Práctica 10

Ejercicio 1: Un programa en un lenguaje procedural es una secuencia de instrucciones o comandos que se van ejecutando y producen cambios en las celdas de memoria, a través de las sentencias de asignación. ¿Qué es un programa escrito en un lenguaje funcional? y ¿Qué rol cumple la computadora?

Un programa escrito en un lenguaje funcional se basa en la evaluación de funciones y expresiones sin modificar directamente el estado de las celdas de memoria. La computadora desempeña el papel de evaluar y simplificar las expresiones, permitiendo resolver problemas de manera eficiente y concisa.

Ejercicio 2: ¿Cómo se define el lugar donde se definen las funciones en un lenguaje funcional?

En un lenguaje funcional, las funciones se definen generalmente en el ámbito global o en el ámbito local de otras funciones. El lugar donde se definen las funciones puede variar dependiendo del lenguaje y del estilo de programación funcional utilizado. A continuación, se describen algunos escenarios comunes:

1. Definición en módulos o archivos: Muchos lenguajes funcionales permiten organizar las funciones en módulos o archivos separados. Cada módulo puede contener un conjunto de funciones relacionadas que se importan en otros módulos cuando sea necesario. Esto ayuda a mantener un código modular y facilita la reutilización de funciones en diferentes partes del programa.

2. Definición en el ámbito global: En muchos lenguajes funcionales, las funciones pueden ser definidas directamente en el ámbito global del programa. Esto significa que las funciones son accesibles desde cualquier parte del programa. Pueden ser invocadas y utilizadas en cualquier momento. Estas funciones globales pueden tener nombres únicos y pueden ser reutilizadas en múltiples partes del programa.

3. Definición local dentro de una función: En muchos lenguajes funcionales, es posible definir funciones dentro del ámbito local de otra función. Estas funciones se conocen como funciones locales o funciones internas. Estas funciones solo son accesibles dentro de la función que las contiene y se utilizan para realizar cálculos específicos o proporcionar una lógica auxiliar para la función principal. Las funciones locales pueden capturar y acceder a las variables definidas en el ámbito de la función que las contiene.

4. Definición mediante expresiones lambda: En algunos lenguajes funcionales, como Lisp o Haskell, las funciones pueden ser definidas mediante expresiones lambda. Una expresión lambda es una función anónima que no requiere un nombre explícito. Estas funciones se definen en línea y se utilizan directamente en el contexto donde son necesarias. Las expresiones lambda son especialmente útiles cuando se requiere una

función simple y no es necesario nombrarla o reutilizarla en otros lugares del programa.

Ejercicio 3: ¿Cuál es el concepto de variables en los lenguajes funcionales?

La noción de variable es la de “variable matemática”, no la de celda de memoria. Las variables son inmutables y no pueden ser modificadas una vez asignado un valor. Esto promueve la seguridad, la concurrencia y facilita el razonamiento y la comprensión del código.

Ejercicio 4: ¿Qué es una expresión en un lenguaje funcional? ¿Su valor de qué depende?

Una expresión es una combinación de valores, variables y funciones que se evalúa para producir un resultado. El valor de una expresión depende únicamente de los valores de las subexpresiones que la componen. Una expresión siempre produce el mismo resultado cuando se evalúa con los mismos valores de entrada, sin importar cuándo o dónde se evalúe.

Ejercicio 5: ¿Cuál es la forma de evaluación que utilizan los lenguajes funcionales?

Orden aplicativo: Aunque no lo necesite siempre evalúa los argumentos. Evalúa las expresiones de inmediato, es decir, tan pronto como las expresiones se encuentran en el flujo de control del programa. Calcula los valores de las expresiones antes de utilizarlos en el programa. Esto significa que los argumentos de las funciones se evalúan antes de que se llame a la función y los resultados se pasan a la función.

Orden normal: No calcula más de lo necesario. En lugar de evaluar una expresión inmediatamente, retrasa la evaluación hasta que el resultado sea necesario en otra parte del programa. Esto permite evitar la evaluación innecesaria de expresiones y mejora la eficiencia en ciertos casos. Una expresión compartida NO es evaluada más de una vez.

Ejercicio 6: ¿Un lenguaje funcional es fuertemente tipado? ¿Qué tipos existen? ¿Por qué?

Sí, en general, los lenguajes funcionales son considerados fuertemente tipados. Esto significa que los lenguajes funcionales imponen restricciones más estrictas en las operaciones y conversiones entre diferentes tipos de datos, y requieren una correspondencia precisa de tipos para realizar operaciones.

En los lenguajes funcionales, los tipos de datos son importantes y se verifican en tiempo de compilación para garantizar la coherencia y la seguridad del programa. Los tipos ayudan a prevenir errores comunes, como la aplicación de operaciones incorrectas o la asignación de valores incompatibles.

Los tipos comunes que se encuentran en los lenguajes funcionales incluyen:

Básicos: Son los primitivos, ejemplo: NUM (INT y FLOAT) (Números) BOOL(Valores de verdad) CHAR(Caracteres)

Derivados: Se construyen de otros tipos, ejemplo:

(num,char) → Tipo de pares de valores

(num → char) → Tipo de una función

Ejercicio 7: ¿Cómo definiría un programa escrito en POO?

Un programa escrito con un lenguaje POO es un conjunto de objectos que interactúan mandándose mensajes.

Ejercicio 8: Diga cuáles son los elementos más importantes y hable sobre ellos en la programación orientada a objetos.

Los elementos son: objetos, mensajes, métodos y clases.

- Los **objetos** son datos abstractos, que tienen estado interno y comportamiento.
- Los **mensajes** son peticiones de un objeto a otro para que este se comporte de una determinada manera.
- Los **métodos** son programas asociados a un objeto y cuya ejecución solo puede desencadenarse a través de un mensaje recibido por este o por sus descendientes
- Las **clases** son tipos definidos por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo. Cada objeto pertenece a una clase y este tiene su funcionalidad.

Ejercicio 9: La posibilidad de ocultamiento y encapsulamiento para los objetos es el primer nivel de abstracción de la POO, ¿cuál es el segundo?

El segundo nivel de abstracción en la programación orientada a objetos (POO) después del ocultamiento y encapsulamiento es la herencia. Permite crear nuevas clases basadas en clases existentes, heredando sus atributos y métodos, y estableciendo relaciones jerárquicas entre las clases. La herencia promueve la reutilización de código, la modularidad y la extensibilidad en el diseño de programas orientados a objetos.

Ejercicio 10: ¿Qué tipos de herencias hay? Cuál usa Smalltalk y C++

Existen diferentes tipos de herencia en la programación orientada a objetos. Los dos tipos más comunes son la herencia simple (single inheritance) y la herencia múltiple (multiple inheritance).

1. Herencia simple: Una clase puede heredar de una única clase base. Esto significa que una clase derivada puede extender y especializar los atributos y métodos de una única clase padre. La herencia simple promueve una relación jerárquica simple entre las clases y se utiliza ampliamente en muchos lenguajes orientados a objetos.
2. Herencia múltiple: Una clase puede heredar de múltiples clases bases. Esto permite que una clase derivada herede atributos y métodos de varias clases padres diferentes. La herencia múltiple ofrece una mayor flexibilidad y capacidad de reutilización de

código, pero también puede ser más compleja de manejar y puede dar lugar a problemas de ambigüedad si existen métodos con el mismo nombre en diferentes clases padres.

- Smalltalk: Utiliza la herencia simple. Una clase puede tener una única clase padre de la cual hereda.
- C++: Admite tanto herencia simple como herencia múltiple. Permite que una clase herede de una única clase base o de varias clases bases diferentes.

Es importante tener en cuenta que algunos lenguajes de programación no admiten la herencia múltiple debido a la complejidad que puede generar. En su lugar, pueden proporcionar alternativas, como interfaces o mixins, para lograr la reutilización de código de manera más controlada.

Ejercicio 11: En el paradigma lógico ¿Qué representa una variable? ¿y las constantes?

En el paradigma lógico, una variable se refiere a elementos indeterminados que pueden sustituirse por cualquier otro. Los nombres de las variables comienzan con mayúsculas y pueden incluir números

“humano(X)”, la X puede ser sustituída por constantes como: juan, pepe, etc.

Ejercicio 12: ¿Cómo se escribe un programa en un lenguaje lógico?

En un lenguaje lógico, los programas son una serie de aserciones lógicas.

El conocimiento se representa a través de reglas y hechos, los objetos se representan por términos, los cuales tienen constantes (determinado) y variables (indeterminado).

Hecho

son relaciones entre objetos y siempre son verdaderas:

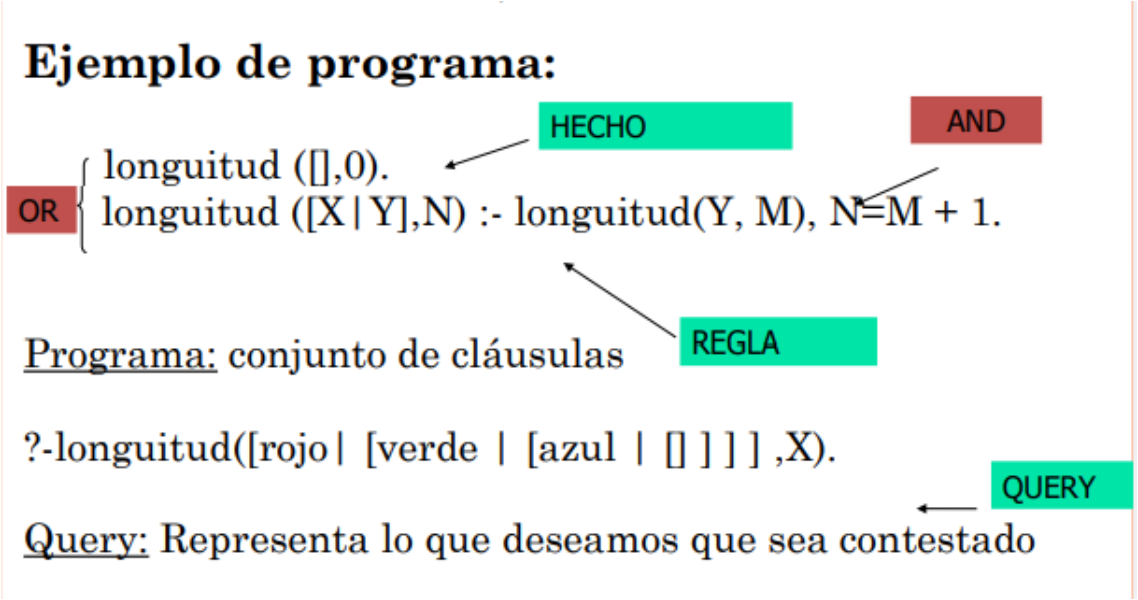
- tiene(coche,ruedas): representa el hecho que un coche tiene ruedas
- longuitud([],0): representa el hecho que una lista vacía tiene longitud cero
- moneda(peso): representa el hecho que peso es una moneda.

Regla: Cláusulas de Horn

La cláusula tiene la forma conclusión :- condición

Donde :- es If, conclusión es un predicado y condición es una conjunción de predicados separados por comas, representando un AND lógico.

Seria como decir if condición else conclusión.



En estos programas se van haciendo consultas, que es un Query. El programa va deduciendo a partir de los hechos y reglas y con eso responde la consulta.

Ejercicio 13: Teniendo en cuenta el siguiente problema, se lee una variable entera por teclado y si es par se imprime "El valor ingresado es PAR" y si es impar imprime "El valor ingresado es impar", implemente este ejemplo en cada uno de los paradigmas presentados en esta práctica.

Lógico:

```
esPar(X):-
  X mod 2 =:= 0,
  write('El valor ingresado es PAR').

 esPar(X):-
  X mod 2 =:= 1,
  write('El valor ingresado es IMPAR').

main:-
    write('Ingrese un numero: '),
    read(X),
    esPar(X).

:- main.
```

Funcional:

```
esPar :: Int -> IO ()
esPar n =
  if even n
    then putStrLn "El valor ingresado es PAR"
    else putStrLn "El valor ingresado es IMPAR"
```

```haskell
main :: IO ()
main = do
  putStrLn "Ingrese un número:"
  n <- readLn
  esPar n
```

Orientado a Objetos:

```java
public class Paridad {
    private int valor;

    public Paridad(int valor) {
        this.valor = valor;
    }

    public void imprimirParidad() {
        if (valor % 2 == 0) {
            System.out.println("El valor ingresado es PAR");
        } else {
            System.out.println("El valor ingresado es IMPAR");
        }
    }

}

public class Programa {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Ingrese un valor entero: ");
        int valor = scanner.nextInt();

        Paridad objeto = new Paridad(valor);
        objeto.imprimirParidad();
    }
}
```

Ejercicio 14: Describa las características más importantes de los Lenguajes Basados en Scripts. Mencione diferentes lenguajes que utilizan este concepto. ¿En general, qué tipificación utilizan?

Los lenguajes basados en scripts son aquellos diseñados para la creación rápida de scripts o guiones que automatizan tareas y permiten la interacción con sistemas o aplicaciones. Algunas de las características más importantes de estos lenguajes son:

1. Sintaxis sencilla: Los lenguajes basados en scripts suelen tener una sintaxis simple y legible que facilita la escritura y comprensión de los scripts.

2. Interpretados: Estos lenguajes son generalmente interpretados en lugar de ser compilados, lo que permite una ejecución más rápida y una mayor flexibilidad en tiempo de desarrollo.

3. Facilidad de uso: Estos lenguajes están diseñados para ser accesibles y fáciles de aprender, lo que los hace ideales para usuarios sin experiencia en programación.

4. Orientados a tareas específicas: Los lenguajes basados en scripts a menudo se centran en tareas específicas, como la automatización de tareas administrativas, el procesamiento de datos o la creación de páginas web dinámicas.

Algunos ejemplos de lenguajes basados en scripts son: Python, JavaScript, Ruby, Perl

En cuanto a la tipificación, algunos utilizan tipado dinámico, como Python y Ruby, donde los tipos de variables se determinan en tiempo de ejecución. Otros pueden tener tipado estático o una combinación de ambos, como TypeScript, una extensión de JavaScript que agrega tipos estáticos opcionales. La elección del enfoque de tipificación depende de los objetivos y las características específicas del lenguaje.

En general, la tipificación en los lenguajes basados en scripts puede variar dependiendo del lenguaje específico y sus objetivos de diseño.

Ejercicio 15: ¿Existen otros paradigmas? Justifique la respuesta

Si, existen otros paradigmas como:

1. Paradigma Orientado a Aspectos (AOP): Este paradigma se centra en la modularidad y separación de preocupaciones. Permite la encapsulación y reutilización de código relacionado con aspectos transversales, como el registro, la seguridad o el rendimiento, mediante la aplicación de aspectos a través de anotaciones o configuraciones.
2. Paradigma Orientado a Eventos: En este paradigma, el flujo de ejecución se basa en la ocurrencia de eventos y la respuesta a ellos. Los programas están compuestos por manipuladores de eventos que se activan cuando ocurre un evento específico. Es comúnmente utilizado en interfaces gráficas de usuario y sistemas distribuidos.
3. Paradigma Funcional Reactivo: Se centra en la programación de sistemas y aplicaciones reactivas, donde la interacción con eventos externos y la propagación de cambios son fundamentales. Utiliza flujos de datos inmutables y funciones puras para describir las transformaciones de datos a lo largo del tiempo.