Cygni 1.0
A Short Reference

He Yanjie

December 26, 2016

Abstract

1 What is Cygni?

Cygni is a script programming language implemented in C#. It is easy to use, has neat grammar and can interacts with C#. It is convenient to wrap C# classes as Cygni class, namely the Cygni libraries are based on the C# class libraries.

Cygni is designed by me. I spent a lot of spare time on it, and I love it very much. I hope you will like it too!

2 Core Language

2.1 Reserved words

- and
- or
- not
- true
- false
- nil
- if
- else
- elif
- while

- for
- foreach
- in
- def
- lambda
- class
- local
- unpack
- break
- continue
- return

2.2 Reserved Symbols

- Add:+
- Subtract:-
- Multiply:*
- Divide:/
- Integer Divide: //
- Modulo:%
- Power: ˆ
- Concatenate: ..
- assign: =
- Equals:==
- Not Equals: !=
- Greater than: >
- Less than: <
- Greater than or Equals: >=
- Less than or Equals: <=
- Goes to: =>

- Parentheses: ( )
- Brackets: [ ]
- Braces: { }
- Colon: :
- Comma: ,

2.3 Identifiers

The first character of identifiers should be underline or letters, the rest can be underlines, letters or numbers. Note that the identifiers should not be the same as the reserved words.

2.4 Comments

Line comment start with #.

2.5 Strings

String should be enclosed by ” or ””. If there is symbol at the start of string, then the escaped characters in the string will be ignored, and the ’ or ” in the string shoule be written twice.

2.6 Types

Variables in Cygni don’t have type. Only the values have.

- integer
- number
- boolean: true or false
- string
- list: lists contain elements from various types.
- dictionary: key-value pairs
- function
- native function: wrapper for C# native functions
- tuple
- struct
- class
- userdata: wrapper for C# native classes.
- nil

2.7 Control Statements

```
if condition {
        # Do something
} else {
        # Do something
}

if condition1 {
        # Branch 1
} elif condition2 {
        # Branch 2
} else {
        # Branch 3
}
```

```
for i = start, end {
        # Do something
}
```

```
foreach item in collection {
        # Do something
}
```

```
while condition {
        # Do something
}
```

Break, Continue, Return break, return exit the loop. continue stays in the loop.

2.8 List Constructor

```
[item1, item2, ...]
```

2.9 Dictionary Constructor

```
{key1: value1, key2:  value2, ...}
```

Note that dictionary only takes values of integer, boolean, string as keys.

2.10 Function Definition

```
def FunctionName (arg1, arg2, ...) {
        # Do something
}

a = lambda(arg1, arg2, ...) => # Expression

a = lambda(arg1, arg2,...) => {
        # Do something
}
```

2.11 Function Call

```
f(arg1, arg2, ...)
```

2.12 Tuple Constructor

```
a = tuple(10, 20)
unpack x, y = a
```

2.13 Struct Constructor

```
a = struct('key1', value1, 'key2', value2, ...)
```

2.14 Class Definition

```
class MyClass {
        # body
}

class DerivedClass: MyClass {
        # body
}
```

2.14.1 Reserved Fields

- __init
- __add
- __sub
- __mul
- __div
- __mod

- __pow
- __unp
- __unm
- __cmp
- __eq
- __get
- __set
- __toStr

3 Basic Library

3.1 Executing

- source
  arguments: fileName [,encoding]
  description: execute a script file.
  return: nil
- require
  arguments: fileName [,encoding]
  description: execute a script file and return it as a module.
  return: module
- import
  arguments: fileName [,encoding]
  description: execute a script file in the current global scope.
  return: nil

3.2 Console Output and input

- print
  arguments: args
  description: print arguments in the console, separated by tab.
  return: nil

- printf

  arguments: content, args

  description: print format string in the console. The arguments can be indexed by {0},{1}... in the string.

  return: nil

- input

  arguments: [content]

  description: write the content in the console and waiting for user to input. The content can be omitted.

  return: string

3.3 Conversion

- int
- number
- str
- list