



Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica



LENGUAJES DE PROGRAMACIÓN Y LAB PIA

Docente: Ing. Karla Patricia Uribe Sierra

Alumno: Bernardo Gutiérrez López

Matrícula: 2222904

Grupo: 005

Día y Hora: M5

Fecha de elaboración: 16/05/2025

Tabla de contenido

Introducción	2
A4-Reporte conceptos fundamentales.	3
A5-Ej.Lenguaje Script	11
A6-Ej.Lenguaje Funcional	27
A7.Ej.Lenguaje Lógico	41
A8.Reporte programas imperativos	45
A9.Ej.Lenguaje orientado a objetos	51
Programa	54
Exposición	66
Conclusión	73

Introducción

En el contexto del estudio de la ciencia de la computación, el dominio de los paradigmas de programación representa un pilar fundamental para el desarrollo de competencias sólidas en el diseño, análisis e implementación de soluciones computacionales. Este reporte tiene como propósito compilar, en un solo documento estructurado, los trabajos elaborados a lo largo del curso, correspondientes a distintos estilos de programación que reflejan enfoques diversos y complementarios para la resolución de problemas.

La programación no es únicamente una actividad técnica; es, ante todo, una forma de pensamiento. Cada paradigma representa una visión distinta sobre cómo entender y modelar el mundo mediante código. En esta recopilación se abordan los lenguajes script, funcional, lógico, imperativo y orientado a objetos, cada uno con su propia filosofía, ventajas y áreas de aplicación. Junto a ellos, se incluye también un reporte introductorio sobre los conceptos fundamentales de la programación, que sirve como base teórica para la comprensión de los paradigmas restantes.

Este documento busca no solo presentar los contenidos de forma ordenada, sino también evidenciar la evolución del aprendizaje y el desarrollo de habilidades a lo largo del curso. Cada sección refleja un proceso de análisis, comparación y aplicación práctica, que permite al lector apreciar las diferencias estructurales y conceptuales entre los paradigmas abordados.

Como complemento técnico y didáctico, se ha implementado una interfaz en HTML que actúa como menú interactivo, permitiendo al usuario acceder de manera directa a cada uno de los trabajos en formato PDF. Esta herramienta facilita la navegación del material y refuerza el componente práctico del curso, integrando conocimientos de desarrollo web básico.

En suma, esta recopilación representa un ejercicio de integración académica y técnica, orientado a reforzar el entendimiento profundo de los distintos paradigmas de programación y a fomentar una visión más crítica y versátil del quehacer informático.

A4-Reporte conceptos fundamentales.

Definición, características, propósitos y relaciones de la sintaxis y la semántica

Definición

- **Sintaxis:** Es el conjunto de reglas y estructuras que determinan la forma correcta de construir oraciones en un lenguaje, ya sea natural (como el español) o de programación.
- **Semántica:** Se refiere al significado de las palabras, frases o símbolos dentro de un lenguaje. En programación, define lo que realmente hace un programa más allá de que esté escrito correctamente.

Características

- **Sintaxis:**
 - Establece el orden y la estructura de las palabras o símbolos.
 - Dicta cómo deben escribirse las instrucciones.
 - Si se viola la sintaxis, se producen errores.
- **Semántica:**
 - Se enfoca en la interpretación.
 - Hay código sintácticamente correcto pero semánticamente incorrecto.

Propósito

- **Sintaxis:** Garantizar que la estructura de las oraciones o instrucciones siga un formato establecido.
- **Semántica:** Asegurar que las expresiones tengan sentido y produzcan el resultado esperado.

Relaciones entre sintaxis y semántica

- La sintaxis es la forma en la que se escriben las expresiones, mientras que la semántica define su significado.
- Un código puede cumplir con la sintaxis del lenguaje, pero si su semántica es incorrecta, no funcionará como se espera.

Lenguaje de representación, definición, características y ejemplos

Definición

Un lenguaje de representación es un sistema formal de símbolos y reglas utilizado para describir, modelar o representar información, conocimientos o estructuras en un dominio específico.

Puede emplearse en diversos campos - como la inteligencia artificial, la lógica, la programación y la representación del conocimiento.

Características

- Uso de símbolos y estructuras formales: Emplea notaciones específicas para expresar ideas.
- Precisión y claridad: Evita ambigüedades, asegurando que la información sea interpretada uniforme.
- Expresividad: Permite representar conceptos complejos mediante estructuras lógicas o matemáticas.
- Interpretabilidad: Puede ser procesado por humanos o computadoras según el contexto de aplicación.
- Reglas bien definidas: Posee una gramática o sintaxis que establece cómo deben construirse las representaciones.

Ejemplos

Lenguajes lógicos

- Lógica proposicional y de primer orden
- OWL (Web Ontology Language)

Lenguajes de modelado y programación

- UML (Unified Modeling Language)
- EPC (Entidad-Relación)

Lenguajes matemáticos y formales

- Lenguaje de conjuntos
- Lenguaje algebraico

Lenguaje de consulta, definición, características, ejemplos

Definición

Un lenguaje de consulta es un tipo de lenguaje utilizado para solicitar, recuperar, modificar o manipular datos almacenados en bases de datos u otros sistemas de información.

Estos lenguajes permiten a los usuarios interactuar con grandes volúmenes de datos de manera estructurada y eficiente.

Características

- Acceso y manipulación de datos: Permite recuperar, insertar, actualizar y eliminar datos.
- Uso de sintaxis recuperada: Siguen reglas específicas para formular consultas y operaciones.
- Independencia de datos: Se pueden utilizar sin necesidad de conocer la estructura interna de la base de datos.
- Eficiencia y optimización: Diseñados para manejar grandes volúmenes de datos de manera rápida.
- Interactividad: Pueden ejecutarse en tiempo real para obtener respuestas inmediatas.

Ejemplos

- SQL: Lenguaje estándar para gestionar bases de datos relacionales.
- GraphQL: Lenguaje para consultar datos en bases de datos XML de manera flexible y eficiente.
- SPARQL: Diseñado para recuperar información de bases de datos basadas en la web semántica.
- Datalog: Lenguaje declarativo en bases de datos deductivos y sistemas de inteligencia artificial.

Lenguaje de alto nivel, definición, características y ejemplos

Definición

Un lenguaje de alto nivel es un tipo de lenguaje de programación diseñado para ser más comprensible para los humanos, utilizando una sintaxis similar al lenguaje natural y abstracciones que facilitan la escritura y comprensión de código. Estos lenguajes permiten desarrollar software sin necesidad de conocer los detalles internos del hardware.

Características

- Sintaxis comprensible y cercana al lenguaje humano
- Independencia del hardware: lo que permite portabilidad entre diferentes sistemas.
- Facilidad de uso y aprendizaje: reduciendo la complejidad de la programación.
- Gestión automática de memoria: en algunos casos como Python o Java.
- Uso de paradigmas de programación como la programación estructurada, orientada a objetos.
- Mayor abstracción: en comparación con los lenguajes de bajo nivel (como el ensamblador).

Ejemplo

- Python: fácil de leer y comprender, usando el desarrollo web, ciencia de datos e IA
- Java: lenguaje orientado a objetos ampliamente utilizado en aplicaciones, módulos
- C#: Usando en desarrollo de software empresarial y videojuegos con Unity
- Ruby: con una sintaxis clara, usando en desarrollo web con Ruby on Rails

Componentes de un lenguaje de programación y sus relaciones

1: Sintaxis

Define las reglas gramaticales del lenguaje, determinando cómo deben estructurarse las instrucciones.

2: Semántica

Se refiere al significado de las instrucciones dentro del lenguaje. Incluso si un código es sintácticamente correcto, puede ser semánticamente inválido.

3: Tipos de datos

Especifican qué tipo de valores pueden almacenar las variables. Enteros, flotantes, cadenas, booleanos, etc.

4: Variables y constantes

Reservan espacios en memoria que almacenan valores. Pueden cambiar las variables, pero no las constantes.

5: Operadores

Permiten realizar operaciones matemáticas, lógicas y de asignación sobre variables y valores. Su relación es que interactúan con expresiones y estructuras de control para definir el comportamiento del programa.

6: Expresiones

Combinaciones de variables, operadores y funciones que producen un valor. Su relación es que se utilizan dentro de estructuras de control y funciones para definir condiciones y cálculos.

7: Estructuras de control

Incluyen condicionales (if, else), bucles (for, while) y estructuras de salto (break, continue). Su relación es que dependen de expresiones y operadores para tomar decisiones y modificar el flujo del programa.

8: Compilador o intérprete

Traduce el código fuente a un formato ejecutable, ya sea en código máquina o intérprete por instrucción.

Definición y características de un compilador e intérprete

Definición

Compilador: es un programa que traduce el código fuente escrito en un lenguaje de programación de alto nivel a código máquina o código intermedio antes de su ejecución.

Intérprete: es un programa que traduce y ejecuta el código fuente línea por línea sin generar un archivo ejecutable previo.

Características

Compilador

- Traducción completa: convierte todo el código fuente en un archivo ejecutable antes de que el programa se ejecute.
- Eficiencia en la ejecución: Como el código ya está traducido, la ejecución es rápida.
- Detección de errores previa: Muestra los errores antes de ejecutar el programa, pero no permite corregirlos en tiempo de ejecución.
- Ejemplo de compiladores: GCC (para C/C++), Java Compiler (javac), Clang.

Intérprete

- Traducción en tiempo real: Ejecuta el código directamente, sin necesidad de generar un archivo.
- Menor velocidad de ejecución: Como traduce línea por línea la velocidad es más lenta que un compilador.
- Facilidad para depuración: Al detectar errores en tiempo de ejecución, permite probar partes del código sin recomilar todo.
- Ejemplo de intérprete: Python, Ruby, JavaScript (V8), PHP.

Estructura, operación y ejemplos de un Interpretador

Estructura de un Interpretador

1. Análisis léxico (lexer): Divide el código fuente en tokens (palabras clave, identificadores, operadores, etc.)
2. Análisis sintáctico (parser): Verifica que los tokens sigan las reglas del lenguaje y construye una estructura (árbol sintáctico).
3. Análisis semántico: Comprueba que las operaciones sean válidas según el tipo de dato y contexto.
4. Ejecutor (Runtime Engine): Evalúa y ejecuta cada línea del código en el entorno adecuado.

Operación de un Interpretador

1. Lee una línea del código fuente.
2. Convierte la línea en tokens.
3. Verifica la sintaxis y genera una estructura de datos.
4. Valida el significado de la instrucción.
5. Ejecuta la instrucción y muestra el resultado.
6. Repite el proceso hasta que finalice el código o se encuentre un error.

Ejemplos de Interpretadores

• Interpretador de Python

Python usa un interpretador para ejecutar el código linea por linea

```
print("Hola, mundo")
```

Salida

Hola, mundo

Si hay un error en cualquier linea, la ejecución se detiene inmediatamente.

```
print ("Hola")
```

Salida

SyntaxError: EOF while scanning string literal

A5-Ej.Lenguaje Script

Reporte: Implementación de programas en diferentes lenguajes script

1- Programa en Python: Suma de dos números

```
a = int(input("Ingrese el primer número: "))
b = int(input("Ingrese el segundo número: "))
suma = a + b
print(f"La suma de {a} y {b} es {suma}")
```

- 1: Se solicita al usuario ingresar dos números enteros
- 2: Se convierten las entradas a tipo int y se almacenan en a y b.
- 3: Se realiza la suma de a+b y se guarda en la variable suma.
- 4: Se muestra el resultado en pantalla

2- Programa en Perl: Contador de palabras en una cadena

```
use strict;
use warnings;
```

```
print "Ingrese una cadena de texto: ";
my $texto = <STDIN>;
chomp($texto);
my @palabras = split(/\s+/, $texto);
my $cantidad = scalar @palabras;
print "La cadena, tiene $cantidad palabras.\n";
```

- 1: Se solicita al usuario ingresar una cadena de texto
- 2: Su uso <STDIN> para capturar la entrada del usuario y chomp para eliminar el salto de línea
- 3: Se divide el texto en palabras utilizando split(/\s+/)
- 4: Se encuentra el número de palabras (@palabras) y se muestra el resultado

3. Programa en Shell Script: Verificar si un archivo existe

```
#!/bin/bash
```

```
echo "Ingrese el nombre del archivo."
```

```
read archivo
```

```
if [-f "$archivo"]; then
```

```
    echo "El archivo '$archivo' existe."
```

```
else
```

```
    echo "El archivo '$archivo' no existe."
```

1: Se solicita al usuario que ingrese un nombre de archivo.

2: Se almacena la entrada en la variable archivo

3: Se uso [-f "\$archivo"] para verificar si el archivo existe en el sistema

4: Se muestra un mensaje indicando si el archivo existe o no.

Introducción de los lenguajes de Scripting

Definición de los lenguajes de Scripting

Los lenguajes de scripting son un tipo de lenguaje de programación diseñado para automatizar la ejecución de tareas dentro de un entorno informático. A diferencia de los lenguajes compilados, los scripts se interpretan en tiempo de ejecución, lo que permite un desarrollo más rápido y una mayor flexibilidad.

Características

- **Interpretados:** No requieren compilación previa; el código se ejecuta línea por línea.
- **Dinámicos:** Permiten la modificación del código durante la ejecución.
- **Sintaxis simple y concisa:** Facilitan el desarrollo rápido de aplicaciones.
- **Automatización de tareas:** Se utilizan para escribir scripts que ejecutan secuencias de comandos repetitivas.

Tipos de lenguajes de Scripting

- **Scripting para sistemas:** Utilizados para la administración de sistemas operativos.
- **Scripting web:** Utilizados en desarrollo web para programación del lado del cliente o servidor.
- **Scripting de automatización:** Empleados para automatizar tareas dentro de aplicaciones.
- **Scripting embedido:** Integrados dentro de otros programas para ampliar su funcionalidad.

Comparación entre lenguajes de Scripting y lenguajes compilados

Los lenguajes compilados como C o Java, requieren un proceso de compilación antes de su ejecución, lo que suele hacerlos más eficientes en términos de rendimiento. Sin embargo, los lenguajes de Scripting ofrecen ventajas como la facilidad de desarrollo, depuración y mantenimiento.

Ejemplos de lenguajes de Scripting

- JavaScript: Utilizado principalmente en desarrollo web para la manipulación del DOM y la interactividad del usuario.
- Python: Ampliamente utilizado en automatización, desarrollo web y análisis de datos.
- Bash: Común en entornos Unix/Linux para la administración de sistemas.
- PowerShell: Lenguaje de scripting de Windows orientado a la administración de sistemas.
- Perl: Usado en el procesamiento del texto y administración de servicios.

Los lenguajes de scripting desempeñan un papel fundamental en la informática moderna debido a su flexibilidad y facilidad de uso.

Python, Perl, Shell

Python

Python es un lenguaje de programación de alto nivel, interpretado y de propósito general, ampliamente utilizado en el desarrollo de software, análisis de datos, inteligencia artificial, automatización y más. Fue creado por Guido van Rossum y lanzado en 1991 con un diseño enfocado en la legibilidad del código y la simplicidad.

Características de Python

- Lenguaje interpretado: no necesita compilación previa, su código se ejecuta línea por línea.
- Sintaxis clara y legible: Python utiliza indentación en lugar de llaves {}, lo que mejora la legibilidad.
- Tipado Dinámico: No es necesario declarar el tipo de las variables, lo asigna automáticamente.
- Multiparadigma: Soporta múltiples paradigmas de programación:
 - Orientado a Objetos: Usa clases y objetos
 - Funcional: Funciones de orden superior, map, filter
 - Imperativo y Procedural: estructuras clásicas de control.
- Gran comunidad y soporte: Es uno de los lenguajes más populares con una gran cantidad de bibliotecas y documentación.

Aplicaciones de Python

- Desarrollo web: Frameworks como Django y Flask permiten crear sitios web dinámicos con Python
- Videojuegos: Librerías como Pygame permiten desarrollar juegos en 2D con Python.

- Ciencia de datos y Machine learning: Bibliotecas como Pandas, NumPy, Matplotlib y Scikit-learn facilitan el análisis de datos y la inteligencia artificial.
- Automatización y Scripting: Python es ideal para escribir scripts que automatizan tareas repetitivas como la manipulación de archivos.
- Cyberseguridad y Hacking Ético: Herramientas como Scapy o Pwntools permiten realizar pruebas de penetración y análisis de seguridad.

Ventajas de Python

- Fácil de aprender gracias a su sintaxis clara
- Gran comunidad y documentación
- Multiplataforma, corre en Windows, Mac y Linux
- Enorme ecosistema de bibliotecas para distintos usos

Desventajas de Python

- Menor rendimiento que lenguajes compilados como C++
- No es el ideal para aplicaciones en tiempo real
- No tiene control de bajo nivel sobre la memoria

Python se ha consolidado como uno de los lenguajes de programación más populares del mundo debido a su facilidad de uso, versatilidad y comunidad activa. Su aplicación en diversos áreos como el desarrollo web lo convierte en una herramienta esencial para programadores de todos los niveles.

Perl

Perl es un lenguaje de programación interpretado, diseñado para la manipulación de texto y ampliamente utilizado en administración de sistemas, desarrollo web y bioinformática. Fue creado en 1987 por Larry Wall como una herramienta para manipular texto y generar informes.

Características de Perl

- Interpretado: No requiere compilación
- Manejo avanzado de cadenas de texto y expresiones regulares
- Compatible con múltiples plataformas (Windows, Mac, Linux)
- Lenguaje de propósito general, útil en desarrollo web, administración de sistemas y análisis de datos.
- Extensa colección de módulos a través de CPAN

Ventajas de Perl

- Excelente para procesamiento de texto y análisis de logs
- Gran cantidad de módulos disponibles en CPAN
- Compatibles con múltiples sistemas operativos
- Mayor flexibilidad en la programación que Bash.

Desventajas de Perl

- Sintaxis menos clara que Python
- El código puede volverse complejo y difícil de mantener.

Shell

Los Shell Scripts nacieron con Unix en los años 70 para automatizar tareas administrativas, el interprete más popular es Bash, desarrollado en 1989 como una mejora del Bourne Shell original, es ampliamente utilizado en Linux y Mac OS para gestionar servidores, manipular archivos y programar tareas.

Características de Shell

- Ligero y eficiente: No requiere compilación, ideal para tareas rápidas
- Interacción con el sistema operativo: Permite ejecutar comandos del sistema.
- Automatización: Muy usado en DevOps y administración de servidores
- Compatible con la mayoría de los sistemas Unix / Linux y mac OS.

Ventajas de Shell

- Ideal para automatizar tareas del sistema
- Ligero y eficiente en entornos Unix / Linux
- Fácil de combinar con otros lenguajes como Python o Perl

Desventajas de Shell

- No es ideal para procesamiento de datos complejos
- Dependiente del sistema operativo
- Sintaxis menos estructurada para programas grandes.

Definición, características y ejemplos de diagramas de flujo

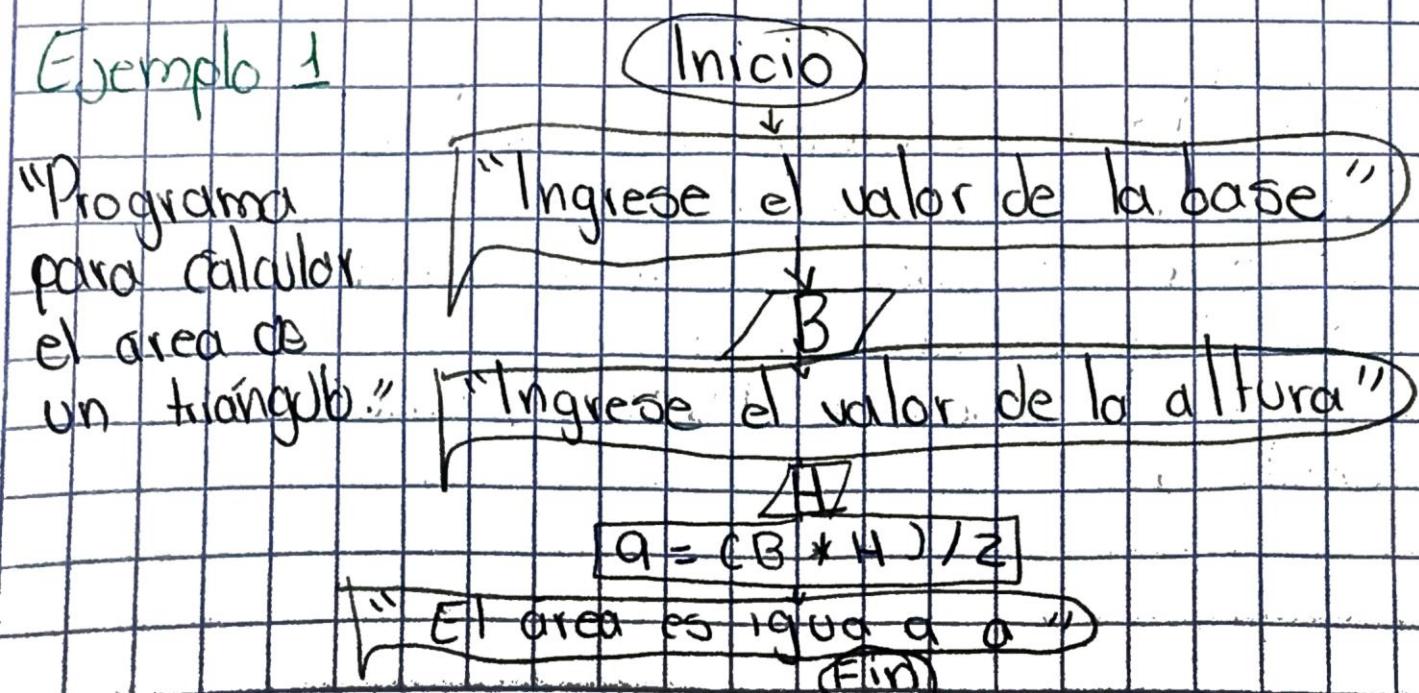
Definición

Un diagrama de flujo es una representación gráfica de un proceso o algoritmo que utiliza símbolos estandarizados para mostrar la secuencia de pasos necesarios para alcanzar un objetivo. Es una herramienta visual utilizada para comprender, analizar, documentar y mejorar procesos en diversos disciplinas, como la informática, la ingeniería y la gestión empresarial.

Características

- Secuencialidad: Utiliza un orden lógico,
- Símbología estandarizada: Figuras geométricas
- Flujo direccional: Usan flechas para indicar la secuencia de ejecución
- Claridad y simplicidad: Comprensibles y fáciles de interpretar.
- Flexibilidad: Adaptables a diferentes niveles de complejidad.

Ejemplo 1



Definición, características, propósito y ejemplos de los lenguajes funcionales

Definición

Los lenguajes funcionales son un paradigma de programación basado en funciones matemáticas y la evaluación de expresiones, en lugar de la ejecución de instrucciones secuenciales. Se enfatiza la inmutabilidad de datos y la ausencia de efectos secundarios, lo que los hace ideales para programación concurrente y sistemas críticos.

Características

- Funciones como ciudadanos de primera clase: Se pueden asignar a variables, pasar como argumentos y devolver como resultados.
- Inmutabilidad: Los datos no cambian después de ser creados.
- Evaluación perezosa: Las expresiones solo se evalúan cuando son necesarias, optimizando el rendimiento.
- Ausencia de efectos secundarios: Evitan modificar el estado global.

Propósitos

- Facilitar la declaración decorativa.
- Optimización y paralelización.
- Mejor legibilidad y mantenimiento.
- Evitar errores comunes.
- Uso en inteligencia artificial y análisis de datos.

Ejemplos

1. Haskell: lenguaje puramente funcional, con tipos fuerte y evaluación perezosa.

```
factoria n = if n == 0 then 1 else n * factorial(n - 1)
```

2. Lisp / Clojure

```
(defn factorial [n]
```

```
(if (zero? n) 1 (* n (factorial (dec n)))))
```

3. Erlang

```
factorial(0) → 1;
```

```
factorial(N) → N * factorial(N - 1);
```

4. Scala

```
def factorial(n: Int) = if (n == 0) 1 else n * factorial(n - 1)
```

5. F#

```
let rec factorial n = if n = 0 then 1 else n * factorial(n - 1)
```

Ejemplos de diagramas de flujo de recursión

Pilas y listas

Diagrama de flujo de recursión

Los diagramas de flujo son una herramienta gráfica utilizada para representar la lógica de algoritmos.

La recursión es una técnica en la que una función se llama a sí misma para resolver un problema.

Ventajas de un diagrama

- Hace que el código sea más elegante y fácil de entender.
- Util para problemas que tiene una estructura recursiva natural

Desventajas de un diagrama

- Usa más memoria (se almacena cada llamada en la pila de ejecución)
- Puede causar desbordamiento de pila si no tiene un caso base adecuado.

Ejemplo

Cálculo del factorial de un número

Resultado = Factorial(N)

No

$N == 0$

Si

Resultado = 1

Fin

(Resultado = $N * \text{Factorial}(N - 1)$)

Pilas (Stacks)

Una pila es una estructura de datos que sigue el principio LIFO (Last In, First Out) donde el último elemento agregado es el primero en salir.

Características de las Pilas

- Push: Agrega un elemento a la pila
- Pop: Elimina el último elemento agregado
- Top: Muestra el elemento en la cima sin eliminarlo.
- Estructura LIFO: Se apilan los elementos y se sacan en orden inverso

Listas

Una lista es una colección de elementos que pueden almacenarse de forma contigua (arrays) o dinámica (listas enlazadas).

Características

- Puede crecer dinámicamente sin un tamaño fijo.
- Se pueden insertar y eliminar elementos sin mover todos los demás.
- Se accede a los elementos recorriendolos uno por uno.

Haskell

Definición

Haskell es un lenguaje de programación funcional puro, de tipado estático y evaluación perezosa. Se ha desarrollado con el propósito de investigar y aplicar conceptos avanzados de programación funcional, proporcionando una sintaxis clara y expresiva.

Historia y desarrollo

Haskell fue concebido en 1987 por un comité de investigadores que buscaban unificar diversas aproximaciones a la programación funcional. Su primera versión estable, Haskell 1.0, fue publicada en 1990.

Características

- Funcional puro. No permite efectos secundarios.
- Tipado estático y fuerte: Utiliza un sistema de tipos basado en Hindley-Milner.
- Evaluación perezosa.

Ventajas

- Mayor seguridad y menor cantidad de errores en tiempo de ejecución.
- Código conciso y expresivo.

Desventajas

- Curva de aprendizaje pronunciada.
- Menos herramientas y bibliotecas.

Scheme

Definición

Scheme es un lenguaje de programación funcional y procedimental, perteneciente a la familia de los lenguajes Lisp. Se caracteriza por su simplicidad, expresividad y el uso extenso de la recursión y la manipulación de listas.

Scheme fue desarrollado en la década de 1970 por Guy L. Steele y Gerald Jay Sussman en el Instituto Tecnológico de Massachusetts (MIT).

Características

- Minimalismo: Posee una sintaxis sencilla basada en S-expressions y un pequeño conjunto de primitivas fundamentales.
- Evaluación basada en expresiones: Todo en Scheme es una expresión, lo que permite composición y reutilización de código.

Ventajas

- Sintaxis simple y consistente
- Alto nivel de expresividad y poder semántico.
- Excelente soporte para recursión
- Facilita el aprendizaje de conceptos.

Desventajas

- Menor adopción en la industria
- Rendimiento inferior en algunos casos
- Pocos herramientas y bibliotecas

A6-Ej.Lenguaje Funcional

Nombre: Gutiérrez López Bernardo

Matrícula: 2222904

A6. Reporte que describe la ejecución de un programa funcional

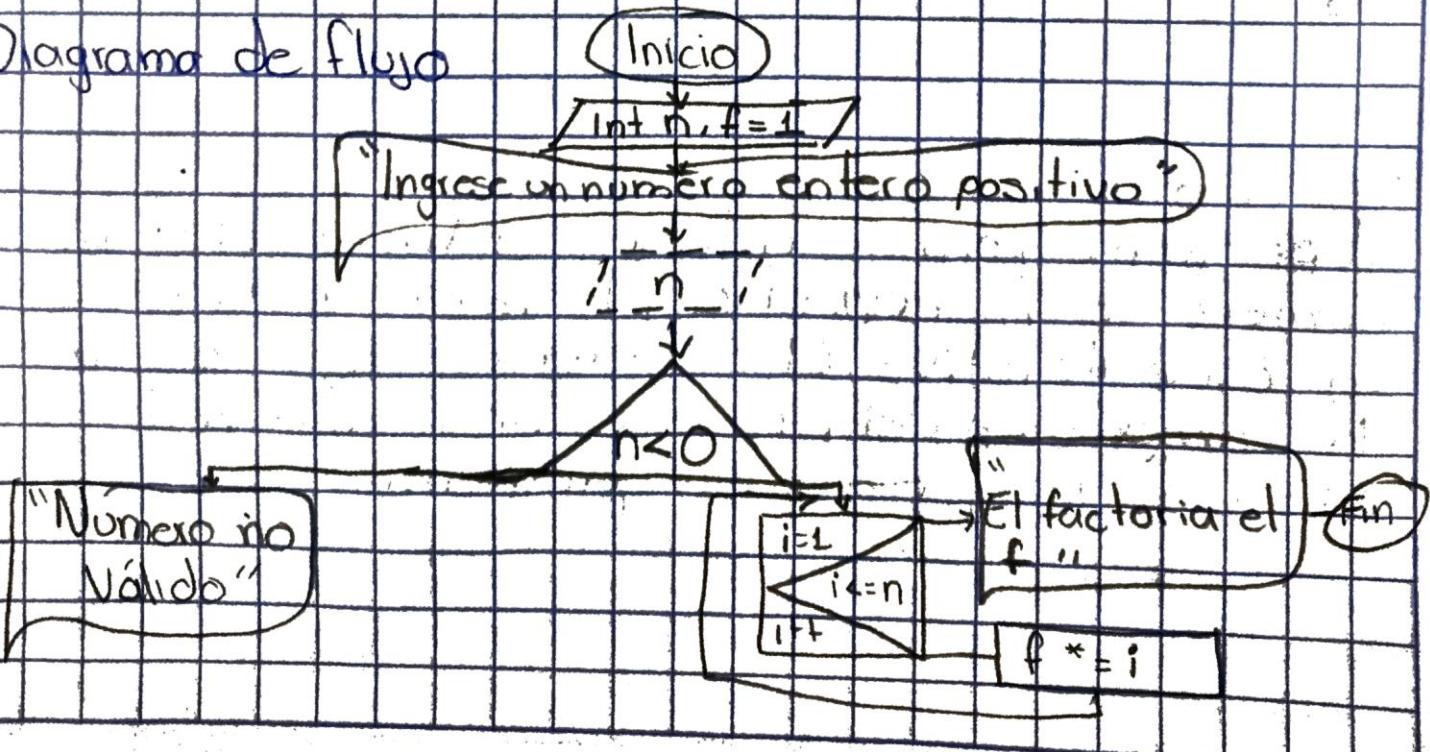
Introducción

En este reporte se describe la ejecución de un programa funcional mediante un diagrama de flujo. Se explicará el funcionamiento del programa, su estructura y el flujo de ejecución, facilitando la comprensión del proceso.

Algoritmo del programa

1. Iniciar el programa
2. Solicitar al usuario un número entero positivo
3. Verificar que el número sea válido (mayor igual a 0)
4. Si el numero es 0 o 1, devolver 1 como resultado
5. Mayor que 1, calcular el factorial
6. Mostrar el resultado en pantalla
7. Finalizar el programa

Diagrama de flujo



Implementación del código

```
#include <stdio.h>
int main () {
    int n, f = 1;
    printf ("Ingrese un número entero positivo ");
    scanf ("%d", &n);
    if (n < 0) {
        printf ("Número no válido ");
    } else {
        for (int i = 1; i <= n; i++) {
            f *= i;
        }
        printf ("El factorial es %.d ", f);
    }
    return 0;
}
```

Conclusión

El programa permite calcular el factorial de un número de manera eficiente mediante el uso de un bucle iterativo. El diagrama de flujo facilitó la comprensión del proceso, mostrando cada paso de la ejecución del programa.

Nombre: Gutiérrez López Bernardo

Matrícula: 2222904

Definición, características y ejemplos de diagramas de flujo

Definición

Un diagrama de flujo es una representación gráfica de un proceso, sistema o algoritmo mediante símbolos y flechas que indican la secuencia de pasos a seguir. Se utilizan para comprender, analizar, documentar y mejorar procesos en diferentes ámbitos, como la informática, la industria y la administración.

Características

1: Uso de símbolos estándar

- Óvalo: Inicio y fin del proceso
- Rectángulo: Representa una actividad o proceso
- Rombo: indica una decisión con opciones de "sí o no"
- Flechas: Señalan el flujo y la dirección del proceso
- Paralelogramo: Representa la entrada o salida de datos.

2: Flujo lógico y secuencial

- Debe seguir un orden claro y preciso.
- Puede incluir bifurcaciones según condiciones.

3: Claridad y simplicidad

- Facilita la comprensión del proceso sin necesidad de explicaciones adicionales.
- Evita el uso excesivo de símbolos o pasos innecesarios.

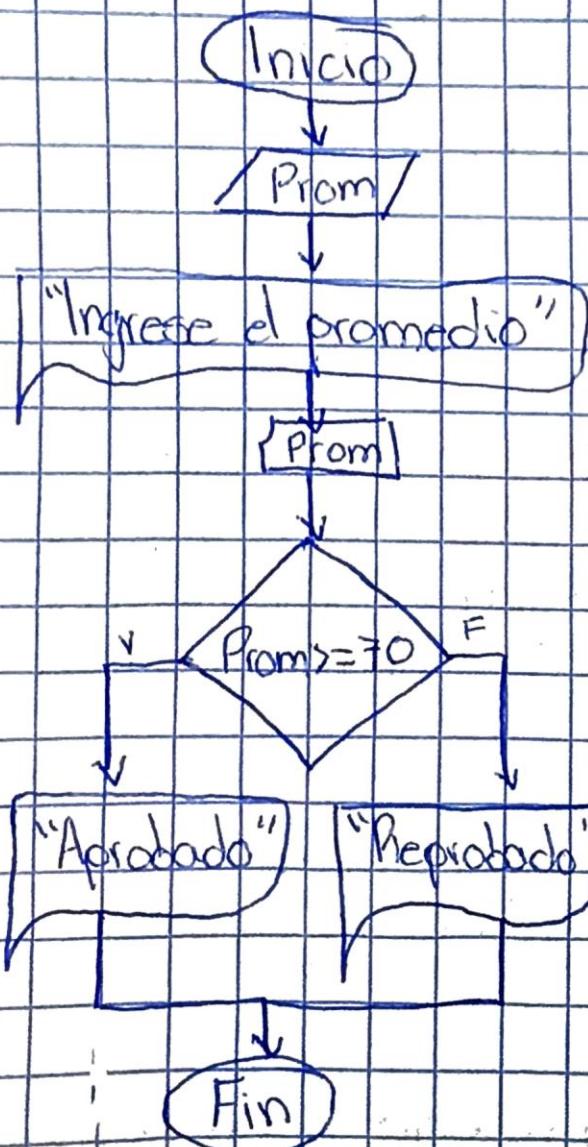
4)

Representación gráfica universal

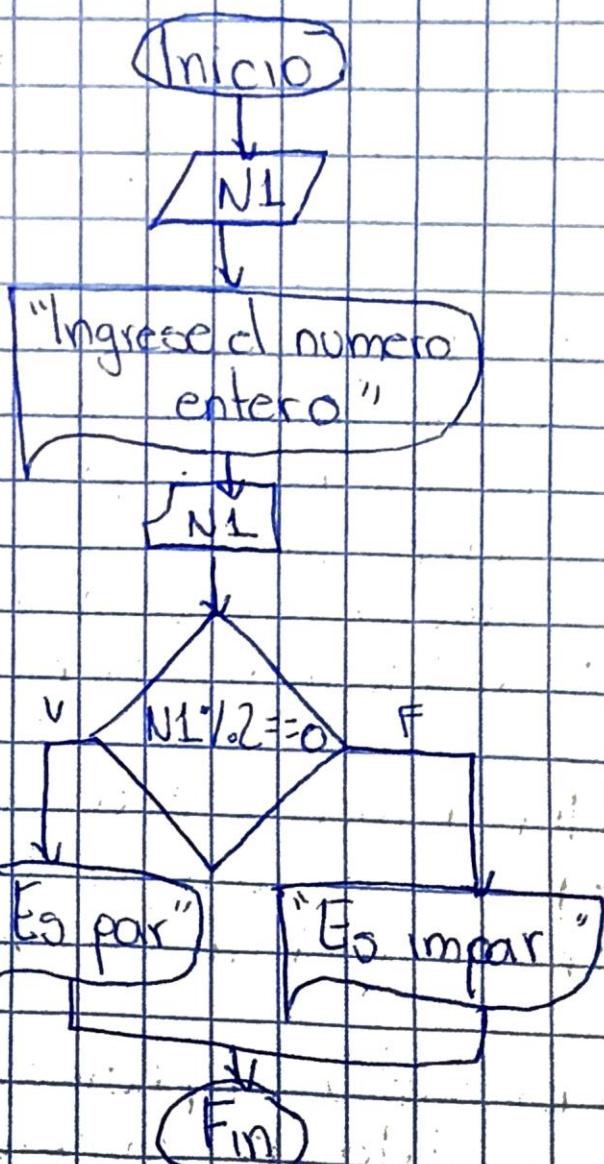
• Se utilizan en diferentes áreas y es comprensible sin importar el idioma

Ejemplos

1) Programa que al recibir el promedio de un alumno imprima "aprobado" si es ≥ 70 y en caso contrario imprima "reprobado".



2) Programa que al recibir un numero entero determine si es un número par e imprima es par.



Nombre : Gutiérrez López Bernardo
Matrícula: 2222904

Definición, características, propósito y ejemplos de lenguajes funcionales

Definición

Los lenguajes funcionales son un paradigma de programación basado en funciones matemáticas y en la evaluación de expresiones. Se centran en el uso de funciones puras, evitando cambios de estado y efectos secundarios. En este modelo, el código se trata como una serie de transformaciones de datos mediante funciones inmutables.

Características

1- Funciones puras

- No dependen de un estado externo y siempre producen el mismo resultado para la misma entrada

2: Inmutabilidad

- Los datos no se modifican en su lugar, se crean nuevas versiones con los cambios aplicados

3: Evaluación perezosa (Lazy Evaluation)

- Los valores solo se calculan cuando son necesarios, lo que optimiza el rendimiento.

4: Funciones de orden superior

- Las funciones pueden recibir otras funciones como argumentos o devolver funciones como resultado

5. Composición de funciones

- Se pueden combinar funciones pequeñas para formar funciones más complejas.

Propósito de los lenguajes funcionales

- Escribir código más claro y mantenible.
- Facilitar la concurrencia y paralelismo gracias a la inmutabilidad.
- Reducir errores al evitar efectos secundarios.
- Aplicaciones en inteligencia artificial, big data y sistemas distribuidos.

Ejemplos de lenguajes funcionales

- 1: Haskell: Lenguaje puramente funcional con evaluación perezosa
- 2: Lisp: Uno de los primeros lenguajes funcionales, base de Clojure y Scheme.
- 3: Clojure: Funcional sobre la máquina virtual de Java (JVM)
- 4: Erlang: Optimizado para sistemas concurrentes y de telecomunicaciones.
- 5: Scala: Mezcla de paradigmas funcional y orientado a objetos.
- 6: F#: Lenguaje funcional desarrollado de Microsoft.

Nombre: Gutiérrez López Bernardo
Matrícula: 2222904

Ejemplos de diagramas de flujo de recursión. Pilas y Listas

Diagrama de flujo de recursión

- Factorial

- Caso base

$$\text{factorial} = 1;$$

- Caso recursivo

```
for( int i=n; i>=1; i++ )
```

$$\text{factorial} *= i;$$

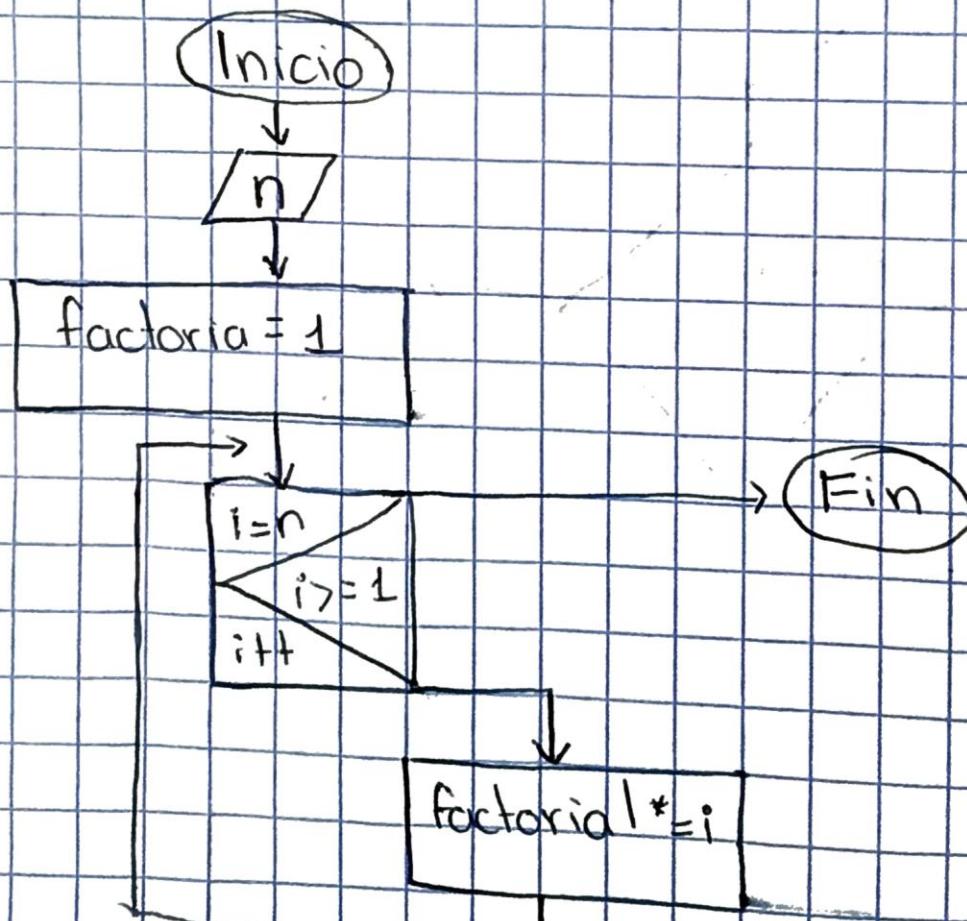


Diagrama de flujo de Pila

Notación postfixa

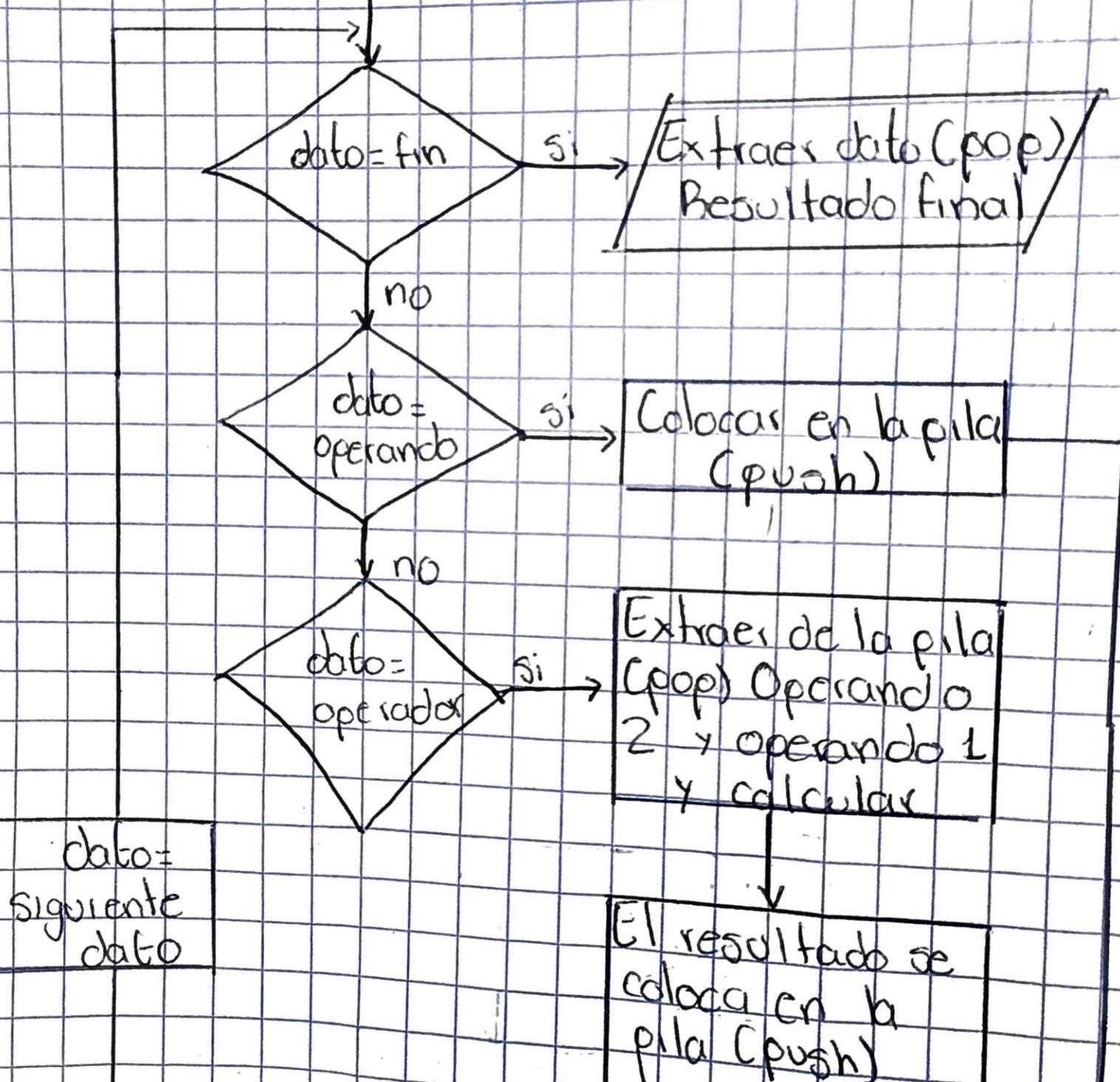


Diagrama de flujo de listas

Asignar valores a todos los elementos de un vector de cinco componentes.

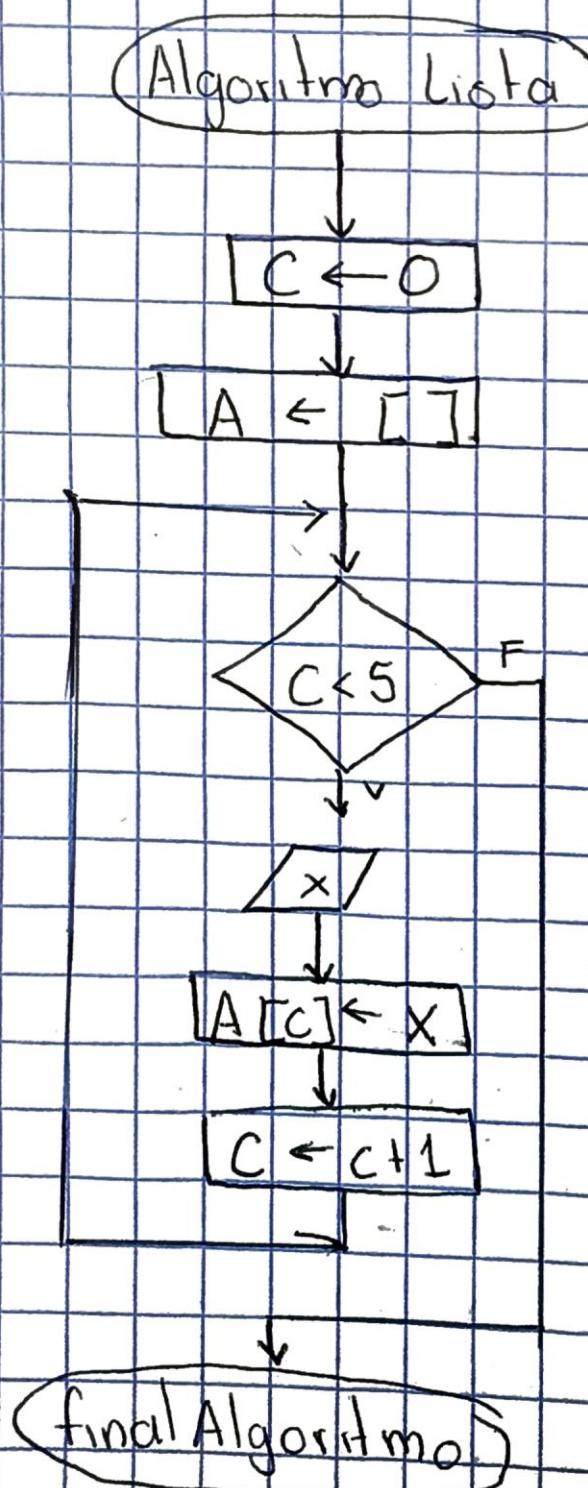
$$A[1] = 5$$

$$A[2] = 7$$

$$A[3] = 8$$

$$A[4] = 12$$

$$A[5] = 18$$



Nombre: Gutiérrez López Bernardo
Matrícula: 2222904

Haskell

¿Qué es Haskell?

Haskell es un lenguaje de programación puramente funcional, con evaluación perezosa y un sistema de tipos fuerte e inferido. Fue creado en 1990 por un comité de académicos e investigadores interesados en un lenguaje basado en cálculos matemáticos y lógica pura.

Principales características

- 1: Lenguaje puramente funcional: los programas se estructuran como funciones matemáticas sin efectos secundarios. No hay variables mutables ni instrucciones imperativas tradicionales como los bucles for o while.
- 2: Evaluación perezosa (Lazy Evaluation): Solo evalúa las expresiones cuando son necesarias. Esto permite definir estructuras de datos infinitas y mejorar la eficiencia del programa.
- 3: Sistema de Tipos fuerte e Inferido: Usa un sistema de tipos estático, lo que significa que los errores de tipo se detectan en tiempo de compilación.
- 4: Funciones de Orden superior: Las funciones pueden recibir otras funciones como parámetros o devolver funciones como resultado.

Ventajas

- Código más seguro
- Código más conciso y expresivo
- Facilidad para concurrencia y paralelismo
- Evaluación perezosa.

Desventajas

- Curva de aprendizaje alta
- Menos soporte empresarial
- Menos bibliotecas y herramientas.

Ejemplo de Código de Haskell factorial de un número con recursión

factorial :: Integer → Integer

factorial 0 = 1

factorial n = n * factorial (n - 1)

main :: IO ()

main = print (factorial 5) -- Salida: 120

Nombre: Gutiérrez López Bernardo
Matrícula 2222904

Scheme

¿Qué es Scheme?

Scheme es un lenguaje de programación funcional que pertenece a la familia de Lisp. Fue desarrollado en la década de 1970 como una versión más simple y elegante de Lisp, con un diseño minimalista y un fuerte énfasis en la recursión, el uso de funciones de orden superior y la evaluación de expresiones.

Principales características

- 1: Sintaxis minimalista: Usa una notación basada en listas, donde el operador aparece antes de los operandos (notación prefija).
- 2: Programación funcional y recursión: Promueve el uso de funciones puros y la recursión en lugar de bucles tradicionales.
- 3: Funciones de Orden superior: las funciones pueden recibir otras funciones como parámetros o devolver funciones como resultado.
- 4: Evaluación perezosa y apilada: permite definir estructuras que no se evalúan hasta que sean necesarias.
- 5: Tipado dinámico: No requiere que se declaren los tipos de datos explícitamente.

Ventajas

- Sintaxis simple y consistente
- Potente para programación funcional y recursión
- Flexible y extensible
- Pequeño y eficiente.

Desventajas

- Menos popular en la industria
- No es tan eficiente como otros lenguajes optimizados para rendimiento.
- Dificultad en manipulación de estructuras complejas.

Ejemplo de código de Scheme

Sumar los elementos de una lista con recursión

(define (suma-lista lst))

 (if (null? lst))

 0

 (+ (car lst) (suma-lista (cdr lst))))

(display (suma-lista '(1 2 3 4 5))) ; salida: 15

A7.Ej.Lenguaje Lógico

Nombre: Bernardo Gutiérrez López

Matrícula: 2222904

AT - Elaborar un reporte lenguaje lógico Prolog

Introducción

Los programas de decisión permiten definir sistemas donde, a partir de condiciones específicas se toman decisiones que conducen a ciertos resultados. El lenguaje Prolog se adapta perfectamente a este enfoque gracias a su naturaleza declarativa y su sistema de inferencia lógica. Este reporte presenta un programa de decisión que determina la recomendación de una actividad según el clima y el tiempo disponible.

Lógica del Programa

Se evalúan dos condiciones:

- Clima soleado, nublado o lluvioso
- Tiempo disponible poco (≤ 1 h), medio (1-3h), mucho (> 3 h)

Según estos datos el sistema recomendará una actividad.

Diagrama de Decisión (Tabla)

Clima	Tiempo	Actividad Recomendada
Soleado	Mucho	Ir a la playa
Soleado	Medio	Pasear en el parque
Soleado	Poco	Leer en el jardín
Nublado	Mucho	Ir al museo
Nublado	Medio	Ver una película
Nublado	Poco	Leer un libro
Lluvioso	Mucho	Cocinar
Lluvioso	Medio	Ver una serie
Lluvioso	Poco	Escuchar música

Código en Prolog

% Base de conocimiento

actividad(soleado, mucho, 'Ir a la playa').

actividad(soleado, medio, 'Pasear en el parque').

actividad(soleado, poco, 'Leer en el jardín').

actividad(nublado, mucho, 'Ir al museo').

actividad(nublado, medio, 'Ver una película').

actividad(nublado, poco, 'Leer un libro').

actividad(lluvioso, mucho, 'Cocinar').

actividad(lluvioso, medio, 'Ver una serie').

actividad(lluvioso, poco, 'Escuchar música').

% Reglas de clasificación de tiempo

Tiempo-disponible(Horas, mucho) :- Horas > 3.

Tiempo-disponible(Horas, medio) :- Horas = < 3, Horas > 1.

Tiempo-disponible(Horas, poco) :- Horas = < 1.

% Reglas principales para sugerir actividad

sugerir-actividad(Clima, Horas, Actividad) :-

Tiempo-disponible(Horas, Tiempo),

actividad(Clima, Tiempo, Actividad).

Consulta en Prolog

?- sugerir_actividad(soleado, 0.5, Actividad).

Actividad = 'Leer en el jardín'.

?- sugerir_actividad(lluvioso, 2, Actividad)

Actividad = 'Ver una serie'.

Conclusión

El programa implementa una lógica de decisión clara basada en condiciones climáticas y tiempo disponible, mostrando cómo Prolog permite representar decisiones de forma declarativa, flexible y sencilla.

A8. Reporte programas imperativos

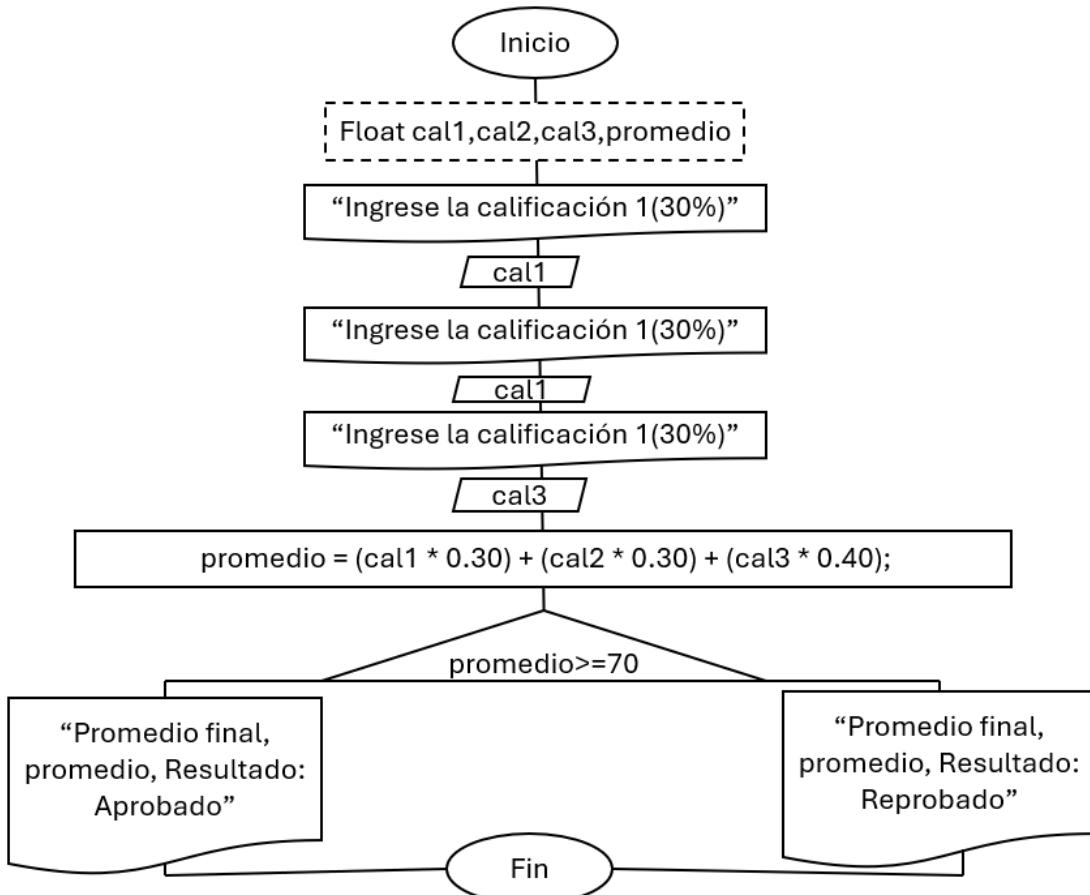
Objetivo: Analizar y describir programas sencillos en diferentes lenguajes imperativos, explicando su funcionamiento y visualizando su lógica mediante diagramas de flujo.

Programa 1 (Lenguaje C)

Descripción

El programa está diseñado para recibir tres calificaciones de un estudiante, cada una con un peso específico en el cálculo del promedio final. La primera calificación tiene un valor del 30%, la segunda también del 30%, y la tercera calificación vale el 40% del total. El usuario debe ingresar las tres calificaciones por teclado. Una vez ingresadas, el programa calcula el promedio ponderado multiplicando cada calificación por su respectivo porcentaje y sumando los resultados. Finalmente, el programa evalúa si el promedio es mayor o igual a 70. Si es así, se muestra en pantalla el mensaje "Aprobado", de lo contrario se muestra "Reprobado". Este tipo de lógica es común en sistemas de evaluación académica, donde no todas las actividades o exámenes tienen el mismo valor.

Diagrama de flujo



Código y salida

```
1 #include <stdio.h>
2 int main() {
3     float cal1, cal2, cal3, promedio;
4     // Entrada de calificaciones
5     printf("Ingrese la calificación 1 (30%): ");
6     scanf("%f", &cal1);
7     printf("Ingrese la calificación 2 (30%): ");
8     scanf("%f", &cal2);
9     printf("Ingrese la calificación 3 (40%): ");
10    scanf("%f", &cal3);
11    // Cálculo del promedio ponderado
12    promedio = (cal1 * 0.30) + (cal2 * 0.30) + (cal3 * 0.40);
13    // Mostrar resultado
14    printf("Promedio final: %.2f\n", promedio);
15    if (promedio >= 70) {
16        printf("Resultado: Aprobado\n");
17    } else {
18        printf("Resultado: Reprobado\n");
19    }
20    return 0;
21 }
```

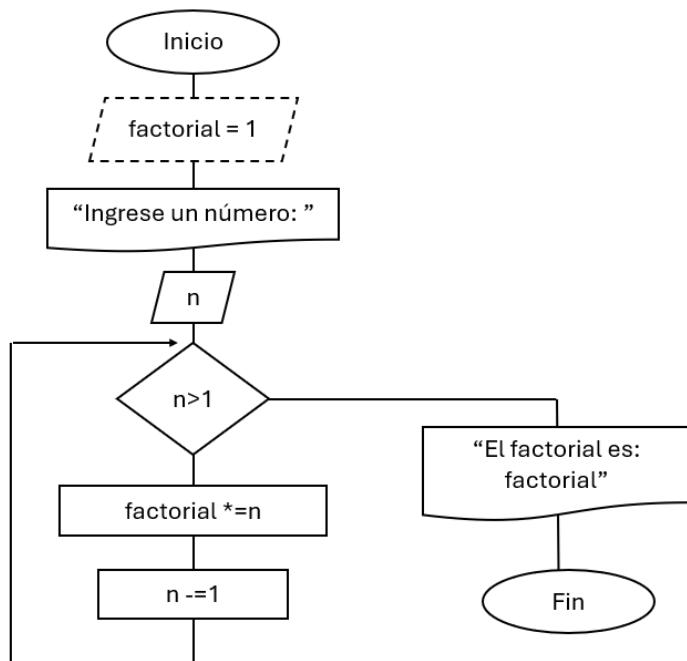
```
input
Ingrese la calificación 1 (30%): 50
Ingrese la calificación 2 (30%): 100
Ingrese la calificación 3 (40%): 100
Promedio final: 85.00
Resultado: Aprobado
```

Programa 2 (Python)

Descripción

Calcula el factorial de un número usando bucles while.

Diagrama de flujo



Código y salida

```
1 n = int(input("Ingrese un número: "))
2 factorial = 1
3
4 while n > 1:
5     factorial *= n
6     n -= 1
7
8 print("El factorial es:", factorial)
9
10
```

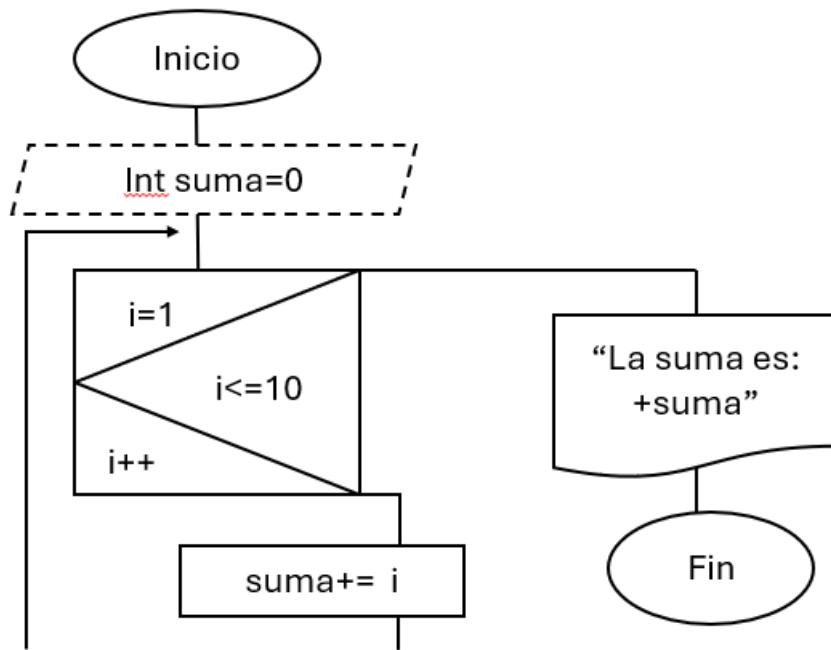
Ingrese un número: 12
El factorial es: 479001600

Programa 3 (Java)

Descripción

Suma los 10 primeros números naturales

Diagrama de flujo



Código y salida

```
1 public class Main {  
2     public static void main(String[] args) {  
3         int suma = 0;  
4  
5         for (int i = 1; i <= 10; i++) {  
6             suma += i;  
7         }  
8  
9         System.out.println("La suma es: " + suma);  
10    }  
11 }  
12
```



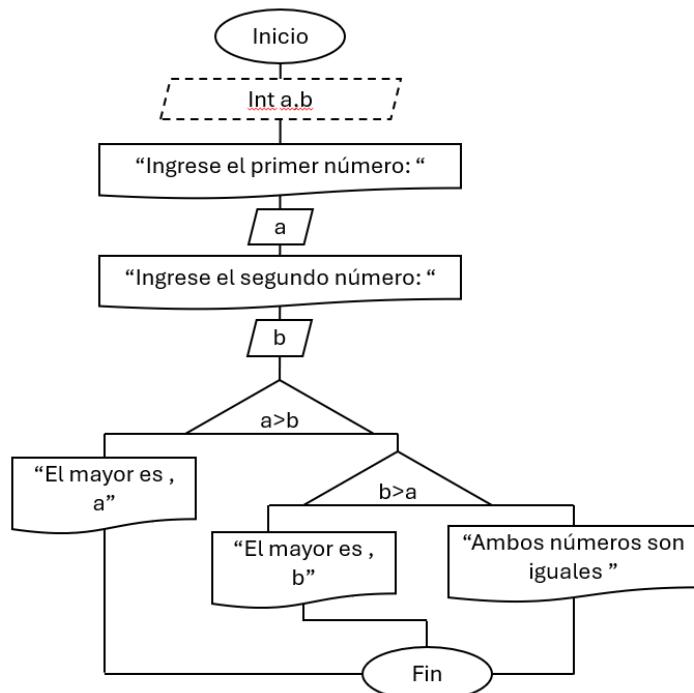
La suma es: 55

Programa 4 (Pascal)

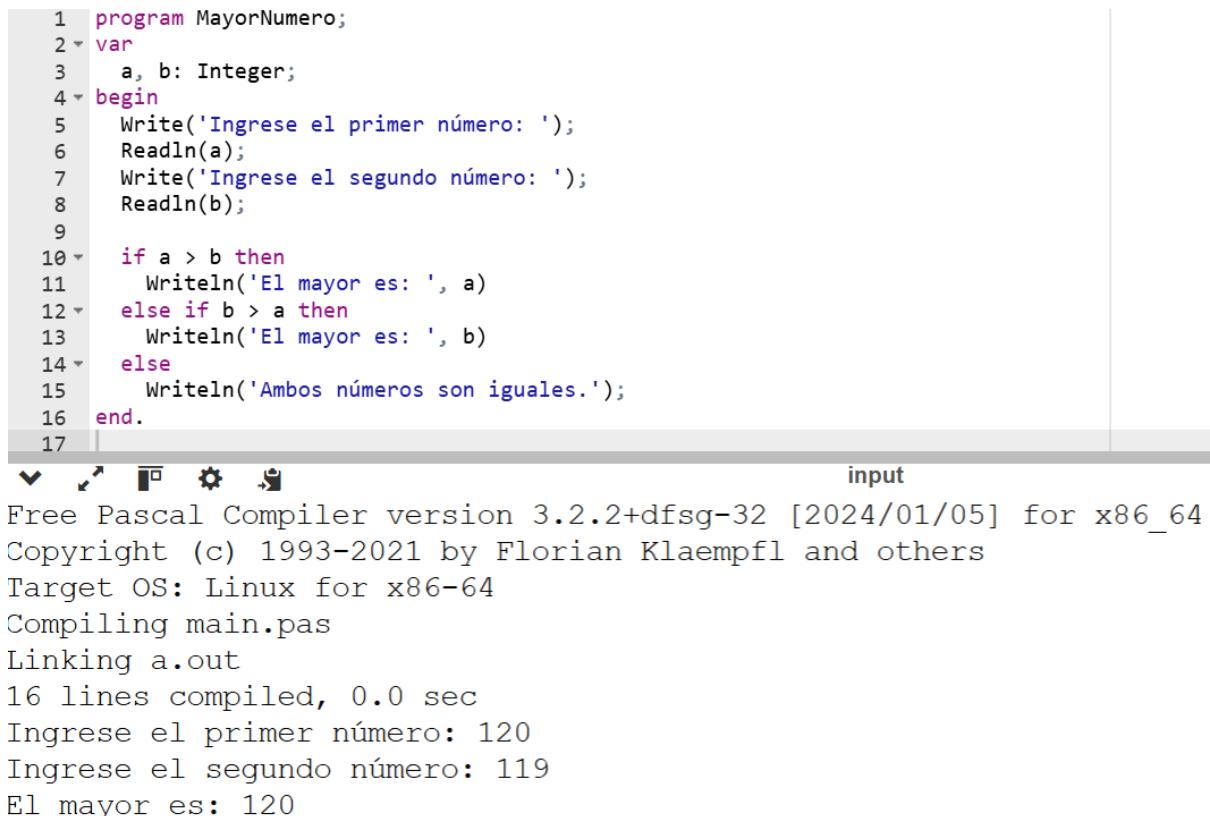
Descripción

Recibe dos números, los lee y muestra cual es el mayor o si son iguales.

Diagrama de flujo



Código y salida



```
1 program MayorNumero;
2 var
3   a, b: Integer;
4 begin
5   Write('Ingrese el primer número: ');
6   Readln(a);
7   Write('Ingrese el segundo número: ');
8   Readln(b);
9
10  if a > b then
11    Writeln('El mayor es: ', a)
12  else if b > a then
13    Writeln('El mayor es: ', b)
14  else
15    Writeln('Ambos números son iguales.');
16 end.
17
```

Free Pascal Compiler version 3.2.2+dfsg-32 [2024/01/05] for x86_64
Copyright (c) 1993-2021 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling main.pas
Linking a.out
16 lines compiled, 0.0 sec
Ingrese el primer número: 120
Ingrese el segundo número: 119
El mayor es: 120

Conclusión

A través del desarrollo de estos cuatro programas en lenguajes imperativos diferentes (C, Java, Python y Pascal), se pudo comprobar cómo, a pesar de las diferencias en sintaxis y estructura, todos comparten los mismos principios fundamentales de la programación imperativa: el uso de variables, estructuras de control (como condicionales y bucles) y entrada/salida de datos. Cada lenguaje tiene sus particularidades y ventajas: C es potente y cercano al hardware; Java es multiplataforma y orientado a objetos; Python destaca por su simplicidad y legibilidad; y Pascal, aunque menos utilizado actualmente, sigue siendo valioso para fines educativos por su claridad estructural. En conjunto, estos ejercicios fortalecieron la comprensión de la lógica algorítmica y demostraron cómo resolver problemas similares en distintos entornos, ampliando la versatilidad y capacidad de adaptación del programador.

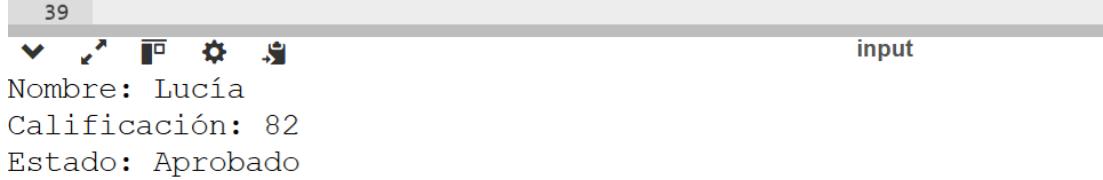
A9.Ej.Lenguaje orientado a objetos

Introducción

La programación orientada a objetos (POO) es un paradigma que organiza el código en torno a clases y objetos, permitiendo representar conceptos del mundo real de manera estructurada. Este paradigma facilita el mantenimiento, la reutilización del código y la escalabilidad del software. En este reporte se explicarán los conceptos de tipos de datos, objetos y encapsulación, utilizando un único programa escrito en Python que simula el manejo de información de un alumno.

Programa

```
1  class Alumno:
2      def __init__(self, nombre: str, calificacion: int):
3          # Atributos privados (encapsulados)
4          self.__nombre = nombre          # Tipo de dato: str
5          self.__calificacion = calificacion  # Tipo de dato: int
6
7      # Getter para nombre
8      def get_nombre(self) -> str:
9          return self.__nombre
10
11     # Setter para nombre
12     def set_nombre(self, nombre: str):
13         self.__nombre = nombre
14
15     # Getter para calificación
16     def get_calificacion(self) -> int:
17         return self.__calificacion
18
19     # Setter para calificación
20     def set_calificacion(self, calificacion: int):
21         self.__calificacion = calificacion
22
23     # Método que evalúa si está aprobado
24     def esta_aprobado(self) -> bool:
25         return self.__calificacion >= 70  # Tipo de dato: bool
26
27     # Crear un objeto de la clase Alumno
28     alumno1 = Alumno("Lucía", 82)
29
30     # Mostrar información usando getters
31     print("Nombre:", alumno1.get_nombre())           # Tipo de dato: str
32     print("Calificación:", alumno1.get_calificacion()) # Tipo de dato: int
33
34     # Evaluar aprobación
35     if alumno1.esta_aprobado():                      # Tipo de dato: bool
36         print("Estado: Aprobado")
37     else:
38         print("Estado: Reprobado")
39
```



The screenshot shows a code editor interface with the Python code in the main pane and its execution results in a separate pane below. The code defines a class 'Alumno' with private attributes '__nombre' and '__calificacion'. It includes methods for getting and setting these values, and a method 'esta_aprobado' to check if the grade is 70 or higher. An object 'alumno1' is created with name 'Lucía' and grade 82. The output pane shows the printed values: 'Nombre: Lucía', 'Calificación: 82', and 'Estado: Aprobado'.

Nombre: Lucía
Calificación: 82
Estado: Aprobado

Tipos de datos utilizados

Elemento	Tipo de dato	Descripción
nombre	str	Cadena de texto que almacena el nombre del alumno.
calificación	int	Número entero que representa la nota.
esta_aprobado()	bool	Devuelve True o False según la nota.

Objeto

- El objeto alumno1 es una instancia de la clase Alumno.
- Tiene estado (nombre, calificacion) y comportamiento (esta_aprobado())

Encapsulación

- Los atributos __nombre y __calificacion están encapsulados (privados).
- Se accede a ellos mediante métodos públicos: get_ y set_, lo cual:
 - >Protege la integridad de los datos.
 - >Permite aplicar reglas (como validar rangos, si se desea).

Conclusión

Este programa muestra cómo un solo ejemplo puede reflejar los tres conceptos clave de la programación orientada a objetos:

- Tipos de datos, para manejar información estructurada.
- Objetos, como instancias que encapsulan atributos y métodos.
- Encapsulación, para proteger los datos internos y permitir su acceso controlado.

Gracias al paradigma (estilo de programación) POO, este programa es fácil de mantener, reutilizar y ampliar, siendo una base sólida para desarrollar aplicaciones más complejas con buena estructura y claridad.

Programa

Tipo de lenguaje

HTML,CSS Y JavaScript

Descripción del lenguaje

HTML, CSS y JavaScript son los tres lenguajes fundamentales para crear sitios web.

- HTML, que significa HyperText Markup Language, se encarga de definir la estructura del contenido. Es como el esqueleto de una página: con él decides qué elementos aparecen, como títulos, párrafos, imágenes, botones o formularios. Sin HTML, simplemente no habría contenido visible.
- CSS, o Cascading Style Sheets, es el lenguaje que se usa para dar estilo a esa estructura. Sirve para modificar cómo se ve todo lo que HTML define. Por ejemplo, puedes cambiar los colores, las tipografías, el tamaño de los textos, el espacio entre los elementos o incluso agregar efectos visuales como sombras o animaciones. En esencia, CSS transforma una página básica en algo visualmente atractivo.
- JavaScript, por otro lado, es lo que le da vida e interactividad a la página. Gracias a este lenguaje, una web puede reaccionar a lo que hace el usuario: mostrar mensajes, cambiar contenido sin recargar la página, validar formularios antes de enviarlos, controlar animaciones, o incluso conectarse con servidores para obtener o enviar datos en tiempo real. JavaScript convierte un sitio estático en una experiencia dinámica y funcional.

En conjunto, HTML pone los bloques, CSS los pinta y acomoda, y JavaScript los hace reaccionar y comportarse como tú quieras. Así se construye prácticamente todo lo que ves y usas en internet.

Descripción del Programa

Este programa muestra una página web interactiva que sirve para visualizar y calcular el promedio ponderado de calificaciones de las distintas actividades del semestre en la materia de Lenguaje de Programación y lab.

Está diseñado para que el usuario (en este caso yo: Bernardo Gutiérrez López) pueda ver los enlaces a cada actividad entregada (como exámenes y reportes), ingresar la calificación obtenida en cada una, y luego presionar un botón para que el sistema calcule el promedio total, considerando que cada actividad tiene un peso distinto en la evaluación final.

La estructura de la página se crea con HTML, el diseño visual se logra con CSS (por ejemplo, colores, bordes redondeados, espaciado, etc.), y la parte dinámica, es decir, la lógica que toma los números ingresados y calcula el promedio ponderado cuando se presiona el botón está hecha con JavaScript.

En resumen, esta página es como una calculadora personalizada de promedio académico, donde cada actividad tiene un valor específico, y yo puedo ver mis archivos, ingresar mis calificaciones y saber cuál es mi promedio final en base a eso.

Análisis del programa (pseudocódigo)

Inicio del documento HTML

Definir idioma: español

Configurar la codificación de caracteres (UTF-8)

Título de la página: "Actividades del Semestre"

Estilos (CSS):

- Se define el diseño general de la página: fondo gris claro, textos centrados, caja blanca con sombra
- Se da formato a cada "actividad" (bloques con enlace y campo de calificación)
- Se define estilo para el botón y el resultado del promedio

En el cuerpo de la página:

Mostrar título con el nombre del estudiante

Crear un contenedor llamado "menu"

Dentro del menú:

Para cada una de las 9 actividades:

- Mostrar un enlace al archivo de la actividad
- Mostrar una caja de texto donde se puede ingresar una calificación (número entre 0 y 100)

Agregar un botón llamado "Calcular Promedio"

Agregar un espacio donde se mostrará el resultado

Fin del cuerpo

Fin del HTML

Explicación del código

1.- Estructura del documento HTML que define que el documento es HTML5 y que el idioma es español.

```
5 LENGPROG.html X
C: > xampp > htdocs > formulario > 5 LENGPROG.html > ...
1 <!DOCTYPE html>
2 <html lang="es">
```

2.- Encabezado que indica la codificación de caracteres (UTF-8) y el título de la pestaña del navegador. Dentro del <style>, se incluye todo el diseño CSS de la página.

```
3 <head>
4   <meta charset="UTF-8">
5   <title>Actividades del Semestre</title>
6 <style>
```

3.- Estilos CSS que le da estilo general a la página (fondo claro, tipografía Arial, y márgenes internos).

```
7   body {
8     font-family: Arial, sans-serif;
9     background: #f2f2f2;
10    padding: 40px;
11  }
```

4.- Centra el título principal y cambia su color.

```
12 <h1>
13   text-align: center;
14   color: #333;
15 </h1>
```

5.- Crea una caja blanca centrada con bordes redondeados y sombra.

```
16 .menu {
17   max-width: 700px;
18   margin: auto;
19   background: white;
20   padding: 20px;
21   border-radius: 10px;
22   box-shadow: 0 0 10px rgba(0,0,0,0.1);
23 }
```

6.- Estiliza cada bloque de actividad con margen, relleno y diseño tipo tarjeta.

```
24 | .actividad {  
25 |   display: flex;  
26 |   align-items: center;  
27 |   justify-content: space-between;  
28 |   margin: 10px 0;  
29 |   padding: 10px;  
30 |   border: 1px solid #ccc;  
31 |   border-radius: 5px;  
32 |   background: #f9f9f9;  
33 | }
```

7.- Estilo para los enlaces: sin subrayado, color azul.

```
34 | .actividad a {  
35 |   text-decoration: none;  
36 |   color: #007bff;  
37 |   margin-right: 10px;  
38 | }
```

8.- Diseño para las cajas donde ingresas las calificaciones.

```
39 | input[type="number"] {  
40 |   width: 60px;  
41 |   padding: 5px;  
42 |   border: 1px solid #ccc;  
43 |   border-radius: 5px;  
44 | }
```

9.- Botón azul para calcular el promedio.

```
45 ▾ #calcularBtn {  
46 |   display: block;  
47 |   margin: 20px auto;  
48 |   padding: 10px 20px;  
49 |   background: #007bff;  
50 |   color: white;  
51 |   border: none;  
52 |   border-radius: 5px;  
53 |   cursor: pointer;  
54 | }
```

10.- Caja donde se muestra el resultado final del promedio.

```
55 | #promedio {  
56 |   text-align: center;  
57 |   font-weight: bold;  
58 |   margin-top: 20px;  
59 | }  
60 | </style>  
61 | </head>
```

11.-Contenido del cuerpo “body”, título principal y contenedor de todas las actividades.

```
62  <body>
63  | <h1>Mis Actividades del Semestre - Bernardo Gutiérrez López - 2222904</h1>
64  | <div class="menu">
```

12.- Cada bloque como este representa una actividad, con su respectivo archivo PDF y un campo para ingresar la calificación. Esto se repite para 9 actividades, cada una con su archivo y su input para la nota.

```
65  | <div class="actividad">
66  | | <a href="MC-2222904-LENGPROG.pdf" target="_blank">Actividad 1 - Examen MC</a>
67  | | <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
68  | </div>
```

13.- Botón para iniciar el cálculo del promedio y un lugar para mostrar el resultado.

```
● 102  | <button id="calcularBtn">Calcular Promedio</button>
103  | <div id="promedio"></div>
104  | </div>
105
```

14.- Script JavaScript para calcular promedio, cuando se hace clic en el botón, se ejecuta esta función.

```
105
106  | <script>
107  | | document.getElementById("calcularBtn").addEventListener("click", function () {
```

15.- Se obtienen todos los campos donde el usuario ingresó calificaciones.

```
108  | | const notas = document.querySelectorAll(".nota");
109  | |
```

16.- Se definen los pesos correspondientes para cada una de las 9 actividades.

```
110  | | const pesos = [0.25, 0.25, 0.20, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05];
111  | |
```

17.- Variables para acumular los cálculos.

```
112  | | let sumaPonderada = 0;
113  | | let sumaPesos = 0;
114  | |
```

18.- Recorre cada calificación: Si es un número válido, la multiplica por su peso y va sumando el total ponderado y los pesos usados.

```
115 |  notas.forEach((nota, index) => {  
116 |  |  const valor = parseFloat(nota.value);  
117 |  |  if (!isNaN(valor)) {  
118 |  |  |  sumaPonderada += valor * pesos[index];  
119 |  |  |  sumaPesos += pesos[index];  
120 |  | }  
121 |  |});  
122 |
```

19.- Se calcula el promedio final y se muestra redondeado a dos decimales.

```
123 |  |  const promedio = sumaPesos > 0 ? (sumaPonderada / sumaPesos).toFixed(2) : 0;  
124 |  |  document.getElementById("promedio").textContent = "Promedio ponderado: " + promedio;  
125 |  |});  
126 |  </script>  
127 |</body>  
128 |</html>  
130 |
```

Código completo

```
1  <!DOCTYPE html>
2  <html lang="es">
3  <head>
4      <meta charset="UTF-8">
5      <title>Actividades del Semestre</title>
6      <style>
7          body {
8              font-family: Arial, sans-serif;
9              background: #f2f2f2;
10             padding: 40px;
11         }
12         h1 {
13             text-align: center;
14             color: #333;
15         }
16         .menu {
17             max-width: 700px;
18             margin: auto;
19             background: white;
20             padding: 20px;
21             border-radius: 10px;
22             box-shadow: 0 0 10px rgba(0,0,0,0.1);
23         }
24         .actividad {
25             display: flex;
26             align-items: center;
27             justify-content: space-between;
28             margin: 10px 0;
29             padding: 10px;
30             border: 1px solid #ccc;
31             border-radius: 5px;
32             background: #f9f9f9;
33         }
34         .actividad a {
35             text-decoration: none;
36             color: #007bff;
37             margin-right: 10px;
38         }
39         input[type="number"] {
40             width: 60px;
41             padding: 5px;
42             border: 1px solid #ccc;
43             border-radius: 5px;
44         }
45         #calcularBtn {
```

```
46 |     display: block;
47 |     margin: 20px auto;
48 |     padding: 10px 20px;
49 |     background: #007bff;
50 |     color: white;
51 |     border: none;
52 |     border-radius: 5px;
53 |     cursor: pointer;
54 |
55 < #promedio {
56 |     text-align: center;
57 |     font-weight: bold;
58 |     margin-top: 20px;
59 | }
60 </style>
61 </head>
62 <body>
63 <h1>Mis Actividades del Semestre - Bernardo Gutiérrez López - 2222904</h1>
64 <div class="menu">
65 <div class="actividad">
66 |     <a href="MC-2222904-LENGPROG.pdf" target="_blank">Actividad 1 - Examen MC</a>
67 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
68 | </div>
69 <div class="actividad">
70 |     <a href="EXAMEN ORDINARIO" target="_blank">Actividad 2 - Examen Ordinario</a>
71 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
72 | </div>
73 <div class="actividad">
74 |     <a href="PIA" target="_blank">Actividad 3 - PIA</a>
75 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
76 | </div>
77 <div class="actividad">
78 |     <a href="A4-2222904-Gutiérrez López Bernardo-LENGPRO.pdf" target="_blank">Actividad 4 - Reporte conceptos fund.</a>
79 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
80 | </div>
81 <div class="actividad">
82 |     <a href="A5-2222904-Gutiérrez López Bernardo-LENGPRO.pdf" target="_blank">Actividad 5 - Ej. Lenguaje Script</a>
83 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
84 | </div>
85 <div class="actividad">
86 |     <a href="A6-2222904-Gutiérrez López Bernardo-LENGPRO.pdf" target="_blank">Actividad 6 - Ej. Lenguaje funcional</a>
```

```
87 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
88 |   </div>
89 |   <div class="actividad">
90 |     <a href="A7-2222904-Gutiérrez Bernardo-LENGPRO.pdf" target="_blank">Actividad 7 - Ej. Lenguaje lógico</a>
91 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
92 |   </div>
93 |   <div class="actividad">
94 |     <a href="A8-Reporte programas imperativos-LENGPRO-2222904.pdf" target="_blank">Actividad 8 - Reporte Prog. Imperativos</a>
95 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
96 |   </div>
97 |   <div class="actividad">
98 |     <a href="A9-Ej.Lenguaje orientado a objetos-LENGPRO.pdf" target="_blank">Actividad 9 - Ej. Lenguaje Orientado a Objetos</a>
99 |     <input type="number" min="0" max="100" class="nota" placeholder="Calificación">
100 |   </div>
101 |
102 |   <button id="calcularBtn">Calcular Promedio</button>
103 |   <div id="promedio"></div>
104 | </div>
105 |
106 | <script>
107 |   document.getElementById("calcularBtn").addEventListener("click", function () {
108 |     const notas = document.querySelectorAll(".nota");
109 |
110 |     const pesos = [0.25, 0.25, 0.20, 0.05, 0.05, 0.05, 0.05, 0.05, 0.05];
111 |
112 |     let sumaPonderada = 0;
113 |     let sumaPesos = 0;
114 |
115 |     notas.forEach((nota, index) => {
116 |       const valor = parseFloat(nota.value);
117 |       if (!isNaN(valor)) {
118 |         sumaPonderada += valor * pesos[index];
119 |         sumaPesos += pesos[index];
120 |       }
121 |     });
122 |
123 |     const promedio = sumaPesos > 0 ? (sumaPonderada / sumaPesos).toFixed(2) : 0;
124 |     document.getElementById("promedio").textContent = "Promedio ponderado: " + promedio;
125 |   });
126 | </script>
127 | </body>
128 | </html>
129 |
130 |
```

Ejecución del programa

Mis Actividades del Semestre - Bernardo Gutiérrez López - 2222904

Actividad 1 - Examen MC	Califica
Actividad 2 - Examen Ordinario	Califica
Actividad 3 - PIA	Califica
Actividad 4 - Reporte conceptos fund.	Califica
Actividad 5 - Ej. Lenguaje Script	Califica
Actividad 6 - Ej. Lenguaje funcional	Califica
Actividad 7 - Ej. Lenguaje lógico	Califica
Actividad 8 - Reporte Prog. Imperativos	Califica
Actividad 9 - Ej. Lenguaje Orientado a Objetos	Califica

Calcular Promedio

Promedio ponderado: 0

Mis Actividades del Semestre - Bernardo Gutiérrez López - 2222904

Actividad 1 - Examen MC	100
Actividad 2 - Examen Ordinario	0
Actividad 3 - PIA	100
Actividad 4 - Reporte conceptos fund.	0
Actividad 5 - Ej. Lenguaje Script	0
Actividad 6 - Ej. Lenguaje funcional	0
Actividad 7 - Ej. Lenguaje lógico	0
Actividad 8 - Reporte Prog. Imperativos	100
Actividad 9 - Ej. Lenguaje Orientado a Objetos	0

Calcular Promedio

Promedio ponderado: 50.00

Examen MC-Lenguajes de Programación

Nombre: Bernardo Gutiérrez López
Matrícula: 2222904
Grupo: 005
Hora: M5
Fecha: 19/03/2025

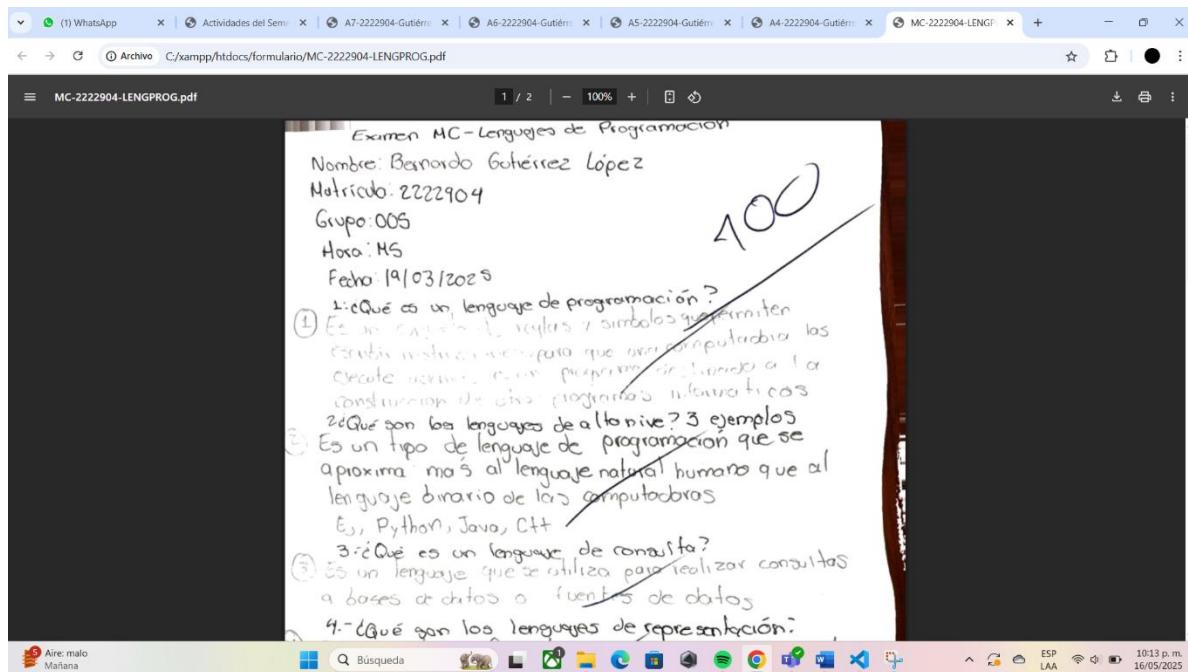
1.-Qué es un lenguaje de programación?
Es un conjunto de reglas y símbolos que permiten escribir instrucciones para que una computadora las execute automáticamente programando la secuencia de los pasos de ejecución de un programa informático.

2.-Qué son los lenguajes de alto nivel? 3 ejemplos
Es un tipo de lenguaje de programación que se aproxima más al lenguaje natural humano que al lenguaje binario de los computadores.
Ej: Python, Java, C++

3.-¿Qué es un lenguaje de consulta?
Es un lenguaje que se utiliza para realizar consultas a bases de datos o fuentes de datos.

4.-¿Qué son los lenguajes de representación?

A 100

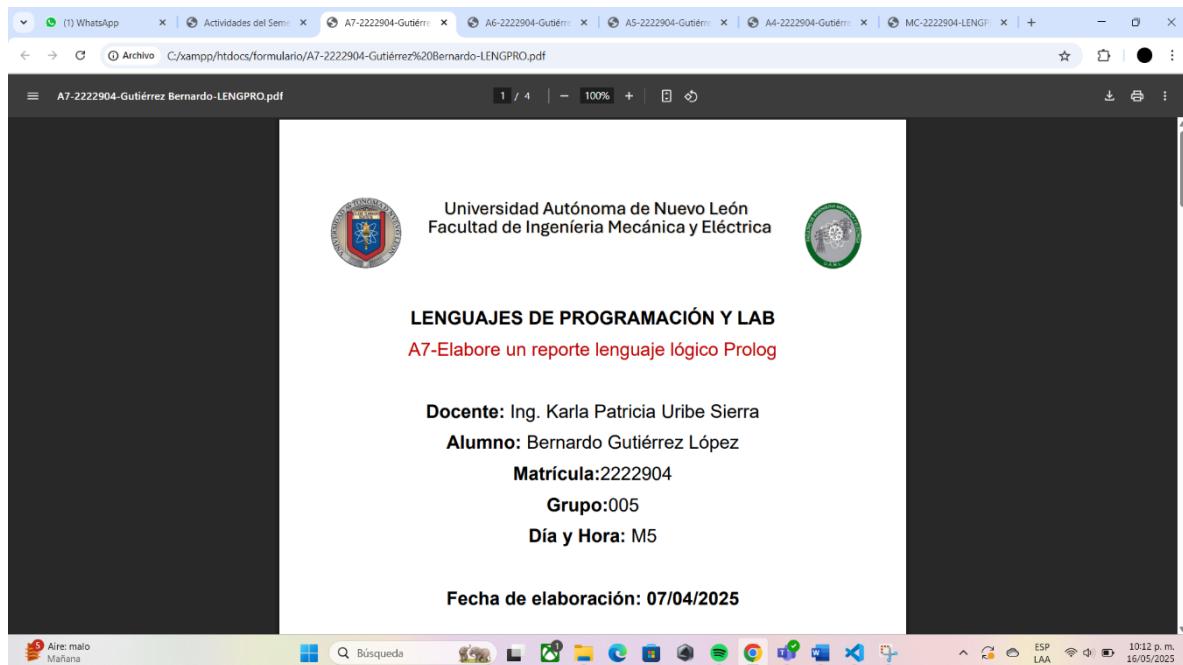


Universidad Autónoma de Nuevo León
Facultad de Ingeniería Mecánica y Eléctrica

LENGUAJES DE PROGRAMACIÓN Y LAB
A7-Elabore un reporte lenguaje lógico Prolog

Docente: Ing. Karla Patricia Uribe Sierra
Alumno: Bernardo Gutiérrez López
Matrícula: 2222904
Grupo: 005
Día y Hora: M5

Fecha de elaboración: 07/04/2025

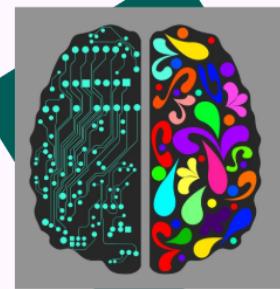


Exposición

Funcion de un lenguaje logico

Función Principal

La principal función de un lenguaje lógico es formalizar el razonamiento, es decir, proporcionar una estructura precisa y sin ambigüedades para representar argumentos, validar inferencias y demostrar teoremas.



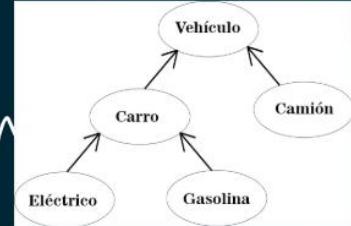
Otras funciones

- **Expresión de conocimiento:** Un lenguaje lógico permite representar hechos, relaciones y estructuras de manera formal.
- **Inferencia y deducción:** Mediante reglas lógicas, se pueden obtener conclusiones válidas a partir de un conjunto de premisas.
- **Verificación de validez:** Uno de los objetivos principales de un lenguaje lógico es comprobar la validez de argumentos y proposiciones.
- **Automatización del razonamiento:** Los lenguajes lógicos permiten el desarrollo de sistemas automáticos que razonan sin intervención humana.
- **Formalización de matemáticas y ciencias:** En disciplinas como la matemática y la ciencia computacional, los lenguajes lógicos se utilizan para definir estructuras, axiomas y teoremas con precisión.

HERENCIA

Definición

La herencia es un principio de la Programación Orientada a Objetos (POO) que permite a una clase (llamada subclase o clase hija) heredar atributos y métodos de otra clase (llamada superclase o clase padre). Esto promueve la reutilización de código y la creación de jerarquías de clases.



Características principales

- Reutilización de código: La subclase no necesita redefinir atributos o métodos ya definidos en la superclase.
- Jerarquía de clases: Permite organizar las clases en estructuras más comprensibles.
- Extensibilidad: Se pueden agregar nuevos atributos y métodos en la subclase sin modificar la superclase.
- Polimorfismo: La subclase puede sobrescribir métodos de la superclase para adaptarlos a sus necesidades.

Conceptos clave

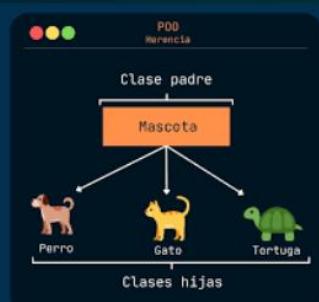
- **Superclase (Clase Padre):** Es la clase base de la cual otras clases pueden heredar. Contiene atributos y métodos que pueden ser compartidos con sus clases derivadas.
- **Subclase (Clase Hija):** Es la clase que hereda los atributos y métodos de la superclase, pudiendo también agregar o modificar funcionalidades.
- **Herencia Simple:** Una subclase hereda de una única superclase.
- **Herencia Múltiple:** Una subclase hereda de múltiples superclases (disponible en algunos lenguajes como Python).
- **Sobreescritura de Métodos (Overriding):** Ocurre cuando una subclase redefine un método de la superclase para modificar su comportamiento.

Ejemplo

Imaginemos que tenemos una familia de animales. Todos los animales tienen algo en común: tienen un nombre y pueden hacer sonidos. Entonces, primero hacemos una "clase" general que se llama Animal.

Después, hacemos subclases más específicos para perros y gatos. Los perros y los gatos son animales, así que heredan las cosas que un animal puede hacer, pero cada uno hace su propio sonido especial: el perro dice "¡Guau!" y el gato dice "¡Miau!".

Así, en lugar de repetir todo desde cero para cada animal, usamos la clase general y solo cambiamos lo que es diferente. Eso es herencia: aprovechar lo que ya existe y agregarle o cambiarle algunas cosas.



• Código del Ejemplo (Python)

```
1 # Clase base o padre
2 class Animal:
3     def __init__(self, nombre):
4         self.nombre = nombre
5
6     def hablar(self):
7         print("El animal hace un sonido")
8
9 # Clase hija que hereda de Animal
10 class Perro(Animal):
11     def hablar(self):
12         print(f"{self.nombre} dice: ¡Guau!")
13
14 # Clase hija que hereda de Animal
15 class Gato(Animal):
16     def hablar(self):
17         print(f"{self.nombre} dice: ¡Miau!")
18
19 # Usamos las clases
20 mi_perro = Perro("Firulais")
21 mi_gato = Gato("Misu")
22
23 mi_perro.hablar() # Firulais dice: ¡Guau!
24 mi_gato.hablar() # Misu dice: ¡Miau!
25
```

Dinámica

Dinamica

1. ¿Qué es un lenguaje lógico?

- A) Está basado en la lógica formal y se utiliza principalmente en inteligencia artificial.
- B) Es un paradigma de programación que se enfoca en la ejecución secuencial de instrucciones.
- C) Es un lenguaje que permite la manipulación de datos en bases de datos relacionales.

2. ¿Cuáles son los lenguajes lógicos más comunes?

- A) Prolog y Datalog.
- B) Lisp y Haskell.
- C) SQL y NoSQL.

3.-¿Qué son los hechos, reglas y consultas en un lenguaje lógico?

4. ¿Qué es la lógica computacional?

- A) Se centra en el uso de la lógica para resolver problemas y formalizar sistemas de razonamiento.
- B) Es el estudio de los sistemas numéricos utilizados en cálculos computacionales.
- C) Se encarga de diseñar modelos de optimización en estructuras de datos.

5. ¿Cuáles son las características de la lógica computacional?

- A) Formalización, automatización y verificación.
- B) Análisis sintáctico, abstracción y herencia de datos.
- C) Modularidad, concurrencia y manejo de excepciones.

¿Por que esta mal la siguiente regla del código?

```
% Hechos de padres  
padre(juan, carlos).  
padre(carlos, pedro).  
  
% Regla para determinar el abuelo por parte de padre  
abuelo(X, Y) :- padre(X, Y), padre(Y, Z).
```

Preguntas

```
1 public class Ejemplo {  
2     public static void main(String args) {  
3         System.out.println("Hola mundo");  
4     }  
5 }
```

¿Qué error tiene el siguiente código?

- A) Falta un punto y coma en el println.
- B) El método main está mal definido.
- C) La clase no puede llamarse Ejemplo.
- D) No se puede usar System.out.println dentro de main.

06:38 a.m.

Preguntas

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

¿Qué responderá Prolog si se hace la siguiente consulta?

- A) true.
- B) false.
- C) juan es padre de luis, no abuelo.
- D) Error de sintaxis.

Preguntas

```
1 public class Main {  
2     public static void main(String args) {  
3         System.out.println("Hola mundo");  
4     }  
5 }
```

- A) Porque System.out.println no está bien escrito.
- B) Porque falta un punto y coma.
- C) Porque el código no está dentro del método main.
- D) Porque Main debe escribirse en minúsculas.

Preguntas

¿Cuál fue el nombre original con el que se referían al lenguaje que hoy conocemos como C++?

- A. C evolucionando
- B. C mejorado
- C. C con clases
- D. C orientado

Preguntas

¿Quién le dio el nombre de C++ al lenguaje?

- A. Bjarne Stroustrup
- B. Dennis Ritchie
- C. Rick Mascitti
- D. Ken Thompson

¿Nombre completo de la Inge?

- A)** Karla Patricia Uribe Guadalupe
- B)** Patricia Sierra Rivera
- C)** Karla Patricia Uribe Sierra
- D)** Carla Patricia Uribe Sanchez
- E)** Patricia Uribe Guadalupe
- F)** Carla Uribe Guadalupe
- G)** Karla Patricia Uribe Diaz
- H)** Carla Patricia Hernandez

Conclusión

El desarrollo de este compendio de actividades representa no solo la integración de conocimientos adquiridos durante el semestre, sino también una muestra del proceso de formación en competencias fundamentales para el entendimiento de los distintos paradigmas de programación. A través de los ejercicios prácticos, reportes teóricos y la implementación de un programa en HTML que centraliza el acceso a los documentos, se consolidó una experiencia de aprendizaje que articula teoría, práctica y organización digital.

Cada actividad permitió explorar un enfoque diferente: desde los conceptos básicos de los lenguajes de programación hasta la práctica directa con lenguajes imperativos, funcionales, lógicos, orientados a objetos y de scripting. Esta diversidad fortaleció la capacidad de análisis y adaptación frente a distintos entornos y estilos de programación, resaltando no solo sus diferencias técnicas, sino también sus ventajas y aplicaciones específicas en la solución de problemas reales.

Por otro lado, el desarrollo del programa HTML refuerza la importancia de la estructuración y presentación de información en entornos digitales, permitiendo al lector navegar de forma clara y eficiente entre los distintos trabajos. Esta herramienta también evidencia el crecimiento en habilidades digitales y de diseño lógico, así como el compromiso por entregar un producto final funcional y estéticamente accesible.

En conclusión, este proyecto integrador refleja el esfuerzo, la evolución académica y el compromiso personal a lo largo del semestre. Más allá de los conocimientos adquiridos, queda una base sólida para enfrentar futuros desafíos en el campo de la programación, con una visión crítica, práctica y versátil sobre los distintos lenguajes y paradigmas que componen el vasto universo del desarrollo de software.