Ejercicio 1

¿Qué es un programa escrito en un lenguaje funcional? ¿Qué rol cumple la computadora?

Un programa escrito en un lenguaje funcional es una **composición de funciones puras**, sin efectos colaterales, que se definen en términos matemáticos. En lugar de ejecutar instrucciones paso a paso como en un lenguaje imperativo, se evalúan expresiones cuyo valor depende únicamente de los valores de entrada.

Se cumple la propiedad de “TRANSPARENCIA REFERENCIAL”: Dos expresiones sintácticamente iguales darán el mismo valor.

Ejercicio 2

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

Las funciones en los lenguajes funcionales se definen dentro de **scripts**, los cuales son listas de definiciones que pueden evaluarse, combinarse o modificarse. Estas funciones pueden escribirse de distintas formas pero mantienen siempre el mismo comportamiento. Por ejemplo, doble x = x + x y doble' x = 2 * x son formas diferentes de definir la misma función.

Además, una función puede tener un tipo explícito (cuadrado :: num → num) o implícito mediante inferencia de tipos. La definición puede incluir parámetros y operaciones matemáticas, y son tratadas como **valores de primer orden**, lo que significa que pueden ser pasadas como argumentos o devueltas como resultado.

Ejercicio 3

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

En la programación funcional, el concepto de variable se acerca más al de una **variable matemática** que a una celda de memoria modificable, como en los lenguajes imperativos. Una vez que se le asigna un valor a una variable, este **no cambia**, lo que garantiza **inmutabilidad**.

Ejercicio 4

¿Qué es una expresión en un lenguaje funcional? ¿De qué depende su valor?

En los lenguajes funcionales, una **expresión es la unidad central del lenguaje**, y su evaluación es el objetivo principal. Una expresión puede ser un número, una función, una aplicación de función, o una combinación de subexpresiones.Las expresiones también pueden contener VARIABLES, (valores desconocidos)

El valor de una expresión **depende únicamente del valor de sus subexpresiones**, sin que influyan el contexto o el estado del sistema. Esto elimina los efectos laterales y permite evaluar expresiones de forma determinista. Esta propiedad también permite aplicar diferentes estrategias de evaluación, como evaluación normal (lazy) o evaluación por aplicación.

● Las expresiones cumplen con la propiedad de “TRANSPARENCIA REFERENCIAL”: Dos expresiones sintácticamente iguales darán el mismo valor

---

Ejercicio 5

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

Los lenguajes funcionales utilizan **evaluación por reducción o simplificación**, lo que significa que las expresiones se evalúan transformándolas paso a paso en otras más simples, hasta obtener un valor final.

Existen dos estrategias principales:

> **Evaluación por orden aplicativo**: evalúa primero los argumentos de una función, aunque luego no se utilicen.
>
> **Evaluación por orden normal (lazy evaluation)**: difiere la evaluación de los argumentos hasta que su valor es necesario, evitando así cálculos innecesarios.

.

Ejercicio 6

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

Sí, los lenguajes funcionales suelen ser **fuertemente tipados**, lo que significa que cada expresión tiene un tipo bien definido y restringe las operaciones entre tipos incompatibles. Por ejemplo, no se puede sumar un número con una cadena sin antes hacer una conversión explícita.

Existen dos categorías de tipos:

> **Tipos básicos**: como num (número), char (carácter), bool (booleano).
>
> **Tipos derivados**: como listas, tuplas, funciones (por ejemplo, num → char indica una función que toma un número y devuelve un carácter).

Además, hay funciones **polimórficas**, como id x = x, que pueden operar sobre cualquier tipo y tienen tipo genérico (por ejemplo, β → β), lo que permite escribir funciones generales reutilizables.

El tipado fuerte ayuda a prevenir errores en tiempo de ejecución y mejora la confiabilidad del software.

Ejercicio 7

¿Cómo definiría un programa escrito en POO (Programación Orientada a Objetos)?

Un programa escrito en POO es un sistema organizado en torno a **objetos que interactúan enviándose mensajes que activan métodos**. Cada objeto representa una entidad que combina datos (estado interno) y comportamiento (métodos).Un método es un programa que está asociado a un objeto determinado y cuya ejecución solo puede desencadenarse a través de un mensaje recibido por éste o por sus descendientes

La lógica del programa se construye sobre estas interacciones: los objetos se envían mensajes que activan métodos, modifican su estado o solicitan información. Esta forma de programar permite representar conceptos del mundo real y estructurar el código en componentes reutilizables y mantenibles.

Ejercicio 8

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

Los elementos fundamentales de la POO son:

> **Objeto**: entidad que encapsula estado interno (atributos) y comportamiento (métodos). Cada objeto es una instancia de una clase.
>
> **Clase**: Es un tipo definido por el usuario que determina las estructuras de datos y las operaciones asociadas con ese tipo.Cada vez que se construye un objeto se está creando una INSTANCIA de esa clase
>
> **Mensaje**: mecanismo mediante el cual un objeto solicita a otro que ejecute un método. TODO el procesamiento en este paradigma es activado por mensajes entre objetos.
>
> **Método**: programa asociado a un objeto, se ejecuta cuando el objeto recibe un mensaje.

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 es la **herencia**, que permite definir jerarquías entre clases. Una clase puede heredar atributos y métodos de otra (superclase), y extender o modificar su comportamiento.

Esto permite **reutilizar código** y crear modelos más complejos a partir de componentes más simples. Por ejemplo, si una clase Empleado hereda de Persona, adquiere automáticamente atributos como nombre o edad.

Otro concepto fundamental en el paradigma orientado a objetos es el de polimorfismo: Es la capacidad que tienen los objetos de distintas clases de responder a mensajes con el mismo nombre. El polimorfismo permite tratar a objetos de diferentes clases como si fueran de una clase común (su superclase) por lo cual requiere de herencia para funcionar, ya que se basa en la jerarquía de clases.

Ejercicio 10

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

Existen distintos tipos de herencia:

Simple: una clase hereda de una única superclase. Usada por **Smalltalk**.

Múltiple: una clase puede heredar de varias superclases. Permitida en **C++**.

Herencia jerárquica: múltiples clases heredan de una misma clase base.

Herencia multinivel: una clase hereda de otra, que a su vez hereda de otra, y así sucesivamente.

Smalltalk favorece simplicidad y coherencia con herencia simple, mientras que C++ brinda mayor flexibilidad con herencia múltiple (aunque más compleja de gestionar).

Ejercicio 11

En el paradigma lógico, ¿qué representa una variable? ¿Y las constantes?

En programación lógica, una **variable** representa un **elemento indeterminado** que puede ser sustituido por cualquier otro. Se usan para establecer relaciones entre hechos o inferencias. Por ejemplo: humano(X) dice que existe un X tal que es humano. Los nombres de las variables comienzan con mayuscula y pueden incluir números

Las **constantes**, en cambio, son valores **determinados**. Representan entidades fijas del dominio, como juan, 3, o perro. La constantes son string de letras en minúsculas (representan objetos atómicos) o string de dígitos (representan números).

Ejercicio 12

¿Cómo se escribe un programa en un lenguaje lógico?

Un programa en un lenguaje lógico (como Prolog) se escribe como una **colección de hechos y reglas**, conocidas como **cláusulas de Horn**. Los hechos son lo que se toman como verdades y las reglas como cosas que se cumple a través de esas verdadesPor ejemplo:

prolog

```prolog
padre(juan, maria).          % hecho
abuelo(X,Y) :- padre(X,Z), padre(Z,Y).   % regla
```

Se puede consultar al programa mediante **queries** (preguntas), como:

prolog

```
?- abuelo(juan, maria).
```

La ejecución del programa consiste en **intentar deducir** la respuesta aplicando reglas de inferencia lógica a partir de los hechos dados.

Ejercicio 13

Problema: se lee una variable entera por teclado y si es par se imprime “El valor ingresado es PAR” y si es impar “El valor ingresado es IMPAR”. Implemente este ejemplo en cada uno de los paradigmas.

Imperativo (C/Java):

```
Scanner sc = new Scanner(System.in);
int n = sc.nextInt();
if (n % 2 == 0)
    System.out.println("El valor ingresado es PAR");
else
    System.out.println("El valor ingresado es IMPAR");
```

Funcional (Haskell):

```
paridad n = if mod n 2 == 0 then "PAR" else "IMPAR"
main = do
    input <- getLine
    putStrLn ("El valor ingresado es " ++ paridad (read input))
```

Lógico (Prolog):

```
par(N) :- 0 is N mod 2, write('El valor ingresado es PAR').
par(N) :- 1 is N mod 2, write('El valor ingresado es IMPAR').

% Consulta:
?- par(4).
```

Orientado a Objetos (Java):

```
class Paridad {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();
        System.out.println("El valor ingresado es " + (n % 2 == 0 ? "PAR" :
"IMPAR"));
    }
}
```

Ejercicio 14

Describa las características más importantes de los Lenguajes Basados en Scripts. Mencione diferentes lenguajes que utilizan este concepto. ¿Qué tipificación utilizan?

Los Lenguajes Basados en Script (LBS) son **interpretados**, dinámicos y pensados para **automatizar tareas** o "pegar" componentes existentes. Tienen una **sintaxis simple**, permiten un desarrollo rápido, y su sistema de tipos suele ser **dinámico**, es decir, no se requiere declarar tipos previamente.

Características:

- Uso en tareas de automatización, procesamiento de texto, integración web, etc.
- Tipado dinámico y flexible.
- Menor eficiencia, pero mayor velocidad de desarrollo.
- Interpretados, por lo general (no compilados).

Lenguajes representativos: **Python, Perl, PHP, JavaScript, Ruby, Tcl**.

Ejercicio 15

¿Existen otros paradigmas? Justifique la respuesta.

Sí, existen otros paradigmas además de los principales (imperativo, funcional, lógico y orientado a objetos). Ejemplos incluyen:

- **Programación orientada a aspectos**: separa responsabilidades transversales (como logs o seguridad) en módulos llamados *aspectos*.
- **Programación reactiva**: se basa en flujos de datos y eventos asíncronos, útil en interfaces o sistemas distribuidos.
- **Programación concurrente y paralela**: enfoca el diseño de programas que pueden ejecutarse en múltiples hilos o núcleos.
- **Programación dirigida por eventos**: el flujo depende de eventos externos (GUI, sensores, etc.).

Estos paradigmas **responden a diferentes necesidades** del desarrollo moderno, como modularidad, concurrencia o reactividad.