**Taller 1: Paradigmas de Programación**

Dahiana Bran Agudelo

Nataly Liévano Arroyo

Elio Esteban Mercado Londoño

Juan Diego Portillo

Lukas Piedrahita Serna

Pablo Serna Valencia

Facultad de Ingeniería, Universidad de Antioquia

Técnicas de Programación y Laboratorio

Profesora Diana Margot López Herrera

Febrero 16, 2024

**Taller 1: Paradigmas de Programación**

Los paradigmas de programación son las diferentes formas o métodos para resolver problemas computacionales. Cada paradigma tiene sus propias reglas, conceptos y técnicas que guían la estructura del código, proporcionando una nueva forma de pensar sobre la organización de un programa de computadora.

Los paradigmas se diferencian en:

**Imperativo**

La programación imperativa consiste en detallar, paso por paso todo lo que realizará el programa. Se centra en describir cómo se debe lograr un resultado específico. Utiliza variables, asignaciones y estructuras de control como bucles y condicionales para modificar el estado del programa.

**Declarativo**

Se enfoca en describir qué resultado se desea obtener, sin especificar cómo obtenerlo. No se detallan los pasos a seguir sino el resultado deseado. Uno de sus subtipos se dará a continuación:

**PARADIGMA FUNCIONAL**

La programación funcional es un paradigma de programación que trata la computación como una evaluación de funciones matemáticas y evita los cambios de estado y la mutabilidad de los datos. En la programación funcional, las funciones son tratadas como ciudadanos de primera clase, lo que significa que pueden ser asignadas a variables, pasadas como argumentos y devueltas como resultados de otras funciones.

La programación funcional es útil en problemas en los que se requiere escalabilidad, concurrencia, y en general para la creación de sistemas robustos y confiables. Es muy utilizada en el desarrollo de aplicaciones distribuidas y en la inteligencia artificial.

Algunos beneficios de la programación funcional:

* Permite escribir código más comprensible y comprobable, ya que se enfoca en el uso de funciones matemáticas puras, que tienen comportamiento predecible.
* Al evitar el cambio de estado y los efectos secundarios, se pueden prevenir errores comunes en programas imperativos.
* Al tratar las funciones como valores, se pueden pasar como argumentos a otras funciones, lo que permite construir programas modulares, compuestos con facilidad.
* Algunas técnicas funcionales, como la programación lógica y la programación de orden superior, permiten expresar soluciones de una manera más concisa y natural.

Desventajas:

* Puede ser más difícil de aprender y entender para los programadores acostumbrados a otros paradigmas de programación, como la programación imperativa o la orientada a objetos.
* Puede ser más compleja de depurar, ya que la falta de efectos secundarios y el cambio de estado pueden dificultar el rastreo de las causas de errores.
* Posiblemente más difícil de escalar en problemas que necesitan un gran uso de recursos, debido a la inmutabilidad de los datos y la ausencia de variables globales.
* Puede ocasionar un menor rendimiento en ciertos casos, por la gran cantidad de objetos y copias creadas en algunas aplicaciones.

Estas desventajas son menos significativas en las aplicaciones modernas, a causa de la optimización de compiladores y lenguajes funcionales.

***CONCEPTOS IMPORTANTES***

**¿Qué es una función?**

En programación, una función es un conjunto de líneas de código que realizan una tarea en concreto cuando es llamada o invocada. Es la relación entre unos valores de entrada y de salida. La idea fundamental detrás de las funciones es la modularidad, es decir, dividir un programa en partes más pequeñas y manejables que realicen tareas específicas.

***Funciones de alto orden:***

El concepto "alto orden" se refiere a funciones que reciben como argumento o retoman funciones, o sea, que pueden manipulase como datos; esta característica se llama "funciones como objetos de primera clase". Este concepto es fundamental en la programación funcional, ya que permite la composición de funciones y la abstracción de patrones comunes.

***Funciones puras:***

Una función pura es aquella que, dado el mismo conjunto de entradas, siempre produce el mismo resultado, es decir que debe ser **determinista** y no tiene efectos secundarios o colaterales. Esto significa que no modifica ningún estado externo ni depende de él.

La salida depende únicamente de:

* Los parámetros de entrada
* El algoritmo interno

***Recursión:***

La recursión se utiliza en lugar de bucles para repetir una serie de instrucciones. Las funciones recursivas se llaman a sí mismas con argumentos modificados hasta que se cumple una condición de salida. En algunos casos, el uso de una función recursiva puede hacer que el código sea más simple y fácil de entender en comparación con un enfoque iterativo. Por ejemplo, algoritmos como el cálculo del factorial o la suma de los elementos de una lista son naturalmente expresados de forma recursiva:

El factorial de un número entero no negativo *n* (denotado como *n!*) se define como el producto de todos los enteros positivos menores o iguales a *n*. Se puede expresar de la siguiente manera en Python:

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n - 1)

Esta función “factorial” calcula el factorial de un número *n* utilizando la definición recursiva. Cuando se llama con un valor de *n* igual a 0, la función devuelve 1 (por definición). Para valores de *n* mayores que 0, la función se llama a sí misma con un argumento n - 1 y multiplica el resultado por *n*.

***Transparencia referencial:***

Una expresión es referencialmente transparente si puede ser reemplazada por su valor sin cambiar el comportamiento del programa. Esto se deriva de la propiedad de las funciones puras. Esto permite probar matemáticamente la corrección de un programa. La posibilidad de escribir programas cuya corrección es probable en vez de gastar el tiempo pescando errores puede revolucionar el proceso de producción de software.

***Composición de funciones:***

Proceso de combinar dos o más funciones con el fin de ejecutar cada una de ellas en secuencia para obtener un resultado en concreto.

Muchos lenguajes de programación modernos admiten características de programación funcional o están diseñados principalmente con paradigmas funcionales en mente. Algunos de los lenguajes más populares que soportan la programación funcional incluyen:

***Haskell:*** Es un lenguaje funcional puro que se basa en el cálculo lambda. Es conocido por su sistema de tipos fuertemente tipado y su énfasis en la inmutabilidad y la programación declarativa.

***Clojure:*** Es un dialecto de Lisp que se ejecuta en la plataforma Java Virtual Machine (JVM). Combina la programación funcional con la programación orientada a objetos y está diseñado para ser simple y expresivo.

***Scala:*** Es un lenguaje de programación multiparadigma que combina la programación orientada a objetos con la programación funcional. Se ejecuta en la JVM y es compatible con Java.

***Erlang:*** Es un lenguaje funcional diseñado para construir sistemas concurrentes y distribuidos altamente escalables y fiables. Es conocido por su modelo de concurrencia basado en actores.

***F#:*** Es un lenguaje funcional de propósito general que se ejecuta en la plataforma .NET. Combina la programación funcional con el paradigma orientado a objetos y es compatible con el desarrollo de aplicaciones web, móviles y de escritorio.

***Scheme:*** Es un dialecto de Lisp que se centra en la simplicidad y la elegancia del código. Es conocido por su sistema de macros y su énfasis en la programación funcional.

***Elixir:*** Es un lenguaje funcional diseñado para construir aplicaciones escalables y tolerantes a fallos en la plataforma Erlang VM. Se basa en la programación concurrente y está diseñado para ser fácil de aprender y usar.

**Efecto de Lado /Colateral:**

Hay efecto cuando un cambio de estado sobrevive a la realización de una operación. Por ejemplo, una operación puede modificar una variable global, modificar uno de sus argumentos, escribir datos a la pantalla o a un archivo, o hacer uso de otras operaciones que tienen efecto de lado.

Otra definición válida es: Si le sacas una foto al sistema (llamémosla F1), después realizas la operación de tu interés, y le vuelves a sacar una foto al sistema (F2). Si F1 y F2 son distintas => la operación que hiciste tiene efecto de lado.

Características cuando hay efectos colaterales:

* Laza una excepción
* La operación se detiene por un error
* Modifica el valor de una variable o de una estructura de datos mutable.

***PARADIGMA FUNCIONAL JUNTO A SCALA***

***Scala:*** Es un lenguaje de programación que combina la programación funcional y la orientada a objetos en un lenguaje conciso.

***Popularidad de Scala:*** En los últimos años, muchas empresas, incluyendo gigantes tecnológicos como Twitter y LinkedIn, han adoptado Scala1. En 2020, Scala entró en el top 10 de los lenguajes que los desarrolladores quieren aprender.

***Ventajas de Scala:*** Scala se ejecuta en una máquina virtual Java (JVM), proporciona interoperabilidad con Java, Groovy, Clojure, etc1. Además, es más fácil de depurar y mantener que muchos otros lenguajes de programación.

***Se especializa en:*** Procesamiento de datos en entornos de Big Data, aplicaciones web, desarrollo de aplicaciones de escritorio, desarrollo de sistemas escalables (hechos para albergar una mayor capacidad con el tiempo).

**REFERENCIAS**

EDTeam. (2020). ¿Qué son los paradigmas de programación? <https://youtu.be/hcuvB58hwlE?si=kmJ7JIUuZI8WpYF7>

Kiko Palomares. (2019). ¿Qué es una función en programación? <https://kikopalomares.com/clases/que-es-una-funcion-en-programacion>

Temática Informática. (2020). Paradigma de Programación Funcional. <https://www.youtube.com/watch?v=k5XARecIDqE>

MyTaskPanel Consulting. (2018). Introducción a la programación funcional: beneficios para el desarrollo de software. <https://www.mytaskpanel.com/programacion-funcional-beneficios/>

Uqbar wiki. (2017). Transparencia referencial efecto de lado y asignacion destructiva. <https://wiki.uqbar.org/wiki/articles/transparencia-referencial--efecto-de-lado-y-asignacion-destructiva.html>

Lucas Spigariol. (2020). Paradigma funcional - Transparencia referencial. <https://www.youtube.com/watch?v=3NOUWzqMWn8>