# Práctica Nro 1

**Historia, evolución y características de Leng. de Programación 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.

RTA: {

### **1951 - 1955: Lenguajes tipo Assembly**

* **Característica:** Programación de bajo nivel con código específico para cada máquina.
* **Lenguajes:** Ensamblador

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

* **Característica:** Introducción de lenguajes de alto nivel.
  + **FORTRAN:** Primer lenguaje de alto nivel orientado a cálculos científicos.
  + **ALGOL 58 y ALGOL 60:** Primera definición formal de sintaxis con BNF y estructura de bloques.
  + **LISP:** Introducción de la programación funcional y estructuras dinámicas como listas.

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

* **Característica:** Expansión a aplicaciones comerciales y manipulación de cadenas.
  + **COBOL:** Diseñado para la gestión de datos en negocios y administración.
  + **SNOBOL:** Introduce el procesamiento avanzado de cadenas de texto.
  + **JOVIAL:** Extensión de ALGOL para aplicaciones militares.

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

* **Característica:** Introducción de lenguajes de propósito general y bases de la programación orientada a objetos.
  + **APL:** Uso de operadores matemáticos avanzados y programación simbólica.
  + **FORTRAN 66:** Primera estandarización de FORTRAN.
  + **BASIC:** Lenguaje sencillo para principiantes.
  + **PL/I:** Combinación de características de FORTRAN y COBOL.
  + **SIMULA 67:** Introducción del concepto de clases y objetos.
  + **ALGOL-W:** Predecesor de Pascal, con estructuras de datos mejoradas.

### **1971 - 1975: Pascal, C, Scheme, Prolog**

* **Característica:** Desarrollo de lenguajes estructurados y lógicos.
  + **Pascal:** Fomenta la programación estructurada y el tipado fuerte.
  + **C:** Introducción de punteros y control directo de memoria.
  + **Scheme:** Variante de LISP con un enfoque minimalista.
  + **Prolog:** Introducción de la programación lógica y la inferencia basada en reglas.

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

* **Característica:** Evolución de la programación orientada a objetos y lenguajes con inferencia de tipos.
  + **Smalltalk:** Populariza la orientación a objetos con un entorno interactivo.
  + **Ada:** Lenguaje para sistemas críticos con fuerte tipado.
  + **FORTRAN 77:** Agrega estructuras de control modernas.
  + **ML:** Introduce inferencia de tipos y programación funcional avanzada.

### **1981 - 1985: Smalltalk 80, Turbo Pascal, PostScript**

* **Característica:** Mejora de la eficiencia y lenguajes para gráficos.
  + **Smalltalk 80:** Primera implementación estable de orientación a objetos.
  + **Turbo Pascal:** Entorno de desarrollo rápido con compilación eficiente.
  + **PostScript:** Lenguaje de descripción de páginas para impresión.

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

* **Característica:** Programación orientada a objetos y lenguajes modulares.
  + **FORTRAN 90:** Introducción de programación modular y manipulación de arreglos avanzada.
  + **C++:** Expande C con orientación a objetos.
  + **SML (Standard ML):** Extiende ML con módulos y tipado fuerte.

### **1991 - 1995: TCL, PERL, HTML**

* **Característica:** Lenguajes para scripting y la web.
  + **TCL:** Lenguaje de scripting embebido.
  + **PERL:** Procesamiento avanzado de texto y scripting flexible.
  + **HTML:** Lenguaje de marcado para la web.

### **1996 - 2000: Java, JavaScript, XML**

* **Característica:** Programación para la web y el desarrollo multiplataforma.
  + **Java:** "Escribir una vez, ejecutar en cualquier lugar" con la JVM.
  + **JavaScript:** Lenguaje dinámico para interactividad en la web.
  + **XML:** Formato estándar para almacenamiento y transmisión de datos.

}

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

RTA {

JavaScript: JavaScript fue creado en 1995 por **Brendan Eich**, mientras trabajaba en **Netscape Communications**. Su propósito inicial era agregar interactividad a las páginas web dentro del navegador **Netscape Navigator**.

Inicialmente llamado **Mocha**, luego **LiveScript**, finalmente se renombró a **JavaScript** por razones de marketing, ya que Java era muy popular en ese momento, aunque ambos lenguajes son independientes.

En 1996, Microsoft lanzó **JScript**, una versión propia de JavaScript para su navegador Internet Explorer. Para evitar fragmentación, en 1997, **ECMAScript** fue estandarizado por **ECMA International** (ECMA-262).

Desde entonces, JavaScript ha evolucionado con mejoras significativas:

* **2009:** Se introduce **Node.js**, permitiendo ejecutar JavaScript fuera del navegador.
* **2015 (ES6):** Se agregan **let, const, arrow functions, clases y promesas**, modernizando el lenguaje.
* **Presente:** JavaScript es clave en el desarrollo web moderno, con frameworks como **React, Angular y Vue**.

}

**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.

RTA {

 **Ortogonalidad**: Un lenguaje es ortogonal cuando sus características pueden combinarse de manera independiente sin efectos inesperados.

* **Ejemplo:** **C** permite combinar estructuras como punteros y arreglos de manera predecible.

 **Expresividad**: Permite escribir código claro y conciso con menos líneas de código.

* **Ejemplo:** **Python**, gracias a su sintaxis limpia y funciones como listas por comprensión.

 **Legibilidad**: Facilita la comprensión del código tanto para el programador como para otros que lo lean en el futuro.

* **Ejemplo:** **Python**, debido a su sintaxis similar al lenguaje natural y su uso de indentación obligatoria.

 **Simplicidad**: Reduce la cantidad de reglas y excepciones en el lenguaje, haciéndolo fácil de aprender y usar.

* **Ejemplo:** **Go**, con una sintaxis minimalista y un conjunto reducido de características.

 **Eficiencia**: Un lenguaje debe permitir que el código se ejecute de manera rápida y eficiente en términos de memoria y procesamiento.

* **Ejemplo:** **C**, ampliamente utilizado en sistemas embebidos y software de alto rendimiento.

 **Portabilidad**: Posibilidad de ejecutar el mismo código en diferentes plataformas con mínimas modificaciones.

* **Ejemplo:** **Java**, gracias a la máquina virtual Java (JVM).

 **Modularidad**: Permite dividir el código en módulos reutilizables y bien estructurados.

* **Ejemplo:** **Rust**, con su sistema de módulos y paquetes (Crates).

 **Seguridad**: Prevención de errores comunes y vulnerabilidades.

* **Ejemplo:** **Rust**, con su sistema de gestión de memoria sin necesidad de recolección de basura.

}

**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.

RTA {

### **Tipos de expresiones en Java**

En Java, se pueden escribir varios tipos de expresiones:

1. **Expresiones aritméticas**: Operaciones matemáticas con operadores como +, -, \*, /, %.
   * Ejemplo: int suma = 5 + 3;
2. **Expresiones relacionales**: Comparaciones que devuelven valores booleanos (true o false).
   * Ejemplo: boolean mayor = (5 > 3);
3. **Expresiones lógicas**: Uso de operadores lógicos (&&, ||, !) para evaluar condiciones booleanas.
   * Ejemplo: boolean resultado = (5 > 3) && (10 < 20);
4. **Expresiones de asignación**: Se utilizan para asignar valores a variables.
   * Ejemplo: int x = 10;
5. **Expresiones de incremento y decremento**: Uso de ++ y -- para modificar el valor de una variable.
   * Ejemplo: x++;
6. **Expresiones condicionales (ternarias)**: Expresiones compactas para evaluar condiciones.
   * Ejemplo: int menor = (a < b) ? a : b;
7. **Expresiones con objetos y métodos**: Llamadas a métodos y acceso a atributos de objetos.
   * Ejemplo: String texto = "Hola".toUpperCase();

### **Facilidades provistas para la organización del programa**

Java proporciona varias características para organizar y estructurar programas:

1. **Clases y objetos**: Java es un lenguaje orientado a objetos, por lo que todo código se organiza en clases y objetos.
   * Ejemplo:

java

CopiarEditar

class Persona {

String nombre;

int edad;

}

1. **Paquetes (package)**: Permiten organizar clases en grupos lógicos, facilitando la reutilización y modularidad.
   * Ejemplo: package com.miapp.utilidades;
2. **Interfaces y herencia**: Facilitan la reutilización de código y la implementación de polimorfismo.
   * Ejemplo:

java

CopiarEditar

interface Vehiculo {

void acelerar();

}

1. **Modificadores de acceso (public, private, protected)**: Controlan la visibilidad y accesibilidad de clases, métodos y atributos.
2. **Manejo de excepciones (try-catch-finally)**: Proporciona un sistema robusto para la gestión de errores.
   * Ejemplo:

java

CopiarEditar

try {

int resultado = 10 / 0;

} catch (ArithmeticException e) {

System.out.println("Error: División por cero");

}

### **Atributos de un buen lenguaje de programación en Java**

| **Atributo** | **¿Java lo posee?** | **Justificación** |
| --- | --- | --- |
| **Ortogonalidad** | ❌ No completamente | Aunque Java es consistente, algunas características (como la sobrecarga de métodos y las conversiones de tipos) pueden generar comportamientos inesperados. |
| **Expresividad** | ✅ Sí | Permite escribir código claro con características como la programación orientada a objetos y expresiones lambda. |
| **Legibilidad** | ✅ Sí | Su sintaxis estructurada y tipado fuerte facilitan la comprensión del código. |
| **Simplicidad** | ❌ No completamente | Aunque Java sigue reglas bien definidas, su sintaxis puede ser más compleja que otros lenguajes como Python. |
| **Eficiencia** | ✅ Sí (moderada) | Java es eficiente gracias a la JVM, pero es más lento que lenguajes como C/C++ debido a la sobrecarga de la máquina virtual. |
| **Portabilidad** | ✅ Sí | Gracias a la JVM, el código Java puede ejecutarse en cualquier sistema sin recompilación. |
| **Modularidad** | ✅ Sí | Con paquetes y clases, Java permite dividir el código en módulos reutilizables. |
| **Seguridad** | ✅ Sí | Java gestiona la memoria automáticamente (evita accesos indebidos) y tiene un fuerte sistema de seguridad en la JVM. |

# 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

RTA {

### **Características más relevantes de Ada**

#### **1. Tipos de datos**

Ada es un lenguaje fuertemente tipado, lo que reduce errores y mejora la seguridad en la manipulación de datos. Sus tipos incluyen:

* **Tipos escalares:** Integer, Float, Boolean, Character, Enumeration (permiten definir conjuntos de valores específicos).
* **Tipos compuestos:** Array, Record, Access (punteros seguros), File.
* **Tipos derivados:** Permiten crear tipos basados en otros, con restricciones específicas.
  + Ejemplo:

ada

CopiarEditar

type Edad is range 0 .. 150;

#### **2. Tipos abstractos de datos – paquetes**

Ada permite definir **paquetes** (package) para encapsular tipos de datos y operaciones, fomentando la modularidad. Los paquetes tienen dos partes:

* **Especificación (package)**: Define la interfaz pública.
* **Cuerpo (package body)**: Implementación de la funcionalidad.

Ejemplo de un **tipo abstracto** encapsulado en un paquete:

ada

CopiarEditar

package Pila is

type Pila\_De\_Enteros is private;

procedure Push(P: in out Pila\_De\_Enteros; X: in Integer);

procedure Pop(P: in out Pila\_De\_Enteros);

private

type Pila\_De\_Enteros is array (1..10) of Integer;

end Pila;

#### **3. Estructuras de datos**

Ada soporta varias estructuras de datos, como:

* **Arreglos (Array)**: Soporta índices personalizados.

ada

CopiarEditar

type Dias is (Lunes, Martes, Miercoles, Jueves, Viernes);

type Horario is array (Dias) of Integer;

* **Registros (Record)**: Estructuras similares a structs en C.

ada

CopiarEditar

type Persona is record

Nombre : String (1..50);

Edad : Integer;

end record;

* **Listas enlazadas y colas**: Usadas con punteros (Access).

#### **4. Manejo de excepciones**

Ada tiene un sistema robusto de **manejo de excepciones**, lo que mejora la seguridad y confiabilidad del código. Se usa el bloque begin ... exception con excepciones predefinidas o personalizadas.

Ejemplo:

ada

CopiarEditar

begin

Put\_Line("División segura");

X := A / B;

exception

when Constraint\_Error =>

Put\_Line("Error: División por cero");

when others =>

Put\_Line("Error desconocido");

end;

#### **5. Manejo de concurrencia**

Ada proporciona soporte **nativo para concurrencia** mediante **tareas (task)**, sincronización con **rendezvous (entry)** y **protected objects** para exclusión mutua.

Ejemplo de una tarea concurrente:

ada

CopiarEditar

task Escritor is

entry Escribir(Mensaje: String);

end Escritor;

task body Escritor is

begin

accept Escribir(Mensaje: String) do

Put\_Line(Mensaje);

end Escribir;

end Escritor;

Ada destaca por su seguridad, modularidad y concurrencia, lo que lo hace ideal para sistemas críticos como aviación y defensa.

}

# 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?

RTA {

Java fue creado en **1995** por **James Gosling** en **Sun Microsystems** con el objetivo de desarrollar un lenguaje que fuera:

* **Portátil**: Capaz de ejecutarse en cualquier dispositivo sin depender del hardware.
* **Seguro**: Con un sistema de administración de memoria automático (recolección de basura).
* **Robusto y simple**: Inspirado en C++ pero eliminando características complejas como punteros explícitos.

Inicialmente, se pensó para dispositivos embebidos, pero rápidamente se convirtió en un estándar para aplicaciones empresariales y web.

### **¿Qué cambios introdujo Java a la Web?**

Java revolucionó la web en varios aspectos:

1. **Applets** (hoy obsoletos): Permitían ejecutar código Java en navegadores, agregando interactividad antes de tecnologías como JavaScript.
2. **Servlets y JSP (Java Server Pages)**: Mejoraron la creación de aplicaciones web dinámicas del lado del servidor.
3. **J2EE (Java 2 Enterprise Edition)**: Introdujo frameworks como **Spring** y **EJB** para aplicaciones empresariales escalables.
4. **Android**: La plataforma de desarrollo de Android se basa en Java, permitiendo la creación de aplicaciones móviles.

### **¿Java es dependiente de la plataforma donde se ejecuta?**

**No, Java es independiente de la plataforma.** Esto se debe a que su código no se compila directamente en código máquina, sino en **bytecode**, que se ejecuta en la **Java Virtual Machine (JVM)**.

* **"Write Once, Run Anywhere" (WORA):** Un programa Java puede ejecutarse en cualquier sistema operativo con una JVM instalada.
* **Ejemplo:** Un código Java compilado en Windows puede ejecutarse en Linux o macOS sin cambios.

Sin embargo, la **JVM sí es dependiente de la plataforma**, ya que debe ser específica para cada sistema operativo.

}

**Ejercicio 7:** ¿Sobre qué lenguajes está basado? RTA { Sobre C}

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

RTA {  
Los **applets** eran pequeños programas escritos en **Java** que se ejecutaban dentro de un navegador web. Fueron introducidos en los años 90 para agregar interactividad a las páginas web antes de que tecnologías como **JavaScript y HTML5** se volvieran populares.

#### **Características de los applets:**

* Se ejecutaban en el lado del **cliente**, dentro del navegador.
* Necesitaban un **plugin de Java** para funcionar.
* Eran usados para gráficos, animaciones y funciones interactivas.
* Se cargaban mediante la etiqueta <applet> en HTML.

Hoy están obsoletos.

Los **servlets** son programas en **Java** que se ejecutan en el **servidor** y generan contenido dinámico para aplicaciones web. Fueron creados para reemplazar los **CGI (Common Gateway Interface)**, ofreciendo mayor eficiencia y mejor manejo de sesiones.

#### **Características de los servlets:**

* Se ejecutan en un **servidor web** (como Apache Tomcat).
* Son parte de la tecnología **Java EE**.
* Se usan para manejar solicitudes HTTP y generar respuestas dinámicas (HTML, JSON, XML).
* Se integran con **JSP (Java Server Pages)** y frameworks como **Spring MVC**.

Aun vigentes.

}

# Lenguajes - C

**Ejercicio 9:** ¿Cómo es la estructura de un programa escrito en C? ¿Existe anidamiento de funciones?

RTA {

### **Estructura de un programa en C**

Un programa en **C** sigue una estructura bien definida que incluye:

1. **Directivas de preprocesador**: Instrucciones para el compilador antes de la compilación.
   * Ejemplo: #include <stdio.h> (para importar la biblioteca estándar de entrada/salida).
2. **Declaraciones globales**: Variables y constantes globales accesibles desde cualquier función.
3. **Funciones**:
   * **main()**: Punto de entrada obligatorio del programa.
   * **Funciones auxiliares**: Definidas por el programador para estructurar mejor el código.
4. **C no permite anidar funciones (definir funciones dentro de otras funciones).**
5. En C, todas las funciones deben definirse **al mismo nivel** dentro del archivo fuente. Sin embargo, se pueden **llamar funciones dentro de otras funciones**, lo que simula un comportamiento similar al anidamiento.

}

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

RTA {

### **1. Expresiones Aritméticas**

Realizan operaciones matemáticas con números enteros (int), de punto flotante (float, double), etc.

**Operadores:**

| **Operador** | **Descripción** |
| --- | --- |
| + | Suma |
| - | Resta |
| \* | Multiplicación |
| / | División |
| % | Módulo (resto de división entera) |

**Ejemplo:**

c

CopiarEditar

int a = 10, b = 3;

int suma = a + b; // 10 + 3 = 13

float division = a / (float)b; // 10 / 3.0 = 3.33

### **2. Expresiones Relacionales**

Comparan valores y devuelven un resultado **booleano** (1 para verdadero, 0 para falso).

**Operadores:**

| **Operador** | **Descripción** |
| --- | --- |
| == | Igualdad |
| != | Desigualdad |
| > | Mayor que |
| < | Menor que |
| >= | Mayor o igual que |
| <= | Menor o igual que |

**Ejemplo:**

c

CopiarEditar

int x = 5, y = 10;

int resultado = (x < y); // resultado = 1 (verdadero)

### **3. Expresiones Lógicas**

Se utilizan en estructuras de control (if, while, for) para evaluar condiciones.

**Operadores:**

| **Operador** | **Descripción** |
| --- | --- |
| && | AND lógico |
| ` |  |
| ! | NOT lógico |

**Ejemplo:**

c

CopiarEditar

int edad = 20;

if (edad >= 18 && edad <= 65) {

printf("Es adulto.\n");

}

### **4. Expresiones de Asignación**

Permiten asignar valores a variables.

**Operadores de asignación:**

| **Operador** | **Equivalente a** |
| --- | --- |
| = | Asignación directa |
| += | x = x + valor |
| -= | x = x - valor |
| \*= | x = x \* valor |
| /= | x = x / valor |
| %= | x = x % valor |

**Ejemplo:**

c

CopiarEditar

int numero = 10;

numero += 5; // numero ahora es 15

### **5. Expresiones de Incremento y Decremento**

Sirven para aumentar o reducir el valor de una variable.

**Operadores:**

| **Operador** | **Descripción** |
| --- | --- |
| ++ | Incremento en 1 |
| -- | Decremento en 1 |

**Ejemplo:**

c

CopiarEditar

int contador = 5;

contador++; // contador ahora es 6

### **6. Expresiones Condicionales (Operador Ternario)**

Es una forma compacta de escribir una estructura if-else.

**Sintaxis:**

c

CopiarEditar

condicion ? expresion\_si\_verdadero : expresion\_si\_falso;

**Ejemplo:**

c

CopiarEditar

int edad = 20;

char \*mensaje = (edad >= 18) ? "Mayor de edad" : "Menor de edad";

printf("%s\n", mensaje);

### **7. Expresiones con Bits (Bitwise)**

Operaciones a nivel de bits, útiles en programación de sistemas y optimización.

**Operadores:**

| **Operador** | **Descripción** |
| --- | --- |
| & | AND bit a bit |
| ` | ` |
| ^ | XOR bit a bit |
| ~ | NOT bit a bit |
| << | Desplazamiento a la izquierda |
| >> | Desplazamiento a la derecha |

**Ejemplo:**

c

CopiarEditar

int a = 5, b = 3;

int resultado = a & b; // 5 (101) & 3 (011) = 1 (001)

}

# 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?

RTA {

### **Python**

* **Tipo de programas**: Aplicaciones web, científicas, automatización, inteligencia artificial, análisis de datos, y más.
* **Paradigma**: Multiparadigma (imperativo, orientado a objetos, funcional).
* **Características**:
  + **Imperativo**: Secuencias de comandos que ejecutan instrucciones paso a paso.
  + **Orientado a objetos**: Soporta clases y objetos.
  + **Funcional**: Permite el uso de funciones como ciudadanos de primera clase.

### **Ruby**

* **Tipo de programas**: Aplicaciones web (principalmente con Ruby on Rails), scripts, automatización.
* **Paradigma**: Orientado a objetos (con soporte para programación funcional).
* **Características**:
  + **Orientado a objetos**: Todo es un objeto, incluyendo tipos primitivos.
  + **Funcional**: Soporte para funciones de primera clase y bloques.

### **PHP**

* **Tipo de programas**: Desarrollo web del lado del servidor, aplicaciones web dinámicas.
* **Paradigma**: Imperativo, orientado a objetos.
* **Características**:
  + **Imperativo**: Código secuencial, control de flujo explícito.
  + **Orientado a objetos**: Introducido a partir de PHP 5, permitiendo la creación de clases y objetos.

}

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

# Lenguaje Javascript

**Ejercicio 13**: ¿A qué tipo de paradigma corresponde este lenguajes? ¿A qué tipo de Lenguaje pertenece?

RTA {

### **JavaScript**

* **Paradigma**: **Multiparadigma**
  + JavaScript soporta varios paradigmas, como:
    - **Imperativo**: Permite escribir código que ejecuta secuencias de instrucciones.
    - **Orientado a objetos**: Soporta la creación de objetos y la programación basada en clases (especialmente con ES6 y la introducción de class).
    - **Funcional**: Permite el uso de funciones de primera clase, funciones anónimas (lambdas), y otros conceptos funcionales como la programación con callbacks y promesas.
* **Tipo de lenguaje**:
  + **Lenguaje de alto nivel**: Facilita la escritura de código con sintaxis amigable y abstrae detalles del hardware y gestión de memoria.
  + **Lenguaje interpretado**: El código JavaScript es interpretado por el navegador o por un entorno de ejecución como **Node.js**.
  + **Lenguaje dinámico**: Los tipos de datos de las variables se determinan en tiempo de ejecución, no de manera estática.

JavaScript es ampliamente utilizado para el desarrollo de aplicaciones web, especialmente en el lado del cliente, pero también puede ser usado en el servidor (con Node.js).

}

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

RTA {

### **1. Tipado de Datos**

* **Dinámico**: JavaScript es un lenguaje de tipado dinámico, lo que significa que las variables no tienen un tipo fijo. El tipo de una variable se determina en tiempo de ejecución.
  + **Ejemplo**:

javascript

CopiarEditar

let x = 5; // x es un número

x = "Hola"; // Ahora x es una cadena

* **Débilmente tipado**: Las conversiones de tipo entre diferentes tipos de datos pueden ocurrir implícitamente, lo que puede llevar a resultados inesperados en algunos casos.
  + **Ejemplo**:

javascript

CopiarEditar

let a = "5";

let b = 2;

let c = a + b; // "52" (concatenación, no suma numérica)

### **2. Manejo de Excepciones**

JavaScript usa un mecanismo de manejo de excepciones con los bloques try, catch y finally.

* **try**: Bloque donde se coloca el código que podría generar una excepción.
* **catch**: Bloque que maneja la excepción si ocurre.
* **finally**: Bloque que se ejecuta independientemente de si ocurrió una excepción o no.

**Ejemplo**:

javascript

CopiarEditar

try {

let x = 10 / 0; // Esto podría causar un error

} catch (error) {

console.log("Ocurrió un error: " + error.message);

} finally {

console.log("Este bloque siempre se ejecuta");

}

### **3. Variables y Ámbito**

* **var**: Declaración de variables de ámbito **global** o **local** dentro de una función. Es susceptible de problemas de hoisting (levantamiento).
  + **Ejemplo**:

javascript

CopiarEditar

var x = 10;

* **let**: Introducido en ES6, permite declarar variables con **ámbito de bloque**, evitando algunos problemas de var.
  + **Ejemplo**:

javascript

CopiarEditar

let y = 5; // Ámbito de bloque

* **const**: Declara variables constantes. Una vez asignado un valor, no puede ser modificado. También tiene **ámbito de bloque**.
  + **Ejemplo**:

javascript

CopiarEditar

const PI = 3.1416; // Constante

### **4. Funciones Flecha (Arrow Functions)**

Introducidas en ES6, las **funciones flecha** proporcionan una forma más concisa de escribir funciones, con la particularidad de que **no tienen su propio contexto de this**, lo que las hace muy útiles en programación funcional y manejo de callbacks.

* **Ejemplo**:

javascript

CopiarEditar

const sumar = (a, b) => a + b;

console.log(sumar(5, 10)); // 15

### **5. Asincronía y Promesas**

* **Promesas**: JavaScript maneja operaciones asincrónicas mediante **promesas** que representan el valor de una operación que no ha terminado, pero que lo hará en el futuro.
  + **Ejemplo**:

javascript

CopiarEditar

let miPromesa = new Promise((resolve, reject) => {

let exito = true;

if (exito) {

resolve("Operación exitosa");

} else {

reject("Algo salió mal");

}

});

miPromesa.then(resultado => {

console.log(resultado); // "Operación exitosa"

}).catch(error => {

console.log(error); // "Algo salió mal"

});

### **6. Objetos y Arrays**

* **Objetos**: Son colecciones de propiedades (pares clave-valor).
  + **Ejemplo**:

javascript

CopiarEditar

let persona = {

nombre: "Juan",

edad: 30

};

console.log(persona.nombre); // "Juan"

* **Arrays**: Estructuras ordenadas que almacenan valores.
  + **Ejemplo**:

javascript

CopiarEditar

let frutas = ["manzana", "banana", "cereza"];

console.log(frutas[1]); // "banana"

### **7. Hoisting**

JavaScript realiza el **hoisting** de las declaraciones de variables y funciones, lo que significa que las declaraciones son "elevadas" al inicio del contexto de ejecución, aunque el código se ejecute en un orden diferente al escrito.

* **Ejemplo**:

javascript

CopiarEditar

console.log(a); // undefined

var a = 10; // La declaración var es elevada, pero no la asignación

### **8. Scope (Ámbito de las Variables)**

El **ámbito** de una variable en JavaScript puede ser:

* **Global**: Si se declara fuera de cualquier función o bloque.
* **Función**: Si se declara dentro de una función.
* **Bloque**: Con let y const, las variables tienen un ámbito de bloque, lo que significa que solo son accesibles dentro de las llaves {} donde fueron definidas.

### **9. Eventos y DOM (Document Object Model)**

JavaScript es comúnmente usado para interactuar con el **DOM** (modelo de objetos del documento) de una página web. Permite la manipulación de elementos HTML y responder a eventos como clics, cambios, etc.

* **Ejemplo**:

javascript

CopiarEditar

document.getElementById("miBoton").addEventListener("click", function() {

alert("¡Botón clickeado!");

});

### **10. Módulos (ES6 Modules)**

En ES6, JavaScript introdujo los módulos, lo que permite dividir el código en archivos separados y **exportar e importar** funcionalidades de forma estructurada.

* **Ejemplo**:

javascript

CopiarEditar

// archivo1.js

export const saludo = "Hola";

// archivo2.js

import { saludo } from './archivo1.js';

console.log(saludo); // "Hola"

}