Practica 1 CyPLP

1_
1951 - 1955: Lenguajes tipo assembly
Uso directo del hardware sin abstracción significativa.
Programación en código máquina con mnemónicos para facilitar la escritura.
Ejemplo: Lenguajes ensambladores específicos para cada arquitectura.

1956 - 1960: FORTRAN, ALGOL 58, ALGOL 60, LISP
FORTRAN: Introducción de estructuras de control como bucles y condicionales.
Primer lenguaje de alto nivel orientado a cálculos científicos.
ALGOL 58 y ALGOL 60: Primera estructura de bloques y anidación de sentencias.
LISP: Introducción del paradigma funcional y listas como estructura de datos principal.

1961 - 1965: COBOL, ALGOL 60, SNOBOL, JOVIAL
COBOL: Lenguaje orientado a negocios, con sintaxis cercana al inglés.
SNOBOL: Introducción de la manipulación avanzada de cadenas de texto.
ALGOL 60: Definición clara de la estructura de programas mediante bloques.
JOVIAL: Extensión de ALGOL 60 para aplicaciones militares y de defensa.

1966 - 1970: APL, FORTRAN 66, BASIC, PL/I, SIMULA 67, ALGOL-W
APL: Introducción de notación matemática compacta y procesamiento matricial.
FORTRAN 66: Estandarización del lenguaje FORTRAN.
BASIC: Facilita la enseñanza de programación con una sintaxis simple.
PL/I: Soporte para concurrencia y excepciones.
SIMULA 67: Primer lenguaje con soporte para programación orientada a objetos (POO).
ALGOL-W: Antecesor de Pascal, mejora en estructuras de datos y control.

1971 - 1975: Pascal, C, Scheme, Prolog
Pascal: Enfoque estructurado y tipado fuerte para enseñanza y desarrollo.
C: Introducción de la programación de sistemas con acceso a memoria de bajo nivel.
Scheme: Extensión de LISP con soporte para funciones de orden superior y recursión más expresiva.
Prolog: Introducción de la programación lógica y basada en reglas.

1976 - 1980: Smalltalk, Ada, FORTRAN 77, ML
Smalltalk: Populariza la programación orientada a objetos (POO) con un entorno gráfico interactivo.
Ada: Soporte para programación concurrente y modularidad en software crítico.
FORTRAN 77: Introducción de estructuras de control mejoradas (IF-THEN-ELSE).
ML: Introducción de la inferencia de tipos en programación funcional.

1981 - 1985: Smalltalk 80, Turbo Pascal, Postscript
Smalltalk 80: Primer entorno completamente basado en objetos con interfaz gráfica.
Turbo Pascal: Introducción de compilación rápida e IDE integrado.
Postscript: Lenguaje para diseño de impresión con capacidades gráficas.

1986 - 1990: FORTRAN 90, C++, SML
FORTRAN 90: Introducción de programación estructurada y matrices dinámicas.
C++: Expansión de C con programación orientada a objetos.
SML (Standard ML): Mejora en la inferencia de tipos y programación funcional estructurada.

1991 - 1995: TCL, PERL, HTML
TCL: Lenguaje de scripting embebido y flexible.
PERL: Manejo avanzado de cadenas y expresiones regulares.
HTML: Estándar para estructurar documentos en la web.

1996 - 2000: Java, Javascript, XML
Java: Introducción del concepto de "escribir una vez, ejecutar en cualquier lugar" mediante la JVM.
Javascript: Popularización del desarrollo web interactivo en el navegador.
XML: Estándar para estructuración de datos con etiquetas personalizadas.

2_
Historia breve del lenguaje de programación Java
Java fue desarrollado por James Gosling y su equipo en Sun Microsystems a principios de la década de 1990.
Originalmente, el proyecto comenzó en 1991 bajo el nombre "Oak", con el objetivo de crear un lenguaje para
dispositivos electrónicos embebidos e interactivos. Sin embargo, este mercado

no despegó, y el enfoque del
lenguaje cambió hacia el desarrollo de aplicaciones para la web.

En 1995, Sun Microsystems lanzó oficialmente Java 1.0, con su lema característico:
"Write Once, Run Anywhere" (Escribe una vez, ejecuta en cualquier lugar). Esta característica se debía a
la Máquina Virtual de Java (JVM), que permitía ejecutar programas en diferentes sistemas operativos sin
necesidad de modificar el código.

Java rápidamente se convirtió en un estándar para el desarrollo de aplicaciones web, empresariales y
móviles. En 2006, Sun liberó gran parte del código de Java como software de código abierto bajo la licencia
GPL. En 2010, Oracle adquirió Sun Microsystems y se convirtió en el nuevo propietario de Java.

Actualmente, Java sigue siendo ampliamente utilizado en el desarrollo de aplicaciones empresariales, Android,
sistemas embebidos y servidores web. Su evolución continúa con versiones regulares que mejoran el rendimiento,
la seguridad y las características del lenguaje.

3_
Un buen lenguaje de programación debe poseer ciertos atributos clave para facilitar el desarrollo, mantenimiento
y eficiencia del software. Aquí están algunos de los atributos más importantes, junto con ejemplos de lenguajes
que los cumplen:

Ortogonalidad
La ortogonalidad se refiere a que un número reducido de conceptos puede combinarse de manera predecible sin
generar efectos secundarios inesperados. Un lenguaje ortogonal permite escribir código más claro y fácil de
entender.
Ejemplo: LISP es altamente ortogonal, ya que sus estructuras básicas (listas y funciones) pueden combinarse sin
restricciones arbitrarias.

Expresividad
Un lenguaje expresivo permite escribir código conciso y claro con el mínimo esfuerzo, sin sacrificar precisión o
eficiencia.
Ejemplo: Python es un lenguaje expresivo, ya que permite escribir código simple y legible con menos líneas en
comparación con otros lenguajes como Java o C.

Legibilidad
La legibilidad es clave para que los programadores puedan comprender y mantener el código con facilidad.
Ejemplo: Pascal fue diseñado con una sintaxis estructurada y clara para facilitar la lectura y enseñanza de la
programación.

Simplicidad
Un lenguaje debe evitar complejidades innecesarias y mantener una sintaxis coherente.
Ejemplo: BASIC fue diseñado para ser fácil de aprender y usar, especialmente para principiantes.

Eficiencia
Un lenguaje eficiente permite ejecutar programas con el menor consumo de recursos posible.
Ejemplo: C es altamente eficiente, ya que permite acceso directo a memoria y optimización a nivel de hardware.

Portabilidad
Un buen lenguaje debe permitir que el código pueda ejecutarse en diferentes plataformas sin modificaciones
significativas.
Ejemplo: Java, gracias a su Máquina Virtual de Java (JVM), permite ejecutar código en distintos sistemas operativos
sin recompilación.

Modularidad
Debe permitir dividir el código en módulos reutilizables y bien organizados.
Ejemplo: Ada fue diseñado con un fuerte enfoque en la modularidad, lo que facilita el desarrollo de software grande y
mantenible.

Seguridad
El lenguaje debe proporcionar mecanismos para evitar errores comunes y vulnerabilidades de seguridad.
Ejemplo: Rust garantiza la seguridad de la memoria sin necesidad de un recolector de basura, evitando errores como
desbordamiento de búfer o uso de punteros nulos.
En la práctica, no existe un lenguaje que cumpla perfectamente con todos estos atributos, pero algunos, como Python,
Java y Rust, han logrado un buen equilibrio entre ellos.

4_
Tipos de expresiones en Python
En Python, una expresión es cualquier combinación de valores, variables, operadores y funciones que devuelven un
resultado. Los principales tipos de expresiones son:

Expresiones aritméticas: Realizan operaciones matemáticas.
Ejemplo: x = 5 + 3 * 2

Expresiones relacionales: Comparan valores y devuelven True o False.
Ejemplo: 10 > 5

Expresiones lógicas: Utilizan operadores lógicos (and, or, not).
Ejemplo: (x > 5) and (y < 10)

Expresiones de cadena: Concatenación y manipulación de textos.
Ejemplo: "Hola" + " Mundo"

Expresiones de lista, tuplas y diccionarios: Definen estructuras de datos.
Ejemplo: [1, 2, 3], ("a", "b"), {"clave": "valor"}

Expresiones lambda: Definen funciones anónimas en una sola línea.
Ejemplo: f = lambda x: x * 2

Expresiones de comprensión: Permiten construir listas, conjuntos o diccionarios de manera concisa.
Ejemplo: [x**2 for x in range(10)]

Facilidades para la organización del programa en Python
Python ofrece varias características para estructurar programas de manera organizada y modular:
Módulos: Se pueden dividir programas en archivos reutilizables (import nombre_modulo).

Paquetes: Agrupación de módulos en directorios organizados.
Funciones: Permiten reutilizar código (def mi_funcion():).
Clases y objetos: Soporte completo para Programación Orientada a Objetos (POO).
Espacios de nombres: Uso de variables locales y globales bien diferenciadas.
Scripts ejecutables: Se pueden escribir scripts independientes o programas más grandes.

Atributos del ejercicio anterior en Python

Atributo	¿Python lo posee?	Justificación
Ortogonalidad	✅ Sí	Python tiene un número reducido de reglas que se combinan de manera coherente.
Expresividad	✅ Sí	Permite escribir código claro y conciso con estructuras como listas por comprensión.
Legibilidad	✅ Sí	Su sintaxis clara, uso de indentación y palabras clave hacen que el código sea fácil de entender.
Simplicidad	✅ Sí	Su diseño minimalista evita sintaxis innecesaria y promueve buenas prácticas.
Eficiencia	❌ No siempre	Es más lento que C o Rust porque usa un intérprete y maneja la memoria automáticamente.
Portabilidad	✅ Sí	Se ejecuta en múltiples plataformas sin modificaciones gracias a la máquina virtual de Python.
Modularidad	✅ Sí	Soporta módulos, paquetes y bibliotecas que facilitan la organización del código.
Seguridad	✅ Sí	Maneja memoria de forma segura y evita errores como punteros nulos o buffer overflow.

En resumen, Python cumple con casi todos los atributos deseables en un lenguaje de programación, excepto en eficiencia, ya que prioriza la simplicidad y legibilidad sobre el rendimiento absoluto.

5_

Ada es un lenguaje de programación desarrollado en la década de 1980 por el Departamento de Defensa de EE.UU., con el objetivo de crear software

confiable y seguro. A continuación, se detallan sus características más relevantes en distintas áreas:

Tipos de datos

Ada es un lenguaje fuertemente tipado, lo que ayuda a evitar errores en tiempo de ejecución. Sus tipos de datos incluyen:

- **Tipos escalares:** Enteros (`Integer`), flotantes (`Float`), booleanos (`Boolean`), caracteres (`Character`).
- **Tipos subrango:** Se pueden definir rangos específicos para evitar valores fuera de un dominio permitido.
  - Ejemplo:

```
ada
CopiarEditar
subtype Edad is Integer range 0 .. 120;
```

- **Tipos enumerados:** Permiten definir un conjunto limitado de valores.
  - Ejemplo:

```
ada
CopiarEditar
type Dia is (Lunes, Martes, Miercoles, Jueves, Viernes);
```

- **Tipos modulares:** Usados para representar valores sin signo.
- **Tipos derivados:** Se pueden crear nuevos tipos basados en otros existentes.
- **Tipos de acceso (punteros):** Manejo seguro de referencias sin punteros nulos.

Tipos abstractos de datos – Paquetes

Ada soporta **encapsulamiento** y **abstracción de datos** mediante **paquetes**, los cuales permiten definir **tipos abstractos de datos (TAD)** de manera modular.

- **Paquetes**: Permiten organizar el código y definir interfaces claras. Se dividen en **especificación** y **cuerpo**.

- Ejemplo de un paquete con un tipo abstracto:

  Aquí, la implementación interna de `Pila` es privada y solo se accede mediante los procedimientos `Apilar` y `Desapilar`.

  ```
  ada
  CopiarEditar
  package Pilas is
     type Pila is private;
     procedure Apilar(P: in out Pila; X: Integer);
     function Desapilar(P: in out Pila) return Integer;
  private
     type Pila is array (1 .. 100) of Integer;
  end Pilas;
  ```

Estructuras de datos

Ada permite definir estructuras de datos complejas de manera segura y organizada:

- **Registros:** Similares a estructuras en C, permiten agrupar datos heterogéneos.
  - Ejemplo:

    ```
    ada
    CopiarEditar
    type Persona is record
       Nombre : String(1..50);
       Edad   : Integer;
    end record;
    ```

- **Matrices:** Permiten definir arreglos con rangos personalizados.
  - Ejemplo:

    ```
    ada
    CopiarEditar
    ```

```
type Vector is array (1..10) of Integer;
```

- **Cadenas:** Ada maneja cadenas de longitud fija o dinámica mediante `Unbounded_String`.
- **Listas enlazadas y estructuras dinámicas:** Se pueden implementar usando **tipos de acceso** (punteros seguros).

Manejo de excepciones

Ada tiene un potente sistema de manejo de excepciones que permite capturar errores en tiempo de ejecución y evitar fallos del sistema.

- Se pueden capturar excepciones mediante `begin ... exception`.
- Existen excepciones predefinidas como:
  - `Constraint_Error`: Error de límites en un array o subrango.
  - `Program_Error`: Llamada a una subrutina no válida.
  - `Storage_Error`: Falta de memoria.
- **Ejemplo de manejo de excepciones:**

```
ada
CopiarEditar
procedure Division_Segura is
   A, B, Resultado: Integer;
begin
   A := 10;
   B := 0;
   Resultado := A / B;
exception
   when Constraint_Error ⇒
      Put_Line("Error: División por cero.");
end Division_Segura;
```

Manejo de concurrencia

Ada fue uno de los primeros lenguajes en soportar concurrencia de forma nativa mediante **tareas** (`task`).

- **Tareas** (`task`): Permiten definir procesos concurrentes.
- **Rendezvous**: Mecanismo de sincronización entre tareas mediante la interacción con **entradas protegidas** (`entry`).
- **Protected Objects**: Alternativa más eficiente a las tareas, similar a monitores en otros lenguajes.
- **Ejemplo de tarea en Ada:**

```
ada
CopiarEditar
task Servidor is
   entry Solicitar;
end Servidor;

task body Servidor is
begin
   loop
      accept Solicitar do
         Put_Line("Solicitud recibida.");
      end Solicitar;
   end loop;
end Servidor;
```

Aquí, el `Servidor` espera a que otra tarea llame a `Solicitar`, ejecutando la sincronización de manera segura.

Conclusión

Ada es un lenguaje robusto que destaca en seguridad, modularidad y concurrencia. Gracias a su fuerte tipado, manejo seguro de memoria y excepciones, es ampliamente utilizado en **sistemas críticos**, como aviación, defensa y sistemas industriales.

6_

¿Para qué fue creado Java?

Java fue creado originalmente por **James Gosling** y su equipo en **Sun Microsystems** en 1991 bajo el proyecto "Oak". Su propósito inicial era

desarrollar un lenguaje para **dispositivos electrónicos embebidos e interactivos**, pero este mercado no prosperó.

En 1995, Java fue lanzado oficialmente con el objetivo de ser un **lenguaje multiplataforma** para el desarrollo de aplicaciones de escritorio y web. Su principal lema fue:

"Write Once, Run Anywhere" (*Escribe una vez, ejecuta en cualquier lugar*), destacando su independencia de la plataforma gracias a la **Máquina Virtual de Java (JVM)**.

¿Qué cambios introdujo Java en la Web?

Java revolucionó la web en los años 90 al introducir nuevas capacidades dinámicas y multiplataforma:

1. **Applets:** Pequeñas aplicaciones embebidas en páginas web que permitían interacción avanzada (aunque hoy en día están obsoletas).
2. **JSP (Java Server Pages):** Permite generar contenido dinámico en servidores web.
3. **Servlets:** Aplicaciones web más eficientes que los CGI tradicionales.
4. **Plataformas como Java EE (Enterprise Edition):** Facilitó el desarrollo de aplicaciones empresariales web seguras y escalables.
5. **Android:** Java se convirtió en el lenguaje base para el desarrollo de aplicaciones móviles con Android.

¿Java es un lenguaje dependiente de la plataforma donde se ejecuta? ¿Por qué?

No, **Java es independiente de la plataforma**, gracias a su mecanismo de ejecución:

- **Código fuente en Java (`.java`)** se compila en **bytecode (`.class`)**, que no es código nativo de ningún sistema operativo.
- Este bytecode es interpretado y ejecutado por la **Máquina Virtual de Java (JVM)**, la cual es específica para cada sistema operativo.

Esto significa que un programa Java puede ejecutarse en **Windows, Linux, macOS, etc., sin cambios en el código fuente**, siempre que haya una JVM disponible en el sistema.

7_

¿Sobre qué lenguajes está basado Java?

Java se inspiró en varios lenguajes de programación, principalmente:

1. **C** → Java tomó su sintaxis básica (estructuras de control como `if`, `for`, `while`).
2. **C++** → Adoptó la programación orientada a objetos (POO), pero eliminó características complejas como herencia múltiple y manejo de punteros.
3. **Smalltalk** → Influenció el uso de objetos puros y la fuerte encapsulación.
4. **Objective-C** → Inspiró la gestión automática de memoria a través del recolector de basura (Garbage Collector).
5. **Modula-3** → Contribuyó con la seguridad en el manejo de excepciones y tipos de datos.

El diseño de Java priorizó la **simplicidad, seguridad y portabilidad**, eliminando características propensas a errores como punteros explícitos y sobrecarga de operadores.

8_

¿Qué son los Applets?

Los **Applets** eran pequeñas aplicaciones Java que se ejecutaban en un navegador web dentro de una página HTML.

🔷 **Características:**

- Se descargaban automáticamente desde un servidor y corrían en el cliente.
- Se ejecutaban dentro de un "sandbox" (entorno seguro) para evitar accesos no autorizados al sistema.
- Eran utilizados para agregar contenido interactivo a las páginas web (gráficos, juegos, animaciones).

🔷 **Ejemplo de código de un Applet en Java:**

```
java
CopiarEditar
```

```
import java.applet.Applet;
import java.awt.Graphics;

public class HolaApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("¡Hola, Applet!", 20, 20);
    }
}
```

📌 **¿Por qué dejaron de usarse?**

- Necesitaban un plugin de Java en el navegador.
- Eran lentos y tenían problemas de seguridad.
- Fueron reemplazados por tecnologías como **JavaScript, HTML5 y WebAssembly**.
- Java eliminó el soporte para applets en 2019 con la versión **Java 11**.

¿Qué son los Servlets?

Los **Servlets** son programas en Java que se ejecutan en un servidor web y generan contenido dinámico en respuesta a solicitudes de clientes (normalmente navegadores).

🔷 **Características:**

- Permiten manejar peticiones HTTP (`GET`, `POST`).
- Son más eficientes que los CGI (Common Gateway Interface).
- Se utilizan en aplicaciones web junto con **JSP (Java Server Pages)**.
- Se ejecutan en un **contenedor de servlets** como **Apache Tomcat**.

9_

Estructura de un programa en C

Un programa en **C** tiene una estructura bien definida que generalmente sigue este esquema:

```
c
CopiarEditar
#include <stdio.h>  // Librerías estándar

// Definición de macros y constantes
#define PI 3.1416

// Declaración de funciones
int suma(int a, int b);

// Función principal
int main() {
    int resultado = suma(5, 3);
    printf("La suma es: %d\n", resultado);
    return 0;
}

// Implementación de funciones
int suma(int a, int b) {
    return a + b;
}
```

Partes principales de un programa en C:

1. **Directivas del preprocesador** (`#include`, `#define`):
   - Se usan para incluir archivos de cabecera (`#include <stdio.h>`) y definir macros (`#define PI 3.1416`).
2. **Definición de constantes y macros:**
   - Se pueden definir valores constantes y macros que se expanden en el código antes de la compilación.
3. **Declaraciones globales:**
   - Variables y funciones pueden declararse globalmente para ser usadas en todo el programa.
4. **Función `main()` (punto de entrada):**
   - Es la función principal de cualquier programa en C.

- Debe devolver un valor entero (`int`), generalmente `return 0;` para indicar una ejecución exitosa.

5. **Definición de funciones:**
   - Se pueden declarar y definir funciones que dividen el código en partes más manejables.

¿Existe anidamiento de funciones en C?

No, **C no permite anidar funciones** (una función dentro de otra).

✅

Alternativa válida:

Para simular el anidamiento, se puede **declarar una función dentro de otra, pero fuera de su bloque.**

10_

Manejo de expresiones en C

En C, una **expresión** es una combinación de **valores, variables, operadores y llamadas a funciones** que produce un resultado. Pueden clasificarse en varios tipos:

---

1. Expresiones aritméticas

Usan operadores matemáticos para realizar cálculos.

```
c
CopiarEditar
int a = 10, b = 5;
int suma = a + b;      // 15
int resta = a - b;     // 5
int multiplicacion = a * b; // 50
int division = a / b;  // 2
int modulo = a % b;    // 0 (resto de la división)
```

📌 **Nota:** En divisiones enteras (`int`), los decimales se descartan.

---

2. Expresiones relacionales (comparación)

Evalúan relaciones entre valores y devuelven `1` (**true**) o `0` (**false**).

```
c
CopiarEditar
int x = 10, y = 5;
int resultado = (x > y);  // 1 (verdadero)
```

🔷 Operadores relacionales:

Operador	Significado
`==`	Igual a
`!=`	Diferente de
`>`	Mayor que
`<`	Menor que
`>=`	Mayor o igual que
`<=`	Menor o igual que

---

3. Expresiones lógicas

Permiten evaluar condiciones compuestas.

```
c
CopiarEditar
int edad = 20;
int es_adulto = (edad >= 18) && (edad < 65);  // 1 (true)
```

🔷 Operadores lógicos:

Operador	Significado
`&&`	AND (ambas condiciones deben ser verdaderas)
`		
`!`	NOT (niega el valor lógico)

---

4. Expresiones de asignación

Asignan valores a variables usando `=` y operadores compuestos.

```
c
CopiarEditar
int x = 10;
x += 5;  // Equivalente a: x = x + 5;
x *= 2;  // Equivalente a: x = x * 2;
```

5. Expresiones condicionales (Operador ternario `?:`)

Forma compacta de escribir `if-else`.

```
c
CopiarEditar
int edad = 20;
char *mensaje = (edad >= 18) ? "Mayor de edad" : "Menor de edad";
printf("%s\n", mensaje);
```

Si `edad >= 18`, asigna `"Mayor de edad"`, si no, `"Menor de edad"`.

6. Expresiones de incremento y decremento

```
c
CopiarEditar
int x = 5;
x++;  // x = 6 (post-incremento)
--x;  // x = 5 (pre-decremento)
```

7. Expresiones de punteros y acceso a memoria

```
c
CopiarEditar
int num = 10;
int *ptr = &num;  // Obtiene la dirección de memoria de num
int valor = *ptr; // Obtiene el valor almacenado en esa dirección
```

`&` → Operador de dirección

`*` → Operador de desreferenciación

11_

Python, Ruby y PHP: Usos, Paradigmas y Características

Lenguaje	Tipo de programas	Paradigma(s) de programación	Características del paradigma
Python	Aplicaciones web, ciencia de datos, inteligencia artificial, automatización, scripting, videojuegos.	Multiparadigma (imperativo, orientado a objetos, funcional)	- Código legible y limpio.- Tipado dinámico y fuerte.- Soporte para programación funcional con `map()`, `filter()`, `lambda`.- Orientado a objetos con clases y herencia.
Ruby	Aplicaciones web (Ruby on Rails), automatización, desarrollo rápido de prototipos.	Multiparadigma (orientado a objetos, funcional, imperativo)	- Todo es un objeto, incluso los números y funciones.- Sintaxis concisa y expresiva.- Metaprogramación avanzada.
PHP	Desarrollo web, sistemas de gestión de contenido (WordPress, Drupal), API backend.	Multiparadigma (imperativo, orientado a objetos, funcional)	- Diseñado para desarrollo web dinámico.- Soporte nativo para bases de datos.- Integración con HTML.- Gestión automática de memoria.

Explicación de los Paradigmas

Imperativo 🏛️ (Python, Ruby, PHP)

- Se ejecutan instrucciones en secuencia.
- Uso de variables, condicionales (`if-else`), bucles (`for`, `while`).
- Ejemplo en Python:

```
python
CopiarEditar
x = 10
if x > 5:
```

```
print("Mayor a 5")
```

Orientado a Objetos (OOP) 🎭 (Python, Ruby, PHP)

- Se organizan los programas en **clases y objetos**.
- Uso de **encapsulación, herencia y polimorfismo**.
- Ejemplo en Ruby:

```
ruby
CopiarEditar
class Persona
  attr_accessor :nombre
  def initialize(nombre)
    @nombre = nombre
  end
end
p = Persona.new("Ana")
puts p.nombre
```

Funcional 🔢 (Python, Ruby)

- Uso de funciones puras, inmutabilidad, `map()`, `reduce()`.
- Ejemplo en Python:

```
python
CopiarEditar
numeros = [1, 2, 3, 4]
cuadrados = list(map(lambda x: x**2, numeros))
print(cuadrados)  # [1, 4, 9, 16]
```

Conclusión

- **Python** es versátil y usado en ciencia de datos e inteligencia artificial.
- **Ruby** es elegante y usado en desarrollo web con Ruby on Rails.

- **PHP** es ideal para desarrollo web dinámico con integración de bases de datos.

12_

Características Importantes de Python, Ruby, PHP, Gobstone y Processing

Lenguaje	Características Importantes
Python	- **Tipado dinámico y fuerte:** No es necesario declarar el tipo de una variable. La comprobación de tipos se realiza en tiempo de ejecución. - **Lenguaje interpretado:** No necesita compilación, se ejecuta directamente en el intérprete. - **Orientación a objetos y funcional:** Admite programación orientada a objetos (OOP) y programación funcional. - **Gran biblioteca estándar:** Ofrece numerosas librerías para diferentes aplicaciones (ciencia de datos, web, automatización, etc.). - **Indentación como delimitador de bloques:** Usa espacios en blanco para definir la estructura de control (en lugar de llaves `{}` o `begin-end`).
Ruby	- **Tipado dinámico:** Similar a Python, el tipo de una variable se determina en tiempo de ejecución. - **Orientado a objetos puro:** Todo en Ruby es un objeto (incluso los números y las clases). - **Sintaxis expresiva y flexible:** Ruby tiene una sintaxis que permite escribir código de manera compacta y legible. - **Metaprogramación:** Permite modificar la estructura de clases y objetos en tiempo de ejecución. - **Recursos para desarrollo web:** Popular en el desarrollo de aplicaciones web mediante el framework Ruby on Rails.
PHP	- **Tipado débil y dinámico:** Las variables pueden cambiar de tipo en tiempo de ejecución. - **Orientado a objetos:** PHP ha incorporado características orientadas a objetos desde PHP 5. - **Llamadas a bases de datos:** Tiene soporte nativo para trabajar con bases de datos, especialmente con MySQL. - **Diseñado para desarrollo web:** Se integra fácilmente con HTML y se ejecuta en el servidor para generar páginas web dinámicas. - **Interactividad en el servidor:** PHP se ejecuta en el servidor, generando contenido dinámico según la solicitud del cliente.
Gobstone	- **Paradigma visual y lógico:** Gobstone es un lenguaje orientado a la lógica y a la programación estructurada. - **Tipado estático:** Los tipos de datos son definidos de manera explícita antes de la ejecución del programa. - **Tareas educativas:** Está diseñado para la enseñanza de programación a estudiantes. - **Sintaxis simple y visual:** Utiliza una

	sintaxis amigable y más intuitiva para facilitar el aprendizaje de conceptos básicos de programación. - **Orientado a tareas específicas:** Ideal para aprender algoritmos y estructuras de datos.
Processing	- **Lenguaje de programación visual:** Está diseñado para la creación de gráficos, animaciones y trabajos interactivos. - **Sintaxis simple:** Similar a Java pero con una interfaz más simple. - **Orientado a gráficos y multimedia:** Utilizado en arte, diseño y visualización de datos. - **Bibliotecas gráficas:** Ofrece bibliotecas muy poderosas para trabajar con gráficos 2D y 3D. - **Ideal para prototipos rápidos:** Permite crear visualizaciones y prototipos de manera eficiente.

13_

JavaScript: Paradigma y Tipo de Lenguaje

Paradigma de JavaScript:

JavaScript es un **lenguaje multiparadigma**, lo que significa que admite varios enfoques de programación. Los paradigmas principales en JavaScript son:

- **Imperativo:** En este paradigma, se especifican las instrucciones o pasos que el programa debe seguir para realizar una tarea. Se trabaja principalmente con sentencias que modifican el estado del programa.
  - **Ejemplo en JavaScript:**

    ```
    javascript
    CopiarEditar
    let x = 10;
    x = x + 5;  // Instrucción imperativa
    console.log(x); // 15
    ```

- **Orientado a Objetos (OOP):** JavaScript es un lenguaje orientado a objetos basado en prototipos. Aunque no utiliza clases de manera tradicional (hasta ES6), los objetos pueden ser creados a partir de otros objetos (prototipos), permitiendo la herencia.
  - **Ejemplo en JavaScript:**

```
javascript
CopiarEditar
let persona = {
  nombre: 'Juan',
  saludar: function() { console.log('Hola ' + this.nombre); }
};
persona.saludar();  // Saluda: Hola Juan
```

- **Funcional:** JavaScript soporta la programación funcional, lo que significa que se pueden usar funciones como objetos de primera clase. Esto implica que las funciones pueden ser pasadas como argumentos, retornadas desde otras funciones y asignadas a variables.
  - **Ejemplo en JavaScript:**

```
javascript
CopiarEditar
const suma = (a, b) => a + b;
console.log(suma(3, 4)); // 7
```

Tipo de Lenguaje:

- **Lenguaje interpretado:** JavaScript es un lenguaje **interpretado**, lo que significa que el código se ejecuta línea por línea sin necesidad de ser compilado previamente. Se ejecuta en tiempo real dentro del navegador o en un entorno de servidor (Node.js).
- **Lenguaje de alto nivel:** JavaScript es un lenguaje de **alto nivel** porque su sintaxis está diseñada para ser fácilmente entendida y manipulada por los humanos, abstrae detalles complejos de hardware y gestión de memoria.
- **Lenguaje de scripting:** JavaScript también es considerado un **lenguaje de scripting** porque está diseñado para realizar tareas automáticas, especialmente en la manipulación de elementos de páginas web en el navegador (por ejemplo, cambiar el contenido dinámico de una página web).

14_

Características Clave:

- **Tipado dinámico:** Las variables no requieren declaración de tipo.
- **Funciones de primera clase:** Las funciones pueden ser asignadas, pasadas y devueltas.
- **Orientado a objetos:** Usando objetos literales y clases ES6.
- **Asincronía:** Manejo de operaciones asíncronas mediante promesas, `async/await`.
- **Sintaxis moderna:** Soporte para nuevas características como clases, desestructuración y módulos.
-