**Paradigma:** Forma de afrontar y resolver un problema, método de trabajo

**Tipos de paradigma**

* Paradigma Imperativo (Como lo resuelvo)
  + Es el más antiguo (1950) y representativo
  + Manda órdenes, impera
  + Define ambiente, asigna memoria, ejecuta linealmente y da resultado
  + Se basa en la máquina de Turing y el concepto de von neumann
  + Habla de cambios de estado
  + Usa instrucciones
  + Usa procedimientos para resolver las cosas
  + La implementación de hardware suele ser imperativa
  + 3 conceptos claves:
    - Celdas de memoria
    - Asignación
    - Repetición
  + Tipos de programación imperativa:
    - Estructurada: Es imperativa, debe ser comprobable, modular y mantenible
    - Procedural: Es imperativa, usa procedimientos y subrutinas, cambia estados.
  + Ventajas de la programación imperativa:
    - Es simple
    - Es reutilizable
    - Fácil de seguir el flujo del programa
    - Puede ser modular
    - Necesita menos memoria que otros tipos
  + Desventajas de la programación imperativa:
    - Datos expuestos a todo el programa (Inseguro)
    - Difícil de relacionar con objetos reales
    - Difícil crear nuevos tipos de datos
    - Más importante el procedimiento que los datos
    - Difícil de testear los programas complejos (Puede llegar a ser imposible)
* Paradigma Declarativo (Que resuelve) “Pensamos como humano, la máquina resuelve y genera las instrucciones”
* Paradigma Inteligencia Artificial (Nuevo. nos dice que necesitamos, nos da órdenes)

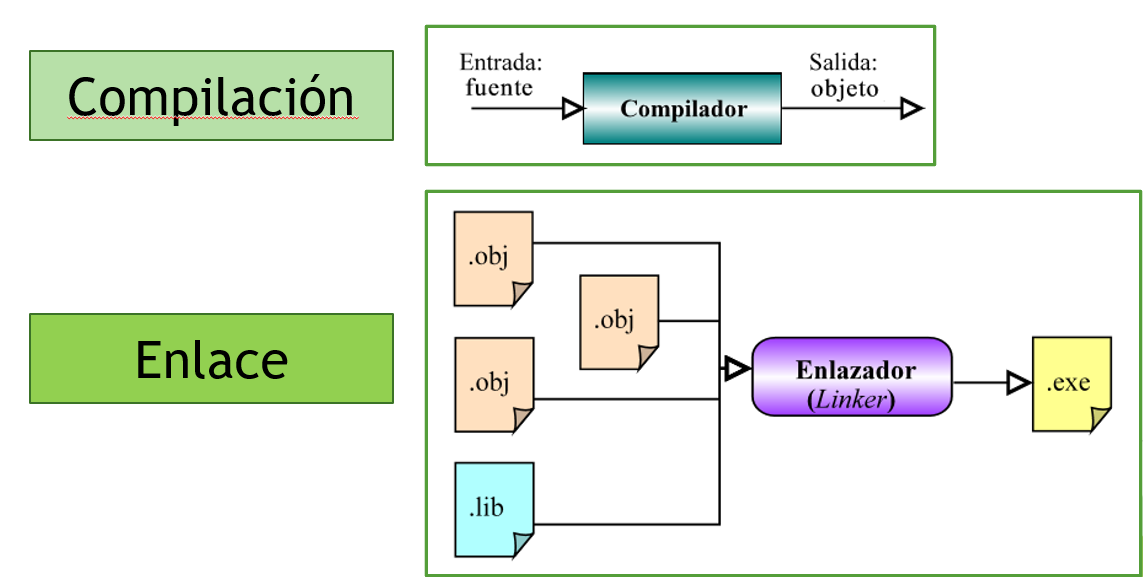
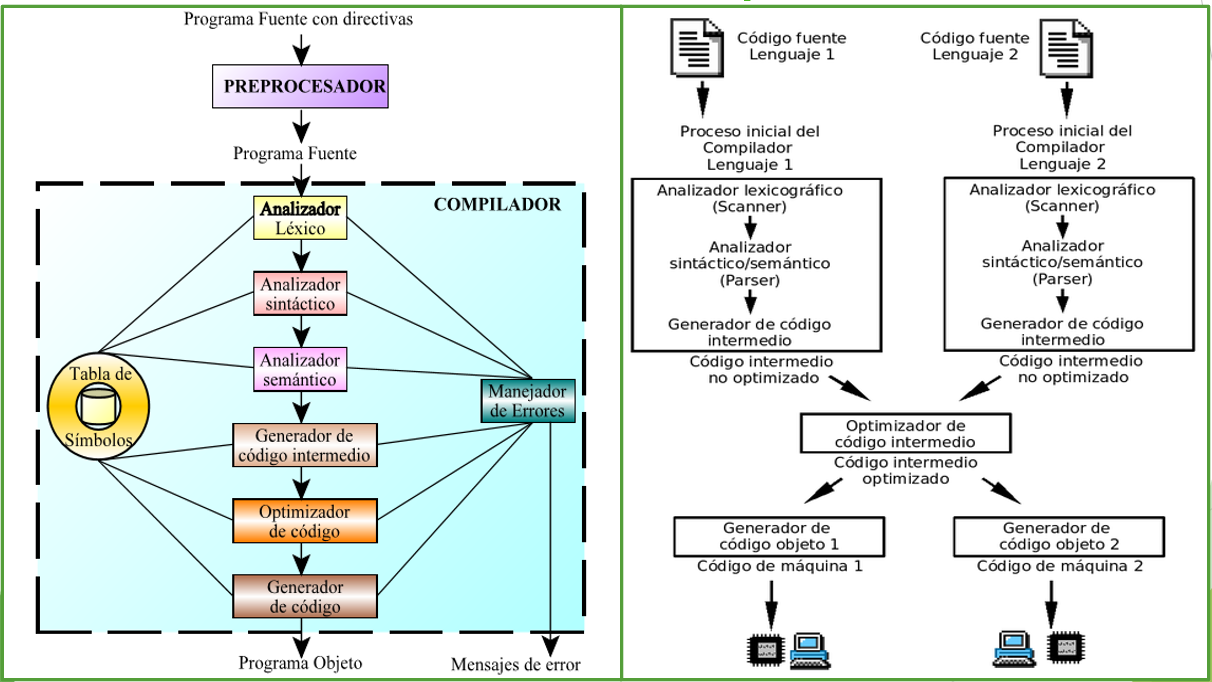
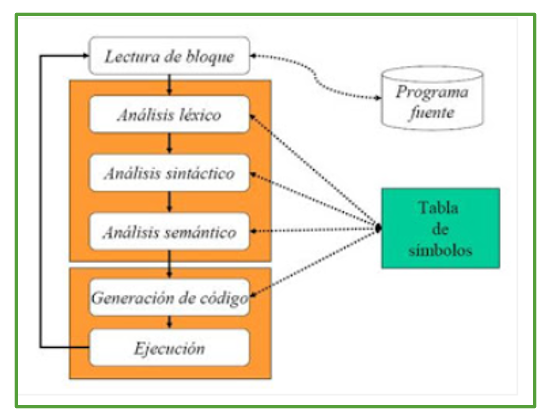
**Criterios de clasificación de lenguajes**

* Nivel de abstracción:
  + Bajo Nivel: Programación a nivel de hardware, complejo de escribir. Ejemplo: Assembler
  + Nivel Medio: Mayor grado de abstracción pero con características de bajo nivel. Ejemplo c
  + Alto Nivel: Más parecido al lenguaje humano. Maneja conceptos, tipos de datos, etc. Totalmente portables e independientes de las características del cpu. Ejemplo: Java.
* Propósito:
  + Propósito general: Aptos para todo tipo de tareas. Ejemplo: c.
  + Propósito Específico: Diseñados para tareas específicas y concretas. Ejemplo: SIMULA.
  + Programación de sistemas: Diseñados para programar sistemas operativos o drivers. Ejemplo: c
  + Para scripting: Para realizar scripts. Ejemplo: JavaScript
* Evolución histórica:
  + Primera Generación: Código de máquina
  + Segunda Generación: Ensamblador y algunos lenguajes no estructurados como fortran.
  + Tercera Generación: La mayoría de los lenguajes modernos. Ejemplo: c, Java
  + Cuarta Generación: Lenguajes diseñados con propósitos concretos. Ejemplo: NATURAL, SQL
  + Quinta Generación: Lenguajes utilizados en inteligencia artificial. La idea es que se declare el problema y los datos y la máquina lo resuelva sola. Ejemplo: prolog
* Formas de ejecución:
  + Compilados (se compilan y ejecutan)
  + Interpretados (no se compilan)
  + Mixtos (se compila en un lenguaje determinado como bytecode y se corre en algún framework específico)
* Formas de desarrollar la tarea:
  + Imperativo
  + Declarativo
* Paradigmas de programación
  + Procedural
  + Orientado a objetos
  + Funcional
  + Lógico
* Lugar de ejecución:
  + Lenguaje de servidor
  + Lenguaje de cliente
* Concurrencia (no tiene nada ese)
* Interactividad:
  + Orientados a sucesos
  + No orientados a sucesos
* Determinismo:
  + Lenguajes deterministas (Todo lo anterior)
  + Lenguajes probabilísticos o No deterministas: Sirven para explorar grandes espacios y en la investigación teórica de hipercomputación

**Traductores**

Un traductor es un programa que recibe un código escrito en algún lenguaje y lo transforma a otro lenguaje. Por lo general traducen un código escrito en algún lenguaje de alto nivel en otro programa, llamado objeto, en código o lenguaje de máquina.

Hay 4 tipos de traductores diferentes:

* Ensambladores:
  + Traduce algo escrito en lenguaje ensamblador a lenguaje de máquina
  + Facilita la escritura de programas ya que es imposible escribir en binario
  + Sustituye todo el codigo por codigo binario
  + Utilizable en la actualidad cuando se requiere control directo de hardware
* PreProcesadores:
  + Traducen el código de un lenguaje de alto nivel a otro para que pueda ser pasado a lenguaje de máquina
  + No analiza el texto original si no que lo divide para formar macros
  + Puede eliminar comentarios, incluir archivos y ejecutar sustituciones de macros
* Compiladores:
  + Traduce un código escrito en un lenguaje de alto nivel a un lenguaje de nivel inferior para que la máquina lo pueda interpretar
  + Usualmente el segundo lenguaje es el lenguaje de máquina pero pueden existir lenguajes intermedios
  + 
  + Existen múltiples compiladores para un solo lenguaje
  + El compilador traduce sentencia a sentencia el programa fuente
  + Etapas del proceso de compilación:
    - Edición: Escribe el programa empleando un lenguaje y un editor y nos da el código fuente de nuestro programa
    - Compilación: Traduce dicho código fuente a código de máquina. En caso de encontrar un error, para y hay que corregirlo
    - Linkeado o enlace: Une el archivo objeto generado en la fase de compilación con rutinas internas del lenguaje para crear el ejecutable. Hay 2 tipos de linkeado:
      * Estático Añade librerías a nuestros binarios compilados para generar el ejecutable
      * Dinámico: No añade librerías si no que hace que se carguen en memoria en el momento en que se necesiten
  + 
  + Partes de un compilador:
    - Front end: Es lo que analiza el código, comprueba su validez, genera el árbol de derivación y completa los valores de la tabla de símbolos. Suele ser independiente de la plataforma y se lo conoce como fase de análisis
    - Back end: Es la parte que genera el código máquina. Es específico de cada plataforma y se basa en los resultados de la fase anterior.
    - Esta división en front y back permite que estos sean utilizados en varios lenguajes y plataformas distintas
    - El código generado en el back end necesita un linker para ser ejecutado
  + Tipos de compiladores:
    - Cruzados: Generan código para un sistema distinto del que están funcionando
    - Optimizadores: Optimizan el código
    - De una sola pasada: Generan el código máquina a partir de una única lectura del código fuente
    - De varias pasadas: Generan el código máquina a partir de varias lecturas del código fuente
    - JIT (just in time): Son parte del intérprete y compilan partes del código cuando son necesarias
  + 
  + Ventajas de compiladores:
    - Producen código optimizado
    - Generan una ejecución más veloz
    - Información de errores más detallada
  + Desventajas de compiladores:
    - Se debe ejecutar muchas veces el código fuente para ver los cambios
    - Mayor consumo de memoria
* Intérpretes:
  + Procesa y traduce cada instrucción o sentencia de un lenguaje de alto nivel a código de máquina y después ejecuta
  + Solo realizan la traducción a medida que sea necesaria y no suelen guardarla
  + Facilitan la búsqueda de errores ya que se puede interrumpir la traducción en cualquier momento y puede modificarse sobre la marcha
  + Permite que un solo código funcione en sistemas diferentes
  + Un programa interpretado suele ser más lento que un programa compilado pero más flexible
  + Permite ofrecer un entorno no dependiente de la máquina (Comúnmente conocido como máquina virtual)
  + Para mejorar el rendimiento a veces se compacta el código fuente para luego ser traducido
  + Los intérpretes realizan 2 operaciones:
    - Traducen el código a un formato interno
    - Ejecutan/Interpretan la traducción
    - 
  + Tipos de intérpretes:
    - Puros: Analizan y ejecutan sentencia a sentencia todo el programa fuente. Utilizados en lenguajes simples, no usan mucha memoria ya que van línea a línea pero al detectar un error deben volver a comenzar el proceso.
    - Avanzados: Incorporan un paso previo al análisis de todo el programa. Generan un lenguaje intermedio que es ejecutado por ellos mismos. Con esto consiguen un mejor manejo de errores.
    - Incrementales: Utilizado cuando el código fuente tiene objetos o funciones que no pueden ser compiladas directamente. Para interpretarlos, primero compila las partes estáticas y luego intenta compilar las partes dinámicas o reemplazarlas y recompilarlas.
  + Ventajas de los intérpretes:
    - Fácil depuración
    - En algunos lenguajes permiten añadir código durante la ejecución
    - Fácil de interrumpir
    - Fácil de modificar y ejecutar nuevamente
    - Necesita menos memoria que un compilador
    - Facilita la búsqueda de errores
  + Desventajas de los compiladores:
    - Lentitud de ejecución
    - Cada instrucción debe ser traducida tantas veces como sea ejecutada
    - Durante la ejecución, el intérprete reside en memoria ya que no guarda código objeto.
* Diferencias entre intérprete y compilador:
  + Intérprete traduce instrucciones a un lenguaje intermedio y el compilador traduce directo a código máquina
  + El intérprete traduce linea a linea y el compilador todo el programa
  + El intérprete detecta errores y permite su depuración mientras interpreta, el compilador da un reporte al final de la compilación.
* Similitudes entre intérprete y compilador:
  + Ambos están disponibles en la mayoría de los lenguajes.

