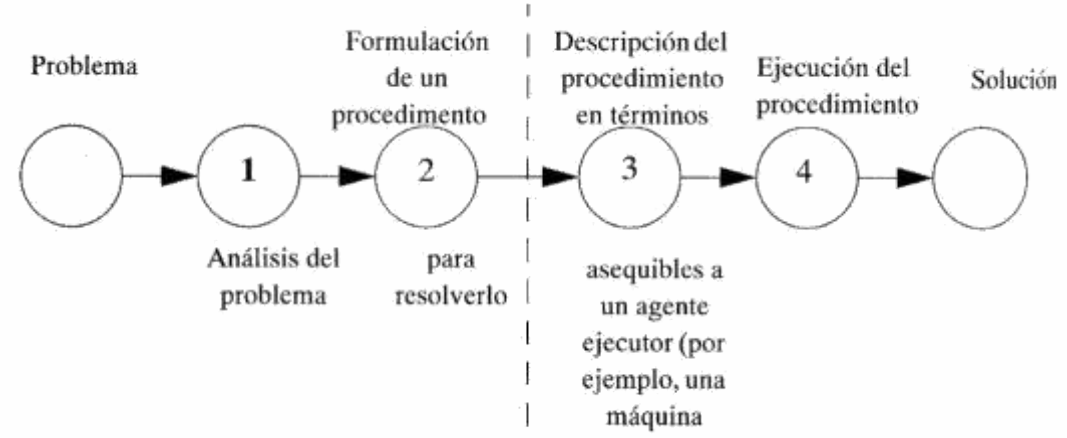
TEMA 2. CONCEPTOS BÁSICOS DE ALGORÍTMICA

2.1 Definición de Algoritmo

Un **algoritmo** *es* una secuencia precisa de operaciones (pasos) que resuelven un problema en un tiempo finito.

∃ Solución(problema) ⇔ ∃ ALGORITMO(Solución(problema))

Pasos para la resolución de un problema:



Los algoritmos son independientes del lenguaje de programación y del ordenador que los ejecuta. Se pueden expresar en multitud de lenguajes y ejecutarse en ordenadores distintos.

2.1.1 Propiedades de los algoritmos

a) Siempre **debe terminar**.

b) Debe contener instrucciones concretas, **sin ninguna ambigüedad**.

c) Todos sus pasos deben ser **simples** y **tener un orden** definido.

d) **Debe funcionar** sean cuales sean los datos de entrada.

e) Debe ser **eficiente y rápido** 🡺 Hay que **Optimizar** 🡸 Para un problema existen múltiples soluciones, y debemos escoger aquella que consuma menos tiempo y recursos.

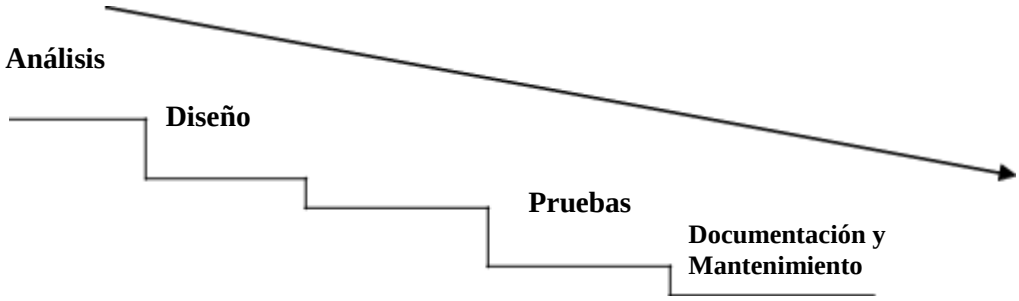
f) Es **independiente** de la **máquina** y del **lenguaje de programación** que se vaya a utilizar. Un algoritmo puede **implementarse** (escribirse) en cualquier lenguaje de programación.

2.2 ¿Qué es un programa?

Un programa es la expresión (transcripción) de un algoritmo en un lenguaje de programación, capaz de ser procesado por un ordenador tras su compilación y linkado y que controla el funcionamiento de un ordenador a la hora de resolver un problema.

2.2.1 Cómo se construye un programa.

El proceso de elaboración de un programa, conlleva varias etapas:



- **Fase de Análisis:** decidir *qué es lo que tenemos que hacer*.
- **Fase de Diseño (desarrollo de la solución):** se define *cómo vamos a hacerlo*. ⇗ Obtención del Algoritmo ⇗ Se utilizará el **Diseño Descendente** o **TOP-DOWN*:*** Un problema complejo se resuelve dividiendo el problema en subproblemas, y así sucesivamente hasta que la resolución de cada subproblema sea fácilmente programable.
- **Fase de Codificación**: ⇗ Implementación del Algoritmo en el lenguaje de programación más adecuado ⇗ Obtención del **Programa**
- **Fase de Pruebas**: No basta que el programa esté terminado ⇗ Hay que comprobar que el programa NO falla y funciona perfectamente en todos los casos posibles que se puedan presentar.
- **Fase de Documentación y Mantenimiento**: ⇗ Se elabora la documentación del programa, y se realizan las actualizaciones oportunas que se vayan necesitando.

TODAS ESTAS FASES HAY QUE REALIZARLAS CON SUMO CUIDADO, PUESTO QUE UN ERROR EN UNA DE ELLAS, PUEDE CONLLEVAR LA VUELTA ATRÁS EN TODO EL PROCESO.

Resumen: Proceso de creación de un programa

- Planteamiento del problema a resolver. Antes de nada debemos conocer perfectamente el problema y los resultados a obtener.
- Representación de los datos. Escoger los tipos de datos a usar.
- Diseño de un algoritmo.
- Comprobación y optimización de algoritmos. Debemos asegurarnos que el algoritmo realiza la tarea correctamente.
- Codificación del programa. Debemos transcribir el algoritmo a un lenguaje de programación concreto para que pueda ser utilizado.
- Depuración del programa. El programa debe estar libre de errores.
- Documentación del programa.

2.3 Definición y uso de herramientas para describir soluciones

Para representar los algoritmos existen dos métodos principales:

- El pseudocódigo
- El diagrama de flujo.

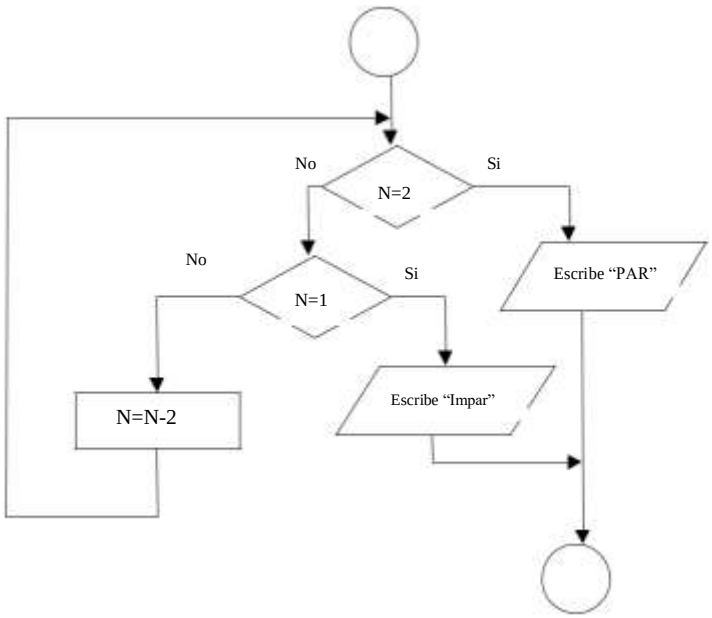
Mientras que el pseudocódigo permite enunciar el algoritmo, los diagramas de flujo (organigramas) permiten visualizarlo de forma gráfica.

2.3.1 Diagramas de flujo (organigrama)

Es una **representación gráfica** de un algoritmo mediante una serie de **símbolos**, que contienen en su interior los pasos del algoritmo, y unas **flechas** que los unen indicando la secuencia (orden) en la que se deben ejecutar. Los símbolos representan acciones y las flechas el flujo del algoritmo.

La descripción de las funciones se puede realizar de forma narrativa, usando un lenguaje natural (conviene que sea parecido al pseudocódigo)

Símbolos principales	Función
	Terminal (representa el comienzo, «inicio», y el final, «fin», de un programa. Puede representar también una parada o interrupción programada que sea necesario realizar en un programa).
	Entrada/Salida (cualquier tipo de introducción de datos en la memoria desde los periféricos, «entrada», o registro de la información procesada en un periférico, «salida»).
	Proceso (cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transferencia, etc.).
NO SÍ	Decisión (indica operaciones lógicas o de comparación entre datos —normalmente dos— y en función del resultado de la misma determina cuál de los distintos caminos alternativos del programa se debe seguir; normalmente tiene dos salidas —respuestas SÍ o NO— pero puede tener tres o más, según los casos).
	Decisión múltiple (en función del resultado de la comparación se seguirá uno de los diferentes caminos de acuerdo con dicho resultado).
	Conector (sirve para enlazar dos partes cualesquiera de un ordinograma a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama).
	Indicador de dirección o línea de flujo (indica el sentido de ejecución de las operaciones).
	Línea conectora (sirve de unión entre dos símbolos).
	Conector (conexión entre dos puntos del organigrama situado en páginas diferentes.
	Llamada subrutina o a un proceso predeterminado (una subrutina es un módulo independiente del programa principal, que recibe una entrada procedente de dicho programa, realiza una tarea determinada y regresa, al terminar, al programa principal).



DFD para indicar si un número es Par o Impar.

2.3.2 El pseudocódigo

El **pseudocódigo** es la representación **narrativa** (no hay reglas sintácticas estrictas) de un algoritmo, escrita en **lenguaje natural** utilizando las **estructuras de control** típicas de algún Lenguaje de Programación y algunos símbolos **algebraicos**.

La utilización de pseudocódigo presenta las ventajas de ser más compacto que un organigrama, ser más fácil de escribir y ser más fácil de transcribir a un lenguaje de programación.

Las estructuras de control deciden qué camino hay que seguir en función de una condición. Son las siguientes:

1. **Estructura secuencial:** consiste en colocar una instrucción tras otra, de manera que se van ejecutando de arriba abajo.
2. **Estructura selectiva o condicional** (**si**, **si no**): permiten ejecutar un conjunto de instrucciones u otras en función de si se cumple o no una condición
3. **Estructura iterativa o de repetición** (**mientras**, **repetir**, **para**): permite repetir una instrucción o grupo de ellas un nº fijo de veces o mientras (o hasta que) una condición sea cierta.

1. Estructura secuencial

Pseudocódigo de un algoritmo que calcule la media de tres números:

```
Leer (n1);
Leer (n2);
Leer (n3);
suma = n1 + n2 + n3;
media = suma / 3;
escribir (media);
```

El orden en el que se realizan las operaciones es importante: no puede calcularse la media sin antes haber leído los números.

2. Estructura selectiva o condicional

El formato de esta estructura es el siguiente:

```
si (se cumple la condición)
        inicio
            conjunto de acciones;
        fin
sino
        inicio
            conjunto de acciones;
        fin
finsi
```

Es decir, primero se examina la condición: si resulta verdadera, se ejecutan las acciones asociadas al **si**, en caso contrario se ejecutan las acciones asociadas al **sino**.

La instrucción **si no** no es obligatoria en una estructura condicional (si no queremos hacer nada en caso que la condición sea falsa).

Algoritmo que calcula la media de 3 nº y devuelve su raíz cuadrada.

```
Leer (n1);
Leer (n2);
Leer (n3);
suma = n1 + n2 + n3;
 media = suma / 3;
si (media >= 0)
inicio
  raiz = RaizCuadrada (media);
  escribir (raiz);
fin
si no
escribir ("No se puede hallar la raiz cuadrada");
```

Antes de hallar la raíz cuadrada hay que ver que la media no es **negativa:**

La estructura condicional permite anidar unas instrucciones en otras. Supongamos que queremos calcular la nota media de la siguiente forma:

- Si teoría >= 5 y practica < 5: media = 0.4 x teoría + 0.6 x práctica
- Si practica >= 5 y teoría < 5: media = 0.6 x teoría + 0.4 x práctica
- En cualquier otro caso se calculara su media normalmente.

```
Leer (teoria);  Leer (practica);
si (teoria >= 5)
inicio
    si (practica < 5)
        media = 0.4 * teoria + 0.6 * practica;
    si no
        media = (teoria + practica) / 2;
fin
si no
inicio
    si (practica >= 5)
        media = 0.6 * teoria + 0.4 * practica;
    si no
        media = (teoria + practica) / 2;
fin
escribir("La media es ", media);
```

Otra posible solución sería:

```
Leer (teoria);  Leer (practica);
media = (teoria + practica) / 2;
si (teoria >= 5)
inicio
        si (practica < 5)
        inicio
            media = 0.4 * teoria + 0.6 * practica;
        fin
fin
si no
inicio
        si (practica >= 5)
            media = 0.6 * teoria + 0.4 * practica;
fin
escribir("La media es ", media);
```

Otra forma de resolverlo es usando el operador **y** en las condiciones. Este operador permite combinar dos condiciones de manera que solo será verdad si ambas condiciones se cumplen:

```
Leer (teoria);  Leer (practica);
si (teoria >= 5 y practica < 5)
    media = 0.4 * teoria + 0.6 * practica;
si no
inicio
    si (practica >= 5 y teoria < 5)
        media = 0.6 * teoria + 0.4 * practica;
    si no
        media = (teoria + practica) / 2;
fin
escribir("La media es ", media);
```

Además del operador **y** también existe el operador **o** el cual permite ejecutar una acción determinada si se verifica una de las condiciones.

```
Leer (edad);
si (edad < 16 o edad >= 65)
    escribir("No puedes trabajar");
si no
    escribir("Puedes trabajar");
```

```
Leer (edad);
si (edad >= 16 y edad < 65)
    escribir("Puedes trabajar");
si no
    escribir("No Puedes trabajar");
```

Es decir, solo trabajan los que tengan 16 o más años y menos de 65.

3. Estructura iterativa o de repetición.

Esta estructura presenta una serie de variantes:

a. Estructura mientras

Esta estructura permite repetir un conjunto de instrucciones 0 o más veces, ya que la condición se verifica antes de entrar en el bucle. El formato de esta estructura es el siguiente:

```
mientras (se cumpla la condición)
    inicio
        conjunto de acciones;
    fin
```

Es decir, primero se examina la condición: si resulta falsa, se pasa directamente a la instrucción que haya tras el fin, de manera que nos saltamos todas las instrucciones que haya dentro del bucle.

b. Estructura repetir … mientras

Esta estructura evalúa la condición una vez realizada la acción. Por tanto, las instrucciones que están dentro se ejecutan al menos una vez. El formato de esta estructura es el siguiente:

```
repetir
    inicio
    conjunto de acciones;
    fin
mientras (se cumpla la condición);
```

Ej: algoritmo que lee por teclado unos números (hasta que introduzcamos un número negativo) y calcula su media.

```
suma = 0;  n = 0;
escribir("Dame un nº no negativo");   leer (numero);
mientras (numero >= 0)
inicio
    suma = suma + numero;
    n = n + 1;
    escribir("Dame un nº no negativo");      leer (numero);
fin
si (n > 0)
inicio
    media = suma / n;
    escribir("La media es ", media);
fin
si no
    escribir ("La media es 0");
```

Ej: Algoritmo anterior usando el **repetir**

```
suma = 0; n = 0;
repetir
inicio
    escribir("Dame un nº no negativo");    leer (numero);
    si (numero >= 0)
    inicio
        suma = suma + numero;
        n = n + 1;
    fin
fin
mientras (numero >= 0);
si (n > 0)
inicio
    media = suma / n;
    escribir("La media es ", media);
fin
si no
    escribir("La media es 0");
```

c. Estructura para

Permite realizar una acción un número determinado de veces.

El formato de esta estructura es el siguiente:

```
para variable de inicio a fin
inicio
   conjunto de acciones;
fin
```

En cada iteración del bucle variable va tomando distintos valores comprendidos entre inicio y fin. En la primera iteración toma el valor inicio, en la segunda inicio+1, y así sucesivamente hasta el valor fin.

Ej: Algoritmo que pide 20 números por teclado y calcula su media.

```
suma = 0;
para n de 1 a 20
inicio
   escribir("Introduzca nº", n);       leer (numero);
   suma = suma + numero;
fin
media = suma / (n-1);
escribir("La media es ", media);
```

Restamos 1 a n ya que se sale del bucle para cuando la variable n sobrepasa el valor 20.

La estructura para puede sustituirse por **mientras** o por **repetir**:

```
suma = 0;
n = 0;
mientras (n < 20)
inicio
   escribir("Introduzca nº", n+1);   leer (numero);
   suma = suma + numero;
   n = n + 1;
fin
media = suma / n;
escribir(" La media es ", media);
```

¿Cual de las tres variantes usar ante un determinado problema?:

```
si (el bucle tiene que ejecutarse un numero fijo de veces)
    Utilizar la estructura para;
si no
inicio
    si (el bucle debe ejecutarse como mínimo una vez)
        Utilizar la estructura repetir...mientras;
    si no
        Utilizar la estructura mientras;
fin
```

Un error muy común con las estructuras de repetición consiste en poner mal la condición de finalización u olvidarse de incrementar el contador, dando lugar a bucles infinitos (bucles que no acaban nunca).

```
suma = 0;
n = 1;
repetir
inicio
    leer (numero);
    suma = suma + numero;
fin
mientras (n <= 20);
media = suma / (n-1);
```

Este bucle nunca finaliza ya que olvidamos incrementar la variable n.

```
suma = 0;
n = 1;
repetir
inicio
    leer (numero);
    suma = suma +
    numero; n = n - 1;
fin
mientras (n <= 20);
media = suma / (n-1);
```

En este caso, la n siempre es menor de 20, ya que la decrementamos en vez de incrementarla.

Ej: Calcular la media de una serie de nº positivos dados por teclado. Un valor de 0, como entrada, indicará el final de la serie de números.

Pseudocódigo

```
contador = 0;
suma = 0;
leer(numero);
mientras (numero <> 0)
inicio
    suma = suma + numero;
    contador = contador + 1;
    leer(numero);
fin
si (contador <> 0)
    media = suma / contador;
sino
    media = 0;
escribir(media);
```

Ej: Calcular la suma de los N primeros números impares, siendo N un nº dado por teclado.

```
suma = 0;
c = 1;
impar = 1;
leer (n);
mientras (c <= n)
inicio
    suma = suma +
    impar; impar = impar
    + 2; c = c + 1;
fin
escribir (suma);
```

```
suma= 0;
impar = 1;
leer (n);
para c de 1 a n
inicio
    suma = suma + impar;
    impar = impar + 2;
fin
escribir(suma);
```

EJEMPLOS

Se desea diseñar un algoritmo para saber si un número es primo o no.

Un número es primo si sólo puede dividirse por sí mismo y por la unidad (es decir, no tiene más divisores que él mismo y la unidad). Por ejemplo, 9, 8, 6, 4, 12, 16, 20, etc.., no son primos, ya que son divisibles por números distintos a ellos mismos y a la unidad. Así, 9 es divisible por 3, 8 lo es por 2, etc.

El algoritmo de resolución del problema pasa por dividir sucesivamente el número por 2, 3, 4, etc.

```
1. Inicio.
2. Poner X igual a 2 (x = 2, x variable que representa a
   los divisores del número que se busca N).
3. Dividir N por X (N/X).
4. Si el resultado de N/X es entero, entonces N es un
   número primo y bifurcar al punto 7; en caso contrario,
   continuar el proceso.
5. Suma 1 a X (X ← X + 1).
6. Si X es igual a N, entonces N es un número primo; en
   caso contrario, bifurcar al punto 3.
7. Fin.
```

Pseudocódigo

```
leer(N);
X=2;
mientras (mod(N / X)<> 0 y X<N)
inicio
  X=X+1;
fin
si (X < N)
  escribir(" N no es primo");
sino
  escribir("N es primo");
```

Realizar la suma de todos los números pares entre 2 y 1.000.

El problema consiste en sumar `2 + 4 + 6 + 8 ... + 1.000`. Utilizaremos las palabras `SUMA` y `NUMERO` (*variables*, serán denominadas más tarde) para representar las sumas sucesivas `(2+4)`, `(2+4+6)`, `(2+4+6+8)`, etc. La solución se puede escribir con el siguiente algoritmo:

```
1. Inicio.
2. establecer SUMA a 0.
3. establecer NUMERO a 2.
4. Sumar NUMERO a SUMA. El resultado será el nuevo valor
   de la suma (SUMA).
5. Incrementar NUMERO en 2 unidades.
6. Si NUMERO =< 1.000 bifurcar al paso 4;
7. en caso contrario, escribir el último valor de SUMA y
   terminar el proceso.
8. Fin.
```

Pseudocódigo

```
SUMA = 0;
N=2;
mientras (N < = 1000)
inicio
 SUMA = SUMA +
 N; N=N+2;
fin
escribir(SUMA);
```