hci+d lab.

Week 04 • 소셜네트워 서비스 개발 및 분석

Flow Control and Array

Joonhwan Lee
human-computer interaction + design lab.

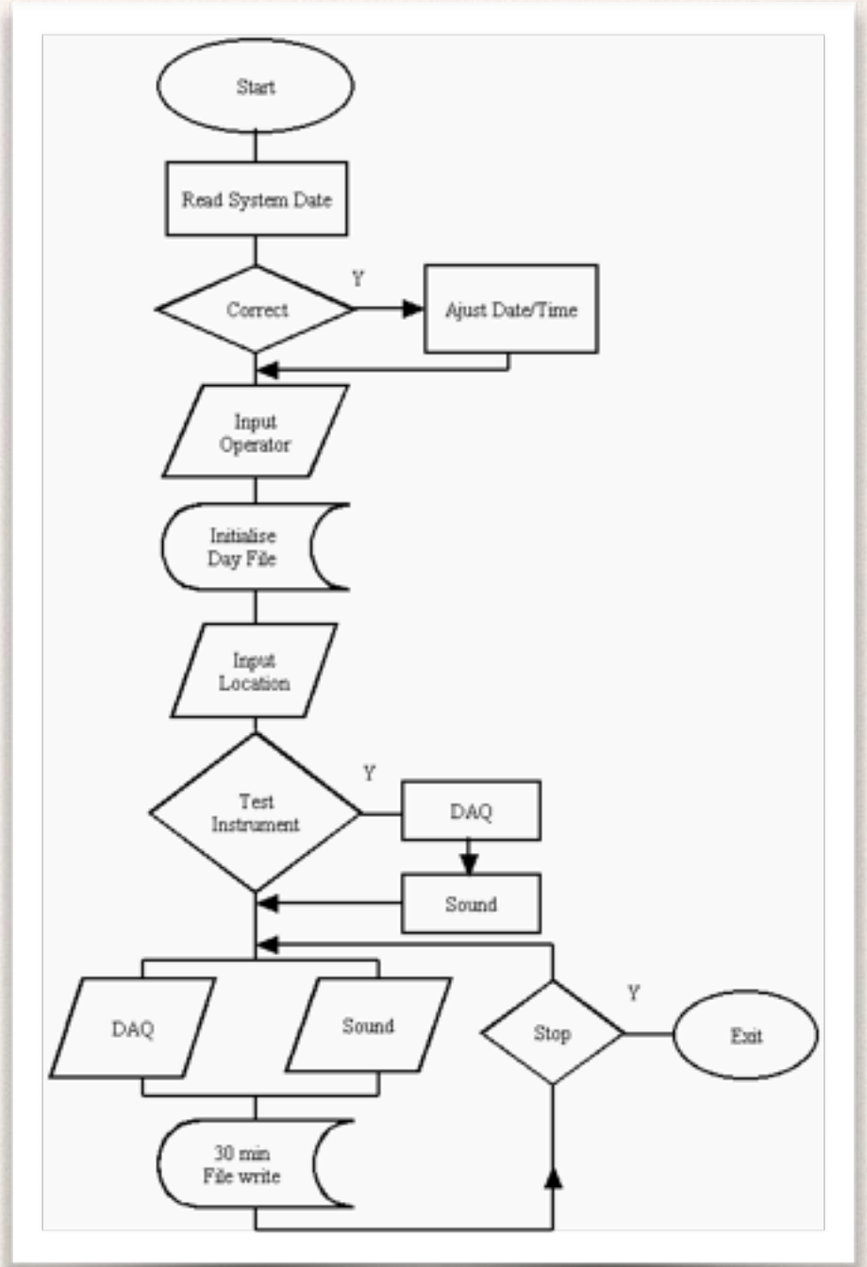
오늘 다룰 내용

- Flow Control
- Array
- Quiz 2
- Quiz 3

1. Flow Control

프로그램의 흐름(flow)

- 지금까지 다룬 프로그램
  - 모두 한줄씩 한번에 실행하고 종료되는 구조 - 단순한 흐름
  - 실제 프로그램은 보다 훨씬 복잡한 흐름을 가지고 있음
- flow 를 만들기 위해서는...
  - 만약에 ooo 라면..
  - ooo 하는 동안에..
  - 예:
    - A 가 B 보다 크면 C1을 하고 A 가 B 보다 작다면 C2를 한다



Boolean Expression

- ‘참/거짓’ 혹은 ‘예/아니오’로 판별되는 문장
  - 15 is greater than 20 => false
  - 5 equals 5 => true
  - 32 is less than or equal to 33 => true
- 논리 자료형으로 불리며 문장의 논리적 구조를 판별할 때 사용됨.

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

AND table

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

OR table

Boolean Comparison Operators

- Similar to Algebra
  - > greater than
  - < less than
  - >= greater than or equal to
  - <= less than or equal to
  - == equality (equal to)
    - Note: '=' is the 'assignment' operator: `x = 5;`
  - != inequality (not equal to)

Comparison Methods

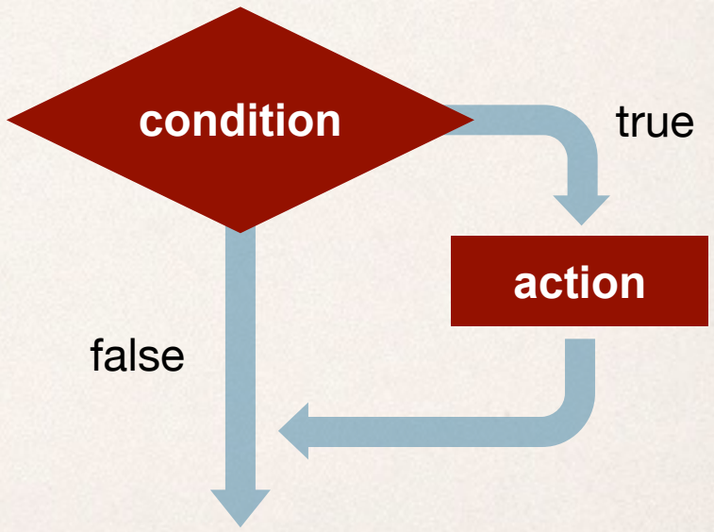
- 비교
  - puts 1 > 2 → false
  - puts 1 < 2 → true
  - puts 5 >= 5 → true
  - puts 5 <= 4 → false
  - puts 1 == 1 → true
  - puts 2 != 1 → true

  - puts 'cat' < 'dog' → true
  - puts 'cat' < 'Dog' → false

Branching: if statement

- 만약 주어진 조건문이 참(true)이라면 실행. 그렇지 않으면 (false) 실행하지 않음.
- **if** *boolean_expression*

```
    execute this line
end
```



Branching: if statement

Lab 1

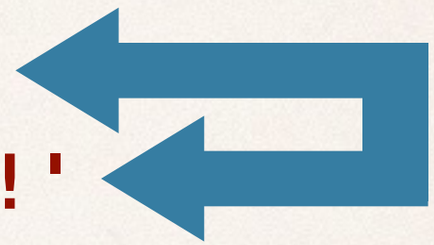
```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if name == 'ruby'
   puts 'What a lovely name!'
end

=>
Hello, what's your name?
ruby
Hello, ruby.
What a lovely name!
```

Branching: if statement

Lab 1

```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if name == 'ruby'
    puts 'What a lovely name!'
end
```




```
=>
Hello, what's your name?
ruby
Hello, ruby.
What a lovely name!
```

Branching: if statement

Lab 1

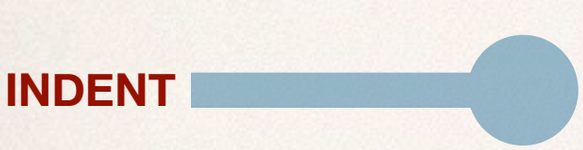
```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if name == 'ruby'
    puts 'What a lovely name!'
end
```

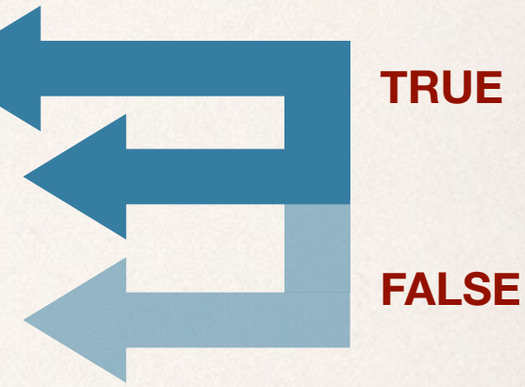

```
=>
Hello, what's your name?
ruby
Hello, ruby.
What a lovely name!
```

Branching: if statement

Lab 1

```
puts 'Hello, what\'s your name?'
name = gets.chomp
puts 'Hello, ' + name + '.'
if name == 'ruby'
  puts 'What a lovely name!'
end
```



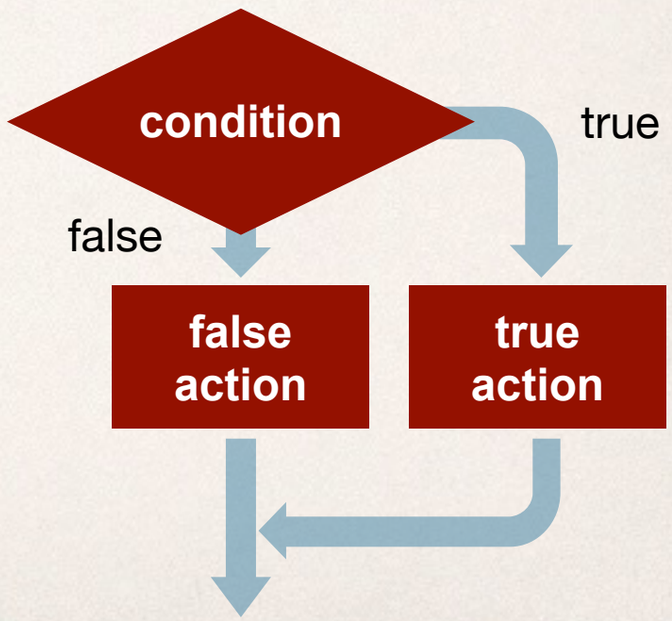



```
=>
Hello, what's your name?
ruby
Hello, ruby.
What a lovely name!
```

Branching: if else statement

- false 인 경우에 실행될 문장을 따로 정의하고 싶을 경우
- **if** *boolean_expression*

```
    execute this line #1
else
    execute this line #2
end
```



Branching: if else statement

Lab 2

```
puts 'I am a fortune-teller. Tell me your name:'
name = gets.chomp
if name == 'ruby'
  puts 'I see great things in your future.'
else
  puts 'Your future is... Oh my!  Look at the time!'
  puts 'I really have to go, sorry!'
end


=>
I am a fortune-teller.  Tell me your name:
java
Your future is... Oh my!  Look at the time!
I really have to go, sorry!
```

Branching: nested if else statement

Lab 3

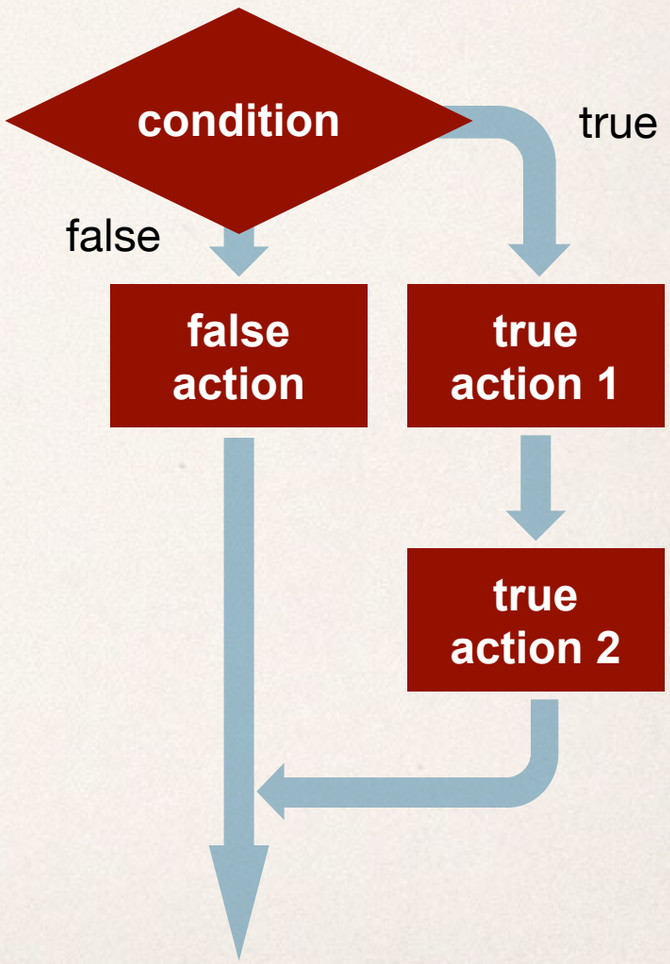
```
puts 'Hello, and welcome to 7th grade English.'
puts 'My name is Mrs. Gabbard. And your name is...?'
name = gets.chomp

if name == name.capitalize
  puts 'Please take a seat, ' + name + '.'
else
  puts name + '?  You mean ' + name.capitalize + ',
right?'
  puts 'Don\'t you even know how to spell your name??'
  reply = gets.chomp

  if reply.downcase == 'yes'
    puts 'Hmmph!  Well, sit down!'
  else
    puts 'GET OUT!!'
  end
end
```

Branching: if elsif statement

- 조건을 여러개 검증해야 할 때 사용
- **if** *boolean_expression*
  execute this line #1
  elsif *boolean_expression*
  execute this line #2
  elsif *boolean_expression*
  execute this line #3
  else *boolean_expression*
  execute this line #4
  end



Branching: if elsif statement

Lab 4

```
puts "Enter your password:"
password = gets.chomp

if password.length <= 4
  puts "Password should be longer than 4 letters."
elsif password.length > 8
  puts "Password should be shorter than 8
letters."
else
  puts "You have entered #{password}."
end
```

Branching: case statement

- if...elsif 보다 보기 쉬운 스위치 문을 만들때

```
case statement
when condition1
  do this line #1
when condition2
  do this line #1
when condition3
  do this line #1
else
  do this line #1
end
```

Branching: case statement

Lab 5

```
puts "What's your age?"
age = gets

case age.to_i
  when 0..2 then
    puts "baby"
  when 3..6 then
    puts "little child"
  when 7..12 then
    puts "child"
  when 12..18 then
    puts "youth"
  else
    puts "adult"
end
```

Logical Operators: AND

- 경우에 따라 조건문을 두개 이상 “모두” 만족시켜야 하는 경우가 있음. 이런 경우 Logical Operator AND를 사용함.
- A **AND** B: 조건 A와 B가 모두 만족하는 경우 → A **&&** B 또는 A **and** B 로 표현

A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

AND table

```
age = 21;
permit = 1;

if age >= 20
  if permit == 1
     puts "OK to Drive";
  else
     puts "Ride the bus";
  end
else
  puts "Ride the bus";
end
```

Nested ifs

```
age = 21;
permit= 1;

if age >= 20 and permit == 1
   puts "OK to Drive";
else
   puts "Ride the bus";
end
```

compound condition

Logical Operators: OR

- 만약 두개 이상의 조건문 중 하나만 만족해도 된다면, OR 를 사용
- A **OR** B: 조건 A와 B 중 하나만이라도 만족하는 경우 → A || B 또는 A or B 로 표현

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

OR table

```
age = 21;
permit = 1;

if age >= 20 or (age >= 20 and permit == 1)
   puts "OK to Drive";
else
   puts "Ride the bus";
end
```

Logical Operators: OR

- 만약 두개 이상의 조건문 중 하나만 만족해도 된다면, OR 를 사용
- A **OR** B: 조건 A와 B 중 하나만이라도 만족하는 경우 → A || B 또는 A or B 로 표현

A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1

OR table

```
age = 21;
permit = 1;

if age >= 20 or (age >= 20 and permit == 1)
   puts "OK to Drive";
else
   puts "Ride the bus";
end
```

괄호의 사용으로 AND 연산자와 OR 연산자를 섞어서 쓸 수 있다.
순서는 ( )가 먼저.

Lab 6: 성적 처리 프로그램

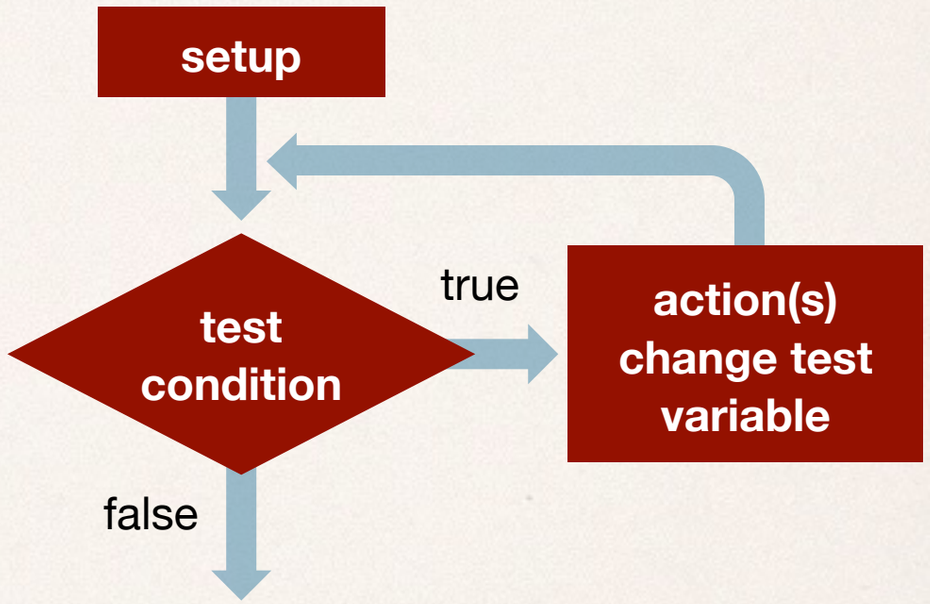
- if 문을 사용해서 다음과 같은 성적처리 프로그램을 만들어 보자.
  - 힌트1: logical operator 를 이용해 조건을 연결
  - 힌트2: get 으로 성적 입력
  - A: 90-100
  - B: 80-89.9999
  - C: 70-79.9999
  - D: 60-69.9999
  - F: below 60

Looping

- 비슷한 일을 반복해서 하기 위해 iteration 이 필요함.
- 각각의 iteration은 ‘아마도’ 조금씩 다른 일을 수행.
- Iteration은 반드시 종료할 수 있는 조건이 있어야 함.
  - 그렇지 않은 경우 무한 루프에 빠질 가능성
- 가장 많이 사용하는 loop는 while, for 등

Looping: while statement

- 모든 loop를 만드는 과정은 다음의 세가지
  - setup variables
  - test condition
  - change test variables



- loop 가 종료될 수 있는 조건을 만들어야 함
  - 종료 조건이 없으면 계속 반복 → 무한루프에 빠지게 됨

Looping: while statement

- while loop

```
while end_condition
   do this line #1
   do this line #2
end
```

```
i = 10                  ← setup variables
while i > 0 do          ← test condition
   puts i               ← action
   i = i - 1            ← change variable
end
```

2. Array (배열)

Array

- 배열은 여러 개의 데이터를 순서대로 보관하기 위한 데이터 타입
- 배열의 생성은 두가지 방법이 있다.
  - `arr1 = Array.new` (Array 클래스를 사용)
  - `arr2 = [1, 2, 3]` (Array 리터럴을 사용)

Array

- Array 클래스를 사용하는 방법

```
arr1 = Array.new
arr1.push(1)
arr1.push("hello")
arr1.push(123.45)
arr1 << 320
arr1 << 'new world'
arr1
=> [1, "hello", 123.45, 320, "new
world"]
```

Array

- 배열 리터럴을 사용하는 방법

  ```
  arr2 = [1, "help", 123.45]
  ```

- 배열 원소의 접근: 배열이 생성되면 숫자 인덱스를 통해 배열의 원소(해당내용)에 접근할 수 있다 (인덱스는 0부터 시작)

  ```
  arr1[0]
  arr2[2]
  ```

- 배열 원소의 치환

  ```
  arr1[0] = 4
  arr2[2] = "hello"
  ```

Array

- 비어있는 인덱스에 객체를 지정 → 빈 공간은 모두 nil 로 채워진다

```
arr2 = [1, "help", 123.45]
arr2[5] = "UFO"
arr2
=> [1, "help", 123.45, nil, nil,
"UFO"]
```

- 배열 크기: size 메소드를 사용해서 배열의 크기를 알 수 있다.

```
arr2.size
```

Array Methods

- 배열이 특정한 객체를 포함하고 있는지를 확인할 때:
  include? 메소드 사용

```
arr2.include? "UFO"   => true
arr2.include? "java"  => false
```

- 배열의 첫번째와 마지막 원소에 접근

```
arr2.first
arr2.last
```

- 배열에 원소를 추가

```
arr2.push("java")
arr2 << "java"
arr2.insert(2, "java") # 특정인덱스에 원소 추가
```

Array Methods

- 배열 원소의 삭제

```
arr2.pop #마지막 원소 삭제
arr2.delete("java")
arr2.delete_at(2)
```

- 배열의 정렬

```
cities = ["seoul", "tokyo",
"pittsburgh", "paris", "new york",
"london", "hong kong"]
cities.sort
numbers = [2, 4, 234.2, -129.34, 23,
31.983, 0]
numbers.sort
numbers.max  or  numbers.min
```

Array Methods

- 배열 원소의 join
- 배열 내의 원소는 join 명령어를 통해 하나의 문자열로 결합할 수 있다.

```
cities = ["seoul", "tokyo", "pittsburgh",
"paris", "new york", "london", "hong kong"]
cities.join
=> "seoultokyopittsburghparisnew
yorklondonhong kong"
cities.join(", ") #결합할 때 사이에 넣는 캐릭터 지정 가능
=> "seoul, tokyo, pittsburgh, paris, new york,
london, hong kong"
```

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

Ranges in Ruby

- `(1..10) : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10`
- `('a'..'f') : 'a', 'b', 'c', 'd', 'e', 'f'`
- `(1..10).to_a`
  `=> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `('a'..'f').to_a`
  `=> ["a", "b", "c", "d", "e", "f"]`

String is Array

- `str = "hello, ruby"`
  `=> "hello, ruby"`
- `str.length`
  `=> 11`
- `str[0]`
  `=> "h"`
- `str[3..9]`
  `=> "lo, rub"`

Quiz 2

- 시간: 30분
- 코드 첫 줄에 본인의 학번, 이름을 comment 로 입력
- quiz2.1_본인영문이름.rb, quiz2.2_본인영문이름.rb 로 저장하고 압축하여 etl 로 제출
- 배점: 각 1.5

Quiz 2.1: 100 little monkeys

- 다음을 출력하는 프로그램을 만드시오.

  100 little monkeys jumping on the bed.
  One fell off and broke his head.
  Mama(or Daddy) called the doctor and the doctor said,
  "No more monkeys jumping on the bed!"
  ...
  1 little monkey jumping on the bed.
  One fell off and broke his head.
  Mama(or Daddy) called the doctor and the doctor said,
  "No more monkeys jumping on the bed!"

- 힌트: rand(range)는 랜돔 넘버를 발생해 주는 함수이다. rand()를 사용하여 Mama 혹은 Daddy 를 선택하게 하자.
  - 예: rand(5) → 0 부터 5까지의 수를 랜돔하게 발생.

Quiz 2.2: 윤년의 계산

- 시작하는 연도와 끝나는 연도를 물어보는 프로그램을 작성하고, 그 사이에 있는 모든 윤년을 출력하라.
  (시작하는 연도와 끝나는 연도가 윤년이면 그 해도 출력)
  - 윤년은 4로 나누어짐 (예: 1984, 2012)
  - 100으로 나누어지면 윤년이 아님 (예: 1800, 1900)
  - 윤년은 400으로 나누어짐 (예: 1600, 2000)

  > Pick a starting year (like 1973 or something):
  => 1973
  > Now pick an ending year:
  => 1977
  > Check it out... these years are leap years:
  => 1976

Quiz 3

- Take Home Quiz
- 코드 첫 줄에 본인의 학번, 이름을 comment 로 입력
- quiz3_본인영문이름.rb로 저장하고 압축하여 etl 로 제출
- 배점: 3 (프로그램이 제대로 실행되면 2.5, 코딩 스타일 0.5)
- 가산점: 2 (wikipedia의 rule을 모두 해결했을 경우)

Quiz 3: Pig Latin

- Language Game.
- 몇가지 간단한 룰에 의해 철자의 순서를 바꾸는 일종의 암호화 방법.
  - e.g., pig → igpay, banana → ananabay, trash → ashtray
- http://en.wikipedia.org/wiki/Pig_Latin

Quiz 3: Pig Latin

- Rules
  - 모음으로 시작되는 단어는 끝에 ay를 붙인다.
    - egg → egg**ay**
    - older → older**ay**
  - 자음으로 시작되는 단어는 첫번째 자음을 맨 뒤로 옮긴 후에 ay를 붙인다.
    - happy → appy**hay**
    - jello → ello**jay**
  - 첫번째와 두번째도 자음이면 두개의 자음을 맨 뒤로 옮긴 후에 ay를 붙인다.
    - thunder → under**thay**
    - school → hool**scay**

Quiz 3: Pig Latin

- Hints
  - `str = 'hello'`
  - `str[0]`
    `=> 'h'`
  - `str[3..4]`
    `=> 'lo'`
  - `arr = ['a', 'b', 'c']`
  - `arr.include?('a')`
    `=> true`

Questions?