

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 el 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 contenidos en sitios web. Ejemplos:

Java Script: 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.

Power Shell: 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.

Fácilidad de integración: Pueden interactuar con otros lenguajes y 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.

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

Multiplataformas: 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.

Analís 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 la aplicación.

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 conocido 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 Comprehensive 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 IO::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 implique 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.

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

Multiplataforma: Funciona en la mayoría de los sistemas 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 entornos.

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 condiciones (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 Visual:

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

2. Estandarización:

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

3. Ejecutabilidad:

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: Flecha que lleva a la acción de inscribir al estudiante.

• No: Flecha que lleva a la acción de solicitar datos adicionales.

• 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.

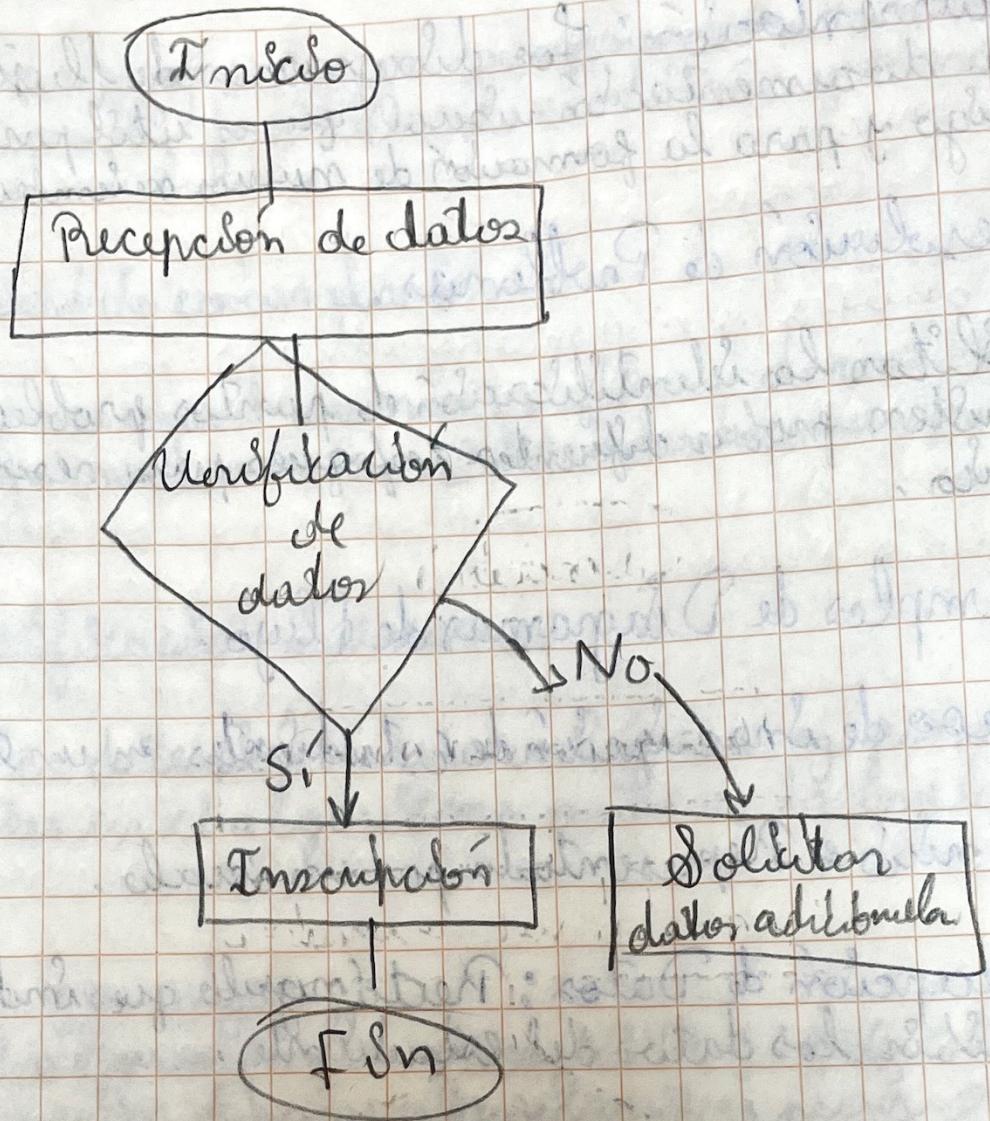
Resolución de Problemas.

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

Ejemplos de Diagramas de Flujo.

Proceso de Inscripción de Estudiantes en un Sistema.

1. **Título**: 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 adicionales.
4. **Inscripción del estudiante**: Rectángulo que indica la acción de inscribir al estudiante.



Algoritmo para calcular el Factorial de un número.

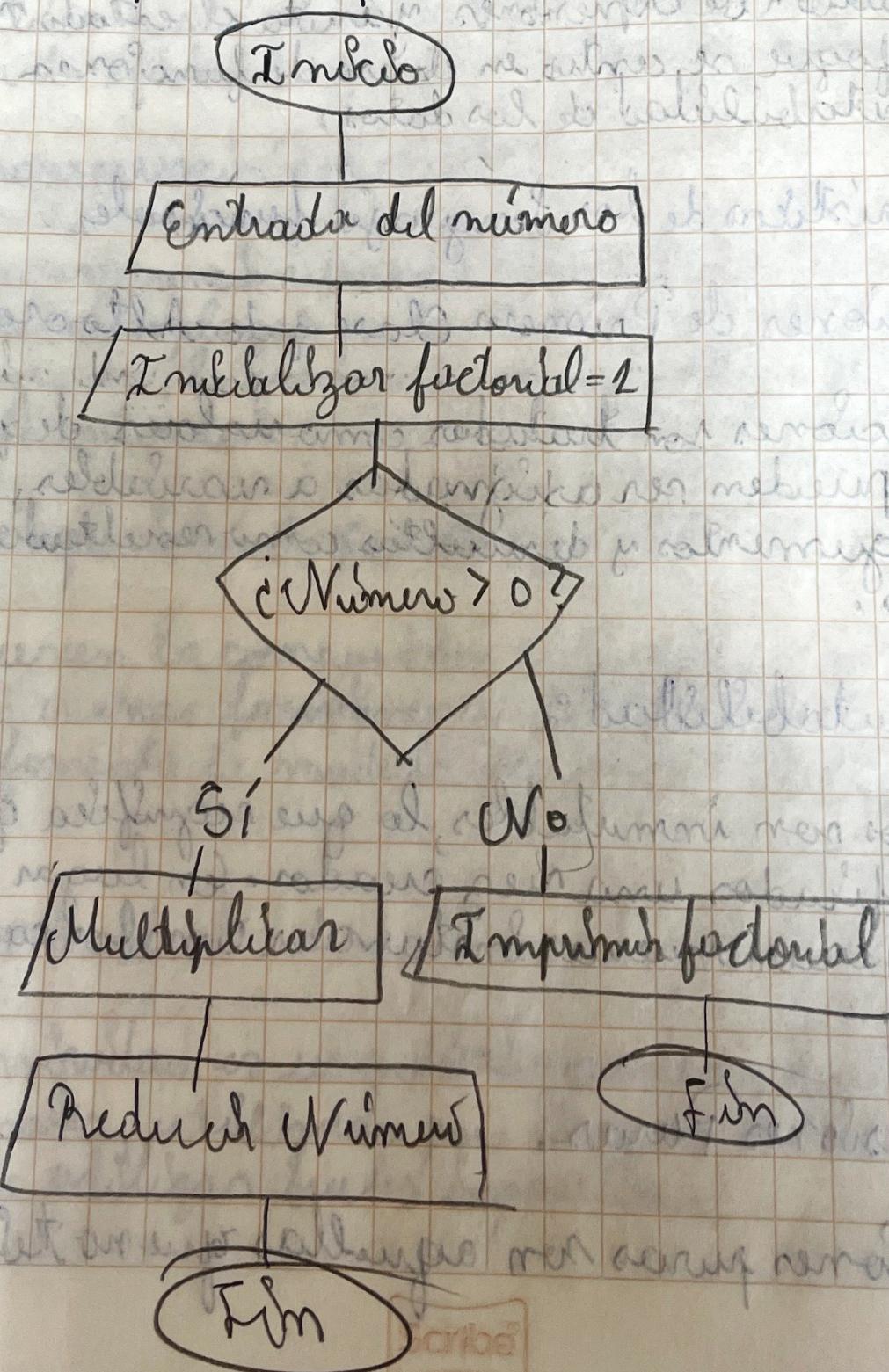
1. **Inicio**: Representado por un óvalo.
2. **Entrada del Número**: Rectángulo que indica la acción de introducir el número.
3. **Inicializar Factorial = 1**: Rectángulo que indica la acción de inicializar una variable factorial a 1.
4. **Si Número > 0 ?**: Rombo que representa la decisión de si el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

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.

Se 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 financieros