Documentación Completa del Proyecto: Traductor de Python a JavaScript

1. Información General del Proyecto

1.1 Título del Proyecto

Traductor de lenguajes de programación: Python 3 a JavaScript ES6 con interfaz Streamlit

1.2 Equipo de Desarrollo

- **Luis Bremer** - T032366
- **Hiram Zuniga** - T036522

1.3 Institución

Centro de Enseñanza Técnica y Superior
Ingeniería en Ciencias Computacionales
Administración de la Tecnología de la Información
Tijuana B.C., 06 de Noviembre del 2025

1.4 Descripción Ejecutiva

El proyecto implementa un traductor completo de Python 3 a JavaScript ES6, siguiendo el pipeline completo de un compilador: análisis léxico, análisis sintáctico, construcción de AST, análisis semántico y generación de código. Incluye una interfaz web desarrollada en Streamlit para facilitar la interacción del usuario.

2. Objetivos del Proyecto

2.1 Objetivo General

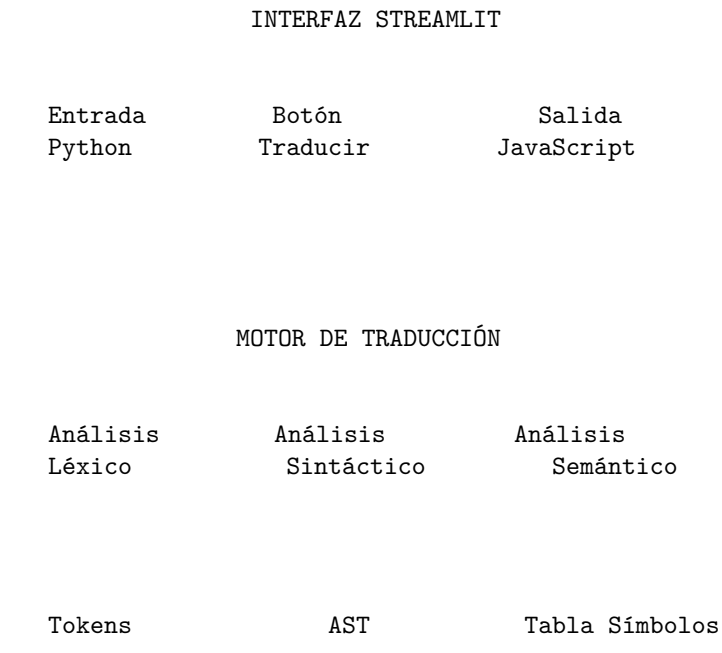
Desarrollar un traductor automático que convierta código Python 3 válido a JavaScript ES6 equivalente, implementando todas las fases de compilación y proporcionando una interfaz web intuitiva.

2.2 Objetivos Específicos

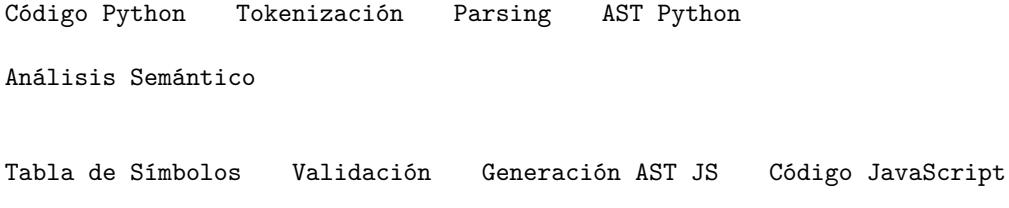
- Implementar un analizador léxico completo para Python 3
- Desarrollar un parser basado en la gramática oficial de Python 3
- Construir un generador de AST (Árbol de Sintaxis Abstracta)
- Implementar análisis semántico con tabla de símbolos
- Crear un generador de código JavaScript ES6
- Desarrollar una interfaz web con Streamlit
- Proporcionar diagnósticos detallados de errores

3. Arquitectura del Sistema

3.1 Componentes Principales



3.2 Flujo de Datos



4. Especificaciones Técnicas

4.1 Lenguajes y Tecnologías

Lenguaje Fuente: Python 3 Construcciones Soportadas: - **Sentencias:** asignaciones, if/elif/else, while, for, def, class, try/except, with, import - **Expresiones:** operadores aritméticos, lógicos, comparación, llamadas a funciones, indexación, comprensiones - **Tipos de datos:** int, float, str, list, dict, tuple, bool - **Estructuras de control:** condicionales, bucles, manejo de excepciones

Lenguaje Destino: JavaScript ES6 Mapeo de Construcciones: - `def función()` → `function función()` o `const función = () =>` - `class MiClase` → `class MiClase` - `print()` → `console.log()` - `if/elif/else` → `if/else if/else` - `for item in lista` → `for (let item of lista)` - `try/except` → `try/catch`

4.2 Gramática Considerada

Python 3 (Subconjunto Principal)

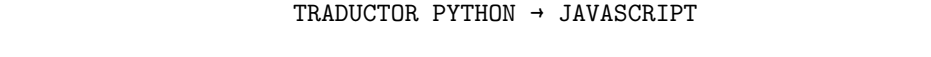
```
programa: sentencia*
sentencia: asignacion | if_stmt | while_stmt | for_stmt | def_stmt | class_stmt | expr_stmt
asignacion: NOMBRE '=' expresion
if_stmt: 'if' expresion ':' bloque ('elif' expresion ':' bloque)* ('else' ':' bloque)?
while_stmt: 'while' expresion ':' bloque
for_stmt: 'for' NOMBRE 'in' expresion ':' bloque
def_stmt: 'def' NOMBRE '(' parametros? ')' ':' bloque
class_stmt: 'class' NOMBRE ('(' NOMBRE ')')? ':' bloque
expresion: termino (('+'|'-') termino)*
termino: factor (('*'|'/'|'%') factor)*
factor: NUMERO | CADENA | NOMBRE | '(' expresion ')' | llamada_funcion
```

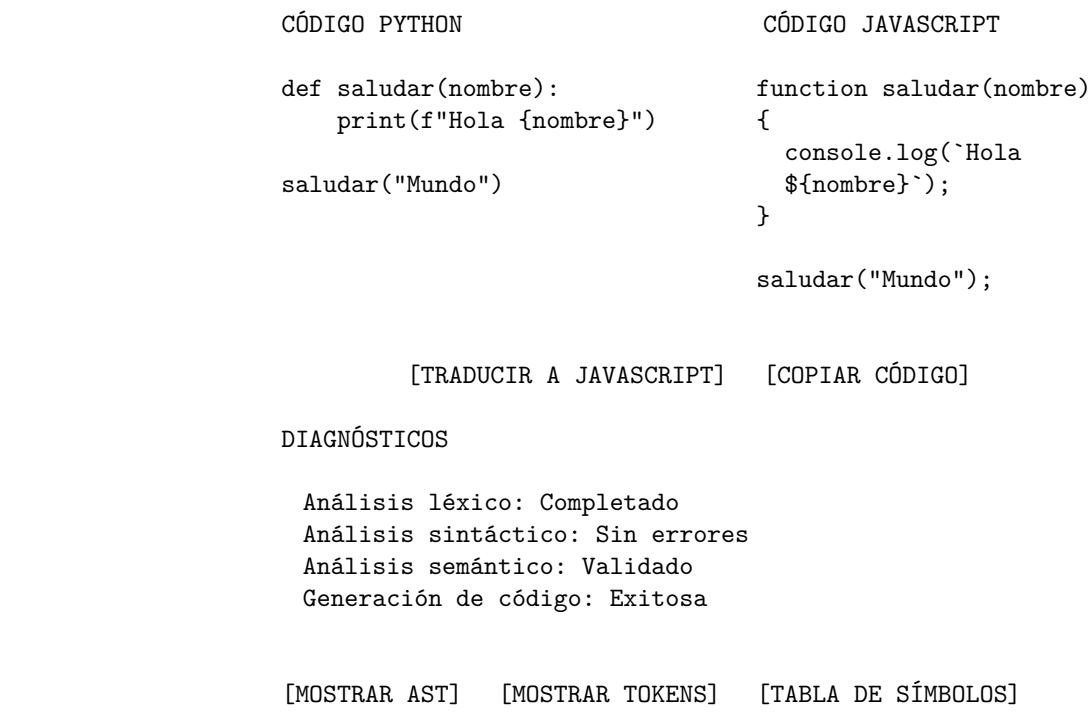
JavaScript ES6 (Salida)

```
programa: declaracion*
declaracion: var_decl | func_decl | class_decl | expr_stmt
var_decl: ('let'|'const'|'var') NOMBRE '=' expresion ';'
func_decl: 'function' NOMBRE '(' parametros? ')' '{' bloque '}'
class_decl: 'class' NOMBRE ('extends' NOMBRE)? '{' metodos* '}'
```

5. Diseño de la Interfaz

5.1 Mockup de la Interfaz Streamlit





5.2 Componentes de la Interfaz

1. **Área de Entrada:** Editor de texto para código Python
2. **Área de Salida:** Visualización del código JavaScript generado
3. **Panel de Control:** Botones para traducir y copiar código
4. **Panel de Diagnósticos:** Estado del proceso de traducción
5. **Herramientas de Debug:** AST, tokens, tabla de símbolos

6. Análisis y Verificación

6.1 Proceso de Análisis Léxico

```python
class AnalizadorLexico:
    def __init__(self, codigo_fuente):
        self.codigo = codigo_fuente
        self.posicion = 0
        self.tokens = []

    def tokenizar(self):
        # Reconoce palabras clave, identificadores, operadores, literales
        pass
```

```
    def siguiente_token(self):
        # Extrae el próximo token del código fuente
        pass
```

6.2 Proceso de Análisis Sintáctico

```
class Parser:
    def __init__(self, tokens):
        self.tokens = tokens
        self.posicion = 0
        self.ast = None

    def parsear_programa(self):
        # Construye el AST siguiendo la gramática de Python
        pass

    def parsear_sentencia(self):
        # Analiza diferentes tipos de sentencias
        pass
```

6.3 Análisis Semántico

```
class AnalizadorSemantico:
    def __init__(self):
        self.tabla_simbolos = {}
        self.errores = []

    def verificar_variables(self, ast):
        # Verifica uso de variables declaradas
        pass

    def verificar_funciones(self, ast):
        # Valida llamadas a funciones
        pass
```

7. Generación de Código

7.1 Mapeo de Construcciones

Python	JavaScript ES6
`def func():`	`function func() {`
`class Clase:`	`class Clase {`
`print(x)`	`console.log(x)`
`if condition:`	`if (condition) {`
`for x in lista:`	`for (let x of lista) {`
`try: ... except:`	`try { ... } catch {`

Python	JavaScript ES6
`x = 5`	`let x = 5;`
`"string"`	`"string"`
`[1, 2, 3]`	`[1, 2, 3]`
`{"a": 1}`	`{"a": 1}`

7.2 Generador de Código

```python
class GeneradorJS:
    def __init__(self, ast_python):
        self.ast = ast_python
        self.codigo_js = ""

    def generar(self):
        # Recorre el AST y genera código JavaScript
        pass

    def visitar_funcion(self, nodo):
        # Convierte función Python a JavaScript
        pass

    def visitar_clase(self, nodo):
        # Convierte clase Python a JavaScript
        pass
```

8. Manejo de Errores

8.1 Tipos de Errores

Errores Léxicos

- Caracteres no reconocidos
- Cadenas sin cerrar
- Números mal formados

Errores Sintácticos

- Indentación incorrecta
- Paréntesis no balanceados
- Palabras clave mal ubicadas

Errores Semánticos

- Variables no declaradas
- Funciones con argumentos incorrectos
- Tipos incompatibles

8.2 Formato de Mensajes de Error

```
Error en línea 5, columna 12:
  Variable 'x' no está definida

  Código:
    4 | def calcular():
  > 5 |     return x + 1
        |            ^
    6 |

  Sugerencia: Declare la variable 'x' antes de usarla
```

9. Casos de Uso

9.1 Caso de Uso Principal: Traducción Exitosa

Actor: Usuario (Profesor/Estudiante)
Precondición: Código Python válido
Flujo Principal: 1. Usuario ingresa código Python en el área de texto 2. Usuario presiona "Traducir a JavaScript" 3. Sistema ejecuta análisis léxico, sintáctico y semántico 4. Sistema genera código JavaScript equivalente 5. Sistema muestra el resultado en el área de salida

9.2 Caso de Uso Alternativo: Error de Sintaxis

Flujo Alternativo: 1. Usuario ingresa código Python con errores 2. Usuario presiona "Traducir a JavaScript" 3. Sistema detecta error en análisis sintáctico 4. Sistema muestra mensaje de error detallado 5. Usuario corrige el código y reintenta

9.3 Caso de Uso: Visualización de AST

Flujo: 1. Usuario completa traducción exitosa 2. Usuario presiona "Mostrar AST" 3. Sistema muestra representación del árbol sintáctico 4. Usuario puede analizar la estructura del código

10. Estructura del Proyecto

```
traductor_python_js/
  src/
      analizador_lexico.py
      parser.py
      ast_nodes.py
      analizador_semantico.py
      generador_js.py
      tabla_simbolos.py
```

```
interfaz/
    app_streamlit.py
tests/
    test_lexico.py
    test_parser.py
    test_semantico.py
    test_generador.py
ejemplos/
    ejemplo_basico.py
    ejemplo_clases.py
    ejemplo_funciones.py
docs/
    manual_usuario.md
    manual_tecnico.md
    ejemplos_traduccion.md
requirements.txt
README.md
main.py
```

11. Cronograma de Desarrollo

Fase 1: Análisis Léxico y Sintáctico (Semanas 1-2)

- Implementar tokenizador
- Desarrollar parser básico
- Crear estructura de AST

Fase 2: Análisis Semántico (Semana 3)

- Implementar tabla de símbolos
- Desarrollar verificaciones semánticas
- Crear sistema de errores

Fase 3: Generación de Código (Semana 4)

- Implementar generador JavaScript
- Crear mapeos de construcciones
- Optimizar salida de código

Fase 4: Interfaz y Testing (Semana 5)

- Desarrollar interfaz Streamlit
- Implementar casos de prueba
- Documentación final

12. Criterios de Aceptación

12.1 Funcionalidad Mínima

- Traducir funciones básicas de Python a JavaScript
- Manejar estructuras de control (if, while, for)
- Procesar tipos de datos básicos
- Detectar y reportar errores sintácticos
- Interfaz web funcional

12.2 Funcionalidad Extendida

- Traducir clases y herencia
- Manejar excepciones (try/except)
- Comprensiones de listas
- Análisis semántico completo
- Visualización de AST

13. Limitaciones y Consideraciones

13.1 Limitaciones Técnicas

- No se traducen todas las librerías de Python
- Algunas construcciones específicas de Python no tienen equivalente directo
- El código JavaScript generado puede requerir ajustes manuales

13.2 Consideraciones de Rendimiento

- Optimizado para archivos de código de tamaño medio
- Tiempo de traducción proporcional a la complejidad del código
- Uso eficiente de memoria para AST

14. Conclusiones

Este proyecto implementa un traductor completo que demuestra los principios fundamentales de compiladores aplicados a la traducción entre lenguajes de programación. La interfaz web facilita su uso educativo y práctico, mientras que el diseño modular permite futuras extensiones y mejoras.

El sistema proporciona una herramienta valiosa para: - Aprender conceptos de compiladores - Migrar código entre lenguajes - Entender equivalencias entre Python y JavaScript - Analizar estructura de programas mediante AST

Documento generado: Noviembre 2025
Versión: 1.0
Autores: Luis Bremer, Hiram Zuniga