

Universidad San Carlos de Guatemala  
Facultad de ingeniería.  
Ingeniería en ciencias y sistemas



# Proyecto 2

## JavaBridge: Traductor de Lenguajes Java a Python

**PONDERACIÓN: 40**

## Índice

### Contenido

1. Resumen Ejecutivo .....	3
2. Competencia que desarrollaremos.....	3
3. Objetivos del Aprendizaje .....	3
3.1 Objetivo General .....	3
3.2 Objetivos Específicos .....	3
4. Enunciado del Proyecto .....	4
4.1 Descripción del problema a resolver .....	4
4.2 Léxico y Sintaxis.....	9
4.3 Gramática Libre de Contexto .....	10
4.4 Política de Errores .....	10
4.5 Ejemplos Archivos .....	11
5 Alcance del proyecto.....	12
6 Reportes HTML.....	14
Metodología .....	15
Desarrollo de Habilidades Blandas .....	16
Cronograma .....	16
Valores.....	17

## 1. Resumen Ejecutivo

Implementar un traductor de código Java básico a Python.

- El sistema debe leer archivos .java (entrada en Java) y producir .py (salida en Python) solo si no hay errores.
- Se deben incluir analizador léxico y analizador sintáctico manuales, reportes HTML (errores léxicos/sintácticos o tokens) y ejecución simulada del código Python generado.

## 2. Competencia que desarrollaremos

- Diseñar AFD y tokens para un subconjunto de Java.
- Construir gramática libre de contexto y parser manual.
- Traducir constructos de Java a Python preservando semántica (tipos, estructuras de control, operaciones).
- Generar reportes HTML y simular ejecución del código traducido.
- Diseñar una interfaz mínima (HTML/JS) para editar/cargar, traducir y visualizar.

## 3. Objetivos del Aprendizaje

### 3.1 Objetivo General

Desarrollar una aplicación con interfaz que traduzca Java a Python integrando análisis léxico y sintáctico manual, con reportes y simulación de ejecución.

### 3.2 Objetivos Específicos

1. Implementar un AFD que reconozca tokens de Java (sin regex ni librerías).
2. Desarrollar un parser manual para un subconjunto acotado de Java.
3. Definir reglas de traducción Java  $\rightarrow$  Python (tipos, estructuras de control, operaciones).
4. Generar HTML de tokens/errores y simular ejecución del código Python.
5. Entregar .py bien formado cuando no haya errores.

## 4. Enunciado del Proyecto

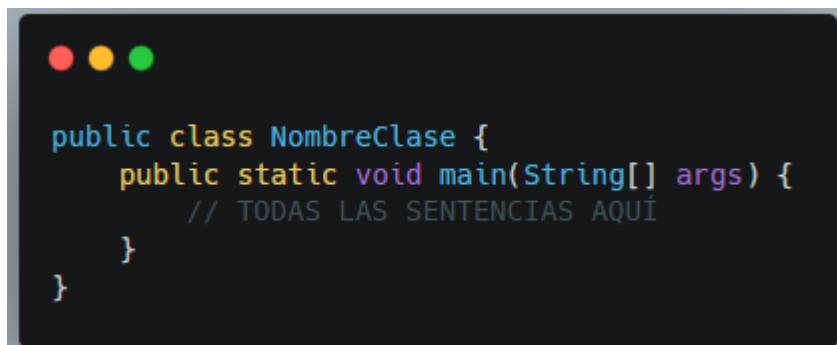
Se solicita desarrollar una aplicación denominada **JavaBridge**, cuyo propósito es traducir un subconjunto de código Java hacia Python.

### 4.1 Descripción del problema a resolver

Se debe soportar la traducción de un subconjunto específico de Java a Python:

#### A) ESTRUCTURA OBLIGATORIA DEL ARCHIVO JAVA

Formato de entrada (.java):



```
public class NombreClase {  
    public static void main(String[] args) {  
        // TODAS LAS SENTENCIAS AQUÍ  
    }  
}
```

Reglas:

- **OBLIGATORIO:** El archivo debe contener exactamente una clase pública
- **OBLIGATORIO:** La clase debe contener exactamente un método main con la firma exacta mostrada
- **OBLIGATORIO:** Todas las sentencias del programa deben estar dentro del método main
- **ERROR LÉXICO:** Si aparecen caracteres no reconocidos
- **ERROR SINTÁCTICO:** Si la estructura no coincide exactamente con el formato

#### B) TIPOS DE DATOS SOPORTADOS

Tipo Java	Tipo Python	Valor por defecto	Notas
int	int	0	Números enteros
double	float	0.0	Números decimales
char	str	' '	Un solo carácter entre comillas simples
String	str	" "	Cadenas entre comillas dobles
boolean	bool	False	true→True, false→False

Ejemplo de traducción de tipos:

## JAVA

```
// ENTRADA
int numero = 42;
double precio = 19.99;
char letra = 'A';
String mensaje = "Hola mundo";
boolean activo = true;
```

## PYTHON

```
# SALIDA
numero = 42
precio = 19.99
letra = 'A'
mensaje = "Hola mundo"
activo = True
```

## C) DECLARACIONES Y ASIGNACIONES

Declaraciones simples:

### JAVA

```
int a, b, c;           // → Variables no inicializadas (valores por defecto)
int x = 5, y = 10;     // → Variables inicializadas
```

Traducción Python:

```
# Comentario automático indicando declaraciones
a = 0 # Declaracion: int
b = 0 # Declaracion: int
c = 0 # Declaracion: int
x = 5 # Declaracion: int
y = 10 # Declaracion: int
```

Asignaciones:

JAVA

```
x = y + 5;
mensaje = "El resultado es: " + x;
```

PYTHON

```
x = y + 5
mensaje = "El resultado es: " + str(x)
```

## D) OPERACIONES ARITMÉTICAS

**Operadores soportados:** +, -, \*, /, ==, !=, >, <, >=, <=

**Precedencia (mayor a menor):**

1. \*, /
2. +, -
3. ==, !=, >, <, >=, <=

**Tabla de compatibilidad para suma (+):**

+	int	double	char	String	boolean
int	int	double	ERROR	String (concatenación)	ERROR
double	double	double	ERROR	String (concatenación)	ERROR
char	ERROR	ERROR	ERROR	String (concatenación)	ERROR
String	String	String	String	String	String
boolean	ERROR	ERROR	ERROR	String	ERROR

### Reglas de conversión automática en Python:

- $\text{int} + \text{String} \rightarrow \text{str}(\text{int}) + \text{String}$
- $\text{double} + \text{String} \rightarrow \text{str}(\text{double}) + \text{String}$
- $\text{boolean} + \text{String} \rightarrow \text{str}(\text{boolean}) + \text{String}$

### E) COMENTARIOS

#### JAVA

```
// Comentario de línea
/* Comentario
de bloque */
```

#### PYTHON

```
# Comentario de línea
'''Comentario
de bloque'''
```

### F) IMPRIMIR EN PANTALLA

#### JAVA

```
System.out.println("Hola");
System.out.println(variable);
System.out.println("El valor es: " + variable);
```

#### PYTHON

```
print("Hola")
print(str(variable))
print("El valor es: " + str(variable))
```

### G) ESTRUCTURAS DE CONTROL

#### IF-ELSE:

## JAVA

```
if (condicion) {  
    // sentencias  
} else {  
    // sentencias  
}
```

## PYTHON

```
if condicion:  
    # sentencias (indentación de 4 espacios)  
else:  
    # sentencias (indentación de 4 espacios)
```

## FOR:

### JAVA

```
for (int i = 0; i < 10; i++) {  
    // sentencias  
}  
  
for (int i = 10; i > 0; i--) {  
    // sentencias  
}
```

### PYTHON

```
i = 0  
while i < 10:  
    # sentencias (indentación de 4 espacios)  
    i += 1  
  
i = 10  
while i > 0:  
    # sentencias (indentación de 4 espacios)  
    i -= 1
```

## WHILE:

### JAVA



```
while (condicion) {  
    // sentencias  
}
```

## PYTHON

```
while condicion:  
    # sentencias (indentación de 4 espacios)
```

## 4.2 Léxico y Sintaxis

### PALABRAS RESERVADAS (case-sensitive):

public, class, static, void, main, String, args, int, double, char, boolean, true, false, if, else, for, while, System, out, println

### SÍMBOLOS:

{, }, (, ), [, ], :, ,, ., =, +, -, \*, /, ==, !=, >, <, >=, <=, ++, --

### IDENTIFICADORES:

**Patrón:** [A-Za-z\_][A-Za-z0-9\_]\*

**Ejemplos válidos:** variable, \_temp, contador1, miVariable

**Ejemplos inválidos:** 1variable, class, for

### LITERALES:

**Enteros:** 123, 0, -45

**Decimales:** 12.34, 0.0, -3.14

**Caracteres:** 'a', '1', ''

**Cadenas:** "hola", "", "texto con espacios"

**Booleanos:** true, false

### COMENTARIOS:

**Línea:** // texto hasta fin de línea

**Bloque:** /\* texto multilínea \*/

### ESPACIOS EN BLANCO:

Espacios, tabs, saltos de línea son ignorados excepto dentro de literales.

## 4.3 Gramática Libre de Contexto

```

PROGRAMA ::= 'public' 'class' ID '{' MAIN '}'
MAIN ::= 'public' 'static' 'void' 'main' '(' 'String' '[' ']' ID ')' '{' SENTENCIAS '}'
SENTENCIAS ::= SENTENCIA SENTENCIAS | ε
SENTENCIA ::= DECLARACION | ASIGNACION | IF | FOR | WHILE | PRINT | ';'
DECLARACION ::= TIPO LISTA_VARS ';'
LISTA_VARS ::= VAR_DECL (',' VAR_DECL)*
VAR_DECL ::= ID ('=' EXPRESION)?
ASIGNACION ::= ID '=' EXPRESION ';'
IF ::= 'if' '(' EXPRESION ')' '{' SENTENCIAS '}' ('else' '{' SENTENCIAS '}')?
FOR ::= 'for' '(' FOR_INIT ';' EXPRESION ';' FOR_UPDATE ')' '{' SENTENCIAS '}'
FOR_INIT ::= TIPO ID '=' EXPRESION
FOR_UPDATE ::= ID ('++' | '--')
WHILE ::= 'while' '(' EXPRESION ')' '{' SENTENCIAS '}'
PRINT ::= 'System' '.' 'out' '.' 'println' '(' EXPRESION ')' ';'
EXPRESION ::= TERMINO (('==' | '!=' | '>' | '<' | '>=' | '<=') TERMINO)*
TERMINO ::= FACTOR (('+' | '-') FACTOR)*
FACTOR ::= PRIMARIO (('*' | '/') PRIMARIO)*
PRIMARIO ::= ID | LITERAL | '(' EXPRESION ')'
TIPO ::= 'int' | 'double' | 'char' | 'String' | 'boolean'
LITERAL ::= ENTERO | DECIMAL | CARACTER | CADENA | BOOLEANO

```

## 4.4 Política de Errores

### ERRORES LÉXICOS:

1. **Carácter inesperado:** @, #, % → "Carácter no reconocido"
2. **Cadena no cerrada:** "hola → "Cadena sin cerrar"
3. **Carácter no cerrado:** 'ab → "Carácter mal formado"
4. **Número mal formado:** 12.34.56 → "Número decimal inválido"

### ERRORES SINTÁCTICOS:

1. **Estructura de clase incorrecta:** private class → "Se esperaba 'public'"

2. **Método main incorrecto:** void main() → "Se esperaba 'public static void main(String[] args)'"
3. **Falta punto y coma:** int x = 5 → "Se esperaba ','"
4. **Llave no balanceada:** if (x > 0) { int y = 5 → "Se esperaba '}'"
5. **Tipo no reconocido:** float x; → "Tipo de dato no soportado"
6. **Variable no declarada:** x = 5; (sin declarar x) → "Variable no declarada"

## 4.5 Ejemplos Archivos

### ENTRADA (.java)

```
public class MiPrograma {
    public static void main(String[] args) {
        // Declaraciones
        int contador = 0;
        String mensaje = "Inicio del programa";
        boolean activo = true;

        System.out.println(mensaje);

        // Ciclo for
        for (int i = 1; i <= 5; i++) {
            System.out.println("Iteracion: " + i);
            contador = contador + i;
        }

        // Condicional
        if (contador > 10) {
            System.out.println("Contador mayor a 10: " + contador);
        } else {
            System.out.println("Contador menor o igual a 10");
        }

        // Ciclo while
        while (activo) {
            System.out.println("Ejecutando while");
            activo = false;
        }
    }
}
```

## SALIDA (.py):

```
# Traducido de Java a Python
# Clase: MiPrograma

# Declaraciones
contador = 0 # Declaracion: int
mensaje = "Inicio del programa" # Declaracion: String
activo = True # Declaracion: boolean

print(str(mensaje))

# Ciclo for traducido a while
i = 1
while i <= 5:
    print("Iteracion: " + str(i))
    contador = contador + i
    i += 1

# Condicional
if contador > 10:
    print("Contador mayor a 10: " + str(contador))
else:
    print("Contador menor o igual a 10")

# Ciclo while
while activo:
    print("Ejecutando while")
    activo = False
```

## 5 Alcance del proyecto

### Interfaz Gráfica Web

#### Componentes obligatorios:

1. **Área de texto principal:** Editor para código
2. **Área de texto secundaria:** Muestra código Python traducido

#### Menús obligatorios:

##### ARCHIVO:

- **Nuevo:** Limpia el editor
- **Abrir:** Carga archivo .java
- **Guardar:** Guarda código Java actual
- **Guardar Python como:** Guarda código Python traducido (solo si traducción exitosa)
- **Salir:** Cierra aplicación

##### TRADUCIR:

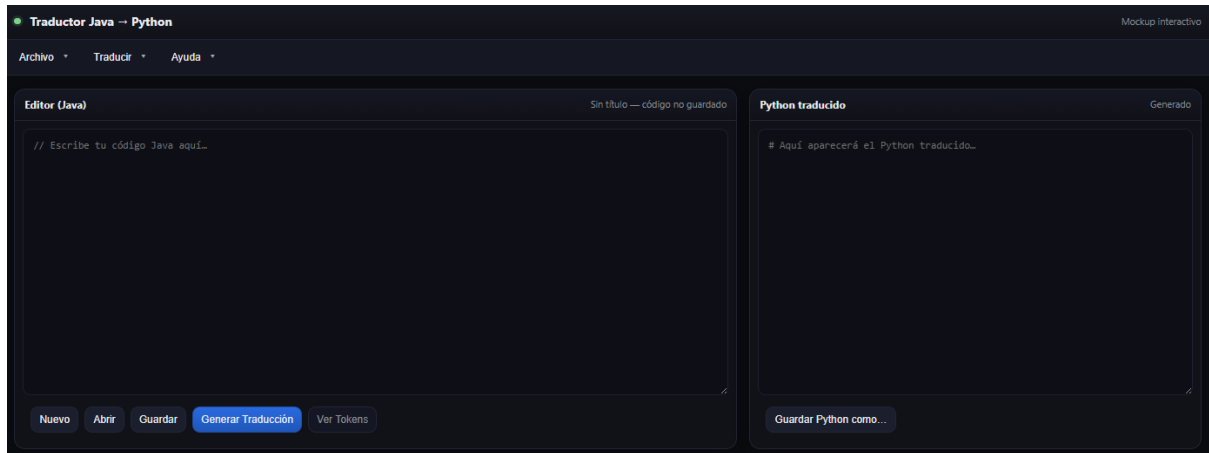
- **Generar Traducción:** Ejecuta análisis léxico → sintáctico → traducción
- **Ver Tokens:** Muestra reporte HTML de tokens

- **Simular Ejecución:** Ejecuta el código Python traducido paso a paso

### AYUDA:

- **Acerca de:** Información del desarrollador

Ejemplo de Interfaz:



**Tokens generados** 128

No.	Lexema	Tipo	Línea	Columna
1	public	PALABRA_RESERVADA	1	1
2	class	PALABRA_RESERVADA	1	8
3	MiPrograma	IDENTIFICADOR	1	14
4	{	LLAVE_ABRE	1	25
5	public	PALABRA_RESERVADA	2	5
6	static	PALABRA_RESERVADA	2	12
7	void	PALABRA_RESERVADA	2	19
8	main	IDENTIFICADOR	2	24
9	(	PAR_ABRE	2	28
10	String	PALABRA_RESERVADA	2	29
11	[	CUB_ABRE	2	35
12	1	CUB_CIERRA	2	36

Descargar HTML Cerrar

## 6 Reportes HTML

### REPORTE DE ERRORES LÉXICOS:

No.	Error	Descripción	Línea	Columna
1	@	Carácter no reconocido	5	10
2	"hola	Cadena sin cerrar	7	15

### REPORTE DE ERRORES SINTÁCTICOS:

No.	Error	Descripción	Línea	Columna
1	}	Se esperaba ';'	12	25

### REPORTE DE TOKENS:

No.	Lexema	Tipo	Línea	Columna
1	public	PALABRA_RESERVADA	1	1
2	class	PALABRA_RESERVADA	1	8
3	MiPrograma	IDENTIFICADOR	1	14

### Requerimientos técnicos

**Prohibido el uso de expresiones regulares:** No está permitido utilizar expresiones regulares para la manipulación y procesamiento de los datos en los archivos de entrada. Los estudiantes deben de implementar sus propias funciones para identificar y manipular tokens o caracteres.

## Entregables

Ejemplo:

Tipo	Descripción
<b>Manual de Usuario</b>	Documento que explica cómo usar el sistema desarrollado, incluyendo capturas de pantalla, pasos detallados y resolución de problemas comunes.
<b>Manual Técnico</b>	Un documento detallado que describe el proceso de desarrollo, análisis de resultados, pruebas realizadas, tabla de token con su respectivo patrón (Expresión Regular) del analizador léxico utilizado en la solución, AFD generado por medio del método del árbol(analizador léxico), gramática tipo 2 (notación BNF) del analizador sintáctico utilizado en la solución y conclusiones obtenidas.
<b>Código Fuente</b>	El código completo del proyecto, con una estructura clara y comentada, que refleja las buenas prácticas de programación y uso de control de versiones.
<b>Diagrama de Flujo</b>	Representación gráfica del flujo de trabajo o funcionamiento del sistema, que permite comprender el recorrido de los datos o acciones dentro del proyecto.

## Metodología

Para el desarrollo del proyecto JavaBridge, se propone una metodología en fases basada en el enfoque cascada, adaptado a proyectos individuales y académicos:

1. **Análisis del Lenguaje y Requisitos**  
Estudio de la estructura sintáctica del lenguaje definido para describir modelos entidad-relación, identificando palabras clave, símbolos, reglas de agrupación y posibles errores léxicos.

2. Diseño del AFD y Diagrama del Sistema  
Diseño del autómata finito determinista que servirá como base del analizador léxico. Se desarrollarán diagramas de flujo para representar el procesamiento de tokens y errores.
3. Pruebas, Corrección y Documentación  
Ejecución de casos de prueba válidos e inválidos. Registro de errores léxicos y verificación de resultados. Redacción del manual técnico y del manual de usuario.

## Desarrollo de Habilidades Blandas

### 6.1 Proyectos Individuales

El trabajo individual en JavaBridge promueve la **autonomía y la toma de decisiones técnicas**, exigiendo al estudiante asumir la totalidad del ciclo de vida del proyecto.

#### 6.2.1 Autogestión del Tiempo

El estudiante planifica y organiza su cronograma personal para cumplir con las fases del desarrollo, entrega y calificación, lo que fortalece su capacidad de organización y responsabilidad.

#### 6.2.2 Responsabilidad y Compromiso

Al ser el único responsable de su producto, el estudiante asume compromisos técnicos y éticos. Esto lo obliga a cumplir estándares de calidad, puntualidad y claridad.

#### 6.2.3 Resolución de Problemas

La implementación manual del analizador léxico, sin herramientas externas, fomenta la búsqueda de soluciones lógicas, estructuradas y autónomas ante desafíos de diseño, sintaxis o integración.

#### 6.2.4 Reflexión Personal

Al finalizar el proyecto, el estudiante evalúa su desempeño, identifica fortalezas y áreas de mejora. Este proceso de reflexión fortalece su capacidad de autoaprendizaje continuo.

## Cronograma

El cronograma describe las etapas clave del proyecto, los plazos estimados para cada una, y el proceso de asignación, elaboración y calificación de las tareas. Los estudiantes deberán seguir este plan para asegurar que el proyecto avance de manera organizada y cumpla con los plazos establecidos. Cada fase incluye la asignación de tareas, el tiempo estimado para su elaboración, y el momento de su calificación.



Tipo	Fecha Inicio	Fecha Fin
Asignación de Proyecto	21/09/2025	25/10/2025

## Valores

Durante el desarrollo del proyecto, se espera que cada estudiante actúe con integridad, compromiso y profesionalismo. Para garantizar una evaluación justa y una experiencia de aprendizaje significativa, se establecen los siguientes principios fundamentales:

### 1. Originalidad del Trabajo

- Cada estudiante debe desarrollar su propio proyecto, partiendo de cero y aplicando los conocimientos adquiridos durante el curso.
- Tanto el código fuente como la documentación entregada deben ser resultado del trabajo personal e individual, conforme a los lineamientos establecidos por el curso.

### 2. Prohibición de Copias y Plagio

- **Se prohíbe estrictamente** la copia total o parcial del código, documentación o cualquier otro entregable de:
  - Compañeros del curso.
  - Semestres anteriores.
  - Fuentes externas no referenciadas.
- Cualquier evidencia de plagio resultará en una **calificación de 0 puntos**, sin excepción.

### 3. Uso Responsable de Recursos Externos

- Está permitido utilizar bibliotecas o recursos externos únicamente cuando:
  - Se mencionan claramente sus fuentes.
  - Se comprende completamente su funcionamiento.
  - **No se usan herramientas automáticas** para generar analizadores léxicos (como ANTLR, JFlex, Lex o similares), conforme a lo indicado en las restricciones del proyecto.

- Se recomienda consultar con el catedrático si existe duda sobre la validez del uso de algún recurso.

#### 4. Revisión y Detección de Plagio

- El proyecto podrá ser sometido a herramientas automáticas de detección de plagio y a revisiones manuales por parte del equipo docente.
- En caso de sospecha:
  - El estudiante deberá explicar y defender el funcionamiento de su código y demostrar su autoría.
  - Si no logra justificar su trabajo adecuadamente, se **anulará su proyecto y se asignará una calificación de 0 puntos**.
- Cualquier irregularidad será reportada al catedrático para que determine las acciones correspondientes, las cuales pueden incluir **notificación a la Escuela de Ciencias y Sistemas**.

#### Restricciones

- El lenguaje de programación a utilizar es JavaScript.
- Queda prohibida cualquier librería que facilite el análisis de la entrada de texto.
- El nombre del repositorio debe tener el siguiente formato LFP\_2S2025\_#Carnet, dentro del repositorio deberá crear una carpeta llamada Proyecto 2.
- Se debe agregar como colaborador al auxiliar de la sección correspondiente.
  - **A+ bryan967132**
  - **A- ATPCOXBAU**
  - **B+ kevinmp2**
  - **B- riverroman**
- Link al repositorio de GitHub en el apartado de UEDI.
- Fecha de entrega: 25 de Octubre de 2025 antes de las 23:59, no se recibirán entregas después de la fecha y hora límite establecidas.