ÁREA 3. DESARROLLO DE SOFTWARE DE APLICACIÓN

SUBÁREA 3.2 LENGUAJES DE PROGRAMACIÓN

TEMAS IMPORTANTE:

Paradigmas de Programación:

Los paradigmas de programación son un conjunto de conceptos y metodologías que guían la forma en que se diseña y desarrolla un programa. Los paradigmas de programación se pueden clasificar en varios tipos:

Programación imperativa: Este paradigma se centra en cómo se realiza una tarea, paso a paso. Se utiliza una secuencia de instrucciones que se ejecutan en un orden específico para cambiar el estado del programa.

Programación orientada a objetos: Este paradigma se centra en los objetos como entidades que pueden tener propiedades y comportamientos. Los objetos interactúan entre sí mediante el intercambio de mensajes.

Programación funcional: Este paradigma se centra en la evaluación de funciones matemáticas. En la programación funcional, las funciones son tratadas como valores y pueden ser pasadas como argumentos a otras funciones.

Programación lógica: Este paradigma se centra en la lógica y la inferencia. Se utilizan reglas y hechos para deducir conclusiones a partir de premisas.

Programación basada en eventos: Este paradigma se centra en la interacción del programa con el entorno mediante la gestión de eventos. Se utilizan callbacks o disparadores para responder a eventos específicos.

Programación concurrente: Este paradigma se centra en la ejecución de múltiples tareas simultáneamente. Se utilizan hilos de ejecución y sincronización para garantizar que las tareas se ejecuten correctamente.

Programación Orientada a Objetos:

La Programación Orientada a Objetos (POO) es un paradigma de programación que se centra en los objetos como entidades clave en la resolución de problemas. En la POO, se modelan los objetos que forman parte del sistema que se está construyendo, y se definen sus propiedades y comportamientos.

Los objetos en la POO son instancias de clases, que son estructuras de datos que definen las propiedades y comportamientos comunes a todos los objetos de esa clase. Las propiedades de un objeto pueden ser valores simples, como números o cadenas de texto, o incluso otros objetos. Los comportamientos de un objeto se definen mediante métodos, que son funciones que pueden acceder y modificar las propiedades del objeto, así como interactuar con otros objetos.

La POO se basa en cuatro conceptos principales: encapsulación, abstracción, herencia y polimorfismo.

* Encapsulación: La encapsulación es el proceso de ocultar los detalles internos de un objeto y exponer solo la interfaz pública. Esto significa que las propiedades y métodos que no son necesarios para el uso del objeto están ocultos, lo que aumenta la seguridad y la modularidad del código.
* Abstracción: La abstracción es el proceso de identificar las características esenciales de un objeto y crear una clase que las represente. En la abstracción, se define la estructura básica de los objetos de una clase, sin entrar en detalles específicos de cada objeto.
* Herencia: La herencia es el proceso de crear una nueva clase a partir de una clase existente. La clase hija hereda todas las propiedades y métodos de la clase padre, y puede agregar nuevos métodos y propiedades propios.
* Polimorfismo: El polimorfismo es la capacidad de los objetos de diferentes clases para responder a un mismo mensaje o llamada de método de forma distinta. Esto permite crear objetos de diferentes clases que comparten una interfaz común, lo que aumenta la flexibilidad y la modularidad del código.

La Programación Orientada a Objetos (POO) es un paradigma de programación que se basa en el concepto de objetos, que son entidades que tienen propiedades y métodos. A continuación, se explican cada uno de estos conceptos clave en la POO:

* Objetos: Los objetos son instancias de una clase, que es una plantilla o molde que define las propiedades y métodos comunes a todos los objetos de ese tipo. Los objetos tienen un estado, que está determinado por sus propiedades, y un comportamiento, que está determinado por sus métodos.
* Métodos: Los métodos son funciones que se definen dentro de una clase y que pueden ser llamados por los objetos de esa clase. Los métodos pueden acceder y modificar las propiedades de un objeto, y también pueden interactuar con otros objetos.
* Encapsulamiento: El encapsulamiento es el proceso de ocultar los detalles internos de un objeto y exponer solo la interfaz pública. Esto significa que las propiedades y métodos que no son necesarios para el uso del objeto están ocultos, lo que aumenta la seguridad y la modularidad del código.
* Herencia: La herencia es el proceso de crear una nueva clase a partir de una clase existente. La clase hija hereda todas las propiedades y métodos de la clase padre, y puede agregar nuevos métodos y propiedades propios. La herencia permite la reutilización de código y la creación de jerarquías de clases.
* Polimorfismo: El polimorfismo es la capacidad de los objetos de diferentes clases para responder a un mismo mensaje o llamada de método de forma distinta. Esto permite crear objetos de diferentes clases que comparten una interfaz común, lo que aumenta la flexibilidad y la modularidad del código.
* Excepciones: Las excepciones son errores que ocurren durante la ejecución del programa y que interrumpen su flujo normal. La POO ofrece un mecanismo de manejo de excepciones que permite capturar y manejar los errores de manera controlada, evitando que el programa se detenga abruptamente.

Programación Funcional:

La Programación Funcional es un paradigma de programación que se basa en el uso de funciones para resolver problemas. En la programación funcional, las funciones son tratadas como valores de primera clase y se utilizan para manipular datos de una forma declarativa, en lugar de utilizar una serie de instrucciones imperativas.

A continuación, se explican algunos de los conceptos clave de la programación funcional:

* Funciones: En la programación funcional, las funciones son entidades fundamentales que se utilizan para transformar datos. Las funciones se definen de tal manera que se pueda llamar a la misma función varias veces con diferentes argumentos, y siempre produzca el mismo resultado.
* Inmutabilidad: En la programación funcional, los datos se tratan como valores inmutables, lo que significa que no se pueden modificar una vez que se han creado. Esto permite evitar problemas de concurrencia y mejora la legibilidad y mantenibilidad del código.
* Composición de funciones: La programación funcional permite la composición de funciones, es decir, combinar varias funciones para crear una nueva función. Esto permite crear funciones más complejas a partir de funciones más simples, lo que aumenta la modularidad y reutilización del código.
* Funciones de orden superior: Las funciones de orden superior son funciones que pueden tomar otras funciones como argumentos o devolver funciones como resultado. Esto permite crear funciones más genéricas y reutilizables.
* Evaluación diferida: En la programación funcional, la evaluación de una función se realiza de forma diferida, es decir, se evalúa solo cuando se necesita el resultado. Esto permite escribir código más eficiente y evitar cálculos innecesarios.

La programación funcional se utiliza en muchos lenguajes de programación, como Haskell, Lisp, Clojure y Scala. La programación funcional ofrece un enfoque diferente a la programación imperativa, y permite escribir código más conciso, legible y mantenible, especialmente en aplicaciones que requieren procesamiento de grandes cantidades de datos.

Programación Web y Móvil: