Práctica 1

Ejercicio 1

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: Lenguajes tipo assembly

- Programación de bajo nivel, control directo del hardware, uso de instrucciones mnemotécnicas en lugar de código binario.
- **Lenguajes representativos**: Ensambladores específicos para diferentes arquitecturas.

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

- FORTRAN: Introducción de estructuras de control como bucles y condicionales, optimización para cálculos científicos.
- ALGOL 58 / ALGOL 60: Primera definición formal de sintaxis, bloques anidados, estructuras de control bien definidas.
- LISP: Primer lenguaje funcional, introducción del procesamiento simbólico y listas enlazadas.

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

- COBOL: Enfoque en aplicaciones comerciales, introducción de estructuras de datos avanzadas y legibilidad en inglés.
- SNOBOL: Manejo avanzado de cadenas de caracteres y expresiones regulares.
- JOVIAL: Uso en sistemas embebidos y de defensa.

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

- APL: Notación matemática concisa y operaciones en arrays.
- BASIC: Introducción de un lenguaje de programación sencillo para aprendizaje.
- SIMULA 67: Introducción del paradigma de programación orientada a objetos.
- PL/I: Lenguaje multipropósito con características de procesamiento de datos científicas y comerciales.

1971 - 1975: Pascal, C, Scheme, Prolog

- Pascal: Programación estructurada con tipado fuerte.
- C: Acceso de bajo nivel con punteros y manipulación de memoria, pero con estructuras de alto nivel.
- Scheme: Introducción del concepto de funciones de orden superior y evaluación diferida.
- Prolog: Introducción de la programación lógica y basada en reglas.

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

- Smalltalk: Expansión de la orientación a objetos con un sistema completamente basado en objetos.
- Ada: Seguridad en la concurrencia y tipado estricto.
- ML: Inferencia de tipos y programación funcional con expresividad matemática.

1981 - 1985: Smalltalk 80, Turbo Pascal, Postscript

- Smalltalk 80: Desarrollo de entornos gráficos interactivos con orientación a objetos.
- Turbo Pascal: Introducción de compiladores rápidos y herramientas para entornos educativos.
- PostScript: Lenguaje para la descripción de páginas y gráficos vectoriales.

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

- FORTRAN 90: Introducción de programación modular y manejo de arrays dinámicos.
- C++: Expansión de C con orientación a objetos.
- SML: Tipado estático fuerte con inferencia de tipos.

1991 - 1995: TCL, PERL, HTML

- TCL: Lenguaje de scripting embebido.
- PERL: Manejo avanzado de texto y expresiones regulares, scripting flexible.
- HTML: Base para el desarrollo web con hipertexto y estructura de documentos.

1996 - 2000: Java, Javascript, XML

- Java: Programación orientada a objetos con ejecución en máquinas virtuales (JVM).
- JavaScript: Lenguaje de scripting dinámico para la web.
- XML: Formato de estructuración de datos para la interoperabilidad en la web.

Ejercicio 2 - Historia de los lenguajes elegidos

Historia de JavaScript

JavaScript fue creado en **1995** por **Brendan Eich** mientras trabajaba en **Netscape Communications**. Su objetivo era agregar interactividad a las páginas web dentro del navegador Netscape Navigator. Originalmente, el lenguaje se llamaba **Mocha**, luego **LiveScript**, y finalmente **JavaScript** por razones de marketing, ya que Java era muy popular en ese momento.

En **1996**, Microsoft creó una versión propia llamada **JScript**, lo que llevó a problemas de compatibilidad entre navegadores. Para solucionar esto, en **1997**, JavaScript fue estandarizado por **ECMA International** bajo el nombre **ECMAScript (ES)**. Desde entonces, ha evolucionado con múltiples versiones:

- **ES5 (2009)** introdujo JSON, `strict mode`, y mejoras en el manejo de objetos.
- **ES6 (2015)** trajo `let`, `const`, clases, promesas y arrow functions, modernizando el lenguaje.
- Versiones posteriores agregaron async/await, módulos nativos y más mejoras.

Hoy en día, JavaScript es un pilar del desarrollo web junto con HTML y CSS, y es utilizado en el backend (Node.js), desarrollo móvil y más.

Historia de Java

Java fue creado en **1991** por un equipo de **Sun Microsystems**, liderado por **James Gosling**. Inicialmente, se llamaba **Oak** y estaba pensado para dispositivos electrónicos, pero luego se enfocó en la web. En **1995**, Sun Microsystems lanzó Java oficialmente con el lema **"Write Once, Run Anywhere"** (WORA), destacando su capacidad para ejecutarse en cualquier plataforma gracias a la **Java Virtual Machine (JVM)**.

Algunos hitos clave en su evolución:

- **1996**: Lanzamiento de JDK 1.0.
- **1999**: Java 2 (J2SE, J2EE, J2ME), introduciendo mejoras en rendimiento y escalabilidad.
- **2004-2011**: Java 5, 6 y 7 añadieron genéricos, mejoras en concurrencia y rendimiento.
- **2017**: Oracle cambió a un modelo de lanzamientos más frecuentes.

Java es ampliamente utilizado en aplicaciones empresariales, desarrollo móvil (Android), servidores y más.

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.

Un buen lenguaje de programación debe poseer varios atributos clave para facilitar su uso, mantenimiento y eficiencia.

1. Simplicidad

- Un lenguaje debe tener un conjunto reducido de conceptos fundamentales, evitando la sobrecarga de reglas innecesarias.
- **Ejemplo**: **Python** es conocido por su sintaxis clara y mínima, lo que facilita su aprendizaje y uso.

2. Legibilidad

- El código debe ser fácil de leer y entender, con una sintaxis intuitiva.
- **Ejemplo**: **Python** usa sangría en lugar de llaves `{}` o `begin-end`, lo que mejora la estructura visual del código.

3. Expresividad

- Un lenguaje debe permitir expresar soluciones de manera concisa y eficiente.
- **Ejemplo**: **JavaScript** con su uso de **funciones flecha (arrow functions)** y **destructuración** permite escribir código más corto y claro.

4. Ortogonalidad

- Un lenguaje es ortogonal si sus características pueden combinarse sin restricciones inesperadas.
- **Ejemplo**: **Scheme**, un dialecto de Lisp, tiene un diseño altamente ortogonal, donde funciones, estructuras de datos y operadores pueden usarse de manera consistente en diferentes contextos.

5. Eficiencia

- Debe permitir que los programas se ejecuten rápidamente y consuman pocos recursos.
- **Ejemplo**: **C** es altamente eficiente, ya que proporciona control directo sobre la memoria y permite optimizaciones de bajo nivel.

6. Portabilidad

- Un lenguaje debe ser capaz de ejecutarse en diferentes plataformas sin modificaciones significativas.
- **Ejemplo**: **Java**, con su JVM, permite ejecutar código en cualquier sistema operativo que tenga una máquina virtual compatible.

7. Seguridad

- Debe prevenir errores comunes y proteger contra vulnerabilidades.
- **Ejemplo**: **Rust** tiene un sistema de gestión de memoria seguro sin necesidad de un recolector de basura, evitando errores como accesos indebidos o fugas de memoria.

8. Modularidad

- Debe permitir dividir el código en módulos reutilizables y organizados.
- **Ejemplo**: **Java** y **C#** ofrecen sistemas de clases y paquetes que facilitan la modularización del código.

Python es un lenguaje que destaca por su **simplicidad, legibilidad, expresividad, portabilidad y seguridad**, lo que lo hace ideal tanto para principiantes como para expertos en múltiples dominios, como desarrollo web, ciencia de datos e inteligencia artificial.

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 ej*

En **Python**, se pueden escribir expresiones aritméticas (`2 + 3`), lógicas (`a > b and b < c`), de cadena (`"Hola" + " Mundo"`), de listas (`[x*2 for x in range(5)]`) y lambda (`lambda x: x*2`). Python organiza programas con módulos (`import math`), paquetes (carpetas con `__init__.py`), y clases (`class Persona:`).

Python tiene **simplicidad** (sintaxis clara), **legibilidad** (indentación forzada), **expresividad** (código conciso), **portabilidad** (funciona en varias plataformas) y **seguridad** (tipado dinámico seguro). No es **ortogonal**, porque algunos tipos de datos y estructuras no combinan de manera uniforme (ej., listas mutables vs. tuplas inmutables).

En **JavaScript**, hay expresiones aritméticas (`5 * 2`), lógicas (`x || y`), de cadena (`"Hola " + nombre`), de objetos (`{nombre: "Ana", edad: 25}`), y funciones (`x => x * 2`). Se organiza con módulos (`import {x} from "./modulo.js"`), objetos (`const obj = {}`) y clases (`class Persona {}` en ES6).

JavaScript tiene **expresividad** (flexibilidad en la escritura del código), **portabilidad** (funciona en todos los navegadores), **modularidad** (import/export de ES6), pero no es tan **seguro**, porque permite conversiones implícitas que pueden generar errores (`"5" - 2 da 3`). También es menos **eficiente** que lenguajes compilados como C o Rust.

Ejercicio 5

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*

Ada es un lenguaje diseñado para sistemas críticos, con fuerte tipado y soporte para concurrencia.

Sus **tipos de datos** incluyen enteros, flotantes, booleanos, caracteres, arreglos (`array(1..10) of Integer`), registros (`record`), punteros (`access`) y enumeraciones (`type Color is (Red, Green, Blue)`).

Los **tipos abstractos de datos y paquetes** permiten encapsular datos y operaciones (`package MiPaquete is ... end MiPaquete;`). Se pueden ocultar detalles

internos y exponer solo lo necesario (`private`).

Las **estructuras de datos** incluyen arreglos, listas enlazadas (`linked list`), pilas y colas, que se pueden implementar con tipos abstractos.

El **manejo de excepciones** se hace con `begin ... exception when` y permite capturar errores (`Constraint_Error`, `Program_Error`, `Storage_Error`).

El **manejo de concurrencia** es fuerte con `task`, que define procesos concurrentes. Se usa `rendezvous` para sincronizar tareas (`accept` bloquea hasta que otra tarea lo llame). También soporta `protected objects` para evitar condiciones de carrera.

Ejercicio 6 - 7 - 8

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?
Ejercicio 7: ¿Sobre qué lenguajes está basado?
Ejercicio 8: ¿Qué son los applets? ¿Qué son los servlets?

Ejercicio 6:

Java fue creado para ser un lenguaje portátil, seguro y orientado a objetos. Inicialmente, estaba pensado para dispositivos electrónicos, pero se popularizó en la web con la llegada de los **applets**.

Cambió la web al permitir aplicaciones más dinámicas y seguras, con código ejecutándose dentro del navegador sin necesidad de recompilarlo en diferentes sistemas. También influyó en el desarrollo de aplicaciones empresariales con **Java EE**.

Java **no es dependiente de la plataforma**, porque el código se ejecuta en la **Java Virtual Machine (JVM)**, que es compatible con múltiples sistemas operativos. Esto permite el lema **"Write Once, Run Anywhere" (WORA)**.

Ejercicio 7:

Java está basado en **C y C++**, heredando su sintaxis y estructura, pero eliminando características complejas como punteros directos y herencia

múltiple. También toma ideas de **Smalltalk** en cuanto a la orientación a objetos y la gestión automática de memoria con el **garbage collector**.

Ejercicio 8:

Los **applets** eran pequeñas aplicaciones escritas en Java que se ejecutaban en el navegador, embebidas en páginas web. Fueron populares en los 90s, pero desaparecieron debido a problemas de seguridad y la evolución de tecnologías como JavaScript y HTML5.

Los **servlets** son programas Java que se ejecutan en un servidor web y generan contenido dinámico. Son la base de **Java EE** y se usan para manejar solicitudes HTTP en aplicaciones web.

Ejercicio 9 - 10

Ejercicio 9: ¿Cómo es la estructura de un programa escrito en C? ¿Existe anidamiento de
funciones?
Ejercicio 10: Describa el manejo de expresiones que brinda el lenguaje.

Ejercicio 9:

La estructura de un programa en **C** sigue este orden:

1. **Directivas de preprocesador** (`#include <stdio.h>`)
2. **Declaraciones globales** (variables, constantes, estructuras)
3. **Funciones** (`int main() { ... }`)

Ejemplo básico:

```
#include <stdio.h>

void saludar() {
    printf("Hola, mundo!\n");
}
```

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

En **C**, **no hay anidamiento de funciones** (una función no puede definirse dentro de otra), pero se pueden llamar entre sí.

Ejercicio 10:

C maneja varios tipos de **expresiones**:

- **Aritméticas**: `a + b`, `x * y - z`
- **Lógicas**: `x > y && y < z`
- **Relacionales**: `a == b`, `x != y`
- **Bit a bit**: `x & y`, `a << 2`
- **Asignación**: `x = 5`, `y += 2`
- **Condicionales**: `x = (a > b) ? a : b;`

C permite combinarlas.

Ejercicio 11 - 12

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?
Ejercicio 12: Cite otras características importantes de Python, Ruby, PHP, Gobstone y Processing.
Por ejemplo: tipado de datos, cómo se organizan los programas, etc

Ejercicio 11:

- **Python** se usa para desarrollo web, ciencia de datos, inteligencia artificial y automatización. Sigue los paradigmas **imperativo, orientado a objetos y**

funcional. Su flexibilidad y tipado dinámico lo hacen apto para múltiples estilos de programación.

- **Ruby** es popular en desarrollo web con **Ruby on Rails**. Es **orientado a objetos** puro, ya que todo en Ruby es un objeto. También admite programación funcional.
- **PHP** se usa para desarrollo web y scripting del lado del servidor. Es **imperativo y orientado a objetos**, con características funcionales.

Las características clave para pertenecer a un paradigma incluyen el uso de objetos y clases en **POO**, funciones de orden superior en **funcional**, y ejecución secuencial con control de flujo en **imperativo**.

▼ **Paradigma Orientado a Objetos, Paradigma Funcional y Paradigma Imperativo**

Paradigma Orientado a Objetos (POO)

- Se basa en la idea de **objetos** que contienen datos (**atributos**) y funciones (**métodos**) que operan sobre esos datos.
- Usa **clases** para definir moldes de objetos.
- Permite conceptos como **herencia**, **polimorfismo** y **encapsulamiento**.
- Ejemplo en Python:

```
python
CopiarEditar
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        return f"Hola, soy {self.nombre}"

p = Persona("Ana")
print(p.saludar())  # "Hola, soy Ana"
```

Paradigma Funcional

- Trata las funciones como valores y permite **funciones de orden superior** (funciones que reciben otras funciones como argumento o devuelven funciones).
- Usa **inmutabilidad** (los datos no cambian, se crean nuevos valores).
- Evita el **estado compartido** para reducir errores.
- Ejemplo en Python usando `map`:

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

Paradigma Imperativo

- Se basa en **ejecución secuencial** con instrucciones que modifican el estado del programa.
- Usa estructuras de control como **bucles (`for`, `while`)** y **condiciones (`if`, `else`)**.
- Es el estilo clásico de programación.
- Ejemplo en PHP:

```
php
CopiarEditar
$suma = 0;
for ($i = 1; $i <= 5; $i++) {
    $suma += $i;
}
echo $suma; // 15
```

Cada lenguaje puede combinar varios paradigmas, pero su diseño favorece más a unos que a otros. Python y Ruby soportan POO y funcional, mientras que PHP es principalmente imperativo y orientado a objetos.

Ejercicio 12:

- **Python**: Tipado dinámico, indentación obligatoria, modularización con `import`.
- **Ruby**: Todo es un objeto, sintaxis flexible, bloques (`do ... end`) para iteraciones.
- **PHP**: Diseñado para la web, incrustado en HTML, ejecución en servidores.
- **Gobstone**: Usado en educación, basado en movimientos de un puntero sobre una grilla, control estructurado del flujo.
- **Processing**: Enfocado en gráficos y animaciones, sintaxis basada en Java, ideal para arte digital e interacción visual.

Ejercicio 13 - 14

Javascript
Ejercicio 13: ¿A qué tipo de paradigma corresponde este lenguajes? ¿A qué tipo de Lenguaje
pertenece?
Ejercicio 14: Cite otras características importantes de javascript. Tipado de datos, excepciones,
variables, etc.

Ejercicio 13:

JavaScript es un lenguaje **multiparadigma**, lo que significa que admite diferentes estilos de programación:

- **Orientado a Objetos**: Usa prototipos en lugar de clases tradicionales para definir objetos. Desde ES6, incluye `class` para una sintaxis más familiar.
- **Funcional**: Permite funciones de orden superior, funciones anónimas (`arrow functions`) y manejo inmutable de datos.
- **Imperativo**: Permite instrucciones secuenciales y estructuras de control (`if`, `for`, `while`).

JavaScript es un **lenguaje interpretado** y de **tipado dinámico**, utilizado principalmente en la web para programación del lado del cliente y también en el backend con Node.js.

Ejercicio 14:

- **Tipado de datos**: Es dinámico y débil, lo que significa que una variable puede cambiar de tipo en tiempo de ejecución.

```
javascript
CopiarEditar
let x = 10; // Número
x = "Hola"; // Ahora es un string
```

- **Manejo de excepciones**: Usa `try-catch` para capturar errores y evitar que el programa se detenga.

```
javascript
CopiarEditar
try {
   throw new Error("Algo salió mal");
} catch (error) {
   console.log(error.message);
}
```

- **Variables**: Puede declarar variables con `var`, `let` y `const`:
  - `var`: Tiene **ámbito de función** y puede redeclararse.
  - `let`: Tiene **ámbito de bloque** y no permite redeclaración.
  - `const`: Similar a `let`, pero su valor no puede cambiar.

```
javascript
CopiarEditar
let nombre = "Ana";
nombre = "Luis"; // Válido

const edad = 25;
edad = 30; // Error: no se puede reasignar un const
```

- **Funciones de orden superior**: Se pueden pasar funciones como argumentos a otras funciones.

```
javascript
CopiarEditar
function operar(n, fn) {
  return fn(n);
}

console.log(operar(5, x => x * 2)); // 10
```

- **Asincronismo**: Usa `callbacks`, `Promises` y `async/await` para manejar tareas asíncronas como llamadas a APIs.

```
javascript
CopiarEditar
async function obtenerDatos() {
  let respuesta = await fetch("https://api.example.com");
  let datos = await respuesta.json();
  console.log(datos);
}
```

JavaScript es uno de los lenguajes más flexibles y usados en la actualidad, especialmente en el desarrollo web.