RocketLang

Mein eigener kleiner BER

Chemnitzer Linux-Tage 2024

/home/robert

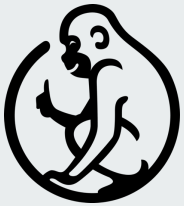
- Robert ‘Flipez’ Müller
- Senior Site Reliability Engineer bei Mozilla
- Software- & Site Reliability Engineer bei Hetzner (Cloud)
- Film- und Fotografie
- Spezi <3 (spezi.auch.cool)

Mail: robert@auch.cool
"Blog": https://auch.cool

1. Sprache ausdenken
2. Interpreter schreiben
3. ???
4. Profit

Monkey

“The programming language that lives in books”



- **Programmiersprache**
- **Tree-walk Interpreter in Go**
- **Optional mit Compiler und VM**

Monkey is a programming language that you can build yourself by reading through Writing An Interpreter In Go and Writing A Compiler In Go.

There is no official implementation of Monkey — it lives in these books and it's up to you, the reader, to implement it. First as a tree-walking interpreter, then as a bytecode compiler and virtual machine.

RocketLang

"It's not rocket science. Is it?"

- Programmiersprache
- Tree-walk Interpreter in Go
- Basiert auf Monkey
- Orientiert sich an Ruby

Started as a joke to learn the inner mechanics of an interpreted language, RocketLang quickly evolved into a long running project with more and more ambitious goals, such as being used for Advent of Code to test its limits.

Was ist ein Interpreter?

Interpreter

- Lexer
  - Wandelt Source Code in Tokens um
- Parser
  - Wandelt Tokens in einen Abstract Syntax Tree (AST) um
- Evaluator (Interpreter)
  - Evaluiert den AST und führt den Code aus

Interpreter - Lexer

- Führt Lexikalische Analyse durch
- Zerteilt den Source Code in einzelne Teile (Tokens)
- Kann unter Umständen automatisch generiert werden

Interpreter - Lexer

```
"let x = 5 + 5;"
```

```
[
  LET,
  IDENTIFIER("x"),
  EQUAL_SIGN,
  INTEGER(5),
  PLUS_SIGN,
  INTEGER(5),
  SEMICOLON
]
```

Interpreter - Lexer

```
// lexer/lexer.go

type Lexer struct {
  input            string
  position         int
  readPosition     int
  ch               byte
  currentLine      int
  positionInLine   int
  file             string
}
```

Interpreter - Tokens

- Enthalten den tatsächlichen Quellcode
  - z.B. + bei PLUS, let bei LET usw.
- Tokens für
  - Zahlen
  - Namen der Variablen (identifier)
  - Worte die wie Variablen aussehen aber keine sind (keywords, z.B. let oder def)
  - Sonderzeichen (Klammern, Kommata, Semikolon, …)
- Jeder Typ einzeln, aber ohne Gruppieren

Interpreter - Tokens

```
// token/token.go (RocketLang 0.22)

type TokenType string

type Token struct {
  Type          TokenType
  Literal       string
  LineNumber    int
  LinePosition  int
  File          string
}
```

Interpreter - Tokens

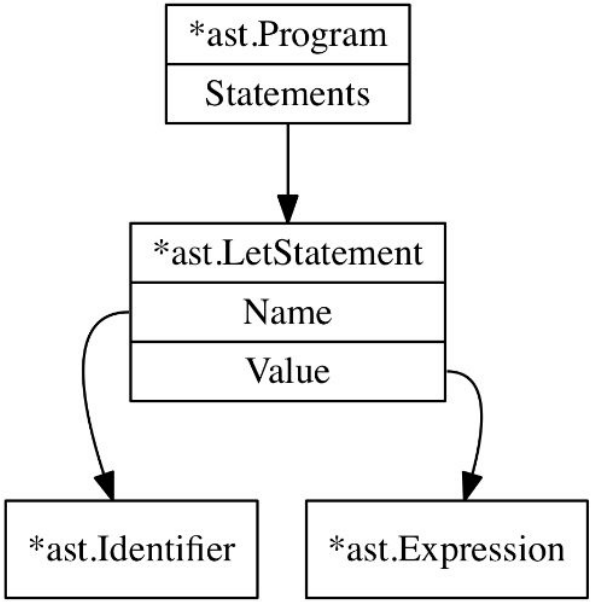
```
// token/token.go (RocketLang 0.22)

const (
  ILLEGAL = "ILLEGAL"
  EOF     = "EOF"

  IDENT  = "IDENT" // add, foobar, x, y
  INT    = "INT"   // 123456
  FLOAT  = "FLOAT" // 123.456
  STRING = "STRING"

  ASSIGN   = "="
  PLUS     = "+"
  MINUS    = "-"
  BANG     = "!"
  ASTERISK = "*"
  SLASH    = "/"
  PERCENT  = "%"

  QUESTION = "?"
...
```

Interpreter - Parser

- Erstellt eine Struktur aus Tokens (AST)
- Kümmert sich um Präzedenz

Interpreter - Parser

```
let x = 5;
```



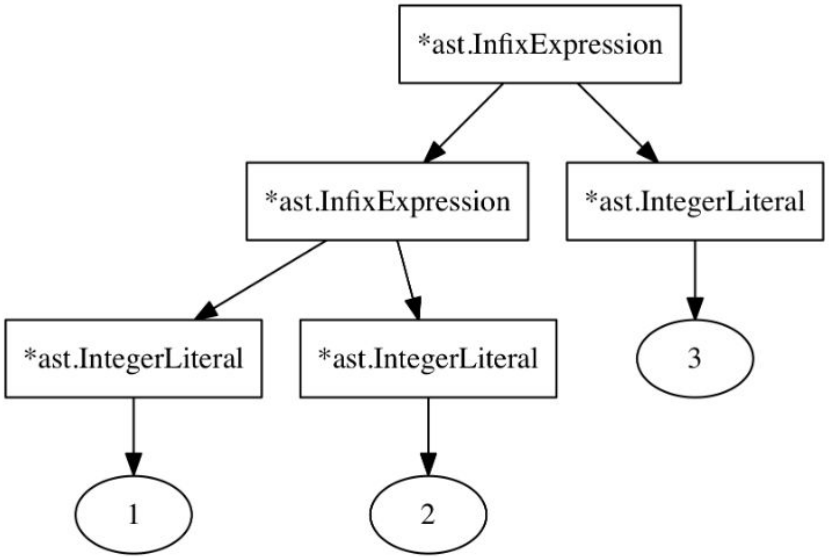
Interpreter - Parser

```
tests := []struct {
  input    string
  expected string
}{
  {
    "-a * b",
    "((-a) * b)",
  }, {
   "!-a",
   "(!(-a))",
  }, {
    "a + b + c",
    "((a + b) + c)",
  }, {
    "a + b - c",
    "((a + b) - c)",
  }, {
    "a * b * c",
    "((a * b) * c)",
  },
```

Interpreter - Parser

```
1 + 2 + 3;
```

```
((1 + 2) + 3)
```



Interpreter - Evaluator

- Evaluiert den AST
  - 1 + 2 sollte 3 ergeben
  - 5 < 1 sollte false sein
- Trifft die Entscheidung wie sich etwas verhält
  - Ist 5 true oder false?

Interpreter - Evaluator

```
func evalPrefixExpression(operator string, right object.Object) object.Object {
  switch operator {
  case "!":
    return evalBangOperatorExpression(right)
  case "-":
    return evalMinusPrefixOperatorExpression(right)
  default:
    return object.NewErrorFormat("unknown operator: %s%s", operator, right.Type())
  }
}
```

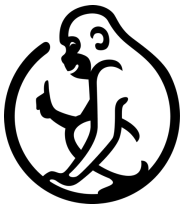
Interpreter - Evaluator

```
func evalBangOperatorExpression(right object.Object) object.Object {
  switch right {
  case object.TRUE:
    return object.FALSE
  case object.FALSE:
    return object.TRUE
  case object.NIL:
    return object.TRUE
  default:
    return object.FALSE
  }
}
```

1. Sprache ausdenken
2. Interpreter schreiben ✔
3. ???
4. Profit

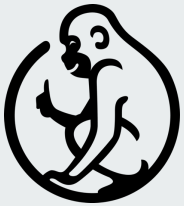
Die Monkey Sprache

- Basiert auf dem Buch “Writing An Interpreter In Go”
- Implementiert die Sprache als einen tree-walking Interpreter
- Bringt ein paar sinnvolle Features mit
    - Integer, Boolean, String, Array, Hash (Maps, Dicts..)
    - Eine Read-eval-print-loop (REPL)
    - Rechenoperationen
    - Built-in functions
    - Rekursion
    - Closures



Monkey Code

Standard Dinge

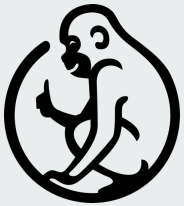


```
// Integers & arithmetic expressions...
let version = 1 + (50 / 2) - (8 * 3);

// ... and strings
let name = "The Monkey programming language";

// ... booleans
let isMonkeyFastNow = true;

// ... arrays & hash maps
let people = [
  {"name": "Anna", "age": 24},
  {"name": "Bob", "age": 99}
];
```

Aus https://monkeylang.org/

Monkey Code

Funktionen

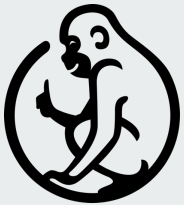


```
// User-defined functions...
let getName = fn(person) {
  person["name"];
};

getName(people[0]); // => "Anna"
getName(people[1]); // => "Bob"

// and built-in functions
puts(len(people))  // prints: 2
```

Aus https://monkeylang.org/

Monkey Code

Conditionals

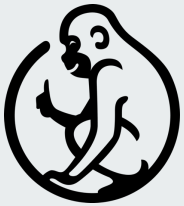


```
let fibonacci = fn(x) {
  if (x == 0) {
    0
  } else {
    if (x == 1) {
      return 1;
    } else {
      fibonacci(x - 1) + fibonacci(x - 2);
    }
  }
};
```

Aus https://monkeylang.org/

Monkey Code

Closures



```
// `newAdder` returns a closure that makes
use of the free variables `a` and `b`:

let newAdder = fn(a, b) {
    fn(c) { a + b + c };
};


// This constructs a new `adder` function:
let adder = newAdder(1, 2);

adder(8); // => 11
```

Aus https://monkeylang.org/

RocketLang 0.9.5 == Monkey

1. Sprache ausdenken ?
2. Interpreter schreiben ✔
3. ???
4. Profit

RocketLang in der “echten” Welt

RocketLang is done (as far as the book is concerned) and I tried to solve a real problem with it. It was quite a journey to get it (sort) of working due to a lot of missing stuff (some crucial, some convenience). So I want to share this piece of code to get a better feeling what is currently missing, what is possible and what not.

```
$ rocket-lang examples/aoc2015_day1.rl
Solution Day 1 Part 1:
127
null
```

Aus https://github.com/Flipez/rocket-lang/pull/2

RocketLang in der “echten” Welt

1. no string[idx] -> I had to convert the input string into an array[]
2. ~~no string comparison ("(" == "(" is currently false) -> a Hash/Dictionary worked~~
    - was fixed in 0.9.6-rc2
3. no loops -> recursion worked
4. no exit or raise
5. running the script always outputs null at the end
6. no comments -> strings everywhere
7. the variable scope is sometimes a bit weird, I had to use more let than I felt was necessary

Aus https://github.com/Flipez/rocket-lang/pull/2

RocketLang in der “echten” Welt

An opening parenthesis, (, means he should go up one floor, and a closing parenthesis,), means he should go down one floor.

For example:

- *(()) and ()() both result in floor 0.*
- *(((and (()(()(both result in floor 3.*
- *))(((((also results in floor 3.*
- *()) and))(both result in floor -1 (the first basement level).*
- *))) and)())()) both result in floor -3.*

Aus https://adventofcode.com/2015/day/1

RocketLang in der “echten” Welt

```
STEPS = INPUT.chars

final_floor = STEPS.count("(") - STEPS.count(")")
solve!("Final floor:", final_floor)
```

Aus https://github.com/ahorner/advent-of-code/blob/main/lib/2015/01.rb

RocketLang in der “echten” Welt

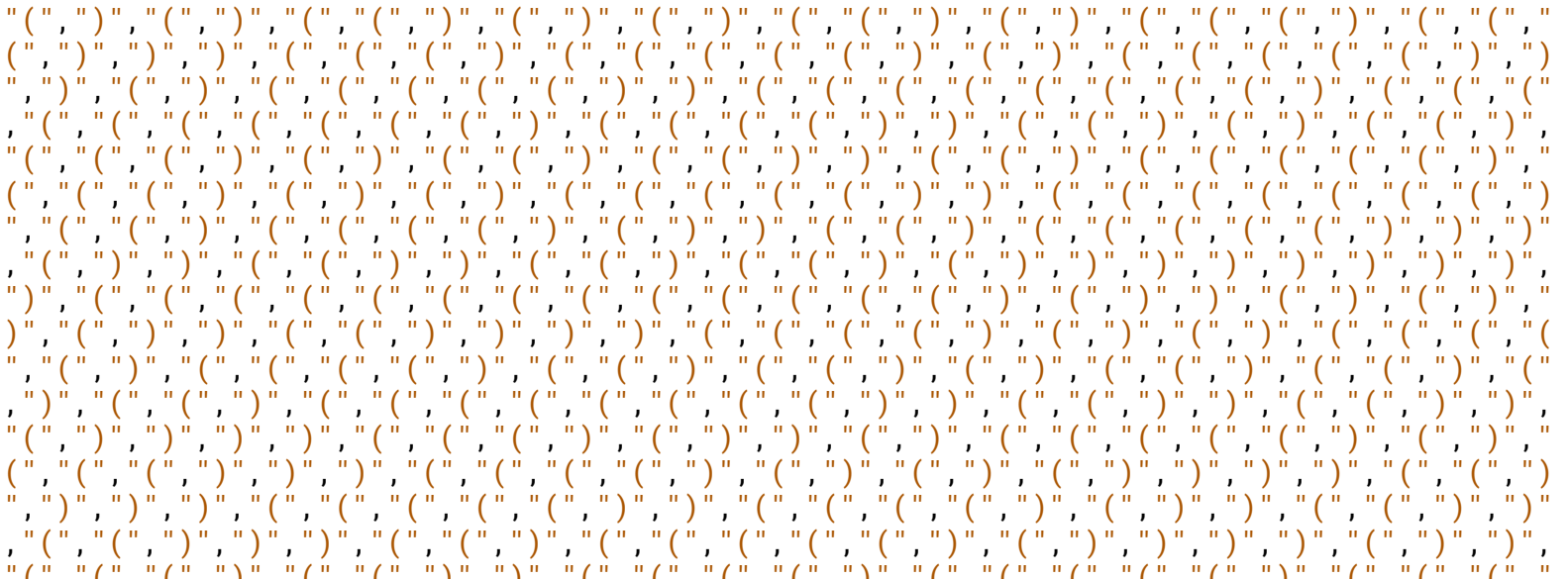
```
let part_one = fn(input) {
  let floor = 0
  let idx = len(input)

  "// There are no loops at the moment
   so use recursion instead"
  calc(input, idx, floor)
}
```

```
let calc = fn(input,idx,floor) {
  let char_to_value = {
    "(": 1,
    ")": -1
  }

  if (idx == 0) {
    return floor
  }

  let new_idx = idx - 1
  let delta = char_to_value[input[new_idx]]

  calc(input, new_idx, floor + delta)
}

puts("Solution Day 1 Part 1: ")
puts(part_one(real_input))
```

Aus https://github.com/ahorner/advent-of-code/blob/main/lib/2015/01.rb

RocketLang in der “echten” Welt

```
"// There aren't any useful string operations in RocketLang yet so you have to prepare"
"// the input code a bit - in this case as an array of strings, like this:"
let real_input = [
```



RocketLang in der “echten” Welt

```
STEPS = INPUT.chars

final_floor = STEPS.count("(") - STEPS.count(")")
solve!("Final floor:", final_floor)
```

```
let part_one = fn(input) {
  let floor = 0
  let idx = len(input)

  "// There are no loops at the moment so use recursion instead"
  calc(input, idx, floor)
}

let calc = fn(input,idx,floor) {
  let char_to_value = {
    "(": 1,
    ")": -1
  }

  if (idx == 0) {
    return floor
  }

  let new_idx = idx - 1
  let delta = char_to_value[input[new_idx]]

  calc(input, new_idx, floor + delta)
}


puts("Solution Day 1 Part 1: ")
puts(part_one(real_input))
```

Aus https://github.com/ahorner/advent-of-code/blob/main/lib/2015/01.rb

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures
- support for index operator on string

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures
- support for index operator on string
- support for thumbs-up, thumbs-down and plus emoji

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures
- support for index operator on string
- support for thumbs-up, thumbs-down and plus emoji

```
var emojis = map[string]TokenType{
      "👍": TRUE,
      "👎": FALSE,
      "➕": PLUS,
}
```

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures
- support for index operator on string
- support for thumbs-up, thumbs-down and plus emoji
- basic ability to parse files
- add pop() builtin
- add `exit()` builtin
- add raise() builtin
- add single line comments
- add support for method call to objects

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures
- support for index operator on string
- support for thumbs-up, thumbs-down and plus emoji
- basic ability to parse files
- add pop() builtin
- add `exit()` builtin

- add raise() builtin
- add single line comments
- add support for method call to objects
- add support for array yeet and yoink
- add array.yoink()
- enhance string.plz_i()

Monkey vs RocketLang

- Integer, Boolean, String, Array, Hash (Maps, Dicts..)
- Eine Read-eval-print-loop (REPL)
- Rechenoperationen
- Built-in functions
- Rekursion
- Closures
- support for index operator on string
- support for thumbs-up, thumbs-down and plus emoji
- basic ability to parse files
- add pop() builtin
- add `exit()` builtin

- add raise() builtin
- add single line comments
- add support for method call to objects
- add support for array yeet and yoink
- add array.yoink()
- enhance string.plz_i()
- add foreach
- object: make array and hash hashable
- object/hash: add values method
- object: add generic method for object comparison
- object/array: add uniq, index, first and last

RocketLang 0.22.0

```
🚀 » puts("hello from rocket-lang!")
"hello from rocket-lang!"
» nil

🚀 » langs = ["ruby", "go", "crystal", "python", "php"]
» ["ruby", "go", "crystal", "python", "php"]

🚀 » langs.pop()
» "php"

🚀 » langs.push("rocket-lang")
» nil

🚀 » langs
» ["ruby", "go", "crystal", "python", "rocket-lang"]
```

RocketLang 0.22.0

JSON

```
🚀 » JSON.parse('{"test": 123}')
» {"test": 123.0}

🚀 » a = {"test": 1234}
» {"test": 1234}

🚀 » a.to_json()
» '{"test":1234}'
```

RocketLang 0.22.0

HTTP Server

```
def test()
  puts(request["body"])
  return("test")
end


HTTP.handle("/", test)


HTTP.listen(3000)
```

RocketLang 0.22.0

Math Builtin

```
🚀 » Math.E
» 2.718281828459045
```

```
🚀 » Math.Pi
» 3.141592653589793
```

```
🚀 » Math.sqrt(3.0 * 3.0 + 4.0 * 4.0)
» 5.0
```

RocketLang 0.22.0

Time Builtin

```
🚀 » Time.format(Time.unix(),
                 "Mon Jan _2 15:04:05 2006")
» "Mon Oct 31 00:08:10 2022"

🚀 » Time.format(Time.unix(),
                 "%a %b %e %H:%M:%S %Y")
» "Mon Oct 31 00:28:43 2022"


🚀 » Time.parse("2022-03-23", "2006-01-02")
» 1647993600

🚀 » Time.parse("2022-03-23", "%Y-%m-%d")
» 1647993600
```

RocketLang 0.22.0

Everything is an object

```
🚀 » "test".type()
» "STRING"

🚀 » true.to_s()
» "true"

🚀 » 1.4.to_s()
» "1.4"
```

Sprache ausdenken

- Monkey als Basis
- Erweiterung mit notwendigen sowie komfortablen Features
  - Schleifen
  - Umwandlung zwischen Typen
  - Mitgelieferte Bibliotheken (Time, Math)
- Wie soll sich X verhalten?
  - z.B jeder Ausdruck returned etwas (zB letzter Wert oder nil)
- Orientierung an Ruby
  - Sowohl Syntax als auch Verhalten

Sprache ausdenken

- Umbenennen von bestehenden Funktionen
  - let streichen
  - fn -> def
  - plz_i(),plz_s() -> to_i(), to_s()
  - [1, 2].yoink(), [1, 2].yeet() -> [1, 2].push(), [1, 2].pop()
- Sinnvolles Errorhandling

```
🔥 Great, you broke it!
 parser errors:
        15:13: no prefix parse function for , found
        15:18: no prefix parse function for , found
        15:25: no prefix parse function for ) found
        30:11: expected assign token to be IDENT, got if instead


🚀 > "test".nope()
=> ERROR: undefined method `.nope()` for STRING
```

Stdlib: All the builtins!

- exit()
- raise()
- HTTP
- JSON

Stdlib: All the builtins!

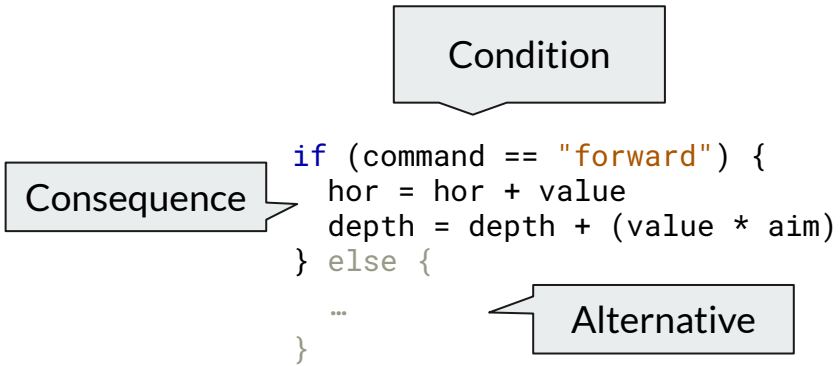
```
func evalIdentifier(node *ast.Identifier, env *object.Environment) object.Object {
      if val, ok := env.Get(node.Value); ok {
            return val
      }

      if fn, ok := stdlib.Functions[node.Value]; ok {
            return fn
      }

      if mod, ok := stdlib.Modules[node.Value]; ok {
            return mod
      }

      return object.NewErrorFormat("%s:%d:%d: identifier not found: " …)
}
```

Playground Beispiel

Stdlib: All the builtins!

```
func raiseFunction(_ object.Environment, args ...object.Object) object.Object {
      return object.NewError(args[0].(*object.String).Value)
}


RegisterFunction(
  "raise",
  object.MethodLayout{
    ArgPattern:    object.Args(object.Arg(object.STRING_OBJ)),
    ReturnPattern: object.Args(object.Arg(object.ERROR_OBJ)),
  },
  raiseFunction,
)
```

Feat: Curly Braces Optional




```
if (command == "forward")
  hor = hor + value
  depth = depth + (value * aim)
end
```

Feat: Curly Braces Optional

```
func (p *Parser) parseIfExpression() ast.Expression {
  // Setup and parse condition
  …
  if !p.expectPeek(token.LBRACE) {
    return nil
  }

  expression.Consequence = p.parseBlockStatement()

  if p.curTokenIs(token.RBRACE) {
    p.nextToken()
  }

  if p.curTokenIs(token.ELSE) {
    p.nextToken()
    if !p.expectPeek(token.LBRACE) {
      return nil
    }

    expression.Alternative = p.parseBlockStatement()

  }
  return expression
}
```

Feat: Curly Braces Optional

```
if !p.expectPeek(token.LBRACE) {
  return nil
}
```

```
if p.peekTokenIs(token.LBRACE) {
  p.nextToken()
}
```

Bug: Print ist seltsam

MarkusFreitag commented on Dec 10, 2022 Collaborator

While `puts` is useful for debugging, it can not really be used for writing stuff to the terminal. The main reason for that is how strings are handled. Lets discuss whether this should change or it stays this way and we add another simple way of writing to the terminal.

```
// what you get
🚀 » puts("1\n2")
"1\n2"
» nil
// what you expect
🚀 » puts("1\n2")
1
2
» nil
```

Bug: Print ist seltsam

```
func putsFunction(env object.Environment, args ...object.Object) object.Object {
	for _, arg := range args {
		fmt.Println(arg.Inspect())
	}

	return object.NIL
}
```

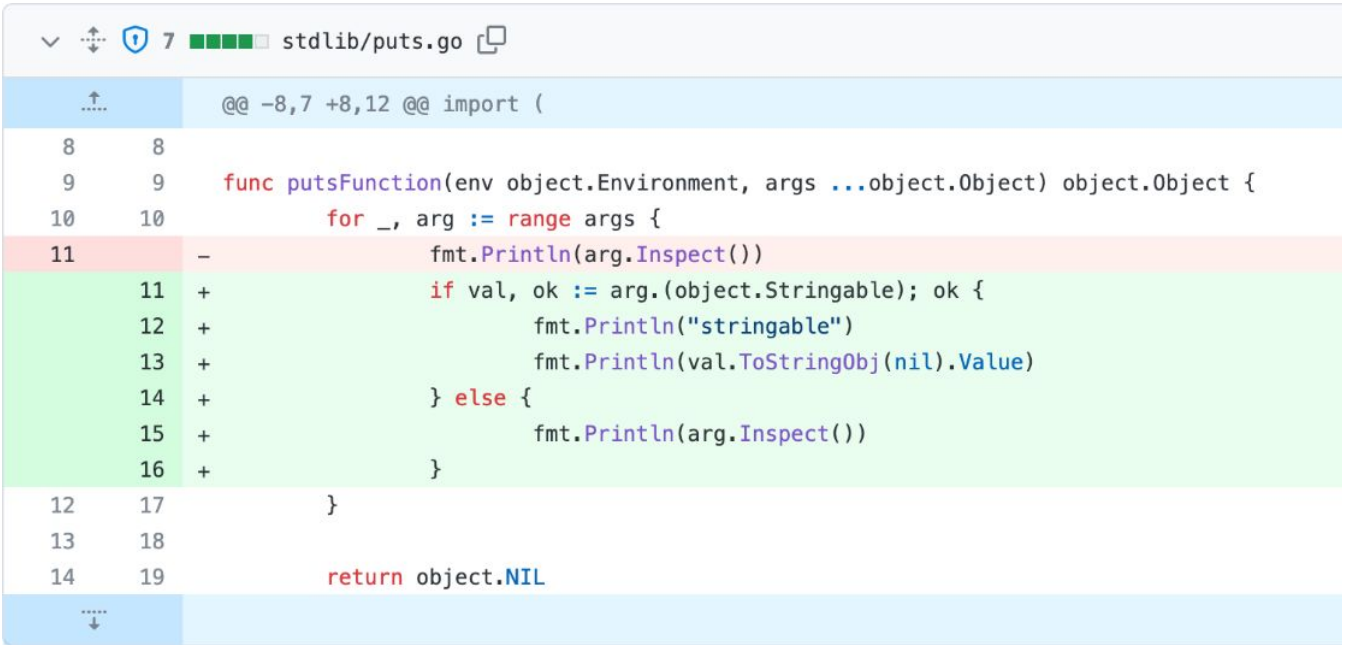
Bug: Print ist seltsam

```
func (b *Boolean) Inspect() string  { return fmt.Sprintf("%t", b.Value) }

func (f *File) Inspect() string  { return fmt.Sprintf("<file:%s>", f.Filename) }

func (f *Float) Inspect() string  { return f.toString() }
```

Bug: Print ist seltsam

```
func (s *String) Inspect() string {
	var output string

	for _, char := range s.Value {
		if char == '"' {
			output += string('\\')
		}

		output += string(char)
	}

	return `"` + output + `"`
}
```

Bug: Print ist seltsam



Was fehlt noch?

- Unterstützung für Klassen
- include/require von anderen Dateien (in wirklich funktionierend)
- Unterstützung für Module/Pakete (ala pip, gems)
- Multiple Return Values
- Mehr Funktionen/Module in der Stdlib

1. Sprache ausdenken ✔
2. Interpreter schreiben ✔
3. ???
4. Profit

Organisation

- Versionskontrolle auf GitHub (https://github.com/Flipez/rocket-lang)
- Issues und Pull Requests aktiv nutzen
- Semantic Versioning
- Tests (!)
- Dokumentation
- Playground

Issues und Pull Requests

- Bei jedem noch so kleinen Problem anlegen
- Am besten mit Beispiel
  - Was habe ich gemacht ?
  - Was ist passiert ?
  - Was habe ich erwartet ?

Kjarrigan commented on Jan 18, 2022 — Contributor

Stats

- RocketLang: `0.14.1`
- Plattform: `Linux home 5.4.0-94-generic`
- OS: `Ubuntu 20.04.3 LTS`
- Installed RocketLang via: `*.deb Package`

Problem

Withouth the optional second argument open works just fine

```
open("/tmp/foo").content()
```

but adding it like shown in the docs

```
open("main.go", "ro").content()
```

breaks with `ERROR: Failed to invoke method: content`

Issues und Pull Requests

MarkusFreitag commented on Jan 18, 2022 — Collaborator

`ro` is not a valid mode, you can choose from `r, w, wa, rw, rwa`.

The real bug here is that the initial error of `open()` is not shown and it tries to call `content()` at the returned error.

Output

```
ERROR: :1:22: undefined method `.content()` for ERROR
```

Test

- Kosten viel Zeit
- Schwierig alles abzudecken
- Super nervig

- Aber:
- Alles was man testet kann nicht kaputt gehen
- Umso wertvoller je komplexer das “Programm”

Tests

codecov bot commented on Dec 19, 2022 • edited

Codecov Report

> Merging #177 (9e5bcaa) into main (71bb1c4) will **increase** coverage by `0.06%`.
> The diff coverage is `100.00%`.

```
@@            Coverage Diff             @@
##             main     #177      +/-   ##
==========================================
+ Coverage   87.36%   87.42%   +0.06%
==========================================
  Files         106      106
  Lines        3681     3699      +18
==========================================
+ Hits         3216     3234      +18
  Misses        398      398
  Partials       67       67
```

Impacted Files	Coverage Δ	
object/array.go	`100.00% <100.00%> (ø)`	

Dokumentation

- Wichtig um selber nicht zu vergessen wie etwas gedacht war
- Hilft anderen das Programm zu benutzen
- Deckt Fehler schneller auf
    - Auch hilfreich Doku zu schreiben wie etwas sein sollte
    - Wenn Edge-Case auftaucht kann man patchen oder weiter dokumentieren

Dokumentation

- So wenig wie möglich Mehraufwand
  - Wenig doppelt verwalten
  - Doku so nah wie möglich an den Code
- Docusaurus Webseite
- Generische Informationen automatisch als Markdown generieren
  - Alle Literals (String, Integer, usw)
  - Deren Methoden (mit Parametern und Ausgabetyp)

Dokumentation

```
// object/string.go

"replace": ObjectMethod{
  Layout: MethodLayout{
    ArgPattern: Args(
      Arg(STRING_OBJ),
      Arg(STRING_OBJ),
    ),
    ReturnPattern: Args(
      Arg(STRING_OBJ),
    ),
  },
  method: func(o Object, args []Object, _ Environment) Object {
    s := o.(*String)
    oldS := args[0].(*String).Value
    newS := args[1].(*String).Value
    return NewString(strings.Replace(s.Value, oldS, newS, -1))
  },
},
```

Dokumentation

```
# docs/literals/string.yml
 replace:
   description: "Replaces the first string with the second string in the given string."
   example: |
     🚀 » "test".replace("t", "f")
     » "fesf"
```

Dokumentation

replace(STRING, STRING)

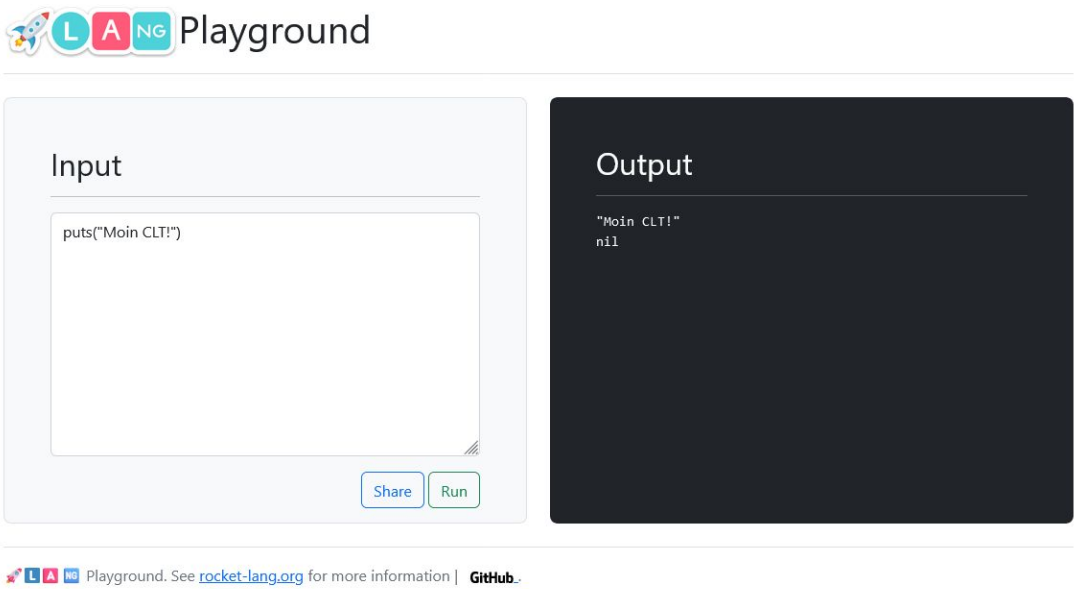
> Returns `STRING`

Replaces the first string with the second string in the given string.

Playground

- Einfaches testen ohne Installation
- Ausführung lokal im Browser
- ‘Share’ Button
  - Perfekt um Bugs nachzuvollziehen
- Kein Mehraufwand
  - WASM Binary
  - Automatisch in der CI gebaut
- https://play.rocket-lang.org/

Playground


Playground
Input
puts("Moin CLT!")
Share
Run
Output
"Moin CLT!"
nil
Playground. See rocket-lang.org for more information | GitHub

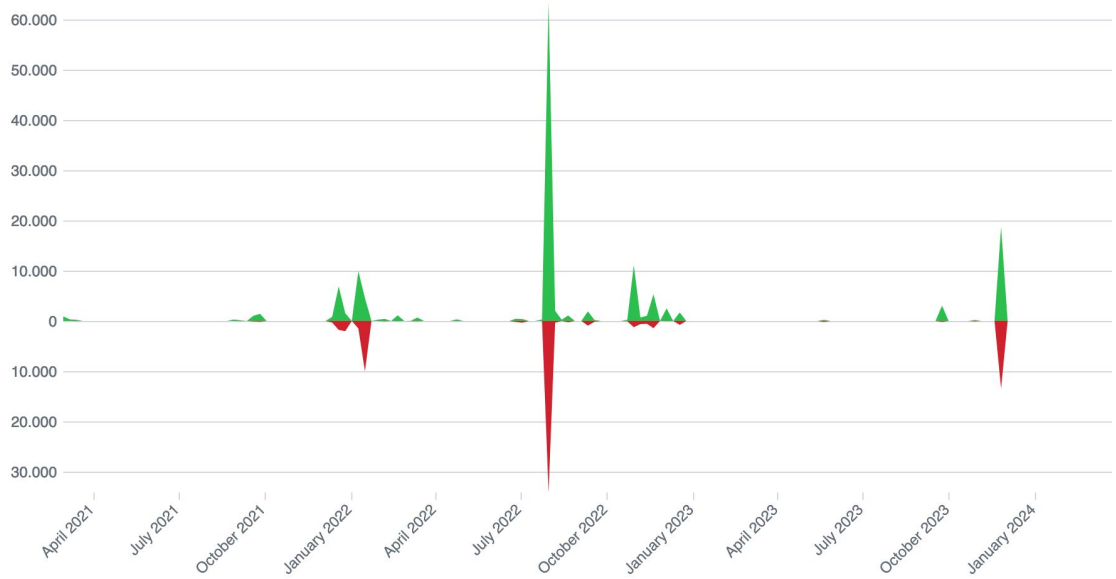
Playground

```
# wasm/vercel.sh

amazon-linux-extras install golang1.11
go build -tags wasm -o main.wasm ../main.go
```

```
// wasm/index.html

this.runButton.addEventListener('click',function(e) {
  window.output.innerText = ""

  const go = new Go();
  go.argv = ['rocket-lang', '-e', window.input.value];
}
```

1. Sprache ausdenken ✔
2. Interpreter schreiben ✔
3. ??? ✔
4. Profit

Auch mal Pause machen



Write more “useless” software

https://ntietz.com/blog/write-more-useless-software/

Vielen Dank!

Fragen?

Mail: robert@auch.cool

Webseite: https://rocket-lang.org
Playground: https://play.rocket-lang.org
GitHub: https://github.com/flipez/rocket-lang