PARADIGMAS DE PROGRAMACIÓN.

PARADIGMAS DE PROGRAMACIÓN.

Autor : Cristian Salazar Ruiz, Cristian David Arbeláez Orozco

*Ing. En sistemas y computación, Universidad Tecnologica De Pereira, Pereira, Colombia*

Correo-e: [cristian.salazar1@utp.edu.co](mailto:cristian.salazar1@utp.edu.co) , c.arbelaez@utp.edu.co

***Resumen*—Un paradigma de programación es un estilo de desarrollo de programas. Es decir, un modelo para resolver problemas computacionales. Los lenguajes de programación, necesariamente, se encuadran en uno o varios paradigmas a la vez a partir del tipo de órdenes que permiten implementar, algo que tiene una relación directa con su sintaxis.**

***Palabras clave—* Paradigma, programación, resolver, implementar.**

***Abstract*— A programming paradigm is a program development style. That is, a model for solving computational problems. Programming languages, necessarily, are framed in one or several paradigms at the same time from the type of commands they allow to implement, something that has a direct relationship with their syntax.**

***Key Word* —** **Paradigm, programming, solving, implementing.**

1. INTRODUCCIÓN

Al hablar de paradigma, hablamos de conjunto de creencias, prácticas y conocimientos que guían el desarrollo de una disciplina durante un período de tiempo En diversas ramas de la ciencia, un conjunto de ideas en vigencia puede ser reemplazado drásticamente por otro que entre en conflicto con él y se demuestre más acertado. La programación tiene sus propios paradigmas, pero el término paradigma de programación no necesariamente representa un modelo único que deba ser respetado hasta que aparezca otro mejor. De hecho, actualmente muchos paradigmas coexisten en armonía.

1. CONTENIDO
2. **¿Qué es un paradigma de programación?**
3. **Programación funcional.**
4. **Programación estructurada.**
5. **Programación Modular.**
6. **Abstracción de datos.**
7. **Programación Orientada a Objetos.**

**¿Qué es un paradigma de programación?**

Un paradigma de programación consiste en un método para llevar a cabo cómputos y la forma en la que deben estructurarse y organizarse las tareas que debe realizar un programa.​ Se trata de una propuesta tecnológica adoptada por una comunidad de programadores, y desarrolladores cuyo núcleo central es incuestionable en cuanto 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. Representa un enfoque particular o filosofía para diseñar soluciones. Los paradigmas difieren unos de otros, en los conceptos y la forma de abstraer los elementos involucrados en un problema, así como en los pasos que integran su solución del problema, en otras palabras, el cómputo. Tiene una estrecha relación con la formalización de determinados lenguajes en su momento de definición. Es un estilo de programación empleado.

Un paradigma de programación está delimitado en el tiempo en cuanto a aceptación y uso, porque nuevos paradigmas aportan nuevas o mejores soluciones que lo sustituyen parcial o totalmente.

El paradigma de programación que actualmente es más utilizado es la "orientación a objetos" (OO). El núcleo central de este paradigma es la unión de datos y procesamiento en una entidad llamada "objeto", relacionable a su vez con otras entidades "objeto".

Tradicionalmente, datos y procesamiento se han separado en áreas diferente del diseño y la implementación de software. Esto provocó que grandes desarrollos tuvieran problemas de fiabilidad, mantenimiento, adaptación a los cambios y escalabilidad. Con la OO y características como el encapsulado, polimorfismo o la herencia, se permitió un avance significativo en el desarrollo de software a cualquier escala de producción. La OO parece estar ligada en sus orígenes con lenguajes como Lisp y Simula, aunque el primero que acuñó el título de "programación orientada a objetos" fue Smalltalk.

A continuación, hablaremos de 5 clases de paradigmas de programación.

**PROGRAMACIÓN FUNCIONAL**

En informática, la programación funcional es un paradigma de programación declarativa basado en el uso de funciones matemáticas, en contraste con la programación imperativa, que enfatiza los cambios de estado mediante la mutación de variables.1 La programación funcional 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. Muchos lenguajes de programación funcionales pueden ser vistos como elaboraciones del cálculo lambda.

En la práctica, la diferencia entre una función matemática y la noción de una "función" utilizada en la programación imperativa, es que las funciones imperativas pueden tener efectos secundarios, como cambiar el valor de cálculos realizados previamente. Por esta razón, carecen de transparencia referencial; es decir, la misma expresión sintáctica puede resultar en valores diferentes en varios momentos de la ejecución del programa. Con código funcional, en contraste, el valor generado por una función depende exclusivamente de los argumentos alimentados a la función. Al eliminar los efectos secundarios se puede entender y predecir el comportamiento de un programa mucho más fácilmente. Esta es una de las principales motivaciones para utilizar la programación funcional.

Los lenguajes de programación funcional, especialmente los puramente funcionales, han sido enfatizados en el ambiente académico y no tanto en el desarrollo comercial o industrial. Sin embargo, lenguajes de programación funcional como Scheme, Erlang, Rust, Objective Caml, Scala, F# y Haskell, han sido utilizados en aplicaciones comerciales e industriales por muchas organizaciones. La programación funcional también es utilizada en la industria a través de lenguajes de dominio específico como R (estadística), Mathematica (matemáticas simbólicas), J y K (análisis financiero).

**Características.**

* Los programas escritos en un lenguaje funcional están constituidos únicamente por definiciones de funciones
* La no existencia de asignaciones de variables y la falta de construcciones estructuradas como la secuencia o la iteración
* Existen dos grandes categorías de lenguajes funcionales: los funcionales puros y los híbridos
* En contraste, los lenguajes funcionales puros tienen una mayor potencia expresiva, conservando a la vez su transparencia referencial.

**Ejemplo:**

boolean found = false;

for(String city: cities){

if(city.equals("Chicago")){

found = true;

break;

}

}

System.out.println("Found chicago?:" + found);

**Ventajas.**

* Ausencia de efectos colaterales
* Proceso de depuración menos problemático
* Pruebas de unidades más confiables
* Mayor facilidad para la ejecución concurrente

**Desventajas.**

* Falta de estandarización
* Bajo rendimiento de los programas.

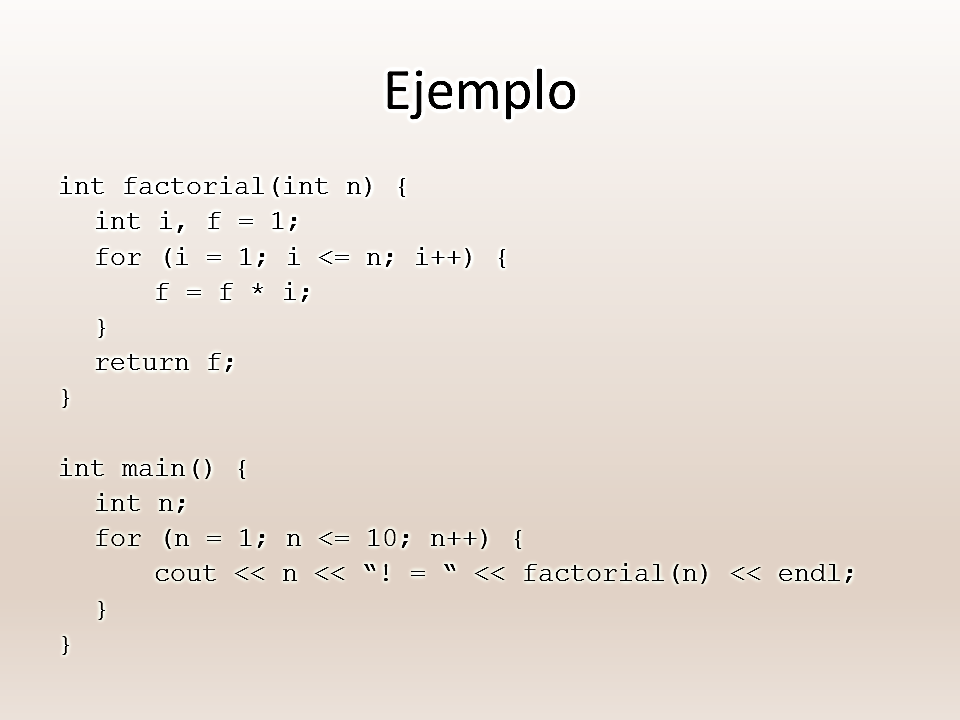
**PROGRAMACIÓN ESTRUCTURADA.**

La programación estructurada es un paradigma de programación orientado a mejorar la claridad, calidad y tiempo de desarrollo de un programa de computadora recurriendo únicamente a subrutinas y tres estructuras básicas: secuencia, selección (if y switch) e iteración (bucles for y while); asimismo, se considera innecesario y contraproducente el uso de la instrucción de transferencia incondicional (GOTO), que podría conducir a código espagueti, mucho más difícil de seguir y de mantener, y fuente de numerosos errores de programación.

Surgió en la década de 1960, particularmente del trabajo de Böhm y Jacopini,​ y un famoso escrito de 1968: «La sentencia goto, considerada perjudicial», de Edsger Dijkstra.​ Sus postulados se verían reforzados, a nivel teórico, por el teorema del programa estructurado y, a nivel práctico, por la aparición de lenguajes como ALGOL, dotado de estructuras de control consistentes y bien formadas.

**Características.**

* La estructura de los programas es clara, puesto que las instrucciones están más ligadas o relacionadas entre sí.
* Los programas son más fáciles de entender, pueden ser leídos de forma secuencial.
* Un programa escrito de acuerdo a los principios de programación estructurada no solamente tendrá una mejor estructura sino también una excelente presentación.



**Ventajas.**

* Reducción de los costos de mantenimiento.
* Reducción del esfuerzo en las pruebas y depuración.
* Los bloques de código son casi auto-explicativos, lo que reduce y facilita la documentación.
* Se incrementa el rendimiento de los programadores.

**Desventajas.**

* El principal inconveniente de este método de programación es que se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar problemático el manejo de su código fuente.
* Se obtiene un único bloque de programa, que cuando se hace demasiado grande puede resultar difícil su manejo.

**PROGRAMACIÓN MODULAR.**

La programación modular es un paradigma de programación que consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.

Se presenta históricamente como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos de lo que esta puede resolver.

Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples. Esto debe hacerse hasta obtener subproblemas lo suficientemente simples como para poder ser resueltos fácilmente con algún lenguaje de programación. Esta técnica se llama refinamiento sucesivo, divide y vencerás o análisis desce­ndente (Top-Down).

Un 'módulo' es cada una de las partes de un programa que resuelve uno de los subproblemas en que se divide el problema complejo original. Cada uno de estos módulos tiene una tarea bien definida y algunos necesitan de otros para poder operar. En caso de que un módulo necesite de otro, puede comunicarse con éste mediante una interfaz de comunicación que también debe estar bien definida.

Si bien un módulo puede entenderse como una parte de un programa en cualquiera de sus formas y variados contextos, en la práctica se los suele tomar como sinónimos de procedimientos y funciones. Pero no necesaria ni estrictamente un módulo es una función o un procedimiento, ya que el mismo puede contener muchos de ellos. No debe confundirse el término "módulo" (en el sentido de programación modular) con términos como "función" o "procedimiento", propios del lenguaje que lo soporte.

**Características.**

* Este paradigma también se conoce como principio de ocultación de procedimientos y datos
* Consiste en dividir un programa en módulos o subprogramas con el fin de hacerlo más legible y manejable.
* Se presenta históricamente como una evolución de la programación estructurada para solucionar problemas de programación más grandes y complejos de lo que ésta puede resolver.

**Ejemplo.**

Prog15.cpp

#include <stdio.h>

#include <conio.h>

#include <string.h>

void proc1(int vector[]);

void main() {

clrscr();

//creando arreglo y cargandolo

int lista[5]= {10,11,12,13,14};

// mandandolo a procedimiento recordar como se manda sin []

proc1(lista);

// desplegando arreglo lista y observar que datos salen

for(int reng=0; reng<=4; reng++)

printf("%d \n",lista[reng]);

getchar();

} // termina main

void proc1(int vector[])

{

// sumandole 50 a lo que se tenia en arreglo lista

// es decir vector queda cargado con 60,61,62,63,64

for(int reng=0; reng<=4; reng++)

vector[reng]=vector[reng]+ 50;

// observar que no se regresa nada

} //termina proc1

**Ventajas.**

* Al aplicar la programación modular, un problema complejo debe ser dividido en varios subproblemas más simples, y estos a su vez en otros subproblemas más simples.
* En caso de que un módulo necesite de otro, puede comunicarse con éste mediante una interfaz de comunicación que también debe estar bien definida.
* Es fácil de mantener y modificar.
* Es más fácil de escribir y depurar.
* Facilidad de controlar es decir descompone un problema en estructuras jerárquicas, de modo que se puede considerar cada estructura desde dos puntos de vista.

**Desventajas.**

* No se dispone de algoritmos formales de modularidad, por lo que aveces los programadores no tienen claras las ideas de los módulos
* La programación modular requiere más memoria y tiempo de ejecución

**ABSTRACCIÓN DE DATOS.**

La abstracción consiste en aislar un elemento de su contexto o del resto de los elementos que lo acompañan. En programación, el término se refiere al énfasis en el "¿qué hace?" más que en el "¿cómo lo hace?" (característica de caja negra). El común denominador en la evolución de los lenguajes de programación, desde los clásicos o imperativos hasta los orientados a objetos, ha sido el nivel de abstracción del que cada uno de ellos hace uso.

Los lenguajes de programación son las herramientas mediante las cuales los diseñadores de lenguajes pueden implementar los modelos abstractos. La abstracción ofrecida por los lenguajes de programación se puede dividir en dos categorías: abstracción de datos (pertenecientes a los datos) y abstracción de control (perteneciente a las estructuras de control).

Los diferentes paradigmas de programación han aumentado su nivel de abstracción, comenzando desde los lenguajes de máquina, lo más próximo al ordenador y más lejano a la comprensión humana; pasando por los lenguajes de comandos, los imperativos, la orientación a objetos (POO), la Programación Orientada a Aspectos (POA); u otros paradigmas como la programación declarativa, etc.

La abstracción encarada desde el punto de vista de la programación orientada a objetos expresa las características esenciales de un objeto, las cuales distinguen al objeto de los demás. Además de distinguir entre los objetos provee límites conceptuales. Entonces se puede decir que la encapsulación separa las características esenciales de las no esenciales dentro de un objeto. Si un objeto tiene más características de las necesarias los mismos resultarán difíciles de usar, modificar, construir y comprender sobre todo cuando es un método de número entero con terminación fraccionaria.

La misma genera una ilusión de simplicidad dado que minimiza la cantidad de características que definen a un objeto.

Durante años, los programadores se han dedicado a construir aplicaciones muy parecidas que resolvían una y otra vez los mismos problemas. Para conseguir que sus esfuerzos pudiesen ser utilizados por otras personas se creó la POO que consiste en una serie de normas para garantizar la interoperabilidad entre usuarios de manera que el código se pueda reutilizar.

**Características.**

* Proporciona un conjunto completo de operaciones válidas y útiles para cada tipo de dato
* La abstracción de datos se logra mediante los tipos de datos que suministra el lenguaje y los subprogramas que van a implementar las operaciones permitidas
* Se encapsula la representación interna de un dato junto con las implementaciones de todas las operaciones que se pueden realizar con ese dato.

**Ejemplo.**

Pensar en términos de objetos es muy parecido a cómo lo haríamos en la vida real. Una analogía sería modelizar un auto en un esquema de POO. Diríamos que el auto es el elemento principal que tiene una serie de características, como podrían ser el color, el modelo o la marca.

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

Estos objetos son utilizables en los programas, por ejemplo en un programa de matemáticas se puede hacer uso de objetos fracción y en un programa que gestione un taller de autos, objetos auto. Los programas orientados a objetos utilizan muchos objetos para realizar las acciones que se desean realizar y ellos mismos también son objetos. Es decir, el taller de autos será un objeto que utilizará objetos auto, herramienta, mecánico, recambios, etc.

**Ventajas.**

* Produce código reutilizable
* Favorece la ausencia de errores, al reutilizar código ya probado y forzar a utilizar la estructura de datos correctamente
* Aumenta la facilidad de uso y la legibilidad del código
* Favorece la extensibilidad del código
* Facilita y hace más rápido el desarrollo de aplicaciones

**Desventajas.**

* Las operaciones requieren mayor tiempo de procesamiento
* En la mayoría no existen herramientas predefinidas, por lo tanto es el programador quien implementa las estructuras
* Al agregar un nuevo elemento en algún lugar que no sea el último, se tiene que desplazar los elementos hacia abajo que están después de la posición de inserción deseada logrando que se genere un espacio para poder agregar un nuevo elemento

**PROGRAMACIÓN ORIENTADA A OBJETOS.**

La programación orientada a objetos (POO, en español; OOP, según sus siglas en inglés) es un paradigma de programación que viene a innovar la forma de obtener resultados. Los objetos manipulan los datos de entrada para la obtención de datos de salida específicos, donde cada objeto ofrece una funcionalidad especial.

Muchos de los objetos prediseñados de los lenguajes de programación actuales permiten la agrupación en bibliotecas o librerías, sin embargo, muchos de estos lenguajes permiten al usuario la creación de sus propias bibliotecas.

Está basada en varias técnicas: herencia, cohesión, abstracción, polimorfismo, acoplamiento y encapsulamiento.

Su uso se popularizó a principios de la década de 1990. En la actualidad, existe una gran variedad de lenguajes de programación que soportan la orientación a objetos.

**Características.**

* Abstracción: denota las características esenciales de un objeto, donde se capturan sus comportamientos
* Encapsulamiento: significa reunir todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción
* Modularidad: propiedad que permite subdividir una aplicación en partes más pequeñas
* Polimorfismo: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre; al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando.

**Ejemplo.**

num\_attempts = 0

number = rand(1..10)

found = false

until found

print "Adivina el número de 1 a 10 que estoy pensando: "

guess = gets.chomp.to\_i

if guess == number

puts "Muy bien! Lo lograste en #{num\_attemps} intentos!"

found = true

else

puts "Lo siento! No es el número, intenta nuevamente."

num\_attempts += 1

end

end

**Ventajas.**

* Permite crear sistemas más complejos
* Agiliza el desarrollo de software
* Proporciona conceptos y herramientas con las cuales se modela y representa el mundo real tan fielmente como sea posible.
* Fomenta la reutilización y extensión del código.

**Desventajas.**

* Complejidad para adaptarse
* Mayor cantidad de código

1. CONCLUSIONES

En conclusión. Cuando se habla de paradigmas de programación estamos hablando de los distintos lenguajes que se usan a la hora de programar en computadora.

RECOMENDACIONES

Se recomienda investigar más a fondo los temas debido a que en este documento solo hay una breve explicación de los distintos lenguajes de programación, lo cual sirve como base introductoria, si quieres comenzar a programar se debe investigar más a fondo el tema de interés.

REFERENCIAS

* <https://www.monografias.com/trabajos108/funciones-y-programacion-estructurada/Diapositiva3.png>
* <https://makeitrealcamp.gitbook.io/programacion-orientada-por-objetos-en-ruby/ejemplo-de-poo>
* <https://es.wikipedia.org/wiki/Programaci%C3%B3n_orientada_a_objetos>
* <https://es.wikipedia.org/wiki/Abstracci%C3%B3n_(inform%C3%A1tica)>
* <http://argetephay.blogspot.com/>
* <https://es.wikipedia.org/wiki/Programaci%C3%B3n_modular>
* <https://www.academia.edu/15737185/Ejemplos_de_Programacion_Estructurada>
* <https://es.wikipedia.org/wiki/Programaci%C3%B3n_estructurada>
* <http://www.clubdetecnologia.net/blog/2014/programacion-funcional-ejemplos/>
* <https://es.wikipedia.org/wiki/Programaci%C3%B3n_funcional>
* <http://desarrollo--software.blogspot.com/2013/01/caracteristicas-de-los-paradigmas-de.html>