YARTBML Language Documentation

YARTBML (Yet Another Re-implementation of Thornsten Ball's Monkey Language) is an interpreter inspired by Thornsten Ball and his book, "Writing An Interpreter in Go". Designed to serve as both an education tool and a practical interpreter, YARTBML builds upon the foundations laid by Ball to explore the intricacies of interpreter design and implementation using Go.

YARTBML is more than just an interpreter; it is a gateway to understanding the underlying mechanisms that make programming languages work. By re-implementing the Monkey language, we aim to provide a hands-on experience that helps programmers at various levels of expertise better understand compilers and interpreters. Whether you are a seasoned developer looking to deepen your understanding of language design, or a beginner eager to learn about the inner workings of interpreters, YARTBML offers a rich, engaging platform to explore and experiment.

Overview

Monkey is a dynamically typed programming language with a syntax akin to JavaScript, enriched with features like first-class functions, closures, and a robust standard library. At its core, YARTBML supports variables, functions, and data structures, enabling users to write expressive code that captures the essence of algorithmic thought. The programming paradigm is heavily inspired by functional languages and it achieves complete compatibility with the SMoL (standard model of languages) spec.

This documentation delves into YARTBML's syntax, built in functionalities, and distinctive features.

Getting Started with YARTBML

In order to get started with YARTBML, ensure you have a Go development environment ready, as the interpreter is developed in Go. Source code for YARTBML can be obtained from the GitHub repository. Follow the provided build instructions to compile the YARTBML interpreter.

github.com/dineshUmasankar/YARTBML

To compile our language, run the following commands from the root of our repository to build a final executable for your host machine.

```
cd internal
go build .
```

Language Features

Semicolons

If C is good enough to have semicolons, then so is our language. Every statement must terminate with a semicolon and any expression that returns a value also terminates with a semicolon.

Data Types

YARTBML supports several primary data types, including:

- Integers: Whole numbers without a decimal component, e.g., 42, -7.
- Booleans: Logical type representing true or false.
- Strings: A sequence of characters enclosed in double quotes, e.g., “YARTBML is awesome!”.
- Arrays: A list of elements, e.g., [1, 2, 3, 4].
- Hashmaps: Key-value pairs, allowing for efficient data lookup, e.g., {“name”: “YARTBML”, “type”: “Interpreter”}.

Variables

Variables in YARTBML are declared using the `let` keyword, enabling the storage and manipulation of values:

```
let version = "1.0.0";
let description = "YARTBML.";
```

Functions

YARTBML treats functions as first-class citizens, allowing them to be assigned to variables, passed as arguments, and returned from other functions:

```
let greet = fn(name) { return "Hello, " + name + "!"; };
let message = greet("World");
puts(message);
```

Control Structures

YARTBML incorporates control structures such as if-else conditionals to direct the flow of execution based on logical conditions:

```
let age = 18;
if (age >= 18) {
     puts("Adult");
} else {
     puts("Minor");
};
```

Datatypes in Action

Our arrays are immutable and only get reassigned when a push or pop function's output is re-assigned to the original identifier that holds an array.

```
let arr = [1, 2, 3];
let arr = push(arr, 4);
puts(arr);
```

The same idea applies to our hashmaps as well.

```
let team = { "name": "Dinesh" };
```

Built-in Functions

YARTBML enriches the Monkey language with a suite of built-in functions designed to facilitate common programming tasks:

- len(s): Determines the length of a string or array s.
- put(s): Outputs the string representation of s to the console.
- first(a), last(a), rest(a), push(a, e): Array manipulation functions for accessing and modifying array elements.

Examples

Let's demonstrate a simple output in YARTBML: Hello World!

```
puts("Hello, World from YARTBML!");
```

We can also implement the Fibonacci sequence to showcase function recursion in YARTBML:

```
let results = [];
let fibonacci = fn(x) {
    if (x == 0) {
        return 0;
    } else {
        if (x == 1) {
            return 1;
        } else {
            fibonacci(x - 1) + fibonacci(x - 2);
        };
    };
};

let results = push(results, fibonacci(5));
puts(results);
```

Conclusion

YARTBML offers a unique perspective on the Monkey programming language, inspired by Thorsten Ball's insightful exploration into interpreter writing. Through its enhanced feature set and optimizations, YARTBML not only pays homage to Ball's original work but also extends its educational and practical applications for enthusiasts and professionals alike.

Further Reading

For those interested in diving deeper into the concepts behind YARTBML and the original Monkey language, Thorsten Ball's "Writing An Interpreter In Go" provides an excellent foundation. Additionally, our GitHub repository hosts the YARTBML source code, offering a hands-on experience in interpreter development and the Monkey programming language.