








CONCEPTOS DE PROGRAMACIÓN

Hardware: Conjunto de elementos físicos o materiales que constituyen una computadora o sistema informático.

Software: Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

Algoritmo: Conjunto ordenado de instrucciones que permiten hacer un cálculo o resolver un problema.

Diagrama de flujo: Representación gráfica de un proceso; utiliza símbolos específicos para representar cada tipo de acción a ejecutar.

Nombre	Símbolo	Función
Terminal		Representa el inicio y fin de un programa. También puede representar una parada o interrupción programada que sea necesaria realizar en un programa.
Entrada / salida		Cualquier tipo de introducción de datos en la memoria desde los periféricos o registro de información procesada en un periférico.
Proceso		Cualquier tipo de operación que pueda originar cambio de valor, formato o posición de la información almacenada en memoria, operaciones aritméticas, de transformaciones, etc.
Decisión		Indica operaciones lógicas o de comparación entre datos (normalmente dos) y en función del resultado de la misma determina (normalmente si y no) cual de los distintos caminos alternativos del programa se debe seguir
Conector Misma Página		Sirve para enlazar dos partes cualesquiera de un diagrama a través de un conector en la salida y otro conector en la entrada. Se refiere a la conexión en la misma página del diagrama
Indicador de dirección o línea de flujo		Indica el sentido de la ejecución de las operaciones
Salida		Se utiliza en ocasiones en lugar del símbolo de salida. El dibujo representa un pedazo de hoja. Es usado para mostrar datos o resultados.

Pseudocódigo: Algoritmo estructurado con el formato de un lenguaje de programación.

Programa: Secuencia de instrucciones realizadas en un lenguaje diseñado para ser interpretado por una computadora.

Lenguaje de programación: conjunto de reglas, símbolos y palabras especiales que permiten construir un programa. Al igual que los lenguajes

humanos, tales como el inglés o el español, los lenguajes de programación poseen una estructura (gramática o sintaxis) y un significado (semántica).

Sintaxis: conjunto de reglas que gobiernan la construcción o formación de sentencias (instrucciones) válidas en un lenguaje. Significa decir cómo se escriben los enunciados, declaraciones y otras construcciones de lenguaje. El estilo sintáctico general de un lenguaje de programación, está dado por:

1.-**Identificadores.** Conjunto de caracteres que sirven para dar nombre a las entidades del programa tales como variables, constantes, clases, funciones, procedimientos, etc. Cuando un identificador se asocia a una entidad concreta, entonces es el "nombre" de dicha entidad y en adelante la representa en el programa. Su longitud y reglas que definen como pueden estar contruidos dependen de cada lenguaje, pero por lo general, deben comenzar con una letra y solo pueden contener, letras, dígitos y el carácter de subrayado (_).

2.-**Operadores.** Son los símbolos que indican al compilador que realice manipulaciones lógicas, relacionales o matemáticas específicas. Los más comunes son: +, -, *, /, >, <, pero dependen de la especificación de cada lenguaje.

3.-**Palabras reservadas.** Tienen un significado especial para el lenguaje y no se pueden utilizar como identificadores.

4.-**Comentarios.** Son anotaciones en el código, significativas para el programador, pero ignoradas por los compiladores e intérpretes. La sintaxis y reglas para los comentarios varían pues son definidas por cada lenguaje.

5.-**Espacios en blanco.**

6.-**Delimitadores.**

7.-**Expresiones.** Combinación de constantes, variables, funciones u operaciones que son interpretadas de acuerdo a las normas particulares de cada lenguaje.

Variables: Es una posición de memoria donde se puede almacenar un valor para uso de un programa.

Constantes: Son valores que no pueden ser modificados durante la ejecución de un programa.

Tipos de datos: Determinan la naturaleza del conjunto de valores que puede tomar una variable.

Sistema Operativo: Conjunto de programas que permiten la administración de los recursos de una computadora, gestionan el hardware desde los niveles más básicos, gestiona el funcionamiento de los programas y aplicaciones instalados y permite la interacción con el usuario.

Compilador: es un programa que traduce el programa fuente (conjunto de instrucciones de un lenguaje de alto nivel) a un programa objeto (instrucciones en lenguaje máquina que la computadora pueda interpretar y ejecutar). El compilador efectúa solo la traducción, no ejecuta el programa. Una vez compilado el programa, el resultado en forma de programa objeto será directamente ejecutable.

Intérprete: es un programa que traduce cada instrucción o sentencia del programa escrito en un lenguaje de alto nivel a código máquina e inmediatamente se ejecuta, y a continuación se ejecuta la siguiente sentencia. El intérprete está diseñado para trabajar con la computadora en modo conversacional o interactivo.

TIPOS DE ERRORES DE PROGRAMACIÓN

Errores de compilación: impiden que el programa se ejecute, se deben a errores cometidos al escribir el código, por ejemplo: escribir mal una palabra reservada, omitir delimitadores necesarios, entre otros.

Errores en tiempo de ejecución: aparecen mientras se ejecuta el programa al intentar realizar operaciones que son imposibles de llevar a cabo. Un ejemplo de esto es la división por cero.

Errores lógicos: son errores que impiden que el programa haga lo que estaba previsto; el código puede compilarse y ejecutarse sin errores, pero el resultado no es el esperado.

EVOLUCIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Lenguaje máquina: Primera generación. La codificación se hace utilizando un lenguaje binario de ceros y unos.

Lenguaje bajo nivel. Segunda generación. Lenguaje ensamblador: Constituye la representación más directa del código máquina, en el cual, se le asigna un código mnemotécnico a cada comando del lenguaje máquina, por lo general formado por tres o cuatro letras para cada comando. En la actualidad, se suele usar cuando se pretende conseguir un uso de recursos de hardware controlado y reducido y/o cuando se pretende conseguir altos rendimientos. Muchos dispositivos programables, (como microcontroladores) aun cuentan con el ensamblador como la única manera de ser manipulados.

Lenguajes alto nivel. Tercera generación. FORTRAN, C, COBOL, PASCAL, BASIC, ADA, RPG, PL1, SIMULA, LISP, Java, entre otros. Estos lenguajes logran la independencia del tipo de máquina y se aproximan al lenguaje natural.

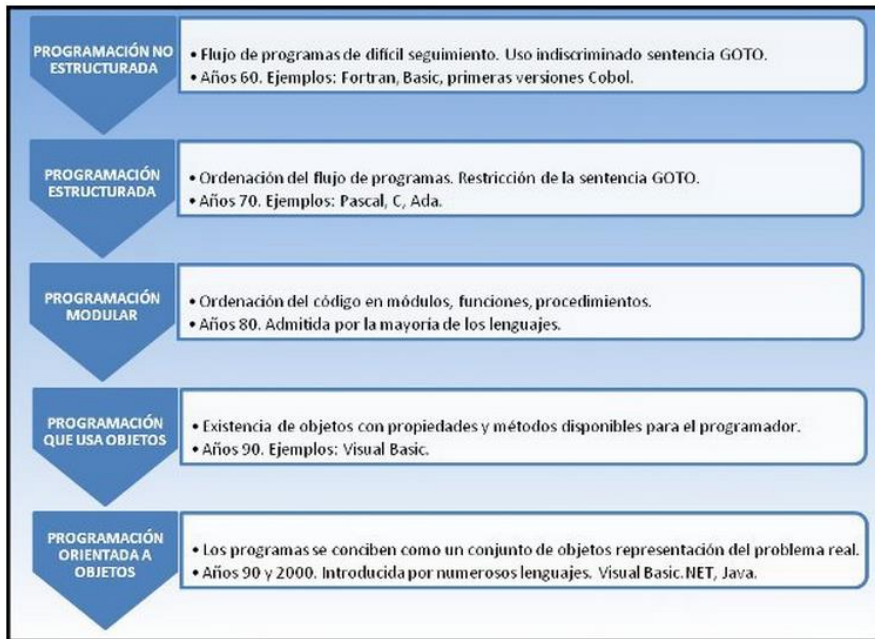
ALGUNAS FORMAS DE PROGRAMACIÓN:

Programación estructurada: Está orientada a mejoras en la claridad, calidad y tiempo de desarrollo del programa a través de la restricción de la sentencia GOTO, promoviendo el uso procedimientos y funciones.

Programación Visual: Brinda un entorno visual amigable para el desarrollo de programas. Incorporan una completa implementación de la programación orientada a objetos y permiten aprovechar al máximo toda la funcionalidad que ofrecen estos lenguajes para el desarrollo de aplicaciones de gestión.

Programación Orientada a Objetos: Define los programas en términos de "clases de objetos". Se basa en la programación de clases, objetos y sus propiedades, como herencia, polimorfismo, encapsulamiento.

Esquema simplificado sobre la evolución de los lenguajes.



Este esquema es meramente orientativo: es imposible reflejar la diversidad de lenguajes y su evolución en un gráfico tan simplificado. También los datos relativos a fechas son orientativos

PROGRAMACIÓN ORIENTADA A OBJETOS.

La Programación Orientada a Objetos (POO u OOP) es un paradigma de programación que define los programas en términos de "clases de objetos", que son entidades que combinan **estado** (propiedades o datos), **comportamiento** (procedimientos o *métodos*) e **identidad** (propiedad del objeto que lo diferencia del resto).

La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que **colaboran** entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

Un **objeto** contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases (e incluso entre objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos). A su vez, dispone de mecanismos de interacción (los llamados métodos) que favorecen la comunicación entre objetos (de una misma clase o de distintas), y en consecuencia, el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan (ni deben separarse) información (datos) y procesamiento (métodos).

Las **clases** son declaraciones o abstracciones de objetos, lo que significa, que una clase es la definición de un objeto. Cuando se programa un objeto y se definen sus características y funcionalidades, realmente se programa una clase.

La programación Orientada a objetos (POO) es una forma especial de programar, más cercana a como expresaríamos las cosas en la vida real que otros tipos de programación. Con la POO tenemos que aprender a pensar las cosas de una manera distinta, para escribir nuestros programas.

Motivación

Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que los esfuerzos de los programadores puedan ser utilizados por otras personas se creó la POO. Que es una serie de normas de realizar las cosas de manera que otras personas puedan utilizarlas y adelantar su trabajo, de manera que consigamos que el código se pueda reutilizar.

La POO no es difícil, pero es una manera especial de pensar, a veces subjetiva de quien la programa, de manera que la forma de hacer las cosas puede ser diferente según el programador. Aunque podamos hacer los programas de formas distintas, no todas ellas son correctas, lo difícil no es programar orientado a objetos sino programar bien. Programar bien es importante porque así podemos aprovechar de todas las ventajas de la POO.

Cómo se piensa en objetos

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Por ejemplo vamos a pensar en un coche para tratar de modelizarlo en un esquema de POO. Diríamos que el coche es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca. Además tiene una serie de funcionalidades asociadas, como pueden ser ponerse en marcha, parar o aparcarse. Pues en un esquema POO el coche sería el objeto, las propiedades serían las características como el color o el modelo y los métodos serían las funcionalidades asociadas como ponerse en marcha o parar.

Por poner otro ejemplo vamos a ver cómo modelizaríamos en un esquema POO una fracción, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo $3/2$. La fracción será el objeto y tendrá dos propiedades, el numerador y el denominador. Luego podría tener varios métodos como simplificarse, sumarse con otra fracción o número, restarse con otra fracción, etc.

Estos objetos se podrán utilizar en los programas, por ejemplo en un programa de matemáticas se haría uso de objetos fracción y en un programa que gestione un taller de coches se utilizarían objetos coche. Los programas Orientados a objetos utilizan muchos objetos para realizar las acciones que se desean realizar y ellos mismos también son objetos. Es decir, el taller de coches será un objeto que utilizará objetos coche, herramienta, mecánico, recambios, etc.

Existe un acuerdo acerca de qué características contempla la "orientación a objetos". Las siguientes características son las más importantes:

Abstracción: Denota las características esenciales de un objeto, donde se capturan sus comportamientos. Se refiere a poder identificar el posible comportamiento de un objeto para posteriormente convertirlo en sus métodos y funciones dentro de una clase.

Encapsulamiento: Consiste en reunir todos los elementos de un posible objeto, los cuales se verán como atributos del mismo, para luego realizar el tratamiento de sus datos de mejor forma. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

Modularidad: Es la propiedad que permite subdividir una aplicación en partes más pequeñas (llamadas módulos), cada una de las cuales debe ser tan independiente como sea posible de la aplicación en sí y de las restantes partes. Estos módulos se pueden compilar por separado, pero tienen conexiones con otros módulos. Al igual que la encapsulación, los lenguajes soportan la modularidad de diversas formas.

Principio de ocultación: Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una "interfaz" a otros objetos que especifica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas; solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no puedan cambiar el estado interno de un objeto de manera inesperada, eliminando efectos secundarios e interacciones inesperadas.

Polimorfismo: Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. Cuando esto ocurre en "tiempo de ejecución", esta última característica se llama **asignación tardía** o **asignación dinámica**.

Herencia: Las clases no se encuentran aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento, permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir (y extender) su comportamiento sin tener que volver a implementarlo.