

AF4, Síntesis.



UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, miércoles 19 de febrero de 2025.

Reporte exposiciones equipo #2.

1) Sintaxis y semántica. (Definición, características, propósitos)

La sintaxis y la semántica son aspectos fundamentales en los lenguajes de programación, ya que determinan tanto la estructura como el significado de las instrucciones escritas. La sintaxis establece las reglas y estructuras gramaticales para la correcta escritura de un programa, asegurando que el código sea comprensible para el compilador o intérprete. Estas reglas incluyen el uso correcto de operadores, delimitadores, palabras clave y estructuras de control.

Por otro lado, la semántica se encarga de definir el significado de las expresiones y sentencias utilizadas dentro de un programa. Mientras que un código puede cumplir con la sintaxis del lenguaje, si su semántica es incorrecta, el programa no producirá los resultados esperados. Un error semántico ocurre cuando una instrucción es escrita correctamente, pero su significado dentro del contexto del programa es erróneo o no válido.

Ambos conceptos son esenciales para garantizar que los programas sean comprensibles y ejecutables sin errores. Una buena práctica en el desarrollo de software es combinar un código sintácticamente correcto con una lógica bien definida para evitar errores en la ejecución y facilitar la depuración del programa.

Lenguaje de representación (Definición, consideraciones y ejemplos).

Existen distintos tipos de lenguajes que cumplen funciones específicas en el ámbito de la informática. Algunos de ellos están diseñados para la representación de información en formatos estructurados, facilitando el almacenamiento, intercambio y visualización de datos en diversos sistemas.

Estos lenguajes utilizan símbolos y reglas específicas que permiten la portabilidad y la compatibilidad entre diferentes plataformas y aplicaciones.

Los lenguajes de representación permiten organizar datos en formatos que pueden ser leídos tanto por humanos como por máquinas. Entre los más utilizados se encuentran XML y JSON. XML (Extensible Markup Language) se emplea para estructurar datos de manera jerárquica mediante etiquetas, lo que facilita su interpretación y modificación. JSON (JavaScript Object Notation), por su parte, ofrece una alternativa más ligera y fácil de manejar, siendo ampliamente usado en el intercambio de datos entre servidores y aplicaciones web.

El uso de estos lenguajes es crucial en entornos donde la interoperabilidad entre sistemas es un requisito, como en bases de datos, servicios web y aplicaciones móviles. Su capacidad de representar información estructurada los hace indispensables en la comunicación de datos entre distintos dispositivos y plataformas.

Lenguaje de consulta. (Definición, características y ejemplo).

Los lenguajes de consulta están orientados a la manipulación y recuperación de datos almacenados en bases de datos. Su propósito principal es facilitar el acceso y la gestión de la información, permitiendo a los usuarios realizar operaciones como selección, filtrado, actualización y eliminación de registros de manera eficiente.

SQL (Structured Query Language) es el lenguaje de consulta más utilizado en bases de datos relacionales. Este lenguaje permite realizar consultas complejas mediante comandos como SELECT, INSERT, UPDATE y DELETE. Su sintaxis estructurada y su capacidad para manejar grandes volúmenes de información lo convierten en una herramienta esencial en el manejo de datos.

Otros lenguajes de consulta incluyen XQuery y XPath, utilizados en bases de datos que almacenan información en formato XML. Estos lenguajes permiten buscar y extraer datos de documentos XML mediante expresiones estructuradas que facilitan la navegación dentro de la jerarquía de elementos.

El conocimiento y uso adecuado de estos lenguajes es fundamental en el ámbito de la informática, ya que permiten optimizar el acceso a la información y garantizar la integridad de los datos almacenados en sistemas de bases de datos.

Lenguaje de alto nivel.

Dentro de la clasificación de los lenguajes de programación, se encuentran aquellos considerados de alto nivel. Estos lenguajes están diseñados para facilitar la escritura de código al proporcionar una sintaxis cercana al lenguaje humano, lo que permite un desarrollo más intuitivo y menos propenso a errores.

Los lenguajes de alto nivel ofrecen abstracción del hardware, permitiendo a los programadores centrarse en la lógica del programa sin preocuparse por los detalles específicos de la arquitectura del sistema. Esto los hace ideales para el desarrollo de aplicaciones en diversos entornos, desde software de escritorio hasta inteligencias artificiales.

Ejemplos de lenguajes de alto nivel incluyen Python, Java y C++. Python es ampliamente utilizado debido a su simplicidad y legibilidad, lo que lo convierte en una opción ideal para principiantes y proyectos de desarrollo rápido. Java, por su parte, es popular en aplicaciones empresariales y móviles gracias a su portabilidad y robustez. C++ combina características de alto y bajo nivel, permitiendo un control detallado sobre la memoria y el rendimiento del programa.

Estos lenguajes han sido fundamentales en el avance de la informática, facilitando el desarrollo de software eficiente y accesible para una amplia gama de aplicaciones.

Componentes de un lenguaje de programación.

Los lenguajes de programación están compuestos por diversos elementos interrelacionados que permiten la construcción de programas funcionales. Entre los principales componentes se encuentran los lexemas, que representan las unidades mínimas de significado en el código fuente. Estos incluyen palabras clave, identificadores, operadores y símbolos especiales. Así como también estructuras de control y las bibliotecas estándar.

Las estructuras de control, como los condicionales y los bucles, permiten modificar el flujo de ejecución del programa, haciendo posible la toma de decisiones y la repetición de instrucciones de manera controlada. Los operadores, por su parte, permiten realizar cálculos y comparaciones entre valores, facilitando la manipulación de datos. Las funciones permiten la reutilización del código al encapsular instrucciones en bloques independientes, lo que mejora la modularidad y mantenibilidad del programa.

Por otro lado, las bibliotecas estándar contienen funciones y clases predefinidas que facilitan la programación al proporcionar herramientas listas para su uso, sin necesidad de implementar soluciones desde cero. La relación entre estos componentes es crucial para garantizar que un programa sea funcional, eficiente y fácil de mantener.

Un diseño bien estructurado que aproveche adecuadamente estos elementos puede mejorar significativamente la legibilidad y rendimiento del código.

Definición y características de un compilador e intérprete.

El compilador y el intérprete son herramientas esenciales en la ejecución de programas escritos en distintos lenguajes de programación. Ambos tienen la función de traducir el código fuente, escrito en un lenguaje de alto nivel, a un formato que puede ser ejecutado por la computadora, aunque difieren en la forma en que realizan esta tarea.

Un compilador es un programa que traduce el código fuente completo a código máquina antes de su ejecución. Esto significa que el programa es procesado en su totalidad, generando un archivo ejecutable que puede ser ejecutado de manera independiente del código fuente. Algunas de sus características incluyen una mayor eficiencia en la ejecución, ya que el código ya está traducido, y la capacidad de detectar errores en la fase de compilación, lo que evita que el programa se ejecute con errores sintácticos.

En contraste, un intérprete traduce y ejecuta el código línea por línea, sin generar un archivo ejecutable. Esto permite una ejecución más flexible, ya que el código puede modificarse y ejecutarse de inmediato sin necesidad de recompilar. Sin embargo, esta característica puede hacer que la ejecución sea más lenta en comparación con un programa compilado, ya que la traducción ocurre en tiempo real.

Los compiladores se utilizan comúnmente en lenguajes como C y C++, mientras que los intérpretes son empleados en lenguajes como Python y JavaScript. La elección entre un compilador o un intérprete depende del propósito del programa y de los requerimientos de ejecución, ya que ambos tienen ventajas y desventajas según el contexto en el que se utilicen.

Estructura, operación y ejemplos de un Intérprete.

Un intérprete es un software que se encarga de traducir y ejecutar código fuente directamente, sin generar un archivo ejecutable intermedio. Su estructura consta principalmente de un analizador léxico, un analizador sintáctico, un evaluador semántico y un ejecutor.

El analizador léxico toma el código fuente y lo divide en tokens, que son las unidades mínimas de significado dentro del lenguaje de programación. Luego, el analizador sintáctico verifica que los tokens estén organizados correctamente de acuerdo con la gramática del lenguaje. Posteriormente, el evaluador semántico revisa que las expresiones tengan un significado válido, asegurando que el código sea coherente. Finalmente, el ejecutor interpreta las instrucciones y las ejecuta en el entorno de ejecución.

El proceso de operación de un intérprete ocurre en tiempo real, lo que permite la ejecución inmediata del código sin necesidad de compilación previa. Esta característica es útil en el desarrollo de software interactivo y en la depuración de programas, ya que los errores pueden corregirse de inmediato sin necesidad de recompilar todo el código. Sin embargo, debido a que cada instrucción es traducida y ejecutada en el momento, los programas interpretados suelen ser más lentos que los compilados.

Algunos ejemplos de intérpretes incluyen el intérprete de Python, que permite ejecutar scripts y probar código de manera interactiva en la consola; el motor V8 de JavaScript, utilizado en navegadores web como Google Chrome para ejecutar código JavaScript en páginas web; y Ruby, cuyo intérprete permite

desarrollar aplicaciones web con rapidez y flexibilidad.

El uso de intérpretes es común en entornos donde la rapidez de desarrollo y la flexibilidad son más importantes que la velocidad de ejecución, como en el desarrollo de scripts, la enseñanza de programación y el análisis de datos. La capacidad de ejecutar código de manera interactiva y modificarlo en tiempo real hace que los intérpretes sean herramientas valiosas en muchas áreas de la Informática.

AF5, Un reporte que describa la implementación de programas script.



UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, viernes 14 de marzo de 2025.

Síntesis exposiciones equipo #2.

Introducción de los lenguajes de Scripting.

Los lenguajes de scripting son un tipo de lenguaje de programación diseñado para automatizar tareas, controlar aplicaciones y gestionar procesos dentro de un sistema. Al contrario de los lenguajes compilados, los lenguajes de scripting son interpretados, lo que significa que su código se ejecuta línea por línea sin necesidad de una compilación previa. Esto los hace ideales para tareas como la administración de sistemas, desarrollo web y automatización de procesos repetitivos.

Características Principales.

1. Interpretados: No requieren compilación, lo que facilita la depuración y modificación rápida del código.
2. Portabilidad: Al no depender de un sistema operativo específico, pueden ejecutarse en diferentes plataformas con el mismo código fuente.
3. Fácil de aprender y usar: Su sintaxis suele ser más sencilla en comparación con otros lenguajes de programación tradicionales.
4. Integración con otros sistemas: Pueden interactuar con aplicaciones y bases de datos para automatizar tareas complejas.
5. Uso en diversas áreas: Se emplean en desarrollo web, administración de servidores, análisis de datos y creación de videojuegos, entre otros.

Tipos de lenguajes de Scripting.

Se pueden clasificar según su propósito y aplicación:

Lenguajes de Scripting para Desarrollo Web: Se utilizan para la creación de páginas dinámicas y la gestión de contenido en sitios web. Ejemplos:

JavaScript: Lenguaje principal para la programación del lado del cliente en navegadores web.

PHP: Ampliamente usado en la creación de sitios web dinámicos y aplicaciones en servidores.

Lenguajes de Scripting para automatización y administración de Sistemas:

Python: Popular por su facilidad de uso y versatilidad, empleado en administración de servidores, análisis de datos y aprendizaje automático.

BASH: Utilizado en sistemas operativos basados en UNIX/Linux para la automatización de tareas del sistema.

Powershell: Diseñado para la gestión y automatización en entornos Windows.

Lenguajes de Scripting para Videojuegos y Aplicaciones específicas:

C++: Utilizado en motores de juegos como Unity y Unreal para programar comportamientos y enemigos.

R: Empleado en análisis estadísticos y ciencia de datos.

Ventajas:

Desarrollo rápido: Permiten escribir y ejecutar código sin necesidad de un largo proceso de compilación.

Facilidad de integración: Pueden interactuar con otros lenguajes, aplicaciones sin problemas.

Gran comunidad y soporte: Al ser populares, cuentan con documentación abundante y foros de ayuda.

Desventajas.

Menor eficiencia. Debido a su interpretación línea por línea, suelen ser más lentos que los lenguajes compilados.

Dependencia de un entorno de ejecución: Requerirán un intérprete adecuado para su ejecución, lo que puede ser una limitación en algunos sistemas.

Seguridad. Al estar ampliamente distribuidos, pueden ser un objetivo frecuente de ataques informáticos si no se configuran adecuadamente.

Python.

Python es un lenguaje de programación de alto nivel y propósito general, conocido por su sintaxis clara y legible. Fue creado por Guido van Rossum y lanzado en 1991. Python es ampliamente utilizado para desarrollo web, análisis de datos, inteligencia artificial, ciencia de datos, y más. Su enfoque en la legibilidad del código permite a los desarrolladores escribir menor código para hacer más. Además, cuenta con una amplia biblioteca estándar que facilita el desarrollo de aplicaciones complejas.

Historia.

1980: Guido van Rossum comienza a trabajar en Python como un proyecto paralelo al ABC.

1991: Se lanza la primera versión de Python.

2000: Lanzamiento de Python 2.0, que introduce la recolección de basura y las listas por comprensión.

2008: Python 3.0 se lanza para corregir errores fundamentales en el diseño de Python 2.x, aunque mantiene compatibilidad con versiones anteriores.

Características.

Sintaxis simple: La sintaxis de Python es fácil de leer y escribir, lo que mejora la productividad del desarrollador.

Interpretado: Se ejecuta línea por línea, lo que facilita la depuración y permite la experimentación interactiva.

Multiplataforma: Funciona en diferentes sistemas operativos como Windows, macOS y Linux.

Comunidad activa: Python cuenta con una amplia documentación y numerosos paquetes de terceros disponibles, lo que facilita el aprendizaje y el desarrollo.

versatilidad: Ideal para diversos campos como desarrollo web (Django, Flask), ciencia de datos (pandas, NumPy), inteligencia artificial (TensorFlow, PyTorch), y automatización de tareas.

Usos comunes:

Desarrollo web: Frameworks como Django y Flask permiten crear aplicaciones web robustas y escalables.

Analisis de datos: Librerías como pandas, NumPy y SciPy facilitan el análisis.

Inteligencia artificial y machine learning: Herramientas como TensorFlow y PyTorch permiten construir y entrenar modelos de machine learning.

Automatización: Scripts de Python se utilizan para automatizar tareas repetitivas y procesos administrativos.

Entornos de Desarrollo:

IDEs Populares: Algunos de los Entornos de Desarrollo Integrado (IDEs) más utilizados para Python incluyen PyCharm, Visual Studio Code, Jupyter Notebook, y Spyder.

Virtual environments: La gestión de entornos virtuales es crucial en Python para mantener las dependencias del proyecto aisladas. Entornos como venv y virtualenv son ampliamente utilizados para este propósito.

B. Bibliotecas y Frameworks Populares.

Django: Un framework de alto nivel que promueve el desarrollo rápido y el diseño limpio y pragmático para aplicaciones web.

Flask: Un microframework ligero para aplicaciones web que permite una mayor flexibilidad en el diseño de las aplicaciones.

pandas: Una biblioteca poderosa para el análisis y la manipulación de datos.

NumPy: Fundamental para la computación científica en Python proporcionando soporte para matrices y operaciones matemáticas avanzadas.

Imperativo: Permite la programación imperativa donde se describen los pasos exactos que el programa debe seguir para alcanzar un objetivo.

Perl.

Perl es un lenguaje de programación dinámico y de propósito general, creado por Larry Wall en 1987. Es considerado por su capacidad para procesar textos y su facilidad para manejar expresiones regulares. Perl se utiliza comúnmente en administración de sistemas, desarrollo web y automatización de tareas. La flexibilidad de Perl permite a los desarrolladores elegir

diferentes enfoques para resolver un problema, lo que a veces puede llevar a un código menos legible, pero muy poderoso.

Historia.

1987: Larry Wall lanza la primera versión de Perl como una herramienta para el procesamiento de texto y la administración del sistema.

1994: Se lanza Perl 5, una revisión significativa que introduce un sistema de módulos y objetos.

2000s: Perl se convierte en un pilar de la administración de sistemas y el desarrollo web, especialmente en el ámbito de CGI (Common Gateway Interface).

Características.

- **Procesamiento de Texto**: Excelente manejo de cadenas y expresiones regulares, ideal para la manipulación de texto.
- **Multiplataforma**: Compatible con diversos sistemas operativos, lo que lo hace versátil para distintos entornos.
- **CPAN**: Comprende el Perl Archive Network, una vasta colección de módulos y bibliotecas que extienden la funcionalidad de Perl.

Flexibilidad: Permite múltiples maneras de hacer las cosas (TMTOWTDI) - There's more than one way to do it -, lo que da a los desarrolladores libertad para abordar problemas desde diferentes ángulos.

Integración: Se integra bien con otros lenguajes y sistemas, permitiendo su uso en una amplia variedad de aplicaciones.

Usos Comunes.

Administración de sistemas: Scripts de Perl se utilizan para tareas de administración y mantenimiento de sistemas.

Desarrollo web: Aunque menos común hoy en día, Perl sigue siendo utilizado en aplicaciones web y CGI.

Automatización: Ideal para escribir scripts que automatizan procesos repetitivos y complejos.

Manipulación de datos: Procesamiento eficiente de archivos de texto y datos log.

Ecosistemas y CPAN.

CPAN: La Comprehensive Perl Archive Network (**CPAN**) es una colección masiva de módulos de Perl que los desarrolladores pueden utilizar para ampliar las funcionalidades de sus aplicaciones. Existen módulos para casi cualquier tarea imaginable, desde procesamiento de texto hasta integración con bases de datos y redes.

Uso de Módulos: Los módulos de Perl se pueden instalar fácilmente utilizando el comando cpan y se pueden importar en scripts como.

Programación de Red.

Sockets y Red: Perl ofrece soporte robusto para la programación de red. Módulos como FD::Socket facilitan la creación de

aplicaciones de red tanto cliente como servidor.

Automatización de Tareas de Red: Scripts de Perl se utilizan frecuentemente para tareas de automatización de red como monitoreo de servidores, análisis de tráfico de red y gestión de configuraciones.

Expresiones Regulares

Potencia en Texto: Perl es famoso por su capacidad de procesamiento de texto, y sus expresiones regulares son extremadamente poderosas y flexibles. Las expresiones regulares son extremadamente poderosas y flexibles. Las expresiones regulares en Perl son una herramienta esencial para cualquier tarea que implica la manipulación o el análisis de texto.

Shell.

El scripting en shell, también conocido como scripting de shell o bash scripting, se refiere a escribir scripts para el intérprete de línea de comandos de Unix / Linux. Bash es una de las shells más populares y su nombre proviene de "Bourne Again Shell", derivado del Bourne Shell original. Los scripts de shell se utilizan para automatizar tareas del sistema, manipulación de archivos, y administración de sistemas.

Historia.

1970's: El Bourne Shell (sh) se convierte en el estándar para la programación de scripts en Unix.

1989: Brian Fox escribe Bash, que se convierte en el shell por defecto en muchas distribuciones de Linux.

Hoy en día: Bash y otros shells como Zsh y Fish son ampliamente utilizados para la automatización y la administración del sistema.

Características

Automatización: Scripts de shell son ideales para automatizar tareas repetitivas y complejas en el sistema operativo.

Interacción directa con el sistema operativo: Acceso a comandos y utilidades del sistema, lo que permite una integración eficiente y directa.

Simplicidad: Permite escribir scripts sencillos y eficientes para tareas complejas, reduciendo el tiempo y el esfuerzo necesario.

Multiplataforma: Funciona en la mayoría de los entornos Unix/Linux y puede adaptarse a otros entornos incluyendo Windows con herramientas como Cygwin o WSL (Windows Subsystem for Linux).

Usos comunes

Automatización de tareas del sistema: Creación de scripts para ejecutar tareas repetitivas como copias de seguridad, actualizaciones de software y mantenimiento del sistema.

Manipulación de archivos: Uso de comandos de shell para copiar, mover, renombrar, y eliminar archivos de manera eficiente.

Administración de sistemas: Monitorización de recursos del sistema, gestión de usuarios y permisos, y configuración de entorno.

Interacción con otras herramientas: Los scripts de shell pueden llamar y controlar otras aplicaciones y comandos, permitiendo una integración profunda en el ecosistema de Unix/Linux.

Comandos Básicos

Gestión de archivos: Comandos como ls, cp, mv, rm y mkdir son fundamentales para la gestión de archivos y directorios.

Permisos de archivo: Comandos como chmod, chown y chgrp son esenciales para manejar los permisos de archivos y la propiedad.

Scripting Avanzado

Condicionales y Bucles: Los scripts de shell pueden incluir condicionales (if, else, elif) y bucles (for, while, until) para controlar el flujo del programa.

Funciones: Los scripts de shell pueden definir funciones para modularizar y reutilizar el código.

Administración del Sistema

Monitoreo del sistema: Comandos como top, htop, ps, df y du se utilizan para monitorear los recursos del sistema y el uso de disco.

Definición, características, propósito de los Diagramas de flujo.

Los Diagramas de flujo son representaciones gráficas que ilustran la secuencia de pasos de un proceso o sistema. Utilizan símbolos normalizados para mostrar las operaciones, decisiones y conexiones entre ellas. Estos usan símbolos estándar para ilustrar operaciones, decisiones y el flujo de control dentro de un programa. Los Diagramas de flujo son fundamentales en la fase de diseño del software, ya que ayudan a los desarrolladores a visualizar y planificar la estructura lógica del código antes de escribirlo.

Características de los Diagramas de Flujo.

1. Claridad Usual:

Representan de manera usual y clara el flujo de control en un programa, lo que facilita la comprensión de la lógica del código.

2. Estandarización:

Empiezan símbolos estandar como óvalos, rectángulos, rombos y flechas, que son ampliamente reconocidos y entendidos en la comunidad de programación.

3. Organizabilidad:

Organizan los pasos de manera lógica y secuencial, lo que es crucial para la correcta implementación del algoritmo.

4. Modularidad:

Permiten descomponer un programa en módulos o sub-procesos, facilitando su diseño y mantenimiento.

5. Facilidad de comunicación:

Son como una herramienta eficaz de comunicación entre programadores, analistas de sistemas y otros interesados.

6. Identificación de errores:

Facilitan la detección de errores lógicos y pasos redundantes en el algoritmo antes de la implementación.

Historia y su origen:

Se popularizaron en los años 1940 y 1950, siendo utilizados para diseñar algoritmos y programas antes de la implementación en lenguaje de máquinas.

Grace Hopper, una pionera en programación, fue una de las primeras en utilizar diagramas de flujo para documentar la lógica de programas en la Marina de los Estados Unidos.

Ventajas:

Diseño Preliminar: Ayudan a los programadores a diseñar la estructura lógica de sus programas antes de codificarlo, lo que ahorra tiempo y reduce errores.

~~Si se le ha~~

Documentación: Los diagramas de flujo proporcionan una documentación visual que es útil para entender el código y para la formación de nuevos miembros del equipo.

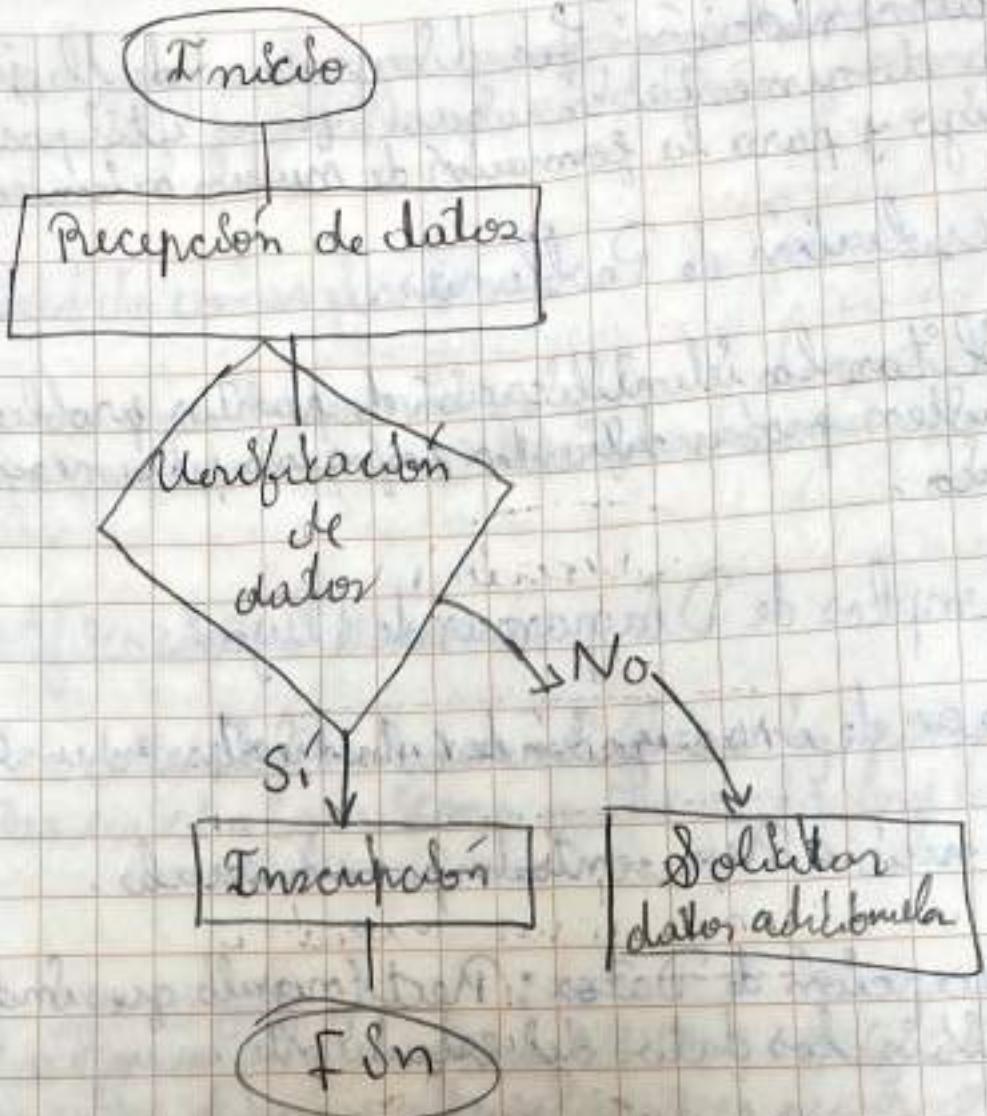
Resolución de Problemas.

Facilitan la identificación de puntos problemáticos y permiten probar diferentes enfoques para resolver problemas lógico.

Ejemplos de Diagramas de Flujo.

Proceso de Inscripción de Estudiantes en un Sistema.

1. **T nube:** Representado por un óvalo.
2. **Recepción de Datos:** Rectángulo que indica la acción de recibir los datos del estudiante.
3. **Verificación de Datos:** Rombo que representa la decisión de si los datos son correctos o no.
 - Sí: Flecha que lleva a la acción de Inscripción.
 - No: Flecha que lleva a la acción de solicitar datos nuevamente.
4. **Inscripción del estudiante:** Rectángulo que indica la acción de inscribir al estudiante.



Algoritmo para calcular el Factorial de un número.

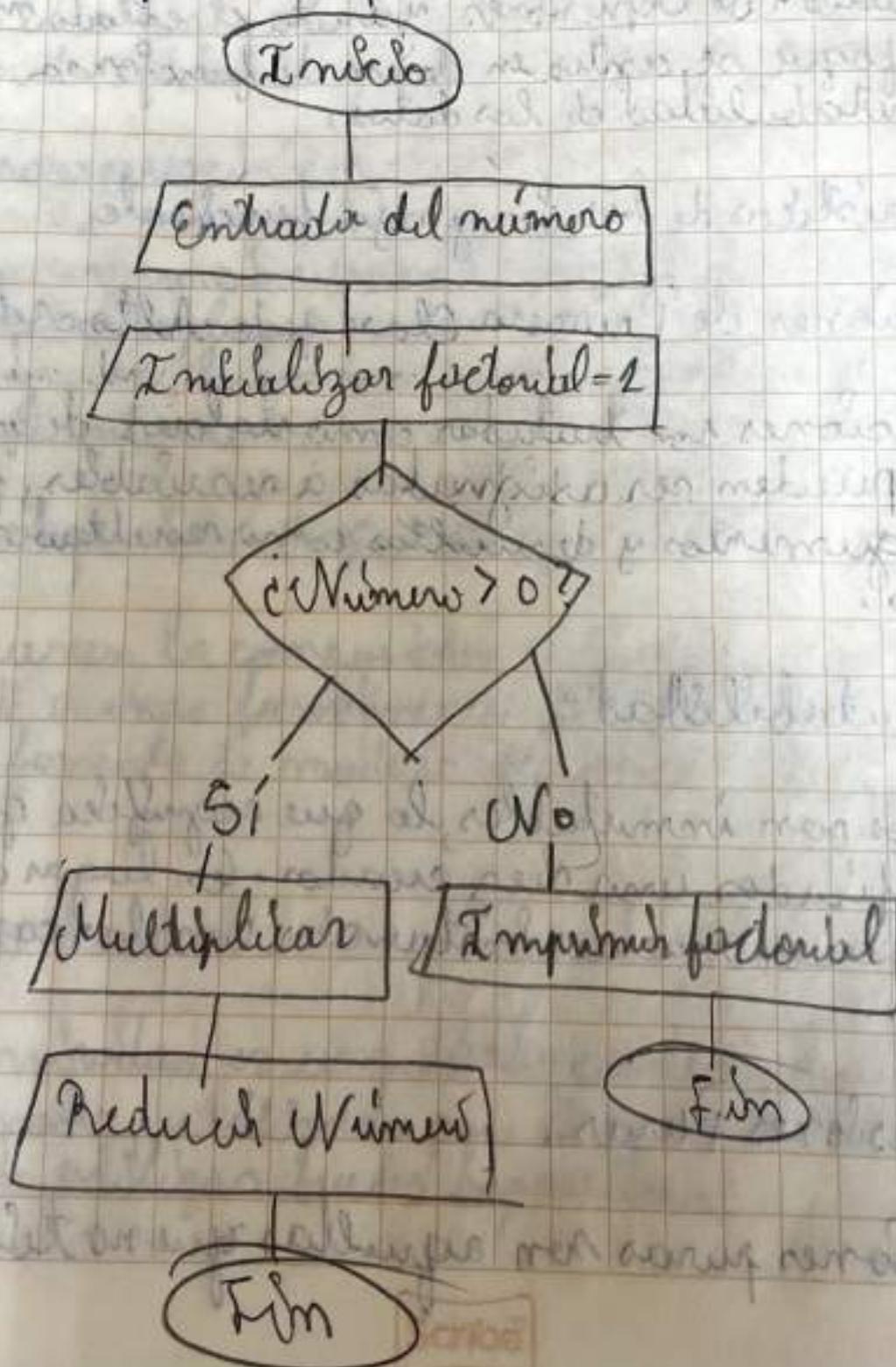
1. Inicio: Representado por un óvalo.
2. Entrada del Número: Rectángulo que indica la acción de introducir el número.
3. Inicializar Factorial = 1: Rectángulo que indica la acción de inicializar una variable factorial a 1.
4. ¿Número > 0?: Rombo que representa la decisión de si el número es mayor que 0.

• Sí: Flecha que lleva a multiplicar factorial por el número y reducir el Número en 1.

No: Flecha que lleva al final del proceso.

5. Imprimir Factorial: Rectángulo que indica la acción de imprimir el resultado factorial.

6. Fim: Representado por un óvalo.



Definición, características, propósito y ejemplos de los lenguajes funcionales.

Los lenguajes funcionales son un paradigma de programación basado en funciones matemáticas y cálculo lambda. A diferencia de la programación imperativa, que se centra en cambiar el estado del programa mediante instrucciones secuenciales, la programación funcional trata la computación como la evaluación de expresiones y evita el estado mutable. Este enfoque se centra en el uso de funciones puros y la inmutabilidad de los datos.

Características de los lenguajes funcionales.

1. Funciones de Primera Clase y de Alto orden.

Las funciones son tratadas como valores de primera clase y pueden ser asignadas a variables, pasadas como argumentos y devueltas como resultados de otras funciones.

2. Inmutabilidad:

Los datos son inmutables, lo que significa que no pueden ser modificados una vez creados. En lugar de modificar datos, se crean nuevas instancias con los cambios necesarios.

3. Expresiones Puras.

Las funciones puros son aquellas que no tienen

efectos secundarios y siempre producen el mismo resultado dado el mismo conjunto de argumentos. Esto facilita el razonamiento sobre el código y la previsibilidad del comportamiento del programa.

4. Evaluación Perezosa.

La evaluación perezosa permite que las expresiones no se evalúen hasta que su valor sea necesario, lo que puede mejorar la eficiencia y evitar cálculos innecesarios.

5. Transparencia Referencial.

La transparencia referencial implica que una expresión puede ser reemplazada por su valor sin cambiar el comportamiento del programa, lo que resulta en un código más predecible y fácil de entender.

6. Composición de Funciones.

Promueven la composición de funciones, permitiendo construir nuevas funciones a partir de otras más simples lo que fomenta la modularidad y la reutilización del código.

7. Recursividad

La recursividad es una técnica común en los lenguajes funcionales, utilizada para iterar a través de datos en lugar de utilizar bucles imperativos.

Propósito de los Lenguajes Funcionales.

El propósito principal de los lenguajes funcionales es proporcionar una forma más declarativa y matemática de escribir programas. Son ideales para tareas que involucran cálculos complejos, transformaciones de datos y programación concurrente. Algunos propósitos específicos incluyen:

Simplicidad y Claridad: Facilitan el razonamiento y la prueba de la corrección del código debido a su enfoque en funciones puras y la ausencia de efectos secundarios.

Concisión: Permiten escribir menos código para lograr el mismo resultado, lo que reduce la posibilidad de errores y hace que el código sea más fácil de mantener.

Eficacia y Parallelismo: Son altamente adecuados para la programación concurrente y paralela, ya que la immutabilidad de los datos y la ausencia de efectos secundarios facilitan la ejecución segura de múltiples hilos.

Reutilización de Código: Promueven la reutilización de funciones y la composición de funciones, lo que resulta en un código más modular y mantenible.

Facilidad de Pruebas: Las funciones pías y la tautología referencial facilitan la escritura de pruebas unitarias y la verificación de la corrección del programa.

Otros aspectos relevantes de los lenguajes.

Optimización mediante Transformaciones Funcionales:

Los compiladores de lenguajes funcionales pueden aplicar transformaciones algebraicas para optimizar el código, como la eliminación de recursión de cola y la fusión de mapas y filtrados.

Notación matemática y simbólica.

Los lenguajes funcionales a menudo permiten una notación matemática y simbólica más cercana a las matemáticas formales, lo que facilita la implementación de algoritmos matemáticos complejos.

Evolución.

Tiene sus raíces en el trabajo de Alonzo Church y su cálculo lambda, desarrollado en la década de 1930. Lisp, uno de los primeros lenguajes funcionales, fue creado en 1958 por John McCarthy.

Compatibilidad con otros paradigmas.

Algunos lenguajes funcionales modernos, como Scala y F#, permiten combinar características funcionales con programación orientada a objetos e imperativa, ofreciendo flexibilidad para diferentes estilos de programación.

Aplicaciones en la industria.

Los lenguajes funcionales son utilizados en una amplia gama de aplicaciones industriales, desde sistemas finanzas

AF6, Reporte que describa la ejecución de un programa funcional.



UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, viernes 21 de marzo de 2025.

Scheme.

Es un lenguaje de programación funcional y uno de los dialectos más conocidos del lenguaje Lisp. Fue desarrollado en los años 70 por Guy L. Steele y Gerald Jay Sussman en el MIT. Scheme es conocido por su simplicidad y elegancia, así como su flexibilidad y poder. Es un lenguaje de propósito general que permite la programación funcional, la programación orientada a objetos y la programación imperativa.

Los primeros trabajos se centraron en la exploración de la semántica de programación y la implementación de intérpretes ejecutables. La primera versión del lenguaje, considerada como MIT Scheme, se publicó en 1975.

Scheme ha evolucionado y ha sido influenciado por las investigaciones y desarrollos en teorías de lenguajes de programación. La versión estandarizada más reciente de Scheme es R7RS, publicada en 2013.

Características.

Simplicidad y elegancia. Scheme tiene una sintaxis sencilla y uniforme. Utiliza una notación de prefijo y paréntesis para todos los expresiones, lo que simplifica la sintaxis y facilita la manipulación del código como datos.

Closures. Significa que las funciones pueden capturar y reírse su entorno de ejecución de funciones anidadas y de orden superior.

Macro Sistema. Potente sistema de macros que permite a los programadores extender el lenguaje y crear nuevas construcciones de control.

Portabilidad. Altamente portable y hay múltiples implementaciones disponibles en diferentes plataformas, lo que lo hace adecuado para una amplia gama de aplicaciones.

Ejemplos de código en Scheme.

Ejemplo 1: Función de Factorial.

```
(define (factorial n)
  (if (= n 1)
      1
      (* n (factorial (- n 1)))))
```

Este ejemplo muestra una implementación recursiva de la función factorial en Scheme. La función toma un entero y devuelve el producto de todos los enteros posteriores menores iguales a ese número.

Ejemplo 2: Uso de cierres (closures).

```
(define (make-counter)
  (let ((count 0))
    (lambda () (set! count (+ count 1)))))
```

Este ejemplo muestra cómo crear una función que mantiene un estado interno utilizando cierres. La función make-counter devuelve una función que incrementa y devuelve un conteo cada vez que es llamada.

hasta telecomunicaciones y procesamiento de datos en grandes volúmenes.

Ejemplos de lenguajes funcionales.

1. Haskell.

Un lenguaje funcional poco conocido por su fuerte tipificación estática y su capacidad de evaluación perezosa. Es ampliamente utilizado en la academia y la investigación.

Ejemplo: Implementación de la función de Fibonacci

Fibonacci :: Int → Int

$$\text{Fibonacci } 0 = 0$$

$$\text{Fibonacci } 1 = 1$$

$$\text{Fibonacci } n = \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2)$$

2. Lisp:

Uno de los lenguajes de programación más antiguos, conocido por su flexibilidad y poder de metaprogramación. Lisp ha influido en muchas otras lenguajes de programación.

Ejemplo: Definición de una función simple

```
( defun factorial (n)
```

```
  ( if (<= n 1)
```

```
    1
```

```
    (* n (factorial (- n 1)) ) ) )
```

3. Erlang.

Un lenguaje funcional diseñado para sistemas concurrentes y de alta disponibilidad. Es popular en el desarrollo de sistemas de telecomunicaciones y aplicaciones de mensajería.

Ejemplo: Función para enviar un mensaje en un sistema concurrente.

```
send_message(Recipient, Message) →  
    Recipient ! {self(), Message}.
```

4. F#. (Fsharp).

Un lenguaje funcional que forma parte del ecosistema .NET. Combina características funcionales con capacidades imperativas y orientadas a objetos.

Ejemplo: Filtrado de una lista.

```
let evenNumbers = list.filter(fun x → x % 2 = 0) [1; 2; 3;  
4; 5; 6]
```

5. Scala.

Un lenguaje que integra paradigmas funcionales y orientados a objetos. Es ampliamente utilizado en el desarrollo de aplicaciones de alto rendimiento y sistemas distribuidos.

Ejemplo: Uso de funciones anónimas.

```
val numbers = list(1, 2, 3, 4, 5, 6)  
val evenNumbers = numbers.filter(_ % 2 == 0)
```

Diagramas de Flujo de Recursión, Pilas y Listas.

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

Ventajas.

- ✓ Hace que el código sea más elegante y fácil de entender.
- ✓ Útil para problemas que tiene una estructura recursiva natural (árboles, grafos).

Desventajas.

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

Aplicaciones.

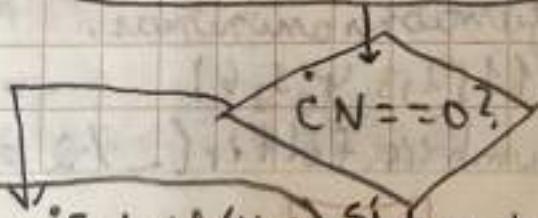
- Cálculo de factorial, fibonacci, potencias.
- Recorridos en estructuras de datos como árboles y grafos.
- Resolución de problemas como Torres de Hanói y búsquedas binaria

Ejemplo de diagrama de flujo

• Si n es 0, se retorna 1 (caso base).

• Si n es mayor

Resultado = factorial (N)



que 0, se multiplica n por factorial($n-1$) que es otra llamada a la misma función.

Resultado = $N \cdot \text{factorial}(N-1)$

Si
Resultado = 1
Fin

- La función se sigue llamando a sí misma hasta llegar a 0!, que es 1.

- Luego, las llamadas regresan, multiplicando los valores acumulados.

Pila (Stack)

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

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.

Ejemplo en Java Script.

```
import java.util.Stack; // Importación necesaria para usar Stack
```

```
public class StackExample {
```

```
    public static void main (String [ ] args) {
```

```
        Stack < Integer > stack = new Stack < > ( );
```

```
        // Agregar elementos a la pila.
```

```
        stack.push (10);
```

```
        stack.push (20);
```

```
        stack.push (30);
```

// Mostrar la pila antes de hacer pop

System.out.println("Pila actual: " + stack);

// Eliminar el elemento superior de la pila

System.out.println("Elemento eliminado: " + stack.pop());

// Mostrar la pila después de hacer pop

System.out.println("Pila después de pop: " + stack);

}

}

Pila actual: [10, 20, 30]

Elemento eliminado: 30

Pila después de pop: [10, 20]

Listas.

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

Tipos de Listas:

Lista Enlazada Simple: cada nodo tiene un puntero al siguiente nodo.

Lista Doblemente Enlazada: cada nodo tiene punteros al siguiente y al anterior.

Lista Circular: el último nodo apunta al primer nodo.

Características.

- ✓ Pueden 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.

Ejemplo en Python.

class Nodo:

```
def __init__(self, dato):  
    self.dato = dato  
    self.siguiente = None
```

class ListaEnlazada:

```
def __init__(self):  
    self.cabeza = None
```

```
def Insertar_ultimo(self, dato):  
    nuevo_nodo = Nodo(dato)  
    nuevo_nodo.siguiente = self.cabeza  
    self.cabeza = nuevo_nodo
```

```
def mostrar_lista(self):
```

```
    actual = self.cabeza  
    while actual:
```

```
        print(actual.dato, end=" → ")
```

```
        actual = actual.siguiente
```

```
    print("None")
```

Uso de la lista enlazada

lista = `Lista Enlazada()`

lista. Insertar_Enlazado(1)

lista. Insertar_Enlazado(2)

lista. Insertar_Enlazado(3)

lista. mostrar = lista(1) # Salida: 3 → 2 → 1 → None

Terminal

PS C:\Users\angel\OneDrive - Universidad Autónoma
de Nuevo León.

3 → 2 → 1 → None.

Haskell

Haskell es un lenguaje de programación funcional puro y de propósito general, conocido por su fuerte sistema de tipos estáticos y su capacidad para realizar evaluaciones perezosas. Fue nombrado en honor al matemático y lógico Haskell Curry. Haskell es ampliamente utilizado en la academia y la investigación, y también ha encontrado aplicaciones en la industria, especialmente en el desarrollo de software crítico y de alta fiabilidad.

Historia y evolución.

Haskell fue desarrollado a finales de los años 80 y principios de los 90 como un esfuerzo colaborativo entre académicos e investigadores interesados en la programación funcional. La primera versión del lenguaje, Haskell 1.0, se publicó en 1990. Desde entonces, Haskell ha evolucionado y se ha mejorado continuamente, con la versión más reciente, Haskell 2010, publicada en 2010.

Características de Haskell.

Funcional Puro.

Haskell es un lenguaje funcional puro, lo que significa que todas las funciones son funciones pures sin efectos secundarios. Esto facilita el razonamiento sobre el código y garantiza la predictibilidad del comportamiento del programa.

2. Evaluación Perezosa:

La evaluación perezosa, también conocida como evaluación diferida, permite que las expresiones no se evalúen hasta que sea necesario. Esto puede mejorar la eficiencia y permitir la creación de estructuras de datos infinitas.

3. Sistema de Tipos Estático y Fuerte:

Haskell utiliza un sistema de tipos estático y fuerte que permite detectar errores de tipo en tiempo de compilación. Además, emplea inferencia de tipos, lo que significa que los tipos pueden deducirse automáticamente sin necesidad de ser explícitamente especificados.

4. Polimorfismo:

Haskell admite polimorfismo paramétrico, lo que permite escribir funciones y tipos generales que pueden trabajar con múltiples tipos de datos.

5. Composición de funciones:

La composición de funciones es una característica fundamental de Haskell, permitiendo construir nuevas funciones a partir de otras más simples.

Propósito de Haskell:

Proporcionar un lenguaje de programación funcional puro y expresivo que permite la creación de software seguro, eficiente y mantenible. Algunos de los objetivos y ventajas de Haskell son:

- ✓ **Seguridad y correctitud:** Haskell facilita la escritura de programas correctos y seguros gracias a su sistema de tipos fuerte y a la inmutabilidad de los datos.
- ✓ **Claridad y claridad:** Permite crear y expresar algoritmos complejos de manera concisa y clara, reduciendo la cantidad de código necesario y facilitando su comprensión.
- ✓ **Aplicaciones concurrentes y paralelas:** Es adecuado para la programación concurrente y paralela, gracias a su enfoque en la inmutabilidad y a la facilidad para manejar la concurrencia mediante monedas y otros constructos.

Ejemplos de código:

Ejemplo 1: Función de Factorial, Ejemplo 2: Filtrado de lista

factorial :: Int \rightarrow Int

Factorial 0 = 1

Factorial n = n * factorial(n-1)

filter even :: [Int] \rightarrow [Int]

filter even xs = filter(x \rightarrow x mod 2 == 0) xs

Ejemplo 3: Uso de Monadas para manejar E/S.

main :: IO()

main = do

putStrLn "¿Cuál es tu nombre?"

name <- getLine

putStrLn ("Hola," ++ name ++ " !")

AF7, Reporte lenguaje lógico Prolog.



UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, miercoles 09 de abril de 2025.

Reporte de Programa lógico en Prolog 09-04-25

Clima	Día	Actividad Recomendada
soleado	fin - semana	ir - a - la - playa
soleado	entre - semana	caminar - en - el parque
lluvioso	cualquier	quedarse - en - casa
nublado	fin - semana	ver - película

•/. Reglas

clima (soleado).

día (fin - semana).

•/. Reglas

actividad (ir - a - la - playa) :- clima (soleado), dia(fin_semana).
actividad (caminar - en - el - parque) :- clima(soleado), dia.
actividad (quedarse - en - casa) :- clima (lluvioso).
actividad (ver - película) :- clima (nublado), dia (fin - semana).

AF8, Lenguajes imperativos.



UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME
FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, viernes 23 de mayo de 2025.

Introducción.

Los lenguajes de programación imperativos son aquellos en los que se le indica a la computadora paso a paso lo que debe hacer para resolver un problema. Se basan en el uso de instrucciones secuenciales, donde el programa cambia su estado a medida que avanza la ejecución.

En este tipo de lenguajes, el programador controla directamente el flujo de ejecución, usando estructuras como:

- Asignaciones (`x = 5`)
- Condicionales (`if, else`)
- Bucles (`for, while`)
- Subrutinas o funciones

Algunos lenguajes clásicos imperativos incluyen Fortran, Pascal, C y Python, cada uno con su propio estilo y sintaxis, pero todos siguiendo el mismo enfoque: dar órdenes claras al computador para que las ejecute en un orden específico.

Este tipo de programación es muy útil para resolver problemas de forma lógica y estructurada, y es la base de muchos otros paradigmas modernos. En este reporte se presentan programas sencillos en diferentes lenguajes imperativos para mostrar cómo se puede aplicar este estilo de programación de manera práctica.

Programa #1.

Calcular área de triángulo. – FORTRAN.

Este programa pide al usuario que ingrese la base y la altura de un triángulo. Luego, calcula su área.



```

1 // welcome to QDB Online,
2 // QDB provides an online compilation and debugger tool for C, C++, Python, Java, PHP, Ruby, Perl,
3 // C#, OCaml, ML, Swift, Pascal, Fortran, Haskell, Object Pascal, Assembly, HTML, CSS, JS, SQL, COBOL, Prolog,
4 // Scala, Elm, Clojure, Racket and many more from anywhere in world.
5
6
7
8 Program calcular_area;
9 implicit none;
10 real :: altura;
11 real :: base;
12 real :: area;
13 write (*,*) "Programa que calcula el area de un triangulo";
14 read (*,*) "Ingres la altura de su triangulo";
15 read (*,*) altura;
16 write (*,*) "Ingres la base de su triangulo";
17 read (*,*) base;
18 area = (base*altura)/2;
19 write (*,*) "El area de su triangulo es igual a", area, "unidades cuadradas";
20 write (*,*) "Presione enter para finalizar el programa";
21 read (*,*)
22 End Program calcular_area;

```

Input

```

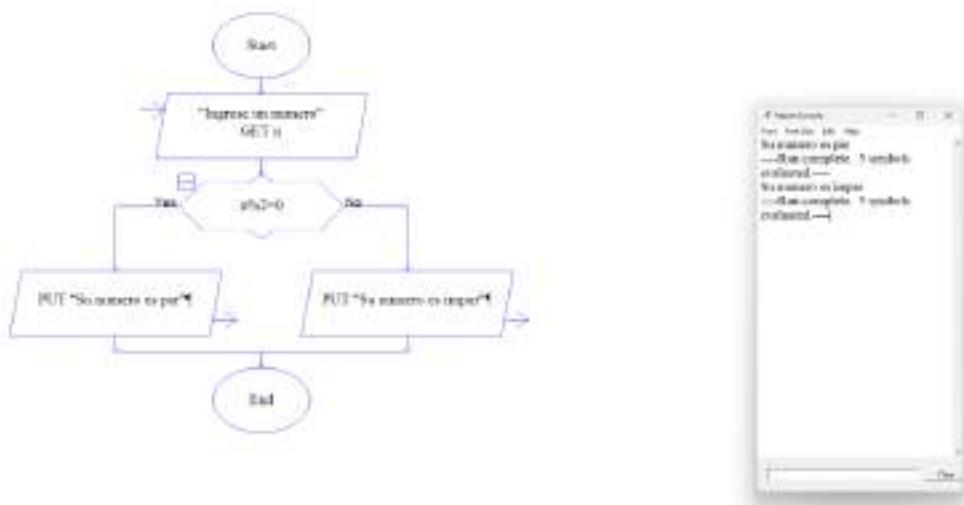
Programa que calcula el area de un triangulo.
Ingres la altura de su triangulo
25
Ingres la base de su triangulo
5
El area de su triangulo es igual a 125.000000 unidades cuadradas
Presione enter para finalizar el programa

```

Programa #2.

Verificar si n es par o impar. – PASCAL

Este programa tiene como funcionamiento leer un numero y mediante estructuras condicionales verifica si es par o impar.



```

main()
1 // Welcome to IDE Online.
2 // IDE Online is an online compiler and debugger tool for C, C++, Python, Java, PHP, Ruby, Perl,
3 // C99, OCaml, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS, SQL (MySQL, PostgreSQL),
4 // Oracle Database, Redis and many more languages ...
5 // Code, Compile, Run and Debug online from anywhere in world.
6
7
8 Program numero ParImpar;
9 var
10   n: Integer;
11 begin
12   Writeln("Ingresa un numero: ");
13   Readln(n);
14
15   if n mod 2 = 0 then
16     Writeln ("El numero es par");
17   else
18     Writeln ("El numero es impar");
19
20 end.
21
22

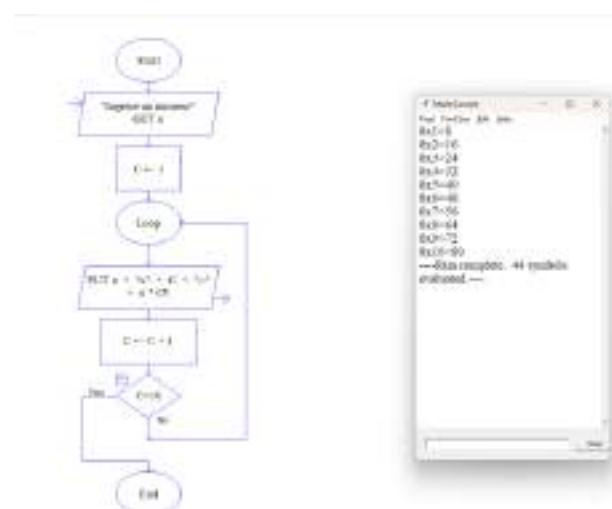
```

Free Pascal Compiler version 3.2.2+dfsg-32 12024/01/05 for x86_64
Copyright (c) 1993-2021 by Florian Klauer and others
Targets: GDB Linux for x86_64
Compiling main.pas
Linking output:
12 lines compiled, 0.0 sec.
Ingresa un numero: 7
El numero es impar.

Programa #3.

Tabla de multiplicación. – Python.

Lo que realiza este programa es, pedirle al usuario que ingrese un número y mediante un bucle while, hace que se muestre en la pantalla su tabla de multiplicación hasta el número 10.



```
1 //4
2
3 // Welcome to GDB Online!
4 // GDB online is an online compiler and debugger tool for C, C++, Python, Java, PHP, Ruby, Perl,
5 // OCaml, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS, JS, SQLite, Prolog,
6 // Code, Compile, Run and Debug online from anywhere in world.
7
8 /**
9  * @author: mari (Ingresar nombre de la tabla)
10 */
11 cont=1;
12
13 while(cont<10):
14     res=mult*cont;
15     print(mult, "<-", cont, "<->", res)
16     cont=cont+1;
```

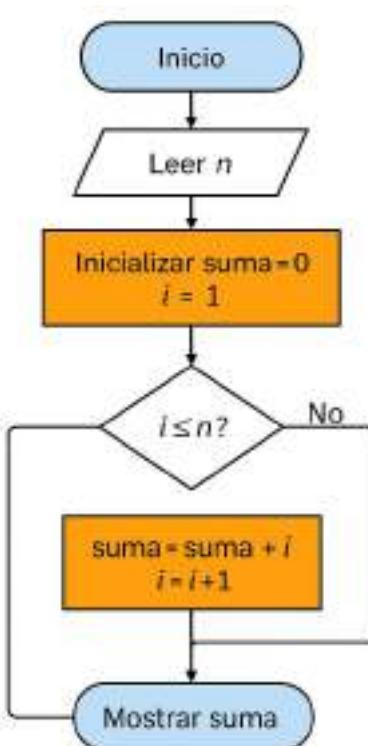


```
✓ ✎ 0 .s
Ingresar numero de la tabla? 7
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
7 x 10 = 70
```

Programa #4.

Suma de números hasta n. – C.

Este programa lee un número n y calcula la suma de todos los enteros desde 1 hasta n, usando un ciclo for.



```

1 //include <iostream.h>
2
3 int sumaC1() {
4     int n, suma = 0;
5
6     cout << "Introduce un número: ";
7     cin >> n;
8
9     for (int i = 1; i <= n; i++) {
10         suma += i;
11     }
12
13     cout << "La suma de 1 a " << n << " es: " << suma;
14     return 0;
15 }

```

A screenshot of a terminal window showing the execution of the C++ program. The user inputs the value 3, and the program outputs the result: "La suma de 1 a 3 es: 6".

AF9, Programación orientada a objetos.



UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, viernes 23 de mayo de 2025.

Introducción.

La Programación Orientada a Objetos (POO) es un paradigma de programación que organiza el software en torno a objetos, los cuales son representaciones de entidades del mundo real. Cada objeto combina datos (atributos) y comportamientos (métodos o funciones), facilitando una programación más estructurada, modular y reutilizable.

En este enfoque, los programas se construyen usando los siguientes principios fundamentales:

- **Clases y Objetos:** Una clase es un molde o plantilla que define las características y comportamientos de un objeto. Un objeto es una instancia concreta de una clase.
- **Encapsulación:** Permite ocultar los detalles internos del objeto, protegiendo los datos y exponiendo solo lo necesario a través de métodos públicos.
- **Herencia:** Es la capacidad de una clase de heredar propiedades y métodos de otra, facilitando la reutilización del código y la creación de jerarquías.
- **Polimorfismo:** Permite que una misma operación actúe de manera diferente dependiendo del tipo de objeto que la ejecute. Esto se logra mediante funciones virtuales y punteros a clases base.
- **Abstracción:** Consiste en modelar objetos complejos ocultando los detalles innecesarios y mostrando solo la información relevante para el uso del objeto.

Programa.

```
1 // Clase Persona
2 using namespace std;
3
4 class Persona {
5 protected:
6     string nombre;
7 public:
8     void setNombre(string n) {
9         nombre = n;
10    }
11     virtual void mostrar() {
12         cout << "Soy una persona y me llamo: " << nombre << endl;
13    }
14 };
15
16 class Estudiante : public Persona {
17 public:
18     void mostrar() override {
19         cout << "Soy un estudiante y me llamo: " << nombre << endl;
20    }
21 };
22
23 int main() {
24     Persona* p;
25     Estudiante e;
26     string nombreUsuario;
27
28     cout << "Ingresá tu nombre: ";
29     getline(cin, nombreUsuario);
30
31     e.setNombre(nombreUsuario);
32     p = &e;
33     p->mostrar();
34     return 0;
35 }
```

Input

```
Ingresá tu nombre: Matthew
Soy un estudiante y me llamo Matthew

...Program finished with exit code 0
Press ENTER to exit console.
```

Elementos aplicados en el código.

1. Encapsulación.

El atributo nombre está protegido y no se puede modificar directamente desde fuera de la clase.

2. Herencia.

La clase estudiante hereda de Persona y estudiante puede usar lo que tenga Persona.

3. Polimorfismo.

El puntero p es de tipo Persona*, pero apunta a un objeto Estudiante. Al llamar p -> mostrar () se ejecuta la versión del método mostrar () de la clase Estudiante gracias al uso de virtual en la clase base.

4. Tipos de datos.

String: Para guardar el nombre del usuario.

Persona*: puntero a un objeto de tipo Persona.

Estudiante: objeto de la clase derivada.

M5. Exámen Medio Curso Lenguaje de Programación. Exp. ØØs.

1. Conteste las siguientes preguntas.

100

1. ¿Qué es un lenguaje de programación?

Es un programa que está destinado para la creación de otros programas. Comprende un lenguaje formal ya que dentro de este lleva el razonamiento lógico y cuantitativo para la creación de estos mismos. Dentro de sus composiciones están los comentarios.

2. ¿Qué son lenguajes de alto nivel?

Es un tipo de lenguaje de programación donde la redacción y la lectura del código es fácil para nosotros los seres humanos ya que la mayoría de las palabras están escritas en el idioma inglés y no es traducido al lenguaje binario de los máquinas.

Ejemplos:
1- Java
2- Python
3- C++

3. ¿Qué es un lenguaje de consulta?

Es un tipo de lenguaje de programación con la funcionalidad de realizar consultas en bases de datos fuentes de datos. Estas mismas pueden ser manipuladas por el usuario, a diferencia de los lenguajes de programación, estas no se centran en el how o how to lo lógica. Son más orientadas a consultar. Ejemplo: SQL y Oracle.

4. ¿Qué son los lenguajes de representación?

Es un sistema dentro de los lenguajes de programación que describe las estructuras y normas que contiene la creación de éstos.

5. ¿Qué es la sintaxis y la semántica de un lenguaje?

La sintaxis son las reglas que indican como deben escribir el código para que la computadora lo entienda. La semántica es el significado de lo que escribimos en el código y nos funciona si y solo si cumplimos correctamente lo que queremos que haga la máquina.

6. Típo de lenguaje: lenguaje de representación.

7- ¿Qué es un lenguaje scripting?

Es un tipo de lenguajes de programación, caracterizado por su función de que no necesita ser compilado, sino, se necesita un intérprete. Estos se ejecutan línea por línea en tiempo real y podemos interrumpirlos o ejecutarlos más tarde. Son muy útiles para corregir errores que se presentan al lanzando. Son muy breves y se puede editar rápidamente.

8. Mencione ejemplos de lenguajes Scripting.

- 1- Java Script
- 2- Python
- 3- PHP
- 4- Powershell
- 5- Pearl
- 6- Shell

9- ¿Qué es un lenguaje funcional?

Es un tipo de lenguajes de programación donde se utilizan únicamente funciones matemáticas y los datos de estar son inmutables, no se pueden crear datos aparte, no pueden ser modificados una vez que se crean.

10 - Mencione y describa ejemplo de lenguaje funcional.

R: Lenguaje utilizado para estadística

Haskell: Lenguaje únicamente funcional y uno de los más rápidos.

Fsharp: Acrónimo de F# y se usa para la creación de softwares procedurales.

Python: Se puede utilizar como lenguaje funcional y funcional para análisis de datos.

PIA.



UANL
UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN



FIME

Lenguajes de Programación.

Ing. Karla Patricia Uribe Sierra.

Grupo 005.

M5.

Semestre enero – junio 2025.

Matthew Alejandro Martínez Zambrana.

2223170.

IAS.

San Nicolás de los Garza, N.L, miércoles 21 de mayo de 2025.

Funcionamiento del programa.

Este programa es una página web tipo portafolio académico, diseñada para presentar las actividades más importantes (llamadas "Actividades Fundamentales") realizadas por el estudiante Matthew Alejandro Martínez Zambrana durante el semestre.

¿Qué hace esta página?

1. **Muestra un título grande** con el nombre del portafolio y del estudiante.
2. **Presenta una lista de enlaces** (uno debajo del otro), donde cada enlace lleva a un archivo PDF diferente. Cada PDF representa una actividad fundamental del curso.
3. **Cuando haces clic en un enlace**, el archivo PDF correspondiente se abre en una nueva pestaña o ventana del navegador.
4. La página **tiene un diseño visual atractivo** gracias al uso de CSS, con colores suaves, una tipografía clara y efectos al pasar el cursor sobre los enlaces (hover).
5. La estructura se mantiene limpia y centrada para que sea fácil de leer y navegar.

¿Para qué sirve?

Esta página sirve como un **reporte digital o carpeta electrónica** donde un estudiante organiza y presenta su trabajo del semestre, facilitando que un maestro, compañero o evaluador acceda fácilmente a los documentos importantes del curso.

Código.

```
WixCode 0 index.html
```

```
0 1 <head> 2 <html> 3 <body> 4 <div> 5 <a href="#" class="contatree">
```

```
6 <!--HTML-->
```

```
7 <meta lang="es-ES" charset="UTF-8" />
```

```
8 <title>Portfolio del semestre</title>
```

```
9 <script>
```

```
10 </script>
```

```
11 <style>
```

```
12 body {
```

```
13   font-family: "Nexa UI", Tahoma, Geneva, Verdana, sans-serif;
```

```
14   background-color: #e6f4ff;
```

```
15   color: #003333;
```

```
16   margin: 0;
```

```
17   padding: 20px;
```

```
18 }
```

```
19 <h1>
```

```
20   text-align: center;
```

```
21   color: #00008B;
```

```
22   margin-bottom: 40px;
```

```
23 </h1>
```

```
24 <.container>
```

```
25   max-width: 200px;
```

```
26   margin: auto;
```

```
27   background: #ffffcc;
```

```
28   padding: 10px;
```

```
29   border-radius: 15px;
```

```
30   box-shadow: 0 0 10px #d9d9d9;
```

```
31 </div>
```

```
32 <a href="#" class="contatree">
```

```
33   display: inline;
```

```
34   padding: 10px 0;
```

```
35   margin: 10px 0;
```

```
36   background-color: #e6f4ff;
```

```
37   color: #00008B;
```

```
38   text-decoration: none;
```

```
39   border-radius: 10px;
```

```
40   transition: background-color 0.1s, transform 0.2s;
```

```
41 </a>
```

```
42 <a href="#" class="contatree" style="background-color: #e6f4ff; transform: scale(1.05);">
```

```
43 </a>
```

```
44 </div>
```

```
45 </body>
```

```
46 </html>
```

```
47 <!--HTML-->
```

```
48 <a href="#" class="contatree">
```

```
10 <h1>Portafolio del semestre - Matthew Alejandro Martínez Zambrana</h1>
11 <a href="EXAMEN MC.pdf" target="_blank">Actividad Fundamental 1 - Exámen Medio curso</a>
12 <a href="EXAMEN ORDINARIO.pdf" target="_blank">Actividad Fundamental 2 - Exámen Ordinario</a>
13 <a href="#" target="_blank">Actividad Fundamental 3 - Proyecto Integrador de aprendizaje</a>
14 <a href="AF4.pdf" target="_blank">Actividad Fundamental 4 - Streams</a>
15 <a href="AF5.pdf" target="_blank">Actividad Fundamental 5 - Implementación de programas</a>
16 <a href="AF6.pdf" target="_blank">Actividad Fundamental 6 - Programa Funcional</a>
17 <a href="AF7.pdf" target="_blank">Actividad Fundamental 7 - Lenguaje Lógico Prolog</a>
18 <a href="AF8_Lenguajes_imperativos_222379.pdf" target="_blank">Actividad Fundamental 8 - Lenguajes Imperativos</a>
19 <a href="AF9.pdf" target="_blank">Actividad Fundamental 9 - Programación orientada a objetos</a>
20 </div>
21 </body>
22 </html>
```

Resultado.





WELCOME TO

LENGUAJES DE PROGRAMACION

Equipo:

Gustavo Degaspari Dos Santos - 2113476

Javier Alexis Suarez Galarza - 2126816

Bernardo Gutiérrez López-2222904

Sherlyn Alexandra Gonzalez Castillo -2032114

Alexsandro Mercado Valdez-2095339

Matthew Alejandro Martínez Zambrana - 2223170

Fatima Del Angel Rivera-2103704

C++

HISTORIA



Bjarne Stroustrup.

- Desarrollado en el año 1979 en los laboratorios AT&T como una extensión orientada a objetos del lenguaje C.
(EL LO LLAMABA C CON CLASES)
- En 1983 es bautizado con el nombre de C++ por Rick Mascitti.
(Significaría incremento de C, aprovechando el operador que tiene el mismo lenguaje.)
- Marca el inicio de una nueva era en 1985 cuando es lanzada su versión comercial.
- Para 1990 se convirtió en uno de los lenguajes más importantes en el desarrollo de software de sistemas y aplicaciones de alto rendimiento.



Características principales.

```
boolean  
'PSI_INTERNAL_XML', false);  
ersion_compare("5.2", PHP_VERSION, ">")) {  
("PHP 5.2 or greater is required!!!");  
xtension_loaded("pcre")) {  
e("phpSysInfo requires the pcre extension to php  
properly.");  
re_once APP_ROOT . '/includes/autoload.inc.php';  
ad configuration  
re_once APP_ROOT . '/config.php';  
!defined('PSI_CONFIG_FILE') || !defined('PSI_D  
$tpl = new Template("/templates/html/error_co  
echo $tpl->fetch();  
ie();  
+ javascript
```



MULTIPARADIGMA.

Nos permite programar de forma imperativa, orientada a objetos, funcional y genérica. Lo que lo hace flexible y adaptable a distintos proyectos.

Uso de punteros.

Nos dan un control profundo de la memoria, lo que es clave en programación de sistemas.

BIBLIOTECA ESTÁNDAR.

Contiene estructuras de datos y algoritmos optimizados, como vectores, listas, pilas y colas, que agilizan el desarrollo.

ALTO RENDIMIENTO.

C++ es un lenguaje compilado, significa que tiene la capacidad de permitir que el código se traduzca a instrucciones de máquina de manera eficiente.

También ofrece un control preciso sobre la memoria y los recursos del sistema, ideal para aplicaciones que requieren velocidad.

COMPATIBILIDAD CON C.

Es compatible con código escrito en C, lo que facilita la migración y reutilización de software antiguo.

Aplicaciones en la vida real.

A white humanoid robot arm is shown from the side, holding a pen and writing mathematical equations on a blackboard. The equations involve sequences $\{x_n\}$ and $\{y_n\}$, limits, and trigonometric functions. A graph of a function is also drawn on the board.

$$\text{CO } \frac{\{x_n\}}{\{y_n\}} = \frac{\{x_n\}}{\{y_n\}}; \quad x + \frac{3n-4}{n^2-2n+3} \quad \{x_n\} \lim_{n \rightarrow \infty} \frac{n^2-x}{3}$$
$$\lim_{n \rightarrow \infty} \sqrt[n]{1 + e^{-\pi} + \pi n + 13} \quad \lim_{n \rightarrow \infty} \sqrt[n]{n^2 - 2n + 3}$$
$$\lim_{n \geq n_0} \sqrt[n]{4^n + \cos 2n} \quad \left(\frac{n^2+n-1}{n^2-2n+3} \right)^5 \quad x: \text{P}$$
$$\forall n \in N \quad x_n < y_n < z_n;$$
$$n \geq n_0: (x_n - g) < \varepsilon \quad \text{lokal. } \{x_n\}: x_n = \frac{1}{n}; \quad \{y_n\} =$$
$$\max; \quad \{x_n\} \sqrt[n]{0+0+0} \leq \sqrt[n]{13^n} \quad x_n \rightarrow R$$
$$\sqrt[n]{4} \cdot \sqrt[n]{13^n} \cdot \sqrt[n]{13^n} \quad R$$
$$\{x_n\} \cdot \{y_n\} = \{x_n + y_n\}; \quad 13$$
$$\{x_n\} \cdot \{y_n\} = \{x_n \cdot y_n\}; \quad 13$$
$$\sum_{n=1}^{\infty} x_n \quad \text{r.}, \quad g \quad \sqrt[n]{13^n}$$

Aplicaciones en la vida real.



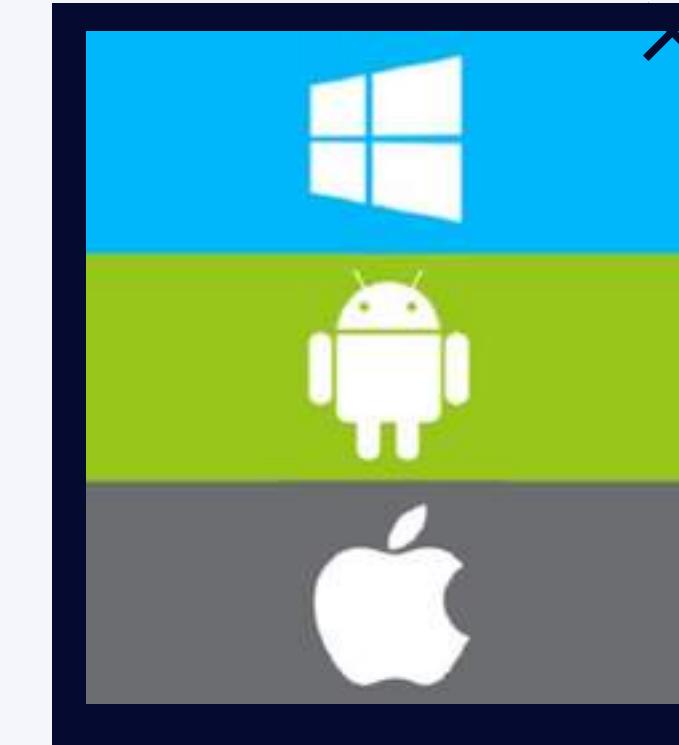
Videojuegos.



Software HP.



AI & ML .



OS.

Fundamentos clave para programar.



Operadores

X

Operador	Sintaxis	Significado
<	a < b	Menor que
<=	a <= b	Menor o igual que
>	a > b	Mayor que
>=	a >= b	Mayor o igual que
==	a == b	Igual a
!=	a != b	Distinto de
!	!a	Negación lógica
&&	a && b	AND lógico
	a b	OR lógico

Estructura básica de un programa.

```
1 #include<iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     cout<<"Hello world!" << endl;
8     return 0;
9 }
```

Tipos de datos y variables.

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int a;
8     float b = 4.7;
9     char c = 'k';
10    bool d = true;
11
12 }
```

Declaración condicional

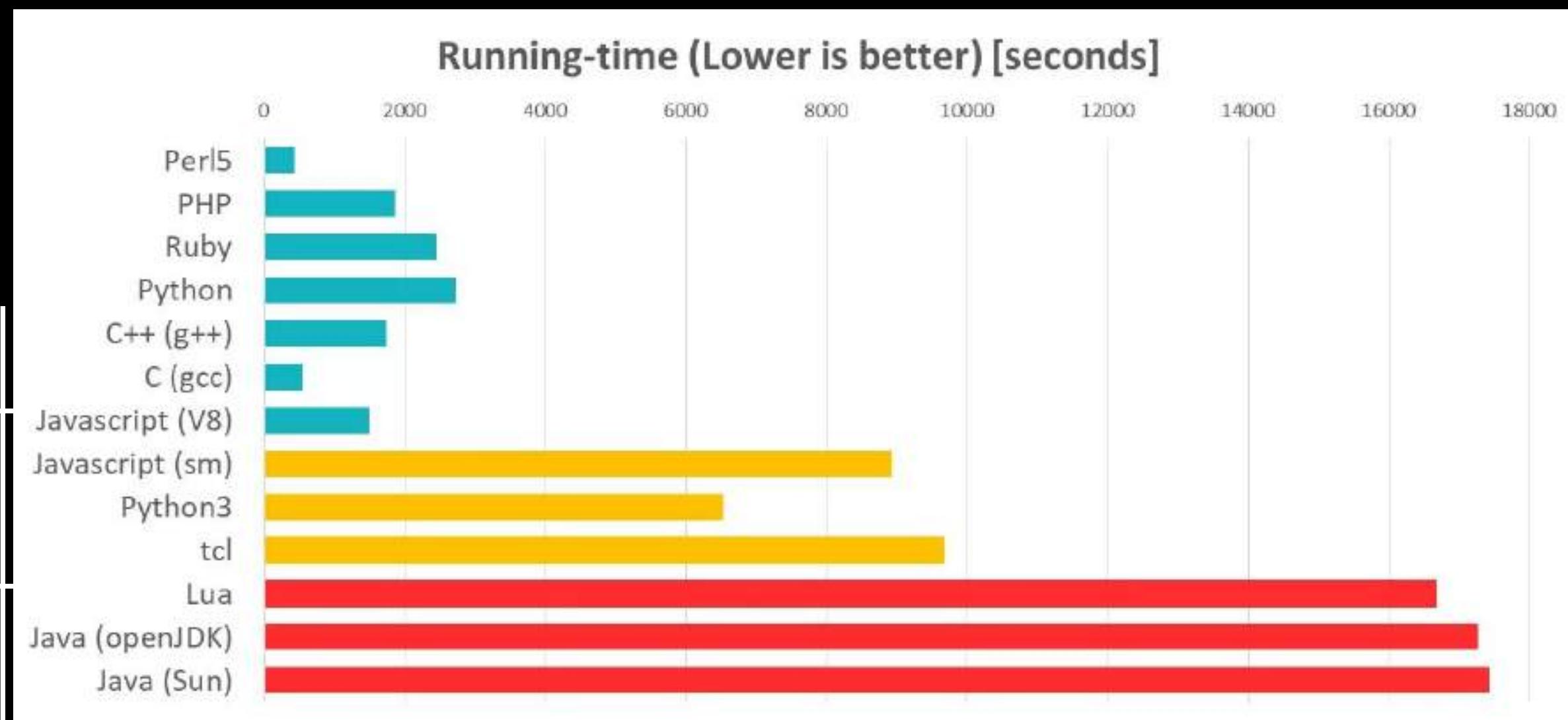
```
#include<stdio.h>
int main()
{
    int marks=83; ①
    ② if(marks>75) {
        printf("First class");
    }
    else if(marks>65){ ③
        printf("Second class");
    }
    else if(marks>55){
        printf("Third class");
    }
    ④ else{
        printf("Fourth class");
    }
    return 0;
}
```

```
1 #include<iostream>
2 using namespace std;
3
4 int main()
5 {
6
7     cout << "Contando del 1 al 5:" << endl;
8     for (int i = 1; i <= 5; i++) {
9         cout << i << " ";
10    }
11
12    return 0;
13 }
```

Funciones

```
1 #include <iostream>
2 using namespace std;
3
4 // Definición de función
5 int sumar(int a, int b) {
6     return a + b;
7 }
8
9 int main() {
10    int resultado = sumar(4, 6);
11    cout << "La suma es: " << resultado << endl;
12    return 0;
13 }
```

Comparación con otros lenguajes.



EJEMPLOS DE C++



Ejemplo 1.

```
1 #include <iostream> // Librería estándar para entrada y salida
2 using namespace std; // Para no escribir std:: antes de cout o cin
3
4 int main() {
5     double num1, num2; // Declaramos dos variables para los números que el usuario va a ingresar
6     char oper; // Variable para el operador matemático que se usará (+, -, *, /)
7
8     // Solicitamos el primer número
9     cout << "Ingrese el primer número: ";
10    cin >> num1;
11
12    // Pedimos al usuario que escriba el operador que desea usar
13    cout << "Ingrese el operador (+, -, *, /): ";
14    cin >> oper;
15
16    // Solicitamos el segundo número
17    cout << "Ingrese el segundo número: ";
18    cin >> num2;
19
20    // Usamos un switch para decidir qué operación se va a realizar según el operador ingresado
21    switch (oper) {
22        case '+':
23            cout << "Resultado: " << num1 + num2;
24            break;
25        case '-':
26            cout << "Resultado: " << num1 - num2;
27            break;
28        case '*':
29            cout << "Resultado: " << num1 * num2;
30            break;
31        case '/':
32            // Aquí usamos una condición if para asegurarnos de que no haya división entre cero
33            if (num2 != 0)
34                cout << "Resultado: " << num1 / num2;
35            else
36                cout << "Error: división por cero."; // Error común cuando el segundo número es 0
37            break;
38        default:
39            // Si el usuario escribió un operador no reconocido
40            cout << "Operador no válido.";
41    }
42
43    return 0; // Fin del programa
44 }
```

Ejemplo 2.

```
1 #include <iostream> // Librería para entrada y salida
2 using namespace std;
3
4 // Creamos una clase llamada Estudiante
5 class Estudiante {
6 private:
7     // Variables privadas (solo se usan dentro de la clase)
8     string nombre;
9     int edad;
10    float promedio;
11
12 public:
13     // Método para pedir al usuario los datos del estudiante
14 void ingresarDatos() {
15     cout << "Nombre: ";
16     cin >> nombre;
17     cout << "Edad: ";
18     cin >> edad;
19     cout << "Promedio: ";
20     cin >> promedio;
21 }
22
23 // Método para mostrar la información almacenada
24 void mostrarDatos() {
25     cout << "\n--- Información del estudiante ---\n";
26     cout << "Nombre: " << nombre << endl;
27     cout << "Edad: " << edad << endl;
28     cout << "Promedio: " << promedio << endl;
29 }
30
31
32 int main() {
33     Estudiante alumno;           // Creamos un objeto de la clase Estudiante
34     alumno.ingresarDatos();    // Llamamos al método para pedir los datos al usuario
35     alumno.mostrarDatos();     // Mostramos los datos que se guardaron
36
37     return 0; // Fin del programa
38 }
```

Ejemplo 3.

```
1 #include <iostream>
2 using namespace std;
3
4 // Clase para representar una cuenta de banco
5 class CuentaBancaria {
6 private:
7     string titular;    // Nombre del dueño de la cuenta
8     double saldo;      // Dinero disponible en la cuenta
9
10 public:
11     // Constructor: se usa al crear el objeto para asignar los valores iniciales
12     CuentaBancaria(string nombre, double montoInicial) {
13         titular = nombre;          // Guardamos el nombre del titular
14         saldo = montoInicial;    // Guardamos el saldo inicial
15     }
16
17     // Método para agregar dinero a la cuenta
18     void depositar(double monto) {
19         saldo += monto; // Sumamos el dinero depositado al saldo
20         cout << "Depósito exitoso. Nuevo saldo: " << saldo << endl;
21     }
22
23     // Método para retirar dinero
24     void retirar(double monto) {
25         // Condición: solo permitimos el retiro si hay suficiente dinero
26         if (monto <= saldo) {
27             saldo -= monto; // Restamos el monto retirado del saldo
28             cout << "Retiro exitoso. Saldo restante: " << saldo << endl;
29         } else {
30             // Si no hay suficiente dinero, se muestra un error
31             cout << "Fondos insuficientes." << endl;
32         }
33     }
34
35     // Método para mostrar el estado actual de la cuenta
36     void mostrarSaldo() {
37         cout << "\nTitular: " << titular << " | Saldo actual: $" << saldo << endl;
```

Ejemplo 3.1.

```
34
35     // Método para mostrar el estado actual de la cuenta
36 void mostrarSaldo() {
37     cout << "\nTitular: " << titular << " | Saldo actual: $" << saldo << endl;
38 }
39 }
40
41 int main() {
42     // Creamos una cuenta bancaria con un nombre y un saldo inicial
43     CuentaBancaria cuenta("Juan Pérez", 1000.0);
44
45     cuenta.mostrarSaldo();      // Mostramos el saldo actual
46     cuenta.depositar(500);      // Agregamos dinero
47     cuenta.retirar(300);        // Retiramos dinero (válido)
48     cuenta.retirar(1500);       // Intentamos retirar más de lo que hay (debería fallar)
49
50     return 0;
51 }
```

Preguntas

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.

padre(juan, maria).

padre(juan, jose).

padre(jose, luis).

abuelo(X, Y) :- padre(X, Z), padre(Z, Y).

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  
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?



QUE COMIENCE EL JUEGO.

- 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