CPLP  
Practica 1



### **951 - 1955: Lenguajes tipo assembly**

* **Características nuevas**:
  + Uso de mnemónicos para representar instrucciones de máquina (por ejemplo, "ADD" en lugar de códigos binarios).
  + Programación simbólica que permitía referirse a direcciones de memoria con etiquetas.
* **Lenguaje que lo incorpora**: Los lenguajes tipo assembly en general fueron los primeros en introducir estas ideas, reemplazando la programación directa en código máquina.

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

* **Características nuevas**:
  + **Abstracción de alto nivel**: FORTRAN introdujo estructuras como bucles y expresiones matemáticas legibles (primer lenguaje de alto nivel compilado).
  + **Estructura de bloques y sintaxis formal**: ALGOL 58 y ALGOL 60 trajeron la idea de bloques de código (begin/end) y una sintaxis bien definida, influyendo en lenguajes modernos.
  + **Programación funcional y listas**: LISP introdujo el procesamiento simbólico, la recursión y las estructuras de datos basadas en listas, siendo pionero en la programación funcional.
* **Lenguajes que las incorporan**:
  + FORTRAN (abstracción de alto nivel).
  + ALGOL 58/60 (estructura de bloques).
  + LISP (programación funcional).

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

* **Características nuevas**:
  + **Lenguaje orientado a negocios**: COBOL introdujo una sintaxis similar al inglés natural para facilitar su uso en aplicaciones empresariales.
  + **Manipulación de cadenas**: SNOBOL destacó por su capacidad avanzada para procesar y manipular texto mediante patrones.
  + **Mejoras en precisión militar**: JOVIAL fue diseñado para sistemas embebidos, con énfasis en control y precisión.
* **Lenguajes que las incorporan**:
  + COBOL (orientación a negocios).
  + SNOBOL (manipulación de cadenas).
  + JOVIAL (aplicaciones embebidas).

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

* **Características nuevas**:
  + **Programación interactiva y matemática**: APL introdujo operadores matemáticos avanzados y notación concisa para cálculos.
  + **Simplicidad para principiantes**: BASIC fue diseñado para ser fácil de aprender, con una sintaxis simple.
  + **Multiparadigma**: PL/I combinó características de FORTRAN (cálculo) y COBOL (negocios).
  + **Programación orientada a objetos**: SIMULA 67 introdujo clases y objetos, sentando las bases de la OOP (Object-Oriented Programming).
* **Lenguajes que las incorporan**:
  + APL (matemática concisa).
  + BASIC (simplicidad).
  + PL/I (multiparadigma).
  + SIMULA 67 (OOP).

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

* **Características nuevas**:
  + **Programación estructurada**: Pascal formalizó el uso de estructuras como procedimientos y funciones con tipado fuerte.
  + **Control de bajo nivel**: C permitió acceso directo a la memoria y gestión manual, ideal para sistemas operativos.
  + **Minimalismo funcional**: Scheme refinó las ideas de LISP con un diseño más simple y elegante.
  + **Programación lógica**: Prolog introdujo la lógica como base para la programación, usando reglas y hechos.
* **Lenguajes que las incorporan**:
  + Pascal (programación estructurada).
  + C (control de bajo nivel).
  + Scheme (funcional minimalista).
  + Prolog (programación lógica).

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

* **Características nuevas**:
  + **OOP completo**: Smalltalk llevó la programación orientada a objetos a un nuevo nivel con conceptos como herencia y mensajes.
  + **Seguridad y concurrencia**: Ada incorporó características para sistemas críticos, como manejo de tareas concurrentes.
  + **Tipado estático funcional**: ML introdujo un sistema de tipos avanzado y polimorfismo en lenguajes funcionales.
* **Lenguajes que las incorporan**:
  + Smalltalk (OOP completo).
  + Ada (seguridad/concurrencia).
  + ML (tipado funcional).

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

* **Características nuevas**:
  + **Entornos gráficos**: Smalltalk 80 integró OOP con interfaces gráficas de usuario (GUI).
  + **Eficiencia y facilidad**: Turbo Pascal combinó la estructura de Pascal con un compilador rápido y un IDE.
  + **Gráficos vectoriales**: Postscript introdujo un lenguaje para describir gráficos e impresión.
* **Lenguajes que las incorporan**:
  + Smalltalk 80 (GUI).
  + Turbo Pascal (IDE eficiente).
  + Postscript (gráficos).

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

* **Características nuevas**:
  + **Programación modular**: FORTRAN 90 añadió módulos y estructuras modernas.
  + **OOP en bajo nivel**: C++ combinó la eficiencia de C con clases y herencia.
  + **Tipado avanzado**: SML refinó los sistemas de tipos en lenguajes funcionales.
* **Lenguajes que las incorporan**:
  + FORTRAN 90 (modularidad).
  + C++ (OOP en bajo nivel).
  + SML (tipado avanzado).

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

* **Características nuevas**:
  + **Scripting y extensibilidad**: TCL fue diseñado para ser embebido y controlar aplicaciones.
  + **Procesamiento de texto**: PERL destacó en manipulación de texto y expresiones regulares.
  + **Marcado web**: HTML introdujo un lenguaje para estructurar contenido en la web.
* **Lenguajes que las incorporan**:
  + TCL (scripting).
  + PERL (texto).
  + HTML (web).

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

* **Características nuevas**:
  + **Portabilidad**: Java introdujo "escribe una vez, ejecuta en cualquier lugar" con su máquina virtual (JVM).
  + **Programación web dinámica**: Javascript permitió interactividad en navegadores.
  + **Datos estructurados**: XML ofreció un formato flexible para intercambio de datos.
* **Lenguajes que las incorporan**:
  + Java (portabilidad).
  + Javascript (web dinámica).
  + XML (datos estructurados).

Ruby es un lenguaje de programación creado por Yukihiro "Matz" Matsumoto en Japón, lanzado por primera vez en 1995. Aquí va un resumen de su historia:

* **Origen (1993-1995)**: Matz comenzó a trabajar en Ruby en 1993, buscando un lenguaje que combinara la simplicidad de Perl con la orientación a objetos de Smalltalk. La primera versión pública, Ruby 0.95, salió el 21 de diciembre de 1995.
* **Filosofía**: Diseñado con énfasis en la productividad del programador y la legibilidad del código, siguiendo el principio de "mínima sorpresa". Matz quería que fuera divertido de usar.
* **Crecimiento (1996-2000)**: Ruby ganó popularidad en Japón durante los años 90. La versión 1.0 se lanzó en 1996, y su comunidad creció con la publicación de documentación en inglés en 1997.
* **Expansión global (2000s)**: Su adopción internacional explotó con el framework Ruby on Rails, creado por David Heinemeier Hansson en 2004. Rails, enfocado en desarrollo web rápido, llevó a Ruby a la fama mundial.
* **Evolución (2010s-presente)**: Ruby ha seguido evolucionando con versiones como 2.0 (2013), que mejoró rendimiento, y 3.0 (2020), que introdujo tipado estático opcional y JIT. Aunque Rails sigue siendo su mayor impulsor, Ruby se usa en diversas áreas como automatización y scripting.



Un ejemplo de un lenguaje que cumple con todas las carecteristicas enunciadas es Python, ya que es ortogonal, es muy simple, ya que incluso marca el cambio de linea con indentacion, es facil de leer y su expresividad es amplia.



### **JavaScript (JS)**

#### **Tipos de expresiones**

* Aritméticas: 3 + 5 \* 2.
* Comparación: x > 10, === (estricta).
* Lógicas: x && y.
* Asignación: x = 5.
* Ternarias: x ? "sí" : "no".
* Funciones: x => x + 1, .map().
* Objetos/Arreglos: { a: 1 }.a, [1, 2].length.
* Especiales: x ?? "defecto", eval().

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

* **Módulos**: import/export (ES6) para estructurar código en archivos.
* **Funciones y closures**: Encapsulan lógica y mantienen estado.
* **Clases**: Soporte OOP con class para herencia y encapsulación.
* **Asincronía**: async/await, promesas para manejar operaciones no bloqueantes.
* **Espacios de nombres**: Objetos o módulos evitan colisiones.

#### **Propiedades**

* **Ortogonalidad**: Media. Tiene inconsistencias (e.g., == vs ===, tipos dinámicos confusos como null vs undefined), lo que limita su uniformidad.
* **Expresividad**: Alta. Permite soluciones concisas (e.g., arrow functions, métodos de arreglos), pero a veces requiere más código que Python.
* **Legibilidad**: Media-Alta. Sintaxis clara, pero puede complicarse con asincronía o abuso de callbacks.
* **Simplicidad**: Media. Flexibilidad y compatibilidad histórica (e.g., var vs let) añaden complejidad.

### **Python**

#### **Tipos de expresiones**

* Aritméticas: 3 + 5 \* 2.
* Comparación: x > 10.
* Lógicas: x and y.
* Asignación: (a := 5) (3.8+).
* Ternarias: "sí" if x else "no".
* Listas por comprensión: [x \* 2 for x in range(5)].
* Funciones: lambda x: x + 1.
* Diccionarios/Listas: {"a": 1}["a"].

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

* **Módulos**: import para dividir código en archivos reutilizables.
* **Funciones y decoradores**: Encapsulan y modifican comportamiento.
* **Clases**: Soporte OOP con herencia y métodos especiales.
* **Paquetes**: Estructuras jerárquicas con \_\_init\_\_.py.
* **Gestión de excepciones**: try/except robusto para control de errores.

#### **Propiedades**

* **Ortogonalidad**: Alta. Reglas consistentes (e.g., tipado dinámico sin ambigüedades como en JS), pocas excepciones.
* **Expresividad**: Muy alta. Listas por comprensión y sintaxis minimalista reducen código (e.g., [x\*\*2 for x in range(5)]).
* **Legibilidad**: Muy alta. Diseñado para ser claro (PEP 8), con indentación obligatoria.
* **Simplicidad**: Alta. Filosofía "una sola forma obvia de hacerlo" evita complejidad innecesaria.

### **Comparación y justificación**

* **Ortogonalidad**: Python > JS. JS tiene más excepciones e inconsistencias heredadas.
* **Expresividad**: Python ≥ JS. Python ofrece construcciones más concisas (comprensiones vs .map()), pero JS es flexible en entornos web.
* **Legibilidad**: Python > JS. La sintaxis de Python es más natural y forzada a ser clara; JS puede ser confuso con mal uso.
* **Simplicidad**: Python > JS. Python evita redundancias y complejidad histórica que JS arrastra por compatibilidad.



ADA cuenta con los siguientes tipos de datos:

**Tipos de datos**

* Integer: un número entero.
* Float: un número decimal.
* Character: una letra o símbolo del teclado (también puede ser un número o varios, pero no se pueden hacer operaciones entre ellos).
* String: consiste en una cadena de caracteres.
* Array: un array o vector es una variable que agrupa varios elementos de cualquiera de los tipos previamente descritos.

**Tipos abstractos de datos (TAD) – Paquetes**: Ada utiliza paquetes como la base para implementar tipos abstractos de datos, ofreciendo una separación clara entre especificación (interfaz pública) y cuerpo (implementación privada). Esto permite encapsulación y abstracción, con tipos privados que ocultan detalles internos. Además, los paquetes genéricos (generics) facilitan la reutilización de TAD parametrizados, como pilas o colas, manteniendo la seguridad del tipado.

**Estructuras de datos**: El lenguaje proporciona arrays (estáticos o dinámicos con índices personalizados), registros (con campos nombrados y variantes) y punteros (restringidos para evitar errores). Estas estructuras, combinadas con el tipado fuerte y las verificaciones en tiempo de compilación, permiten construir datos complejos como listas o árboles de forma segura y eficiente, sin las vulnerabilidades típicas de lenguajes como C.

**Manejo de excepciones**: Ada integra un sistema robusto de excepciones, con opciones predefinidas (como Constraint\_Error) y la capacidad de definir excepciones propias. Mediante bloques begin ... exception ... end, los errores se capturan y manejan fácilmente, con propagación automática a niveles superiores si no se tratan, lo que asegura un control predecible y seguro de fallos.

**Manejo de concurrencia**: La concurrencia en Ada se gestiona con "tasks" (tareas), unidades nativas de ejecución paralela que se sincronizan mediante el mecanismo "rendezvous" para intercambio seguro de datos. Desde Ada 95, los "protected types" complementan esto, protegiendo datos compartidos contra accesos concurrentes incorrectos. Este enfoque garantiza seguridad y evita problemas como condiciones de carrera, siendo ideal para sistemas críticos.



**Java** fue creado con el propósito de ser un lenguaje de programación portátil, robusto y seguro, inicialmente enfocado en dispositivos electrónicos y sistemas embebidos.

Java trajo un cambio significativo a la Web al introducir los **applets**, pequeños programas escritos en Java que podían ejecutarse en navegadores mediante la JVM, permitiendo contenido interactivo y dinámico más allá del HTML estático.

No, Java no es dependiente de la plataforma donde se ejecuta, gracias a su diseño basado en la **Java Virtual Machine (JVM)**. El código Java se compila a un bytecode intermedio que la JVM interpreta o compila a código máquina nativo en tiempo de ejecución, adaptándose al sistema operativo y hardware subyacente (Windows, Linux, macOS, etc.).

1. Esta basado en C y C++, aunque tiene menos funcionalidades de bajo nivel que estos dos.
2. Los servlets son programas escritos en Java que se ejecutan en un servidor web para ampliar sus capacidades. Se utilizan para generar contenido dinámico y responder a las peticiones de los clientes.

### 

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

Un programa en C se organiza con directivas del preprocesador (e.g., #include), declaraciones globales de variables o funciones, la función main como punto de entrada, y definiciones de funciones adicionales. Todo se estructura en bloques delimitados por llaves, con variables locales dentro de funciones.

### **¿Existe anidamiento de funciones?**

No, C no permite definir funciones dentro de otras funciones. Solo se pueden anidar bloques de código dentro de una función para limitar el alcance de variables, pero las funciones deben definirse independientemente en el ámbito global. Esto mantiene el diseño simple y lineal de C.

En **C**, el manejo de expresiones es directo y eficiente, típico de un lenguaje de bajo nivel con influencia en su diseño estructurado. Soporta:

* **Aritméticas**: Operaciones como suma (+), resta (-), multiplicación (\*), división (/), y módulo (%), con prioridad estándar y paréntesis para control.
* **Relacionales**: Comparaciones (>, <, ==, etc.) que devuelven 0 (falso) o 1 (verdadero).
* **Lógicas**: Operadores && (and), || (or), ! (not) para combinar condiciones.
* **Asignación**: = y compuestas como +=, -=, etc., que también devuelven valores.
* **Bit a bit**: Operaciones como &, |, ^, ~, <<, >> para manipulación de bits.
* **Punteros y acceso**: Uso de \* (desreferencia) y & (dirección), integrados en expresiones.

Las expresiones se evalúan en un orden definido por precedencia y asociatividad, y son fuertemente tipadas, requiriendo conversiones explícitas (casts) si los tipos no coinciden. Su simplicidad y cercanía al hardware lo hacen eficiente, pero carece de expresiones avanzadas como lambdas o comprensiones de otros lenguajes.

**PHP**

* **Tipos de programas**: Aplicaciones web (e.g., sitios dinámicos, CMS como WordPress), APIs, sistemas de gestión de contenido, comercio electrónico.
* **Paradigma**: Imperativo, orientado a objetos (desde PHP 5), con elementos procedimentales.
* **Características**:
  + **Imperativo/Procedimental**: Código secuencial y funciones como base en sus primeras versiones.
  + **Orientado a objetos**: Clases, herencia y encapsulación añadidas en PHP 5, con soporte completo para OOP.
  + **Tipado dinámico**: Variables sin declaración estricta de tipo, lo que lo hace flexible pero menos riguroso.
  + **Integración web**: Diseñado para generar HTML dinámico, con sintaxis embebida en páginas web.
  + Determinación: Su evolución desde scripts simples a OOP y su enfoque en procesamiento del lado del servidor lo alinean con estos paradigmas.

### **Ruby**

* **Tipos de programas**: Aplicaciones web (e.g., con Ruby on Rails), herramientas de automatización, scripting, prototipos rápidos.
* **Paradigma**: Orientado a objetos puro, con elementos funcionales e imperativos.
* **Características**:
  + **Orientado a objetos**: Todo es un objeto (incluso números y cadenas), con métodos, herencia y polimorfismo integrados.
  + **Funcional**: Soporta bloques, lambdas y funciones como ciudadanos de primera clase.
  + **Tipado dinámico**: Flexibilidad en tipos, sin necesidad de declaración previa.
  + **Sintaxis expresiva**: Diseñada para ser legible y minimalista, siguiendo el principio de "mínima sorpresa".
  + Determinación: Su diseño centrado en objetos y la incorporación de conceptos funcionales lo hacen un lenguaje OOP con influencias modernas.



### **Python**

* **Tipado**: Dinámico y fuerte (no requiere declaración de tipos, pero respeta compatibilidad estricta).
* **Organización**: Módulos (import), funciones, clases; indentación obligatoria define bloques.
* **Otras**: Multiparadigma (OOP, funcional, imperativo), legibilidad como prioridad (PEP 8), intérprete interactivo.

### **Ruby**

* **Tipado**: Dinámico y fuerte (todo es objeto, incluso primitivos).
* **Organización**: Clases y módulos (require), bloques para estructuras concisas, sintaxis flexible.
* **Otras**: Orientado a objetos puro, expresividad alta, soporte para metaprogramación (modificar código en ejecución).

### **PHP**

* **Tipado**: Dinámico y débil (conversión automática de tipos, menos rigurosidad).
* **Organización**: Scripts embebidos en HTML, funciones, clases; archivos organizados por inclusión (include, require).
* **Otras**: Enfocado en web, soporte OOP desde PHP 5, ejecución del lado del servidor, amplia integración con bases de datos.

### **Gobstone**

* **Tipado**: No aplica en sentido clásico; enfocado en comandos y estados del tablero, sin variables tipadas explícitas.
* **Organización**: Secuencia de comandos imperativos (mover, poner, sacar), procedimientos para reutilización.
* **Otras**: Educativo, basado en un tablero gráfico, sintaxis simple para enseñar lógica y algoritmos.

### **Processing**

* **Tipado**: Estático y débil (heredado de Java, pero con flexibilidad en ciertas conversiones).
* **Organización**: Funciones como setup() y draw() para inicialización y bucle gráfico, estructurado en sketches.
* **Otras**: Diseñado para gráficos y animaciones, entorno visual interactivo, basado en Java pero simplificado.



**Paradigma**: Multiparadigma (orientado a objetos con prototipos, funcional, imperativo).

**Tipo de lenguaje**: Alto nivel, interpretado, dinámico, débilmente tipado.

### **Características importantes de JavaScript**

* **Tipado**: Dinámico y débil (conversión automática, e.g., 5 + "5" = "55").
* **Excepciones**: Manejo con try, catch, throw para errores controlados.
* **Variables**: Declaradas con var (ámbito función), let/const (ámbito bloque), sin tipado fijo.
* **Otras**: Asincronía (async/await), funciones de primera clase, prototipos, ejecución en event loop.