2024_compiler_final - Mini Lisp Interpreter

Overview

This project implements a basic Lisp-inspired interpreter in Python using `lark`. It supports arithmetic, logical operations, variable definitions, functions, conditionals, and printing. The interpreter parses code, builds an Abstract Syntax Tree (AST), and evaluates it.

Features

1. **Arithmetic Operations**: `+`, `-`, `*`, `/`, `mod`, comparisons like `>`, `<`, `=`.
2. **Logical Operations**: `and`, `or`, `not`.
3. **Variables**: Define using `(define var_name value)`.
4. **Conditionals**: `(if condition true_branch false_branch)`.
5. **Functions**: Anonymous: `(fun (params) body)`; Named: `(define name (fun (params) body))`.
6. **Printing**: `(print-num value)`, `(print-bool value)`.

Requirements

- Python 3.7+
- `lark` library (`pip install lark`)
- Vscode 2021

Run

type `python mini.py test_data.lsp` on terminal

Implementation

Basic

- [x] **Syntax Validation** (10 分)
- [x] **Print** (10 分)

- [x] **Numerical Operations** (25 分
- [x] **Logical Operations** (25 分)
- [x] **if Expression** (8 分)
- [x] **Variable Definition** (8 分)
- [x] **Function** (8 分)
- [x] **Named Function** (6 分)

Bonus

- [ ] **Recursion** (5 分)
- [x] **Type Checking** (5 分)
- [ ] **Nested Function** (5 分)
- [ ] **First-class Function** (5 分)

Usage

1. **Write Code**: Create a `.lsp` file. Example:

   ```
   (define x 10)
   (define y 20)
   (print-num (+ x y))
   ```

2. **Run Interpreter**:

   ```
   python mini_lisp.py <file.lsp>
   ```

   Replace `<file.lsp>` with your file path.

3. **Output**: Results will print to the console.

Code Structure

- `mini_lisp.py`: Main script for parsing, transforming, and interpreting.
- `grammar_v2.lark`: Defines Mini Lisp syntax.

Components

1. **AST Nodes**: Represent elements like numbers, variables, operations.
2. **Transformer**: Converts parsed syntax into an AST.
3. **Interpreter**: Evaluates the AST based on language rules.

Syntax Examples

Variables

```
(define x 42)
```

Functions

```
(define add (fun (a b) (+ a b)))
(add 10 20)
```

Conditionals

```
(if (> x 0) (print-num x) (print-num (- x)))
```

Error Handling

- **Syntax Errors**: Prints `syntax error` and exits.
- **Type Errors**: Prints `Type Error` and exits.
- **Undefined Variables/Functions**: Raises an error.

Examples

Arithmetic

```
(define a 5)
(define b 10)
(print-num (+ a b))
```

Output:

```
15
```

Logic

```
(print-bool (and #t #f))
(print-bool (or #t #f))
(print-bool (not #t))
```

Output:

```
#f
#t
#f
```

Functions

```
(define abs (fun (x) (if (< x 0) (- 0 x) x)))
(print-num (abs -42))
```

Output:

```
42
```