

# Elementos de un programa informático.



## Objetivos

En esta primera unidad comenzarás con los conceptos generales y fundamentales de la programación. Estudiarás los diferentes enfoques a la hora de desarrollar software, las fases genéricas del proceso de programación, el ciclo de vida del software, así como una descripción de los diferentes lenguajes de programación.

Posteriormente, empezarás a construir pseudocódigos, que serán el paso inicial antes de empezar a realizar programas en el lenguaje de programación Java.

# 1.- Introducción.

---

¿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 radio-despertador, has preparado el desayuno utilizando el microondas, mientras desayunabas has visto u oído las últimas noticias a través de tu receptor de televisión digital terrestre, te has vestido y puede que hayas utilizado el ascensor para bajar al portal y 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.

El volumen de datos que actualmente manejamos y sus innumerables posibilidades de tratamiento constituyen un vasto territorio en el que los programadores tienen mucho que decir.

En esta primera unidad realizaremos un recorrido por los conceptos fundamentales de la programación de aplicaciones. Iniciaremos nuestro camino conociendo con qué vamos a trabajar, qué técnicas podemos emplear y qué es lo que pretendemos conseguir. 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 cada uno 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, escogiendo entre una de ellas para ponernos manos a la obra utilizando el lenguaje Java.



## 2.- Programas y programación.

---

## 2.1.- Buscando una solución.

Generalmente, la primera razón que mueve a una persona hacia el aprendizaje de la programación es utilizar el ordenador como herramienta para resolver problemas concretos. Como en la vida real, la búsqueda y obtención de una solución a un problema determinado, utilizando medios informáticos, se lleva a cabo siguiendo unos pasos fundamentales. En la siguiente tabla podemos ver estas analogías.



### Resolución de problemas

En la vida real...	En Programación...
Observación de la situación o problema.	<b>Análisis del problema:</b> requiere que el problema sea definido y comprendido claramente para que pueda ser analizado con todo detalle.
Pensamos en una o varias posibles soluciones.	<b>Diseño o desarrollo de algoritmos:</b> procedimiento paso a paso para solucionar el problema dado.
Aplicamos la solución que más estimamos adecuada.	<b>Resolución del algoritmo elegido en la computadora:</b> consiste en convertir el algoritmo en programa, ejecutarlo y comprobar que soluciona verdaderamente el problema.

## 2.2.- Algoritmos y programas.

Después de analizar en detalle el problema a solucionar, hemos de diseñar y desarrollar el algoritmo adecuado. Pero, ¿Qué es un algoritmo?

**Algoritmo:** secuencia ordenada de pasos, descrita sin ambigüedades, que conducen a la solución de un problema dado.

Los algoritmos son independientes de los lenguajes de programación y de las computadoras donde se ejecutan. Un mismo algoritmo puede ser expresado en diferentes lenguajes de programación y podría ser ejecutado en diferentes dispositivos. Piensa en una receta de cocina, ésta puede ser expresada en castellano, inglés o francés, podría ser cocinada en fogón o vitrocerámica, por un cocinero o más, etc. Pero independientemente de todas estas circunstancias, el plato se preparará siguiendo los mismos pasos.



La diferencia fundamental entre algoritmo y **programa** es que, en el segundo, los pasos que permiten resolver el problema, deben escribirse en un determinado **lenguaje de programación** para que puedan ser ejecutados en el ordenador y así obtener la solución.

**Los lenguajes de programación** son sólo un medio para expresar el algoritmo y el ordenador un procesador para ejecutarlo. El diseño de los algoritmos será una tarea que necesitará de la creatividad y conocimientos de las técnicas de programación. Estilos distintos, de distintos programadores a la hora de obtener la solución del problema, darán lugar a algoritmos diferentes, igualmente válidos.

Pero cuando los problemas son complejos, es necesario descomponer éstos en subproblemas más simples y, a su vez, en otros más pequeños. Estas estrategias reciben el nombre de diseño **descendente** o diseño **modular** (top-down). Este sistema se basa en el lema **divide y vencerás**.

Para representar gráficamente los algoritmos que vamos a diseñar, tenemos a nuestra disposición diferentes herramientas que ayudarán a describir su comportamiento de una forma precisa y genérica, para luego poder codificarlos con el lenguaje que nos interese. Entre otras tenemos:

- ✓ **Diagramas de flujo:** Esta técnica utiliza símbolos gráficos para la representación del algoritmo. Suele utilizarse en las fases de análisis.
- ✓ **Pseudocódigo:** Esta técnica se basa en el uso de palabras clave en lenguaje natural, constantes, variables, otros objetos, instrucciones y estructuras de programación que expresan de forma escrita la solución del problema. Es la técnica más utilizada actualmente.
- ✓ **Tablas de decisión:** En una tabla son representadas las posibles condiciones del problema con sus respectivas acciones. Suele ser una técnica de apoyo al pseudocódigo cuando existen situaciones condicionales complejas.



### Diagramas de flujo

Aquí tienes un vídeo en el que se explica cómo son.



## Autoevaluación

**Rellena los huecos con los conceptos adecuados:**

A los pasos que permiten resolver el problema, escritos en un lenguaje de programación, para que puedan ser ejecutados en el ordenador y así obtener la solución, se les denomina:

## 3.- Fases de la programación.

---

Sea cual sea el estilo que escojamos a la hora de automatizar una determinada tarea, debemos realizar el proceso aplicando un método a nuestro trabajo. Es decir, sabemos que vamos a dar solución a un problema, aplicando una filosofía de desarrollo y lo haremos dando una serie de pasos que deben estar bien definidos.

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

- ✓ **Fase de resolución del problema.**
- ✓ **Fase de implementación.**
- ✓ **Fase de explotación y mantenimiento.**

A continuación, analizaremos cada una de ellas.

## 3.1.- 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:

### a. 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:

- ✓ ¿Cuál es la información que ofrecerá la resolución del problema?
- ✓ ¿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.



### Ejercicio resuelto

Vamos a ilustrar esta fase realizando el análisis del siguiente problema:

“Leer el radio de un círculo y calcular e imprimir su superficie y circunferencia.”

Está claro que las entradas de datos en este problema se reducen al radio del círculo, pero piensa ¿qué salidas de datos ofrecerá la solución?

### b. 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.

## 3.2.- 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.



### a. 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.

### b. 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.



## Autoevaluación

Rellena los huecos con los conceptos adecuados:

En la fase de codificación, hemos de tener en cuenta la [ ] del lenguaje para obtener el código fuente o programa. Posteriormente, éste deberá ser [ ] o [ ] para que pueda ser ejecutado posteriormente.

### 3.3.- 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.

Será imprescindible añadir una documentación adecuada que facilite al programador la comprensión, uso y modificación de dichos programas.

## 4.- Ciclo de vida del software.

Sean cuales sean las fases en las que realicemos el proceso de desarrollo de software, y casi independientemente de él, siempre se debe aplicar un modelo de ciclo de vida.

**Ciclo de vida del software:** es una sucesión de estados o fases por las cuales pasa un software a lo largo de su "vida".

El proceso de desarrollo puede involucrar siempre las siguientes etapas mínimas:

- ✓ Especificación y Análisis de requisitos.
- ✓ Diseño.
- ✓ Codificación.
- ✓ Pruebas.
- ✓ Instalación y paso a Producción.
- ✓ Mantenimiento.

## 5.- Lenguajes de programación.

Como hemos visto, en todo el proceso de resolución de un problema mediante la creación de software, después del análisis del problema y del diseño del algoritmo que pueda resolverlo, es necesario traducir éste a un lenguaje que exprese claramente cada uno de los pasos a seguir para su correcta ejecución. Este lenguaje recibe el nombre de lenguaje de programación.

**Lenguaje de programación:** Conjunto de reglas sintácticas y semánticas, símbolos y palabras especiales establecidas para la construcción de programas. Es un lenguaje artificial, una construcción mental del ser humano para expresar programas.

Los lenguajes de programación pueden ser clasificados en función de lo cerca que estén del lenguaje humano o del lenguaje de los computadores. El lenguaje de los computadores son códigos binarios, es decir, secuencias de unos y ceros. Detallaremos seguidamente las características principales de los lenguajes de programación.

## 5.1.- Lenguaje máquina.

Este es el lenguaje utilizado directamente por el procesador, consta de un conjunto de instrucciones codificadas en binario. Es el sistema de códigos directamente interpretable por un circuito microprogramable.

Este fue el primer lenguaje utilizado para la programación de computadores. De hecho, cada máquina tenía su propio conjunto de instrucciones codificadas en ceros y unos. Cuando un algoritmo está escrito en este tipo de lenguaje, decimos que está en **código máquina**.

```

1
1 0
1 1
1 0 0
1 0 1
1 0 0 1

```

Programar en este tipo de lenguaje presentaba los siguientes inconvenientes:

- ✓ Cada programa era válido sólo para un tipo de procesador u ordenador.
- ✓ La lectura o interpretación de los programas era extremadamente difícil y, por tanto, insertar modificaciones resultaba muy costoso.
- ✓ Los programadores de la época debían memorizar largas combinaciones de ceros y unos, que equivalían a las instrucciones disponibles para los diferentes tipos de procesadores.
- ✓ Los programadores se encargaban de introducir los códigos binarios en el computador, lo que provocaba largos tiempos de preparación y posibles errores.

A continuación, se muestran algunos códigos binarios equivalentes a las operaciones de suma, resta y movimiento de datos en lenguaje máquina.

### Algunas operaciones en lenguaje máquina.

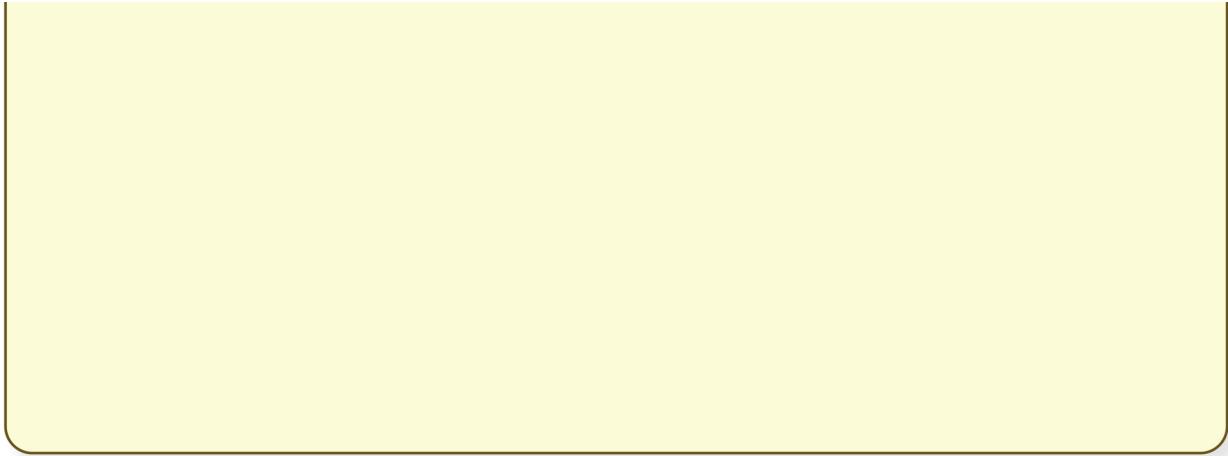
Operación	Lenguaje máquina
SUMAR	00101101
RESTAR	00010011
MOVER	00111010

Dada la complejidad y dificultades que ofrecía este lenguaje, fue sustituido por otros más sencillos y fáciles utilizar. No obstante, hay que tener en cuenta que todos los programas para poder ser ejecutados, han de traducirse siempre al lenguaje máquina que es el único que entiende la computadora.



### Para saber más

Como recordatorio, te proponemos el siguiente enlace sobre cómo funciona el sistema binario.



## 5.2.- Lenguaje Ensamblador.

La evolución del lenguaje máquina fue el lenguaje ensamblador. Las instrucciones ya no son secuencias binarias, se sustituyen por códigos de operación que describen una operación elemental del procesador. Es un lenguaje de bajo nivel, al igual que el lenguaje máquina, ya que dependen directamente del hardware donde son ejecutados.

**Mnemotécnico:** son palabras especiales, que sustituyen largas secuencias de ceros y unos, utilizadas para referirse a diferentes operaciones disponibles en el juego de instrucciones que soporta cada máquina en particular.

En ensamblador, cada instrucción (mnemotécnico) se corresponde a una instrucción del procesador. En la siguiente tabla se muestran algunos ejemplos.

### Algunas operaciones y su mnemotécnico en lenguaje Ensamblador.

Operación	Lenguaje Ensamblador
MULTIPLICAR	MUL
DIVIDIR	DIV
MOVER	MOV

En el siguiente gráfico puedes ver parte de un programa escrito en lenguaje ensamblador. En color rojo se ha resaltado el código máquina en hexadecimal, en magenta el código escrito en ensamblador y en azul, las direcciones de memoria donde se encuentra el código.



Pero aunque ensamblador fue un intento por aproximar el lenguaje de los procesadores al lenguaje humano, presentaba múltiples dificultades:

- ✓ Los programas seguían dependiendo directamente del hardware que los soportaba.
- ✓ Los programadores tenían que conocer detalladamente la máquina sobre la que programaban, ya que debían hacer un uso adecuado de los recursos de dichos sistemas.
- ✓ La lectura, interpretación o modificación de los programas seguía presentando dificultades.

Todo programa escrito en lenguaje ensamblador necesita de un intermediario, que realice la traducción de cada una de las instrucciones que componen su código al lenguaje máquina correspondiente. Este intermediario es el programa ensamblador. El programa original escrito en lenguaje ensamblador constituye el código fuente y el programa traducido al lenguaje máquina se conoce como programa objeto que será directamente ejecutado por la computadora.

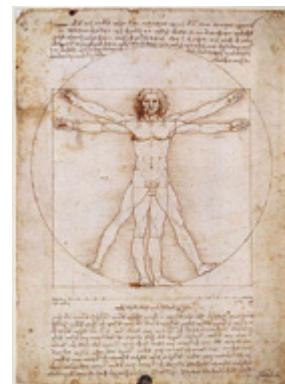
## 5.3.- Lenguajes compilados.

Para paliar los problemas derivados del uso del lenguaje ensamblador y con el objetivo de acercar la programación hacia el uso de un lenguaje más cercano al humano que al del computador, nacieron los lenguajes compilados. Algunos ejemplos de este tipo de lenguajes son: Pascal, Fortran, Algol, C, C++, etc.

Al ser lenguajes más cercanos al humano, también se les denomina **lenguajes de alto nivel**. Son más fáciles de utilizar y comprender, las instrucciones que forman parte de estos lenguajes utilizan palabras y signos reconocibles por el programador.

¿Cuáles son sus **ventajas**?

- ✓ Son mucho más fáciles de aprender y de utilizar que sus predecesores.
- ✓ Se reduce el tiempo para desarrollar programas, así como los costes.
- ✓ Son independientes del hardware, los programas pueden ejecutarse en diferentes tipos de máquina.
- ✓ La lectura, interpretación y modificación de los programas es mucho más sencilla.



Pero un programa que está escrito en un lenguaje de alto nivel también tiene que traducirse a un código que pueda utilizar la máquina. Los programas traductores que pueden realizar esta operación se llaman compiladores.

**Compilador:** Es un programa cuya función consiste en traducir el código fuente de un programa escrito en un lenguaje de alto nivel a lenguaje máquina. Al proceso de traducción se le conoce con el nombre de compilación.

El compilador realizará la traducción y además informará de los posibles errores. Una vez subsanados, se generará el programa traducido a código máquina, conocido como **código objeto**. Este programa aún no podrá ser ejecutado hasta que no se le añadan los módulos de enlace o bibliotecas, durante el proceso de enlazado. Una vez finalizado el enlazado, se obtiene el **código ejecutable**.



### Autoevaluación

Durante la fase de enlazado, se incluyen en el código fuente determinados módulos (bibliotecas) que son necesarios para que el programa pueda realizar ciertas tareas, posteriormente se obtendrá el código ejecutable.

- Verdadero  Falso

## 5.4.- Lenguajes interpretados.

Se caracterizan por estar diseñados para que su ejecución se realice a través de un **intérprete**. Cada instrucción escrita en un lenguaje interpretado se analiza, traduce y ejecuta tras haber sido verificada. Una vez realizado el proceso por el intérprete, la instrucción se ejecuta, pero no se guarda en memoria.

**Intérprete:** Es un programa traductor de un lenguaje de alto nivel en el que el proceso de traducción y de ejecución se llevan a cabo simultáneamente, es decir, la instrucción se pasa a lenguaje máquina y se ejecuta directamente. No se genera programa objeto, ni programa ejecutable.

Los lenguajes interpretados generan programas de menor tamaño que los generados por un compilador, al no guardar el programa traducido a código máquina. Pero presentan el inconveniente de ser algo más lentos, ya que han de ser traducidos durante su ejecución. Por otra parte, necesitan disponer en la máquina del programa intérprete ejecutándose, algo que no es necesario en el caso de un programa compilado, para los que sólo es necesario tener el programa ejecutable para poder utilizarlo.



Ejemplos de lenguajes interpretados son: **Perl, PHP, Python, JavaScript**, etc.

A medio camino entre los lenguajes compilados y los interpretados, existen los lenguajes que podemos denominar **pseudo-compilados o pseudo-interpretados**, es el caso del **Lenguaje Java**. Java puede verse como compilado e interpretado a la vez, ya que su código fuente se compila para obtener el código binario en forma de bytecodes, que son estructuras parecidas a las instrucciones máquina, con la importante propiedad de no ser dependientes de ningún tipo de máquina (se detallarán más adelante). La Máquina Virtual Java se encargará de interpretar este código y, para su ejecución, lo traducirá a código máquina del procesador en particular sobre el que se esté trabajando.



### Debes conocer

Puedes entender por qué Java es un lenguaje compilado e interpretado a través del siguiente esquema.

El lenguaje Java es compilado e interpretado.

## 6.- Estructuras básicas de datos

---

Son elementos de un programa todos los datos y resultados manipulados por las instrucciones de un programa. Tienen tres atributos:

- ✓ **Nombre:** con el que se identifica.
- ✓ **Tipo:** conjunto de valores que puede tomar.
- ✓ **Valor:** elemento del tipo que se le asigna en un determinado momento.

Ejemplo: Nombre: sueldo      Tipo: real      Valor: 70000.45

## 6.1.- Identificadores

---

Son palabras creadas por el programador para dar nombre a los elementos que necesita declarar en un programa.

Existen unas normas generales para su empleo:

- ✓ Pueden estar constituidos por letras, dígitos y el carácter subrayado (\_), aunque algunas herramientas permiten otros caracteres especiales.
- ✓ Deben comenzar por una letra y en algunos compiladores también por (\_).
- ✓ No deben tener espacios en blanco.
- ✓ El número máximo de caracteres que se pueden emplear depende del compilador utilizado.
- ✓ El nombre asignado conviene que tenga relación con la información que contiene, pudiéndose emplear abreviaturas que sean significativas.

Aparte de estas normas de obligado cumplimiento, el sentido común nos indica que el identificador de una variable debe indicar o reflejar de alguna forma el contenido de la misma.

## 6.2.- Datos. Tipos

Dato es toda información que se aporta a un programa. Los programas procesan datos a fin de obtener resultados.

Podemos definir **Dato** como un conjunto de símbolos que representan valores, hechos, objetos o ideas de forma adecuada para ser **tratados**

Incluso, en el ámbito de la informática, podemos definir dato de forma similar, como cualquier “objeto” manipulable por la computadora.

**Un dato puede ser un carácter leído desde teclado, un número, la información almacenada en un CD, una foto almacenada en un fichero, una canción, el nombre de un alumno almacenado en la memoria del ordenador, etc.**

Los tipos **básicos** de datos que incluyen la mayoría de los lenguajes de programación, (aunque con diferencias significativas entre unos y otros) son:

- ✓ Numéricos.
- ✓ No numéricos: Carácter, Lógicos.
- ✓ Estructurados.

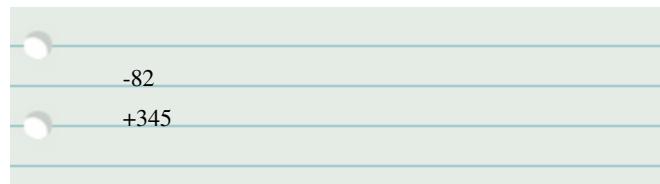
Por ejemplo, la **edad** de un trabajador la podremos representar con un tipo básico, concretamente un tipo **entero**, ya que la edad es un valor numérico positivo y sin decimales (que es justo lo que define el tipo entero, como se verá más adelante).

## 6.2.1.- Datos numéricos

Se utilizan para contener magnitudes y se clasifican en enteros y reales:

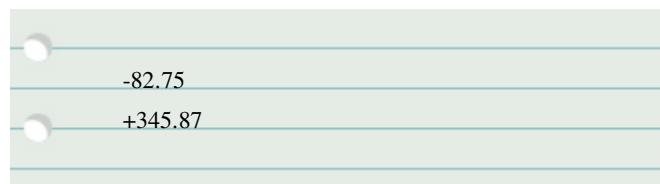
- ✓ **Enteros:** se emplean para representar números enteros cuyo rango o tamaño dependen del lenguaje y del ordenador utilizado.

Los datos de este tipo se expresan mediante una serie de dígitos, pudiendo estar precedidos del signo (+ o -).

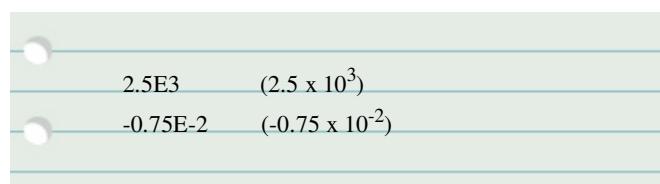


- ✓ **Reales:** se emplea para representar los números con parte decimal o los números muy grandes o muy pequeños que no pueden ser contenidos en un entero. Se pueden representar de dos formas:

- º **Punto decimal:** emplea los dígitos del 0 al 9 con su signo correspondiente y un punto para separar la parte entera de la decimal.



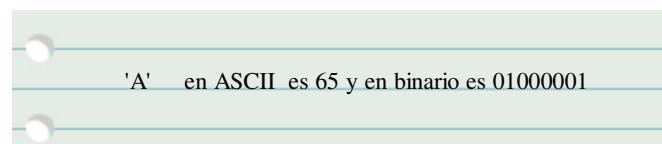
- º **Científica o exponencial:**



## 6.2.2.- Datos no numéricos

- ✓ **Carácter:** Se emplea para representar un símbolo dentro de un código definido por el fabricante del ordenador, de tal forma que cada uno de ellos se corresponde con un número entero sin signo según un determinado código.

La representación interna depende del código utilizado. Los códigos más empleados son los que utilizan 8 bits.



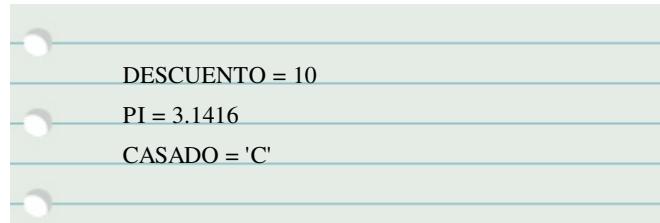
- ✓ **Lógico:** se emplea para representar dos valores opuestos, Verdadero o Falso, True o False, SI o NO, V o F, 1 ó 0. Internamente se considera 1 como verdadero y 0 como falso.

## 6.3.- Constantes

---

Son datos cuyo valor no cambia durante la ejecución del programa. Las constantes pueden ser:

- ✓ Enteras
- ✓ Reales
- ✓ Alfanuméricas o cadena de caracteres. (Su valor se indica entre comilla simple ' o doble ").

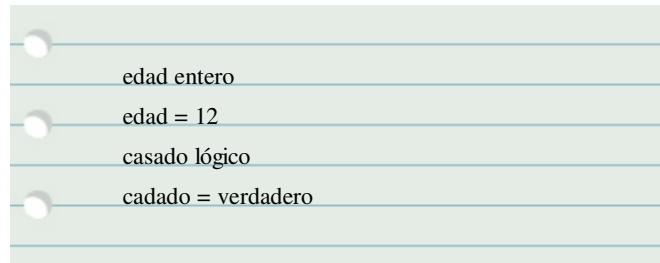


## 6.4.- Variables

Son datos cuya información puede ser variable durante la ejecución del programa. Estos datos deben ser definidos con un identificador y un tipo de dato.

El identificador es elegido por el programador y permite referenciar la variable para su uso en el programa pudiéndose modificar su valor. El tipo de dato permite determinar el tamaño de la variable en memoria. Según el tipo de dato que almacenan las variables pueden ser:

- ✓ Numéricas: nombre-variable = valor
- ✓ Alfanuméricas: nombre-variable = 'comentario'
- ✓ Lógicas: nombre-variable = valor lógico



Antes de utilizar una variable en el programa, ésta debe contener un valor que puede ser asignado inicialmente o bien durante la ejecución del programa.

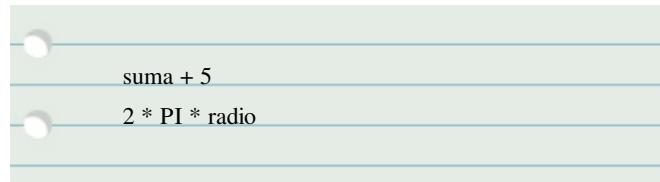
## 6.5.- Expresiones

---

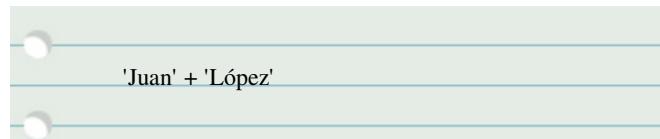
Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Cada expresión toma un valor que se determina tomando los valores de las variables y constantes implicadas y la ejecución de las operaciones existentes.

Por tanto una expresión es un conjunto de datos (operándos) y operadores, con unas reglas específicas de construcción. Según sea el resultado que producen y los operadores que utilizan se clasifican en:

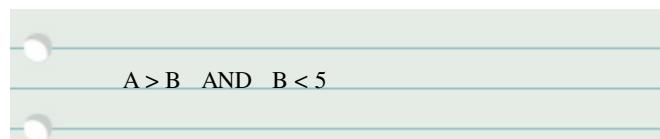
- ✓ Numéricas: son las que producen resultados de tipo numérico. Se construyen mediante operadores aritméticos.



- ✓ Alfanuméricos: son las que producen resultados alfanuméricicos. Se construyen mediante operadores alfanuméricos.



- ✓ Lógicas: son las que producen resultados verdadero o falso. Se construyen mediante los operadores lógicos y/o relationales.



## 6.6.- Funciones

---

Las operaciones que se requieren en los programas exigen en numerosas ocasiones, además de las operaciones aritméticas básicas, un número determinado de operadores espaciales que se denominan “funciones internas”, incorporadas o estándar.

Cada lenguaje de programación tiene sus propias funciones, entre las comunes y más utilizadas están las siguientes:

- ✓  $\text{sqrt}(x)$  raíz cuadrada de un número positivo
- ✓  $\text{abs}(x)$  valor absoluto
- ✓  $\cos(x)$  coseno
- ✓  $\sin(x)$  seno

## 6.7.- Operadores

Los operadores son símbolos que sirven para conectar los datos haciendo diversas clases de operaciones.

En función de las operaciones a realizar los operadores se clasifican en:

Los operadores **aritméticos** son aquellos operados que combinados con los operandos forman expresiones matemáticas o aritméticas.

### Operadores aritméticos básicos

Operador	Significado	Expresión	Resultado
-	Operador unario de cambio de signo	-10	-10
+	Adición	1.2 + 9.3	10.5
-	Sustracción	312.5 – 12.3	300.2
*	Multiplicación	1.7 * 1.2	2.04
/	División (entera o real)	0.5 / 0.2	0.25
%	Resto de la división entera	25 % 3	1
** o ^	Potencia	3 ^ 2	9

Los operadores **relacionales** se utilizan para comparar datos de tipo primitivo (numérico, carácter y booleano). El resultado se utilizará en otras expresiones o sentencias, que ejecutarán una acción u otra en función de si se cumple o no la relación.

### Operadores relationales

Operador	Ejemplo	Significado
== o =	op1 == op2	op1 igual a op2
!= o <>	op1 != op2	op1 distinto de op2
>	op1 > op2	op1 mayor que op2
<	op1 < op2	op1 menor que op2
>=	op1 >= op2	op1 mayor o igual que op2
<=	op1 <= op2	op1 menor o igual que op2

Los operadores **lógicos** realizan operaciones sobre valores lógicos, o resultados de expresiones relationales, dando como resultado un valor lógico.

### Operadores lógicos

Operador	Ejemplo	Significado
----------	---------	-------------

Operador	Ejemplo	Significado
NOT o No	NOT op	Negación
AND o Y	op1 AND op2	Conjunción
OR o O	op1 OR op2	Disyunción

Operadores **alfanuméricicos** realizan operaciones sobre cadenas de caracteres.

### Operadores alfanuméricicos

Operador	Ejemplo	Significado
+	'Luis' + 'Pérez'	Concatenación

## 6.7.1.- Orden de precedencia

Dentro de las expresiones hay que tener un orden de **prioridad de los operadores**, que depende del lenguaje de programación utilizado, pero que de forma general se puede establecer de mayor a menor prioridad de la siguiente forma:

### Orden de precedencia de operadores

Operador	Tipo
( )	Paréntesis de izquierda a derecha
-	Signo
**	Potencia
* / %	Producto, división, módulo
* -	Suma, resta
+	Concatenación
NOT	Negación
AND	Conjunción
OR	Disyunción

## 6.7.2.- Tablas de verdad

---

En las operaciones lógicas se determina su resultado por medio de las tablas de la verdad:

**V:** verdadero    **F:** falso

**Negación**

A	not A	
V	F	
F	V	

**Conjunción**

A	B	A and B
V	V	V
V	F	F
F	V	F
F	F	F

**Disyunción**

A	B	A or B
V	V	V
V	F	V
F	V	V
F	F	F

## 7.- El pseudocódigo

Para la representación de un algoritmo se debe utilizar algún método que permita independizar dicho algoritmo del lenguaje de programación elegido. Esto permitirá que un algoritmo pueda ser codificado indistintamente en cualquier lenguaje.

Nosotros utilizaremos el pseudocódigo.

**Pseudocódigo** es una técnica utilizada para la descripción de un algoritmo utilizando un lenguaje intermedio entre el lenguaje natural y el lenguaje de programación.

La escritura o diseño de un algoritmo mediante pseudocódigo, exige la “**indentación**” o “sangria” del texto en el margen izquierdo de las diferentes líneas, lo que facilita el entendimiento y comprensión del diseño realizado.

Todo algoritmo representado en pseudocódigo deberá reflejar las siguientes partes:

- ✓ **Cabecera:** Es el área o bloque informativo donde quedará reflejado el nombre del programa.
- ✓ **Cuerpo:** Es el resto del diseño, el cual queda dividido en dos bloques, el bloque de datos, donde deberán quedar descritos todos los elementos de trabajo necesarios para la ejecución del programa (**Entorno**), y el bloque de acciones que es la zona en la que se deben describir con máxima claridad y detalle todas aquellas acciones que el ordenador deberá realizar durante la ejecución del programa(**Algoritmo**).

En el algoritmo se suelen considerar tres partes:

- ✓ **Entrada:** Información dada al algoritmo.
- ✓ **Proceso:** Operaciones o cálculos necesarios para encontrar la solución del problema.
- ✓ **Salida:** Respuestas dadas por el algoritmo o resultados finales de los cálculos.



### Ejercicio resuelto

Realizar el pseudocódigo de un programa que permita calcular el **área de un rectángulo**, introduciremos por teclado el valor de la base y de la altura. Etiquetar tanto la entrada de datos como la salida de resultados.

Lo primero que debes hacer es plantearte y contestar a las siguientes preguntas:

Especificaciones de **entrada**: ¿Qué datos son de entrada?, ¿Cuántos datos se introducirán?, ¿Cuántos son datos de entrada válidos?

Especificaciones de **salida**: ¿Cuáles son los datos de salida?, ¿Cuántos datos de salida se producirán?, ¿Qué precisión tendrán los resultados?, ¿Se debe imprimir una cabecera o etiquetar los datos?

- ✓ Paso 1.- Entrada desde periférico, por ejemplo teclado, de base y altura.
- ✓ Paso 2. Cálculo de la superficie, multiplicando base por la altura.
- ✓ Paso 3. Salida por pantalla de área.



## Debes conocer

A continuación te ofrecemos varios enlaces interesantes:

- ✓ Aquí tienes información sobre PSelint, una herramienta educativa, utilizada para aprender los fundamentos de la programación. <http://es.wikipedia.org/wiki/PSelint>  
Incluye ejemplos de pseudocódigos.  
Podrás descargarlo en esta dirección: <http://pseint.uptodown.com/>
- ✓ Aquí tienes un vídeo en el que se explica cómo utilizarlo.

## 7.1.- Tipos de instrucciones

---

Las instrucciones disponibles en un lenguaje de programación dependen del tipo de lenguaje.

Hay una serie de instrucciones llamadas instrucciones **básicas** que se utilizan de modo general en un algoritmo y que esencialmente soportan todos los lenguajes. Estas instrucciones básicas se clasifican en:

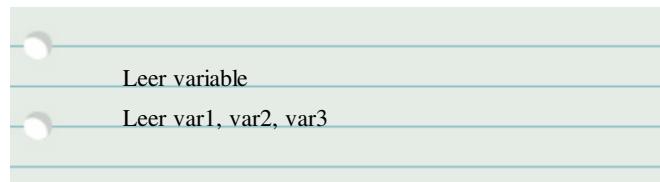
- ✓ Instrucciones de **definición** de datos. Son aquellas instrucciones utilizadas para informar al procesador del espacio que debe reservar en memoria con la finalidad de almacenar un dato mediante el uso de variables simples o estructuras más complejas. La definición consiste en indicar un nombre a través del cual haremos referencia a un dato y un tipo a través del cual informaremos al procesador de las características y espacio que deberá reservar la memoria.
- ✓ Instrucciones **primitivas**. Son aquellas que ejecuta el ordenador de modo inmediato, ya que no dependen de otra cosa que de su propia aparición en el programa. Pueden ser: de entrada, de asignación y de salida.
- ✓ Instrucciones **compuestas**. Son aquellas instrucciones que no pueden ser ejecutadas directamente por el procesador, y están constituidas por un bloque de acciones agrupadas en subrutinas, subprogramas, funciones o módulos.
- ✓ Instrucciones **de control**. Son instrucciones que controlan el flujo de ejecución de otras instrucciones.

## 7.1.1.- Instrucciones primitivas

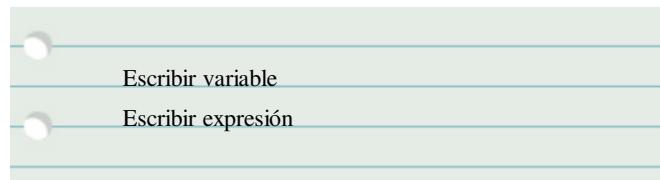
Son aquellas que ejecuta el ordenador de modo inmediato, ya que no dependen de otra cosa que de su propia aparición en el programa. Pueden ser: de entrada, de asignación y de salida.

- ✓ Sentencias de **entrada**. Son aquellas instrucciones encargadas de recoger datos de uno o varios dispositivos de entrada y almacenarlos en la memoria central en las variables que aparecen en la propia instrucción y que previamente han sido definidas.

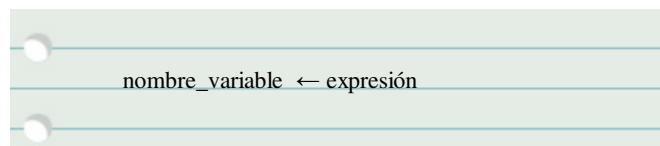
En pseudocódigo se utiliza de la siguiente forma:



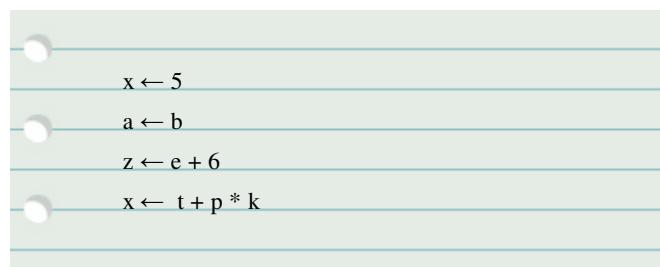
- ✓ Sentencias de **salida**. Son aquellas instrucciones encargadas de recoger los datos procedentes de variables (previamente definidas) a los resultados obtenidos de una expresión evaluada y depositarlos en un periférico o dispositivo de salida.



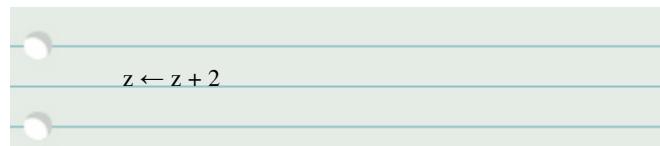
- ✓ Sentencias de **asignación**. Son instrucciones que se utilizan para asignar valores a variables, o cambiar el valor almacenado en una variable obtenido de la evaluación de una expresión.



La operación de asignación es destructiva ya que al almacenarse un nuevo valor en la variable se borra el que tenía antes.



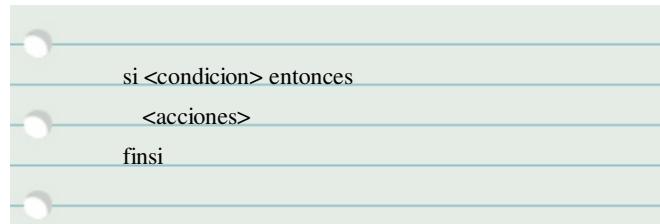
Es posible utilizar el mismo nombre de variable en ambos lados del operador de asignación.



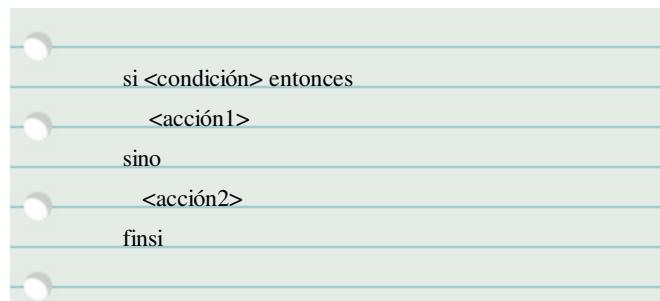
## 7.1.2.- Instrucciones alternativas

Controlan la ejecución o la no ejecución de uno o varios bloques de instrucciones dependiendo del cumplimiento o no de una condición o del valor de una expresión. Pueden ser:

- ✓ **Simples**: controlan la ejecución de un conjunto de instrucciones por el cumplimiento o no de una condición. Si la condición se cumple se ejecuta el grupo de instrucciones y si no se cumple no se ejecuta nada.



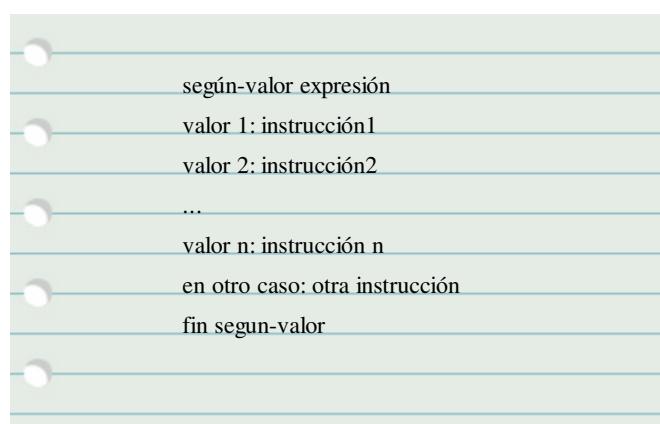
- ✓ **Dobles**: controla la ejecución de dos conjuntos de instrucciones dependiendo del cumplimiento o no de una condición. Si se cumple se ejecutan unas instrucciones y si no se cumplen se ejecutan otras.



- ✓ **Múltiples**: controla la ejecución de varios conjuntos de instrucciones, por el valor de una expresión, de forma que cada conjunto de instrucciones está ligado a un posible valor de dicha expresión.

Se ejecutará el conjunto de instrucciones que se encuentra relacionado con el valor que resulte de evaluar la expresión.

Las distintas opciones tienen que ser disjuntas, es decir, solo pueden ejecutarse un conjunto de instrucciones a la vez.



En otras ocasiones, a esta instrucción de alternativa múltiple se la denomina “En caso de”.

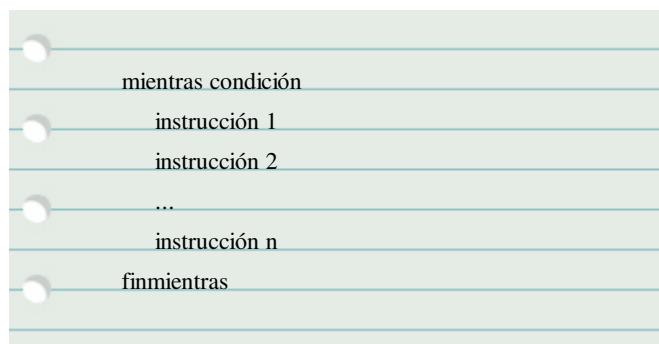
## 7.1.3.- Instrucciones repetitivas

Controlan la ejecución repetitiva de un conjunto de instrucciones mediante una condición que determina el número de veces que se ha de repetir esta ejecución.

El valor de la condición tiene que estar afectado por la ejecución de las instrucciones para asegurar la terminación de esta repetición. Existen varios tipos de estructuras repetitivas:

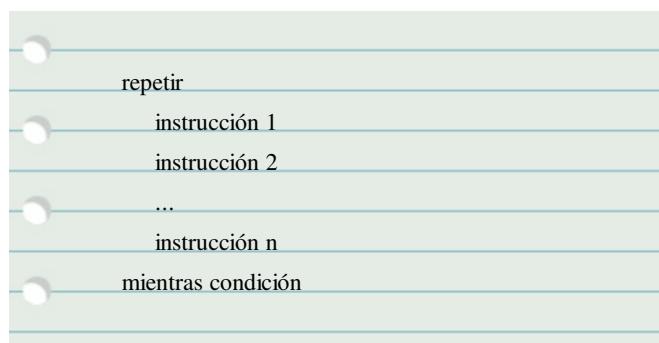
- ✓ Estructura **mientras**: controla la ejecución de un conjunto de instrucciones de tal forma que estas se ejecutan mientras se cumpla la condición, que será evaluada siempre antes de cada repetición.

Puede que las instrucciones no se ejecuten nunca.



- ✓ Estructura **repetir**: controla la ejecución de un conjunto de instrucciones, de tal forma que estas se ejecutan mientras que se cumpla la condición que será evaluada siempre después de cada repetición.

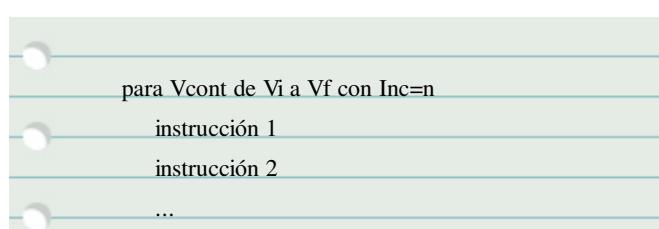
Las instrucciones se ejecutan al menos una vez.



- ✓ Estructura **para**: controla la ejecución de un conjunto de instrucciones, de tal forma que estas se ejecutan un número determinado de veces que se conoce de antemano.

Esta estructura lleva asociada una variable que actúa como contador. El contador parte de un valor inicial y se irá incrementando en cada repetición hasta llegar al valor final que es cuando se detiene la ejecución.

A este contador se le llama variable de control.



instrucción n  
finpara

## 7.2.- Elementos auxiliares

Los elementos auxiliares de un programa son variables que realizan funciones específicas dentro de un algoritmo. Las más importantes son los **contadores**, **acumuladores** e **interruptores**.

Un **contador** es una variable cuyo valor se incrementa en una cantidad fija, positiva o negativa. Se utiliza en los siguientes casos:

- ✓ Para contar la ocurrencia de un suceso particular. Por ejemplo, contar cuantos números son positivos de una serie.
- ✓ Para contabilizar el número de veces que es necesario repetir una acción. Normalmente, esta utilidad viene asociada a un bucle o sentencia de control repetitiva.

Es necesario darles un valor inicial, que en bastantes casos es cero.

Se utilizan expresiones del tipo     $\text{CONT} = \text{CONT} + 1$

Un **acumulador** es una variable cuyo valor se incrementa sucesivas veces en cantidades variables.

Se utiliza en aquellos casos en los que se desea obtener el total acumulado de un conjunto de cantidades. También es precisa su inicialización, habitualmente a cero.

También en las situaciones en las que hay que obtener un total como producto de distintas cantidades se utiliza un acumulador, pero en este caso el valor inicial que debe darse al mismo es 1.

Se utilizan expresiones del tipo     $\text{ACUM} = \text{ACUM} + \text{VAR}$

Un **interruptor** o **switch** es una variable que solamente puede tomar dos valores exclusivos (0 y 1, verdadero y falso, si y no).

Se utiliza en los siguientes casos:

Recordar en un determinado punto de un programa la ocurrencia o no de un suceso anterior, para salir de un bucle o iteración, o para decidir en una instrucción alternativa que acción realizar. Se inicializa el interruptor a un valor y cuando se produzca un suceso se modifica el valor del mismo, siendo analizado posteriormente en cualquier lugar del programa.

Para hacer que dos acciones diferentes se ejecuten alternativamente dentro de un bucle o iteración.

## 8.- Programación modular

La programación modular es el resultado de la aplicación del concepto de diseño descendente **TOP-DOWN**. El diseño descendente consiste en descomponer sucesivamente el problema original, partiendo de la base de que es más fácil dar una solución a cada subdivisión del problema, para posteriormente implementarlos en un único problema.

El resultado final de la programación modular es la división de un programa en un conjunto de módulos independientes que se comunican entre sí a través de llamadas.

Un módulo representa una tarea determinada dentro del conjunto de tareas que han de ser realizadas en un programa y estará formado por un conjunto de instrucciones identificadas mediante un nombre (nombre del módulo) por el cual serán invocadas desde otros módulos del programa. Un módulo deberá tener siempre un nombre de inicio donde comience a ejecutarse y un punto de salida y retorno, donde el módulo devuelva el control al módulo desde el cual fue invocado.

Dentro de los distintos módulos en los que puede ser dividido un programa, siempre tiene que haber uno que describa la solución completa del problema y que servirá de nexo de unión entre todos los demás, es el módulo o **programa principal**.

Cuando un módulo es un programa independiente, se dice que tenemos un módulo externo. En este caso el módulo genera su propio objeto que se unirá al resto del código objeto del programa durante la fase del linkado. Por el contrario, cuando los módulos se encuentran dentro del mismo código fuente, se dice que los módulos son internos. Los diferentes lenguajes tienen diferentes tipos de módulos, como funciones, procedimientos, métodos, dependiendo del lenguaje y de la utilización del módulo. La programación modular exige una comunicación entre el módulo llamador y el módulo llamado. Este tipo de comunicación se realiza a través de variables de enlace a las que se denominan **parámetros** o argumentos.

En general, un módulo se comporta como una caja negra, que recibe a través de los argumentos una serie de datos que le ayudarán a realizar el cometido para el que fue diseñado. El programador que invoque al módulo únicamente debe saber que datos tiene que pasarle y que hace el módulo, sin preocuparse de su código interno.

**Función.** En programación una función es un módulo que nos proporciona un valor a partir de los valores de unos argumentos que le son pasados.

acumulador = acumulador + **potencia (numero1, numero2)**

donde **potencia(numero1,numero2)** es una llamada a una función a la que se pasan dos argumentos.

Es conveniente declarar las funciones en el entorno del programa, esto facilita la detección de errores. Igualmente es conveniente declararlas en el código de los programas fuente, puesto que permite optimizar la gestión de la memoria y detectar errores de tipo en las llamadas a las funciones.

Entorno

entero FUNCION potencia (entero x, entero y)

**Procedimiento.** Es similar a la función salvo que no devuelve ningún valor. La obtención de un resultado en un procedimiento tiene que realizarse a través de los argumentos. El paso de parámetros, cada lenguaje establece los mecanismos de cómo realizar esta tarea, puede realizarse de dos maneras:

- ✓ Paso de parámetros **por valor**: los parámetros se pasan entre los módulos, no estableciéndose ninguna relación entre los mismos, por lo que si realizamos modificaciones en el valor pasado, los cambios se pierden al retornar al módulo de llamada.
- ✓ Paso de parámetros **por dirección**: los parámetros se pasan entre los módulos a través de la dirección de memoria donde residen por lo que si realizamos modificaciones en el valor pasado, los cambios no se pierden al retornar al módulo de llamada.

**Variables Globales**.- Son aquellas variables que pueden ser accedidas y modificadas desde cualquier módulo o función. En general las variables globales se declaran fuera de cualquier módulo.

**Variables Locales**.- Son aquellas variables que pueden ser accedidas y modificadas únicamente desde el módulo o función en que fueron declaradas.