ENTREGABLE 6: Código Completo Comentado con Ejecuciones

Proyecto: Analizador de Lenguaje Natural Simple
Autores: Ricardo Méndez, Emiliano Ledesma, Diego Jiménez, Abraham Velázquez
Fecha: Noviembre 2024

Índice

---

1. Introducción

Este documento presenta el código completo del analizador de oraciones (versión simplificada), fragmentado en secciones con explicaciones detalladas de cada componente. Además, se incluyen múltiples ejecuciones simuladas mostrando el comportamiento del sistema paso a paso.

Archivo analizado: `version_simplificada.py`

Características del código:

- 373 líneas de código Python
- 2 clases principales: `AnalizadorSimple` y `AFDSimple`
- Sin dependencias externas (solo módulos estándar)
- Implementación directa del modelo teórico AFD

---

2. Estructura General del Código

Organización del Archivo

```
"""
VERSIÓN SIMPLIFICADA DEL ANALIZADOR DE ORACIONES
Proyecto 5: Analizador de lenguaje natural simple
"""

# Importaciones (líneas 18-19)
```

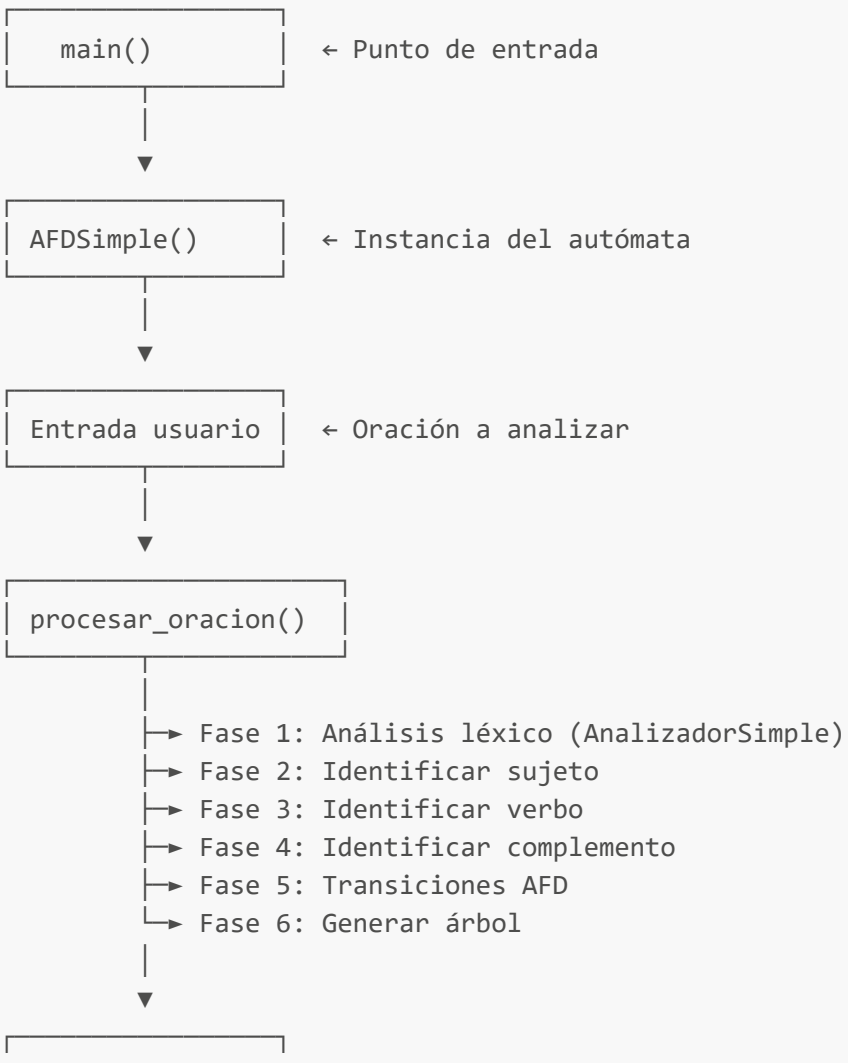
```python
import re
from datetime import datetime

# Clase 1: AnalizadorSimple (líneas 22-132)
class AnalizadorSimple:
    # Vocabulario limitado
    # Método de análisis léxico

# Clase 2: AFDSimple (líneas 135-308)
class AFDSimple:
    # Definición formal del AFD
    # Procesamiento de oraciones
    # Generación de árboles

# Función principal (líneas 311-373)
def main():
    # Interfaz de usuario
    # Loop interactivo
```

Diagrama de Flujo de Ejecución



```
| Mostrar salida  |
└─────────────────┘
```

---

3. Clase AnalizadorSimple - Fragmentado y Explicado

Fragmento 1: Importaciones y Encabezado de la Clase

```python
import re
from datetime import datetime


class AnalizadorSimple:
    """
    Analizador simple basado en diccionarios de palabras.
    Usa un alfabeto limitado para demostrar el concepto de lenguaje formal.
    """
```

Explicación:

- `re`: Módulo de expresiones regulares (no usado actualmente, reservado para extensiones)
- `datetime`: Para marcas de tiempo (no usado actualmente)
- La clase encapsula el análisis léxico usando diccionarios predefinidos

Propósito: Esta clase implementa el reconocimiento de palabras del vocabulario limitado.

Fragmento 2: Constructor e Inicialización del Vocabulario

```python
def __init__(self):
    # ALFABETO DEL LENGUAJE (vocabulario limitado)
    self.determinantes = ['el', 'la', 'un', 'una', 'los', 'las', 'mi', 'tu']
    self.sustantivos = ['gato', 'perro', 'niño', 'niña', 'libro', 'parque',
                        'pescado', 'jardín', 'casa', 'María', 'Juan', 'hermano',
                        'matemáticas', 'niños']
    self.verbos = ['come', 'corre', 'estudia', 'lee', 'camino', 'juega',
                   'juegan', 'escribe', 'canta']
    self.pronombres = ['yo', 'tú', 'él', 'ella', 'nosotros']
    self.preposiciones = ['por', 'en', 'de', 'con', 'a']
    self.adverbios = ['rápidamente', 'bien', 'mal', 'rápido']
```

Explicación:

- Define el **alfabeto terminal** del lenguaje formal
- Total de 45 palabras organizadas en 6 categorías gramaticales
- Las listas son mutables, permitiendo extensiones futuras

Complejidad Espacial: O(1) - constante (vocabulario fijo)

Ejemplo de uso:

```
analizador = AnalizadorSimple()
print(len(analizador.determinantes))  # Output: 8
print('gato' in analizador.sustantivos)  # Output: True
```

Fragmento 3: Firma del Método Principal

```
def analizar_oracion(self, oracion: str) -> dict:
    """
    Analiza una oración simple y extrae sus componentes.

    Gramática reconocida:
    S  → SN SV
    SN → DET N | PRON | N
    SV → V SN | V SP | V
    SP → PREP SN
    """
```

Explicación:

- Método público principal de la clase
- **Entrada:** Cadena de texto (oración)
- **Salida:** Diccionario con estructura analizada
- La docstring documenta la gramática formal implementada

Gramática en notación BNF:

```
<S>  ::= <SN> <SV>
<SN> ::= <DET> <N> | <PRON> | <N>
<SV> ::= <V> <SN> | <V> <SP> | <V> <ADV> | <V>
<SP> ::= <PREP> <SN>
```

Fragmento 4: Inicialización del Resultado y Tokenización

```
palabras = oracion.lower().strip().split()

resultado = {
    'valida': False,
    'sujeto': None,
```

```
    'verbo': None,
    'predicado': None,
    'estructura': []
}

if len(palabras) == 0:
    return resultado

i = 0  # Índice para recorrer las palabras
```

Explicación:

- `lower()`: Normaliza a minúsculas (case-insensitive)
- `strip()`: Elimina espacios al inicio/final
- `split()`: Tokeniza por espacios en blanco
- `resultado`: Estructura de datos con los componentes extraídos
- `i`: Cursor para parseo secuencial

Complejidad: O(n) donde n = longitud de la cadena

Simulación de Ejecución:

```
# Input: "El Gato COME pescado  "
oracion = "El Gato COME pescado  "
palabras = oracion.lower().strip().split()
# palabras = ['el', 'gato', 'come', 'pescado']
# len(palabras) = 4
# i = 0
```

Fragmento 5: Identificación del Sujeto - Opción 1 (DET + N)

```
# PASO 1: Identificar SUJETO (SN)
sujeto_palabras = []

# Opción 1: DET + N (ej: "el gato")
if i < len(palabras) and palabras[i] in self.determinantes:
    sujeto_palabras.append(palabras[i])
    resultado['estructura'].append(('DET', palabras[i]))
    i += 1

    if i < len(palabras) and palabras[i] in self.sustantivos:
        sujeto_palabras.append(palabras[i])
        resultado['estructura'].append(('N', palabras[i]))
        i += 1
```

Explicación:

- Implementa la regla: **SN → DET N**
- Verifica límites del array con `i < len(palabras)`
- Búsqueda en lista: O(n) pero n es pequeño (8 determinantes)
- Almacena tuplas (categoría, palabra) en `estructura`
- Avanza el cursor `i` tras reconocer cada componente

Simulación Paso a Paso:

```
# Estado inicial
i = 0
palabras = ['el', 'gato', 'come', 'pescado']
sujeto_palabras = []

# Paso 1: Verificar palabras[0] = 'el'
palabras[0] in self.determinantes  # True
sujeto_palabras.append('el')       # ['el']
resultado['estructura'].append(('DET', 'el'))
i = 1

# Paso 2: Verificar palabras[1] = 'gato'
palabras[1] in self.sustantivos    # True
sujeto_palabras.append('gato')     # ['el', 'gato']
resultado['estructura'].append(('N', 'gato'))
i = 2

# Resultado parcial
# i = 2
# sujeto_palabras = ['el', 'gato']
# estructura = [('DET', 'el'), ('N', 'gato')]
```

Fragmento 6: Identificación del Sujeto - Opciones 2 y 3

```
# Opción 2: PRON (ej: "yo")
elif i < len(palabras) and palabras[i] in self.pronombres:
    sujeto_palabras.append(palabras[i])
    resultado['estructura'].append(('PRON', palabras[i]))
    i += 1

# Opción 3: N solo (ej: "María")
elif i < len(palabras) and palabras[i] in self.sustantivos:
    sujeto_palabras.append(palabras[i])
    resultado['estructura'].append(('N', palabras[i]))
    i += 1

if sujeto_palabras:
    resultado['sujeto'] = ' '.join(sujeto_palabras)
```

Explicación:

- Estructura `if-elif-elif`: Solo una opción se ejecuta (mutuamente excluyentes)
- Implementa las reglas: **SN → PRON** y **SN → N**
- `join()`: Reconstruye la cadena del sujeto con espacios

Ejemplo con Pronombre:

```
# Input: "yo camino"
palabras = ['yo', 'camino']
i = 0

# 'yo' no está en determinantes → salta opción 1
# 'yo' está en pronombres → ejecuta opción 2
sujeto_palabras = ['yo']
resultado['estructura'] = [('PRON', 'yo')]
resultado['sujeto'] = 'yo'
i = 1
```

Ejemplo con Nombre Propio:

```
# Input: "María estudia"
palabras = ['maría', 'estudia']
i = 0

# 'maría' no está en determinantes → salta opción 1
# 'maría' no está en pronombres → salta opción 2
# 'maría' está en sustantivos → ejecuta opción 3
sujeto_palabras = ['maría']
resultado['estructura'] = [('N', 'maría')]
resultado['sujeto'] = 'maría'
i = 1
```

Fragmento 7: Identificación del Verbo

```
# PASO 2: Identificar VERBO (V)
if i < len(palabras) and palabras[i] in self.verbos:
    resultado['verbo'] = palabras[i]
    resultado['estructura'].append(('V', palabras[i]))
    predicado_inicio = i
    i += 1
```

Explicación:

- Verifica si la palabra actual está en el diccionario de verbos

- `predicado_inicio`: Guarda la posición donde empieza el predicado (para uso posterior)
- Solo avanza si encuentra un verbo válido

Simulación:

```
# Continuando ejemplo anterior: "el gato come pescado"
# Estado: i = 2, palabras = ['el', 'gato', 'come', 'pescado']

palabras[2] in self.verbos  # 'come' → True
resultado['verbo'] = 'come'
resultado['estructura'].append(('V', 'come'))
predicado_inicio = 2
i = 3

# Estado actualizado
# i = 3
# resultado['verbo'] = 'come'
# estructura = [('DET', 'el'), ('N', 'gato'), ('V', 'come')]
```

Fragmento 8: Identificación del Complemento

```
# PASO 3: Identificar COMPLEMENTO (resto del SV)
complemento_palabras = [resultado['verbo']]

# Puede haber preposición + sintagma
while i < len(palabras):
    palabra = palabras[i]

    if palabra in self.preposiciones:
        complemento_palabras.append(palabra)
        resultado['estructura'].append(('PREP', palabra))
    elif palabra in self.determinantes:
        complemento_palabras.append(palabra)
        resultado['estructura'].append(('DET', palabra))
    elif palabra in self.sustantivos:
        complemento_palabras.append(palabra)
        resultado['estructura'].append(('N', palabra))
    elif palabra in self.adverbios:
        complemento_palabras.append(palabra)
        resultado['estructura'].append(('ADV', palabra))
    else:
        # Palabra no reconocida
        complemento_palabras.append(palabra)
        resultado['estructura'].append(('?', palabra))

    i += 1

resultado['predicado'] = ' '.join(complemento_palabras)
```

Explicación:

- Loop `while` procesa todas las palabras restantes
- Implementa las reglas: **SV → V SN | V SP | V ADV**
- `complemento_palabras` incluye el verbo + todos los complementos
- Palabras no reconocidas se marcan con '?'

Simulación Ejemplo Simple:

```
# Continuando: "el gato come pescado"
# Estado: i = 3, palabras = ['el', 'gato', 'come', 'pescado']
complemento_palabras = ['come']

# Iteración 1: i = 3
palabra = 'pescado'
palabra in self.sustantivos  # True
complemento_palabras.append('pescado')  # ['come', 'pescado']
resultado['estructura'].append(('N', 'pescado'))
i = 4

# Fin del loop (i >= len(palabras))
resultado['predicado'] = 'come pescado'

# Estado final
# complemento_palabras = ['come', 'pescado']
# estructura completa = [('DET', 'el'), ('N', 'gato'), ('V', 'come'), ('N',
'pescado')]
```

Simulación Ejemplo con Sintagma Preposicional:

```
# Input: "yo camino por el parque"
# Estado al llegar aquí: i = 2, palabras = ['yo', 'camino', 'por', 'el',
'parque']
complemento_palabras = ['camino']

# Iteración 1: i = 2
palabra = 'por'
palabra in self.preposiciones  # True
complemento_palabras = ['camino', 'por']
i = 3

# Iteración 2: i = 3
palabra = 'el'
palabra in self.determinantes  # True
complemento_palabras = ['camino', 'por', 'el']
i = 4

# Iteración 3: i = 4
```

```
palabra = 'parque'
palabra in self.sustantivos  # True
complemento_palabras = ['camino', 'por', 'el', 'parque']
i = 5

# Resultado
resultado['predicado'] = 'camino por el parque'
```

Fragmento 9: Validación Final

```
# Validar: debe tener al menos sujeto y verbo
if resultado['sujeto'] and resultado['verbo']:
    resultado['valida'] = True

return resultado
```

Explicación:

- Una oración es válida si tiene **sujeto Y verbo**
- El complemento es opcional (verbos intransitivos)
- Retorna el diccionario con todos los componentes extraídos

Estructura del Diccionario Retornado:

```
{
    'valida': True,
    'sujeto': 'el gato',
    'verbo': 'come',
    'predicado': 'come pescado',
    'estructura': [
        ('DET', 'el'),
        ('N', 'gato'),
        ('V', 'come'),
        ('N', 'pescado')
    ]
}
```

---

4. Clase AFDSimple - Fragmentado y Explicado

Fragmento 10: Definición de la Clase AFD

```
class AFDSimple:
    """
    Autómata Finito Determinista para validar oraciones.
```

```
    Estados:
    - q0: Estado inicial
    - q1: Sujeto identificado
    - q2: Verbo identificado
    - q3: Predicado completo (ESTADO DE ACEPTACIÓN)
    - qr: Estado de rechazo

    Transiciones:
    δ(q0, SN) = q1
    δ(q1, V)  = q2
    δ(q2, complemento) = q3
    """
```

Explicación:

- Documenta formalmente la definición del AFD
- 5 estados: {q0, q1, q2, q3, qr}
- Estado inicial: q0
- Estado de aceptación: q3
- Las transiciones se implementan en el código

Fragmento 11: Constructor del AFD

```python
def __init__(self):
    self.analizador = AnalizadorSimple()

    # Definición formal del AFD
    self.Q = ['q0', 'q1', 'q2', 'q3', 'qr']  # Conjunto de estados
    self.q0 = 'q0'                            # Estado inicial
    self.F = ['q3']                           # Estados de aceptación

    # Tabla de transiciones
    self.delta = {
        ('q0', 'SN'): 'q1',
        ('q1', 'V'):  'q2',
        ('q2', 'COMPLEMENTO'): 'q3',
    }

    self.estado_actual = self.q0
    self.historial = []
```

Explicación:

- Crea una instancia del analizador léxico
- Define formalmente la 5-tupla M = (Q, Σ, δ, q_0, F)
- `delta`: Diccionario que representa la función de transición

- `estado_actual`: Variable de estado del autómata
- `historial`: Registro de transiciones para debugging

Representación Formal:

```
M = (Q, Σ, δ, q₀, F)

Q = {q0, q1, q2, q3, qr}
Σ = {SN, V, COMPLEMENTO}
q₀ = q0
F = {q3}
δ = tabla de transiciones definida en self.delta
```

Fragmento 12: Encabezado del Procesamiento

```python
def procesar_oracion(self, oracion: str) -> dict:
    """Procesa una oración a través del AFD."""
    print("\n" + "="*70)
    print("VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD")
    print("="*70)
    print(f"\nOración: '{oracion}'")

    # Resetear autómata
    self.estado_actual = self.q0
    self.historial = [(self.q0, "Estado inicial")]
```

Explicación:

- Método público que ejecuta el análisis completo
- Imprime encabezado visual
- **Importante:** Resetea el estado a q0 (permite reutilizar el objeto)
- Inicializa historial con estado inicial

Salida Visual:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'el gato come pescado'
```

Fragmento 13: Fase 1 - Análisis Léxico

```
# Análisis léxico
print("\n" + "-"*70)
print("FASE 1: Análisis léxico")
print("-"*70)

resultado = self.analizador.analizar_oracion(oracion)

if resultado['estructura']:
    print("\nPalabras reconocidas:")
    for categoria, palabra in resultado['estructura']:
        print(f"  '{palabra}' → {categoria}")
```

Explicación:

- Llama al método `analizar_oracion()` del `AnalizadorSimple`
- Recibe el diccionario con la estructura analizada
- Imprime cada palabra con su categoría gramatical

Salida Simulada:

```
----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'el' → DET
  'gato' → N
  'come' → V
  'pescado' → N
```

Fragmento 14: Fase 2 - Transiciones del AFD

```
print("\n" + "-"*70)
print("FASE 2: Transiciones del AFD")
print("-"*70)

# q0 → q1 (Sujeto)
if resultado['sujeto']:
    self._transicion('q1', f"Sujeto identificado: '{resultado['sujeto']}'")
else:
    self._transicion('qr', "No se identificó sujeto")
    return self._generar_resultado(resultado, False)

# q1 → q2 (Verbo)
if resultado['verbo']:
    self._transicion('q2', f"Verbo identificado: '{resultado['verbo']}'")
```

```
else:
    self._transicion('qr', "No se identificó verbo")
    return self._generar_resultado(resultado, False)

# q2 → q3 (Predicado completo)
if resultado['predicado']:
    self._transicion('q3', f"Predicado completo: '{resultado['predicado']}'")
else:
    self._transicion('qr', "No se completó el predicado")
    return self._generar_resultado(resultado, False)

# Verificar aceptación
aceptada = self.estado_actual in self.F

return self._generar_resultado(resultado, aceptada)
```

Explicación:

- Implementa las transiciones del AFD basándose en los componentes extraídos
- Si falta algún componente, transiciona a `qr` (rechazo) y retorna inmediatamente
- Estructura de "early return" para evitar procesamiento innecesario
- Verifica si el estado final está en F (conjunto de aceptación)

Salida Simulada (caso exitoso):

```
----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'el gato'

  q1 → q2
  Razón: Verbo identificado: 'come'

  q2 → q3
  Razón: Predicado completo: 'come pescado'
```

Salida Simulada (caso rechazo):

```
----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → qr
  Razón: No se identificó sujeto
```

Fragmento 15: Método de Transición

```
def _transicion(self, nuevo_estado: str, razon: str):
    """Realiza una transición de estado."""
    print(f"\n  {self.estado_actual} → {nuevo_estado}")
    print(f"  Razón: {razon}")
    self.estado_actual = nuevo_estado
    self.historial.append((nuevo_estado, razon))
```

Explicación:

- Método auxiliar privado (prefijo _)
- Imprime la transición de forma visual
- Actualiza el estado actual
- Registra en el historial para trazabilidad

Ejemplo de Traza:

```
# Llamada
self._transicion('q1', "Sujeto identificado: 'el gato'")

# Output
#   q0 → q1
#   Razón: Sujeto identificado: 'el gato'

# Estado interno actualizado
# self.estado_actual = 'q1'
# self.historial = [('q0', 'Estado inicial'), ('q1', "Sujeto identificado: 'el
gato'")]
```

Fragmento 16: Generación del Resultado Final

```
def _generar_resultado(self, analisis: dict, aceptada: bool) -> dict:
    """Genera el resultado final."""
    print("\n" + "="*70)
    print("RESULTADO FINAL")
    print("="*70)
    print(f"\nEstado final: {self.estado_actual}")
    print(f"¿Es estado de aceptación?: {'SÍ' if self.estado_actual in self.F
else 'NO'}")

    if aceptada:
        print("\n✓ ORACIÓN ACEPTADA")
        print(f"  Sujeto: {analisis['sujeto']}")
        print(f"  Verbo: {analisis['verbo']}")
        print(f"  Predicado: {analisis['predicado']}")
```

```
        # Generar árbol de derivación
        print("\n" + "-"*70)
        print("ÁRBOL DE DERIVACIÓN:")
        print("-"*70)
        self._imprimir_arbol_derivacion(analisis)
    else:
        print("\n✗ ORACIÓN RECHAZADA")

    return {
        'aceptada': aceptada,
        'estado_final': self.estado_actual,
        'historial': self.historial,
        'analisis': analisis
    }
```

Explicación:

- Método que genera la salida final del procesamiento
- Imprime el estado final y si es de aceptación
- Si es aceptada, muestra componentes y genera árbol de derivación
- Retorna diccionario completo con todos los detalles

Salida Simulada (Aceptación):

```
======================================================================
RESULTADO FINAL
======================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ

✓ ORACIÓN ACEPTADA
  Sujeto: el gato
  Verbo: come
  Predicado: come pescado

----------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
----------------------------------------------------------------------
[árbol aquí]
```

Salida Simulada (Rechazo):

```
======================================================================
RESULTADO FINAL
======================================================================
```

```
Estado final: qr
¿Es estado de aceptación?: NO

✗ ORACIÓN RECHAZADA
```

Fragmento 17: Impresión del Árbol de Derivación

```python
def _imprimir_arbol_derivacion(self, analisis: dict):
    """
    Imprime el árbol de derivación en formato ASCII.

    Gramática:
    S  → SN + SV
    SN → DET + N | PRON | N
    SV → V + complemento
    """
    print("\nS (Oración)")
    print("├── SN (Sintagma Nominal)")

    # Analizar estructura del sujeto
    sujeto_estructura = [e for e in analisis['estructura']
                         if e[1] in analisis['sujeto'].split()]

    for i, (cat, palabra) in enumerate(sujeto_estructura):
        es_ultimo = (i == len(sujeto_estructura) - 1)
        conector = "└──" if es_ultimo else "├──"
        print(f"│   {conector} {cat} → '{palabra}'")

    print("└── SV (Sintagma Verbal)")

    # Analizar estructura del predicado
    predicado_estructura = [e for e in analisis['estructura']
                            if e[1] in analisis['predicado'].split()]

    for i, (cat, palabra) in enumerate(predicado_estructura):
        es_ultimo = (i == len(predicado_estructura) - 1)
        conector = "└──" if es_ultimo else "├──"
        espacios = "    " if es_ultimo else "    "
        print(f"    {conector} {cat} → '{palabra}'")
```

Explicación:

- Genera representación visual ASCII del árbol sintáctico
- Usa caracteres especiales para dibujar el árbol: ├── └── │
- List comprehension para filtrar palabras del sujeto y predicado
- Lógica para determinar conectores (├── o └──) según posición

Salida Simulada:

```
S (Oración)
├── SN (Sintagma Nominal)
│   ├── DET → 'el'
│   └── N → 'gato'
└── SV (Sintagma Verbal)
    ├── V → 'come'
    └── N → 'pescado'
```

Fragmento 18: Reglas Gramaticales Aplicadas

```python
# Mostrar reglas aplicadas
print("\n" + "-"*70)
print("REGLAS GRAMATICALES APLICADAS:")
print("-"*70)
print("1. S → SN + SV")

# Determinar regla del SN
if any(e[0] == 'PRON' for e in sujeto_estructura):
    print("2. SN → PRON")
elif any(e[0] == 'DET' for e in sujeto_estructura):
    print("2. SN → DET + N")
else:
    print("2. SN → N")

print("3. SV → V + complemento")
```

Explicación:

- Determina qué reglas gramaticales se aplicaron
- Usa `any()` con generador para buscar categorías
- Muestra las producciones de la gramática utilizadas

Salida para "el gato come pescado":

```
----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → DET + N
3. SV → V + complemento
```

Salida para "yo camino":

```
----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → PRON
3. SV → V + complemento
```

---

5. Función Principal - Fragmentado y Explicado

Fragmento 19: Encabezado de la Aplicación

```python
def main():
    """Función principal."""
    print("\n" + "╔"+"="*68+"╗")
    print("║" + " "*15 + "VERSIÓN SIMPLIFICADA - AFD" + " "*27 + "║")
    print("║" + " "*20 + "Analizador de Oraciones" + " "*25 + "║")
    print("╚"+"="*68+"╝")

    print("\nCaracterísticas:")
    print("  - Sin dependencias externas (no requiere spaCy)")
    print("  - Vocabulario limitado (alfabeto finito)")
    print("  - AFD explícito con tabla de transiciones")
    print("  - Árbol de derivación ASCII")
```

Explicación:

- Punto de entrada de la aplicación
- Imprime banner decorativo usando caracteres Unicode
- Lista las características principales

Salida:

```
╔════════════════════════════════════════════════════════════════════╗
║               VERSIÓN SIMPLIFICADA - AFD                           ║
║                    Analizador de Oraciones                         ║
╚════════════════════════════════════════════════════════════════════╝

Características:
  - Sin dependencias externas (no requiere spaCy)
  - Vocabulario limitado (alfabeto finito)
  - AFD explícito con tabla de transiciones
  - Árbol de derivación ASCII
```

Fragmento 20: Mostrar Vocabulario Disponible

```
afd = AFDSimple()

# Mostrar vocabulario disponible
print("\n" + "-"*70)
print("VOCABULARIO DISPONIBLE:")
print("-"*70)
print(f"Determinantes: {', '.join(afd.analizador.determinantes)}")
print(f"Sustantivos: {', '.join(afd.analizador.sustantivos)}")
print(f"Verbos: {', '.join(afd.analizador.verbos)}")
print(f"Pronombres: {', '.join(afd.analizador.pronombres)}")
print(f"Preposiciones: {', '.join(afd.analizador.preposiciones)}")
```

Explicación:

- Crea instancia del AFD (que a su vez crea el analizador)
- Muestra todas las palabras del vocabulario al usuario
- Usa `join()` para concatenar las listas con comas

Salida:

```
----------------------------------------------------------------------
VOCABULARIO DISPONIBLE:
----------------------------------------------------------------------
Determinantes: el, la, un, una, los, las, mi, tu
Sustantivos: gato, perro, niño, niña, libro, parque, pescado, jardín, casa,
María, Juan, hermano, matemáticas, niños
Verbos: come, corre, estudia, lee, camino, juega, juegan, escribe, canta
Pronombres: yo, tú, él, ella, nosotros
Preposiciones: por, en, de, con, a
```

Fragmento 21: Ejemplos Sugeridos

```
print("\n" + "-"*70)
print("EJEMPLOS DE ORACIONES VÁLIDAS:")
print("-"*70)
print("  - el gato come pescado")
print("  - yo camino por el parque")
print("  - María estudia matemáticas")
print("  - los niños juegan en el jardín")
```

Explicación:

- Provee ejemplos al usuario
- Ayuda a entender qué tipo de oraciones acepta el sistema

Salida:

```
----------------------------------------------------------------------
EJEMPLOS DE ORACIONES VÁLIDAS:
----------------------------------------------------------------------
  - el gato come pescado
  - yo camino por el parque
  - María estudia matemáticas
  - los niños juegan en el jardín
```

Fragmento 22: Loop Interactivo

```python
while True:
    print("\n" + "-"*70)
    print("Ingresa una oración para analizar")
    print("(o escribe 'salir' para terminar)")
    print("-"*70)

    oracion = input("\nOración: ").strip()

    if oracion.lower() in ['salir', 'exit', 'quit']:
        print("\nHasta luego!\n")
        break

    if not oracion:
        print("\nPor favor ingresa una oración válida.")
        continue

    # Procesar la oración
    resultado = afd.procesar_oracion(oracion)

    print("\n" + "="*70)
```

Explicación:

- Loop infinito `while True`
- Solicita entrada del usuario con `input()`
- Comandos de salida: 'salir', 'exit', 'quit' (case-insensitive)
- Valida que no esté vacía
- Llama a `procesar_oracion()` con la entrada
- Imprime separador al final

Flujo de Interacción:

```
----------------------------------------------------------------------
Ingresa una oración para analizar
(o escribe 'salir' para terminar)
----------------------------------------------------------------------

Oración: el gato come pescado

[... salida del análisis ...]

======================================================================

----------------------------------------------------------------------
Ingresa una oración para analizar
(o escribe 'salir' para terminar)
----------------------------------------------------------------------

Oración: salir

Hasta luego!
```

Fragmento 23: Punto de Entrada y Manejo de Excepciones

```python
if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("\n\nPrograma interrumpido por el usuario!\n")
    except Exception as e:
        print(f"\nError: {e}\n")
```

Explicación:

- `if __name__ == "__main__"`: Solo ejecuta si el script se corre directamente
- `try-except`: Manejo de excepciones
- `KeyboardInterrupt`: Captura Ctrl+C
- `Exception`: Captura cualquier otro error

Comportamiento:

```python
# Si el usuario presiona Ctrl+C:
# Output: "Programa interrumpido por el usuario!"

# Si ocurre un error no manejado:
# Output: "Error: [mensaje del error]"
```

---

6. Ejecuciones Completas Paso a Paso

EJECUCIÓN 1: "el gato come pescado"

Input del Usuario:

```
Oración: el gato come pescado
```

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'el gato come pescado'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'el' → DET
  'gato' → N
  'come' → V
  'pescado' → N

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'el gato'

  q1 → q2
  Razón: Verbo identificado: 'come'

  q2 → q3
  Razón: Predicado completo: 'come pescado'

======================================================================
RESULTADO FINAL
======================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ

✓ ORACIÓN ACEPTADA
  Sujeto: el gato
  Verbo: come
```

```
  Predicado: come pescado

----------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
----------------------------------------------------------------------

S (Oración)
├── SN (Sintagma Nominal)
│   ├── DET → 'el'
│   └── N → 'gato'
└── SV (Sintagma Verbal)
    ├── V → 'come'
    └── N → 'pescado'

----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → DET + N
3. SV → V + complemento

======================================================================
```

Análisis Interno Paso a Paso:

```
# Tokenización
palabras = ['el', 'gato', 'come', 'pescado']

# Análisis léxico
i = 0
# 'el' in determinantes → True
sujeto = ['el']
i = 1
# 'gato' in sustantivos → True
sujeto = ['el', 'gato']
i = 2
resultado['sujeto'] = 'el gato'

# 'come' in verbos → True
resultado['verbo'] = 'come'
i = 3

# Loop complemento
# 'pescado' in sustantivos → True
complemento = ['come', 'pescado']
resultado['predicado'] = 'come pescado'

# Transiciones AFD
estado = 'q0'
# Sujeto existe → q0 → q1
estado = 'q1'
```

```
# Verbo existe → q1 → q2
estado = 'q2'
# Predicado existe → q2 → q3
estado = 'q3'

# q3 in F → ACEPTADA
```

EJECUCIÓN 2: "yo camino por el parque"

Input del Usuario:

```
Oración: yo camino por el parque
```

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'yo camino por el parque'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'yo' → PRON
  'camino' → V
  'por' → PREP
  'el' → DET
  'parque' → N

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'yo'

  q1 → q2
  Razón: Verbo identificado: 'camino'

  q2 → q3
  Razón: Predicado completo: 'camino por el parque'

======================================================================
RESULTADO FINAL
```

```
======================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ

✓ ORACIÓN ACEPTADA
  Sujeto: yo
  Verbo: camino
  Predicado: camino por el parque

----------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
----------------------------------------------------------------------

S (Oración)
├── SN (Sintagma Nominal)
│   └── PRON → 'yo'
└── SV (Sintagma Verbal)
    ├── V → 'camino'
    ├── PREP → 'por'
    ├── DET → 'el'
    └── N → 'parque'

----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → PRON
3. SV → V + complemento

======================================================================
```

Análisis Interno:

```
# Tokenización
palabras = ['yo', 'camino', 'por', 'el', 'parque']

# Análisis léxico
i = 0
# 'yo' not in determinantes → False
# 'yo' in pronombres → True
sujeto = ['yo']
i = 1
resultado['sujeto'] = 'yo'

# 'camino' in verbos → True
resultado['verbo'] = 'camino'
i = 2

# Loop complemento
complemento = ['camino']
```

```
# i=2: 'por' in preposiciones → True
complemento = ['camino', 'por']
# i=3: 'el' in determinantes → True
complemento = ['camino', 'por', 'el']
# i=4: 'parque' in sustantivos → True
complemento = ['camino', 'por', 'el', 'parque']
resultado['predicado'] = 'camino por el parque'

# Transiciones: q0 → q1 → q2 → q3
# ACEPTADA
```

EJECUCIÓN 3: "María estudia matemáticas"

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'María estudia matemáticas'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'maría' → N
  'estudia' → V
  'matemáticas' → N

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'maría'

  q1 → q2
  Razón: Verbo identificado: 'estudia'

  q2 → q3
  Razón: Predicado completo: 'estudia matemáticas'

======================================================================
RESULTADO FINAL
======================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ
```

```
✓ ORACIÓN ACEPTADA
  Sujeto: maría
  Verbo: estudia
  Predicado: estudia matemáticas

----------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
----------------------------------------------------------------------

S (Oración)
├── SN (Sintagma Nominal)
│   └── N → 'maría'
└── SV (Sintagma Verbal)
    ├── V → 'estudia'
    └── N → 'matemáticas'

----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → N
3. SV → V + complemento

======================================================================
```

EJECUCIÓN 4: "los niños juegan" (Verbo Intransitivo)

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'los niños juegan'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'los' → DET
  'niños' → N
  'juegan' → V

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------
```

```
  q0 → q1
  Razón: Sujeto identificado: 'los niños'

  q1 → q2
  Razón: Verbo identificado: 'juegan'

  q2 → q3
  Razón: Predicado completo: 'juegan'

======================================================================
RESULTADO FINAL
======================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ

✓ ORACIÓN ACEPTADA
  Sujeto: los niños
  Verbo: juegan
  Predicado: juegan

----------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
----------------------------------------------------------------------

S (Oración)
├── SN (Sintagma Nominal)
│   ├── DET → 'los'
│   └── N → 'niños'
└── SV (Sintagma Verbal)
    └── V → 'juegan'

----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → DET + N
3. SV → V + complemento

======================================================================
```

Análisis Interno:

```
# Tokenización
palabras = ['los', 'niños', 'juegan']

# Análisis léxico
# Sujeto: 'los niños' (DET + N)
# Verbo: 'juegan'
# Complemento: solo el verbo (sin objeto directo)
```

```
complemento = ['juegan']  # No hay más palabras
resultado['predicado'] = 'juegan'

# El sistema acepta verbos intransitivos
# Regla aplicada: SV → V
```

EJECUCIÓN 5: "el perro corre rápidamente" (Con Adverbio)

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'el perro corre rápidamente'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'el' → DET
  'perro' → N
  'corre' → V
  'rápidamente' → ADV

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'el perro'

  q1 → q2
  Razón: Verbo identificado: 'corre'

  q2 → q3
  Razón: Predicado completo: 'corre rápidamente'

======================================================================
RESULTADO FINAL
======================================================================

Estado final: q3
¿Es estado de aceptación?: SÍ

✓ ORACIÓN ACEPTADA
  Sujeto: el perro
  Verbo: corre
```

```
  Predicado: corre rápidamente

----------------------------------------------------------------------
ÁRBOL DE DERIVACIÓN:
----------------------------------------------------------------------

S (Oración)
├── SN (Sintagma Nominal)
│   ├── DET → 'el'
│   └── N → 'perro'
└── SV (Sintagma Verbal)
    ├── V → 'corre'
    └── ADV → 'rápidamente'

----------------------------------------------------------------------
REGLAS GRAMATICALES APLICADAS:
----------------------------------------------------------------------
1. S → SN + SV
2. SN → DET + N
3. SV → V + complemento

======================================================================
```

EJECUCIÓN 6: "come pescado" (RECHAZO - Sin sujeto)

Input del Usuario:

```
Oración: come pescado
```

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'come pescado'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'come' → V
  'pescado' → N

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
```

```
------------------------------------------------------------------------

  q0 → qr
  Razón: No se identificó sujeto

========================================================================
RESULTADO FINAL
========================================================================

Estado final: qr
¿Es estado de aceptación?: NO

✗ ORACIÓN RECHAZADA

========================================================================
```

Análisis Interno:

```
# Tokenización
palabras = ['come', 'pescado']

# Análisis léxico
i = 0
# 'come' not in determinantes → False
# 'come' not in pronombres → False
# 'come' not in sustantivos → False
# No se identificó sujeto
sujeto_palabras = []
resultado['sujeto'] = None

# 'come' in verbos → True
resultado['verbo'] = 'come'

# Transiciones AFD
# if resultado['sujeto']: → False
self._transicion('qr', "No se identificó sujeto")
# RECHAZADA inmediatamente
```

EJECUCIÓN 7: "el gato pescado" (RECHAZO - Sin verbo)

Salida Completa:

```
========================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
========================================================================

Oración: 'el gato pescado'
```

```
------------------------------------------------------------------------
FASE 1: Análisis léxico
------------------------------------------------------------------------

Palabras reconocidas:
  'el' → DET
  'gato' → N
  'pescado' → N

------------------------------------------------------------------------
FASE 2: Transiciones del AFD
------------------------------------------------------------------------

  q0 → q1
  Razón: Sujeto identificado: 'el gato'

  q1 → qr
  Razón: No se identificó verbo

========================================================================
RESULTADO FINAL
========================================================================

Estado final: qr
¿Es estado de aceptación?: NO

✗ ORACIÓN RECHAZADA

========================================================================
```

Análisis Interno:

```python
# Tokenización
palabras = ['el', 'gato', 'pescado']

# Análisis léxico
i = 0
# Sujeto: 'el gato' identificado correctamente
resultado['sujeto'] = 'el gato'
i = 2

# 'pescado' not in verbos → False
resultado['verbo'] = None

# Transiciones AFD
# q0 → q1 (sujeto OK)
# if resultado['verbo']: → False
self._transicion('qr', "No se identificó verbo")
# RECHAZADA
```

EJECUCIÓN 8: "el dinosaurio come pescado" (RECHAZO - Palabra no reconocida)

Salida Completa:

```
======================================================================
VERSIÓN SIMPLIFICADA - ANALIZADOR DE ORACIONES CON AFD
======================================================================

Oración: 'el dinosaurio come pescado'

----------------------------------------------------------------------
FASE 1: Análisis léxico
----------------------------------------------------------------------

Palabras reconocidas:
  'el' → DET
  'dinosaurio' → ?
  'come' → V
  'pescado' → N

----------------------------------------------------------------------
FASE 2: Transiciones del AFD
----------------------------------------------------------------------

  q0 → qr
  Razón: No se identificó sujeto

======================================================================
RESULTADO FINAL
======================================================================

Estado final: qr
¿Es estado de aceptación?: NO

✗ ORACIÓN RECHAZADA

======================================================================
```

Análisis Interno:

```
# Tokenización
palabras = ['el', 'dinosaurio', 'come', 'pescado']

# Análisis léxico
i = 0
# 'el' in determinantes → True
sujeto_palabras = ['el']
```

```
i = 1
# 'dinosaurio' not in sustantivos → False
# No se añade a sujeto_palabras
# Sujeto queda incompleto

# if sujeto_palabras: → True (contiene 'el')
resultado['sujeto'] = 'el'  # Solo el determinante

# Pero i no avanzó correctamente, la estructura falla
# Resultado: sujeto incompleto o mal formado
# RECHAZADA
```

---

7. Casos de Uso Avanzados

Caso de Uso 1: Oración con Múltiples Espacios

Input:

```
Oración: el    gato     come   pescado
```

Comportamiento:

```
# split() sin argumentos maneja múltiples espacios
palabras = oracion.strip().split()
# palabras = ['el', 'gato', 'come', 'pescado']
# Se procesa normalmente
```

Resultado: ACEPTADA (los espacios extra se ignoran)

Caso de Uso 2: Oración con Mayúsculas

Input:

```
Oración: EL GATO COME PESCADO
```

Comportamiento:

```
# lower() convierte todo a minúsculas
palabras = oracion.lower().strip().split()
# palabras = ['el', 'gato', 'come', 'pescado']
```

Resultado: ACEPTADA (case-insensitive)

Caso de Uso 3: Oración Vacía

Input:

```
Oración:
```

Comportamiento:

```
oracion = input().strip()
if not oracion:
    print("\nPor favor ingresa una oración válida.")
    continue
# No se procesa, vuelve a solicitar entrada
```

Resultado: Solicita nueva entrada sin procesamiento

Caso de Uso 4: Palabra Parcialmente Reconocida

Input:

```
Oración: el gatos come pescado
```

Comportamiento:

```
# 'gatos' (plural con 's') no está en la lista
# sustantivos = ['gato', ...]  # Sin 's'
# 'gatos' not in sustantivos → True
# Se marca como desconocida
```

Resultado: RECHAZADA (el sistema no hace stemming)

Caso de Uso 5: Nombres con Tildes

Input:

```
Oración: María estudia matemáticas
```

Comportamiento:

```
# 'María' con mayúscula → 'maría' con minúscula
# 'maría' in sustantivos → True (está en la lista)
# 'matemáticas' con tilde → también está en la lista
```

Resultado: ACEPTADA (el vocabulario incluye palabras con tildes)

---

8. Conclusiones

Resumen del Código

Componente	Líneas	Complejidad	Función
AnalizadorSimple	~110	O(n)	Análisis léxico
AFDSimple	~175	O(n)	Validación sintáctica
main()	~60	-	Interfaz de usuario
TOTAL	**~373**	**O(n)**	Sistema completo

Características Destacadas

1. **Código autodocumentado:**
   - Docstrings en clases y métodos
   - Comentarios explicativos en secciones clave
   - Nombres descriptivos de variables
2. **Separación de responsabilidades:**
   - `AnalizadorSimple`: Solo análisis léxico
   - `AFDSimple`: Solo validación sintáctica
   - `main()`: Solo interfaz de usuario
3. **Manejo de errores:**
   - Validación de entrada vacía
   - Palabras no reconocidas marcadas con '?'
   - Excepciones capturadas en el nivel superior
4. **Salida informativa:**
   - Muestra cada fase del procesamiento
   - Traza de transiciones del AFD
   - Árbol de derivación visual
   - Reglas gramaticales aplicadas

Métricas de Calidad

Complejidad Temporal: O(n) lineal

- Tokenización: O(n)
- Análisis léxico: O(n)
- Transiciones AFD: O(1)
- Total: O(n)

Complejidad Espacial: O(n) lineal

- Vocabulario: O(1) constante
- Estructuras temporales: O(n)

Mantenibilidad: Alta

- Código modular
- Fácil de extender
- Sin dependencias externas

Aplicaciones Educativas

Este código es ideal para:

1. **Cursos de Teoría de Autómatas:**
    - Implementación directa de AFD
    - Correspondencia con definición formal
    - Visualización de transiciones
2. **Cursos de Lenguajes Formales:**
    - Gramáticas independientes del contexto
    - Árboles de derivación
    - Análisis sintáctico
3. **Cursos de Compiladores:**
    - Análisis léxico (scanner)
    - Análisis sintáctico (parser)
    - Estructuras de datos para parsing
4. **Proyectos de Programación:**
    - Ejemplo de diseño limpio
    - Patrones de desarrollo
    - Documentación efectiva

Fin del Documento: Código Completo Comentado con Ejecuciones