

으뜸 파이썬

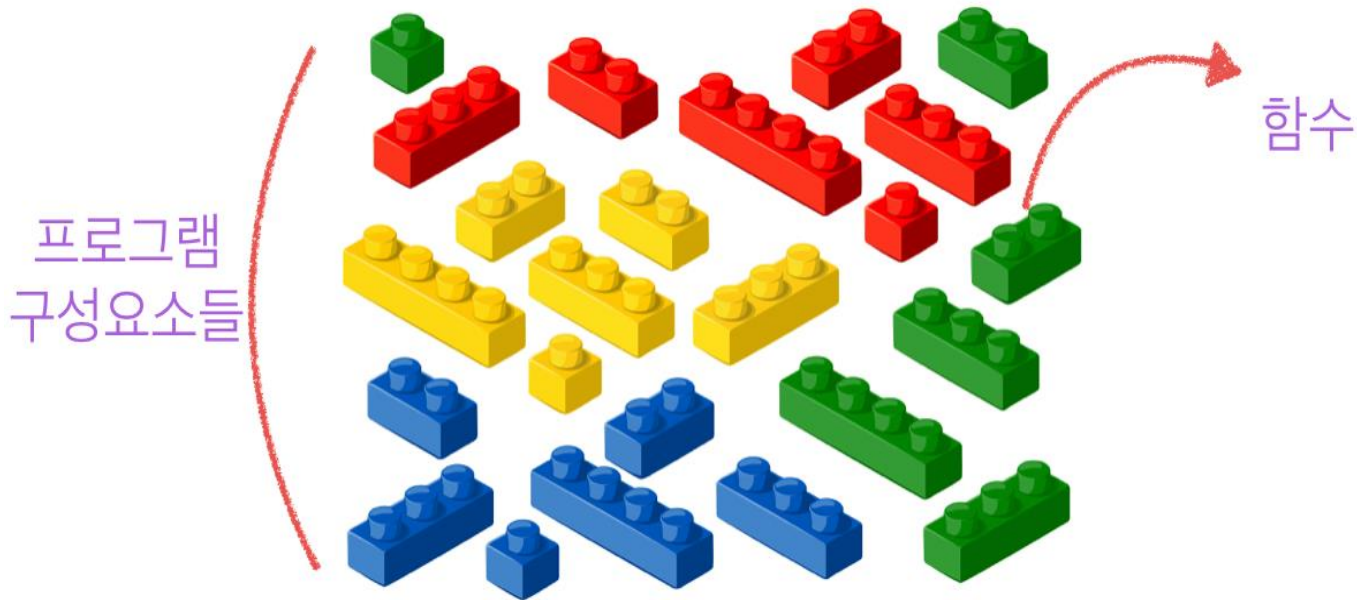


4장 함수와 입출력

학습목표

- 함수에 대해 이해하고 그 필요성을 설명할 수 있다.
- 내장 함수와 사용자 정의 함수를 이해하고 설명할 수 있다.
- 사용자 정의 함수를 def 문을 이용하여 정의하고 호출할 수 있다.
- 호출되는 함수에 값을 전달하기 위하여 매개변수를 사용할 수 있다.
- 함수의 반환문에 대해 이해하고 그 필요성을 설명할 수 있다.
- 다중 반환문을 사용하여 여러 개의 값을 반환할 수 있다.
- 지역변수와 전역변수를 올바르게 사용할 수 있다.
- 순서 매개변수와 키워드 매개변수를 이용하여 효율적으로 값을 전달할 수 있다.
- 함수에 전달되는 가변 인자를 처리하는 방법을 익힌다.
- 출력 함수와 입력 함수의 목적과 사용방법을 설명할 수 있다.
- input() 함수를 이용한 입력 방법에 대해 이해하고 사용할 수 있다.
- format() 메소드와 플레이스홀더를 이용한 출력을 할 수 있다.
- 여러 가지 출력서식을 사용할 수 있다.
- 파이썬의 다양한 내장 함수를 사용할 수 있다.

4.1 함수의 역할



[그림 4-1] 레고 블록

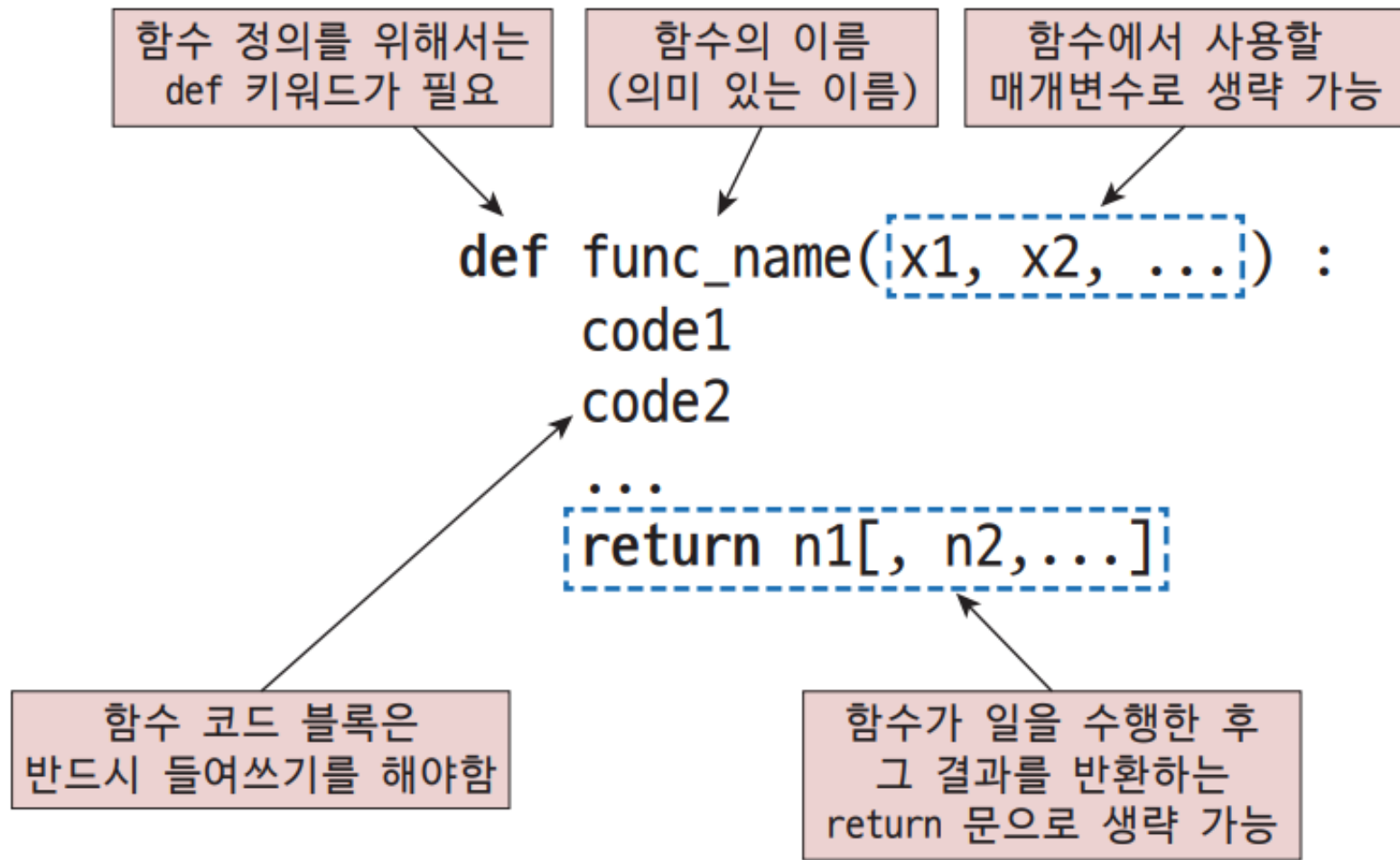


[그림 4-2] 레고 블록을 조립해서 만든 자동차

(출처: bricklink.com)

여러분이 사용하는 프로그램도 많은 부품
(함수나 클래스)으로 이루어져 있습니다

- 반복적으로 사용되는 코드 - 덩어리(혹은 **블록**block) 이라고 함
- 기능에 따라 미리 만들어진 블록은 필요할 때 **호출**function call 함
- 파이썬에서 미리 만들어서 제공하는 함수는 인터프리터에 포함되어 배포되는데 이러한 함수를 **내장함수**built-in function 라고 함
 - 대표적으로 print()가 있음
- 사용자가 직접 필요한 함수를 만들 수 있음
- 이러한 함수를 **사용자 정의 함수**user defined function라고 함
- def 키워드 사용 : define의 약자
 - def를 이용한 함수 정의 방법을 배워볼 예정



[그림 4-3] 파이썬에서 함수를 정의하는 문법

- return문이 없는 간단한 코드로 함수를 정의하고 호출하기

코드 4-1 : 별표 출력을 위한 함수 정의와 호출

print_star_func.py

```
def print_star():                                # 별표 출력을 위한 함수 정의
    print('*****')

print_star()                                     # 별표 출력을 위한 함수 호출
```

실행결과

```
*****
```

코드 4-2 : 별표 출력을 위한 함수 정의와 반복 호출

print_star_4.py

```
def print_star():
```

```
    print('*****')
```

```
print_star()    # 별표 출력함수 호출 1
```

```
print_star()    # 별표 출력함수 호출 2
```

```
print_star()    # 별표 출력함수 호출 3
```

```
print_star()    # 별표 출력함수 호출 4
```

실행결과

```
*****
```

```
*****
```

```
*****
```

```
*****
```

- print_star()라는 함수는 어떤 일을 하도록 정의된 명령어들의 집합(혹은 블록)이며 이 집합은 외부에서 호출할 때마다 수행되는 것을 확인해 볼 수 있다.



LAB 4-1 : 함수 정의와 호출

1. [코드 4-1]의 함수 호출문을 삭제하면 어떻게 되는가?
2. [코드 4-2]를 수정하여 6줄의 별표를 출력해 보시오. 이때 함수 호출을 6회 하시오.

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*****
```


코드 4-3 : 3줄 별표 출력을 위한 함수 정의와 호출 방법

print_star3.py

```
def print_star3():
```

```
    print('*****')
```

```
    print('*****')
```

```
    print('*****')
```

```
print_star3() # 3줄의 별표가 출력됨
```

```
print_star3() # 3줄의 별표가 출력됨
```

```
print_star3() # 3줄의 별표가 출력됨
```

실행결과

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



LAB 4-2 : 함수 정의와 호출

1. [코드 4-3]을 수정하여 함수 호출 두 번으로 10줄의 별표를 출력해 보시오.

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

코드 4-4 : 별표 출력을 위한 함수 정의와 호출 방법의 수정

print_star_plus.py

```
def print_star(): # 별표 기호를 한 줄 출력함
    print('*****')

def print_plus(): # 더하기 기호를 한 줄 출력함
    print('+++++')

print_star()      # 별표 기호 출력
print_plus()      # 더하기 기호 출력
print_star()
print_plus()
```

실행결과

```
*****
+++++
*****
+++++
```

- 한번 만들어진 함수는 다른 프로그램에서 재사용이 가능
- 프로그램 개발의 시간과 비용을 절약할 수 있다



LAB 4-3 : 함수 정의와 호출

1. [코드 4-4]를 수정하여 해시마크(#)를 한 줄 출력하는 `print_hash()` 함수를 추가로 구현하십시오.
2. `print_star()`, `print_plus()`, `print_hash()` 함수를 모두 이용하여 다음과 같은 출력이 나타나도록 함수를 호출하십시오.

```
#####  
*****  
+++++  
+++++  
*****  
#####
```

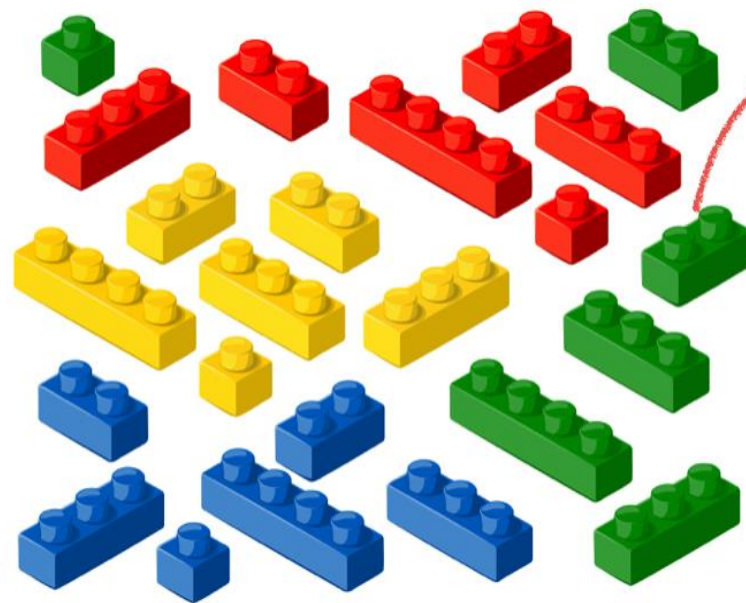


NOTE : 함수를 사용할 때의 장점

이제까지 배운 내용을 바탕으로 함수의 장점을 정리해 보면 다음과 같은 결론을 얻을 수 있을 것이다.

1. 하나의 큰 프로그램을 여러 부분으로 나누어 작성할 수 있기 때문에 구조적인 프로그래밍이 가능하다.
2. 다른 프로그램에서 함수를 재사용하는 것이 가능하다.
3. 코드의 가독성이 증가한다.
4. 프로그램 수정 시에도 일부 함수만 수정하면 되기 때문에 유지 관리가 쉬워진다.
5. 이미 개발된 함수를 사용하게 되면 프로그램 개발의 시간과 비용을 절약할 수 있다.

프로그램
구성요소들



함수

4.2 함수와 매개변수

전달받을 값 3, 4를 가지는 변수 m, n : 매개변수

```
def foo(m, n) :  
    code  
    ...  
    return n1[, n2,...]
```

foo(3, 4)

foo 라는 함수에 넘겨줄 값 3, 4 : 인자

[그림 4-4] 매개변수와 인자의 개념과 사용방법



NOTE : 인자와 매개변수

- **매개변수**Parameter : 함수나 메소드 헤더부에 정의된 변수로 함수가 호출될 때 실제 값을 전달받는 변수이다.

예: def foo(m, n): 의 m과 n

- **전달인자**Argument : 함수나 메소드가 호출될 때 전달되는 실제 값을 말하며, 간단하게 인자라고도 한다.

예: foo(3, 4)의 3과 4

```
def print_star():  
    print('*****')
```

```
print_star() # 별표 출력 1
```

```
print_star() # 별표 출력 2
```

```
print_star() # 별표 출력 3
```

```
print_star() # 별표 출력 4
```

코드 4-5 : 매개변수를 가진 별표 출력 함수와 인자를 이용한 호출

print_star_param.py

```
# 별표 출력을 매개변수 n번만큼 반복하는 프로그램
def print_star(n):
    for _ in range(n):
        print('*****')

print_star(4) # 별표 출력을 위해 4라는 인자 값을 준다.
```

실행결과

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




LAB 4-4 : 다양한 별표와 패턴 출력

1. [코드 4-5]를 수정하여 별표(*)줄이 10개 출력되도록 인자를 변경하시오.
2. [코드 4-5]를 수정하여 별표(*) 표시 대신 해시마크(#)가 출력되도록 하시오. 이를 위해 함수 이름을 print_hash(n)으로 수정하고 수정된 함수를 print_hash(10)과 같이 호출하시오.
3. print_hash(6)을 호출하여 다음과 같은 출력이 나타나도록 하여라.

```
#####  
#####  
#####  
#####  
#####  
#####
```

4. print_hash(6)을 호출하여 다음과 같은 출력이 나타나도록 하여라. 다음 화면과 같이 매번 해시가 출력될 때마다 앞 칸에 줄 번호를 0부터 표시하도록 하여라.

```
0 #####  
1 #####  
2 #####  
3 #####  
4 #####  
5 #####
```

코드 4-6 : 매개변수를 사용하여 지정된 문자를 인자 값만큼 반복 출력하기

print_hello_n_times.py

```
def print_hello(n):          # 매개변수를 이용한 반복 출력
```

```
    print('Hello ' * n)
```

```
print('Hello를 두 번 출력합니다.')
```

인자가 2이면 Hello를 2회 출력

```
print_hello(2)
```

```
print('Hello를 세 번 출력합니다.')
```

인자가 3이면 Hello를 3회 출력

```
print_hello(3)
```

```
print('Hello를 네 번 출력합니다.')
```

인자가 4이면 Hello를 4회 출력

```
print_hello(4)
```

실행결과

Hello를 두 번 출력합니다.

Hello Hello

Hello를 세 번 출력합니다.

Hello Hello Hello

Hello를 네 번 출력합니다.

Hello Hello Hello Hello

코드 4-7 : 매개변수를 사용한 인자 값의 합계

sum_func.py

```
def print_sum(a, b):                # 두 개의 매개변수를 가진 함수
    result = a + b
    print(a, '과', b, '의 합은', result, '입니다.')

print_sum(10, 20)
print_sum(100, 200)
```

실행결과

10 과 20 의 합은 30 입니다.
100 과 200 의 합은 300 입니다.

- 함수 호출시에 인자를 하나만 넣어주게 되면 함수에서 필요한 b라는 매개변수의 값이 없으므로 TypeError 메시지가 나타나며 실행되지 않음

이 경우 인자의 수가 일치하지 않음!!

print_sum(10) # 에러



TypeError: print_sum() missing 1 required positional argument: 'b'



LAB 4-5 : 두 수 연산을 수행하는 함수

1. 두 개의 매개변수 a , b 를 받아서 두 수의 차를 구하여 출력하는 `print_sub(a, b)` 함수를 구현하여라. `print_sum(10, 20)`을 호출한 결과 다음과 같은 출력이 나타나도록 하여라.

10과 20의 차는 -10입니다.

2. 두 개의 매개변수 a , b 를 받아서 두 수의 곱을 구하여 출력하는 `print_mult(a, b)` 함수를 구현하여라. `print_mult(10, 20)`을 호출한 결과 다음과 같은 출력이 나타나도록 하여라.

10과 20의 곱은 200입니다.

4.3 매개변수를 활용한 2차 방정식의 근 구하기

$$ax^2 + bx + c = 0$$

[수식 4-1] x에 대한 2차 방정식

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

[수식 4-2] 2차 방정식의 근의 공식

코드 4-8 : 2차 방정식의 근을 구하는 기능

root_ex1.py

```
a = 1
b = 2
c = -8

# ( a * x^2 ) + ( b * x ) + c = 0

r1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
r2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)

print('해는', r1, '또는', r2)
```

실행결과

해는 2.0 또는 -4.0

- $a(a \neq 0)$, b , c 를 해당하는 값을 방정식에 맞게 입력
- a , b , c 에 해당하는 해를 변수 $r1$, $r2$ 에 저장하여 출력

- 변수 a, b, c의 값을 2, -6, -8로 바꾼 방정식의 해를 구하고 싶을 때

코드 4-9 : 2차 방정식의 근을 구하는 기능의 반복 사용

root_ex2.py

```
a = 1
b = 2
c = -8

# 근의 공식으로 해를 한 번 더 구한다.(반복되는 코드)
r1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
r2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
print('해는', r1, '또는', r2)

a = 2
b = -6
c = -8

# 근의 공식으로 해를 한 번 더 구한다.(반복되는 코드)
r1 = (-b + (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
r2 = (-b - (b ** 2 - 4 * a * c) ** 0.5) / (2 * a)
print('해는', r1, '또는', r2)
```

실행결과

해는 2.0 또는 -4.0

해는 4.0 또는 -1.0

- 문제점

- 변수 a, b, c 에 원하는 계수를 입력하고, 다시 $r1, r2$ 의 수식을 구해줘야 함
- 복사, 붙여 넣기를 한다 해도 코드가 중복되는 부분이 많고 불필요하게 긴 것을 한눈에 알 수 있다.

코드 4-10 : 2차 방정식의 근을 구하는 기능을 함수로 만들기

root_ex3.py

```
def print_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)  
    print('해는', r1, '또는', r2)
```

계수 값이 다른 2차 방정식의 해를 구함

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

```
print_root(2, -6, -8)
```

- 밖에서 넘겨준 계수 3개 a, b, c를 매개변수로 받고, 함수 몸체에 근의 공식 연산을 한 후, 결과 r1, r2를 출력하는 코드
- 코드가 훨씬 간결해지고, 사용하기 편리함

실행결과

해는 2.0 또는 -4.0

해는 4.0 또는 -1.0



LAB 4-6 : 함수의 사용

1. 다음 2차 방정식의 근을 [코드 4-10]의 함수를 사용하여 출력하여라.

(1) $x^2 + 4x - 21 = 0$

(2) $x^2 - 6x + 8 = 0$

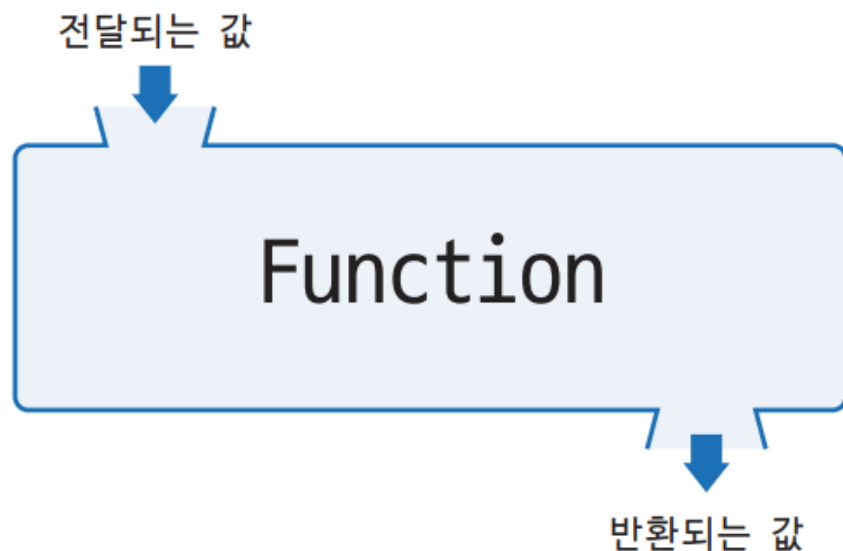
2. 삼각형의 면적을 구하기 위하여 밑변과 높이를 사용하려고 한다. 두 개의 매개변수 width, height를 받아서 삼각형의 면적을 출력하는 함수 print_area(width, height)를 구현하여라. 이때 print_area(10, 20)을 호출하여 다음과 같은 출력이 나타나도록 하여라.

```
print_area(10, 20) # 삼각형의 면적을 구하는 함수 호출
```

출력 결과 :

```
밑변 10, 높이 20인 삼각형의 면적은 : 100
```

4.4 반환문 return



[그림 4-5] 값의 전달과 반환 : 함수는 값을 전달받아 처리하고 결과를 반환할 수 있다

```
def func_name([x1, x2, ...]) :  
    code1  
    code2  
    ...  
    return n1[, n2,...]
```

함수가 일을 수행한 후 그 결과를 반환하는 기능

[그림 4-6] 함수의 구성과 반환문 return

반환문 return

- 일반적으로 함수 내부는 **블랙박스** **black box**라고 가정
- 함수의 내부는 특정한 코드를 가지고 있으며 주어진 일을 수행하고 결과를 반환할 수 있음
- **return** 키워드를 사용하여 하나 이상의 값을 반환해 줄 수 있음

코드 4-11 : 두 값의 합을 반환하는 get_sum() 함수와 return 문의 사용

sum_with_return1.py

```
def get_sum(a, b):                # 두 수의 합을 반환하는 함수
    result = a + b
    return result                 # return 문을 사용하여 result를 반환

n1 = get_sum(10, 20)
print('10과 20의 합 =', n1)

n2 = get_sum(100, 200)
print('100과 200의 합 =', n2)
```

실행결과

10과 20의 합 = 30

100과 200의 합 = 300

- return을 이용하여 값을 반환할 때는 다음과 같이 return 키워드 다음에 수식을 입력하여 수식의 결과를 반환하는 것도 가능

코드 4-12 : 두 값의 합을 반환하는 get_sum() 함수와 return 문의 사용2

sum_with_return2.py

```
def get_sum(a, b):  
    return a + b
```

```
result = get_sum(100, 200)  
print('두 수의 합', result)
```

```
result = get_sum(100, 200)  
print('두 수의 합', result)
```

```
print('두 수의 합', get_sum(100, 200))
```

두가지 방법의 결과는 동일함

코드 4-13 : 두 개의 값을 튜플로 반환하는 방법과 전달받는 방법

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
```

두 근 r1, r2를 반환함

```
# 함수 호출시 인자를 상수 값을 사용함  
# result1, result2를 이용해서 결과 값을 반환 받아온다.
```

```
result1, result2 = get_root(1, 2, -8)  
print('해는', result1, '또는', result2)
```

result1, result2가 r1, r2의 값을 받아온다

실행결과

해는 2.0 또는 -4.0



LAB 4-7 : 원의 면적과 둘레를 반환하는 함수

1. 원의 면적과 둘레를 구하기 위하여 원의 반지름을 사용하려고 한다. 이 때 사용할 `circle_area_circum(radius)` 함수는 다음과 같이 반지름(radius) 10을 입력으로 받아서 면적(area)과 둘레(circum)의 두 값을 반환하도록 구현하여라.

```
radius = 10  
area, circum = circle_area_circum(radius) # 원의 면적과 둘레를 반환
```

출력 결과 :

반지름 10인 원의 면적은 314.0, 원의 둘레는 62.8

- 위의 코드에서는 계수 a, b, c 를 매개변수로 받아서 근의 공식으로 해를 계산해 변수 $r1, r2$ 에 저장, return문으로 두 값을 반환
- 함수 외부에서는 함수 `get_root`를 호출해 그 해를 변수 `result1, result2`에 저장하여 출력
- 이와 같이 두 개 이상의 값을 반환하는 반환문을 **다중 반환문**
`multiple return statement`이라고 함
- 다중 반환문에서 쉼표로 구분되는 두 개의 값은 튜플 형으로 반환이 이루어짐



LAB 4-8 : 다수의 결과를 반환하는 함수 만들기

1. 어떤 정수 n 과 m 을 입력하면, n 의 배수 m 개를 반환하는 `multiplies(n, m)` 함수를 구현하고 다음과 같이 호출하여 결과를 출력하여라.

```
r1, r2, r3, r4 = multiples(3, 4) # 3의 배수 4 개를 구하라
print(r1, r2, r3, r4)
r1, r2, r3, r4, r5 = multiples(2, 5) # 2의 배수 5 개를 구하라
print(r1, r2, r3, r4, r5)
```

출력 결과 :

```
3 6 9 12
2 4 6 8 10
```

4.5 전역 변수

- 전역변수 **global variable**
 - 함수 바깥에서 선언되거나 전체 영역에서 사용 가능한 변수

코드 4-14 : 매개변수를 사용하지 않고 외부 변수를 사용하는 경우

sum_func_global1.py

```
def print_sum():  
    result = a + b  
    print('print_sum() 내부 :', a, '과', b, '의 합은', result, '입니다.')
```

a = 10 # 전역변수 a
b = 20 # 전역변수 b

```
print_sum()  
result = a + b  
print('print_sum() 외부 :', a, '과', b, '의 합은', result, '입니다.')
```

실행결과

print_sum() 내부 : 10 과 20 의 합은 30 입니다.

print_sum() 외부 : 10 과 20 의 합은 30 입니다.

코드 4-15 : 함수 외부에서 정의된 값을 함수 내부에서 변경하는 경우

sum_func_global2.py

```
def print_sum():  
    a = 100  
    b = 200  
    result = a + b  
    print('print_sum() 내부 :', a, '과', b, '의 합은', result, '입니다.')
```

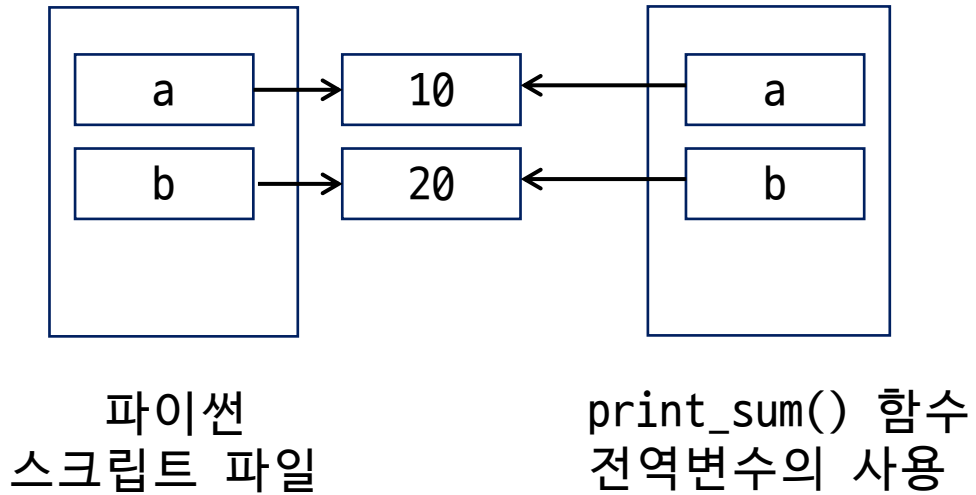
a = 10

b = 20

print_sum()

실행결과

print_sum() 내부 : 100 과 200 의 합은 300 입니다.



[그림 4-8] 파이썬 스크립트 파일과 전역변수,
그리고 이 전역변수를 사용하는
print_sum() 함수

코드 4-16 : 함수 내부에서 값을 변경하고, 그 값을 외부에서 확인하기

sum_func_global3.py

```
def print_sum():  
    a = 100  
    b = 200  
    result = a + b  
    print('print_sum() 내부 :', a, '과', b, '의 합은', result, '입니다.')
```

a = 10
b = 20
print_sum()
result = a + b
print('print_sum() 외부 :', a, '과', b, '의 합은', result, '입니다.')

100, 200을 참조하는 새로운
a, b 변수 생성

10, 20을 참조하는 a, b 변수 생성

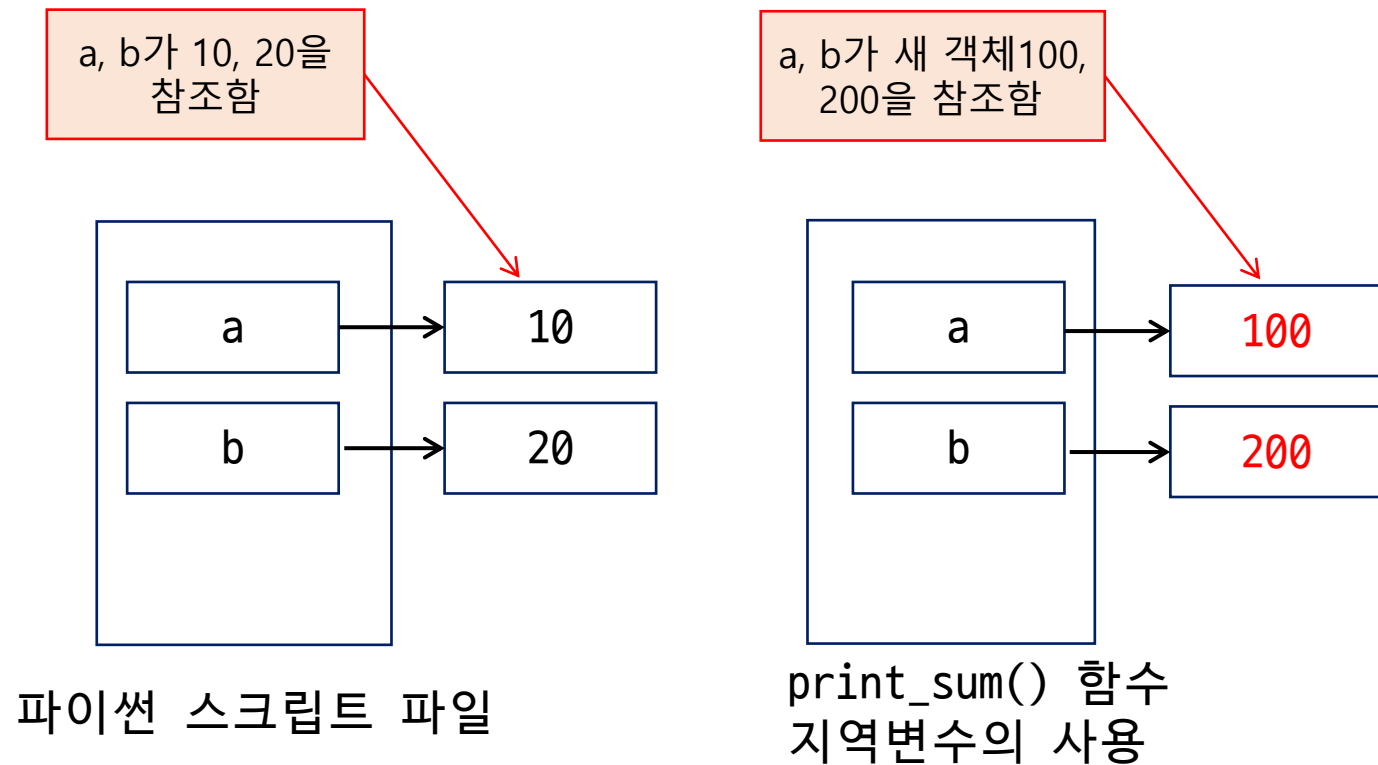
실행결과

print_sum() 내부 : 100 과 200 의 합은 300 입니다.

print_sum() 외부 : 10 과 20 의 합은 30 입니다.

- print_sum()을 수행한 다음 함수 외부에서 다시 한번 a와 b를 합하여 result에 대입하고 그 결과를 출력

- 할당 **assign**
 - $a = 100, b = 200$
- 지역 변수 **local variable**
- 참조 **reference**



[그림 4-9] 파이썬 스크립트 파일과 전역변수, 그리고 이 지역변수를 사용하는 print_sum() 함수. 이 함수 내부의 a, b는 지역변수가 참조하는 객체가 아닌 별개의 객체를 참조함

코드 4-17 : global 키워드를 사용한 전역변수의 참조 방법

sum_func_global4.py

```
def print_sum():  
    global a, b  
    # a, b는 함수외부에서 선언된 a, b를 사용한다.  
    a = 100  
    b = 200  
    result = a + b  
    print('print_sum() 내부 :', a, '과', b, '의 합은', result, '입니다.')
```

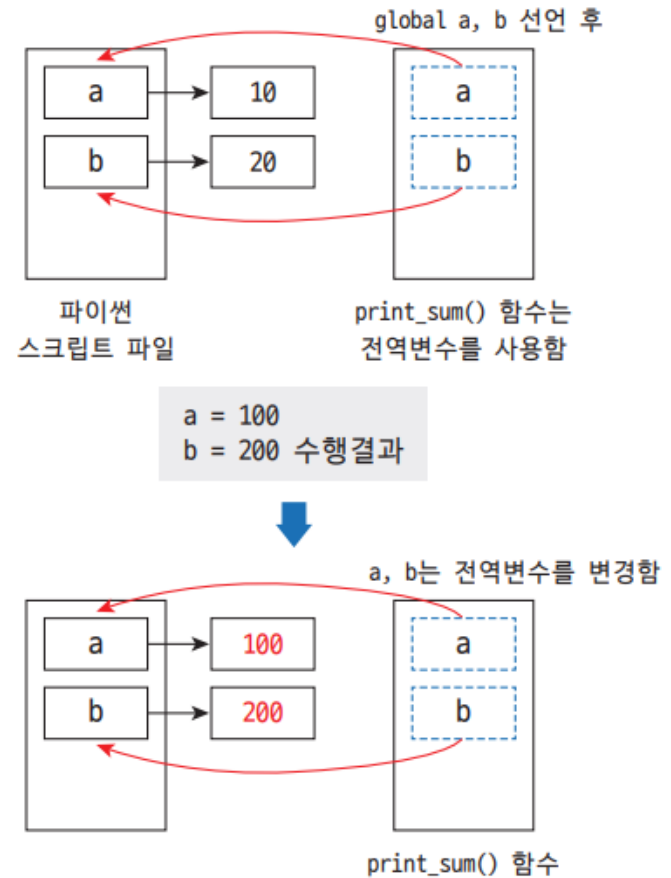
```
a = 10  
b = 20  
print_sum()  
result = a + b  
print('print_sum() 외부 :', a, '과', b, '의 합은', result, '입니다.')
```

실행결과

print_sum() 내부 : 100 과 200 의 합은 300 입니다.

print_sum() 외부 : 100 과 200 의 합은 300 입니다.

global a = 100 # 문법 오류 발생



[그림 4-9] 명시적 global 선언을 통하여 함수 내에서 전역변수 a, b를 사용하는 과정



주의 : 전역변수와 전역상수

전역변수를 사용하는 것은 파이썬뿐만 아니라 모든 프로그래밍 언어에서 매우 나쁜 습관이다. 특히 코드의 길이가 길어질 경우 전역변수는 예외의 주요 원인이 된다.

그러나 **전역상수** `global constant`의 경우는 반드시 나쁘다고 볼 수 없다. 전역상수는 다음과 같이 `global`이라는 키워드로 선언하는데 함수의 외부에서 선언해서 모듈 전체에서 참조할 수 있다. 전역 상수값은 일반적으로 **대문자**를 사용한다.

아래의 코드를 살펴보면, `GLOBAL_VALUE`라는 이름의 변수에 1024라는 값을 할당한 후 `foo()` 함수에서 이 변수 값을 불러서 사용하기 위해 `global GLOBAL_VALUE`라는 이름으로 선언했다.

전역상수의 예 :

```
GLOBAL_VALUE = 1024
...
def foo():
    global GLOBAL_VALUE
    a = GLOBAL_VALUE * 100
```

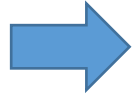
수학 연산을 위해 사용되는 `math` 모듈의 경우 원주율 `pi`와 오일러 상수 `e`를 프로그램 전체에서 참조하여 사용하는데, 이러한 상수 값의 경우는 예외적으로 소문자로 표기한다.

4.6 함수의 인자 전달 방식

코드 4-18 : 인자를 빠뜨린 호출

```
print_star_param_error.py
def print_star(n):    # 인자를 필요로 함
    for _ in range(n):
        print('*****')

print_star()          # 인자가 없으므로 에러 발생
```



코드 4-19 : 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'

- 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 매개변수** default parameter
- [코드 4-19]와 같이 매개변수에 = 1과 같이 디폴트 값을 할당

- 인자가 없이 호출해도 디폴트 값 1을 매개변수 n에 전달하므로 한 줄의 별표 라인이 정상적으로 출력됨

코드 4-19 : 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

```
*****
```

- print_star() 함수 호출시 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

```
*****
```

```
*****
```

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

코드 4-20 : 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
```

- 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.

코드 4-21 : 매개변수에 디폴트 값을 2개 사용한 div() 함수

default_param2.py

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

```
print('div() =', div()) # 인자가 없을 경우 div(1, 2)로 간주  
print('div(4) =', div(4)) # div(4, 2)로 간주함  
print('div(6, 3) =', div(6, 3))
```

실행결과

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

[표 4-1] 디폴트 인자를 사용하는 함수에 적용된 인자의 예와 수행 결과

함수 호출문	실제 수행시 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

[표 4-1] 디폴트 매개변수를 고려한 함수 호출과 실제 수행시 가지는 인자 값(파란색 1, 2는 디폴트 인자로 함수호출시 빼먹을 경우 넘겨짐)

4.6.2 키워드 인자 keyword argument

코드 4-22 : 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 호출시 인자를 상수 값을 사용함.

result1, result2를 이용해서 결과 값을 반환 받아온다.

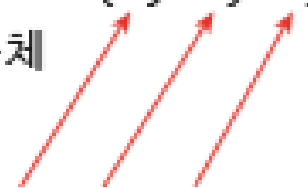
```
result1, result2 = get_root(1, 2, -8)  
print('해는', result1, '또는', result2)
```

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


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



[그림 4-10] 위치 인자의 전달 방식 : 매개변수에 전달할 값을 전달할 때 a, b, c 순서에 따라 전달하므로 순서가 중요함

```
result1, result2 = get_root(-8, 2, 1) # 1, 2, -8을 인자로 줄 때와 그 결과가 다름
```

실행결과

해는 -0.25 또는 0.5

```
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

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



[그림 4-11] 키워드 인자의 전달 방식 : a, b, c의 키워드를 통해서 매개변수에 전달할 값을 명시해 주므로 순서는 중요하지 않음

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

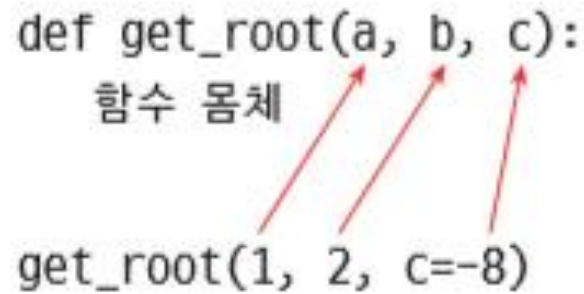
키워드 인자와 위치 인자를 섞어서 사용할 적에는 반드시 위치인자가 먼저 나타나야 한다(위의 경우는 오류)

SyntaxError: positional argument follows keyword argument

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

키워드 인자와 위치 인자를 섞어서 사용할 적에 위치인자를 먼저 적어주면 된다

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



[그림 4-12] 위치 인자와 키워드 인자의 혼용 : 1, 2는 a, b에 전달되고 -8은 키워드를 통해 명시해준 c에 전달됨



NOTE : 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. 그러나 두 가지 방식을 혼합하는 경우 **반드시** 위치 인자 뒤에 키워드 인자가 와야 한다.

```
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

코드 4-23 : 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))
```

실행결과

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

```
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

따라서 아래와 같이 수정할 것

```
print("circle area =", func(1, radius=5))
```



LAB 4-9 : 키워드 인자

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


코드 4-24 : 인자를 하나 가지는 함수

arg_greet1.py

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

실행결과

안녕하세요 홍길동 씨

코드 4-25 : 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?

4.6.3 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 4-26 : 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

코드 4-27 : 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

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

코드 4-28 : 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



LAB 4-10 : 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

4.7 재귀함수

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

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

코드 4-29 : 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

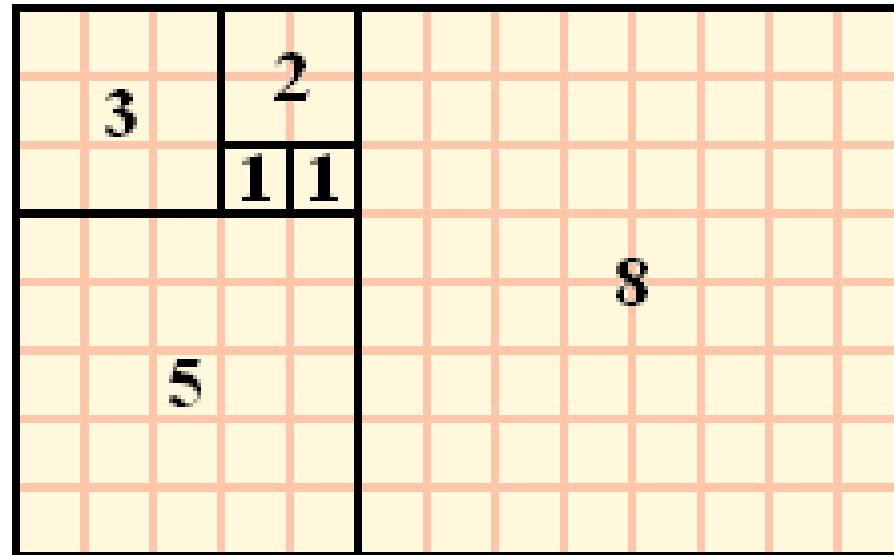
5! = 120

피보나치 수

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

$$F_n = F_{n-1} + F_{n-2}$$

- 이 정의에 따르면 피보나치 수열은 다음과 같다
 - 0, 1, 1, 2, 3, 5, 8, 13, 21,...



출처:위키백과

코드 4-30 : 재귀함수를 이용하여 정의한 피보나치 수열

fibonacci_recursion.py

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

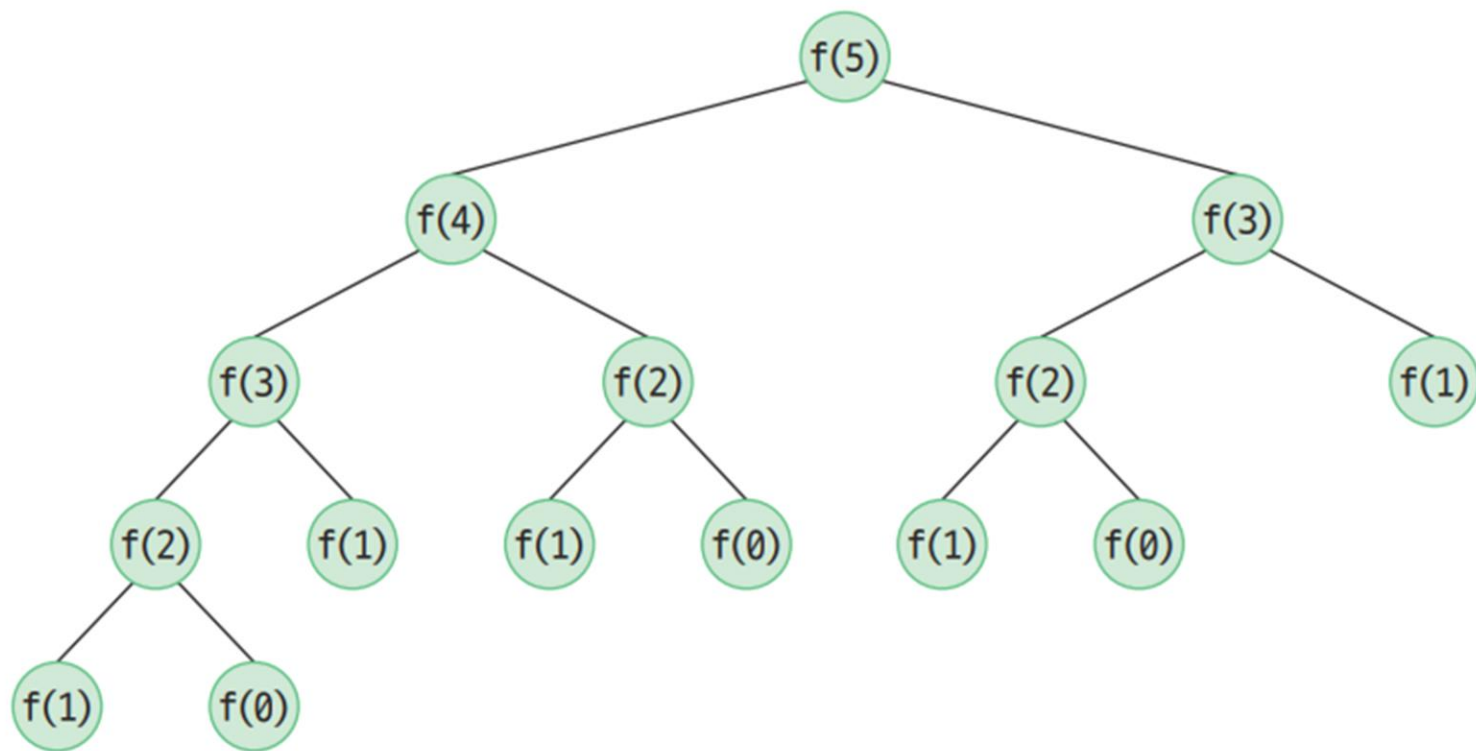
nterms = int(input("몇 개의 피보나치수를 원하세요? "))

# 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



[그림 4-13] fibonacci(5) 함수의 수행과정 : 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.

4.8 입력함수와 출력함수

- `input()` 함수
 - 사용자로부터 입력을 받기 위한 함수

코드 4-31 : `input()` 입력함수를 사용한 `name`의 입력 방법

`input_name1.py`

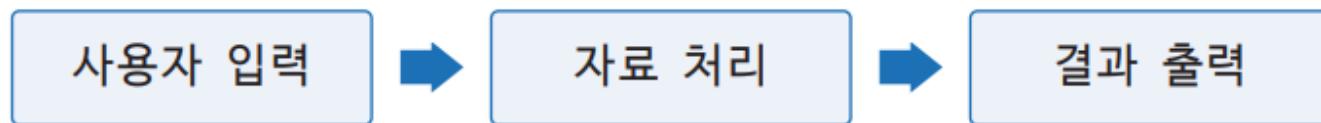
```
print('Enter your name : ')\nname = input()    # 사용자의 키보드 입력 값을 name이 참조함\nprint('Hello', name, '!')
```

실행결과

Enter your name :

Hong GilDong

Hello Hong GilDong !



[그림 4-14] 사용자 입력과 처리, 결과 출력을 가지는 프로그램의 흐름

코드 4-32 : 문자열을 이용한 input 입력함수의 사용법

input_name2.py

```
name = input('Enter your name : ')\nprint('Hello', name, '!')
```

실행결과

```
Enter your name : Hong GilDong\nHello Hong GilDong !
```

4.8.1 input() 함수와 int() 함수

대화창 실습 : 정수형과 문자열형의 덧셈, 서로 다른 자료형의 덧셈

```
>>> 100 + 1      # 정수의 덧셈으로 수+수
```

```
101
```

```
>>> '100' + '1'  # 문자열의 덧셈으로 문자열+문자열 연산을 통해 문자를 연결
```

```
'1001'
```

```
>>> '100' + 1    # 문자열과 정수의 덧셈(오류 발생)
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: must be str, not int
```

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

```
>>> int('100') + 1    # 문자열 '100' 의 덧셈을 위해 정수(int)로 변환
```

```
101
```

```
>>> float('100') + 1  # 문자열 '100'을 덧셈을 위해 실수(float)로 변환
```

```
101.0
```

```
>>> '100' + str(1)    # 문자열 덧셈을 위해서 1을 문자열 '1'로 변환
```

```
'1001'
```

대화창 실습 : 대화형 모드를 통한 int() 함수로 감싸야 정수값이 됨

```
>>> num1 = int(input("숫자를 입력하세요: "))
```

```
숫자를 입력하세요: 100
```

int() 함수로 감싸야 정수값이 됨

```
>>> num2 = int(input("숫자를 입력하세요: "))
```

```
숫자를 입력하세요: 200
```

```
>>> num3 = num1 + num2
```

```
>>> print("두 수의 합은", num3, "입니다.")
```

```
두 수의 합은 300 입니다.
```

- '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

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

```
>>> int('100') + 1
```

```
101
```

```
>> float('100') + 1
```

```
101.0
```

- 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

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

대화창 실습 : 공백 구분자를 사용한 input() 함수 실습

```
>>> 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)
```

대화창 실습 : TIP으로 다음과 같은 방법도 가능(10장의 내용)

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

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

대화창 실습 : 공백 구분자를 사용한 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)
```

4.8.2 여러가지 문자열 처리 메소드

대화창 실습 : 여러가지 문자열 메소드

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

```
'HELLO'
```

```
>>> 'HELLO'.lower()
```

```
'hello'
```

```
>>> 'Guido Van Rossum'.split()
```

```
['Guido', 'Van', 'Rossum']
```

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

```
['Apple', 'Banana', 'Orange']
```

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

```
['Apple', 'Banana', 'Orange', 'Kiwi']
```

구분자 `separator`

한 문자열을 하나 이상의 개별 문자열로 나누는 문자

디폴트 구분자는 공백

구분자로 쉼표를 사용

구분자로 세로바를 사용



NOTE : 함수와 메소드 method

함수와 메소드는 특정한 일을 하도록 정의된 동작을 수행한다는 점에서 동일하다. 그런데, 함수는 호출하여 매개변수를 전달하여 특정한 동작을 수행하고 필요한 경우 그 결과를 반환하는 모든 프로그래밍 모듈을 함수라고 부를 수 있는 반면, 메소드는 객체지향 프로그래밍에서 다루는 **객체object**에 소속되는 함수이다. 이 메소드는 자신을 가지는 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 소속된 인스턴스의 현재 데이터(멤버 변수)들에 접근하여 일을 하게 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 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 출현횟수
3
>>> s2.count('a')        # s2 인스턴스(객체)의 알파벳 'a'의 출현횟수
6
```

함수와 메소드 method

- 함수와 메소드는 특정한 일을 하도록 정의된 동작을 수행한다는 점에서 동일함
- 함수 - 호출될때 매개변수를 전달받아(생략가능) 특정한 동작을 수행하고 필요한 경우 그 결과를 반환하는 모든 프로그래밍 모듈
- 메소드 - 객체지향 프로그래밍에서 다루는 객체 object에 소속되는 함수
- 메소드는 인스턴스 instance라고 하는 각각 다른 값을 가진 객체들마다 호출할 수 있음
- 9장에서 상세히 다루게 됨

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

대화창 실습 : format() 메소드와 플레이스홀더의 인덱스

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

플레이스 홀더에 들어갈 내용

```
'Hello Python!'
```

플레이스 홀더

```
>>> '{} Python!'.format('Hi')
```

```
'Hi Python!'
```

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

```
'Hello Python!'
```

베이스 문자열, 템플릿 문자열

대화창 실습 : format() 메소드와 플레이스홀더

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

```
'I like Python and Java'
```

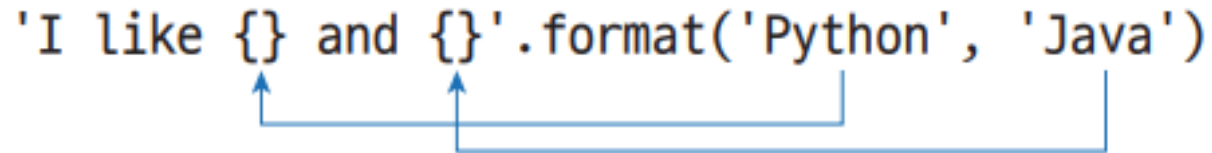
```
>>> 'I like {0} and {1}'.format('Python', 'Java')
```

```
'I like Python and Java'
```

인덱스에 따라 Python, Java가
각각 들어감

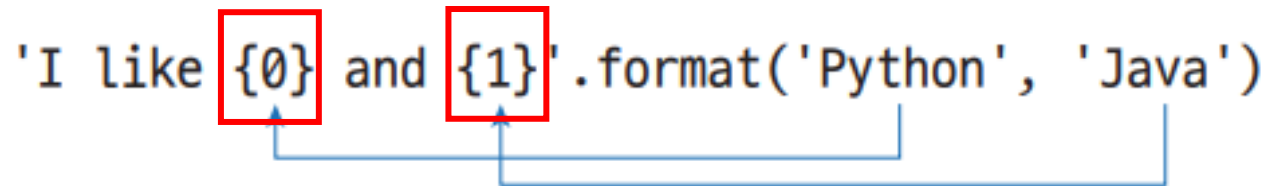
- 플레이스홀더 내에 필요한 정수 값(인덱스)을 할당하여 출력 순서를 제어할 수 있다(디폴트로 0, 1, .. 이 할당됨)

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



A diagram showing the mapping of arguments to placeholders. A blue line connects the first placeholder `{}` to the argument 'Python'. Another blue line connects the second placeholder `{}` to the argument 'Java'.

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



A diagram showing the mapping of arguments to indexed placeholders. The placeholder `{0}` is highlighted with a red box, and a blue line connects it to the argument 'Python'. The placeholder `{1}` is also highlighted with a red box, and a blue line connects it to the argument 'Java'.

[그림 4-15] 플레이스홀더와 인덱스를 이용한 출력제어

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



A diagram showing the mapping of arguments to placeholders in reverse order. The placeholder `{1}` is highlighted with a red box, and a blue line connects it to the argument 'Java'. The placeholder `{0}` is also highlighted with a red box, and a blue line connects it to the argument 'Python'.

[그림 4-16] 플레이스홀더와 전달인자

- {0}, {1}, {2}와 같이 플레이스홀더의 번호를 이용하여 다양한 출력력을 할 수 있다
- 플레이스홀더에는 문자열뿐만 아니라 100, 200과 같은 정수형이나 실수형등 임의의 자료형도 올 수 있음

대화창 실습 : format() 메소드와 플레이스홀더의 인덱스

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

```
'I like Java and Python'
```

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!'
```

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

대화창 실습 : 플레이스홀더 내의 객체 출력

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

코드 4-33 : 플레이스홀더와 format() 메소드의 사용

```
print_format1.py
name = 'Hong GilDong'
print('My Name is {}!'.format(name))
```

실행결과

```
My Name is Hong GilDong!
```

코드 4-34 : 플레이스홀더와 format() 메소드의 사용

print_format2.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.

- print() 함수 내부에 있는 {}의 의미는 {}가 있는 곳에 format() 함수의 인자 값 name을 출력하라는 뜻
- 사용자의 이름과 나이를 입력으로 받아서 name, age라는 이름의 변수에 저장한 후 format() 메소드를 이용하여 출력이 가능
- 문자열의 format() 메소드를 사용해 {} 필드에 각각 이름(name)과 나이(age), 직업(job)이 위치하도록 문자열 포매팅을 함
- format() 메소드를 사용하지 않는다면 출력문은 복잡한 형태로 나타남. 이 경우 쉼표가 너무 많고 따옴표도 많아서 코드를 읽기도 힘들고 오류가 날 가능성도 매우 높다

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



LAB 4-11 : format() 메소드의 활용

1. 위의 코드를 수정하여 이름, 나이, 직업, 사는 곳까지 다음과 같이 화면에 출력하려고 한다.

당신의 이름을 입력해주세요 : 김철수

나이를 입력해주세요 : 21

직업을 입력해주세요 : 학생

사는 곳을 입력해 주세요 : 창원시

당신의 이름은 김철수, 나이는 21살, 직업은 학생, 사는 곳은 창원시입니다.

- a) 출력문에서 format() 메소드를 사용하여 출력하여라.
- b) 출력문에서 format() 메소드를 사용하지 말고 쉼표로 구분하여 출력하여라.

4.9 고급 format() 메소드

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

대화창 실습 : 정수 표현을 위한 기본 포매팅

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

별도의 정렬기능이 없음

2 4 8

4 16 64

6 36 216

8 64 512

10 100 1000

12 144 1728

14 196 2744

보기 불편한 출력

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
```

오른쪽 정렬기능이 없음

오른쪽으로 정렬된 보기
좋은 출력

대화창 실습 : 소수점 아래 자리수를 조절하는 실수 포매팅

```
>>> print('소수점 두 자리로 표현한 원주율 = {0:10.2f}'.format(3.1415926))
```

소수점 두 자리로 표현한 원주율 = 3.14

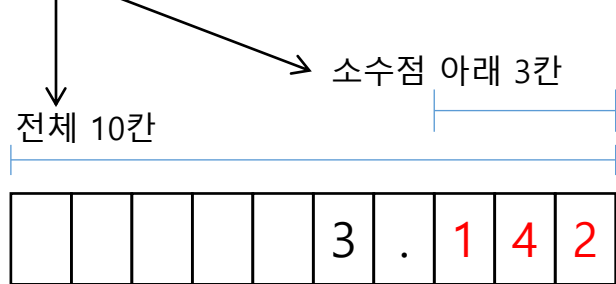
```
>>> print('소수점 세 자리로 표현한 원주율 = {0:10.3f}'.format(3.1415926))
```

소수점 세 자리로 표현한 원주율 = 3.142

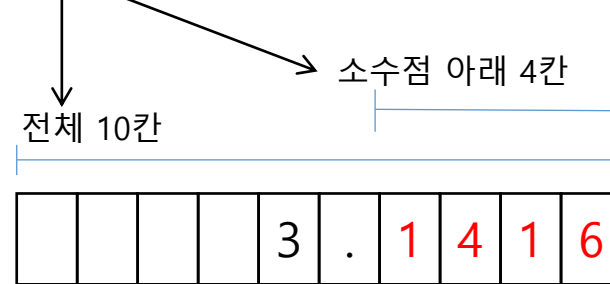
```
>>> print('소수점 네 자리로 표현한 원주율 = {0:10.4f}'.format(3.1415926))
```

소수점 네 자리로 표현한 원주율 = 3.1416

'{0:10.3f}'.format(3.1415926)



'{0:10.4f}'.format(3.1415926)



[그림 4-19] {0:10.3f}과 {0:10.4f} 포매팅 출력의 예

대화창 실습 : 실수 표현을 위한 포매팅에서 소수점

```
>>> print('1/3은 {:.3f}'.format(1/3))
```

소수점 아래 출력만 지정

1/3은 0.333

대화창 실습 : 1000 단위 쉼표 출력방법

```
>>> print('{:,}'.format(1234567890))
```

1000단위 쉼표 출력

1,234,567,890

- 플레이스홀더에 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

4.10 문자열의 다양한 메소드

- 특정한 객체 유형에 적용할 수 있는 기능을 **메소드** `method`라고 부름
- 문자열 자료형의 함수(메소드) - `format()`
- 파싱을 위한 `parse()`, `capitalize()`, `casefold()`, `count()`, `endswith()`, `expandtabs()`, `encode()`, `find()`, `index()`, `isalnum()`, `isdecimal()` 등

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' 문자의 위치를 반환한다.
```

```
2
```

- 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

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']
>>> 'Hello ' + 'Python!'        # + 연산자를 이용하여 문자를 연결함
'Hello Python!'
```

구분자separator

한 문자열을 하나 이상의 개별 문자열로 나누는 문자

여기서는 콜론 문자가 구분자임



LAB 4-12 : 문자열의 여러 메소드 활용

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.

- join() : 매개변수로 넘어온 문자열을 해당 문자를 구분자로 나누어서 반환
- rstrip() : 해당 문자열 오른쪽 끝에 있는 공백을 제거해 반환
- lstrip() : 해당 문자열의 왼쪽 끝에 있는 공백 제거해 반환
- strip() : 해당 문자열의 양쪽 끝에 있는 공백을 제거해 반환
- replace() : 해당 문자열 내에서 교환하고 싶은 문자열과 교환할 문자열을 각각 매개변수로 받아 원하는 문자열로 교환해 반환
- split() : 매개변수로 받은 문자를 구분자로 해당 문자열을 분리시켜 리스트로 반환

4.11 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 4-18] 입력에 대해 정해진 출력을 내보내는 블랙박스 역할을 하는 함수



[그림 4-19] 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

[표 4-2] 파이썬의 내장함수 목록

파이썬의 내장 함수				
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()	

대화창 실습 : 대화형 모드를 통한 여러 가지 내장 함수 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]
```

- `abs()` 함수는 -100이라는 정수를 입력 받아서 그 절댓값인 100을 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
- `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

- 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

대화창 실습 : 대화형 모드를 통한 id() 함수 실습

```
>>> a_str = "Hello Python!"
```

```
>>> id(a_str)
```

```
4549938992
```

```
>>> n = 300
```

```
>>> id(n)
```

```
4549735536
```

- 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'>
```

- 인자로 들어온 string 값을 그대로 파이썬 인터프리터가 실행하여 그 결과를 출력하는 eval() 함수
- eval() 함수는 스크립트 자체를 수행하는 등 기능이 매우 강력
- 시스템의 명령을 직접 호출하여 파이썬이 수행되는 시스템의 모든 파일의 정보를 얻거나 파일을 삭제하는 등의 기능까지 수행할 수 있으므로 **주의해서 사용할 것**

대화창 실습 : 수식을 가진 문자열과 eval() 함수

```
>>> eval('10 + 20 ')          # 10 + 20 문장을 파이썬 번역기가 수행함
30

>>> eval('(5 * 20) / 2')      # (5 * 20) / 2 문장을 파이썬 번역기가 수행함
50.0

>>> chr(65)                   # 유니코드 값 65는 알파벳 'A'이며, chr() 함수는 이를 반환함
'A'

>>> ord('A')                  # 알파벳 'A'의 유니코드 값 65를 반환함
65
```



Questions?