# Unidad 01: Algoritmos, programas y metodología de la programación

## 1.1 Introducción

En este capítulo se introduce la metodología que hay que seguir para la resolución de problemas con computadoras. La resolución de un problema con una computadora se hace escribiendo un programa, que exige al menos los siguientes pasos:

1. Definición o análisis del problema.
2. Diseño del algoritmo.
3. Transformación del algoritmo en un programa.
4. Ejecución y validación del programa.

Uno de los objetivos fundamentales de esta unida es el aprendizaje y diseño de los algoritmos. Este capítulo introduce al lector en el concepto de algoritmo y de programa, así como a las herramientas que permiten “dialogar” al usuario con la máquina: **los lenguajes de programación**.

## 1.2 Resolución de problemas con computadoras: fases

El proceso de resolución de un problema con una computadora conduce a la escritura de un programa y a su ejecución. Las fases de resolución de un problema con computadora son: *Análisis del problema. Diseño del algoritmo. Codificación. Compilación y ejecución. Verificación. Depuración. Mantenimiento. Documentación.*

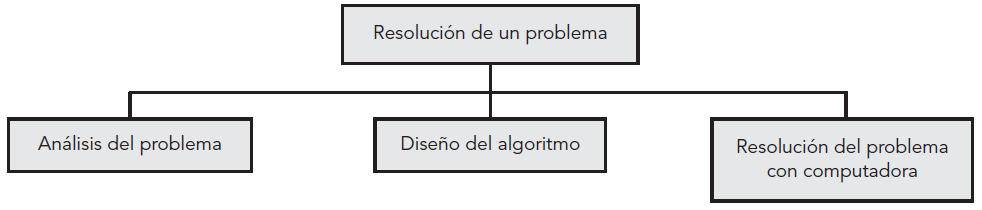
* **Análisis.** El problema se analiza teniendo presente la especificación de los requisitos dados por el cliente de la empresa o por la persona que encarga el programa.
* **Diseño.** Una vez analizado el problema, se diseña una solución que conducirá a un algoritmo que resuelva el problema.
* **Codificación (implementación).** La solución se escribe en la sintaxis del lenguaje de alto nivel y se obtiene un programa fuente que se compila a continuación.
* **Ejecución, verificación y depuración.** El programa se ejecuta, se comprueba rigurosamente y se eliminan todos los errores que puedan aparecer.
* **Mantenimiento.** El programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.
* **Documentación.** Escritura de las diferentes fases del ciclo de vida del software.

**Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.**

Ejemplos de algoritmos son: instrucciones para montar en una bicicleta, hacer una receta de cocina, obtener el máximo común divisor de dos números, etc. Los algoritmos se pueden expresar por fórmulas, diagramas de flujo o N-S y pseudocódigos. Esta última representación es la más utilizada para su uso con lenguajes estructurados.

## 1.2.1 Análisis del problema

Esta fase requiere una clara definición, donde se contemple exactamente lo que debe hacer el programa y la solución deseada.



Para poder identificar y definir bien un problema es conveniente responder a las siguientes preguntas:

* ¿Qué entradas se requieren? Tipos de datos.
* ¿Cuál es la salida deseada? También, tipos de datos
* ¿Qué método produce la salida deseada?
* Requisitos adicionales para la solución.

## 1.2.2 Diseño del problema

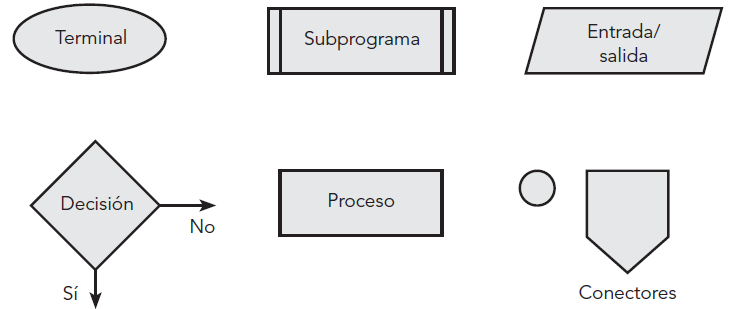
En la etapa de análisis del proceso de programación se determina **qué hace** el programa. En la etapa de diseño se determina **cómo hace** el programa la tarea solicitada. Los métodos más eficaces para el proceso de diseño se basan en el conocido divide y vencerás. Es decir, la resolución de un problema complejo se realiza dividiendo el problema en subproblemas y a continuación dividiendo estos subproblemas en otros de nivel más bajo, hasta que pueda ser implementada una solución en la computadora. Este método se conoce técnicamente como **diseño descendente.**

El proceso que convierte los resultados del **análisis del problema en un diseño modular** con refinamientos sucesivos que permitan una posterior traducción a un lenguaje se denomina **diseño del algoritmo**. El diseño del algoritmo es independiente del lenguaje de programación en el que se vaya a codificar posteriormente.

## 1.2.3 Herramientas gráficas y alfanuméricas

Dos herramientas: Diagramas de flujos y Pseudocódigos

* Un diagrama de flujo es una representación gráfica de un algoritmo. Los símbolos más frecuentemente empleados se muestran en la figura.



* El pseudocódigo es una herramienta de programación en la que las instrucciones se escriben en palabras similares al inglés o español, que facilitan tanto la escritura como la lectura de programas. En esencia, el pseudocódigo se puede definir como un lenguaje de especificaciones de algoritmos.

*Ejemplo de Pseudocódigo:* Calcular el valor de la suma 1 + 2 + 3 + ... + 100. **Algoritmo** Se utiliza una variable Contador como un contador que genere los sucesivos números enteros, y Suma para almacenar las sumas parciales 1, 1 + 2, 1 + 2 + 3 ...

1. Establecer Contador a 1
2. Establecer Suma a 0
3. Mientras Contador < = 100 hacer
4. Sumar Contador a Suma
5. Incrementar Contador en 1
6. fin mientras
7. Visualizar Suma

## 1.2.4 Codificación de un programa

La codificación es la escritura en un lenguaje de programación de la representación del algoritmo desarrollada en las etapas precedentes. Dado que el diseño de un algoritmo es independiente del lenguaje de programación utilizada para su implementación, el código puede ser escrito con igual facilidad en un lenguaje o en otro.

Para realizar la conversión del algoritmo en programa se deben sustituir las palabras reservadas en español por sus homónimos en inglés, y las operaciones/instrucciones indicadas en lenguaje natural por el lenguaje de programación correspondiente.

## 1.2.5 Compilación y ejecución de un programa

Una vez que el algoritmo se ha convertido en un programa fuente, es preciso introducirlo en memoria mediante el teclado y almacenarlo posteriormente en un disco. Esta operación se realiza con un programa editor. Después el programa fuente se convierte en un archivo de programa que se guarda (graba) en disco.

El programa fuente debe ser traducido a lenguaje máquina, este proceso se realiza con el **compilador** y el sistema operativo que se encarga prácticamente de la compilación.

Si tras la compilación se presentan errores (errores de compilación) en el programa fuente, es preciso volver a editar el programa, corregir los errores y compilar de nuevo. Este proceso se repite hasta que no se producen errores, obteniéndose el programa objeto que todavía no es ejecutable directamente. Suponiendo que no existen errores en el programa fuente, se debe instruir al sistema operativo para que realice la fase de montaje o enlace (link), carga, del programa objeto con las bibliotecas del programa del compilador.

Una vez que el programa ejecutable se ha creado, ya se puede ejecutar (correr o rodar) desde el sistema operativo con solo teclear su nombre (en el caso de DOS). Suponiendo que no existen errores durante la ejecución (llamados errores en tiempo de ejecución), se obtendrá la salida de resultados del programa.

## 1.2.6 Verificación y depuración de un programa

La verificación o depuración de un programa es el proceso de ejecución del programa con una amplia variedad de datos de entrada, llamados datos de test o prueba, que determinarán si el programa tiene o no errores. Para realizar la verificación se debe desarrollar una amplia gama de datos de test: valores normales de entrada, valores extremos de entrada que comprueben los límites del programa y valores de entrada que comprueben aspectos especiales del programa.

## 1.2.7 Mantenimiento y documentación

La documentación de un programa puede ser **interna y externa**. La documentación interna es la contenida en líneas de comentarios. La documentación externa incluye análisis, diagramas de flujo y(o) pseudocódigos, manuales de usuario con instrucciones para ejecutar el programa y para interpretar los resultados.

## 1.3 Algoritmos: conceptos y propiedades

Un algoritmo es un método para resolver un problema. La resolución de un problema exige el diseño de un algoritmo que resuelva el problema propuesto. Los pasos para la resolución de un problema son:

1. **Diseño del algoritmo**, que describe la secuencia ordenada de pasos, sin ambigüedades, que conducen a la solución de un problema dado. (Análisis del problema y desarrollo del algoritmo.)
2. Expresar el algoritmo como un programa en un lenguaje de programación adecuado. (Fase de codificación.)
3. Ejecución y validación del programa por la computadora.

Programa de computadora

Diseño Algoritmo

Problema

Los algoritmos son independientes tanto del lenguaje de programación en que se expresan como de la computadora que los ejecuta. En cada problema el algoritmo se puede expresar en un lenguaje diferente de programación y ejecutarse en una computadora distinta; sin embargo, el algoritmo será siempre el mismo. Así, por ejemplo, en una analogía con la vida diaria, una receta de un plato de cocina se puede expresar en español, inglés o francés, pero cualquiera que sea el lenguaje, los pasos para la elaboración del plato se realizarán sin importar el idioma del cocinero.

En la ciencia de la computación y en la programación, los algoritmos son más importantes que los lenguajes de programación o las computadoras. Un lenguaje de programación es tan solo un medio para expresar un algoritmo y una computadora es solo un procesador para ejecutarlo. Tanto el lenguaje de programación como la computadora son los medios para obtener un fin: conseguir que el algoritmo se ejecute y se efectúe el proceso correspondiente.

## 1.3.1 Características de los algoritmos

* Debe ser preciso
* Debe tener un orden: Entrada (ingredientes), Proceso (elaboración de la receta) y Salida (asado)

Ejemplo: Se desea diseñar un algoritmo para saber si un número es primo o no. Un número es primo si solo puede dividirse entre sí mismo y entre 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 entre números distintos a ellos mismos y a la unidad. Así, 9 es divisible entre 3, 8 lo es entre 2, etc. El algoritmo de resolución del problema pasa por dividir sucesivamente el número entre 2, 3, 4..., etcétera.

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 entre X (N/X).

4. Si el resultado de N/X es entero, entonces N no es número primo y se bifurca

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 primo; en caso contrario bifurcar al punto

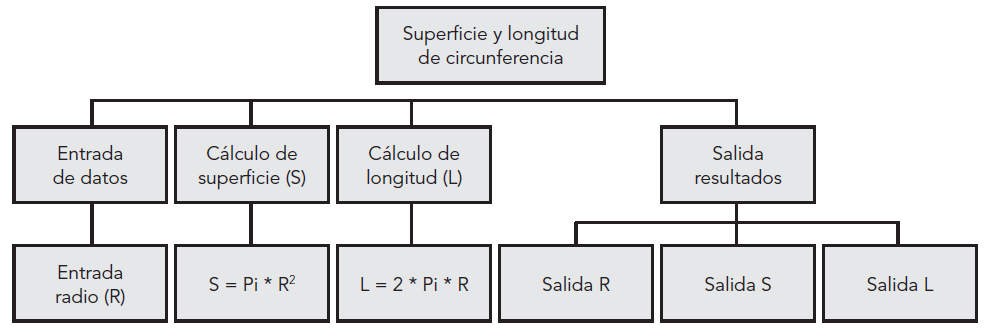
3.

7. Fin.

## 1.3.2 Diseño de algoritmos

Una computadora no tiene capacidad para solucionar problemas más que cuando se le proporcionan los sucesivos pasos a realizar. Estos pasos sucesivos que indican las instrucciones a ejecutar por la máquina constituyen, como ya conocemos, el algoritmo. La información proporcionada al algoritmo constituye su entrada y la información producida por el algoritmo constituye su salida.

Los problemas complejos se pueden resolver más eficazmente con la computadora cuando se dividen en subproblemas que sean más fáciles de solucionar que el original. Así, el problema de encontrar la superficie y la longitud de un círculo se puede dividir en tres problemas más simples o subproblemas.



El problema de cálculo de la circunferencia y superficie de un círculo se puede descomponer en subproblemas más simples: 1) leer datos de entrada; 2) calcular superficie y longitud de circunferencia, y 3) escribir resultados (datos de salida).

## 1.3.3 Escritura de algoritmos

El algoritmo consta de cuatro acciones básicas, cada una de las cuales debe ser ejecutada antes de realizar la siguiente. En términos de computadora, cada acción se codificará en una o varias sentencias que ejecutan una tarea particular.

El algoritmo descrito es muy sencillo; sin embargo, como ya se ha indicado en párrafos anteriores, el algoritmo general se descompondrá en pasos más simples en un procedimiento denominado **refinamiento** sucesivo, ya que cada acción puede descomponerse a su vez en otras acciones simples. Así, por ejemplo, un primer refinamiento del algoritmo ir al cine se puede describir de la forma siguiente:

1. **inicio**

2. ver la cartelera de cines en el periodico

3. **si** no proyectan “Harry Potter” **entonces**

3.1. decidir otra actividad

3.2. bifurcar al paso 7

**si\_no**

3.3. ir al cine

**fin\_si**

4. **si** hay cola **entonces**

4.1. formarse en ella

4.2. **mientras** haya personas delante **hacer**

4.2.1. avanzar en la cola

**fin\_mientras**

**fin\_si**

5. **si** hay localidades **entonces**

5.1. comprar un boleto

5.2. pasar a la sala

5.3. localizar la(s) butaca(s)

5.4. **mientras** proyectan la pelicula **hacer**

5.4.1. ver la pelicula

**fin\_mientras**

5.5. abandonar el cine

**si\_no**

5.6. refunfunar

**fin\_si**

6. volver a casa

7. **fin**

En el algoritmo anterior existen diferentes aspectos a considerar. En primer lugar, ciertas palabras reservadas se han escrito deliberadamente en negrita (mientras, si\_no; etc.). Estas palabras describen las estructuras de control fundamentales y procesos de toma de decisión en el algoritmo. Estas incluyen los conceptos importantes de selección (expresadas por si-entonces-si\_no, if-then-else) y de repetición (expresadas con mientras-hacer o a veces repetir-hasta e iterar-fin\_iterar, en inglés, while-do y repeat-until) que se encuentran en casi todos los algoritmos, especialmente en los de proceso de datos. La capacidad de decisión permite seleccionar alternativas de acciones a seguir o bien la repetición una y otra vez de operaciones básicas.

Otro aspecto a considerar es el método elegido para describir los algoritmos: empleo de sangrado o justificación en escritura de algoritmos. En la actualidad es tan importante la escritura de programa como su posterior lectura. Ello se facilita con el sangrado de las acciones interiores a las estructuras fundamentales citadas: selectivas y repetitivas.

## 1.3.4 Representación gráfica de los algoritmos

Para representar un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Ello permitirá que un algoritmo pueda ser codificado de manera indistinta en cualquier lenguaje. Para conseguir este objetivo se precisa que el algoritmo sea representado gráfica o numéricamente, de modo que las sucesivas acciones no dependan de la sintaxis de ningún lenguaje de programación, sino que la descripción pueda servir fácilmente para su transformación en un programa, es decir, **su codificación**.

Los métodos usuales para representar un algoritmo son:

1. diagrama de flujo,
2. lenguaje de especificación de algoritmos: pseudocódigo,
3. lenguaje español, inglés...
4. fórmulas.

## 1.4 Pseudocódigo

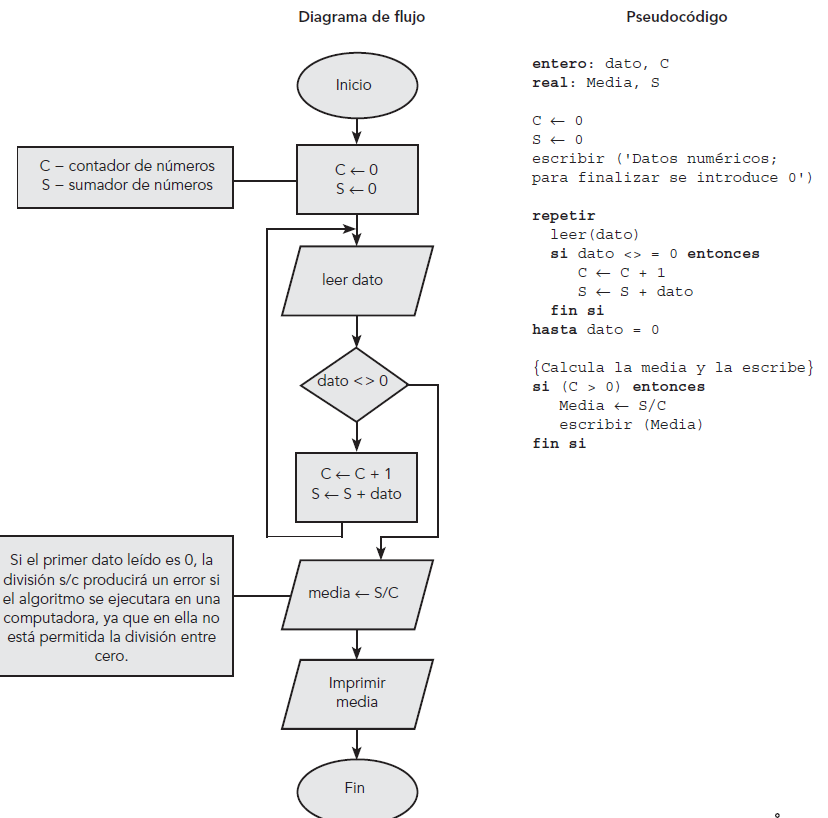
El pseudocódigo es un lenguaje de especificación (descripción) de algoritmos. El uso de tal lenguaje hace el paso de codificación final (esto es, la traducción a un lenguaje de programación) relativamente fácil. Los lenguajes APL, Pascal y Ada se utilizan a veces como lenguajes de especificación de algoritmos. También considerado como primer borrador.

El pseudocódigo no puede ser ejecutado por una computadora. La ventaja del pseudocódigo es que en su uso, en la planificación de un programa, el programador se puede concentrar en la lógica y en las estructuras de control y no preocuparse de las reglas de un lenguaje específico.

## 1.5 Diagramas de flujos

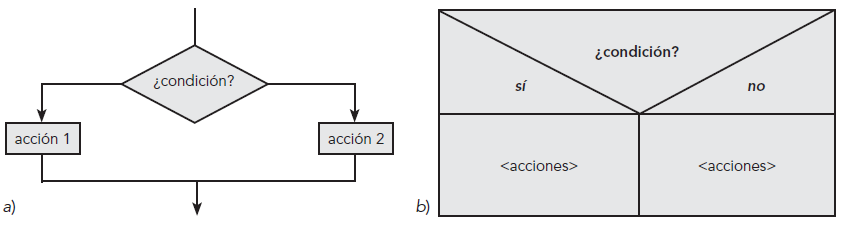
Un diagrama de flujo (flowchart) es una de las técnicas de representación de algoritmos más antigua y a la vez más utilizada, aunque su empleo ha disminuido considerablemente, sobre todo, desde la aparición de lenguajes de programación estructurados.

VER tabla



## 1.6 Diagramas de Nassi-Schneiderman (N-S)

El diagrama N-S de (también conocido como diagrama de Chapin) es como un diagrama de flujo en el que se omiten las flechas de unión y las cajas son contiguas. Las acciones sucesivas se escriben en cajas sucesivas y, como en los diagramas de flujo, se pueden escribir diferentes acciones en una caja.



## 1.7 Metodología de programación

Existen dos enfoques muy populares para el diseño y construcción de programas: el enfoque **estructurado** y el enfoque **orientado a objetos**. Estos dos enfoques conducen a dos tipos o metodologías de programación: programación estructurada y programación orientada a objetos.1 Un tercer enfoque, la **programación modular** está directamente conectado con los otros dos enfoques.

## 1.7.1 Programación modular

La programación modular es uno de los métodos de diseño más flexible y potente para mejorar la productividad de un programa. En programación modular el programa se divide en **módulos** (partes independientes), cada uno de los cuales ejecuta una única actividad o tarea.

## 1.7.2 Programación Estructurada

La programación estructurada es un enfoque específico que, normalmente, produce programas bien escritos y muy legibles, aunque no necesariamente un programa bien escrito y fácil de leer ha de ser estructurado. La programación estructurada trata de escribir un programa de acuerdo con unas reglas y un conjunto de técnicas.

Las reglas de programación estructurada o diseño estructurado se basan en la modularización; es decir, en la división de un problema en subproblemas más pequeños (módulos).

Las técnicas de programación estructurada incluyen construcciones o estructuras (instrucciones) básicas de control. **Secuencia**. **Decisión** (también denominada selección). **Bucles** o lazos (también denominada repetición o iteración).

## 1.7.3 Programación orientada a objetos

La Programación orientada a objetos (POO) es el paradigma de programación dominante en la actualidad y ha sustituido las técnicas de programación estructurada.

La programación orientada a objetos se compone de objetos. Un objeto es un elemento autosuficiente de un programa de computadora que representa un grupo de características relacionadas entre sí y está diseñado para realizar una tarea específica. Cada objeto tiene una funcionalidad específica que se expone a sus usuarios y una implementación oculta al usuario.

## 1.8 Herramientas de programación

Las herramientas más usuales son **editor, compilador y depurador de errores** y puesta a punto del programa, sobre todo en el caso de desarrollo profesional. Estas herramientas pueden ser independientes y utilizadas de esta forma o bien estar incluidas en entornos de desarrollo integradas y utilizadas como un todo.

## 1.8.1 Editor de texto

Un editor de textos es un programa de aplicación que permite escribir programas. Los editores que sirven para la escritura de programas en lenguajes de alto nivel son diferentes de los procesadores de texto tradicionales.

Un editor de software ya sea independiente o integrado es un entorno de desarrollo que normalmente debe proporcionar las características adecuadas para la adaptación de las normas de escritura de la sintaxis del lenguaje de programación correspondiente y también algunas propiedades relacionadas con la sintaxis de este lenguaje; que reconozca sangrados de línea y que reconozca y complete automáticamente palabras clave (reservadas) del lenguaje después de que los programadores hayan tecleado los primeros caracteres de la palabra.

## 1.8.2 Programa ejecutable

El programa o archivo ejecutable es el archivo binario (código máquina) cuyo contenido es interpretado por la computadora como un programa. El ejecutable contiene instrucciones en código máquina de un procesador específico.

## 1.8.3 Proceso de compilación

Las computadoras solo entienden el lenguaje máquina. Por consiguiente, para ejecutar un programa con éxito, el código fuente se debe traducir al lenguaje máquina, mediante un compilador o en su caso un intérprete.

Existen dos métodos para procesar programas completos (compilación y ejecución). Uno es los programas de consola, normalmente conocidos como consola de línea de comandos, que son herramientas en las que los programadores deben teclear los comandos (las órdenes) en una consola o ventana Shell y ejecutar paso a paso las diferentes etapas de compilación. El segundo método es el más utilizado ya que suele ser más cómodo de usar: son los EDI (Integrated Development Environment), entorno de desarrollo integrado, que permiten la edición, compilación y ejecución de un programa de modo directo.