

Practica 1 - CPLP

Objetivo: Conocer la evolución de los lenguajes de programación y sus características.

Ejercicio 1: Los lenguajes de programación más representativos son:

1951 - 1955: Lenguajes tipo assembly

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

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

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

1971 - 1975: Pascal, C, Scheme, Prolog

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

1981 - 1985: Smalltalk 80, Turbo Pascal, Postscript

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

1991 - 1995: TCL, PERL, HTML

1996 - 2000: Java, Javascript, XML

Indique para cada uno de los períodos presentados cuales son las características nuevas que se incorporan y cual de ellos la incorpora

- 1951 - 1955:
 - Programación en lenguaje de bajo nivel, cercano al hardware.
 - Uso de mnemónicos en lugar de código de máquina puro.
 - Mejora en la legibilidad respecto al código binario.
- 1956 - 1960:
 - **FORTRAN:** Primer lenguaje de alto nivel, optimizado para cálculos científicos y numéricos.
 - **ALGOL 58 / ALGOL 60:** Introducción de la estructura de bloques y recursividad.
 - **LISP:** Introducción de la programación funcional y manipulación de listas.
- 1961 - 1965:

- **COBOL**: Orientado a la gestión de datos y procesos empresariales, con sintaxis cercana al inglés.
- **SNOBOL**: Procesamiento avanzado de cadenas de caracteres.
- **JOVIAL**: Lenguaje para sistemas embebidos en aplicaciones militares y aeroespaciales.
- 1966 - 1970:
 - **APL**: Uso de notación matemática compacta y programación orientada a matrices.
 - **FORTRAN 66**: Primer estándar oficial del lenguaje.
 - **BASIC**: Introducción de la programación interactiva y accesible para principiantes.
 - **PL/I**: Integración de características de lenguajes científicos y de negocio.
 - **SIMULA 67**: Introducción de la programación orientada a objetos (clases, objetos y herencia).
- 1971 - 1975:
 - **Pascal**: Introducción de estructuras de datos y control más estructuradas.
 - **C**: Mayor control sobre la memoria y portabilidad.
 - **Scheme**: Enfoque funcional con soporte para closures y evaluación perezosa.
 - **Prolog**: Introducción de la programación lógica basada en reglas.
- 1976 - 1980:
 - **Smalltalk**: Expansión de la programación orientada a objetos con un entorno interactivo.
 - **Ada**: Seguridad en tipos, modularidad y concurrencia.
 - **FORTRAN 77**: Mejoras en estructuras de control y manejo de cadenas.
 - **ML**: Introducción de la inferencia de tipos en programación funcional.
- 1981 - 1985:

- **Smalltalk 80**: Popularización de la programación orientada a objetos con un entorno gráfico.
- **Turbo Pascal**: Rápida compilación e integración de herramientas de desarrollo.
- **Postscript**: Lenguaje interpretado para la generación de gráficos e impresión.
- 1986 - 1990:
 - **FORTRAN 90**: Soporte para programación modular y arrays dinámicos.
 - **C++**: Expansión de C con orientación a objetos (clases, herencia y polimorfismo).
 - **SML**: Refinamiento de la inferencia de tipos en lenguajes funcionales.
- 1991 - 1995:
 - **TCL**: Lenguaje de scripting embebido para automatización.
 - **PERL**: Manejo avanzado de cadenas y expresiones regulares.
 - **HTML**: Definición de la estructura de documentos web.
- 1996 - 2000:
 - **Java**: Independencia de plataforma con la máquina virtual JVM.
 - **Javascript**: Introducción de la programación dinámica en el navegador.
 - **XML**: Estandarización para el intercambio de datos estructurados.

Ejercicio 2: Escriba brevemente la historia del lenguaje de programación que eligió en la encuesta u otro de su preferencia

Python es un lenguaje de programación de alto nivel, interpretado y con una sintaxis clara y legible. Fue creado por **Guido van Rossum** en el **Centro para las Matemáticas y la Informática (CWI) en los Países Bajos** a finales de los años 80 y lanzado en **1991**. Su objetivo era diseñar un lenguaje que combinara la **simplicidad y legibilidad** con una gran capacidad de programación.

Principales hitos en su evolución:

- **1991:** Se lanza Python 1.0, con características como manejo de excepciones, funciones y módulos.
- **2000:** Se lanza Python 2.0, agregando recolección de basura y listas por comprensión, aunque eventualmente sería reemplazado.
- **2008:** Se publica Python 3.0, una versión incompatible con Python 2, mejorando la gestión de Unicode y la sintaxis general.
- **2010-2020:** Python se populariza en ciencia de datos, inteligencia artificial y desarrollo web con frameworks como Django y Flask.
- **2020:** Se declara el fin del soporte para Python 2, consolidando Python 3 como el estándar.

Hoy en día, Python es uno de los lenguajes más utilizados en el mundo, gracias a su facilidad de aprendizaje y su amplia aplicación en áreas como **desarrollo web, automatización, análisis de datos, aprendizaje automático y ciberseguridad**.

Ejercicio 3: ¿Qué atributos debería tener un buen lenguaje de programación?

Por ejemplo,

ortogonalidad, expresividad, legibilidad, simplicidad, etc. De al menos un ejemplo de un lenguaje que cumple con las características citadas.

1. **Ortogonalidad:** Un pequeño conjunto de reglas y combinaciones permite expresar una gran cantidad de funcionalidades sin excepciones innecesarias.
 - **Ejemplo: LISP**, donde unas pocas estructuras básicas permiten definir programas complejos.
2. **Expresividad:** Permite escribir código claro y conciso, con menos esfuerzo y sin ambigüedad.
 - **Ejemplo: Python**, con su sintaxis intuitiva y legible.
3. **Legibilidad:** Un código fácil de leer reduce la posibilidad de errores y facilita su mantenimiento.
 - **Ejemplo: Python**, que fomenta la indentación y evita el uso excesivo de símbolos.

4. **Simplicidad:** Evita una sintaxis excesivamente compleja o redundante, facilitando el aprendizaje y la escritura de código.
 - **Ejemplo: BASIC**, diseñado para ser fácil de entender por principiantes.
5. **Portabilidad:** Permite que el mismo código pueda ejecutarse en diferentes plataformas sin modificaciones.
 - **Ejemplo: Java**, gracias a la Máquina Virtual de Java (JVM).
6. **Eficiencia:** Un buen lenguaje debe permitir escribir programas que utilicen recursos computacionales de manera óptima.
 - **Ejemplo: C**, ampliamente usado en sistemas operativos por su alto rendimiento.
7. **Seguridad:** Previene errores comunes y vulnerabilidades mediante restricciones en el uso de memoria y estructuras de datos.
 - **Ejemplo: Rust**, que evita errores de acceso a memoria sin necesidad de un recolector de basura.
8. **Modularidad:** Debe permitir la reutilización y organización del código mediante funciones, clases o módulos.
 - **Ejemplo: Python**, con su sistema de módulos y paquetes.

Ejemplo de un lenguaje que cumple con varias características

Python es un lenguaje que destaca por su **legibilidad, expresividad, simplicidad, modularidad y portabilidad**, lo que lo hace ideal tanto para principiantes como para expertos en diversas áreas de la programación.

Ejercicio 4: Tome uno o dos lenguajes de los que ud. Conozca y

- Describa los tipos de expresiones que se pueden escribir en él/ellos
- Describa las facilidades provistas para la organización del programa
- Indique cuáles de los atributos del ejercicio anterior posee el/los lenguaje/s elegidos y cuáles no posee, justifique en cada caso.

Python

1. Tipos de expresiones en Python

Python permite diferentes tipos de expresiones, entre ellas:

- **Expresiones aritméticas:** `a + b` , `x * y - 5` , `3 ** 2`
- **Expresiones booleanas:** `x > 5` , `a == b and c != d`
- **Expresiones de cadenas:** `"Hola" + " mundo"` , `nombre.upper()`
- **Expresiones de listas y diccionarios:** `[x for x in range(10)]` , `{clave: valor for clave, valor in datos}`
- **Expresiones lambda:** `lambda x: x * 2`

2. Facilidades para la organización del programa

Python permite organizar el código mediante:

- **Módulos y paquetes** (`import mi_modulo`)
- **Funciones y clases** (`def funcion():` , `class MiClase:`)
- **Espacios de nombres y ámbitos** (global, local, no local)
- **Programación orientada a objetos y programación funcional**

Java

1. Tipos de expresiones en Java

- **Expresiones aritméticas:** `a + b` , `x * y - 5` , `Math.pow(3, 2)`
- **Expresiones booleanas:** `x > 5` , `a == b && c != d`
- **Expresiones de objetos:** `new Persona("Juan")`
- **Expresiones lambda:** `(x) → x * 2`

2. Facilidades para la organización del programa

Java proporciona:

- **Clases y objetos** (programación orientada a objetos obligatoria)
- **Paquetes y módulos** (`package mi.paquete;`)
- **Interfaces y herencia** (`implements` , `extends`)
- **Manejo estricto de tipos**

Atributo	Python	Java
Ortogonalidad	✓	✓
Expresividad	✓	✓
Legibilidad	✓	✓
Simplicidad	✓	✗
Portabilidad	✓	✓
Eficiencia	✗	✓
Seguridad	✗	✓
Modularidad	✓	✗

Lenguajes - ADA

Ejercicio 5: Describa las características más relevantes de Ada, referida a:

- Tipos de datos
- Tipos abstractos de datos – paquetes
- Estructuras de datos
- Manejo de excepciones
- Manejo de concurrencia

1. Tipos de datos en Ada

Ada es un lenguaje **fuertemente tipado**, lo que significa que **no permite conversiones implícitas entre tipos incompatibles**. Algunos de sus tipos de datos son:

- **Enteros:** `Integer`, `Natural`, `Positive`
- **Reales:** `Float`, `Fixed` (precisión fija)
- **Caracteres y cadenas:** `Character`, `String`
- **Booleanos:** `Boolean` (`True` o `False`)
- **Enumerados:** Se pueden definir tipos con valores específicos, por ejemplo:

```
type Dias_Semana is (Lunes, Martes, Miercoles, Jueves, Viernes);
```

- **Tipos derivados y subrangos:** Permiten definir restricciones adicionales:

```
subtype Pequeño is Integer range 1..10;
```

- **Registros:** Similares a estructuras en C o clases en otros lenguajes.

2. Tipos abstractos de datos y paquetes

Ada permite la encapsulación de datos mediante **paquetes (packages)**, los cuales funcionan como módulos reutilizables.

Ejemplo de un **paquete** que define un tipo abstracto:

```
package Mi_Paquete is
  type Contador is private;
  procedure Incrementar (C: in out Contador);
  function Obtener_Valor (C: Contador) return Integer;
private
  type Contador is record
    Valor: Integer := 0;
  end record;
end Mi_Paquete;
```

Los paquetes **permiten ocultar la implementación interna** y exponer solo las operaciones necesarias.

3. Estructuras de datos

Ada soporta diversas estructuras de datos, tales como:

- **Registros** (similares a structs en C):

```
type Persona is record
  Nombre: String(1..20);
  Edad: Integer;
end record;
```

- **Arreglos:**

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


- **Listas enlazadas** (a través de tipos de acceso, equivalentes a punteros en C):

```
type Nodo;  
type Nodo_Access is access Nodo;  
type Nodo is record  
  Valor: Integer;  
  Siguiente: Nodo_Access;  
end record;
```

4. Manejo de excepciones

Ada tiene un sistema de manejo de excepciones robusto, lo que permite capturar errores en tiempo de ejecución de manera estructurada.

Ejemplo de uso de excepciones:

```
begin  
  -- Código que puede generar una excepción  
  X := 10 / 0; -- División por cero  
exception  
  when Constraint_Error =>  
    Put_Line("Error: División por cero detectada.");  
  when others =>  
    Put_Line("Error desconocido.");  
end;
```

Las excepciones permiten mejorar la seguridad del código evitando fallos críticos.

5. Manejo de concurrencia

Ada fue diseñado con **soporte nativo para concurrencia**, a través de **tasks** (tareas) y **protected objects** (objetos protegidos).

Ejemplo de una tarea concurrente:

```
ask Tipo_Tarea is  
  entry Iniciar;
```

```
end Tipo_Tarea;

task body Tipo_Tarea is
begin
    accept Iniciar;
    Put_Line("Tarea ejecutándose...");
end Tipo_Tarea;
```

- **Las tareas** pueden ejecutarse en paralelo.
- **Los objetos protegidos** permiten sincronización segura entre hilos de ejecución.

Lenguajes - JAVA

Ejercicio 6

: Diga para qué fue, básicamente, creado Java. ¿Qué cambios le introdujo a la Web?

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

1. ¿Para qué fue creado Java?

Java fue creado en **1995** por **James Gosling** y su equipo en **Sun Microsystems**. Originalmente, fue diseñado para dispositivos electrónicos y sistemas embebidos, pero rápidamente evolucionó hacia el desarrollo de **software multiplataforma, aplicaciones empresariales y desarrollo web**. Su principal objetivo era permitir el desarrollo de software **portátil, seguro y eficiente**.

2. ¿Qué cambios introdujo Java en la Web?

Java revolucionó la Web al introducir la **programación dinámica y la ejecución de código en el lado del cliente y servidor**. Algunos de los cambios más importantes incluyen:

- **Applets de Java:** Permitieron ejecutar programas en navegadores web sin necesidad de instalación. Aunque hoy en día han caído en desuso, fueron revolucionarios en su momento.
- **Java Servlets y JSP:** Permitieron el desarrollo de aplicaciones web dinámicas y robustas en el lado del servidor.

- **JEE (Java Enterprise Edition):** Facilitó el desarrollo de aplicaciones empresariales escalables y seguras.
- **Frameworks como Spring, Hibernate y Struts:** Mejoraron la productividad en el desarrollo web y empresarial.

3. ¿Java es dependiente de la plataforma en donde se ejecuta? ¿Por qué?

No, **Java es un lenguaje independiente de la plataforma** gracias a su arquitectura basada en la **Java Virtual Machine (JVM)**.

Esto se debe a su principio de "**Write Once, Run Anywhere**" (**WORA**), que significa que el código fuente de Java se compila a un **bytecode intermedio**, el cual puede ejecutarse en cualquier dispositivo que tenga una JVM, sin importar el sistema operativo o el hardware.

Sin embargo, la **JVM sí es dependiente de la plataforma**, ya que hay versiones específicas de la JVM para cada sistema operativo.

Ejercicio 7: ¿Sobre qué lenguajes está basado?

Java tomó inspiración de varios lenguajes de programación previos, incorporando sus mejores características mientras eliminaba aspectos problemáticos. Los principales lenguajes en los que se basó son:

1. **C** → Java heredó su sintaxis básica, estructuras de control (`if`, `for`, `while`), operadores y muchas convenciones de programación.
2. **C++** → Java adoptó el modelo de programación orientada a objetos (POO) de C++, pero eliminó características complejas como punteros y herencia múltiple, para mejorar la seguridad y simplicidad.
3. **Smalltalk** → Influenció el enfoque **totalmente orientado a objetos** de Java y su concepto de gestión automática de memoria mediante el **garbage collector**.
4. **Objective-C** → Inspiró la idea de manejar excepciones y el uso de mensajes entre objetos.
5. **Ada** → Aportó ideas sobre seguridad en la programación, modularidad y tipado fuerte.

Ejercicio 8: ¿Qué son los applets? ¿Qué son los servlets?

1. ¿Qué son los Applets?

Los **Applets** eran programas escritos en Java que se ejecutaban dentro de un navegador web. Eran pequeñas aplicaciones embebidas en páginas HTML, capaces de realizar operaciones interactivas y dinámicas.

Características de los Applets:

- Se ejecutaban en el navegador a través de la **Java Virtual Machine (JVM)**.
- Usaban el paquete `java.applet` y la clase base `Applet`.
- Tenían restricciones de seguridad estrictas (sandbox) para evitar accesos no autorizados al sistema del usuario.
- Fueron populares en los 90s y principios de los 2000s, pero fueron reemplazados por tecnologías más modernas como **JavaScript, HTML5 y WebAssembly**.
- Actualmente, los navegadores ya **no soportan Applets** debido a problemas de seguridad y rendimiento.

2. ¿Qué son los Servlets?

Los **Servlets** son programas en Java que **se ejecutan en el servidor** y generan contenido dinámico para páginas web. Son la base de muchas aplicaciones web basadas en Java.

Características de los Servlets:

- Se ejecutan en un **servidor web o de aplicaciones** (por ejemplo, **Apache Tomcat, WildFly, GlassFish**).
- Usan el paquete `javax.servlet` y la clase base `HttpServlet`.
- Son una alternativa eficiente a los **CGI Scripts** (Common Gateway Interface).
- Pueden manejar solicitudes HTTP (`GET`, `POST`, etc.) y generar respuestas dinámicas en HTML o JSON.
- Son la base de tecnologías como **JavaServer Pages (JSP)**, **JavaServer Faces (JSF)** y frameworks como **Spring MVC**.

Diferencias entre Applets y Servlets

Característica	Applets	Servlets
Ubicación de ejecución	Navegador web (cliente)	Servidor web
Dependencia del navegador	Sí, requiere JVM en el navegador	No, solo el servidor necesita JVM
Interacción con el usuario	Interfaz gráfica en la página web	Genera contenido dinámico (HTML, JSON, etc.)
Seguridad	Ejecutado en una sandbox con restricciones	Seguridad manejada por el servidor
Estado actual	Obsoleto y discontinuado	Sigue siendo ampliamente usado en aplicaciones web

Lenguajes - C

Ejercicio 9

: ¿Cómo es la estructura de un programa escrito en C? ¿Existe anidamiento de funciones?

1. Estructura de un programa en C

Un programa en C sigue una estructura bien definida que generalmente incluye las siguientes secciones:

```
#include <stdio.h> // 1. Directivas de preprocesador

// 2. Definiciones de macros y constantes
#define PI 3.1416

// 3. Declaración de funciones
void saludar();

// 4. Función principal (punto de entrada del programa)
int main() {
    printf("¡Hola, mundo!\n"); // 5. Cuerpo del programa
    saludar();
    return 0;
}

// 6. Definiciones de funciones
```

```
void saludar() {  
    printf("Bienvenido a la programación en C.\n");  
}
```


Explicación de las partes del programa:

1. **Directivas de preprocesador (`#include`)**: Importan bibliotecas necesarias para el programa, como `stdio.h` para entrada y salida estándar.
2. **Definiciones de macros y constantes (`#define` , `const`)**: Se pueden definir valores constantes.
3. **Declaración de funciones**: Se especifican las funciones que se utilizarán más adelante.
4. **Función `main()`**: Es el punto de entrada del programa. Siempre debe retornar un `int`.
5. **Cuerpo del programa**: Contiene instrucciones que ejecuta el programa.
6. **Definiciones de funciones**: Implementación de las funciones declaradas anteriormente.

2. ¿Existe anidamiento de funciones en C?

No, **C no permite el anidamiento de funciones**, es decir, **no se pueden definir funciones dentro de otras funciones**.

Ejemplo inválido en C (esto no compila):

```
int main() {  
    void funcionAnidada() { //  Error: No se permiten funciones dentro de  
    funciones en C  
        printf("Esto no es válido en C.");  
    }  
    funcionAnidada();  
    return 0;  
}
```

Alternativa válida en C:

Si se necesita usar una función dentro de otra, la solución es **declararla fuera de `main()`** y llamarla desde la función principal.

```
#include <stdio.h>

void funcionSeparada() {
    printf("Esto sí es válido en C.\n");
}

int main() {
    funcionSeparada();
    return 0;
}
```

Ejercicio 10: Describa el manejo de expresiones que brinda el lenguaje.


1. Tipos de expresiones en C

Expresiones aritméticas

Son operaciones matemáticas que usan operadores aritméticos.

Ejemplo:

```
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:** La división entre enteros trunca los decimales (`10 / 4` da `2` y no `2.5`).

Expresiones relacionales

Se usan en comparaciones y devuelven un valor booleano (`0` o `1`).

Ejemplo:

```
int x = 10, y = 5;
int resultado = (x > y); // 1 (true)
```

Operadores relacionales:

- `>` (mayor que), `<` (menor que)
- `>=` (mayor o igual que), `<=` (menor o igual que)
- `==` (igual a), `!=` (distinto de)

Expresiones lógicas

Se usan para combinar condiciones en estructuras de control (`if`, `while`, `for`).

Ejemplo:

```
int a = 5, b = 10, c = 15;
if (a < b && b < c) {
    printf("La condición es verdadera\n");
}
```

Operadores lógicos:

- `&&` (AND) → `true` si ambas condiciones son `true`
- `||` (OR) → `true` si al menos una condición es `true`
- `!` (NOT) → Invierte el valor de la condición

Expresiones de asignación

Asignan valores a variables usando `=` y combinaciones con operadores (`+=`, `-`, `=`, `*=`, etc.).

Ejemplo:

```
int x = 10;
x += 5; // x ahora es 15 (equivalente a x = x + 5)
x *= 2; // x ahora es 30 (equivalente a x = x * 2)
```

Expresiones condicionales (Operador ternario)

Permite evaluar una condición en una sola línea.

Ejemplo:


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

Sintaxis:

```
(condición) ? expresión_si_verdadero : expresión_si_falso;
```


Expresiones con punteros

Manejo de memoria mediante punteros.

Ejemplo:

```
int x = 10;
int *ptr = &x; // ptr almacena la dirección de x
printf("Valor de x: %d\n", *ptr); // Accede al valor de x
```

Operadores:




-  (direccionamiento) → Obtiene la dirección de memoria
- (desreferenciación) → Accede al valor almacenado en la dirección

2. Orden de evaluación y precedencia de operadores

C sigue un **orden de evaluación** según la **precedencia de operadores**.



Ejemplo con prioridad de operadores:

```
int resultado = 10 + 5 * 2; // ¿25 o 30?
```

 La multiplicación () tiene mayor precedencia que la suma () , por lo que se evalúa primero:

```
int resultado = 10 + (5 * 2); // 10 + 10 = 20
```

Precedencia de operadores en C (de mayor a menor prioridad):

1.  (paréntesis)
2.  (multiplicación, división, módulo)

3. `+-` (suma y resta)
4. `<<>>` (desplazamiento de bits)
5. `<<= >=>` (relacionales)
6. `== !=` (igualdad)
7. `&` (AND bit a bit)
8. `^` (XOR bit a bit)
9. `|` (OR bit a bit)
10. `&&` (AND lógico)
11. `||` (OR lógico)
12. `?:` (operador ternario)
13. `= += -= *= /=` (asignación)

Lenguajes - Python - RUBY - PHP

Ejercicio 11:

¿Qué tipo de programas se pueden escribir con cada uno de estos lenguajes?

¿A qué

paradigma responde cada uno? ¿Qué características determinan la pertenencia a cada paradigma?

1 Python

📌 Tipos de programas que se pueden escribir:

- ✓ Aplicaciones de escritorio
- ✓ Desarrollo web (Django, Flask)
- ✓ Ciencia de datos e inteligencia artificial (NumPy, Pandas, TensorFlow)
- ✓ Scripts de automatización y administración de sistemas
- ✓ Videojuegos (Pygame)
- ✓ Aplicaciones embebidas (MicroPython en Raspberry Pi)

📌 Paradigmas:

✓ **Programación imperativa:** Uso de instrucciones secuenciales para modificar el estado del programa.

✓ **Programación orientada a objetos:** Uso de clases, objetos, encapsulación, herencia y polimorfismo.

✓ **Programación funcional:** Uso de funciones de orden superior (`map()` , `filter()` , `reduce()`), inmutabilidad y expresiones lambda.

📌 **Características que determinan estos paradigmas en Python:**

- **Imperativo:** Uso de variables y estructuras de control (`if` , `for` , `while`).
- **Orientado a objetos:** Soporta clases y objetos (`class` , `self`).
- **Funcional:** Funciones como ciudadanos de primera clase, permite recursión y programación declarativa.

2 Ruby

📌 **Tipos de programas que se pueden escribir:**

- ✓ Aplicaciones web (Ruby on Rails)
- ✓ Aplicaciones de servidor y API REST
- ✓ Scripts de automatización
- ✓ Desarrollo de juegos (Gosu)

📌 **Paradigmas:**

✓ **Orientado a objetos** (Todo en Ruby es un objeto, incluyendo números y funciones).

✓ **Funcional** (Soporta lambdas, closures, bloques y funciones de orden superior).

✓ **Metaprogramación** (Capacidad de modificar la estructura del código en tiempo de ejecución).

📌 **Características que determinan estos paradigmas en Ruby:**

- **Orientado a objetos:** Todo es un objeto, incluso los tipos primitivos.
- **Funcional:** Permite el uso de bloques (`each` , `map`), lambdas y `Proc` .
- **Metaprogramación:** Puede redefinir clases y métodos en tiempo de ejecución.

3 PHP

📌 **Tipos de programas que se pueden escribir:**

✓ Aplicaciones web dinámicas (WordPress, Laravel, Symfony)

✓ Aplicaciones de backend y API REST

✓ Gestión de bases de datos con MySQL y PostgreSQL

✓ CMS y e-commerce

📌 Paradigmas:

✓ **Programación imperativa** (Uso de variables, estructuras de control y bucles).

✓ **Orientado a objetos** (Clases, objetos, herencia, polimorfismo).

✓ **Funcional (limitado)** (Funciones de orden superior como `array_map()`, `array_filter()`).

📌 Características que determinan estos paradigmas en PHP:

- **Imperativo:** Código secuencial con asignaciones y control de flujo.
- **Orientado a objetos:** Uso de clases y objetos (`class` , `extends`).
- **Funcional:** Funciones anónimas (`function() use`), `array_map()` y `array_filter()` .

Ejercicio 12: Cite otras características importantes de Python, Ruby, PHP, Golang y Processing.

Por ejemplo: tipado de datos, cómo se organizan los programas, etc.

1 Python

📌 Tipado de datos:

✓ **Dinamicamente tipado** (No se necesita especificar el tipo de variable).

✓ **Fuertemente tipado** (No convierte automáticamente entre tipos incompatibles).

✓ Soporta tipos primitivos (`int` , `float` , `str` , `bool`) y estructuras complejas (`list` , `dict` , `set` , `tuple`).

📌 Organización de programas:

- Basado en **módulos y paquetes** (`import module_name`).
- Se pueden definir funciones (`def nombre_funcion()`) y clases (`class NombreClase`).
- Entrada principal con `if __name__ == "__main__":` .

Otras características:

- ✓ Lenguaje **interpretado** y multiplataforma.
 - ✓ Soporte nativo para **programación funcional y orientada a objetos**.
 - ✓ Gran cantidad de **librerías estándar** y externas (NumPy, Django, Flask).
-

Ruby

Tipado de datos:

- ✓ **Dinámicamente tipado** (Las variables no requieren declaración de tipo).
- ✓ **Fuertemente tipado** (No convierte implícitamente entre tipos incompatibles).
- ✓ Todo en Ruby es un **objeto**, incluidos números y funciones.

Organización de programas:

- Código basado en **clases y módulos** (`module NombreModulo`).
- Métodos definidos con `def nombre_metodo` .
- Soporte para **bloques, lambdas y metaprogramación**.

Otras características:

- ✓ Lenguaje **interpretado** con sintaxis sencilla y expresiva.
 - ✓ Soporta **metaprogramación** (definir clases y métodos en tiempo de ejecución).
 - ✓ Ampliamente utilizado en desarrollo web con **Ruby on Rails**.
-

PHP

Tipado de datos:

- ✓ **Dinamicamente tipado**, pero con soporte para **tipado estricto opcional** (`declare(strict_types=1);`).
- ✓ Soporta tipos primitivos (`int` , `float` , `bool` , `string` , `array`) y compuestos (`object` , `callable`).

Organización de programas:

- Código estructurado en **archivos** `.php` que pueden incluir HTML y scripts embebidos.

- Se pueden definir **clases y funciones**.
- Manejo de dependencias con **Composer**.

📌 Otras características:

- ✓ Ampliamente usado en desarrollo **backend y web**.
 - ✓ Soporte para **bases de datos** (MySQL, PostgreSQL).
 - ✓ Interpretado en **servidores web** (Apache, Nginx).
-

4 Gobstone

📌 Tipado de datos:

- ✓ Lenguaje con un conjunto limitado de tipos (`Color` , `Dirección` , `Bool`).
- ✓ No tiene variables tradicionales, solo una memoria de estado.

📌 Organización de programas:

- Programación **imperativa** con comandos secuenciales.
- Estructura basada en **procedimientos** (`procedure nombre { ... }`).
- No tiene orientación a objetos ni funciones de alto nivel.

📌 Otras características:

- ✓ Diseñado para enseñanza de **pensamiento computacional**.
 - ✓ Ejecuta instrucciones en un **tablero con robots y colores**.
 - ✓ Basado en **acciones simples como mover, poner y sacar fichas**.
-

5 Processing

📌 Tipado de datos:

- ✓ **Estáticamente tipado** (Las variables deben declarar su tipo: `int` , `float` , `boolean`).
- ✓ Soporta tipos gráficos como `PImage` , `PVector` .

📌 Organización de programas:

- Código basado en **funciones principales**:
 - `setup()` (configuración inicial).
 - `draw()` (ejecución continua).

- Utiliza **clases y objetos** para estructurar programas.

Otras características:

- ✓ **Orientado a gráficos y multimedia** (usado para animaciones e interactividad).
- ✓ Basado en **Java**, pero con una sintaxis simplificada.
- ✓ Usado en **arte digital, visualización de datos y educación**.

Lenguaje Javascript

Ejercicio 13

: ¿A qué tipo de paradigma corresponde este lenguajes? ¿A qué tipo de Lenguaje pertenece?

¿A qué tipo de paradigma corresponde?

JavaScript es un lenguaje **multiparadigma**, ya que permite programar de varias formas:

1. **Programación Imperativa**: Uso de estructuras de control como bucles y condicionales.
2. **Programación Orientada a Objetos (POO)**: Basada en **prototipos** en lugar de clases tradicionales. Desde **ES6**, permite definir clases (`class`).
3. **Programación Funcional**: Uso de funciones de orden superior (`map` , `filter` , `reduce`), funciones anónimas y callbacks.
4. **Programación Asíncrona**: Uso de **promesas** (`Promise`) y **async/await** para manejar operaciones no bloqueantes.

¿A qué tipo de lenguaje pertenece?

- ✓ **Lenguaje Interpretado**: No necesita compilación previa; se ejecuta en un **motor de JavaScript** (como V8 en Chrome).
- ✓ **Lenguaje de Alto Nivel**: Fácil de leer y escribir, con abstracciones para manejar memoria y ejecución.
- ✓ **Lenguaje Débilmente Tipado y Dinámico**: Las variables no tienen un tipo fijo y pueden cambiar en tiempo de ejecución (`var` , `let` , `const`).
- ✓ **Lenguaje de Scripting**: Originalmente diseñado para **interactividad en la web**, pero con Node.js también se usa en **backend**.

Ejercicio 14: Cite otras características importantes de javascript. Tipado de datos, excepciones, variables, etc

1. Tipado de datos

✓ **Débilmente Tipado:** No es necesario declarar el tipo de dato de una variable.

✓ **Dinámicamente Tipado:** Una variable puede cambiar de tipo en tiempo de ejecución.

✓ Soporta **tipos primitivos:**

- **Number** (Enteros y flotantes) → `let num = 10;`
- **String** (Cadenas de texto) → `let text = "Hola";`
- **Boolean** (Valores lógicos) → `let isActive = true;`
- **Undefined** (Sin valor asignado) → `let x;`
- **Null** (Valor intencionalmente vacío) → `let y = null;`
- **Symbol** (Identificadores únicos) → `let sym = Symbol("id");`
- **BigInt** (Números grandes) → `let bigNumber = 123n;`

✓ Soporta **tipos compuestos:**

- **Object** (Colección de datos clave-valor) → `let obj = { name: "Fabio", age: 25 };`
- **Array** (Listas ordenadas) → `let arr = [1, 2, 3];`
- **Function** (Bloques de código reutilizables) → `function suma(a, b) { return a + b; }`

2. Manejo de excepciones

JavaScript usa `try...catch` para capturar y manejar errores en tiempo de ejecución.

```
try {  
  let result = x / 0; // Error si x no está definida  
} catch (error) {  
  console.error("Ocurrió un error:", error.message);  
}
```



```
} finally {  
  console.log("Este bloque se ejecuta siempre.");  
}
```

3. Declaración de variables

JavaScript permite declarar variables con tres palabras clave principales:

- ◆ `var` → **Ámbito de función, redeclarable** (evitar su uso en ES6 en adelante).
- ◆ `let` → **Ámbito de bloque, reasignable** (buena práctica para variables cambiantes).
- ◆ `const` → **Ámbito de bloque, inmutable** (para valores constantes).

```
var globalVar = "Soy global";  
let localVar = "Solo dentro del bloque";  
const constantVar = "No cambio";
```

4. Funciones y programación funcional

✓ JavaScript permite **funciones de primera clase**, lo que significa que las funciones pueden ser asignadas a variables, pasadas como argumentos y devueltas como valores.

◆ Funciones tradicionales:

```
function saludar(nombre) {  
  return "Hola " + nombre;  
}
```

◆ Funciones flecha (ES6+):

```
const saludar = (nombre) => `Hola ${nombre}`;
```

5. Programación orientada a objetos (POO) basada en prototipos

JavaScript implementa la POO usando **prototipos** en lugar de clases tradicionales, aunque desde ES6 se introdujo `class` como una forma más familiar de definir objetos.

```
class Persona {  
  constructor(nombre, edad) {  
    this.nombre = nombre;  
    this.edad = edad;  
  }  
  
  saludar() {  
    return `Hola, soy ${this.nombre}`;  
  }  
}  
  
const persona1 = new Persona("Fabio", 25);  
console.log(persona1.saludar()); // "Hola, soy Fabio"
```

6. Asincronía y manejo de promesas

JavaScript permite manejar código asíncrono mediante **callbacks, promesas y `async/await`**.

◆ Promesas:

```
let promesa = new Promise((resolve, reject) => {  
  setTimeout(() => resolve("Éxito!"), 2000);  
});  
  
promesa.then((mensaje) => console.log(mensaje)); // "Éxito!" después de 2 segundos
```

◆ Async/Await:

```
async function obtenerDatos() {  
  let respuesta = await fetch("https://jsonplaceholder.typicode.com/todos/1");  
  let datos = await respuesta.json();  
  console.log(datos);  
}
```

```
}  
obtenerDatos();
```

7. JSON (JavaScript Object Notation)

JavaScript usa JSON como formato ligero para el intercambio de datos.

```
let usuario = { nombre: "Fabio", edad: 25 };  
let jsonStr = JSON.stringify(usuario); // Convertir objeto a JSON  
let jsonObj = JSON.parse(jsonStr); // Convertir JSON a objeto
```

8. Manipulación del DOM (Document Object Model)

JavaScript permite modificar el contenido y estilo de una página web dinámicamente.

```
document.getElementById("titulo").innerText = "Nuevo Título";  
document.querySelector("button").addEventListener("click", () => alert("Clic!"));
```