

Practica 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?

Ejercicio 1: Un programa escrito en un **lenguaje funcional** se basa en la aplicación de **funciones matemáticas puras**, sin realizar cambios de estado ni usar asignaciones de variables como en la programación procedural. En este paradigma, el énfasis está en la **evaluación de expresiones** y no en la ejecución de instrucciones paso a paso.

En lugar de modificar directamente el estado de la memoria, un programa funcional se construye a partir de funciones que reciben entradas y devuelven salidas sin causar efectos secundarios. Esto permite mayor claridad, facilidad de prueba, y un enfoque más declarativo.

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

Ejercicio2: En un lenguaje funcional, el lugar donde se definen las funciones se denomina **entorno** o **ámbito léxico (lexical scope)**. Este entorno representa el **contexto en el que una función es declarada**, y determina qué nombres (como variables o funciones) están disponibles dentro de esa función.

Las funciones en lenguajes funcionales no se definen dentro de bloques de instrucciones, sino como **declaraciones independientes**, que pueden estar:

- En el **ámbito global**, accesibles desde todo el programa.
- En un **entorno local**, por ejemplo, dentro de otra función (esto permite funciones anidadas y cierres o **closures**).

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

Ejercicio 3: En los lenguajes funcionales, el concepto de **variables** es **muy diferente al de los lenguajes imperativos o procedurales**. En lugar de representar espacios de memoria que pueden cambiar de valor (como en la programación imperativa), una ****variable** en programación funcional representa una **asociación inmutable entre un nombre y un valor**.

Es decir, una vez que a una variable se le asigna un valor, **ese valor no puede cambiar**. Por esta razón, en los lenguajes funcionales se habla más apropiadamente de **"nombres" o "vínculos"** que de variables mutables.

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

Ejercicio 4: En un lenguaje funcional, una **expresión** es cualquier construcción del lenguaje que puede ser evaluada para producir un **valor**. A diferencia de los lenguajes imperativos, donde se ejecutan instrucciones paso a paso, en los lenguajes funcionales **todo programa está compuesto por expresiones**, y no por comandos o sentencias.

El **valor de una expresión** en un lenguaje funcional depende únicamente de:

1. **Los valores de sus subexpresiones.**
2. **Las definiciones de las funciones utilizadas.**
3. **El entorno léxico (scope) en que fue definida.**

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

Ejercicio 5: Los lenguajes funcionales utilizan principalmente dos formas de evaluación: **evaluación perezosa (lazy evaluation)** y **evaluación estricta (eager o strict evaluation)**.

La

evaluación perezosa consiste en **posponer la evaluación de una expresión hasta que su valor sea estrictamente necesario**.

En cambio, cuando hablamos de evaluación estricta, significa que **los argumentos de las funciones se evalúan tan pronto como se pasan**.

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

Ejercicio 6: Sí, la mayoría de los lenguajes funcionales son **fuertemente tipados**, lo que significa que **los errores de tipos son detectados en tiempo de compilación** y que no es posible realizar operaciones entre valores incompatibles sin producir un error.

Un sistema de tipos fuerte proporciona **mayor seguridad, robustez y fiabilidad**, al evitar errores comunes como sumar un número con una cadena de texto o acceder a campos inexistentes en una estructura.

✓ ¿Qué tipos existen en un lenguaje funcional?

Los lenguajes funcionales suelen ofrecer una **amplia variedad de tipos**, incluyendo:

1. Tipos primitivos:

- Números (`Int` , `Float`)
- Cadenas (`String` , `Char`)
- Booleanos (`Bool`)

2. Tipos compuestos:

- Tuplas: `("Hola", 3)`
- Listas: `[1, 2, 3]`
- Registros o estructuras (dependiendo del lenguaje)

3. Tipos algebraicos (en lenguajes como Haskell):

- **Tipos suma** (definiciones con alternativas):

```
haskell
CopiarEditar
data Color = Rojo | Verde | Azul
```

Tipos producto (combinación de campos):

```
haskell
CopiarEditar
data Persona = Persona String Int
```

4. Funciones como tipos de primera clase:

- Las funciones se tratan como valores y también tienen tipos, por ejemplo:

```
haskell
CopiarEditar
suma :: Int → Int → Int
```

5. Tipos polimórficos:

- Permiten escribir funciones genéricas. Ejemplo en Haskell:

```
haskell
CopiarEditar
longitud :: [a] → Int
```

6. Tipos de orden superior y funciones anónimas:

- Las funciones pueden tomar y devolver otras funciones como parámetros.

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

Ejercicio 7: Un programa escrito en POO se basa en clases y objetos que colaboran entre sí mediante el envío de mensajes para realizar una tarea o resolver un problema.

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

Ejercicio 8:

- Clase: define los atributos y comportamientos de un objeto. Sirve como molde.
- Objeto: Es una entidad del mundo real que conoce o trabaja con otros objetos para resolver un problema del mundo real.
- Atributo: Es una característica del objeto que se guarda en una variable.
- Método: Define el comportamiento de un objeto.

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

Ejercicio 9: El segundo nivel de abstracción esta formado por:

- Herencia: Permite declarar una clase con comportamiento generales, y luego crear clases derivadas (subclases) que heredan esos comportamientos y los especializan si es necesario.
- Polimorfismo: Permite que diferentes clases respondan de manera diferente al mismo mensaje o método, según el tipo del objeto que lo invoque.

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

Ejercicio 10:

1. Herencia simple

- Una clase hija hereda de una sola clase padre.

2. Herencia múltiple

- Una clase hija hereda de **dos o más** clases padres.

3. Herencia multinivel

- Una cadena de herencia donde una clase hereda de otra que a su vez hereda de otra.

4. Herencia jerárquica

- Varias clases heredan de una misma clase base.

5. Herencia híbrida

- Combinación de dos o más tipos anteriores (por ejemplo, jerárquica y múltiple).

◆ ¿Qué tipo de herencia usa Smalltalk?

- **Smalltalk usa solo herencia simple.**

No permite herencia múltiple, lo que mantiene su modelo más limpio y coherente.

◆ ¿Qué tipo de herencia permite C++?

- **C++ permite herencia múltiple**, además de todos los otros tipos (simple, multinivel, jerárquica, híbrida).
- Soporta también mecanismos para resolver **conflictos de ambigüedad** en herencia múltiple (por ejemplo, mediante clases virtuales).

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

Ejercicio 11:

¿Qué representa una *variable*?

- Una **variable** representa un **valor desconocido o por determinar**, que el sistema intentará **unificar** (hacer coincidir) con valores concretos al evaluar una consulta. Las variables son **universales** y pueden tomar **cualquier valor** que haga verdadera una cláusula.

¿Qué representa una *constante*?

- Una **constante** representa un **valor fijo, específico e inmutable** dentro del programa lógico. Pueden representar objetos, personas, números, etc., y **no cambian ni se sustituyen** durante la ejecución.

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

Ejercicio 12: En un lenguaje lógico se escribe declarando:

- Hechos: declaran conocimientos que se consideran verdaderos.

- Reglas: definen relaciones lógicas entre los hechos. Se escribe con "!=" y se lee "si...".
- Consultas: preguntan que se hacen al sistema para que resuelva usando hechos y reglas.

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.

Ejercicio 13:



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?

Ejercicio 14: Son lenguajes diseñados principalmente para automatizar tareas, controlar otros programas o realizar operaciones de forma rápida, sin necesidad de compilación. Se utilizan comúnmente en desarrollo web, administración de sistemas, automatización y tareas de integración.

◆ Características más importantes:

1. Interpretados

- Se ejecutan directamente línea por línea sin necesidad de compilación previa.
-  Ventaja: Desarrollo rápido y prueba inmediata.
-  Desventaja: Suelen ser más lentos que los lenguajes compilados.

2. Tipado dinámico

- No es necesario declarar el tipo de las variables. El tipo se determina en tiempo de ejecución.

3. Sintaxis sencilla y flexible

- Diseñados para ser fáciles de escribir, leer y mantener.

4. Multiparadigma

- Muchos lenguajes de script permiten programación estructurada, orientada a objetos y funcional.

5. Alta productividad

- Permiten escribir menos líneas de código para hacer tareas comunes.

6. Uso en tareas específicas o entornos embebidos

- Pueden servir como lenguajes de configuración o extensión dentro de otros programas (como scripts de juegos, macros en hojas de cálculo, etc.).


7. No requieren compilación

- Se guardan como archivos de texto plano y se ejecutan directamente por un intérprete.

Ejercicio 15: ¿Existen otros paradigmas? Justifique la respuesta


Ejercicio 15: **Principales paradigmas adicionales:**

1. Paradigma imperativo


- **Enfoque:** Indica paso a paso *cómo* debe hacerse una tarea.
- **Características:** Uso de variables, asignaciones, bucles y estructuras de control.
- **Lenguajes:** C, Pascal, Fortran.
-  Muy común en programación de bajo nivel o cercana al hardware.

2. Paradigma funcional


- **Enfoque:** Basado en funciones puras y en la matemática. Evita cambios de estado y variables mutables.
- **Características:** No hay efectos secundarios, usa funciones como ciudadanos de primera clase.
- **Lenguajes:** Haskell, Lisp, Elixir, F#.

-  Muy útil en programación concurrente y para mantener código predecible.


3. Paradigma declarativo

- **Enfoque:** Se describe *qué* debe lograrse, no *cómo* hacerlo.
- **Lenguajes:** SQL, HTML, lenguajes funcionales y lógicos también entran en esta categoría.
-  Muy útil cuando se desea expresar intenciones sin detallar la implementación.

4. Paradigma orientado a eventos

- **Enfoque:** Las acciones del programa se disparan como respuesta a eventos (por ejemplo, clics del usuario).
- **Lenguajes:** JavaScript, Visual Basic, C# (con interfaces gráficas).
-  Común en aplicaciones gráficas e interfaces de usuario.

5. Paradigma concurrente y paralelo

- **Enfoque:** Permite que múltiples procesos o hilos se ejecuten al mismo tiempo.
- **Lenguajes:** Go, Erlang, Rust (con concurrencia segura), Java (con hilos).
-  Fundamental en sistemas modernos y aplicaciones distribuidas.