

# Python (1)



VICEPRESIDENCIA  
PRIMERA DEL GOBIERNO  
MINISTERIO  
DE ASUNTOS ECONÓMICOS  
Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO  
DE DIGITALIZACIÓN  
E INTELIGENCIA ARTIFICIAL

red.es

Centro de  
Referencia Nacional  
en Comercio Electrónico  
y Marketing

CRN  
Digital

Centro de  
Garantía Juvenil



UNIÓN EUROPEA

 Barrabés

 The Valley

*"El FSE invierte en tu futuro"*

Fondo Social Europeo

# Introducción



# Definiciones

## Programación

Proceso utilizado para idear las **acciones** necesarias para resolver un problema mediante el uso de un **ordenador**.

## Algoritmo

Conjunto ordenado y finito de **acciones** que permiten resolver un problema.

## Entorno

Conjunto de **elementos necesarios** para llevar a cabo un **algoritmo**.

## Estado del entorno

Descripción de los elementos del **entorno** en un momento dado. Los algoritmos **modifican** progresivamente el estado de su entorno desde su **estado inicial** hasta alcanzar el **estado final** deseado.

# Definiciones

## Acción

Suceso **finito en el tiempo** que tiene un **efecto definido**. Una acción puede actuar sobre el **entorno y modificarlo**.

## Proceso

Ejecución de una o varias **acciones**. El encargado de llevar a cabo el proceso es el **procesador**.

## Procesador

Entidad capaz de **comprender** y ejecutar eficazmente un **algoritmo**. El procesador es el destinatario del **algoritmo**.

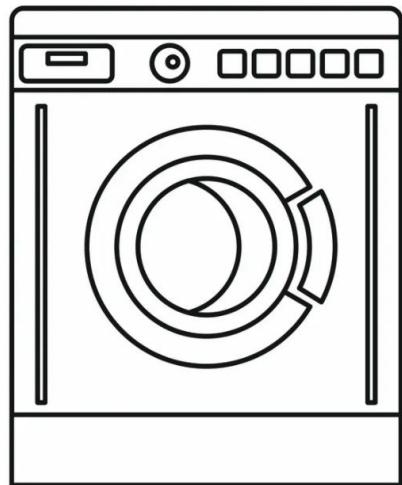
## Ordenador

**Máquina** que tiene la capacidad de **resolver problemas** según unas instrucciones dadas. Un ordenador (o computador) se compone de **procesador, memoria, y dispositivos de entrada y salida** que le permiten comunicarse con el exterior.

# Definiciones

## Ejemplo

Consideremos el siguiente conjunto de acciones que representa el **algoritmo** llevado a cabo por una lavadora para lavar la ropa:



1. Cargar agua.
2. Cargar jabón.
3. Calentar el agua a 30 °C.
4. Girar el bombo lentamente durante 20 minutos (lavado).
5. Expulsar el agua.
6. Cargar agua.
7. Girar el bombo lentamente durante 5 minutos (aclarado).
8. Expulsar el agua.
9. Girar el bombo rápidamente durante 1 minuto (centrifugado).

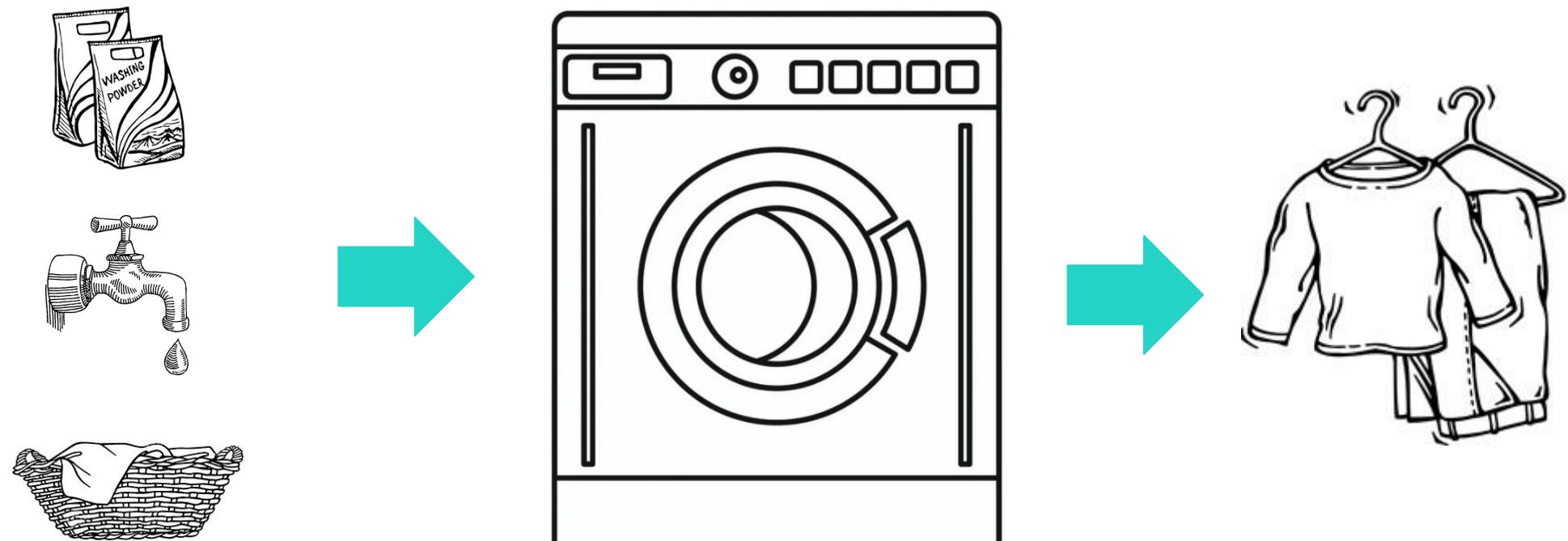
En este ejemplo el **problema a resolver** sería lavar la ropa, el **procesador** sería la propia lavadora, y el **entorno** serían todos los elementos necesarios para poder llevar a cabo el algoritmo (ropa, agua, jabón, bombo, etc.).

Durante el **procesamiento del algoritmo** se irán ejecutando de forma secuencial las distintas **acciones** (ej.: cargar agua, girar el bombo...), y consecuentemente el **estado del entorno** irá cambiando desde su estado inicial hasta el estado final deseado, que en este ejemplo sería tener la ropa limpia.

# Definiciones

## Ejemplo

Para lograr el **estado final** deseado es necesario que el **estado inicial** cumpla una serie de **condiciones**, como por ejemplo que la ropa esté dentro de la lavadora, que la lavadora esté conectada a la corriente eléctrica y a la toma de agua, que haya jabón en el cajón, etc.



# Representación de algoritmos

A grandes rasgos los algoritmos pueden representarse de dos modos:

## Representación gráfica

Los algoritmos pueden representarse **gráficamente** mediante los llamados **diagramas de flujo**.

Los diagramas de flujo son útiles para describir **algoritmos sencillos**, **algoritmos simplificados**, o **partes específicas** de un algoritmo, pero generalmente no resultan adecuados para describir **algoritmos complejos en su totalidad**.

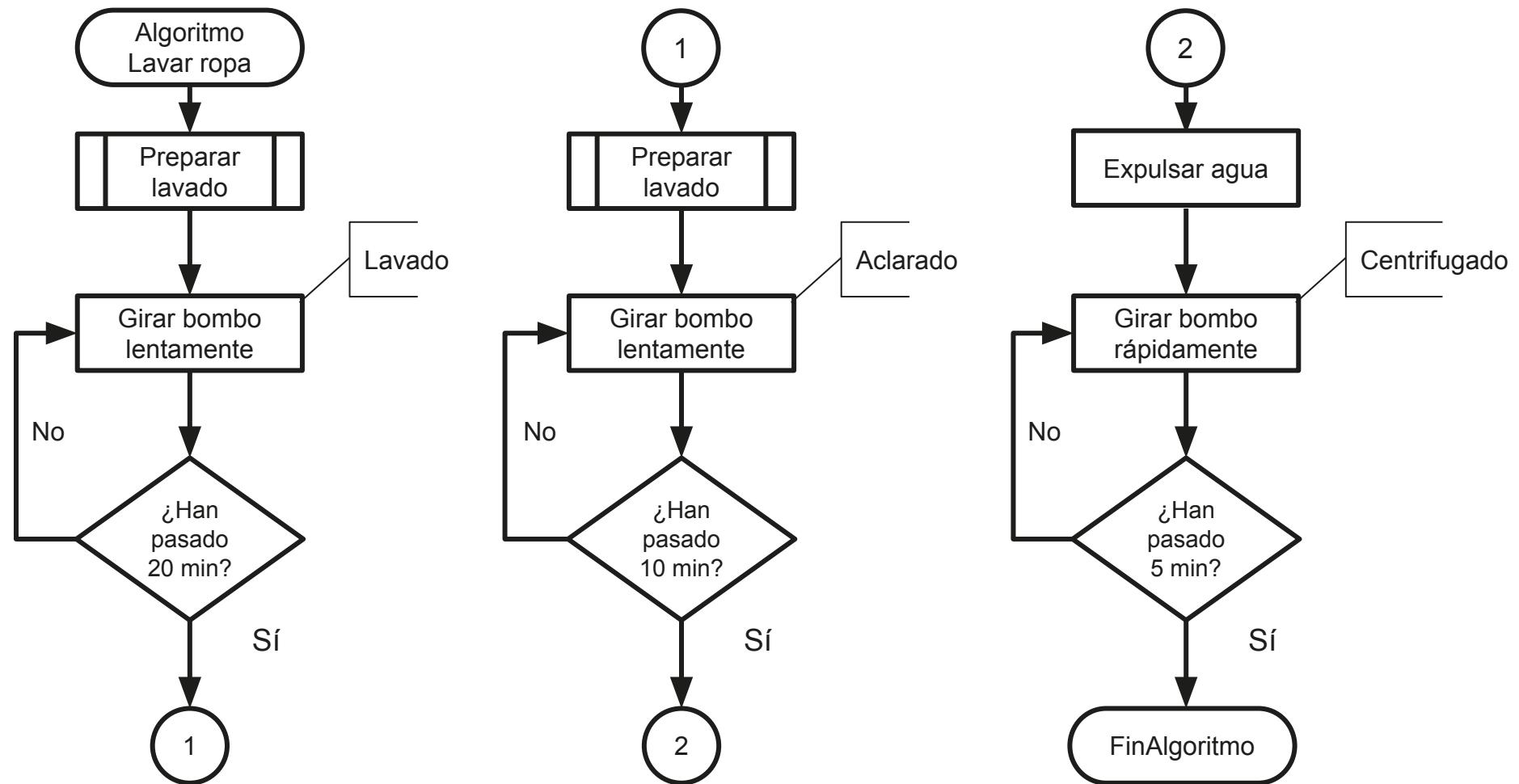
## Representación textual

La representación gráfica no es adecuada para expresar **algoritmos complejos** o que requieren un **elevado nivel de formalidad**. En estos casos deberemos emplear el **lenguaje algorítmico**.

El lenguaje algorítmico, también denominado **pseudocódigo** permite representar **textualmente** un algoritmo mediante el uso de un lenguaje que, aunque utiliza **estructuras** similares a las empleadas por los **lenguajes de programación**, es independiente de los mismos.

No existe una **sintaxis normalizada** para el lenguaje algorítmico y se puede utilizar con distintos niveles de formalidad.

# Representación gráfica de algoritmos



Representación gráfica del algoritmo “Lavar ropa”

# Representación textual de algoritmos

Para representar **algoritmos complejos** o con un **elevado nivel de formalidad**, es necesario utilizar una **representación textual**, para lo cual se usa el **lenguaje algorítmico**.

## Lenguaje natural

Es el lenguaje que utilizamos habitualmente para comunicarnos (ej.: español, inglés). Debido a su **complejidad y ambigüedad**, el lenguaje natural generalmente no resulta adecuado para describir un algoritmo.

## Lenguaje algorítmico

Los algoritmos deben ser expresados de forma **clara y precisa**. Para ello utilizaremos un lenguaje más específico que el lenguaje natural denominado **lenguaje algorítmico, pseudolenguaje o pseudocódigo**. El lenguaje algorítmico no puede ser entendido directamente por un ordenador, pero se puede **traducir** fácilmente a cualquier **lenguaje de programación**.

## Lenguaje de programación

Para que un ordenador pueda procesar nuestros algoritmos es necesario que los entienda. Para ello debemos transcribir nuestros algoritmos en un lenguaje capaz de ser comprendido por un ordenador. Estos lenguajes se denominan **lenguajes de programación**.

# Representación textual de algoritmos

## Algoritmo LavarRopa

```
    prepararLavado()
    // Lavado
    Mientras minutos_transcurridos ≤ 20 Hacer
        girarBomboLentamente
    Fin Mientras
    prepararLavado()
    // Aclarado
    Mientras minutos_transcurridos ≤ 10 Hacer
        girarBomboLentamente
    Fin Mientras
    expulsarAgua
    // Centrifugado
    Mientras minutos_transcurridos ≤ 5 Hacer
        girarBomboRapidamente
    Fin Mientras
FinAlgoritmo
```

*Representación en lenguaje algorítmico o pseudocódigo del algoritmo “Lavar ropa”*

# Propiedades de un algoritmo

Un algoritmo **correctamente diseñado** debe cumplir las siguientes **propiedades**:

## Corrección

El algoritmo debe **resolver correctamente el problema** para el cual fue diseñado.

## Inteligibilidad

El algoritmo debe ser **claro y fácil de entender**, con el fin de facilitar su mantenimiento y modificaciones.

## Eficiencia

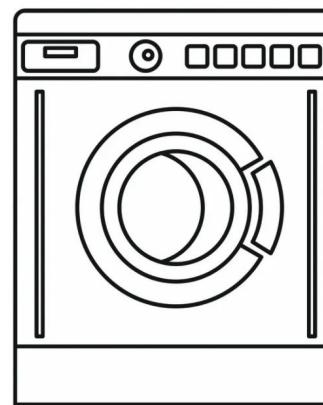
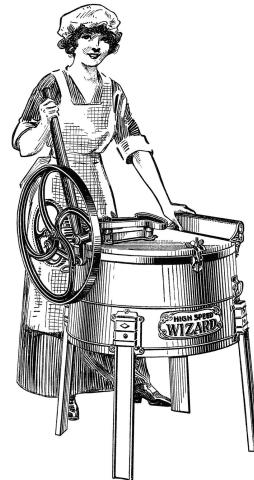
El algoritmo debe resolver el problema deseado en un **tiempo y con un consumo de recursos razonable**.

## Generalidad

El algoritmo debe poder **adaptarse a otros problemas similares** con pocos cambios.

# Lenguajes de programación

Un mismo algoritmo puede ser expresado en **diferentes lenguajes de programación**, en función de factores como el ámbito de aplicación o la tecnología disponible.



# Líneas de comandos, programas y scripts

Durante este módulo hablaremos de **líneas de comandos, programas y scripts**.

## Línea de comandos

Herramienta que permite dar instrucciones de **forma interactiva** a un programa o sistema operativo mediante la escritura de comandos de texto.

## Programa

**Secuencia de instrucciones** escritas en un lenguaje de programación que permiten realizar una tarea específica en un ordenador.

## Script

Término que se utiliza tanto para referirse a una **secuencia de comandos** como a un **programa relativamente simple**.

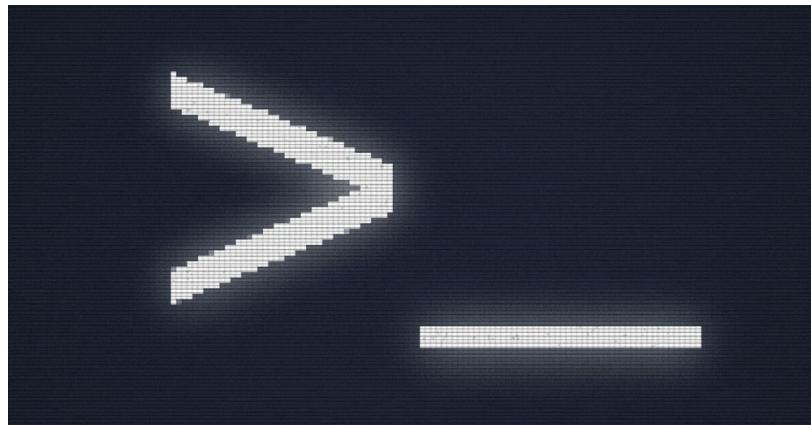
# Línea de comandos

Una **interfaz de línea de comandos** o ***command-line interface (CLI)*** es una interfaz de usuario que permite dar órdenes a un sistema por medio de comandos de texto.

Las interfaces de línea de comandos existen desde los inicios de la computación y todavía son una parte fundamental de los sistemas operativos actuales como Windows, MacOS o GNU/Linux.

La interfaz de línea de comandos se conoce también como **línea de comandos**, **shell**, **terminal**, ***prompt***, **consola**, o **símbolo del sistema**.

El modo de funcionamiento más simple de una interfaz de línea de comandos se denomina **modo interactivo**. En este modo, el usuario teclea las órdenes una a una y las ejecuta pulsando la tecla Enter.



## Funcionamiento de una línea de comandos

Las interfaces de línea de comandos actuales **emulan el funcionamiento de las antiguas terminales o consolas**.

Estas terminales eran dispositivos electromecánicos autónomos consistentes en un **teclado** y una **impresora** que actuaban como **dispositivos de entrada y salida de datos**, en formato de texto, en los primeros ordenadores. Con el paso del tiempo las impresoras se fueron sustituyendo por pantallas de vídeo.

El flujo de datos de entrada que reciben estos dispositivos, generalmente consistente en texto introducido a través del teclado, se conoce como **entrada estándar**, mientras que el flujo de datos de salida se conoce como **salida estándar**. En el caso de que el terminal genere alguna información de error lo hace a través de un tercer canal o flujo denominado **error estándar**.

La mayoría de los terminales muestran tanto la salida estándar como el error estándar en el mismo dispositivo, que generalmente es la pantalla.



# Scripts

Además del modo interactivo, en el cual se interpretan uno a uno los comandos u órdenes introducidas por el usuario a través del teclado, la mayoría de las interfaces de línea de comandos permiten **procesar archivos** que contienen una **secuencia de instrucciones** que se conoce como **script**.

```
GNU nano 2.8.6                               File: First.sh                               Modified

#!/bin/bash
echo "Hello World"

^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos
^X Exit      ^R Read File  ^\ Replace   ^U Uncut Text^T To Linter ^  Go To Line
```

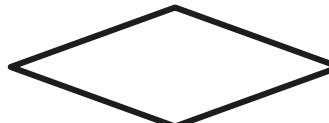
# Representación gráfica de algoritmos



# Representación gráfica de algoritmos

Los algoritmos se pueden **representar de forma gráfica** mediante los llamados **diagramas de flujo**.

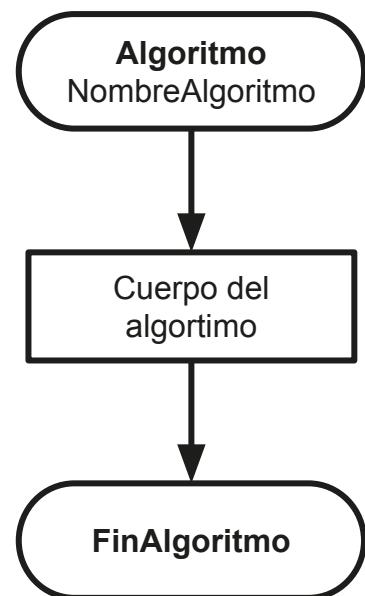
Los diagramas de flujo utilizan **símbolos normalizados** en la norma **ISO 5807**.

	Línea de flujo
	Símbolos terminales de inicio / fin
	Acción
	Subalgoritmo
	Leer / Escribir
	Decisión
	Comentario
	Conector on-page
	Conector off-page

Algunos de los símbolos más utilizados para la representación gráfica de algoritmos (ISO 5807).

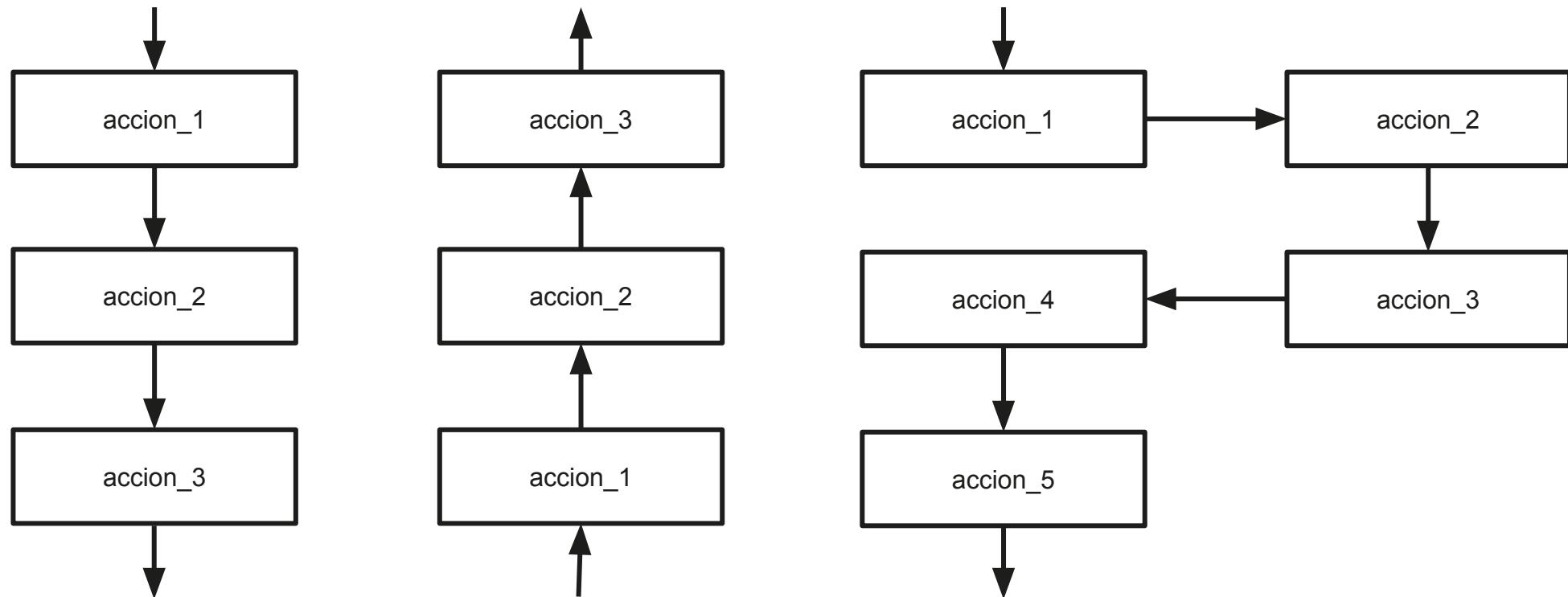
# Símbolos terminales

Todo algoritmo debe indicar dónde comienza y dónde termina mediante la utilización de los símbolos terminales de **inicio** y **fin**.



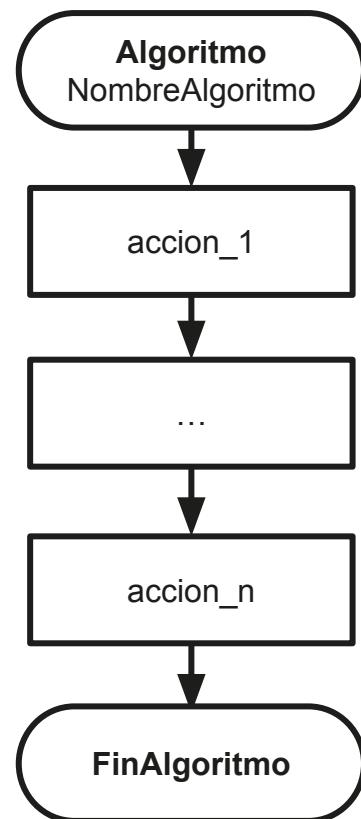
# Líneas de flujo

El **orden concreto de procesamiento** de las acciones se especifica mediante las **líneas o flechas de flujo**.



# Estructura secuencial

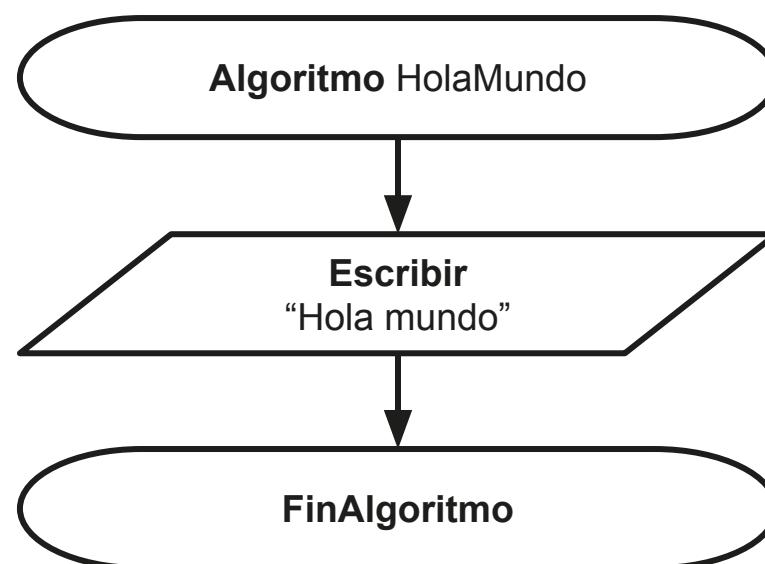
La estructura más básica de un algoritmo es la **estructura secuencial**, que representa una lista de acciones que se deben **procesar en orden**, una a continuación de otra.



## Escribir

El símbolo **Escribir** nos permite mostrar información al usuario.

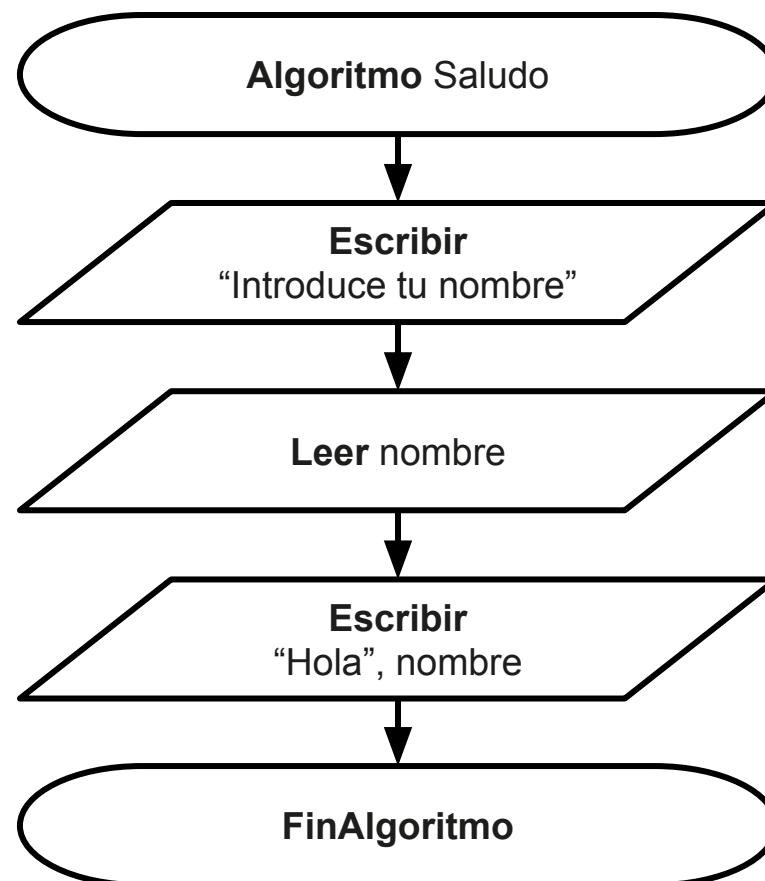
Por ejemplo, el siguiente algoritmo mostrará al usuario el texto “Hola Mundo”.



## Leer

El símbolo **Leer** nos permite obtener información del usuario y almacenarla en una **variable**.

Por ejemplo, el siguiente algoritmo solicitará al usuario que introduzca su nombre y seguidamente lo saludará.

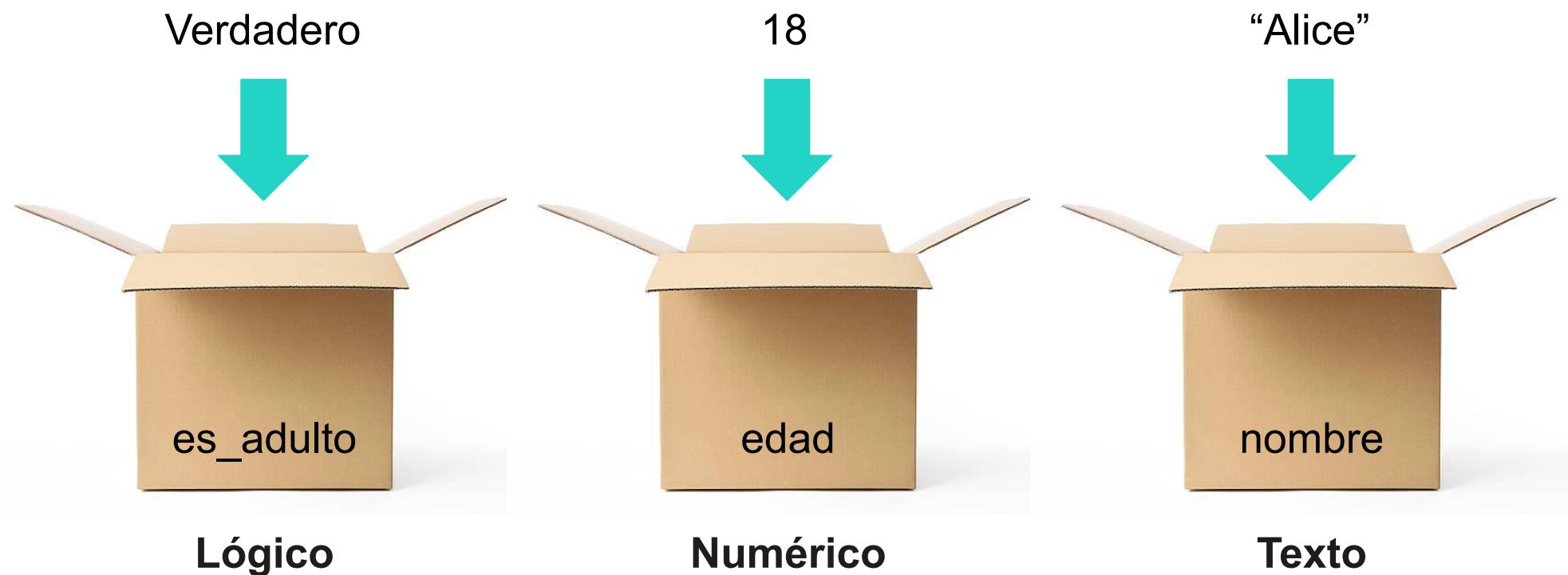


# ¿Qué es una variable?

Una variable es un **identificador o nombre que representa un valor** y que se utiliza para **guardar los datos** que necesita nuestro programa y poder operar con ellos.

En un programa informático, **toda variable tiene asociado un tipo**, que identifica el tipo de datos que puede almacenar.

Cuando el valor de una variable no va a cambiar durante toda la ejecución del algoritmo la denominamos **constante**.



# Tipos de datos

A la hora de crear nuestros algoritmos consideraremos los siguientes tipos de datos elementales:

## Lógico

El tipo lógico o tipo booleano tiene su origen en el álgebra de Boole. Este tipo únicamente puede tener **dos valores posibles: VERDADERO y FALSO.**

## Entero

Permite representar **números enteros** (sin decimales), que pueden ser positivos o negativos.

Ej.: 1, -2, 123, 0

## Real

Permite representar **números reales** (con decimales), que pueden ser positivos o negativos. Para separar decimales utilizaremos el punto (“.”).

Ej.: 1.0, -2.5

## Carácter

Permite representar **caracteres o cadenas de caracteres**. Para indicar que un valor es de tipo carácter lo representaremos encerrado entre comillas simples o dobles.

Ej.: “A”, “a”, “12”, “Hola mundo”, ‘Hola mundo’

# Reglas de nomenclatura

Todos los lenguajes de programación tienen ciertas **reglas** que debemos respetar a la hora de nombrar nuestras variables. De lo contrario nuestro programa no funcionará. Las reglas más comunes son las siguientes:

- Los caracteres del identificador o nombre pueden ser una **letra de alfabeto inglés**, mayúscula o minúscula (a-z, A-Z), un **dígito** (0-9), o el **carácter “\_”**.
- No puede haber **espacios** entre caracteres.
- El primer carácter del identificador no puede ser un **dígito**.
- No se pueden usar **palabras reservadas** del lenguaje (ej.: algoritmo, leer, escribir, ...) como identificadores de variables o constantes.

Ejemplos.:

 Aclarado1	 1Aclarado
 Aclarado_1	 Aclarado-1
 Numero_Lavados	 Numero Lavados
 NumeroLavados	 Leer

# Convenciones

Además, cada lenguaje tiene sus **recomendaciones** o **convenciones de estilo** que se recomiendan respetar para **facilitar la legibilidad** de nuestros programas y **comunicarnos mejor** con otros profesionales.

Las más comunes que afectan a los identificadores de variables y constantes son las siguientes:

- Los identificadores de las **variables** se representan con letras **minúsculas**.
- Los identificadores de las **constantes** se representan con letras **mayúsculas**.
- Se deben utilizar identificadores o nombres **significativos**.
- Las palabras se separan con un **guión bajo** (“\_”).

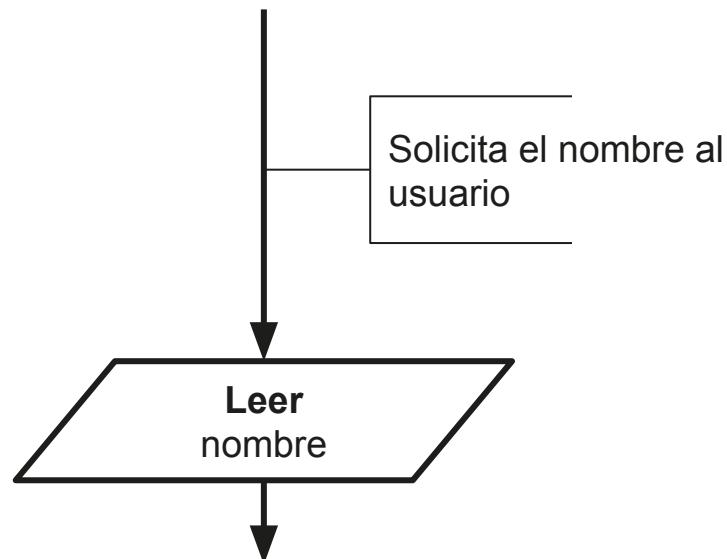
Ejemplos.:

- ✓ aclarado
- ✓ aclarado\_1
- ✓ numero\_lavados
- ✓ PI

# Comentarios

Los comentarios son **anotaciones** que hacemos en nuestro algoritmo con el fin de **aclarar su funcionamiento**.

Los comentarios son **ignorados por el procesador**.



# Estructuras de control

Además de la estructura secuencial existen dos tipos de **estructuras de control** que nos permiten construir algoritmos más complejos:

## Estructura selectiva

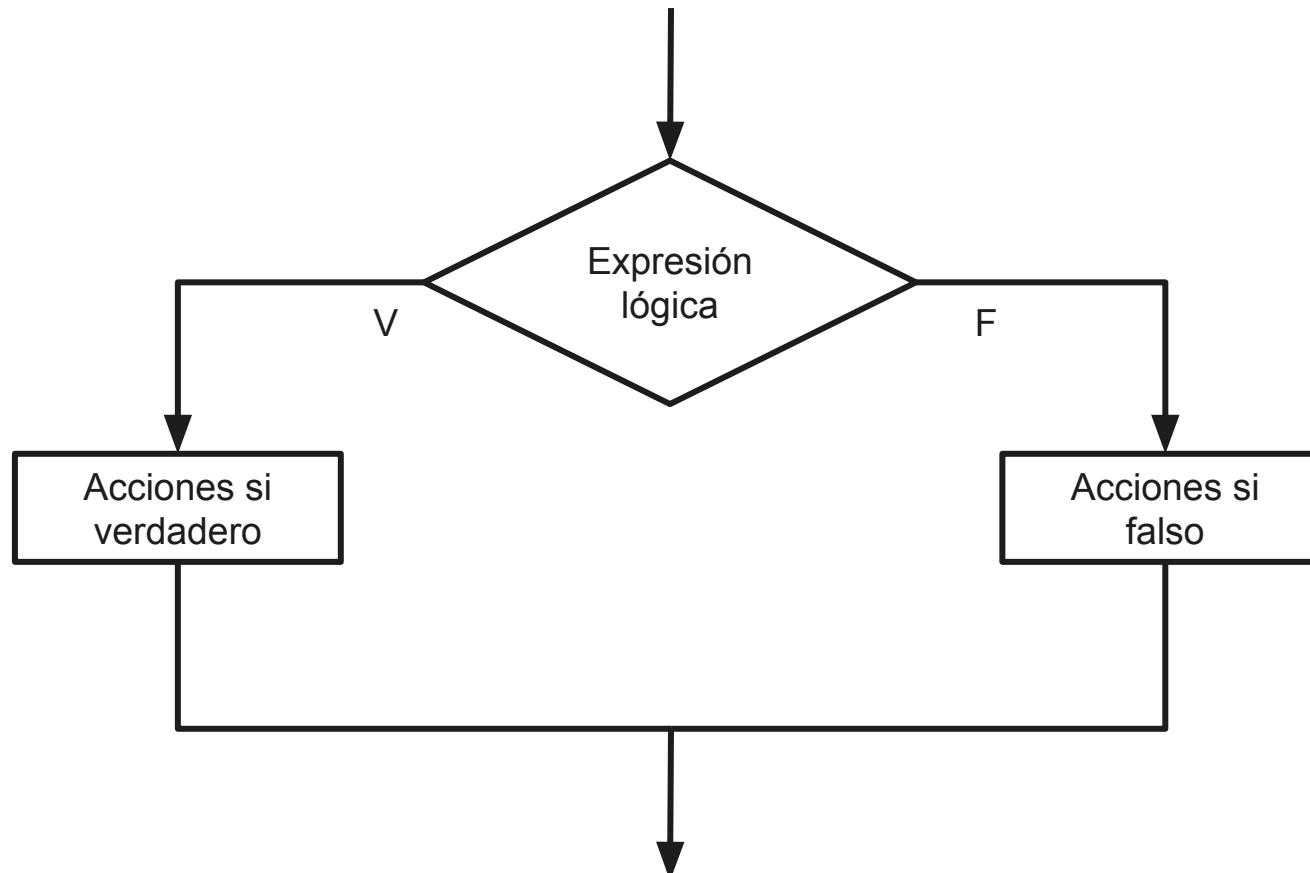
También denominada **decisión**, **estructura condicional** o **bifurcación**. Permite ejecutar un conjunto diferente de acciones en función de si se cumple o no una determinada condición.

## Estructura iterativa

También denominada **repeticIÓN** o **bucle**. Permite ejecutar repetidamente un conjunto de acciones. Generalmente se utiliza junto con una **estructura selectiva** que comprueba si se han ejecutado las repeticiones deseadas.

# Estructura selectiva

La **estructura selectiva** nos permite **ejecutar diferentes acciones** en función de si se cumple o no una determinada **condición**, que se representa mediante una expresión lógica.



# Operadores relacionales

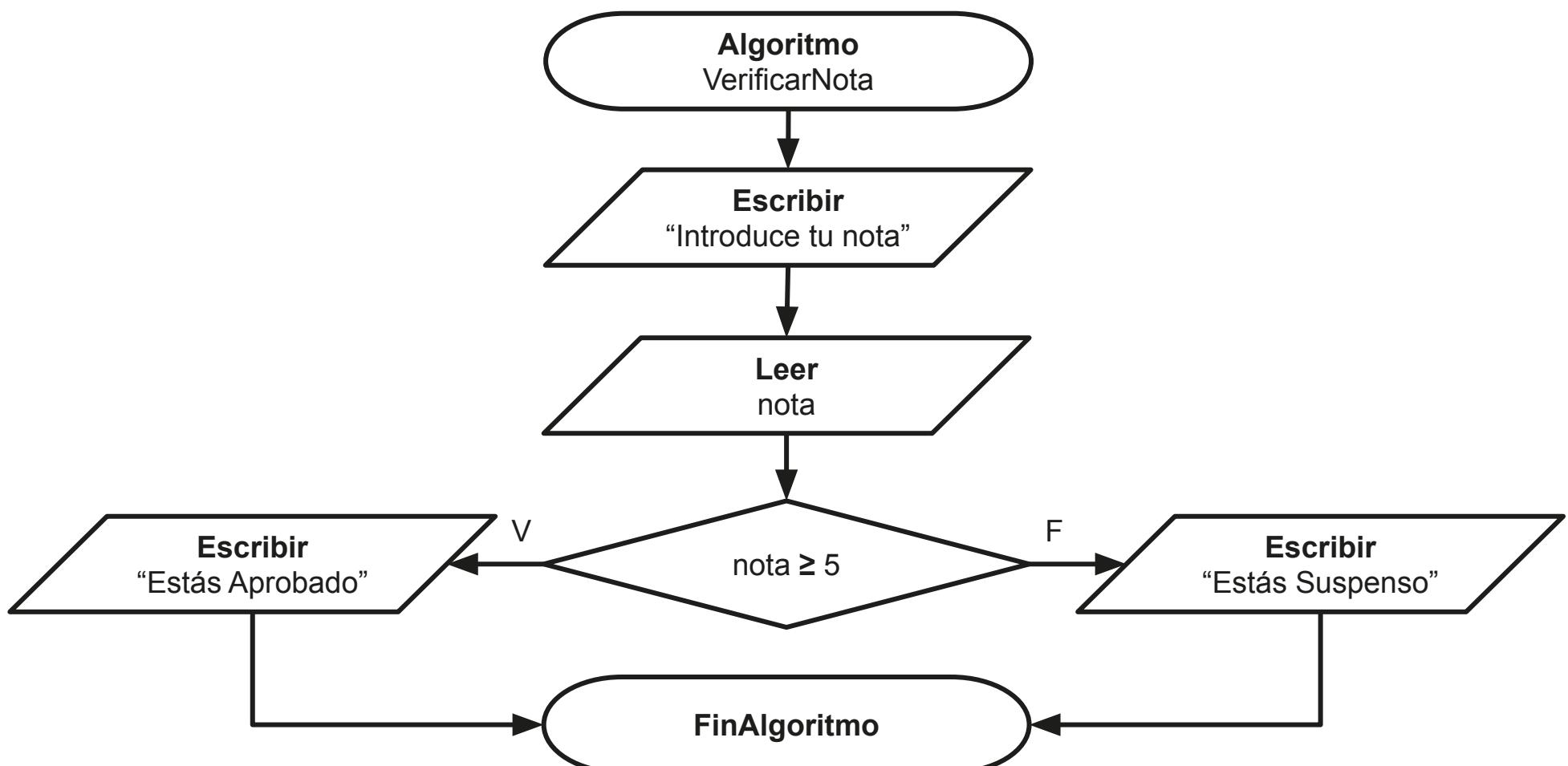
Una expresión lógica es aquélla que como resultado de su evaluación podemos obtener únicamente dos valores: **VERDADERO** o **FALSO**.

Las expresiones lógicas más sencillas son aquéllas que obtenemos como resultado de **comparar dos valores** mediante la utilización de un **operador relacional**:

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
=	Igual	$5 = 3$	FALSO
≠	Distinto	$5 \neq 3$	VERDADERO
<	Menor	$5 < 3$	FALSO
≤	Menor o igual	$5 \leq 3$	FALSO
>	Mayor	$5 > 3$	VERDADERO
≥	Mayor o igual	$5 \geq 3$	VERDADERO

# Estructura selectiva

## Ejemplo



# Estructuras iterativas

Existen dos tipos básicos de estructuras iterativas:

## Mientras

Se procesan repetidamente las acciones mientras la condición sea cierta. Si inicialmente la condición es falsa, las acciones no se procesan ninguna vez.

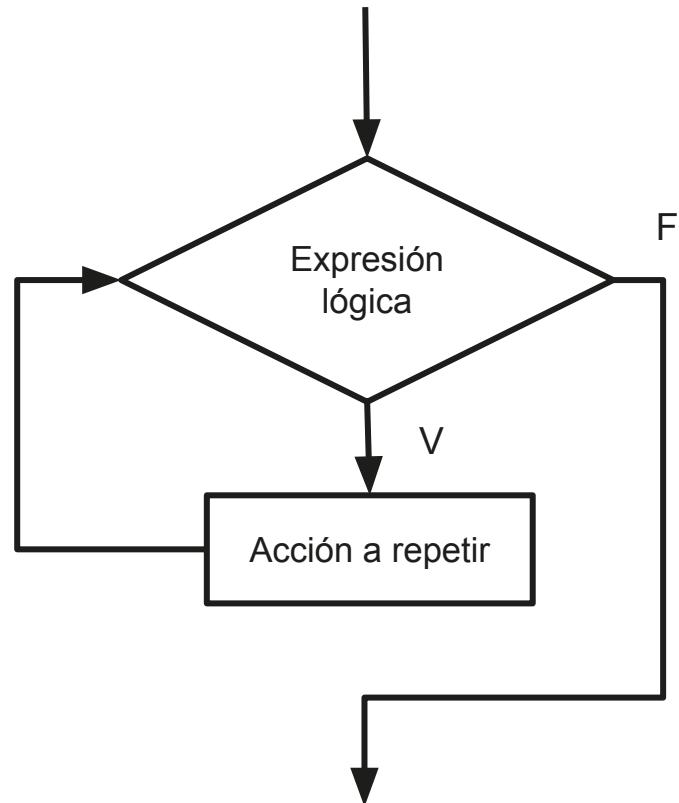
## Repetir

Se procesan las acciones al menos una vez y se repite mientras la condición sea cierta.

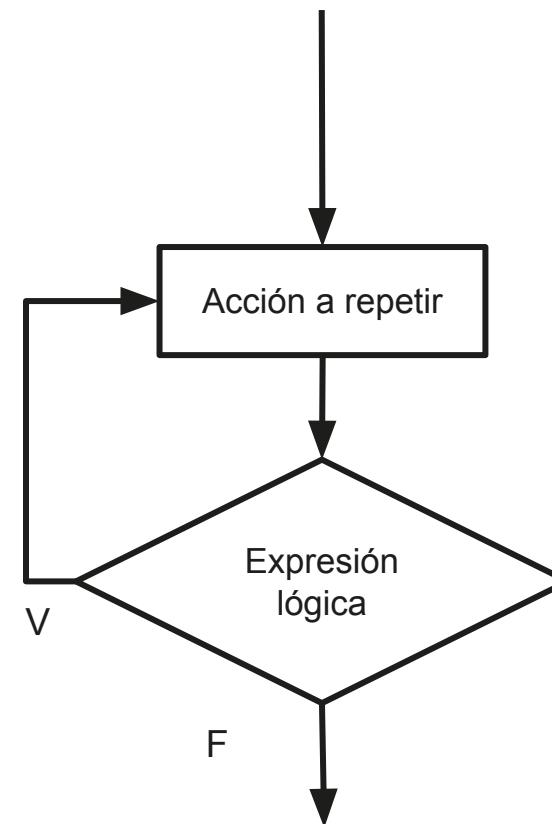
Se conoce también como Hacer ... mientras.

# Estructuras iterativas

Estructuras iterativas Mientras y Repetir ... mientras:



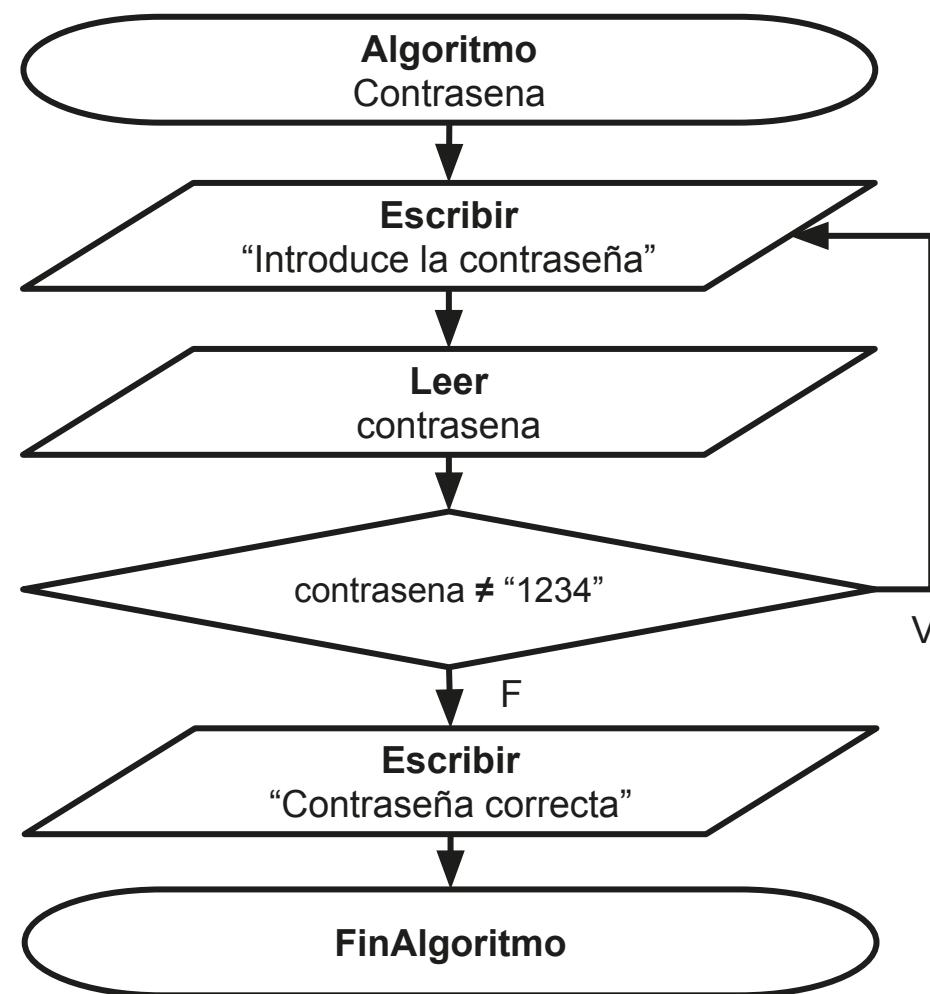
**Mientras**



**Repetir ... mientras**

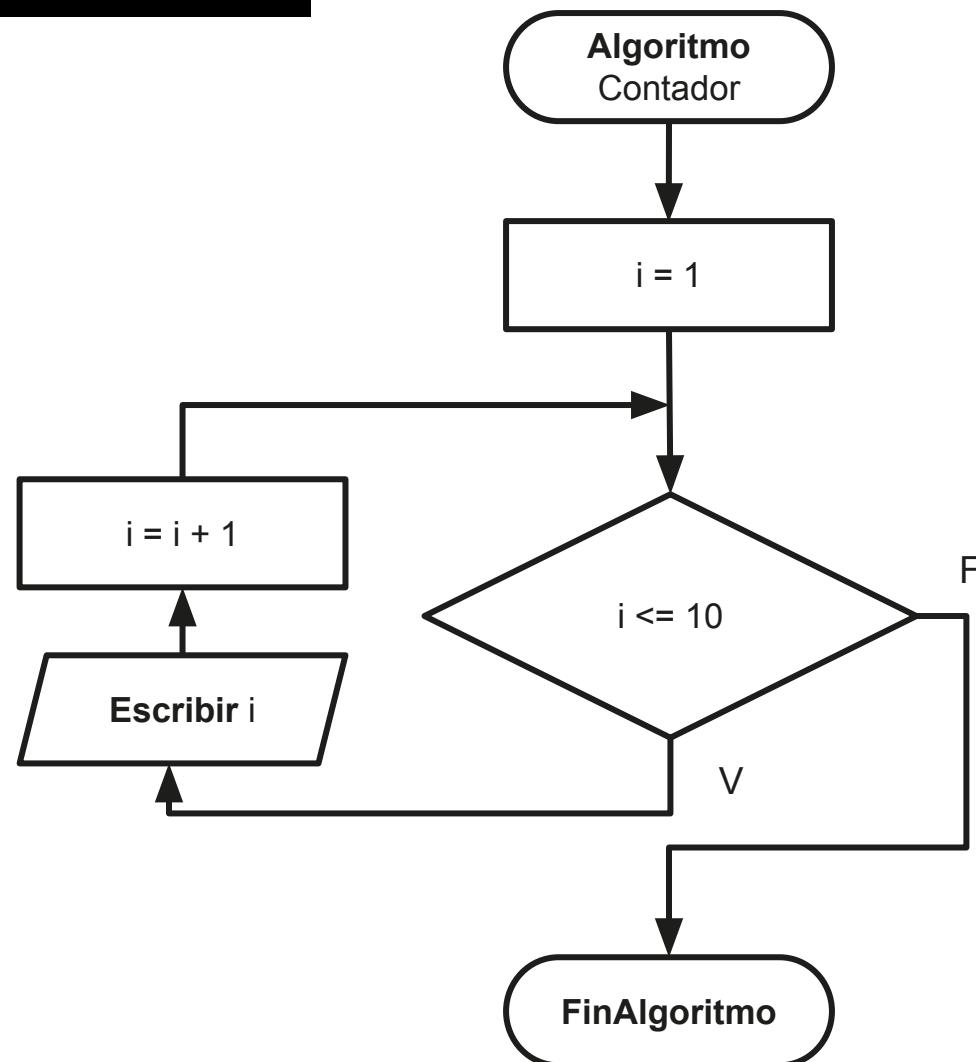
# Estructura iterativa

## Ejemplo



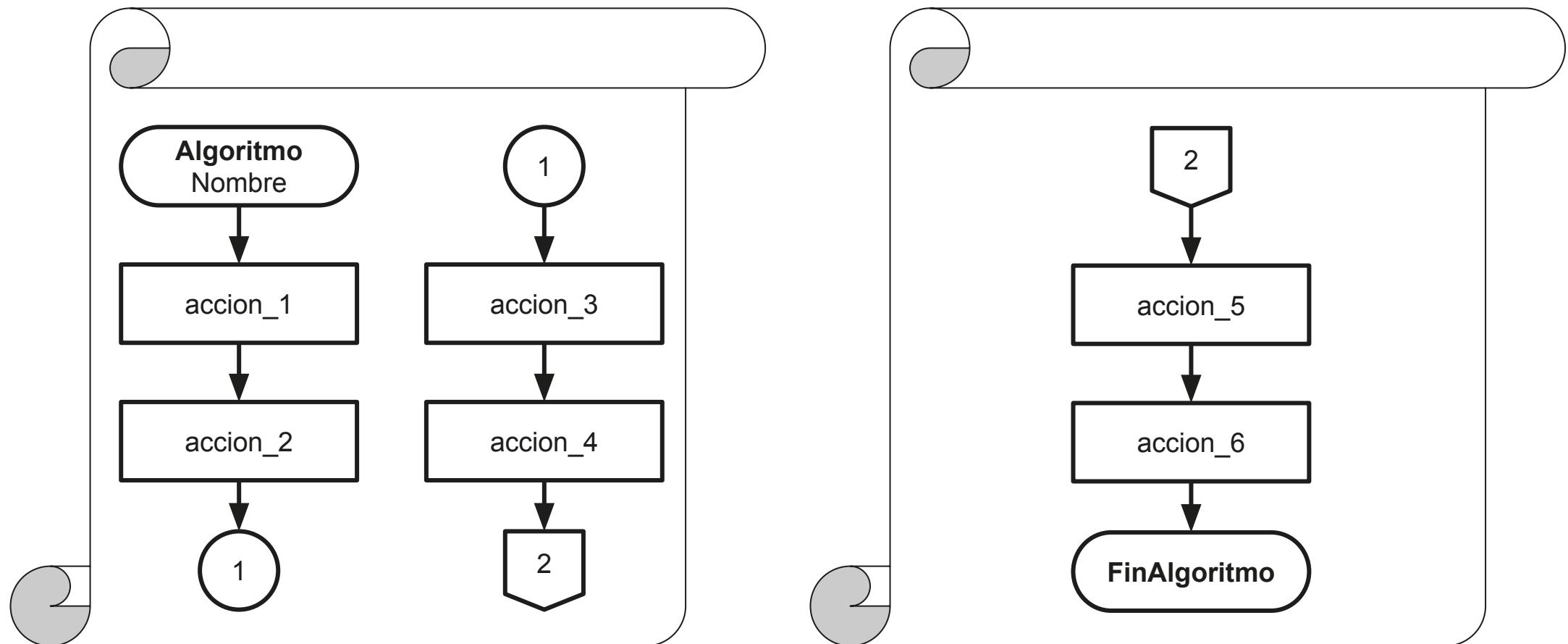
# Estructura iterativa

## Ejemplo



# Conectores

Para organizar mejor nuestros algoritmos podemos dividirlos en fragmentos y simbolizar su unión mediante el uso de **conectores**. Utilizaremos conectores diferentes en función de si el fragmento conectado se encuentra en la **misma página** o en una **página diferente**.



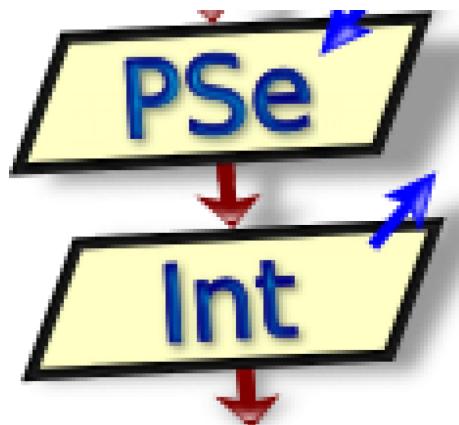
# Representación gráfica de algoritmos con PSelnt



## ¿Qué es PSelnt?

PSelnt (**Pseudocódigo Intérprete**) es una herramienta para asistir al estudiante en sus primeros pasos en programación.

PSelnt consiste en:



- Un **editor gráfico** para la descripción de algoritmos mediante **diagramas de flujo**.
- Un **editor de texto** para la descripción de algoritmos mediante **pseudocódigo**.
- Un **intérprete** que permite **procesar los algoritmos** descritos.
- Una serie de **herramientas de ayuda** al aprendizaje.

PSelnt **no es un lenguaje de programación real**, sino una **herramienta de aprendizaje** que permite iniciarse en el mundo de la programación mediante la escritura y ejecución de algoritmos sencillos.

# Ventajas de PSelnt

## Idioma

Los lenguajes de programación suelen estar inspirados en el idioma inglés. **PSelnt está escrito en español**, de modo que no es necesario aprender términos en inglés.

## Sencillez

PSelnt utiliza un **pseudocódigo sencillo y limitado**, pero suficiente para aprender los fundamentos de la programación.

## Centrado en la lógica

Los estudiantes de programación pasan más tiempo corrigiendo cuestiones de sintaxis propias del lenguaje que desarrollando la lógica del algoritmo. **PSelnt permite centrarse en los aspectos lógicos del problema** sin tener que lidiar con las particularidades de la sintaxis de un lenguaje real.

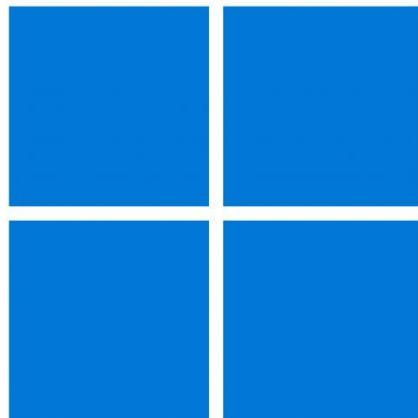
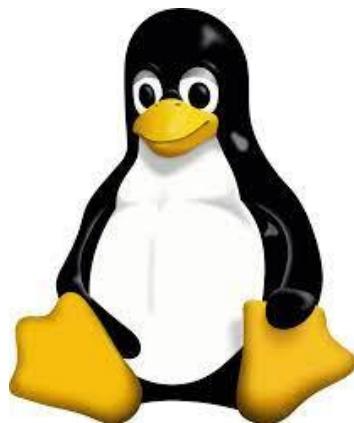
## Intérprete

Aunque podemos escribir pseudocódigo y dibujar diagramas de flujo utilizando simplemente lápiz y papel, no hay forma de saber si estos funcionan. **PSelnt incluye un intérprete que permite ejecutar nuestros algoritmos** y comprobar que funcionan correctamente.

## Instalación de PSelnt

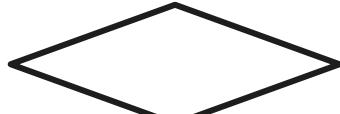
PSelnt es una aplicación de **software libre y multiplataforma**, disponible de forma gratuita para su descarga desde su página oficial de Source Forge:

- <http://pseint.sourceforge.net/>



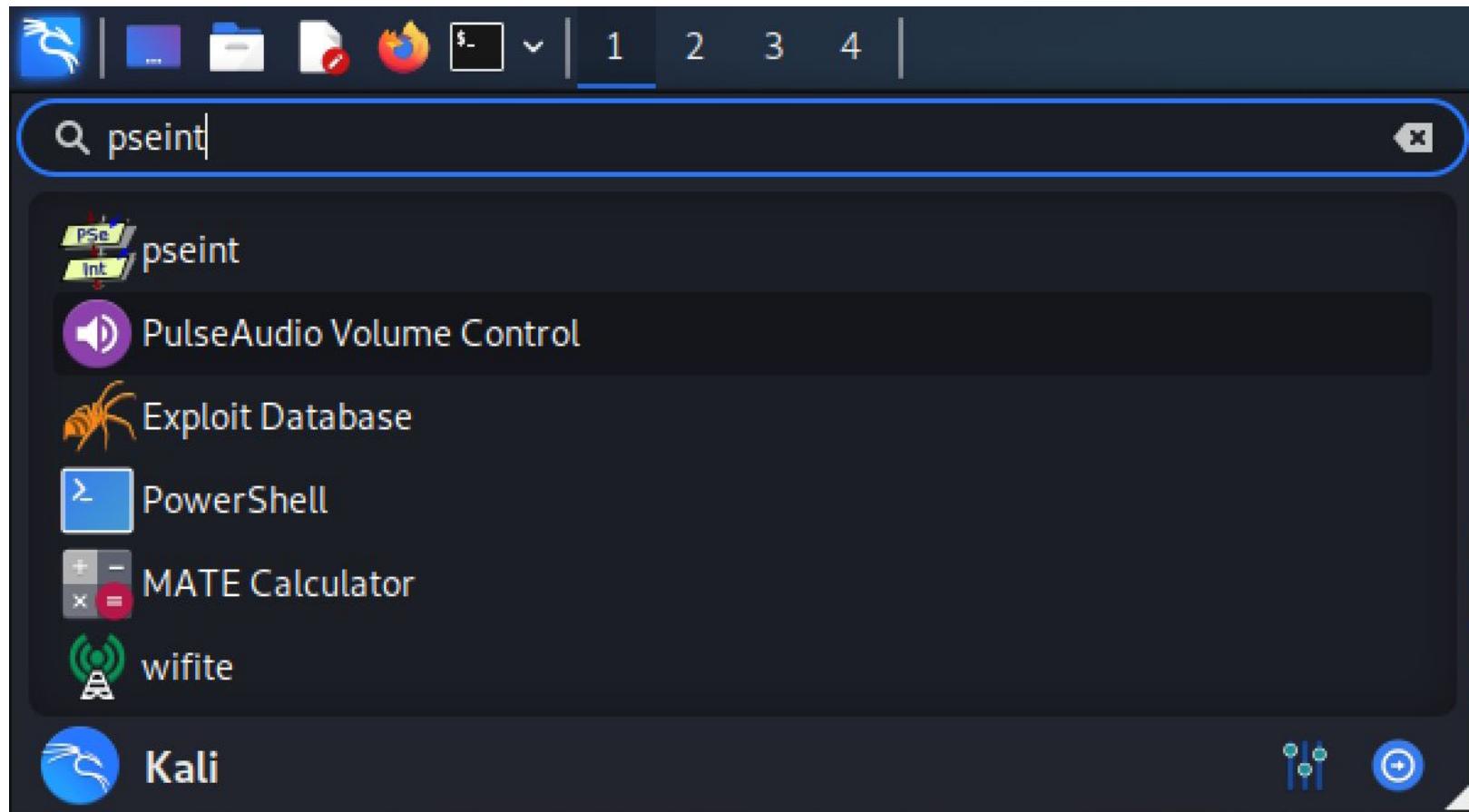
# Símbolos utilizados por PSelnt

PSelnt utiliza el siguiente conjunto de símbolos para describir los diagramas de flujo:

	Línea de flujo
	Símbolos terminales de inicio / fin
	Acción
	Subalgoritmo (Función)
	Comentario
	Leer
	Escribir
	Decisión
	Según
	Para

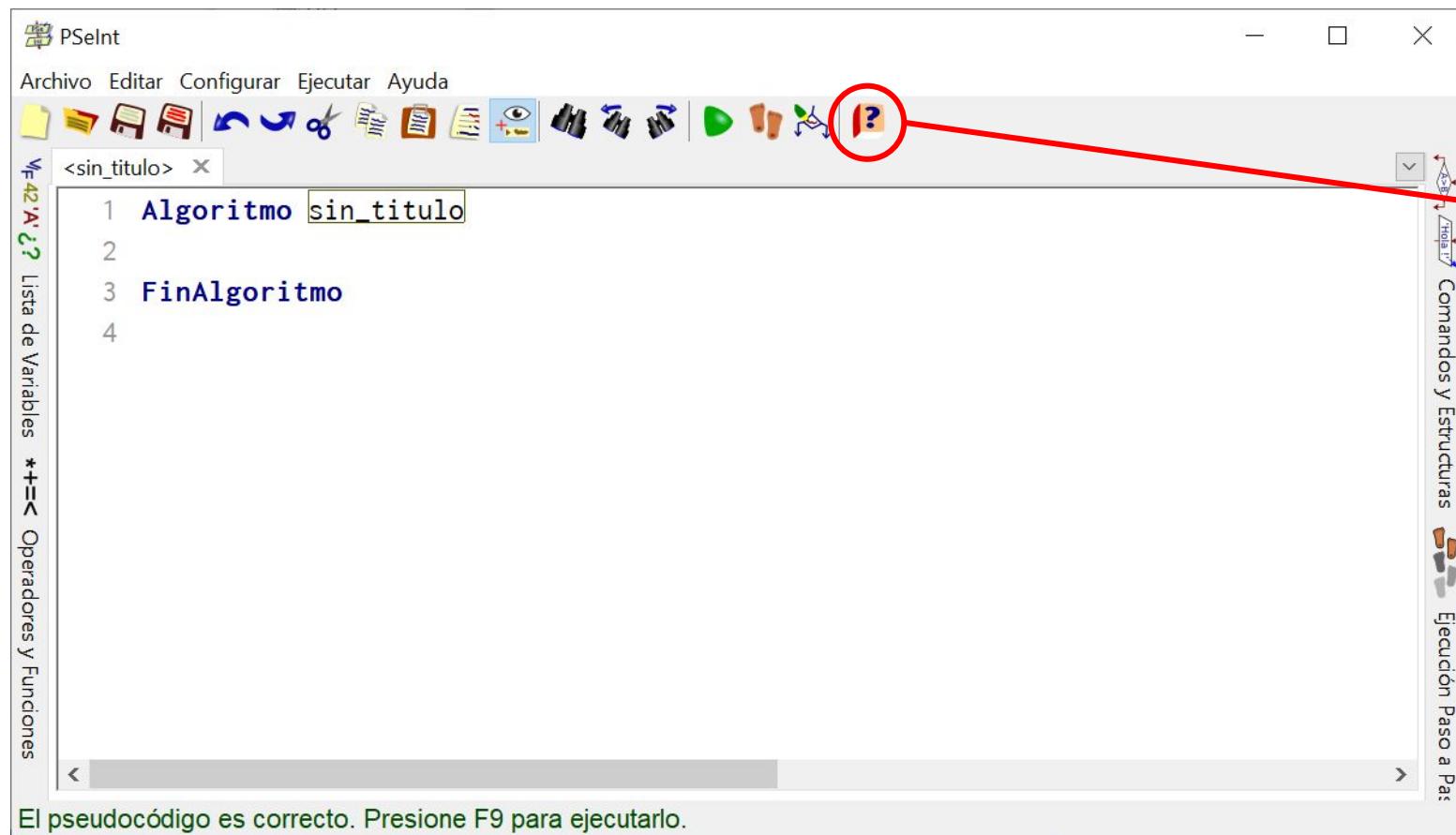
## Ejecución de PSelInt

PSelInt se encuentra instalado en la máquina virtual M2\_Kali de los materiales de este módulo.



# Ayuda de PSelInt

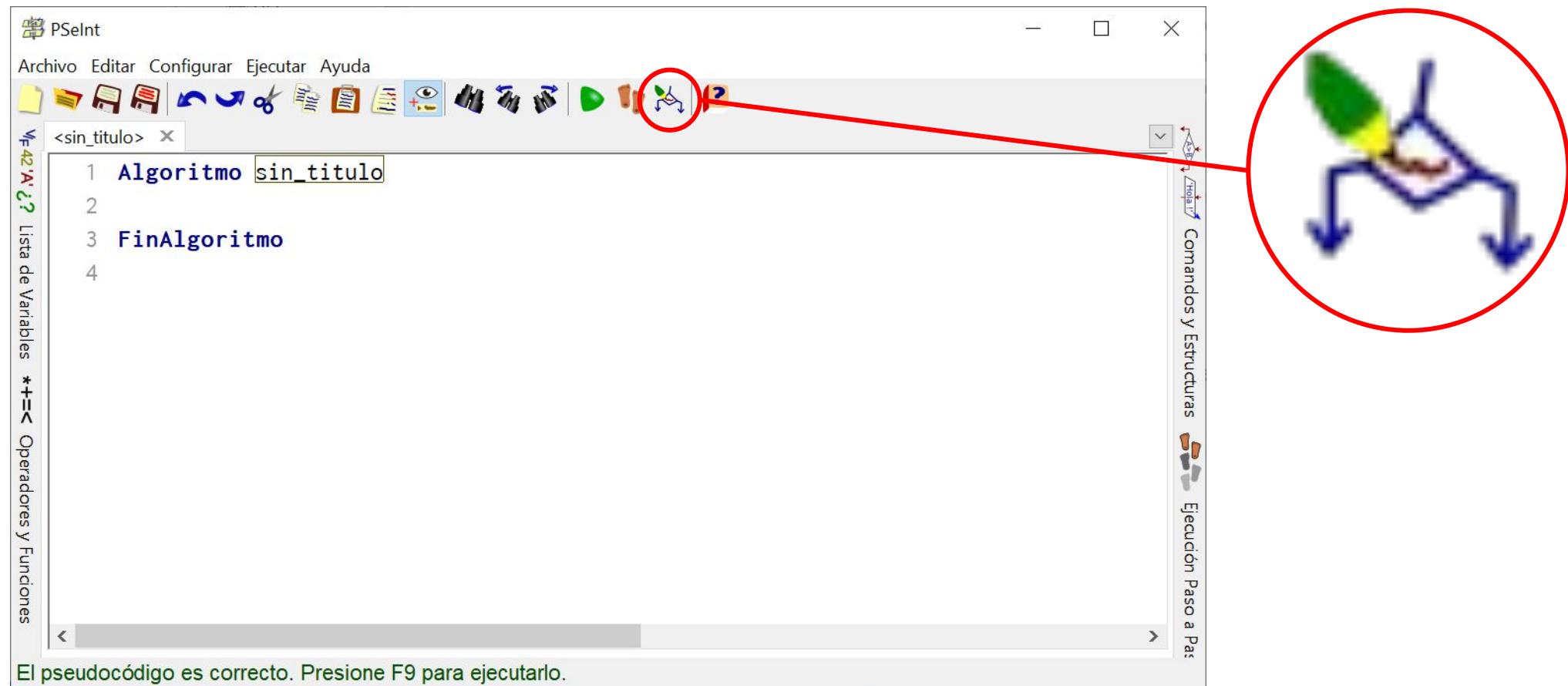
Podemos acceder en cualquier momento a la ayuda de PSelInt pulsando el icono “Ayuda”.



# Interfaz de PSeInt

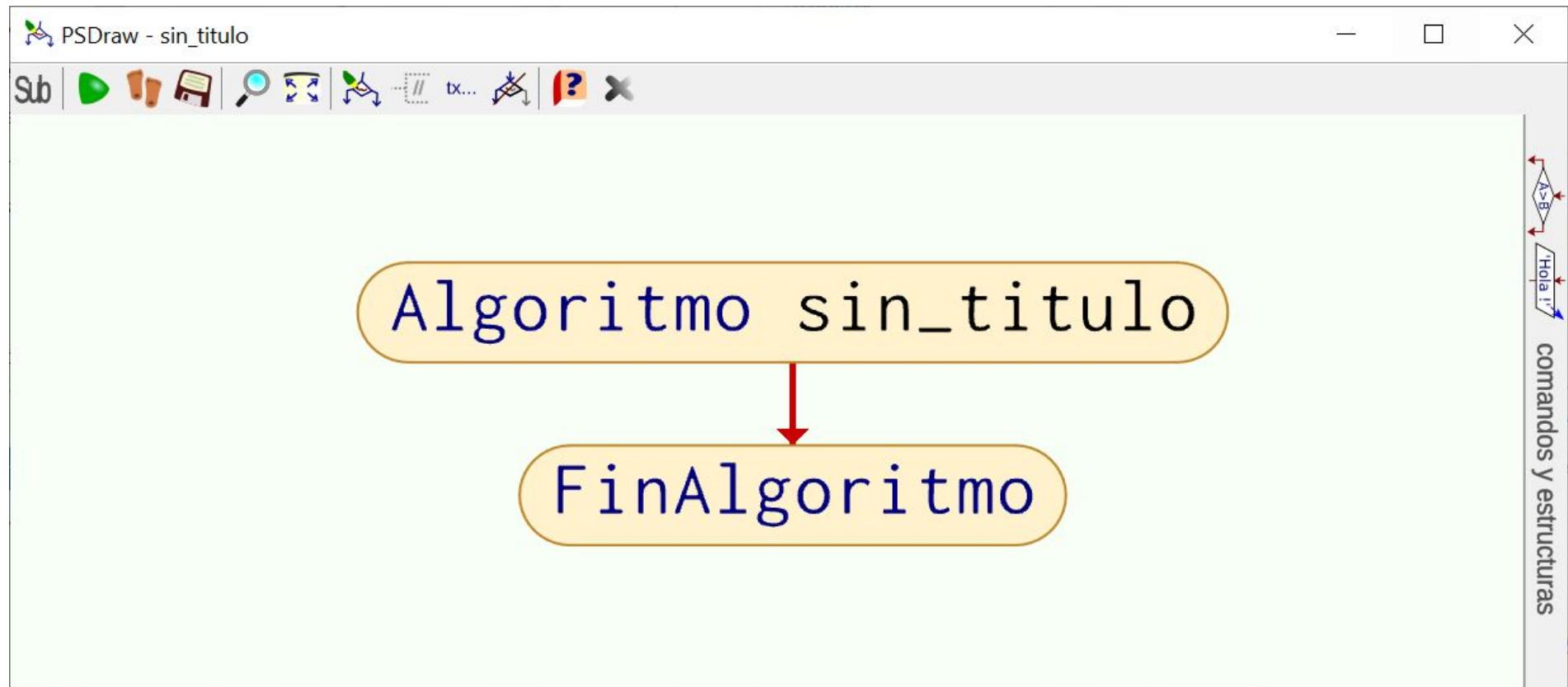
En la pantalla principal de PSeInt encontraremos el editor de pseudocódigo.

Para dibujar un diagrama de flujo abriremos PSDraw pulsando el icono “Dibujar Diagrama de Flujo”.



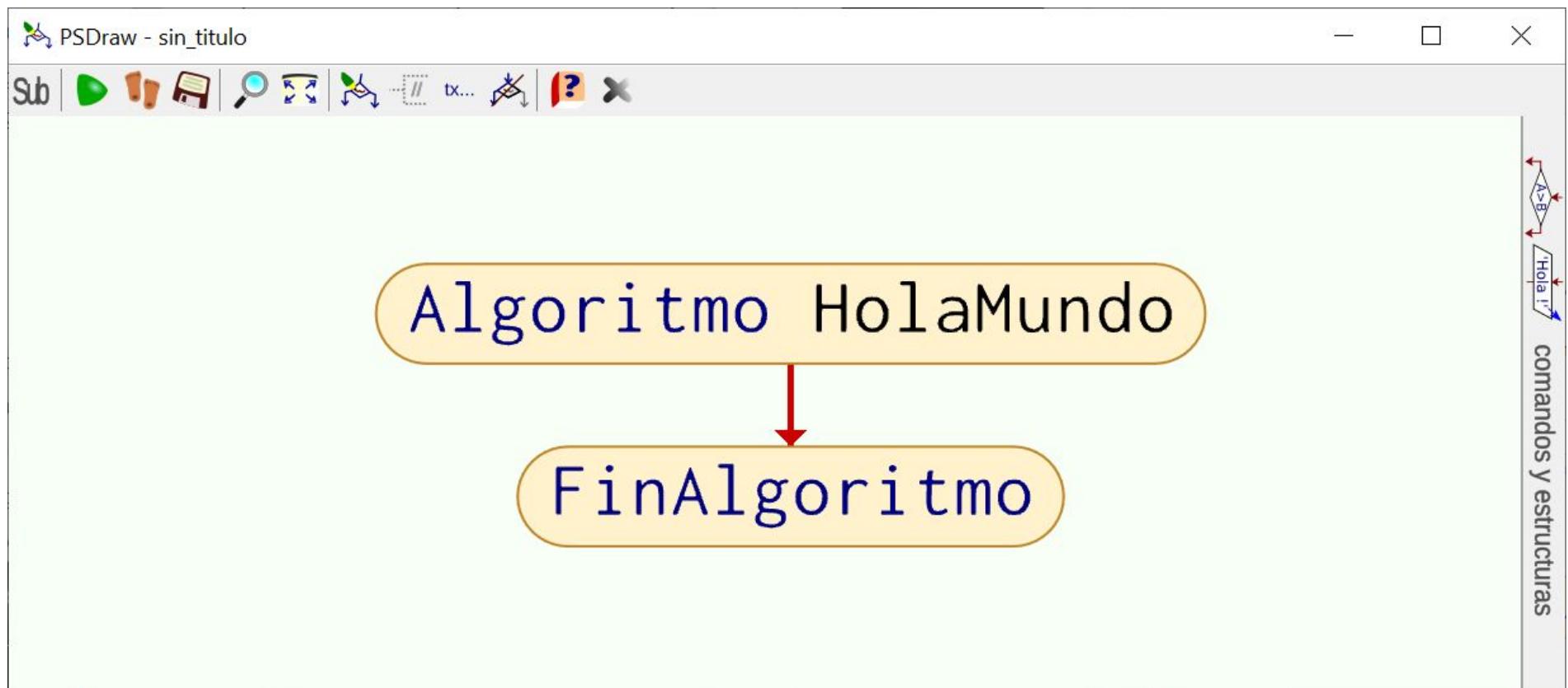
## Interfaz de PSelnt

La interfaz de PSDraw nos mostrará un algoritmo vacío con los símbolos terminales de inicio y fin.



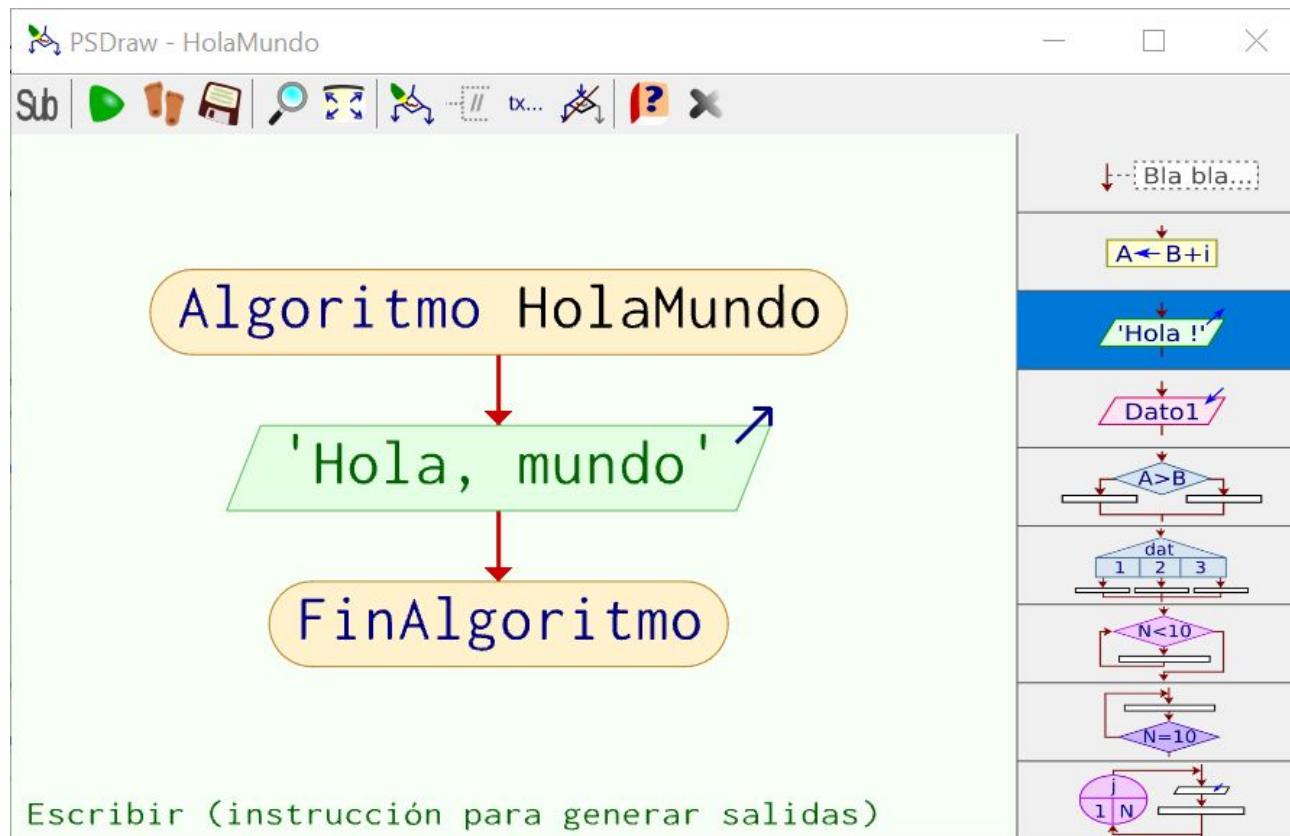
# Algoritmo HolaMundo

Editaremos el nombre del algoritmo y lo nombraremos como “HolaMundo”.



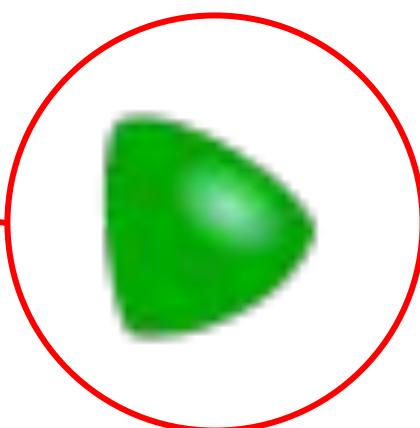
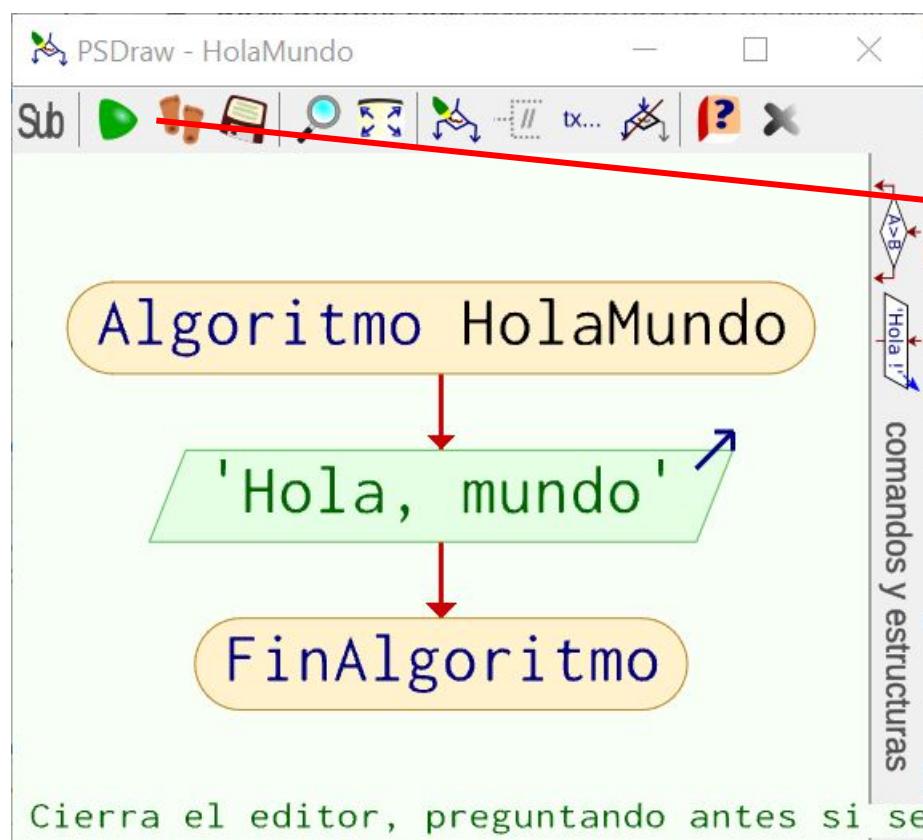
# Algoritmo HolaMundo

Arrastraremos un símbolo “Escribir” al área de dibujo e introduciremos la expresión “Hola, mundo”, escribiendo el texto entre comillas (simples o dobles).



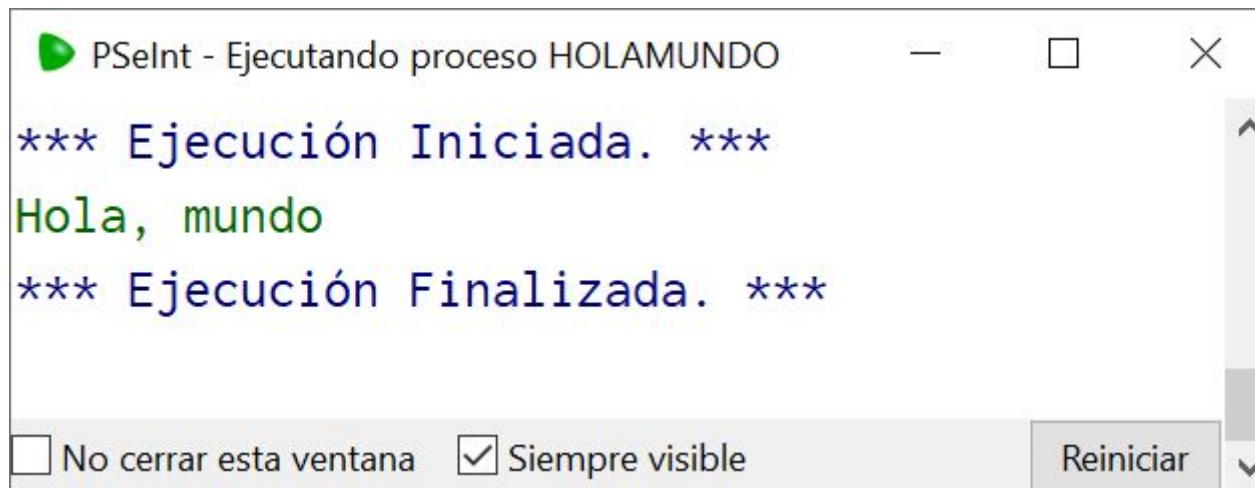
# Algoritmo HolaMundo

Para ejecutar nuestro algoritmo pulsaremos el icono “Ejecutar”.



## Algoritmo HolaMundo

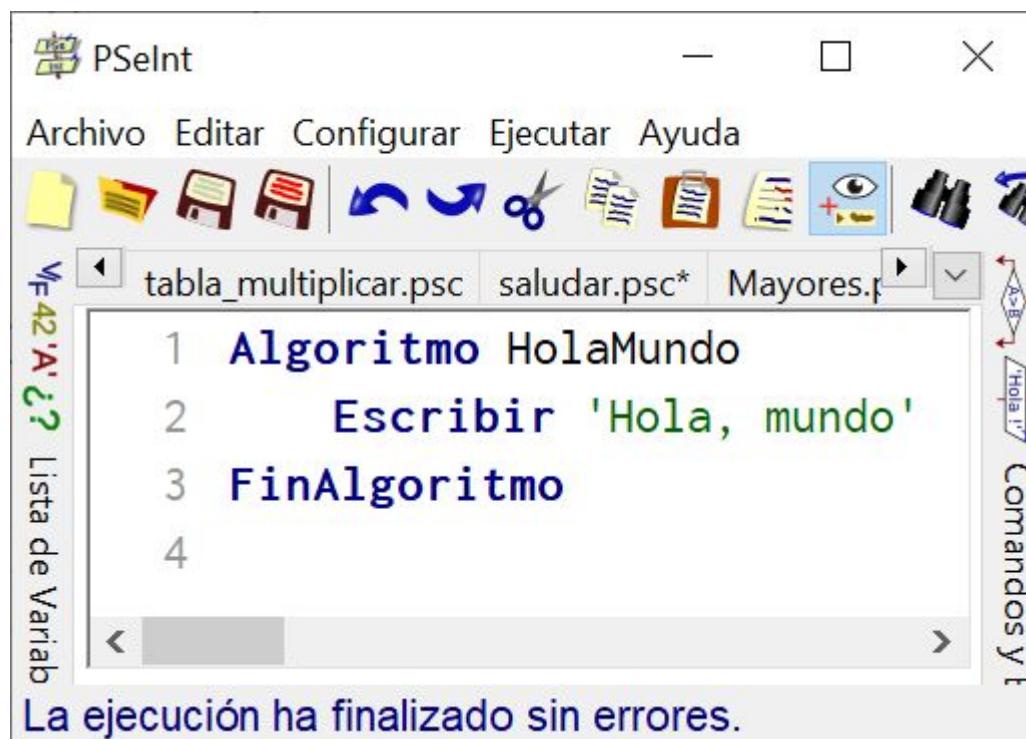
Veremos el resultado de nuestro algoritmo en la consola de PSeInt. Es recomendable marcar la opción “Siempre visible” para evitar que esta ventana quede oculta tras otras.



# Algoritmo HolaMundo

Si volvemos a la ventana principal de PSelnt veremos que se habrá generado el pseudocódigo equivalente al diagrama de flujo dibujado. Desde esta ventana podemos guardar nuestro algoritmo pulsando el icono “Guardar”, así como crear y editar algoritmos en pseudocódigo.

Por el momento utilizaremos únicamente el editor gráfico PSDraw para crear nuestros algoritmos.



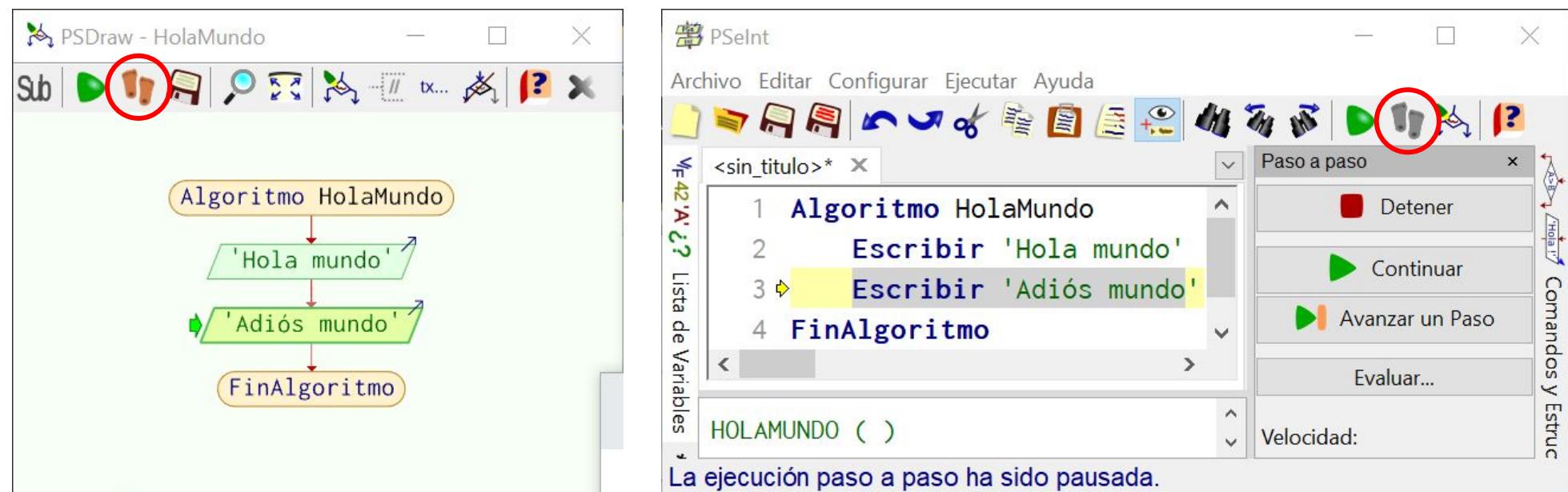
The screenshot shows the PSelnt application window. The menu bar includes Archivo, Editar, Configurar, Ejecutar, and Ayuda. The toolbar contains icons for new file, open file, save file, cut, copy, paste, find, and others. The file list at the top shows 'tabla\_multiplicar.psc', 'saludar.psc\*', and 'Mayores.psc'. The main pane displays the pseudocode:

```
1 Algoritmo HolaMundo
2   Escribir 'Hola, mundo'
3 FinAlgoritmo
4
```

A vertical sidebar on the left shows variable definitions: **VF**, **42**, **'A'**, **?**, and **Lista de Variables**. A vertical sidebar on the right shows flowchart symbols: **Comandos y Estructuras**, **Algoritmo**, **Condición**, **Bucle**, and **Salida**. A status message at the bottom says "La ejecución ha finalizado sin errores."

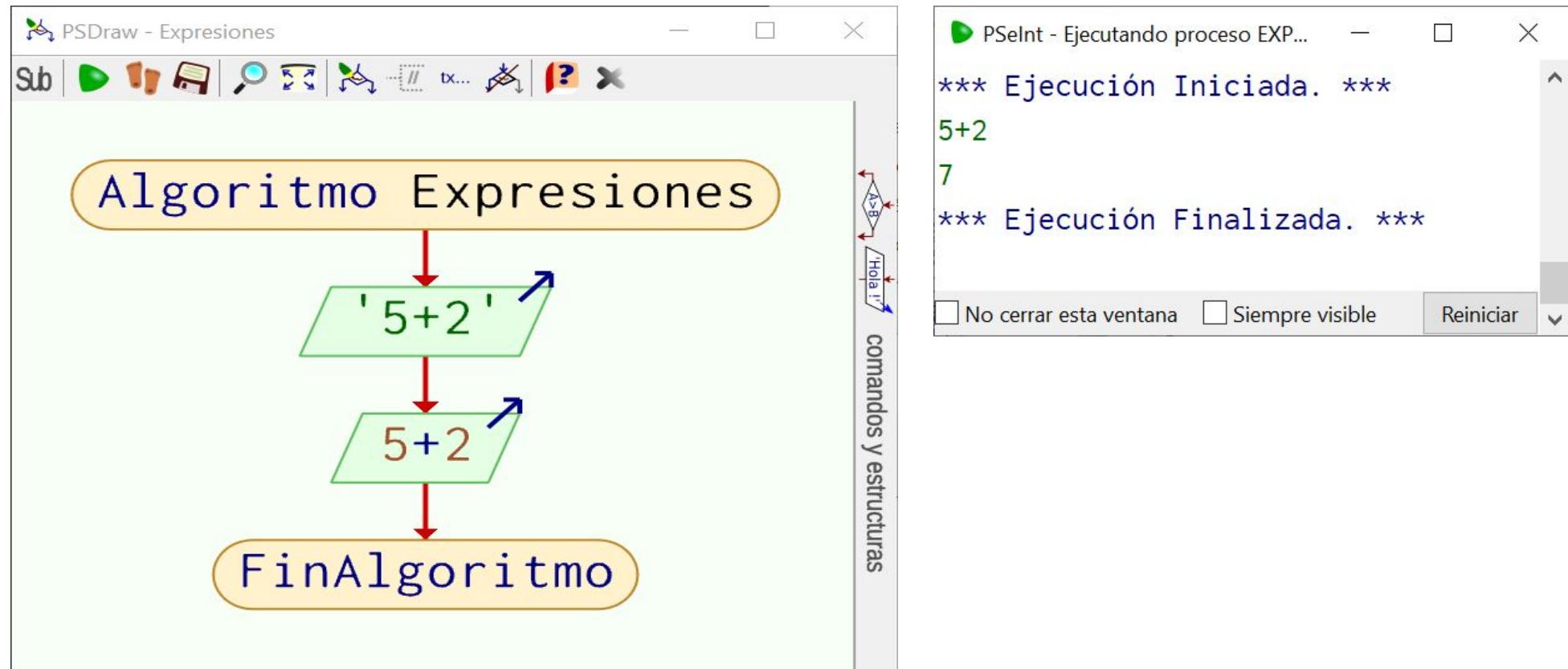
# Ejecución paso a paso

Pulsando el icono “Ejecutar paso a paso”, PSeInt ejecutará las acciones del algoritmo una a una y podremos seguir su evolución tanto en la pantalla principal de pseudocódigo como en el diagrama de flujo que se muestra en la pantalla de PSDraw.



# Expresiones

Los valores representados entre comillas, simples o dobles, son interpretados como texto. Si omitimos las comillas PSeInt interpretará que se trata de una expresión e intentará evaluarla.



# Operadores aritméticos

Los operadores aritméticos permiten realizar operaciones aritméticas entre dos valores numéricos.

PSelnt utiliza los siguientes operadores aritméticos:

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
<b>+</b>	Suma	<code>3 + 2</code>	5
<b>-</b>	Resta	<code>3 - 2</code>	1
<b>*</b>	Multiplicación	<code>3 * 2</code>	6
<b>/</b>	División	<code>3 / 2</code>	1.5
<b>↑ (^)</b>	Potencia	<code>3 ↑ 2 (3 ^ 2)</code>	9
<b>MOD</b>	Resto/Módulo	<code>3 MOD 2</code>	1

# Operadores aritméticos

Podemos comprobar el funcionamiento de los operadores aritméticos en PSelInt con el siguiente algoritmo.



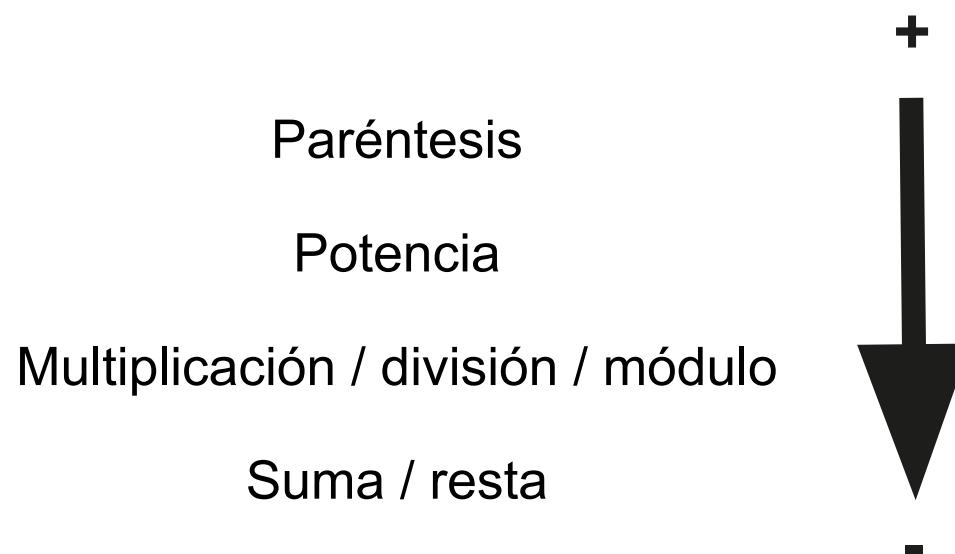
```

    *** Ejecución Iniciada. ***
    5
    1
    6
    1.5
    9
    1
    *** Ejecución Finalizada. ***
    
```

No cerrar esta ventana  Siempre visible Reiniciar

# Precedencia de los operadores

En PSeInt, al igual que en la mayoría de los lenguajes de programación, los operadores aritméticos se evalúan según el siguiente orden de precedencia:



A igualdad de prioridad los operadores se evalúan **de izquierda a derecha**.

Se puede determinar el orden en el que queremos que se apliquen los operadores mediante **el uso de paréntesis**.

# Precedencia de los operadores

$1 + 2 ^ 3 / 4 * 5$



$1 + 8 / 4 * 5$



$1 + 2 * 5$



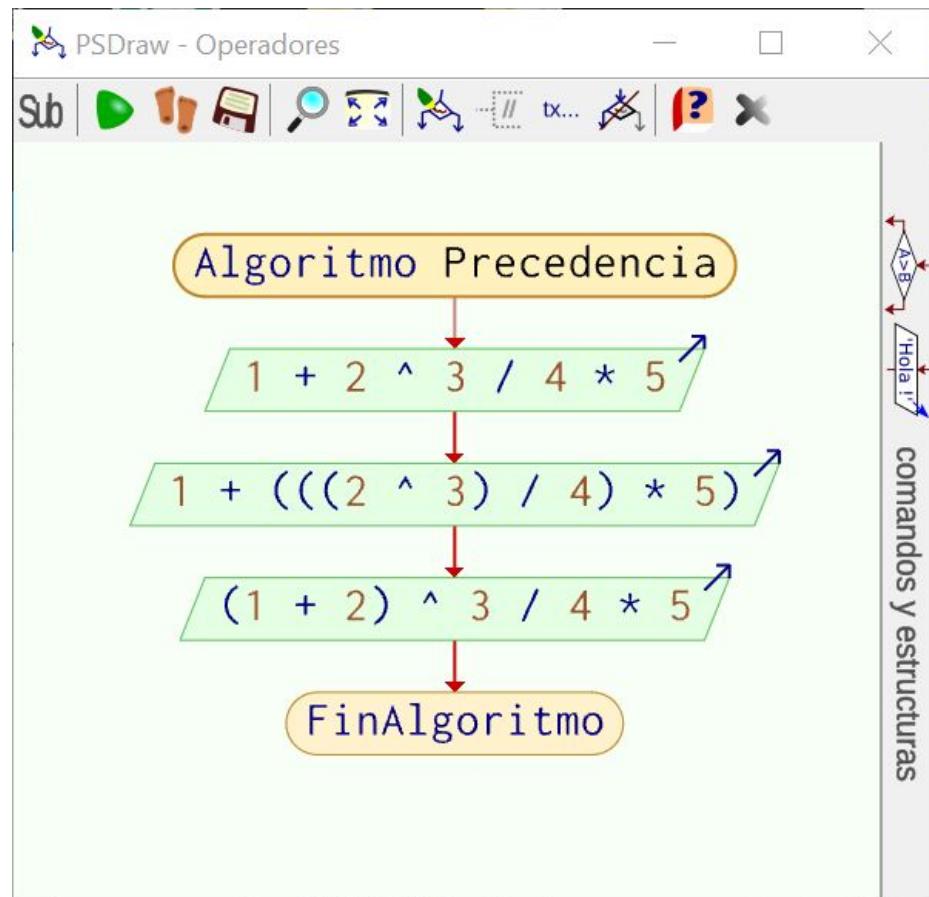
$1 + 10$



11

# Precedencia de los operadores

Podemos comprobar la precedencia de los operadores aritméticos en PSeInt con el siguiente algoritmo.



PSelint - Ejecutando proceso PRECEDENCIA

\*\*\* Ejecución Iniciada. \*\*\*

11  
11  
33.75

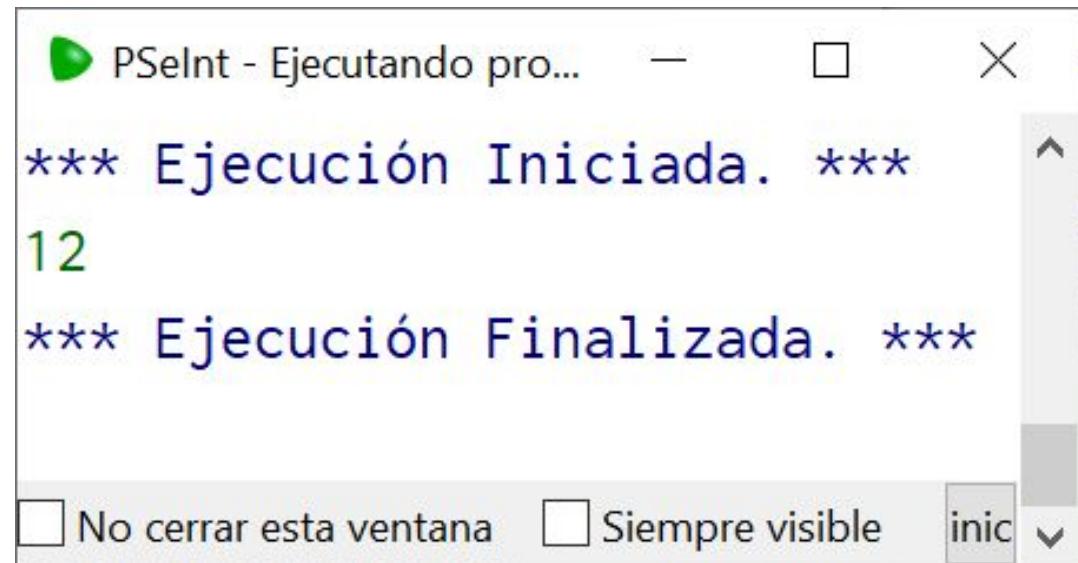
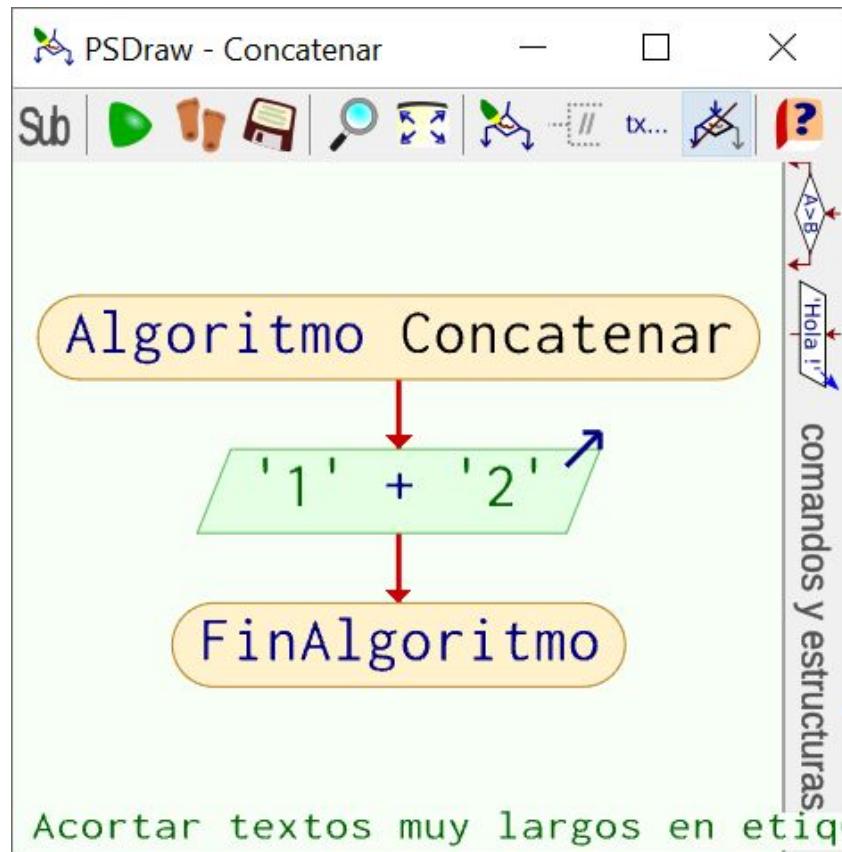
\*\*\* Ejecución Finalizada. \*\*\*

No cerrar esta ventana  Siempre visible Reiniciar

Este cuadro de diálogo muestra la ejecución del algoritmo. Se informa que la ejecución ha comenzado y finalizado. Los resultados mostrados son 11, 11 y 33.75. Al final, se presentan las opciones para no cerrar la ventana, hacerla siempre visible y reiniciar.

# Operadores y tipos

Un mismo operador se puede comportar de modo diferente en función del tipo de datos de los operandos.



# Asignación

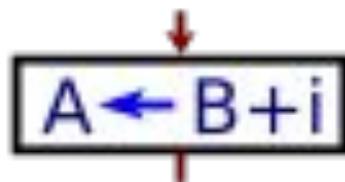
La operación de asignación permite asignar a una variable el resultado de una expresión.

La sintaxis de una asignación es la siguiente:

```
nombre_variable ← expresion
```

La expresión puede ser un valor, una expresión matemática u otra variable.

Para representar una asignación en un diagrama de flujo de PSeInt debemos utilizar el símbolo gráfico de Asignación / Dimensión / Definición:

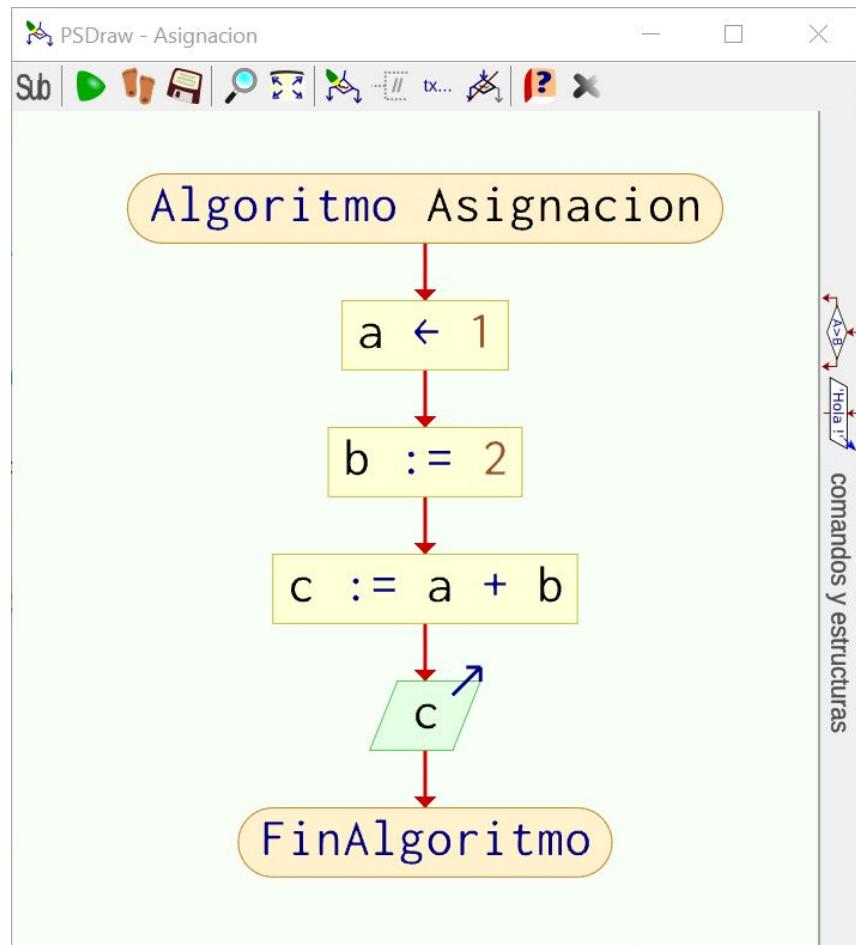


Como operador de asignación utilizaremos, indistintamente, los siguientes:

- $\leftarrow$  (se escribe tecleando un símbolo “menor que”, “<” seguido de un símbolo “menos”, “-”)
- $:=$

# Asignación

Podemos comprobar el funcionamiento del operador de asignación con el siguiente algoritmo.



PSeInt - Ejecutando proceso ASIGNAC... — X

\*\*\* Ejecución Iniciada. \*\*\*

3

\*\*\* Ejecución Finalizada. \*\*\*

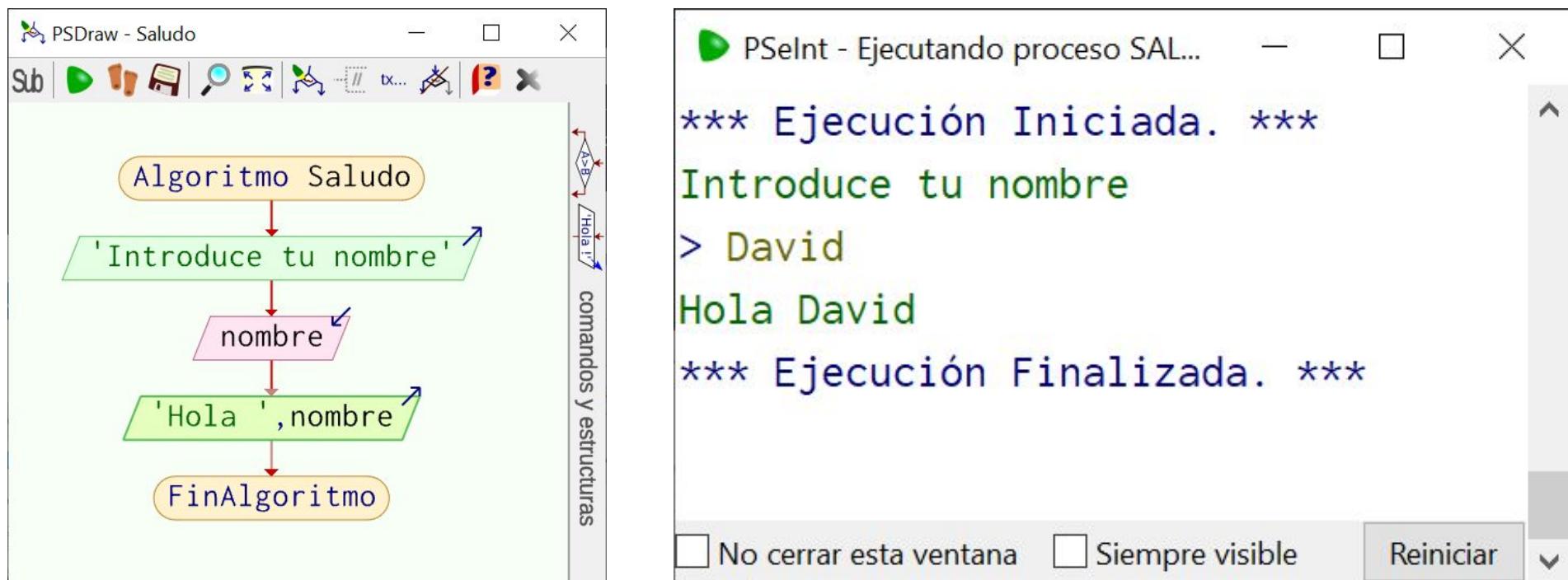
No cerrar esta ventana Siempre visible Reiniciar

Este cuadro de ejecución muestra el resultado de la ejecución del algoritmo. Se indica que la ejecución ha comenzado y finalizado exitosamente. El resultado de la ejecución es el número 3. En la parte inferior, hay opciones para no cerrar la ventana, hacerla siempre visible y reiniciar.

# Algoritmo Saludo

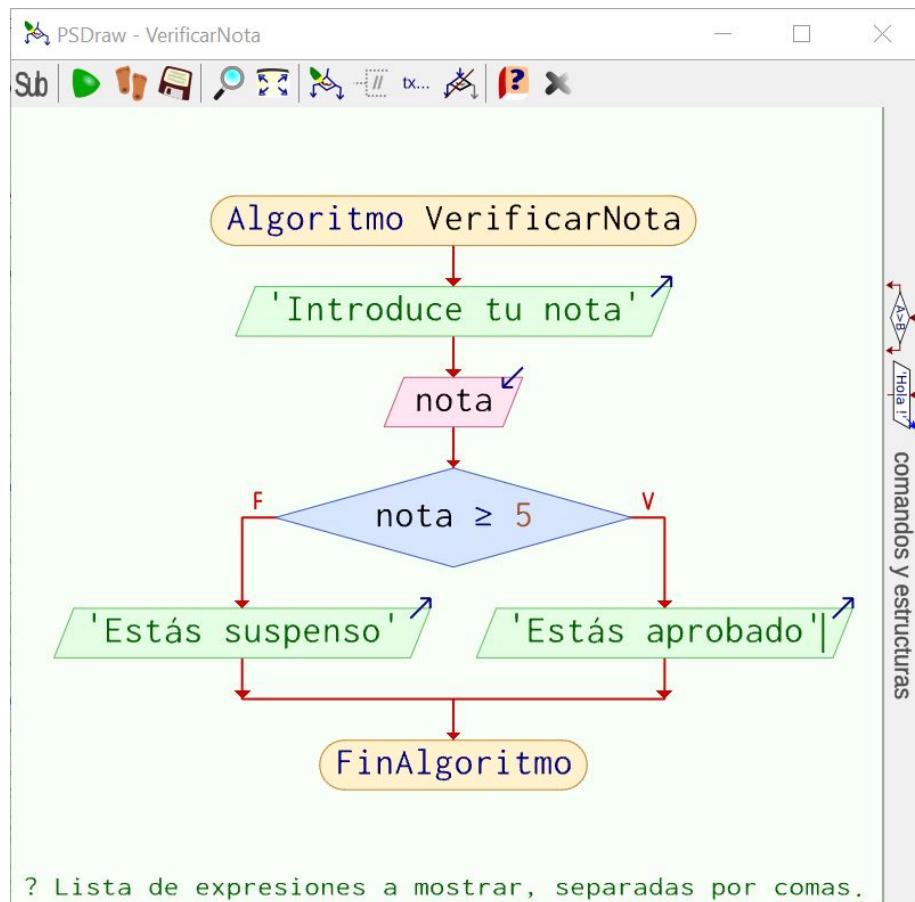
El algoritmo saludo nos permite obtener información del usuario y utilizarla en nuestro programa. Utilizamos una variable para almacenar la información proporcionada por el usuario.

Observa que el texto literal (Hola) se representa entre comillas, mientras que la variable (nombre) no.



# Algoritmo VerificarNota

El algoritmo VerificarNota utiliza una estructura selectiva **Si ... Entonces** para comprobar si se cumple o no una determinada condición (que la nota sea mayor o igual que 5) e informar si se ha aprobado o suspendido.



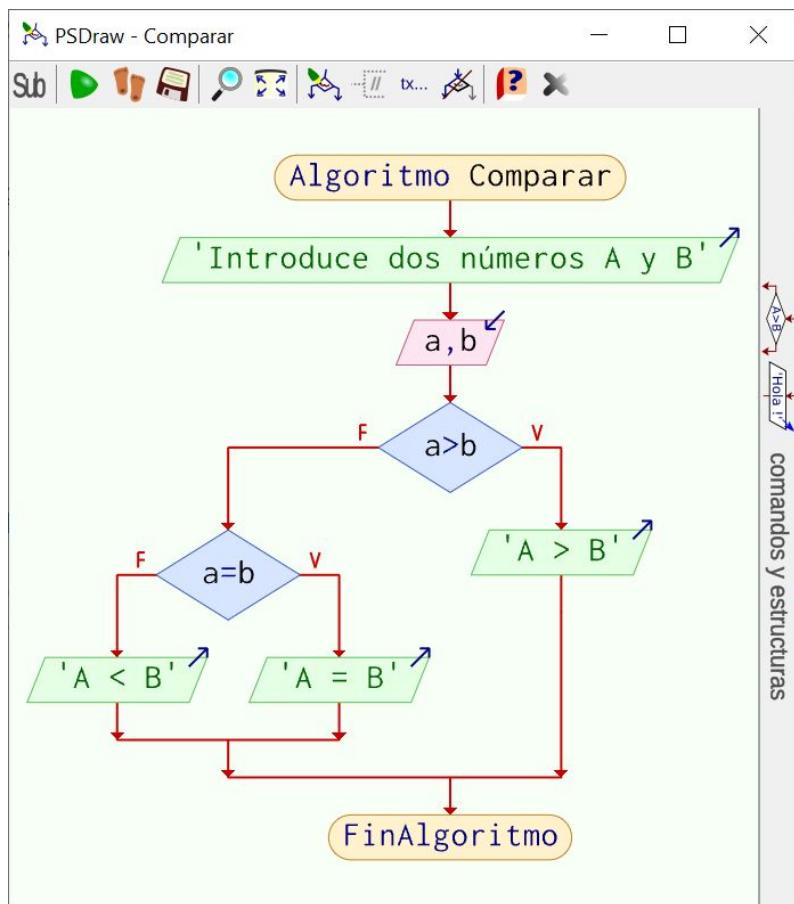
Se muestran dos capturas de pantalla de la ejecución del algoritmo:

- Ejecución 1:** Introduce la nota "4". El resultado es "Estás suspenso".
- Ejecución 2:** Introduce la nota "5". El resultado es "Estás aprobado".

Cada ejecución incluye un terminal de comandos y estructuras y una barra de control con opciones para no cerrar la ventana, hacerla siempre visible y reiniciar.

# Estructuras selectivas anidadas

Cuando nuestro algoritmo necesita realizar una selección entre más de dos opciones posibles, podemos anidar las estructuras selectivas necesarias.



```

*** Ejecución Iniciada. ***
Introduce dos números A y B
> 1
> 2
A < B
*** Ejecución Finalizada. ***

```

```

*** Ejecución Iniciada. ***
Introduce dos números A y B
> 1
> 1
A = B
*** Ejecución Finalizada. ***

```

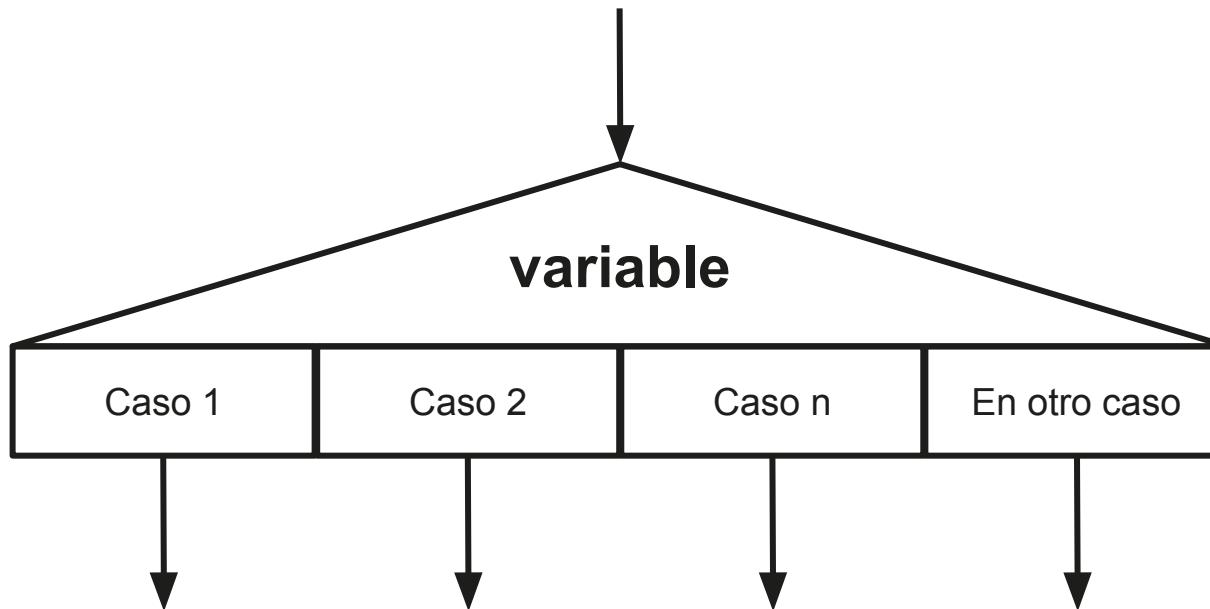
```

*** Ejecución Iniciada. ***
Introduce dos números A y B
> 2
> 1
A > B
*** Ejecución Finalizada. ***

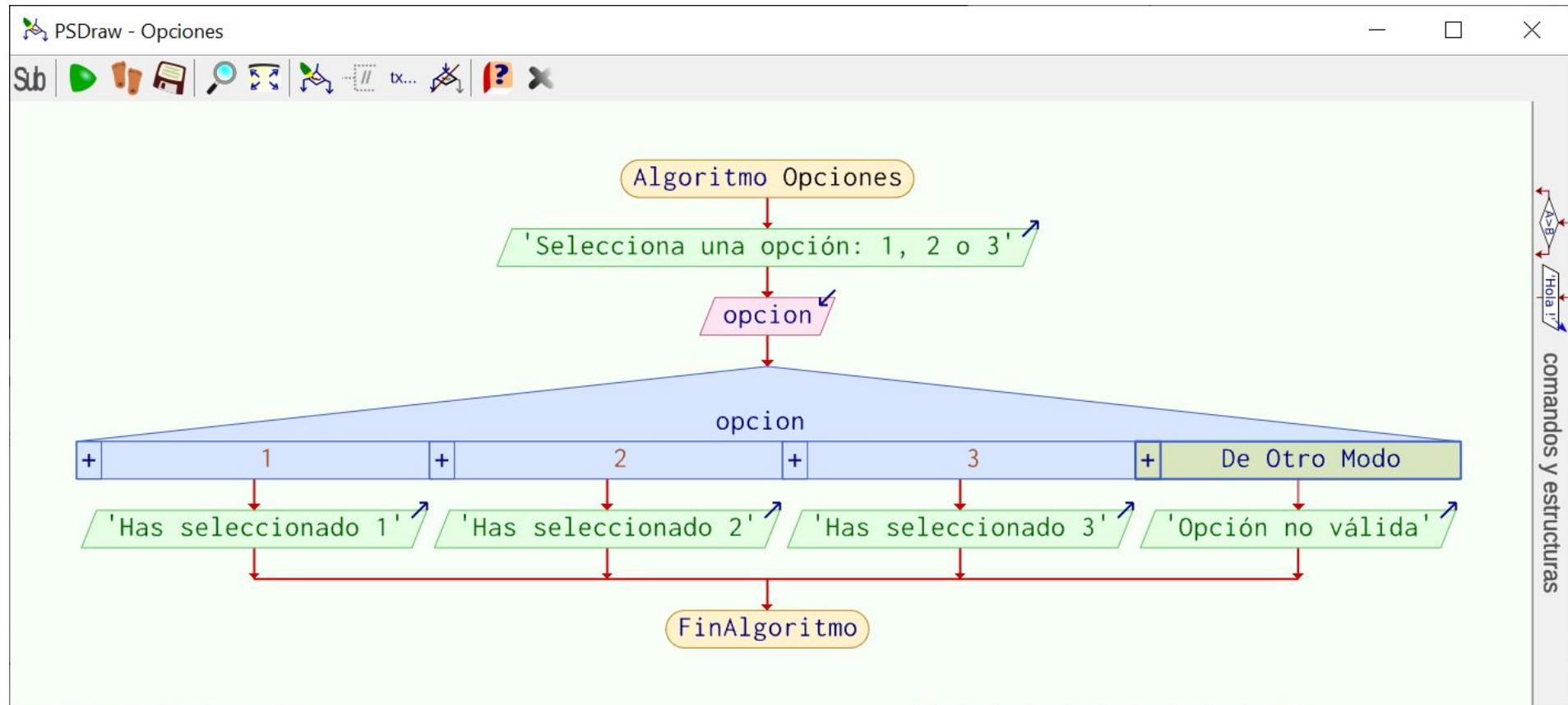
```

## Estructura selectiva Según

En el caso de que la selección múltiple dependa del valor que tome una variable, podemos utilizar una estructura **Según**.



# Algoritmo Opciones



# Algoritmo Opciones

```
PSelnt - Ejecutando proceso OPC... — ×
*** Ejecución Iniciada. ***
Selecciona una opción: 1, 2 o 3
> 1
Has seleccionado 1
*** Ejecución Finalizada. ***

 No cerrar esta ventana  Siempre visible Reiniciar ▾
```

```
PSelnt - Ejecutando proceso OPC... — ×
*** Ejecución Iniciada. ***
Selecciona una opción: 1, 2 o 3
> 3
Has seleccionado 3
*** Ejecución Finalizada. ***

 No cerrar esta ventana  Siempre visible Reiniciar ▾
```

```
PSelnt - Ejecutando proceso OPC... — ×
*** Ejecución Iniciada. ***
Selecciona una opción: 1, 2 o 3
> 2
Has seleccionado 2
*** Ejecución Finalizada. ***

 No cerrar esta ventana  Siempre visible Reiniciar ▾
```

```
PSelnt - Ejecutando proceso OPC... — ×
*** Ejecución Iniciada. ***
Selecciona una opción: 1, 2 o 3
> 4
Opción no válida
*** Ejecución Finalizada. ***

 No cerrar esta ventana  Siempre visible Reiniciar ▾
```

## Ejercicio individual

### Representación gráfica de algoritmos

Representa, mediante un diagrama de flujo en PSeInt, un algoritmo que solicite al usuario que introduzca un valor numérico que representa su nota y le devuelva su calificación según la siguiente tabla:

- $0 \leq \text{nota} < 5$ : Suspensos
- $5 \leq \text{nota} < 7$ : Aprobado
- $7 \leq \text{nota} < 9$ : Notable
- $9 \leq \text{nota} \leq 10$ : Sobresaliente

Ten en cuenta que el usuario puede introducir un valor con decimales (Ej.: 7.5)

# Ejercicio individual

## Representación gráfica de algoritmos

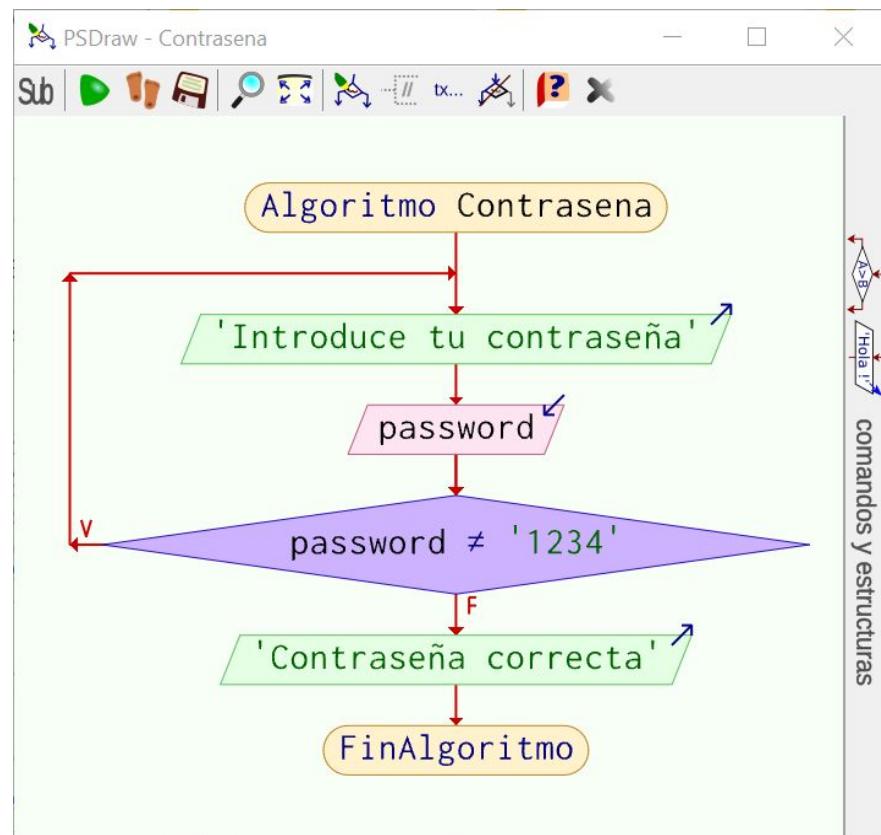
Representa, mediante un diagrama de flujo, un algoritmo que solicite al usuario que introduzca un número del uno al siete y que devuelva el día de la semana que le corresponde, según la siguiente tabla:

- 1: Lunes
- 2: Martes
- 3: Miércoles
- 4: Jueves
- 5: Viernes
- 6: Sábado
- 7: Domingo

En caso de que el usuario introduzca un número que no esté en en rango 1-7, el algoritmo debe mostrar el texto “Número no válido”.

# Algoritmo Contraseña

El algoritmo Contraseña utiliza una estructura iterativa **Repetir ... mientras**, que repite la solicitud de la contraseña al usuario mientras esta sea incorrecta.



PSelnt - Ejecutando proceso CO...

\*\*\* Ejecución Iniciada. \*\*\*

Introduce tu contraseña  
> 1235  
Introduce tu contraseña  
> 1234  
Contraseña correcta

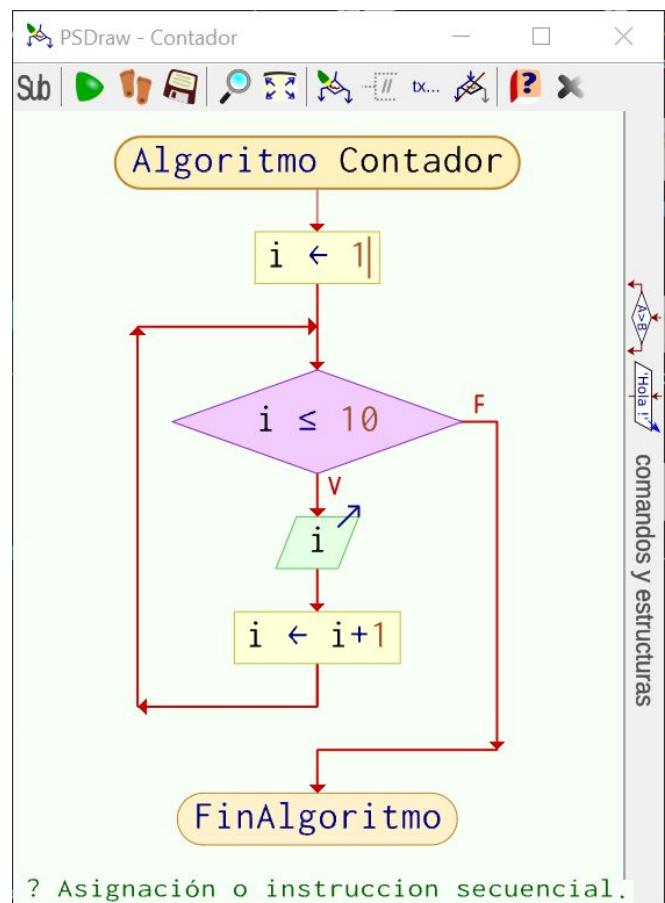
\*\*\* Ejecución Finalizada. \*\*\*

No cerrar esta ventana  Siempre visible Reiniciar

Este cuadro de ejecución muestra la interacción entre el usuario y el programa. Se presentan instrucciones para introducir la contraseña, lo cual se repite dos veces con los valores '1235' y '1234'. La respuesta '1234' es considerada correcta ('Contraseña correcta'). Una vez que se cumple la condición de la estructura iterativa, el programa finaliza ('\*\*\* Ejecución Finalizada. \*\*\*').

# Algoritmo Contador

El algoritmo Contador utiliza una estructura iterativa **Mientras** que incrementa el valor de una variable mientras esta sea menor o igual que una cierta cantidad.



Captura de pantalla de la ejecución del algoritmo en PSeInt:

\*\*\* Ejecución Iniciada. \*\*\*

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

\*\*\* Ejecución Finalizada. \*\*\*

Comandos y estructuras

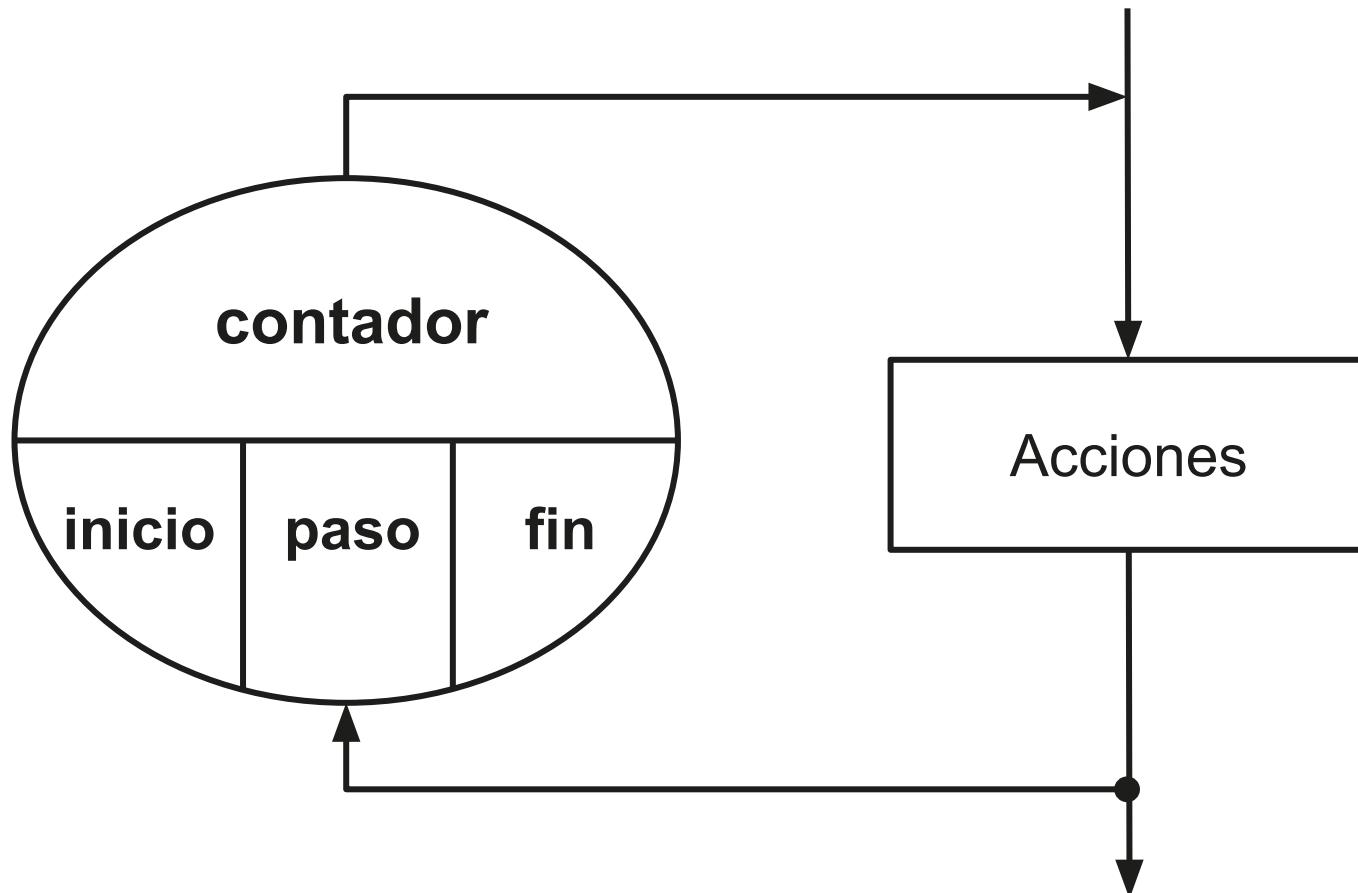
PSelnt - Ejecutando proceso CO...

Reiniciar

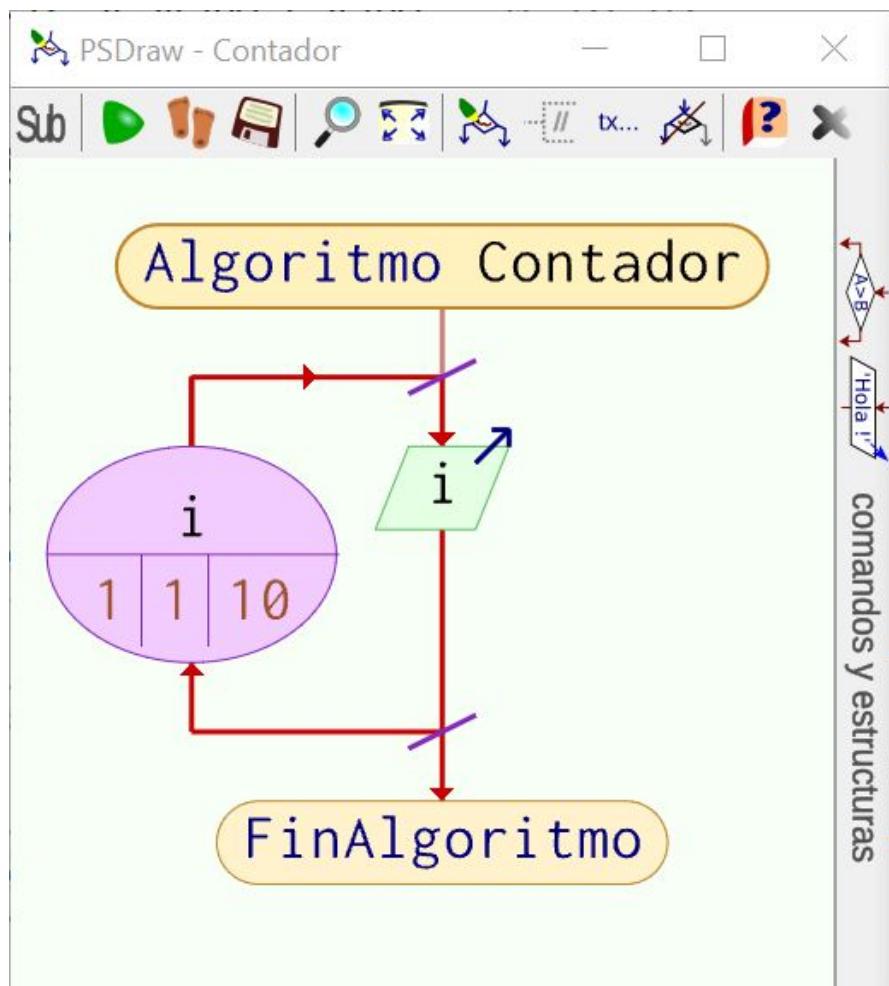
Este cuadro de diálogo muestra el resultado de la ejecución del algoritmo. Se observan los números 1 a 10, lo que indica que el bucle se ejecutó 10 veces. La ejecución comenzó y finalizó exitosamente.

## Estructura iterativa Para

Cuando el **número de iteraciones es conocido**, o cuando deseamos **iterar entre unos límites establecidos**, existe una forma simplificada de representar la iteración: la estructura iterativa **Para**.



# Algoritmo Contador utilizando la estructura Para



PSelnt - Ejecutando proceso CO...

\*\*\* Ejecución Iniciada. \*\*\*

1  
2  
3  
4  
5  
6  
7  
8  
9  
10

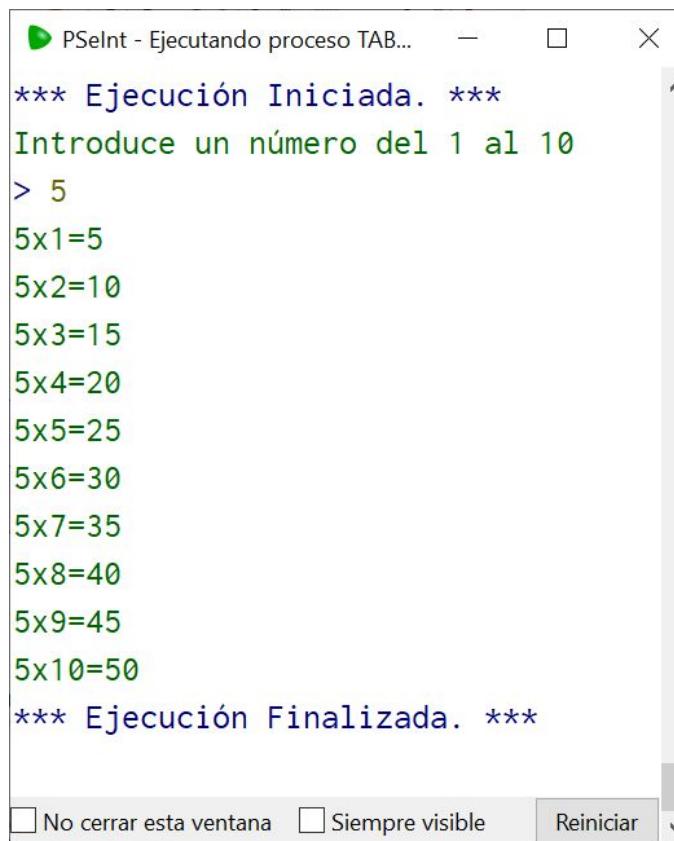
\*\*\* Ejecución Finalizada. \*\*\*

No cerrar esta ventana  Siempre visible  Reiniciar

# Ejercicio individual

## Representación gráfica de algoritmos

Representa, mediante un diagrama de flujo, un algoritmo que pregunte al usuario un número del 1 al 10 y devuelva la tabla de multiplicar de dicho número, tal y como se muestra en la siguiente captura:



The screenshot shows a terminal window titled "PSelnt - Ejecutando proceso TAB...". The window displays the following text:  
\*\*\* Ejecución Iniciada. \*\*\*  
Introduce un número del 1 al 10  
> 5  
5x1=5  
5x2=10  
5x3=15  
5x4=20  
5x5=25  
5x6=30  
5x7=35  
5x8=40  
5x9=45  
5x10=50  
\*\*\* Ejecución Finalizada. \*\*\*

At the bottom of the window, there are three checkboxes: "No cerrar esta ventana", "Siempre visible", and "Reiniciar".

# Representación textual de algoritmos



# Representación textual de algoritmos

Además de la representación gráfica, los algoritmos también se pueden representar de forma textual mediante el denominado **lenguaje algorítmico, pseudolenguaje o pseudocódigo**.

La representación textual nos permite describir **algoritmos más complejos y con mayor detalle** que la representación gráfica.

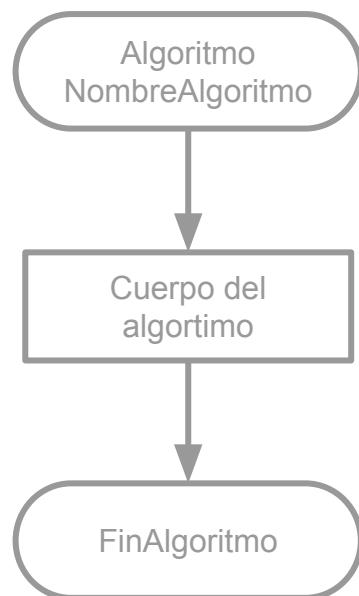
**No existe un estándar** generalmente aceptado para el pseudocódigo, sino que hay diversas propuestas realizadas por diferentes autores y organismos.

PSelInt permite generar automáticamente pseudocódigo a partir del diagrama de flujo y viceversa, de modo que en este modo utilizaremos el pseudocódigo utilizado por PSelInt.

Comenzaremos presentando las estructuras de pseudocódigo que equivalen a los símbolos ya presentados para los diagramas de flujo.

## Inicio y fin

Representaremos el **inicio** y **fin** de nuestro algoritmo mediante el uso de las palabras clave **Algoritmo** *NombreAlgoritmo* y **FinAlgoritmo**, que equivalen a los símbolos terminales de los diagramas de flujo.



**Algoritmo** *NombreAlgoritmo*

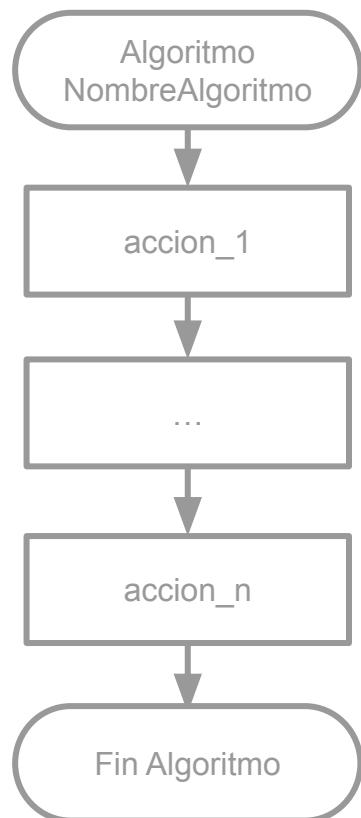
// Cuerpo del algoritmo

**FinAlgoritmo**

# Estructura secuencial

En una representación textual, las acciones de un algoritmo se escriben **secuencialmente de arriba abajo**.

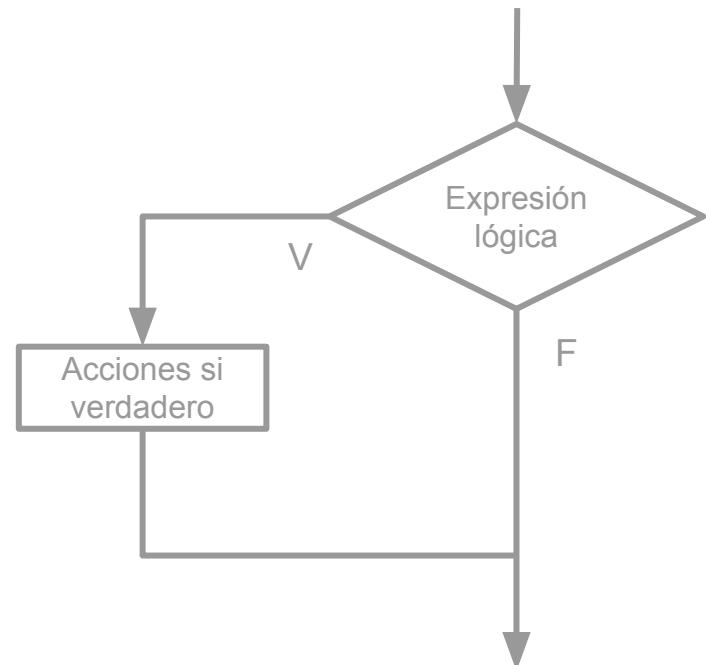
Utilizaremos la **sangría o indentación** para facilitar la lectura de estructuras anidadas.



```
Algoritmo NombreAlgoritmo
accion_1
accion_2
...
accion_n
FinAlgoritmo
```

# Estructura selectiva simple

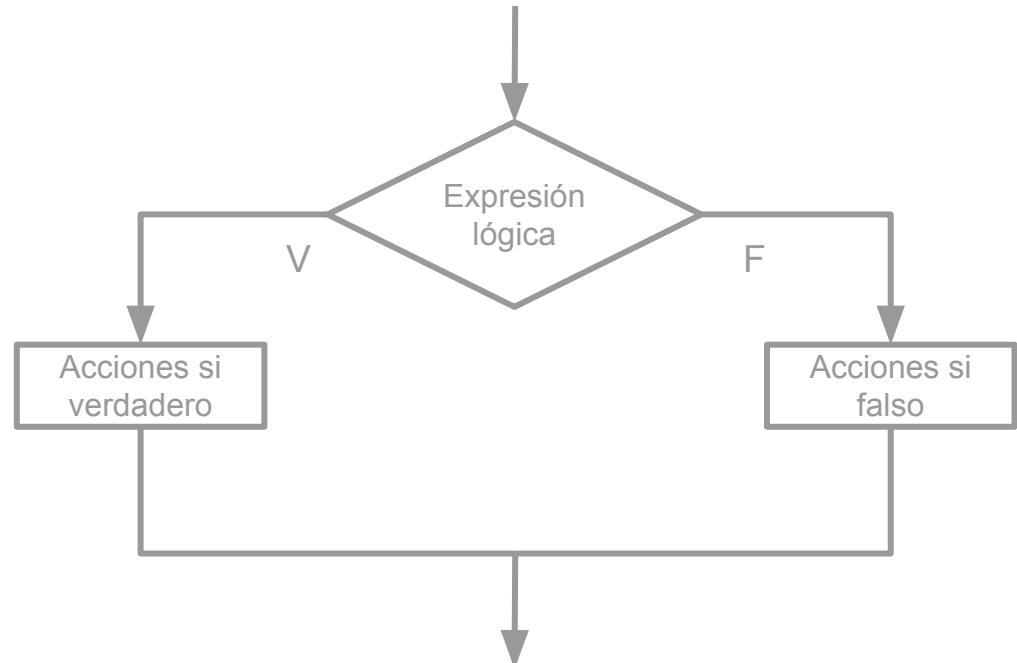
Representaremos la **estructura selectiva simple** mediante las palabras clave **Si** y **FinSi**.



```
Si expresion_logica Entonces  
acciones_si_verdadero  
FinSi
```

# Estructura selectiva doble o alternativa

Representaremos la **estructura selectiva doble** mediante las palabras clave **si**, **SiNo** y **FinSi**.

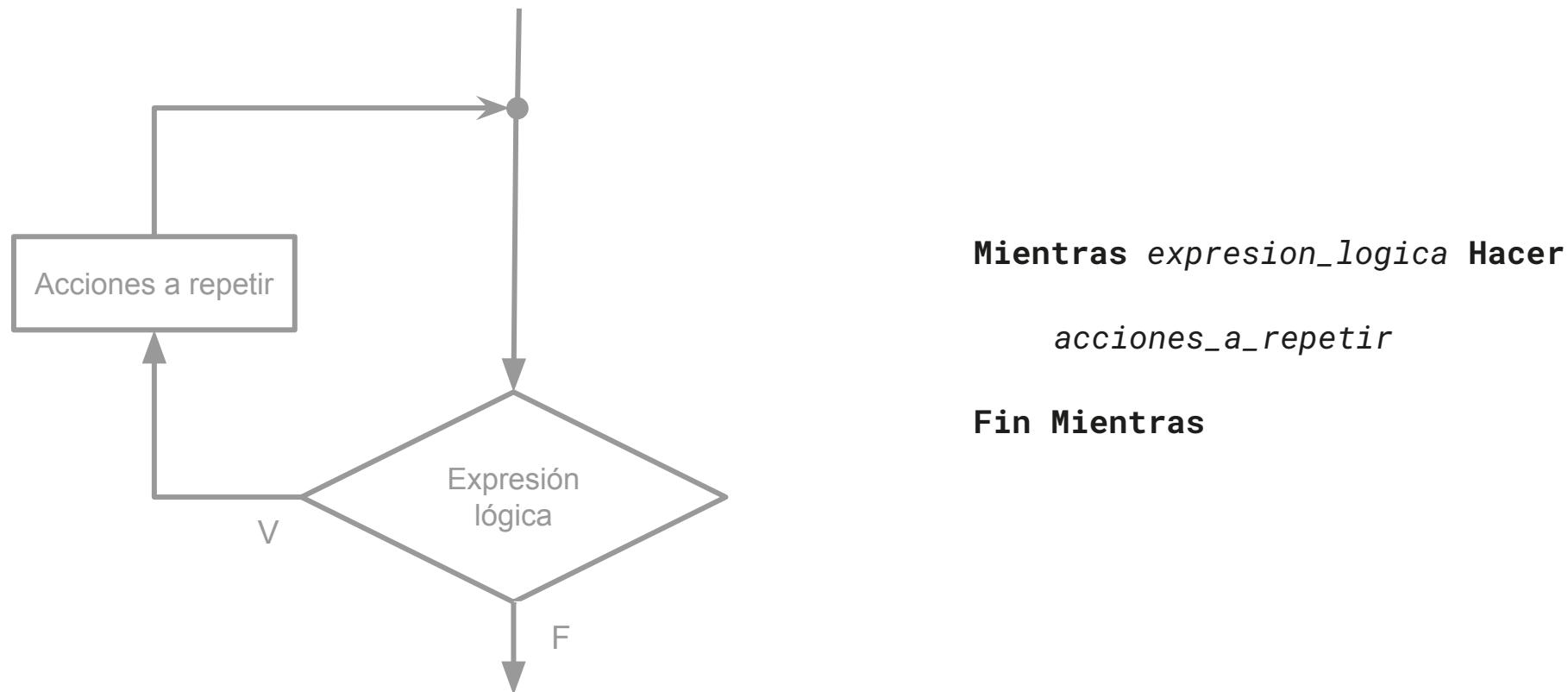


```
Si expresion_logica Entonces  
    acciones_si_verdadero  
  
SiNo  
    acciones_si_falso  
  
FinSi
```

## Estructura iterativa Mientras

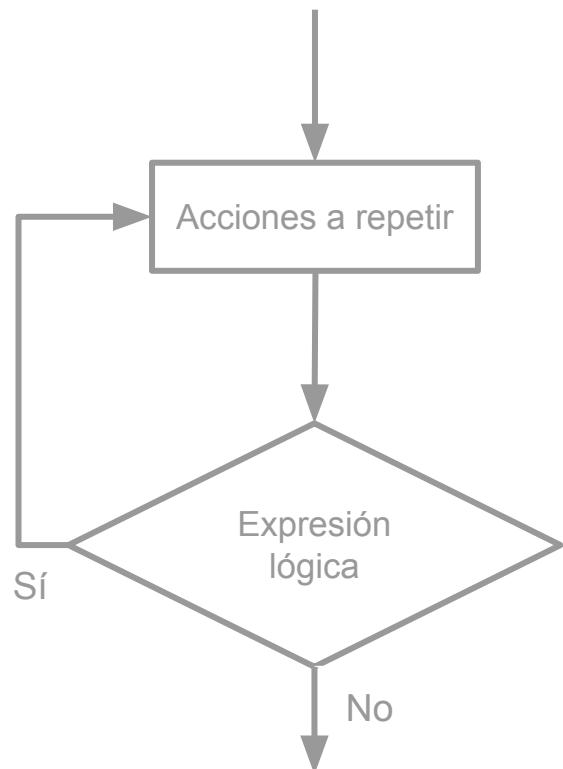
En una estructura iterativa **Mientras** se procesan repetidamente las acciones mientras la condición sea verdadera.

**Si inicialmente la condición es falsa, las acciones no se procesan ninguna vez.**



## Estructura iterativa Repetir

En una estructura iterativa **Repetir** se procesan las acciones al **menos una vez** y se repite mientras la condición sea cierta.



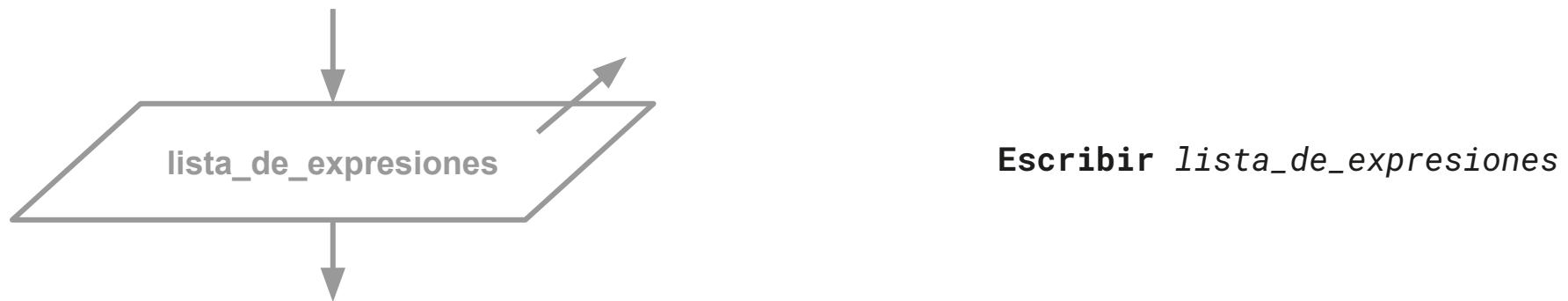
**Repetir**

*acciones\_a\_repetir*

**Mientras Que expresión\_lógica**

# Escribir

La palabra reservada **Escribir** permite **mostrar información** en la pantalla.



## Leer

La palabra reservada **Leer** permite **obtener información** del usuario a través del teclado y asignarla a variables.



# Comentarios

Los comentarios son anotaciones que **el procesador no tiene en cuenta**.

Representaremos los comentarios comenzando los mismos con una doble barra (//).



# Representación textual de algoritmos con PSelnt



# Algoritmo HolaMundo

The image shows two side-by-side software windows. The left window is titled "PSDraw - HolaMundo" and displays a flowchart for the "Algoritmo HolaMundo". The flowchart consists of three rounded rectangular boxes: the top box is labeled "Algoritmo HolaMundo", the middle box contains the text "'Hola, mundo'", and the bottom box is labeled "FinAlgoritmo". Arrows indicate a flow from the top box down to the middle, and from the middle down to the bottom. To the right of the flowchart is a vertical bar with the text "comandos y estructuras" and a small icon. The right window is titled "PSeInt" and shows a pseudocode editor with a file named "hola\_mundo.psc\*". The pseudocode is as follows:

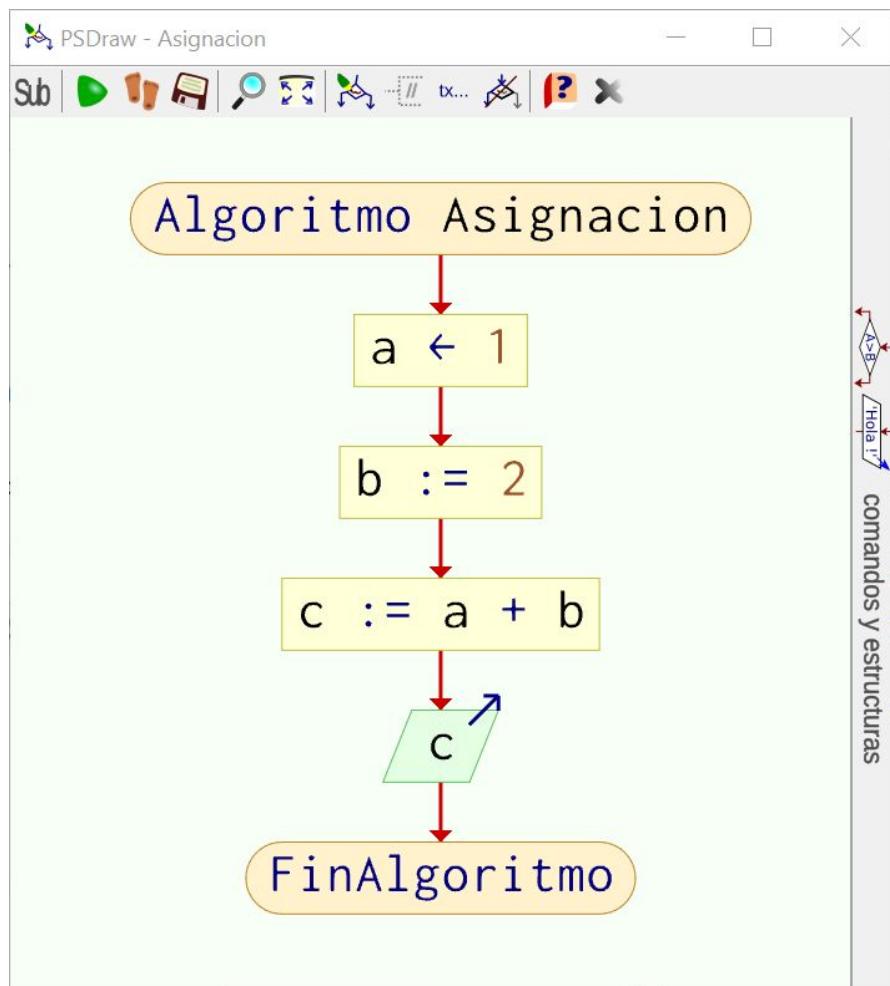
```
1 Algoritmo HolaMundo
2   Escribir "Hola, mundo"
3 FinAlgoritmo
```

The PSeInt interface includes a toolbar with various icons, a menu bar with "Archivo", "Editar", "Configurar", "Ejecutar", and "Ayuda", and a status bar at the bottom.

**Cierre el editor, preguntando antes si se**

**El pseudocódigo ha cambiado. Presione F9 para ver los cambios**

# Asignación



PSelnt

Archivo Editar Configurar Ejecutar Ayuda

asignacion.psc\*

```

1 Algoritmo Asignacion
2   a <- 1
3   b := 2
4   c := a + b
5   Escribir c
6 FinAlgoritmo
7

```

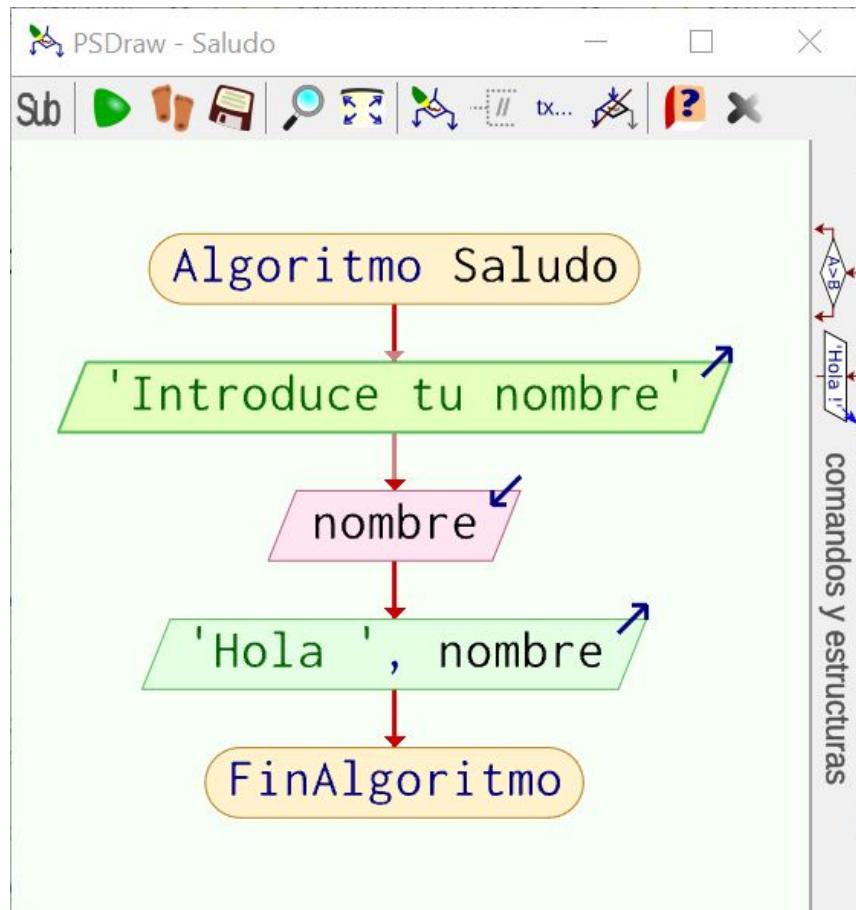
Lista de Variables \*+=< Operac

Comandos y Estructuras

La ejecución ha finalizado sin errores.

Este captura de pantalla muestra el editor de PSeInt con el archivo "asignacion.psc". El código fuente es idéntico al diagrama de flujo anterior. Se incluyen paneles para "Lista de Variables", "Operac" y "Comandos y Estructuras". Una barra de status informa que la ejecución ha finalizado sin errores.

# Algoritmo Saludo



Ejecución del algoritmo "saludo.psc" en PSelnt:

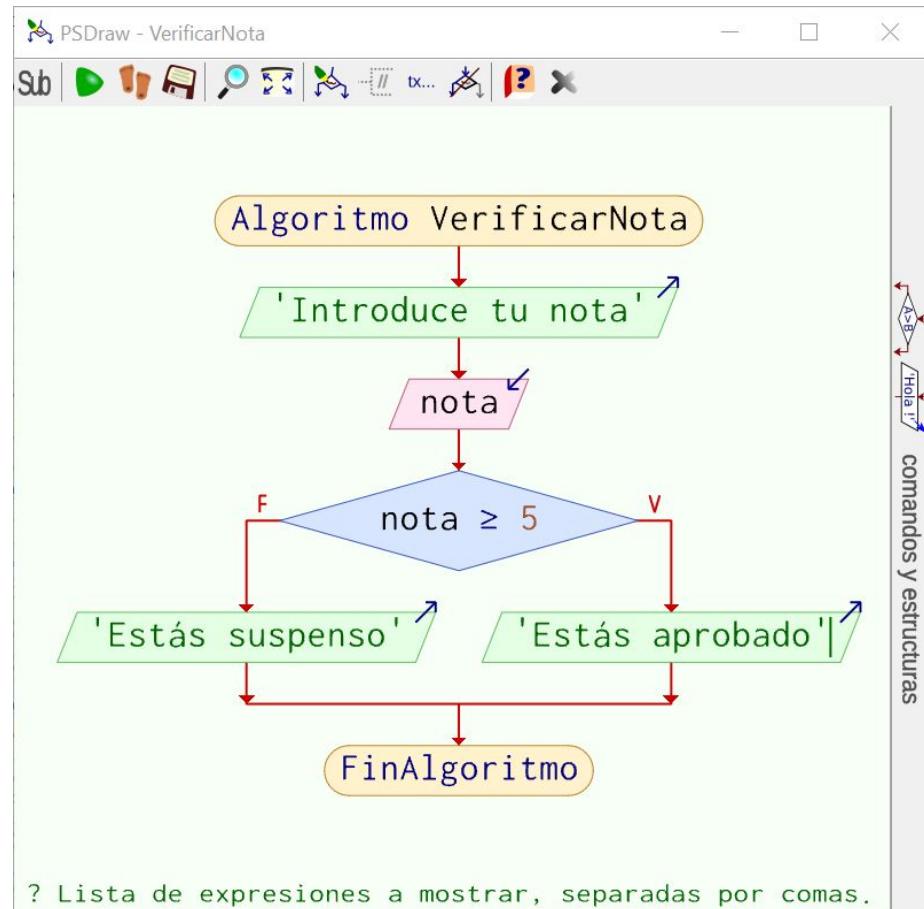
```

1 Algoritmo Saludo
2   Escribir 'Introduce tu nombre'
3   Leer nombre
4   Escribir 'Hola ', nombre
5 FinAlgoritmo
6
    
```

La ejecución ha finalizado sin errores.

Este cuadro de diálogo muestra el código fuente del algoritmo "saludo.psc". El código define un algoritmo que imprime "Introduce tu nombre", lee el nombre del usuario y luego imprime "Hola ", nombre. La ejecución se completó sin errores.

# Algoritmo VerificarNota

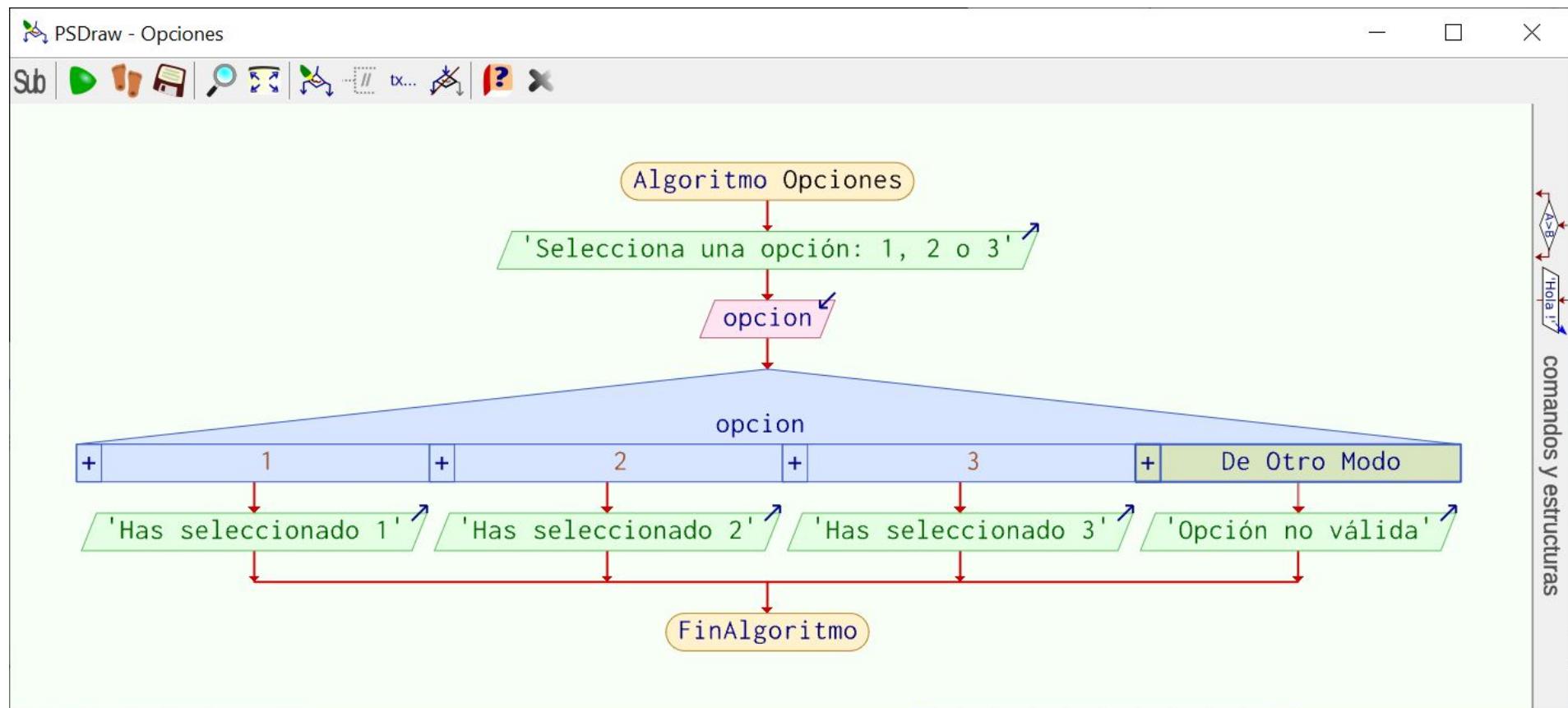


```

1 Algoritmo VerificarNota
2   Escribir 'Introduce tu nota'
3   Leer nota
4   Si nota ≥ 5 Entonces
5     Escribir 'Estás aprobado'
6   SiNo
7     Escribir 'Estás suspenso'
8   FinSi
9 FinAlgoritmo
10
    
```

La ejecución ha finalizado sin errores.

# Algoritmo Opciones



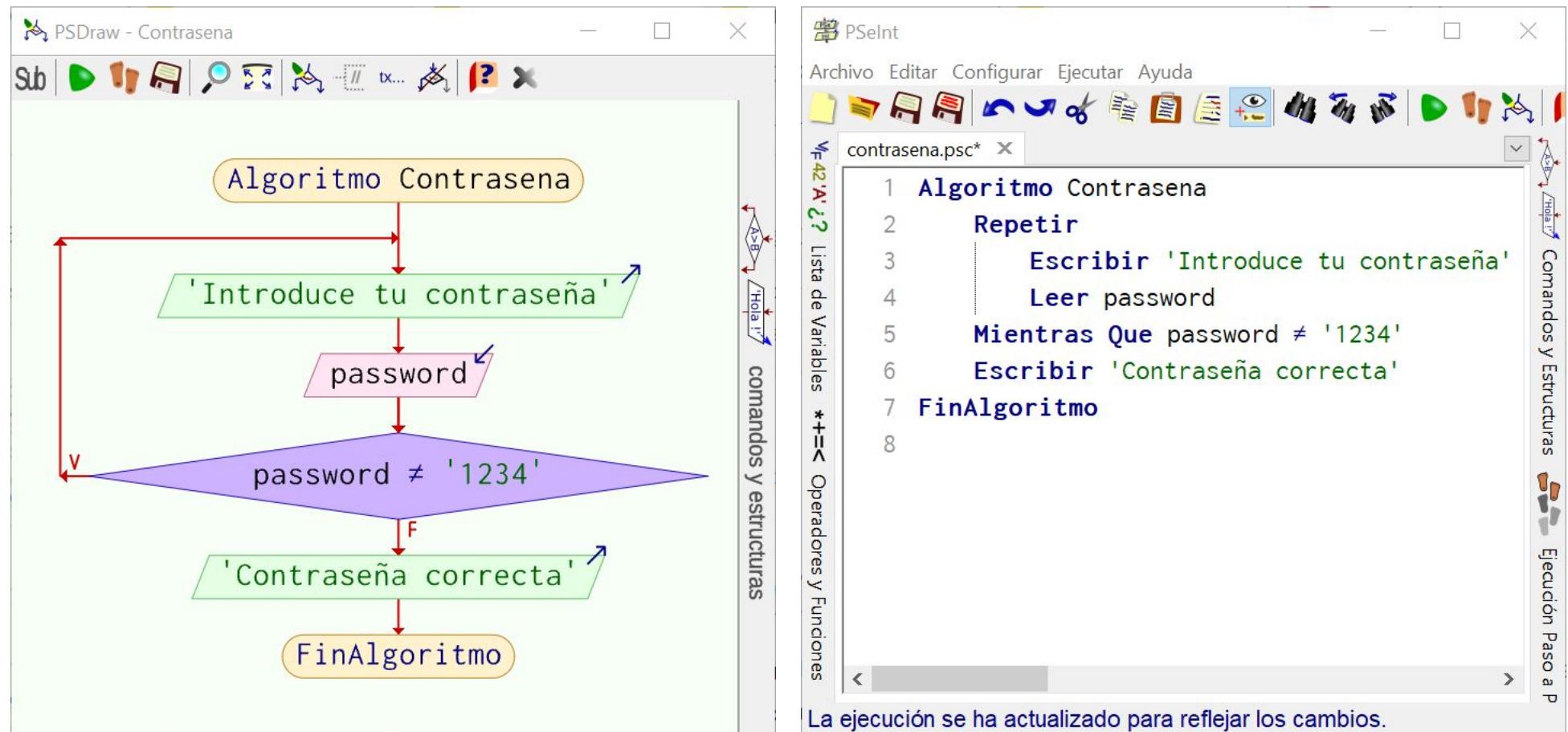
# Algoritmo Opciones

The screenshot shows the PSeInt programming environment. The window title is "PSeInt". The menu bar includes "Archivo", "Editar", "Configurar", "Ejecutar", and "Ayuda". The toolbar contains various icons for file operations like new, open, save, cut, copy, paste, and search. The status bar at the bottom displays the message "La ejecución ha finalizado sin errores." (Execution has ended without errors).

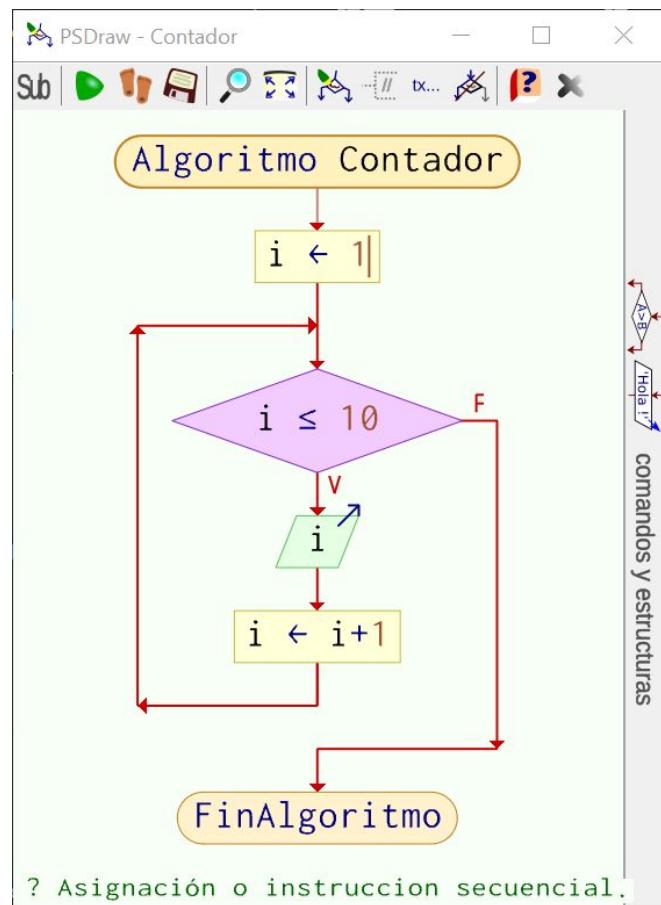
```
1 Algoritmo Opciones
2   Escribir 'Selecciona una opción: 1, 2 o 3'
3   Leer opcion
4   Segun opcion Hacer
5     1:
6       Escribir 'Has seleccionado 1'
7     2:
8       Escribir 'Has seleccionado 2'
9     3:
10    Escribir 'Has seleccionado 3'
11    De Otro Modo:
12      Escribir 'Opción no válida'
13    FinSegun
14  FinAlgoritmo
15
```

The code implements a selection structure where the user is prompted to choose between three options (1, 2, or 3). If the input matches one of the options, a message is displayed indicating the selection. If the input does not match any of the options, a message stating "Opción no válida" (Invalid option) is displayed.

# Algoritmo Contraseña



# Algoritmo Contador (estructura Mientras)



PSeInt

Archivo Editar Configurar Ejecutar Ayuda

contraseña.psc\* contador.psc\* x

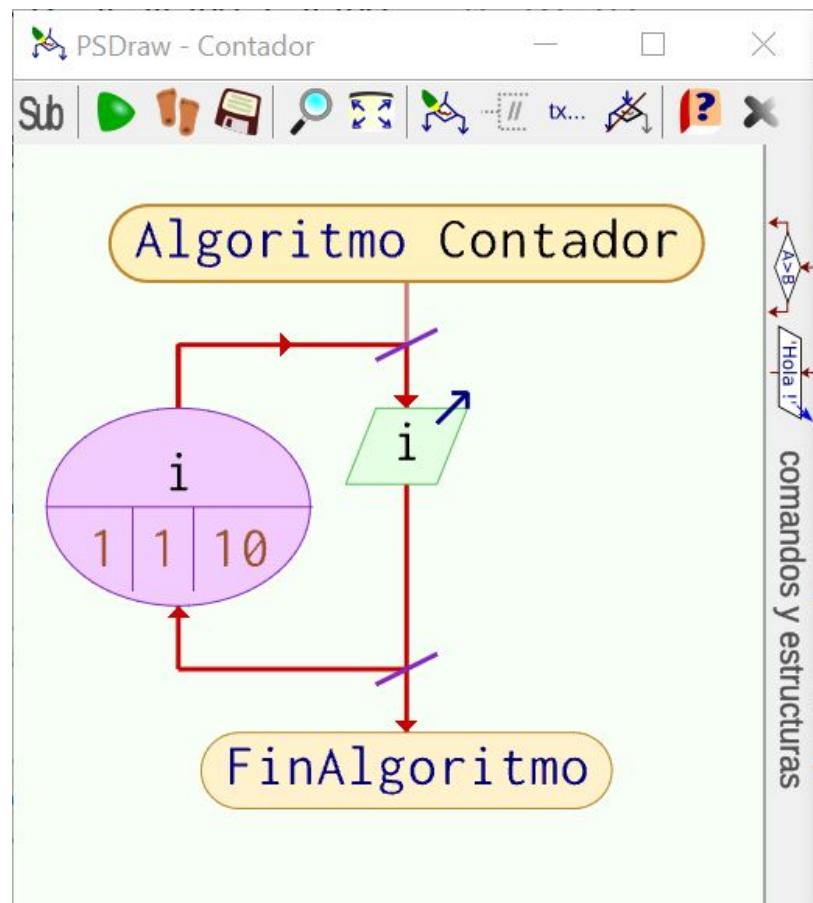
```

1 Algoritmo Contador
2   i ← 1
3   Mientras i≤10 Hacer
4     Escribir i
5     i ← i+1
6   FinMientras
7 FinAlgoritmo
8
    
```

Comandos y Estructuras

La ejecución se ha actualizado para reflejar los car

# Algoritmo Contador (estructura Para)



PSeInt

Archivo Editar Configurar Ejecutar Ayuda

contraseña.psc\* contador.psc\* contador\_pa.psc

```

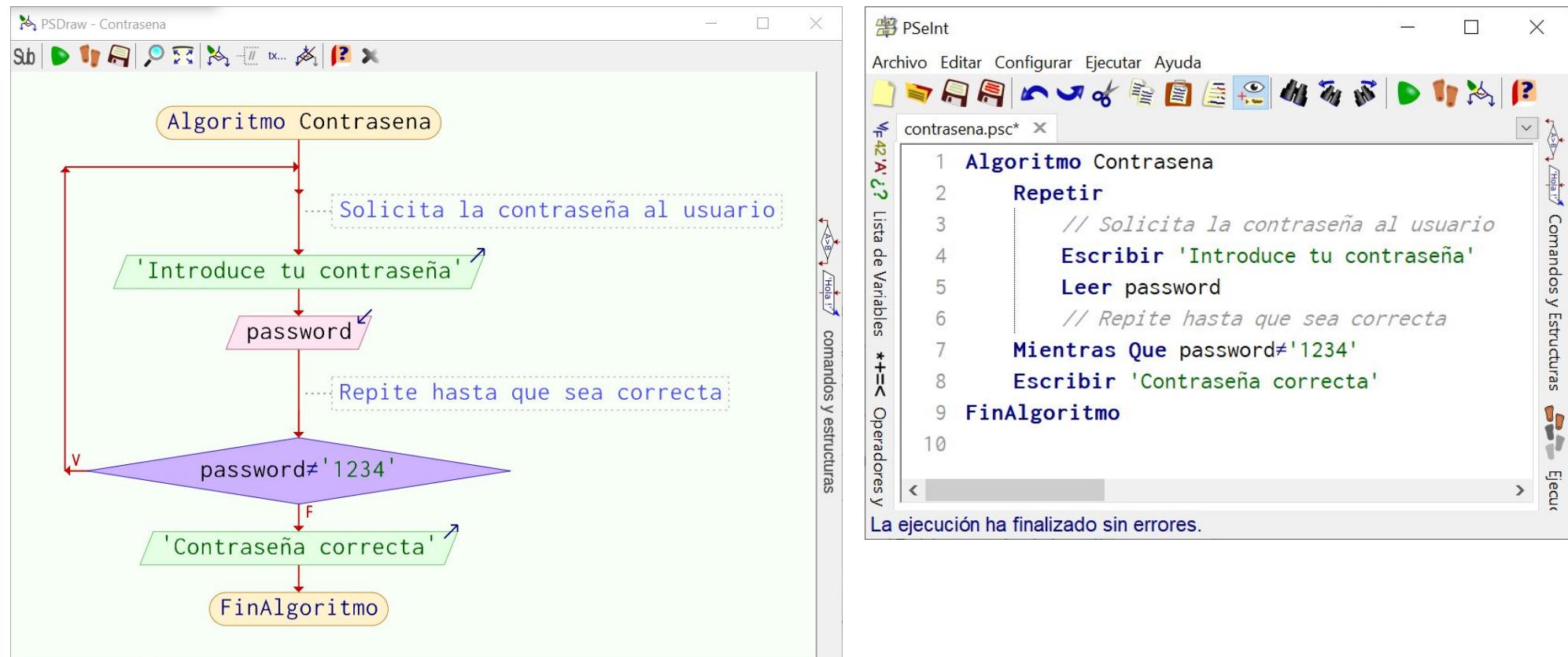
1 Algoritmo Contador
2 Para i<1 Hasta 10 Hacer
3     Escribir i
4 FinPara
5 FinAlgoritmo
6
    
```

Lista de Variables \*+:

Comandos y Estructur.

La ejecución ha finalizado sin errores.

# Comentarios



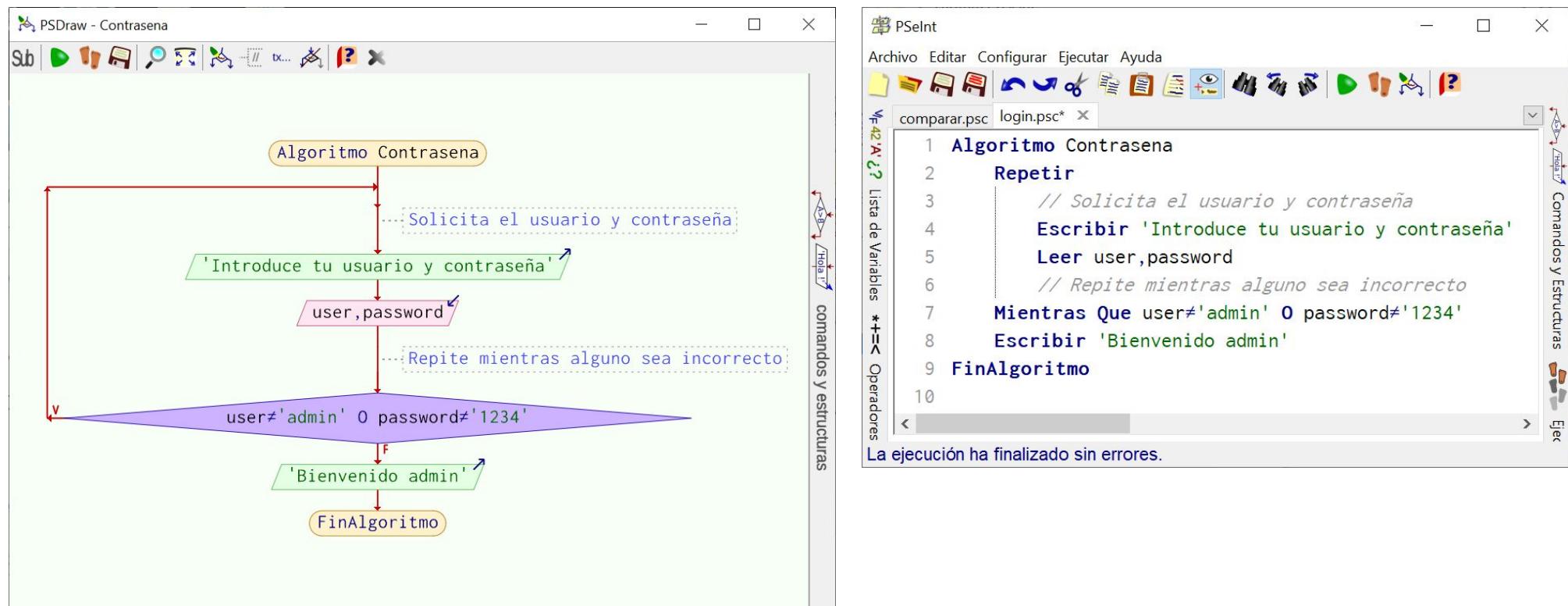
# Operadores lógicos

Los operadores lógicos o booleanos son la **negación** (NO), el “**Y**” lógico, y el “**O**” lógico, y se aplican únicamente sobre valores booleanos.

Los operadores lógicos pueden ser **unarios**, cuando operan sobre un único valor, como por ejemplo la negación (NO); o **binarios**, cuando operan sobre dos valores, como por ejemplo el “Y” lógico ( $a \text{ Y } b$ ).

<b>a</b>	<b>b</b>	<b>NO a</b>	<b>a Y b</b>	<b>a O b</b>
<b>FALSO</b>	<b>FALSO</b>	VERDADERO	FALSO	FALSO
<b>FALSO</b>	<b>VERDADERO</b>	VERDADERO	FALSO	VERDADERO
<b>VERDADERO</b>	<b>FALSO</b>	FALSO	FALSO	VERDADERO
<b>VERDADERO</b>	<b>cierto</b>	FALSO	VERDADERO	VERDADERO

# Operadores lógicos



# Ejercicio individual

## Representación textual de algoritmos

Representa mediante pseudocódigo un algoritmo que solicite al usuario que introduzca su altura en m y su peso en kg y que calcule su índice de masa corporal según la siguiente fórmula:

$$\text{IMC} = \frac{\text{Peso}}{\text{Altura}^2}$$

En función del índice de masa corporal calculado, el algoritmo debe devolver una de las siguientes salidas:

IMC	SALIDA A MOSTRAR
< 18,5	Peso bajo
18,5 ≤ IMC < 25	Peso normal
IMC ≥ 25	Peso alto

# Funciones y procedimientos



# Funciones y procedimientos

Las **funciones** y los **procedimientos** son **subalgoritmos** que ejecutan una secuencia específica de acciones.

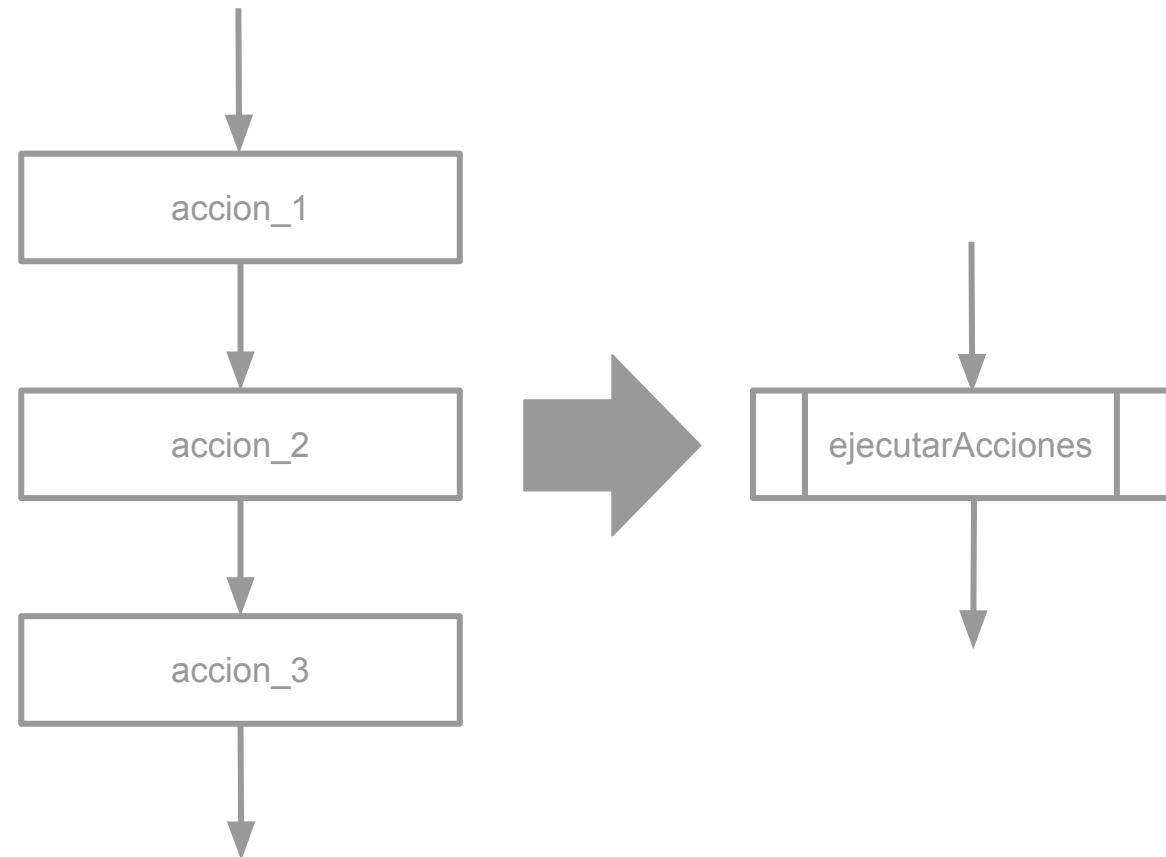
De este modo podemos crear una función que realice una tarea específica e **invocarla** desde nuestro algoritmo cada vez que necesitemos realizar dicha tarea.

La diferencia entre una función y un procedimiento es que la **función devuelve un valor de salida**, mientras que el procedimiento no.

En este módulo **consideraremos los procedimientos como un tipo específico de funciones** que no devuelven ningún valor, y nos referiremos a ambos como funciones.

Las funciones y los procedimientos nos permiten **reutilizar código** y que nuestros programas sean más legibles y fáciles de mantener.

# Funciones y procedimientos



**Funcion ejecutarAcciones()**

accion\_1

accion\_2

accion\_3

**FinFuncion**

# Función sin parámetros ni retorno

The image shows two windows from the PSeInt IDE. The left window is the code editor with the file 'imprimir\_saludo.psc' open. It contains the following pseudocode:

```

1 Funcion imprimirSaludo()
2   Escribir 'Hola'
3 FinFuncion
4
5 Algoritmo Saludar
6   Escribir 'Introduce el nombre del usuario 1'
7   Leer usuario_1
8   imprimirSaludo()
9   Escribir 'Introduce el nombre del usuario 2'
10  Leer usuario_2
11  imprimirSaludo()
12
13 FinAlgoritmo
  
```

The status bar at the bottom of the editor says: "La ejecución ha finalizado sin errores."

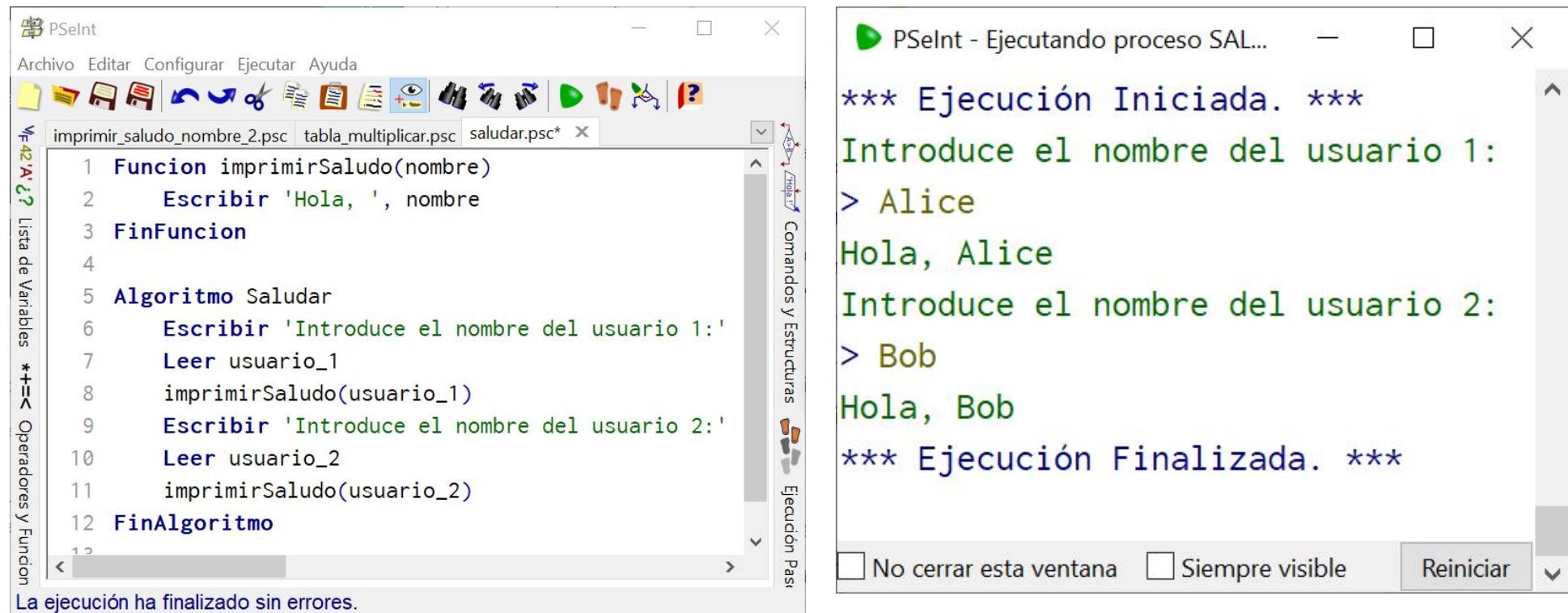
The right window is a terminal window titled "PSeInt - Ejecutando proceso SAL...". It displays the following output:

```

*** Ejecución Iniciada. ***
Introduce el nombre del usuario 1:
> Alice
Hola
Introduce el nombre del usuario 2:
> Bob
Hola
*** Ejecución Finalizada. ***
  
```

At the bottom of the terminal window, there are three checkboxes: "No cerrar esta ventana", "Siempre visible", and a "Reiniciar" button.

# Función con parámetros sin retorno



The screenshot displays the PSeInt Integrated Development Environment (IDE) and its execution window.

**IDE Window (Left):**

- Title Bar:** PSeInt
- Menu Bar:** Archivo, Editar, Configurar, Ejecutar, Ayuda
- Toolbar:** Includes icons for New, Open, Save, Print, Run, Stop, and Help.
- Project Explorer:** Shows files: imprimir\_saludo\_nombre\_2.psc, tabla\_multiplicar.psc, and saladar.psc\*.
- Code Editor:**

```

1 Funcion imprimirSaludo(nombre)
2   Escribir 'Hola, ', nombre
3 FinFuncion
4
5 Algoritmo Saludar
6   Escribir 'Introduce el nombre del usuario 1:'
7   Leer usuario_1
8   imprimirSaludo(usuario_1)
9   Escribir 'Introduce el nombre del usuario 2:'
10  Leer usuario_2
11  imprimirSaludo(usuario_2)
12 FinAlgoritmo
    
```
- Status Bar:** La ejecución ha finalizado sin errores.

**Execution Window (Right):**

- Title Bar:** PSeInt - Ejecutando proceso SAL...
- Text Output:**

```

*** Ejecución Iniciada. ***
Introduce el nombre del usuario 1:
> Alice
Hola, Alice
Introduce el nombre del usuario 2:
> Bob
Hola, Bob
*** Ejecución Finalizada. ***
    
```
- Checkboxes at the bottom:**
 No cerrar esta ventana    Siempre visible
- Buttons at the bottom:** Reiniciar

# Función con parámetros y retorno

The image shows two windows of the PSelint IDE. The left window displays a script named 'saludar.psc' containing the following pseudocode:

```

1 Funcion saludo < obtenerSaludo(nombre)
2     saludo < 'Hola, ' + nombre
3 FinFuncion
4
5 Algoritmo Saludar
6     Escribir 'Introduce el nombre del usuario 1:'
7     Leer usuario_1
8     Escribir obtenerSaludo(usuario_1)
9     Escribir 'Introduce el nombre del usuario 2:'
10    Leer usuario_2
11    Escribir obtenerSaludo(usuario_2)
12 FinAlgoritmo
  
```

The right window shows the execution process, starting with 'Ejecución Iniciada.' It prompts the user to 'Introduce el nombre del usuario 1:' followed by the input 'Alice'. It then outputs 'Hola, Alice'. It then prompts 'Introduce el nombre del usuario 2:' followed by the input 'Bob'. It outputs 'Hola, Bob'. Finally, it concludes with 'Ejecución Finalizada.'

**PSelint**

Archivo Editar Configurar Ejecutar Ayuda

imprimir\_saludo\_nombre\_2.psc tabla\_multiplicar.psc saludar.psc\*

Lista de Variables Operadores y Funciones

La ejecución ha finalizado sin errores.

PSelint - Ejecutando proceso SAL...

\*\*\* Ejecución Iniciada. \*\*\*

Introduce el nombre del usuario 1:  
> Alice  
Hola, Alice

Introduce el nombre del usuario 2:  
> Bob  
Hola, Bob

\*\*\* Ejecución Finalizada. \*\*\*

No cerrar esta ventana  Siempre visible Reiniciar

# Ejercicio individual

## Representación textual de algoritmos

Se desea modificar el algoritmo del ejercicio anterior de tal modo que el cálculo del IMC pueda ser reutilizado en otros algoritmos.

Para ello calcularemos el IMC dentro de una función calcularIMC() que reciba como parámetros el peso en kg y la altura en m del usuario y devuelva su IMC, calculado según la siguiente fórmula:

$$\text{IMC} = \frac{\text{Peso}}{\text{Altura}^2}$$

El algoritmo debe solicitar el peso y la altura al usuario, hacer uso de la función indicada y, en función del valor devuelto por dicha función, mostrar una de las siguientes salidas:

IMC	SALIDA A MOSTRAR
< 18,5	Peso bajo
18,5 ≤ IMC < 25	Peso normal
IMC ≥ 25	Peso alto

# Python



## Breve historia



Python fue creado en 1989 por el informático neerlandés Guido Van Rossum.

Van Rossum creó Python en su tiempo libre y le puso el nombre en honor a los Monty Python, de los que era fan.

Python está inspirado en los lenguajes ABC y, de acuerdo con su creador debe ser:

- Fácil, intuitivo y potente.
- De código abierto.
- El código escrito en Python debe ser tan comprensible como cualquier texto en inglés.
- Apto para las actividades diarias, permitiendo construir prototipos en poco tiempo.

Python es hoy en día el lenguaje de programación más popular según los índices TIOBE y PYPL.

# Usos



Python abarca prácticamente todos los segmentos del mercado

- Programación de sistemas
- Aplicaciones web
- Aplicaciones de escritorio
- Inteligencia artificial
- Cálculo científico
- Procesamiento de imágenes
- Big Data
- Redes
- Sistemas empotrados
- ...

Además, incluso en otros contextos donde se prefieran otras opciones, Python sigue siendo una herramienta de prototipado apreciada.

# Ventajas y desventajas



## Puntos fuertes

- Gramática.
- Su biblioteca estándar permite cubrir un amplio espectro de funcionalidades.
- Amplia variedad de proyectos externos sencillos de integrar.



## Puntos débiles

- Lenguaje no compilado > lentitud.
- Relativa baja difusión respecto a otros lenguajes, como C, C++ o Java, aunque cada vez tiene más peso.

# Ventajas y desventajas

Python es un lenguaje que dispone de **varias implementaciones**, siendo la más común CPython, que está escrita en C.

 python	<b>CPython</b>	Escrita en C. Es la implementación más común y de referencia, y es a la que nos referimos cuando simplemente decimos “Python”
 Jython	<b>Jython</b>	Escrita en Java.
 IronPython	<b>IronPython</b>	Escrita en C#
 PyPy	<b>PyPy</b>	Escrita ¡en Python! Utiliza un subconjunto limitado del lenguaje Python llamado RPython junto con compilación en tiempo de ejecución o just-in-time (JIT) para mejorar su rendimiento

# Características de Python

- Interpretado
- Multiparadigma
- De alto nivel
- Tipado dinámico

# El Zen de Python



- Hermoso es mejor que feo
- Explícito es mejor implícito
- Simple es mejor que complejo
- Complejo es mejor que complicado
- Plano es mejor que anidado
- Disperso es mejor que lento
- La legibilidad cuenta
- Los casos especiales no son suficientemente especiales como para romper las reglas
- Aunque lo pragmático gana a la pureza
- Los errores nunca deberían dejarse pasar silenciosamente
- A menos que se silencien explícitamente
- Cuando te enfrentes a la ambigüedad, rechaza la tentación de adivinar
- Deberá haber una, y preferiblemente solo una- manera obvia de hacerlo
- Aunque puede que no se obvia a primera vista a menos que seas holandés
- Ahora es mejor que nunca
- Aunque muchas veces nunca es mejor que “ahora mismo”
- Si la implementación es difícil de explicar, puede que sea una mala idea.
- Si la implementación es sencilla de explicar, puede que sea una buena idea.
- Los espacios de nombres son una gran idea ¡tengamos más de esas!
- Está todo dicho

# Ramas de Python

Actualmente Python dispone de dos ramas activas

- 2.x, cuya última versión es la 2.7 y se encuentra en el final de su ciclo de vida.
- 3.x, cuya última versión es la 3.10.

Existen algunas diferencias importantes entre Python 2 y 3, en este curso únicamente estudiaremos Python 3.

Es importante tener en cuenta que muchas veces la bibliografía que utilizamos o los fragmentos de código que encontramos en internet pueden estar desarrollados para otras versiones de Python y no funcionar en nuestro sistema.

VERSIÓN	ESTADO	LANZAMIENTO	FINAL SOPORTE
3.10	bugfix	04/10/2021	10/2026
3.9	bugfix	05/10/2020	10/2025
3.8	security	14/10/2019	10/2024
3.7	security	27/06/2018	06/2023
2.7	end-of-life	03/07/2010	01/2020

# Modo de gobierno

- Creador del lenguaje: Guido Van Rossum
- BDFL, Benevolent Dictator for Life (Dictador Benevolente Vitalicio)
- PEP: Python Enhancement Proposal (Propuestas de mejora para Python)

Son documentos que describen una propuesta que pretende mejorar algún aspecto de Python, y pueden ser de tres tipos:

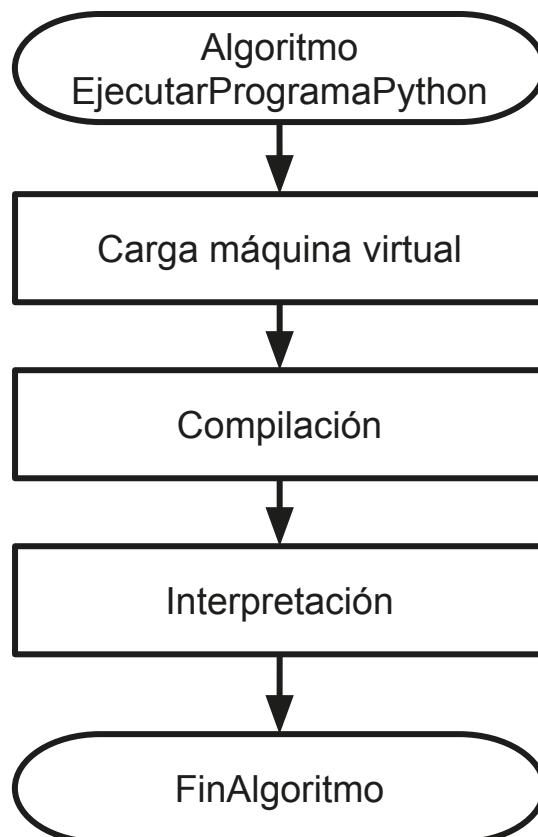
- Standard track PEP: propuestas de tipo técnico
- Process PEP: propuestas estratégicas
- Informational PEP: recomendaciones

Cada PEP es revisada por los responsables de la comunidad y son calificadas como “a tener en cuenta”, “rechazadas” o “a implementar”

- Toma de decisiones

Cualquier persona puede aportar su contribución a Python (ej.: reportando un bug, enviando una PEP, escribiendo un parche, etc.). No obstante solo unas pocas personas tienen permiso de escritura en el repositorio oficial de Python.

# Fases de ejecución de un programa Python



- 1. Carga de la máquina virtual:** cuando se inicia un programa Python se carga la máquina virtual.
- 2. Compilación:** a continuación se compila el programa a bytecodes. Para evitar tener que compilarlo de nuevo con cada ejecución, se guarda su versión compilada en un fichero .pyc. Cada vez que se ejecuta el programa se comprueba si se ha realizado algún cambio y solo en caso afirmativo se vuelve a compilar.
- 3. Interpretación:** por último la máquina virtual ejecuta el bytecode compilado y procesa el algoritmo.

# Primeros pasos



# Entorno de trabajo

En el presente curso solo abordaremos CPython, la implementación de referencia de Python.

Comenzaremos utilizando un intérprete online de Python para familiarizarnos con el lenguaje y después veremos cómo instalar Python en nuestro equipo para los siguientes sistemas:

- Windows
- macOS
- GNU/Linux

Por último utilizaremos Python en nuestro entorno local a través de una máquina virtual Kali Linux con Python preinstalado.

# Entorno de trabajo online

<https://replit.com/languages/python3>

The screenshot shows a web-based Python development environment. At the top left is a dropdown menu set to "Python". In the center is a large green "Run" button. To the right is a "Share" button. The main area is divided into two panes. The left pane contains the Python code:

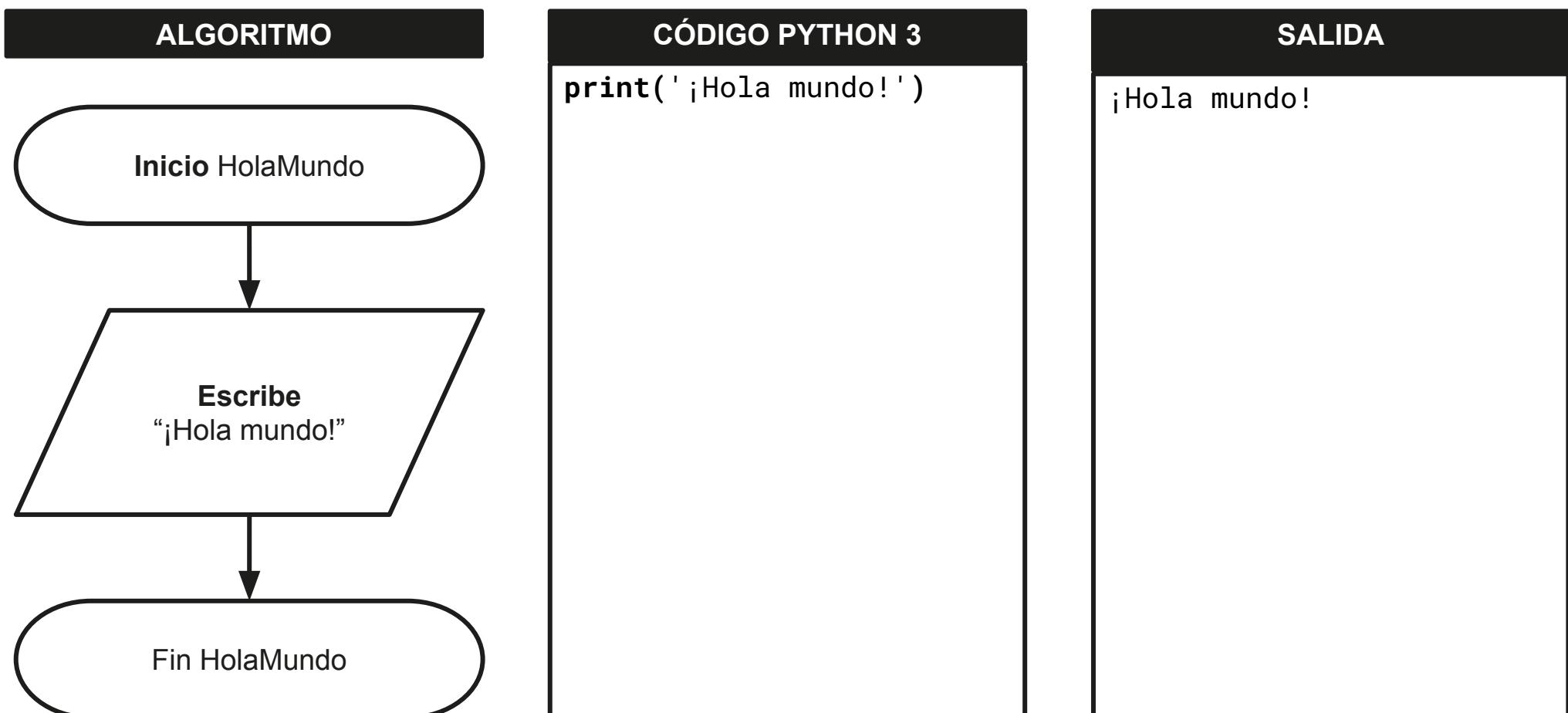
```
1 print('Hello, world!')
```

The right pane shows the output of the code execution:

```
Python 3.8.2 (default, Feb 26 2020, 02:56:10)
>
```

# Primer programa en Python

La función `print()` de Python nos permite escribir información en la salida estándar (pantalla).



# Ayuda de Python

La función **help()** de Python invoca el sistema de ayuda del lenguaje. Si no pasamos ningún argumento a la función nos devolverá un mensaje de bienvenida explicando el uso de la misma.

## CÓDIGO PYTHON 3

```
help()
```

## SALIDA

Welcome to Python 3.8's help utility!

If this is your first time using Python, you should definitely check out the tutorial on the Internet at <https://docs.python.org/3.8/tutorial/>.

Enter the name of any module, keyword, or topic to get help on writing Python programs and using Python modules. To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, symbols, or topics, type "modules", "keywords", "symbols", or "topics". Each module also comes with a one-line summary of what it does; to list the modules whose name or summary contain a given string such as "spam", type "modules spam".

# Ayuda de Python: palabras reservadas

Por ejemplo, `help('keywords')` nos devuelve información sobre las palabras reservadas del lenguaje.

## CÓDIGO PYTHON 3

```
help('keywords')
```

## SALIDA

Here is a list of the Python keywords. Enter any keyword to get more help.

False	class	from	or
None	continue	global	pass
True	def	if	raise
and	del	import	return
as	elif	in	try
assert	else	is	while
async	except	lambda	with
await	finally	nonlocal	yield
break	for	not	

# Ayuda de Python: for

Y `help('for')` nos devolverá información sobre la sentencia `for`.

## CÓDIGO PYTHON 3

```
help('for')
```

## SALIDA

The "for" statement

\*\*\*\*\*

The "for" statement is used to iterate over the elements of a sequence (such as a string, tuple or list) or other iterable object:

```
for_stmt ::= "for" target_list "in"  
expression_list ":" suite ["else" ":" suite]
```

...

# Documentación oficial

Python pone a disposición de sus usuarios documentación oficial en varios idiomas, incluido el español:

<https://docs.python.org/es/3/>

The screenshot shows the Python documentation website for version 3.10.1. The top navigation bar includes links for "Python", "Spanish" (selected), "3.10.1" (selected), and "3.10.1 Documentation". On the left sidebar, there are sections for "Descarga" (Download) with a link to "Descarga esta documentación", "Documentos por versión" (Documents by version) listing versions from 3.11 down to 3.6, and "Otros recursos" (Other resources) including links to PEP indices, beginner's guides, book lists, and developer guides. The main content area features a large title "documentación de Python – 3.10.1" and a welcome message: "Welcome! This is the official documentation for Python 3.10.1." Below this, a section titled "Áreas de la documentación:" lists several topics with their descriptions: "¿Qué hay de nuevo en Python 3.10?" (What's new in Python 3.10?), "Tutorial" (Tutorial, starting here), "Referencia de la Biblioteca" (Reference Library, keep it under your pillow), "Referencia del lenguaje" (Language Reference, description of syntax and language elements), "Configuración y uso de Python" (Configuration and Usage, how to use Python on different platforms), "Códigos (HOWTOs) de Python" (Python HOWTOs, detailed documents on specific topics), "Instalación de módulos de Python" (Module Installation, installation from the Python Package Index and other sources), "Distribuir módulos de Python" (Module Distribution, publication of modules for others to install), "Extender e incrustar" (Extending and Embedding, tutorial for C/C++ programmers), "Python/C API" (Python/C API, reference for C/C++ programmers), and "Preguntas frecuentes" (FAQ, frequently asked questions).

# Reglas y convenciones de nomenclatura

Los nombres de los objetos de Python deben seguir ciertas reglas:

- Solo usaremos **letras en minúscula** (a - z), **letras en MAYÚSCULA** (A - Z), **dígitos** (0 - 9) o el **guión bajo** o **underscore** (\_). Aunque Python lo acepta, **no utilizaremos caracteres especiales** como la ñ, **letras con tilde** (á, é, í, ó, ú) u otros signos ortográficos.
- El primer carácter no puede ser un dígito.
- Los nombres que empiezan por guión bajo (\_) o doble guión bajo (\_\_) están reservados para variables con un significado especial.
- No pueden usarse palabras reservadas del lenguaje como nombres de variable.

# Reglas y convenciones de nomenclatura

En general los lenguajes de programación **no permiten usar espacios** en los nombres o identificadores de los objetos, como variables, constantes, funciones, etc.

Por otra parte es recomendable utilizar **nombres descriptivos**, y estos suelen estar compuestos de varias palabras.

Para facilitar la lectura de los nombres compuestos por varias palabras existen diversas técnicas. Estas son las más utilizadas en el ámbito de la programación:

TIPO	DESCRIPCIÓN	EJEMPLO
<b>UpperCamelCase</b>	Las palabras no se separan. La primera letra de cada palabra en mayúscula, el resto en minúsculas.	MiClase
<b>lowerCamelCase</b>	Igual que la anterior, pero la primera letra del identificador se escribe en minúscula	miFuncion
<b>snake_case</b>	Las palabras se separan con un guión bajo	mi_variable, MI_CONSTANTE

# Convención de nombres

En Python se recomienda seguir la siguiente convención de nombres:

ELEMENTO	TIPO	DESCRIPCIÓN	EJEMPLO
<b>Variable</b>	snake_case (minúsculas)	Letras minúsculas. Palabras separadas mediante un guión bajo.	variable, mi_variable
<b>Constante</b>	snake_case (mayúsculas)	Letras mayúsculas. Palabras separadas mediante un guión bajo	CONSTANTE, MI_CONSTANTE
<b>Funciones</b>	snake_case (minúsculas)	Letras minúsculas. Palabras separadas mediante un guión bajo.	funcion, mi_funcion
<b>Clases</b>	UpperCamelCase	La primera letra de cada palabra en mayúsculas, el resto en minúsculas. Camel case	Clase, MiClase

# Escribir

En Python podemos escribir datos en pantalla mediante la función `print()`. Los parámetros que especifiquemos entre comillas serán considerados texto literal. De lo contrario Python lo considerará una expresión y tratará de evaluarla. Es posible especificar varios parámetros separados por comas.

## CÓDIGO PYTHON 3

```
print("3+2")
print(3+2)
print("texto1", "texto2")
```

## SALIDA

```
3+2
5
texto1 texto2
```

# Asignación

En Python la asignación se representa mediante el símbolo ‘=’. Para diferenciarlo del operador relacional “es igual a”, este último se representa como “==”.

CÓDIGO PYTHON 3

```
x = 1  
print(x)
```

SALIDA

```
1
```

# Composición secuencial

Al igual que en el lenguaje algorítmico, las expresiones de Python se procesan secuencialmente y de arriba a abajo. NOTA: En Python no es necesario terminar las expresiones con un punto y coma (';').

CÓDIGO PYTHON 3

```
x = 1
print(x)
x = x + 1
print(x)
x = x + 1
print(x)
```

SALIDA

```
1
2
3
```

# Comentarios

Los comentarios en Python se pueden escribir de dos formas, según ocupen una o varias líneas.

## CÓDIGO PYTHON 3

```
# Este es un comentario de una línea
x = 1 # Este también
'''
Este es un
comentario de
varias líneas
'''
print(x)
```

## SALIDA

```
1
```

# Tipos

Python es un lenguaje de tipado dinámico. No es preciso declarar el tipo de una variable, y este puede cambiar dinámicamente en tiempo de ejecución. Podemos consultar el tipo de un objeto mediante la función **type()**.

## CÓDIGO PYTHON 3

```
# Ahora x es de tipo entero (integer)
x = 1
print(x)
tipo = type(x)
print(tipo)
# Y ahora de tipo cadena (string)
x = "Hola"
print(x)
tipo = type(x)
print(tipo)
```

## SALIDA

```
1
<class 'int'>
Hola
<class 'str'>
```

# Tipos

Recuerda que todo aquello que especifiquemos entre comillas se considerará texto literal, y por tanto será de tipo `string` (cadena de texto).

## CÓDIGO PYTHON 3

```
x = 1
# Imprime el valor de la variable x
print(x)
# Imprime el carácter (letra) x
print("x")
```

## SALIDA

```
1
x
```

# Leer

En Python podemos leer los datos introducidos por el usuario mediante la función **input()**.

Adicionalmente la función **input()** permite escribir un texto en la pantalla antes de llevar a cabo la lectura.

**NOTA:** Recuerda que **input** se escribe con “n” antes de “p”.

## CÓDIGO PYTHON 3

```
print('Introduce tu nombre: ')
nombre = input()
print('Hola', nombre)

# Imprime una línea en blanco
print()

nombre= input('Introduce tu nombre: ')
print('Hola', nombre)
```

## SALIDA

Introduce tu nombre:  
Alice  
Hola Alice

Introduce tu nombre: Bob  
Hola Bob

# Leer

La función **input()** siempre devuelve un valor de tipo string (cadena de texto).

Si deseamos obtener un valor numérico podemos convertir el tipo de los datos obtenidos mediante la función **int()**.

## CÓDIGO PYTHON 3

```
numero= input('Introduce un número: ')
tipo = type(numero)
print(tipo)

# Imprime una línea en blanco
print()

numero= int(input('Introduce un número: '))
print(type(numero))
```

## SALIDA

Introduce un número: 1  
<class 'str'>

Introduce un número: 1  
<class 'int'>

# Leer

Si realizamos operaciones entre variables cuyo tipo no es el adecuado podemos obtener resultados inesperados.

## CÓDIGO PYTHON 3

```
numero1= input('Introduce un número: ')
numero2= input('Introduce otro número: ')
print(numero1+numero2)

# Imprime una línea en blanco
print()

numero1= int(input('Introduce un número: '))
numero2= int(input('Introduce otro número: '))
print(numero1+numero2)
```

## SALIDA

```
Introduce un número: 1
Introduce otro número: 2
12
```

```
Introduce un número: 1
Introduce otro número: 2
3
```

# Ejercicio individual

## Asignación, lectura y escritura

Escribe un programa en Python que lleve a cabo las siguientes acciones:

- Solicite al usuario que introduzca un número y lo asigne a una variable a.
- Solicite al usuario que introduzca un número y lo asigne a una variable b.
- Calcule el resultado de sumar las variables a + b y lo asigne a una variable c.
- Imprima el siguiente texto: "El resultado de sumar a + b es ", seguido del valor de c.

# Estructuras de control



# Indentación

La mayoría de los lenguajes de programación utilizan la **sangría** o **indentación** para mejorar la legibilidad. Python la utiliza para identificar bloques de código, y por tanto es muy importante respetarla.

## LENGUAJE ALGORÍTMICO

```
Si condición
    accion_1;
    accion_2;
Fin Si
accion_3
```

## CÓDIGO PYTHON 3

```
if condición:
    accion_1
    accion_2
accion_3
```

# Estructura selectiva simple

En Python podemos representar la estructura selectiva simple mediante la palabra reservada **if**. Prueba el siguiente ejemplo. Algo falla ¿qué ha sucedido? ¿cómo lo podemos solucionar?

## CÓDIGO PYTHON 3

```
# Ejemplo 1
numero = input('Introduce un número: ')
if numero == 10:
    print('Has acertado')
```

## SALIDA

```
Introduce un número: 10
```

# Estructura selectiva doble y múltiple

En Python podemos representar la estructura selectiva múltiple mediante las palabras reservadas `if`, `elif` y `else`:

## CÓDIGO PYTHON 3

```
# Estructura selectiva doble
numero = int(input('Introduce un número: '))
if numero == 10:
    print('Has acertado')
else:
    print('No has acertado')

# Imprime una línea en blanco
print()

# Estructura selectiva múltiple
numero = int(input('Introduce un número: '))
if numero < 10:
    print('Demasiado pequeño')
elif numero > 10:
    print('Demasiado Grande')
else:
    print('Has acertado')
```

## SALIDA

Introduce un número: 5  
No has acertado

Introduce un número: 5  
Demasiado pequeño

# Estructura iterativa Mientras

Python implementa la estructura iterativa Mientras mediante la palabra reservada while.

## CÓDIGO PYTHON 3

```
# Estructura iterativa Mientras
i = 1
while i <= 10:
    print(i)
    i = i + 1
```

## SALIDA

```
1
2
3
4
5
6
7
8
9
10
```

# Estructura iterativa Mientras

Mediante las sentencias **break** y **continue** podemos controlar el flujo de una iteración **while**.

## CÓDIGO PYTHON 3

```
# Sentencia break
i = 0
while i <= 10:
    i = i + 1
    if i == 5:
        break
    print(i)

# Imprime una línea en blanco
print()

# Sentencia continue
i = 0
while i < 10:
    i = i + 1
    if i == 5:
        continue
    print(i)
```

## SALIDA

```
1
2
3
4

1
2
3
4
6
7
8
9
10
```

# Estructura iterativa Para Cada

Python implementa la estructura iterativa **Para Cada** mediante las palabras reservadas **for**, **in**.

La sintaxis es la siguiente:

```
for elemento in iterable:  
    acciones
```

Donde **iterable** es un objeto que permite recorrer sus elementos uno a uno.

A continuación se muestra un ejemplo.

## CÓDIGO PYTHON 3

```
# Definimos una lista con los números del 1 al 5  
numeros = [1,2,3,4,5]  
# n toma los valores de la lista  
for n in numeros:  
    # para cada valor de la lista imprime n  
    print(n)
```

## SALIDA

```
1  
2  
3  
4  
5
```

# Ejercicio individual

## Estructura iterativa Para (bucle for)

Escribe un programa en Python que lleve a cabo las siguientes acciones:

- Sigue el siguiente formato de respuesta:
- Solicita al usuario un número de inicio.
- Solicita al usuario un número final.
- Imprime los números desde inicio hasta final.

Pista: investiga el uso del bucle **for** junto con la función `range()` en Python.

# Trabajo en grupo

## Estructura selectiva múltiple por casos

Investigad cómo implementar una estructura de selección múltiple por casos en Python.

Pista: esta estructura se conoce en inglés como **switch - case**.

Se pide:

1. ¿Hay alguna estructura del lenguaje Python equivalente a una estructura de selección múltiple por casos?
2. ¿Es posible utilizarla en nuestro entorno de trabajo online?
3. En caso afirmativo, escribe un ejemplo.
4. En caso negativo, escribe un ejemplo con una estructura alternativa.

# Ejercicio individual

## Estructura iterativa Hacer ... mientras

Investiga cómo implementar una estructura iterativa **Hacer ... Mientras** (repetir) en Python.

Pista: esta estructura se conoce en inglés como **do - while**.

Se pide:

1. ¿Hay alguna estructura del lenguaje Python equivalente a una estructura iterativa **Hacer ... Mientras**?
2. ¿Es posible utilizarla en nuestro entorno de trabajo online?
3. En caso afirmativo, escribe un ejemplo.
4. En caso negativo, escribe un ejemplo con una estructura alternativa.

# Debate

## Tipado dinámico vs tipado estático

Hemos visto que Python es un lenguaje de tipado dinámico (el tipo de una variable puede cambiar en tiempo de ejecución).

- Investiga las ventajas y desventajas del tipado dinámico frente al tipado estático.
- ¿Cuál crees que es mejor? Razona tu respuesta
- ¿Es posible utilizar Python como un lenguaje de tipado estático?

# Operadores



# Operadores

Estos son los principales tipos de operadores que podemos utilizar en Python:

- **Aritméticos**
- **Asignación**
- **Relacionales**
- **Lógicos**
- **De pertenencia**

# Operadores Aritméticos

Los operadores aritméticos realizan una operación aritmética sobre dos valores numéricos.

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
<code>+</code>	Adición o suma	<code>2 + 3</code>	5
<code>-</code>	Substracción o resta	<code>5 - 3</code>	2
<code>*</code>	Multiplicación	<code>2 * 3</code>	6
<code>/</code>	División	<code>5 / 2</code>	2.5
<code>%</code>	Módulo, residuo o resto	<code>5 % 3</code>	2
<code>**</code>	Potencia o exponenciación	<code>2 ** 3</code>	8
<code>//</code>	División entera	<code>5 // 3</code>	1

# Operadores Aritméticos

Comprobemos el uso de algunos de los operadores de aritméticos.

## CÓDIGO PYTHON 3

```
a = 5  
b = 2  
  
# Suma  
print(a + b)  
  
# Resta  
print(a - b)  
  
# Multiplicación  
print(a * b)  
  
# División  
print(a / b)  
  
# Módulo  
print(a % b)  
  
# Potencia o exponenciación  
print(a ** b)  
  
# División entera  
print(a // b)
```

## SALIDA

```
7  
3  
10  
2.5  
1  
25  
2
```

# Operadores de Asignación

El operador de asignación asigna un valor. Puede combinarse con un operador aritmético.

OPERADOR	EJEMPLO	EQUIVALENTE
=	x = 5	
+=	x += 5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5
%=	x %= 5	x = x % 5
**=	x **= 5	x = x ** 5
//=	x //= 5	x = x // 5

# Operadores de Asignación

Comprobemos el uso de algunos de los operadores de asignación.

## CÓDIGO PYTHON 3

```
# Asignamos a x el valor 5
x = 5
print(x)

print() # Imprime una línea en blanco

# x = x + 5
x += 5
print(x)

print() # Imprime una línea en blanco

# x = x - 5
x -= 5
print(x)

print() # Imprime una línea en blanco

# x = x * 5
x *= 5
print(x)
```

## SALIDA

```
5
10
5
25
```

# Operadores Relacionales

Los operadores relacionales comparan dos valores.

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
>	Mayor que	5 > 3	True
<	Menor que	5 < 3	False
==	Igual a	5 == 5	True
>=	Mayor o igual que	5 >= 5	True
<=	Menor o igual que	5 <= 3	False
!=	No es igual a	5 != 5	False

# Operadores Relacionales

Comprobemos el uso de algunos operadores relacionales.

## CÓDIGO PYTHON 3

```
a = 1
b = 2

# ¿a es mayor que b?
print(a > b)

# ¿a es menor que b?
print(a < b)

# ¿a es igual a b? (nótese el ==)
print(a == b)

# ¿a es menor o igual a b?
print(a <= b)

# ¿a es distinto de b?
print(a != b)
```

## SALIDA

```
False
True
False
True
True
```

# Operadores Lógicos

Los operadores lógicos operan sobre los tipos booleanos.

OPERADOR	DESCRIPCIÓN	EJEMPLO	RESULTADO
<b>and</b>	‘Y’ lógico	True and False	False
<b>or</b>	‘O’ lógico	True or False	True
<b>not</b>	negación	not True	False

# Operadores Lógicos

Comprobemos el uso de algunos operadores lógicos.

## CÓDIGO PYTHON 3

```
a = 1
b = 2
c = 3

# ¿a es mayor que b?
print(a > b)

# Negación de la expresión anterior
print(not(a > b))

# ¿b es menor que c?
print(b < c)

# ¿a es mayor que b, y b es menor que c?
print(a > b and b < c)

# ¿a es mayor que b, o b es menor que c?
print(a > b or b < c)
```

## SALIDA

```
False
True
True
False
True
```

# Operadores de Pertenencia

Los operadores de pertenencia identifican si un operando se encuentra presente en un determinado objeto.

## CÓDIGO PYTHON 3

```
# Comprobar si un número está en una lista
mi_lista = [1,2,3,4,5]
print(3 in mi_lista)
print(9 not in mi_lista)

print() # Imprime una línea en blanco

# Comprobar si una cadena contiene otra
mi_cadena = 'Hola Mundo'
print('Hola' in mi_cadena)
print('hola' in mi_cadena)
```

## SALIDA

```
True
True

True
False
```

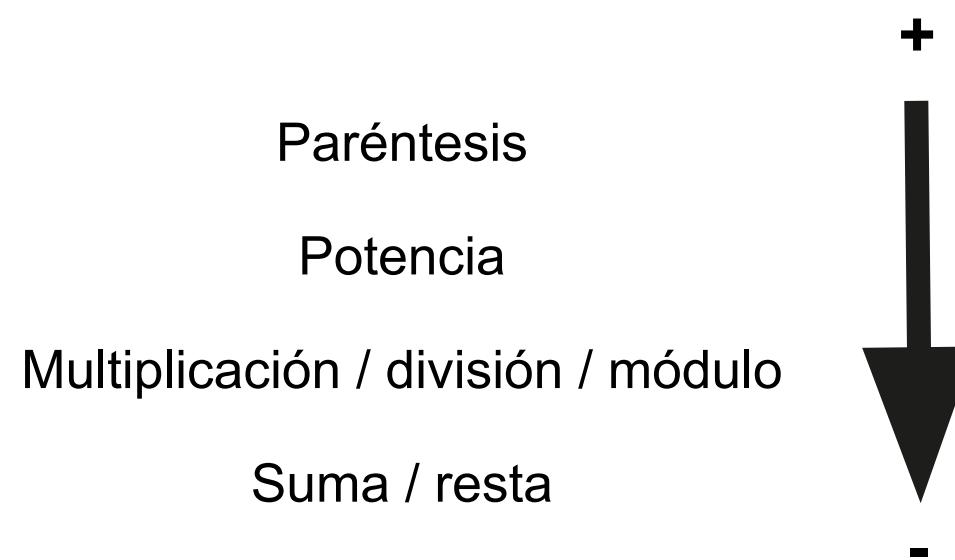
# Prioridad de los operadores

Los operadores de Python se evalúan según el siguiente orden de prioridad o precedencia:

- - (cambio de signo)
- \*, /, //, %
- +, - (resta)
- in, not in, <, <=, >, >=, ==, !=
- not
- and
- or

Recuerda que en igualdad de prioridad los operadores se evalúan **de izquierda a derecha**, y que se puede determinar el orden en el que queremos que se apliquen los operadores mediante el uso de paréntesis.

# Prioridad de los operadores



# Prioridad de los operadores

`1 + 2 ** 3 / 4 * 5`



`1 + 8 / 4 * 5`



`1 + 2.0 * 5`



`1 + 10.0`



`11.0`

# Prioridad de los operadores

Cuando representamos una expresión compleja es una buena práctica indicar el orden de precedencia mediante paréntesis, aunque no sean necesarios.

CÓDIGO PYTHON 3	SALIDA
<pre># Esta expresión es difícil de entender x = 1 + 2 ** 3 / 4 * 5 print(x)  print()  # Escrita así es más legible x = 1 + (((2 ** 3) / 4) * 5) print(x)</pre>	<pre>11.0  11.0</pre>

# Operadores de cadenas

Python permite utilizar los operadores de adición (+) y multiplicación (\*) en cadenas de caracteres. Obsérvese el uso del carácter de control \n, que imprime un salto de línea.

## CÓDIGO PYTHON 3

```
# Concatenar cadenas
mensaje = 'Hola '
mensaje += 'Mundo'
print(mensaje)

print()

# Multiplicar cadenas
castigo = 'No volveré a hablar en clase\n'
castigo *= 10
print(castigo)
```

## SALIDA

```
Hola Mundo

No volveré a hablar en clase
```

# Operadores y tipos

Cuando escribamos una expresión debemos tener cuidado con los tipos. En ocasiones Python **convertirá los tipos implícitamente**.

Recuerda que podemos saber el tipo de una variable con la función **type()**.

## CÓDIGO PYTHON 3

```
# Ejemplo 1: suma de un entero y un real
a = 1
b = 2.0
c = a + b
print(c)
print(type(a))
print(type(b))
print(type(c))

print()

# Ejemplo 2: división real de dos enteros
a = 4
b = 2
c = a/b
print(c)
print(type(a))
print(type(b))
print(type(c))
```

## SALIDA

```
3.0
<class 'int'>
<class 'float'>
<class 'float'>

2.0
<class 'int'>
<class 'int'>
<class 'float'>
```

# Operadores y tipos

En otros casos Python no convertirá los tipos y nos devolverá un error.

## CÓDIGO PYTHON 3

```
# Ejemplo 3: concatenar entero y cadena
a = 4
b = 'dos'
print(type(a))
print(type(b))
c = a + b
print(c)
```

## SALIDA

```
<class 'int'>
<class 'str'>
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    c = a + b
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

# Operadores y tipos

Podemos forzar la conversión mediante las funciones de conversión de tipo.

## CÓDIGO PYTHON 3

```
# Convertir de int a float
i = 4
print(i)
print(type(i))
f = float(i)
print(f)
print(type(f))

print()

# Convertir de float a int
f = 4.0
print(f)
print(type(f))
i = int(f)
print(i)
print(type(i))
```

## SALIDA

```
4
<class 'int'>
4.0
<class 'float'>

4.0
<class 'float'>
4
<class 'int'>
```

# Gestión de excepciones



# Gestión de excepciones

La conversión de tipos resulta muy útil cuando deseamos leer datos del exterior.

## CÓDIGO PYTHON 3

```
# Adivina el número
numero_pensado = 65
print('Adivina el número')
while True:
    numero = int(input('Número?: '))
    if numero < numero_pensado:
        print('Demasiado pequeño')
    elif numero > numero_pensado:
        print('Demasiado grande')
    else:
        break
print('¡Lo has adivinado!')
```

## SALIDA

```
Adivina el número
Número?: 50
Demasiado pequeño
Número?: 75
Demasiado grande
Número?: 63
Demasiado pequeño
Número?: 69
Demasiado grande
Número?: 66
Demasiado grande
Número?: 64
Demasiado pequeño
Número?: 65
¡Lo has adivinado!
```

# Gestión de excepciones

Pero ¿qué sucede si el valor introducido por el usuario no se puede convertir a un número?

## CÓDIGO PYTHON 3

```
# Adivina el número
numero_pensado = 65
print('Adivina el número')
while True:
    numero = int(input('Número?: '))
    if numero < numero_pensado:
        print('Demasiado pequeño')
    elif numero > numero_pensado:
        print('Demasiado grande')
    else:
        break
print('¡Lo has adivinado!')
```

## SALIDA

```
Adivina el número
Número?: a
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    numero = int(input('Número?: '))
ValueError: invalid literal for int()
with base 10: 'a'
```

# Gestión de excepciones

Una forma de evitar estos errores es mediante la captura de excepciones

## CÓDIGO PYTHON 3

```
# Adivina el número
numero_pensado = 65
print('Adivina el número')
while True:
    try:
        numero = int(input('Número?: '))
    except:
        print('Número no válido')
    else:
        if numero < numero_pensado:
            print('Demasiado pequeño')
        elif numero > numero_pensado:
            print('Demasiado grande')
        else:
            break
print('¡Lo has adivinado!')
```

## SALIDA

```
Adivina el número
Número?: a
Número no válido
Número?: 65
¡Lo has adivinado!
```

# Gestión de excepciones

Las excepciones nos permiten controlar el comportamiento de un programa cuando se produce un error.

Siempre que ejecutemos una acción que pueda producir algún tipo de excepción (situación de error) es importante capturarla para evitar que el flujo del programa se detenga.

La sintaxis para capturar una excepción es la siguiente:

```
try:  
    # Acciones que pueden provocar la excepción  
except NombreExcepcion:  
    # Este código se ejecuta si se produce una excepción de tipo NombreExcepcion  
except:  
    # Este código se ejecuta si se ha producido cualquier tipo de excepción  
else:  
    # Este código se ejecuta en caso contrario  
finally:  
    # Este código se ejecuta haya habido o no una excepción
```

A continuación veremos un ejemplo.

# Gestión de excepciones

Una forma de evitar estos errores es mediante la captura de excepciones

## CÓDIGO PYTHON 3

```
# Divide dos números
while True:
    try:
        dividendo = int(input('Dividendo?: '))
        divisor = int(input('Divisor?: '))
        resultado = dividendo/divisor
    except ValueError:
        print('Debes introducir un número válido')
    except ZeroDivisionError:
        print('El divisor no puede ser cero')
    except:
        print('Se ha producido un error inesperado')
    else:
        print('Resultado:', resultado)
```

## SALIDA

```
Dividendo?: 4
Divisor?: 2
Resultado: 2.0
Dividendo?: a
Debes introducir un número válido
Dividendo?: 4
Divisor?: 0
El divisor no puede ser cero
Dividendo?:
```

PYTHON

# Módulos



# Importación de módulos

Un **módulo** de Python es un fichero que contiene **funciones**, **variables** o **clases** y que puede ser usado por nuestro programa.

Los módulos nos permiten **organizar** mejor nuestro código y facilitar su **reutilización** en otras aplicaciones.

Además, Python proporciona una biblioteca estándar con módulos que extienden la funcionalidad del lenguaje, y que podemos consultar en la siguiente URL: <https://docs.python.org/es/3.10/library/index.html>

Para poder utilizar un módulo de la biblioteca estándar en nuestro programa debemos importarlo.

La sintaxis para importar un módulo completo es la siguiente:

```
# Mi programa  
import nombre_modulo
```

# Ejercicio individual

## Adivina el número

Modifica el programa “Adivina el número” para que la variable `numero_pensado` escoja un número **entero** aleatorio entre 1 y 100.

Pista: échale un vistazo al módulo `random` de la biblioteca estándar de Python.

El funcionamiento del programa debe ser el siguiente:

1. El programa “piensa” un número entre 1 y 100.
2. Se solicita al usuario que escriba un número.
3. Si el usuario no ha acertado el número, el programa debe informar si es demasiado grande o demasiado pequeño y solicitar un nuevo número.
4. Si el usuario ha acertado el número, el programa debe informar de que lo ha adivinado y debe volver a comenzar, pensando un nuevo número.

Una vez hayas resuelto el programa, mejóralo añadiendo captura de excepciones para evitar que el programa se detenga en caso de que el usuario no introduzca un número válido.



GOBIERNO  
DE ESPAÑA

VICEPRESIDENCIA  
PRIMERA DEL GOBIERNO  
MINISTERIO  
DE ASUNTOS ECONÓMICOS  
Y TRANSFORMACIÓN DIGITAL

SECRETARÍA DE ESTADO  
DE DIGITALIZACIÓN  
E INTELIGENCIA ARTIFICIAL

red.es

Dirección de  
Innovación y Desarrollo  
en Comercio Electrónico  
y Marketing  
**CRN**  
Digital



UNIÓN EUROPEA

*"El FSE invierte en tu futuro"*

Fondo Social Europeo

  
Barrabés

 The Valley