「わこつ言語」チュートリアル

1. わこつ言語とは

・ 概要・理念

わこつ言語(WaQuotes-language、わ"言語)は鈴木雄大により考案、実装された言語及びその処理系である。WaQuotes に由来して内部的には wq、wqlang と表現される。

ruby と coffeescript から影響を受けており、セミコロンがなく、実行に必要な括弧も最小限である等、快適なタイピングをメインコンセプトとした。

関数宣言をはじめとして、if 処理や while 処理の際には内部の関数群を proc オブジェクトとして保持し、これを変数として保持できるため、coffeescript に近い感覚でコーディングすることができ、また proc オブジェクトも->と q で囲われるため、スムーズにタイプすることができる。(多くのプログラマはアロー演算子->を入力するキーバインドを設定しているか、入力に慣れていると思います。)

また、処理系の名前にもある「wq」、proc の終了を表す「q」は vim を使用するプログラマにとっては馴染みがあり、自然と指が伸びる文字であることも、わこつ言語を快適にする 1 つの理由である。

わこつ言語で不自由のないタイピングによるプログラミングをお楽しみください！

・ 実装

実装言語は ruby で、LL(1)言語として実装されている。

実行環境として 1.9.3 以降の ruby 処理系が必要である。

プログラムファイルの実行には ruby wqlang.rb [ファイル名]と実行する。

処理系の詳細な処理については『「わこつ言語」処理系に関するソース解説』を参照。

・ 機能

通常のプログラム実行の他に、シェル対話モードを備えている。

引数としてファイル指定をせずに実行するとシェル対話モードとして起動し、ユーザの入力をプログラムコードとして解釈し実行する。

シェル対話モードでは以下で紹介される制御構造、組み込み関数の他に、デバッグを行う::debug 関数を備えており、namespace 及び code と引数を指定することでそれぞれ確認が可能。

終了コマンドとしては、quit の他に.q、:q、:q!及び Ctrl-C で終了可能である。

2. 基本文法・制御構造

一部の制御構造を除き、改行文字を処理の終了とする。そのため、各処理は基本的に 1 行で示される。

要素と要素の間はスペースで区切って記述しなければならない。ただしカンマ、コロン、括弧の前後の空白はなくとも良い。

なお、変数名、関数名には半角英数文字、ハイフン(-)、アンダースコア(_)のみ使用できる。

・ 変数スコープ

通常の変数はグローバル空間において宣言されるが、関数呼び出し内部での名前空間はグローバル空間とは隔絶される。そのため内部で宣言した変数はスコープを抜ける際に消滅し、内部ではグローバル変数の値を変更することはできない。

・ コメントアウト

単一行コメント(行頭からのみ使用可能)

```
#= comment out
```

複数行コメント

```
###
comment out
###
```

・ 文字列: "*" または '*'

クォートで囲われたものを文字列として扱う。

文字列内ではその文字列を囲っているクォート文字を使用することはできない。そのため内部でクォート文字を用いる際は、囲っているものと別のクォート文字を使用する。

ダブルクォートに囲われている場合、改行文字、タブ文字、空白文字(\n, \r\n, \t, \s)を使用できる。

・ 式

数値、文字列同士の計算、比較を表す。型エラーの発生する計算は実行時エラーとなる。

・ 予約語

```
while, if, else, func, print, println, scan, call, q, true, false
```

・ 代入: =

処理の終了は改行。演算子の前に代入される変数、後に代入する値を指定する。

代入する値には変数、文字列、式、call 文が対応。call 文の場合実行結果が代入される。

- 条件: if

  条件句の終了は'->'。内部処理の終了は'q'。内部処理は 1 つの proc オブジェクトとして格納される。

  if 句の後に条件を表す式を指定する。
- 繰り返し: while

  条件句の終了は'->'。内部処理の終了は'q'。内部処理は 1 つの proc オブジェクトとして格納される。

  while 句の後に条件を表す式を指定する。
- 関数宣言: func

  宣言句の終了は'->'。内部処理の終了は'q'。内部処理は 1 つの proc オブジェクトとして格納される。

  func 句の後に関数名を指定する。引数を与える場合は関数名に続けて':'で引数句を開始し、引数は','で区切る。

3. 組み込み関数

- 出力: print, println

  処理の終了は改行。それぞれ改行なし出力、改行あり出力を行う。

  引数は変数、文字列、式、引数なしの call 関数が対応。

  引数ありの call 関数を渡すことはできない。

- 入力: scan

  処理の終了は改行。ユーザからの入力を受け付ける。

  変数を引数として呼び出した場合、その引数に入力値を代入する。

  変数代入の対象として scan が呼ぶことも可能。

  この場合も引数として変数を 1 つ持つことが可能。(非推奨)

- 関数呼び出し: call

  処理の終了は改行。

  引数を与える場合は関数名に続けて:で引数句を開始し、引数は','で区切る。

  関数呼び出しの結果を変数に代入可能。代入せずに出力することはできない。

  引数なしの関数を呼ぶ場合は call をつけずに呼ぶことが可能。(非推奨)

- ループ終了: break

  処理の終了は改行。現在のループ処理を抜け元の処理に戻る。

  while 内で使用可能。それ以外の場所では動作しない(シンタックスエラーとはならない)。

- プログラム終了: exit

  処理の終了は改行。プログラムを終了する。

4. 推奨される実装、されない実装

・ カンマ、コロンについて

カンマ、コロンの前の要素との間には空白を空けず、後ろの要素との間には空白を空ける実装を推奨

```
func sample: x, y ->
```

・ 括弧について

括弧内部の要素に対しては空白を空けず、外部とは空白を空ける実装を推奨

```
(1 + 2) + (3 + 4)
```

・ call 文なしでの関数呼び出しについて

変数呼び出しと同様に扱えるが、引数ありの場合と文法が大きく異なり可読性が下がるため、非推奨。

5. サンプルコード

以下は基本文法を示した tutorial.wq である。

上記以外にも非推奨な実装があるが、基本的にはサンプルコードの文法での記述を推奨する。

```
#= IO
print "Hello World\n"
println "Hi"

scan x
println x
y = scan
println y

#= Assignment
number = 42
bool = true
calc = 1 * 10

#= Function
func hello_world ->
   println 'Hello World'
q
call hello_world

func square：x ->
   result = x * x
q
x = 'out of scope'
result = call square：3
println result
#=> 9
println x
#=> 'out of scope'
```

```
#= Condition
println 1 >> 0
println (1 >> 0) && (1 << 0)

if 1 >> 0 ->
   println 'true'
q else ->
   println 'false'
q

while number <= 50 ->
   println number
   if number == 47 ->
      break
   q
   number = number + 1
q

println 'end'
```