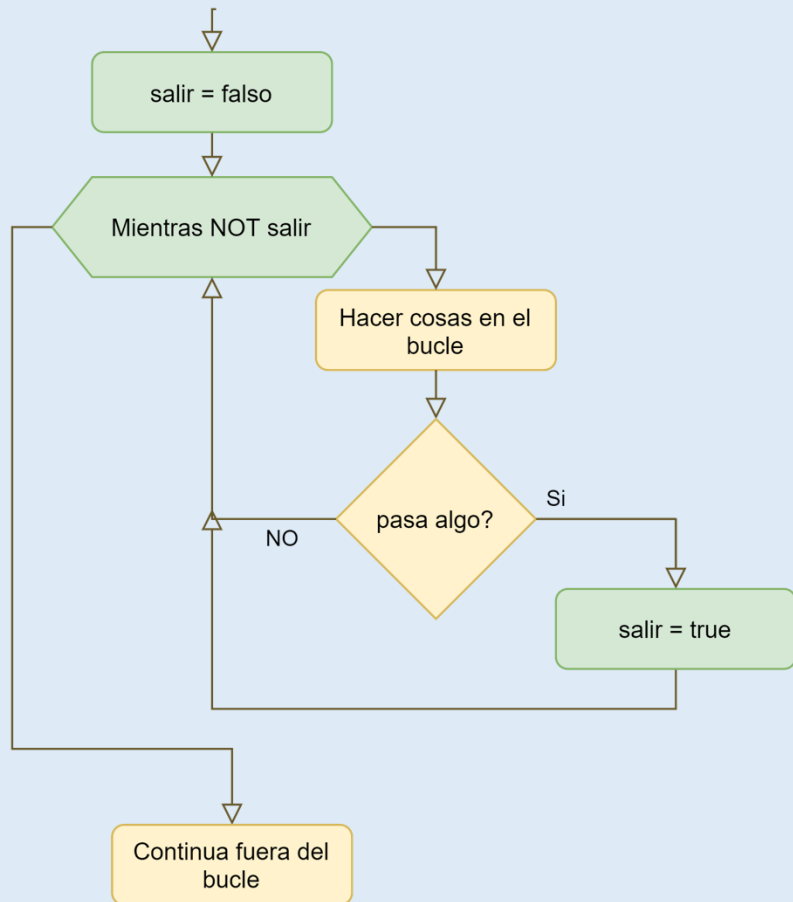
¿Qué estructura iterativa utilizar?



¿Qué estructura iterativa utilizar?



Implementación - bandera



...

salir = Falso;

Mientras !salir Hacer

//Hacer cosas dentro del bucle

Si <condicion> Entonces

salir = Verdadero;

FinSi

FinMientras

//Continua fuera del bucle

...

Subrutinas

- Una subrutina es una secuencia de instrucciones, un subprograma, que se invoca desde un programa para ejecutar una tarea específica tantas veces como sea necesario.
- Una vez que finaliza, regresa el control al lugar de donde fue llamada.
- Puede ser definida por el programador o bien proporcionada por el lenguaje de programación.
- Para realizar la tarea, la subrutina, puede que necesite datos de entrada (parámetros) y puede que necesite devolver algún resultado (retorno). Ni los parámetros ni el retorno son siempre necesarios.