Cygni Primer

He Yanjie

November 2016

Contents

Chapter 1

Introduction

1.1 What is Cygni

Cygni is a scripting language, implemented in C#. It supports both procedure-oriented and object-oriented programming.

Cygni is inspired by some scripting language, such as Python, Lua, etc.

If you have any questions, please feel free to discuss with me. My github: https://github.com/JasonHe0727

Chapter 2

Core Cygni

Ok, if you have some programming experience before, I think maybe you don't have to go through all of the contents. Some small examples may be more helpful. Let's begin.

2.1 Your First Cygni Program

Here is a simple script which writes message to the screen.

2.1.1 The Code

```
print("Hello Cygni!")
```

2.1.2 Running the Program

Input the previous code in the interpreter, then you will see the input "Hello Cygni!" in the screen.

2.2 Variables

The variables in Cygni are dynamic, which means they can be any type, such as number, boolean, string, etc.

You declare variables in Cygni by simply assigning values to them:

```
a = 12
b = true
```

You may assign one value to several variables:

```
a = b = 89.32
```

Name	CTS Type	Description	Range(Approximate
number	System.Double	64-bit, double precision floating point	$\pm 4.9 \times 10^{-324} \sim 1.7 \times$
boolean	System.Boolean	Represents true or false	true or false
string	System.String	Unicode character string	

Table 2.1: Predefined Types

2.3 Variable Scope

The scope of a variable is the region of code from which variable can be accessed. In Cygni, the scope is determined by the following rules:

- A class, which contains fields, has its own scope.
- A function, which contains local variables, has its own scope.
- If a variable is not defined in a class or a scope, then it is a global variable, which stays in the global scope.
- Built-in variables are in the built-in scope, which is immutable in the runtime.

The rule of finding variables is: local -¿ field -¿ global -¿ built-in

2.4 Types

See table 2.4.

2.4.1 The number type

The number type is actually the double type in C#.

2.4.2 The boolean type

The boolean type can only be true or false.

2.4.3 The string type

Literals of the string type should be enclosed by ” or ””. Backslashes(’
’) represents escape sequence. If you write symbol ’@’ at the beginning of the sentence, the ’
’ will lose its meanning.

```
s1 = "a string"
s2 = 'another string'
s3 = @'c:\test.txt' # If literals start with '@', the backslashes will lo
s4 = 'c:\\test.txt' # The 's4' equals to 's3;
```

2.5 Flow Control

2.5.1 Conditional Statement: The if Statement

The syntax of 'if' statement is as followings:

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

The final 'else' statement is not a must. If you have several branches, it is convenient to use 'elif' statement for represent them.

```
if condition {
        # Branch One
} elif {
        # Branch Two
} elif {
        # Branch Three
}
```

2.5.2 Loops

The for loop

The foreach loop

The while loop

2.6 Factorial

This is a very traditional example.

```
def fact(n){
        if n == 0 { 1 }
        else { n * fact(n - 1) }
```

```
}
fact(10)
=> 3628800
```

2.7 Position

Have a look at another example.

```
class Position{
        def __INIT__(nx, ny){
                this.x = nx
                this.y = ny
        }

        def move(nx, ny){
                this.x = nx
                this.y = ny
        }

        def __TOSTRING__(){
                printf("({0}, {1})", this.x, this.y)
        }

}

p1 = Position(10,20)
print(p1)
p1.move(35,47)
print(p1)
```

Hope you have got a feel about Cygni!

Chapter 3

Built-in Types

3.1 Basic Types

3.1.1 Number

Number can be integer or float.

```
a = 10
b = 99.2
```

3.1.2 Boolean

Boolean can be two values: true and false.

```
a = true
b = false
```

3.1.3 String

You can use ' or "" to enclose a string.

```
str1 = "Hello Cygni!"
str1
=> "Hello Cygni!"
```

3.2 Collection

3.2.1 List

The syntax of initializing a list is the same as Python. You can put objects from various types into a list.

A list can be indexed by a non-negative integer. The index starts from zero.

```
list1 = [1, 2, 3, 4, 5]
list2 = [18, false, "Judy"]
list1[0] = 789
```

3.2.2 HashTable

Using function 'hashtable' to initialize a hash table by key-value pairs. The key can only be integer, boolean and string. A hash table can be indexed by the key.

```
ht1 = hashtable("key1",123,"key2",789)
ht1["key1"]
```

3.3 Structure Array

The stucture array type is inspired by the Matlab/Octave. You can get element from a struct by the field or by the integer index.

```
s1 = struct("name","Judy","age",16)
s1.name # The same as s1[0]
s1.age
```

3.4 Function

3.4.1 Cygni Function

Cygni function should be initialized by the 'def' statement. The function can be passed as a parameter.

```
def mul(x){
        return x * y
}
```

```
def mul2(f, x){
        return f(x, 2)
}
```

3.4.2 Native Function

Native function is imported from C#.

3.5 Class

Cygni class can be initialized by the 'class' statement. It supports inheritance. There are some built-in functions to be overrided as followings.

- __INIT__: Constructor for the class. The default constructor is a non-arg constructor.
- __TOSTRING__: Output the class instance as a string.
- __ADD__: override '+' operator.
- __SUBTRACT__: override '-' operator.
- __MULTIPLY__: override '*' operator.
- __DIVIDE__: override '/' operator.
- __MODULO__: override '%' operator.
- __POWER__: override 'ˆ operator.
- __COMPARETO__: return a integer to indicate the comparision result. This function will override the '¿', '¡', '¿=', '¡=' operators.
- __INDEXER__: This function takes a list as indexes, and return an element.

3.6 User Data

User data is a wrapper for the C# data type.

Chapter 4

Operators

4.1 Arithmetic Operators

Cygni supports the following arithmetic operators:

- Add: +
- Subtract: -
- Multiply: *
- Divide: /
- Modulo: %
- Power: ^
- Unary Plus: +
- Unary Minus: -

```
a = 10
b = 99.2
a + b
=> 109.2
```

4.2 Logical Operators

Cygni supports the following logical operators:

- and

- or
- not

```
a and b
=> False
```

4.3 Relation Operators

The following relation operators will return a boolean value:

- '¿'
- '¡'
- '¿='
- '¡='
- '=='
- '!='

Chapter 5

Control Flow Statements

5.1 Condition Statement

Use if to start a condition statement. There are three keywords: if, else, elif.

```
if x > 10 {
        print('x is greater than 10')
} else {
        print('x is not greater than 10')
}

if y == 5 {
        print(5)
}
elif y == 10 {
        print(10)
}
elif y == 20 {
        print(20)
}
else {
        print('y is not 5, 10 and 20.')
}
```

5.2 Loop Statement

There are three loop statement in Cygni: for, while, foreach. The followings are three examples using different statements to print 1 to 9 in the console.

```
for i = 0, 10 {
        print(i)
}


i = 0
while i < 10 {
        print(i)
        i = i + 1
}

foreach i in range(0,10) {
        print(i)
}
```

5.2.1 for

The 'for' statement should take two or three arguments, and it needs a named value as the iterator.

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

for i in start, end, step {
        # Do something
}
```

If the step is positive, the iterator will increase the step at one time, and break out when the iterator is greater than or equal to the end. If the step is negative, the loop will break when the iterator is less than or equal to the end.

5.2.2 while

The 'while' loop will not break until the condition is false;

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

5.2.3 foreach

'foreach' statement tranverse every element in the collection.

```
foreach i in [1,2,3,4,5] {
        # Do something
}
```

5.3 Jump statement

There are three jump statement in Cygni: break, continue, and return. 'break' can jump out from the current loop, 'continue' can start a new round in the loop. 'return' is used to return value from a function.

5.4 def

'def' statement is used to define a function.

5.5 class

'class' statement is used to define a class.

5.6 Recursion

Chapter 6

Built-in Commands

6.1 Do File

TO DO