Práctica 10 Paradigmas de Leng. de Programación

> **Objetivo**: poder reconocer las diferentes características que poseen los paradigmas de programación

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?

Un programa escrito en un lenguaje funcional se construye a partir de funciones. Está basado en el uso de funciones. Popular en la resolución de problemas de AI, matemática, lógica, etc.

¿Qué rol cumple la computadora?

La computadora desempeña el papel de evaluar las expresiones y aplicar las funciones de manera eficiente para obtener los resultados deseados.

Ejercicio 2

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

En un lenguaje funcional, el lugar donde se definen las funciones se conoce como contexto léxico o ámbito léxico. El contexto léxico es el entorno en el que una función es definida y determina las reglas de visibilidad y alcance de dicha función.

En general, en un lenguaje funcional, las funciones se pueden definir en los siguientes lugares:

1. A nivel de archivo: Las funciones se definen en un archivo fuente independiente y pueden ser accedidas desde otros archivos o módulos mediante mecanismos de importación o inclusión.
2. A nivel de módulo: Un módulo es una unidad de organización que agrupa funciones relacionadas. Las funciones se definen dentro del módulo y pueden ser importadas y utilizadas en otros módulos según las reglas de importación del lenguaje.
3. A nivel local: Una función también puede ser definida dentro del alcance de otra función. En este caso, se le llama función local o función anidada. Estas funciones están disponibles solo dentro del alcance de la función que las contiene y no se pueden acceder directamente desde fuera de ella.

El lugar donde se definen las funciones determina su visibilidad y alcance. Las funciones definidas a nivel de archivo o módulo son accesibles desde otros archivos o módulos, mientras que las funciones locales solo son accesibles dentro del contexto en el que están definidas.

Ejercicio 3

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

En los lenguajes funcionales, las variables son inmutables y se utilizan para asociar un valor a un nombre. A diferencia de los lenguajes imperativos, no se modifican directamente, sino que se crean nuevas variables o se aplican transformaciones a los datos existentes para obtener nuevos valores. En lugar de trabajar con variables mutables, se enfatiza el uso de expresiones y funciones puras que no tienen efectos secundarios y producen resultados basados únicamente en los valores de entrada. La inmutabilidad y el enfoque en funciones puras facilitan el razonamiento sobre el código, evitan problemas de concurrencia y permiten optimizaciones automáticas.

Ejercicio 4

¿Qué es una expresión en un lenguaje funcional?

Una expresión en un lenguaje funcional es una combinación de valores, variables y operaciones que se evalúa para producir un resultado.

¿Su valor de qué depende?

El valor de una expresión depende únicamente de los valores de entrada y de las reglas de evaluación del lenguaje, sin depender de ningún estado o efecto secundario.

Ejercicio 5

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

La forma de evaluar las expresiones es a través de un mecanismo de **reducción o simplificación.** Existen dos formas de reducción

- Orden aplicativo: Aunque no sea necesario, siempre evalúa todos los argumentos.
- Orden normal (lazy evaluation): No calcula más de lo necesario. La expresión no es evaluada hasta que su valor lo necesite. Una expresión compartida no es evaluado más de una vez.
- Diferida (Haskell): es una evolución del orden normal.

Ejercicio 6

¿Un lenguaje funcional es fuertemente tipado?

Sí, un lenguaje funcional es generalmente considerado como un lenguaje fuertemente tipado. Esto significa que los tipos de datos son estrictamente aplicados y verificados en tiempo de compilación o ejecución, lo que implica que no se permiten operaciones o conversiones automáticas entre tipos incompatibles.

¿Qué tipos existen? ¿Por qué?

Toda función tiene asociado un tipo. Los tipos son:

- **Básicos**: primitivos (num, bool, char).
- **Derivados**: se construyen a través de otros tipos
  - (num, char) → tipos de pares de valores
  - (num → char) → tipo de una función

Además de estos tipos básicos, los lenguajes funcionales a menudo admiten tipos de datos más complejos, como registros, conjuntos, funciones y tipos de datos definidos por el usuario.

La fuerte tipificación en los lenguajes funcionales proporciona ventajas en términos de seguridad y robustez del código.

Ejercicio 7

¿Cómo definiría un programa escrito en POO?

Un programa escrito en Programación Orientada a Objetos es un conjunto de objetos que interactúan entre sí enviándose mensajes para realizar tareas. Los objetos son instancias de clases que encapsulan datos y comportamientos relacionados. El programa se estructura en base a la modularidad y la reutilización de código a través de la creación y manipulación de objetos.

Ejercicio 8

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

Los elementos que lo componen son

- **Mensajes** Petición de un objeto a otro para que este se comporte de una determinada manera, ejecutando uno de sus métodos. Todo el procesamiento de este modelo es activado por mensajes.
- **Métodos** Es una porción de programa asociado a un objeto determinado y cuya ejecución sola puede desencadenarse a través de un mensaje
- **Clases** Es un tipo definido por el usuario que determina las estructuras de datos y las operaciones de ese tipo.
    - Primer nivel de abstracción de datos: Definimos estructura, comportamientos y tenemos ocultamiento.
      La información del objeto solo puede ser accedida por la ejecución de los métodos correspondientes.

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) es la herencia. La herencia permite crear nuevas clases basadas en clases existentes, heredando sus atributos y comportamientos. Esta relación jerárquica de clases permite la reutilización de código y la creación de una estructura de clases más especializada y organizada.

Ejercicio 10

¿Qué tipos de herencias hay?

Existen varios tipos de herencia en la Programación Orientada a Objetos (POO). Dos de los tipos más comunes son la herencia simple y la herencia múltiple.

La herencia simple permite que una clase herede los atributos y comportamientos de una única clase padre. En este caso, una clase puede tener una única clase padre de la cual hereda.

La herencia múltiple, por otro lado, permite que una clase herede los atributos y comportamientos de múltiples clases padres. En este caso, una clase puede tener varios padres y heredar características de todas ellas. La herencia múltiple puede llevar a problemas de ambigüedad cuando dos clases padres tienen atributos o métodos con el mismo nombre.

Smalltalk

En el caso de Smalltalk, un lenguaje de programación orientado a objetos puro, se utiliza la herencia simple. Smalltalk no admite la herencia múltiple de forma nativa, lo que promueve un diseño más simple y evita los problemas de ambigüedad asociados con la herencia múltiple.

C++

En el caso de C++, un lenguaje de programación orientado a objetos e imperativo, se permite tanto la herencia simple como la herencia múltiple. C++ admite la herencia múltiple, lo que significa que una clase puede heredar atributos y comportamientos de múltiples clases padres. Sin embargo, la herencia múltiple en C++ puede ser complicada y puede llevar a problemas de ambigüedad, por lo que se deben tomar precauciones para resolverlos, como el uso de especificadores de acceso o la resolución explícita de nombres.

Ejercicio 11

En el paradigma lógico ¿Qué representa una variable?

En el paradigma lógico, una variable representa un símbolo que puede tomar diferentes valores. A diferencia de las variables en otros paradigmas, en el paradigma lógico, las variables no se modifican una vez que se les ha asignado un valor. En cambio, las variables se utilizan para expresar relaciones y patrones en un sistema lógico, y su valor se determina mediante un proceso de unificación basado en reglas lógicas. Son elementos indeterminados que pueden sustituirse por otro, por ejemplo “humano(X)”, donde X puede sustituirse por constantes como “Juan”. Los nombres de las variables comienzan con mayúsculas y pueden tener números

¿y las constantes?

Las constantes, por otro lado, representan valores fijos y predefinidos en el sistema lógico. Estos valores no cambian durante la ejecución y se utilizan para representar elementos concretos o hechos establecidos en el dominio de problema. Las constantes pueden ser números, símbolos, cadenas de texto u otros tipos de datos, dependiendo del lenguaje de programación lógica utilizado.

Ejercicio 12

¿Cómo se escribe un programa en un lenguaje lógico?

Un programa escrito en lenguaje lógico es una secuencia de “cláusulas”, estas pueden ser hechos o reglas.

- Hechos: representan una relación entre objetos y se consideran verdaderos siempre
- Reglas: tiene la forma de conclusión .- (sí) condición. Conclusión es siempre un predicado y condición es un conjunto de predicados separados por comas. Representan un AND lógico. Análogamente, es el if de un lenguaje imperativo/procedural.

```
longuitud ([],0).
longuitud ([X|Y],N) :- longuitud(Y, M), N=M + 1.
?-longuitud([rojo| [verde | [azul | [] ] ] ] , X)
```

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.

Enunciado: **se lee una variable entera por teclado y si es par imprime “El valor ingresado es PAR**” y si es impar imprime “**El valor ingresado es impar**

Funcional:

```
main :: IO ()
main = do
  putStrLn "Ingrese un número:"
  input <- getLine
  let number = read input :: Int
  putStrLn (checkParity number)

checkParity :: Int -> String
checkParity n
  | even n = "El valor ingresado es PAR"
  | otherwise = "El valor ingresado es impar"
```

Lógico:

```
par_impar :-
  write('Ingrese un número: '),
  read(N),
  check_parity(N, Result),
  write(Result).

check_parity(N, 'El valor ingresado es PAR') :-
  0 is N mod 2.
check_parity(N, 'El valor ingresado es impar') :-
  1 is N mod 2.
```

POO (usando Java):

Script (usando Python):

```java
import java.util.Scanner;

public class Paridad {

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    System.out.println("Ingrese un número:");
    int number = scanner.nextInt();
    checkParity(number);
  }

  public static void checkParity(int number) {
    if (number % 2 == 0) {
      System.out.println("El valor ingresado es PAR");
    } else {
      System.out.println("El valor ingresado es impar");
    }
  }
}
```

```python
number = int(input("Ingrese un número: "))
if number % 2 == 0:
    print("El valor ingresado es PAR")
else:
    print("El valor ingresado es impar")
```

Ejercicio 14

Describa las características más importantes de los Lenguajes Basados en Scripts.

Los lenguajes basados en scripts son aquellos diseñados para la escritura de scripts, que son programas pequeños y ágiles utilizados para automatizar tareas o realizar operaciones específicas. Algunas características importantes de los lenguajes basados en scripts son:

1. **Sintaxis sencilla y legible**: Los lenguajes de scripting suelen tener una sintaxis simplificada que permite una escritura rápida y comprensible. Se enfocan en la facilidad de uso y la legibilidad del código.
2. **Ejecución interpretada**: Los scripts se ejecutan en tiempo de ejecución mediante un intérprete, lo que permite una ejecución rápida y una fácil iteración y depuración del código.
3. **Amplias bibliotecas y módulos**: Los lenguajes de scripting suelen contar con una amplia gama de bibliotecas y módulos que facilitan la realización de tareas específicas, como manipulación de archivos, comunicación en red, procesamiento de datos, entre otros.
4. **Orientados a tareas específicas**: Los scripts están diseñados para tareas particulares, como administración de sistemas, automatización de tareas, scripting web, procesamiento de datos, entre otros. Se enfocan en resolver problemas prácticos de manera eficiente.
5. **Flexibilidad y dinamicidad**: Los lenguajes de scripting suelen ser flexibles en cuanto a la manipulación de tipos de datos y estructuras. Permiten cambios dinámicos en tiempo de ejecución y no requieren una definición de tipos estricta.

Mencione diferentes lenguajes que utilizan este concepto.

Algunos ejemplos de lenguajes basados en scripts son: Python, JavaScript, Perl, Ruby, PowerShell, Shell scripting, etc.

¿En general, qué tipificación utilizan?

En cuanto a la tipificación, en general, los lenguajes basados en scripts suelen utilizar una tipificación dinámica o flexible, lo que significa que los tipos de datos se infieren automáticamente durante la ejecución y se pueden cambiar dinámicamente. Esto permite una mayor flexibilidad y rapidez en el desarrollo de scripts, aunque puede aumentar el riesgo de errores de tipo en tiempo de ejecución.

Ejercicio 15

¿Existen otros paradigmas? Justifique la respuesta

Además de los paradigmas de Programación Orientada a Objetos, Programación Funcional, Programación Lógica y Programación de Scripting, existen otros paradigmas de programación. Algunos ejemplos incluyen la Programación Imperativa, Estructurada, Orientada a Aspectos, Reactiva y Paralela/Concurrente. Cada paradigma tiene características únicas y se adapta mejor a diferentes problemas y

contextos de desarrollo. La elección del paradigma depende de los requisitos del proyecto y las características específicas del problema a resolver.