“Año del Bicentenario, de la consolidación de nuestra Independencia, y de la conmemoración de las heroicas batallas de Junín y Ayacucho”

CARRERA: INGENIERÍA DE SOFTWARE
WAYRASIMI: IMPLEMENTACIÓN DE SU ANALIZADOR LÉXICO
COMPILADORES

ESTUDIANTES:

Ortiz Castañeda Jorge Luis
Huamani Huamani Jhordan Steven Octavio
Flores Leon Miguel Angel

DOCENTE:
Vicente Enrique Machaca Arceda

Arequipa, Perú
23 de marzo de 2025

Índice

1 Introducción **2**

2 WayraSimi **3**

3 Justificación y Descripción del Lenguaje **3**
- 3.1 Sintaxis Python-inspirada . . . 3
- 3.2 Tipado estático e inferencia de tipos . . . 3
- 3.3 Concurrencia integrada . . . 3

4 Tipos de Datos en WayraSimi **3**

5 Ejemplos **4**

6 Código en Python **4**
- 6.1 pruebita.py . . . 4
- 6.2 Explicación pruebita.py . . . 7
  - 6.2.1 Importación de la librería . . . 7
  - 6.2.2 Definición de los tokens . . . 7
  - 6.2.3 Expresiones regulares para tokens simples . . . 7
  - 6.2.4 Funciones para tokens complejos . . . 7
  - 6.2.5 Manejo de comentarios y espacios . . . 7
  - 6.2.6 Manejo de errores . . . 7
  - 6.2.7 Creación del lexer . . . 7
  - 6.2.8 Pruebas y análisis de archivos . . . 7
- 6.3 Ejecución principal . . . 7

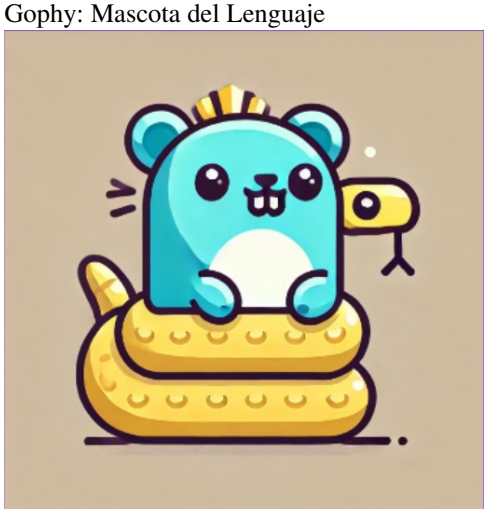
1. Introducción

En el presente informe, se expondrá el desarrollo de nuestro lenguaje en quechua, denominado "WayraSimi", durante su fase de análisis y pruebas con el analizador léxico. Para llevar a cabo esta implementación, se ha utilizado el lenguaje de programación Python, junto con la librería "ply.lex", con el objetivo de construir un compilador básico para nuestro lenguaje propuesto.

2. WayraSimi

Presentamos WayraSimi, un lenguaje de programación compilado que fusiona la claridad de Python con la eficiencia de Go. Diseñado para aplicaciones de alto rendimiento y sistemas concurrentes, WayraSimi busca ser una herramienta poderosa y accesible para desarrolladores.

- Ejemplo de documento: WayraSimi.ws
- Composición: Wayra (Viento, Aire) + Simi (Palabra, Lenguaje)
- Significado: Lenguaje del Viento o Lenguaje Veloz, implicando rapidez y eficiencia.
- Gophy: Mascota del Lenguaje



3. Justificación y Descripción del Lenguaje

3.1. Sintaxis Python-inspirada

Utiliza la indentación para definir bloques de código, buscando la legibilidad y simplicidad sintáctica de Python.

3.2. Tipado estático e inferencia de tipos

Similar a Go, WayraSimi será un lenguaje de tipado estático para garantizar la seguridad y el rendimiento.

3.3. Concurrencia integrada

Inspirado en Go, WayraSimi tendrá soporte nativo para concurrencia ligera (gorutinas) y comunicación entre procesos (canales).

4. Tipos de Datos en WayraSimi

- **Yupay (entero)**: Números enteros, sin parte decimal (int, int32, int64).
- **Chiqi_kay (flotante)**: Números con parte decimal (punto flotante, float32, float64).
- **Qillga (texto)**: Cadenas de caracteres, texto (string).

WayraSimi soporta varios tipos de datos, incluyendo enteros, flotantes, texto, booleanos, listas y mapas, cada uno con su propia representación y uso.

5. Ejemplos

```
ruray hatunRuray() {
    imprimiy(" Allin  punchaw, Pachamama!");
}
```

Listing 1: Hola Mundo

```
ruray hatunRuray() {
    para i := 0; i < 5; i++ {
        para j := 0; j < 5; j++ {
            imprimiy(i, j);
        }
    }
}
```

Listing 2: Bucles Anidados

```
ruray factorial(n yupay) yupay {
    sichus n == 0 {
        kutipay 1;
    }
    kutipay n * factorial(n-1);
}

ruray hatunRuray() {
    imprimiy(factorial(5));
}
```

Listing 3: Recursividad

6. Código en Python

6.1. pruebita.py

```

import ply.lex as lex

# Tabla 1: Lexemas WayraSimi
tokens = [
    'YUPAY_TOKEN',  # Número, Decimal
    'CHIQAP_TOKEN', # Booleano
    'QILLQA_TOKEN', # Texto, Letra
    'CHIQAQP_TOKEN', # Verdad, Realidad
    'IDENTIFICADOR_TOKEN', # Variable, function_1
    'OPERADOR_MAS', # +
    'OPERADOR_MENOS', # -
    'OPERADOR_PACHA', # *
    'OPERADOR_RAKI', # /
    'OPERADOR_MODULO', # %
    'OPERADOR_ASIGNACION', # =
    'OPERADOR_IGUALDAD', # ==
    'OPERADOR_MANA_IGUAL', # !=
    'OPERADOR_MENOR', # <
    'OPERADOR_MAYOR', # >
    'OPERADOR_MENOR_IGUAL', # <=
    'OPERADOR_MAYOR_IGUAL', # >=
    'OPERADOR_LOGICO_WAN', # and
    'OPERADOR_LOGICO_UTAQ', # or
    'OPERADOR_LOGICO_MANA', # not
    'PARENTESIS_ABRE', # (
    'PARENTESIS_CIERRA', # )
    'LLAVE_ABRE', # {
    'LLAVE_CIERRA', # }
    'CORCHETE_ABRE', # [
    'CORCHETE_CIERRA', # ]
    'COMA_TOKEN', # ,
    'PUNTO_TOKEN', # .
```

```
    'DOS_PUNTOS_TOKEN', # :
    'PUNTO_Y_COMA_TOKEN', # ;
    'COMENTARIO_TOKEN_LINEA', # # ...
    'COMENTARIO_TOKEN_BLOQUE_ABRE', # /*
    'COMENTARIO_TOKEN_BLOQUE_CIERRA', # */
    'PALABRA_RESERVADA_SICHUS', # sichus
    'PALABRA_RESERVADA_MANA_SICHUS', # mana sichus
    'PALABRA_RESERVADA_MANA', # mana
    'PALABRA_RESERVADA_PARA', # para
    'PALABRA_RESERVADA_RURAY', # ruray
    'PALABRA_RESERVADA_KUTIPAY', # kutipay
    'PALABRA_RESERVADA_IMPRIMIY', # imprimiy
    'PALABRA_RESERVADA_AYLLU', # ayllu
    'PALABRA_RESERVADA_VAR', # variable
    'PALABRA_RESERVADA_TAKYAQ', # takyaq
    'CADENA_TOKEN' # String literals
]

# Regular expressions for simple tokens
t_OPERADOR_MAS     = r'\+'
t_OPERADOR_MENOS   = r'-'
t_OPERADOR_PACHA   = r'\*'
t_OPERADOR_RAKI    = r'/'
t_OPERADOR_MODULO  = r'%'
t_OPERADOR_ASIGNACION = r'='
t_OPERADOR_IGUALDAD = r'=='
t_OPERADOR_MANA_IGUAL = r'!='
t_OPERADOR_MENOR   = r'<'
t_OPERADOR_MAYOR   = r'>'
t_OPERADOR_MENOR_IGUAL = r'<='
t_OPERADOR_MAYOR_IGUAL = r'>='
t_OPERADOR_LOGICO_WAN = r'wan'
t_OPERADOR_LOGICO_UTAQ = r'utaq'
t_OPERADOR_LOGICO_MANA = r'mana'
t_PARENTESIS_ABRE  = r'\('
t_PARENTESIS_CIERRA = r'\)'
t_LLAVE_ABRE       = r'\{'
t_LLAVE_CIERRA      = r'\}'
t_CORCHETE_ABRE    = r'\['
t_CORCHETE_CIERRA   = r'\]'
t_COMA_TOKEN       = r','
t_PUNTO_TOKEN      = r'\.'
t_DOS_PUNTOS_TOKEN = r':'
t_PUNTO_Y_COMA_TOKEN = r';'
t_PALABRA_RESERVADA_SICHUS = r'sichus'
t_PALABRA_RESERVADA_MANA_SICHUS = r'mana\s+sichus' # Handle space between mana and
    ↪ sichus
t_PALABRA_RESERVADA_MANA = r'mana'
t_PALABRA_RESERVADA_PARA = r'para'
t_PALABRA_RESERVADA_RURAY = r'ruray'
t_PALABRA_RESERVADA_KUTIPAY = r'kutipay'
t_PALABRA_RESERVADA_IMPRIMIY = r'imprimiy'
t_PALABRA_RESERVADA_AYLLU = r'ayllu'
t_PALABRA_RESERVADA_VAR = r'variable'
t_PALABRA_RESERVADA_TAKYAQ = r'takyaq'

def t_YUPAY_TOKEN(t):
    r'\d+(\.\d+)?'
    t.value = float(t.value) if '.' in t.value else int(t.value)
    return t

def t_CHIQAP_TOKEN(t):
    r'chiqap|mana_chiqap'
    t.value = True if t.value == 'chiqap' else False
    return t

def t_QILLQA_TOKEN(t):
    r'\'[^\']*\'|\"[^\"]*\"' # Single or double quoted strings
    t.value = t.value[1:-1] # Remove quotes
    return t

def t_CHIQAQP_TOKEN(t):
    r'chiqaqp' # Assuming 'chiqaqp' represents truthiness in code, adjust if needed.
```

```
    return t

def t_IDENTIFICADOR_TOKEN(t):
    r'[a-zA-Z_][a-zA-Z0-9_]*'
    # Check for reserved words to avoid misclassification as identifier if needed,
    # but based on the table, keywords are already defined as tokens.
    return t

def t_COMENTARIO_TOKEN_LINEA(t):
    r'\#.*'
    # No return value. Token is discarded
    pass

def t_newline(t):
    r'\n+'
    t.lexer.lineno += len(t.value)

t_ignore = ' \t'

# Regla para manejar caracteres ilegales
def t_error(t):
    print(f"Error léxico: Carácter ilegal '{t.value[0]}' en línea {t.lineno},
        ↪ posición {t.lexpos}")
    t.lexer.skip(1)

lexer = lex.lex()

# Test it out
def test_lexer(data):
    lexer.lineno = 1  # Reset line number for each test
    lexer.input(data)
    tokens_list = []
    while True:
        tok = lexer.token()
        if not tok:
            break  # No more input
        tokens_list.append(tok)
        print(tok)  # Imprimir para visualización inmediata
    return tokens_list

# Función para leer archivo y analizar tokens
def analyze_file(filepath):
    try:
        with open(filepath, 'r', encoding='utf-8') as file:
            data = file.read()
    except FileNotFoundError:
        print(f"Error: No se encontró el archivo '{filepath}'.")
        return []

    lexer.lineno = 1  # Reset line number for each analysis
    lexer.input(data)
    tokens_list = []

    while True:
        tok = lexer.token()
        if not tok:
            break  # No more input
        tokens_list.append({
            'type': tok.type,
            'value': tok.value,
            'line': tok.lineno,
            'position': tok.lexpos
        })

    return tokens_list

if __name__ == '__main__':
    # Archivos de ejemplo
    example_files = [
        "D:\\uSalle\\Software\\ejemplito1.txt",
        "D:\\uSalle\\Software\\ejemplito2.txt",
        "D:\\uSalle\\Software\\ejemplito3.txt"
   ]
```

```
178
179     for filename in example_files:
180         print(f"\n--- Analizando léxicamente {filename} ---")
181         tokens = analyze_file(filename)
182         for token in tokens:
183             print(token)
```

Listing 4: Lexema en lenguaje Python

6.2. Explicación pruebita.py

6.2.1. Importación de la librería

Se importa la librería **ply.lex**, que es una herramienta para construir analizadores léxicos en Python.

6.2.2. Definición de los tokens

Se define una lista de tokens (**'YUPAY_TOKEN'**, **'CHIQAP_TOKEN'**, etc), que son los componentes léxicos que el analizador reconocerá. Cada token representa un elemento del lenguaje, como números, operadores, palabras reservadas, etc.

6.2.3. Expresiones regulares para tokens simples

Aquí se definen las expresiones regulares para los tokens simples, como operadores (+, -, *, etc.), símbolos de puntuación (",", ";", ":", etc.) y palabras reservadas (sichus, mana, ruray, etc.).

6.2.4. Funciones para tokens complejos

Algunos tokens requieren lógica adicional para ser reconocidos. Por ejemplo:

- **Números (YUPAY_TOKEN):** Reconoce números enteros o decimales y los convierte a int o float.
- **Booleanos (CHIQAP_TOKEN):** Reconoce los valores booleanos chiqap (verdadero) y mana_chiqap (falso).
- **Cadenas de texto (QILLQA_TOKEN):**Reconoce cadenas de texto entre comillas simples o dobles.
- **Identificadores (IDENTIFICADOR_TOKEN):**Reconoce nombres de variables o funciones.

6.2.5. Manejo de comentarios y espacios

- Comentarios de línea
- Espacios y tabulaciones
- Saltos de línea

6.2.6. Manejo de errores

Si se encuentra un carácter no reconocido, se imprime un mensaje de error y se ignora el carácter.

6.2.7. Creación del lexer

Se crea una instancia del analizador léxico.

6.2.8. Pruebas y análisis de archivos

- **Función test_lexer:** Prueba el lexer con una cadena de entrada.
- **Función analyze_file:** Lee un archivo y analiza su contenido léxicamente.

6.3. Ejecución principal

Se analizan varios archivos de ejemplo y se imprimen los tokens encontrados.