

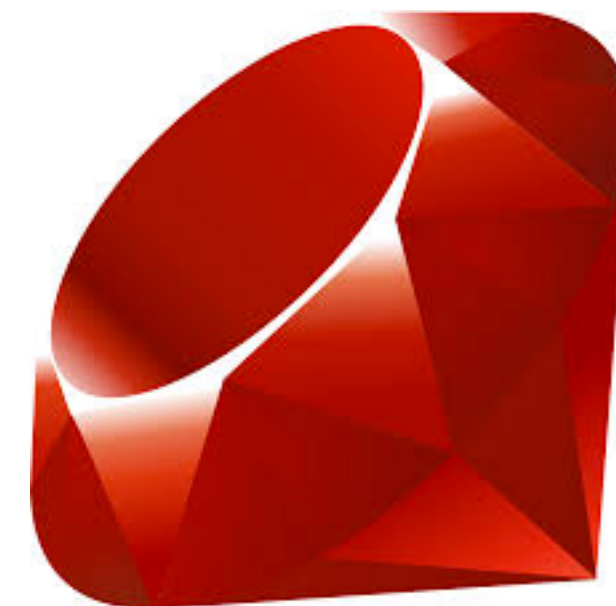
멋쟁이사자처럼 Ruby

1. Ruby 기초

#기초 #OOP #StyleGuide #IRB

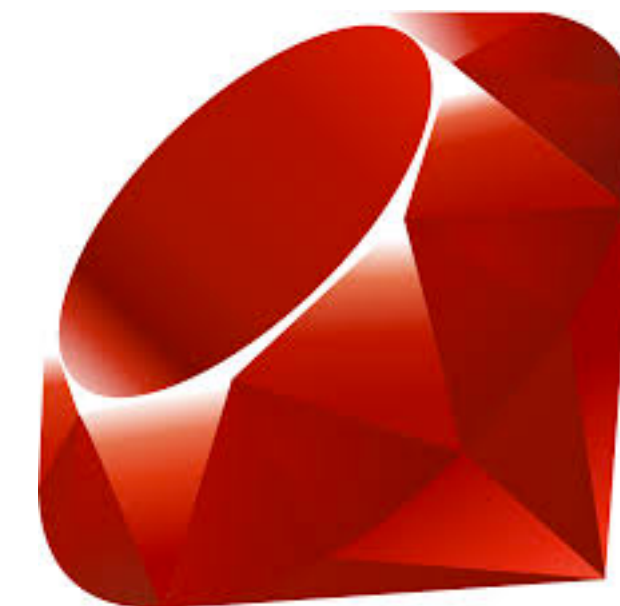
Ruby

- **마츠모토 유키히로가 개발한
동적 객체 지향 스크립트 프로그래밍 언어**
- **순수 객체 지향 언어이므로, 정수나 문자열 등을
포함한 데이터 형식 등 모든 것이 객체이다.**
- **1996년 Ver1.0 배포 및 현재 2.5.1**
- **2005년 Ruby on Rails 프레임워크 등장으로
유명해짐**



Ruby

- 마츠모토 유키히로가 개발한
동적 객체 지향 스크립트 프로그래밍 언어
- **순수 객체 지향 언어**이므로, 정수나 문자열 등을
포함한 데이터 형식 등 **모든 것이 객체**이다.
- 1996년 Ver1.0 배포 및 현재 2.4.2
- 2005년 **Ruby on Rails 프레임워크** 등장으로
유명해짐



Ruby 주요 특징

- **High-level**
: 사람말(영어) 같다.
- **Interpreted**
: 컴파일러가 필요없다.
- **Object-oriented**
: 객체지향적이다. 거의 모든 것이 '객체'
- **Easy to use**
: 사용하기 쉽다.

Ruby 언어 비교 (Java vs Ruby)

```
public class Print3Times {  
    public static void main(String[] args) {  
        for(int i = 0; i < 3; i++) {  
            System.out.println("Hello World!");  
        }  
    }  
}
```

```
3.times { puts "Hello Wolrd"}
```

23 Feb How to Apply to a Job

Here's the cover letter I sent for applying to my previous job.

```
#!/usr/bin/env ruby
I = []; fit = [:your, :offer, ' http://www.standoutjobs.com/jobs.php?id=1409']
just = checkout = %w(macournoyer cv)
its = you = me = 1_000_000_000 #$.
class Array
  def can i, help = :you; true end
  def me love = ruby
    $my = self; love='creating world'.class.new 'stuff'
  end
  def see _for = :yourself
    `open http://#{ $my.join '.com/' }/`
  end
  alias :am :to_a
  alias :in? :include?
end

I.fill(0).each { |criterion| } && I.am.uniq! and :passionate

its.upto(you) do |i| fit.in? :StandoutJobs
  just.send :me, :an => 'email'
end; %w(e ll).see if I.can :help => you

=begin
[... snip ...]

Marc-André Cournoyer
http://macournoyer.com

Run this ruby script to get my résumé
=end
```

Ruby 기초

- **2칸 띄어쓰기(space indentation) 추천**
: 파이썬과 달리 강제 사항은 아님.
- **주석처리는 '#'**
- **더이상의 세미콜론(;)은 없다.**
: 한 줄 쓰기를 위해 사용할 수는 있지만, 추천하지 않음.

<https://github.com/github/rubocop-github/blob/master/STYLEGUIDE.md>

<https://github.com/airbnb/ruby>

<https://github.com/bbatsov/ruby-style-guide>

Ruby 기초

콘솔에 print하기

- **puts**
: 문자를 콘솔에 출력하는 가장 기본 method
- 개행문자(New line `'\n'`)이 입력됨.
- Java의 `System.out.println()`과 유사
- 거의 대부분 puts를 활용

Ruby 기초

- puts vs print

콘솔에 print하기

```
3.times {print 3}
```

```
3.times {puts 3}
```

Ruby 기초

• puts vs print

콘솔에 print하기

```
3.times {print 3}
```

```
3.times {puts 3}
```

print는 개행문자를 자동으로 삽입시켜주지 않아 한 줄로 보인다.

Ruby 기초

콘솔에 print하기

- puts vs p

```
array = [1,2,3]  
puts array  
p array
```

```
var = 'abc'  
puts var  
p var
```

Ruby 기초

콘솔에 print하기

• puts vs p

```
array = [1,2,3]  
puts array  
p array
```

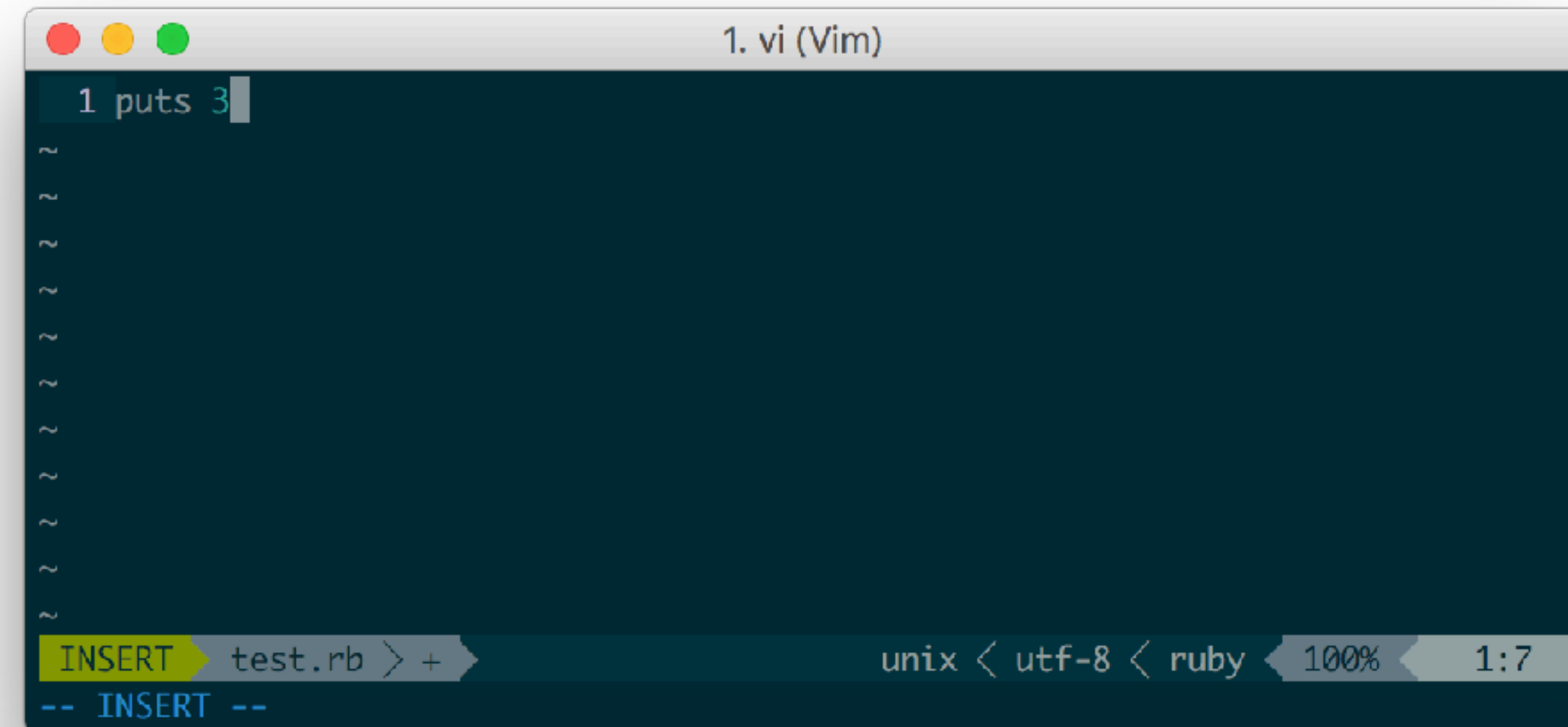
```
var = 'abc'  
puts var  
p var
```

p는 object.inspect

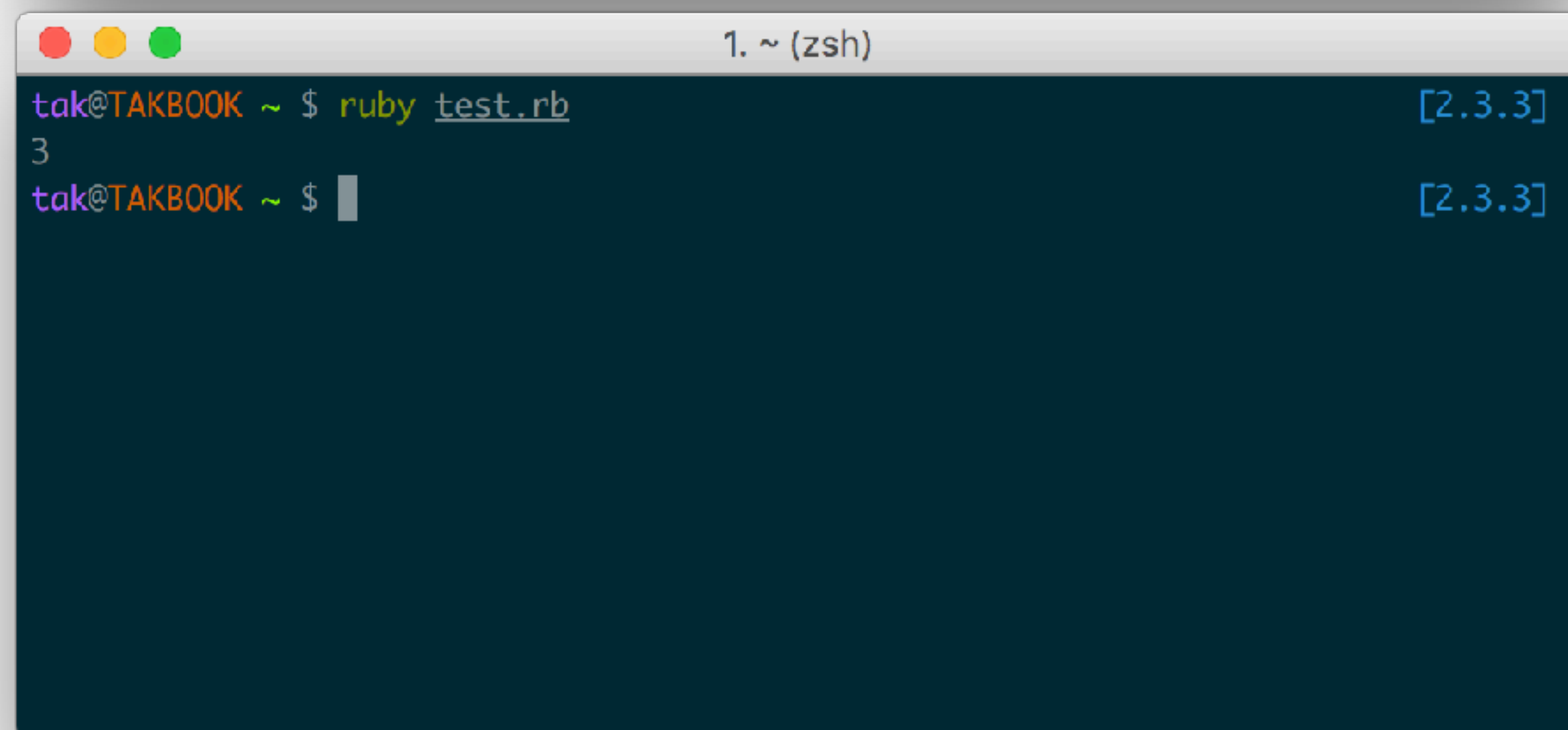
즉, object의 내부 표현방식을 보여준다.

Ruby 기초

Script 실행하기



```
1 puts 3
```



```
tak@TAKBOOK ~ $ ruby test.rb
3
tak@TAKBOOK ~ $
```

Ruby 기초

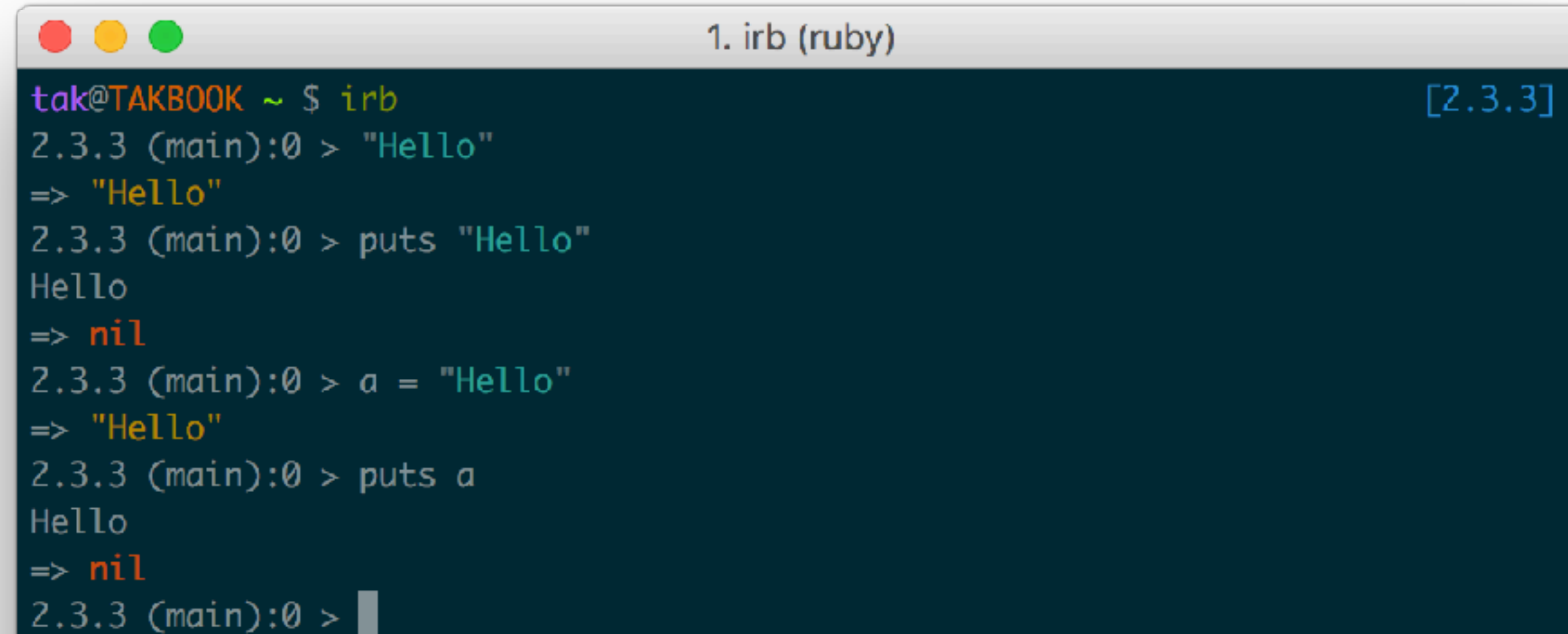
Naming Conventions

- 변수(varibales)
: 소문자, 여러 단어의 경우 snake_case
- 상수(Constants)
: 모두 대문자
- 클래스(Classses or Modules)
: CamelCase

Ruby 기초

IRB

- **IRB (Interactive Ruby)**
: Python IDLE과 유사함.
- **콘솔 기반으로 Ruby 사용이 가능함.**
- **Ruby 설치시 자동으로 사용 가능.**



```
1. irb (ruby)
tak@TAKBOOK ~ $ irb
2.3.3 (main):0 > "Hello"
=> "Hello"
2.3.3 (main):0 > puts "Hello"
Hello
=> nil
2.3.3 (main):0 > a = "Hello"
=> "Hello"
2.3.3 (main):0 > puts a
Hello
=> nil
2.3.3 (main):0 > 
```


2. Ruby

Flow of Control

#조건 #반복

2. Ruby FoC

if문

- if / unless / elsif / else
- {중괄호} 없음
- if 문은 end로 종료함.

```
a = 5 # declare a variable

if a == 3
  puts "a is 3"
elsif a == 5
  puts "a is 5"
else
  puts "a is not 3 or 5"
end

# => a is 5
```

```
a = 5

unless a == 6
  puts "a is not 6"
end

# => a is not 6
```

2. Ruby FoC

while /until

```
a = 10

while a > 9
  puts a
  a -= 1
  # same as a = a - 1
end

# => 10
```

```
a = 9

until a >= 10
  puts a
  a += 1
end

# => 9
```

2. Ruby FoC

inline statement

```
# if modifier form  
  
a = 5  
b = 0  
  
puts "One liner" if a == 5 and b == 0  
# => One liner
```

```
# while modifier form  
  
times_2 = 2  
times_2 *= 2 while times_2 < 100  
puts times_2 # => ?
```

2. Ruby FoC

true / false

- false / nil 을 제외한 모두는 true!

```
puts "0 is true" if 0 # => 0 is true
puts "false is true?" if "false" # => false is true?
puts "no way - false is false" if false # => NOTHING PRINTED
puts "empty string is true" if "" # => empty string is true
puts "nil is true?" if "nil" # => nil is true?
puts "no way - nil is false" if nil # => NOTHING PRINTED
```

2. Ruby FoC

===

- == && 의미상으로 맞을 때 true.
- Case문에서 활용하여 case equality operator

```
if /lion/ === 'likelion'  
  puts '/lion === likelion'  
end  
  
if /lion/ == 'likelion'  
  puts '/lion/ == likelion'  
end  
  
if 'likelion' === 'likelion'  
  puts 'likelion === likelion'  
end  
  
if 'likelion' == 'likelion'  
  puts 'likelion == likelion'  
end  
  
if Integer === 3  
  puts 'Integer === 3'  
end  
  
if Integer == 3  
  puts 'Integer == 3'  
end
```

2. Ruby FoC

===

```
if /lion/ === 'likelion'  
  puts '/lion/ === likelion'  
end
```

true

```
if /lion/ == 'likelion'  
  puts '/lion/ == likelion'  
end
```

false

```
if 'likelion' === 'likelion'  
  puts 'likelion === likelion'  
end
```

true

```
if 'likelion' == 'likelion'  
  puts 'likelion == likelion'  
end
```

true

```
if Integer === 3  
  puts 'Integer === 3'  
end
```

true

```
if Integer == 3  
  puts 'Integer == 3'  
end
```

false

2. Ruby FoC Case문

- **=== 을 통해 비교하며, 조건에 만족하면 종료한다.**
- **1) if문과 유사하게 활용**

```
age = 21

case # 1ST FLAVOR
  when age >= 21
    puts "You can buy a drink"
  when 1 == 0
    puts "Written by a drunk programmer"
  else
    puts "Default condition"
  end
end
# => You can buy a drink
```


2. Ruby FoC

• 2) 특정 값 target

Case문

```
name = 'Fisher'
case name # 2nd FLAVOR
  when /fish/i then puts "Something is fishy here"
  when 'Smith' then puts "Your name is Smith"
end

#=> Something is fishy here
```

2. Ruby FoC

for loop

- 루비에서는 **for**은 거의 안 쓰임.
- **Each /times** 를 더 많이 사용함.

2. Ruby FoC

for loop

- 루비에서는 **for**은 거의 안 쓰임.
- **Each /times** 를 더 많이 사용함.

3. Ruby Functions / Methods

#함수

3. Ruby Methods

- 대부분의 언어에서는 다음과 같이 정의한다.
function : 클래스 밖에서 정의 됨(def)
method : 클래스 내에서 정의 됨(def)
- 루비에서는 모든 function은 메소드이다.
(루비에서는 모든 function/method는
적어도 하나의 클래스에 속해있다.)

3. Ruby Methods

- [소괄호]는 정의할 때나 호출할 때 선택적으로 사용
- 명확성을 위해 사용하기도 함.

```
def simple
  puts "no parens"
end

def simple1()
  puts "yes parens"
end

simple() # => no parens
simple # => no parens
simple1 # => yes parens
```

3. Ruby Methods

Return

- **return 키워드는 선택적으로 사용**
(마지막에 실행된 줄이 return 됨)
- **원하는 무엇이든 return 가능하며,**

```
def add(one, two)
  one + two
end

def divide(one, two)
  return "I don't think so" if two == 0
  one / two
end

puts add(2, 2) # => 4
puts divide(2, 0) # => I don't think so
puts divide(12, 4) # => 3
```

3. Ruby Methods

메소드명

- 메소드명은 '?' 나 '!'와 함께 사용 가능하다.
- ? : 예측하기 위한 메소드
- ! : 추후에 string 학습시 사용.

```
def can_divide_by?(number)
  return false if number.zero?
  true
end

puts can_divide_by? 3 # => true
puts can_divide_by? 0 # => false
```


3. Ruby Methods

기본 매개변수 (default argument)

*** 삼항연산자**

condition ? True : false

- 메소드는 기본 매개변수(default arguments)를
가질 수 있다.

```
def factorial (n)
  n == 0? 1 : n * factorial(n - 1)
end

def factorial_with_default (n = 5)
  n == 0? 1 : n * factorial_with_default(n - 1)
end

puts factorial 5 # => 120
puts factorial_with_default # => 120
puts factorial_with_default(3) # => 6
```

3. Ruby Methods

스플랫 (splat)

- * prefix를 가진 매개변수를 가지도록 할 수 있다.
(가운데 매개변수도 가능함)

```
def max(one_param, *numbers, another)
  # Variable length parameters passed in
  # become an array
  numbers.max
end

puts max("something", 7, 32, -4, "more") # => 32
```

3. Ruby Methods

<https://ruby-doc.org/core-2.3.3/>

4. Ruby Blocks

#블록 #코드덩어리 #메소드로 code를 넘길 수 있다.

Ruby Blocks

개요

- 코드 모음 (Chunks of code)
- {중괄호} 혹은 do ... end 사이에 있는 코드
- 하나의 파라미터로 메소드에 넘겨준다.

Ruby Blocks

개요

```
1.times { puts "hello world" }
```

```
2.times do |index|  
  if index > 0  
    puts index  
  end  
end
```

|| 사이에 parameter를 설정하여 값을 넘길 수 있다.

```
2.times { |index| puts index if index > 0 }
```

* Convention

코드가 한 줄일 경우 {중괄호}를 사용하고,

여러 줄 일 경우 do... end를 사용한다.

Ruby Blocks

사용방법

- 사용법 1. 함축적(implicit)
- 1) **block_given?**을 통해 block이 있는지 확인
- 2) **yield**를 활용하여 block 호출

```
def two_times_implicit
  return "No block" unless block_given?
  yield
  yield
end

puts two_times_implicit { print "Hello " } # => Hello
                                           # => Hello
puts two_times_implicit # => No block
```


Ruby Blocks

사용방법

- 사용법 2. 명시적(explicit)
- 1) 마지막 매개변수 앞에 **&**를 사용
- 2) **call** method를 활용하여 block 호출

```
def two_times_explicit (&i_am_a_block)
  return "No block" if i_am_a_block.nil?
  i_am_a_block.call
  i_am_a_block.call
end

puts two_times_explicit # => No block
two_times_explicit { puts "Hello" } # => Hello
                                     # => Hello
```


5. Ruby Files and ENV

#파일 R/W #예외처리 #환경변수 #Environment Variable

<https://ruby-doc.org/core-2.3.3/File.html>

Ruby Files

• 1. 파일 읽기

```
File.foreach( 'test.txt' ) do |line|  
  puts line  
  p line  
  p line.chomp # chops off newline character  
  p line.split # array of words in line  
end
```

Ruby Files

```
2.3.3 (main):0 > File.foreach('a.txt') do |line|
2.3.3 (main):0 *   puts lines.chomp
2.3.3 (main):0 * end
Errno::ENOENT: No such file or directory @ rb_sysopen - a.txt
from (pry):31:in `foreach'
```

Ruby Files

- 1. 파일 읽기
- 파일이 존재하지 않을 시 **예외처리 예시1.**

```
begin

  File.foreach( 'do_not_exist.txt' ) do |line|
    puts line.chomp
  end

rescue Exception => e
  puts e.message
  puts "Let's pretend this didn't happen..."
end
```

Ruby Files

- 1. 파일 읽기
- 파일이 존재하지 않을 시 예외처리 예시2.

```
if File.exist? 'test.txt'  
  File.foreach( 'test.txt' ) do |line|  
    puts line.chomp  
  end  
end
```

<http://ruby-doc.org/core-2.4.2/IO.html#method-c-new>

Ruby Files

• 2. 파일 쓰기

• **File.open("파일명", "옵션")**

"r" (read)

"w" (write)

"a" (append)

Mode	Meaning
"r"	Read-only, starts at beginning of file (default mode).
"r+"	Read-write, starts at beginning of file.
"w"	Write-only, truncates existing file to zero length or creates a new file for writing.
"w+"	Read-write, truncates existing file to zero length or creates a new file for reading and writing.
"a"	Write-only, starts at end of file if file exists, otherwise creates a new file for writing.
"a+"	Read-write, starts at end of file if file exists, otherwise creates a new file for reading and writing.
"b"	Binary file mode (may appear with any of the key letters listed above). Suppresses EOL <--> CRLF conversion on Windows. And sets external encoding to ASCII-8BIT unless explicitly specified.
"t"	Text file mode (may appear with any of the key letters listed above except "b").

+가 없으면 해당 기능만 가능, 있으면 읽고 쓰기 모두!

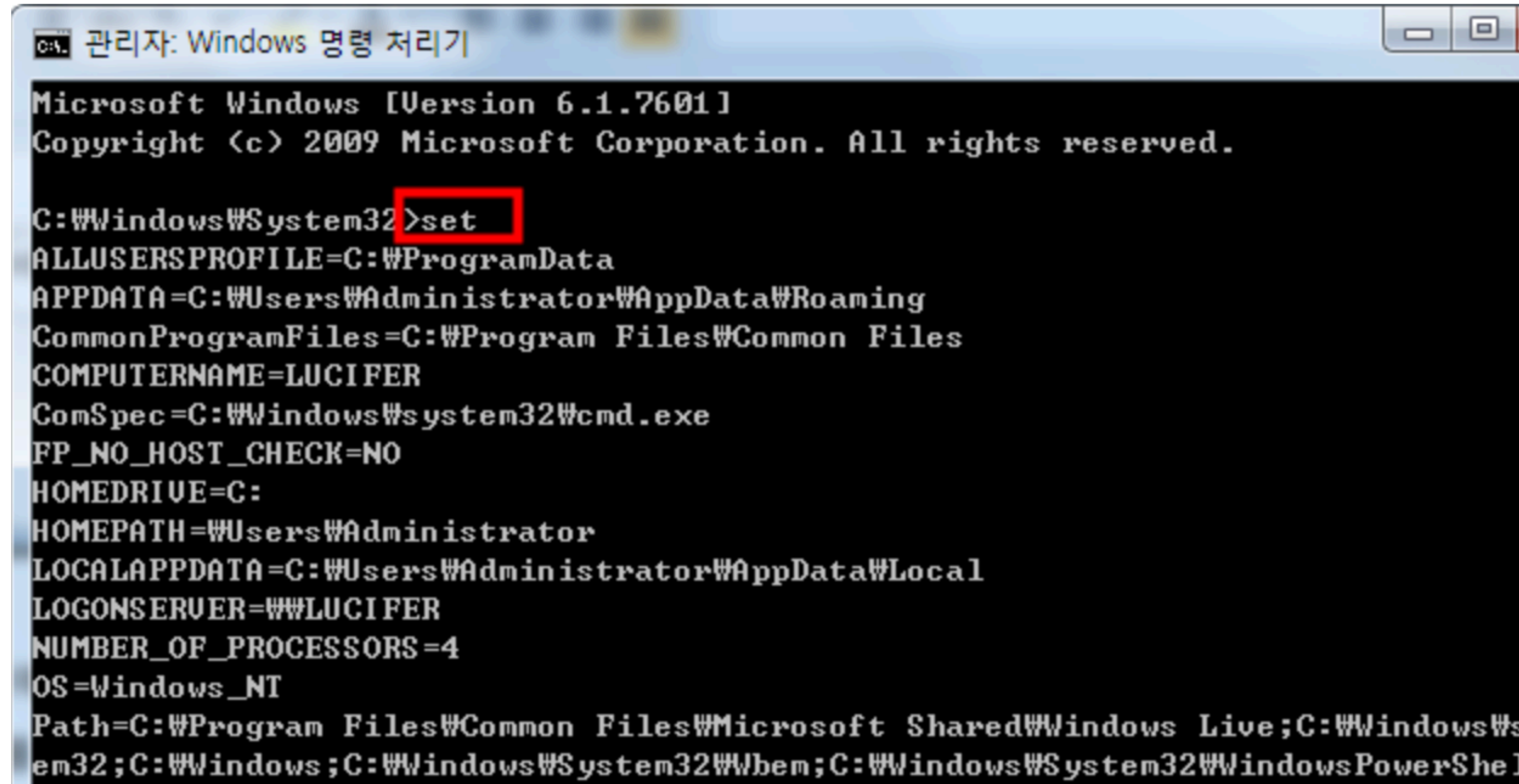
Ruby ENV

- OS에 설정된 환경변수를 사용할 수 있음

```
puts ENV["EDITOR"]
```


Ruby ENV

- Tip) 컴퓨터에 저장된 환경변수 확인하기



```
C:\> 관리자: Windows 명령 처리기

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Windows\System32>set
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\Administrator\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
COMPUTERNAME=LUCIFER
ComSpec=C:\Windows\system32\cmd.exe
FP_NO_HOST_CHECK=NO
HOMEDRIVE=C:
HOMEPATH=\Users\Administrator
LOCALAPPDATA=C:\Users\Administrator\AppData\Local
LOGONSERVER=\\LUCIFER
NUMBER_OF_PROCESSORS=4
OS=Windows_NT
Path=C:\Program Files\Common Files\Microsoft Shared\Windows Live;C:\Windows\S
em32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShe
```


6. Ruby Strings

#문자열 조작 #String API를 활용하면 편하게!
#Symbol

<https://ruby-doc.org/core-2.3.3/String.html>

“Hello”

Ruby Strings

- 시작하기전에,
 - 1) 'Single-quote'
 - ' 를 사용하기 위해서 \'
 - 거의 대부분 그대로 보여줌
 - 2) "Double-quote" %Q{ }도 큰따옴표와 같다!
 - 개행문자 사용 가능 ("wn" "wt")
 - 문자열 끼워넣기 사용가능
(String Interpolation)

Ruby Strings

```
2.3.3 (main):0 > a = "안녕하세요 . \n 멋쟁이사자처럼입니다 ."  
=> "안녕하세요 . \n 멋쟁이사자처럼입니다 ."  
2.3.3 (main):0 > b = '안녕하세요 . \n 멋쟁이사자처럼입니다 .'  
=> "안녕하세요 . \\n 멋쟁이사자처럼입니다 ."  
2.3.3 (main):0 > puts a  
안녕하세요 .  
  멋쟁이사자처럼입니다 .  
=> nil  
2.3.3 (main):0 > puts b  
안녕하세요 . \n 멋쟁이사자처럼입니다 .  
=> nil
```

Ruby Strings

```
2.3.3 (main):0 > name = '홍길동'
=> "홍길동"
2.3.3 (main):0 > a = "#{name}님 만나서 반가워요 ."
=> "홍길동님 만나서 반가워요 ."
2.3.3 (main):0 > b = '#{name}님 만나서 반가워요 .'
=> "\#{name}님 만나서 반가워요 ."
```

Ruby Strings

! (Bang Methods)

- 루비에서,
!는 메소드를 활용하여 객체의 내용을 바꾼다.

```
2.3.3 (main):0 > name = 'hong gildong'
=> "hong gildong"
2.3.3 (main):0 > name.upcase
=> "HONG GILDONG"
2.3.3 (main):0 > name
=> "hong gildong"
2.3.3 (main):0 > name.upcase!
=> "HONG GILDONG"
2.3.3 (main):0 > name
=> "HONG GILDONG"
```

`:foo`
`"foo".to_sym`
`:foo.to_s`

Symbol – string

Ruby Strings

Symbols

- Symbol은 문자열은 아니다.
- Symbols은 고유하고(unique) 변경이 불가능하다.
(immutable)
- Symbol – String은 서로 변환이 가능하다.

Ruby Strings Symbols

- **String**은 동일한 문자열도 서로 다른 공간(메모리)에 존재하나, **Symbol**은 변경 불가능(**immutable**)하기 때문에 메모리 상에서 동일한 객체로 존재
- 따라서, **Symbol**은 메모리나 성능에서 **String**보다 유리하므로 나중에 **Hash**에서 키는 **Symbol**로 생성
- **Symbol**은 따로 관리됨.

Symbol.all_symbols

Ruby Strings

Symbols

```
2.3.3 (main):0 > puts "string".object_id
70298024134500
=> nil
2.3.3 (main):0 > puts "string".object_id
70298020576060
=> nil
2.3.3 (main):0 >
2.3.3 (main):0 > puts :symbol.object_id
795548
=> nil
2.3.3 (main):0 > puts :symbol.object_id
795548
=> nil
```

Ruby Strings

Symbols

```
2.3.3 (main):0 > Symbol.all_symbols.last(10)
=> [
  [0] :goto,
  [1] :cursor,
  [2] :cursor=,
  [3] :pressed?,
  [4] :getpass,
  [5] :홍길동,
  [6] :uppercase,
  [7] :@__awesome_methods__,
  [8] :foo,
  [9] :claer
]
2.3.3 (main):0 > "hi".to_sym
=> :hi
2.3.3 (main):0 > Symbol.all_symbols.last(10)
=> [
  [0] :cursor,
  [1] :cursor=,
  [2] :pressed?,
  [3] :getpass,
  [4] :홍길동,
  [5] :uppercase,
  [6] :@__awesome_methods__,
  [7] :foo,
  [8] :claer,
  [9] :hi
]
```

[https://ruby-doc.org/
core-2.3.3/String.html](https://ruby-doc.org/core-2.3.3/String.html)

String은 유용한 것들이 많으므로

직접 한번씩 해보고 필요할 때마다 찾아봐야함.

7. Ruby Arrays

#배열

<https://ruby-doc.org/core-2.3.3/Array.html>

Ruby Arrays

- Object reference들의 집합.
- 자동적으로 배열의 크기가 늘어난다.

```
het_arr = [1, "two", :three] # heterogeneous types
puts het_arr[1] # => two (array indices start at 0)

arr_words = %w{ what a great day today! }
puts arr_words[-2] # => day
puts "#{arr_words.first} - #{arr_words.last}" # => what - today!
p arr_words[-3, 2] # => ["great", "day"] (go back 3 and take 2)

# (Range type covered later...)
p arr_words[2..4] # => ["great", "day", "today"]

# Make a String out of array elements separated by ','
puts arr_words.join(',') # => what,a,great,day,today!
```

Ruby Arrays

생성 및 Index

- 생성
 - `[obj1, obj2, ...]`
 - `Array.new(len)`
 - `%w {str1 str2}`
 - 생성시, 이질 배열(Heterogeneous type) 가능
`[1, "hi", :foo]`
- Index
 - `[index]` : index에는 음수 및 범위(range 가능)

Ruby Arrays

원소 추가/삭제

- 추가(Append)
 - `.push`
 - `<<`
- 삭제(Remove)
 - `.pop`
 - `.shift`

Ruby Arrays

기타

- 반복 (each, each_with_index)
- 모든 원소 수정 (map)
- 필터링 (select, reject, find)
- 등..

```
a = [1, 3, 4, 7, 8, 10]
a.each { |num| print num } # => 1347810 (no new line)
puts # => (print new line)

new_arr = a.select { |num| num > 4 }
p new_arr # => [7, 8, 10]
new_arr = a.select { |num| num < 10 }
              .reject{ |num| num.even? }
p new_arr # => [1, 3, 7]

# Multiply each element by 3 producing new array
new_arr = a.map { |x| x * 3 }
p new_arr # => [3, 9, 12, 21, 24, 30]
```

<https://ruby-doc.org/core-2.3.3/Array.html>

Array는 유용한 것들이 많으므로

직접 한번씩 해보고 필요할 때마다 찾아봐야함.

8. Ruby Range

#범위

<https://ruby-doc.org/core-2.3.3/Range.html>

Ruby Range

```
2.3.3 (main):0 > (1..5).to_a
=> [
  [0] 1,
  [1] 2,
  [2] 3,
  [3] 4,
  [4] 5
]
2.3.3 (main):0 > (1...5).to_a
=> [
  [0] 1,
  [1] 2,
  [2] 3,
  [3] 4
]
```

- 구간/조건 등에서 연속적인 것을 표현
Ex) 1..45, 'a'..'z'
- 메모리 효율적이다. (앞/끝 데이터만 저장)
- to_a를 통해 배열로 바꿀 수 있다.
- .. : included – included
- ... : included – excluded

```
some_range = 1..3
puts some_range.max # => 3
puts some_range.include? 2 # => true

puts (1...10) == 5.3 # => true
puts ('a'...'r') == "r" # => false (end-exclusive)

p ('k'..'z').to_a.sample(2) # => ["k", "w"]
# or another random array with 2 letters in range

age = 55
case age
  when 0..12 then puts "Still a baby"
  when 13..99 then puts "Teenager at heart!"
  else puts "You are getting older..."
end
# => Teenager at heart!
```

9. Ruby Hash

#해쉬

<https://ruby-doc.org/core-2.3.3/Hash.html>

Ruby Hashes

- **Object reference들의 인덱스된 집합**

배열 : Object reference들의 집합.

- **파이썬 딕셔너리와 같으며, Index(Key)는 배열과 같은 숫자를 제외한 어떠한 것도 가능하다.**
- **Key-Value로 구성된다.**

```
editor_props = { "font" => "Arial", "size" => 12, "color" => "red"}

# THE ABOVE IS NOT A BLOCK - IT'S A HASH
puts editor_props.length # => 3
puts editor_props["font"] # => Arial
```


Ruby Hashes

생성 / Index

- 생성
 - `Hash.new()`
 - `{key1: val1, ..}`
- Index
 - `[key]` 보통 `key`는 `string`이 아니라 `symbol`이다.
 - 존재하지 않는 `key`로 접근하게 되면,
`nil`이 리턴된다.

만약, `Hash.new()`로 생성하게 되면, `0`이 리턴됨.

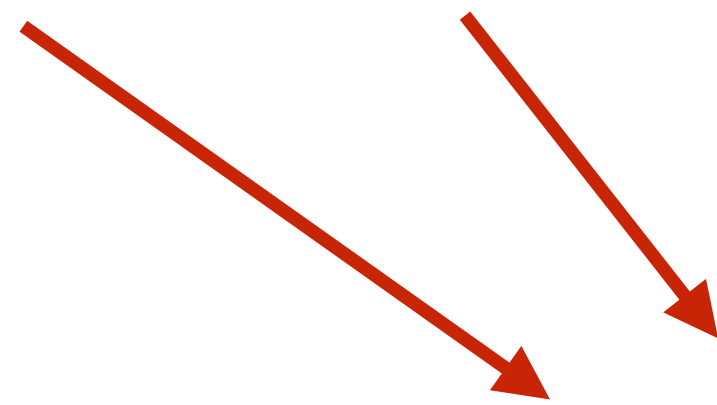
```
2.3.3 (main):0 > options = { :font_size => 10, :font_family => "Arial" }  
=> {  
  :font_family => "Arial",  
  :font_size => 10  
}  
2.3.3 (main):0 > options = { font_size: 10, font_family: "Arial" }  
=> {  
  :font_family => "Arial",  
  :font_size => 10  
}  
2.3.3 (main):0 > grades = { "Jane Doe" => 10, "Jim Doe" => 6 }  
=> {  
  "Jane Doe" => 10,  
  "Jim Doe" => 6  
}
```

```
2.3.3 (main):0 > h0 = Hash.new(0)
=> {}
2.3.3 (main):0 > h1 = Hash.new
=> {}
2.3.3 (main):0 > h0['key']
=> 0
2.3.3 (main):0 > h1['key']
=> nil
2.3.3 (main):0 > h0[:key] = 'value'
=> "value"
2.3.3 (main):0 > h0['key']
=> 0
2.3.3 (main):0 > h0[:key]
=> "value"
```

Ruby Hashes

반복

- Hash는 반복시 인자를 **key, value** 순으로 받는다.



```
2.3.3 (main):0 > h2.each do |k, v|
2.3.3 (main):0 *   puts "#{k}: #{v}"
2.3.3 (main):0 * end
a: 1
b: 2
c: 3
```

Ruby Hashes

- **Block과 헷갈리지 말자! 루비는 혼동한다...**

```
# Let's say you have a Hash
a_hash = { :one => "one" }

# Then, you output it
puts a_hash # => {:one=>"one"}

# If you try to do it in one step - you get a SyntaxError
# puts { :one => "one" }

# RUBY GETS CONFUSED AND THINKS {} IS A BLOCK!!!

# To get around this - you can use parens
puts ({ :one => "one" }) # => {:one=>"one"}

# Or drop the {} altogether...
puts one: "one" # => {:one=>"one"}
```

<https://ruby-doc.org/core-2.3.3/Hash.html>

Hash는 유용한 것들이 많으므로

직접 한번씩 해보고 필요할 때마다 찾아봐야함.

*** 알아두면 도움이 되는 55가지 루비 기법**

**[https://gist.github.com/
nacyot/7624036](https://gist.github.com/nacyot/7624036)**