

Paradigmas y Lenguajes

Introducción



Msc. Ricardo Monzón

PARADIGMAS Y LENGUAJES

**LICENCIATURA EN SISTEMAS DE INFORMACION
FACULTAD DE CIENCIAS EXACTAS Y NATURALES Y AGRIMENSURA-UNNE**

2DO. AÑO PRIMER CUATRIMESTRE

PROFESOR TEORIA:

- **Msc. RICARDO MONZON**

PROFESORES DE PRACTICO

- **LIC. SONIA CORRALES**
- **LIC. MONICA RINGA**
- **LIC. LEANDRO RODRIGUEZ**

AYUDANTES DE PRACTICA

- **LIC. MATIAS MASCAZZINI**
- **LIC. WALTER RAMIREZ**

PARADIGMAS Y LENGUAJES

LICENCIATURA EN SISTEMAS DE INFORMACION
FACULTAD DE CIENCIAS EXACTAS Y NATURALES Y AGRIMENSURA-UNNE

HORARIOS

Teoría

Lunes de 14:00 a 17:00

Sábados de 09:00 a 12:00

Práctica

Lunes de 17:00 a 20:00 – Grupo 1

Miércoles de 14:00 a 17:00 – Grupo 2

Laboratorios

Laboratorios

Jueves 14:00 a 16:00, 16:00 a 18:00, 20:00 a 22:00.

Viernes de 18:00 a 20:00



PARTE I: INTRODUCCION.

TEMA 1: PARADIGMAS y Paradigmas de Programación



Definición Teórica - Paradigma

*«Un **paradigma** está constituido por los supuestos teóricos generales, las leyes y las técnicas para su aplicación que adoptan los miembros de una determinada comunidad científica.»*

Podemos decir que, los paradigmas son marcos de referencia que imponen reglas sobre cómo se deben hacer las cosas, indican qué es válido dentro del paradigma y qué está fuera de sus límites.

Un paradigma distinto implica nuevas reglas, elementos, límites y maneras de pensar, o sea implica un cambio.

Definición Teórica – Paradigma de Programación

Los paradigmas pueden ser considerados como patrones de pensamiento para la resolución de problemas. Desde luego siempre teniendo en cuenta los lenguajes de programación, según nuestro interés de estudio.

*Un **paradigma de programación** es entonces una forma de representar y manipular el conocimiento. Representa un enfoque particular o filosofía para la construcción del software.*

No es mejor uno que otro sino que cada uno tiene ventajas y desventajas. También hay situaciones donde un paradigma resulta más apropiado que otro.

Ejemplos de Paradigmas de Programación



Tipos de Paradigmas de Programación

1. **Paradigma Imperativo:** Modelo abstracto que consiste en un gran almacenamiento de memoria donde la computadora almacena una representación codificada de un cálculo y ejecuta una secuencia de comandos que modifican el contenido de ese almacenamiento. Algoritmos + Estructura de Datos = Programa.
2. **Paradigma Procedimental:** es un enfoque en programación que se basa en descomponer un problema en una serie de pasos o procedimientos bien definidos. Estos procedimientos, llamados funciones o subrutinas, se utilizan para estructurar el código y resolver tareas específicas. Es una extensión del paradigma imperativo y es muy intuitivo, ya que sigue un flujo lógico y secuencial.

Tipos de Paradigmas de Programación

3. Paradigma Declarativo. - Modelos de Desarrollo: Funcional y Lógico. Se construye señalando hechos, reglas, restricciones, ecuaciones, transformaciones y otras propiedades derivadas del conjunto de valores que configuran la solución.

3.1. Paradigma Funcional: Modelo matemático de composición funcional donde el resultado de un cálculo es la entrada del siguiente, y así sucesivamente hasta que una composición produce el valor deseado. Como su nombre lo dice operan solamente a través de funciones. Cada función devuelve un solo valor, dada una lista de parámetros. La programación funcional proporciona la capacidad para que un programa se modifique así mismo, es decir que pueda aprender.

Tipos de Paradigmas de Programación

3.2. Paradigma Lógico: Esta programación se basa en un subconjunto del cálculo de predicados, incluyendo instrucciones escritas en formas conocidas como cláusulas de Horn (sentencias de lógica de primer orden). Este paradigma puede deducir nuevos hechos a partir de otros hechos conocidos. Permite un método particularmente mecánico de demostración llamado resolución. Representa conocimiento mediante relaciones (predicados) entre objetos (datos).

Un programa lógico consiste en un conjunto de relaciones, y su ejecución demuestra que una nueva relación se sigue de las que constituía el programa. Las relaciones se especifican con reglas y hechos. La ejecución consiste en la demostración de hechos sobre las relaciones por medio de preguntas.

Tipos de Paradigmas de Programación

4. Paradigma Demostrativo.- Modelos de Desarrollo: Genético. Cuando se programa bajo un paradigma demostrativo (también llamada programación por ejemplos), el programador no especifica procedimentalmente cómo construir una solución sino que presentan soluciones de problemas similares.

5. Paradigma Orientado a Objetos: disciplina de ingeniería de desarrollo y modelado de software que permite construir más fácilmente sistemas complejos a partir de componentes individuales. Objetos + Mensajes = Programa. Describen los lenguajes que soportan objetos en interacción. Un objeto es un grupo de procedimientos que comparten un estado.

Tipos de Paradigmas de Programación

6. Paradigma Reactivo: Es un enfoque de programación diseñado para manejar flujos de datos y eventos de manera eficiente y en tiempo real. Se enfoca en la idea de que un sistema debe reaccionar a cambios en su entorno o a flujos de datos que se actualizan continuamente.

7. Paradigma Basado en Componentes: es un enfoque en que se centra en construir aplicaciones a partir de componentes reutilizables e independientes. Un componente es una unidad modular que encapsula funcionalidad y datos, y se integra fácilmente con otros para formar sistemas más grandes.

8. Paradigma Concurrente y Paralelo: es un enfoque en la programación que permite la ejecución de múltiples tareas al mismo tiempo, aunque ambos conceptos tienen diferencias clave.

Ejemplos de Lenguajes asociados a paradigmas de programación:

- El paradigma imperativo es considerado el más común y está representado, por ejemplo, por el C, por BASIC o Pascal.
- El paradigma funcional está representado por la familia de lenguajes LISP, en particular Scheme.
- El paradigma lógico, un ejemplo es PROLOG.
- El paradigma orientado a objetos. Un lenguaje completamente orientado a objetos es Smalltalk.
- Paradigma reactivo: herramientas y frameworks tales como RxJS (Reactive Extensions for JavaScript), REACT, Akka Streams.
- Paradigma basado en componentes: REACT, Angula, VUE.
- Paradigma Concurrente: Java, Phyton.
- Paradigma Paralelo: Go, CUDA, OpenMP, MPI.

PARADIGMA DECLARATIVO

Es un paradigma de programación que está basado en el desarrollo de programas especificando o "declarando" un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. La solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla.

Diferencia entre imperativo y declarativo

En el paradigma imperativo se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y hallar la solución, es decir, un algoritmo en el que se **describen los pasos necesarios para solucionar el problema**.

En el declarativo las sentencias que se utilizan lo que hacen es **describir el problema** que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información.



PARTE I: INTRODUCCION.

TEMA 2: Lenguajes de Programación






Definición Teórica – Lenguaje de Programación

Un conjunto de sintaxis y reglas semánticas que definen los programas del computador. Es una técnica estándar de comunicación para entregarle instrucciones al computador.

Un lenguaje le da la capacidad al programador de especificarle al computador, qué tipo de datos actúan y que acciones tomar bajo una variada gama de circunstancias, utilizando un lenguaje relativamente próximo al lenguaje humano.



Un programa escrito en un lenguaje de programación necesita pasar por un proceso de compilación, interpretación o intermedio, es decir, ser traducido al lenguaje de máquina para que pueda ser ejecutado por el ordenador.

Clasificación de Lenguajes de Programación

Los lenguajes de programación se pueden clasificar de diferentes maneras dependiendo del criterio utilizado.

Según su Nivel de Abstracción

Lenguajes de Bajo Nivel: Cerca del lenguaje máquina, como el ensamblador. Son más rápidos pero difíciles de comprender.

Lenguajes de Alto Nivel: Más fáciles de entender y escribir, como Python, Java o C#. Se parecen al lenguaje humano y abstraen los detalles del hardware.

Según el Paradigma de Programación.

Según su Uso o Propósito.

De propósito general: C++, Python, Java.

De Dominio específico: SQL, HTML.

Según su forma de ejecución: Compilados, Interpretados, Híbridos.

Según su tipado.


Estáticamente tipado: C++, Java. (Tiempo de compilación)

Dinámicamente tipado: Python, Javascript. (Tiempo de ejecución)




Clasificación de Lenguajes de Programación

Según su forma de ejecución:

- **Lenguajes interpretados** (Interpretes) como Basic, Dbase.
 - **Lenguajes compilados** (Compiladores) como C, C++, Clipper.
 - **Lenguajes interpretados con recolectores de basura** (Maquina Virtual) como Smalltalk, Java, Ocaml.
 - **Lenguajes Scripts** (Motor de ejecución) como Perl, PHP, SQL.
- 

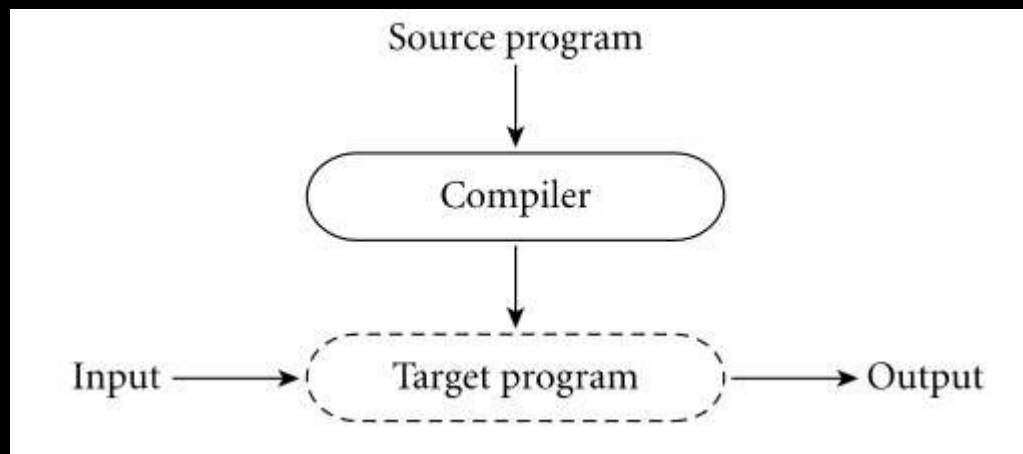


Clasificación de Lenguajes de Programación

- Existe una gran variedad de estrategias para conseguir que un programa se ejecute en un computador.
 - Todas se basan en los "meta-programas" (compiladores, intérpretes, etc.) cuyos datos de entrada son el código fuente de otros programas.
 - Estos meta-programas procesan otros programas y realizan múltiples tareas.
 - Estas estrategias son: Compilación, Interpretación, Ejecución en Máquina Virtual, Enlazado de Rutinas y Librerías y PreProcesamiento.
- 

Clasificación de Lenguajes de Programación

- **Compilación:** La siguiente figura muestra el proceso de generación y ejecución de un programa compilado.

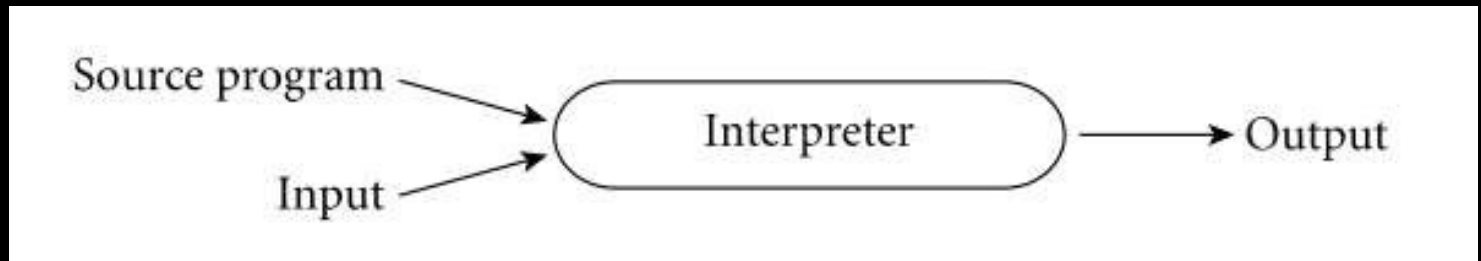


Ejemplos: C, C++.

- Diferentes momentos en la vida de un programa: tiempo de compilación y tiempo de ejecución.
- Mayor eficiencia

Clasificación de Lenguajes de Programación

- **Interpretación:** La siguiente figura muestra el proceso de ejecución de un programa interpretado

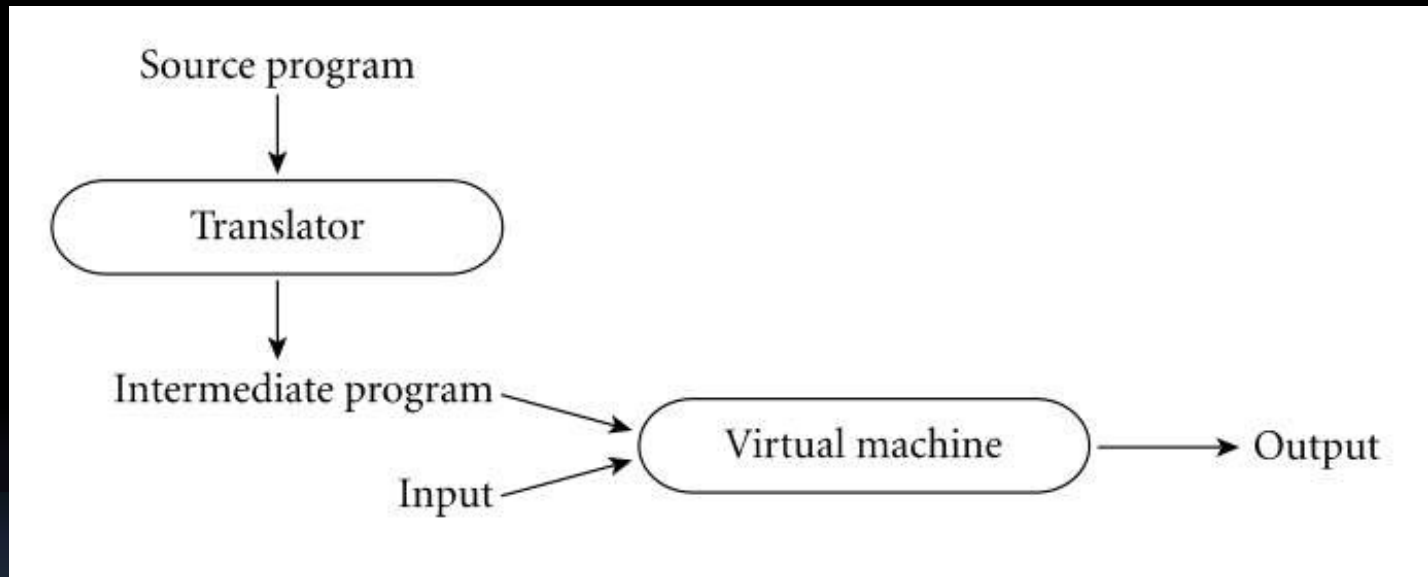


Ejemplos: BASIC, LISP, Scheme, Python, Ruby.

- No hay diferencia entre el tiempo de compilación y el tiempo de ejecución.
- Mayor flexibilidad: el código se puede construir y ejecutar "on the fly" (funciones lambda o clousures).

Clasificación de Lenguajes de Programación

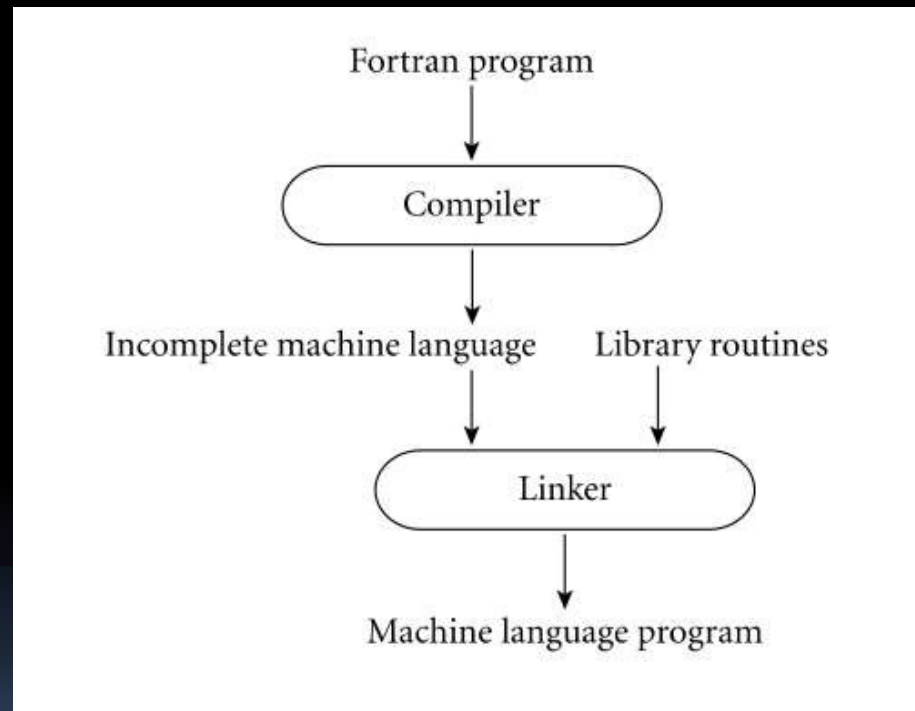
- **Ejecución en Máquina Virtual:** La siguiente figura muestra el proceso de traducción y ejecución de un programa en máquina virtual.



Ejemplos: Java, Scala.

Clasificación de Lenguajes de Programación

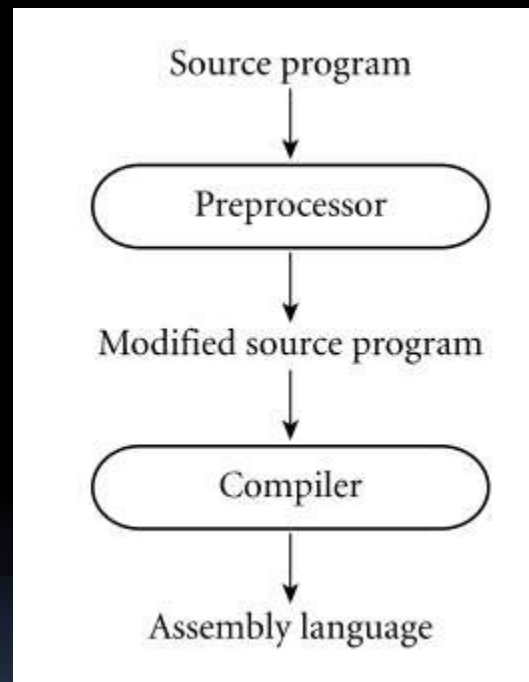
- **Enlazado de Rutinas y Librerías:** La siguiente figura muestra el proceso de compilación y enlazado de un programa de este tipo.



Ejemplos: Java, C++

Clasificación de Lenguajes de Programación

- **Preprocesamiento:** El preprocesador analiza el código y sustituye macros.



Ejemplos: C, C++. Scala hace algo parecido con Java.

Otra Clasificación de Lenguajes de Program.

Tipos de Programación

Programación imperativa

Programación orientada a objetos

Programación dinámica

Programación dirigida por eventos

Programación declarativa

Programación funcional

Programación lógica

Lenguaje específico del dominio

Programación reactiva

Programación multiparadigma

Programación con restricciones



Otra Clasificación de Lenguajes de Program.

Programación imperativa

Es el más usado en general, se basa en dar instrucciones al ordenador de como hacer las cosas en forma de algoritmos. La programación imperativa es la más usada y la más antigua, el ejemplo principal es el lenguaje de máquina. Ejemplos de lenguajes puros de este paradigma serían el C, BASIC o Pascal.

Programación orientada a objetos

Está basada en el imperativo, pero encapsula elementos denominados objetos que incluyen tanto variables como funciones. Está representado por C++, C#, Java o Python entre otros, pero el más representativo sería el Smalltalk que está completamente orientado a objetos.

Programación dinámica

Está definida como el proceso de romper problemas en partes pequeñas para analizarlos y resolverlos de forma lo más cercana al óptimo, busca resolver problemas sin usar métodos recursivos. Este paradigma está más basado en el modo de realizar los algoritmos, por lo que se puede usar con cualquier lenguaje imperativo.

Otra Clasificación de Lenguajes de Program.

Programación dirigida por eventos

Es un paradigma de programación en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema, definidos por el usuario o que ellos mismos provoquen.

Programación declarativa

Está basado en describir el problema declarando propiedades y reglas que deben cumplirse, en lugar de instrucciones. Hay lenguajes para la programación funcional, la programación lógica, o la combinación lógico-funcional. Unos de los primeros lenguajes funcionales fueron Lisp y Prolog.

Programación funcional

Basada en la definición los predicados y es de corte más matemático, está representado por Scheme (una variante de Lisp) o Haskell. Python también representa este paradigma.

Programación lógica

Basado en la definición de relaciones lógicas, está representado por Prolog.

Programación con restricciones

Similar a la lógica usando ecuaciones. Casi todos los lenguajes son variantes del Prolog.

Otra Clasificación de Lenguajes de Program.

Programación multiparadigma

Es el uso de dos o más paradigmas dentro de un programa. El lenguaje Lisp se considera multiparadigma. Al igual que Python, que es orientado a objetos, reflexivo, imperativo y funcional.

Programación reactiva

Este paradigma se basa en la declaración de una serie de objetos emisores de eventos asíncronos y otra serie de objetos que se "suscriben" a los primeros (es decir, quedan a la escucha de la emisión de eventos de estos) y *reaccionan* a los valores que reciben. Es muy común usar la librería Rx de Microsoft (Acrónimo de Reactive Extensions), disponible para múltiples lenguajes de programación.

Lenguaje específico del dominio

Se denomina así a los lenguajes desarrollados para resolver un problema específico, pudiendo entrar dentro de cualquier grupo anterior. El más representativo sería SQL para el manejo de las bases de datos, de tipo declarativo, pero los hay imperativos, como el Logo.

GENERACIONES DE LOS LENGUAJES

Los equipos de ordenador (el hardware) han pasado por cuatro generaciones, de las que las tres primeras (ordenadores con válvulas, transistores y circuitos integrados) están muy claras, la cuarta (circuitos integrados a gran escala) es más discutible.

Algo parecido ha ocurrido con la programación de los ordenadores (el software), que se realiza en lenguajes que suelen clasificarse en cinco generaciones, de las que las tres primeras son evidentes, mientras no todo el mundo está de acuerdo en las otras dos. Estas generaciones no coincidieron exactamente en el tiempo con las de hardware, pero sí de forma aproximada, y son las siguientes:

GENERACIONES DE LOS LENGUAJES

Primera generación: Los primeros ordenadores se programaban directamente en código binario (0s y 1s), que puede representarse mediante secuencias de ceros y unos sistema binario. Cada modelo de ordenador tiene su propio código, por esa razón se llama lenguaje de máquina.

Segunda generación: Los lenguajes simbólicos o de ensamblador, así mismo propios de la máquina, simplifican la escritura de las instrucciones y las hacen más legibles. (add, move).

Tercera generación: Los lenguajes de alto nivel sustituyen las instrucciones simbólicas por códigos independientes de la máquina, parecidas al lenguaje humano o al de las Matemáticas. (C, Java, Python).

GENERACIONES DE LOS LENGUAJES

Cuarta generación: se ha dado este nombre a ciertas herramientas que permiten construir aplicaciones sencillas combinando piezas prefabricadas. Hoy se piensa que estas herramientas no son, propiamente hablando, lenguajes. Algunos proponen reservar el nombre de cuarta generación para la programación orientada a objetos. (SQL, MATLAB, ABAP, R, SAS).

Quinta generación: Se usan para resolver problemas mediante Programación Lógica o Sistemas Expertos, delegando mas trabajo al software. Se enfoca en la resolución automática de problemas complejos. Se asocia con conceptos de Inteligencia Artificial. Prolog, OPS5, Mercury, LISP.

ESTILOS DE PROGRAMACIÓN

Los **ESTILOS**, son las convenciones utilizadas para escribir código fuente en los distintos lenguajes. También llamados estándares o guías. Ej. Nomenclatura, indentación, comentarios, estructuras.

La **Programación Imperativa**, es un paradigma de programación que describe la programación en términos del estado del programa y sentencias que cambian dicho estado. Los programas imperativos son un conjunto de instrucciones que le indican al computador cómo realizar una tarea.

La implementación de hardware de la mayoría de computadores es imperativa; prácticamente todo el hardware de los computadores está diseñado para ejecutar código de máquina, que es nativo al computador, escrito en una forma imperativa.

Los primeros lenguajes imperativos fueron los lenguajes de máquina de los computadores originales.

Ej: Lenguaje C, Pascal.

ESTILOS DE PROGRAMACIÓN - Imperativa

Características: Definición de procedimientos, Definición de tipos de datos, Chequeo de tipos en tiempo de compilación, Cambio de estado de variables, Pasos de ejecución de un proceso.

```
type
  tDimension = 1..100;
  eMatriz(f,c: tDimension) = array [1..f,1..c] of real;

  tRango = record
    f,c: tDimension value 1;
  end;

  tpMatriz = ^eMatriz;

procedure EscribirMatriz(var m: tpMatriz);
var filas,col : integer;
begin
  for filas := 1 to m^.f do begin
    for col := 1 to m^.c do
      write(m^[filas,col]:7:2);
      writeln(resultado);
      writeln(resultado)
    end;
  end;
end;
```

ESTILOS DE PROGRAMACIÓN - Estructurada

La **programación estructurada** es una forma de escribir programación de ordenador de forma clara, para ello utiliza únicamente tres estructuras: secuencial, selectiva e iterativa; siendo innecesario y no permitiéndose el uso de la instrucción o instrucciones de transferencia incondicional (GOTO).

Surge del Teorema de Dijkstra, demostrado por Edsger Dijkstra en los años sesenta, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control siguientes:

- Secuencia
- Instrucción condicional.
- Iteración, o bucle de instrucciones.

ESTILOS DE PROGRAMACIÓN - Estructurada

Ejemplo de **programación estructurada**.

```
/* ****  
 * Programa que usa estructura selectiva simple  
 **** */  
//librerías  
#include <stdio.h>  
#define TAM 8 //constante simbólica  
  
//definición de tipos de datos abstractos  
//prototipos de función  
//declaración de variables globales, aunque no recomendado.  
  
int main() { //función principal  
    int num; //declaración de variable global  
    system("cls");  
    printf("Programa que determina si un numero es positivo:");  
    printf("\nTeclea un numero:");  
    scanf("%d", &num);  
  
    if (num > 0)  
        printf("El numero es positivo\n\n");  
  
    system("pause");  
    return 0;  
}
```

ESTILOS DE PROGRAMACIÓN - Funcional

La **Programación Funcional** es un paradigma de programación declarativa basado en la utilización de funciones matemáticas.

Sus orígenes provienen del Cálculo Lambda (o λ -cálculo), una teoría matemática elaborada por Alonzo Church como apoyo a sus estudios sobre computabilidad.

Existen dos grandes categorías de lenguajes funcionales: los funcionales *puros* y los *híbridos*. La diferencia entre ambos estriba en que los lenguajes funcionales híbridos son menos dogmáticos que los puros, al admitir conceptos tomados de los lenguajes imperativos, como las secuencias de instrucciones o la asignación de variables.

Entre los lenguajes funcionales puros, cabe destacar a Haskell y Miranda. Los lenguajes funcionales híbridos más conocidos son Lisp, Scheme, Ocaml y Standard ML (estos dos últimos, descendientes del lenguaje ML).

ESTILOS DE PROGRAMACIÓN - Funcional

Ejemplo de Código LISP para el Cálculo de una Potencia (utilizando recursividad).

```
(defun potencia-aux (b e)
  (if (= e 0) 1
      (* b (potencia-aux b (- e 1)))))
```

```
(potencia-aux '3 '4)
81
```

ESTILOS DE PROGRAMACIÓN - Lógica

La **Programación Lógica** consiste en la aplicación del *corpus* de conocimiento sobre lógica para el diseño de lenguajes de programación; no debe confundirse con la disciplina de la lógica computacional. Gira en torno al concepto de **predicado**, o relación entre elementos.

La programación lógica encuentra su *hábitat* natural en aplicaciones de inteligencia artificial o relacionadas:

- **Sistemas expertos**, donde un sistema de información imita las recomendaciones de un experto sobre algún dominio de conocimiento.
- **Demostración automática de teoremas**, donde un programa genera nuevos teoremas sobre una teoría existente.
- **Reconocimiento de lenguaje natural**, donde un programa es capaz de comprender (con limitaciones) la información contenida en una expresión lingüística humana.

ESTILOS DE PROGRAMACIÓN - Lógica

El lenguaje de programación lógica por excelencia es **Prolog**. Los programas construidos un lenguaje lógico están contruidos únicamente por expresiones lógicas, es decir, que son ciertas o falsas, en oposición a un expresión interrogativa (una pregunta) o expresiones imperativas (una orden).

Prolog, proveniente del inglés **Programming in Logic**, es un lenguaje lógico bastante popular en el medio de investigación en Inteligencia Artificial.

Las instrucciones de Prolog se llaman **"reglas o cláusulas de Horn"**. En Prolog el orden de ejecución de las instrucciones no tiene nada que ver con el orden en que fueron escritas. Tampoco hay instrucciones de control propiamente dichas. Para trabajar con este lenguaje, un programador debe acostumbrarse a pensar de una manera muy diferente a la que se utiliza en los lenguajes clásicos.

ESTILOS DE PROGRAMACIÓN - Lógica

%% declaraciones

padrede('juan', 'maria'). % juan es padre de maria

padrede('pablo', 'juan'). % pablo es padre de juan

padrede('pablo', 'marcela').

padrede('carlos', 'debora').

% A es hijo de B si B es padre de A

hijode(A,B) :- padrede(B,A).

% A es abuelo de B si A es padre de C y C es padre B

abuelode(A,B) :- padrede(A,C), padrede(C, B).

% A y B son hermanos si el padre de A es tambien el padre de B y si
A y B no son lo mismo

hermanode(A,B) :- padrede(C,A) , padrede(C,B), A <> B.

ESTILOS DE PROGRAMACIÓN - Lógica

%% CONSULTAS

% juan es hermano de marcela?

?- **hermanode('juan', 'marcela').**

si

% carlos es hermano de juan?

?- **hermanode('carlos', 'juan').**

no

% pablo es abuelo de maria?

?- **abuelode('pablo', 'maria').**

si

% maria es abuelo de pablo?

?- **abuelode('maria', 'pablo').**

no

ESTILOS DE PROGRAMACIÓN - Objetos

La **Programación Orientada a Objetos** (POO u OOP según siglas en inglés) es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan **estado** (es decir, datos), **comportamiento** (esto es, 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.

ESTILOS DE PROGRAMACIÓN - Objetos

De esta forma, un objeto contiene toda la información, (los denominados atributos) 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).

ESTILOS DE PROGRAMACIÓN - Objetos

Las principales diferencias entre la programación estructurada y la orientada a objetos son:

- La programación orientada a objetos es más moderna.
- En la programación estructurada se escriben funciones y después les pasan datos. En la orientada a objetos se definen objetos con datos y métodos y después se envían mensajes a los objetos diciendo que realicen esos métodos en sí mismos.
- La programación orientada a objetos se basa en lenguajes que soportan sintáctica y semánticamente la unión entre los tipos abstractos de datos y sus operaciones (a esta unión se la suele llamar clase).
- La programación orientada a objetos incorpora en su entorno de ejecución mecanismos tales como el polimorfismo y el envío de mensajes entre objetos.

ESTILOS DE PROGRAMACIÓN - Objetos

Lenguajes orientados a objetos

Entre los lenguajes orientados a objetos destacan los siguientes:

Ada , C++ , C# , VB.NET , Visual FoxPro , Clarion , Delphi , Harbour , Eiffel , Java , Lexico (en castellano) , Objective-C , Ocaml , Oz , Perl (soporta herencia múltiple) , PHP (en su versión 5) , PowerBuilder , Python , Ruby , Smalltalk , Magik (SmallWorld) .

ESTILOS DE PROGRAMACIÓN - Objetos

```
public Bicicleta(int velocidadInicial, int marchaInicial) {
    marcha = marchaInicial;
    velocidad = velocidadInicial;
}

public void setMarcha(int nuevoValor) {
    marcha = nuevoValor;
}

public void frenar(int decremento) {
    velocidad -= decremento;
}

public void acelerar(int incremento) {
    velocidad += incremento;
}
}

public class MountainBike extends Bicicleta {
    public int alturaSillin;

    public MountainBike(int alturaInicial, int velocidadInicial, int marchaInicial) {
        super(velocidadInicial, marchaInicial);
        alturaSillin = alturaInicial;
    }

    public void setAltura(int nuevoValor) {
        alturaSillin = nuevoValor;
    }
}

public class Excursion {

    public static void main(String[] args) {
        MountainBike miBicicleta = new MountainBike(10,10,3);
        miBicicleta.acelerar(10);
        miBicicleta.setMarcha(4);
        miBicicleta.frenar(10);
    }
}
```

ESTILOS DE PROGRAMACIÓN - Aspectos

La **programación orientada a aspectos (POA)** es un paradigma de programación que basa su filosofía en tratar las **obligaciones transversales** de nuestros programas como módulos separados (aspectos) para lograr una correcta separación de responsabilidades.

Una obligación transversal es aquella que **se repite en varias partes de un programa** independientemente de si las secciones en las que aparece tienen relación directa; imagina un método que actualice lo que se muestra en la pantalla de un programa de dibujo, este puede ser llamado desde los métodos encargados de dibujar algo en pantalla, hasta métodos encargados de recortar, redimensionar, guardar, exportar, deshacer, etc.

Aunque es un paradigma relativamente moderno, tuvo muchas influencias que le ayudaron a marcar unas bases muy sólidas, entre las que cabe resaltar la reflexión computacional, programación adaptativa y la programación orientada a objetos (POO).

ESTILOS DE PROGRAMACIÓN - Aspectos

Con la programación orientada a objetos (**POO**) cambiamos nuestra forma de pensar para abstraer problemas de una forma más simbólica para el humano, ahora los objetos eran los que manipulaban los datos, y las clases son las que nos ofrecen sus funcionalidades. **Con la POA** tomamos un enfoque parecido al de la **POO** pero con la diferencia que la **POA** nos dice que si alguna funcionalidad de nuestro código se repite en diferentes módulos, lo mejor sería extraerla del programa principal y hacer de ella un aspecto en lugar de crear jerarquías complicadas.

El fuerte de la **POA** se obtiene al combinarla con otros paradigmas de programación, así compensamos las deficiencias de uno con las fortalezas de otro. Entre todos los paradigmas, la **POO** es el más popular con el que se suele combinar, es por eso que puedes encontrar fácilmente implementaciones de **POA** para C++, Java y Perl.

ESTILOS DE PROGRAMACIÓN

```
@Component
@Aspect
public class MiAspect {

    @Before("execution( * Probatina.*())")
    public void interceptaClaseProbatinaSinParametros(JoinPoint jp) {
        System.out.println("ANTES de " + jp.toString());
    }

    @After("execution(public * get*())")
    public void interceptaGetters(JoinPoint jp) {
        System.out.println("DESPUES de " + jp.toString());
    }
}
```

Ventajas:

- Provee una fuerte herramienta para modularizar programas sin importar lo extensos y complicados que estos sean.
- Vuelve más limpio el código fuente.
- Permite agilizar el proceso de creación de programas cuando muchas personas están involucradas en el mismo proyecto, y/o están en lugares geográficos diferentes.

ESTILOS DE PROGRAMACIÓN - Aspectos

Ventajas:

- Puede mezclarse con cualquier otro paradigma de programación.
- Permite la comunicación entre diferentes lenguajes de programación que comparten aspectos.

Desventajas:

- Sufre de un antipatrón de diseño: acciones a distancia.
- Vuelve difícil de comprender el código puesto que el programa hace tareas que no están en los métodos que deberían estar.
- Es un poco complicado identificar cuándo es óptimo utilizar POA de forma eficiente.

Ejemplos: [AspectC++](#) para C++; [AspectJ](#) extensión de Eclipse para Java; [Aspect](#) módulo para Perl; [Spring](#) para Java; [aspectlib](#) para Python; [AspectGo](#) framework en desarrollo para Golang.

ESTILOS DE PROGRAMACIÓN - Eventos

- La **programación dirigida por eventos** es un paradigma en el que tanto la estructura como la ejecución de los programas van determinados por los sucesos que ocurran en el sistema o que ellos mismos provoquen.
- El creador de este tipo de programas define los eventos que manejará y las acciones ante cada uno de ellos, conocido como el manejador de eventos. Los eventos soportados estarán determinados por el lenguaje de programación, por el S.O. y por eventos propios.
- Un ejemplo son los Programas del tipo VISUAL (Visual C, Visual Basic, etc., en el que a cada elemento del programa (objetos, controles, etc.) se le asignan una serie de eventos que generará dicho elemento, como la pulsación de un botón del ratón sobre él o el redibujado del control.
- La programación dirigida por eventos es la base de lo que llamamos interfaz de usuario, aunque puede emplearse también para desarrollar interfaces entre componentes de Software o módulos del núcleo.

ESTILOS DE PROGRAMACIÓN - Eventos

En este tipo, al comenzar la ejecución del programa se llevarán a cabo las inicializaciones y a continuación el programa quedará bloqueado hasta que se produzca algún evento.

Cuando alguno de los eventos esperados por el programa tenga lugar, el programa pasará a ejecutar el código del correspondiente administrador de evento. Por ej., si el evento es hacer click en el botón de play de un reproductor de películas, se ejecutará el código del administrador de evento, que será el que haga que la película se muestre por pantalla.

Ejemplo de programa orientado a eventos en pseudo lenguaje:

```
While (true){  
    Switch (event){  
        case mouse_button_down:  
        case mouse_click:  
        case keypressed:  
        case Else:  
    }  
}
```

ESTILOS DE PROGRAMACIÓN - Restricciones

La **Programación con restricciones** es un paradigma de programación, donde las relaciones entre las variables son expresadas en términos de restricciones (ecuaciones).

Actualmente es usada como una tecnología de software para la descripción y resolución de problemas combinatorios, especialmente en las áreas de planificación y programación de tareas (calendarización).

Se trata de un paradigma de programación basado en la especificación de un conjunto de restricciones, las cuales deben ser satisfechas por cualquier solución del problema planteado, en lugar de especificar los pasos para obtener dicha solución. Se relaciona mucho con la programación lógica y con la investigación operativa.

ESTILOS DE PROGRAMACIÓN - Restricciones

Pertenece al paradigma de programación declarativa, es decir, en lugar de indicar cómo queremos que se hagan las cosas, nos limitamos a decir qué queremos que pase y el sistema/librería/lenguaje se encargará de hacerlo.

Se basa en crear un modelo formado por variables, para las cuales deberemos indicar qué posibles valores pueden tomar, y restricciones, que expresan relaciones que deben cumplirse entre esas variables. Una vez creado el modelo, el sistema se encarga de encontrar aquellas soluciones que se ajustan a las restricciones.

Un ejemplo sencillo sería el modelo formado por las siguientes restricciones:

ESTILOS DE PROGRAMACIÓN - Restricciones

x es un valor entre 1 y 10, y es un valor entre 1 y 10 , $x < y$, $x + y = 4$

Partiendo de estas restricciones, el sistema sería capaz de encontrar todas las posibles soluciones al problema (en este caso, sólo hay una):

$$x = 1 \text{ y } y = 3$$

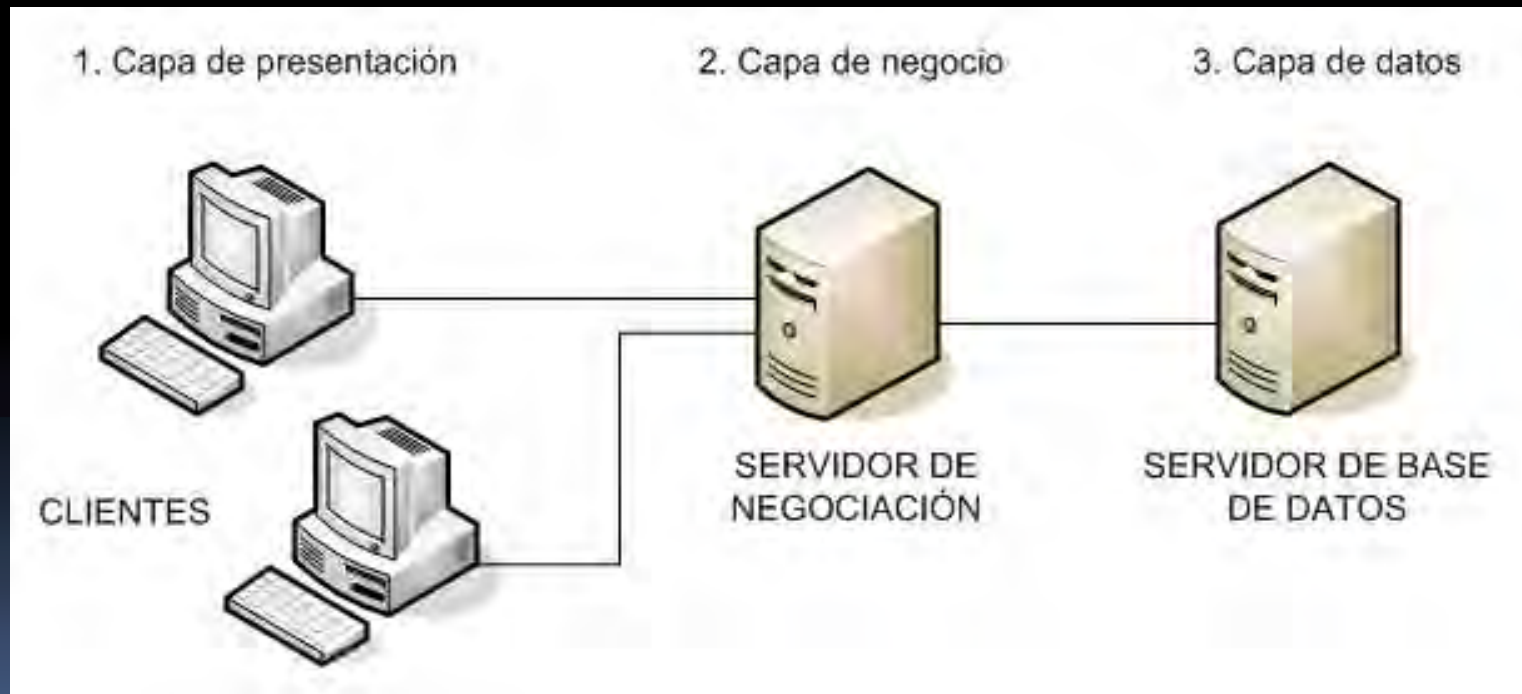
En CLOJURE (leng. de programación de propósito general dialecto Lisp), sería:

```
1 | (def model
2 |   [($in :x 1 10)
3 |    ($in :y 1 10)
4 |    ($ < :x :y)
5 |    ($= ($+ :x :y ) 4)])
```

```
1 | (solution model)
2 | ;; => {:x 1 :y 3}
```

ESTILOS DE PROGRAMACIÓN - Capas

La **programación por capas** es un estilo de programación en la que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño, un ejemplo básico de esto es separar la capa de datos de la capa de presentación al usuario.



ESTILOS DE PROGRAMACIÓN - Capas

La ventaja principal de este estilo, es que el desarrollo se puede llevar a cabo en varios niveles y en caso de algún cambio sólo se ataca al nivel requerido sin tener que revisar entre código mezclado. Un buen ejemplo de este método de programación seria:

Modelo de interconexión de sistemas abiertos

Además permite distribuir el trabajo de creación de una aplicación por niveles, de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, simplemente es necesario conocer la API que existe entre niveles.

CONCLUSIONES

La elección de un lenguaje de programación depende completamente del tipo de problema que desees resolver, las necesidades específicas y las características de cada lenguaje. Esta es una pequeña guía para decidir qué lenguaje utilizar según las situaciones más comunes:

CONCLUSIONES

| Situación | Lenguaje Recomendado |
|--|--|
| Desarrollo Web | - Frontend: HTML, CSS, JavaScript (React, Angular, Vue). |
| | - Backend: Node.js (JavaScript), Python (Django, Flask), PHP, Ruby (Rails). |
| Desarrollo de Aplicaciones de Escritorio | - Multiplataforma: Java, C#, Python (PyQt, Tkinter). |
| | - Específico del sistema operativo: Swift (iOS/macOS), C++ o C# (.NET). |
| Desarrollo de Aplicaciones Móviles | - Nativo: Swift (iOS), Kotlin/Java (Android). |
| | - Multiplataforma: Flutter (Dart), React Native (JavaScript). |
| Inteligencia Artificial y Ciencia de Datos | - Lenguaje clave: Python (TensorFlow, PyTorch, pandas), R. |
| | - Alto rendimiento: C++. |
| Desarrollo de Videojuegos | - Motores de juegos: C# (Unity), C++ (Unreal Engine). |
| | - Prototipado: Python. |
| Software Empresarial o Grandes Sistemas | - Java, C#, Python. |
| | - Bases de datos: SQL. |
| Programación de Sistemas o Hardware | - Bajo nivel: C, C++, Rust. |
| | - Sistemas embebidos: C, Assembly. |
| Scripts y Automatización | - Python, Bash, PowerShell. |
| Aplicaciones Científicas o Matemáticas | - MATLAB, Python, Julia, R. |