

Python

함수와 입출력

Spring 2025



AI융합학과

Seongbok Baik

sbbaik@dju.kr

00 Text Book



교재명	으뜸 파이썬
저자	박동규, 강영민
출판사	생능출판사
발행년	2024.06.14



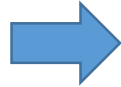
5.6 함수의 인자 전달 방식



코드 5-18: 인자를 빠뜨린 호출
print_star_param_error.py

```
def print_star(n): # 인자가 하나 필요함
    for _ in range(n):
        print('*****')

print_star() # 인자가 없으므로 에러 발생
```



코드 5-19: 디폴트 값을 가지는 print_star() 함수
print_default_param.py

```
def print_star(n = 1): # 매개변수 n은 디폴트 값 1을 가짐
    for _ in range(n):
        print('*****')
print_star() # 인자가 없더라도 에러 없이 수행됨
```

실행결과

```
*****
```

`TypeError: print_star() missing 1 required positional argument: 'n'`

5.6 함수의 인자 전달 방식

- 함수에 특정한 작업을 위임하기 위하여 정확한 인자를 넣어주는 것도 필요하지만 가끔씩은 위와 같은 에러를 예방하고, 좀 더 유연성 있는 작업을 위해서 디폴트 값을 사용하는 것이 편리할 때가 있음
- 이때 사용하는 것이 **디폴트 인자** `default argument`
- [코드 5-19]와 같이 매개변수에 `= 1`과 같이 디폴트 값을 할당

5.6 함수의 인자 전달 방식

- 인자 없이 호출해도 디폴트 값 1을 매개변수 n 에 전달하므로 한 줄의 별표 라인이 정상적으로 출력됨



코드 5-19: 디폴트 값을 가지는 print_star() 함수
print_default_param.py

```
def print_star(n = 1): # 매개변수 n은 디폴트 값 1을 가짐
    for _ in range(n):
        print('*****')
print_star() # 인자가 없더라도 에러 없이 수행됨
```

실행결과

```
*****
```

5.6 함수의 인자 전달 방식

- `print_star()` 함수 호출 시 인자 2를 입력하게 되면 디폴트 값 1을 취하지 않고 인자 값 2를 `n` 값으로 가짐
- 이 경우 2개의 별표 줄을 출력

```
print_star(2) # 인자 값이 2이므로 디폴트 매개변수 n = 1은 수행되지 않음
```

`print_star(2)`의 실행 결과

```
*****  
  
*****
```

5.6 함수의 인자 전달 방식

- 두 개의 매개변수를 사용하는 div()라는 함수
- 두 번째 매개변수에 2라는 디폴트 값을 할당



코드 5-20: 디폴트 인자를 1개 사용한 div() 함수
default_param1.py

```
def div(a, b = 2):  
    return a / b  
  
print('div(4) =', div(4))  
print('div(6, 3) =', div(6, 3))
```

실행결과

```
div(4) = 2.0  
div(6, 3) = 2.0
```

5.6 함수의 인자 전달 방식

- 다음과 같이 첫 번째 매개변수에 디폴트 값 2를 할당하고 두 번째 매개변수를 생략한다면?

```
def div(a = 2, b):  
    return a / b
```

- 에러메시지를 출력함

SyntaxError: non-default argument follows default argument

- 디폴트 매개변수는 전체 변수에 대해 모두 할당하거나 매개변수의 출현 순서상 뒤에 있는 변수부터 할당하여야 한다.

5.6 함수의 인자 전달 방식



코드 5-21: 매개변수에 디폴트 값을 2개 사용한 div() 함수
default_param2.py

```
def div(a = 1, b = 2):  
    return a / b
```

```
print('div() =', div())  
print('div(4) =', div(4))  
print('div(6, 3) =', div(6, 3))
```

[표 5-1] 디폴트 인자를 사용하는 함수에 적용된 인자의 예와 수행 결과

함수 호출문	실제 수행 시 넘겨지는 인자(파란색 인자는 디폴트 값)	반환값
div()	div(1, 2)	0.5
div(4)	div(4, 2)	2.0
div(6, 3)	div(6, 3)	2.0

실행결과

```
div() = 0.5  
div(4) = 2.0  
div(6, 3) = 2.0
```

5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 **keyword argument**



코드 5-22: 2차 방정식의 근을 구하는 함수와 함수 호출문
`root_func.py`

```
def get_root(a, b, c):  
    r1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)  
    r2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)  
    return r1, r2  
  
# 함수 호출 시 1, 2, -8 인자를 사용한다.  
# result1, result2를 이용해서 결과 값을 반환 받는다.  
result1, result2 = get_root(1, 2, -8)  
print('해는', result1, '또는', result2)
```

실행결과

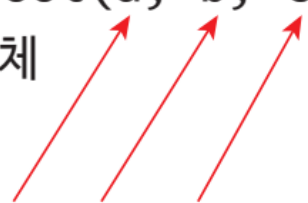
해는 2.0 또는 -4.0

- 함수를 호출할 때 인자의 값만을 전달하는 것이 아니라 그 인자의 이름을 함께 명시하여 전달하는 방식
- 파이썬의 기본 인자 전달 방식을 **위치 인자 **positional argument**** 방식이라고 함

5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 **keyword argument**

```
def get_root(a, b, c):  
    함수 몸체
```



```
get_root(1, 2, -8)
```

[그림 5-10] 위치 인자의 전달 방식: 매개변수에 전달할 값을 a, b, c 순서에 따라 전달하므로 순서가 중요함

```
result1, result2 = get_root(-8, 2, 1)
```

실행결과

해는 -0.25 또는 0.5

5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 **keyword argument**

위치와 상관없이 키워드에 의해서 인자 값이 결정됨

```
result1, result2 = get_root(a = 1, b = 2, c = -8)
```

실행결과

해는 2.0 또는 -4.0

```
result1, result2 = get_root(a = 1, c = -8, b = 2)
```

위의 코드와 아래 코드는 그 결과가 동일하다,
키워드 인자를 사용하면 인자의 위치는 중요하지 않다

```
result1, result2 = get_root(c = -8, b = 2, a = 1)
```

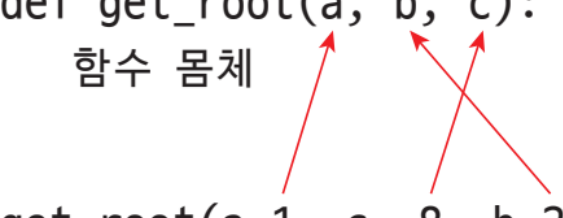
실행결과

해는 2.0 또는 -4.0

5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 keyword argument

```
def get_root(a, b, c):  
    함수 몸체  
  
get_root(a=1, c=-8, b=2)
```



[그림 5-11] 키워드 인자의 전달 방식: a, b, c의 키워드를 통해서 매개변수에 전달할 값을 명시해 줌으로 순서는 중요하지 않음

```
result1, result2 = get_root(c = -8, b = 2, 1)
```

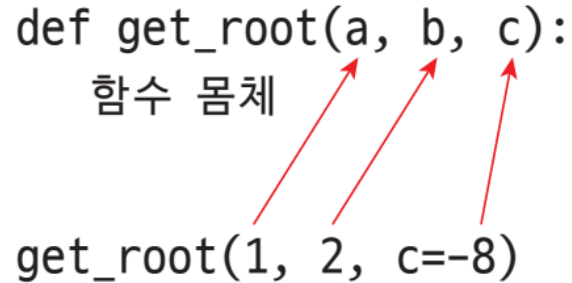


- 키워드 인자와 위치 인자를 섞어서 사용할 적에는 반드시 위치인자가 먼저 나타나야 한다(위의 경우는 오류)

5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 keyword argument

```
def get_root(a, b, c):  
    함수 몸체  
  
get_root(1, 2, c=-8)
```



[그림 5-12] 위치 인자와 키워드 인자의 혼용: 1, 2는 a, b에 전달되고 -8은 키워드를 통해 명시해 준 c에 전달됨



잠깐 - 위치 인자와 키워드 인자로 인자를 전달하기

파이썬의 함수에서는 인자를 전달할 때 위치 인자로 전달하는 방식과 키워드 인자로 전달하는 방식이 있다. 그리고 두 가지 방식을 혼합하는 방식도 있다. 두 가지 방식을 혼합하는 경우 키워드 인자는 반드시 위치 인자 뒤에 와야 한다.

5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 **keyword argument**

```
result1, result2 = get_root(1, -8, b = 2)
```

TypeError: get_root() got multiple values for argument 'b'

```
>>> def func(a, b, c) :  
...     print(a, b, c)  
...
```

```
>>> func(1, 2, 3)
```

```
1 2 3
```

```
>>> func(1, c=2, b=3)
```

```
1 3 2
```

```
>>> func(1, b=2, 3)
```

SyntaxError: positional argument follows keyword argument



5.6 함수의 인자 전달 방식

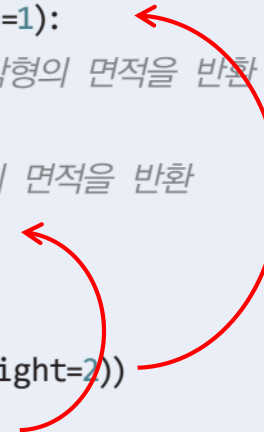
5.6.2 키워드 인자 keyword argument



코드 5-23: 직사각형과 원의 면적을 구하는 함수의 구현

areas_of_shapes.py

```
def func(shape, width=1, height=1, radius=1):  
    if shape == 0 : # shape 값이 0이면 사각형의 면적을 반환  
        return width * height  
    if shape == 1 : # shape 값이 1이면 원의 면적을 반환  
        return 3.14 * radius ** 2  
  
print("rect area =", func(0, width=10, height=2))  
print("circle area =", func(1, radius=5))
```



첫번째 인자의 값에 의해 반환값이 결정됨

실행결과

```
rect area = 20  
circle area = 78.5
```


5.6 함수의 인자 전달 방식

5.6.2 키워드 인자 **keyword argument**

```
print("rect area =", func(0, radius=5))
```

- 위의 경우 func(0, radius=5)를 입력하였는데 첫 인자 0은 함수 내부에서 rect shape을 의미함.

```
rect area = 1
```

- 따라서 문법적인 오류는 없으나 논리적인 문제점이 있음, 첫 매개변수가 0일 경우
- 디폴트 인자 width, height가 각각 1로 처리된 후 아래 문장

```
return width * height
```

이 실행되어 rect_area는 1이 됨

5.6 함수의 인자 전달 방식



LAB 5-9: 키워드 인자

1. 다음과 같이 성(last name)과 이름(first name), 존칭(honorifics)을 매개변수로 받아서 출력하는 함수 `print_name`이 있다.

```
def print_name(honorifics, first_name, last_name):  
    ''' 키워드 인자를 이용한 출력용 프로그램 '''  
    print(honorifics, first_name, last_name)
```

- a) 다음과 같은 함수 호출의 결과는 무엇인가?

```
print_name(first_name='Gildong', last_name='Hong', honorifics='Dr.')
```

- b) 다음과 같은 함수 호출의 결과는 무엇인가?

```
print_name('Gildong', 'Hong', 'Dr.')
```

5.6 함수의 인자 전달 방식

5.6.3 가변적인 인자전달



코드 5-24: 인자를 하나 가지는 함수
arg_greet1.py

```
def greet1(name):  
    print('안녕하세요', name, '씨')  
  
greet1('홍길동')
```

실행결과

안녕하세요 홍길동 씨



코드 5-25: 인자를 2개 가지는 함수
arg_greet2.py

```
def greet2(name1, name2):  
    print('안녕하세요', name1, '씨')  
    print('안녕하세요', name2, '씨')  
  
greet2('홍길동', '홍길순')
```

실행결과

안녕하세요 홍길동 씨
안녕하세요 홍길순 씨

- 인자의 개수를 미리 알 수 없을 경우에는 어떻게 해야만 할까?

5.6 함수의 인자 전달 방식

5.6.3 가변적인 인자전달

- 인자의 수가 정해지지 않은
가변 인자 `arbitrary argument`

→ 별표(*)를 매개변수의
앞에 넣어 사용

- 가변적 인자는 튜플이나 리스트와 비슷하게 for - in문에서 사용가능



코드 5-26: 가변 인자를 가지는 함수의 정의와 호출
`arg_greet.py`

```
def greet(*names):
    for name in names:
        print('안녕하세요', name, '씨')

greet('홍길동', '양만춘', '이순신') # 인자가 3개
greet('James', 'Thomas')          # 인자가 2개
```

실행결과

```
안녕하세요 홍길동 씨
안녕하세요 양만춘 씨
안녕하세요 이순신 씨
안녕하세요 James 씨
안녕하세요 Thomas 씨
```

5.6 함수의 인자 전달 방식

5.6.3 가변적인 인자전달



코드 5-27: 가변 인자를 가지는 함수에서 len() 함수 활용
arg_foo.py

```
def foo(*args):  
    print('인자의 개수:', len(args))  
    print('인자들 :', args)  
  
foo(10, 20, 30)
```

실행결과

```
인자의 개수: 3  
인자들 : (10, 20, 30)
```

- len() 함수를 이용하여 다음과 같이 가변적으로 전달된 인자의 개수를 출력하는 것도 가능

5.6 함수의 인자 전달 방식

5.6.3 가변적인 인자전달

- 숫자의 합을 구하는 프로그램
- `sum_nums()` 함수에 전달될 인자의 개수를 미리 알 수 없는 경우, 가변인자를 받는 `*numbers`라는 매개변수를 사용하여 전체 인자를 튜플 형식으로 받을 수 있음



코드 5-28: 가변 인자를 가지는 함수를 이용한 합계 구하기
`arg_sum_nums.py`

```
def sum_nums(*numbers):  
    result = 0  
    for n in numbers:  
        result += n  
    return result  
print(sum_nums(10, 20, 30))           # 10, 20, 30 인자들의 합을 출력  
print(sum_nums(10, 20, 30, 40, 50))  # 10, 20, 30, 40, 50 인자들의 합을 출력
```

실행결과

```
60  
150
```

5.6 함수의 인자 전달 방식

5.6.3 가변적인 인자전달



LAB 5-10: 가변 인자의 활용

1. 가변 인자를 사용하는 `sum_nums()` 함수를 수정하여 인자들을 튜플 형식으로 출력한 후 모든 값들의 합과 평균을 다음과 같이 출력하시오.

```
3 개의 인자 (10, 20, 30)
합계 : 60 , 평균 : 20.0
5 개의 인자 (10, 20, 30, 40, 50)
합계 : 150 , 평균 : 30.0
```

2. 가변 인자를 사용하는 함수 `min_nums()` 함수를 구현하시오. 이 함수는 정수를 인자로 받을 수 있는데 이 인자의 개수가 가변적이다. 이 함수의 호출문이 다음과 같을 경우

```
min_nums(20, 40, 50, 10)
```

다음과 같은 출력이 나타나도록 하시오.

```
최솟값은 10
```

5.7 재귀함수

- 재귀함수 **recursion**란 함수 내부에서 자기 자신을 호출하는 함수를 말함
- 절차적 기법으로 해결하기 어려운 문제를 직관적이고 간단하게 해결 가능

5.7 재귀함수

- 함수 `factorial()`은 $n! = n * (n-1)!$ 이라는 정의에 맞게 다음과 같이 다시 정의가 가능함



코드 5-29: 재귀함수를 이용하여 정의한 팩토리얼
`factorial_recursion.py`

```
def factorial(n): # n!의 재귀적 구현
    if n <= 1 :   # 종료조건이 반드시 필요하다.
        return 1
    else :
        return n * factorial(n-1) # n * (n-1)! 정의에 따른 구현
n = 5
print('{}! = {}'.format(n, factorial(n)))
```

실행결과

5! = 120

5.7 재귀함수

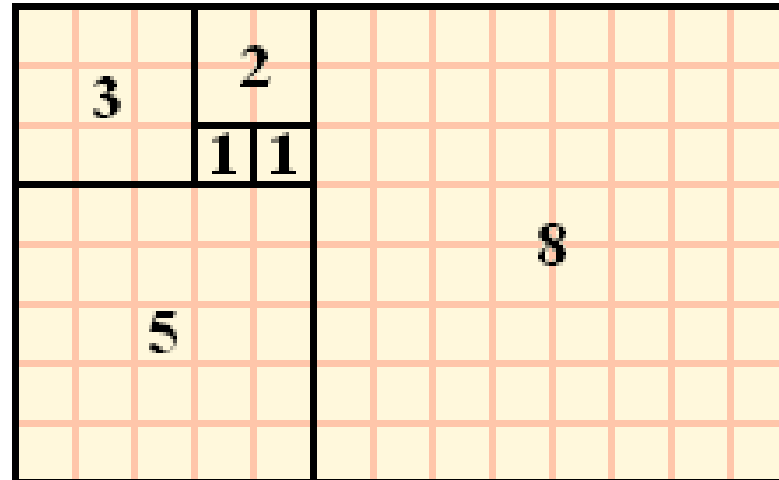
피보나치 수

- 피보나치 수열의 첫번째 항 F_0 은 0이고, F_1 은 1이며, F_n 은 다음과 같이 정의된다.

$$F_n = F_{n-1} + F_{n-2}, F_0 = 0, F_1 = 1$$

- 이 정의에 따르면 피보나치 수열은 다음과 같다

- 0, 1, 1, 2, 3, 5, 8, 13, 21,...



출처: 위키백과

5.7 재귀함수

피보나치 수



코드 5-30: 재귀함수를 이용하여 정의한 피보나치 수열
fibonacci_recursion.py

```
def fibonacci(n): # 피보나치 함수의 재귀적 구현
    if n <= 1:    # 피보나치 함수의 종료 조건
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))
        #  $F_n = F_{(n-1)} + F_{(n-2)}$ 

nterms = int(input("몇 개의 피보나치수를 원하세요? "))

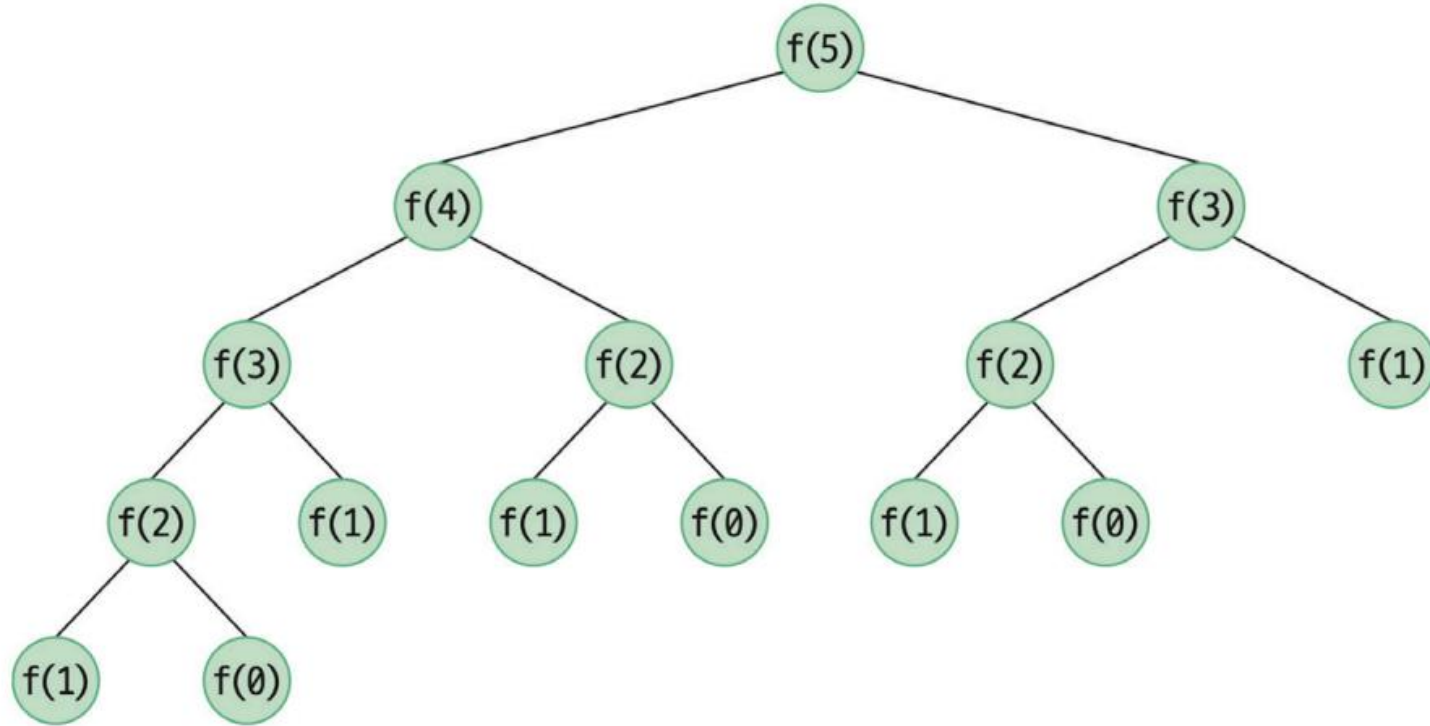
# 음수일 경우 피보나치 수를 구할 수 없음
if nterms <= 0:
    print("오류 : 양수를 입력하세요.")
else:
    print("Fibonacci 수열: ", end = '')
    for i in range(nterms):
        print(fibonacci(i), end=' ')
```

실행결과

```
몇 개의 피보나치수를 원하세요? 10
Fibonacci 수열: 0 1 1 2 3 5 8 13 21 34
```

5.7 재귀함수

피보나치 수



[그림 5-13] `fibonacci(5)` 함수의 수행 과정: `f(5)`는 `f(4)`와 `f(3)`을 호출하며, `f(4)`는 `f(3)`과 `f(2)`를 재귀적으로 호출하여 전체적으로는 매우 큰 실행 트리구조가 된다

5.8 입력함수와 출력함수

input() 함수

- 사용자로부터 입력을 받기 위한 함수

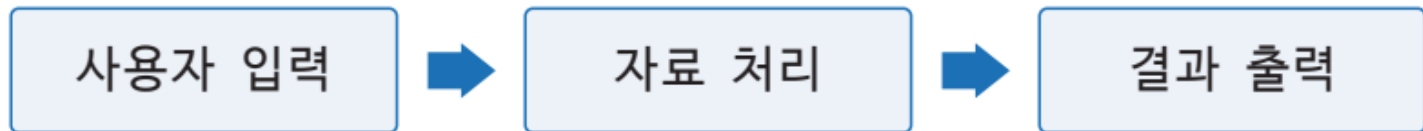


코드 5-31: input() 입력함수를 사용한 name의 입력 방법
input_name1.py

```
print('Enter your name : ')
name = input() # 사용자의 키보드 입력 값을 name0 참조함
print('Hello', name, '!')
```

실행결과

```
Enter your name :
Hong GilDong
Hello Hong GilDong !
```



[그림 5-14] 사용자 입력과 처리, 결과 출력을 가지는 프로그램의 흐름

5.8 입력함수와 출력함수

5.8.1 input() 함수와 int() 함수



대화창 실습: int() 함수와 float(), str() 함수를 이용한 문자열의 변환

```
>>> int('100') + 1
101
>>> float('100') + 1
101.0
>>> '100' + str(1)
'1001'
```



대화창 실습: 대화형 모드를 통한 input()과 int() 함수

```
>>> num1 = int(input("숫자를 입력하세요: "))
숫자를 입력하세요: 100
>>> num2 = int(input("숫자를 입력하세요: "))
숫자를 입력하세요: 200
>>> num3 = num1 + num2
>>> print("두 수의 합은", num3, "입니다.")
두 수의 합은 300 입니다
```

int() 함수로 감싸야 정수 값이 됨

int() 함수로 감싸야 정수 값이 됨

5.8 입력함수와 출력함수

5.8.1 input() 함수와 int() 함수

- '100'이라는 문자형을 정수 100으로 변환할 수 있는 방법은?
- int() 함수를 이용하여 괄호 안의 문자열을 정수형으로 변환하기
- int() 함수 대신에 float() 함수를 사용하면 실수 값으로 변환할 수 있다.

5.8 입력함수와 출력함수

5.8.1 input() 함수와 int() 함수

- 한꺼번에 여러 개의 입력 값 받기
- 이를 위해서는 앞에서 배운 `split()` 메소드를 사용하여 입력된 문자를 공백 단위로 나누어주는 작업이 필요



대화창 실습: `input()` 함수와 공백 구분자를 사용한 `split()` 메소드

```
>>> s1, s2 = input('문자열 2개를 입력하세요: ').split()
```

문자열 2개를 입력하세요: Hello Python

```
>>> s1
```

```
'Hello'
```

```
>>> s2
```

```
'Python'
```

공백으로 구분하여 입력함

5.8 입력함수와 출력함수

5.8.1 input() 함수와 int() 함수

- 정수형 다중 입력과 문자열 다중입력과의 차이점은 각 변수에 문자열을 할당한 뒤 int() 함수를 이용해서 정수형으로 형 변환을 해야 함



대화창 실습: input() 함수와 공백 구분자를 사용한 split() 메소드

```
>>> num1, num2, num3 = input('세 정수를 입력하세요: ').split()
세 정수를 입력하세요: 100 200 300
>>> num1, num2, num3 = int(num1), int(num2), int(num3)
>>> num1, num2, num3
(100, 200, 300)
```

5.8 입력함수와 출력함수

5.8.1 input() 함수와 int() 함수

- split() 메소드의 디폴트 구분자 **separator**인 공백 대신 쉼표를 사용



대화창 실습: 쉼표 구분자를 이용한 input() 함수 실습

```
>>> num1, num2, num3 = input('세 정수를 ,로 구분하여 입력하세요: ').split(',')
세 정수를 ,로 구분하여 입력하세요: 100,200,300
>>> num1, num2, num3 = int(num1), int(num2), int(num3)
>>> num1, num2, num3
(100, 200, 300)
```

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드



대화창 실습: 쉼표 구분자를 이용한 input() 함수 실습

```
>>> 'hello'.upper()  
'HELLO'
```

구분자 **separator**

한 문자열을 하나 이상의 개별 문자열로 나누는 문자

```
>>> 'Guido Van Rossum'.split()  
['Guido', 'Van', 'Rossum']
```

디폴트 구분자는 공백

```
>>> 'Apple,Banana,Orange'.split(',')  
['Apple', 'Banana', 'Orange']
```

구분자로 쉼표를 사용

```
>>> 'Apple|Banana|Orange|Kiwi'.split('|')  
['Apple', 'Banana', 'Orange', 'Kiwi']
```

구분자로 세로바를 사용

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드



잠깐 - 함수와 메소드

함수와 메소드는 특정한 일을 하도록 정의된 동작을 수행한다는 점에서 동일하다. 그런데 함수는 호출하고 인자를 전달하여 특정한 동작을 수행하고 필요한 경우 그 결과를 반환하는 일을 한다. 반면, 메소드는 객체지향 프로그래밍에서 다루는 **객체object**에 부속되는 함수이다. 이 메소드는 자신이 가지고 있는 객체의 속성 혹은 멤버 변수들에 대해 접근할 수 있다. 메소드는 **인스턴스instance**라고 하는 각각 다른 값을 가진 객체들마다 호출할 수 있으며, 이렇게 호출된 메소드는 동일한 이름의 메소드라고 하더라도 자신이 부속된 인스턴스의 현재 데이터(멤버 변수)들에 접근하여 일하게 된다. 예를 들어, 각각의 문자열이 인스턴스라고 할 때, 자신이 가진 특정 알파벳의 수를 출력하는 count()라는 메소드가 정의되어 있다고 하자. 어떤 문자열 s1은 "aaa"이고 다른 문자열 s2는 "aaaaaa"라고 할 때, 두 문자열이 각각 count() 메소드를 부르는 방법은 s1.count('a'), s2.count('a') 형태가 된다. 이때 두 문자열은 동일한 메소드를 동일한 인자로 불렀지만, 그 결과는 인스턴스 각각의 데이터에 근거하여 3과 6을 출력하게 된다.

```
>>> s1 = "aaa"      # "aaa" 인스턴스를 참조하는 s1 변수 생성
>>> s2 = "aaaaaa"  # "aaaaaa" 인스턴스를 참조하는 s2 변수 생성
>>> s1.count('a')   # s1 인스턴스(객체)의 알파벳 'a'의 출현 횟수
3
>>> s2.count('a')   # s2 인스턴스(객체)의 알파벳 'a'의 출현 횟수
6
```

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드

- 템플릿 문자열 `template string` 혹은 베이스 문자열 `base string`
 - 코드를 대화창에 입력할 때 사용되는 문자열
 - 출력 메소드 `format`을 호출하는 문자열
- 플레이스 홀더 `placeholder`
 - 인자의 출력을 목적으로 사용되는 중괄호

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드



대화창 실습: format() 메소드와 플레이스홀더

```
>>> '{} Python!'.format('Hello') # '{} Python!'가 베이스 문자열  
'Hello Python!'
```

플레이스 홀더에 들어갈 내용

플레이스 홀더

```
>>> '{0} Python!'.format('Hello')  
'Hello Python!'
```

베이스 문자열, 템플릿 문자열



대화창 실습: format() 메소드와 플레이스홀더의 인덱스

```
>>> 'I like {} and {}'.format('Python', 'Java')  
'I like Python and Java'  
>>> 'I like {0} and {1}'.format('Python', 'Java')  
'I like Python and Java'
```

인덱스에 따라 Python, Java가 각각 들어감

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드

- 플레이스 홀더 내에 필요한 정수 값(인덱스)을 할당하여 출력 순서를 제어할 수 있다.
(디폴트로 0, 1, ... 이 할당됨)

```
'I like {} and {}'.format('Python', 'Java')
```



```
'I like {0} and {1}'.format('Python', 'Java')
```



[그림 5-15] 플레이스홀더와 인덱스를 이용한 출력 제어

```
'I like {1} and {0}'.format('Python', 'Java')
```



[그림 5-16] 플레이스홀더와 전달인자

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드

- {0}, {1}, {2}와 같이 플레이스 홀더의 번호를 이용하여 다양한 출력을 할 수 있다
- 플레이스 홀더에는 문자열뿐만 아니라 100, 200과 같은 정수형이나 실수형 등 임의의 자료형도 올 수 있음



대화창 실습: format() 메소드와 플레이스홀더의 인덱스

```
>>> 'I like {1} and {0}'.format('Python', 'Java')  
'I like Java and Python'
```


5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드

인덱스를 중복할 수 있음



대화창 실습: format() 메소드와 플레이스홀더의 인덱스 사용법

```
>>> '{0}, {0}, {0}! Python'.format('Hello')
'Hello, Hello, Hello! Python'
>>> '{0}, {0}, {0}! Python'.format('Hello', 'Hi')
'Hello, Hello, Hello! Python'
>>> '{0} {1}, {0} {1}, {0} {1}'.format('Hello', 'Python')
'Hello Python, Hello Python, Hello Python!'
>>> '{0} {1}, {0} {1}, {0} {1}'.format(100, 200)
'100 200, 100 200, 100 200!'
```

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드

- format() 내부에는 문자열 리터럴뿐만 아니라 다음과 같이 변수나 객체를 넣을 수 있음



대화창 실습: 플레이스홀더 내의 객체 출력

```
>>> greet = 'Hello'
>>> '{} World!'.format(greet)
'Hello World!'
```



코드 5-32: 플레이스홀더와 format() 메소드의 사용
print_format1.py

```
name = input('당신의 이름을 입력해주세요 : ')
age = input('나이를 입력해주세요 : ')
job = input('직업을 입력해주세요 : ')

print('당신의 이름은 {}, 나이는 {}살, 직업은 {}입니다.'.format(name, age, job))
```

실행결과

```
당신의 이름을 입력해주세요 : 김철수
나이를 입력해주세요 : 21
직업을 입력해주세요 : 학생
당신의 이름은 김철수, 나이는 21살, 직업은 학생입니다.
```

5.8 입력함수와 출력함수

f-문자열을 알아보자 : format() 메소드보다 편리하다



코드 5-33: 플레이스홀더와 format() 메소드의 사용
print_format2.py

```
name = input('당신의 이름을 입력해주세요 : ')
age = input('나이를 입력해주세요 : ')
job = input('직업을 입력해주세요 : ')
# 문자열의 앞에 f를 적어준다(f-문자열의 사용).
print(f'당신의 이름은 {name}, 나이는 {age}살, 직업은 {job}입니다.')
```

f문자열

중괄호 안에 변수명 사용가능

실행결과

```
당신의 이름을 입력해주세요 : 김철수
나이를 입력해주세요 : 21
직업을 입력해주세요 : 학생
당신의 이름은 김철수, 나이는 21살, 직업은 학생입니다.
```

5.8 입력함수와 출력함수

5.8.2 형식출력을 위한 format() 메소드



LAB 5-11: 다양한 문자열 포매팅

1. 위의 코드를 수정하여 이름, 나이, 직업, 사는 곳까지 다음과 같이 화면에 출력하려고 한다.
 - a) 출력문에서 format() 메소드를 사용하여 출력하여라.
 - b) 출력문에서 f-문자열을 사용하여 출력하여라.
 - c) 출력문에서 format() 메소드나 f-문자열을 사용하지 말고 심표로 구분하여 출력하여라.

5.8 입력함수와 출력함수

5.8.3 고급 format() 메소드

- 문자열 포매팅을 더욱더 세밀하게 하기 위하여 format() 메소드의 고급 기능에 대하여 알아보기
- format() 메소드는 플레이스 홀더 내에 콜론(:)을 찍고 출력의 크기와 형식 지정을 할 수 있다.

5.8 입력함수와 출력함수

5.8.3 고급 format() 메소드



대화창 실습: 출력 간의 크기 지정을 통한 정수 포매팅

```
>>> for i in range(2, 15, 2):
...     print('{0:3d} {1:4d} {2:5d}'.format(i, i*i, i*i*i))
... 
```

```

2      4      8
4     16     64
6     36    216
8     64    512
10    100   1000
12    144   1728
14    196   2744

```

출력 칸 수 지정과 오른쪽
정렬 기능이 있음

오른쪽으로 정렬된
보기 좋은 출력

```
>>> for i in range(2, 15, 2): # f-문자열을 사용함
...     print(f'{i:3d} {i*i:4d} {i*i*i:5d}') # 위의 결과와 동일
```

5.8 입력함수와 출력함수

5.8.3 고급 format() 메소드

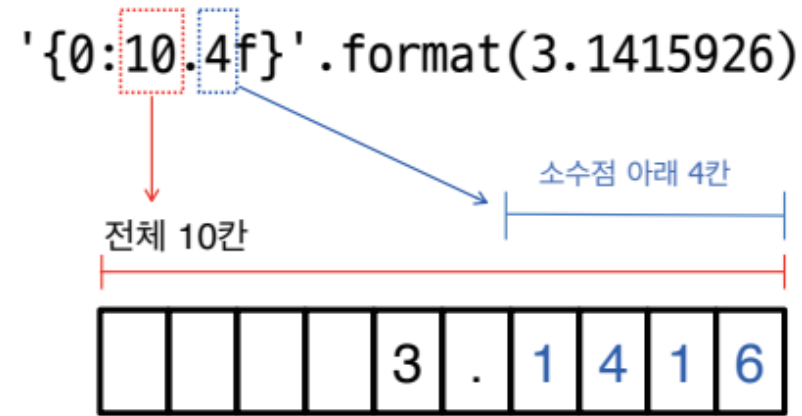
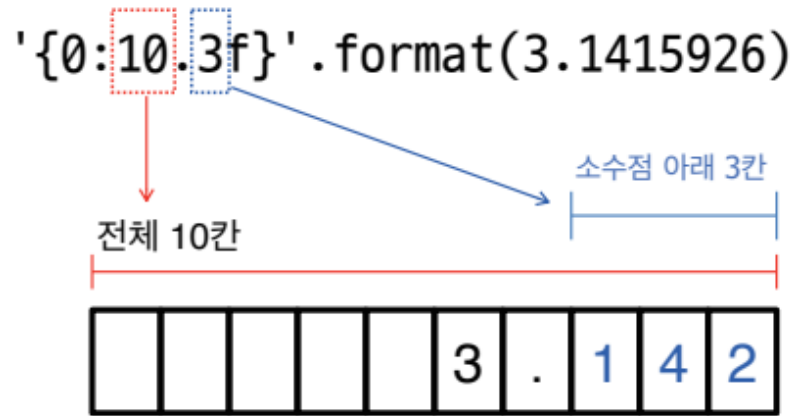


대화창 실습: 소수점 아래 자리 수를 조절하는 실수 포매팅

```
>>> print('소수점 세 자리로 표현한 원주율 = {0:10.3f}'.format(3.1415926))  
소수점 세 자리로 표현한 원주율 = 3.142  
>>> print('소수점 네 자리로 표현한 원주율 = {0:10.4f}'.format(3.1415926))  
소수점 네 자리로 표현한 원주율 = 3.1416
```

5.8 입력함수와 출력함수

5.8.3 고급 format() 메소드



[그림 5-17] {0:10.3f}와 {0:10.4f} 포매팅 출력의 예



대화창 실습: 1,000 단위 쉼표 출력 방법

```
>>> print('{:,}'.format(1234567890)) # 1,000 단위 쉼표 출력
1,234,567,890
```


5.8 입력함수와 출력함수

5.8.3 고급 format() 메소드

- 플레이스 홀더에 출력을 할 때는 key=value와 같이 키와 값을 인자로 넘겨주고 이 키를 이용한 출력도 가능
- 부산시의 위도 35.17N와 경도 129.07E를 출력하는 경우
- 순서에 상관없이 키와 값 형식으로 입력된 정보가 유지되기만 하면 정상적인 출력을 얻을 수 있다.



대화창 실습: 플레이스홀더 내에 키-값 형식으로 인자를 전달하는 방법

```
>>> print('위도 : {0}, 경도: {1}'.format('35.17N', '129.07E'))
위도 : 35.17N, 경도: 129.07E
>>> print('위도 : {lat}, 경도: {long}'.format(lat='35.17N', long='129.07E'))
위도 : 35.17N, 경도: 129.07E
>>> print('위도 : {lat}, 경도: {long}'.format(long='129.07E', lat='35.17N'))
위도 : 35.17N, 경도: 129.07E
```

키워드 인자와 유사

키워드 인자와 유사

5.8 입력함수와 출력함수

5.8.4 다양한 문자열 메소드

- 파싱을 위한 `parse()`, `capitalize()`, `casefold()`, `count()`, `endswith()`, `expandtabs()`, `encode()`, `find()`, `index()`, `isalnum()`, `isdecimal()` 등
- 이런 특수한 기능을 가지는 함수는 사용자에게 큰 편의성을 제공함
- 파이썬의 장점이 될 수 있음

5.8 입력함수와 출력함수

5.8.4 다양한 문자열 메소드



대화창 실습: 문자열의 다양한 메소드

```
>>> 'abc'.upper()      # 대문자로 만든다.  
'ABC'  
>>> 'ABC'.lower()     # 소문자로 만든다.  
'abc'  
>>> 'hobby'.count('h') # 'h' 문자가 나타나는 횟수를 구한다.  
1  
>>> 'hobby'.count('b') # 'b' 문자가 나타나는 횟수를 구한다.  
2  
>>> 'hobby'.find('h')  # 'h' 문자의 위치를 반환한다.  
0  
>>> 'hobby'.find('b')  # 'b' 문자가 최초로 나타나는 위치를 반환한다.  
2
```

5.8 입력함수와 출력함수

5.8.4 다양한 문자열 메소드

- upper() : 해당 문자열을 **대문자**로 변환
- lower() : 해당 문자열을 **소문자**로 변환
- count() : 매개변수로 넘어온 문자와 동일한 문자가 해당 문자열 내에 몇 번 등장하는지 그 **횟수**를 반환
- find() : 매개변수로 넘어온 문자와 동일한 문자가 문자열에서 어디에서 있는지 그 **인덱스**를 반환

5.8 입력함수와 출력함수

5.8.4 다양한 문자열 메소드



대화창 실습: 문자열의 메소드 실습

```
>>> ','.join('ABCD')
'A,B,C,D'
>>> '   hello   '.rstrip() # 오른쪽 공백 지우기
'   hello'
>>> '   hello   '.lstrip() # 왼쪽 공백 지우기
'hello   '
>>> '   hello   '.strip() # 공백 지우기
'hello'
>>> s1 = 'Long live the King!'
>>> s1.replace('King', 'Queen') # 문자열 교환
'Long live the Queen!'
>>> s1.title() # 타이틀 문자열로 변환
'Long Live The King!'
>>> s1.capitalize() # 첫 문자만 대문자로 변환
'Long live the king!'
>>> s2 = "X:Y:Z"
>>> s2.split(':') # :를 구분자로 하여 s2 문자를 리스트로 분리함
['X', 'Y', 'Z']
```

구분자separator

한 문자열을 하나 이상의 개별 문자열로 나누는 문자

여기서는 콜론 문자가 구분자임

5.8 입력함수와 출력함수

5.8.4 다양한 문자열 메소드



LAB 5-12: 다양한 식별자를 활용한 변수 사용

1. `join()` 메소드를 사용하여 'ABCD'의 문자열을 'A_B_C_D'와 같이 출력하여라.
2. 'My favorite thing is monsters.'라는 문자열 `s`에 `replace()` 메소드를 적용하여 'My favorite thing is cartoons.'로 수정된 `t` 문자열을 얻도록 하여라.

5.8 입력함수와 출력함수

5.8.5 내장함수

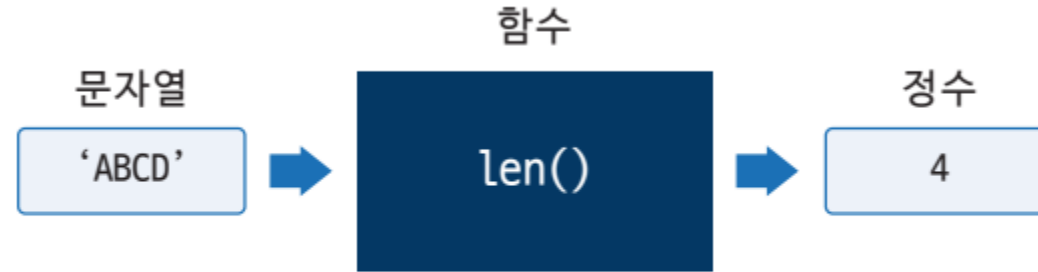
- 프로그래밍에서 함수는 흔히 **블랙박스** **black box**라는 비유를 함
- 함수를 호출하는 사용자는 제대로 된 입력 값을 주고 그 결과인 출력(결과 값)을 사용만 하면 되기 때문



[그림 5-18] 입력에 대해 정해진 출력을 내보내는 블랙박스 역할을 하는 함수

5.8 입력함수와 출력함수

5.8.5 내장함수



[그림 5-19] 내장함수 len()의 동작

- len() 함수가 어떤 값을 반환하는지 알고 있기 때문에 자세한 동작 과정을 몰라도 사용함
- print() 함수, input() 함수도 이런 함수에 속함
- 파이썬에서 기본으로 구현되어 있어 제공하는 함수를 파이썬의 **내장함수** **built-in function**라고 한다

5.8 입력함수와 출력함수

5.8.5 내장함수

[표 5-2] 파이썬의 내장함수 목록

파이썬의 내장함수				
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	_import_()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

5.8 입력함수와 출력함수



대화창 실습: 대화형 모드를 통한 여러 가지 내장함수 실습

```
>>> abs(-100) # 절대값을 반환하는 함수
100
>>> min(200, 100, 300, 400) # 여러 원소들 중 최솟값을 반환하는 함수
100
>>> max(200, 100, 300, 400) # 여러 원소들 중 최댓값을 반환하는 함수
400
>>> str1 = "F00" # "Foo" 혹은 'F00' 형식으로 문자열 객체를 생성함
>>> len(str1) # 문자열의 길이를 반환
3
>>> eval("100+200+300") # 문자열을 수치값과 연산자로 변환하여 평가
600
>>> sorted("EABFD") # 문자열을 정렬
['A', 'B', 'D', 'E', 'F']
>>> list = [200, 100, 300, 400]
>>> sorted(list)
[100, 200, 300, 400]
>>> sorted(list, reverse = True)
[400, 300, 200, 100]
```

5.8 입력함수와 출력함수

- `abs()` 함수는 `-100`이라는 정수를 입력 받아서 그 절댓값인 `100`을 반환
- `min()` 함수는 `200, 100, 300, 400` 의 값을 가지는 정수들 중에서 가장 작은 값을 반환
- `max()` 함수는 `200, 100, 300, 400` 의 값을 가지는 정수들 중에서 가장 큰 값을 반환
- `id()` 함수는 "FOO"라는 문자열이 저장된 변수 `str1`의 identity를 반환
- `type()` 함수는 해당 변수의 자료형을 반환
`len()` 함수는 변수 `str1`에 저장된 문자열의 길이를 반환
- `eval()` 함수는 문자열을 받아와 해당 문자열의 내용을 수식화해서 평가(evaluate)한 다음 평가한 값을 반환
- `sorted()` 함수는 문자열을 받아와 해당 문자열을 구성하는 문자들을 알파벳순으로 정렬해 반환

5.8 입력함수와 출력함수

- 파이썬은 **객체지향 프로그래밍 언어** **object oriented programming language**
- 다양한 속성과 기능을 가진 객체들이 프로그램을 구성하는 패러다임이 객체지향 언어의 핵심
- 파이썬의 객체는 다른 객체와 구별되는 고유한 **식별값** **identity**을 가지고 있으며 `id()` 함수는 이 객체의 식별 값을 정수형으로 반환

5.8 입력함수와 출력함수



대화창 실습: 대화형 모드를 통한 id() 함수 실습

```
>>> a_str = "Hello Python!"  
>>> id(a_str)      # a_str 변수가 참조하는 객체의 id  
4549938992
```

5.8 입력함수와 출력함수

type() 함수는 객체의 자료형을 반환



대화창 실습: 여러가지 자료형에 대한 type() 함수의 적용

```
>>> type(123)
<class 'int'>
>>> type('Hello String!')
<class 'str'>
>>> type(120.3)
<class 'float'>
>>> type([100, 300, 600])
<class 'list'>
```

Leistung ist nicht alles / Keinen Studierenden zurücklassen

