Conceptos y Paradigmas de Lenguajes de Programación - Práctica 1

EJERCICIO 1

1951 - 1955: Lenguajes tipo Assembly

- **Lenguajes de bajo nivel**: Los lenguajes tipo Assembly se crearon para interactuar directamente con la máquina. Esto proporcionaba un control más directo sobre el hardware.
- **Programación más cercana al hardware**: A diferencia de los lenguajes de alto nivel que utilizan abstracciones, los lenguajes tipo assembly traducen instrucciones casi directamente a las operaciones de la máquina.

1956 - 1960: FORTRAN, ALGOL 58, ALGOL 60, LISP

- **Lenguajes de alto nivel**: Estos lenguajes permiten a los programadores escribir código más abstracto, lo que facilita la programación.
- **Primeras lenguas científicas y matemáticas**: **FORTRAN** fue diseñado para cálculos científicos y numéricos.
- **Estructuras de control más avanzadas**: **ALGOL** introdujo conceptos como las estructuras de control de flujo (condicionales y bucles).
- **Soporte para programación funcional**: **LISP** introdujo la programación funcional, utilizando funciones matemáticas y recursividad.
- **Abstracción y formalización de algoritmos**: **ALGOL** ayudó a establecer la base de muchos conceptos de programación moderna, como las definiciones de variables y las funciones.

1961 - 1965: COBOL, ALGOL 60, SNOBOL, JOVIAL

- **Lenguajes orientados a negocios**: **COBOL** fue diseñado específicamente para aplicaciones de negocios, con un enfoque en la manipulación de datos y registros.
- **Expresión de patrones y cadenas de texto**: **SNOBOL** se centró en el procesamiento de cadenas de texto y patrones, lo que permitió manipular y analizar datos textuales de manera más fácil.
- **Lenguajes de sistemas y aplicaciones militares**: **JOVIAL** fue diseñado para la programación de aplicaciones militares.
- **Ampliación de ALGOL**: **ALGOL 60** fue extendido y perfeccionado, con más formalización en las estructuras de control y la gramática.

1966 - 1970: APL, FORTRAN 66, BASIC, PL/I, SIMULA 67, ALGOL-W

- **Notación matemática avanzada**: **APL** introdujo una notación matemática compacta para la manipulación de arrays y matrices.
- **Simplificación para principiantes**: **BASIC** fue diseñado para facilitar la programación a los estudiantes y personas sin experiencia previa.
- **Programación orientada a objetos**: **SIMULA 67** introdujo la programación orientada a objetos (OOP), un modelo de programación que usa objetos y clases.
- **Lenguaje multi-paradigma**: **PL/I** combinó características de varios paradigmas de programación, como el imperativo, el estructurado y el orientado a datos.

1971 - 1975: Pascal, C, Scheme, Prolog

- **Programación estructurada**: **Pascal** promovió un enfoque estructurado para la programación, con control explícito sobre las estructuras de datos y la lógica de ejecución.
- **C, lenguaje de sistemas**: **C** introdujo un lenguaje cercano al sistema operativo y a la arquitectura de la computadora, permitiendo la programación de bajo nivel y la creación de sistemas operativos.
- **Lenguaje lógico**: **Prolog** introdujo la programación lógica, un paradigma basado en la lógica formal y el razonamiento deductivo.
- **Lenguaje de programación funcional**: **Scheme** es una versión de Lisp, que profundizó en la programación funcional, haciendo hincapié en la recursión y las funciones de primer orden.

1976 - 1980: Smalltalk, Ada, FORTRAN 77, ML

- **Programación orientada a objetos**: **Smalltalk** fue uno de los primeros lenguajes puramente orientados a objetos, con una fuerte integración de clases, objetos y mensajes.
- **Seguridad y confiabilidad**: **Ada** fue diseñado para aplicaciones críticas, con un enfoque en la confiabilidad, seguridad y manejo de excepciones.
- **Programación funcional avanzada**: **ML** (Meta Language) introdujo una potente sistema de tipos y soporte para la programación funcional.
- **Actualización de FORTRAN**: **FORTRAN 77** actualizó FORTRAN con soporte para cadenas, estructuras de control más avanzadas y mejores capacidades de modularización.

1981 - 1985: Smalltalk 80, Turbo Pascal, Postscript

- **Estandarización de OOP**: **Smalltalk 80** consolidó el modelo de programación orientada a objetos, incorporando clases, objetos, herencia y polimorfismo como elementos fundamentales.
- **Compilación eficiente**: **Turbo Pascal** trajo un compilador extremadamente rápido que favoreció el desarrollo ágil de aplicaciones.
- **Lenguaje de descripción de páginas**: **Postscript** introdujo un lenguaje de programación para describir gráficos e imágenes de manera precisa, utilizado en impresoras láser y gráficos.

1986 - 1990: FORTRAN 90, C++, SML

- **Orientación a objetos en C++**: **C++** introdujo la programación orientada a objetos en un lenguaje basado en C, permitiendo la creación de clases, objetos y la herencia.
- **Mejoras en FORTRAN**: **FORTRAN 90** mejoró la capacidad para manejar arrays multidimensionales y la modularización del código.
- **Programación funcional avanzada**: **SML** (Standard ML) fortaleció el paradigma funcional, con un sistema de tipos muy robusto.

1991 - 1995: TCL, PERL, HTML

- **Lenguaje para interfaces gráficas**: **TCL** se diseñó como un lenguaje para crear interfaces gráficas de usuario (GUI).
- **Manipulación de cadenas y expresiones regulares**: **PERL** se destacó por su capacidad para manipular cadenas de texto y el uso extensivo de expresiones regulares.
- **Lenguaje de marcas**: **HTML** introdujo un lenguaje de marcado para estructurar contenido web, convirtiéndose en la base para la creación de sitios web.

1996 - 2000: Java, Javascript, XML

- **Portabilidad y desarrollo de aplicaciones web**: **Java** introdujo la idea de "escribir una vez, ejecutar en cualquier parte", permitiendo que las aplicaciones se ejecuten en cualquier plataforma mediante la máquina virtual Java (JVM).
- **Interactividad web**: **Javascript** permitió la creación de páginas web interactivas, integrándose en el navegador para el desarrollo del frontend dinámico.
- **Formato de datos estructurados**: **XML** proporcionó un estándar para el intercambio de datos estructurados entre sistemas, siendo extensible y fácil de leer tanto para humanos como para máquinas.

EJERCICIO 2

Java fue creado en 1991 por un equipo de ingenieros de Sun Microsystems, liderado por James Gosling. Originalmente, Java fue diseñado bajo el nombre de Oak como parte de un proyecto para dispositivos electrónicos de consumo, como televisores y electrodomésticos inteligentes. Sin embargo, al principio no tuvo mucho éxito en este campo.

En 1995, Sun Microsystems cambió el nombre del lenguaje a Java y lanzó una versión mejorada que sería utilizada para desarrollo web, aprovechando su portabilidad y capacidad para ejecutarse en cualquier plataforma. El lema Write Once, Run Anywhere (WORA) se popularizó, destacando la capacidad de las aplicaciones Java de ejecutarse en cualquier sistema que tuviera la Máquina Virtual de Java (JVM), sin importar el sistema operativo subyacente.

En ese mismo año, Java fue integrado en los navegadores web, permitiendo que se crearan aplicaciones interactivas y dinámicas en la web, lo que provocó una gran expansión de su uso. Java 1.0, lanzado en 1996, fue la primera versión oficial y sentó las bases para el desarrollo de aplicaciones empresariales, sistemas distribuidos, y aplicaciones web.

Durante los años 2000 y 2010, Java continuó evolucionando, mejorando la seguridad, el rendimiento y las capacidades de programación orientada a objetos (OOP). En 2004, Sun Microsystems lanzó Java 5, que introdujo nuevas características como generics, metadata annotations y enumeraciones.

En 2009, Oracle Corporation adquirió Sun Microsystems, y Java pasó a ser propiedad de Oracle, lo que impulsó nuevas versiones del lenguaje, incluyendo mejoras en el rendimiento y nuevas características como lambdas (Java 8, 2014).

Hoy en día, Java sigue siendo uno de los lenguajes más populares y ampliamente utilizados, especialmente en el desarrollo de aplicaciones empresariales, aplicaciones

móviles (Android) y sistemas grandes y distribuidos. Con el OpenJDK, el desarrollo de Java ha sido más accesible y abierto a la comunidad, permitiendo contribuciones continuas al lenguaje, y se ha mantenido relevante y actualizado a lo largo de los años, consolidándose como un pilar fundamental en la industria del software.

EJERCICIO 3

Un buen lenguaje de programación debería tener los siguientes atributos:

- Simplicidad y Legibilidad
- Claridad en los bindings
- Confiabilidad
- Soporte
- Abstracción
- Ortogonalidad
- Eficiencia

Simplicidad y Legibilidad: Los lenguajes de programación deberían:
- Poder producir programas fáciles de escribir y de leer
- Resultar fáciles a la hora de aprenderlo y/o enseñarlo
- La estructura subyacente del algoritmo y los datos que el programa representa deben quedar en manifiesto al inspeccionar el texto del programa

Claridad en los bindings: Los elementos de los lenguajes de programación pueden ligarse a sus atributos o propiedades en diferentes momentos:
- Definición del lenguaje
- Implementación del lenguaje
- Escritura del programa
- Compilación
- Cargado del programa
- Ejecución

La ligadura en cualquier caso debe ser clara.

Confiabilidad: Relacionada con la seguridad.
- Chequeo de tipos: cuanto antes se encuentren errores menos costoso resulta realizar arreglos.
- Manejo de excepciones: la habilidad para interceptar errores en tiempo de ejecución, tomar medidas correctivas y continuar.

Soporte:
- Debería ser accesible para cualquiera que quiera usarlo o instalarlo (lo ideal sería que su compilador o intérprete sea de dominio público).
- Debería poder ser implementado en diferentes plataformas
- Deberían existir diferentes medios para familiarizarse con el mismo: tutoriales, textos, cursos, etc.

Abstracción:
- Capacidad de definir y usar estructuras u operaciones complicadas de manera que sea posible ignorar muchos de los detalles.
- Concepto clave para manejar la complejidad, abstracción de procesos y de datos.

- Le ahorra al programador tiempo de desarrollo al proporcionar algoritmos ya implementados.

Ortogonalidad:
- Refiere a la posibilidad que ofrece el lenguaje de poder combinar sus elementos sin producir errores.
- Significa que un conjunto pequeño de constructores primitivos, puede ser combinado en número relativamente pequeño a la hora de construir estructuras de control y datos. Cada combinación es legal y con sentido.

Eficiencia: se relaciona con:
- Tiempo y espacio: capacidad de un programa para realizar una tarea de manera correcta y consumiendo pocos recursos de memoria, espacio en disco, tiempo de ejecución, tráfico de red, etc.
- Esfuerzo humano: que mejore el rendimiento del programador.
- Optimizable: capacidad del lenguaje de programación de venir optimizado para tareas específicas.

Java cumple con las características clave que lo hacen un buen lenguaje de programación.

1. **Simplicidad y legibilidad**: Aunque tiene una sintaxis más estricta que otros lenguajes como Python, su estructura organizada (con clases y métodos) y su enfoque orientado a objetos facilitan el desarrollo y el mantenimiento del código. Tiene como desventaja que su verbosidad puede reducir la simplicidad.

2. **Claridad en los Bindings**: En Java, los nombres de variables, métodos y clases están claramente definidos y se asocian de forma explícita con sus respectivos valores o comportamientos, lo que elimina ambigüedades.

3. **Confiabilidad**: Java es conocido por ser confiable gracias a su manejo de excepciones, la gestión automática de memoria (garbage collection) y tiene tipado fuerte y estático, que previene errores comunes de programación.

4. **Soporte**: Java tiene un extenso ecosistema de bibliotecas, frameworks y herramientas, lo que facilita el desarrollo y mantenimiento de aplicaciones. Además, es ampliamente utilizado y respaldado por una gran comunidad y documentación.

5. **Abstracción**: Java permite la abstracción mediante la programación orientada a objetos (POO), ocultando los detalles internos y proporcionando interfaces claras para interactuar con las clases y objetos.

6. **Ortogonalidad**: Java es ortogonal en su diseño, lo que significa que sus características no se interfieren entre sí. Las operaciones funcionan de manera coherente, lo que hace que el lenguaje sea más predecible y fácil de usar.

7. **Eficiencia**: Aunque no es tan eficiente como lenguajes compilados como C, Java es bastante eficiente debido a la JVM (Java Virtual Machine) y las optimizaciones de la compilación Just-In-Time (JIT), lo que permite un buen rendimiento en la mayoría de los casos.

En resumen, Java es un lenguaje **sencillo, legible, confiable y eficiente**, ideal para aplicaciones grandes, y tiene un fuerte **soporte** con una amplia comunidad y herramientas.

EJERCICIO 4

Lenguaje de programación: Java y Python.

Tipos de expresiones

Java: es un lenguaje de tipado estático, por lo que las expresiones en Java deben ser compatibles con los tipos de datos definidos en tiempo de compilación.

- Expresiones aritméticas: Se usan operadores como `+`, `-`, `*`, `/`, `%` para realizar operaciones matemáticas.
  - Ejemplo: int result = 10 + 5;

- Expresiones lógicas: Se usan para combinar condiciones, con operadores como `&&`, `||`, `!`.
  - Ejemplo: `boolean isValid = (x > 10) && (y < 20);`

- Expresiones de comparación: Usan operadores como `==`, `!=`, `<`, `>`, `<=`, `>=` para comparar valores.
  - Ejemplo: `boolean isEqual = (x == y);`

- Expresiones de asignación: Se utilizan para asignar valores a variables.
  - Ejemplo: `int a = 5;`

- Expresiones de llamada a métodos: Se invocan métodos para obtener valores.
  - Ejemplo: `String result = myMethod("Hello");`

Python: es un lenguaje de tipado dinámico, lo que significa que las expresiones no requieren que los tipos se definan explícitamente, y Python los determina en tiempo de ejecución.

- Expresiones aritméticas: Al igual que en Java, se pueden usar operadores matemáticos.
  - Ejemplo: `result = 10 + 5`

- Expresiones lógicas: Usan operadores lógicos como `and`, `or`, `not`.
  - Ejemplo: `is_valid = (x > 10) and (y < 20)`

- Expresiones de comparación: Python también permite comparar valores con operadores como `==`, `!=`, `<`, `>`, `<=`, `>=`.
  - Ejemplo: `is_equal = (x == y)`

- Expresiones de asignación: Python usa una sintaxis simple para asignación sin necesidad de declarar tipos.
  - Ejemplo: `a = 5`

- Expresiones de llamada a funciones: Invocar funciones es común en Python.
  - Ejemplo: `result = my_function("Hello")`

Facilidades para la organización del programa

Java: tiene una fuerte orientación a objetos y está diseñado para estructurar el código en clases y objetos. A continuación, algunas de las principales facilidades para organizar programas en Java:

- Clases y objetos: Todo en Java se organiza dentro de clases. Las clases permiten agrupar datos (variables) y comportamientos (métodos) en un solo contenedor.

- Paquetes (Packages): Java usa paquetes para agrupar clases relacionadas, lo que facilita la organización de proyectos grandes y la modularización del código.

- Interfaz de programación orientada a objetos (POO): Java implementa características de la POO como la herencia, polimorfismo, encapsulación y abstracción, lo que permite organizar el código de manera estructurada y reutilizable.

- Excepciones: Java permite gestionar errores y excepciones usando bloques `try-catch`, lo que ayuda a mantener el código organizado y a gestionar fallos de manera controlada.

Python: también permite organizar programas de manera eficiente, pero con un enfoque más sencillo que Java. Algunas de las facilidades incluyen:

- Funciones y módulos: En Python, los programas pueden dividirse en funciones (definidas con `def`) y módulos (archivos `.py`). Los módulos permiten agrupar funciones relacionadas, lo que facilita la modularización del código.

- Clases y objetos: Python soporta la programación orientada a objetos, permitiendo definir clases y crear objetos, aunque no es tan estricta como Java.

- Paquetes (Packages): Al igual que Java, Python permite usar paquetes para organizar el código y facilitar la gestión de proyectos grandes.

- Excepciones: Python maneja los errores usando bloques `try-except`, similar a Java, lo que mejora la organización y el control de errores.

Atributos del ejercicio anterior

Simplicidad y Legibilidad:
- Java: Cumple moderadamente. Aunque tiene una sintaxis más estricta que Python, su estructura organizada (con clases y métodos) y su enfoque orientado a objetos hacen que sea fácil de leer y entender para los programadores con experiencia. Sin embargo, la verbosidad de Java puede reducir la simplicidad.

- Python: Cumple plenamente. La sintaxis de Python es clara, concisa y fácil de entender, lo que facilita tanto su aprendizaje como su lectura. Su enfoque minimalista permite escribir código de manera eficiente y comprensible.

Claridad en los bindings:
- Java: Cumple. El uso de tipos explícitos y la declaración previa de variables hace que la asociación entre nombres y valores sea clara. Además, la orientación a objetos proporciona una forma clara de asociar comportamientos con objetos.

- Python: Cumple. Aunque no requiere declarar tipos explícitos, la claridad en los nombres de variables y funciones permite que los bindings sean fácilmente comprensibles.

Confiabilidad:
- Java: Cumple plenamente. La tipificación estática y el manejo robusto de excepciones en Java ayudan a evitar errores en tiempo de ejecución. Además, la JVM (Java Virtual Machine) proporciona un entorno controlado y fiable.

- Python: Cumple parcialmente. Python es confiable en términos de ejecución, pero su tipado dinámico puede conducir a errores que solo se detectan en tiempo de ejecución. A pesar de eso, el manejo de excepciones contribuye a la confiabilidad.

Soporte:
- Java: Cumple plenamente. Java tiene un soporte excepcional, con una gran comunidad de desarrolladores, vastas bibliotecas y frameworks (como Spring y Hibernate), y una excelente documentación.

- Python: Cumple plenamente. Python también tiene un ecosistema robusto, con una gran comunidad de usuarios y una amplia variedad de bibliotecas y frameworks (como Django, Flask, NumPy, Pandas) que facilitan su uso en diversos campos.

Abstracción:
- Java: Cumple plenamente. La programación orientada a objetos de Java permite una excelente abstracción de datos y comportamientos. Las clases y las interfaces proporcionan un fuerte mecanismo para abstraer detalles complejos.

- Python: Cumple. Aunque Python no es tan estricto como Java, también permite la abstracción mediante clases y funciones, y su sintaxis más flexible hace que la creación de abstracciones sea más sencilla.

Ortogonalidad:
- Java: Cumple. Las características de Java están diseñadas para ser coherentes y no se interfieren entre sí, lo que permite trabajar con diferentes elementos (clases, objetos, interfaces, excepciones) de forma independiente.

- Python: Cumple. Python es ortogonal en su diseño. Sus funcionalidades y características no se superponen de manera innecesaria, y se pueden usar de manera independiente sin que interfieran entre sí.

Eficiencia:
- Java: Cumple moderadamente. Java es más eficiente que Python debido a la compilación de bytecode y las optimizaciones realizadas por la JVM. Sin embargo, no es tan rápido como los lenguajes de bajo nivel como C o C++.

- Python: No cumple completamente. Python es más lento debido a que es un lenguaje interpretado. Aunque se pueden usar herramientas como Cython o NumPy para mejorar su rendimiento en ciertos casos, en términos generales, Python es menos eficiente que Java.

Conclusión:
Java es confiable, eficiente y estructurado, adecuado para aplicaciones grandes, mientras que Python destaca por su simplicidad, legibilidad y flexibilidad, ideal para desarrollo rápido y proyectos más pequeños. Ambos cumplen bien con la mayoría de los atributos clave, pero Java es más eficiente y Python es más fácil de aprender y usar.

EJERCICIO 5

Características relevantes de **ADA**

Tipos de datos
ADA es un lenguaje fuertemente tipado, que incluye tipos estándar como Integer, *Float*, *Boolean*, *Character*, y *String*. También permite definir tipos personalizados mediante registros y enumeraciones, y ofrece tipos de acceso (*punteros*) para gestionar memoria.

Tipos abstractos de datos – Paquetes
ADA soporta paquetes para organizar y encapsular tipos de datos y operaciones. Un paquete tiene una especificación (interfaz pública) y un cuerpo (implementación), lo que promueve la modularidad y la reutilización de código.

Estructuras de datos
ADA ofrece arrays (arreglos), registros (similares a estructuras), y soporta listas enlazadas mediante *punteros*. Además, se pueden definir tipos de datos con acceso controlado para un manejo eficiente de memoria.

Manejo de excepciones
ADA permite un manejo robusto de excepciones con bloques *`begin ... exception`*. Los programadores pueden definir excepciones personalizadas y manejar errores como *Constraint_Error* o *Storage_Error*, lo que mejora la confiabilidad del software.

Manejo de concurrencia

ADA tiene soporte nativo para concurrencia con tareas, que son unidades de ejecución concurrente. Las tareas se sincronizan mediante *entries y protected objects* para garantizar la exclusión mutua, lo que facilita la programación multitarea segura.

Conclusión

En resumen, ADA es un lenguaje seguro y eficiente, con un enfoque en la modularidad, gestión de errores y concurrencia, ideal para aplicaciones críticas y sistemas en tiempo real.

EJERCICIOS 6

¿Para qué fue creado Java?
Java fue creado para ser un lenguaje independiente de la plataforma, permitiendo que las aplicaciones se ejecutaran en cualquier dispositivo o sistema operativo sin modificaciones, gracias a la Java Virtual Machine (JVM).

¿Qué cambios le introdujo a la Web?
Java introdujo *applets y servlets* para hacer las páginas web más interactivas y dinámicas. Los applets permitieron ejecutar pequeños programas en los navegadores, y los servlets habilitaron la creación de aplicaciones web dinámicas en el servidor.

¿Java es un lenguaje dependiente de la plataforma?
No, Java es independiente de la plataforma. El código Java se compila en *bytecode*, que puede ejecutarse en cualquier dispositivo con una JVM instalada, independientemente del sistema operativo.

EJERCICIO 7

¿Sobre qué lenguajes está basado Java?
Java está basado principalmente en C (por su sintaxis), C++ (por la programación orientada a objetos) y Smalltalk (por sus conceptos de objetos).

EJERCICIO 8

¿Qué son los applets? ¿Qué son los servlets?

Applets
Son pequeñas aplicaciones en Java que se ejecutan dentro de un navegador web para añadir interactividad a las páginas.

Servlets
Son programas en Java que se ejecutan en el servidor web, respondiendo a solicitudes HTTP y generando contenido dinámico para las páginas web.

EJERCICIO 9

Estructura de un programa escrito en C

Un programa en C tiene una estructura básica que incluye:

1. Directivas de preprocesador: Instrucciones que se procesan antes de la compilación, como *#include* para incluir bibliotecas estándar o *#define* para definir macros.

```
#include <stdio.h>
```

2. Función principal: Todo programa en C debe tener una función *main()*, que es el punto de entrada del programa.

```
int main() {
    // Código del programa
    return 0;
}
```

3. Declaración de variables: Dentro de la función *main()* (o cualquier otra función), se declaran las variables que se utilizarán.

```
int a, b;
```

4. Cuerpo del programa: El código del programa, con instrucciones de control (*if, for, while*), operaciones, llamadas a funciones, etc.

```
printf("Hello, World!");
```

5. Funciones adicionales: Aparte de *main()*, se pueden definir otras funciones para modularizar el programa.

¿Existe anidamiento de funciones en C?
No, en C no es posible anidar funciones. Es decir, no puedes definir una función dentro de otra. Las funciones deben ser definidas de forma independiente en el mismo nivel de código, fuera de otras funciones.

EJERCICIO 10

Manejo de expresiones en C

C maneja una amplia variedad de expresiones que incluyen:

1. Aritméticas: Son las expresiones que realizan operaciones matemáticas como suma, resta, multiplicación, división y módulo.

```
int sum = a + b;
```

2. Relacionales: Se utilizan para comparar dos valores, como mayor que (>), menor que (<), igual a (==), etc.

```
if (a > b) { /* hacer algo */ }
```

3. Lógicas: Las expresiones lógicas incluyen operadores como *&& (Y lógico), || (O lógico), y ! (NO lógico).*

if (a > b && c < d) { /* hacer algo */ }

4. Asociación de valores: La asignación de valores se hace con el operador =, y se pueden usar operadores compuestos como *+=, -=,* etc.

a += 5; // Es equivalente a a = a + 5;

5. Incremento y decremento: C soporta los operadores *++ (incremento) y -- (decremento)* para aumentar o disminuir una variable en uno.

a++; // Incrementa a en 1
b--; // Decrementa b en 1

6. Bit a bit: C permite operar sobre bits con operadores como *& (AND), | (OR), ^ (XOR), << (desplazamiento a la izquierda) y >> (desplazamiento a la derecha).*

int x = a & b; // Operación AND a nivel de bits

7. Expresiones condicionales: Se pueden usar expresiones condicionales con el operador ternario *? :* para elegir entre dos valores dependiendo de una condición.

int result = (a > b) ? a : b; // Si a > b, result = a, si no, result = b

Conclusión

En resumen, C ofrece una variedad de expresiones para trabajar con datos, realizar cálculos, y controlar el flujo del programa, utilizando operadores aritméticos, lógicos, relacionales, bit a bit y condicionales.

EJERCICIO 11

Tipos de programas y paradigmas

Python
- Tipos de programas: Desarrollo web, ciencia de datos, IA, automatización.
- Paradigma: Multiparadigma, especialmente orientado a objetos y funcional.
- POO: Permite el uso de clases, objetos, herencia y polimorfismo.
- Simplicidad y legibilidad: Su sintaxis clara y sencilla facilita la programación.

Ruby
- Tipos de programas: Desarrollo web con Ruby on Rails, aplicaciones de automatización, análisis de datos y scripts.
- Paradigma: Orientado a objetos puro (todo es un objeto).
- Flexible y dinámico: Permite modificar clases y objetos en tiempo de ejecución.
- Sintaxis expresiva: La sintaxis de Ruby es muy flexible, permitiendo escribir código de manera más intuitiva.

PHP
- Tipos de programas: Desarrollo web dinámico, CMS como WordPress.
- Paradigma: Principalmente procedural con soporte a orientación a objetos.

- Integración con HTML: PHP es conocido por su facilidad para integrarse con HTML y manejar formularios, bases de datos y otros elementos web.

EJERCICIO 12

Características importantes

Python
- Tipado: Dinámico. Se utiliza una jerarquía de directorios.
- Organización: Módulos y paquetes.
- Excepciones: Muy robusto, utilizando `try-except`.

Ruby
- Tipado: Dinámico.
- Organización: Clases y módulos. Los proyectos suelen usar el enfoque de gemas (librerías) para extender la funcionalidad.
- Excepciones: `begin-rescue-end`.

PHP
- Tipado: Dinámico, soporte para tipos fuertes.
- Organización: Archivos con funciones/clases.
- Sintaxis: Similar a C, con muchas funciones y estructuras integradas.
- Excepciones: `try-catch`.

Gobstone
- Tipado: Dinámico.
- Organización: Acciones secuenciales.
- Paradigma: Programación estructurada (educativa).

Processing
- Tipado: Dinámico, pero proporciona una sintaxis para trabajar con tipos de datos gráficos y visuales.
- Organización: Basado en funciones.
- Paradigma: Orientado a gráficos y arte visual.

RESUMEN

Python: Multiparadigma, especialmente funcional y orientado a objetos. Ideal para web, ciencia de datos y automatización.

Ruby: Orientado a objetos puro, conocido por su flexibilidad y el framework Rails.

PHP: Procedural y orientado a objetos, enfocado en desarrollo web.

Gobstone: Lenguaje educativo basado en programación estructurada.

Processing: Enfocado en arte visual interactivo, con una sintaxis simple similar a Java.

EJERCICIO 13

Paradigma y tipo de lenguaje de JavaScript

Paradigma
JavaScript es un lenguaje multiparadigma, lo que significa que soporta varios paradigmas de programación, principalmente:

- Programación imperativa: Se basa en la ejecución secuencial de instrucciones.
- Programación orientada a objetos (POO): Permite el uso de objetos y clases, aunque con un enfoque más flexible que en otros lenguajes como Java o C++.
- Programación funcional: Soporta funciones de primera clase, lo que permite pasar funciones como argumentos, devolver funciones y usar conceptos como *cerraduras (closures).*

Tipo de lenguaje
JavaScript es un lenguaje de alto nivel, interpretado, dinámico y event-driven (basado en eventos), utilizado principalmente para desarrollo web. Originalmente diseñado para la manipulación de páginas web, ahora se usa también en el lado del servidor (con Node.js).

EJERCICIO 14

Características importantes de JavaScript

Tipado de datos
JavaScript tiene tipado dinámico, lo que significa que las variables no requieren una declaración explícita de tipo y su tipo puede cambiar durante la ejecución del programa. Los tipos de datos básicos incluyen *números, cadenas, booleanos, objetos, arreglos y null.*

```
let x = 10;  // Number
x = "Hola";  // String
```

Excepciones
JavaScript maneja excepciones mediante *try-catch* para capturar y manejar errores en tiempo de ejecución.

```
try {
  let result = someFunction();
} catch (error) {
  console.error(error);
}
```

Variables
JavaScript utiliza tres tipos de declaración de variables:
1. var: Declaración global o de función (antiguo, menos recomendado).
2. let: Declaración de bloque, más segura que `var`.
3. const: Para declarar variables que no se pueden reasignar (constantes).

```
let a = 5;
const b = 10;
```

Otros aspectos importantes
- Asincronía: JavaScript maneja tareas asíncronas a través de *promesas y async/await.*
- Closures: Funciones internas que mantienen acceso al contexto de la función externa, incluso después de que esta haya terminado de ejecutarse.
- Hoisting: Las declaraciones de variables y funciones se "suben" al inicio de su contexto, permitiendo su uso antes de su definición.

JavaScript es flexible, potente y esencial para la programación web moderna, con un enfoque en la interacción con el usuario y el manejo de eventos.