Ruby学习日志

基础语法:

**注释**：

“#注释内容”或者

“=begin

这里是有注释很多很多

=end”

**输出**：

Print，puts，p都可以

变量：

本地变量： time 或者是 \_time

实例对象： @time

类： @@time

全局变量： $time

数据类型：

Numeric，String，Symbol，Boolean，Array，Hash。

变量技巧：

“hello #{name}” a,b = b,a; 3.times{puts “hello”} “hello” \* 3

这些分别是什么意思？

if：

这里标注一下特殊的 if a > 5 then puts a end

特殊的第二版（超简洁） puts a if a > 5

Unless

他不再是else，这个关键字的作用等同于if（！XXX） === unless（XXX）

If elsif else：

这里不啰嗦，注意一下关键字 elsif 即可。

Switch：

改变后的switch完全不一样。

Case xxxx

When “keyValue” then “result”

When ….

Else “result” #这里等同于defualt

End

注意：首字母不用大写，这里是word排版默认大写，我未作更改。

Loop：

**循环：**

3.times{ puts “hello world” }

搞定：等同于 ×3

**For：**

For x in 数组

Puts

End

**While：**

While i > 5 do

I -= 1

End

等同于

I -= 1 while i > 5

**Untile:**

语法通while。

使用loop do 代替 while ture

Loop do puts “OK” end

中断循环关键字

Break，next，redo，retry。

疑问：redo，retry么意思？

方法：

def关键字

def plus（x，y）

z = x + y

return z

end

调用

Plus（3,4）

**简洁版**

**Def plus x，y**

**X+y**

**End**

**调用**

**Plus 3,4**

只有overriding 没有loading，这个是什么意思？

带有block的方法：

Def hello

Yield

End

Hello {puts “hello,block”}

另外一种形式，但是不完全一致

Def hello &block

Block.call

End

Hello {puts “hello,block”}

Lambda 和 Proc

Lambda{}

Proc.new{}

区别：

Proc中的return会跳出当前作用域

Lambda中的return不会跳出当前作用域

Class：

正常定义，使用new 实例化调用

属性访问级别：attr\_accessor:name 读写

Attr\_reader:name

Attr\_writer:name

继承

关键字 “<”

Class A < B

Def xxx

End

End

A继承自B

Module 更强大的interface

Module Eat

Def eat

P “I can eat”

End

End

Class Pig

Include Eat

End

Pig.new.eat

常量访问：

Module Math

PI = 3.24

End

调用 Math::PI

Module命名空间：

在module外面再嵌套一层就认为是命名空间了

比如

Module Foo

Module Bar

Def self.say

P “sss0”

End

End

End

调用：Foo::Bar.say

备注：当然module中也可以嵌套class

module的特殊用法singleton:

module Item extend self

def name

p “I’m item”

end

end

调用： Item.name

扩展方法

class Fixnum

def plus n

self + n

end

end

1.plus 3

更多:

[Fiber](https://gist.github.com/1050566)

[Class](https://gist.github.com/872171)

[System](https://gist.github.com/822091)

[Proc](<https://gist.github.com/874438>)

ruby中，一切都是对象，比如 “Lampard”.reverse。

特点：在ruby中可以省略（），{}等。

irb：irb命令是启动ruby

p puts print 的共性与区别

共性：都是向屏幕打印东西

区别：

print:打印一行内容，不追加换行符，其他两个命令会追加换行符

p：不解释“”中间的字符，原封不动的输出

puts：对字符串中的\n等转义字符做了解释

网上有资料说p无法打印中文，在2.0.0版本无此问题

格式控制符：

％　　格式说明引导符。

－　　指定左对齐输出。

０　　指定空位填零。 eg. %0md，m目标是位数

m.n 指定输出域宽度及精度。 eg. %m.nd，m目标是位数，n表示小数位数。

格式字符：指定输出项的数据类型和输出格式。

d　　有符号十进制整数。

o　　无符号八进制数。

x　　无符号十六进制数。(小写的x格式中用小写字母a,b,c,d,e,f来表示10到15之间的数，大写的X则用大写的ABCDEF来表示10到15之间的数)

u　　不带符号的十进制整数。

c 　 输出一个字符。

s　　输出一个字符串。

e　　以指数形式输出实型数。

f 以小数形式输出实型数。

g　　 自动决定输出格式为e和f中较短的一种，不打印无效的零。

%　　输出%。

注：“+” 拼接时要注意两边的变量时同类型或者是可以隐式转换的。

输出段落可以使用 <<PARAGRAPH 段落文字 PARAGRAPH

在ruby中 get与get.chomp() 的区别

gets和gets.chomp()都表示读入用户的输入并用于输出，但两者还是有所不同，其中gets是得到的内容后，在输出时后面接着换行；而gets.chmop()得到的内容输出时后面不带空格和换行。

在ruby中 include，require，expend，load四个好兄弟。

*require，load用于包含文件，include则用于包含的模块。*

*require加载一次，load可加载多次。*

*require加载Ruby代码文件时可以不加后缀名，load加载代码文件时必须加后缀名。*

*require一般情况下用于加载库文件，而load用于加载配置文件。*

*————引子http://bbs.chinaunix.net/thread-3625799-1-1.html*

没有搞明白他们的用途和用法，待练习

文件操作

Close – 保存并且关闭文件

read – 读取文档全部内容

readline – 读取一行

truncate – 清空

write(stuff) – 写入内容

在ruby中存在这些关键字，比较有趣

and ，那么请问and和&&有什么区别？

通过字面理解一下（肯定不一样，要不然我也不会问了），and是“和”的意思，&&是“并且”，好像也没差啥啊。那么请回答我 puts true and false 的输出结果（true），接着告诉我 puts false and true 的输出结果（false）。××××，我要讲脏话了，这是什么情况。

想知道答案请猛击下面连接

<http://zhaoningbo.iteye.com/blog/1982761>

看了这个地址虽然有些懵懵懂懂，但是还是不能完全明白，

后来通过http://stackoverflow.com/questions/2802494/i-true-and-false-in-ruby-is-true

完全明白了。感谢教练！

Array

::[]

::new

::try\_convert

#&

#\*

#+

#-

#<<

#<=>

#==

#[]

#[]=

#assoc

#at

#bsearch

#clear

#collect

#collect!

#combination

#compact

#compact!

#concat

#count

#cycle

#delete

#delete\_at

#delete\_if

#drop

#drop\_while

#each

#each\_index

#empty?

#eql?

#fetch

#fill

#find\_index

#first

#flatten

#flatten!

#frozen?

#hash

#include?

#index

#initialize\_copy

#insert

#inspect

#join

#keep\_if

#last

#length

#map

#map!

#pack

#permutation

#pop

#product

#push

#rassoc

#reject

#reject!

#repeated\_combination

#repeated\_permutation

#replace

#reverse

#reverse!

#reverse\_each

#rindex

#rotate

#rotate!

#sample

#select

#select!

#shift

#shuffle

#shuffle!

#size

#slice

#slice!

#sort

#sort!

#sort\_by!

#take

#take\_while

#to\_a

#to\_ary

#to\_h

#to\_s

#transpose

#uniq

#uniq!

#unshift

#values\_at

#zip

#|