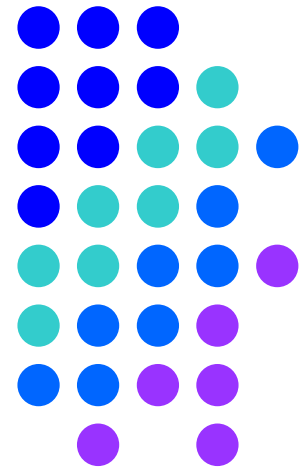


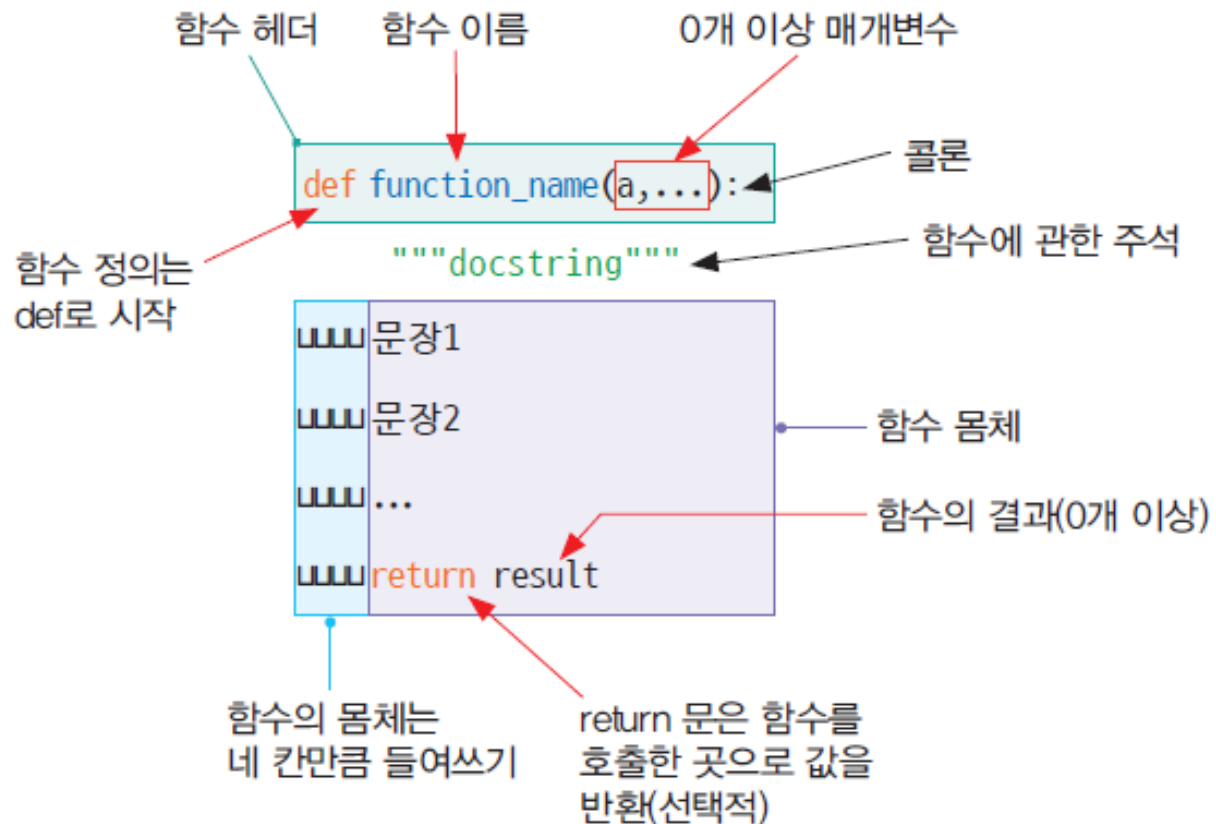
4주차. 함수

- 6.2 함수 정의
- 6.3 함수 호출
- 6.4 지역 변수
- 6.5 전역 변수
- 6.7 순환 함수



6.2 함수 정의

형식



[그림 6-5] 함수 정의문의 형식

6.2 함수 정의

■ 함수 정의문

- 함수 헤더

 - ◆ 키워드 `def`로 시작

 - ◆ 그 다음에 함수 이름, 매개변수 리스트, 콜론

- 함수 이름: 변수 이름을 정하는 규칙과 동일

- 독스트링

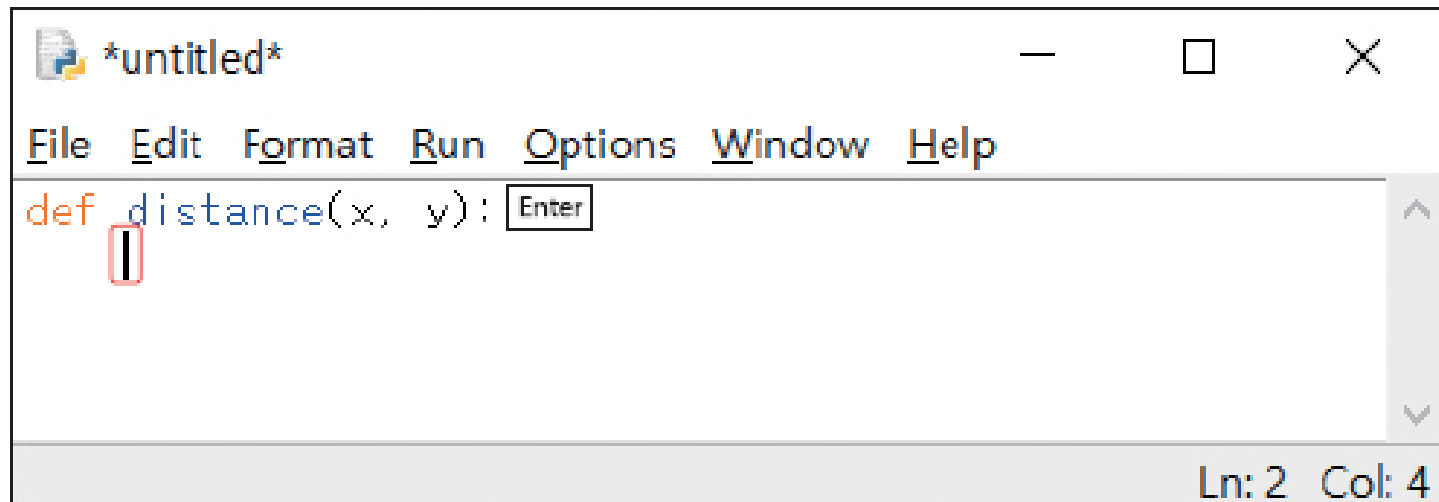
 - ◆ 함수의 헤더 밑에 함수가 어떤 일을 수행하는가에 관한 주석

 - ◆ 삼중 따옴표

- 함수 몸체: 함수를 구성하는 문장(들)

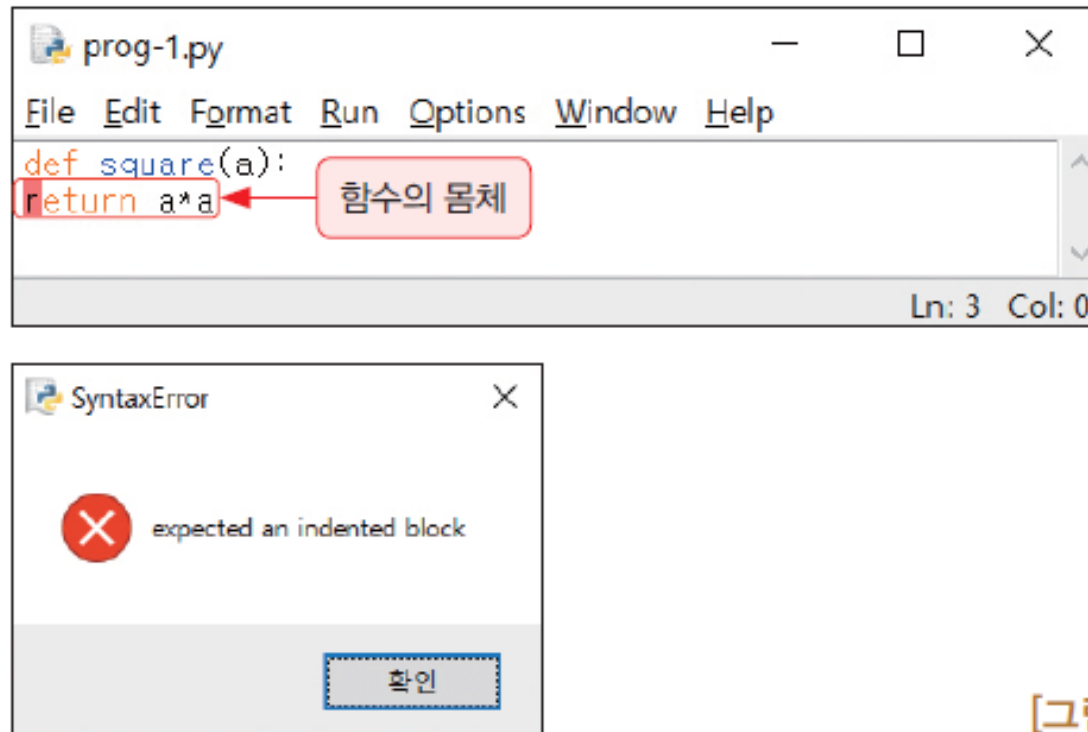
- `return` 문: 함수의 실행이 종료되면서 함수 호출문으로 결과를 반환

6.2 함수 정의



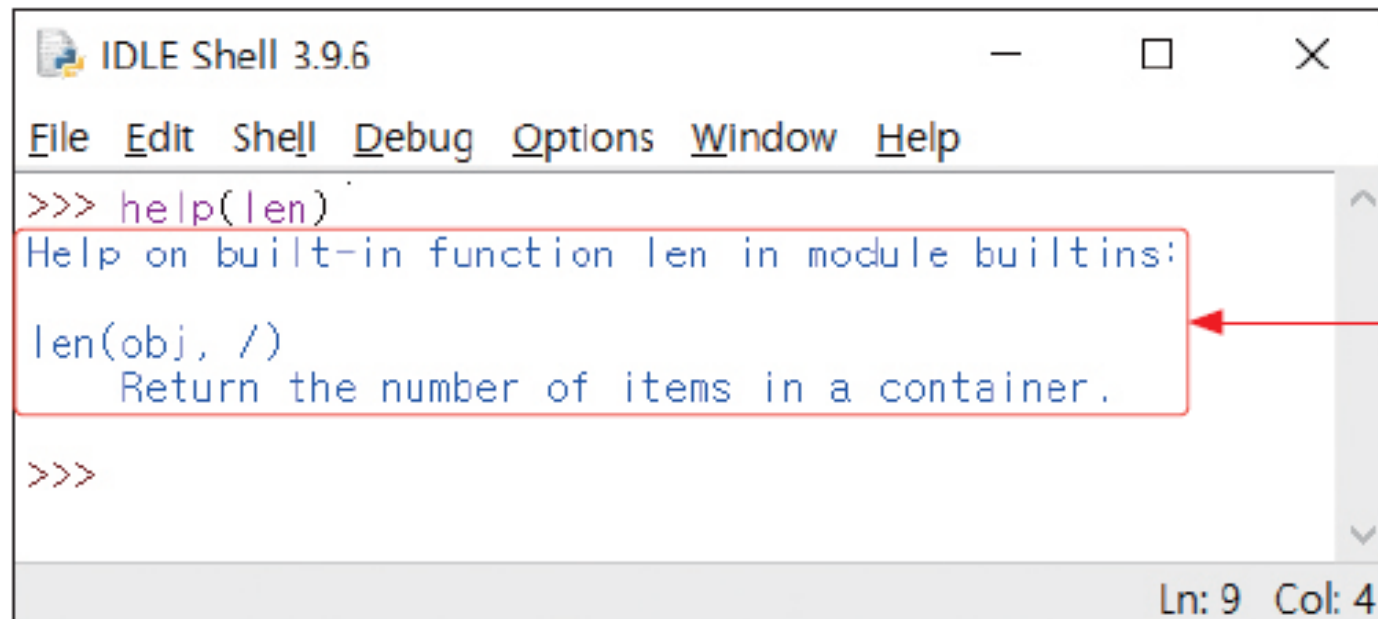
[그림 6-6] 편집기 창에서 함수의 헤더 입력

6.2 함수 정의



[그림 6-7] 들여쓰기 무시

6.2 함수 정의



The screenshot shows the IDLE Shell 3.9.6 window. The menu bar includes File, Edit, Shell, Debug, Options, Window, and Help. The command prompt shows the user entering `>>> help(len)`. The output displays the help text for the built-in function `len`: `Help on built-in function len in module builtins:`, followed by the function signature `len(obj, /)` and its description: `Return the number of items in a container.`. A red box highlights this help text, and a red arrow points from a callout box to it. The status bar at the bottom right indicates `Ln: 9 Col: 4`.

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>> help(len)
Help on built-in function len in module builtins:
len(obj, /)
    Return the number of items in a container.
>>>
```

Ln: 9 Col: 4

내장 함수 len의
독스트링

[그림 6-8] 내장 함수 len의 독스트링

6.2 함수 정의

■ 함수 작성 예

- 원점 (0, 0)에서 좌표 (x, y)까지의 거리를 계산하는 함수

$$\sqrt{(x-0)^2 + (y-0)^2}$$

```
def distance(x, y):  
    """좌표 (0, 0)에서 (x, y)까지의 거리를 계산"""  
    a = x**2  
    b = y**2  
    c = (a + b) ** 0.5  
    return c
```

함수 이름 매개변수 독스트링

6.2 함수 정의

함수_1

...

함수_2

...

...

함수_n

...

메인 프로그램

[그림 6-9] 함수(들)을 포함한 프로그램의 일반적인 구조

6.3 함수 호출

■ 함수 호출

- 정의한 함수를 실행하려면 함수를 호출해야 함
- 함수를 호출하면 해당 함수의 헤더로 제어가 넘어가서 그 함수의 몸체를 구성하는 문장들을 실행하고, 그 함수의 끝에 도달하거나 return 문을 만나면 함수를 호출했던 곳으로 되돌아와서 다음 문장들을 실행
- 인수: 함수 호출문의 () 안에 사용된 식(또는 변수, 값)
- 매개변수: 피호출 함수 헤더의 () 안에 사용된 변수
- 정수, 실수, 문자열 등의 불변 데이터는 값의 복사(call by value) 방식으로 전달
- 리스트 등의 가변 데이터는 주소 복사(call by reference) 방식으로 전달

6.3 함수 호출

함수를 호출하는 두 가지 형식

- 함수로부터 값을 반환받지 않을 경우
함수(인수_리스트)
- 함수로부터 값을 반환받을 경우(배정문)
변수 = 함수(인수_리스트)

6.3 함수 호출

- 함수를 포함한 프로그램의 실행 과정
 - 정의한 함수를 실행하려면 함수를 호출해야 함

```
def distance(x, y):  
    a = x**2  
    b = y**2  
    c = (a + b) ** 0.5  
    return c
```

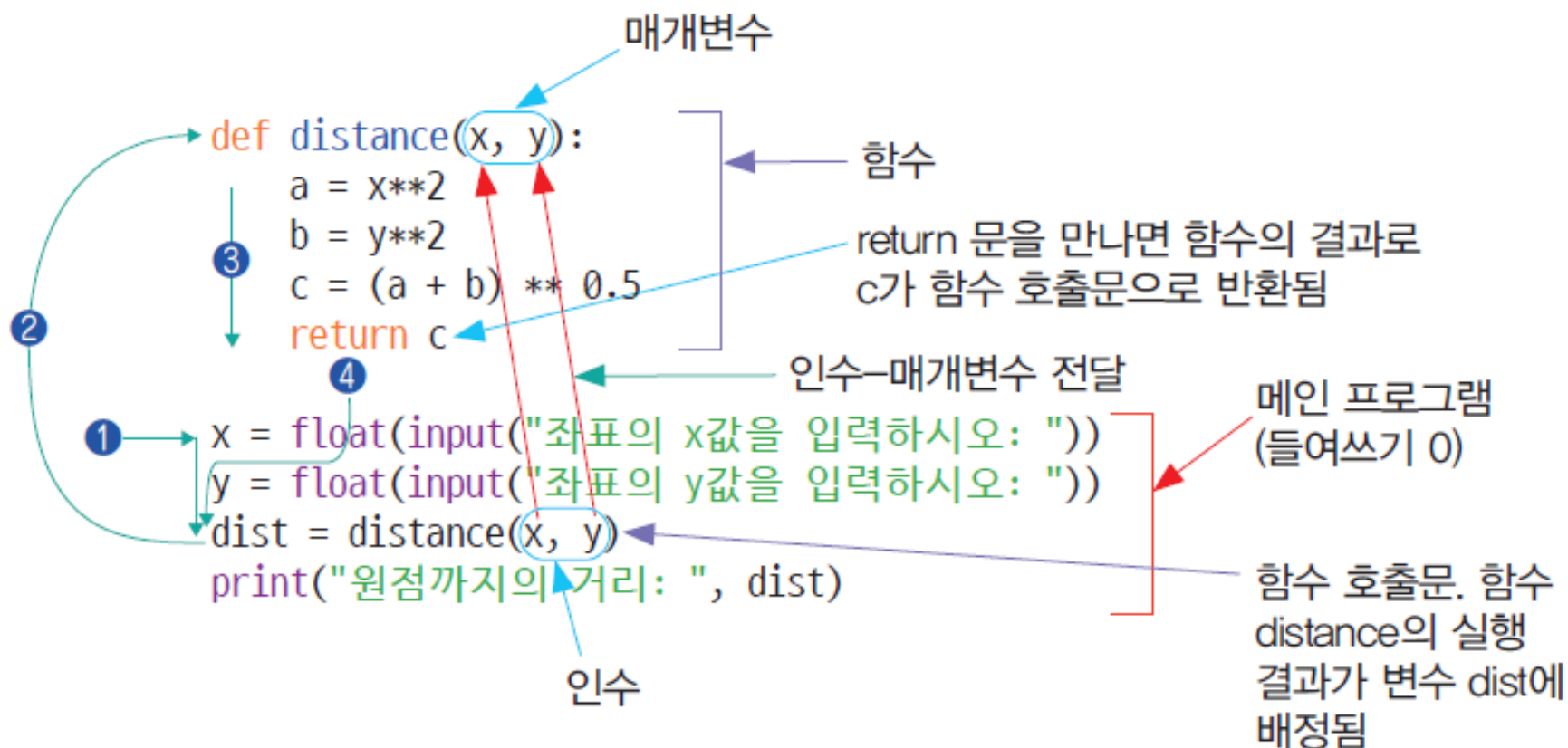
← 함수

```
x = float(input("좌표의 x값을 입력하시오: "))  
y = float(input("좌표의 y값을 입력하시오: "))  
dist = distance(x, y)  
print("원점까지의 거리 : ", dist)
```

← 메인 프로그램

[그림 6-10] 함수와 메인 프로그램

6.3 함수 호출



[그림 6-11] 함수를 포함한 프로그램이 실행되는 과정

6.3 함수 호출

```
>>> distance(1 + 3, 9 - 6)  # 1 + 3과 9 - 6은 식  
5.0
```

```
distance(1 + 3, 9 - 6)  
→ distance(4, 3)  
→ 5.0
```

6.3 함수 호출

■ return 문

- return 문은 어떤 내용도 화면에 출력하지 않은 채 함수 호출문으로 값을 반환
- 하나의 함수 안에 여러 개의 return 문이 포함될 수 있음
- return 문이 함수의 마지막 문장일 필요는 없음
- return문의 포함 여부는 선택사항
- return 문을 포함하지 않은 함수는 함수 호출문으로 None을 반환

return 문의 형식

- return 식

식의 계산 결과는 하나의 값이 되고, 함수를 호출한 곳으로 이 값이 반환된다. 식 자리에 임의의 변수, 값이 올 수 있다.

- return

함수를 호출한 곳으로 반환되는 값은 None이다.

6.3 함수 호출

예 return 문

```
def adder(x, y):  
    z = x + y  
    return z
```

예 반환받은 값이 식에 사용됨

```
3 * distance(1 + 3, 9 - 6) - 5  
→ 3 * distance(4, 3) - 5  
→ 3 * 5 - 5  
→ 15 - 5  
→ 10
```

6.3 함수 호출

예 반환된 값을 출력하거나 변수에 저장

```
>>> print(distance(6, 8))  
10.0  
>>> dist = distance(12, 9)  
>>> print(dist)  
15.0
```


6.3 함수 호출

예 함수 안에서 print 문을 사용하고, return 문을 사용하지 않음

편집기 창

```
def rec_area(length, width):  
    print(length * width)  
  
length = 6  
width = 7  
area = rec_area(length, width)  
print("직사각형의 면적: ", area)
```

Ln: 7 Col: 25

IDLE 셸 창

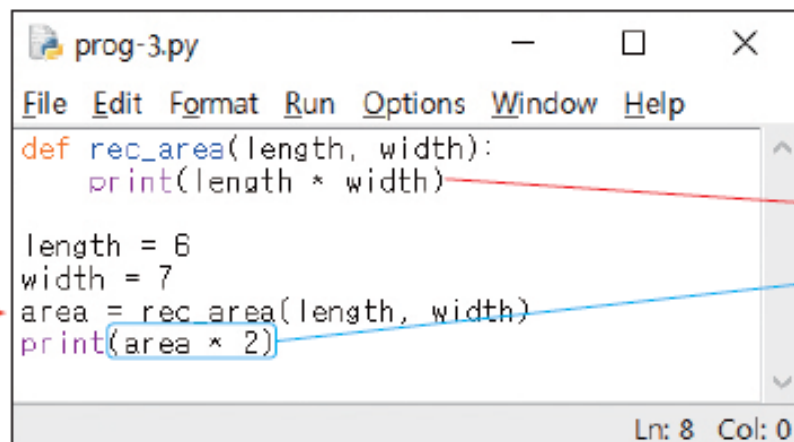
```
IDLE Shell 3.9.6  
File Edit Shell Debug Options Window Help  
>>>  
===== RESTART: 0:/파이썬 소스/6장/p  
rog-2.py =====  
42  
직사각형의 면적: None  
>>>
```

Ln: 7 Col: 4

[그림 6-12] return 문이 없는 함수에서 None이 반환됨

6.3 함수 호출

편집기 창



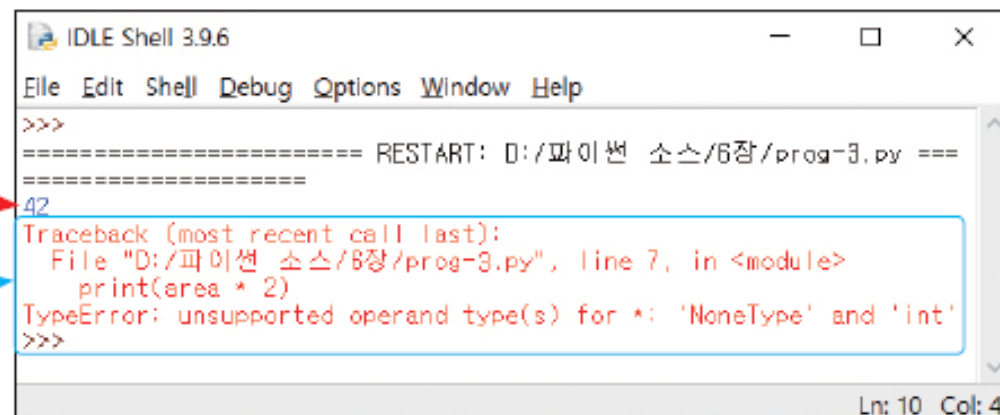
```
File Edit Format Run Options Window Help
def rec_area(length, width):
    print(length * width)

length = 6
width = 7
area = rec_area(length, width)
print(area * 2)
```

Ln: 8 Col: 0

area에 None이 지정됨

IDLE 셸 창



```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/파이썬 소스/8장/prog-3.py =====
42
Traceback (most recent call last):
  File "D:/파이썬 소스/8장/prog-3.py", line 7, in <module>
    print(area * 2)
TypeError: unsupported operand type(s) for *: 'NoneType' and 'int'
>>>
```

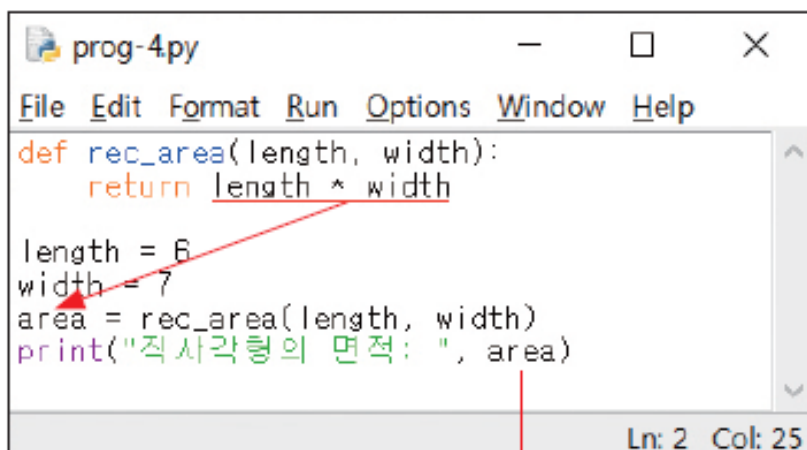
Ln: 10 Col: 4

[그림 6-13] 메인 프로그램에서 None에 곱셈 연산을 실행

6.3 함수 호출

예 함수에서 return 문을 사용하여 결과를 반환함

편집기 창

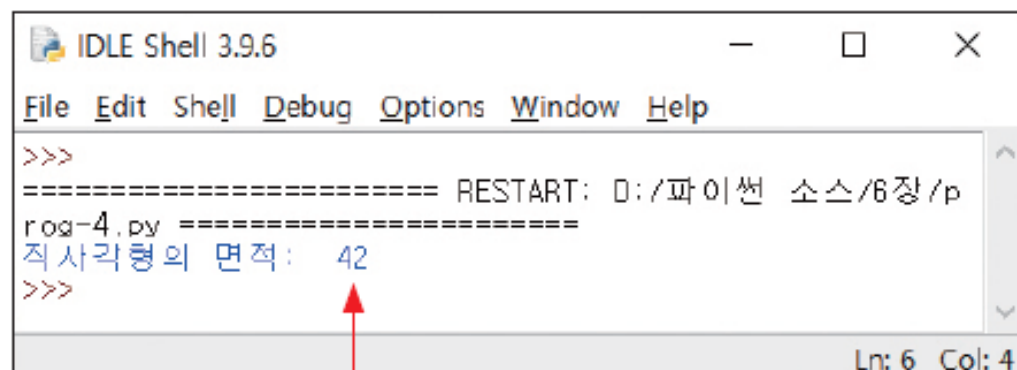


```
File Edit Format Run Options Window Help
def rec_area(length, width):
    return length * width

length = 6
width = 7
area = rec_area(length, width)
print("직사각형의 면적: ", area)
```

Ln: 2 Col: 25

IDLE 셸 창



```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/파이썬 소스/6장/p
rog-4.py =====
직사각형의 면적: 42
>>>
```

Ln: 6 Col: 4

[그림 6-14] 함수에서 return 문이 면적을 반환

6.3 함수 호출

예 return 문에서 여러 개의 값을 반환

```
def multiple_return(...):  
    ...  
    return a, b+10, c
```

여러 개의 값을
반환하는 함수

...

함수에서 3개의 값을 반환하고, 함수 호출문에서 이들을
변수 3개에 차례대로 지정함

x, y, z = multiple_return(...) ← 함수 호출문

6.3 함수 호출

```
def sum_mult(a, b):
```

```
    sum = a + b
```

```
    mult = a * b
```

```
    return (sum, mult)
```

```
...
```

```
(s, m) = sum_mult(num1, num2)
```

튜플



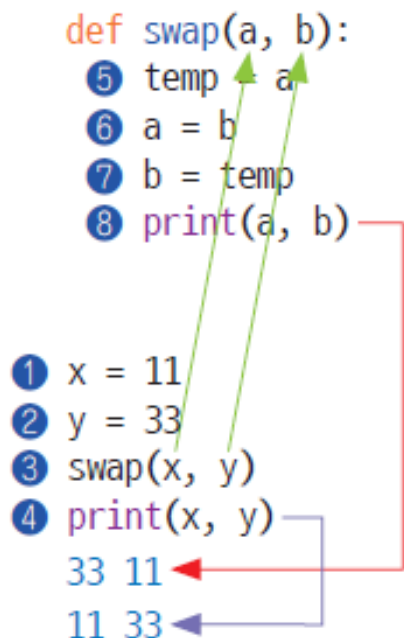
6.3 함수 호출

```
temp = a  
a = b  
b = temp
```

■ 함수와 불변 데이터 타입

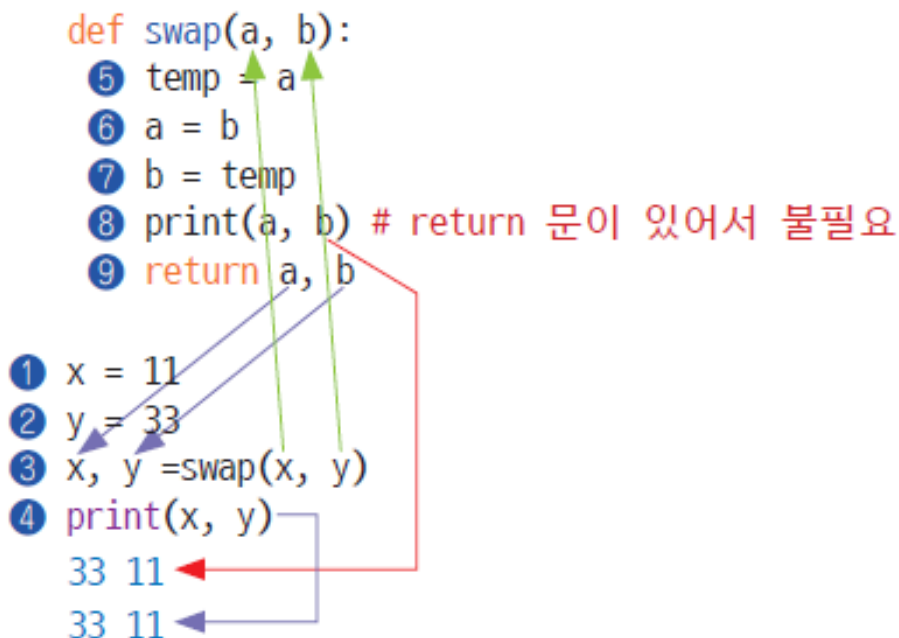
함수 swap에서 return 문을 사용하지 않음

```
def swap(a, b):  
    ⑤ temp ← a  
    ⑥ a = b  
    ⑦ b = temp  
    ⑧ print(a, b)  
  
① x = 11  
② y = 33  
③ swap(x, y)  
④ print(x, y)  
33 11  
11 33
```



함수 swap에서 return 문을 사용함

```
def swap(a, b):  
    ⑤ temp ← a  
    ⑥ a = b  
    ⑦ b = temp  
    ⑧ print(a, b) # return 문이 있어서 불필요  
    ⑨ return a, b  
  
① x = 11  
② y = 33  
③ x, y = swap(x, y)  
④ print(x, y)  
33 11  
33 11
```



[그림 6-15] swap 함수의 두 가지 버전

6.3 함수 호출

인수 -3이 함수의 radius로 전달되고, 함수 안의 if 문에서 radius가 음수이므로 print 문을 실행하고, 바로 return함. None이 반환됨. 메인 프로그램의 print 문의 결과로 None이 출력됨

```
prog-7.py
File Edit Format Run Options Window Help
def circle_area(radius):
    if radius < 0:
        print("반지름은 양수이어야 함")
        return
    area = 3.14 * radius * radius
    return area

print("면적: ", circle_area(-3))
print("면적: ", circle_area(4))
Ln: 9 Col: 29
```

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/파이썬 소스/6장/p
rog-7.py =====
반지름은 양수이어야 함
면적: None
면적: 50.24
>>>
Ln: 8 Col: 4
```

인수 4가 함수의 radius로 전달되고, 함수 안의 if 문에서 radius가 양수이므로 if 문의 두 문장이 실행되지 않음. area가 계산되고, return 문이 area를 반환함. 메인 프로그램의 print 문의 결과로 50.24가 출력됨

[그림 6-16] return 문이 두 개 포함된 함수

6.4 지역 변수

■ 변수의 범위

- 그 변수를 접근할 수 있는 프로그램의 일부분



[그림 6-17] 변수의 두 가지 유형

- 전역 변수: 메인 프로그램에서 값을 배정한 변수
- 지역 변수: 어떤 함수 안에서 값을 배정한 변수 (매개변수 포함)

6.4 지역 변수

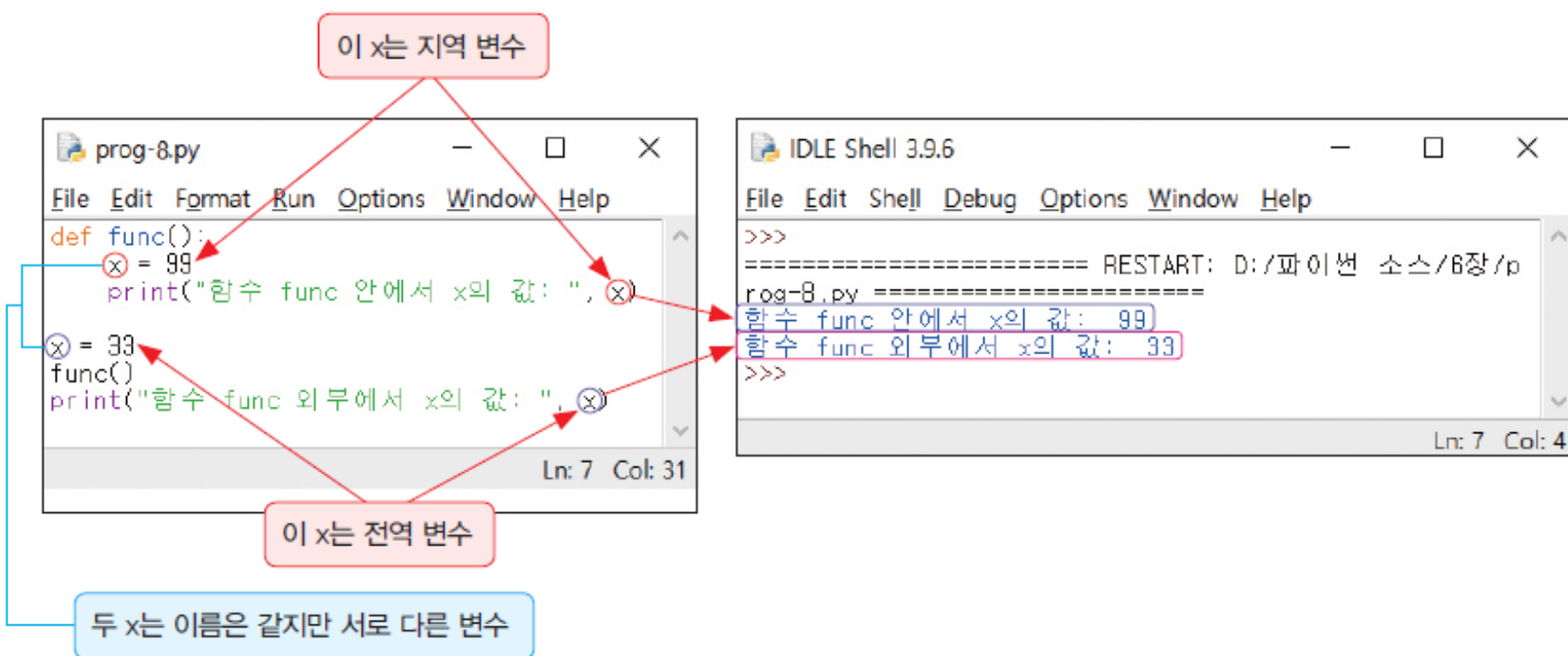
```
def func(a, b)
    x = a + b
    y = a * b
    result = x + y
    return result
```

매개변수 a, b와 함수 안에서 정의한 x, y, result는 모두 지역 변수이고, 함수가 실행되는 동안에만 존재한다.

[그림 6-18] 지역 변수

6.4 지역 변수

예 전역 변수와 지역 변수



[그림 6-19] 같은 이름을 갖는 전역 변수와 지역 변수

6.4 지역 변수

예 함수의 지역 변수를 메인 프로그램에서 참조

```
prog-9.py
File Edit Format Run Options Window Help
def func():
    y = 99
    print("함수 func 안에서 y의 값: ", y)
func()
print("함수 func 외부에서 y의 값: ", y)
Ln: 5 Col: 0
```

```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/파이썬 소스/6장/pr
og-9.py =====
함수 func 안에서 y의 값: 99
Traceback (most recent call last):
  File "D:/파이썬 소스/6장/prog-9.py", line 6, in <mo
dule>
    print("함수 func 외부에서 y의 값: ", y)
NameError: name 'y' is not defined
>>>
Ln: 10 Col: 4
```

변수 y는 함수 안에서 정의되었으므로 함수의 지역 변수이고, 함수의 실행이 끝나면 사라진다. 메인 프로그램에서 함수의 지역 변수 y를 접근(출력)하려고 하면 에러 메시지가 나타난다.

[그림 6-20] 함수의 지역 변수 y를 메인 프로그램에서 접근할 때 발생하는 에러 메시지

6.4 지역 변수

예 함수 안에서 지역 변수를 생성하기 전에 참조

```
def wrong_func():  
    print('x의 값: ', x) # 에러 메시지가 발생함  
    x = 99
```

6.5 전역 변수

■ 전역 변수

- 어떤 함수에도 속하지 않는, 메인 프로그램에서 선언한 변수
- 메인 프로그램 및 메인 프로그램에 속한 모든 함수들이 접근할 수 있는 변수

6.5 전역 변수

```
File Edit Format Run Options Window Help
def func(b):
    3 a = a + b
    return a
1 a = 5
2 c = func(3)
print(c)
print(a)
```

지역 변수

전역 변수

Ln: 5 Col: 5

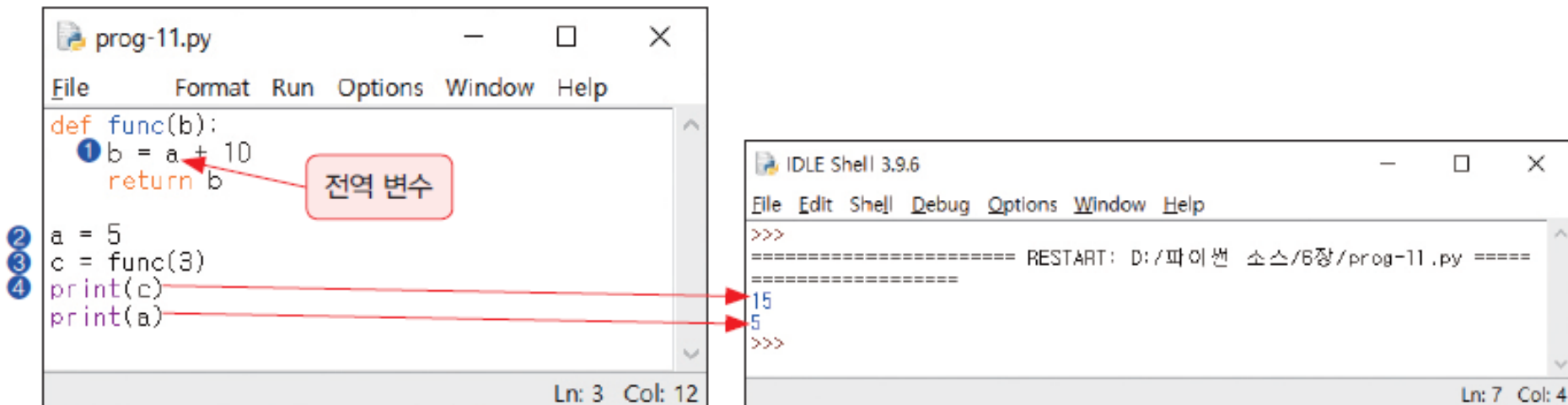


```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/파이썬 소스/6장/prog-10.py =====
Traceback (most recent call last):
  File "D:/파이썬 소스/6장/prog-10.py", line 6, in <module>
    c = func(3)
  File "D:/파이썬 소스/6장/prog-10.py", line 2, in func
    a = a + b
UnboundLocalError: local variable 'a' referenced before assignment
>>>
```

Ln: 11 Col: 4

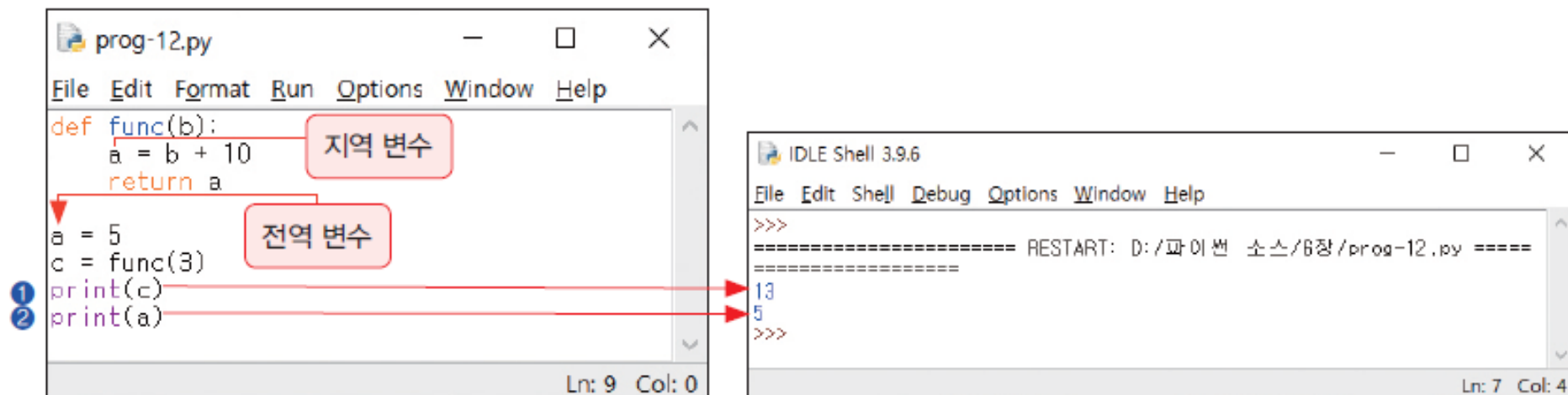
[그림 6-21] 함수에서 값을 지정하지 않은 지역 변수를 사용하려고 함

6.5 전역 변수



[그림 6-22] 함수에서 전역 변수를 읽음

6.5 전역 변수

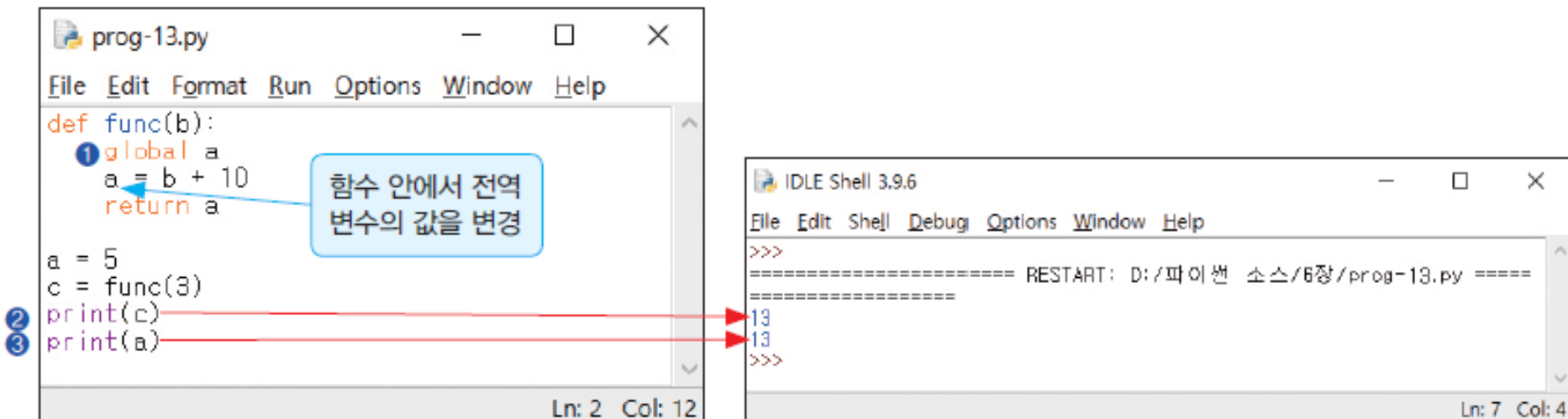


[그림 6-23] 같은 이름 a를 갖는 전역 변수와 지역 변수

6.5 전역 변수

■ global 키워드

- 함수 func 안에서 새로운 지역 변수 a를 생성하지 않고, 메인 프로그램의 전역 변수 a의 값을 변경하려면 함수 func의 ❶에서 global 키워드를 사용하여 a가 전역 변수임을 명시



[그림 6-24] 함수 안에서 global 키워드로 전역 변수를 명시

6.5 전역 변수

■ 전역 상수

- 함수 수정할 수 없는 값
- 관례상 전역 상수를 위한 이름은 모두 대문자로 나타냄

예 전역 상수

```
PI = 3.14                                # 전역 상수
def circle_area(radius):
    return PI*radius*radius              # 원의 면적을 반환

def circle_circum(radius):
    return 2*PI*radius                   # 원의 둘레를 반환

radius = int(input("반지름을 입력하시오: "))
print("원의 면적: ", circle_area(radius))
print("원의 둘레: ", circle_circum(radius))
```

6.7 순환 함수

■ 순환 함수

- 자기 자신을 호출하는 함수
- 흔히 순환 알고리즘을 순환 함수로 구현
- 순환 알고리즘(함수)에는 두 가지 부분이 포함되어 있어야 함
 - ◆ 순환 호출을 하지 않고 직접 문제를 해결할 수 있는 부분(기본 사례)
 - ◆ 순환 호출을 하는 부분 - 기본 사례를 향해 나아가야 함

6.7 순환 함수

순환 알고리즘의 예

- 피보나치 수열

$$fib(n) = \begin{cases} 0 & \text{if } n=0 \\ 1 & \text{if } n=1 \\ fib(n-2) + fib(n-1) & \text{otherwise} \end{cases}$$

- $sum(n) = 1 + 2 + 3 + \dots + n$

6.7 순환 함수

예 순환 함수

팩토리얼의 정의

$$n! = \begin{cases} 1 & n \leq 1 \\ n * (n-1)! & n \geq 2 \end{cases}$$

```
def factorial(n):
```

```
    if n <= 1:
```

```
        return 1
```

```
    return n * factorial(n-1)
```

순환 호출을 멈추는 부분
(기본 사례)

순환 호출을 하는 부분
(순환 호출을 할 때마다 n이 1씩 줄
어들어서 기본 사례에 도달하게 됨)

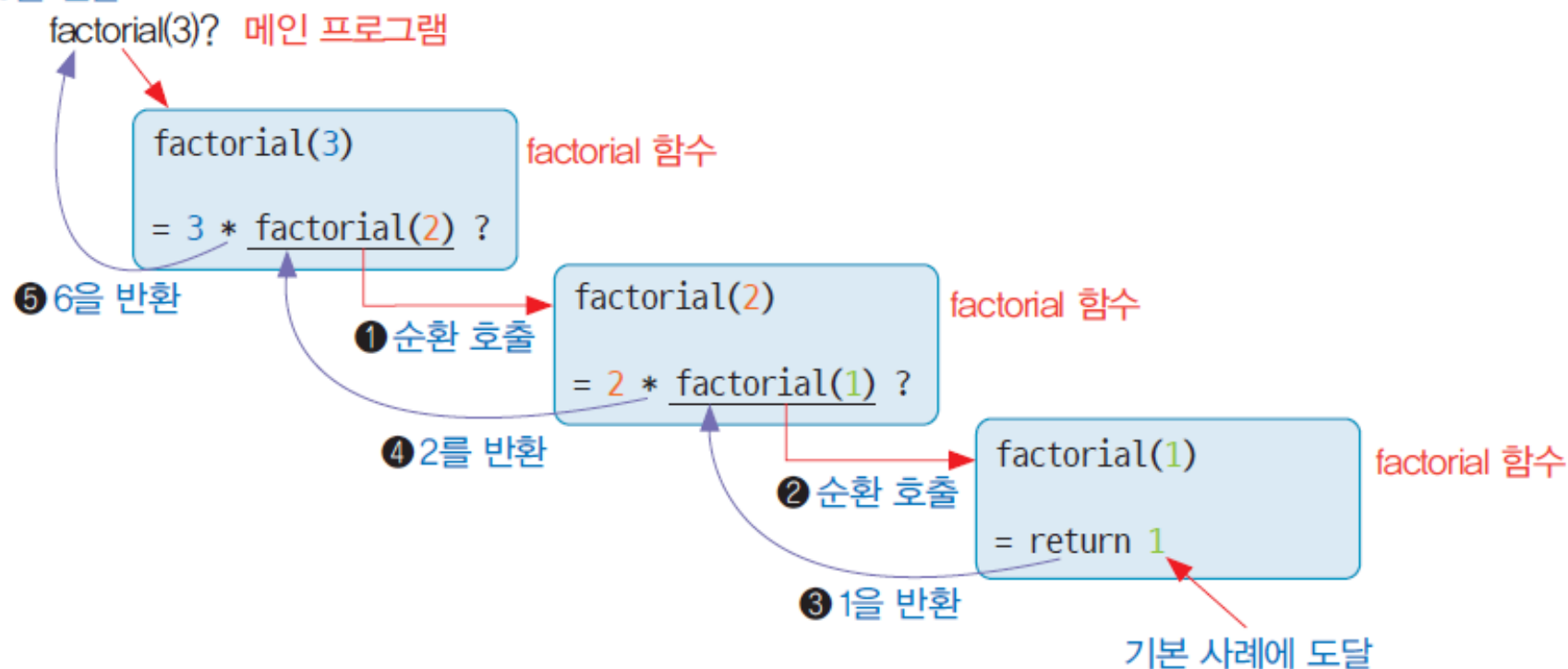
```
print(factorial(3)) # 메인 프로그램에서 함수 호출
```

6.7 순환 함수

```
factorial(3) = 3 * factorial(2)
             = 3 * (2 * factorial(1)) # 1은 기본 사례
             = 3 * (2 * (1))
             = 6
```

6.7 순환 함수

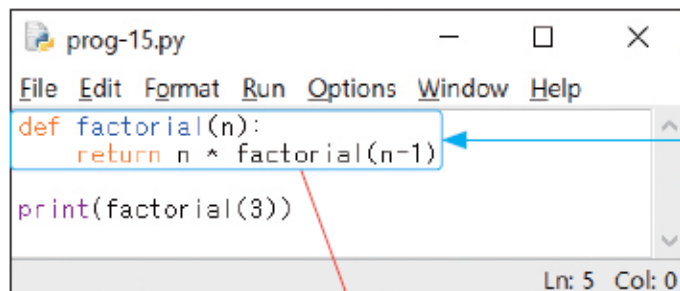
메인 프로그램에서 함수를
호출하면서 매개변수에
3을 전달



[그림 6-27] $n=3$ 일 때 순환 호출 및 반환되는 값

6.7 순환 함수

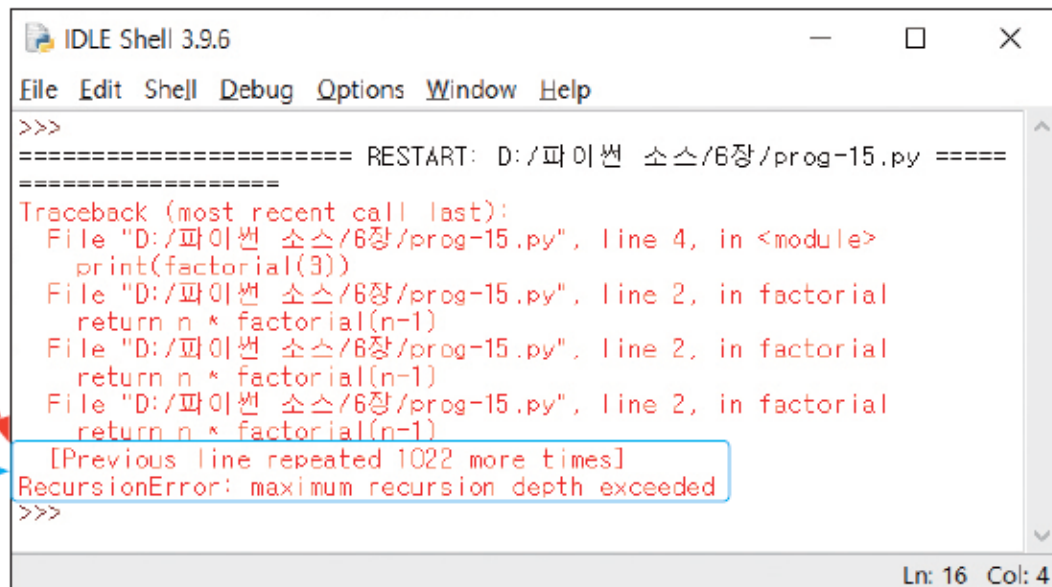
■ 기본 사례를 빠뜨린 순환 함수



```
def factorial(n):  
    return n * factorial(n-1)  
  
print(factorial(3))
```

기본 사례가 빠진
순환 함수

기본 사례에 도달하지 못하고 n 이
1씩 줄어들면서 계속 순환 호출하
다가 최대 순환 횟수를 초과해서
에러가 발생함

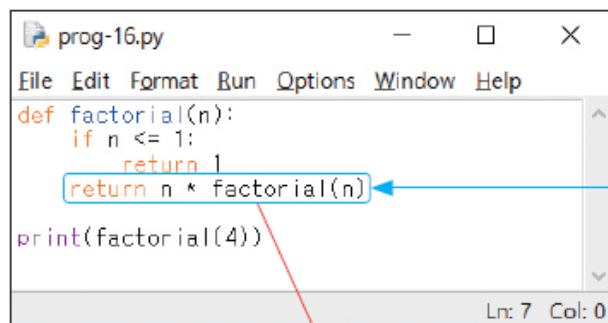


```
>>>  
===== RESTART: D:/파이썬 소스/6장/prog-15.py =====  
Traceback (most recent call last):  
  File "D:/파이썬 소스/6장/prog-15.py", line 4, in <module>  
    print(factorial(3))  
  File "D:/파이썬 소스/6장/prog-15.py", line 2, in factorial  
    return n * factorial(n-1)  
  File "D:/파이썬 소스/6장/prog-15.py", line 2, in factorial  
    return n * factorial(n-1)  
  File "D:/파이썬 소스/6장/prog-15.py", line 2, in factorial  
    return n * factorial(n-1)  
[Previous line repeated 1022 more times]  
RecursionError: maximum recursion depth exceeded  
>>>
```

[그림 6-28] 순환 함수 정의에 기본 사례 누락

6.7 순환 함수

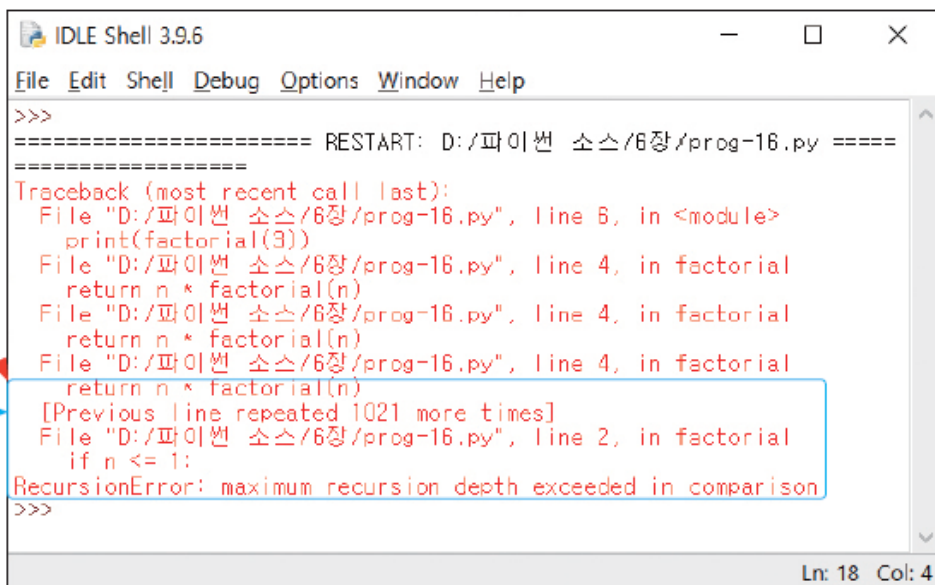
■ 기본 사례로 나아가지 못하는 순환 함수



```
prog-16.py
File Edit Format Run Options Window Help
def factorial(n):
    if n <= 1:
        return 1
    return n * factorial(n)
print(factorial(4))
Ln: 7 Col: 0
```

순환 호출을 할 때 n이 줄어들지 않아서
기본 사례에 도달할 수 없음

기본 사례에 도달하지 못하고 계속
반복하다가 최대 순환 횟수를 초과
해서 에러가 발생함



```
IDLE Shell 3.9.6
File Edit Shell Debug Options Window Help
>>>
===== RESTART: D:/파이썬 소스/6장/prog-16.py =====
Traceback (most recent call last):
  File "D:/파이썬 소스/6장/prog-16.py", line 5, in <module>
    print(factorial(3))
  File "D:/파이썬 소스/6장/prog-16.py", line 4, in factorial
    return n * factorial(n)
  File "D:/파이썬 소스/6장/prog-16.py", line 4, in factorial
    return n * factorial(n)
  File "D:/파이썬 소스/6장/prog-16.py", line 4, in factorial
    return n * factorial(n)
  [Previous line repeated 1021 more times]
  File "D:/파이썬 소스/6장/prog-16.py", line 2, in factorial
    if n <= 1:
RecursionError: maximum recursion depth exceeded in comparison
>>>
Ln: 18 Col: 4
```

[그림 6-29] 순환 함수 정의가 기본 사례를 향해 나아가지 못함

6.7 순환 함수

■ 순환을 반복으로

- 순환적인 알고리즘을 순환 함수로 작성하면 자연스럽고, 가독성을 높일 수 있다는 장점
- 매개변수들과 지역 변수들을 위한 공간 할당, 되돌아갈 주소 저장 등 일반적으로 순환 알고리즘은 반복 알고리즘에 비해서 비효율적
- 대부분의 순환 문제는 반복문을 사용하여 해결 가능

6.7 순환 함수

2부터 n까지 1씩 증가
하면서 반복됨

fact = 1
fact = fact * 2 (= 1 * 2)
fact = fact * 3 (= 1 * 2 * 3)
...
fact = fact * (n-1) (= 1 * 2 * ... * (n-1))
fact = fact * n (= 1 * 2 * ... * n)

또는 n부터 2까지 1씩
감소하면서 반복됨

```
def factorial(n)
    fact = 1
    for i in range(2, n+1)
        fact = fact * i
    return fact
```

```
def factorial(n)
    fact = 1
    for i in range(n, 1, -1)
        fact = fact * i
    return fact
```

[그림 6-30] 팩토리얼 함수에 존재하는 반복적인 특성