Ratscript Documentation

Release 0.1.1

MousePaw Media

Jun 03, 2022

CONTENTS

1 Syntax and Design **3**
- 1.1 Vision . . . 3
- 1.2 Document Conventions . . . 4

2 Language Structure **5**
- 2.1 Whitespace . . . 5
- 2.2 Comments . . . 6
- 2.3 Subordination . . . 7
- 2.4 Attributes . . . 8
- 2.5 Keywords . . . 9

3 Indices and tables **11**

Ratscript is a programming language based on the idea that a programming language should be *intentionally designed*. It embraces the goals that the language should resemble *common logic* more than *computer logic*, have a shallow initial learning curve to make it easy to grasp by non-programmers, and should foster conscientious programming habits, encouraging less-experienced programmers to default to safer and more effective methods and algorithms.

See Ratscript's `README.md`, `CHANGELOG.md`, `BUILDING.md`, and `LICENSE.md` for more technical information.

Contents:

CHAPTER

ONE

SYNTAX AND DESIGN

1.1 Vision

The design of Ratscript's syntax has three goals:

- Explicit: Make behavior obvious from the syntax.
- Simple: Lower the learning curve for beginning users.
- Elegant: Maintain usability by experienced programmers.

1.1.1 Glossary

- *compound statement*: a statement made up of one or more *clauses*. For example, an `if` statement.
- *clause*; a single header-suite pair in a compound statement, such as the `else` clause in an `if`/`else` compound statement.
- *expression*: a unit of code that can be evaluated to a value.
- *header*: the top part of a clause, usually defining how and when the clause will be executed.
- *name*: a textual reference to a value in memory, in the context of a variable. Names have both scope and type, and are *bound* to a value.
- *statement*: a single, executable unit of code.
- *suite*: the body part of a clause, subordinated to a header.
- *value*: any object or piece of data in memory, which can be bound to one or more *names*. Values have type, but no scope.
- *variable*: a value in memory associated with a name.

Note: In Python, names have scope, but no type; values have type, but no scope. In Ratscript, names have both scope and type, while values only have scope. Ratscript names can be "rebound" to a new value, as long as that value is the same type.

1.2 Document Conventions

Note: Notes, TODOs, and proposed statements are formatted like this (blue block) to make them easier to find.

Warning: Design Principles: These are notes about design principles.

```
# This is a code example.
```

CHAPTER

TWO

LANGUAGE STRUCTURE

Ratscript draws inspiration from many languages, but it has no direct parents. It is most closely comparable with Python, although many of its core features differentiate it even there.

> **Warning: Design Principle:** Syntax is chosen for suitability and ease-of-use, *not* for familiarity or tradition.

2.1 Whitespace

The Ratscript language largely ignores whitespace, except as important for determining the boundaries of names.

We will formalize a style, which will include whitespace recommendations.

2.1.1 Line Termination

All lines of code in Ratscript are simply terminated by the newline character `\n`.

```
print("Hello, world!")
print("I am Ratscript.")

#> Hello, world!
#> I am Ratscript.
```

If the user wants to continue typing beyond one line, they can do so with the ellipsis `...` token, as in the following example.

```
print("Hello, world! I...
am Ratscript.")

#> Hello, world! I am Ratscript.
```

However, the `...` must be the last characters (except whitespace) on a line for line continuation. The following would throw an error.

```
COUNTEREXAMPLE
print("My name is Bond... James
      Bond.")
```

The line continuation should work in all situations, as long as it's at the end of a line.

```
let x = 5 + 7 ...
        + 13 + 9

x
#> 34
```

2.2 Comments

2.2.1 Line and Inline Comments

A line comment is denoted with the hash # at the start of the comment. This symbol tells the compiler that everything until the newline is a comment, allowing for both line and inline comments.

```
# This is a single line comment.
print("Hello, world!") # This is an inline comment.
```

2.2.2 Multiline Comments

```
##
This is a multiline comment.
A multiline comment is denoted with `##` at the start,
and `##` at the end.
##
```

After the initial ##, all subsequent # (and whitespace) are ignored until another character is encountered. This is a valid multiline comment then:

```
### ### ### ### ###
FANCY
BANNER
HERE
### ### ### ### ###
```

2.2.3 Documentation Commenting

Documentation comments are denoted with #!.

```
#! This is a documentation comment.
```

```
##!
You can create multiline documentation comment like this.
The closing tag is the same as a normal multiline comment.
##
```

2.2.4 Terminal Output

All command line output begins with #> to make it a valid line comment, for convenience when copy-pasting.

Error messages are also preceded with /!\ to make them easier to spot.

```
#> This is command line output.
#> /!\ This is an error message.
```

2.3 Subordination

Ratscript uses a very unique way of defining "blocks": the **subordination operator**. While this is unusual (hey, brackets were too, once!), it offers a few advantages, as you'll see.

2.3.1 Subordination Operator

The subordination operator is the semicolon ;, chosen because of its location on the home row on QWERTY keyboards.

The whitespace around the ; operator is ignored, so you can use traditional indenting as it suits your needs and preferences. The recommended style is used herein, and outlined in style.

Here's the subordination operator in use:

```
if foo
;  if bar
;  ;  if baz
;  ;  ; do_thing()
;  ;  else
;  ;  ; do_other_thing()
;  ;  end if
;  end if
end if
```

This offers the unique advantage of being able to see how deeply nested ANY line is out of context, merely by the number of ; operators before it. What's further, when space-padded in the recommended style depicted, it aids the eye in drawing a direct line between statements at the same level. Third, it's more visible than whitespace indentation.

By being able to "subordinate" any line to any other line, we can add additional information to virtually any line. (See attributes.)

```
make chinese_name as string
; <encoding "utf-16">
```

```
make fibonacci(num as integer!)
;  <recursion MAX>
;  <return as integer>
;  if num < 2
;  ; return num
;  else
;  ; return fibonacci(num-1) + fibonacci(n-2)
;  end if
```

Multi-lining Statements

The subordination operator may appear in the middle of a line, for creating one-line statements. (This is one other reason we chose the semicolon.)

For example, the following…

```
if answer == 42
;  print("What's the question?")
```

…could be rewritten as…

```
if answer == 42 ; print("What's the question?")
```

The depth of nesting still matters however, as seen here:

```
if searching
; if answer == 42 ;; print("What's the question?")
```

We have to use two `;;` to separate, otherwise the suite will become disconnected from its header.

Here are some more examples:

```
if foo == 42; do_thing(); do_other_thing()

if foo == 42
;  do_thing()
;  do_other_thing()

make name = "Jason" ; <encoding=utf-8>
```

These, however, do not work:

```
do_thing_one(); do_thing_two()  # invalid!
do_thing_one();  # invalid, don't end with a semicolon
```

This helps enforce good style, as it's already considered bad practice in nearly all languages to combine two disparate statements on the same line.

2.4 Attributes

Ratscript allows applying attributes to anything. (This can be used for many things.)

Attributes are defined in corner brackets < >, and are subordinated to the definition of whatever they modify.

```
make name = "Jason"
;  <encoding="utf-8">

make num = 65.9
;  <precision="double">

make do_thing()
;  <recursion=100>
;  print("Hi!")
;  doThing() # this would stop recursing after a depth of 100
```

These are basically just compile-time variables/properties.

2.4.1 Built-In Attributes

Some attributes are built-in. You can also define a custom attribute on a class or function. (See classes)

Class Attributes	
`<key=member>`	The key member of a class, used for casting and hashing
`<private>`	A class member that is only visible from within the class
`<protected>`	A class member that is only writable from within the class
`<static>`	A static class member

Function Attributes	
`<throw>`	Function has the potential to throw an error

Variable Attributes	
`<encoding="utf-8">`	`string` encoding
`<precision="double">`	`float` precision

2.5 Keywords

The following are keywords in Ratscript. Actual behavior will typically be described elsewhere.

- `assert`
- `and`
- `break`
- `case`
- `class`
- `continue`
- `define`
- `elif` or `else if`
- `else`
- `end`
- `fall`
- `false`
- `finally`
- `for`
- `if`
- `let`
- `make`
- `nil`

- or
- panic
- pass
- print
- return
- switch
- test
- this
- until
- warn
- while

CHAPTER

THREE

INDICES AND TABLES

- genindex
- search