

UT. 1

Introducción a la programación

ÍNDICE

1. Introducción a la programación.

1.1 ¿Que es la programación ?.

1.2 ¿Qué es un programador, ¿qué y cómo lo hace?.

1.3 ¿Qué es un programa?

1.4 Lenguajes de programación .

1.4.1 ¿Qué es un lenguaje de programación?

1.4.2 Un poco de evolución de los lenguajes de programación.

1.4.3 Clasificación de los lenguajes de programación .

1.5 Estructura de un programa.

1.6 Fases de la programación.

1.6.1 Resolución del problema.

1.6.2 Implementación.

1.6.3 Explotación.

1.7 Algoritmos.

1.7.1 Características.

1.7.2 Elementos básicos.

1.7.3 Estructura.

1.7.4 Medios de Expresión.

1. Introducción a la programación.

Se suele decir que una persona no entiende algo de verdad hasta que puede explicárselo a otro. En realidad, no lo entiende de verdad hasta que puede explicárselo a un computador. —Donald Knuth.

En esta primera unidad realizaremos un breve recorrido por los conceptos fundamentales de la programación de aplicaciones. Iniciaremos nuestro camino conociendo a grandes rasgos los orígenes y evolución de la programación hasta el presente. Continuando con el análisis de las **diferentes formas de programación existentes**, identificaremos qué **fases** conforman el desarrollo de un programa, avanzaremos detallando las características relevantes de los **lenguajes de programación disponibles**, para posteriormente, realizar una visión general del lenguaje de programación Java. Finalmente, tendremos la oportunidad de conocer con qué herramientas podríamos desarrollar nuestros programas (eclipse, netsBeans, intelliJ, etc), escogiendo entre una de ellas para ponernos manos a la obra utilizando el lenguaje Java.

¿Cuántas acciones de las que has realizado hoy, crees que están relacionadas con la programación?

Hagamos un repaso de los primeros instantes del día: te ha despertado la alarma de tu teléfono móvil o despertador; has preparado el desayuno utilizando el microondas, mientras desayunabas has visto u oído las últimas noticias a través de tu televisión digital terrestre, te has vestido y puede que hayas utilizado el ascensor para bajar al portal ó has abierto la puerta del garaje con tu mando para salir a la calle, etc. Quizá no es necesario que continuemos más para darnos cuenta de que casi todo lo que nos rodea, en alguna medida, está relacionado con la programación, los programas y el tratamiento de algún tipo de información.

1.1 ¿Que es la programación ? .

La programación es el acto de programar, es decir, **organizar una secuencia de pasos a seguir, ordenados según un criterio lógico para lograr un objetivo concreto**. Este término puede utilizarse en muchos contextos, es común hablar de programación a la hora de organizar una salida, las vacaciones o de la lista de programas con sus días y horarios de emisión de los canales de televisión o la lista de películas de un cine, etc.

En el ámbito de la informática, la programación se refiere a la **acción de crear programas o aplicaciones en un lenguaje de programación que da como resultado un código fuente que una vez interpretado o compilado, genera un conjunto de ordenes que el ordenador puede ejecutar** .

Por sí misma, la programación informática es la relación que existe entre un **programador** y un **ordenador** , por la cual , a través de un lenguaje específico (Lenguajes de programación) , una persona da instrucciones a un ordenador, y como consecuencia hacer que cumpla con un objetivo establecido.

1.2 ¿Qué es un programador, ¿qué y cómo lo hace?

Ya podemos dirigirnos a un ordenador en nuestro idioma y conseguir que nos resuelva tareas para los que ha sido programado , *por un programador*, pretender que nos resuelva tareas para las que no ha sido programado aún no se ha conseguido..... pero se llegará ... y entonces puede que sean los propios ordenadores quienes hagan sus propios programas. Hasta entonces serán los programadores quienes sigan haciendo aplicaciones para las máquinas , contribuyendo así al desarrollo tecnológico de nuestra sociedad.

La variable mas importante para determinar el bienestar de la población ahora, comparándola con hace 100 años, no es la política , no es la economíaes el desarrollo de la tecnología...

No podemos decirle así como si nada al ordenador qué es lo que queremos en nuestro idioma nativo, necesitaremos un programador informático para que nos traduzca lo que queremos hacer, mediante el análisis, diseño y codificación en un lenguaje determinado, a un programa que sea entendible por la maquina. *Un programador informático es aquella persona que comunica sus ideas e instrucciones a un ordenador siguiendo una serie de normas preestablecidas por cada lenguaje de programación*, haciendo que los sitios web, juegos y otros programas cumplan con el objetivo para el que fueron pensados.

También se podría definir al programador como un políglota que se especializa en hablar con ordenadores en distintos lenguajes. Saben exactamente qué decir y cómo decirlo para que el ordenador los entienda. Como sabrás, cada idioma posee un vocabulario, una gramática y un conjunto de reglas propias. Los lenguajes de programación también tienen su propia manera de decir cosas, la cual se llama **sintaxis**. Si bien, una persona puede entenderte aunque pronunciaras mal una palabra, si usaras la conjugación incorrecta o incluso si arruinaras tu gramática, un ordenador sería mucho menos tolerante con los errores. Tan pronto cometes un error de sintaxis, la computadora dejará de escucharte y dejará de ejecutar el programa, mostrándote un error en la mayoría de los casos.

Para realizar esta tarea existen una serie de pautas y herramientas que nos ayudan en nuestro trabajo y que iremos descubriendo poco a poco. En definitiva, *el programador realiza su trabajo analizando el problema, diseñando la solución (algoritmo) y por ultimo codificando el programa en un lenguaje determinado (java, c, c++, python...).*

1.3 ¿Qué es un programa?

Un programa es una serie de órdenes o instrucciones ordenadas que realizan una función determinada. De entre las muchas definiciones que se le pueden dar:

*“Es un conjunto de órdenes diseñadas y creadas a través del razonamiento lógico, almacenadas en ficheros de texto respetando la sintaxis de un determinado **lenguaje de programación**, que una vez ejecutadas realizarán una o varias tareas en un ordenador.”*

1.4 Lenguajes de programación .

1.4.1 ¿ qué es un lenguaje de programación?

“Un lenguaje de programación es un conjunto de instrucciones, operadores y reglas de sintaxis y semánticas, que se ponen a disposición del programador para que este pueda comunicarse con los dispositivos de hardware y software existentes.”

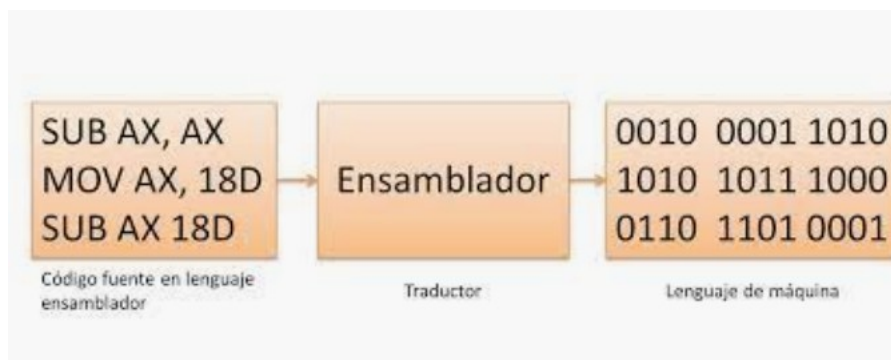
1.4.2 Un poco de evolución de los lenguajes de programación.

La historia de la programación y por consiguiente de los lenguajes de programación se encuentra fuertemente relacionada con la computación, puesto que a partir de la aparición de los ordenadores personales fue cuando esta empezó a tener relevancia para la sociedad, debido a la velocidad con la cual se podían ejecutar tareas mecánicas mediante la codificación de comandos.

El conjunto de órdenes e instrucciones que se dan al ordenador para que resuelva un problema o ejecute una determinada misión, recibe el nombre de **programa ó aplicación** . En los primeros tiempos de la informática, la programación se efectuaba en el único lenguaje que entiende el microprocesador: su propio código binario, también denominado lenguaje máquina o código máquina.



Pero la programación en lenguaje máquina resulta muy lenta y tediosa, pues los datos e instrucciones se deben introducir en sistema binario y, además, obliga a conocer las posiciones de memoria donde se almacenan los datos. Como puede imaginar, este tipo de programación conlleva gran número de errores y la tarea de depuración exige bastante tiempo y dedicación. Por este motivo, a principios de los 50 se creó una notación simbólica, denominada código de ensamblaje (ASSEMBLY), que utiliza una serie de abreviaturas nemotécnicas para representar las operaciones :



Al principio, la traducción del código de ensamblaje al código máquina se realizaba manualmente, pero enseguida se vio que el ordenador también podía encargarse de esa traducción; se desarrolló así un programa traductor, llamado ensamblador (ASSEMBLER).

Conforme los ordenadores fueron introduciéndose en el mundo empresarial y académico, aquellos primitivos lenguajes fueron sustituidos por otros más sencillos de aprender y más cómodos de emplear. Estos lenguajes, llamados de alto nivel, tienen una estructura que se adapta más al pensamiento humano que a la forma de trabajar del ordenador. Por ejemplo, seguro que le suenan lenguajes como BASIC, Java, C, Python etc. En la actualidad, se acostumbra identificar el ensamblador, que es el programa traductor, con el código de ensamblaje. ¿Y cuántos lenguajes de programación existen? Pues sucede algo así como con los lenguajes humanos: existen centenares, si bien sólo unos pocos de ellos son ampliamente utilizados. [Lectura recomendada](#)

```
1 package ventanas;
2
3 import java.awt.Graphics; // importar la clase Graphics
4 import javax.swing.JFrame;
5
6 public class Ventana extends JFrame {
7
8     // constructora
9     public Ventana() {
10         setTitle("Gráfica"); // título
11         setSize(600,250); // ventana de 600x250
12         setDefaultCloseOperation(EXIT_ON_CLOSE);
13     }
14
15     // función que dibuja la gráfica
16     public void paint(Graphics g) {
17         // para cada coordenada x ...
18         for (int x = 0 ; x < 600 ; x++)
19             // dibujamos una línea entre (x,f(x)) y (x+1,f(x+1))
20             g.drawLine(x, (int)f(x), x + 1, (int)f(x + 1));
21     }
22
23     // función a dibujar
24     double f(double x) {
25         // en Java "cos" se escribe "Math.cos" y "sen" "Math.sin"
26         return 250-(Math.cos(x/3) + Math.sin(x/5) +3) * 40 ;
27     }
28 }
```

1.4.3 Clasificación de los lenguajes de programación .

Según nivel de Abstracción

Una capa de abstracción (o nivel de abstracción) es una forma de ocultar los detalles de implementación de ciertas funcionalidades.

1º El Lenguaje Máquina: es el lenguaje de programación que entiende directamente la máquina . Este lenguaje de programación utiliza el alfabeto binario, es decir, el 0 y el 1.

Con estos dos únicos dígitos, también conocidos como bits, se forman lo que se conoce como cadenas binarias (combinaciones de ceros y unos) son con las que se escriben las instrucciones, y a través de estas instrucciones el microprocesador de la computadora entiende nuestra peticiones. El lenguaje máquina fue el primer lenguaje de programación. Este lenguaje de programación dejó de utilizarse por su gran dificultad y por la facilidad para cometer errores al escribir las cadenas binarias.

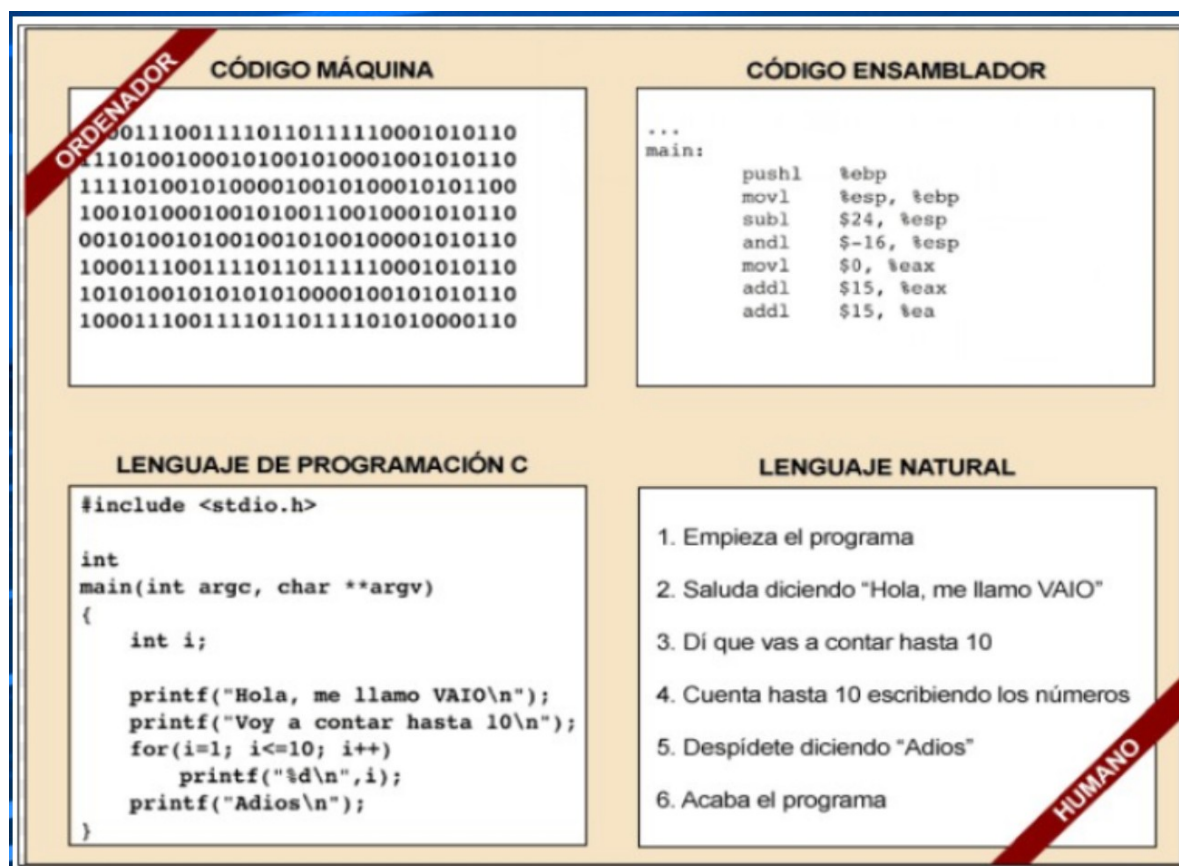
2º Lenguajes de programación de bajo nivel: Son mucho mas fáciles de utilizar que el lenguaje máquina, pero dependen mucho de la máquina o computadora como sucedía con el lenguaje máquina. El lenguaje ensamblador fue el primer lenguaje de programación que trato de sustituir el lenguaje máquina por otro lenguaje que fuese más parecido al de los seres humanos.

En este lenguaje se conoce como ***programa fuente***, que traduce las instrucciones a un programa escrito en lenguaje ensamblador por el programador, y el programa es la traducción a lenguaje máquina del programa fuente.

Los lenguajes de este tipo pueden crear programas muy rápidos, pero son difíciles de aprender, son específicos de cada procesador (de cada máquina), si nos llevamos el programa a otro computador será preciso reescribir el programa desde el comienzo.

3º Lenguajes de programación de alto nivel: Este tipo de lenguajes de programación son independientes de la máquina, los podemos usar en cualquier computador con muy pocas modificaciones o sin ellas, son muy similares al lenguaje humano, pero precisan de un programa interprete o compilador que traduzca este lenguaje de programación de alto nivel a uno de bajo nivel como el lenguaje de máquina que la computadora pueda entender.

Los lenguajes de programación de alto nivel son más fáciles de aprender porque se usan palabras o comandos del lenguaje natural, generalmente del inglés. Este es el caso del BASIC, Java, C, C#, etc.



Según tipo de Ejecución.

- Interpretes

Un intérprete es un programa informático que *procesa el código fuente de un proyecto de software durante su tiempo de ejecución*, es decir, mientras el software se está ejecutando, y actúa como una **interfaz** entre ese proyecto y el procesador. Un intérprete siempre procesa el código línea por línea, de modo que lee, analiza y prepara cada secuencia de forma consecutiva para el procesador. Este principio también se aplica a las secuencias recurrentes, que se ejecutan de nuevo cada vez que vuelven a aparecer en el código. Para procesar el código fuente del software, el intérprete recurre a sus propias bibliotecas internas: en cuanto una *línea de código fuente se ha traducido a los correspondientes comandos legibles por máquina*, esta se envía directamente al procesador.

El proceso de conversión no finaliza hasta que se ha interpretado todo el código. Solo se interrumpe prematuramente si se produce un fallo durante el procesamiento, lo que simplifica mucho la resolución de los errores, ya que la línea de código problemática se detecta inmediatamente después de ocurrir el fallo.

- **Compiladores**

Un compilador es un programa informático que *traduce todo el código fuente de un proyecto de software a código máquina antes de ejecutarlo*. Solo entonces el procesador ejecuta el software, obteniendo todas las instrucciones en código máquina antes de comenzar. De esta manera, el procesador cuenta con todos los componentes necesarios para ejecutar el software, procesar las entradas y generar los resultados. No obstante, en muchos casos, durante el proceso de compilación tiene lugar un paso intermedio fundamental: antes de generar la traducción final en código máquina, la mayoría de los compiladores suelen convertir el código fuente en un **código intermedio** (también llamado *código objeto*) que, a menudo, es compatible con diversas plataformas y que, además, también puede ser utilizado por un intérprete.

Al producir el código, el compilador determina **qué instrucciones** van a enviarse al procesador y **en qué orden**. Si las instrucciones no son interdependientes, incluso es posible que puedan procesarse en paralelo.

Compilador versus intérprete

Tanto los compiladores como los intérpretes cumplen la función de *convertir el código de software que se ha escrito a un formato ejecutable y legible por máquina*. Sin esta traducción, los procesadores informáticos no podrían ejecutar el software en lenguajes como C, C++, PHP, Python o Ruby, lo que convierte estos programas en unos componentes imprescindibles para utilizar ordenadores, portátiles o smartphones. Hemos visto que compiladores e intérpretes presentan algunas diferencias básicas, algo que debe tenerse especialmente en cuenta a la hora de elegir un **lenguaje de programación adecuado** para desarrollar un nuevo software. En la siguiente tabla, se resumen los aspectos clave que caracterizan intérpretes y compiladores:

	Intérprete	Compilador
Momento en que se traduce el código fuente	Durante el tiempo de ejecución del software	Antes de ejecutar el software
Procedimiento de traducción	Línea por línea	Siempre todo el código
Presentación de errores de código	Después de cada línea	En conjunto, después de toda la compilación
Velocidad de traducción	Alta	Baja
Eficiencia de traducción	Baja	Alta
Coste de desarrollo	Bajo	Alto
Lenguajes típicos	PHP, Perl, Python, Ruby, BASIC	C, C++, Pascal

Si observamos las diferencias entre compilador e intérprete, vemos claramente los puntos fuertes y débiles de cada solución para traducir el código fuente: con el intérprete, los programas se pueden ejecutar de inmediato y, por lo tanto, se inician mucho más rápido. Además, el desarrollo es mucho más fácil que con un compilador, porque el proceso de depuración (es decir, la corrección de errores) se lleva a cabo igual que la traducción, línea por línea. En el caso del compilador, primero

debe traducirse todo el código antes de poder resolver los errores o iniciar la aplicación. Sin embargo, una vez que se ejecuta el programa, los servicios del compilador ya no son necesarios, mientras que el intérprete continúa utilizando los recursos informáticos.

	Ventaja	Inconveniente
Intérprete	Proceso de desarrollo sencillo (sobre todo en términos de depuración)	Proceso de traducción poco eficiente y velocidad de ejecución lenta
Compilador	Proporciona al procesador el código máquina completo y listo para ejecutar	Cualquier modificación del código (resolución de errores, desarrollo del software, etc.) requiere volverlo a traducir

• Solución híbrida de intérprete y compilador: compilación en tiempo de ejecución

Para compensar los puntos débiles de ambas soluciones, también existe el llamado modelo de compilación en tiempo de ejecución (en inglés, **just-in-time-compiler**, o “compilador justo a tiempo”). Este tipo de compilador, que a veces también se conoce por el término inglés **compreter** (acrónimo de **compiler** e **interpreter**), rompe con el modelo habitual de compilación y traduce el código del programa durante el tiempo de ejecución, al igual que el intérprete. De esta forma, la **alta velocidad de ejecución** típica de los compiladores se complementa con la **simplificación del proceso de desarrollo**.

Son lenguajes en los que el código fuente no se compila directamente a código máquina, sino que se compila un código intermedio y **se ejecuta en una máquina virtual**. Ésta, se encarga de convertir las instrucciones en código intermedio (lenguaje máquina de la máquina virtual), a código máquina nativo (lenguaje máquina de la máquina real).

Desde la perspectiva de la máquina virtual, el código fuente es compilado, pero, desde la perspectiva de la máquina real, el código intermedio es interpretado.

Generalmente el paso de código intermedio a código nativo, se realiza usando los denominados **compiladores Just in Time** (al vuelo), que optimizan el proceso, utilizando cachés, o almacenes temporales, donde guardan las conversiones ya realizadas para optimizar el proceso.

Ejemplos de este último grupo serían Java o los lenguajes para la plataforma .net de Microsoft (Visual Basic .net, Visual C#, etc).

Java es uno de los ejemplos más conocidos de lenguaje basado en compilación en tiempo de ejecución: el compilador JIT, que figura entre los componentes del **Java Runtime Environment (JRE)**, mejora el **rendimiento de las aplicaciones Java** traduciendo el código de bytes en código máquina de manera dinámica.



1.5 Estructura de un programa.

Un programa informático (programa) es una secuencia de acciones (instrucciones) que manipulan un conjunto de objetos (datos).

Existen dos partes o bloques que componen un programa:

1. **Bloque de declaraciones:** en este se detallan todos los objetos que utiliza el programa (constantes, variables, archivos, etc).
2. **Bloque de instrucciones:** conjunto de acciones u operaciones que se han de llevar a cabo para conseguir los resultados esperados.

El bloque de instrucciones está compuesto a su vez por tres partes, aunque en ocasiones no están perfectamente delimitadas, y aparecerán entremezcladas en la secuencia del programa, podemos localizarlas según su función. Estas son:

- 1.**Entrada de datos:** instrucciones que almacenan en la memoria interna datos procedentes de un dispositivo externo.
- 2.**Proceso o algoritmo:** instrucciones que modifican los objetos de entrada y, en ocasiones, creando otros nuevos.
- 3.**Salida de resultados:** conjunto de instrucciones que toman los datos finales de la memoria interna y los envían a los dispositivos externos.

Partes del bloque de instrucciones

Entrada	--> Algoritmo -->	Salida
Inicio de programa: datos	Proceso de programa: cálculos	Fin de programa: resultados

En la siguiente tabla detallamos la estructura básica de un programa informático:

Estructura de un programa informático

Cabecera	A modo de comentarios se suele especificar: <ul style="list-style-type: none"> Nombre del programa Datos de entrada Datos de salida
Funciones, Librerías, Referencias	Definición de funciones propias creadas por el programador para usarlas en varias ocasiones Librerías del lenguaje o referencias.
Declaraciones	Definiciones y tipos de: <ul style="list-style-type: none"> variables constantes nuevos tipos de datos
Asignaciones	Valores iniciales de los identificadores declarados previamente
Entradas	Instrucciones para almacenar en memoria los valores de algunos identificadores
Control	Instrucciones de control de flujo del programa. Pueden ser: <ul style="list-style-type: none"> Alternativas Repetitivas
Salidas	Instrucciones para devolver los resultados obtenidos

1.6 Fases de la programación.

El proceso de creación de software puede dividirse en diferentes fases:

1. *Fase de resolución del problema.*
2. *Fase de implementación.*
3. *Fase de explotación y mantenimiento.*

A continuación, analizaremos cada una de ellas.

1.6.1 Resolución del problema.

Para el comienzo de esta fase, es necesario que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle. A su vez, la fase de resolución del problema puede dividirse en dos etapas:

Análisis

Por lo general, el análisis indicará la especificación de requisitos que se deben cubrir. Los contactos entre el analista/programador y el cliente/usuario serán numerosos, de esta forma podrán ser conocidas todas las necesidades que precisa la aplicación. Se especificarán los procesos y estructuras de datos que se van a emplear. La creación de prototipos será muy útil para saber con mayor exactitud los puntos a tratar.

El análisis inicial ofrecerá una idea general de lo que se solicita, realizando posteriormente sucesivos refinamientos que servirán para dar respuesta a las siguientes cuestiones:

1. ¿Cuál es la información que ofrecerá la resolución del problema?
2. ¿Qué datos son necesarios para resolver el problema?

La respuesta a la primera pregunta se identifica con los resultados deseados o las salidas del problema. La respuesta a la segunda pregunta indicará qué datos se proporcionan o las entradas del problema.

En esta fase debemos aprender a analizar la documentación de la empresa , investigar, observar todo lo que rodea el problema y recopilar cualquier información útil.

Diseño

En esta etapa se convierte la especificación realizada en la fase de análisis en un diseño más detallado, indicando el comportamiento o la secuencia lógica de instrucciones capaz de resolver el problema planteado. Estos pasos sucesivos, que indican las instrucciones a ejecutar por la máquina, constituyen lo que conocemos como *algoritmo*.

Consiste en plantear la aplicación como una única operación global, e ir descomponiéndola en operaciones más sencillas, detalladas y específicas. En cada nivel de refinamiento, las operaciones identificadas se asignan a módulos separados.

Hay que tener en cuenta que antes de pasar a la implementación del algoritmo, hemos de asegurarnos que tenemos una solución adecuada. Para ello, todo diseño requerirá de la realización de la **prueba o traza** del programa. Este proceso consistirá en un seguimiento paso a paso de las instrucciones del algoritmo utilizando datos concretos. Si la solución aportada tiene errores, tendremos que volver a la fase de análisis para realizar las modificaciones necesarias o tomar un nuevo camino para la solución. Sólo cuando el algoritmo cumpla los requisitos y objetivos especificados en la fase de análisis se pasará a la fase de implementación.

Herramientas de diseño para realizar el algoritmo.

1.6.2 Implementación.

Si la fase de resolución del problema requiere un especial cuidado en la realización del análisis y el posterior diseño de la solución, la fase de implementación cobra también una especial relevancia. Llevar a la realidad nuestro algoritmo implicará cubrir algunas etapas más que se detallan a continuación.

Codificación o construcción

Esta etapa consiste en transformar o traducir los resultados obtenidos a un determinado lenguaje de programación. Para comprobar la calidad y estabilidad de la aplicación se han de realizar una serie de pruebas que comprueben las funciones de cada módulo (pruebas unitarias), que los módulos funcionan bien entre ellos (pruebas de interconexión) y que todos funcionan en conjunto correctamente (pruebas de integración).

Cuando realizamos la traducción del algoritmo al lenguaje de programación debemos tener en cuenta las reglas gramaticales y la sintaxis de dicho lenguaje. Obtendremos entonces el código fuente, lo que normalmente conocemos por programa.

Pero para que nuestro programa comience a funcionar, antes debe ser traducido a un lenguaje que la máquina entienda. Este proceso de traducción puede hacerse de dos formas, compilando o interpretando el código del programa.

Compilación: *Es el proceso por el cual se traducen las instrucciones escritas en un determinado lenguaje de programación a lenguaje que la máquina es capaz de interpretar.*

Compilador: *programa informático que realiza la traducción. Recibe el código fuente, realiza un análisis lexicográfico, semántico y sintáctico, genera un código intermedio no optimizado, optimiza dicho código y finalmente, genera el código objeto para una plataforma específica.*

Intérprete: *programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los intérpretes sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.*

Una vez traducido, sea a través de un proceso de compilación o de interpretación, el programa podrá ser ejecutado.

Prueba de ejecución y validación

Para esta etapa es necesario implantar la aplicación en el sistema donde va a funcionar, debe ponerse en marcha y comprobar si su funcionamiento es correcto. Utilizando diferentes datos de prueba se verá si el programa responde a los requerimientos especificados, si se detectan nuevos errores, si éstos son bien gestionados y si la interfaz es amigable. Se trata de poner a prueba nuestro programa para ver su respuesta en situaciones difíciles.

Mientras se detecten errores y éstos no se subsanen no podremos avanzar a la siguiente fase. Una vez corregido el programa y testeado se documentará mediante:

Documentación interna: Encabezados, descripciones, declaraciones del problema y comentarios que se incluyen dentro del código fuente.

Documentación externa: Son los manuales que se crean para una mejor ejecución y utilización del programa.

1.6.3 Explotación.

Cuando el programa ya está instalado en el sistema y está siendo de utilidad para los usuarios, decimos que se encuentra en fase de explotación.

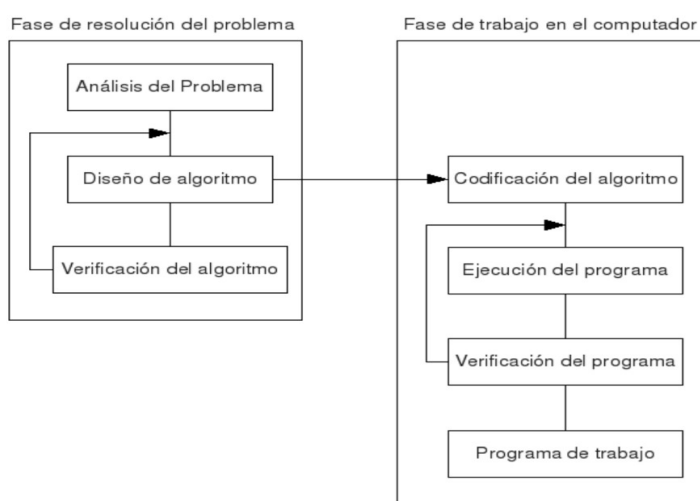
Periódicamente será necesario realizar evaluaciones y, si es necesario, llevar a cabo modificaciones para que el programa se adapte o actualice a nuevas necesidades, pudiendo también corregirse errores no detectados anteriormente. Este proceso recibe el nombre de mantenimiento del software.

Mantenimiento del software: es el proceso de mejora y optimización del software después de su entrega al usuario final. Involucra cambios al software en orden de corregir defectos y dependencias encontradas durante su uso, así como la adición de nuevas funcionalidades para mejorar la usabilidad y aplicabilidad del software.

1.7 Algoritmos .

Un *algoritmo informático* es un conjunto de instrucciones definidas, ordenadas y acotadas para resolver un problema, realizar un cálculo o desarrollar una tarea. Es decir, un algoritmo es un procedimiento paso a paso para conseguir un fin. A partir de un estado e información iniciales, se siguen una serie de pasos ordenados para llegar a la solución de una situación.

En **programación**, un algoritmo supone el **paso previo a ponerse a escribir el código**. De este modo, un programa informático no sería más que un **conjunto de algoritmos ordenados y codificados** en un **lenguaje de programación** para poder ser ejecutados en un ordenador.



1.7.1 Características

- **Carácter finito.** Un algoritmo siempre debe terminar después de un número finito de pasos.
- **Precisión.** Cada paso de un algoritmo debe estar precisamente definido; las operaciones a llevar a cabo deben ser especificadas de manera rigurosa y no ambigua para cada caso.
- **Entrada.** Un algoritmo tiene cero o más entradas: cantidades que le son dadas antes de que el algoritmo comience, o dinámicamente mientras el algoritmo corre. Estas entradas son tomadas de conjuntos específicos de objetos.
- **Salida.** Un algoritmo tiene una o más salidas: cantidades que tienen una relación específica con las entradas.
- **Eficacia.** También se espera que un algoritmo sea eficaz, en el sentido de que todas las operaciones a realizar en un algoritmo deben ser suficientemente básicas como para que en principio puedan ser hechas de manera exacta y en un tiempo finito por un hombre usando lápiz y papel.
- Los problemas complejos se pueden resolver de manera más eficaz cuando se dividen en subproblemas que sean más fáciles de solucionar que el inicial (diseño descendente).

1.7.2 Elementos básicos

- **Los comentarios** sirven para documentar el algoritmo y en ellos se escriben anotaciones generalmente sobre su funcionamiento. En pseudocódigo, cuando se coloque un comentario de una sola línea se escribirá precedido de `//`. Si el comentario es multilínea, lo pondremos entre `{}`.

- **Las palabras reservadas** o palabras clave (Keywords) son palabras que tienen un significado especial, como: `inicio` y `fin`, que marcan el principio y fin del algoritmo.

- **Identificadores** son los nombres que se dan a las constantes simbólicas, variables, funciones, procedimientos, u otros objetos que manipula el algoritmo. La regla para construir un identificador establece que:

- Debe resultar significativo, sugiriendo lo que representa.
- No podrá coincidir con palabras reservadas, propias del lenguaje algorítmico. Como se verá más adelante, la representación de algoritmos mediante pseudocódigo va a requerir la utilización de palabras reservadas.
- Se recomienda un máximo de 50 caracteres.
- Comenzará siempre por un carácter alfabético y los siguientes podrán ser letras, dígitos o el símbolo de subrayado.
- Podrá ser utilizado indistintamente escrito en mayúsculas o en minúsculas

- **Dato** es la expresión general que describe los objetos con los cuales opera el algoritmo. El tipo de un dato determina su forma de almacenamiento en memoria y las operaciones que van a poder ser efectuadas con él. En principio hay que tener en cuenta que, prácticamente en cualquier lenguaje y por tanto en cualquier algoritmo, se podrán usar datos de los siguientes tipos:

- *entero*. Subconjunto finito de los números enteros, cuyo rango o tamaño dependerá del lenguaje en el que posteriormente codifiquemos el algoritmo y de la computadora utilizada.
- *real*. Subconjunto de los números reales limitado no sólo en cuanto al tamaño, sino también en cuanto a la precisión.
- *lógico*. Conjunto formado por los valores verdad y falso.
- *carácter*. Conjunto finito y ordenado de los caracteres que ordenador reconoce.
- *cadena*. Los datos (objetos) de este tipo contendrán una serie finita de caracteres, que podrán ser directamente traídos o enviados a/desde la consola.

En los algoritmos para indicar que un dato es de uno de estos tipos se declarará utilizando directamente el identificador o nombre del tipo.

Los datos pueden venir expresados como constantes, variables, expresiones o funciones.

Las **constantes** son datos cuyo valor no cambia durante todo el desarrollo del algoritmo.

Una **variable** es un objeto cuyo valor puede cambiar durante el desarrollo del algoritmo. Se identifica por su nombre y por su tipo, que podrá ser cualquiera, y es el que determina el conjunto de valores que podrá tomar la variable. En los algoritmos se deben declarar las variables que se van a usar, especificando su tipo.

Una **expresión** es una combinación de operadores y operandos. Los operandos podrán ser constantes, variables u otras expresiones y los operadores de cadena, aritméticos, relacionales o lógicos. Las expresiones se clasifican, según el resultado que producen, en:

- Numéricas. Los operandos que intervienen en ellas son numéricos, el resultado es también de tipo numérico y se construyen mediante los operadores aritméticos. Se pueden considerar análogas a las fórmulas matemáticas. Debido a que son los que se encuentran en la mayor parte de los lenguajes de programación, los algoritmos utilizarán los siguientes operadores aritméticos: menos unario (-), multiplicación (*), división real (/), adición (+), resta (-), módulo de la división entera (mod) y cociente de la división entera (div). Tenga en cuenta que la división real siempre dará un resultado real y que los operadores mod y div sólo operan con enteros y el resultado es entero.
- Alfanuméricas. Los operandos son de tipo alfanumérico y producen resultados también de dicho tipo. Se construyen mediante el operador de concatenación, representado por el operador ampersand (&) o con el mismo símbolo utilizado en las expresiones aritméticas para la suma.
- Booleanas. Su resultado podrá ser verdad o falso. Se construyen mediante los operadores relacionales y lógicos. Los operadores de relación son: igual (=), distinto (<>), menor que (<), mayor que (>), mayor o igual (>=), menor o igual (<=). Actúan sobre operandos del mismo tipo y siempre devuelven un resultado de tipo lógico. Los operadores lógicos básicos

son: negación lógica (no), multiplicación lógica (y), suma lógica (o). Actúan sobre operandos de tipo lógico y devuelven resultados del mismo tipo, determinados por las tablas de verdad correspondientes a cada uno de ellos.

En los lenguajes de programación existen ciertas **funciones** predefinidas o internas que aceptan unos argumentos y producen un valor denominado resultado.

1.7.3 Estructura general.

El pseudocódigo es la herramienta más adecuada para la representación de algoritmos. El algoritmo en pseudocódigo debe tener una estructura muy clara y similar a un programa, de modo que se facilite al máximo su posterior codificación. Interesa por tanto conocer las secciones en las que se divide un programa, que habitualmente son:

- La cabecera.
- El cuerpo del programa:
 - Bloque de declaraciones.
 - Bloque de instrucciones.

La **cabecera** contiene el nombre del programa.

El **cuerpo del programa** contiene a su vez otras dos partes: el bloque de declaraciones y el bloque de instrucciones.

En el **bloque de declaraciones** se definen o declaran las constantes con nombre, los tipos de datos definidos por el usuario y también las variables.

El **bloque de instrucciones** contiene las acciones a ejecutar para la obtención de los resultados. Las instrucciones o acciones básicas a colocar en este bloque se podrían clasificar del siguiente modo:

- De **inicio/fin**. La primera instrucción de este bloque será siempre la de inicio y la última la de fin.
- De **asignación**. Esta instrucción se utiliza para dar valor a una variable en el interior de un programa.
- De **lectura**. Toma uno o varios valores desde un dispositivo de entrada y los almacena en memoria en las variables que aparecen listadas en la propia instrucción.
- De **escritura**. Envía datos a un dispositivo de salida.

1.7.4 Medios de expresión.

- **Máquina de Turing:** Es un modelo matemático diseñado por Alan Turing que formaliza el concepto de algoritmo, es una descripción que se puede considerar de abstracción de bajo nivel.
- **Pseudocódigo:** Es la descripción, de tal forma que se asemeja a un lenguaje de programación pero con algunas convenciones del lenguaje natural.

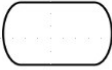



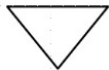
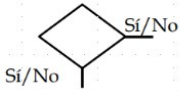



En la informática, el pseudocódigo es una forma de describir instrucciones lógicas que estructuralmente se asemeja a los lenguajes de programación. No obstante, el pseudocódigo se creó con el fin de que las personas puedan leer y escribir dichas instrucciones de forma sencilla, donde en la mayoría de los casos, se utiliza como estructura básica para el posterior desarrollo en un lenguaje de programación propiamente dicho.

La representación de un algoritmo mediante pseudocódigo se muestra a continuación. Esta representación es muy similar a la que se empleará en la escritura al programar.

```
algoritmo <nombre_algoritmo>
const
  <nombre_de_constante 1> = valor1
  ...
var
  <tipo_de_dato1>: <nombre_de_variable1> [, <nombre_de_variable2>, .....]
  ...
  //Los datos han de ser declarados antes de poder ser utilizados
inicio
  <acción 1>
  <acción 2>
  .....
  <acción N>
  // Algunas acciones podrían ser de lectura: leer(<lista_de_variables>)
  // Algunas acciones podrían ser de escritura:
  escribir(<lista_de_expresiones>)
  // Algunas acciones podrían ser de asignación: <nombre_de_variable> <-
  <expresión>
fin
```

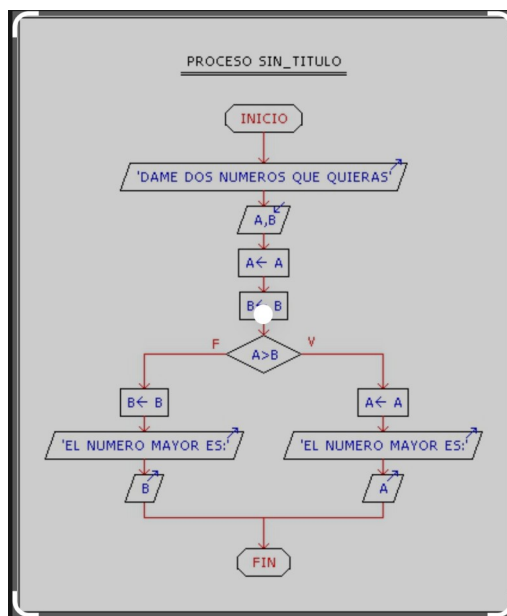
- **Diagrama de Flujo:** El *diagrama de flujo* o *flujograma* o *diagrama de actividades* es la representación gráfica de un algoritmo o proceso. Se utiliza en disciplinas como programación, economía, procesos industriales y psicología cognitiva. En Lenguaje Unificado de Modelado (UML), es un diagrama de actividades que representa los flujos de trabajo paso a paso.

Simbología ANSI

Símbolo	Significado	¿Para que se utiliza?
	Inicio / Fin	Indica el inicio y el final del diagrama de flujo.
	Operación / Actividad	Símbolo de proceso, representa la realización de una operación o actividad relativas a un procedimiento.
	Documento	Representa cualquier tipo de documento que entra, se utilice, se genere o salga del procedimiento.
	Datos	Indica la salida y entrada de datos.
	Almacenamiento / Archivo	Indica el depósito permanente de un documento o información dentro de un archivo.
	Decisión	Indica un punto dentro del flujo en que son posibles varios caminos alternativos.
	Líneas de flujo	Conecta los símbolos señalando el orden en que se deben realizar las distintas operaciones.
	Conector	Conector dentro de página. Representa la continuidad del diagrama dentro de la misma página. Enlaza dos pasos no consecutivos en una misma página.
	Conector de página	Representa la continuidad del diagrama en otra página. Representa una conexión o enlace con otra hoja diferente en la que continua el diagrama de flujo.

Fuente: Elaborado a partir de la página <http://www.ansi.org/>

Ejemplo:



- **Descripción de Alto Nivel:** Se establece el problema, se selecciona el modelo matemático y se explica el algoritmo de manera verbal.

