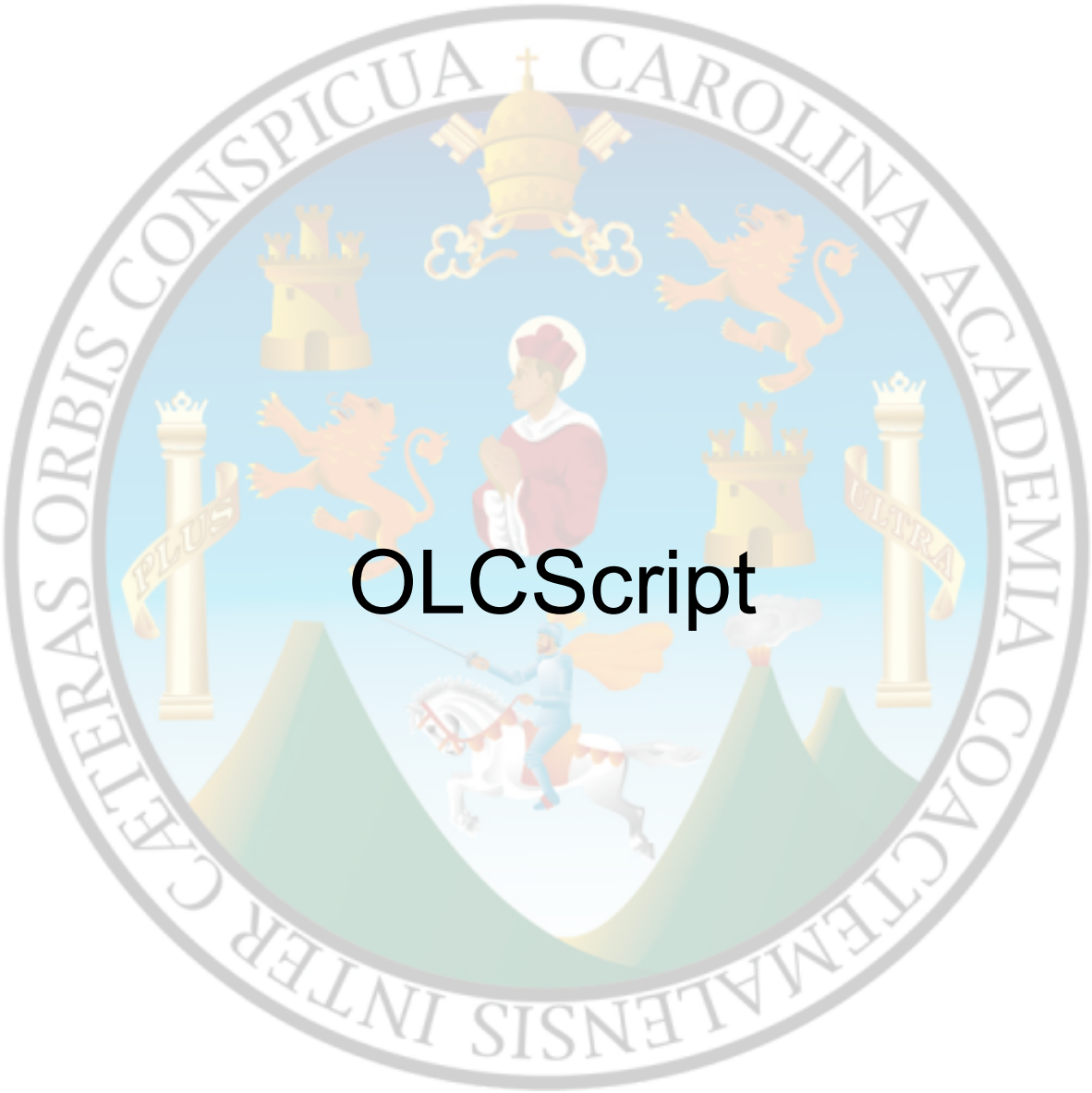
MANUAL TECNICO



OLCScript

Tabla de Contenido

Introducción

El presente documento describe la serie de pasos que realizan algunos de los métodos o funciones que conforman el programa computacional (OLCScript Interpreter) por medio de fragmentos de código, como también se detalla los conocimientos previos que debe tener el lector de este manual para comprender de una mejor manera el funcionamiento de cada una de las partes del código que conforman el programa.

Objetivos

Instruir el uso adecuado del programa computacional, describiendo el diseño y la lógica del programa por medio de fragmentos de códigos.

Describir al usuario el funcionamiento del programa para el mejor uso de él y demostrar el proceso necesario para su ejecución.

Orientar al usuario a entender la estructura del programa, como lo son sus clases y cada uno de los métodos que componen dicha clase.

Información Destacada

El manual técnico hace referencia a la información necesaria con el fin de orientar al personal en la concepción, planteamiento análisis programación e instalación del sistema. Es de notar que la redacción propia del manual técnico está orientada a personal con conocimientos en sistemas y tecnologías de información, conocimientos de programación avanzada sobre Javascript, responsables del mantenimiento e instalación del programa computacional en el computador.

Conocimientos Previos

Los conocimientos mínimos que deben tener las personas que operarán las páginas y deberán utilizar este manual son:

- Conocimientos y entendimientos en logaritmos
- Conocimientos en Python
- Conocimientos en Ply
- Conocimientos en Tkinter
- Conocimientos del patrón de diseño Patrón Intérprete
- Conocimiento básico de Windows

Requerimientos

El sistema puede ser instalado en cualquier sistema operativo que cumpla con los siguientes requerimientos:

- Sistema Operativo: Windows 7 o superior
- Procesador mínimo Intel Pentium (800 MHz Intel Pentium)
- Mínimo 1GB en RAM
- Visual Studio IDE, o compatible con Python.
- Exploradores: Internet Explorer 9 y superior

Instalación y Configuración

Para el proceso de instalación de esta aplicación únicamente es necesario tener instalado un IDE que sea compatible con el lenguaje Python para ejecutar la aplicación desde la terminal de este.

No es necesario tener alguna configuración ya que la configuración que trae por determinado el IDE es la necesaria para que el funcionamiento del programa sea posible.

Fragmentos de Código

En esta sección, se proporciona una explicación detallada de los métodos y funciones clave que componen el código del programa. El propósito principal es facilitar la comprensión de cómo funciona el programa, para que las personas que lo utilicen puedan brindar un mejor soporte y tener una comprensión más clara de la lógica subyacente en el software. En otras palabras, se busca documentar y explicar de manera exhaustiva las partes esenciales del código para hacer que su mantenimiento y uso sean más accesibles y comprensibles.

1. Código del método “open_file”:

```
def open_file(self):
    filename = filedialog.askopenfilename(initialdir="/Desktop", title="Selecciona un archivo", filetypes=(("Archivos OLC"
    if filename:
        self.path_file = filename
        with open(self.path_file, 'r') as file:
            self.content_file = file.read()
            self.text_area.delete("1.0", tk.END)
            self.text_area.insert("0.0", self.content_file)
```

Este método inicia su proceso verificando si el usuario ha elegido un archivo. En caso afirmativo, guarda la ubicación del archivo en la variable llamada "path_file". Si el usuario no ha seleccionado ningún archivo, la función no realiza ninguna acción adicional.

Una vez que se ha almacenado la ruta del archivo, se procede a comprobar si la extensión del archivo seleccionado es la correcta, es decir, si es ".olc". Si la extensión no es la adecuada, se mostrará un mensaje de error en la pantalla para informar al usuario del problema.

Si la extensión del archivo es correcta, el método continúa leyendo el contenido del archivo línea por línea y almacena toda esta información en una variable. Posteriormente, se mostrará el contenido completo en el área de texto de la aplicación para que el usuario pueda visualizarlo.

2. Código del método “save_file”:

```
def save_file(self):
    if self.path_file != "":
        with open(self.path_file, 'w') as file:
            file.write(self.text_area.get("1.0", tk.END))
    else:
        self.save_file_as()
```

El método “save_file” funciona de la siguiente manera:

Este método es llamado cuando el usuario presiona el botón “Guardar” que se muestra en la interfaz gráfica de la aplicación y funciona de la siguiente manera:

Guarda todo el contenido que tiene el área de texto de la aplicación en una variable, luego verifica si el contenido pertenece a un archivo abierto por medio de la aplicación y de ser así realiza los cambios que el usuario ingresó en el archivo. En caso de que no haya un archivo abierto el programa le preguntará al usuario si desea guardar el contenido en un archivo nuevo.

3. Código del método “run”

```
def run(self):
    self.console.config(state="normal")
    self.console.delete(1.0, tk.END)
    self.console.insert(tk.END, "")
    self.console.config(state="disabled")
    self.symbols = {}
    self.errors = []
    content_text_area = self.text_area.get("1.0", tk.END)
    if content_text_area == "\n":
        self.show_console()
        messagebox.showerror("¡Error!", "No hay código para ejecutar")
        return
    env = Environment(None, 'Global')
    ast = Ast()
    parser = Parser()
    instructions = parser.interpret(content_text_area)
    self.errors = parser.getErrors()

    if len(self.errors) > 0:
        self.show_errors()
    else:
        for instruction in instructions:
            instruction.execute(ast, env)
        self.errors.extend(ast.getErrors())
        if len(self.errors) > 0:
            self.show_errors()
        else:
            self.symbols.update(env.symbols_table)
            self.symbols.update(env.functions)
```

Este método se encarga de examinar el texto ingresado en el área de texto. En primer lugar, verifica si hay errores en el código. Si encuentra errores serán mostrados en una tabla de errores.

Si no se detectan errores, el método procede a mostrar los resultados obtenidos del código ingresado en la consola de entrada, en la consola de salida..

4. Código de la gramática:

```
def p_declaration_type_value(t):
    '''declaration : data_symbol ID COLON data_type EQUAL expression SEMICOLON'''
    params = get_position(t)
    t[0] = Declaration(t[1], t[2], t[4], t[6], params.line, params.column)
```

El código anterior define la gramática de una instrucción declaración en el lenguaje OLC. Las reglas definen el orden correcto en el que la instrucción debe estar escrita. Verifica que primero venga un tipo de símbolo el cual puede ser “var” o “const”, luego verifica que venga un id, dos puntos, el tipo de dato el cual puede ser entero, decimal, cadena, carácter o booleano. Por último verifica que venga el signo igual, una expresión y punto y coma.

5. Código del método “saveSymbol”:

```
def saveSymbol(self, ast, id, symbol):
    if id in self.symbols_table:
        ast.setErrors(Error("Semantico", f"Variable {id} ya existe", self.id, symbol.line, symbol.column))
        return
    self.symbols_table[id] = symbol
```

Este método se encarga de almacenar las variables, constantes y arreglos en un diccionario para luego poder ser accedidos a ellos cuando sea necesario.

6. Código de la clase “Ternary”:

```
class Ternary(Expression):
    def __init__(self, exp1, exp2, exp3, line, column):
        self.exp1 = exp1
        self.exp2 = exp2
        self.exp3 = exp3
        self.line = line
        self.column = column

    def execute(self, ast, env):
        condition = self.exp1.execute(ast, env)
        if condition.value:
            return self.exp2.execute(ast, env)
        return self.exp3.execute(ast, env)
```

La clase “Ternary” obtiene como parámetros tres expresiones y línea y columna. Su funcionamiento es el siguiente:

Si la primera expresión se cumple ejecuta la segunda expresión y si de lo contrario la primera expresión no se cumple se ejecuta la tercera expresión.