proyecto 2
Manual técnico

Marvin obidio perez Larios

201903712

Objetivos

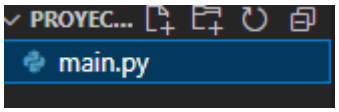
• implementar un software en Python que permitan plasmar los conocimientos sobre el analizador léxico

• Implementar la programación orientada a objetos en Python.

Requisitos

- Visual Studio
- python
- Graphviz
- Tkinker

Estructura del programa: todas las clases creadas



Esta parte del código es para crear el menú

```
def create_menu(self):
    menubar = tk.Menu(self)

    file_menu = tk.Menu(menubar, tearoff=0)
    file_menu.add_command(label="Nuevo", command=self.new_file)
    file_menu.add_command(label="Abrir", command=self.open_file)
    file_menu.add_command(label="Guardar", command=self.save_file)
    file_menu.add_command(label="Guardar Como", command=self.save_file_as)
    file_menu.add_separator()
    file_menu.add_command(label="Salir", command=self.quit)
    menubar.add_cascade(label="Archivo", menu=file_menu)

    analysis_menu = tk.Menu(menubar, tearoff=0)
    analysis_menu.add_command(label="Generar sentencias MongoDB", command=self.generate_mongodb_statements)
    menubar.add_cascade(label="Análisis", menu=analysis_menu)

    tokens_menu = tk.Menu(menubar, tearoff=0)
    tokens_menu.add_command(label="Ver Tokens", command=self.show_tokens)
    menubar.add_cascade(label="Tokens", menu=tokens_menu)

    self.config(menu=menubar)
```

Esta parte es para traducir

```
def generate_mongodb_statements(self):

    code = self.editor.get("1.0", tk.END)
    self.parser.parse(code)
    if self.parser.errors:

        self.error_area.delete("1.0", tk.END)
        for error in self.parser.errors:
            self.error_area.insert(tk.END, f"{error['type']}: Línea {error['line']}: {error['description']}\n")
    else:
        self.error_area.delete("1.0", tk.END)

        if self.parser.statements:

            for statement in self.parser.statements:
                messagebox.showinfo("Sentencia Generada", statement)
        else:
            messagebox.showinfo("No se Generaron Sentencias", "No se generaron sentencias de MongoDB.")
```

Esta parte genera los tokens

```
    def show_tokens(self):
        code = self.editor.get("1.0", tk.END)
        tokens = self.tokenize(code)

        tokens_window = tk.Toplevel(self)
        tokens_window.title("Tokens Reconocidos")

        tokens_text = tk.Text(tokens_window)
        tokens_text.pack(fill="both", expand=True)

        tokens_text.insert(tk.END, "Número correlativo\tToken\tNúmero de Token\tLexema\n")
        for i, token in enumerate(tokens, start=1):
            tokens_text.insert(tk.END, f"{i}\t{token['type']}\t{token['number']}\t{token['lexeme']}\n")

    def tokenize(self, code):
        tokens = []
        token_patterns = [
            (r'(CrearBD|EliminarBD|CrearColeccion|EliminarColeccion|InsertarUnico|ActualizarUnico|EliminarUnico|BuscarTodo|
            (r'("[^"]*"|\w+)', 'Lexema')
        ]
        for line_number, line in enumerate(code.split('\n'), start=1):

            for pattern, token_type in token_patterns:

                for match in re.finditer(pattern, line):
                    lexeme = match.group()
```

```
        token_patterns = [
            (r'(CrearBD|EliminarBD|CrearColeccion|EliminarColeccion|InsertarUnico|ActualizarUnico|EliminarUnico|BuscarTodo|
            (r'("[^"]*"|\w+)', 'Lexema')
        ]
        for line_number, line in enumerate(code.split('\n'), start=1):

            for pattern, token_type in token_patterns:

                for match in re.finditer(pattern, line):
                    lexeme = match.group()
                    start = match.start()
                    end = match.end()

                    tokens.append({
                        'type': token_type,
                        'number': line_number,
                        'lexeme': lexeme,
                        'start': start,
                        'end': end
                    })

        return tokens
```