Python

함수와 입출력

Spring 2025

AI융합학과
Seongbok Baik
sbbaik@dju.kr

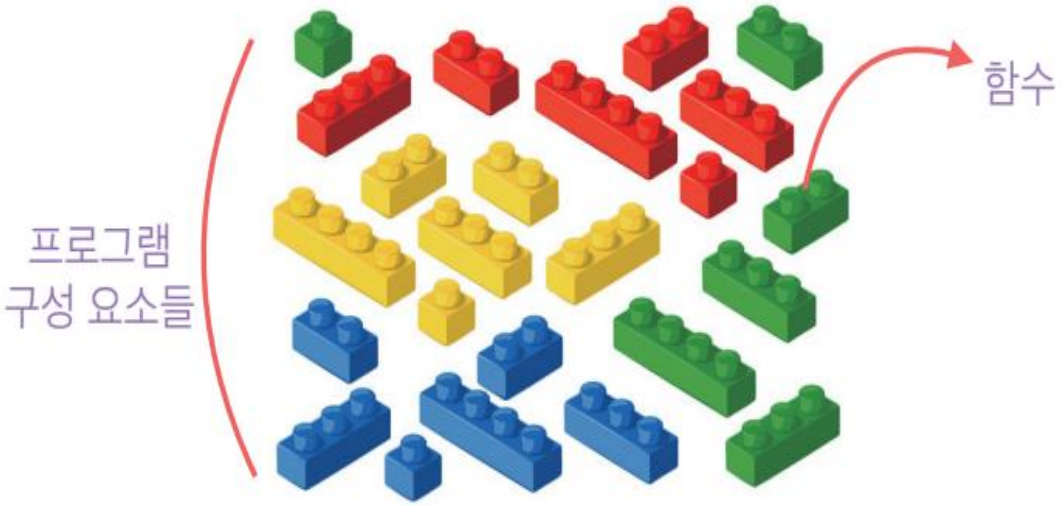
00 Text Book

교재명	으뜸 파이썬
저자	박동규, 강영민
출판사	생능출판사
발행년	2024.06.14

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.
- 지역변수와 전역변수를 올바르게 사용할 수 있다.
- 함수의 반환문에 대해 이해하고 그 필요성을 설명할 수 있다.
- 다중 반환문을 사용하여 여러 개의 값을 반환할 수 있다.
- 출력 함수와 입력 함수의 목적과 사용 방법을 설명할 수 있다.
- input() 함수를 이용한 입력 방법에 대해 이해하고 사용할 수 있다.
- format() 메소드와 플레이스 홀더를 이용한 출력을 할 수 있다.
- 파이썬의 다양한 내장 함수를 사용할 수 있다.

5.1 함수의 역할



[그림 5-1] 프로그램의 구성 요소와 함수와의 관계

[그림 5-2] 레고 블록을 조립해서 만든 자동차(출처: bricklink.com)

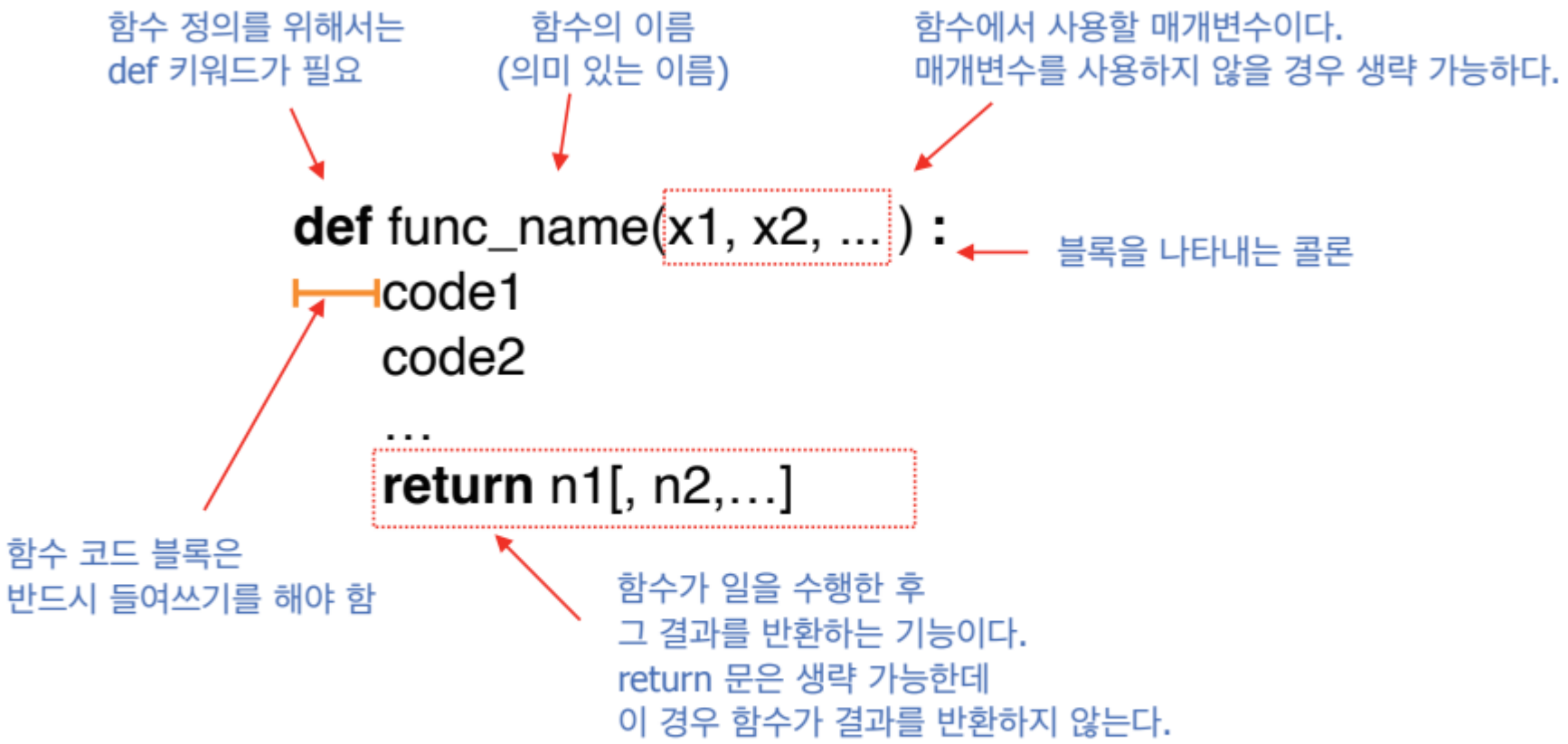
여러분이 사용하는 프로그램도 많은 부품(함수나 클래스)으로 이루어져 있습니다

(출처: bricklink.com)

5.1 함수의 역할

- 반복적으로 사용되는 코드 - 덩어리(혹은 **블록**block)라고 함
- 기능에 따라 미리 만들어진 블록은 필요할 때 **호출**function call함
- 파이썬에서 미리 만들어서 제공하는 함수는 인터프리터에 포함되어 배포되는데 이러한 함수를 **내장함수**built-in function라고 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

5.1 함수의 역할



[그림 5-3] 파이썬에서 함수를 정의하는 문법

5.1 함수의 역할

- return문이 없는 간단한 코드로 함수를 정의하고 호출하기

코드 5-1: 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

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

5.1 함수의 역할

코드 5-2: 별표 출력을 위한 함수 정의와 반복 호출

print_star_4.py

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

print_star() # 별표 출력 함수 호출 1
print_star() # 별표 출력 함수 호출 2
print_star() # 별표 출력 함수 호출 3
print_star() # 별표 출력 함수 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.

5.1 함수의 역할

LAB 5-1: 함수 정의와 호출

1. **[코드 5-1]**의 함수 호출문을 삭제하면 어떻게 되는가?

2. **[코드 5-2]**를 수정하여 6줄의 별표를 출력해 보시오. 이때 함수 호출을 6회 하시오.

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

5.1 함수의 역할

코드 5-3: 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

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

5.1 함수의 역할

LAB 5-2: 함수 정의와 여러 번 호출하기

1. **[코드 5-3]**을 수정하여 함수 호출 두 번으로 10줄의 별표를 출력해 보시오.

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

5.1 함수의 역할

코드 5-4: 별표 출력과 더하기 기호 출력을 번갈아 호출하기
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.

5.1 함수의 역할

LAB 5-3: 여러 함수 정의와 호출

1. **[코드 5-4]**를 수정하여 해시마크(#)를 한 줄 출력하는 `print_hash()` 함수를 추가로 구현하시오.

2. `print_star()`, `print_plus()`, `print_hash()` 함수를 모두 이용하여 다음과 같은 출력이 나타나도록 함수를 호출하시오.

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

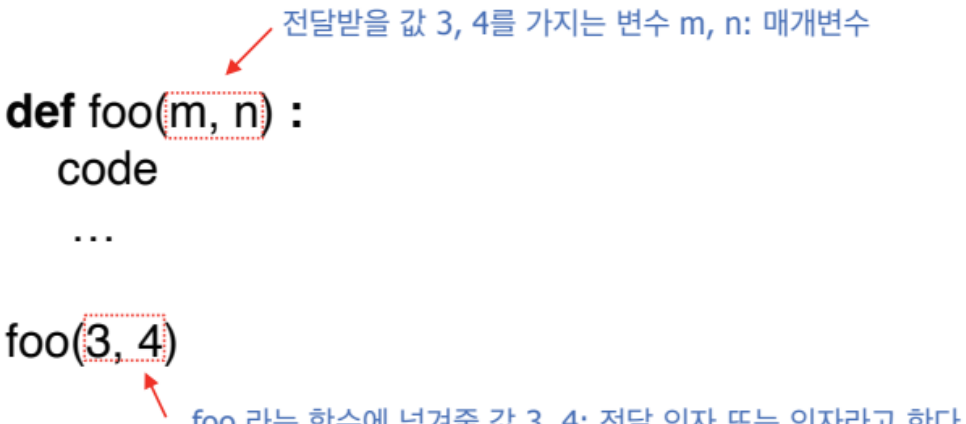
5.1 함수의 역할

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.

5.2 함수와 매개변수



[그림 5-4] 매개변수와 인자의 개념과 사용 방법

NOTE: 인자와 매개변수

- **매개변수**parameter: 함수나 메소드 헤더부에 정의된 변수로 함수가 호출될 때 실제 값을 전달받는 변수이다.
  예: `def foo(m, n):`의 `m`과 `n`
- **인자**argument: 함수나 메소드가 호출될 때 전달되는 실제 값을 말하며, **전달 인자**라고도 한다.
  예: `foo(3, 4)`의 `3`과 `4`

5.2 함수와 매개변수

복습 : 코드 5-2

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

print_star() # 별표 출력 함수 호출 1
print_star() # 별표 출력 함수 호출 2
print_star() # 별표 출력 함수 호출 3
print_star() # 별표 출력 함수 호출 4
```

별표를 출력하는 함수를 4번 호출한다.
이 출력 줄의 수를 효과적으로 제어할 수 있는 방법이 없을까?

5.2 함수와 매개변수

코드 5-5: 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

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

인자를 사용하자.
이 출력 줄의 수를 효과적으로 제어할 수 있다.

5.2 함수와 매개변수

LAB 5-4: 다양한 별표와 패턴 출력

1. **[코드 5-5]**를 수정하여 별표(*)줄이 10개 출력되도록 인자를 변경하시오.
2. **[코드 5-5]**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 ##################
```

5.2 함수와 매개변수

코드 5-6: 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.')
print_hello(2)          # 전달 인자로 2를 넘겨준다.
print('Hello를 세 번 출력합니다.')
print_hello(3)          # 전달 인자로 3을 넘겨준다.
print('Hello를 네 번 출력합니다.')
print_hello(4)          # 전달 인자로 4를 넘겨준다.
```

인자가 2이면 Hello를 2회 출력

인자가 3이면 Hello를 3회 출력

인자가 4이면 Hello를 4회 출력

실행결과

```
Hello를 두 번 출력합니다.
Hello Hello
Hello를 세 번 출력합니다.
Hello Hello Hello
Hello를 네 번 출력합니다.
Hello Hello Hello Hello
```

5.2 함수와 매개변수

코드 5-7: 매개변수를 사용한 인자값의 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 호출 시에 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!! 전달 인자가 하나밖에 없음
```

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

5.2 함수와 매개변수

LAB 5-5: 두 수를 전달받아 연산을 수행하는 함수

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

```
10과 20의 차는 -10입니다.
```

2. 두 개의 매개변수 a, b를 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.
```

5.3 매개변수를 활용한 2차 방정식의 근 구하기

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

[수식 5-1] 미지수 x에 대한 2차 방정식

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

[수식 5-2] 2차 방정식의 근의 공식

5.3 매개변수를 활용한 2차 방정식의 근 구하기

코드 5-8: 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 != 0), b, c를 해당하는 값을 방정식에 맞게 입력
- a, b, c에 해당하는 해를 변수 r1, r2에 저장하여 출력

5.3 매개변수를 활용한 2차 방정식의 근 구하기

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

코드 5-9: 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
```

5.3 매개변수를 활용한 2차 방정식의 근 구하기

- 문제점
  - 변수 a, b, c에 원하는 계수를 입력하고, 다시 r1, r2의 수식을 구해 줘야 함
  - 복사, 붙여넣기를 한다 해도 코드가 중복되는 부분이 많고 불필요하게 긴 것을 한눈에 알 수 있다.

5.3 매개변수를 활용한 2차 방정식의 근 구하기

코드 5-10: 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
```

5.3 매개변수를 활용한 2차 방정식의 근 구하기

LAB 5-6: 함수의 사용

1. 다음 2차 방정식의 근을 **[코드 5-10]**의 함수를 사용하여 출력하여라.
   (1) $x^2 + 4x - 21 = 0$
   (2) $x^2 - 6x + 8 = 0$

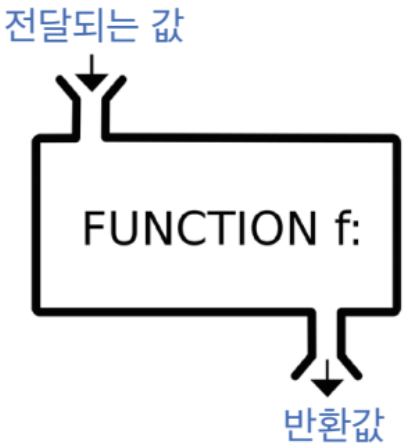
2. 삼각형의 면적을 구하기 위하여 밑변과 높이를 사용하려고 한다. 두 개의 매개변수 `width`, `height`를 받아서 삼각형의 면적을 출력하는 함수 `print_area(width, height)`를 구현하여라. 이때 `print_area(10, 20)`을 호출하여 다음과 같은 출력이 나타나도록 하여라.

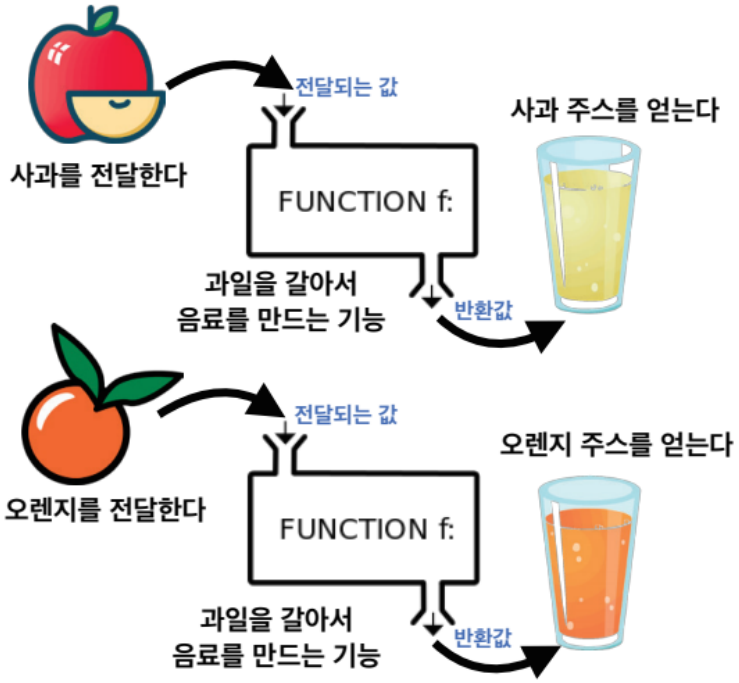
```
print_area(10, 20) # 삼각형의 면적을 구하는 함수 호출
```

실행결과

```
밑변 10, 높이 20인 삼각형의 면적은 : 100
```

5.4 return을 이용한 결과값 반환과 튜플





[그림 5-5] 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 5-6] 함수의 구성과 반환문 return

5.4 return을 이용한 결과값 반환과 튜플

반환문 return

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

5.4 return을 이용한 결과값 반환과 튜플

반환문 return

코드 5-11: 두 값의 합을 반환하는 get_sum() 함수와 return 문의 사용 1

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
```

반환문 return

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

코드 5-12: 두 값의 합을 반환하는 get_sum() 함수와 return 문의 사용 2
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))
```

두 가지 방법의 결과는 동일함

5.4 return을 이용한 결과값 반환과 튜플

반환문 return

코드 5-13: 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

# 함수 호출 시 1, 2, -8 인자를 사용함
# result1, result2를 이용해서 결과 값을 반환 받는다.
result1, result2 = get_root(1, 2, -8)
print('해는', result1, '또는', result2)
```

두 근 r1, r2를 반환함

result1, result2가 r1, r2의 값을 받아온다

실행결과

```
해는 2.0 또는 -4.0
```

5.4 return을 이용한 결과값 반환과 튜플

LAB 5-7: 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
```

실행결과

```
반지름 10인 원의 면적은 314.0, 원의 둘레는 62.8
```

5.4 return을 이용한 결과값 반환과 튜플

- 위의 코드에서는 계수 a, b, c를 매개변수로 받아서 근의 공식으로 해를 계산해 변수 r1, r2에 저장, return문으로 두 값을 반환
- 함수 외부에서는 함수 get_root를 호출해 그 해를 변수 result1, result2에 저장하여 출력
- 이와 같이 두 개 이상의 값을 반환하는 반환문을 **다중 반환문**multiple return statement이라고 함
- 다중 반환문에서 쉼표로 구분되는 두 개의 값은 튜플 형으로 반환이 이루어짐

5.4 return을 이용한 결과값 반환과 튜플

LAB 5-8: 다수의 결과를 반환하는 함수 만들기

1. 어떤 정수 n과 m을 입력하면, n의 배수 m개를 반환하는 multiples(n, m) 함수를 구현하고 다음과 같이 호출하여 결과를 출력하여라.

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

실행결과

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

5.5 전역 변수

전역변수global variable

- 함수 바깥에서 선언되거나 전체 영역에서 사용 가능한 변수

코드 5-14: 매개변수를 사용하지 않고 외부 변수를 사용하는 경우

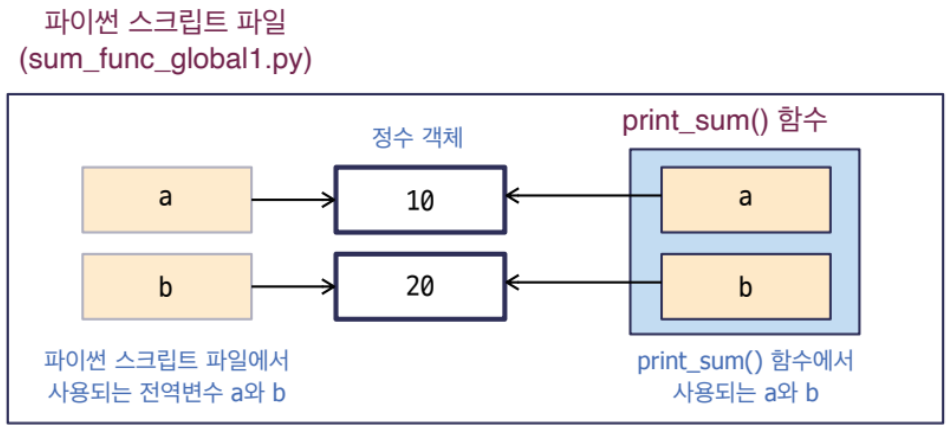
sum_func_global1.py

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

a = 10
b = 20
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.
```

5.5 전역 변수



[그림 5-7] 파이썬 스크립트 파일과 전역변수, 그리고 이 전역변수를 사용하는 print_sum() 함수

코드 5-15: 함수 외부에서 정의된 값을 함수 내부에서 변경하는 경우

sum_func_global2.py

```
def print_sum():
    a = 100       # 새로운 객체 100이 생성되고 a가 이것을 참조한다.
    b = 200       # 새로운 객체 200이 생성되고 b가 이것을 참조한다.
    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.
```

5.5 전역 변수

코드 5-16: 함수 내부에서 값을 변경하고, 그 값을 외부에서 확인하기

sum_func_global3.py

```
def print_sum():
    a = 100     # 새로운 객체 100이 생성되고 a가 이것을 참조한다.
    b = 200     # 새로운 객체 200이 생성되고 b가 이것을 참조한다.
    result = a + b # 함수 내부에서 a와 b를 더하면 지역변수의 값이 더해진다.
    print('print_sum() 내부 :', a, '과', b, '의 합은', result, '입니다.')

a = 10
b = 20
print_sum()
result = a + b     # 함수 외부에서 a와 b를 더하면 전역변수의 값이 더해진다.
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

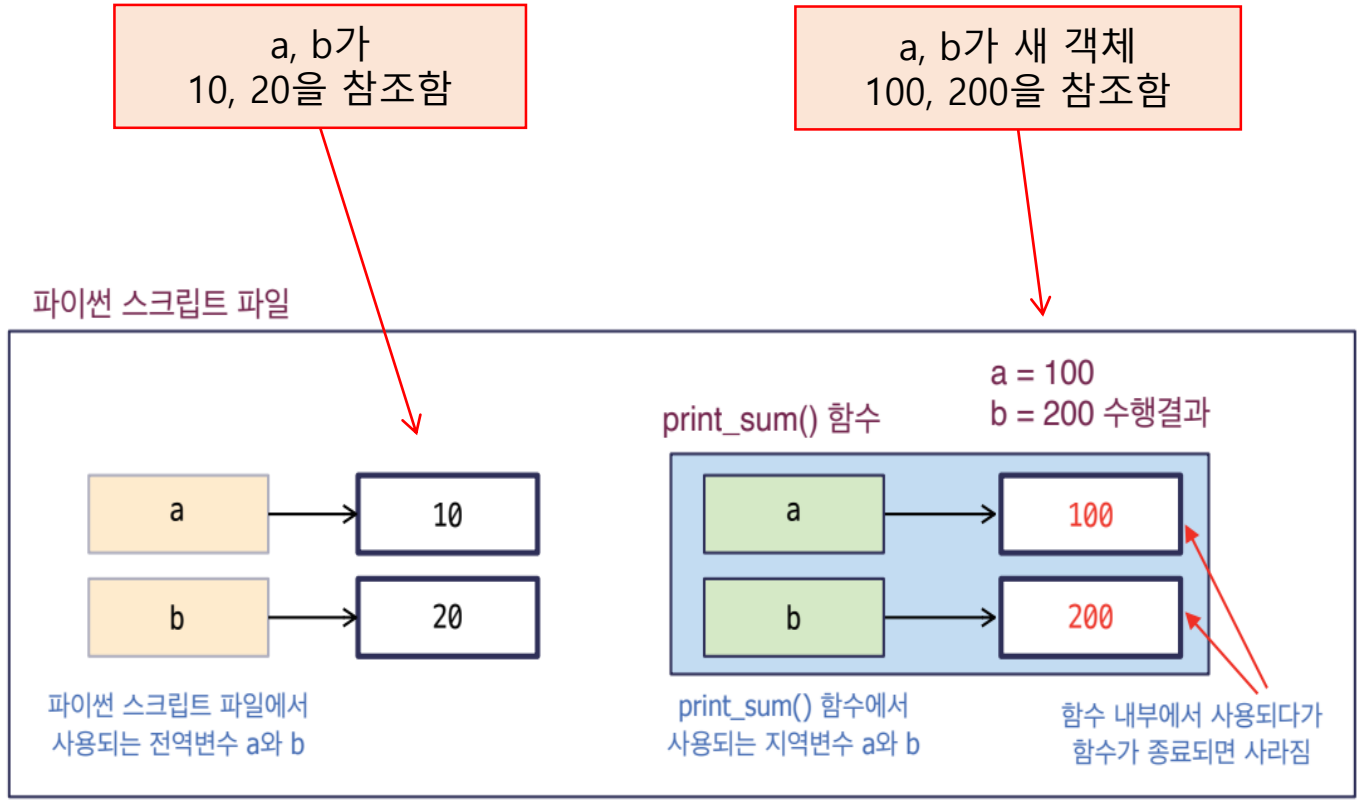
실행결과

```
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

5.5 전역 변수

- 할당assign
  - a = 100, b = 200
- 지역 변수local variable
- 참조reference



[그림 5-8] 파이썬 스크립트 파일과 전역변수 a, b 그리고 지역변수를 사용하는 print_sum() 함수. 이 함수 내부에서 a = 100, b = 200과 같은 할당 연산을 하면 a, b는 지역변수가 되어 함수 내부에서 생성된 100과 200을 참조하며, 스크립트 파일의 a, b와 별개의 변수가 된다

5.5 전역 변수

코드 5-17: global 키워드를 사용한 전역변수의 참조 방법

sum_func_global4.py

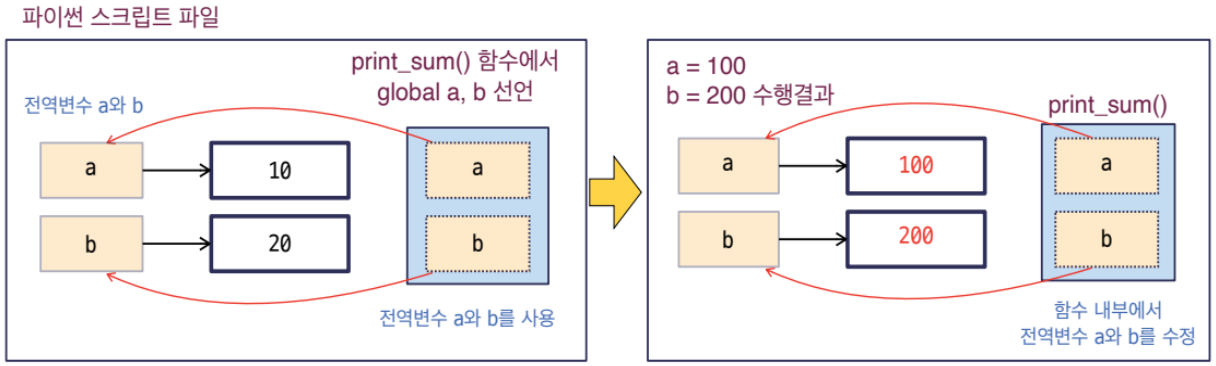
```
def print_sum():
    global a, b # a, b는 함수 외부에서 선언된 a, b를 사용한다.
    a = 100
    b = 200
    result = a + b
    print('print_sum() 내부 :', a, '과', b, '의 합은', result, '입니다.')

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

전역변수 a, b가 100, 200을 참조함

```
global a = 100 # 문법 오류 발생
```

실행결과

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



[그림 5-9] 명시적 global 선언을 통하여 함수 내에서 전역변수 a, b를 사용하는 과정

5.5 전역 변수

주의 - 전역변수와 전역상수

전역변수를 사용하는 것은 파이썬뿐만 아니라 모든 프로그래밍 언어에서 매우 나쁜 습관이다. 특히, 코드의 길이가 길어질 경우 전역변수는 에러의 주요 원인이 된다. 그러나 **전역상수**global constant의 경우는 반드시 나쁘다고 볼 수 없다. 전역상수는 다음과 같이 global이라는 키워드로 선언하는데, 함수의 외부에서 선언해서 모듈 전체에서 참조할 수 있다. 전역 상수값은 일반적으로 대문자를 사용한다. 아래의 코드를 살펴보면, GLOBAL_VALUE라는 이름의 변수에 1024라는 값을 할당한 후 foo() 함수에서 이 변수 값을 불러서 사용하기 위해 global GLOBAL_VALUE라는 이름으로 선언했다.

전역상수의 예:

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

수학 연산을 위해 사용되는 math 모듈의 경우 원주율 pi와 오일러 상수 e를 프로그램 전체에서 참조하여 사용하는데, 이러한 상수 값의 경우는 예외적으로 소문자로 표기한다.

Leistung ist nicht alles / Keinen Studierenden zurücklassen

DAEJEON UNIVERSITY
SINCE 1980
대전대학교