hci+d lab.

Week 05 • 소셜네트워크 데이터마이닝과 분석

Iterators, More Methods and Classes
Hash, Regex, File I/O

Joonhwan Lee
human-computer interaction + design lab.

오늘 다룰 내용

- Iterators
- Writing Methods
- Classes & Objects
- Hash
- File I/O
- Quiz 4

1. Iterators

Array Iterators: each

- each: loop의 역할을 하는 메소드로 Array 클래스가 제공
  - *array*`.each do |variable|`
    `code`
    `end`

```
array = [1, 2, 3, 4, 5]
array.each do | x |
  puts x
end
```

Number Iterators: times

- times: loop의 역할을 하는 메소드로 Integer 클래스가 제공
  - *integer*.**times** do
    code
    end

```
i = 0
10.times do
  puts i
  i = i + 1
end
```

Range

- Range
  - 1st ~ 31st
  - January to December
  - line number 23 to 32
  - a 부터 f 까지
- ~, to, ~부터~까지 : range 를 표현하는 말
  - o부터 x까지 값이 하나씩 증가한다고 인식
- in Ruby
  - **(1..9)** => 1, 2, 3, 4, 5, 6, 7, 8, 9
  - **(1...9)** => 1, 2, 3, 4, 5, 6, 7, 8
  - **('a'..'d')** => 'a', 'b', 'c', 'd'

Range

```
(1..5).each do |x|
  puts x
end
```

```
(1..5).to_a
```

```
('a'..'f').min
```

```
('a'..'f').max
```

```
('a'..'f').include?('c')
```

Range

http://www.ruby-doc.org/core-2.0.0/Range.html

```
(1..5).each do |x|          => 1
  puts x                      2
end                           3
                              4
                              5
```

```
(1..5).to_a                 => [1, 2, 3, 4, 5]

('a'..'f').min              => 'a'

('a'..'f').max              => 'f'

('a'..'f').include?('c')    => true
```

Ranges as Conditions

- ```
  for i in 0..5
    puts "current i value is #{i}"
  end
  ```

- ```
  (0..5).each do | i |
    puts "current i value is #{i}"
  end
  ```

Lab 1: 1부터 1000까지 계산

- 1부터 1000까지 더한 값과 곱한 값을 각각 출력하시오.

- Result of addition from 1 to 1000 is: 500500.
- Result of multiplication from 1 to 1000 is: 402387260077093.....

2. Writing Methods

Iterators & Loops

- Iterator와 Loop는 동일한 코드를 계속 입력하지 않고도 같은 일을 반복할 수 있다.
- 그럼에도 불구하고 iterator 나 loop로는 해결이 안되는 문제들이 있음.
  - lab2.rb ← 비슷한 반복이 끊임없이 일어남.
  - Lab 2와 같은 경우, 메소드를 직접 작성하는 방법으로 프로그램을 간단하게 만들 수 있음.
  - String, Integer, Array 등의 클래스 → 계속 반복 사용되는 코드를 메소드로 정의하여 간단하게 호출해서 사용.

메소드를 만들자

- 메소드를 만드는 방법
  - def ***method_name***
    code
    code
    ...
    end
  - def hello
    puts "hello, ruby programmer~!"
    end
- 메소드의 호출
  - hello
    => "hello, ruby programmer~!"

메소드에 값을 전달

- 메소드 이름 옆에 패러미터를 정의하여 전달하고자 하는 값을 제공할 수 있다.

- `def method_name value`

```
    code
end
```

Lab 3.1

- **def hello name**

```
  puts "hello, #{name}"
end
=> hello "ruby"
   hello "java"
   hello "programmer"
```

메소드에 값은 두개 이상 전달 가능

- 패러미터를 하나 이상 만들 수 있다.

- def *method_name* ***value1, value2***
  code
  end

Lab 3.2

- **def sayName name, times**
  puts "hello, #{name*times}"
  end
  => sayName "ruby", 4
  sayName "java", 10
  sayName "programmer", 0

메소드의 로컬변수

- 메소드 밖에서 사용된 변수와 메소드 안에서 사용된 변수가 서로 이름이 같더라도 둘은 **완전히** 다른 변수

Lab 3.4

```
num = 10
def change_value num
  num = 2
  puts num
end

change_value num => 2
puts num => 10
```

메소드가 반환하는 값

- a = "12345"
  b = a.to_i

- 어떤 메소드는 처리된 결과를 "반환(return)"하여 준다.
- 반환된 값은 다른 변수에 값을 지정할 수 있음.
- 예:
  - x = rand(1..5)
  - y = Math.sqrt(128)

메소드가 반환하는 값

- 메소드가 값을 반환하기 위해선 return을 위한 변수가 있어야 함.

Lab 4.1

```
def square(x)
   puts (x * x)
end

square(5)
a = square(10)
puts a
```

메소드가 반환하는 값

- 메소드가 값을 반환하기 위해선 return을 위한 변수가 있어야 함.

Lab 4.1

```
def square(x)
   puts (x * x)
end

square(5)          => 25
a = square(10)     => 100
puts a             => nil
```

메소드가 반환하는 값

- square 메소드를 다음과 같이 수정

Lab 4.2

- **def square1(x)**

```
  x * x
end

square1(5)
a1 = square1(10)
puts a1
```

메소드가 반환하는 값

- square 메소드를 다음과 같이 수정

Lab 4.2

- **def square1(x)**

```
  x * x
end

square1(5)              =>
a1 = square1(10)        =>
puts a1                 => 100
```

메소드가 반환하는 값

Lab 4.3

```
def square2(x)
    result = x * x
    result
end
```

Lab 4.4

```
def square3(x)
    return x * x
end
```

Lab 5: 메소드의 사용

- 다음의 메소드를 사용해서 lab2.rb 를 수정해 보자.

```
def ask(question)
  while true
    puts question
    answer = gets.chomp.downcase
    if (answer == 'yes' || answer == 'no')
      if answer == 'yes'
        wets_bed_result = true
      else
        wets_bed_result = false
      end
      break
    else
      puts 'Please answer "yes" or "no".'
    end
  end
  return wets_bed_result
end
```

3. Class

클래스와 오브젝트

- 루비에서 데이터의 기본 단위를 “오브젝트(객체)”라고 부름.
- 변수, 배열, 모든 것이 오브젝트.
- 루비에서 주로 사용되는 오브젝트의 종류.

오브젝트	오브젝트가 하는 일	클래스 이름
수치 오브젝트	수치를 표현	Numeric, Integer, Float
문자열 오브젝트	문자열을 표현	String
정규 표현 오브젝트	문자열 매칭을 위한 패턴 표현	Regexp
시간 오브젝트	시간을 표현	Time
파일 오브젝트	파일 읽기/쓰기에 사용	File
해시 오브젝트	해시를 표현	Hash
배열 오브젝트	배열을 표현	Array

클래스

- 오브젝트의 성질이나 기능을 나타내는 일종의 설계도.
- 클래스로 부터 오브젝트를 생성해 낼 수 있음.
- Time 클래스는 시간이라는 데이터를 표현하기 위해 만들어 짐.
- Time이라는 클래스로 부터 현재 시간, 특정 시간의 오브젝트를 생성할 수 있음.
- `now = Time.new` # 현재의 시간을 표현하는 now 라는 오브젝트 생성

Object Oriented Programming (OOP)

- 객체지향 프로그래밍 (OOP)
  - 객체지향 프로그래밍은 여러 단위의 객체 간의 협업을 통해 프로그램이 수행하고자 하는 목적을 이루는 방법.
  - 각각의 클래스는 서로 다른 역할을 담당하고 있고, 여러 클래스로 만들어진 객체(object)가 모여서 프로그램을 완성.
  - 각각의 객체는 서로 메시지를 주고 받음.
  - 예: 자동차
    - 엔진 객체: 엑셀로 부터 메시지를 받아 작동, 바퀴에 메시지 전달
    - 바퀴 객체: 엔진으로 부터 메시지 받아 회전, 브레이크에서 메시지 받으면 회전을 멈춤. 운전대로부터 좌/우 이동 메시지 받음.
    - 운전대 객체: 좌/우로 움직이며 바퀴에게 메시지 전달.
    - 페달 객체 → 악셀 객체 / 브레이크 객체: 밟았다가 뗄 수 있으며 메시지를 바퀴와 엔진에 전달.

Object Oriented Programming (OOP)

- 클래스는 오브젝트를 생성하기 위한 청사진.
  - 오브젝트의 상태를 저장하는데 사용될 속성을 정의
    - 운전대 클래스: 좌/우로 움직임. 버튼이 하나 있고 누르면 소리가 남.
      → 차종에 관계없이 비슷한 기능을 제공, 그러나 차종에 따라 추가되는 속성이 있을 수 있음 (예: 오디오 콘트롤 버튼)
    - 엔진 클래스: 회전을 하여 동력을 바퀴로 전달. 멈춤 속성과 시동 속성이 있음.
  - 오브젝트가 이해할 수 있는 메시지와 메시지에 응답하는 과정을 정의 (메소드의 역할)
    - 엔진 클래스: 엑셀 오브젝트로 부터 메시지가 전달되면 엔진 시동 메소드를 실행하여 엔진을 움직임. 브레이크 오브젝트로부터 메시지가 전달되면 엔진 멈춤 메소드를 실행.
  - 이 과정에서 클래스가 내부적으로 어떻게 움직이는지 클래스 외부에서 알 필요가 없음 → 예: String에서 sort 메소드.

클래스 만들기

- 클래스를 만들기 위해서는 class 라는 키워드를 사용하며 다음과 같은 구조를 따른다.

```
class Die
  def roll
    1 + rand(6)      ← class method
  end
end
```

- 클래스는 다음과 같이 오브젝트를 생성하여 사용한다.
  - `die1 = Die.new` # 주사위 하나 생성
  - `die1.roll` # 주사위의 메소드인 roll 호출

클래스의 인스턴스 변수

- 인스턴스 변수는 클래스의 변수. @로 시작.
- 클래스 내에서는 글로벌하게 사용 (어디서나 참조 가능)

Lab 6

```
class Die
  def roll
    @number = 1 + rand(6)
  end
  def showing
    @number
  end
end

puts Die.new.showing
```

클래스의 인스턴스 변수

- 인스턴스 변수는 클래스의 변수. @로 시작.
- 클래스 내에서는 글로벌하게 사용 (어디서나 참조 가능)

Lab 6

```
class Die
  def roll
    @number = 1 + rand(6)
  end
  def showing
    @number
  end
end

puts Die.new.showing
```

=> nil !!!

클래스의 인스턴스 변수

- 인스턴스 변수는 클래스의 변수. @로 시작.
- 클래스 내에서는 글로벌하게 사용 (어디서나 참조 가능)

Lab 6

```
class Die
  def roll
    @number = 1 + rand(6)
  end
  def showing
    @number
  end
end

puts Die.new.showing
```

=> nil !!!

이 경우, 반드시 다음과 같이 써야 함.

```
die = Die.new
die.roll
die.showing
```

클래스의 인스턴스 변수 초기화

- 오브젝트가 생성될 때, 인스턴스 변수를 초기화 해주어야.

Lab 6

```
class Die
  def initialize
    roll
  end
  def roll
    @number = 1 + rand(6)
  end
  def showing
    @number
  end
end
```

클래스의 인스턴스 변수 초기화

- 오브젝트가 생성될 때, 인스턴스 변수를 초기화 해주어야.

Lab 6

```
class Die
  def initialize
    roll
  end
  def roll
    @number = 1 + rand(6)
  end
  def showing
    @number
  end
end
```

die = Die.new 를 했을 때
initialize 메소드가 자동 호출되면서
roll 함수가 역시 호출된다.

Lab 7: Dragon 키우기

- 다마고치와 같은 Dragon 키우기 프로그램을 만들어 보자.
  - Dragon은 이름을 가진다.
  - 밥을 주면 배가 가득 찬다.
  - 소화관이 차면 산책을 가서 용변을 봐야 한다.
  - 재우면 3번 코를 골며 자다가 곧 일어난다.
  - 하늘높이 던지면(toss) 좋아하고, 안아서 흔들어주면(rock) 잠이 든다. 곧 잠에서 깨지만.
  - 시간이 지날 수록 (한번 뭔가 액션을 취할 수록) 배는 고파지고 소화관은 차오른다.
  - 배가 너무 고프면 게임은 끝. 소화관이 너무 차면 실례를... (Lab7.rb에 클래스 설계)

Lab 8: Dragon 게임을 보다 interactive 하게...

- Lab 7의 프로그램을 interactive 하게 바꾸어 보자.
  - Lab 5의 ask 메소드를 참고하여 다음과 같이 명령어의 보기를 주고 선택하게 하자.

  - Dragon의 이름을 입력해 주세요.
    G-Dragon
    'G-Dragon'이 태어났습니다.
  - commands: feed, toss, walk, rock, put to bed, exit
    feed
    'G-Dragon'에게 밥을 주고 있어요.
  - commands: feed, toss, walk, rock, put to bed, exit
    toss
    'G-Dragon'을 하늘 높이 던졌어요!
    즐거워 하네요

4. Hash

Hash Class

- Hash는 루비에서 가장 많이 사용되는 데이터 타입.
- Array와 마찬가지로 하나의 변수에 여러 데이터를 담을 수 있으나, array가 데이터를 단순히 순서대로 담는 반면

hash는 데이터를 key-value 쌍으로 담는다.

- 특정 데이터를 참조할 때, 데이터가 담긴 인덱스를 알지 못해도 해당 데이터로의 접근이 가능.
- ["name"] = "홍길동" (key-value)
- ["address"] = "관악구 관악로" (key-value)

Array vs. Hash

Lab 9.1

```
contact = Array.new
contact = ["홍길동", 24, "male", "관악구 관악로"]
puts contact[0]  => 홍길동
puts contact[2]  => male
```

Lab 9.2

```
contact = Hash.new
contact["name"] = "홍길동"
contact["age"] = 24
contact["sex"] = "male"
contact["address"] = "관악구 관악로"
puts contact["age"]  => 24
puts contact["address"]  => 관악구 관악로
```

Symbol 과 Hash

- 심볼(symbol)은 루비의 독특한 데이터 타입
- 기능적으로 문자열과 유사하다.

```
:name
  => :name
:name.to_s
  => name
"name".to_sym
  => :name
```

Symbol을 사용하는 이유

- 문자열은 한번 생성될 때마다 메모리를 차지.
  - ```
    hash["key"] = value1
    hash["key"] = value2
    hash["key"] = value3
    hash["key"] = value4
    hash["key"] = value5
    ...
    ```
  - 매번 key라는 문자열이 메모리에 저장됨.
  - ```
    hash[:key] = value1
    hash[:key] = value2
    hash[:key] = value3
    ...
    ```
  - 심볼을 사용하면 key 가 단 한번만 메모리에 저장.

Hash의 사용

- Hash 는 다음과 같이 생성하고 사용

Lab 10.1

```
person = Hash.new
person[:name] = "Hong Gil Dong"
person[:phone_number] = "555-1234"
```

```
person = {:name => "Hong Gil Dong", :phone_number => "555-1234"}
```

```
puts person[:name]
```

```
puts person[:phone_number]
```

Hash의 사용

Lab 10.2

- Hash 값의 변경
  - `person[:phone_number] = "010-555-1234"`
- Hash 값의 추가
  - `person[:sex] = "male"`

```
p person
=> {:name => "Hong Gil Dong", :phone_number
    => "010-555-1234", :sex => "male"}
```

Hash의 Iterator

Lab 10.3

- Array의 경우
  - contact = ["홍길동", 24, "male", "관악구 관악로"]

```
contact.each do | v |
  puts v
end
```

- Hash의 경우
  - contact = {:name => "홍길동", :sex => "male", :address => "관악구 관악로"}

```
contact.each do | k, v |
  puts "#{k}: #{v}"
end
```

5. File I/O

IO Class

- 지금까지의 프로그램은 모두 결과물을 화면에 출력하고 종료됨.
- 결과물을 파일로 출력하기 위해서는 File 클래스를 사용한다.
- 그동안 사용해온 gets, puts, print 등은 모두 IO 클래스에서 정의된 메소드
- IO 클래스는 그 밖에 read, write, readline 등의 메소드 제공.
- 파일의 저장과 열기를 수행할 때는 IO 클래스의 서브클래스인 File 클래스를 사용.

File의 save 와 open

- File 클래스는 파일의 열기와 저장을 지원.
- 파일 객체 생성
  - file = File.new("filename", "mode")
    - mode 에 따라 파일 객체의 용도가 결정됨.
      - `r : read-only`
      - `r+ : read-write`
      - `w : write-only`
      - `w+ : read-write`
- 파일 닫기
  - file.close

File 저장

- 파일 객체를 다음과 같이 만든다.
  - `file = File.new("test.txt", "w")`
- 파일 객체에 저장하고자 하는 객체를 전달한다.
  - `file << "hello, ruby"`
- 마지막으로 파일을 닫는다.
  - `file.close`

Lab 11: One Little Monkey 저장

- 다음의 문장을 저장하는 프로그램을 만드시오.

  - One little monkey jumping on the bed
  - He fell off and bumped his head
  - So Momma called the doctor and the doctor said
  - No more monkeys jumping on the bed!

  - Printed on 2013/10/12 at 10:45AM

File 읽기

- 파일 객체를 다음과 같이 만든다.
  - `file = File.new("test.txt", "r")`
- 파일 내용 전체를 읽어 스트링에 저장한다.
  - `a = file.read`
- 파일 내용을 한줄 씩 읽어 처리한다.

```
file.each do | line |
  puts line.upcase
end
```

- 마지막으로 파일을 닫는다.
  - `file.close`

Lab 12: 저장된 One Little Monkey 불러오기

- Lab 11에서 저장한 One Little Monkey 파일을 불러와서 모두 대문자 처리하여 출력하시오.

Lab 13: Jobs' Speech 분석

- jobs_speech.txt 에는 스티브잡스의 유명한 스탠포드 연설이 담겨 있다. 이 파일의 내용을 읽어 단어만 따로 한줄씩 추출한 후, 대문자로 변환하여 출력하시오.
- " , . 등의 부호 삭제할 것.
- 예:

```
I
AM
HONORED
TO
BE
...
```

힌트:

`gsub(',','')` → , 부호 삭제
`split()` → 단어 분리

Quiz 4

- Take Home Quiz
- 코드 첫 줄에 본인의 학번, 이름을 comment 로 입력
- quiz4_본인영문이름.rb로 저장하고 압축하여 etl 로 제출
- 배점: 3 (프로그램이 제대로 실행되면 2.5, 코딩 스타일 0.5)

Quiz 4: 단어의 빈도수 계산

- 스티브 잡스와 빌 게이츠의 연설 파일을 읽어 각 단어의 빈도수를 계산한 후 파일로 저장하시오.
  - 힌트
    - Hash 를 사용, Key를 단어로 지정, 빈도수가 같은 key를 만나면 value를 1씩 증가.
    - Hash 클래스의 함수인 sort_by를 사용하여 value를 기준으로 sort.
- THE: 91
  I: 86
  TO: 71
  AND: 49
  WAS: 48

Questions?