1. Introduction to Ruby

2018년 3월 20일 화요일 오후 6:37

Introduction to Ruby

1. 루비란?

high-level: 영어를 읽듯이 읽고 쓰기 편하다.

interpreted: 컴파일러가 필요하지 않다.

Object-oriented: 객체지향언어 (객체라는 데이터구조)

Easy to use: 인간의 필요를 강조한 언어

2. Data type (데이터 형)

컴퓨터 프로그램은 데이터를 신속하게 분석하고 조작하기 위해!

=> 정보를 다른 유형으로 분리한다.

Numbers : 그냥 숫자로만 쓴다.

Booleans : true/false (이 때 따옴표를 치면 안된다.)

Strings : 'I'm learning Ruby!' : 따옴표로 string을 나타낸다.

3. Variables (변수)

변수는 단어 또는 이름으로 생각하면된다.

Ruby에서 변수를 선언하기 = 이름쓰기

Ruby에서 값을 할당하기 = 할당기호!!

4. Math (연산)

산술 연산자

추가	뺄셈	곱셈	몫	지수화	나머지
+	-	*	/	**	%

5. 출력

print : 당신의 줄에 그대로 화면에 출력한다.

```
print 'Oxnard Montalvo'
```

puts : 새로운 행을 추가해서 화면에 출력한다.

```
puts 'What's up?'
```

6. 루비는 모든 것이 객체입니다.

Ruby는 모든것이 하나의 객체이다.

Ruby의 모든 기능에는 메소드라고 내장된 기능이 있습니다.

메소드의 예시: 문자열의 길이, 문자열의 역방향을 알려주세요(내장메서드)

Interpreter: 당신이 쓴 코드를 받아서 run하는 것
editor에 code를 입력한다.
때 Interpreter가 editor의 code를 읽어서 그것의 결과를 console에 출력한다.

7. '.length' 메소드
메소드는 . 으로 호출한다
문자열의 길이를 알려준다.
(은행 암호입력에서 암호의 길이를 제한하는 경우)

8. '.reverse' 메소드
.length 메소드와 같은 방식으로 호출한다.
문자열의 역방향 버전으로 만든다.
(가장 낮은 값에서 높은 값으로 값 목록을 정렬하는 경우)

9. '.upcase' vs '.downcase'메소드
문자열을 모두 대문자로 변환한다.
문자열을 모두 소문자로 변환한다.

10. 한 줄 주석
Ruby의 주석 : #
코드를 명확히 할 수 있다.
코드를 일, 달 또는 수년전에 작성했을때 수행중인 작업을 상기 시켜준다.
=> 코드도 실행해도 실행이 되지 않는다.

11. 여러 줄 주석
=begin
=end
이때 공부를 넣지 않는게 중요하다.

12. 이름 지정 규칙
현재는 지역변수만 설명을 할 것이다.
- 변수는 소문자로 시작해야한다.
- 단어와 같이 밑줄로 구분해야한다.
- $나 @로 시작하는것은 다른 의미를 가진 변수이다.

13. 변수 및 데이터 유형
변수의 이름을 말하면 변수를 선언하는것
이를 이용해서 변수를 설정한다는 것

Introduction to Ruby

첫 번째 프로젝트

```
print "What's your first name? "
first_name = gets.chomp
first_name.capitalize!

print "What's your last name? "
last_name = gets.chomp
last_name.capitalize!

print "What city are you from? "
city = gets.chomp
city.capitalize!

print "What state or province are you
from? "
state = gets.chomp
state.upcase!

puts "Your name is #{first_name} #
{last_name} and you're from #{city}, #
{state}!"
```

3. Getting Input

gets: 사용자로부터 입력을 받는 루비 메소드

입력을 받으면 ruby는 각 입력 비트 뒤에 공백행을 추가한다.

.chomp: 여분의 줄을 제거한다.

5. Printing the Output

String interpolation: 문자열 보간법

#{변수이름}을 ""안에 넣으면 해당 변수의 값이 문자열에 대체되서 나온다.

이때 ''(작은따옴표)로 잘못쓰지 않도록 조심한다.

6. Formatting with String Methods

.capitalize!

변수 자체에 포함된 값을 수정한다. => 메소드 사용후 변수 할당을 따로 안해도 된다.

2. Control Flow in Ruby

2018년 3월 20일 화요일 오후 6:38

Control Flow in Ruby

1. How it Works

입력에 따라 항상 동일한 결과를 생성한다: 환경에 대한 반응을 변화시키지 않았다.

Control Flow : 사용자가 입력하는 정보, 계산 결과 또는 프로그램의 다른 부분에서 반환 한 값에 따라 다른 결과를 선택할 수 있다.

2. If

true나 false로 값을 판단할 수 있는 표현식을 받는다.

이때, true이면 그 다음에 따르는 코드를 실행한다.

>> false인 경우에는 해당 코드 블록을 실행하지 않는다. (다음으로 넘어간다.)

ruby는 공백을 신쓰지 않기 때문에 print문을 들여쓰는것이 필요하지 않는다. // 그치만 습관!

if문은 end로 끝낸다.

```
if 1 < 2
  print "I'm getting printed
because one is less than two!"
end
```

3. Else

if의 파트너는 else이다.

if/else문은?

"이 표현이 사실이라면 if블록을 실행한다. 그렇지 않으면 else문에 있는 코드를 실행한다."

```
if 1 > 2
  print "I won't get printed
because one is less than two."
else
  print "That means I'll get
printed!"
end
```

4. Elsif

두 개 이상의 옵션을 원할 때 elsif 구조를 사용한다. => 대안의 수 추가

```ruby
if x < y  # Assumes x and y are
defined
  puts "x is less than y!"
elsif x > y
  puts "x is greater than y!"
else
  puts "x equals y!"
end
```

5. Unless

거짓인지를 확인할때! 사용한다. (if에서 else사용도 가능하지만 이것이 더유용하다!)

```ruby
unless hungry
  # Write some sweet programs
else
  # Have some noms
end
```

6. Equal or Not?

= : 변수를 할당할때 (할당 연산자: assignment operator)

== : 두개의 값이 같은지를 확인할 때 비교!! (관계 연산자: relational operator)

!= : 두개의 값이 다른지를 확인 할 때!

```ruby
x = 2
y = 2
if x == y
  print "x and y are equal!"
end
```

7. Less Than or Creater Than

미만	작거나 같음	큰	크거나 같음
<	<=	>	>=

8. 논리연산자 AND (&&)

true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

두개다 true일때 true의 결과가 나온다.

9. 논리연산자 OR (||) => inclusive

true	true	**true**
true	false	**true**
false	true	**true**

false	false	**false**

둘중 하나라도 true일때 true의 결과가 나온다.

10. 논리연산자 NOT (!)

!true	**false**
!false	**true**

반대의 값을 가진다.

11. Combining Boolean Operators

표현식에 부울 연산자를 결합 할 수 있다.

=> 평가 순서를 제어하기 위해 괄호를 사용한다. => 괄호안이 괄호 밖보다 먼저 계산

Thith Meanth War!

1. How it Works

```
print "Thtring, pleathe!: "
user_input = gets.chomp
user_input.downcase!

if user_input.include? "s"
  user_input.gsub!(/s/, "th")
else
  puts "Nothing to do here!"
end

puts "Your string is: #{user_input}"
```

4. Setting Up the 'If' Branch

.include? "char" :

>>문자열에 "char"가 들어갔는지 체크하는 메소드 : 따옴표 안은 수정할 수 있다.

gsub(/char/, "replace") :

>>특정 문자열 대신에 다른 문자열을 삽입하고 싶을 때 사용한다. (global substitution)

이때 ""대신에 //가 들어감을 주의하자!

gsub!와 괄호사이에 공백을 주면안된다.

메소드 이름 끝에 '!'는 ruby가 바뀐 값을 변수에 바로 저장하도록한다.

3. Looping with Ruby

2018년 3월 20일 화요일 오후 6:38

Loops & Iterators

1. While Loop

때로는 특정 조건이 참인 동안 Ruby에서 액션을 반복하고 싶지만 반복 횟수를 모를 수 있다.

ex) 사용자에게 특정 유형의 입력을 요구한다. (잘못된 타입인 경우 여러 번 다시 요청한다)

While: 특정 조건이 참인지 확인하고 실행을 계속한다. / 이 조건에 맞지 않으면 loop를 멈춘다.

```
counter = 1
while counter < 11
  puts counter
  counter = counter + 1
end
```

2. Danger: Infinite Loops

만약 while에서 counter가 계속 1이었다면? 계속 counter<11의 조건을 만족해서 loop가 끝나지 않았을 것이다. 이렇게 loop가 무한히 반복되는 것을 infinite loop라고 한다.

3. Until Loop

while루프의 보안 버전이다!

```
i = 0
until i == 6
  i = i + 1
end
puts i
```

1) 변수i를 0값으로 초기화 한다.

2) i가 6이 될 때까지 코드를 실행하는데 이때 i의 값이 1씩 증가한다.

3) i가 6과 같다면 해당 block이 멈춘다.

4) 결국 i의 값으로 6이 콘솔에 프린트 된다.

4. Assignment operaotr(대입 연산자)

+=	-=	*=	/=
var = var + 1 var += 1	var = var - 1 var -= 1	var = var * 1 var *= 1	var = var / 1 var /= 1

5. For Loop

```
for num in 1...10
  puts num
end
```

loop를 몇번 반복 해야하는지 알고있는 경우 필요하다

6. Inclusive and Exclusive Ranges

```
for num in 1...10
```

3개의 점(...)을 사용하면 num의 1에서 9까지의 값을 취한다. // 반복에서 10은 제외한다.

```
for num in 1..10
```

2개의 점(..)을 사용하면 num의 1에서 10까지의 값을 취한다. // 반복에서 10을 포함한다.

8. The Loop Method

iterator : 코드 블록(명령어들)을 반복 할 수 있는 ruby의 메서드이다.

가장 간단한 iterator은 loop 메소드이다.

```
loop { print "Hello, world!" }
```

ruby의 중괄호 {}는 일반적으로 do end와 바꿔서 사용할 수 있다.

do : 여는 괄호 {

end: 닫는 괄호 }

```
i = 0
loop do
  i += 1
  print "#{i}"
  break if i > 5
end
```

break 키워드 : 루프를 탈출한다 (여기서는 if를 사용했으므로 조건이 충족되면 루프를 탈출한다.)

9. Next!

next 키워드 : 루프의 특정 단계를 건너 뛸 수 있다.

```
for i in 1..5
  next if i % 2 == 0
  print i
end
```

짝수를 인쇄하고 싶지 않을 때 // 짝수 조건일 경우 루프의 다음 반복으로 간다.

10. Saving Multiple Values

배열 : 여러 변수를 다일 변수에 묶을 수 있다.

대괄호를 통해 항목들의 목록을 묶을 수 있다.

순서대로 정렬할 필요가 없다.

```
[1, 2, 3, 4]
```

11. The .each Iterator

.each : 한번에 하나씩 개체의 각 요소에 식을 적용할 수 있는 메서드이다. (iterator)

```
object.each do |item|
  # Do something
end
```

변수이름은 원하는 대로 ||사이에 지정 할 수 있다. => 이 변수는 해당 루프에서만 사용된다.

13. The .times Iterator

.times : 초소형 for루프와 같다. 지정된 횟수만큼 반복한다.

```
10.times { print "Chunky bacon!" }
```

Redacted!

1. What You'll Be Building

```
puts "Text to search through: "
text = gets.chomp
puts "Word to redact: "
redact = gets.chomp

words = text.split(" ")

words.each do |word|
  if word != redact
    print word + " "
  else
    print "REDACTED "
  end
end
```

3. The .split Method

.split: 문자열을 받아 배열을 반환한다.

괄호 안에 텍스트를 전달하면 그 텍스트를 볼 때마다 문자열을 나눈다.

```
text.split(",")
```

, 를 볼 때마다 문자열을 분할하도록 ruby에 지시한다 (사용자의 입력을 개별 단어로 나눌 수 있다.)

4. Arrays and Hashes

2018년 3월 20일 화요일 오후 6:38

Data Structures

1. Creating Arrays

배열: 하나의 변수에 값의 목록을 저장 할 수 있다.

2. Access by Index

배열은 각요소에 index가 부여되어있다.

```
        +---+---+---+---+---
+
array   | 5 | 7 | 9 | 2 | 0
|
        +---+---+---+---+---
+
index     0   1   2   3   4
```

index를 이용해서 배열의 특정 요소에 접근 할 수 있다.

```
array = [5, 7, 9, 2, 0]
array[2]
```

3. Arrays

배열에는 숫자만 넣을 수 있는게 아니다. 객체배열, 문자열 배열 등등을 만들 수 있다.

4. Arrays Of Arrays

다차원배열: 배열안에 배열이 들어갈 수 있다.

```
array = [[thing, thing],
[thing, thing]]
```

6. Introduction to Hashes

배열에는 0부터 배열길이 - 1 까지의 인덱스가 오는 숫자로 값을 불러낼 수 있다.

하지만 숫자가 아니라 값으로 인덱스를 사용하고 싶을 때는 어떻게 해야할까?

해시(Hash) : 키 - 값 쌍의 집합.

```
hash = {
  key1 => value1,
  key2 => value2,
  key3 => value3
}
```

인덱스로 값을 호출하듯이 키로 값을 호출 할 수 있다.

```
puts  my_hash [ "name" ]
```

해쉬 리터럴 표기법 (Hash Literal Notation) : 해시에서 원하는 것을 문자 그대로 묘사해서

7. Using Hash.new

```
my_hash = Hash.new
```

다음과 같이 새로운 해시를 생성할 수 있다. (대괄호로 해야하는 것을 꼭 기억하자)
다음과 같이 설정하면 my_hash에 빈 중괄호{} 가 생성된다.

8. Adding to a Hash

ㄱ. 리터럴 표기법: 중괄호 사이에 새로운 키-값 쌍을 직접 추가한다.
ㄴ. Hash.new : 아래와 같이 추가한다.

```
pets = Hash.new
pets["Stevie"] = "cat"
```

9. Accessing Hash Values

배열과 마찬가지로 해시 값을 접근 할 수 있다.

```
pets = {
  "Stevie" => "cat",
  "Bowser" => "hamster",
  "Kevin Sorbo" => "fish"
}

puts pets["Stevie"]
# will print "cat"
```

10. (Re)Introduction to Iteration

loop에서 iteratior를 사용했을 때처럼 해시에서도 사용할 수 있다.
.each : 해당 섹션 안에있는 배열과 해시에 사용한다.

```
friends = ["Milhouse", "Ralph", "Nelson",
"Otto"]

family = { "Homer" => "dad",
  "Marge" => "mom",
  "Lisa" => "sister",
  "Maggie" => "sister",
  "Abe" => "grandpa",
  "Santa's Little Helper" => "dog"
}

friends.each { |x| puts "#{x}" }
family.each { |x, y| puts "#{x}: #{y}" }
```

11. Iterating Over Arrays

```
numbers = [1, 2, 3, 4, 5]
numbers.each { |element| puts element }
```

배열 반복

12. Iterating Over Multidimensional Arrays

```ruby
s = [["ham", "swiss"], ["turkey", "cheddar"],
["roast beef", "gruyere"]]
```

해당 다차원 배열에서 "swiss"에 접근하고 싶다면, s[0][1]로 접근 할 수 있다.

다차원 배열을 iterator를 사용해서 각 요소에 접근하고 싶다면 다음과 같이 2중으로 사용

```ruby
s.each do | sub_array |
  sub_array.each do | y |
    puts y
  end
end
```

13. Iterating Over Hashes

Iterator를 해시에서 사용할 때는 각 키-값 쌍을 나타내는 두개의 변수가 필요하다.

```ruby
restaurant_menu = {
  "noodles" => 4,
  "soup" => 3,
  "salad" => 2
}

restaurant_menu.each do |item, price|
  puts "#{item}: #{price}"
end
```

다음과 같이 앞에 변수에는 키가, 뒤에 변수에는 값이 각각 할당 된다

Create Histogram

1. What You'll Be Building

```ruby
puts "Text please: "
text = gets.chomp

words = text.split(" ")
frequencies = Hash.new(0)
words.each { |word| frequencies[word] += 1 }
frequencies = frequencies.sort_by {|a, b| b }
frequencies.reverse!
frequencies.each { |word, frequency| puts
word + " " + frequency.to_s }
```

4 Creating the Frequencies Hash

```ruby
h = Hash.new("nothing here")
```

해시 값의 기본값 설정은 () 안에 한다.

만약 존재하지 않는 키에 접근하려고 하면 해당 기본값이 나온다.

위 예제에서는 "nothing here"이라는 값이 출력이 된다.

5. Iterating Over the Array

each를 사용해서 해시에 할당할 값을 1씩 증가하면서 저장 할 수 있다.

=> 1번에서는 사용자 입력 받은 문자열의 단어들은 배열로 words에 저장한후.

each로 각 단어에 접근해서. frequencies[word] += 1 을 통해

해당 해시에 접근할 때 마다 숫자가 1씩 증가한다!

=> 초기값을 0으로 지정했기 때문에 접근되지 않으면 해시의 값은 0이다.

6. Sorting the Hash

```
colors = {
  "blue" => 3,
  "green" => 1,
  "red" => 2
}
colors = colors.sort_by do |color, count|
  count
end
colors.reverse!
```

.sort_by : 배열을 우리의 목적에 맞는 배열로 반환한다.

=> 여기서는 |color, count| count로 썼기 때문에 count기준으로 정렬

```
[["green", 1], ["red", 2], ["blue", 3]]
```

다음 배열로 출력된다.

.reverse! : 배열을 역순으로 정렬한다.

7. Iterating Over the Hash

```
frequencies.each do |word, frequency|
  puts word + " " + frequency.to_s
end
```

.each를 사용해서 키-값 쌍을 반복 할 수 있다.

.to_s : string타입으로 변환한다. (숫자에 ""를 붙인다.)

5. Blocks and Sorting

2018년 3월 20일 화요일 오후 6:38

Methods, Blocks & Sorting

1. Why Methods?

Method? : 프로그램에서 특정 작업을 수행하기 위해 작성된 코드를 재사용하기 위한 것

=> 프로그램의 부분을 분리해서 필요한 부분만 사용한다.

Method의 장점

ㄱ. 코드에서 문제가 발생하면 버그를 찾고 수정하는 것이 쉽다. (메소드로 조직화가 되어있어서)

ㄴ. 특정 작업을 별도의 method에 할당하면 프로그램을 덜 중복화 하고,
코드를 재사용 할 수 있게 해준다. 매번 다시 작성하지 않고, 한 프로그램에서 반복적으로!

ㄷ. 객체에 대해 배울때 ruby의 메소드로 할 수 있는 흥미로운 점이 많다는 것을 알게 될 것이다.

2. Method Syntax

메소드는 키워드 def를 사용하여 정의된다. (define)

메소드의 3가지 부분

ㄱ. Header: def를 포함한 부분, method의 이름, method의 인수(arguments)부분

ㄴ. Body: code block의 부분, 각종 공백과 규칙으로 이루어져있다. (for, if, elsif, else)

ㄷ. end : method가 끝나는 부분

```ruby
def welcome
  puts "Welcome to Ruby!"
end
```

4. Call it

메소드를 정의하는 건 좋지만, 호출을 해서 프로그램에 실행을 지시했을 때 유용하다.

메소드를 호출할때는 그저 메소드의 이름을 타이핑 하면된다.

5. Parameters and Arguments

만약 메소드가 인수(arguments)를 취하면 우리는 인수를 accepts or expects 했다고 한다.

인수 (arguments): method를 호출할 때 실제로 그 메소드의 괄호안에 넣은 코드!

매개변수 (parameter) : method를 정의할 때 그 괄호안에 정의한 이름!

```ruby
def square(n)
  puts n ** 2
end
square(12)
# ==> prints "144"
```

다음에서는 매개변수 n을 지정하고 method를 호출할 때 12라는 인수를 전달했다.

6. Splat!

Splat argument : 앞에 *가 붙은 인수

메소드가 하나 이상의 인수를 수신 할 수 있음을 프로그램에 알려준다.

```
def what_up(greeting, *friends)
  friends.each { |friend| puts "#{greeting}, #{friend}!" }
end

what_up("What up", "Ian", "Zoe", "Zenas", "Eleanor")
```

7. Let's Learn Return

return : method를 이용해서 출력만하기 보다는, 해당 메소드의 실질적인 값을 돌려준다.

```
def double(n)
  return n * 2
end

output = double(6)
```

output에는 12가 저장이 되어있다. (실제로 값을 나타냄)

9. Blocks Are Like Nameless Methods

대부분의 메소드는 사용자 또는 다른사람이 지정한 이름이 있다. ([array], sort(), "string", downcase())

block: 이름이 없는 메소드를 만드는 방법

```
1.times do
  puts "I'm a code block!"
end

1.times { puts "As am I!" }
```

block은 do / end로 혹은 { }으로 정의 될 수 있다.

10. How Blocks Differ from Methods

```
# method that capitalizes a word
def capitalize(string)
  puts "#{string[0].upcase}#{string[1..-1]}"
end

capitalize("ryan") # prints "Ryan"
capitalize("jane") # prints "Jane"

# block that capitalizes each string in the array
["ryan", "jane"].each {|string| puts "#{string[0].upcase}#
{string[1..-1]}"} # prints "Ryan", then "Jane"
```

capitalize method같은 경우, 해당 메소드의 이름으로 호출을 할 수 있다.

block(.each)는 오로지 1번만 호출 할 수 있다.

11. Using Code Blocks

메소드는 블록을 매개변수로 사용할 수 있다.

메소드에 블록을 전달한다: 우리가 메소드를 호출 할 때 abstracting한 방법!

=> abstraction: making something simpler

=> 나열 대신 .each를 사용하는 것처럼 단순화

12. Introduction to Sorting

.sort! : ruby의 기본적인 정렬 알고리즘

=> 낮은 수 부터 높은 수 까지

=> 알파벳의 a에서부터 z까지

14. The Combined Comparison Operator (결합된 비교연산자)

a <=> b

0: a와 b의 값이 같다.

1: a가 b보다 크다.

-1: a가 b보다 작다.

15. Getting Technical

```
books.sort! do |firstBook, secondBook|
  firstBook <=> secondBook
end
```

이 sort 메서드는 기본적으로 오름차순으로 정렬하려고한다고 가정하지만 프로그래머가 두 항목의 비교 방법을 지정할 수있는 선택적 인수로 블록을 허용합니다.

my : 오름차순 정렬 후 reverse!

Ordering Your Library

1. What You'll Be Building

```
def alphabetize(arr, rev=false)
  if rev
    arr.sort { |item1, item2| item2 <=> item1 }
  else
    arr.sort { |item1, item2| item1 <=> item2 }
  end
end

books = ["Heart of Darkness", "Code Complete", "The Lorax",
"The Prophet", "Absalom, Absalom!"]

puts "A-Z: #{alphabetize(books)}"
puts "Z-A: #{alphabetize(books, true)}"
```

3. Default Parameters

첫번째 매개변수 arr, 두번째 매개변수 rev

여기서 두번째 매개변수 rev=false의 의미는

사용자가 두개의 인수를 입력하지 않으면 rev의 기본값이 false가 된다.

=> 두번째 인수값에 false, true를 입력해야한다. (false이면 원래대로 / true이면 rev대로)

6. Hahses and Symbols

2018년 3월 20일 화요일 오후 6:38

Hashes and Symbols

1. The story So Far

ㄱ. 해시 리터럴 표기법 (hash literal notation)

```
new_hash = { "one" => 1 }
```

ㄴ. 해시 생성자 표기법 (hash constructor notation)

```
new_hash = Hash.new
```

2. Iterating Over Hashes

.each 메소드로 해시를 반복 할 수 있다.

```
my_hash.each do |key, value|
  puts my_hash[]
end
```

키와 값의 목록이 my_hash 각각의 행에 인쇄된다.

3. Nil: a Formal Introduction

각종 언어에서 몇몇의 error는 ruby에서는 모두 특별값인 nil로 표기가된다.

루비애서 false와 nil은 non-true라고 생각하면된다.

그렇지만 두개의 뜻은 같지 않다.

false: not true

nil: "nothing at all"

4. Setting Your Own Default

그래서 default값을 nil로 두지 않기위해 Hash.new()에서 괄호안에 초기값을 설정 할 수 있다.

5. A Key of a Different Color

그동안은 문자열("")을 해시의 키로 사용해왔다.

이때 key를 다른 색으로 지정하기위해 다른 symbol을 사용 할 수 있다. (:)

```
menagerie = { :foxes => 2,
  :giraffe => 1,
  :weezards => 17,
  :elves => 1,
  :canaries => 4,
  :ham => 1
}
```

6. What's a Symbol?

루비의 심볼을 일종의 이름으로 생각할 수 있다. (Symbol은 문자열이 아니다.)

```ruby
"string" == :string # false
```

같은 이름을 가진 string은 다른 것으로 인식하지만, 같은이름을 가진 symbol은 값은 것으로 인식한다.

```ruby
puts "string".object_id
puts "string".object_id

puts :symbol.object_id
puts :symbol.object_id
```

```
18032800
18022200
802268
802268
```

.object_id : 객체의 ID를 가져온다. (두 객체가 정확히 같은 객체인가?)

7. Symbol Syntax

ㄱ. 심볼은 항상 콜론(:)으로 시작한다.

ㄴ. 심볼의 콜론 뒤에 오는 첫번째 문자는 문자 또는 밑줄(_)이어야 한다.

ㄷ. 기호이름에 공백을 넣지 마시오

```ruby
:my symbol # Don't do this!
:my_symbol # Do this instead.
```

8. What are Symbols Used For?

심볼은 주로 해시 키 (hash key) 혹은 참조 메소드 (reference method)이름으로 사용된다.

```ruby
sounds = {
  :cat => "meow",
  :dog => "woof",
  :computer => 10010110,
}
```

심볼이 해시키를 만드는 이유

ㄱ. 변경 불가능하다 (생성되면 변경할 수 없다.)

ㄴ. 한번에 하나의 심볼 사본만 존재해서 메모리가 절약된다.

ㄷ. 위의 두가지 이유로 문자열로 된 키보다 심볼로 된 키가 빠르다.

9. Converting Between Symbols and Strings

문자열과 심볼간의 변환

```ruby
:sasquatch.to_s
# ==> "sasquatch"

"sasquatch".to_sym
# ==> :sasquatch
```

10. Many Paths to the Same Summit

ruby에는 항상 뭔가를 성취하기 위한 다양한 방법이 존재한다.

.to_sym대신 .intern이 있다.

문자열을 기호로 내재화한다.

```
"hello".intern
# ==> :hello
```

11. All Aboard the Hash Rocket!

해시 로켓 스타일(Hash Rocket) : 키와 값 사이에 =>기호가 있는 해시구문

=> 가 작은 로켓처럼 보이기 때문이다.

```
numbers = {
  :one => 1,
  :two => "two",
  :three => 3,
}
```

12. The Hash Rocket Has Landed

그러나 해시 구문이 ruby 1.9v에서 변경되었다.

```
new_hash = {
  one: 1,
  two: 2,
  three: 3
}
```

ㄱ. 시작 부분이 아니라 기호의 끝에 콜론을 넣는다.

ㄴ. 더 이상 해시 로켓이 필요하지 않는다.

이때 키의 시작부분 대신 끝에 콜론이 있지만 이것이 여전히 symbol임을 유의한다.

14. Becoming More Selective

.select : 특정 기준을 충족하는 값에 대해 해시를 필터링 하는 방법을 쓰기 위한 메소드

자신이 설정한 조건과 일치하는 키-값 쌍을 선택 할 수 있다.

```
grades = { alice: 100,
  bob: 92,
  chris: 95,
  dave: 97
}

grades.select { |name, grade| grade <  97 }
# ==> { :bob => 92, :chris => 95 }

grades.select { |k, v| k == :alice }
# ==> { :alice => 100 }
```

15. More Methods, More Solutions

.each_key : 단지 키로만 작업하는 방법

.each_value : 단지 값으로만 작업하는 방법

```ruby
my_hash = { one: 1, two: 2, three: 3 }

my_hash.each_key { |k| print k, " " }
# ==> one two three

my_hash.each_value { |v| print v, " " }
# ==> 1 2 3
```

A Night At The Movies

1. What You'll Be Building

```ruby
movies = {
  Memento: 3,
  Primer: 4,
  Ishtar: 1
}

puts "What would you like to do?"
puts "-- Type 'add' to add a movie."
puts "-- Type 'update' to update a movie."
puts "-- Type 'display' to display all movies."
puts "-- Type 'delete' to delete a movie."

choice = gets.chomp.downcase
case choice
when 'add'
  puts "What movie do you want to add?"
  title = gets.chomp
  if movies[title.to_sym].nil?
    puts "What's the rating? (Type a number 0 to 4.)"
    rating = gets.chomp
    movies[title.to_sym] = rating.to_i
    puts "#{title} has been added with a rating of #
{rating}."
  else
    puts "That movie already exists! Its rating is #
{movies[title.to_sym]}."
  end
when 'update'
  puts "What movie do you want to update?"
  title = gets.chomp
  if movies[title.to_sym].nil?
    puts "Movie not found!"
  else
    puts "What's the new rating? (Type a number 0 to 4.)"
    rating = gets.chomp
    movies[title.to_sym] = rating.to_i
    puts "#{title} has been updated with new rating of #
{rating}."
  end
when 'display'
  movies.each do |movie, rating|
    puts "#{movie}: #{rating}"
  end
```

3. The Case Statement

case: if/else가 강격하지만 if와 elsif수가 많은경우 속도가 느릴 수 있다. => 대안으로 사용 case

```
case language
  when "JS"
    puts "Websites!"
  when "Python"
    puts "Science!"
  when "Ruby"
    puts "Web apps!"
  else
    puts "I don't know!"
end
```

10. Nice Work!

프로그램의 add, display, update, delete는 보편적이다.

CRUD : Create Read Update Delete

각각 데이터베이스에서 항목을 업데이트하거나, 웹 사이트에 정보를 요청하거나 블로그에 게시물을 쓸 때 취하는 조치이다. 이 설정에 익숙하면 API호출부터 Ruby on rails와 같은 웹 프레임 워크에 이르기까지 모든 것을 볼 수 있다.