gibica Documentation

Release 0.8.2

Matthieu Gouel

May 11, 2019

Contents

Gibica is an interpreter developped in Python 3 (v3.6+).

The aim of this project is to better understand how a basic interpreter works by implementing a simple language.

CHAPTER 1

Getting Started

1.1 Installation

You can install the package using *pip* or *pipenv*. It requires Python 3.6.0+ to run.

```
pip install gibica
```

That's it ! Now you have everything to build and run your Gibica programs.

1.2 Hello, numbers !

For now Gibica doesn't understand strings so we cannot actually write the traditionnal "Hello, world!" program. Nonetheless, we can write a first piece of code that display a number. Pretty much the same right ?!

```
print(1);
```

Yes that's it. You can save that super fancy program in a file named *hello.gbc* and execute it with the following command :

```
gibica hello.gbc
```

if **1** is printed on you screen, congratulation ! You have just written your first Gibica program !

CHAPTER 2

Main features

2.1 Overview

- Like many programming languages, Gibica impose the semicolon at the end of expressions.

```
print(1);
```

- You can write comments in your programs. Comments are code that will be ignored during the interpretation. To do that, just put the character **#** before your comment.

```
# This is a comment !

print(1); # Here is an other.
```

2.2 Variable and mutability

Gibica uses the dynamic type feature from Python so you don't have to declare explicitly the type of your variables. Let's declare a first variable with an integer number.

```
let integer = 1;
```

By default, all Gibica variables are *immutables*. Yes it seems tough but in fact it protects from many surprises.

I admit that not be able to use mutability on variables can be very unconveniant, so it's possible to explicitly enable the mutabily of a variable at the declaration with the keyword **mut**.

```
let mut interger = 1;
integer = integer + 1;
```

There is currently three implicit types in the Gibica implementation.

Integer type

```
let integer = 10;
```

FLoat type

```
let float = 1.0;
```

Boolean type

```
let boolean1 = true;
let boolean2 = false;
```

2.3 Control flow

For now Gibica provides two types of control flows.

conditional statement

```
let mut result = 0;
let i = 5;
if i <= 4 {
    result = 1;
} else if i == 5  {
    result = 2;
}
else {
    result = 3;
}
```

loop statement

```
let i = 0;
while i < 5 {
    i = i + 1;
}
```

2.4 Functions

Here is a basic example of a function declaration.

```
def add(a, b) {
    return a + b;
}

let result = add(1, 1);
```

Moreover, you can specify the mutability nature of a parameter.

```
def increment(mut n) {
    n = n + 1;
    return n;
}

let result = increment(1);
```

CHAPTER 3

Advanced usage

3.1 Grammar

This is the current grammar of the Gibica language.

```
program: (statement)*

statement: function_declaration
         | variable_declaration
         | expression_statement
         | if_statement
         | while_statement
         | jump_statement

function_declaration: 'def' ID parameters compound

parameters: '(' logical_or_expr (',' logical_or_expr)* ')'

function_body: '{' (statement)* '}'

variable_declaration: 'let' assignment ';'

expression_statement: assignment ';'

assignment: logical_or_expr ['=' logical_or_expr]

if_statement: 'if' logical_or_expr compound
            ('else' 'if' local_or_expr compound)*
            ['else' compound]

while_statement: 'while' local_or_expr compound

compound: '{' (statement)* '}'
```

(continues on next page)

(continued from previous page)

```
jump_statement: 'return' expression_statement

logical_or_expr: logical_and_expr ('or' logical_and_expr)*

logical_and_expr: logical_not_expr ('and' logical_not_expr)*

logical_not_expr: 'not' logical_not_expr
                | comparison

comparison: expr (('==' | '!=' | '<=' | '>=' | '<' | '>') expr)*

expr: term (('+' | '-') term)*

term: atom (('*' | '/' | '//') atom)*

call: ['mut'] ID [parameters]

atom: '+' atom
    | '-' atom
    | call
    | INT_NUMBER
    | FLOAT_NUMBER
    | '(' logical_or_expr ')'
    | TRUE
    | FALSE
```

3.2 Package content

3.2.1 gibica

gibica package

Gibica package.

Submodules

gibica.ast module

gibica.builtins module

gibica.entrypoint module

gibica.exceptions module

Exceptions module.

exception gibica.exceptions.**InterpreterError**
Bases: exceptions.Exception

Interpreter error.

exception gibica.exceptions.**LexicalError**
Bases: exceptions.Exception

Lexical error.

exception gibica.exceptions.**ObjectError**
Bases: exceptions.Exception

Object error.

exception gibica.exceptions.**SementicError**
Bases: exceptions.Exception

Sementic error.

exception gibica.exceptions.**SyntaxError**
Bases: exceptions.Exception

Syntax error.

exception gibica.exceptions.**TypeError**
Bases: exceptions.Exception

Type error.

gibica.interpreter module

gibica.lexer module

gibica.memory module

Memory module.

class gibica.memory.**Frame**(**args*, ***kwargs*)
Bases: list

Frame of *Scope* objects.

current
Get the current scope of the frame.

class gibica.memory.**Memory**(***kwags*)
Bases: object

Memory object representation.

append_frame(***kwargs*)
Create a new frame.

append_scope()
Create a new scope in the current frame.

pop_frame()
Delete the current frame.

pop_scope()
Delete the current scope in the current frame.

class gibica.memory.**Scope**
Bases: dict

Memory scope object.

class gibica.memory.**Stack**(**args*, ***kwargs*)
Bases: `list`

Stack of *Frame* objects.

current
Get the current frame of the stack.

gibica.parser module

gibica.sementic module

gibica.tokens module

gibica.types module

Python Module Index

g

Index