

- Fonaments de Programació (PDF)
  - ¿Qué es programar?
  - Què és un llenguatge de programació?
    - Llenguatge de màquina
    - Llenguatge de Baix Nivell
    - Llenguatge d'Alt Nivell
- Fundamentos de algoritmia (PDF)
  - ¿Qué es un algoritmo?
  - Regles que han de complir els algoritmes
  - Parts d'un algoritme
  - Algoritmo de la vida diària
  - Algoritmo computacional
  - Llenguatges algorítmics
  - ¿Qué es una variable?
  - ¿Cómo nombramos a una variable?
  - ¿Que es un tipo de dato?
  - Ejemplos de programaEn Python -> Las arrays no existen (matrices), pero si diccionarios etc
  - Prueba de escritorio EXAMEN\*\*\*\*\*

\*\*07/05/24

# Fonaments de Programació (PDF)

---

## ¿Qué es programar?

---

Programar: Programar és escriure instruccions específiques perquè una màquina entengui, processi i executi una sèrie de tasques.

Todos los procesos que hacemos “és programar”, entonces todo lo que hacemos en el ordenador lo es. Incluso las IA al calcular o al hacer búsquedas en ellas, es programar, lo que lo hace muy rápido porque ya lo tienen codificado.

Quan parlem de màquines ens referim a tots els “sistemes” que puguin processar informació (Ordinador, tablet,smartphone, electrodoméstics,...)

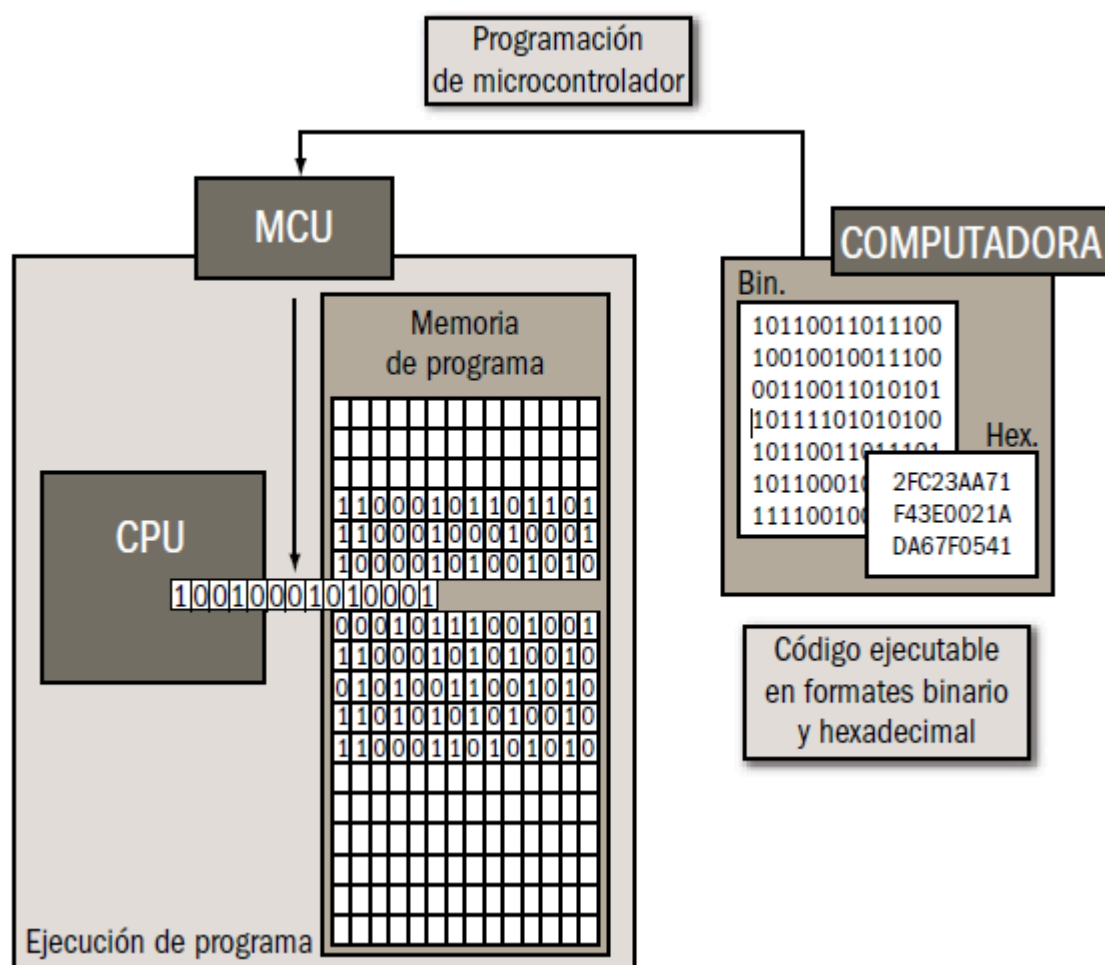
Todo tiene sistemas de procesamiento. Un smartwatch es mucho más potente en unidades de procesamiento que un ordenador de la nasa de hace 30 años

# Què és un llenguatge de programació?

Un llenguatge de programació és un llenguatge que ens permet comunicar-nos amb una màquina i escriure programes que permeti a aquesta interpretar les nostres instruccions

Podem classificar els **llenguatges** de programació en **tres grans categories**:

## Llenguatge de màquina



- Aquest tipus de llenguatge està escrit perquè sigui entès directament pel processador
- Lo que interpreta la màquina
- Les seves instruccions són cadenes binàries (0 i 1), que indiquen les operacions i la direcció de memòria que es farà servir

- També podem programar fent servir codi hexadecimal que converteix allò que escribim en termes binaris perquè l'ordinador ho pugui entendre

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

```

Administrator: Command Prompt Development - debug
C:\ics>debug
-e 100 b4 09 ba 09 01 cd 21 cd 20 48 6f 6c 61 2c 20 6d 75 6e 64 6f 24
-g
Hola, mundo
Program terminated normally

```

## Ventajas:

- Possibilitat de carregar (transferir un programa a la memòria) sense la necessitat de traduir les instruccions perquè l'ordinador les entengui.
- La velocitat d'execució és superior a qualsevol altre llenguatge de programació, perquè no cal traduir-lo.

## Inconvenientes:

- Els programes només s'executen al mateix procesador (CPU).
- Codificació més complexa i lenta.
- Dificultat per verificar i posar a punt els programes.

# Llenguatge de Baix Nivell

# Exemple ensamblador: Hola Mundo!

```
HOLAMU-1.ASM X
HOLAMU-1.ASM
1  ;model small // Acosta el espacio de memoria
2  ;el punto y coma (;) define comentarios
3  .....
4  .model small ;se define el tipo de modelo, corto
5  .data ;aquí se definen las variables que se van a ocupar, es el segmento de datos
6
7  mensaje db 'Este es mi Hola mundo!$' ;Mensaje
8
9  .code ;bloque de código
10 inicio: ;se define un comienzo del cuerpo del programa
11
12 mov ax,@data ;movemos al registro dx, los datos a utilizar
13 mov ds,ax ; de nuevo modemos los registros de derecha a izquierda
14
15 ;en ah siempre se cargan las funciones, en este caso la 09h, permite desplegar un mensaje en pantalla
16 mov ah,09h
17 mov dx,offset mensaje ;movemos al registro dx el contenido de mensaje
18 ; se llama a la interrupcion 21h
19 int 21h ;interrupcion
20
21 ; con esta instruccion terminamos el programa
22 mov ax, 4c00h
23 int 21h ;interrupcion
24 .stack ;segmento de pila
25
26 end inicio ;termina el segmento definido como inicio
27
28
29
30
```

- Són llenguatges que ensamblen grups de conmutadors necessaris per expressar una mínima lògica aritmètica (vinculats íntimament al hardware)
- En microprocesadors etc, el ordinador tampoc no l'interpreta com a tal
- 16 bits, Ensamblen conjunt de conmutadors
- Requereixen una fase de traducció al llenguatge màquina per poder ser executat directament per la computadora. (se necessita ensamblar)
- El llenguatge per excel·lència és l'ensamblador. Les instruccions es coneixen com nomenclàtics, per exemple:
  - Operacions aritmètiques: ADD (sumar), SUB (resta), DIV (divisió),...
- Actualment es fa servir en espais acadèmics o d'investigació, així com al treball amb micro-controladors (CMOS/BIOS) i electrònica (firmware).

## Ventajas:

- Major velocitat de codificació
- Major velocitat de càlcul (una instrucció en un llenguatge de Baix Nivell probablement equival a una instrucció en codi màquina)

## Inconvenientes:

- Dependència total de la màquina (programar en llenguatge ensamblador en PC és diferent de programar en Mac).
- Els programadors estan obligats a conèixer aspectes del hardware.

## Llenguatge d'Alt Nivell

En 1 sola línia puedes imprimir un hello word

## Exemple Python: Hola Mundo!

```
1 print("Hello Python World!")
```

## Altres exemples: Hola Mundo!

**C#**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hola Mundo");
            Console.ReadLine();
        }
    }
}

```

**Java**

```

1 public class HolaMundo
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Hola Mundo");
6     }
7 }

```

**PHP**

```

1 <html>
2 <head></head>
3 <body>
4 <?php
5     echo "Hola Mundo";
6 ?>
7 </body>
8 </html>

```

**VisualBasic**

```

Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
        label1.text = "Hola Mundo"
    End Sub
End Class

```

- Qualquiera que se asemeja al inglés y luego el ordenador lo interpreta.
- Aquests llenguatges són els més utilitzats pels programadors, estan dissenyats perquè les persones escriguin i entenguin els programes d'un mode molt més fàcil que els llenguatges màquina i ensambladors.
- Los que usan programadores.

**Ventajas:**

- Independència de la màquina. Independiente del Sistema Operativo (SO).
- Processos o funcions previamente definides. En entorno gráfico y hay definiciones ya preseteadas
- Temps d'aprenentatge relativament més curt. (puede que sí, pero depende y puede ser largo el aprendizaje)
- Permeten més flexibilitat al desenvolupador.

### Inconvenientes:

- No del todo real si trabajas bien con recursos de memoria
- No s'aprofita el 100% dels recursos de la màquina en comparació amb els llenguatges anteriors.
- Augment de l'ús de memoria.
- Temps d'execució relativament major (una línea de codi equival a vàries línies de codi màquina). aunque es rápida

(Esta clase entra en teoría)

## Fundamentos de algoritmia (PDF)

---

### ¿Qué es un algoritmo?

---

Un algoritme és una seqüència lògica de passos a fer per poder resoldre un problema determinat

- A los programadores nos viene bien el papel y boli, porque tenemos que desarrollar algoritmos y cuantos más micropasos tengamos puede ser más facil de resolver un problema más grande.
- La algoritmia sirve para tener los pasos necesarios para resolver un problema.
- A mejor definido, mejor será nuestro código.

## Regles que han de cumplir els algoritmes

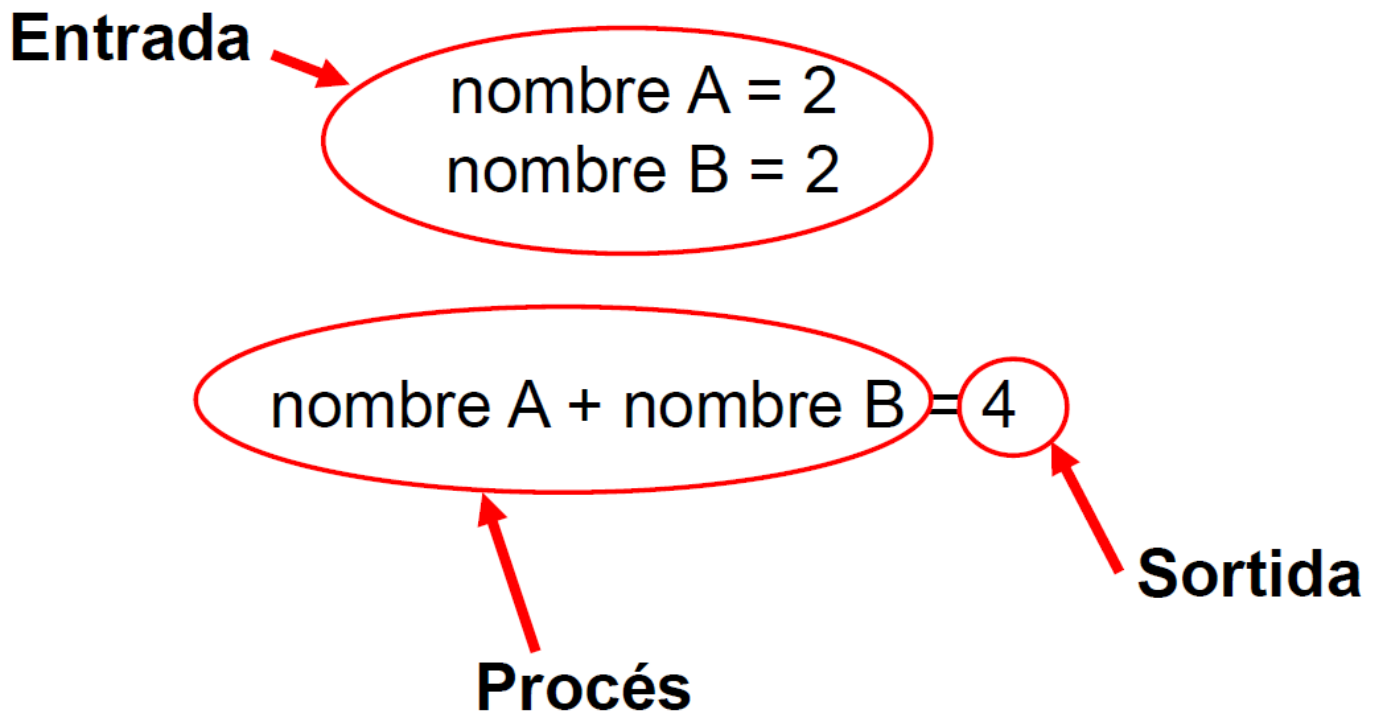
---

- Han de ser **precisos**: tenir un pas a pas lògic i puntual.
- Han de ser **definites**: l'algorisme té que comportar-se de la mateixa forma sempre.

- Han de ser **finits**: l'algorisme ha de tenir un nombre finit de passos, té que acabar en un moment.

## Parts d'un algoritme

---



- **Entrada**: corresponen les dades que l'algorisme rep.
- **Procés**: equivalen les accions que es realitzen sobre les dades d'entrada.
- **Sortida**: el resultat de les accions sobre les dades d'entrada.

Ayer definíamos variables, asignamos entrada, procesos y al final una salida de datos

## Algoritmo de la vida diaria

---

- Aquest tipus d'algoritme és aquell que ens ajuda a resoldre problemes quotidians, i els fem sense donar-nos compte seguint una metodologia per resoldre'ls.

Ejemplo:

# Ex: algorisme per raspallar les dents

1. INICI
2. Anar al bany
3. Agafar el raspall de dents
4. Agafar la pasta de dents
5. Posar pasta de dents al raspall
6. Raspallar les dents el temps desitjat
7. Esbandir-se la boca
8. Guardar el raspall i la crema
9. FIN

**Important: podem donar solució a un problema de moltes formes diferents**

Se podría añadir más procesos y sería mejor.

Puede haber más de una solución posible.

## Algoritmo computacional

---

- Els algorismes computacionals permeten definir els processos per donar solució a problemàtiques mitjançant operacions lògiques a un computador.
- Aquests a diferència dels anteriors han de ser desenvolupats seguint una **metodologia definida per a la solució de problemes**. (enfocament de la solució, sintaxi...)

Al final los procesos van a ser bastante mecánicos

## Llenguatges algorítmics

---

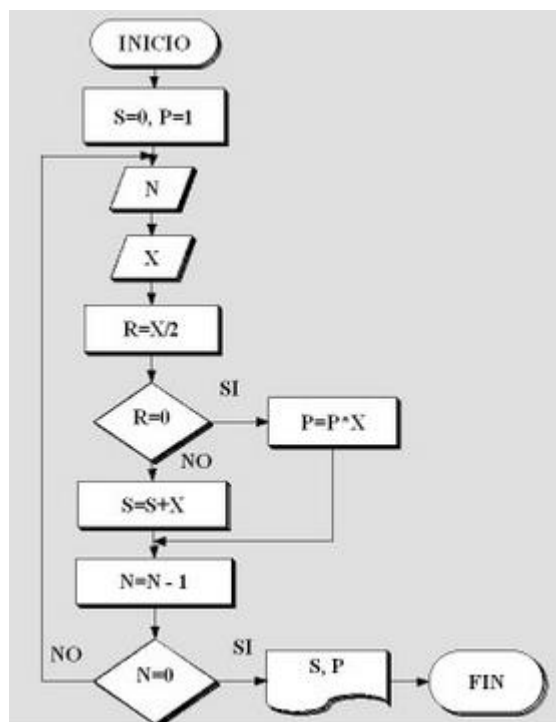
- Defineixen la forma com ens comunicarem per mitjà d'algoritmes, amb l'objectiu que sigui la màquina la que ens entengui.
- Puedes tener errores que has de interpretar
- Els algoritmes es poden resoldre mitjançant l'aplicació de diferents tècniques, en aquest cas podem ressaltar **4 llenguatges** que ens permetran descriure els



passos d'un algoritme de manera més detallada i estructurada (estos 3 més el de **programació**):

- **Llenguatge Natural**

- Aquest llenguatge ens permet descriure la seqüència lògica de passos d'una forma més natural, fem servir un vocabulari quotidià en descriure els passos de forma simple sense tecnicismes.

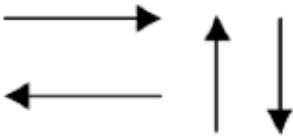
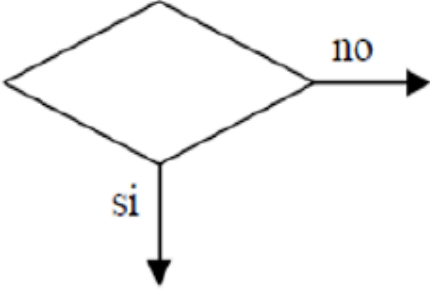
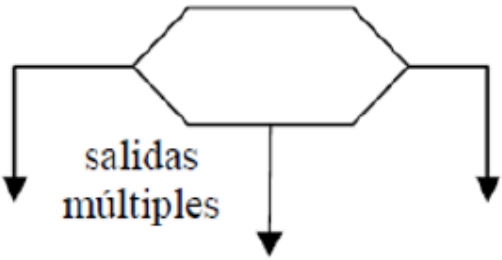



- Introducir los datos, etc

- **Diagrames de flux**

- Representen els algorismes per mitjà de símbols que faciliten l'enteniment de la solució o el procés plantejat.
- Diferentes simbolos con descripción:

SIMBOLO	DESCRIPCION
	Representa el inicio y el final de un diagrama de flujo
	Símbolo usado para representar cualquier tipo de entrada o lectura de datos y también puede ser usado para la asignación de valores.
	Utilizado para representar cualquier operación o proceso lógico, por lo regular usado para asignar valores, realizar operaciones matemáticas.
	Este símbolo se usa para representar las entradas o salidas del sistema, permite imprimir el resultado de un proceso

SIMBOLO	DESCRIPCION
	Las flechas representan el flujo del sistema, la dirección indica cual es el sentido al momento de ejecutar o seguir el diagrama.
	Representa una decisión u operación de comparación entre datos, en este símbolo define una condición y dependiendo del resultado se determina cuál de los caminos debe tomar el sistema.
	Representa decisiones múltiples o bifurcación, dependiendo del resultado de una comparación que depende de un dato de entrada el sistema tomará uno de los caminos propuestos.
	Este símbolo permite conectar o enlazar dos partes de un diagrama (en caso de que sea muy grande y no lo podamos ver completamente) se usa mediante un conector de entrada y otro de salida.

### • Pseudocódigo

- El pseudocodi compleix la mateixa funció però orientat a definir la solució d'un problema d'una forma més precisa i buscant definicions formals, generalment es fan servir per resoldre problemes mitjançant algorismes computacionals.
- El pseudocodi ha de complir amb les següents característiques:
  - Ser precís i definit.
  - Evitar diverses interpretacions (ambigüitat). Evitar siempre ambigüedad
  - Fer servir termes formals però familiars al sentit comú.
  - Eliminar instruccions innecessàries.

- Regles bàsiques:

# INICIO

## Declaració de **variables**;

## Expresions i operacions;

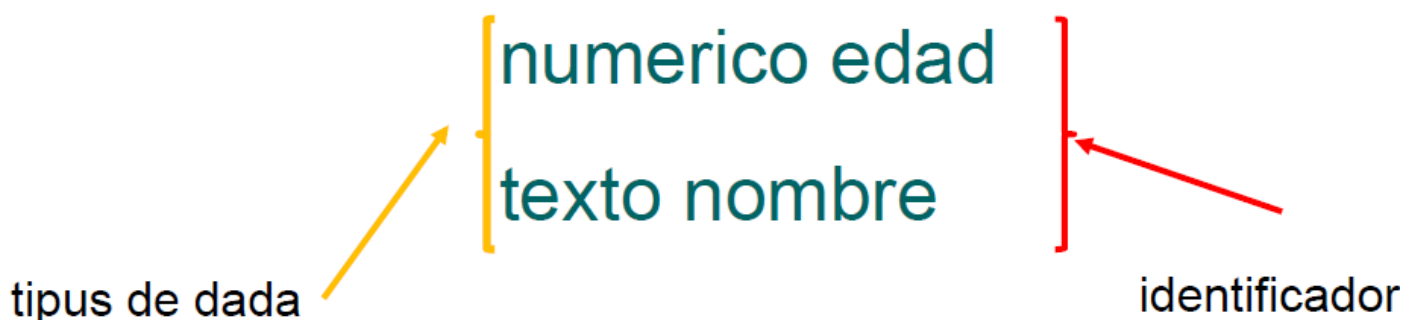
# FINAL

- Limitarem amb les paraules **INICIO** i **FINAL** l'algoritme.
- Al final de sentència farem servir “;” (punto y coma)
- Les **variables** les declararem abans de fer-les servir. Como el diagrama de flujo.
- Tenim que indicar el tipus de dada de cada variable.

Al final de sentència (en python no), se usa el punto y coma (como javascript etc)

## ¿Qué es una variable?

- Una variable és un contenidor que pot emmagatzemar informació i pot canviar en el temps, ja que el seu contingut pot variar.
  - ¡En Python una variable es una etiqueta a la que le asociamos a un tipo de dato!
- Per regla general, les variables les podem declarar amb un identificador i un tipus de dada que l'acompanya



A ver, el tipo de dada serían los números (Enteros, reales...) o letras.

El identificador es el nombre que le damos a ese tipo de datos.

## ¿Cómo nombramos a una variable?

---

- El primer carácter ha de ser alfabètic(a...z, A..Z) o \$.
- Després poden anar caràcters alfanumèrics.
- Els identificadors no poden ser paraules reservades del llenguatge.
- És molt recomanable fer servir la regla **camelCase**.
  - Tiene que empezar con una letra y ha de ser minúscula.
  - Todas las siguientes palabras irán con la primera letra en mayúscula.
- No poden haver-hi espais ni signes de puntuació.
  - el guión bajo si se acepta como signo de puntuación
- La mayoría de llenguatges són **case sensitive**.
  - El método en que escribes los nombres (identificadores) de las variables, atributos, métodos de clase y clases.
  - Ej.: camelCase

## ¿Que es un tipo de dato?

---

- Les variables corresponen a contenidors de memoria on enmagatzemem valors.
- Aquests valors estàn associats a un tipus de dada específica:
  - **Numéric** (sencer, decimal).
  - **Text** (un carácter o una cadena de caràcters).
  - **Lògic** (veritato mentida / SI o NO)

**Ejemplos de programa**En Python -> Las arrays no existen (matrices), pero si diccionarios etc

---

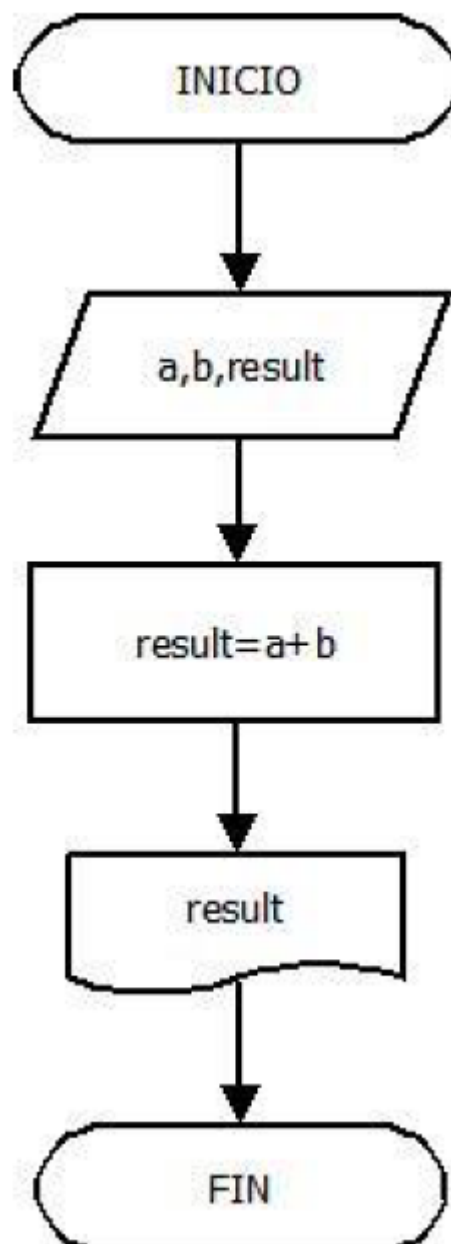
- Realitzarem amb llenguatge natural, diagrama de flux i pseudocodi, un algoritme que faci la suma de 2 nombres i mostri el resultat.

- Lenguaje natural

1. Introduzco el primer valor
2. Introduzco el segundo valor
3. Realizo la operación suma
4. Muestro el resultado

- Diagrama de flujo

- inicio, declaración, operaciones e imprimir resultado + fin



- Pseudocódigo

INICIO

numerico a, b, result;

result = a + b;

imprimir result;

FINAL

- Toda linea acaba en punto y coma.

## Prueba de escritorio EXAMEN\*\*\*\*\*

---

- Quan es realitza un algoritme, l'ideal és verificar el procés per assegurar-nos que compleix allò que s'esperava, per això s'aplica una tècnica anomenada “Prova d'Esriptori”.
- La prova d'escriptori **consisteix en un seguiment pas a pas de l'execució de l'algorisme, per validar aquest procés, determinar quins són els valors pas a**

pas i verificar així el funcionament desitjat.

INICIO

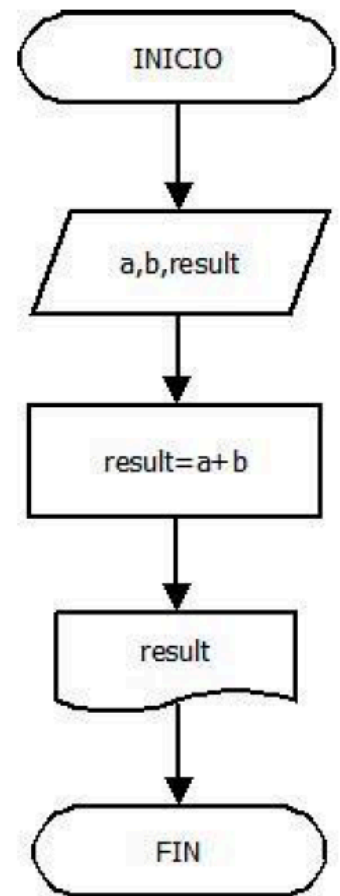
numerico a, b, result;

result = a + b;

imprimir result;

FINAL

a	b	result
2	2	4
1	4	5



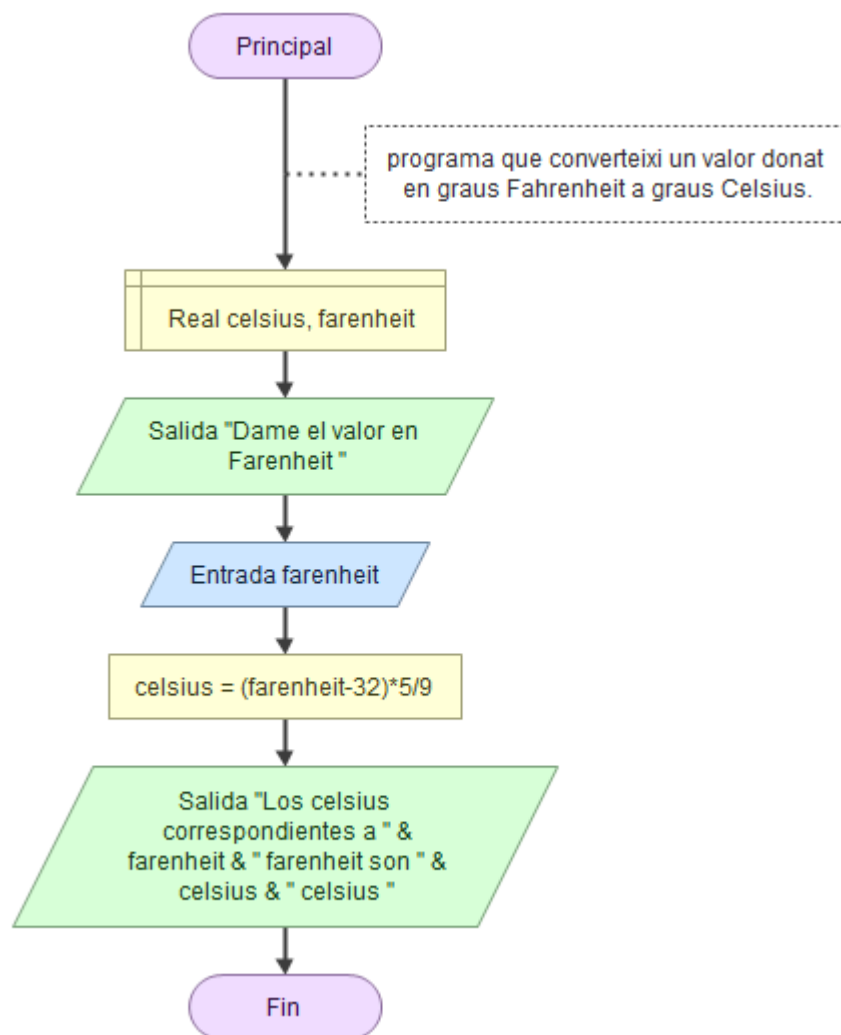
\*\* \*\*Ej. Prueba escritorio ----->

Ejercicios:

Realitzar els següents algorismes:

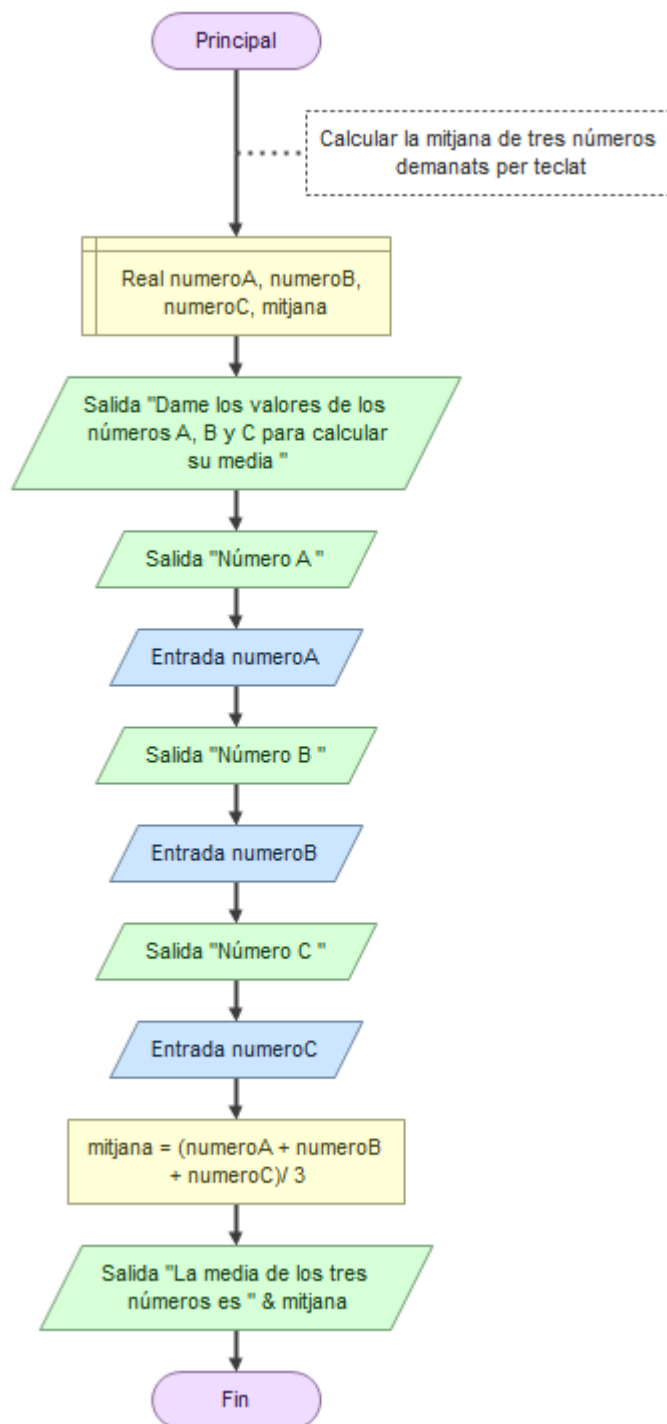
- Escriure un programa que converteixi un valor donat en graus Fahrenheit a graus Celsius. Recordeu que la fórmula per a la conversió és:

$$C = (F-32)*5/9$$

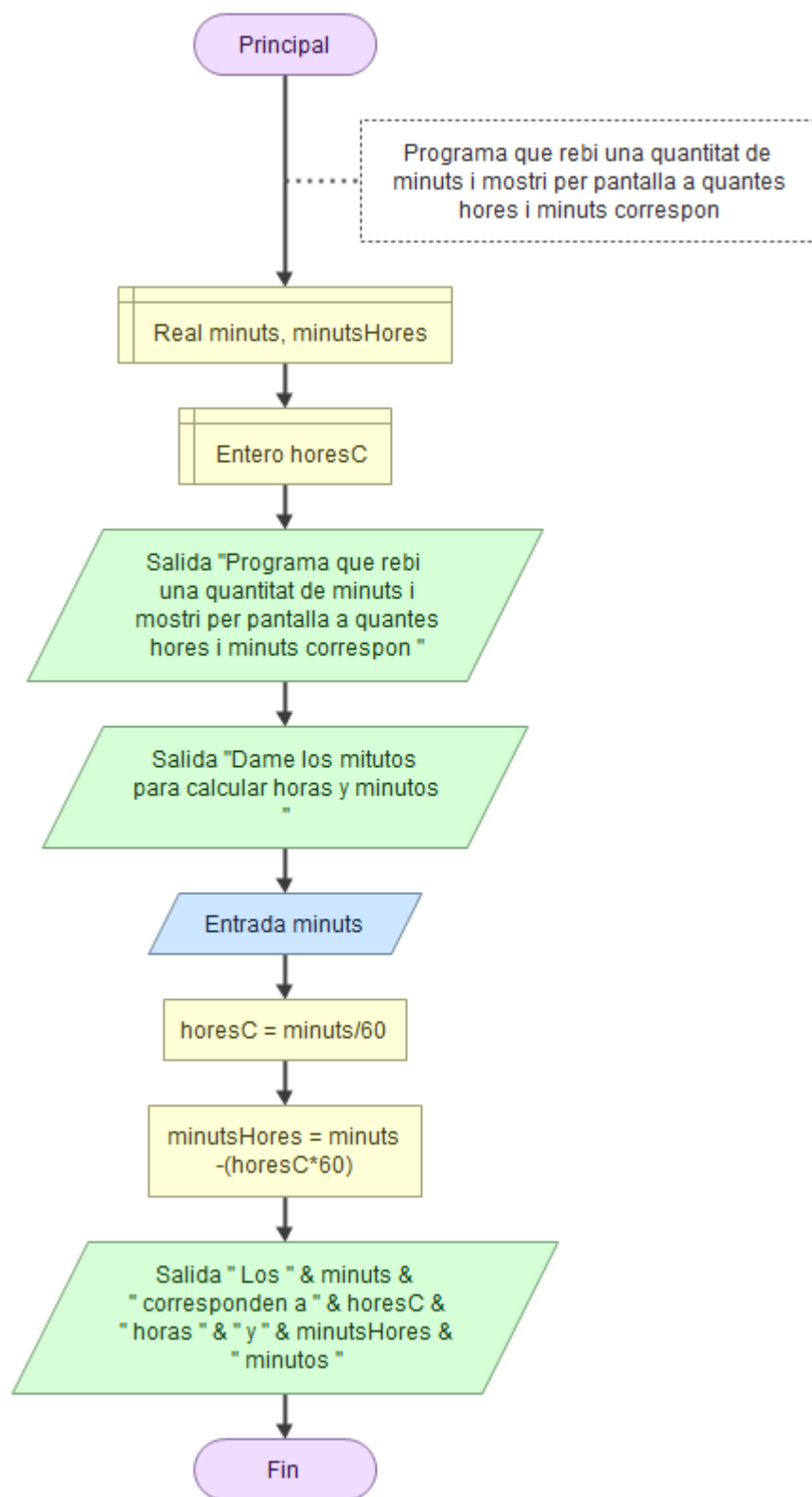


- Calcular la mitjana de tres números demanats per teclat.

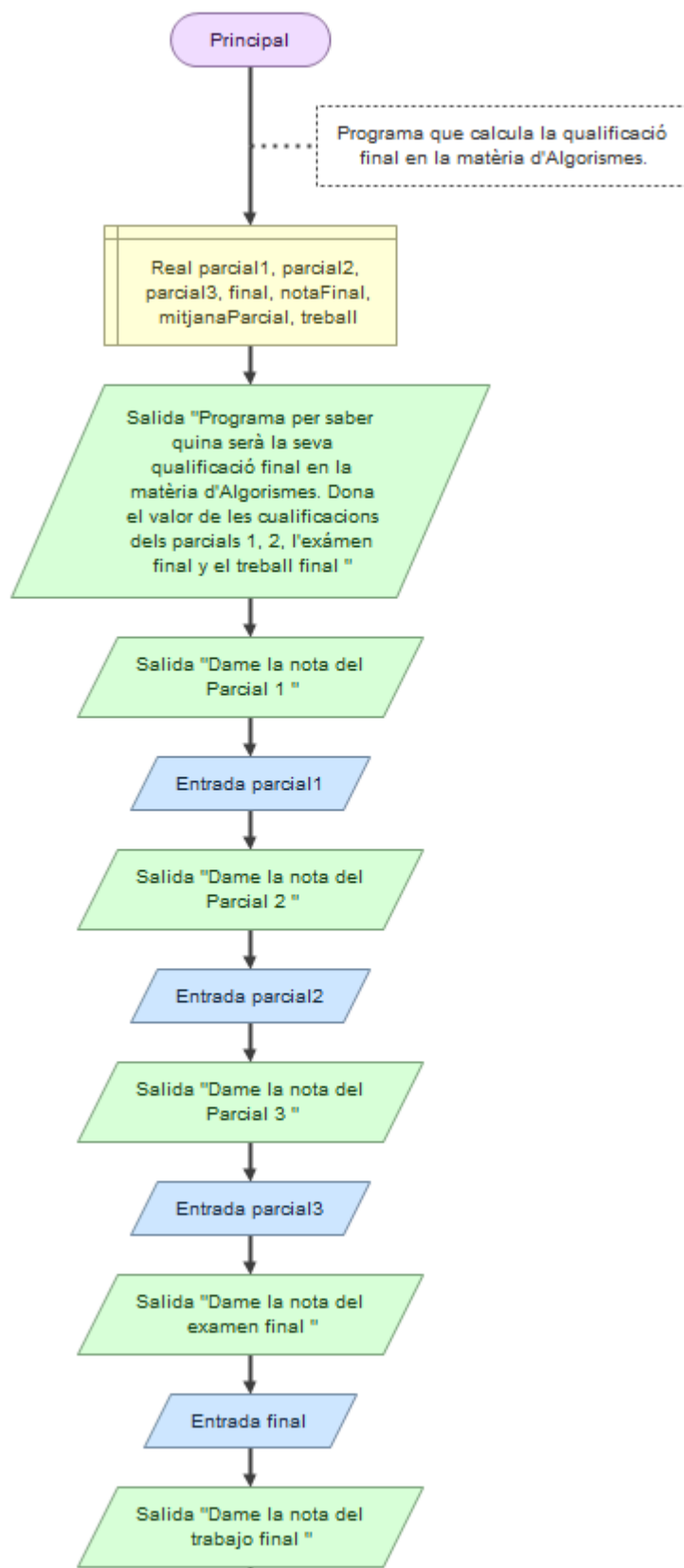


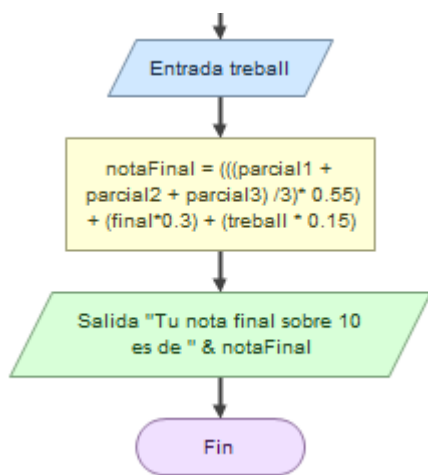


- Realitza un programa que rebi una quantitat de minuts i mostri per pantalla a quantes hores i minuts correspon. Per exemple: 1000 minuts són 16 hores i 40 minuts.

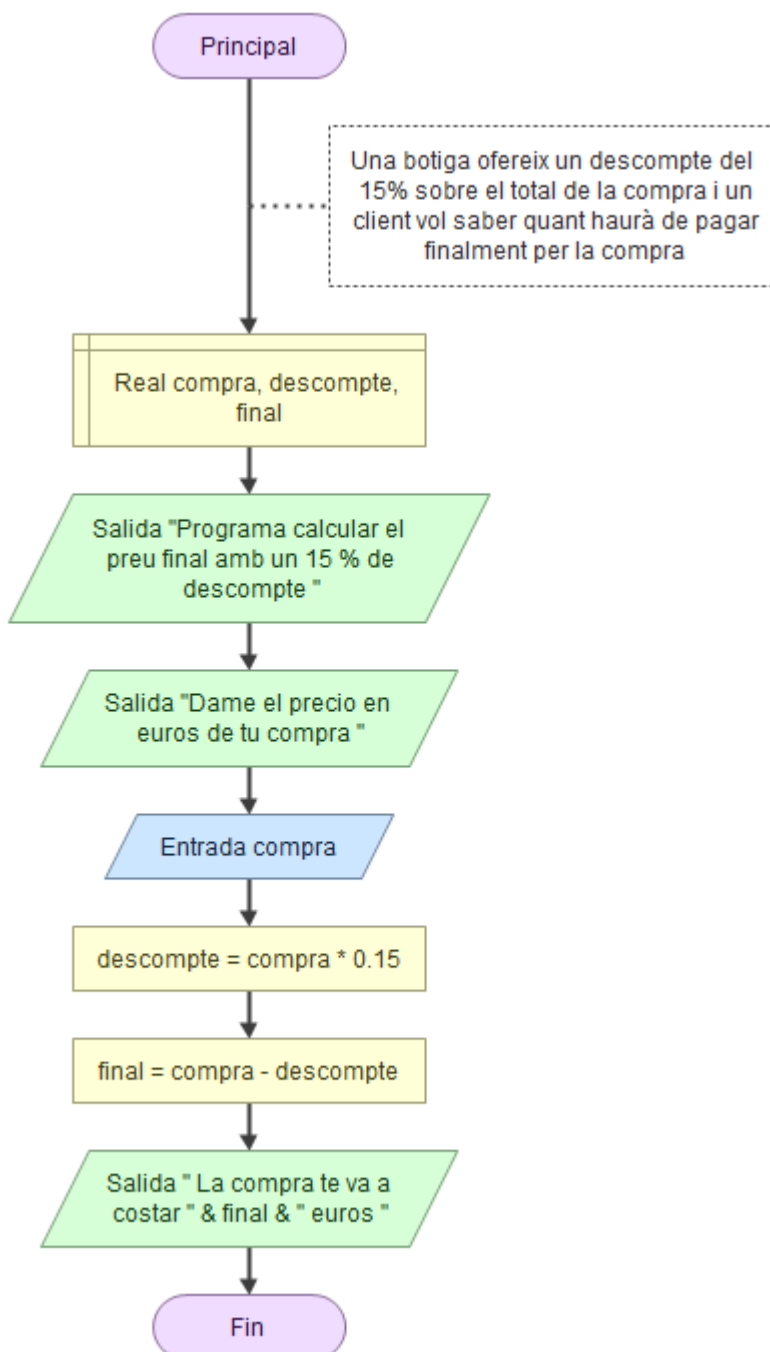


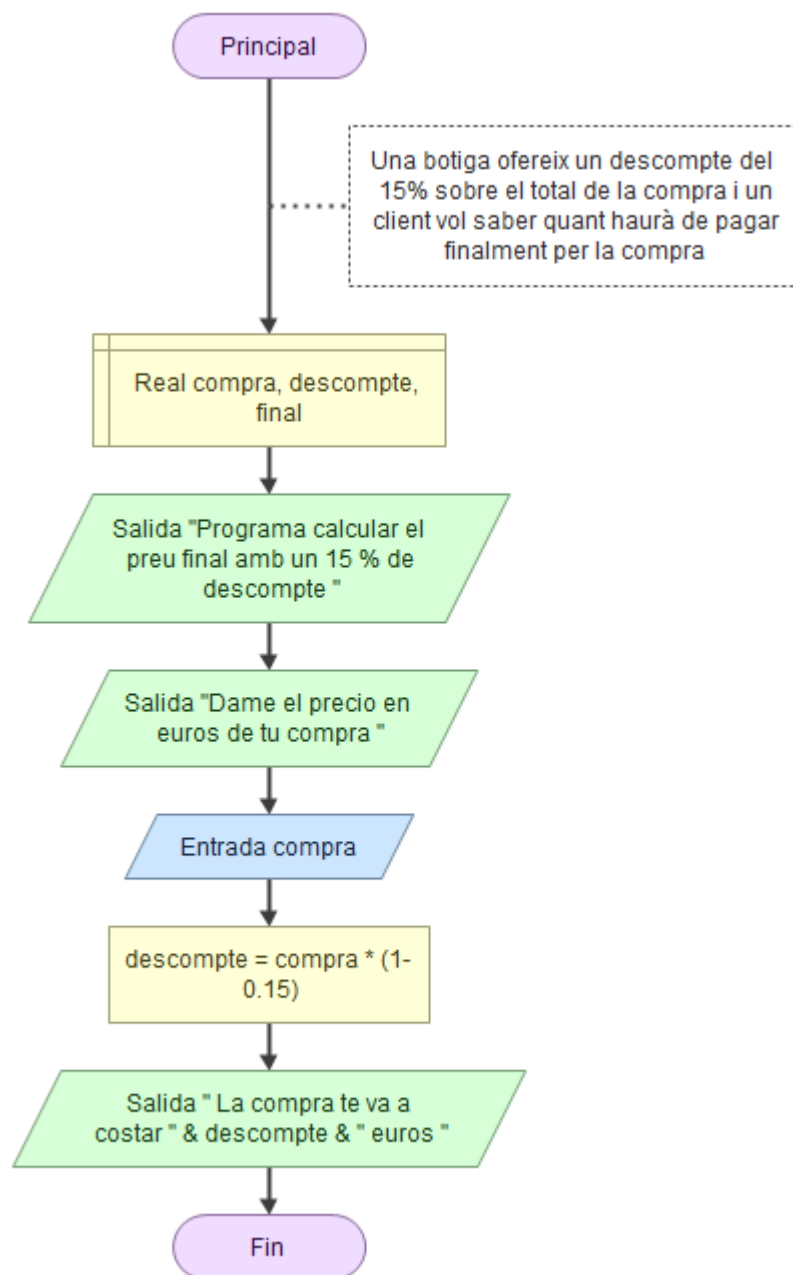
- Un alumne vol saber quina serà la seva qualificació final en la matèria d'Algorismes. Aquesta qualificació es compon dels percentatges següents:
  - 55% de la mitjana de les tres qualificacions parcials.
  - 30% de la qualificació de l'examen final.
  - 15% de la qualificació d'un treball final.



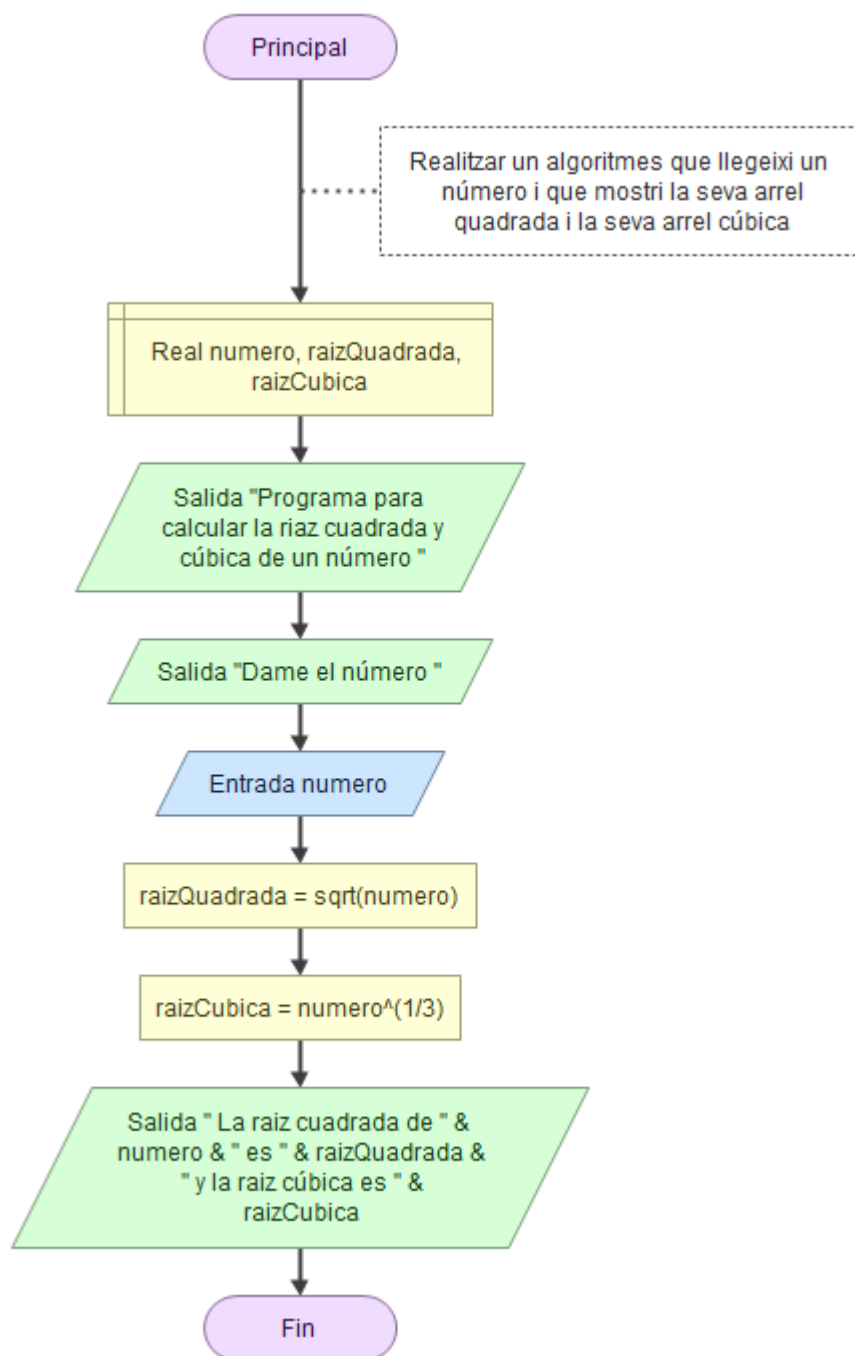


- Una botiga ofereix un descompte del 15% sobre el total de la compra i un client vol saber quant haurà de pagar finalment per la compra.





- Realitzar un algoritme que llegeixi un número i que mostri la seva arrel quadrada i la seva arrel cúbica.



Estructuras condicionales:

Hasta ahora, en estructura secuencial una orden va detrás de otra. Ahora queremos estructuras de control, donde se pueden tomar decisiones.

Condicionales simples:

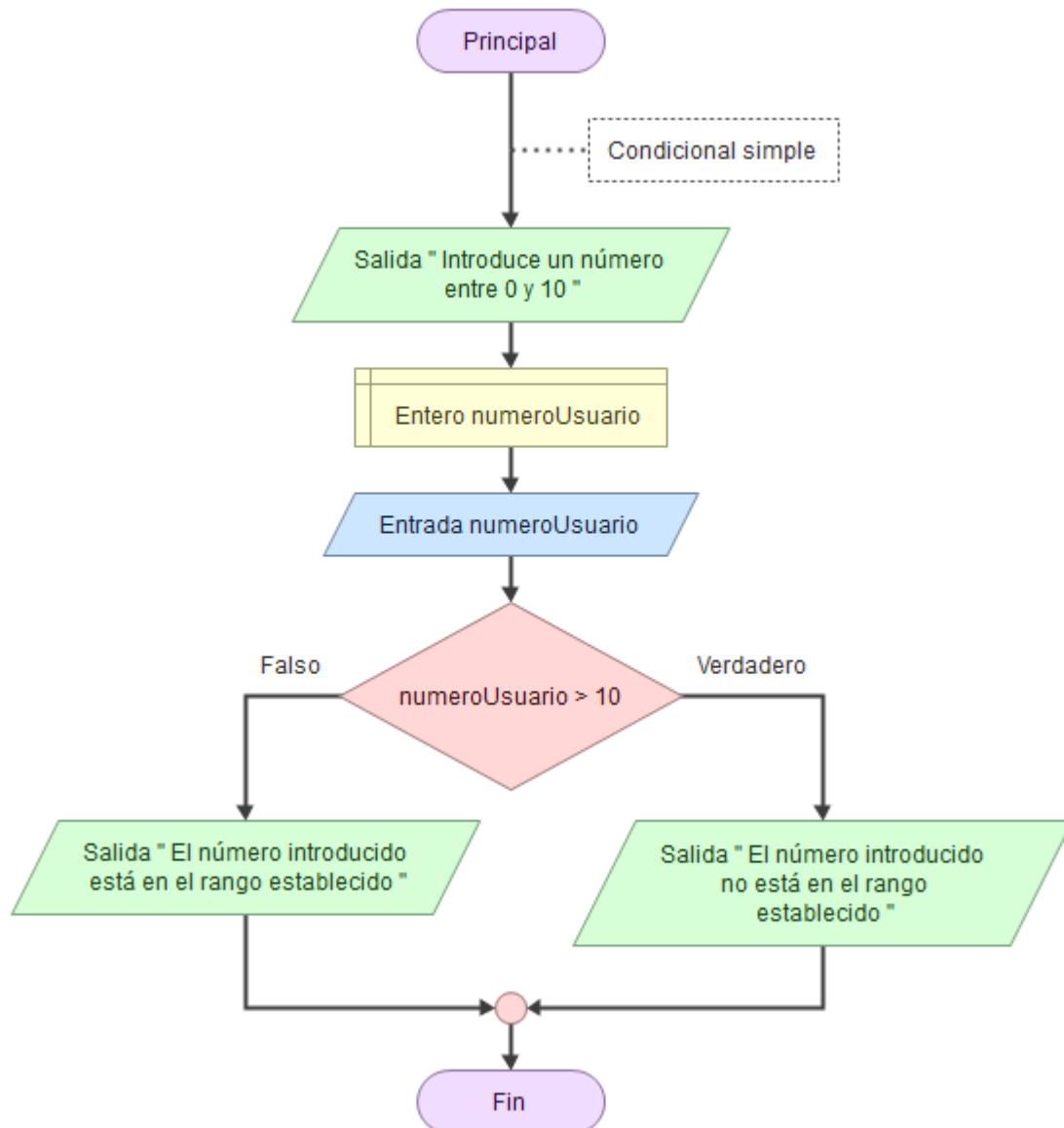
\*\* \*\*En lenguaje en C -> estructura "sweetcase", en python no existe

\*\* \*\*FlowGorithm no lo tiene tampoco

Se usa la herramienta de control Si



**\*\* \*\***El resultado siempre será true o false, es un resultado booleano



podemos poner  $\leq$  a un número

Para comparaciones se usan los siguientes símbolos:

$==$  igual a

$\leq$  menor o igual a

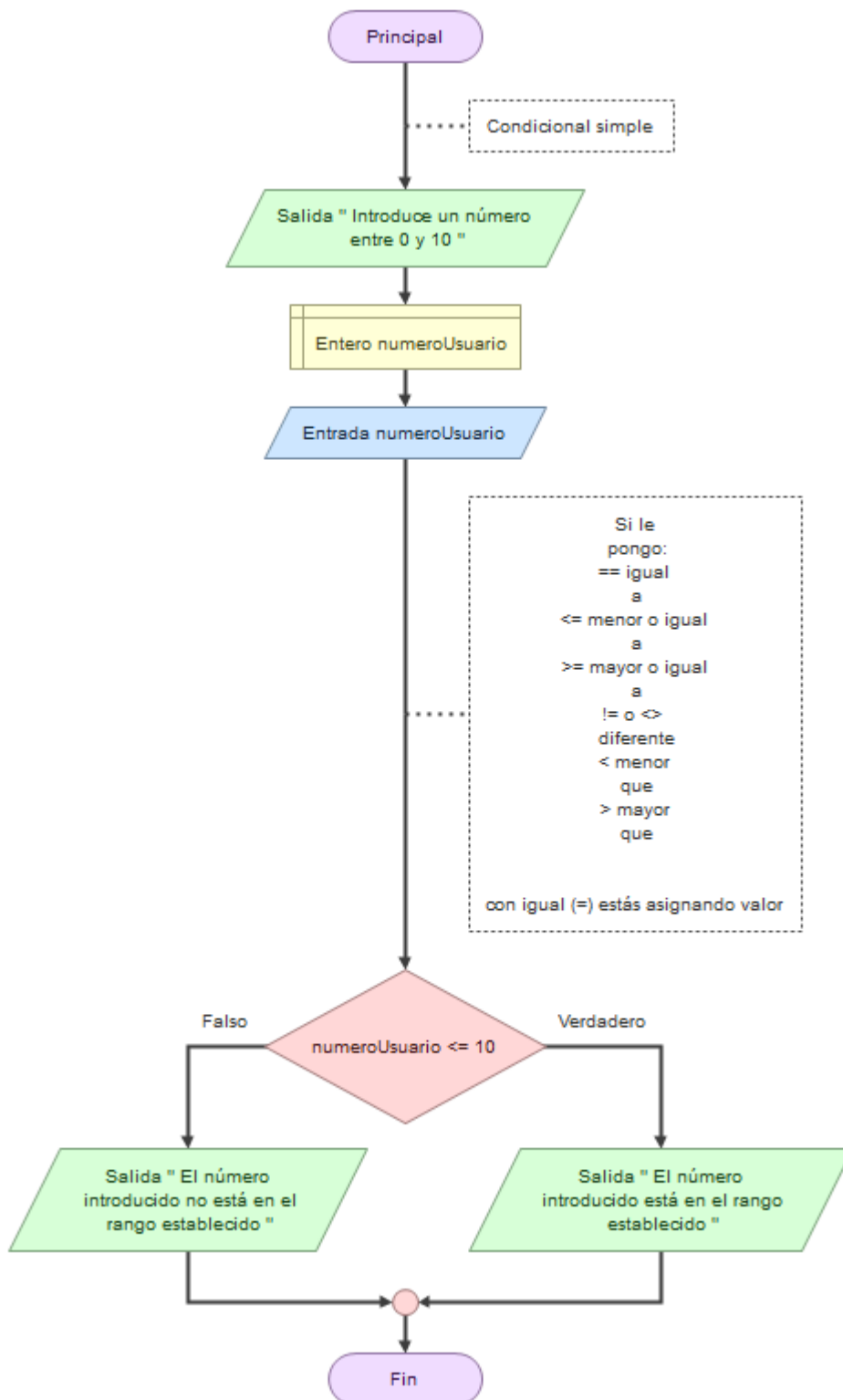
$\geq$  mayor o igual a

$!=$  o  $<>$  diferente

< menor que

mayor que

con igual (=) estás asignando valor, no es igual a!

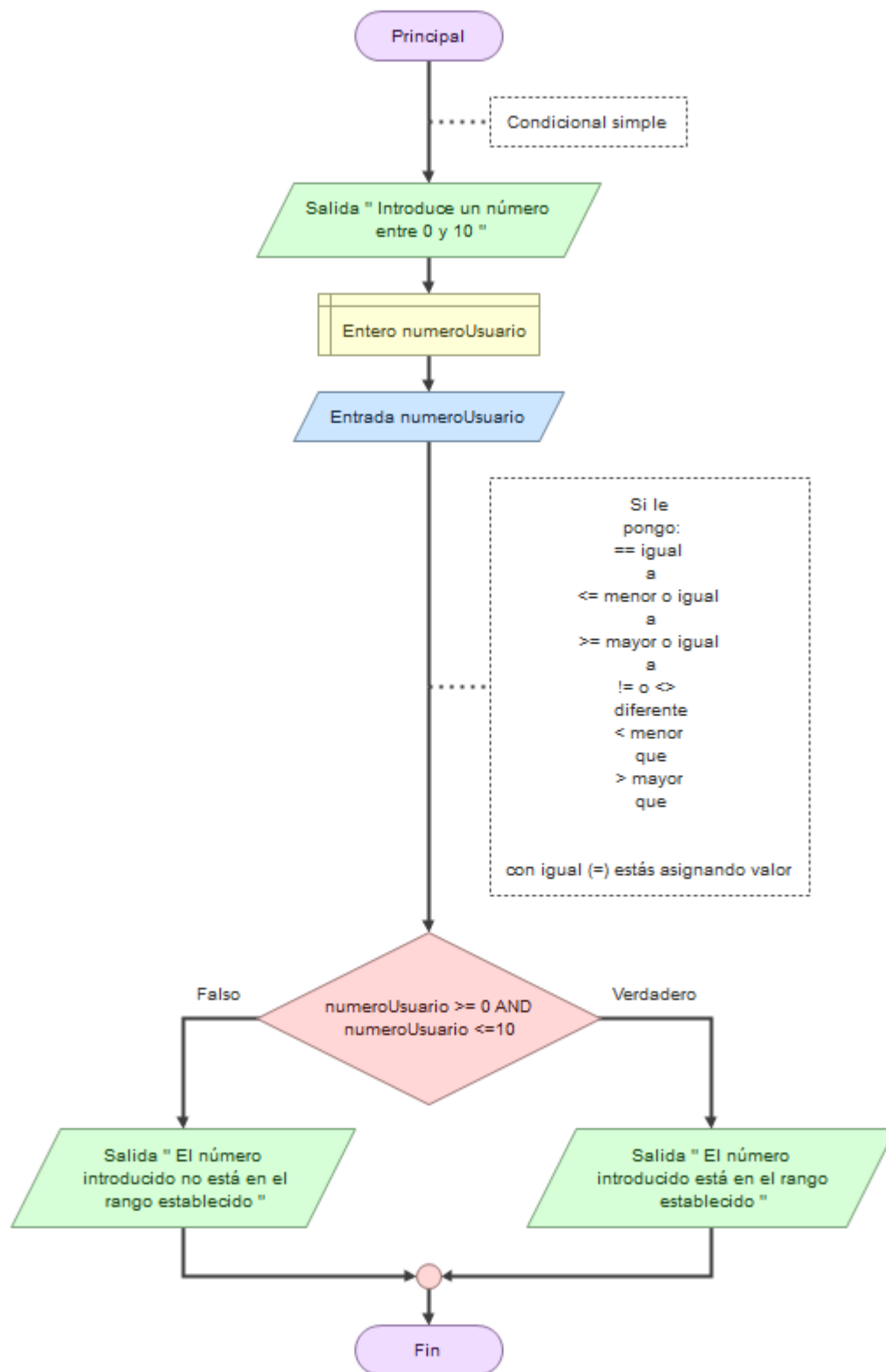


Cumplir dos condiciones o más:

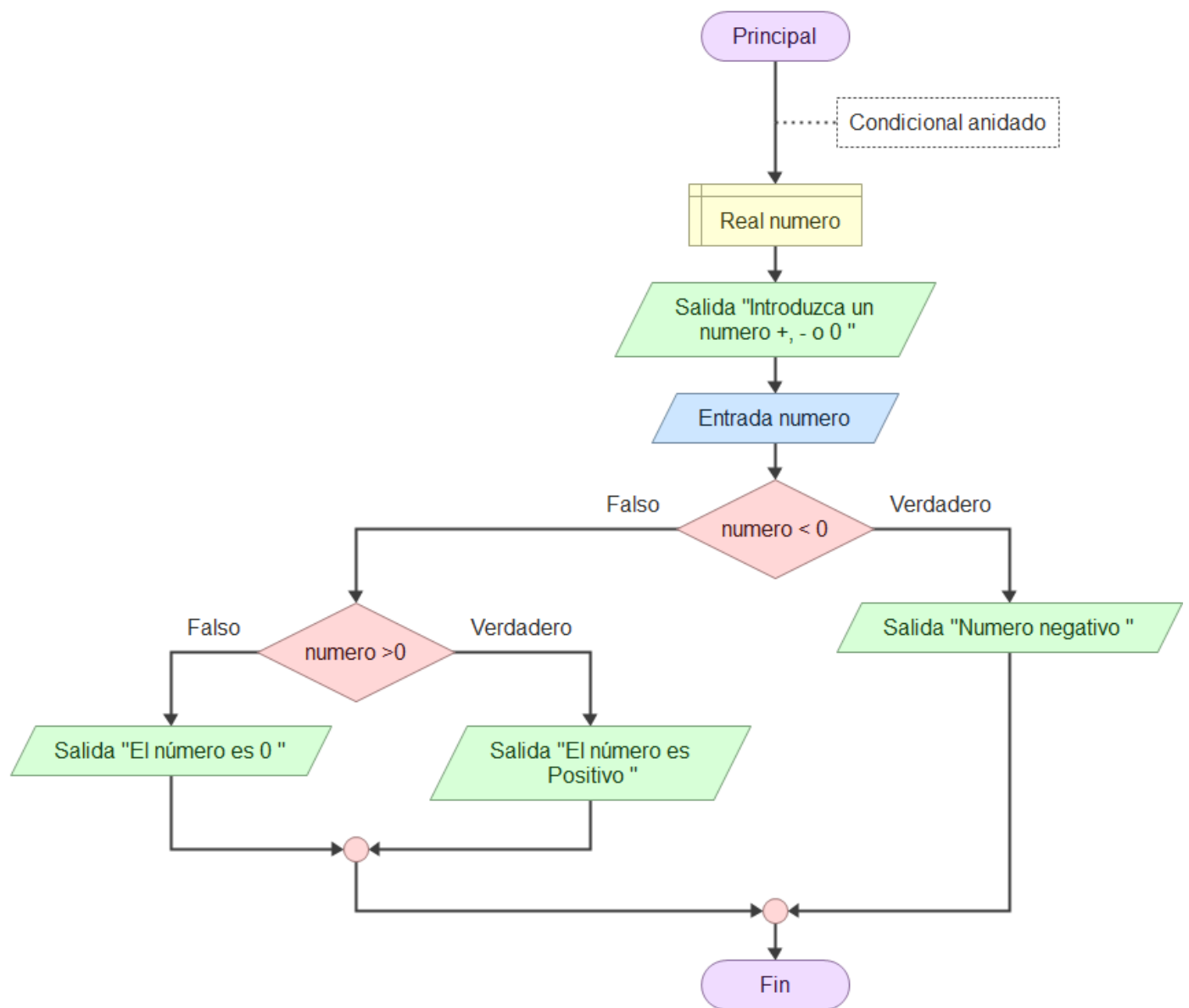
AND && : Se tiene que cumplir las dos condiciones

OR || (alt gr + 1) : Se tiene que cumplir una u otra condición





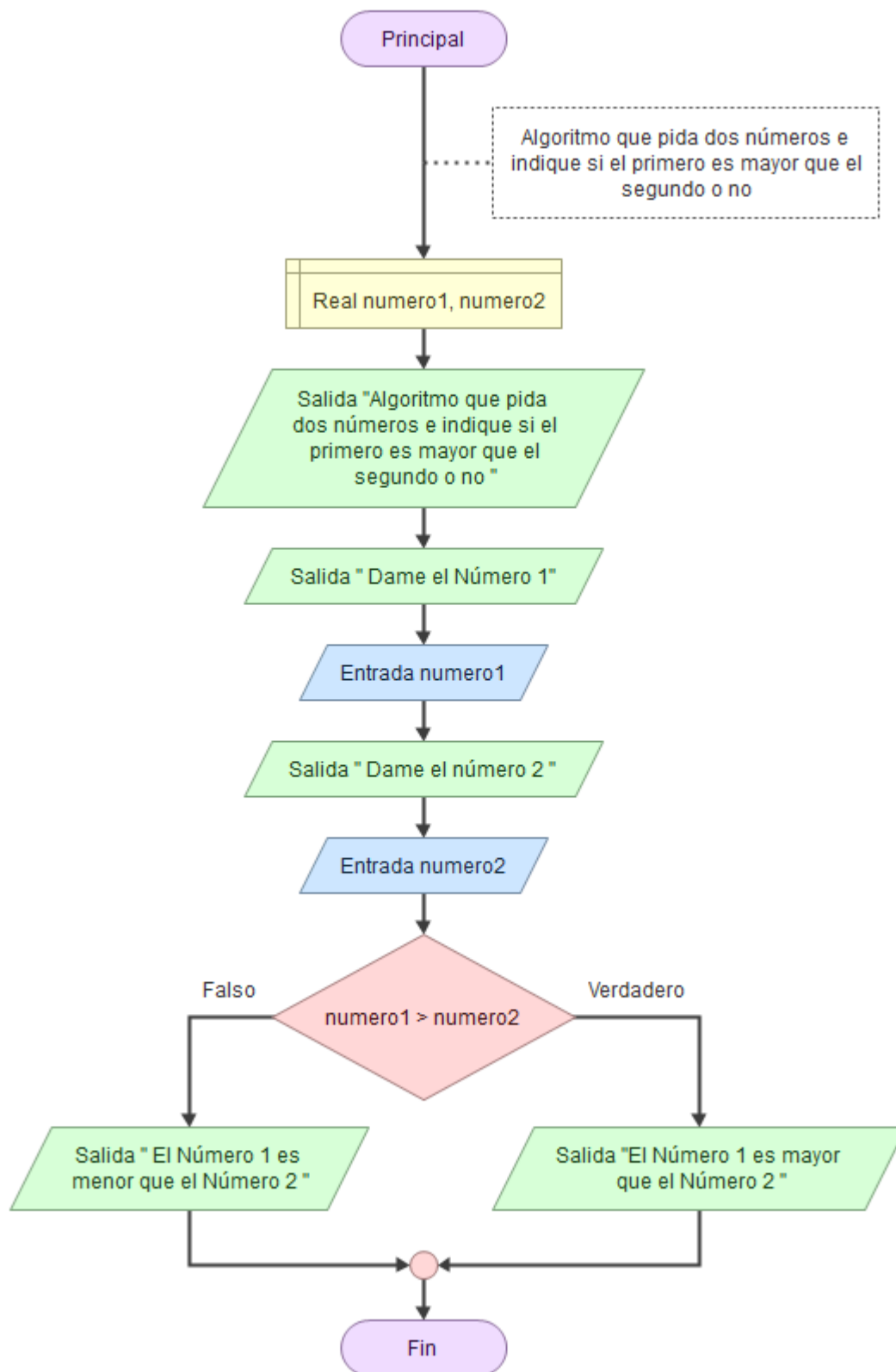
Condicional anidado:



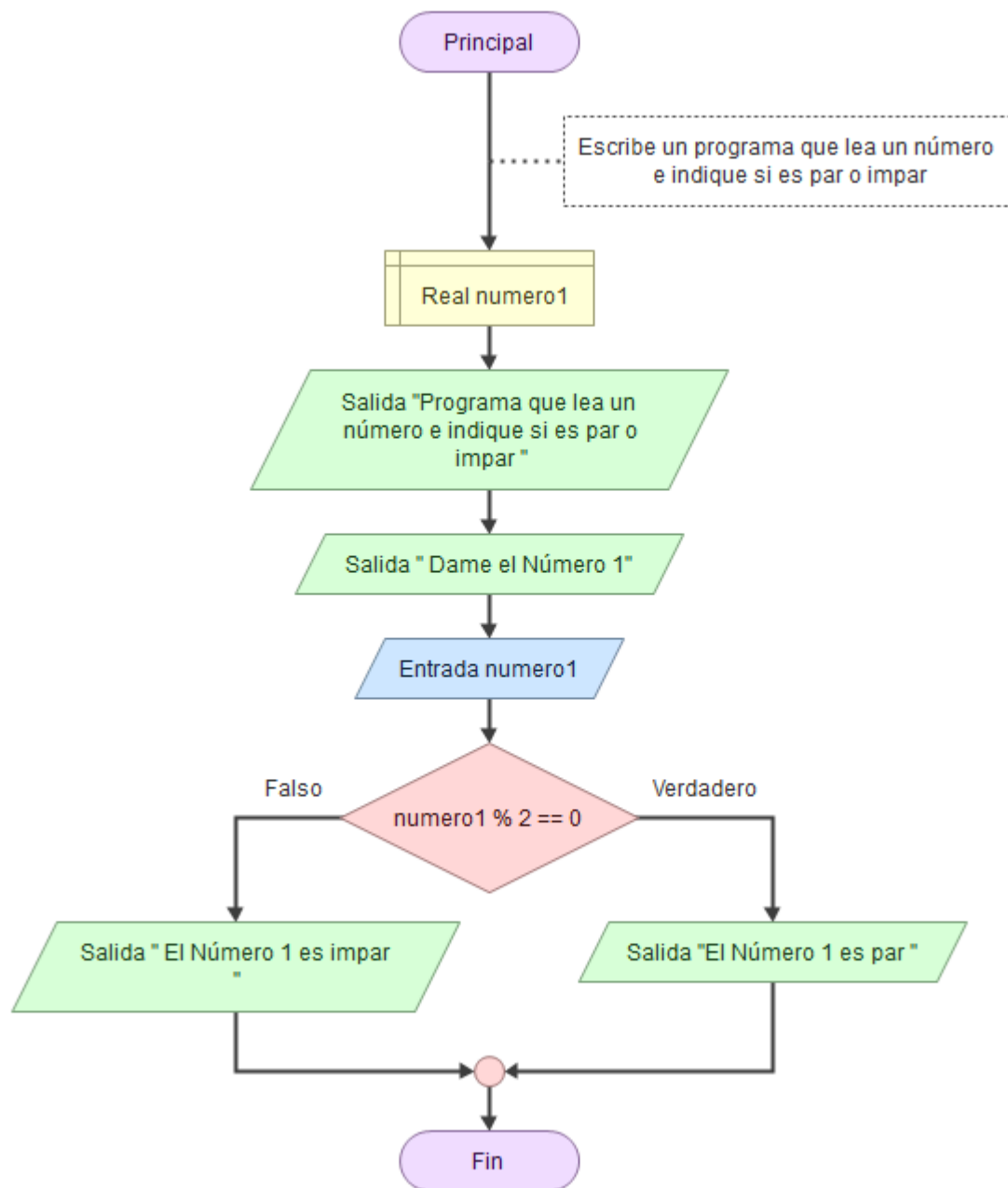
Esto sirve mucho para entrar en una cuenta, un control de usuario

Ejercicio:

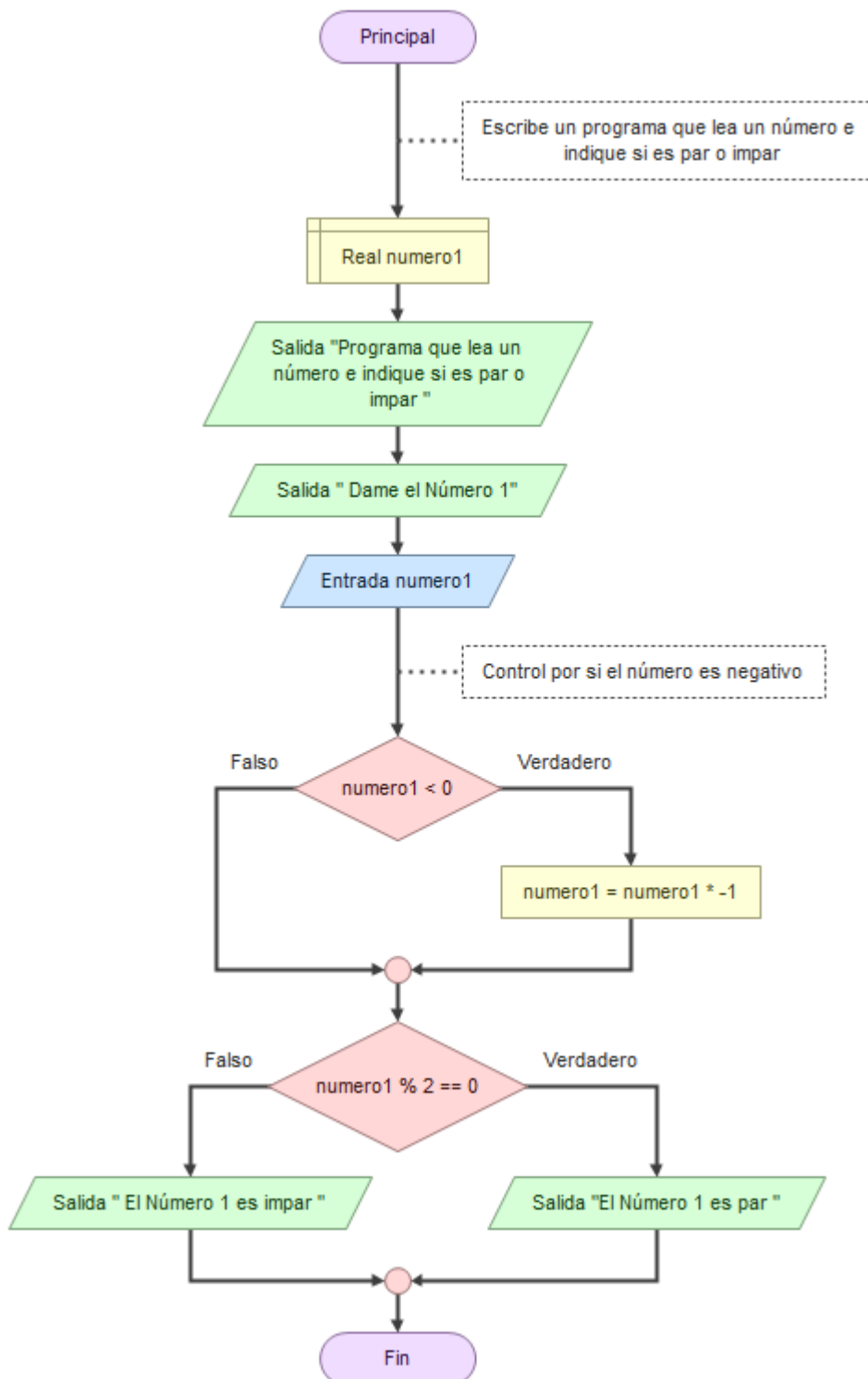
Algoritmo que pida dos números e indique si el primero es mayor que el segundo o no:



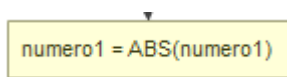
Escribe un programa que lea un número e indique si es par o impar:



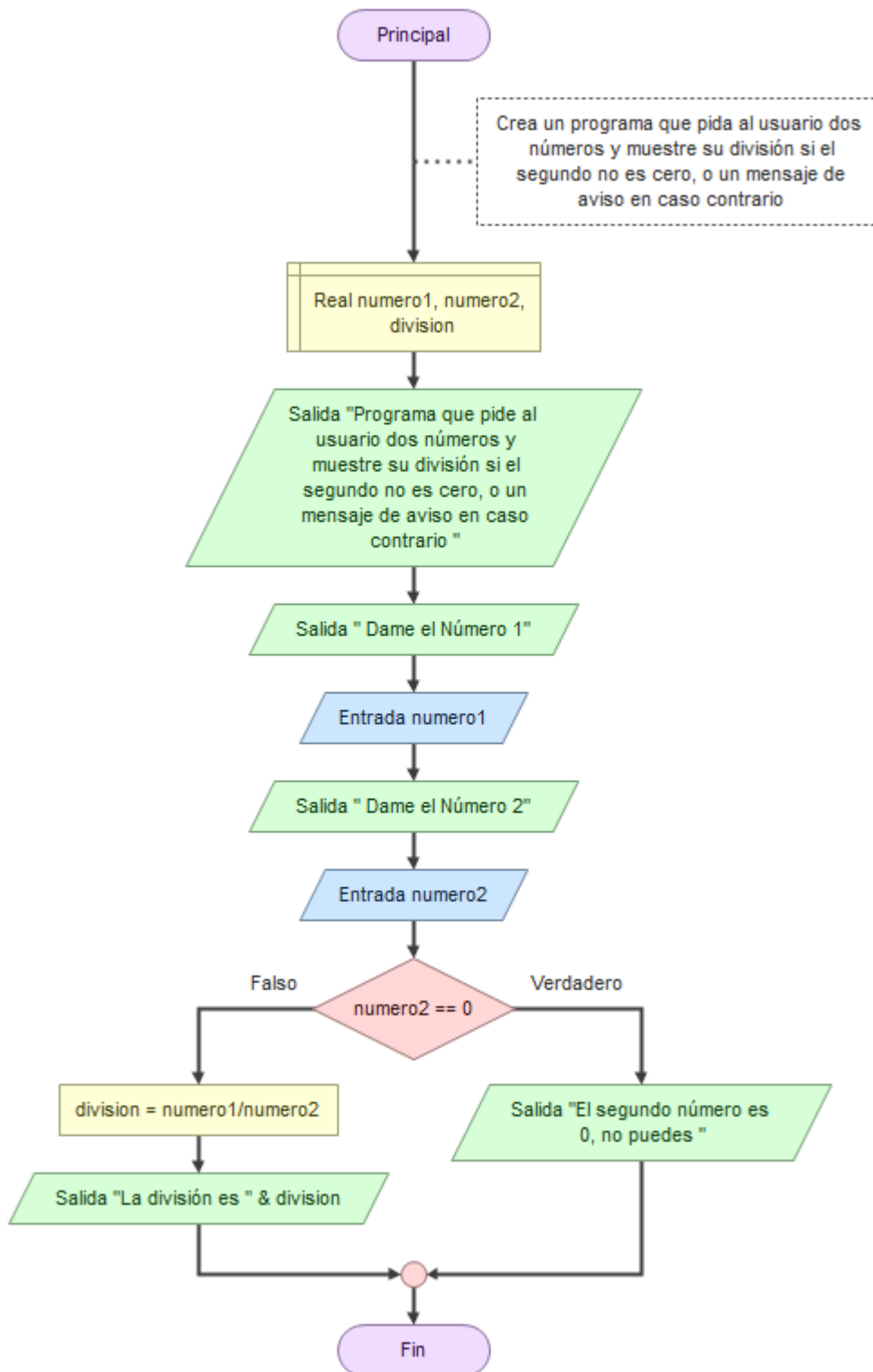
Si es negativo, tenemos que dar otro condicional o da error.



para hacer número absoluto existe la función `ABS(número)` o `abs ()`



Crea un programa que pida al usuario dos números y muestre su división si el segundo no es cero, o un mensaje de aviso en caso contrario.



\*\*