



# Red Programming Language

Guaracy Monteiro  
guaracy.bm@gmail.com

Outubro 2015

## Sumário

<b>1</b>	<b>Introdução</b>	<b>3</b>
1.1	Objetivo . . . . .	3
1.2	A Linguagem . . . . .	3
1.3	Configuração . . . . .	3
<b>2</b>	<b>Sintaxe</b>	<b>4</b>
2.1	Delimitadores . . . . .	4
2.2	Sintaxe livre . . . . .	5
2.3	Comentários . . . . .	5
<b>3</b>	<b>REPL</b>	<b>6</b>
<b>4</b>	<b>Variáveis</b>	<b>9</b>
<b>5</b>	<b>Tipos de dados</b>	<b>9</b>
<b>6</b>	<b>Expressões</b>	<b>9</b>
<b>7</b>	<b>Funções</b>	<b>9</b>
<b>8</b>	<b>Escopo</b>	<b>9</b>
<b>9</b>	<b>Operadores</b>	<b>9</b>
<b>10</b>	<b>Controle de fluxo</b>	<b>9</b>
<b>11</b>	<b>Excessões</b>	<b>9</b>
<b>12</b>	<b>Pilha</b>	<b>9</b>
<b>13</b>	<b>Depuração</b>	<b>9</b>
<b>14</b>	<b>Estrutura do sistema</b>	<b>9</b>
<b>15</b>	<b>Palavras reservadas</b>	<b>9</b>
<b>16</b>	<b>VID</b>	<b>9</b>

## Listagens

1	Free-form syntax . . . . .	5
---	----------------------------	---

Rascunho

# 1 Introdução

**ATENÇÃO:** A linguagem encontra-se em desenvolvimento e não está apta para ser usada em produção. Muita coisa poderá ser alterada neste documento.

**OBS:** Para salientar a sintaxe dos programas em Red no L<sup>A</sup>T<sub>E</sub>X, estou usando o “*minted*” que usa o “*pygmentize*” para efetuar a tarefa. Para ficar correto, é necessário que seja colocado no início **Red** [ ].

## 1.1 Objetivo

O objetivo inicial não é o de ser um manual, livro ou algo do gênero sobre a linguagem. Apenas um local para que eu possa agrupar as informações e o conhecimento sobre a linguagem. Em segundo lugar, compartilhar a linguagem com quem estiver interessado. Vender e ficar rico está fora de cogitação. :D

## 1.2 A Linguagem

A linguagem Red é fortemente baseada em REBOL compartilhando, entre outros, a homoiconicidade, o grande número de tipos de dados, a mistura código+data como em Lisp. Como diferenças é possível citar a possibilidade de gerar executáveis em código nativo (não precisa ser na mesma plataforma de desenvolvimento) e a tipagem opcional para parâmetros nas funções.

## 1.3 Configuração

Para criar um ambiente de desenvolvimento não são necessários poderes especiais. A primeira coisa a fazer é baixar de [www.red-lang.org](http://www.red-lang.org) a versão para o seu sistema operacional e colocar no local que ficar mais conveniente. Note que para o Linux, a versão disponível é de 32 bits. Para rodar em uma instalação de 64 bits é necessário instalar algumas bibliotecas para suportar a versão. As formas mais comuns de executar o programa são:

- Apenas executar o programa **red** e entrará no REPL (ead-eval-print loop), isto é, um ambiente interativo onde você vai digitando e executando as instruções.

- Executando **red** <arquivo.red> o script existente no arquivo será executado.
- Executando **red -c** <arquivo.red> o script existente no arquivo será compilado e irá gerar um executável para a plataforma atual.
- Executando **red -c -t** <plataforma><arquivo.red>, o script será compilado para a plataforma especificada. Assim você pode estar no Linux e gerar executáveis, por exemplo, para Linux, Windows, Android e OSX, sem a necessidade de trocar de ambiente. As plataformas disponíveis são:
  - **MSDOS** : Windows, x86, aplicações console (+ GUI)
  - **Windows** : Windows, x86, GUI applications
  - **Linux** : GNU/Linux, x86
  - **Linux-ARM** : GNU/Linux, ARMv5, armel (soft-float)
  - **RPi** : GNU/Linux, ARMv5, armhf (hard-float)
  - **Darwin** : MacOSX Intel, apenas aplicações console
  - **Syllable** : Syllable OS, x86
  - **FreeBSD** : FreeBSD, x86
  - **Android** : Android, ARMv5
  - **Android-x86** : Android, x86

## 2 Sintaxe

Antes de começar qualquer coisa, aprender um pouco da sintaxe é importante. Até porque você deve estar acostumado com aquelas linguagens complicadas onde é necessário separar algumas coisas com vírgula, outras com ponto e vírgula, outras com chaves, outras com colchetes, etc., etc., etc.. Então vamos lá:

### 2.1 Delimitadores

Basicamente são três os delimitadores. Para string, blocos e caminho.

- **Strings** : utiliza-se aspas ( `"` ) para strings que não possuam quebra de linha no interior ou chaves ( `{ }` ) caso a string tenha mais de uma linha.
- **Blocos** : os blocos são delimitados por colchetes ( `[ ]` )
- **Caminhos** : são delimitados (ou concatenados) com a barra invertida ( `\` )

## 2.2 Sintaxe livre

O delimitador padrão é o espaço e, a única restrição é separar os tokens por um ou mais espaços. Os códigos abaixo são todos válidos e possuem a mesma avaliação:

```
1 Red []
2 while [a > 0][print "loop" a: a - 1]
3
4 while [a > 0]
5     [print "loop" a: a - 1]
6
7 while [
8     a > 0
9 ] [
10     print "loop"
11     a: a - 1
12 ]
13
14 while [
15     a > 0
16 ] [
17     print "loop"
18     a: a - 1
19 ]
```

Listagem 1: Free-form syntax

Note que, se você entrar com `a<0` ou `a-1` (sem espaços) causará um erro. Ou melhor, poderá causar um erro já que serão consideradas como palavras (variáveis) e poderão existir e conter um valor válido.

## 2.3 Comentários

Existem dois tipos de comentários (trechos que são ignorados pelo programa):

- O comentário que inicia com ponto e vírgula ( ; ) e vai até o final da linha e pode ser utilizado em qualquer parte do programa e
- o comentário com mais de uma linha que inicia com **comment** { e termina com um fecho chave ( } ) pode ser utilizado em qualquer parte do programa menos no meio de uma expressão.

### 3 REPL

Em vez de criar um script em um editor, executar e/ou compilar, acredito que o mais interessante no início seja digitar e ver o resultado. Para tal, basta usar o REPL (read-eval-print-loop). Como o nome já diz, ele lê uma entrada efetuada pelo usuário, efetua uma avaliação, mostra o resultado e fica esperando uma nova entrada. Para iniciar, basta executar **red** sem nenhum argumento e deverá aparecer algo como:

```
--== Red 0.5.4 ==--  
Type HELP for starting information.  
  
red>>
```

Para sair digite **q** ou **quit** e pressione enter. Digitando help e enter, serão apresentadas algumas opções para auxílio. Lembre-se que o

Apesar de não necessitar a digitação de **Red** [ ] que aparecem nas listagens para efeitos de salientar a sintaxe do script, se entrar no REPL não terá problema nenhum. O REPL entende que a entrada de uma nova linha seja a indicação para que ele avalie a entrada. Faz-se necessário que o comando seja digitado em uma linha a menos que ele termine com a abertura de um bloco [. Neste caso, ele mudará o prompt de **red>>** para uma abertura de colchetes [ indicando que está esperando o fechamento para avaliar a expressão.

```
red>> a: 5  
== 5
```

```
red>> while [a > 0]
*** Script error: while is missing its body argument
*** Where: while
red>> while [a > 0] [
[   print
[   "loop"
[   a: a - 1
[   ]
loop
loop
loop
loop
loop
loop
red>>
```

Utilizando as setas para cima e para baixo é possível navegar no histórico para a execução de expressões informadas anteriormente. Se você digitar algo e pressionar tab, o REPL irá mostrar uma relação das possíveis funções que podem ser entradas, inclusive as definidas pelo usuário. Se for digitado a e tab, teremos algo como:

```
red>> action! any any-type! all absolute add and~ append at
any-object! any-string! any-word! any-function! any-block!
arcsine arccosine arctangent arctangent2 as-pair any-path!
a attempt action? ask a-an acos asin atan aqua any-block?
any-function? any-object? any-path? any-string? any-word?
atan2 and about
red>> a
```

Se você digita **help** ou **?** seguido de uma função, será mostrado um resumo da função informando como ela é utilizada, uma breve descrição da função, os argumentos e alguns refinamentos. Para insert, temos:



```
red>> help insert

USAGE:
    insert series value /part length /only /dup count

DESCRIPTION:
    Inserts value(s) at series index; returns series head.
    insert is of type: action!

ARGUMENTS:
    series [series! bitset! map!]
    value  [any-type!]

REFINEMENTS:
    /part => Limit the number of values inserted.
            length [number! series!]
    /only => Insert block types as single values (overrides /part).
    /dup  => Duplicate the inserted values.
            count  [number!]
```

red>>

- 4 Variáveis
- 5 Tipos de dados
- 6 Expressões
- 7 Funções
- 8 Escopo
- 9 Operadores
- 10 Controle de fluxo
- 11 Excessões
- 12 Pilha
- 13 Depuração
- 14 Estrutura do sistema
- 15 Palavras reservadas
- 16 VID