PARADIGMAS DE PROGRAMACIóN

POR

CARACUEL LLABRES, KEVIN
FERRER MAYOL, MIQUEL
RIVERA RIVERA, DIEGO
VALENS CORRó, XAVIER
VILLALONGA JIMENEZ, DAVID

Indice

1.0-INTRODUCCIóN

1.1-Definición de Paradigma

Según Thomas Kuhn, los paradigmas son "*realizaciones científicas universalmente reconocidas que, durante cierto tiempo, proporcionan modelos de problemas y soluciones a una comunidad científicas*". Pero ésta es una visión muy cerrada del concepto. Luego se fue ampliando a medida que le llegaban las críticas, desde la incorporación del concepto de inconmensurabilidad o incomparabilidad hasta un abandono casi total en la filosofía actual de autores como Mario Bunge, que llega a afirmar "*que las revoluciones totales imaginadas por Thomas Kuhn, Paul K. Feyerabend y sus seguidores son sólo ficciones de su imaginación. Las ideas de inconmensurable (incomparable), paradigmas y cambios irracionales semejantes a conversiones religiosas son lógicamente inválidas e históricamente falsas. Todo avance revolucionario en un campo de conocimiento ha utilizado descubrimientos provenientes de otros campos*".

Dirigiéndonos al asunto que nos atañe particularmente, cabe indicar que la Informática tiene una particularidad como ciencia, su alta velocidad de cambio, es decir su volatilidad. La programación que es posible ejecutar en un ordenador determinado, se ve ampliada al volverse más poderosos los equipos. Pero ésta no es la única razón para la gran cantidad de paradigmas que se encuentran en las ciencias computacionales, la gran libertad que ofrecen los lenguajes existentes, permiten la creación de nuevas formas de ver cada lenguaje e interpretarlo en una máquina y esa libertad es la que permite que para cada tipo de problema específico haya uno o varios lenguajes adscritos a uno o más de un paradigma de programación. Ésta, entre otras, es la razón de que haya tantos paradigmas, tantos lenguajes y tantas maneras de resolver un mismo problema. Por lo tanto, en esta ciencia, el concepto que nos atañe, pasaría a significar un esquema formal de organización, y es utilizado como sinónimo de *marco teórico*.

1.2-Definición de Paradigma de Programación

Así llegamos a la concepción de Paradigma en la programación, que refiere a la propuesta metodológica que es adoptada por una comunidad de programadores cuyo núcleo central es incuestionable en cuanto a que únicamente trata de resolver uno o varios problemas claramente delimitados. La resolución de estos problemas debe suponer consecuentemente un avance significativo en al menos un parámetro que afecte a la ingeniería de software. Un paradigma vendría a representar un enfoque particular para diseñar soluciones.

Los lenguajes de programación están definidos por los diversos paradigmas existentes. Nos podemos encontrar lenguajes diseñados específicamente para explotar un paradigma, como por ejemplo el Haskel que se diseñó específicamente para el paradigma funcional, y luego, nos podemos encontrar con lenguajes amplios y versátiles como Java o C++ que pueden albergar paradigmas estructurados, orientado a objetos o multiparadigmas, siendo el programador usuario del lenguaje quien elige y decide qué usar en cada caso y como enfocar la mejor solución a un problema.

Cada paradigma tiene sus particularidades que lo hacen eficiente a la hora de abordar un problema determinado y tienen sus ventajas y desventajas. No existe el paradigma perfecto en el que se puedan encontrar las mejores soluciones a todos los problemas, cada tipo de problema necesita unas ciertas herramientas que conduzcan a su solución de una manera eficaz y elegante.

Apéndice: Programación: ¿Arte o Ciencia?
La programación es el corazón de todo proyecto de software y depende del talento del ser humano que la ejecuta.
Muchos autores como Donald Kunth, siguen considerando que la programación aún no está en condiciones de recibir el título de ciencia y han señalado que a pesar de todos los formalismos creados, el diseño de algoritmos y su codificación, ésta sigue siendo un arte.

Todos los conocimientos, técnicas, herramientas y métodos son imprescindibles para formar a un ingeniero de sistemas. Son el equivalente en los pintores a dominar la teoría del color y las técnicas pictóricas, pero ese conocimiento no puede, ni podrá jamás, guiar su mano para crear una obra maestra. El propio programador sería el encargado de darle el toque personal a un programa.

Edsger Dijkstra, por su parte, nos recuerda que la programación se basa en la investigación de las áreas de desarrollo software y en la creación de programas con bases matemáticas, lingüísticas y lógicas.

La opinión de quien escribe este texto es que es necesaria la aplicación de la noción de complejidad a este dilema. No se puede dar una valoración simple ya que se caería en un dualismo en el que ambas opciones son mutuamente excluyentes. Así, la programación es un conocimiento complejo y para poder hacer uso de él es imprescindible el manejo de las herramientas metodológicas y científicas fruto de la investigación y, por otra parte, una cierta pericia dada por la experiencia y la creatividad.

Un ejemplo clarificador es el de la música, para dominar un instrumento no basta con saberse las escalas y reglas de ritmo, armonía, melodía al dedillo sino que es necesario un alto componente de experiencia, llamado en el mundillo ''tablas'' y otro componente de creatividad. Y la cuestión es que para un músico ambas visiones son necesarias.

1.3-Teoría de la Programación

La teoría de la programación es el conjunto de principios que rigen un paradigma y definen las características semánticas de cierto conjunto de lenguajes y aspira a dar certidumbre sobre las formas más adecuadas de enfrentar un problema, modelarlo y encontrar su solución para poder codificarlo en algún lenguaje y ejecutar la solución en una máquina.

Se puede retroceder en el tiempo hasta 1830 para comenzar a ver los primeros atisbos de los que ya en el s.XX serán los paradigmas de programación en toda su esencia.

Charles Babbage desarrolla el concepto de Máquina Analítica cuya arquitectura está formada por cinco módulos: entrada, cálculo, control, memoria y salida. Esta fue la base de la ENIAC, en 1947, considerada la primera computadora digital de la historia. La técnica de programación se hacía mediante un cableado para resolver alguna tarea específica y si se deseaba tener un programa diferente se tenía que modificar ese cableado.

Algunos años después de la ENIAC, John von Neumann crea un modelo que consiste en permitir la coexistencia de datos con instrucciones en la memoria para que la computadora pueda ser programada sin cables. Esta idea es de tal envergadura que obliga a los fabricantes de computadoras a revisar su arquitectura, generando el nacimiento del paradigma de programación libre y los primeros lenguajes de programación.

Es en este momento en el que nace el paradigma de programación como un todo, que luego se bifurcará en paradigmas completos con sus propias teorías de programación, pero este es el punto de inflexión, el giro copernicano que facilita la rápida evolución de los sistemas informáticos y da lugar a muchos tipos diferentes de programación para resolver gran cantidad de problemas.

2.0-PARADIGMAS

2.1-Paradigma Declarativo

La primera gran división que de forma clara se puede hacer es la de los paradigmas imperativo y declarativo. Comenzaremos con el paradigma declarativo y seguiremos a partir de ahí con los imperativos ya que en la programación web el paradigma dominante es consecuente de este último.

La Programación Declarativa está basada en el desarrollo de programas especificando un conjunto de condiciones, proposiciones, afirmaciones, restricciones, ecuaciones o transformaciones que describen el problema y detallan su solución. Luego la solución es obtenida mediante mecanismos internos de control, sin especificar exactamente cómo encontrarla. Se le indica al ordenador qué es lo que se desea obtener o qué es lo que se está buscando y éste nos da una respuesta.

Las sentencias que se utilizan describen el problema que se quiere solucionar, pero no las instrucciones necesarias para solucionarlo. Esto último se realizará mediante mecanismos internos de inferencia de información a partir de la descripción realizada.

Una de las principales características de este tipo de lenguajes es que minimiza los efectos secundarios imprevistos que puedan surgir al crear un programa.

Al aplicar este estilo no se usan algoritmos, sino que se le dice a la máquina qué es lo que tiene que hacer y no cómo hacerlo o qué pasos seguir. El cómo es dado por los métodos de implementación del lenguaje.

La programación declarativa suele considerar los programas como teorías de una lógica formal y computaciones y deducciones de ese espacio lógico. Bases de Datos Consultivas, tipo SQL, programación lógica, programación funcional, programación con restricciones, programas DSL (de dominio específico) son claros ejemplos de este paradigma.

2.2-Paradigma Lógico

La programación Lógica propone la implementación de programas usando lógica matemática o lógica formal. Este paradigma está basado en la esa lógica formal, es decir, una serie de sentencias que expresan reglas y hechos acerca de un problema. Las reglas se escriben en forma de condiciones del tipo H : - B1 (cabeza : – cuerpo)(condición : - conclusiones). Los hechos son sólo encabezados: H. .

La interpretación de la lógica predicativa como lenguaje de programación, se basa en la interpretación de implicaciones del tipo H si B1 y … y Hn como declaraciones de procedimientos. Los problemas individuales se exponen como teoremas que han de ser probados.

Las pruebas son computaciones que se generan en los axiomas. Uno de los lenguajes más usados es el prolog:

- Prolog: Es un lenguaje para programar artefactos electrónicos con técnicas de producción final interpretada y es muy usado en las investigaciones en Inteligencia Artificial. Las instrucciones se componen de cláusulas de Horn que constituyen reglas del tipo modus ponens, es decir, “Si es verdad el antecedente, entonces es verdad el consecuente" y se lee como "La cabeza es verdad si el cuerpo es verdad".
  Ejemplo:

```
profesor(Joan)                    <--  Hecho, también expresable como:
profesor(Joan) :- true            <--  A modo de conclusión
```

- Luego se puede preguntar:

```
?- profesor(Joan).
Yes   <-- Respuesta
?- profesor(X).
X = Joan
```

2.3-Paradigma Funcional

Es un paradigma de programación declarativa que se basa en el concepto de funciones aritméticas, usadas como una evolución de predicados de corte matemático que tiene sus raíces en el cálculo lambda, un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión. Al usar un lenguaje funcional, el valor generado por una función depende exclusivamente de los argumentos alimentados a la función. Así se eliminan los efectos secundarios y se puede entender y predecir el comportamiento de un programa mucho más fácilmente. La secuencia de computaciones llevadas a cabo por el programa se rige única y exclusivamente por la reescritura de definiciones más amplias a otras cada vez más concretas y definidas, usando lo que se denominan "definiciones dirigidas".

Estos lenguajes son usados en ambientes académicos, en algunas aplicaciones comerciales e industriales y a través de lenguajes de dominio específico para estadísticas, matemáticas simbólicas y análisis financiero.

Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones en las que se verifican ciertas propiedades como la transparencia referencial, es decir, que el significado de una expresión depende únicamente del significado de sus subexpresiones. No hay asignaciones de variables ni iteraciones.

Se utilizan funciones de orden superior y de primera clase, que son funciones que pueden tomar otras funciones como argumentos o devolverlos como resultados, expresiones o la recursividad para lograr las iteraciones.

Podemos encontrar lenguajes funcionales híbridos como Lisp, Scheme, SAP y lenguajes funcionales puros como Miranda y Haskell.

- Haskell: Es un lenguaje de programación estandarizado multi-propósito puramente funcional con semánticas no estrictas y fuertemente tipificado (typeful). No decimos al ordenador lo que tiene que hacer, sino más bien, decimos cómo son las cosas. Lo único que puede hacer una función es calcular y devolver algo como resultado. En Haskell se ven los programas como una serie de transformaciones de datos. Además usa un sistema de tipos que puede deducir de forma inteligente qué tipos usar en cada momento: enteros, cadenas de texto, etc. Ahora veremos un ejemplo:

El siguiente es un ejemplo de una función de multiplicación con una sentencia If … Else (Si … Sino)

```
    doubleSmallNumber
x = if x > 100

 then x
 else x*2
```

2.4-Paradigma Imperativo

Definición:

Es el paradigma de programación que describe la programación como una secuencia de sentencias u órdenes que cambian el estado del programa. Los programas imperativos son un conjunto de instrucciones secuenciales que indican al computador cómo realizar una tarea o encontrar la solución a un problema. También puede utilizar subrutinas como veremos en lenguajes mas avanzados mas adelante.

Historia:

La programación imperativa es muy cercana a la arquitectura física del computador. La arquitectura que se utiliza en los computadores y que es la base de la programación imperativa es la arquitectura tradicional propuesta por Von Neumann. En esta arquitectura los datos se almacenan en una memoria a la que se accede desde una unidad de control ejecutando instrucciones de forma secuencial.

El estilo de programación imperativo constituye la primera forma de programar ordenadores. El ensamblador y el código máquina que se utilizaban antes de desarrollarse los primeros lenguajes de programación, tienen un enfoque totalmente imperativo.

Los primeros lenguajes de programación de alto nivel (como el Fortran) eran abstracciones del lenguaje ensamblador y mantenían este enfoque. Lenguajes más modernos como el BASIC o el C han continuado esta idea.

En los años 60 se introducen conceptos de programación procedural en los lenguajes de programación. La programación procedural es un tipo de programación imperativa en el que los programas se descomponen procedimientos(también llamados subrutinas).
A finales de la década de los 60 Edsger W. Dijkstra, una de las figuras más importantes en la historia de la computación, publicó en la revista '*Communications of the ACM*'el importante artículo que propone que la sentencia 'GOTO' se elimine de los futuros lenguajes de programación.
El propósito de la instrucción 'GOTO' es transferir el control a un punto determinado del código, donde debe continuar la ejecución. El punto al que se salta, viene indicado por una etiqueta. GOTO es una instrucción de salto incondicional.

Esta instrucción ha sido menospreciada en los lenguajes de alto nivel, debido a la dificultad que presenta para poder seguir adecuadamente el flujo del programa, tan necesario para verificar y corregir los programas.
Este artículo marca el inicio de una nueva tendencia de programación denominada 'programación estructurada' que, manteniendo la programación imperativa, intenta conseguir lenguajes que promuevan programas correctos, modulares y mantenibles.

Características:

-Elementos:

Los elementos más importantes de este paradigma son:

• **Tipos de datos:** Es un atributo de una parte de los datos que indica al ordenador (y/o al programador) algo sobre la clase de datos sobre los que se va a procesar. Esto incluye imponer restricciones en los datos, como qué valores pueden tomar y qué operaciones se pueden realizar.

Los tipos de datos primitivos son :

Caracteres:

- char: ((a-z),(0-9) y símbolos ($,_*. etc.)).

Numéricos:

Para expresar números enteros (sin decimal) utilizamos.
-byte: 8 bits.
-short: 16 bits.
-int: 32 bits.
-long: 64 bits.
Para expresar números reales (con parte entera y parte decimal)
-float: 32 bits.
-double: 64 bits.

Booleanos:

Para representar valores lógicos, nos devuelven resultados lógicos.
-boolean: Rango, true-false.

• **Variables:** Se encargan de almacenar o dar referencia al estado del programa. Son objetos cuyo valor depende de una localidad de memoria o celda y que puede cambiar a lo largo de la ejecución del programa. Los estados de un programa son representados y diferenciados por su conjunto de variables y sus contenidos.

• **Expresiones:** En la programación imperativa su papel más importante está en las sentencias de asignación. Una sentencia de asignación sirve para modificar el valor de una variable y así cambiar el estado de un programa. Los operadores empleados pueden ser matemáticos, lógicos o una combinación de ambos.

-Explicación:

El paradigma imperativo se caracteriza por:

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

En el paradigma imperativo se determinan los datos que se requieren para el cálculo, estos datos son asociados a unas variables, para luego ejecutar paso a paso las transformaciones de los datos almacenados en la memoria, de forma que el valor final de la variable sea el resultado que buscamos. Estas transformaciones realizadas sobre los datos originales asociados a las variables las llamamos cálculos, que estos son generados por los algoritmos que transforman los datos originales en la solución al problema en cuestión.

- Flujo de datos:

Para guiar estos cálculos, el paradigma imperativo proporciona en su versión básica las siguientes estructuras de control de flujos de datos:

• **Secuencial:** Las instrucciones del programa se ejecutan una a continuación de la otra.

• **Selección condicional:** Produce una bifurcación del flujo de datos, cuando una condición previamente definida se cumple, la ejecución del programa se bifurca, cuando es el caso contrario el programa sigue su curso secuencial. Seria el caso del *if, if else*.

• **Selección incondicional:** La estructura secuencial del flujo de datos se bifurca obligadamente o imperativamente, obligando a ejecutar una instrucción distinta a la siguiente en orden que el programador ha especificado. Seria el caso de una subrutina.

La evolución ha generado otros nuevos tipos de estructuras de control de flujo, tales como while, do etc. que simplifican la tarea del programador, creando nuevas sentencias de control.

• **Sentencias de iteración :** Definen instrucciones que se ejecutan de forma repetitiva hasta que se cumple una determinada condición.

-Límites del paradigma imperativo:

-El paradigma imperativo tiene su límite natural en una de sus características intrínsecas:

• **Efectos laterales:** Estos efectos son los producidos por la modificación continua de las celdas de memoria durante el programa, donde un error puede desencadenar una cadena de errores que harán que el resultado que necesitamos sea erróneo y por tanto que nuestro programa no sea seguro y más difícil de predecir. Por otra parte el concepto de una única memoria global que se actualiza por cada instrucción dificulta la portabilidad y reusabilidad del código, es decir, no tiene transparencia referencial. Si cambiamos una expresión *E* por *V* la semántica del programa va a diferir de la original.

-Tipos de lenguaje basados en el paradigma imperativo.

.Lenguaje ensamblador:

Es un lenguaje de programación de bajo nivel para los computadores, microprocesadores, microcontroladores y otros circuitos integrados programables. Es la representación más directa del código máquina específico para cada arquitectura legible por un programador.

```
; ------------------------------------------
; Programa que imprime un string en la pantalla
; ------------------------------------------
        .model small              ; modelo de memoria

        .stack                    ; segmento del stack

        .data                     ; segmento de datos
        Cadena1 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 Cadena1  ; 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
```

•Fortran:

Fortran es un lenguaje de programación de alto nivel está especialmente adaptado al cálculo numérico y a la computación científica. Desarrollado originalmente por IBM en 1957.Este lenguaje es característico de los años 60 ,como explicamos en la historia de los paradigmas imperativos, porque introduce el sistema de subrutinas.

Subrutinas: Es un subalgoritmo que forma parte de algoritmo principal que resuelve una tarea particular necesaria para que dicho algoritmo alcance el objetivo para el que fue diseñado.Es un segmento de código separado del bloque principal que puede ser invocado por este o por otra subrutina. Cuando es invocada hace que la ejecución del código principal se detenga y pase a ejecutar el código de la subrutina.Cuando acabe la ejecución de la subrutina, esta volverá donde fue invocada a través de la instruccion *'return'*.

```
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
```

-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.

2.5-Programación dinámica

- Introducción:

La programación dinámica es a la vez un método de optimización matemática y un método de programación de computadoras. En ambos contextos se refiere a la simplificación de un problema complicado dividiéndolo en subproblemas más simples de una manera recursiva. Esto fue nombrado el "Principio de optimalidad". Del mismo modo, un problema que se puede resolver de forma óptima dividiéndola en subproblemas y luego encontrar de forma recursiva las soluciones óptimas a los subproblemas se dice que tiene subestructura óptima.

Un concepto a tener en cuenta en este tipo de programación es la recursividad, este concepto engloba que un hecho puede repetirse indefinidamente.

- Historia:

El término programación dinámica fue utilizado originalmente por Richard Bellman en los años 40, para describir el proceso de resolver problemas donde se necesita encontrar las mejores definiciones una tras otra. En 1953 modificó su significado a uno más moderno, el cual se refiere específicamente en encontrar pequeños problemas de decisión dentro de grandes decisiones.

- Características:

- Hace uso de:

•Subproblemas: En programación se dice que un problema puede tener subproblemas superpuestos si el problema se puede dividir en subproblemas que se reutilizan varias veces y que por tanto serán más sencillos de resolver.

•Subestructura óptima: En informática, se dice que un problema tener subestructura óptima si una solución óptima se puede construir de manera eficiente de soluciones óptimas de sus subproblemas.
En general, se pueden resolver problemas con subestructuras óptimas siguiendo estos tres pasos:
1-Dividir el problema en subproblemas más pequeños.
2-Resolver estos problemas de manera óptima usando este proceso de tres pasos recursivamente.(Dividiendo los problemas en problemas más pequeños).
3-Usar estas soluciones óptimas para construir una solución óptima al problema original.

Los subproblemas se resuelven a su vez dividiéndolos en subproblemas más pequeños hasta que se alcance el caso fácil, donde la solución al problema es trivial, es decir, que la solución tiene una estructura muy simple. Por ejemplo s=1.

•**Memoización:** Esta técnica de optimización es utilizada principalmente para acelerar los programas de ordenador mediante el almacenamiento de los resultados de las llamadas de las funciones más costosas en la memoria **caché** y devolviendo este resultado cuando las mismas entradas vuelvan a ocurrir.

Caché: En informática, la caché es la memoria de acceso rápido de una computadora, que guarda temporalmente las últimas informaciones procesadas.

- Enfoques:

La programación dinámica toma normalmente uno de los dos siguientes enfoques:

•**Top-down:** El problema se divide en subproblemas, y estos se resuelven recordando las soluciones por si fueran necesarias nuevamente. Es una combinación de memoización y recursividad.
•**Bottom-up:** Todos los problemas que puedan ser necesarios se resuelven de antemano y después se usan para resolver las soluciones a problemas mayores. Este enfoque es ligeramente mejor en consumo de espacio y llamadas a funciones, pero a veces resulta poco intuitivo encontrar todos los subproblemas necesarios para resolver un problema dado.

- Ejemplos:

Ejemplo de programa dinámico enfocado a una organización Bottom-up:

```
FUNC Fibonacci (↓n: NATURAL): NATURAL
VARIABLES
    tabla: ARRAY [0..n] DE NATURALES
    i: NATURAL
INICIO
    SI n = 0 ENTONCES
        DEVOLVER 0
    SINOSI n = 1 ENTONCES
        DEVOLVER 1
    SINO
        tabla[0] := 0
        tabla[1] := 1
        PARA i = 2 HASTA n HACER
            tabla[i] := tabla[i-1] + tabla[i-2]
        FINPARA
        DEVOLVER tabla[n]
    FINSI
FIN
```

Ejemplo de programa dinámico enfocado a una organización Top-down:

```
FUNC Fibonacci (↓n: NATURAL, ↕tabla: ARRAY [0..n] DE NATURALES): NATURAL
 VARIABLES
    i: NATURAL
 INICIO
     SI n <= 1 ENTONCES
         devolver n
    FINSI
     SI tabla[n-1] = -1 ENTONCES
        tabla[n-1] := Fibonacci(n-1, tabla)
    FINSI
     SI tabla[n-2] = -1 ENTONCES
        tabla[n-2] := Fibonacci(n-2, tabla)
    FINSI
        tabla[n] := tabla[n-1] + tabla[n-2]
    devolver tabla[n]
FIN
```

- Conclusión:

La programación dinámica se aplica cuando la subdivisión de un problema conduce a:

•Una enorme cantidad de problemas.
•Problemas cuyas soluciones parciales se solapan.
•Grupos de problemas de muy distinta complejidad.

2.6-Programación estructurada

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** 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 mas 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:

1. **Secuencia**: ejecución de una instrucción tras otra.

2. **Selección**: ejecución de una de dos instrucciones (o conjuntos), según el valor de una variable booleana.

3. **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
```

- **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#
- Fortan
- Java
- Perl
- PHP
- Lua

Ejemplo codigo C, calcular el area 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;
```

2.7-Programación orientada a objetos

-Definición: Es un paradigma de programación que representa el concepto de **objetos** y los usa en sus interacciones.

-Explicación: Los objetos que son representados por los paradigmas de programación tienen campos de datos que son **atributos** que describen el objeto y además, tiene procedimientos asociados llamados métodos. Para diseñar aplicaciones y/o programas los objetos (los cuales, son utilizados por lo general como **instancias** de clases) interaccionan con otros objetos para así, poder empezar con el diseño. La POO (programación orientada a objetos) intenta enseñar un modelo de programación basado en objetos. Incorpora código y datos utilizando el concepto de objeto. Un objeto es un tipo de dato abstracto con el añadido del **polimorfismo** y la **herencia.**

Un objeto puede tener un estado (datos) y un comportamiento (código). A veces los objetos corresponden a cosas que existen en la vida real. Por ejemplo, un programa de gráficos puede tener objetos como «círculo», «cuadrado», «menú». Un sistema de compra online tendrá objetos tales como «carro de la compra», «cliente» y «producto». El sistema de compra soportará comportamientos tales como «añadir producto», «pagar el producto» o «oferta de descuento».
Los objetos son designados en jerarquías de clases. Por ejemplo, con el sistema de compra puede haber **clases** de alto nivel tales como «productos electrónicos», «productos de cocina», y «libros». También puede haber subclases debajo de algunas clases como «productos electrónicos» entre otros. Estas clases y subclases corresponden a conjuntos y subconjuntos en la lógica matemática. En lugar de utilizar tablas de bases de datos y subrutinas de programación, el desarrollador utiliza los objetos los cuales el usuario puede estar más familiarizado: por ejemplo, objetos de su aplicación favorita.
La orientación de objetos utiliza encapsulación. e **ocultación de información.** La programación orientada a objetos fusiona los tipos de datos abstractos con programación estructurada y divide los sistemas en objetos modulares con sus propios datos y comportamientos. Esto se conoce como encapsulación. Con la encapsulación, los datos de dos objetos están divididos, así que un cambio en el primer objeto no puede afectar al segundo.

-Motivación (como se piensa en objetos)

Durante mucho tiempo los programadores se han dedicado a crear aplicaciones para resolver problemas. Para conseguir que estas aplicaciones puedan ser utilizadas por otras personas se creó la POO, también conocida como programación orientada a objetos.
La programación orientada a objetos es una serie de normas sobre como realizar las cosas de manera que otra gente pueda utilizarlas y adelantar su trabajo. Así el código se puede reutilizar.
La POO no es tan difícil como parece, se necesita una manera especial de pensar, a veces subjetiva. Así que la forma de hacer las cosas puede ser diferente dependiendo de cada persona. Aunque podamos hacer los programas de formas distintas, no quiere decir que todas las formas sean correctas, lo difícil no es programar, sino programar bien.
Pensar en objetos es muy parecido a como lo haríamos en la vida real. Por ejemplo vamos a pensar en un monitor para tratar de hacernos una idea. El monitor es el elemento principal que tiene una serie de características, como la marca, el modelo, el tamaño de la pantalla. Además tiene una serie de funciones como encenderse, apagarse, variar el brillo , etc.

Pues en un esquema de POO el monitor sería el objeto, las propiedades (atributos) serían las características como el modelo, la marca..., y los métodos serían las funciones como encenderse o apagarse.

-Historia de la POO

La aparición de términos como “objetos” y “orientado” en el ámbito moderno de la programación orientada a objetos hizo su primera aparición en el MIT a finales de los años 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 referían al apartado de interacción gráfica.

-Características de la POO

-Estructura de un objeto

Un objeto puede dividirse en tres partes:

1- **Relaciones**
2- **Propiedades**
3- **Métodos**

Cada uno de esos componentes es independiente del otro:

Las **relaciones** permiten que el objeto se integre en la organización
Las **propiedades** distinguen un objeto determinado de los restantes que forman parte de la misma organización y tiene valores que dependen de la propiedad que se trate
Los **métodos** son las operaciones que pueden realizarse sobre el objeto

Normalmente, los objetos forman siempre una organización jerárquica, es decir, ciertos objetos son superiores a otros de cierta manera. Existen varios tipos de jerarquías:
serán **simples** cuando su estructura puede representarse con un "árbol". En otros casos es más compleja.

De todos modos, sea la estructura simple o compleja siempre se podrán distinguir tres niveles de objetos:

La raíz
Objetos intermedios
Objetos terminales

La raíz (de la jerarquía): Es un objeto que se caracteriza por estar en el nivel más alto de la estructura. Se le suele llamar "objeto madre" o "Entidad"
Objetos intermedios: Son aquellos objetos que se sitúan debajo de la raíz y que a su vez tienen a otras clases o conjuntos de objetos debajo. Representan clases o conjuntos de objetos. Ejemplo: Ventana, cuenta, fichero...
Objetos terminales: Son aquellos objetos que descienden de una clase o subclase y que no tienen ninguna clase debajo. Suelen ser instancias.

Profundicemos más en los elementos de la estructura de un objeto mencionados anteriormente:

Relaciones: Son los enlaces que permiten a un objeto relacionarse con aquellos objetos que formen parte de la misma organización.

Los hay de dos tipos:

Relaciones jerárquicas
Relaciones semánticas

Relaciones jerárquicas: Son vitales para la existencia de la aplicación ya que la construyen. Son bidireccionales, es decir, un objeto es padre de otro cuando el primer objeto se encuentra situado encima del segundo en la organización en la que forman parte, también, si un objeto es padre de otro, el segundo es hijo del primero.

Relaciones semánticas: Se refieren a las relaciones que no tienen nada que ver con la organización de la que forman parte los objetos que las establecen.

Propiedades: Todo objeto puede tener cierto número de propiedades, cada una de las cuales tendrá uno o varios valores. En POO las propiedades corresponden a las clásicas "variables" de la programación estructurada.

Existen diferentes tipos de propiedades:

Propiedades propias
Propiedades heredadas

Propiedades propias: Están formadas dentro del objeto (con encapsulación)
Propiedades heredadas: Están definidas por un objeto diferente, familiar de este ("padre", "abuelo", etc.)

Métodos: Son las operaciones que realizan acceso a los datos. Podemos definirlo mejor como un programa procedimental escrito en cualquier lenguaje que está asociado a un objeto determinado y cuya ejecución sólo puede desencadenarse a través de un mensaje recibido por este objeto o sus descendientes.

Existen diferentes tipos de métodos:

Métodos propios
Métodos heredados

Métodos propios: Están incluidos dentro del objeto
Métodos heredados: Están definidos por un objeto diferente, familiar de este ("padre", "abuelo", etc.)

-Características secundarias de la POO

Tipificación
Concurrencia
Persistencia

Tipificación: Permite la agrupación de objetos en tipos
Concurrencia: Varios objetos pueden actuar al mismo tiempo
Persistencia: Un objeto puede seguir existiendo tras desaparecer su antecesor.

-Ventajas de la POO

Reutilización
Mantenimiento
Modificación
Fiabilidad

Reutilización: Cuando las clases están diseñadas adecuadamente, se pueden usar en distintas partes del programa y en otros proyectos

Mantenimiento: Debido a su sencillez para abstraer el problema, los Programas OO (orientados a objetos) son más fáciles de leer y entender ya que nos permiten ocultar detalles de implementación dejando visibles los detalles más importantes.

Modificación: Es la facilidad de añadir, suprimir o modificar nuevos objetos permitiendo modificarlos de manera fácil e intuitiva.

Fiabilidad: Se divide el problema en partes más pequeñas para probarlas de manera independiente y aislar mejor los posibles errores que pueda tener.

Puntos fuertes:

Fomenta la reutilización y extensión del código
Permite crear sistemas más complejos
Relaciona el sistema con el mundo real
Facilita la creación de programas visuales
Permite la construcción de prototipos
Agiliza el desarrollo de software
Facilita el trabajo en equipo y el mantenimiento del software.

Puntos débiles:

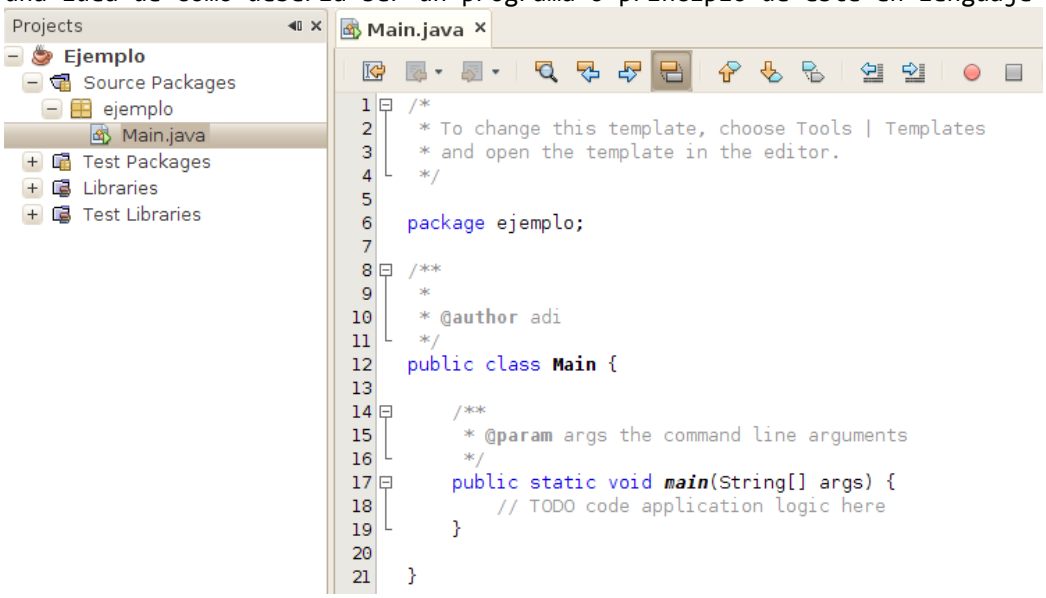
Cambio en la forma de pensar de la POO tradicional
La ejecución de programas OO (orientados a objetos) es más lenta
La necesidad de utilizar bibliotecas de clases obliga a los programadores a aprender a usarlas
Exige conocer a fondo la teoría de objetos
Requiere mayor capacidad de los programadores

-Ejemplos de lenguajes orientados a objetos

Voy a poner ejemplos sobre dos lenguajes de programación orientada a objetos. Java y Python.

Java

Este es un trozo de código escrito en Java. Se trata del código fuente de un programa (aunque no está empezado aun) Solo es un código generado por el programa NetBeans que genera un código con solo decirle un tipo de proyecto pero aún así, podemos hacernos una idea de como debería ser un programa o principio de este en lenguaje OO java



Python

Este es un ejemplo de una clase en lenguaje de programación Python

```
class Objeto:
    pass

class Antena:
    pass

class Pelo:
    pass

class Ojo:
    pass
```

2.7-Programación Modular

-Introducción

La Programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas para así, hacerlo más fácil de entender y más manejable. Se presenta como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos.

-Historia

Los lenguajes de programación tradicionales han sido usados para hacer de soporte a la programación modular desde los años 60. La programación modular es un concepto que no tiene definición oficial. Se puede definir como el predecesor natural de la POO, o una evolución de este (dependiendo del punto de vista de cada uno).

-Características

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 el problema de una manera mucho más fácil. 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. Un módulo puede entenderse como una parte de un programa en cualquiera de sus formas.

-Ejemplos

Ejemplo de "subprograma" (procedimientos) en programación modular. Solo es el esquema:

```
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 ;
```

Definiciones:

Objetos: Son programas de ordenador que constan de un estado y un comportamiento.

Atributos: Son las características individuales que diferencian un objeto de otro y determinan su apariencia.

Instancia: Es una realización específica de una clase o prototipo determinado.

Clase: Es una plantilla para la creación de objetos de datos según un modelo específico.

Polimorfismo: Es el dominio de una sola interfaz para entidades de diferentes tipos. Un tipo de polimorfismo sería uno cuyas operaciones pueden ser también aplicadas a valores de algún otro tipo o tipos.

Herencia: En la programación orientada a objetos, la herencia es cuando un objeto o clase esta basado en otro objeto o clase, utilizando la misma implementación (heredandolo de una clase) o especificandolo en una aplicación para mantener el mismo comportamiento.

Información oculta: En programación, información oculta es la ocultación de decisiones de diseño en un programa para proteger partes de su código si hay algún cambio (ya sea o no inesperado)

LISP: Lenguaje de programación OO y multiparadigma.

Alan Kay: Informático estadounidense nacido el 17 de mayo de 1940. Es conocido por sus trabajos pioneros en la POO y el diseño de sistemas de interfaz gráfica de usuario (GUI)

Sketchpad: Fue el primer programa informático que permitía la manipulación directa de objetos gráficos; viene a ser el primer programa de dibujo por ordenador.

-Otras definiciones

Demonios: Es un tipo especial de métodos, relativamente poco frecuente en los sistemas de POO, que se activa automáticamente cuando sucede algo especial. Es decir, es un programa, como los métodos ordinarios, pero se diferencia de estos porque su ejecución no se activa con un mensaje, sino que se desencadena automáticamente cuando ocurre un suceso determinado. Ejemplo: la asignación de un valor a una propiedad de un objeto,la lectura de un valor determinado, etc.

Abstracción: Es la capacidad de aislar y encapsular la información del diseño y la ejecución. También identifica los atributos y los métodos.

2.8-Programación genérica

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.
Es decir, que es un sistema en el cual los datos serán introducidos más tarde, no durante la creación del algoritmo.

Ejemplo:

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

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

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.

```
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.
```

2.9-Programación distribuida

La programación distribuida va ligada a la computación distribuida, y está enfocada a desarrollar sistemas distribuidos.
Los sistemas distribuidos funcionaran utilizando un gran numero de ordenadores organizados en clústeres, separados físicamente pero conectados entre sí y funcionando como uno solo.

Lenguajes diseñados específicamente para este tipo de programación son:

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

2.10-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 el, pero desconoce su gestión de recursos.

Como paradigma se podría decir que es información que se almacena permanentemente en servidores alrededor del mundo y se envía al cliente a través de caches temporales.

Ejemplos claros son las aplicaciones de Google o Amazon.

2.11-Lenguajes esotéricos

La naturaleza de los lenguajes esotéricos es probar los limites de la programación, ya sea creando lenguajes a modo de broma, arte o como un concepto profundo.
Es dudable la utilidad de los lenguajes estoreticos en grandes proyectos serios, y es que usualmente, el creador de el lenguaje lo hace a modo de prueba o sencillamente para alejarse todo lo posible de los cánones de los lenguajes comunes.

Podemos encontrar varios ejemplos realmente distintos de lenguajes esotericos:

- **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 linea)
- **Oz** (Que incluye prácticamente todos los paradigmas existentes)
- **Piet** (Sus programas son mapas de bits en formato arte abstracto)

3.0-MULTIPARADIGMAS

Cualquier lenguaje de programación que soporte más de un paradigma puede ser considerado multiparadigma.
Estos lenguajes permiten crear programas usando más de un estilo de programación.
No significa que vayan a utilizar distintos paradigmas combinados o que los vayan a utilizar siempre, sencillamente disponen de esa herramienta. Aunque existen casos aislados, como **Oz**, que permiten incluso mezclarlos de forma natural.

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.

4.0-BIBLIOGRAFIA

http://www.uvmnet.edu/investigacion/episteme/numero1-05/enfoque/a_vision.asp
http://cyt-ar.com.ar/cyt-ar/index.php/Paradigma
''La estructura de las revoluciones científicas'', Thomas Kuhn (1962)
''Las pseudociencias ¡vaya timo!'', Mario Bunge (2010)
http://people.cs.aau.dk/~normark/prog3-03/html/notes/theme-index.html
''The Art of Computer Programming. Vol. 1'', Donald Knuth (1997)
http://cala.unex.es/cala/epistemowikia/index.php?title=Programaci%C3%B3n:_%C2%BFarte_o_ciencia%3F
http://foldoc.org/declarative+language
''Predicate Logic as a Programming Language'', Robert Kowalski (http://www.doc.ic.ac.uk/~rak/papers/IFIP%2074.pdf)
''The repeated demise of logic programming and why it will be reincarnated''. Carl Hewitt.
''Aplicaciones de la Inteligencia Artificial en la Actividad Empresarial, la Ciencia y la Industria''. Wendy B. Rauch-Hindin
Una Introducción agradable a Haskell http://www.lcc.uma.es/~blas/pfHaskell/gentle/index.html
¡Aprende Haskell por el bien de todos! http://aprendehaskell.es/main.html
http://es.wikipedia.org/wiki/Programaci%C3%B3n_imperativa
http://web.ua.es/dccia/
http://es.wikipedia.org/wiki/Fortran
http://es.wikipedia.org/wiki/Tipo_de_dato
http://es.wikipedia.org/wiki/Expresi%C3%B3n_(inform%C3%A1tica)
http://es.wikipedia.org/wiki/Efecto_secundario_(inform%C3%A1tica)
http://www.mindmeister.com/176377774/paradigmas-de-programaci-n
http://es.wikipedia.org/wiki/Lenguaje_ensamblador
http://es.wikipedia.org/wiki/GOTO
http://centrodeartigo.com/articulos-noticias-consejos/article_140108.html
http://es.wikipedia.org/wiki/Historia_de_los_lenguajes_de_programaci%C3%B3n
http://es.wikipedia.org/wiki/Arquitectura_de_von_Neumann
http://es.wikipedia.org/wiki/Programaci%C3%B3n_din%C3%A1mica
http://es.wikipedia.org/wiki/Recursi%C3%B3n
http://www.wordreference.com/definicion/recursivo
http://es.wikipedia.org/wiki/Cach%C3%A9_(inform%C3%A1tica)
http://en.wikipedia.org/wiki/Memoization
http://es.wikipedia.org/wiki/Trivial_(matem%C3%A1tica)
http://en.wikipedia.org/wiki/Dynamic_programming
http://es.wikipedia.org/wiki/Richard_Bellman
http://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos
http://www.desarrolloweb.com/articulos/499.php
http://www.monografias.com/trabajos/objetos/objetos.shtml
http://librosweb.es/libro/python/capitulo_5/programacion_orientada_a_objetos.html
http://programacion.net/articulo/programacion_orientada_a_objetos_279
http://es.slideshare.net/mayibarra10/paradigmas-de-programacin-14133043
http://es.slideshare.net/Pxnditx10/programacin-modular-22705773
http://aprendiendojee.wordpress.com/2010/08/02/operadores-y-tipos-de-datos-i/
http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/clases.htm