1) Explicar la ventaja de escribir un programa usando Programacion Funcional, con respecto al Paradigma Imperativo.

Las ventajas de la programación funcional en comparación con la programación imperativa:

Transparencia referencial: Las funciones siempre devuelven el mismo resultado para los mismos argumentos, lo que facilita el razonamiento y la prueba del código.

Menos efectos secundarios: Las funciones no modifican el estado externo, reduciendo el riesgo de errores y mejorando la claridad.

Facilidad de paralelización: La ausencia de estado compartido permite ejecutar funciones en paralelo de manera más sencilla.

Abstracción y reutilización: Las funciones de alto orden permiten crear abstracciones más potentes y reutilizar código de manera efectiva.

2) Describir la forma en la que Programacion Funcional define los tipos de datos

En programación funcional, los tipos de datos se definen y organizan de manera que reflejan su estructura y comportamiento. Aquí hay algunas características clave:

Tipos Algebraicos: La programación funcional utiliza tipos algebraicos, como los tipos sumas y productos, para combinar diferentes tipos de datos. Por ejemplo, un tipo "Producto" puede combinar múltiples tipos (como un `par` de números), mientras que un tipo "Suma" puede representar múltiples opciones (como un tipo `Opción` que puede ser `Nada` o `Algo`).

Inmutabilidad: Los datos son generalmente inmutables, lo que significa que una vez creados, no se pueden modificar. Esto promueve la seguridad en el manejo de datos y evita efectos secundarios indeseados.

Funciones como Ciudadanos de Primera Clase: En muchos lenguajes funcionales, las funciones son tratadas como tipos de datos. Esto permite que las funciones se pasen como argumentos, se retornen como resultados y se almacenen en estructuras de datos.

Patrones de Coincidencia: La programación funcional utiliza patrones de coincidencia para descomponer y trabajar con tipos de datos complejos. Esto permite extraer valores de estructuras de datos de manera clara y concisa.

Tipado Estático y Dinámico: Dependiendo del lenguaje, se pueden utilizar sistemas de tipos estáticos (como Haskell) o dinámicos (como JavaScript). En ambos casos, la verificación de tipos ayuda a prevenir errores en tiempo de compilación o ejecución.

Estos enfoques permiten una manipulación más clara y segura de los datos, facilitando la creación de programas más robustos y mantenibles.

3) Para que se usa el head y tail. Ejemplificar.

En programación funcional, especialmente en lenguajes como Haskell, `head` y `tail` son funciones utilizadas para operar sobre listas:

`head`: Devuelve el primer elemento de una lista.

`tail`: Devuelve todos los elementos de la lista excepto el primero.

Ejemplo:

Dada la lista `lista = [1, 2, 3, 4]`:

`head lista` devolverá `1`.

`tail lista` devolverá `[2, 3, 4]`.

Esto es útil para descomponer listas y realizar operaciones recursivas.

4) Que caracteristicas distingue al Paradigma logico con respecto al resto de los Paradigmas?

El paradigma lógico se distingue de otros paradigmas por las siguientes características:

Declaratividad: En lugar de especificar cómo realizar tareas (como en el paradigma imperativo), se centra en describir qué resultados se desean, expresando relaciones y hechos.

Uso de reglas y hechos: Utiliza una base de conocimientos compuesta por hechos y reglas lógicas. Los programas se construyen como conjuntos de estas afirmaciones.

Resolución de consultas: Permite hacer consultas sobre la base de conocimientos, utilizando un motor de inferencia para deducir respuestas a partir de los hechos y reglas.

No secuencial: No se basa en una secuencia de comandos para el flujo del programa, lo que puede facilitar la paralelización.

Programación simbólica: Se enfoca en el uso de símbolos y estructuras lógicas en lugar de manipulación directa de datos.

Un ejemplo representativo de este paradigma es Prolog, que se basa en la lógica de predicados.

5) Dar un ejemplo de un hecho y una regla con variables y constantes.

Aquí tienes un ejemplo de un hecho y una regla en un contexto lógico:

Hecho: `padre(juan, maria).`

Esto significa que "Juan es padre de María", donde "juan" es una constante y "maria" es otra constante.

Regla: `abuelo(X, Y) :- padre(X, Z), padre(Z, Y).`

Esta regla establece que "X es abuelo de Y si X es padre de Z y Z es padre de Y". Aquí, "X" y "Y" son variables, mientras que "Z" es una constante que puede tomar el valor de cualquier padre.

Ejemplo de uso:

Si se tiene el hecho `padre(jose, juan).` (José es padre de Juan), y se usa la regla, podemos inferir que "José es abuelo de María".

6) En que situaciones el motor de inferencias responde Yes? Ejemplificar.

El motor de inferencias responde **Yes** (o **verdadero**) en las siguientes situaciones:

Hechos Directos: Cuando se consulta un hecho que ya está explícitamente declarado en la base de datos.
Ejemplo: `padre(juan, maria). ?- padre(juan, maria).` Respuesta: **Yes**.

Reglas Verificadas: Cuando una consulta coincide con una regla cuyas condiciones son todas verdaderas.
Ejemplo: `abuelo(X, Y) :- padre(X, Z), padre(Z, Y). padre(juan, maria). padre(maria, pedro). ?- abuelo(juan, pedro).` Respuesta: **Yes**.

Inferencias Válidas: Cuando el motor de inferencias deduce la verdad de un enunciado a partir de hechos y reglas.
Ejemplo: `hermano(X, Y) :- padre(Z, X), padre(Z, Y). padre(juan, maria). padre(juan, pedro). ?- hermano(maria, pedro).` Respuesta: **Yes**.

En resumen, el motor de inferencias responde **Yes** cuando se puede verificar un hecho directamente o cuando se satisfacen todas las condiciones de una regla.

7) Clasificar los distintos tipos de herencia en la Programacion orientada a objetos.

Herencia: Es la capacidad de crear nuevas clases a partir de otras ya existentes, ampliando su estructura y comportamiento. Las clases están organizadas en jerarquías de clases, las subclases heredan estado y comportamiento de las superclases. Las subclases pueden agregar nuevos atributos y comportamiento y además pueden cambiar el comportamiento de los métodos heredados.

Clasificación de herencia: • De estado: se heredan solo los atributos de la superclase.

• De comportamiento: se heredan solo los métodos de la superclase.

• Total o parcial: se heredan todos los atributos y métodos, o solo parte de ellos.

• Simple o múltiple: se hereda de una sola superclase (tiene 1 clase padre), o se hereda de varias superclases (+ de una clase padre).

8) Que es el polimorfismo? Ejemplificar.

El polimorfismo es un concepto de la programación orientada a objetos que permite que una misma operación se comporte de diferentes maneras según el objeto que la invoque. Esto se logra a través de la sobrecarga de métodos y la redefinición de métodos en clases derivadas.

Ejemplo: Supongamos que tenemos una clase base llamada `Animal` con un método `hacerSonido()`. Las clases derivadas `Perro` y `Gato` pueden implementar este método de la siguiente manera:

Perro: `hacerSonido()` devuelve "Guau".

Gato: `hacerSonido()` devuelve "Miau".

Cuando se llama a `hacerSonido()` en un objeto de tipo `Animal`, el resultado varía dependiendo si es un `Perro` o un `Gato`.

9) Que diferencia hay entre un metodo y un mensaje? Mencionar los tipos de metodos que existen.

La diferencia entre un método y un mensaje en programación orientada a objetos es:

Método: Es una función o procedimiento asociado a una clase que define el comportamiento de los objetos de esa clase. Es una implementación específica de una acción que puede ser invocada.

Mensaje: Es la forma en que un objeto solicita a otro objeto que ejecute un método. En este sentido, enviar un mensaje a un objeto es invocar un método de ese objeto.

Tipos de Metodos: Métodos de instancia: Operan sobre instancias específicas de una clase y pueden acceder a los atributos de esa instancia.

Métodos de clase: Se aplican a la clase en sí y no requieren una instancia; suelen ser utilizados para operaciones que afectan a la clase completa.

Métodos estáticos: Similar a los métodos de clase, pero se asocian a la clase en el contexto de ciertos lenguajes de programación; no pueden acceder a los atributos de instancia.

Métodos abstractos: Declarados en una clase base sin implementación; deben ser implementados en las clases derivadas.

Métodos sobrecargados: Tienen el mismo nombre pero diferentes parámetros, permitiendo múltiples definiciones de un método.

10) Como se ordena una OrderedCollecton? Ejemplificar.

Para ordenar una `OrderedCollection` en Smalltalk, se puede utilizar el método `sort:`. Este método requiere un bloque que define la comparación entre elementos.

Ejemplo

```
| collection sortedCollection | collection := OrderedCollection with: 5 with: 2 with: 8 with: 1.
sortedCollection := collection sort: [:a :b | a < b].
```

En este ejemplo, `collection` contiene los números 5, 2, 8 y 1. La llamada a `sort:` ordena los elementos de menor a mayor, y `sortedCollection` contendrá los valores ordenados: 1, 2, 5, 8.