**PROCESAMIENTO DE LENGUAJES**

**Traducción**:

* Traduce todas las instrucciones a lenguaje de máquina y guarda el resultado.
* Tiene dos etapas principales:
  + - **Análisis** (front-end): analiza léxico, sintaxis y significado de las instrucciones.
    - **Síntesis** (back-end): genera el programa en lenguaje de máquina.
* Cada lenguaje necesita su propio compilador.

**Interpretación**:

* Traduce y ejecuta línea por línea, sin guardar la traducción.
* Toma una instrucción.
* La traduce.
* La ejecuta.
* Si una instrucción se repite, la traduce y ejecuta cada vez.
* No crea un programa en lenguaje de máquina.
* Ejemplos: **Python, JavaScript.**

**Traducción vs. Interpretación**:

**Traducción**:

* + Usa más memoria, pero es más rápida porque guarda las instrucciones ya traducidas.
  + Ejemplo: C, C++.

**Interpretación**:

* + Usa menos memoria y es más fácil corregir errores, pero es más lenta porque traduce mientras ejecuta.
  + Traduce repetidamente las instrucciones que se repiten.

**TRADUCCIÓN**

**Análisis Léxico**

· Se lee el programa **carácter por carácter** para identificar sus partes (tokens).

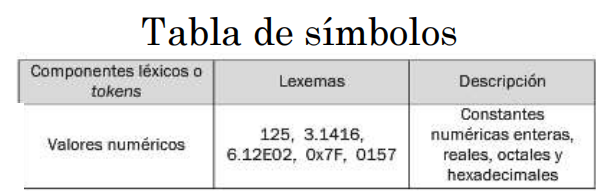
· Los **tokens** son palabras o símbolos:

* + Identificadores, variables, etc.

· **Comentarios se ignoran.**

· Los tokens encontrados se registran en la **tabla de símbolos**, que almacena toda la información del programa.

· Para validar los tokens, se usan **autómatas finitos**.



**Análisis Sintáctico**:

1. Verifica que las **combinaciones de tokens** forman **sentencias correctas** según la gramática del lenguaje.
2. Si hay errores, los **informa**.
3. Usa **gramáticas** y construye un **árbol de derivación** para validar que las secuencias sean válidas.

**Análisis Semántico**:

1. Comprueba el programa **sintácticamente correcto** y reúne información sobre **tipos de variables**.
2. Detecta errores como:

* Tipos incorrectos.
* Problemas de flujo de control.
* Incoherencias en nombres de identificadores o argumentos.
* Posibles errores en ejecución.

1. Se apoya en **gramáticas con atributos** para implementar esta fase.

**Síntesis**:

1. Traduce las instrucciones a **código intermedio** (assembler).
2. Luego, las convierte a **código máquina**.
3. El resultado es el **programa objeto compilado**.
4. Ejemplos de lenguajes compilados: **C, C++, C#, etc**

**Orden** traducción **de** **programa**: léxico, sintáctico, semántico, tabla de símbolos, síntesis y generacion de código.

**LENGUAJES CON PROCESAMIENTO HÍBRIDO O MIXTO**

**Lenguajes que combinan compilación e interpretación**:

1. Usan ambas estrategias:
   1. **Compilan** el código fuente a un **código intermedio**.
   2. Luego, **interpretan línea por línea** ese código intermedio.
2. **Java** es un ejemplo:
   1. El **JDK** compila el código fuente a **bytecode**.
   2. El **JRE** interpreta y ejecuta el bytecode.
   3. usa una **máquina virtual** para esta tarea.

**Análisis Semántico y Gramáticas con Atributos**:

1. Las **gramáticas con atributos** combinan propiedades sintácticas y semánticas:
   1. Asignan **atributos** a símbolos terminales y no terminales.
   2. Cada regla de producción tiene:
      1. **Reglas semánticas**: especifican cómo cambian los atributos.
      2. **Condiciones**: deben cumplirse para validar el contexto.
2. Una **sentencia correcta sintáctica y semánticamente** cumple todas las condiciones de los atributos.
3. El **analizador semántico** toma el árbol sintáctico generado por el análisis previo y:
   1. Determina los valores de los atributos.
   2. Aplica las reglas semánticas definidas.
4. Ejemplos de atributos:
   1. Tipo de una variable.
   2. Valor asignado.
   3. Dirección de memoria.
   4. Número y tipos de argumentos de una función.