3
함수 제작 및 활용 - 연습
Function Abstraction & Application - More Practice

실습 문제에 쓸 지식

문자열 메소드 : partition(〈분리문자열〉)

partition 메소드는 〈분리문자열〉을 중심으로 문자열을 세 조각으로 나눈다. 예를 들어, 대상 문자열이 "3.14159"이고, 〈분리문자열〉이 "."인 경우, "3.14159".partition(".")를 실행하면 ("3", ".", "14159")와 같이 〈분리문자열〉 를 중심으로 3조각으로 나눈 뭉치를 내준다. 이 조각 뭉치를 **튜플(tuple)**이라고 하는데, 튜플도 문자열과 같은 요령으로 위치번호(index) 0, 1, 2를 사용하여 조각으로 분리할 수 있다. 다음을 실행하여 확인해보자.

```
pi = "3.14159".partition(".")
print(pi)
print(pi[0])
print(pi[1])
print(pi[2])
```

만약 분리문자열이 대상 문자열에 없으면, 첫째 조각에 대상 문자열 전체, 둘째와 셋째 조각에는 빈문자열을 들어있는 세조각 뭉치 튜플을 내준다. 예를 들어 분리문자열이 "."이고, 대상 문자열이 "365"이면, ("365", "", "")와 같이 3조각으로 나눈다. 다음도 실행하여 확인해보자.

```
year = "365".partition(".")
print(year)
print(year[0])
print(year[1])
print(year[2])
```

실습 문제 1. 정수 입력확인

다음 get_int 함수는 정수 입력을 받아서 확인 후 정수로 변환하여 내주는 함수이다.

```
def get_int(message) :
    s = input(message)
    while not s.isdigit() :
        s = input(message)
    return int(s)
```

이 함수를 다양한 입력을 가지고 호출하여 써보면서 이해하자.

그리고 나서, + 또는 - 부호를 앞에 붙인 정수도 허용하도록 개선한 get_int_signed 함수를 다음과 같이 작성하였다.

```
def get_int_signed(message) :
    s = input(message)
    while not (remove_sign(s).isdigit()):
        s = input(message)
    return int(s)
```

앞에서 공부한 방법과는 다르게 입력 문자열 맨 앞에 있는 부호를 떼어버린 다음, 남은 문자열이 숫자로만 구성되어 있는지 확인하는 방식으로 입력확인을 하도록 remove_sign 함수를 만들어보자.

실습 문제 2. 정수+실수 입력확인

다음 isfloat 함수는 문자열 인수 s가 소수점이 있는 실수 형태인지 확인하는 함수이다.

```
def isfloat(s) :
    (m,_,n) = s.partition(".")
    return (m.isdigit() and (n.isdigit() or n == "")) or m == "" and n.isdigit()
```

이 함수를 다양한 입력을 가지고 호출하여 써보면서 어떤 형태의 실수를 확인하는지 이해하자.

다음 get_float 함수는 정수 또는 소숫점이 있는 실수 형태로 입력을 받아서 확인 후 부동소수점수(float)로 변환하여 내주는 함수이다.

```
def get_float(message) :
    s = input(message)
    while not (s.isdigit() or isfloat(s)) :
        s = input(message)
    return float(s)
```

"+" 또는 "-" 부호를 앞에 붙인 정수 또는 고정소수점수도 허용하도록 get_fixed 함수를 개선한 get_fixed_signed 함수를 빈칸을 채워 완성하라.

```
def get_fixed_signed(message) :
    s = input(message)

    while
        s = input(message)

    return float(s)
```

- 귀띔 : 실습문제 1에서 만든 함수도 사용해도 좋다.

실습 문제 3. 안전한 제곱근 함수

표준입력창에서 정수 또는 실수를 입력받아 제곱근을 구하여 표준출력창에 프린트하는 프로시저 safe_sqrt() 를 작성하자. safe_sqrt() 프로시저를 호출하면 표준입출력창을 통하여 사용자와 대화식으로 프로그램이 아래와 같이 작동한다. 보통 폰트로 된 부분은 컴퓨터가 사용자에게 보여주는 부분이고, 파란색으로 볼드체로 진하게 보이는 부분이 사용자가 키보드로 입력한 부분이다.

```
제곱근을 구해드립니다.
0이상의 수를 입력하세요.
수를 입력하세요.
4
4 의 제곱근은 2.0 입니다.
계속하시겠습니까? (y/n) Y
계속하시겠습니까? (y/n) y
수를 입력하세요.
4.8
4.8 의 제곱근은 2.1909 입니다.
계속하시겠습니까? (y/n) 예
계속하시겠습니까? (y/n) y
수를 입력하세요.
-45
수를 입력하세요.
+38
수를 입력하세요.
twenty seven
수를 입력하세요.
0
0 의 제곱근은 0.0 입니다.
계속하시겠습니까? (y/n) n
안녕히 가세요.
```

사용자의 키보드 입력은 확인하여 0이상의 정수와 실수만 통과시키고 나머지는 재입력하도록 한다. 위의 사례를 보면 4, 4.8, 0은 제곱근 계산이 가능한 정수와 실수이므로 입력확인을 통과하여 제곱근 계산을 하여 결과를 보여준다. 그러나 -45, +38, twenty seven은 허용하는 입력이 아니므로 재입력을 요청한다. 한번 계산이 끝나면 계속할지를 물어본다. y를 입력하면 계속하고, n를 입력하면 프로그램을 종료한다.

코딩 가이드

- 제곱근 계산은 표준 라이브러리 math 모듈의 sqrt 함수를 사용한다.
- 계산한 제곱근은 실행사례에서 보여주는 대로 소수점 4째자리 미만은 반올림하여 표준출력창에 프린트한다.
- 강의 시간 및 실습 시간에 완성한 isfloat() 함수와 stop() 함수를 사용하여야 한다.

기 제작 함수

```
def isfloat(s) :
    (m,_,n) = s.partition(".")
    return m.isdigit() and (n.isdigit() or n == "") or \
           m == "" and n.isdigit()

def stop():
    cont = input('계속하시겠습니까? (y/n) ')
    while not (cont == 'y' or cont == 'n'):
        cont = input('계속하시겠습니까? (y/n) ')
    return cont == 'n'
```

완성해야 할 함수

```
def safe_sqrt():
    import math
    print("제곱근을 구해드립니다.")
    print("0이상의 수를 입력하세요.")
    while True:
        
        
        
        
        
        
    print("안녕히 가세요.")
```