PROGRAMACIÓN, 1º CFGS DAM/DAW
Federico Huércano Ruiz

1 Introducción a la programación

OBJETIVOS DE LA UNIDAD:

- Entender qué son problemas, algoritmos y programas.
- Reconocer las características de un buen algoritmo.
- Conocer los elementos de los lenguajes de programación.
- Comprender el ciclo de vida del software.
- Practicar con herramientas para crear algoritmos.

1. Introducción a la Programación

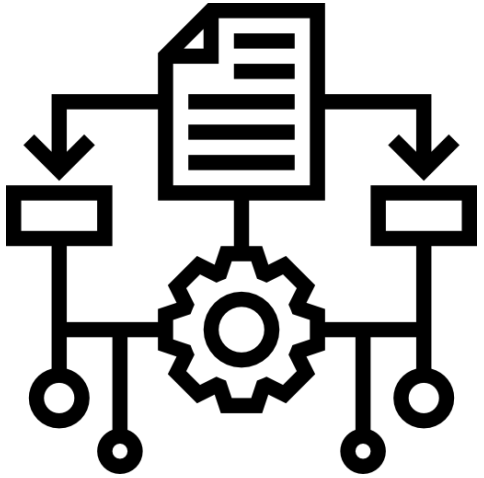
1. Introducción.

La razón principal por la que una persona utiliza un **ordenador** es para **resolver problemas** (en el sentido más general de la palabra), o en otras palabras, procesar una información para obtener un resultado a partir de unos datos de entrada.

Los ordenadores resuelven los problemas mediante la **utilización de programas escritos** por los programadores. Los programas de ordenador no son entonces más que **métodos para resolver problemas**. Por ello, para escribir un programa, lo primero es que el programador sepa **resolver el problema** que estamos tratando.

El programador debe **identificar** cuáles son los **datos de entrada** y a partir de ellos obtener los datos de **salida.**

La **solución**, a la que se llegará por medio del procesamiento de la información que se realizará mediante la utilización de un método para resolver el problema que denominaremos **algoritmo**



1. Introducción a la Programación

2. Problemas, algoritmos y programas

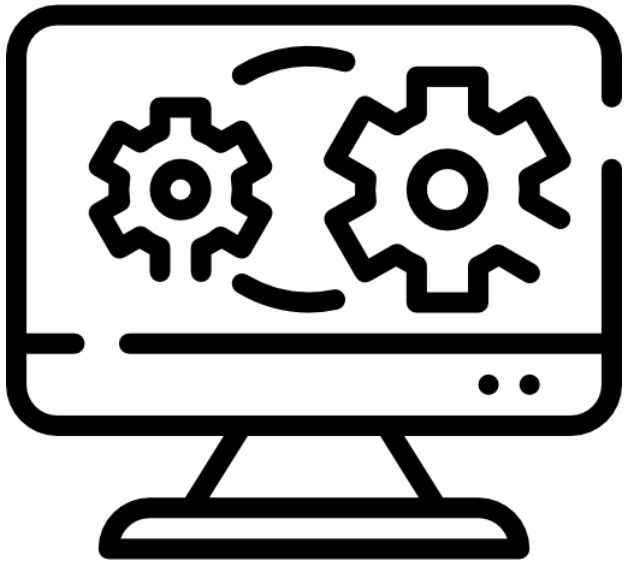
Un **algoritmo** es una **secuencia de acciones** que se realizan para resolver un problema. La tarea de programar en realidad se trata de **resolver problemas**.

La **programación** se encarga de resolver problemas a través de algoritmos

El algoritmo debe especificar **las acciones** que se deben ejecutar y en **qué orden** deben ejecutarse.

Un algoritmo es **independiente** del lenguaje de programación. Es decir, una vez tengamos un algoritmo que resuelve un problema, podemos aplicarlo sobre cualquier **lenguaje de programación**.

Un programa es un algoritmo escrito con las sentencias específicas de un lenguaje de **programación concreto**.

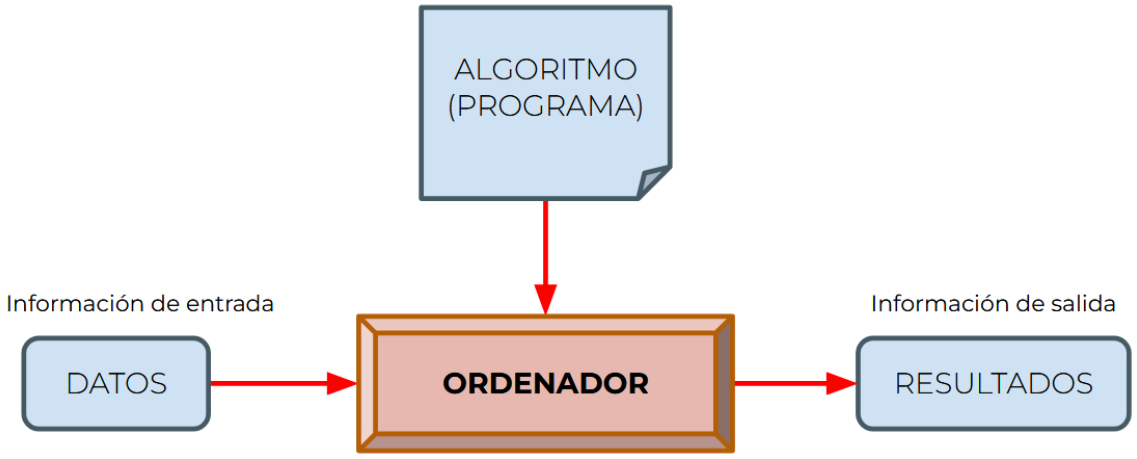


1. Introducción a la Programación

2.1. Algoritmo

Las características principales de un algoritmo son:

- **Precisos:** Las instrucciones tienen que ser **concretas.** No puede haber ambigüedad.
- **Finitos:** El algoritmo tiene que tener un **principio y un fin**. Debe contener un número limitado de pasos.
- **Definidos:** Ante una misma entrada, siempre tiene que dar el mismo resultado.
- **Orden lógico:** Los pasos deben seguir una secuencia coherente



1. Introducción a la Programación

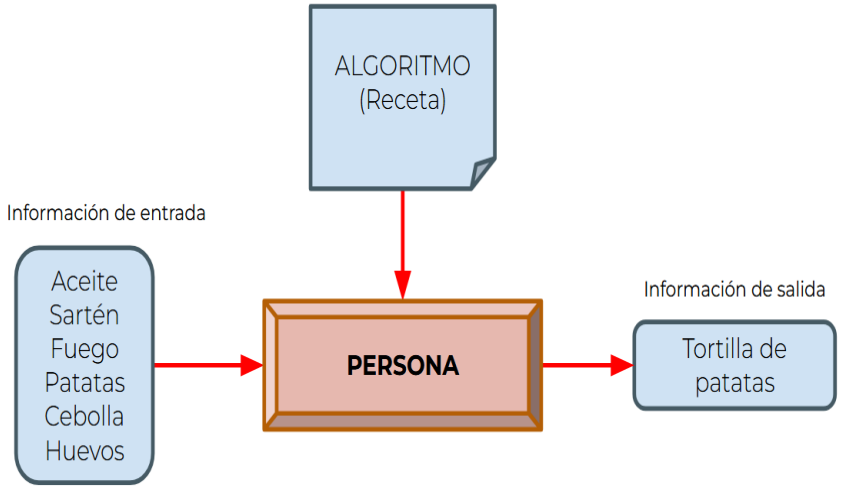
2.1.1 Ejemplo de algoritmo

A continuación, vamos a ver un **ejemplo de algoritmo** para elaborar una **tortilla de patatas**. Tendremos unos datos de entrada, luego el algoritmo que procesa los datos de entrada, y finalmente unos datos de salida o resultado.

- **Datos de entrada:** Huevo, aceite, patatas, cebolla, sartén, fuego.
- **Datos de salida:** Tortilla de patatas.

Algoritmo:

1. Poner el aceite en la sartén.
2. Poner la sartén al fuego.
3. Cuando el aceite esté caliente, freír las patatas y la cebolla.
4. Batir los huevos y añadir las patatas y la cebolla cuando estén al gusto.
5. Añadir los huevos a la sartén y darle la vuelta varias veces.



1. Introducción a la Programación

2.1.2 Otro ejemplo de algoritmo

Otro ejemplo de algoritmo son las **secuencias de reglas** que utilizamos para realizar **operaciones aritméticas**: sumas, restas, productos y divisiones. Son algoritmos porque definen de manera precisa los pasos que se deben seguir para encontrar una solución en un tiempo finito.

A **continuación,** podemos ver un ejemplo de algoritmo que nos permite realizar multiplicaciones de varias cifras.

El algoritmo que define como hacerlo sería el siguiente:

```
   981
x  1234
------
  3924
 2943
1962
 981
------
1210554
```

1. *Se multiplica sucesivamente el **multiplicando** (**981**) por cada una de las cifras del **multiplicador** (**1234**), obtenidas de derecha a izquierda.*
2. *Escribimos los resultados intermedios uno bajo del otro, desplazando cada línea un lugar a la izquierda,*
3. *Se suman todas las filas para obtener el producto*

1. Introducción a la Programación

2.2 Calidad del algoritmo.

Para resolver un **mismo problema** puede haber infinidad de algoritmos distintos que lo resuelvan.

La **calidad de los algoritmos** se podrá medir atendiendo a dos factores:

- El **tiempo necesario** para ejecutarlo.
- Los **recursos**, en términos de memoria principal y de almacenamiento que se necesitan para implementarlo.

Ejemplo: Sumar los números del 1 al 100

Algoritmo 1 (con bucle): Ir sumando uno por uno → 1 + 2 + 3 + ... + 100.

- Tiempo: más lento porque hace 100 operaciones.
- Recursos: casi nada de memoria.

Algoritmo 2 (fórmula matemática): Usar la fórmula n*(n+1)/2.

- Tiempo: instantáneo, solo hace una operación.
- Recursos: mínimo.

1. Introducción a la programación

CASO PRÁCTICO

Diseña un algoritmo de la vida diaria

Instrucciones

1. Formad grupos de 3-4 personas.
2. Elegid una tarea cotidiana sencilla que todos conozcáis
3. Definid vuestro algoritmo en la plantilla:
   - **Entradas:** ¿Qué materiales, datos u objetos necesito para empezar la tarea?
   - **Pasos:** escribid las acciones a realizar de manera clara, ordenada y sin ambigüedades.
   - **Salida:** el resultado final que se obtiene al terminar el proceso.
4. Antes de dar por finalizado el algoritmo, comprobad que cumple estas características
   - **Preciso:** las instrucciones no dejan lugar a dudas.
   - **Finito:** tiene un principio y un final.
   - **Definido:** siempre que se ejecute da el mismo resultado.
   - **Ordenado:** los pasos siguen una secuencia lógica.

1. Introducción a la programación

CASO PRÁCTICO

Algoritmo para encontrar el número mayor de una lista de valores numéricos

Instrucciones

1. Formad grupos de 3-4 personas.
2. Indicar:
   - **Entradas:** ¿qué datos necesitamos para resolver el problema?
   - **Pasos (acciones ordenadas):** pensad en cómo recorrer la lista paso a paso hasta encontrar el mayor valor.
   - **Salida:** el número más grande de la lista.
3. Antes de dar por finalizado el algoritmo, comprobad que cumple estas características
   - **Preciso:** las instrucciones no dejan lugar a dudas.
   - **Finito:** tiene un principio y un final.
   - **Definido:** siempre que se ejecute da el mismo resultado.
   - **Ordenado:** los pasos siguen una secuencia lógica.

1. Introducción a la Programación

3. Conceptos básicos en programación

Un **procesador** es cualquier entidad capaz de interpretar y ejecutar un cierto repertorio de instrucciones. En el caso de un PC sería la CPU.

- Un **programa** está formado por uno o más algoritmos escritos con una notación precisa para que puedan ser ejecutados por un **procesador en concreto (ordenador).**
- Un **lenguaje de programación** es una notación, conjunto de reglas y definiciones que determinan aquello que se puede escribir para desarrollar un programa.
- La **programación** es la actividad de resolución de problemas a través de un ordenador

1. Introducción a la Programación

3. Conceptos básicos en programación

• Un **proceso** es un algoritmo en ejecución que se caracteriza por una sucesión de estados.

• El **estado de un programa** es un conjunto de valores en un momento determinado. El estado indica el grado de progreso en un proceso.

• El **cómputo** es la transformación del estado de un programa al ejecutarse una o más instrucciones.

• Una **instrucción** es una expresión que indica, dentro de un programa, qué operación se debe realizar y qué datos se deben utilizar.

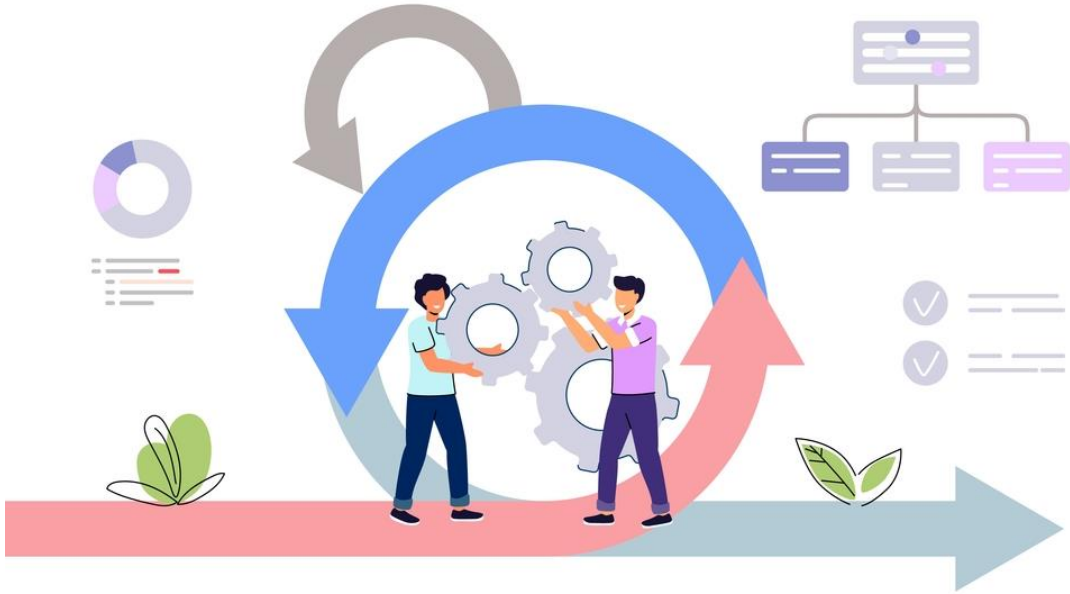
• Los **datos** son cualquier tipo de información dispuesta de forma adecuada para su tratamiento por un ordenador .

1. Introducción a la Programación

4. Ciclo de vida del software

La creación de cualquier programa (**software o sw**) implica la realización de tres pasos genéricos:

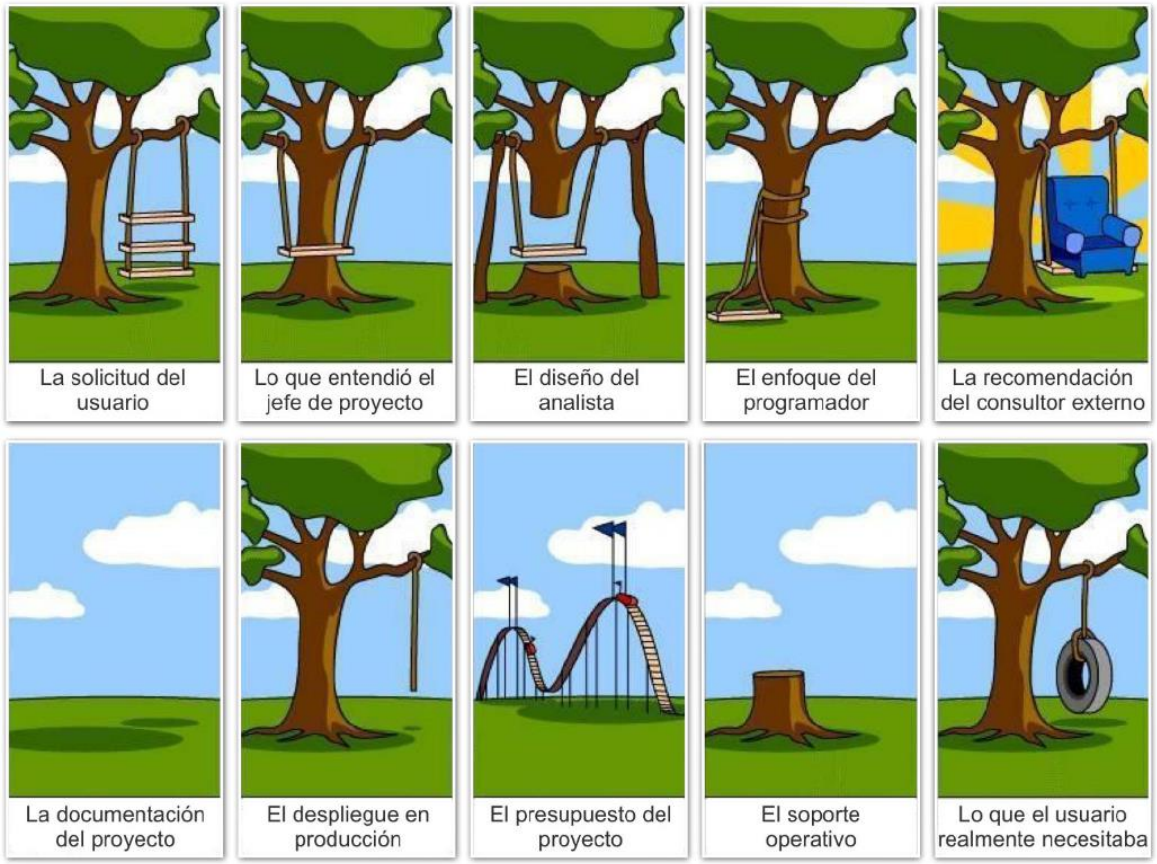
- **Definición** ¿Qué hay que desarrollar?
- **Desarrollo**. ¿Cómo lo implemento?
- **Mantenimiento.** ¿Qué problemas surgen en producción?



1. Introducción a la Programación

4.1 Fase de definición.

Se intenta **caracterizar el sistema que se ha de construir, analizando:**



- La **información** que ha de usar el sistema.
- Qué **funciones** debe realizar
- Qué **condiciones** existen
- Cuáles son las **interfaces** del sistema
- Qué **criterios de validación** se utilizarán.

Este ejemplo representa una mala definición de requisitos y como de importante es la fase de definición

1. Introducción a la Programación

4.2 Fase de desarrollo

En esta fase:

- Se **diseñan** estructuras de datos y de los programas.
- Se **escriben y documentan** éstos.
- Se prueba el **software.**

4.3 Fase de mantenimiento

En esta fase:

- **Correcciones** y cambios a partir del desarrollo inicial
- **Adaptación** a nuevos entornos (mantenimiento adaptativo)
- Gestión de **incidencias** y soporte al usuario
- **Documentación continua**

1. Introducción a la Programación

4.2 Documentación

Todas las fases, desde la recolección de requisitos hasta la codificación, deben estar documentadas, de modo que se **puedan consultar** cada vez que se estime oportuno.

En la fase de codificación, además, se deben **añadir en el código**, todos los **comentarios** que se consideren necesarios para una buena comprensión de otro programador ajeno al proyecto.

Finalmente, se deben incluir **manuales de usuario y programador**, con el fin de completar todo el expediente del proyecto desarrollado.

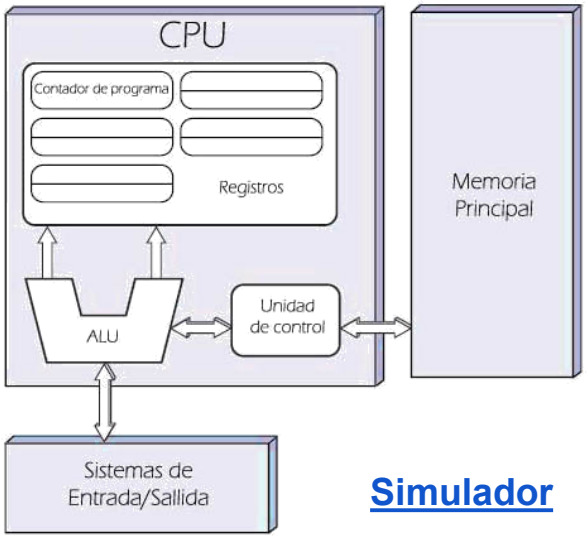
1. Introducción a la Programación

5. Lenguajes de programación

Los orígenes de los lenguajes de programación se encuentran en las **máquinas de cálculo**.

Charles Babbage junto con **Ada Lovelace** son los que diseñan la primera máquina analógica para computar.

Pero es la arquitectura de **John Von Neumann** la que supone un cambio definitivo. Dicha arquitectura divide a un computador digital en **varias partes** que se encargan de hacer tareas distintas.



Simulador

1. Introducción a la Programación

5.1 Lenguajes de bajo y alto nivel.

Los lenguajes de programación pueden clasificarse principalmente en **dos grandes grupos**:

Lenguajes de bajo nivel

- Son aquellos que están **más próximos** al funcionamiento de la máquina.
- El más básico es el **lenguaje máquina**, formado por secuencias de **ceros y unos** que representan directamente instrucciones y datos en memoria.
- Por su **dificultad de lectura**, surgieron los **lenguajes ensambladores**, que utilizan mnemotécnicos e identificadores para facilitar la escritura de instrucciones.
- Tanto el lenguaje máquina como el ensamblador se consideran de bajo nivel porque **dependen fuertemente del hardware** y resultan poco legibles para las personas

Ejemplo código ensamblador

```
L0:  MOV   R1, #a      ;
     MOV   R2, #b      ;

L1:  LD    R3, (R1)    ;
     CMP   R3, #0      ;
     BNE   L3          ;

L2:  MOV   R4, #1      ;
     JMP   L4          ;

L3:  MOV   R4, #0      ;

L4:  ST    (R2), R4    ;
     JMP   L1          ;
```

Ejemplo código binario

```
01010100 01101000 01101001 01110011
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110100 01110101 01110100 01101111
01110010 01101001 01100001 01101100
00100000 01110100 01101111 00100000
01101100 01100101 01100001 01110010
01101110 00100000 01100010 01101001
01101110 01100001 01110010 01111001
00101110 00100000 01001001 00100000
01101000 01101111 01110000 01100101
00100000 01111001 01101111 01110101
00100000 01100101 01101110 01101010
01101111 01111001 00100000 01101001
01110100 00100001
```

1. Introducción a la Programación

5.1 Lenguajes de bajo y alto nivel.

Lenguajes de alto nivel

- Son los que se acercan más al **lenguaje humano** y facilitan la programación.
- Permiten **construir instrucciones** de forma más sencilla y comprensible.
- Son **más seguros**, al reducir errores frecuentes.
- Son portables, es decir, el **mismo código** puede ejecutarse en distintas máquinas.
- Además, son **mucho más legibles** para el programador.

Ejemplo en Java (lenguaje de alto nivel)

```
int cantidad = 6;
int precio = 10;
int total = cantidad * precio;
System.out.println("El precio total es: " + total);

int base = 3;
int altura = 10;
int area = base * altura;
System.out.println("El área del rectángulo es: " + area);
```

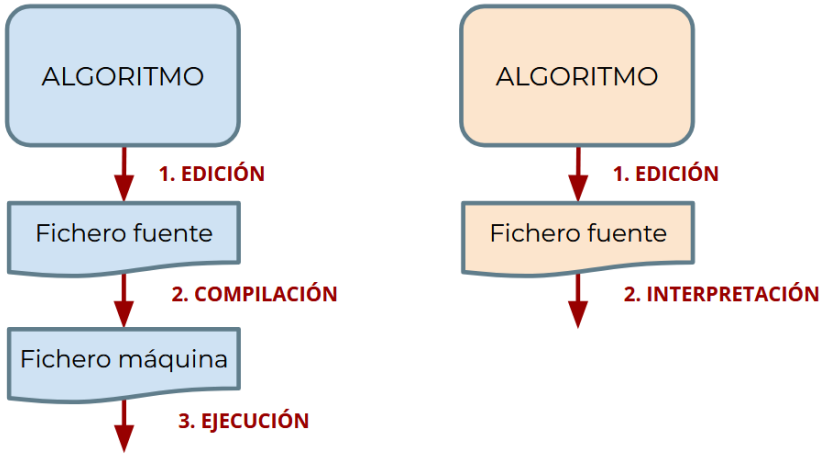
1. Introducción a la Programación

5.2. Compiladores e intérpretes.

Hay dos formas de traducir un programa escrito en un lenguaje de alto nivel al lenguaje máquina: **interpretar y compilar**.

- En la **interpretación** se traduce a lenguaje máquina cada instrucción de manera individual en **tiempo de ejecución** (de una en una).

- En la **compilación** se traducen de **manera global** todas las instrucciones por medio de un **programa (compilador).** En los lenguajes compilados se debe hacer siempre una **compilación previa** para poder ejecutar el programa resultante.



1. Introducción a la Programación

5.3. Lenguajes de programación más populares.

Existe un **ranking** que se basa en la información proporcionada por **TIOBE**, un indicador que mide la popularidad de los lenguajes de programación.

Su clasificación se actualiza mensualmente y se fundamenta en criterios como la **cantidad de ingenieros** especializados, **cursos disponibles** y **proveedores** en todo el mundo.

Enero 2025	Enero 2024	Cambio	Lenguaje	Ratings	Cambio
1	1		Python	23.28%	+9.32%
2	3	↑	C++	10.29%	+0.33%
3	4	↑	Java	10.15%	+2.28%
4	2	↓	C	8.86%	-2.59%
5	5		C#	4.45%	-2.71%
6	6		JavaScript	4.20%	+1.43%
7	11	⇈	Go	2.61%	+1.24%
8	9	↓	SQL	2.41%	+0.95%
			Visual Basic	2.37%	+0.77%
			Fortran	2.04%	+0.94%

Índice

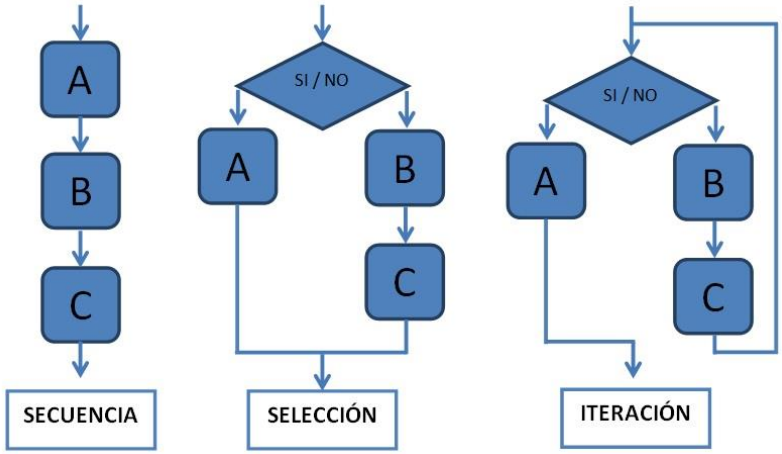
1. Introducción a la Programación

5.4 Lenguajes de programación estructurados y orientado a objetos.

Programación estructurada

Se trata de un paradigma de programación que organiza el **código en subrutinas** y utiliza únicamente **tres estructuras de control fundamentales**:

- **Secuencia**
- **Selección (por ejemplo, if)**
- **Iteración (como for o while)**



1. Introducción a la Programación

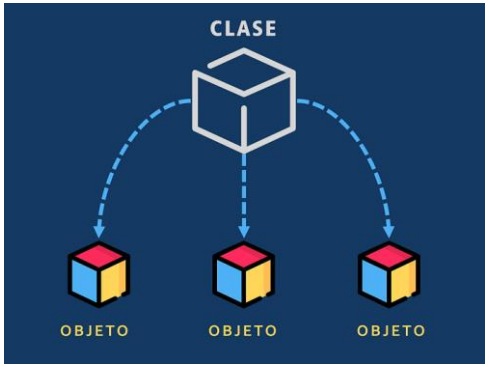
5.5 Lenguajes de programación estructurados y orientado a objetos.

Programación orientada a objetos (POO)

Se trata de un paradigma de programación que estructura el software alrededor de **objetos**, los cuales combinan

- **datos (atributos)**
- **comportamiento (métodos).**

En lugar de centrarse en funciones, se enfoca en **modelar entidades del mundo real** como objetos que interactúan entre sí. Este estilo promueve la **reutilización del código**, facilita el mantenimiento y es ideal para proyectos grandes y complejos.



1. Introducción a la Programación

5.6 Elementos de un lenguaje de programación.

Hay una serie de elementos que están presentes en casi todo lenguaje de programación:

- **Tipos de datos:** enteros, reales, cadena de caracteres…
- **Palabras reservadas:** conjunto de **palabras propias** del lenguaje de programación. No pueden utilizarse, por ejemplo, para nombrar variables o constantes.
- **Operadores:** Crean instrucciones utilizando cálculos matemáticos (operadores aritméticos), comparaciones (operadores relacionales) u operaciones lógicas (operadores lógicos).
- **Constantes y variables:** Posiciones de memoria que almacenan los datos con los que trabaja nuestro programa. Las constantes no pueden variar a lo largo de la ejecución del programa, mientras que las variables sí.
- **Estructuras de control:** Secuenciales, condicionales o bucles.
- **Funciones:** Bloques de código empaquetados con una cabecera que realizan una tarea específica, lo que permite, entre otras cosas, la reutilización del código.
- **Comentarios:** Bloques dentro del código que no se ejecutan. Su función es la de aclarar partes del programa.

1. Introducción a la programación

CASO PRÁCTICO

Lenguajes interpretados y compilados

Enunciado

1. Investiga por internet **lenguajes compilados e interpretados y enumera varios de cada tipo.**
2. Para cada uno, responde:
   - ¿En qué **tipo de proyectos** se usa habitualmente?
   - ¿Qué **ventaja** principal tiene?
   - ¿Qué **desventaja principal** tiene?
3. Investiga en qué consiste el **modelo cliente-servidor** de manera muy básica e indica donde se utilizan los lenguajes que has indicado anteriormente.
4. Investigar qué **lenguajes de programación** utilizan las aplicaciones más usadas en la actualidad (Instagram, videojuegos, Spotify, tik-tok, etc.).

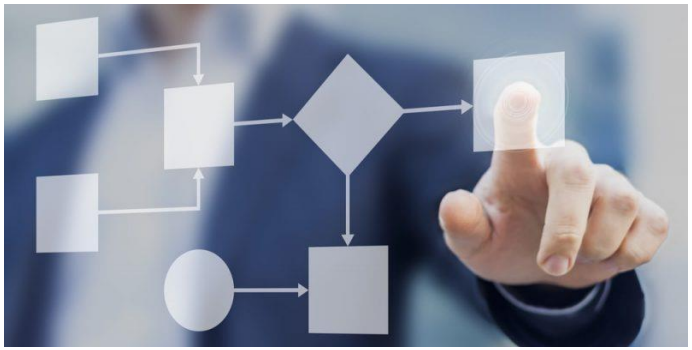
1. Introducción a la Programación

6. Representación de algoritmos.

Para representar un algoritmo se suele utilizar algún método que permite **independizarlo del lenguaje de programación** utilizado. Esto nos permitirá codificar más tarde a través de cualquier lenguaje de programación.

Existen diversas técnicas, podemos destacar:

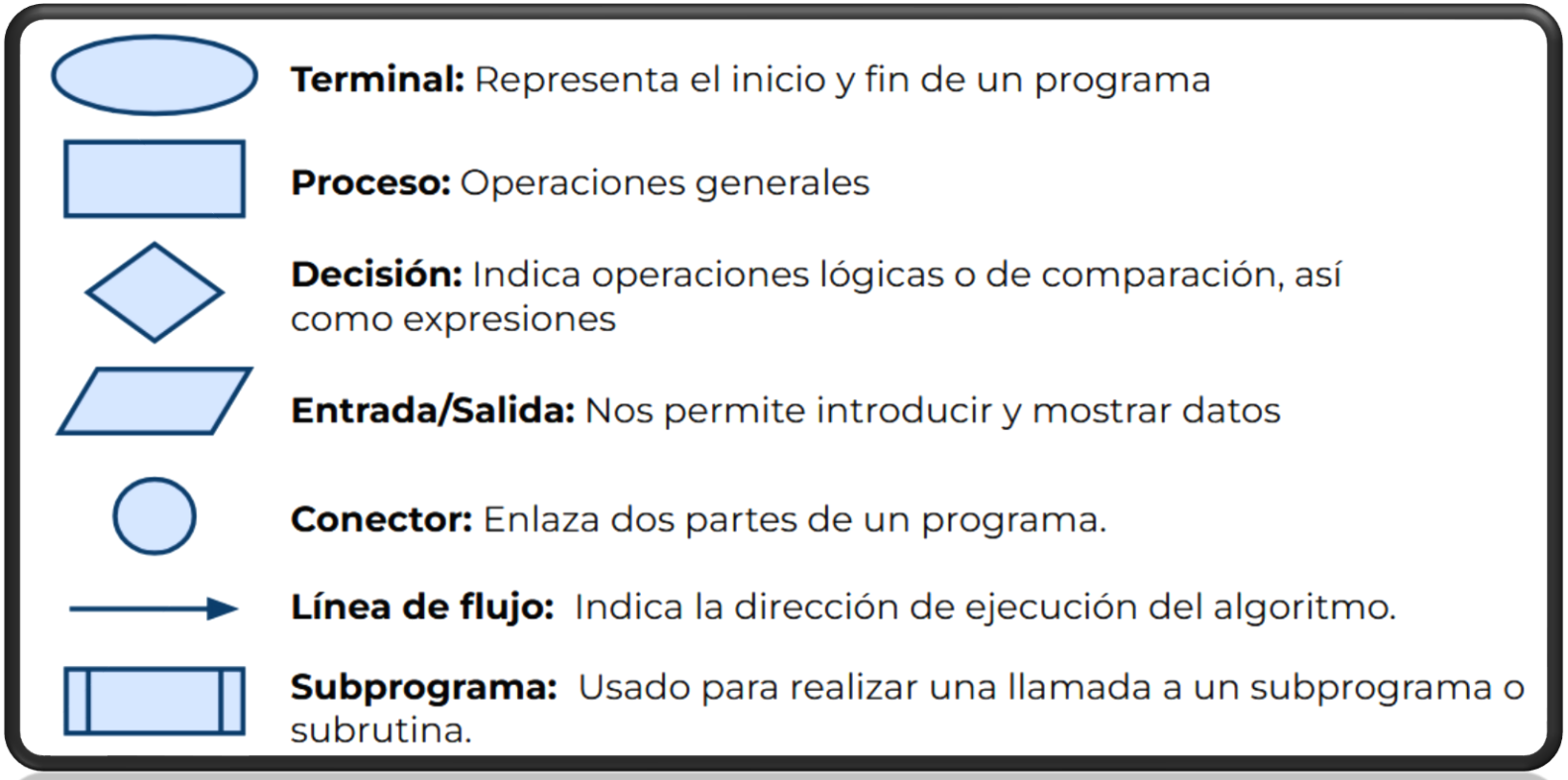
- **Diagramas de flujo (ordinogramas):** Utilizan símbolos gráficos estándar y escriben los pasos del algoritmo dentro de los símbolos. A través de las líneas de flujo se indican la secuencia en que se deben ejecutar
- **Lenguajes de descripción de algoritmos (Pseudocódigo).** Es un lenguaje entre el lenguaje natural y lenguaje de programación



1. Introducción a la Programación

6.1. Diagramas de flujo.

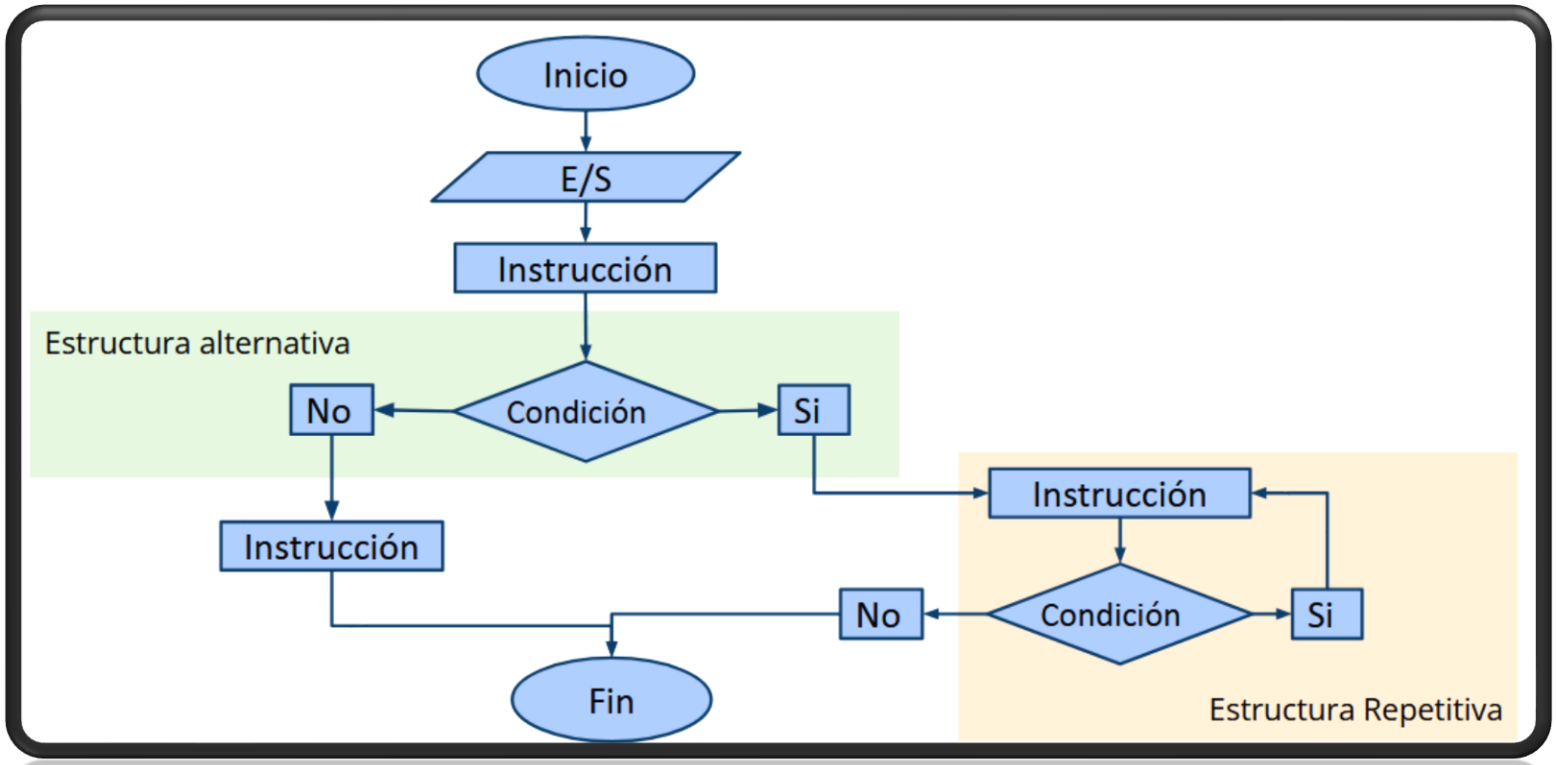
Estos son los **símbolos** más utilizados en los diagramas de flujo:



1. Introducción a la Programación

6.1. Diagramas de flujo.

En la siguiente imagen se muestra los elementos principales usados en la programación estructurada, a través de un diagrama de flujo.



1. Introducción a la Programación

6.2. Pseudocódigo.

Es un lenguaje para la descripción de algoritmos que procura mantener las propiedades de los diagramas de flujo: **sencillez, claridad, normalización y flexibilidad**.

- Su notación se basa en el **lenguaje natural**.
- La estructura general del pseudocódigo está dividida en **dos partes**:
  - **Nombre y entorno del programa:** tipos, nombre de las variables y constantes que se utilizan.
  - **Desarrollo ordenado y estructurado del algoritmo:** esta segunda parte se corresponde con el diagrama de flujo.

```
PROGRAMA NombreDelPrograma
VARIABLES
    Declaración de variables
ALGORITMO
    Cuerpo del programa
FIN
```

1. Introducción a la Programación

6.3. Ejemplos de diagrama de flujo y pseudocódigo.

En la siguiente imagen se muestre un ejemplo de algoritmo representado de las dos formas indicadas anteriormente.

Diseñar un algoritmo que obtenga el área de un rectángulo

```
1  Algoritmo Rectangulo
2      Definir base,altura Como Real;
3      Escribir "Introduce la base:";
4      Leer base;
5      Escribir "Introduce la altura:";
6      Leer altura;
7      area ← base * altura;
8      Escribir "El área es " ,area;
9  FinAlgoritmo
```

Pseudocódigo

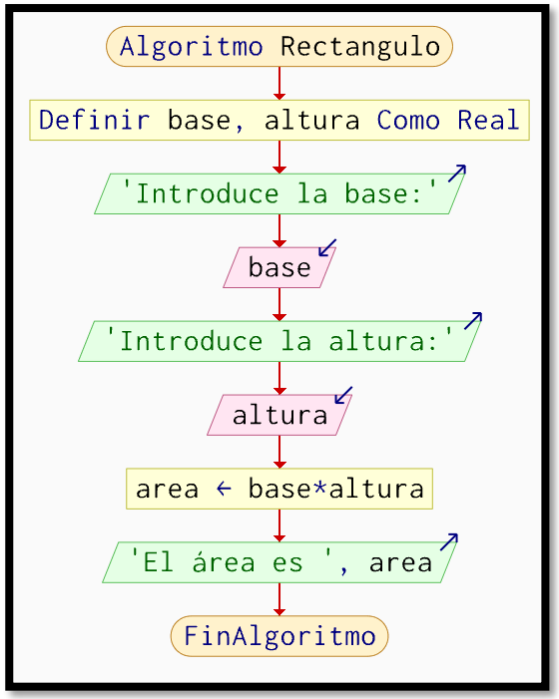


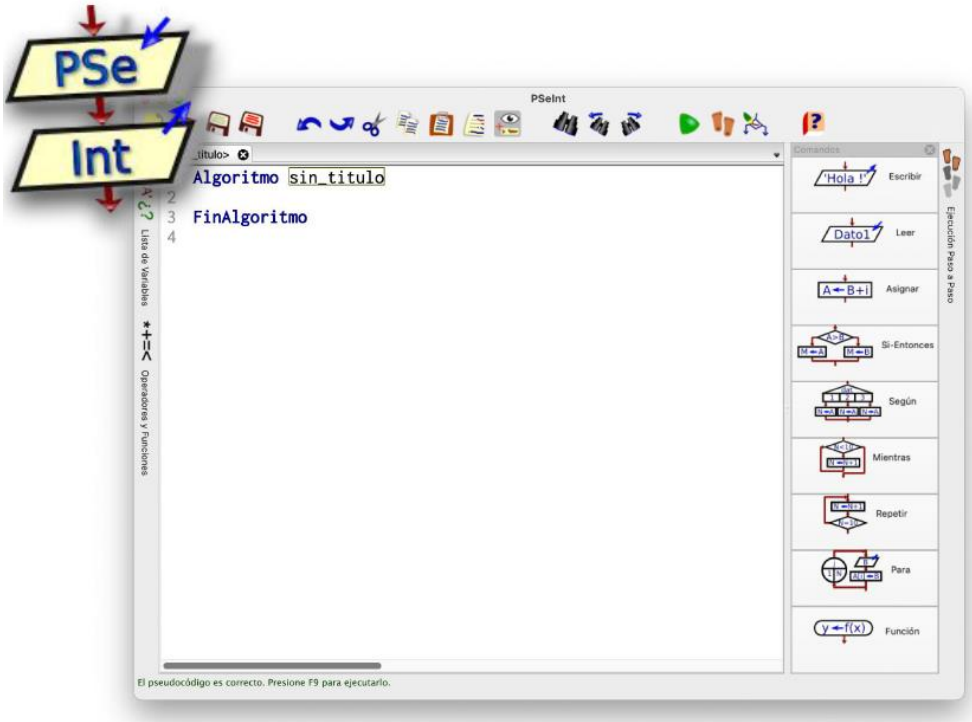
Diagrama de flujo

1. Introducción a la Programación

7. Software Pseint

PSeInt es un editor e intérprete de **pseudocódigo y diagramas de flujo multiplataforma** que permite trabajar tanto con algoritmos sencillos como complejos.

Se trata de una aplicación bajo licencia **GPLv2** y puede descargarse desde **aquí.**

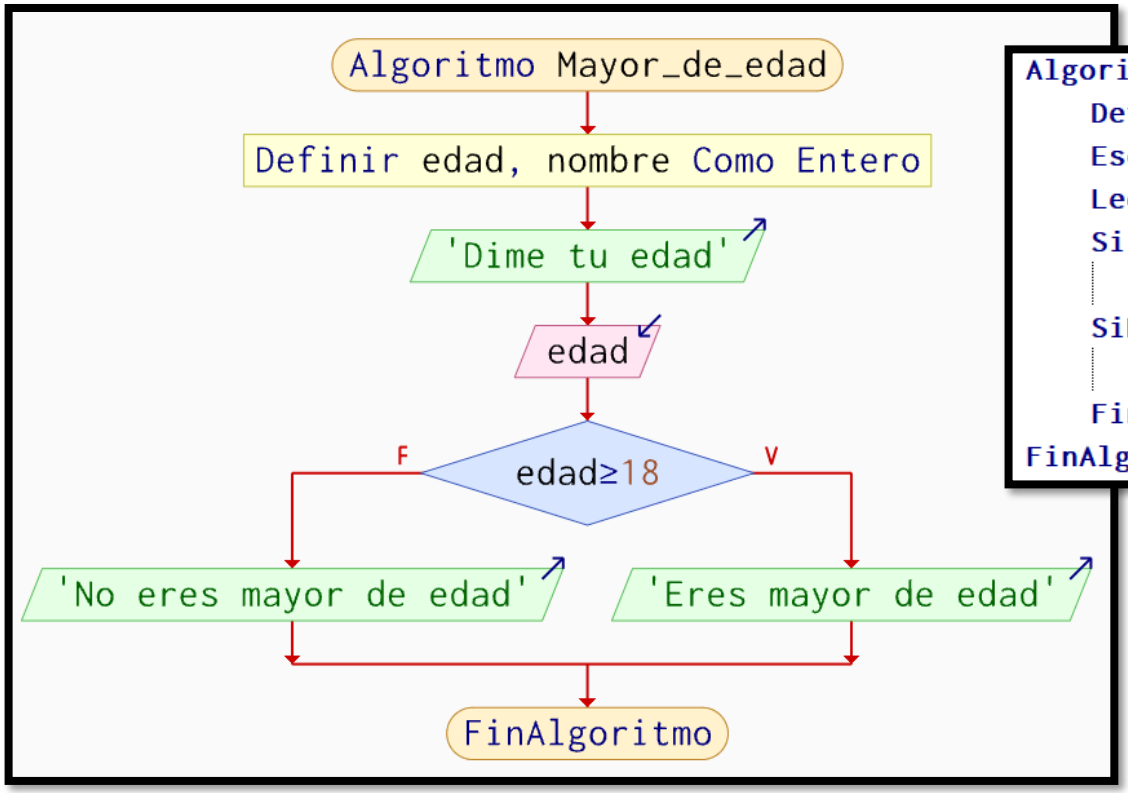


El propio programa dispone de una **ayuda muy completa** que explica de manera clara las ideas principales de la programación estructurada, junto con **numerosos ejemplos**.

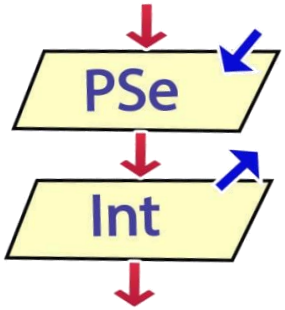
1. Introducción a la Programación

7.1 Ejemplo de estructura alternativa en Pseint

Se muestra un ejemplo para indicar si eres mayor de edad o no.




```
Algoritmo Mayor_de_edad
    Definir edad, nombre Como Entero
    Escribir 'Dime tu edad'
    Leer edad
    Si edad≥18 Entonces
        Escribir 'Eres mayor de edad'
    SiNo
        Escribir 'No eres mayor de edad'
    FinSi
FinAlgoritmo
```

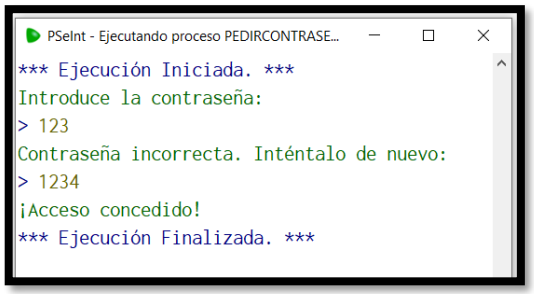


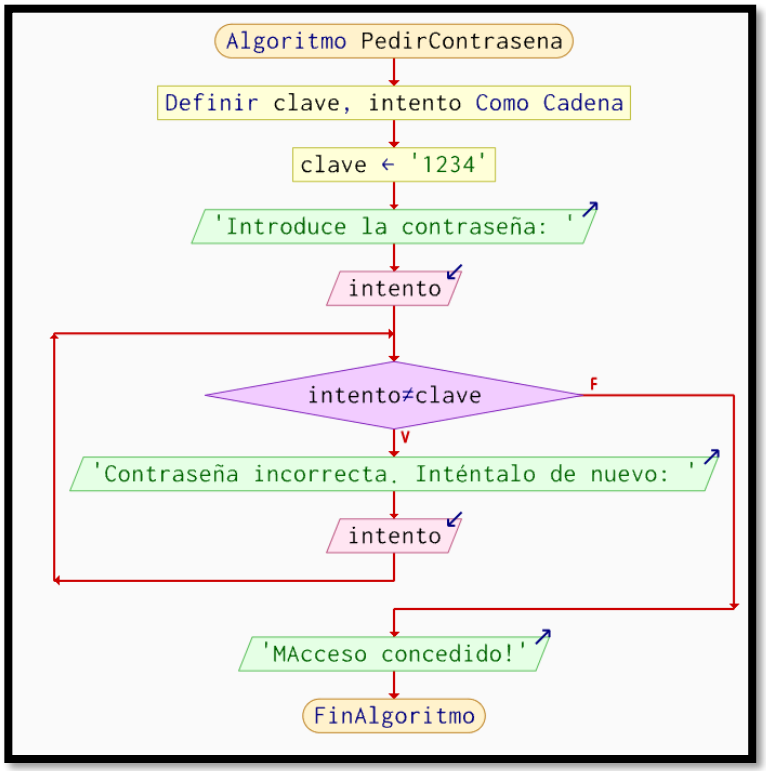
1. Introducción a la Programación

7.2 Ejemplo de estructura repetitiva en Pseint

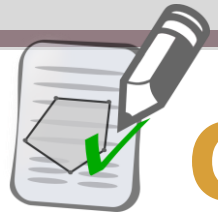
Se muestra un ejemplo para averiguar si la contraseña es correcta o no.

```
Algoritmo PedirContrasena
    Definir clave, intento Como Cadena
    clave ← "1234"

    Escribir "Introduce la contraseña: "
    Leer intento

    Mientras intento ≠ clave Hacer
        Escribir "Contraseña incorrecta. Inténtalo de nuevo: "
        Leer intento
    FinMientras

    Escribir "¡Acceso concedido!"
FinAlgoritmo
```

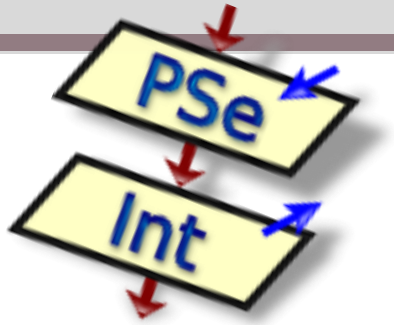




1. Introducción a la programación



CASO PRÁCTICO



Realiza los siguientes algoritmos en Pseint

1. Escribe un algoritmo en pseudocódigo que, dada una **distancia en millas marinas**, las escriba en metros, se debe tener en cuenta que 1 milla marina es igual a 1852 metros.
2. Diseñar un algoritmo expresado en pseudocódigo que, introduciendo por teclado el **precio base de un producto** y el porcentaje de iva, escriba el precio total del producto.
3. Diseñar un algoritmo que determine si un **número es par**
4. Leer dos números enteros, A y B. Calcular C = A*B mediante **sumas sucesivas** e imprimir el resultado.
5. Leer dos **números A y B** e intercambiar el valor de sus variables
6. Una persona se encuentra en el **kilómetro 70** de una carretera, otra se encuentra en el **km 150**, los coches tienen sentido opuesto y tienen la misma velocidad. Realizar un programa para determinar en **qué kilómetro** de esa carretera se encontrarán. Utilizar estructura repetitiva.

1. Introducción a la programación

CASO PRÁCTICO

- REALIZA LA ACTIVIDAD 1 DE LA MOODLE -

moodle centros

1. Introducción a la programación