

# Paradigmas de Programación

# Definición de Paradigma

Definición: Desde Thomas Kuhn hasta Mario Bunge.  
Particularidad de la ciencia Informática: Mutabilidad.  
Libertad en los lenguajes.  
A cada problema una solución distinta en diversos  
lenguajes de diferentes paradigmas.  
Significado en Informática:  
    Marco Teórico.  
    Esquema formal de organización.

# Paradigma de Programación

Propuesta metodológica para resolver problemas.

Es un enfoque para diseñar soluciones.

Lenguajes específicos - Lenguajes versátiles.

Ventajas y desventajas. Eficiencia en cada problema.

No existe el paradigma perfecto.

Buscar eficacia y elegancia en la solución.

Programación: ¿Arte o ciencia?

Kunth VS. Dijkstra. Concepto de complejidad.

Ejemplos: 1-Música y 2-Breaking Bad

# Teoría de la Programación

Principios que rigen un paradigma.

Características semánticas que definen lenguajes.

Busca las soluciones más adecuadas de enfrentarse a un problema para poder codificarlo y ejecutarlo.

Babbage: Máquina analítica. 5 módulos:

entrada, cálculo, control, memoria y salida.

ENIAC: 1ª computadora digital. Programación cableada

John von Neumann: Sin cables. Coexisten datos e instrucciones en la memoria. Giro copernicano.

# Paradigma Declarativo

Gran división: Paradigmas Imperativos y Declarativos

Se describe el problema y se detalla la solución, que es obtenida mediante mecanismos internos.

No se dan instrucciones para llegar a soluciones.

Minimiza los efectos secundarios imprevistos.

Se le dice a la máquina *qué* es lo que tiene que hacer y no *cómo* hacerlo.

Los programas son teorías de la lógica formal.

Ejemplos BD consultivas (SQL), programación con restricciones, Programas DSL.

# Paradigma Lógico

Implementación de programas usando lógica formal.

Una serie de sentencias que expresan reglas y hechos acerca de un problema. Hechos: H.

Reglas: (cabeza :- cuerpo)  $H_1 :- B_1, \dots, B_n$ .

Los problemas individuales se exponen como teoremas que han de ser probados.

Prolog: Lenguaje para programar artefactos electrónicos y muy usado en Inteligencia Artificial.

Modus Ponens - Si es verdad el antecedente, entonces es verdad el consecuente.

# Paradigma Funcional

Se basa en el concepto de funciones aritméticas.

El valor generado por una función depende de los argumentos alimentados a la función.

Las computaciones van de la reescritura de definiciones más amplias a otras más concretas.

Usos: ambientes académicos, aplicaciones comerciales e industriales y lenguajes de dominio específico.

Haskell: Los programas se ven como transformaciones de datos.

Es puramente funcional con semánticas no estrictas y fuertemente tipificado.

# Paradigma de programación imperativa

- **Definición:** Describe la programación como una secuencia de sentencias o órdenes que cambian el estado del programa. Conjunto de instrucciones que indican al computador cómo realizar o encontrar la solución de una tarea. Más adelante utilizará subrutinas.
- **Historia:** Años 50 , basado en la arquitectura Von Neumann. Constituye la primera forma de programar ordenadores (ensamblador ,máquina).Años 60 se introduce programación procedural y a finales Dijkstra propone la eliminación de la instrucción GOTO, que marcará el inicio de la programación estructurada.



# Características del paradigma imperativo

## • Elementos:

- Tipos de datos: Caracteres,numéricos,booleanos.
- Variables: Se encargan de almacenar o dar referencia al estado del programa
- Expresiones: En la programación imperativa su papel más importante está en las sentencias de asignación.

## • Explicación:

- Operaciones de asignación:Cada valor calculado es asignado o almacenado en una celda de memoria, de manera de actualizar el valor grabado.
- Repetición: El programa efectúa su tarea iterando o repitiendo secuencias de pasos elementales.

## • Flujo de datos:

- Secuencial:El programa se ejecuta secuencialmente.
- Selección condicional:Según la condición la ejecución se bifurca.
- Selección incondicional:La ejecución se bifurca imperativamente.
- Sentencias de iteración:La ejecución se repite hasta que se cumple la condición.

## • Límites del paradigma:

- Efecto lateral:Producidos por la modificación continua de las celdas de memoria durante el programa, provocan que el programa sea inseguro y menos predecible.

# Tipos de lenguaje imperativos

## Ensamblador(1)

## Fortran(2)

-Introduce las subrutinas.

```
-----  
; Programa que imprime un string en la pantalla  
-----  
.model small          ; modelo de memoria  
  
.stack                ; segmento del stack  
  
.data                 ; segmento de datos  
Cadenal DB 'Hola Mundo.$' ; string a imprimir (finalizado en $)  
  
.code                 ; segmento del código  
  
-----  
; Inicio del programa  
-----  
programa:  
; -----  
; inicia el segmento de datos  
; -----  
MOV AX, @data          ; carga en AX la dirección del segmento de datos  
MOV DS, AX             ; mueve la dirección al registro de segmento por medio de AX  
  
; -----  
; Imprime un string en pantalla  
; -----  
MOV DX, offset Cadenal ; mueve a DX la dirección del string a imprimir  
MOV AH, 9              ; AH = código para indicar al MS DOS que imprima en la pantalla, el string en DS:DX  
INT 21h                ; llamada al MS DOS para ejecutar la función (en este caso especificada en AH)  
  
; -----  
; Finaliza el programa  
; -----  
INT 20h                ; llamada al MS DOS para finalizar el programa  
  
end programa
```

(1)

```
REGRESION LINEAL.FORTRANS  
APLICACION  
    DIMENSION TIEMPO(1000),PROD(1000)  
    OPEN(1,FILE='HISTORIA.txt')  
    I=0  
10  READ(1,*,END=80)T,P  
    I=I+1  
    TIEMPO(I)=T  
    PROD(I)=P  
    GO TO 10  
80  NDATOS=I  
    CALL AJULIN(TIEMPO,PROD,NDATOS,A,B)  
    WRITE(*,90)A,B  
90  FORMAT('LA ECUACION ES:Y=',F10.2,'+',F10.2,'X')  
20  FORMAT(20F10.0)  
    END  
  
SUBROUTINE AJULIN(X,Y,N,A,B)  
    DIMENSION X(1),Y(1)  
    SUMX=0.  
    SUMY=0.  
    SUMX2=0.  
    SUMY2=0  
    SUMXY=0  
    DO 20 I=1,N  
        SUMX=SUMX+X(I)  
        SUMY=SUMY+Y(I)  
        SUMX2=SUMX2+(X(I)*X(I))  
        SUMY2=SUMY2+Y(I)**2  
        SUMXY=SUMXY+(X(I)*Y(I))  
20  CONTINUE  
    PROD=SUMX*SUMY  
    B=(SUMXY-PROD/N)/(SUMX2-SUMX**2/N)  
    A=(SUMY/N-B*SUMX/N)  
    RETURN  
    END
```

(2)

# Ventajas y desventajas

## •Ventajas:

- Se adapta bien para trabajar con proyectos grandes y en grupo.
- Excelente cuando se necesitan resolver muchos problemas similares.
- Control de datos de manera secuencial.

## •Desventajas:

- Falta de flexibilidad debido a la secuencialidad de las instrucciones.
- Y como hemos comentado antes, este paradigma tiene el problema del efecto lateral

# Paradigma de programación dinámica

- **Introducción:** Se refiere a la simplificación de un problema complicado dividiéndolo en subproblemas más simples de una manera recursiva. Si estos subproblemas tienen soluciones óptimas se dice que el problema tiene subestructura óptima.
- **Historia:** En 1953 Richard Bellman creó el significado de este tipo de programación. Este consiste específicamente en encontrar pequeños problemas de decisión dentro de grandes decisiones.

# Características

- Hace uso de:

- Subproblemas.

- Subestructura óptima

1. Dividir el problema en subproblemas

2. Resolver estos subproblemas de manera óptima

3. Usar estas soluciones óptimas para construir una solución óptima al problema original.

- Memoización: Acelerar los programas de ordenador mediante el almacenamiento en la memoria caché

- Enfoques:

- Top-down: Se resuelven recordando las soluciones de los subproblemas.

- Bottom-up: Se resuelven solo los problemas necesarios de antemano, que luego se usan para resolver problemas mayores.

- Conclusión.

# Diferencias en paradigma imperativo y declarativo

- **Imperativo:** Describe las instrucciones para variar el estado de un programa para así alcanzar la solución de un problema.Efectos secundarios.
- **Declarativo:** Describe la solución de un problema, no las instrucciones. Estas últimas se realizarán a través de mecanismos internos de inferencia de información a partir de la descripción realizada.Transparencia referencial.

# Paradigma de programación estructurada

Definición: La **programación estructurada** es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa, utilizando subrutinas y tres estructuras: **secuencia**, **selección (if y switch)** e **iteración (bucles for y while)**, considerando innecesario el uso de la instrucción de transferencia (GOTO), que podría llevar a "código espagueti", que es mucho más difícil de seguir y de mantener, y era la causa de muchos errores de programación.

# El código espagueti

Definición:El **código espagueti** es un término despectivo para referirse a los programas de computación que tienen una estructura de control de flujo compleja e incomprensible.

Suele asociarse este estilo de programación con lenguajes básicos y antiguos, donde el flujo se controlaba mediante sentencias de control muy primitivas como goto y utilizando números de línea. Un ejemplo de este lenguaje era el QBasic de Microsoft en sus primeras versiones.)



# Orígenes de la programación estructurada

A finales de los años 1970 surgió una nueva forma de programar dando lugar a programas más fiables y eficientes, escritos de manera que facilitaba su mejor comprensión durante la fase de desarrollo y posibilitando una más sencilla modificación posterior.

El teorema del programa estructurado, propuesto por Böhm-Jacopini, demuestra que todo programa puede escribirse utilizando únicamente las tres instrucciones de control.

# El teorema del programa estructurado

Es un resultado en la teoría de lenguajes de programación. Establece que toda función computable puede ser implementada en un lenguaje de programación que combine sólo tres estructuras lógicas. Esas tres formas también llamadas estructuras de control específicamente son:

- **Secuencia:** ejecución de una instrucción tras otra.
- **Selección:** ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.
- **Iteración:** ejecución de una instrucción (o conjunto) mientras una variable booleana sea 'verdadera'. Esta estructura lógica también se conoce como ciclo o bucle.

Este teorema demuestra que la instrucción GOTO no es estrictamente necesaria y que para todo programa que la utilice existe otro equivalente que no hace uso de dicha instrucción.

# Ejemplos códigos estructuras de control

- Selección:

```
IF (Condición) THEN
    (Bloque de sentencias 1)
ELSE
    (Bloque de sentencias 2)
END IF
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
SELECT (Expresión)
    CASE Valor1
        (Bloque de sentencias 1)
    CASE Valor2
        (Bloque de sentencias 2)
    CASE Valor n
        (Bloque de sentencias n)
    CASE ELSE
        (Bloque de sentencias "Else")
END SELECT
```

# Ejemplos códigos estructuras de control

- Iteración

```
DO WHILE (Condición)
    (Bloque de sentencias)
LOOP
////////////////////////////////////
WHILE (Condición)
    (Bloque de sentencias)
WEND
////////////////////////////////////
DO
    (Bloque de sentencias)
LOOP UNTIL (Condición)
////////////////////////////////////
FOR (Variable) = (Expresión1) TO (Expresión2) STEP (Salto)
    (Bloque de sentencias)
NEXT
```

# Ventajas de la programación estructurada

Ventajas de la programación estructurada:

- Los programas son más fáciles de entender, pueden ser leídos de forma secuencial y no hay necesidad de hacer seguimientos en saltos de líneas (GOTO) dentro de los bloques de código para intentar entender la lógica.
- La estructura de los programas es clara.
- Reducción del esfuerzo en las pruebas y depuración. El seguimiento de los fallos o errores del programa ("debugging") se facilita debido a su estructura más sencilla y comprensible, por lo que los errores se pueden detectar y corregir más fácilmente.
- Reducción de los costos de mantenimiento. A la hora modificar o extender los programas resulta más fácil.
- Los programas son más sencillos y más rápidos de confeccionar.
- Se incrementa el rendimiento de los programadores, comparado con la forma anterior que utiliza GOTO.

# Programación estructurada de bajo nivel

En un bajo nivel, los programas estructurados con frecuencia están compuestos de simples estructuras de flujo de programa jerárquicas. Estas son secuencia, selección y repetición:

- "Secuencia" se refiere a una ejecución ordenada de instrucciones.
- En "selección", una de una serie de sentencias es ejecutada dependiendo del estado del programa. Esto es usualmente expresado con palabras clave como `if..then..else..endif`, `switch`, o `case`.

En la "repetición" se ejecuta una sentencia hasta que el programa alcance un estado determinado, o las operaciones han sido aplicadas a cada elemento de una colección. Esto es usualmente expresado con palabras clave como `while`, `repeat`, `for` o `do..until`.

# Lenguajes de programación estructurada

Es posible hacer la programación estructurada en cualquier lenguaje de programación, aunque es preferible usar algo como un lenguaje de programación procedimental. Algunos de los lenguajes utilizados inicialmente para programación estructurada incluyen: ALGOL, Pascal, PL/I y Ada.

## Otros lenguajes:

- ASP
- BASIC
- C
- C#
- Fortran
- Java
- Perl
- PHP
- Lua

# Ejemplo código C, calcular el área de un triángulo:

```
#include <stdio.h>
#include <stdlib.h>

float area(float, float);

int main(){
    float base=0, altura=0;

    printf ("Introdueix base: ");
    scanf (" %f", &base);
    printf ("Introdueix altura: ");
    scanf (" %f", &altura);

    if (base<0||altura<0){
        printf ("\n-1\n");
        return 0;
    }
    else
        printf ("El resultat es: %.2f cm2", area(altura, base));
    }
    float area(float altura, float base)
    {
        float resultat;
        resultat=base*altura/2;
    } return resultat;
```



# Paradigmas de la P00 y el modular

La P00 es un paradigma de programación que representa el concepto de objetos y los usa en sus interacciones.

Los objetos que son representados por los paradigmas de programación tienen campos de datos que son atributos que describen el objeto y tienen procedimientos llamados métodos.

Para diseñar aplicaciones o programas los objetos son utilizados como instancias de clases e interaccionan con otros objetos para empezar el diseño

Un objeto puede tener un estado (datos) y un comportamiento (codigo)

La orientación de objetos utiliza encapsulación y ocultación de información.

La programación orientada a objetos es una serie de normas sobre como realizar las cosas de manera que otra gente pueda reutilizarlas.

# Orígenes de la P00

La aparición de términos como “objetos” y “orientado” en el ámbito moderno de la P00 hizo su primera aparición en el MIT a finales de los 50 y principios de los 60. En el entorno del grupo de inteligencia artificial, a principios de los 60, “objeto” podría referirse a objetos de identidad (en lenguaje de programación LISP) con propiedades (atributos). Alan Kay fue el último en dar a entender una comprensión detallada del funcionamiento interno de LISP en 1966. Otro ejemplo fue el Sketchpad creado por Ivan Sutherland en 1960-1961; en el glosario del informe técnico del sketchpad en 1963, Sutherland mencionó y definió palabras como “objeto” e “instancia” aunque se refieren al apartado de interacción gráfica.

# Características de la P00

Estructura de un objeto.

Un objeto puede dividirse en tres partes:

1-Relaciones

2-Propiedades

3-Métodos

Cada uno de estos componentes es independiente del otro. Los objetos tienen jerarquías, unas simples, otras complejas. De todos modos, sea cual sea su jerarquía, siempre habrá 3 niveles de objetos:

La raíz de la jerarquía

Los objetos intermedios

Los objetos terminales

# Características secundarias de la P00

Hay 3 conceptos fundamentales en la P00:

- 1- Tipificación
- 2- Concurrencia
- 3- Persistencia

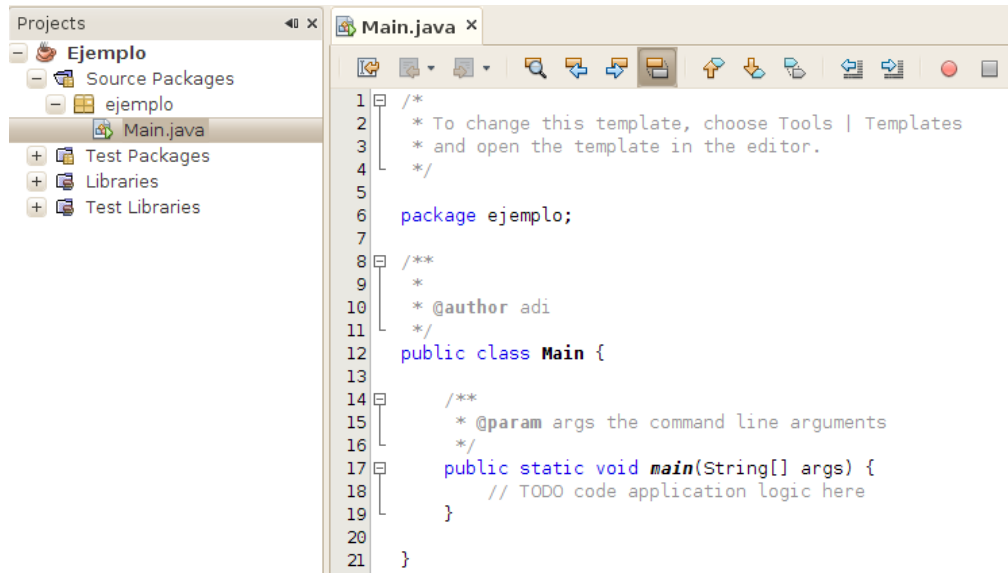
También hay que tener en cuenta los pros/contras de la utilización de la P00:

Pros: Reutilización, Mantenimiento, Modificación y Fiabilidad

Contras: Cambio en la forma de pensar, ejecución lenta de programas, necesidad de utilizar bibliotecas, amplios conocimientos de la teoría de objetos y mayor capacidad de los programadores

# Ejemplos de lenguajes de P00

Java:



```
1  /*
2  * To change this template, choose Tools | Templates
3  * and open the template in the editor.
4  */
5
6  package ejemplo;
7
8  /**
9   *
10  * @author adi
11  */
12  public class Main {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21  }
```

Python:

```
class Objeto:  
    pass
```

```
class Antena:  
    pass
```

```
class Pelo:  
    pass
```

```
class Ojo:  
    pass
```

# Programación modular

Es un paradigma de la programación que consiste en dividir un programa en módulos o subprogramas, para así, hacerlo más fácil de entender y más manejable. Se utiliza para solucionar problemas grandes y complejos.

Los lenguajes de programación tradicionales han sido usados para hacer de soporte a la programación modular desde los años 60. La programación modular no tiene definición oficial, se puede definir como el predecesor natural de la POO.

Cuando un problema complejo se divide en varios subproblemas más simples y estos en otros aún más simples se espera que el resultado sea lo suficientemente simple como para poder resolverse rápidamente. Los “módulos” son cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el programa complejo original.

## Ejemplo de programación modular

```
program nombre_programa;  
const declarar_ctes;  
type declarar_tipos;  
var declarar_vars;
```

(\*aquí irían los subprogramas\*)

```
begin  
    cuerpo_programa  
end .
```

```
procedure nombre (lista_parametros);  
const declarar_ctes;  
type declarar_tipos;  
var declarar_vars;
```

(\*aquí irían los subprogramas\*)

```
begin  
    cuerpo_procedimiento  
end ;
```



# Otros paradigmas

## Programación genérica

- **Definición:** Es un tipo de programación cuyo objetivo es establecer unos parámetros en una serie de algoritmos de tal manera que funcionen independientemente de los datos introducidos.
- **Ejemplo:**

```
if (usuario == "tunombre")  
    mensaje = "Eres tunombre";  
else  
    mensaje = "No eres tunombre";
```

```
if (usuario == "otronombre")  
    mensaje = "Eres otronombre";  
else  
    mensaje = "No eres otronombre";
```

```
función saberNombre(nombre)  
{  
    if (usuario == nombre)  
        mensaje = "Eres " + usuario;  
    else  
        mensaje = "No eres " + usuario;  
}  
  
saberNombre(tuNombre); // Podemos usar esta llamada para cualquier tipo de nombre.
```

En la primer imagen, los algoritmos *if* i *else* deberían repetirse una y otra vez por cada nombre, y en la segunda pasarían a formar una sola función genérica valida para cualquier nombre.

## Programación distribuida

La programación distribuida va ligada a la computación distribuida, y está enfocada a desarrollar sistemas distribuidos.

Los sistemas distribuidos funcionarían utilizando un gran número de ordenadores organizados en clústeres, separados físicamente pero conectados entre sí y funcionando como uno solo.

Lenguajes diseñados para este tipo de programación son:

- Ada
- Alef
- E
- Erlang
- Limbo
- Oz

# Otros paradigmas

## Computación en la nube

También llamado Cloud computing, es un sistema informático que es ofrecido a través de Internet como servicio, de modo que el usuario puede acceder a él, pero desconoce su gestión de recursos.

## Lenguajes esotéricos

La naturaleza de los lenguajes esotéricos es probar los límites de la programación, ya sea creando lenguajes a modo de broma, arte o como un concepto profundo.

Podemos encontrar varios ejemplos realmente distintos de lenguajes esotéricos:

- **LOLCODE** (Cuya sintaxis está basada en Lolcat)
- **INTERCAL** (Creado para ser realmente difícil de entender)
- **Whitespace** (Que usa como palabras clave espacios, tabs y saltos de línea)
- **Oz** (Que incluye prácticamente todos los paradigmas existentes)
- **Piet** (Sus programas son mapas de bits en formato arte abstracto)

```
HAI
CAN HAS STDIO?
PLZ OPEN FILE "LOLCATS.TXT"?
AWSUM THX
        VISIBLE FILE
O NOES
        INVISIBLE "ERROR!"
KTHXBYE
```

# Multiparadigmas

Cualquier lenguaje de programación que soporte más de un paradigma puede ser considerado multiparadigma.

Ejemplos de lenguaje multiparadigma y sus paradigmas son:

- **PHP:** Orientado a objetos, Imperativo, funcional y reflexivo.
- **Java:** Genérico, Imperativo, orientado a objetos y reflexivo.
- **Erlang:** Funcional, concurrente y distribuido.
- **C++:** Imperativo, orientado a objetos, funcional y genérico.
- **C#:** Imperativo, funcional, orientado a objetos, reflexivo y genérico .
- **Perl:** Imperativo, orientado a objetos y funcional.
- **JavaScript:** Imperativo, orientado a objetos y funcional.
- **Python y Ruby:** Imperativo, orientado a objetos, reflexivo y funcional.
- **Scala:** Imperativo, orientado a objetos, funcional, genérico y concurrente.