정보교육을 위한 파이썬

정보 탐색

Version 0.0.9-d2

저자: Charles Severance
번역: 이광춘, 한정수
(xwmooc)

Printing history:

October 2013: Major revision to Chapters 13 and 14 to switch to JSON and use OAuth. Added new chapter on Visualization.

September 2013: Published book on Amazon CreateSpace

January 2010: Published book using the University of Michigan Espresso Book machine.

December 2009: Major revision to chapters 2-10 from *Think Python: How to Think Like a Computer Scientist* and writing chapters 1 and 11-15 to produce *Python for Informatics: Exploring Information*

June 2008: Major revision, changed title to *Think Python: How to Think Like a Computer Scientist*.

August 2007: Major revision, changed title to *How to Think Like a (Python) Programmer*.

April 2002: First edition of *How to Think Like a Computer Scientist*.

The LaTeX source for the *Think Python: How to Think Like a Computer Scientist* version of this book is available from `http://www.thinkpython.com`.

Chapter 1

조건부 실행

1.1 불 연산식(Boolean expressions)

불 연산식(boolean expression)은 참(True) 혹은 거짓(False)를 가진 연산 표현식이다. 다음 예제는 == 연산자를 사용하는데 두개의 피연산자를 비교하여 값이 동일하면 참(True), 그렇지 않으면 거짓(False)을 출력한다.

```
>>> 5 == 5
True
>>> 5 == 6
False
```

참(True)과 거짓(False)은 불(bool) 형식에 속하는 특별한 값들이고, 문자열은 아니다.

```
>>> type(True)
<type 'bool'>
>>> type(False)
<type 'bool'>
```

==연산자는 **비교(comparison operators)** 연산자 중의 하나이고, 다른 연산자는 다음과 같다.

```
x != y               # x는 y와 값이 같지 않다.
x > y                # x는 y보다 크다.
x < y                # x는 y보다 작다.
x >= y               # x는 y보다 크거나 같다.
x <= y               # x는 y보다 작거나 같다.
x is y               # x는 y와 같다.
x is not y           # x는 y와 개체가 동일하지 않다.
```

여러분에게 이들 연산자가 친숙할지 모르지만, 파이썬 기호는 수학 기호와는 다르다. 일반적인 오류에는 비교의 같다의 의미로 == 연산자 대신에 =를 사용하는 것이다. = 연산자는 할당 연산자이고, ==연산자는 비교 연산자이다. =<, => 비교 연산자는 파이썬에는 없다.

1.2 논리 연산자

3개의 **논리 연산자(logical operators)**: `and`, `or`, `not`가 있다. 논리 연산자 의미는 영어 의미와 유사하다. 예를 들어,

```
x > 0 and x < 10
```

`x`이 0 보다 크다. *그리고(and)*, 10 보다 작으면 참이다.

`n%2 == 0 or n%3 == 0`은 두 조건문 중의 하나만 참이되면, 즉, 숫자가 2 *혹은 (or)* 3으로 나누어진다면 참이다.

마지막으로 `not` 연산자는 불 연산 표현을 부정한다. `x > y`이 거짓이면, `not (x > y)`은 참이다. 즉, `x`이 `y` 보다 작거나 같으면 참이다.

엄밀히 말해서, 논리 연산자의 두 피연산자는 모두 불 연산 표현이여야 하지만, 파이썬은 그렇게 엄격하지는 않는다. 어떤 0이 아닌 숫자 모두 "true"로 해석된다.

```
>>> 17 and True
True
```

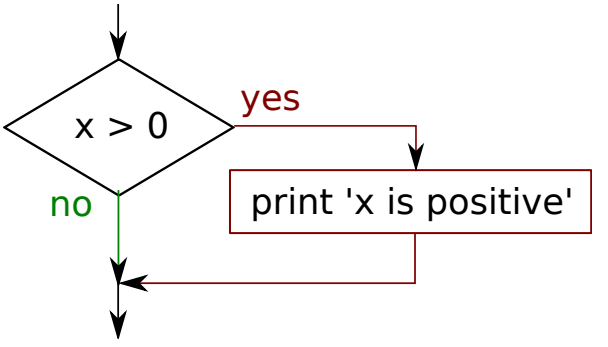
이러한 유연함이 유용할 수 있으나, 혼란을 줄 수도 있으니 유의해서 사용해야 됩니다. 무슨 일을 하고 있는지 정확하게 알지 못한다면 피하는게 좋습니다.

1.3 조건문 실행

유용한 프로그램을 작성하기 위해서는 조건을 확인하고 조건에 따라 프로그램의 실행을 바꿀 수 있어야 한다. **조건문(Conditional statements)**은 그럴 수 있는 능력을 부여한다. 가장 간단한 형태는 `if` 문이다.

```
if x > 0 :
    print 'x is positive'
```

`if`문 뒤에 불 연산 표현문을 **조건(condition)**이라고 한다.



만약 조건문이 참이면, 들여쓰기 된 스테이트먼트가 실행된다. 만약 조건문이 거짓이면, 들여쓰기 된 스테이트먼트의 실행을 생략한다.

if문은 함수 정의나 for 반복문과 동일한 구조를 가진다. if문은 콜론(:)으로 끝나는 헤더 머리부문과 들여쓰기 블록으로 구성된다. if문과 같은 구문을 한 줄 이상에 걸쳐 작성되기 때문에 **복합문(compound statements)**이라고 한다.

if문 본문에 실행 명령문의 제한은 없으나 최소한 한 줄은 있어야 한다. 때때로, 본문에 하나의 실행명령문이 없는 경우가 있다. 아직 코드를 작성하지 않고 자리를 잡아 놓는 경우로, 아무것도 수행하지 않는 pass문을 사용할 수 있다.

```
if x < 0 :
    pass          # 음수값을 처리 예정!
```

if문을 파이썬 인터프리터에서 타이핑하고 엔터를 치게 되면 명령 프롬프트가 갈매기 세마리에서 점 세개로 바뀌어 본문을 작성중에 있다는 것을 다음과 같이 보여줍니다.

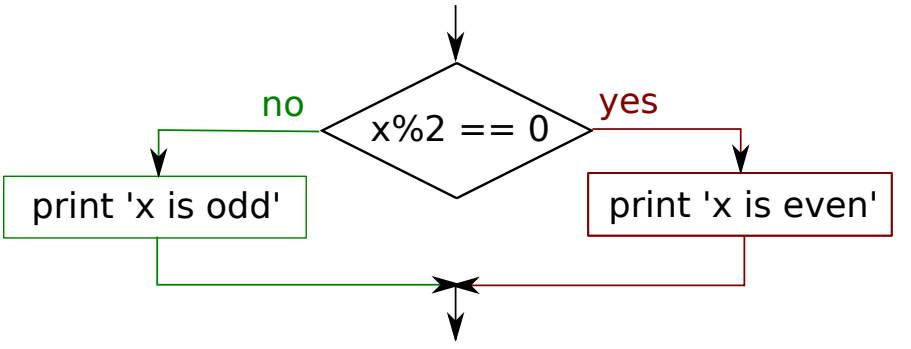
```
>>> x = 3
>>> if x < 10:
...     print 'Small'
...
Small
>>>
```

1.4 대안 실행

if문의 두 번째 형태는 **대안 실행(alternative execution)**으로 두 가지 경우의 수가 존재하고, 조건이 어느 방향인지를 결정한다. 구문(Syntax)은 아래와 같다.

```
if x%2 == 0 :
    print 'x is even'
else :
    print 'x is odd'
```

x가 2로 나누었을 때, 0이되면, x는 짝수이고, 프로그램은 짝수 결과 메시지를 출력한다. 만약 조건이 거짓이라면, 두 번째 명령문 블록이 실행된다.



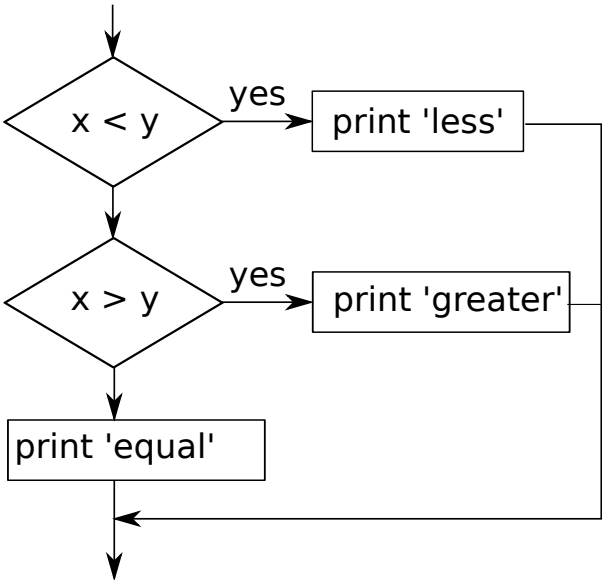
조건은 참 혹은 거짓이서, 대안 중 하나만 정확하게 실행될 것이다. 대안을 **분기(Branch)**라고도 하는데 이유는 실행 흐름의 분기가 되기 때문이다.

1.5 연쇄 조건문

때때로, 두 가지 이상의 경우의 수가 있으며, 두 가지 이상의 분기가 필요하다. 이 같은 연산을 표현하는 방법이 **연쇄 조건문(chained conditional)**이다.

```
if x < y:
    print 'x is less than y'
elif x > y:
    print 'x is greater than y'
else:
    print 'x and y are equal'
```

elif는 "else if"의 축약어이다. 이번에도 단 한번의 분기만 실행된다.



elif문의 갯수에 제한은 없다. else문이 있다면, 거기서 끝마쳐야 하지만, 연쇄 조건문에 필히 있어야 하는 것은 아니다.

```
if choice == 'a':
    print 'Bad guess'
elif choice == 'b':
    print 'Good guess'
elif choice == 'c':
    print 'Close, but not correct'
```

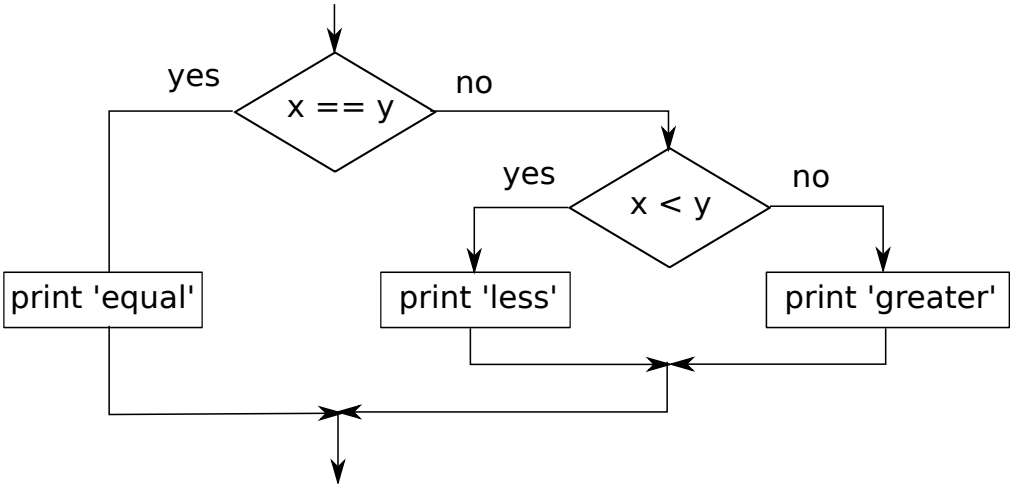
각 조건은 순서대로 점검한다. 만약 첫 번째가 거짓이면, 다음을 점검하고 계속 점검해 나간다. 순서대로 진행 주에 하나의 조건이 참이면, 해당 분기가 수행되고, if문 전체는 종료된다. 설사 하나 이상의 조건이 참이라고 하더라도, 첫 번째 참 분기만 수행된다.

1.6 중첩 조건문

하나의 조건문이 조건문 내부에 중첨될 수도 있다. 다음처럼 삼분 예제를 작성할 수 있다.

```
if x == y:
    print 'x and y are equal'
else:
    if x < y:
        print 'x is less than y'
    else:
        print 'x is greater than y'
```

바깥 조건문에 두 개의 분기가 있다. 첫 분기는 간단한 명령 실행문을 담고 있다. 두 번째 분기는 자체가 두 개의 분기를 가지고 있는 또 다른 `if`문을 담고 있다. 자체로 둘다 조건문이지만, 두 분기 모두 간단한 실행 명령문이다.



명령문 블록을 들여쓰는 것이 구조를 명확히 하지만, 중첩 조건문의 경우 가독성이 급격히 저하된다. 일반적으로, 가능하면 중첩 조건문을 피하는 것을 권장한다.

논리 연산자를 사용하여 중첩 조건문을 간략히 할 수 있다. 예를 들어, 단일 조건문으로 가지고 앞의 코드를 다시 작성할 수 있다.

```
if 0 < x:
    if x < 10:
        print 'x is a positive single-digit number.'
```

`print`문은 두개의 조건문이 통과될 때만 실행돼서, `and` 연산자와 동일한 효과를 거둘 수 있다.

```
if 0 < x and x < 10:
    print 'x is a positive single-digit number.'
```

1.7 try와 catch를 활용한 예외 처리

앞에서 사용자가 타이핑한 것을 읽어 정수로 파싱하기 위해서 함수 `raw_input`와 `int`을 사용한 프로그램 코드를 살펴 보았다. 또한 이렇게 하는 코딩하는 것이 얼마나 위험한 것인지도 살펴보았다.

```
>>> speed = raw_input(prompt)
What...is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int()
>>>
```

파이썬 인터프리터에서 상기 명령문을 실행할 때, "이런" 잠시 있다가 다음 명령 실행문으로 넘어가는 새로운 명령 프롬프트를 보게 된다.

하지만, 코드가 파이썬 스크립트로 실행이 되면 오류가 발생하고 즉시, 그 지점에서 멈추게 된다. 다음의 명령 실행문은 실행하지 않는다.

화씨 온도를 섭씨 온도로 변환하는 간단한 프로그램이 있다.

```
inp = raw_input('Enter Fahrenheit Temperature:')
fahr = float(inp)
cel = (fahr - 32.0) * 5.0 / 9.0
print cel
```

이 코드를 실행하고 적절하지 않은 입력값을 타이핑하게되면, 다소 불친절한 오류 메시지와 함께 작동을 멈춘다.

```
python fahren.py
Enter Fahrenheit Temperature:72
22.2222222222

python fahren.py
Enter Fahrenheit Temperature:fred
Traceback (most recent call last):
  File "fahren.py", line 2, in <module>
    fahr = float(inp)
ValueError: invalid literal for float(): fred
```

이런 종류의 예측되거나, 예측 못한 오류를 다루는 "try / except"로 불리는 조건 실행 구조가 파이썬에 내장되어 있다. `try`와 `except`의 기본 사항은 일부 명령문이 문제를 가 있다는 것을 사전에 알고 있고, 만약 그 때문에 오류가 발생하게 된다면 대신 실행할 명령문을 프로그램에 추가하는 것이다. `except` 블록의 명령문은 오류가 없다면 실행되지 않는다.

파이썬의 `try`, `except` 기능을 프로그램 코드의 실행에 보험을 든다고 생각할 수 있다.

온도 변환기 프로그램을 다음과 같이 다시 작성할 수 있다.

```
inp = raw_input('Enter Fahrenheit Temperature:')
try:
    fahr = float(inp)
    cel = (fahr - 32.0) * 5.0 / 9.0
    print cel
except:
    print 'Please enter a number'
```

파이썬은 `try` 블록의 명령문을 우선 실행합니다. 만약 모든 것이 순조롭다면, `except` 블록을 건너뛰고, 다음 코드를 실행합니다.

except이 try 블록에서 발생하면, 파이썬은 try블록을 건너뛰고 except블록의 명령문을 수행합니다.

```
python fahren2.py
Enter Fahrenheit Temperature:72
22.2222222222

python fahren2.py
Enter Fahrenheit Temperature:fred
Please enter a number
```

try문으로 예외사항을 다루는 것을 예외 처리한다**catching an exception**고 부릅니다. 예제에서는 except 절에서는 단순히 오류 메시지를 출력만 합니다. 대체로, 예외 처리는 오류를 고치거나, 다시 시작하거나, 최소한 프로그램이 정상적으로 종료될 수 있게 합니다.

1.8 논리 연산식의 단락(Short circuit) 평가

파이썬이 x >= 2 and (x/y) > 2와 같은 논리 연산식을 처리할 때, 왼쪽에서부터 오른쪽으로 연산식을 평가한다. and의 정의 때문에 x이 2보다 작다면, x >= 2은 거짓(False)이 되서, 전체는 (x/y) > 2이 참(True) 혹은 거짓(False)에 관계없이 거짓(False)이 된다. 파이썬이 논리 연산식의 나머지 부분을 평가해도 나아지는 것이 없다고 탐지할 때, 평가를 멈추고 나머지 논리 연산식에 대한 연산도 중지한다. 최종 논리 연산식의 값이 이미 알려졌기 때문에 더 이상의 연산을 멈출 때, 이를 단락(Short-circuiting) 평가라고 한다.

이것이 세심해 보일 수 있지만, 단락 행동이 가디언 **패턴(guardian pattern)**으로 불리는 좀 더 똑똑한 기술로 연결된다. 파이썬 인터프리터의 다음 코드를 살펴보자.

```
>>> x = 6
>>> y = 2
>>> x >= 2 and (x/y) > 2
True
>>> x = 1
>>> y = 0
>>> x >= 2 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and (x/y) > 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>>
```

파이썬이 (x/y) 연산을 평가할 때 y이 0 이어서 실행오류 발생시켜서, 세번째 연산은 수행되지 않습니다. 하지만, 두 번째 예제의 경우 x >= 2 이 거짓(False) 이어서 전체가 거짓(False)이 되어 (x/y) 평가가 실행되지 않고, 단락(Short-circuiting) 평가 규칙에 의해서 오류도 발생하지 않습니다.

오류가 발생할 것 같은 평가식 앞에 **가디언(gardian)** 평가식을 전략적으로 배치해서 논리 평가식을 아래와 같이 구성합니다.

```
>>> x = 1
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x = 6
>>> y = 0
>>> x >= 2 and y != 0 and (x/y) > 2
False
>>> x >= 2 and (x/y) > 2 and y != 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>>
```

첫 번째 논리 표현식은 `x >= 2`이 거짓(`False`)이라 `and`에서 멈춥니다. 두 번째 논리 표현식은 `x >= 2`이 참(`True`), `y != 0`은 거짓(`False`)이어서 `(x/y)`까지 갈 필요는 없습니다. 세 번째 논리 표현식은 `(x/y)` 연산이 끝난 후에 `y != 0`이 수행되어서 오류가 발생합니다.

두 번째 논리 표현식에서 `y != 0`이 '0'이 아니어만 `(x/y)`이 실행될 수 있도록 **가디언(gardian)** 역할을 수행한다고 할 수 있다.

1.9 디버깅(Debugging)

오류가 발생했을 때, 파이썬이 화면에 출력하는 트레이스백(traceback)은 상당한 정보를 담고 있지만, 특히 스택에 많은 프레임이 있는 경우 엄청나게 보일 수도 있다. 대체로 유용한 정보는 다음과 같다.

- 어떤 종류의 오류인가.
- 어디서 발생했는가.

구문 오류는 대체로 발견하기 쉽지만, 몇 가지는 애매합니다. 공백 오류가 대표적인데, 공백(space)과 탭(tab)은 구별이 되지 않고, 통상 무시하고 넘어가기 쉽기 때문입니다.

```
>>> x = 5
>>>  y = 6
  File "<stdin>", line 1
    y = 6
    ^
SyntaxError: invalid syntax
```

이 예제의 문제는 두 번째 줄이 한 칸 공백으로 들여써서 발생하는 것입니다. 하지만, `y`에 오류 메시지가 있는데 프로그래머를 잘못된 곳으로 인도합니다. 대체로 오류 메시지는 문제가 어디에서 발견되었는지를 지칭하지만, 실제 오류는 코드 앞에서 종종 선행하는 줄에 있을 수 있습니다.

동일한 문제가 실행 오류에도 있습니다. 데시벨(decibels)로 신호 대비 잡음비를 계산한다고 가정합시다. 공식은 $SNR_{db} = 10\log_{10}(P_{signal}/P_{noise})$입니다. 파이썬에서 아래와 같이 작성할 수 있습니다.

```
import math
signal_power = 9
noise_power = 10
ratio = signal_power / noise_power
decibels = 10 * math.log10(ratio)
print decibels
```

하지만, 실행하게 되면, 다음과 같은 오류 메시지[1]가 발생합니다.

```
Traceback (most recent call last):
  File "snr.py", line 5, in ?
    decibels = 10 * math.log10(ratio)
OverflowError: math range error
```

오류 메지지가 5번째 줄에 있지만, 잘못된 것은 없습니다. 실제 오류를 발견하기 위해서, 출력값은 0인 `ratio`의 값을 `print`문을 사용해서 출력해 보는 것이 도움이 됩니다. 문제는 4번째 줄에 있는데, 두 정수를 나누기를 소수점 연산을 했기 때문입니다. `signal_power` 와 `noise_power` 를 부동 소수점값으로 표현하는게 해결책입니다.

대체로, 오류 메시지는 어디에 문제가 발견되었는지를 말해주지만, 종종 어디서 원인이 발생했는지는 말해주지 않습니다.

1.10 용어 정의

몸통부문(body): 복합문 내부에 열련의 명령문 실행을 기술한 부문

불 표현식(boolean expression): 참(`True`) 혹은 거짓(`False`)의 값을 가지는 표현식

분기(branch): 조건문에서 대안 실행 명령문의 한 흐름

연쇄 조건문(chained conditional): 일련의 대안 분기가 있는 조건문

비교 연산자(comparison operator): 피연산자를 ==, !=, >, <, >=, <=로 비교하는 연산자

조건문(conditional statement): 조건에 따라 명령의 흐름을 조절하는 명령문

조건(condition): 조건문에서 어느 분기를 실행할지 결정하는 불 표현식

복합문(compound statement): 머리부문(head)와 몸통부문(body)으로 구성된 스테이트먼트. 머리부문은 콜론(:)으로 끝나며, 몸통부문은 머리부문을 기준으로 들여쓰기로 구별된다.

[1]파이썬 3.0에서는 오류 메시지가 발생하지 않습니다. 정수 피연산자인 경우에도 나눗셈 연산자가 부동 소수점 나눗셈을 수행합니다.

가디언 패턴(guardian pattern): 단락(short circuit) 행동을 잘 이용하도록 논리 표현식을 구성하는 것

논리 연산자(logical operator): 불 표현식을 결합하는 연산자 중의 하나 (`and`, `or`, `not`)

중첩 조건문(nested conditional): 하나의 조건문이 다른 조건문 분기에 나타나는 조건문.

트레이스백(traceback): 예외 사항이 발생했을 때 실행되고, 출력되는 함수 리스트

단락(short circuit): 왜냐하면 파이썬이 나머지 조건 표현식 평가를 할 필요없이 최종 결과를 알기 때문에, 파이썬이 논리 표현식 평가를 일부 진행하고, 더 이상의 평가를 멈출 때.

1.11 연습문제

Exercise 1.1 40시간 이상 일할 경우 시급을 1.5배 더 종업원에게 지급하는 봉급계산 프로그램을 다시 작성하세요.

```
Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

Exercise 1.2 `try`, `except`를 사용하여 봉급계산 프로그램을 다시 작성하세요. 숫자가 아닌 입력값을 잘 처리해서 메시지를 출력하고 프로그램을 종료하게 됩니다. 다음은 프로그램의 출력 결과를 보여줍니다.

```
Enter Hours: 20
Enter Rate: nine
Error, please enter numeric input

Enter Hours: forty
Error, please enter numeric input
```

Exercise 1.3 0.0과 1.0 사이의 점수를 출력하는 프로그램을 작성하세요. 만약 점수가 범위 밖이면 오류를 출력합니다. 만약 점수가 0.0과 1.0 사이라면, 다음의 테이블에 따라 등급을 출력합니다.

```
Score   Grade
>= 0.9     A
>= 0.8     B
>= 0.7     C
>= 0.6     D
< 0.6    F

Enter score: 0.95
```

```
A

Enter score: perfect
Bad score

Enter score: 10.0
Bad score

Enter score: 0.75
C

Enter score: 0.5
F
```

입력값으로 다양한 다른 값을 출력하도록 반복적으로 보이는 것처럼 프로그램을 실행하세요.