Nex Programming Language

A powerful, interpreted programming language built with Python.

Team Members

Jomar Lazaro
Jarren Aceret
Derek Luis Lagadon
Gabriel Andrei Adajar

Introduction

Nex features:

- Clean syntax
- Dynamic typing
- Modular architecture

Getting Started

Prerequisites

- Python 3.x

Running the Shell

To start the interactive REPL:

```
python shell.py
```

Variables & Data Types

Nex supports dynamic typing. Use `VAR` to declare variables.

```
VAR age = 25
VAR pi = 3.14
VAR name = "Nex"
VAR numbers = [1, 2, 3]
```

Arithmetic Operators

- `+` : Addition
- `-` : Subtraction
- `*` : Multiplication
- `/` : Division
- `^` : Power

Comparison & Logical Operators

Comparison	Logical
`==` Equal	`AND` Logical AND
`!=` Not Equal	`OR` Logical OR
`<` Less than	`NOT` Logical NOT
`>` Greater than	
`<=` Less or equal	
`>=` Greater or equal	

Control Flow: If-Elif-Else

```
VAR x = 10

IF x > 5 THEN
    PRINT("Greater than 5")
ELIF x == 5 THEN
    PRINT("Equal to 5")
ELSE
    PRINT("Less than 5")
END
```

Loops

For Loops

```
FOR i = 0 TO 4 THEN
    PRINT(i)
END
```

While Loops

```
VAR i = 0
WHILE i < 5 THEN
    PRINT(i)
    VAR i = i + 1
END
```

Functions

Block Syntax

```
FUN add(a, b)
    RETURN a + b
END
```

Arrow Syntax

```
FUN multiply(a, b) -> a * b
```

Built-in Functions

- `PRINT(value)`
- `INPUT()`, `INPUT_INT()`
- `LEN(list)`, `APPEND(list, value)`, `POP(list, index)`
- `IS_NUM(value)`, `IS_STR(value)`, `IS_LIST(value)`
- `RUN(filename)`

Project Structure

The project is modularized for maintainability:

- **Core**: `nex.py`, `shell.py`
- **Lexer**: `lexer/lexer.py`, `lexer/tokens.py`
- **Parser**: `parser/parser.py`, `parser/nodes.py`, `parser/grammar.txt`
- **Interpreter**: `interpreter/interpreter.py`, `interpreter/values.py`
- **Utils**: `utils/errors.py`, `utils/constants.py`, `utils/string_with_arrows.py`

Project Highlights

- **Modular Architecture**: Decoupled Lexer, Parser, and Interpreter for better scalability.
- **Organized File Structure**: Logical separation of concerns into `lexer`, `parser`, `interpreter`, and `utils` directories.
- **Centralized Error Handling**: Unified error reporting mechanism for consistent debugging.

Conclusion

The **Nex** programming language project successfully demonstrates the implementation of a functional interpreter using Python.

Key Takeaways:

- Practical application of lexing, parsing, and interpreting.
- Understanding of language design principles.
- Creation of a modular and extensible codebase.

Thank You!

Start coding with **Nex** today.