

스크립트 프로그래밍

06 함수

2016 2학기 (02분반)

강승우

학습 목표

- 형식 매개변수를 가진 함수를 정의할 수 있다(6.2 절).
- 실매개변수(즉, 인자)를 가진 함수를 호출할 수 있다(6.3절).
- 값을 반환하는 함수와 값을 반환하지 않는 함수를 구별할 수 있다(6.4절).
- 위치 인자 혹은 키워드 인자를 사용하여 함수를 호출할 수 있다(6.5절).
- 참조값을 인자로 전달할 수 있다(6.6절).
- 읽기, 디버깅, 관리에 편리한 재사용 코드를 개발할 수 있다(6.7절).
- 함수를 재사용하기 위한 모듈을 만들 수 있다(6.7-6.9절).
- 변수의 스코프를 결정할 수 있다(6.9절).
- 기본 인자를 가진 함수를 정의할 수 있다(6.10절).
- 다중값을 반환하는 함수를 정의할 수 있다(6.11절).
- 소프트웨어 개발에서 함수 추상화의 개념을 적용할 수 있다 (6.12절).
- 단계적 개선을 사용한 함수의 설계 및 구현할 수 있다 (6.13절).
- 재사용 함수를 이용하여 그래픽 프로그램을 만들 수 있다 (6.14절).

함수

- 함수
 - 재사용 코드를 정의하여 코드를 단순화하는데 사용
- 사례
 - 1부터 10까지 정수의 합, 20부터 37까지 정수의 합, 35부터 49까지 정수의 합을 구하는 프로그램을 작성하고자 할 때,

```
sum = 0
for i in range(1, 10):
    sum += i
print("1에서 10까지의 합은 ", sum, " 입니다. ")

sum = 0
for i in range(20, 37):
    sum += i
print("20에서 37까지의 합은 ", sum, " 입니다. ")

sum = 0
for i in range(35, 49):
    sum += i
print("35에서 49까지의 합은 ", sum, " 입니다. ")
```

DRY!!!

Don't Repeat Yourself

함수 사용 예제

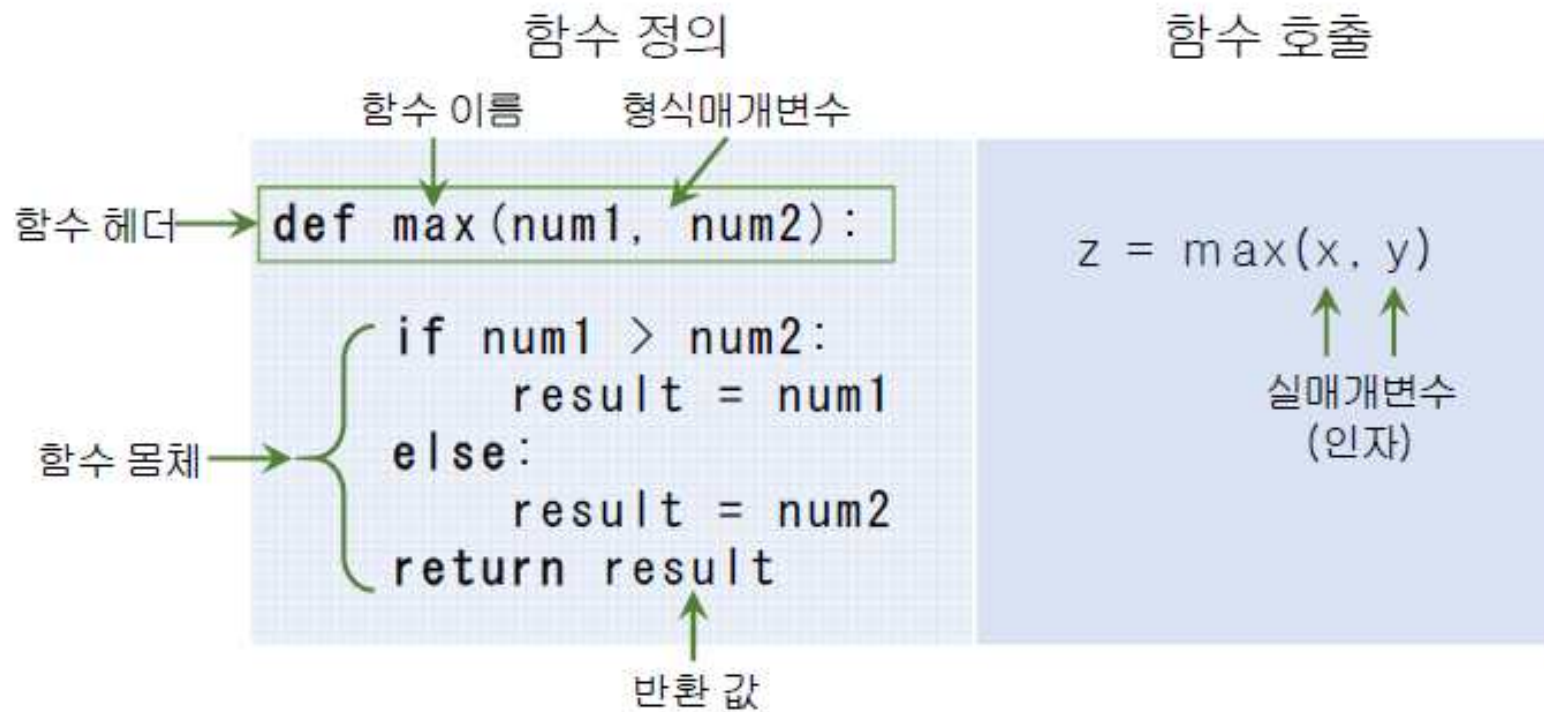
```
def sum(i1, i2):  
    result = 0  
    for i in range(i1, i2):  
        result += i  
    return result
```

```
def main():  
    print( "1부터 10까지의 합은 ", sum(1, 10))  
    print( "20부터 37까지의 합은", sum(20, 37))  
    print( "35부터 49까지의 합은", sum(35, 49))
```

```
main() # main 함수를 호출한다.
```

함수 정의

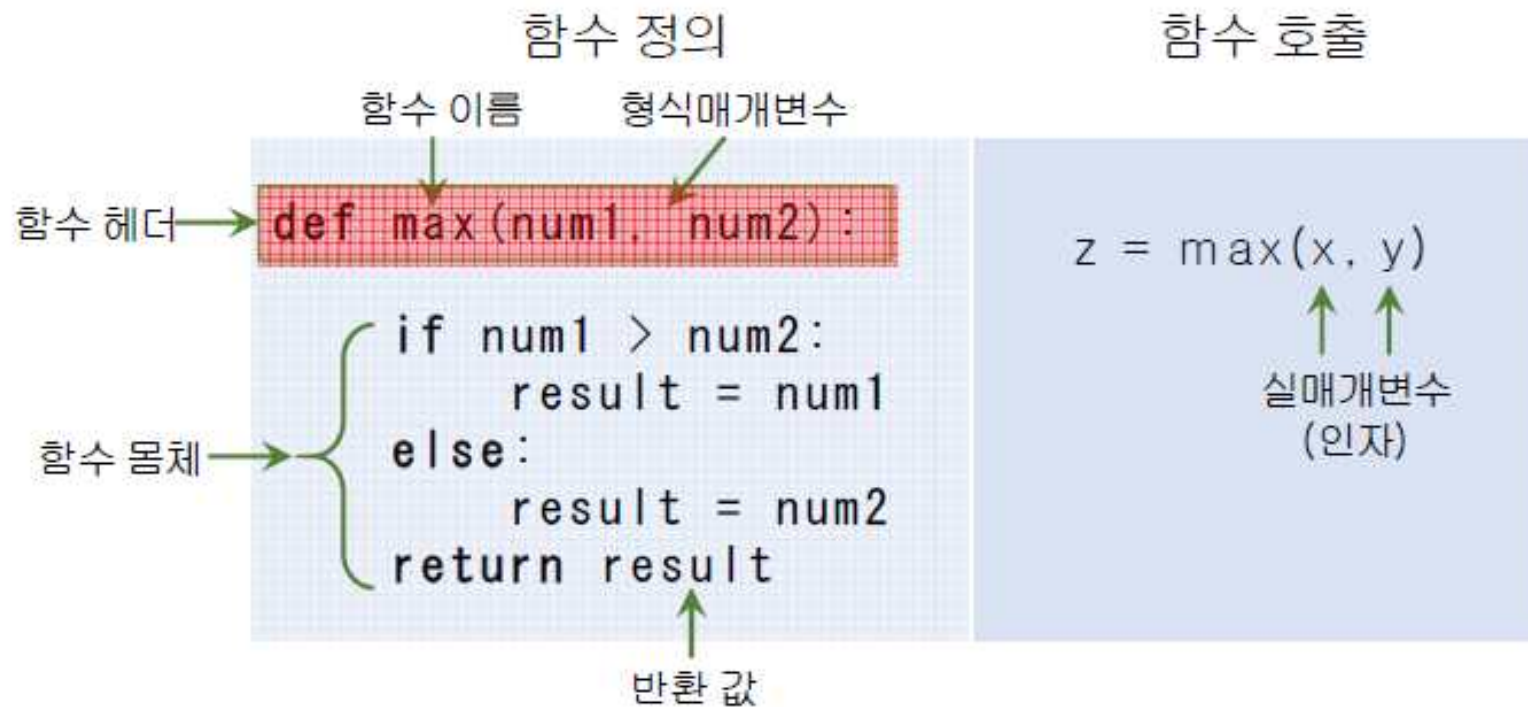
- 함수(function)
 - 프로그램에서 동작을 수행하기 위한 명령문들의 모음



함수 헤더

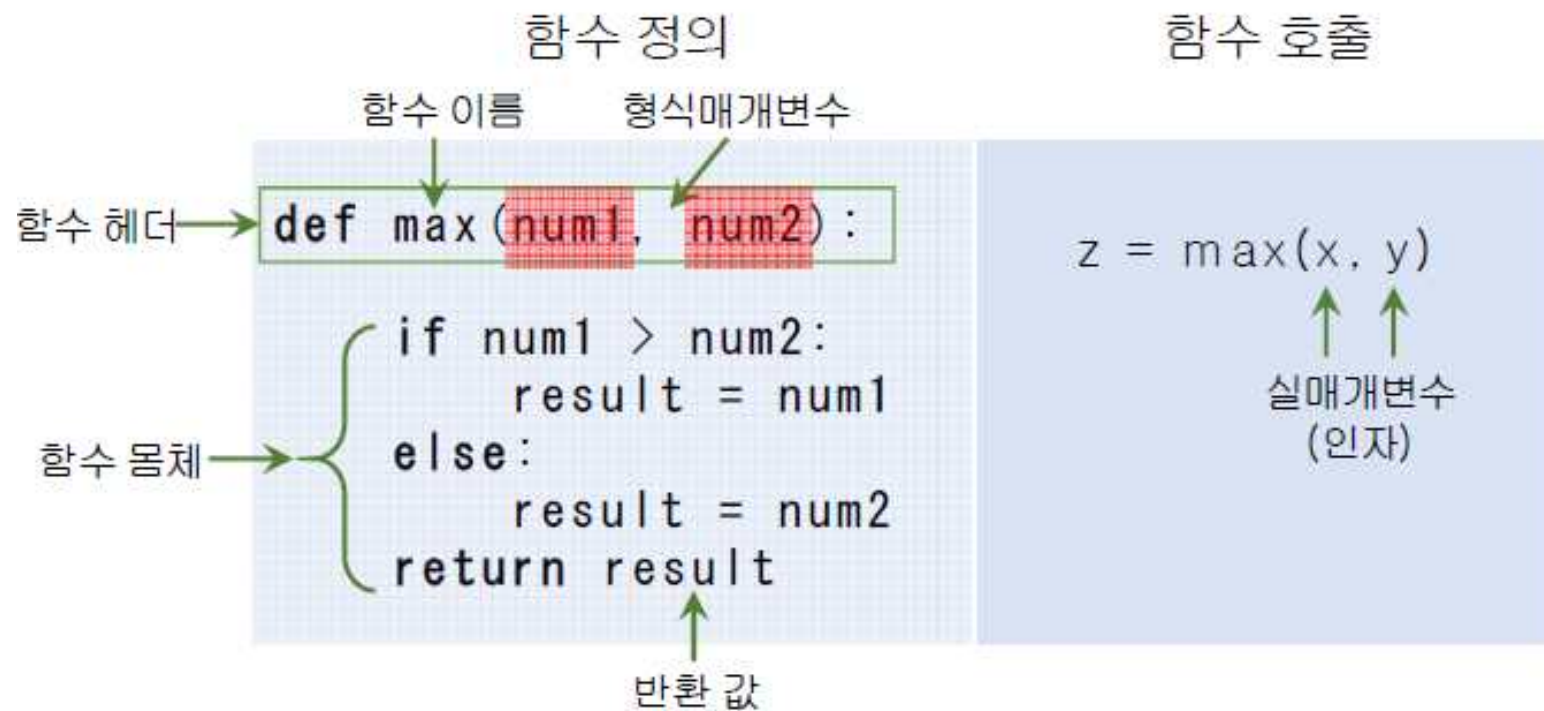
- 헤더(header)

- def 키워드로 시작, 그 다음 함수의 이름과 매개변수가 나오며, 콜론(:)으로 끝남



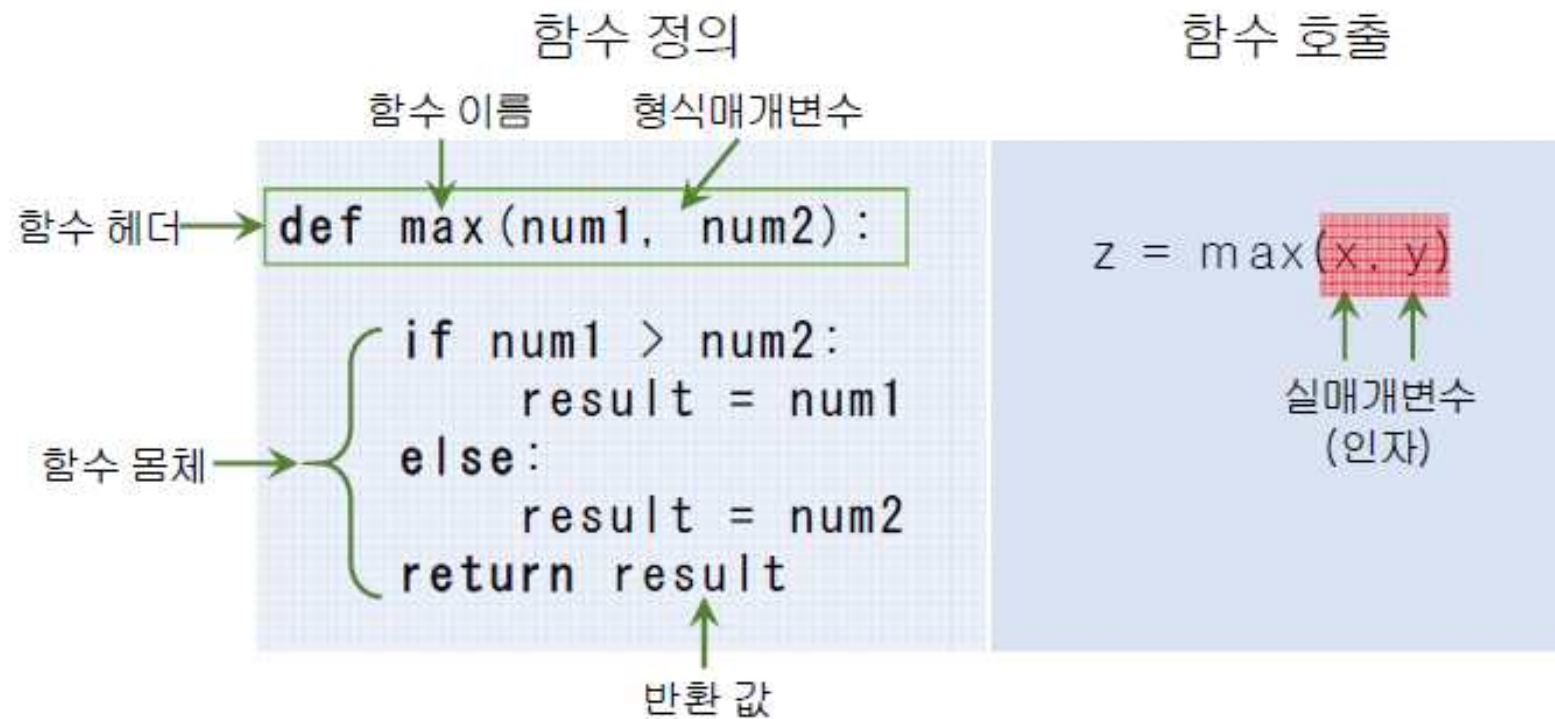
형식 매개변수

- 형식 매개변수(formal parameter) 혹은 매개변수
 - 함수 헤더 내의 변수



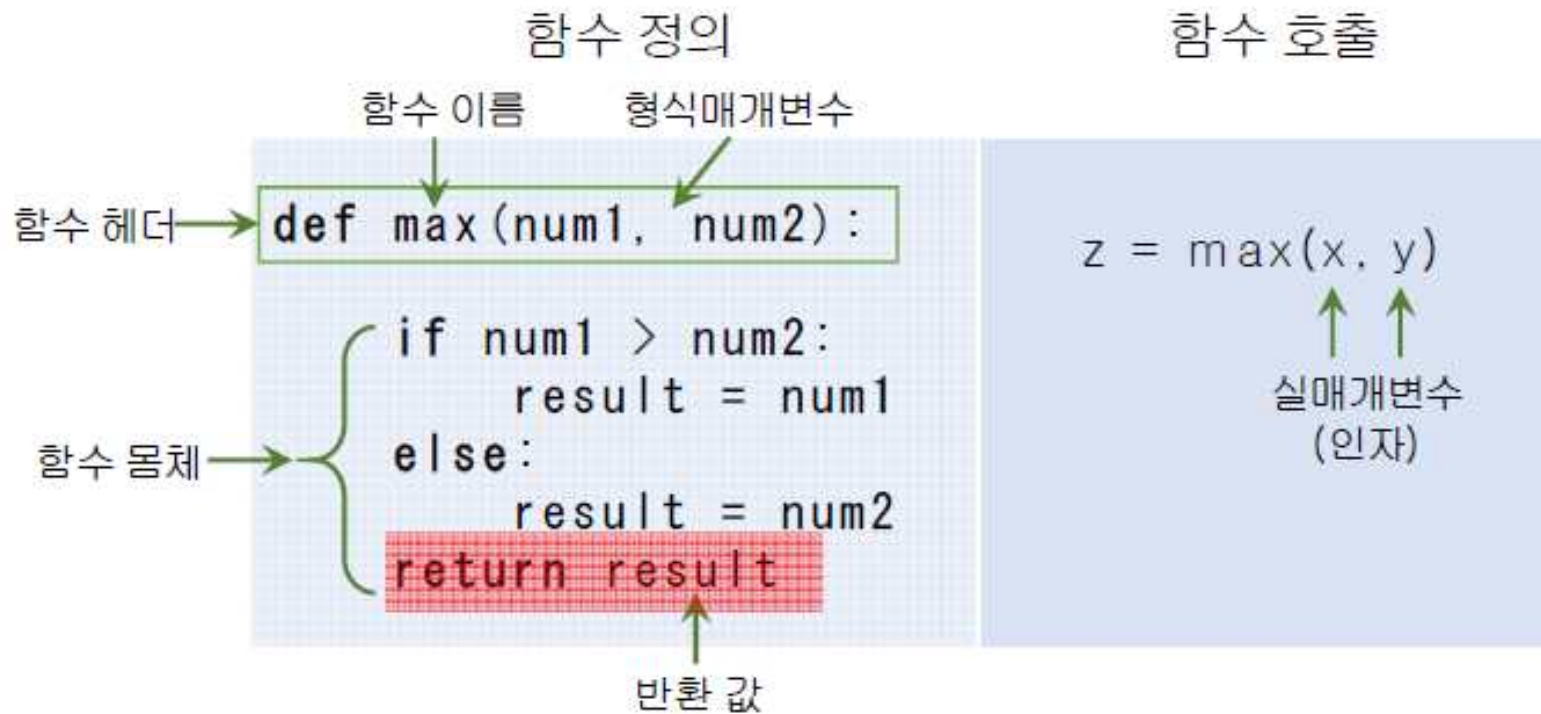
실매개변수

- 실매개변수(actual parameter) 또는 인자(argument)
 - 함수가 호출될 때 매개변수로 전달되는 값



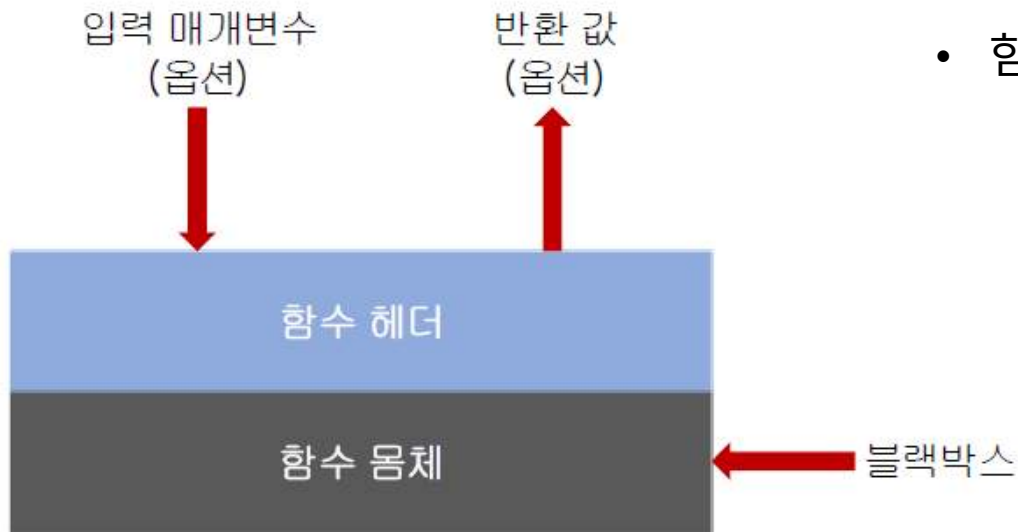
반환 명령문

- return 키워드 사용
- 값-반환 함수에 필수적



함수 추상화

- 함수 추상화: 함수 사용과 함수 구현을 분리시키는 것
 - 우리는 이미 print, sqrt, randint 함수 등이 어떻게 구현되어 있는지 알지 못하지만 사용했음
- 함수의 몸체를 함수의 구체적인 구현 방법을 포함하고 있는 블랙 박스로 생각할 수 있다.

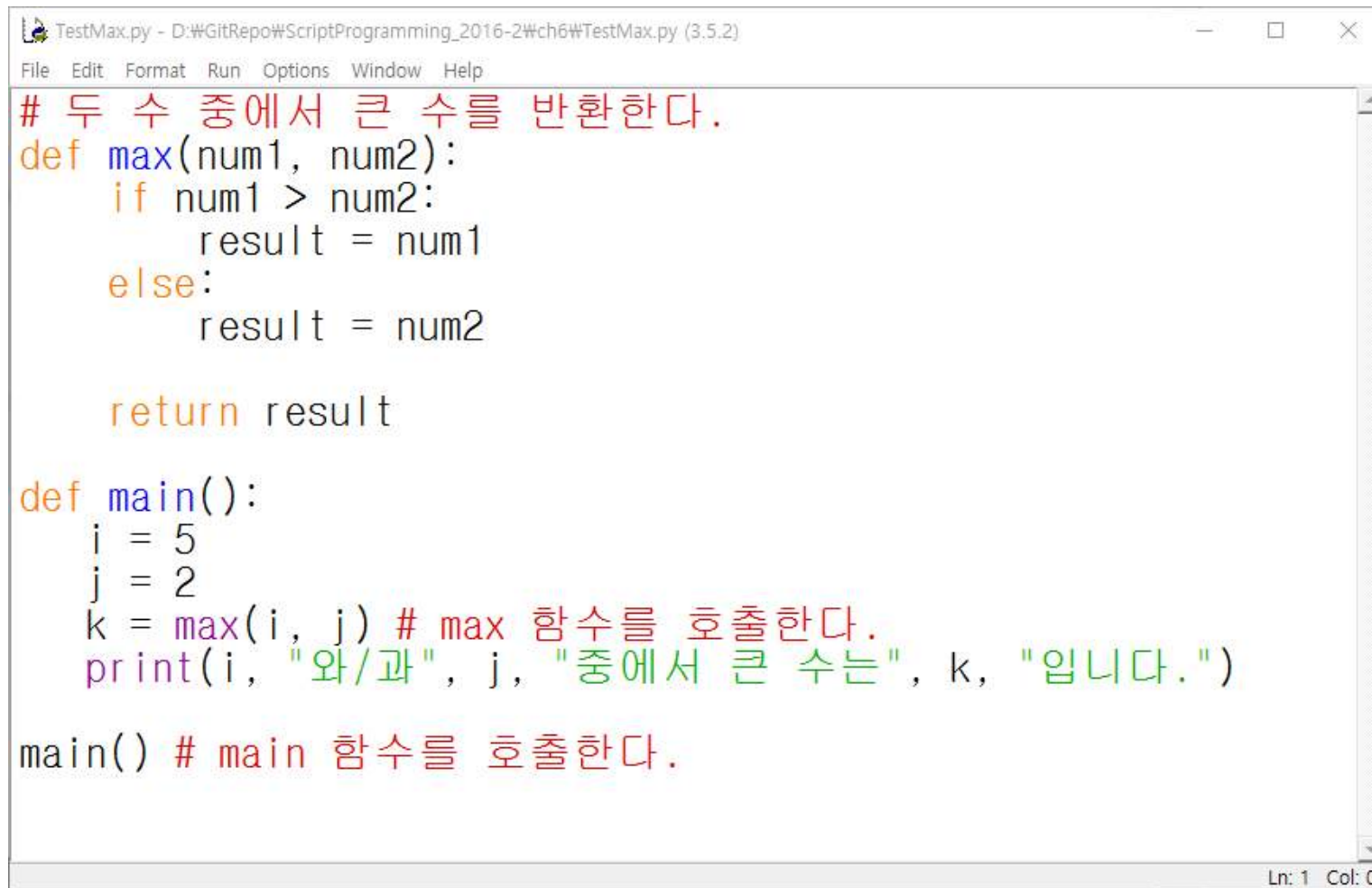


- 함수 사용의 장점
 - 재사용
 - 정보 은닉(캡슐화): 구현 내용을 감출 수 있음
 - 복잡도 감소

함수 호출하기

- 함수 정의
 - 함수가 어떤 연산을 수행하는 것인지 정하는 것
- 함수 호출
 - 함수를 실제로 사용하기 위해서 호출(call, invoke) 필요
- 값을 반환하는 함수의 경우 함수 호출의 결과는 반환값
 - `larger = max(3, 4)`
 - → `max(3,4)`를 호출하고 그 결과를 `larger` 변수에 할당
- 값을 반환하지 않는 함수의 경우 함수 호출은 명령문의 실행
 - `print("프로그래밍은 재미있습니다!")`

함수 호출하기 예제



```
TestMax.py - D:\GitRepo\ScriptProgramming_2016-2\ch6\TestMax.py (3.5.2)
File Edit Format Run Options Window Help
# 두 수 중에서 큰 수를 반환한다.
def max(num1, num2):
    if num1 > num2:
        result = num1
    else:
        result = num2

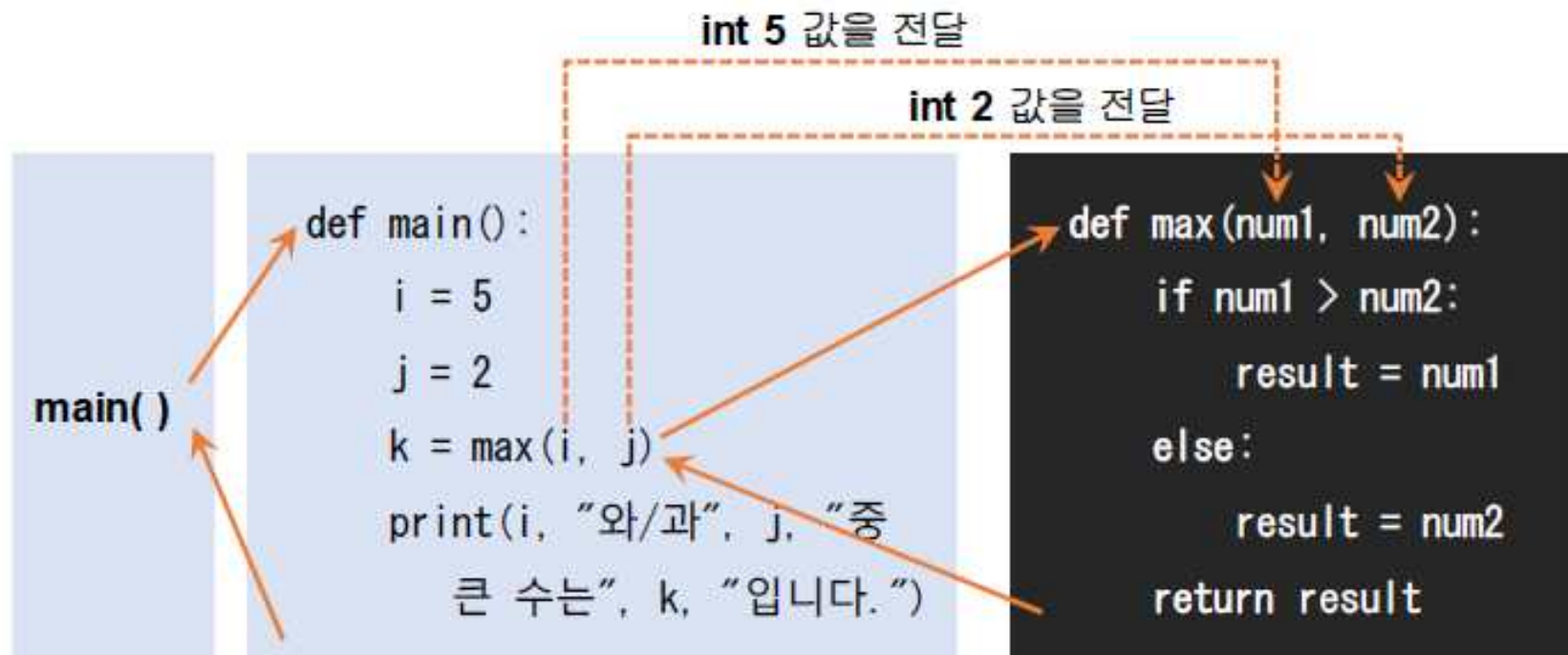
    return result

def main():
    i = 5
    j = 2
    k = max(i, j) # max 함수를 호출한다.
    print(i, "와/과", j, "중에서 큰 수는", k, "입니다.")

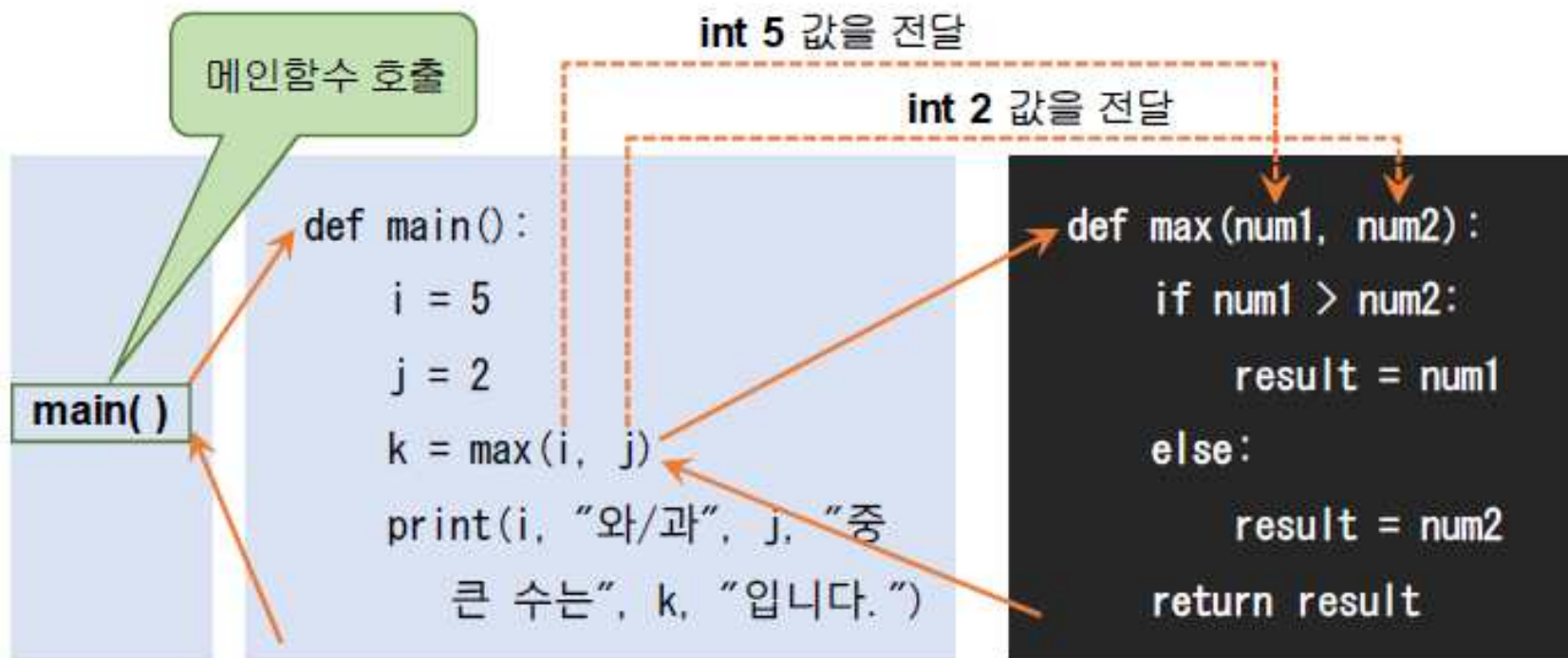
main() # main 함수를 호출한다.
```

Ln: 1 Col: 0

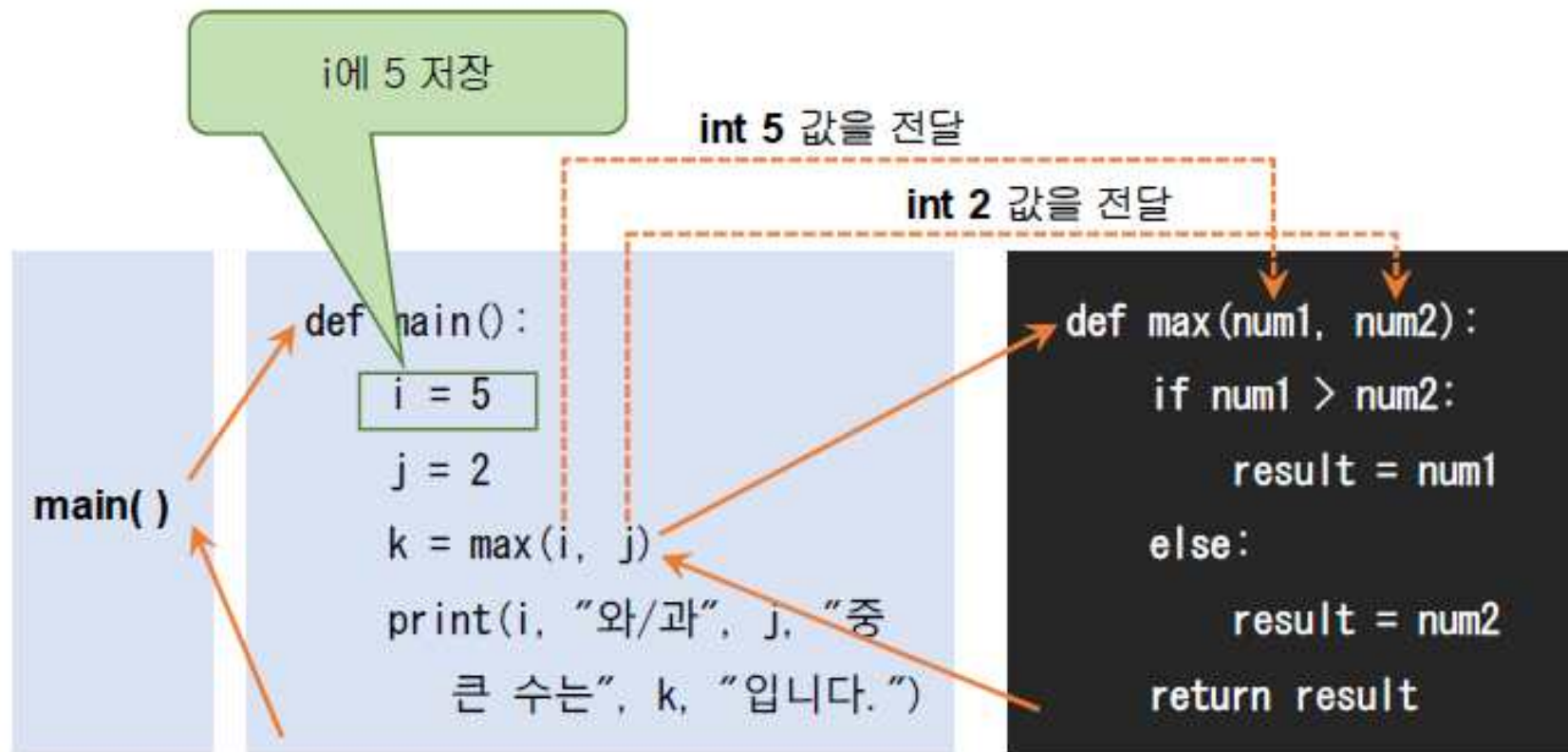
함수 호출 트레이스



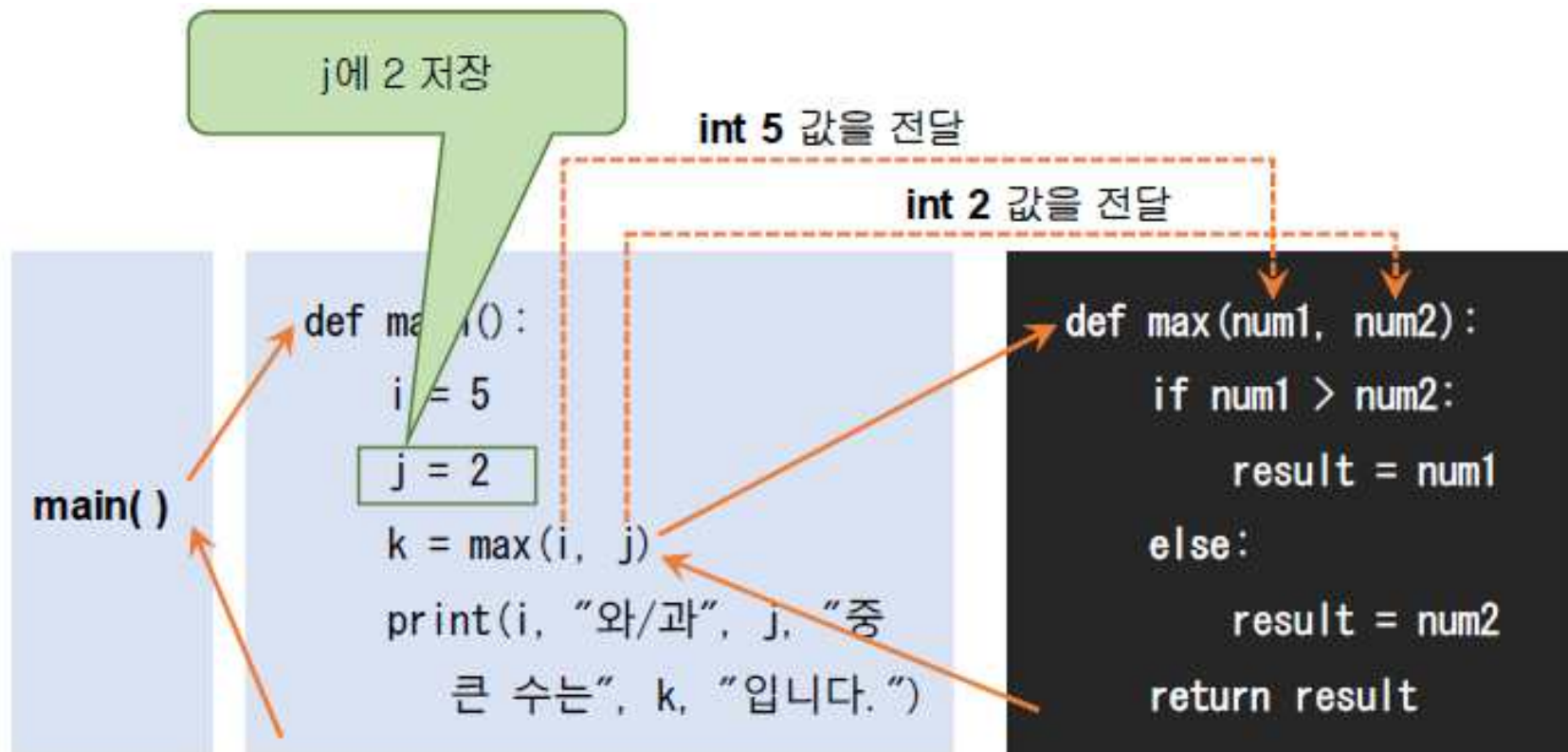
함수 호출 트레이스



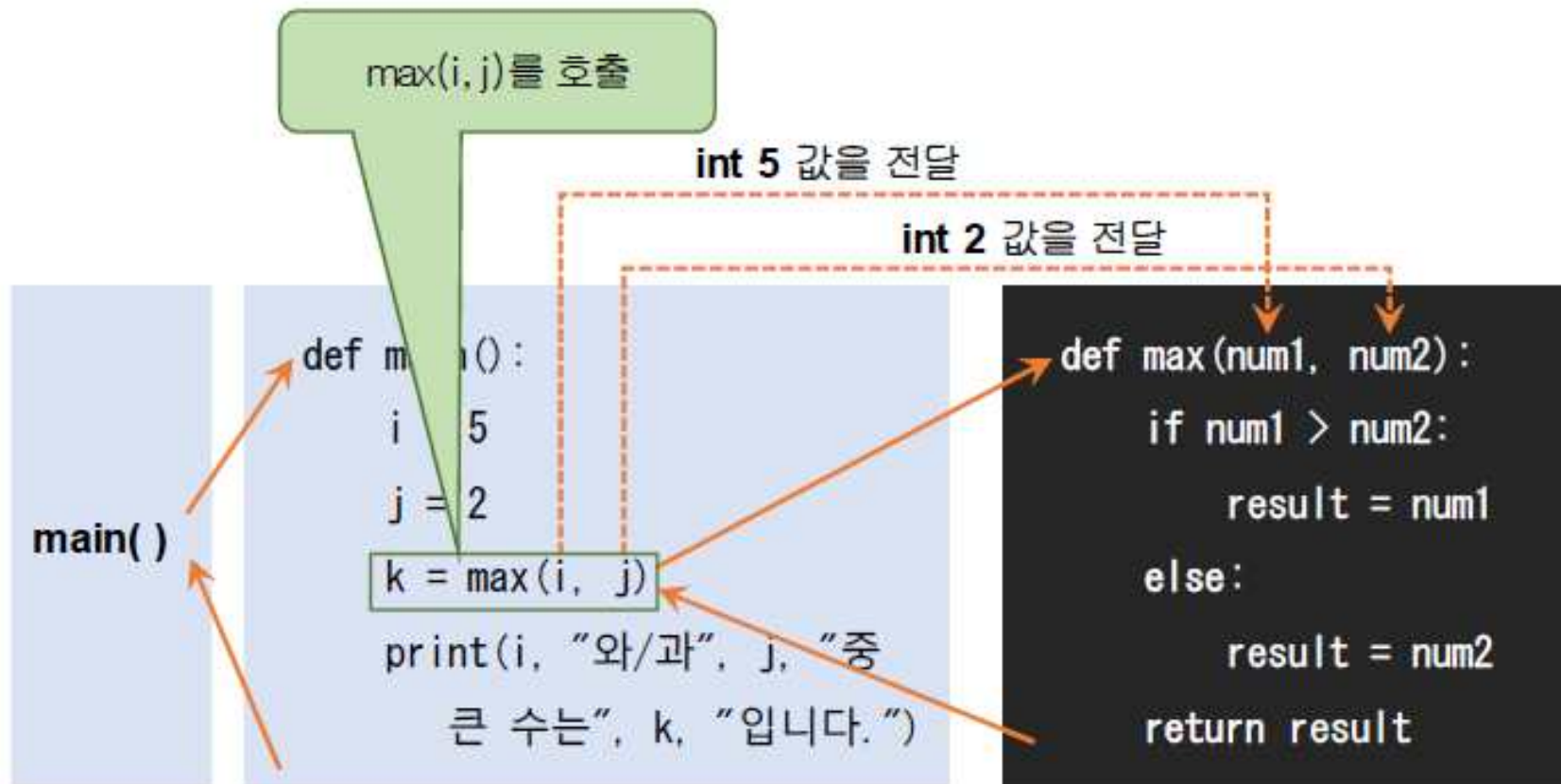
함수 호출 트레이스



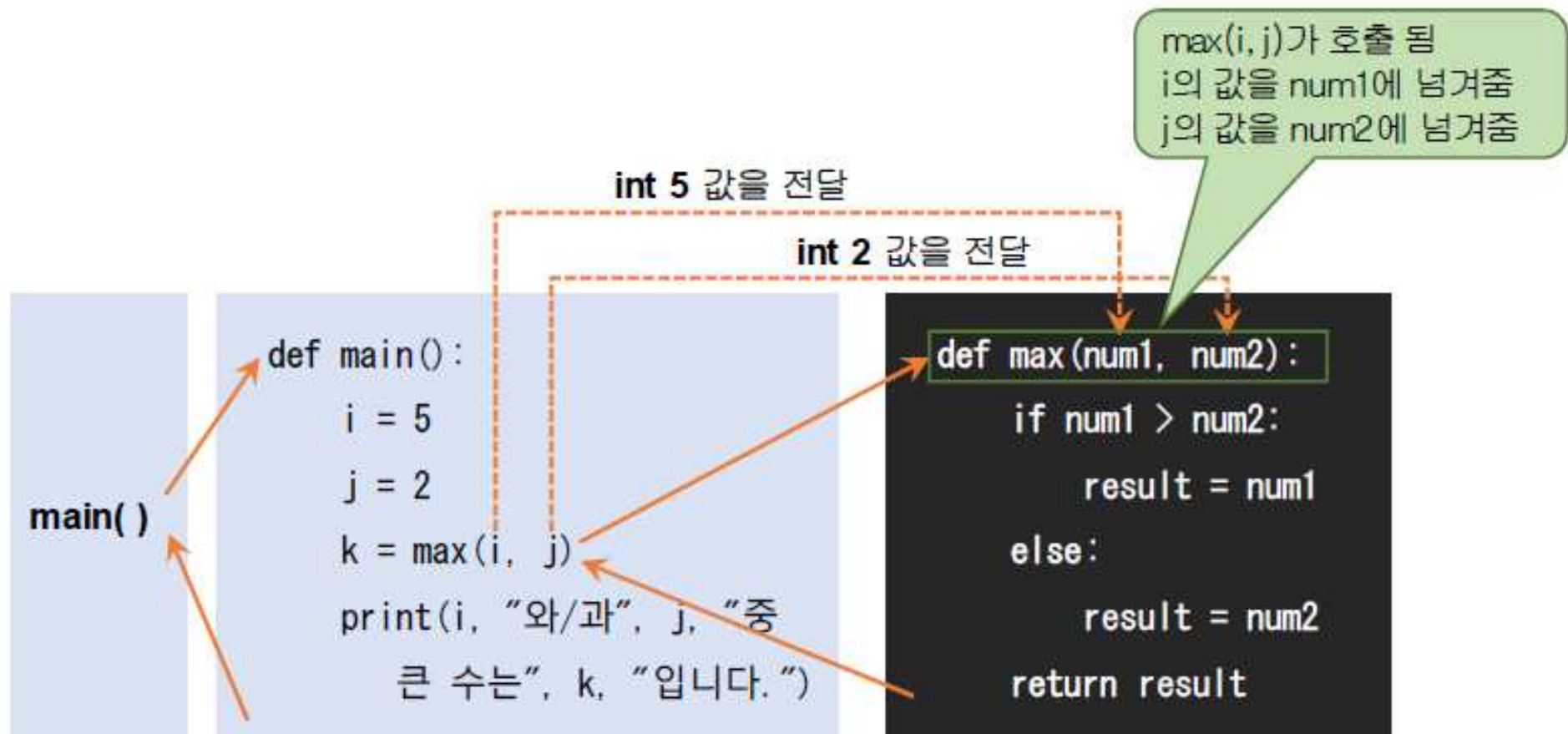
함수 호출 트레이스



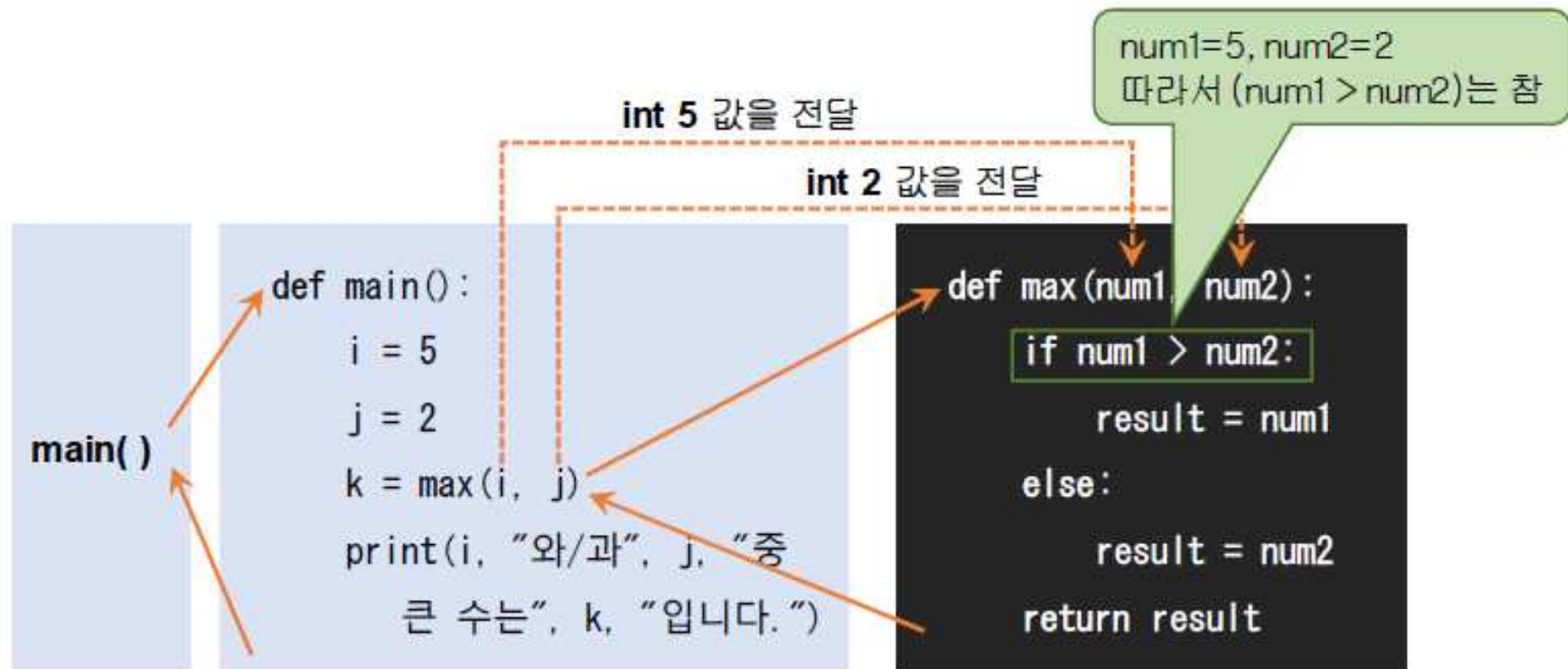
함수 호출 트레이스



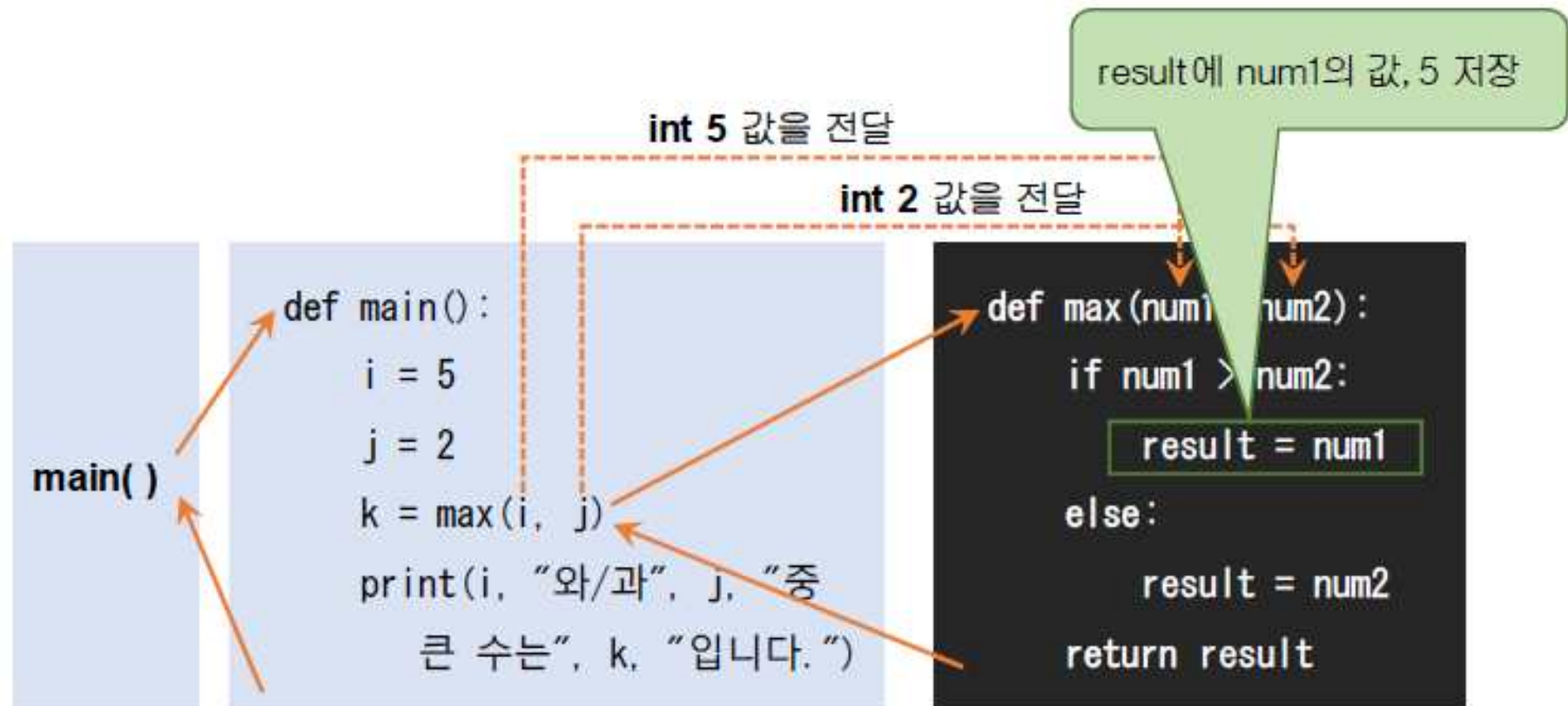
함수 호출 트레이스



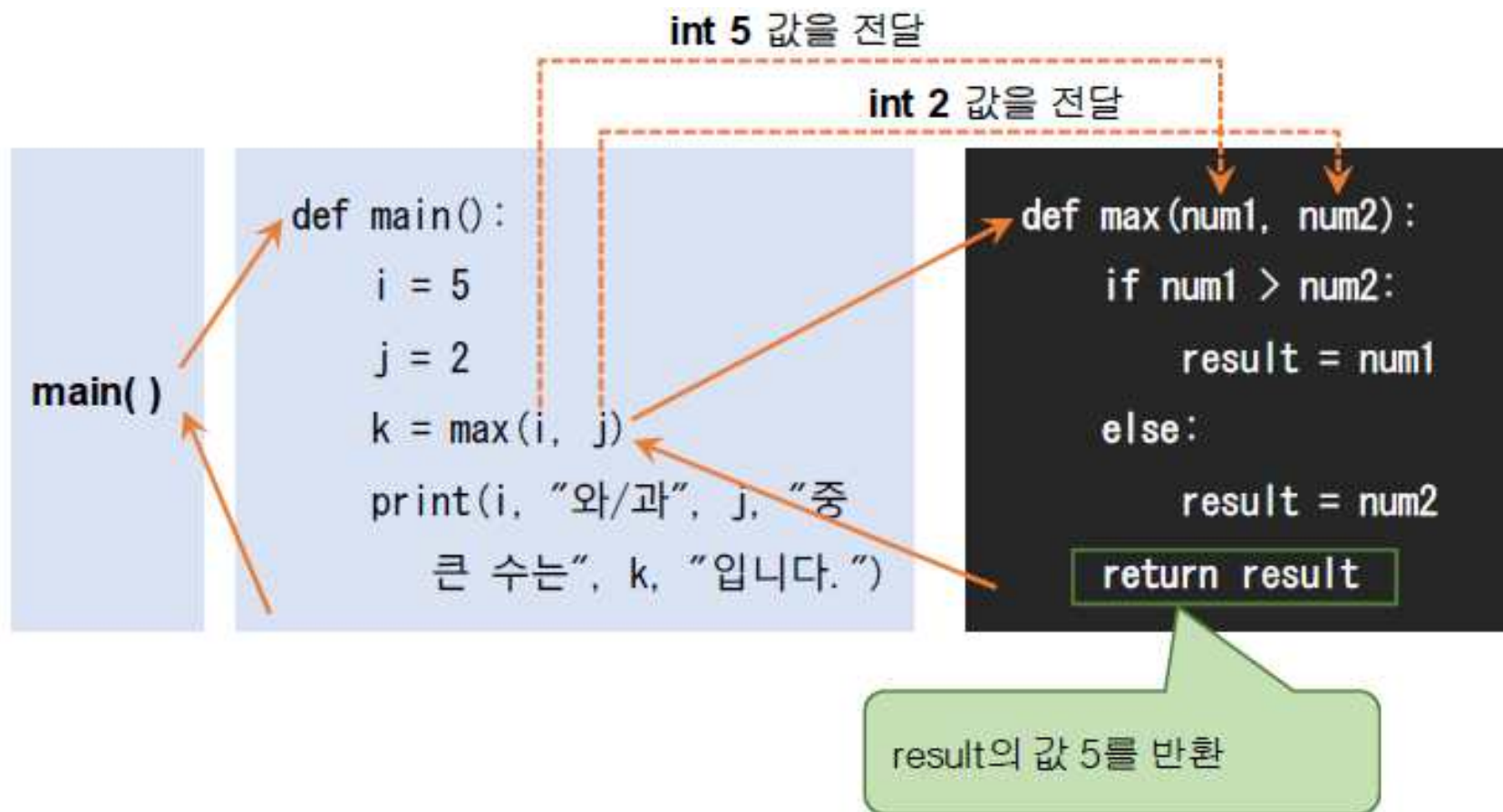
함수 호출 트레이스



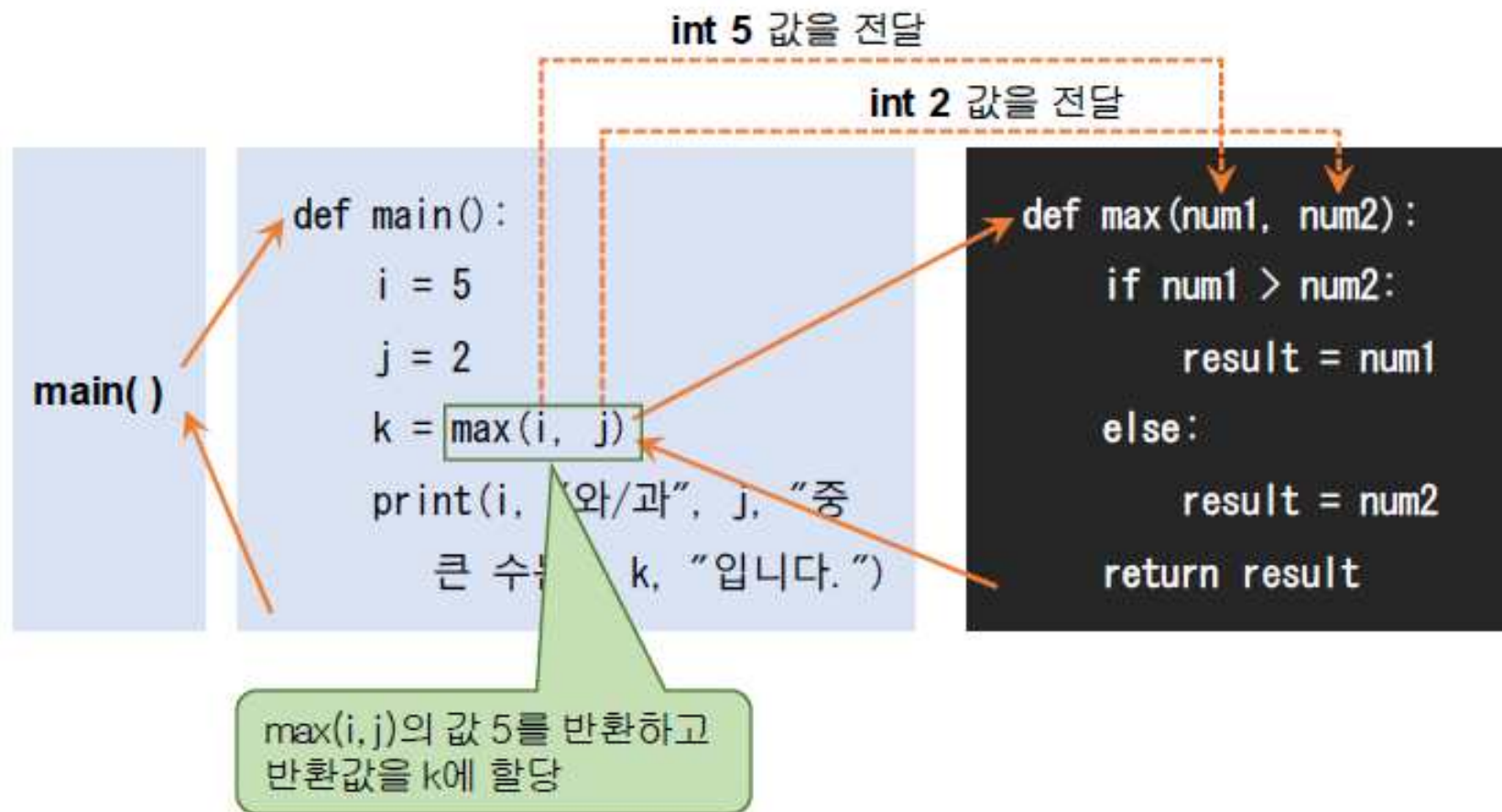
함수 호출 트레이스



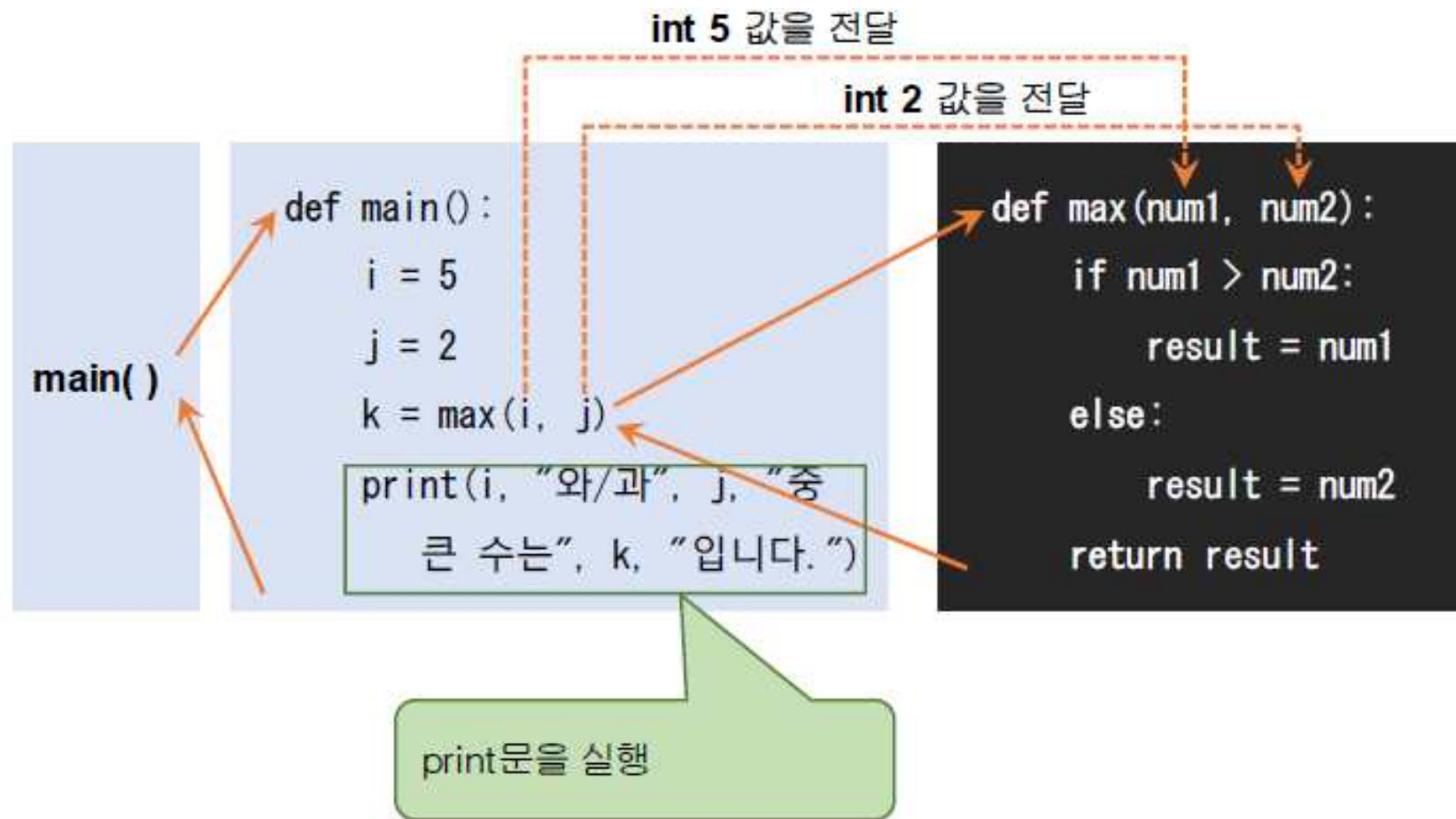
함수 호출 트레이스



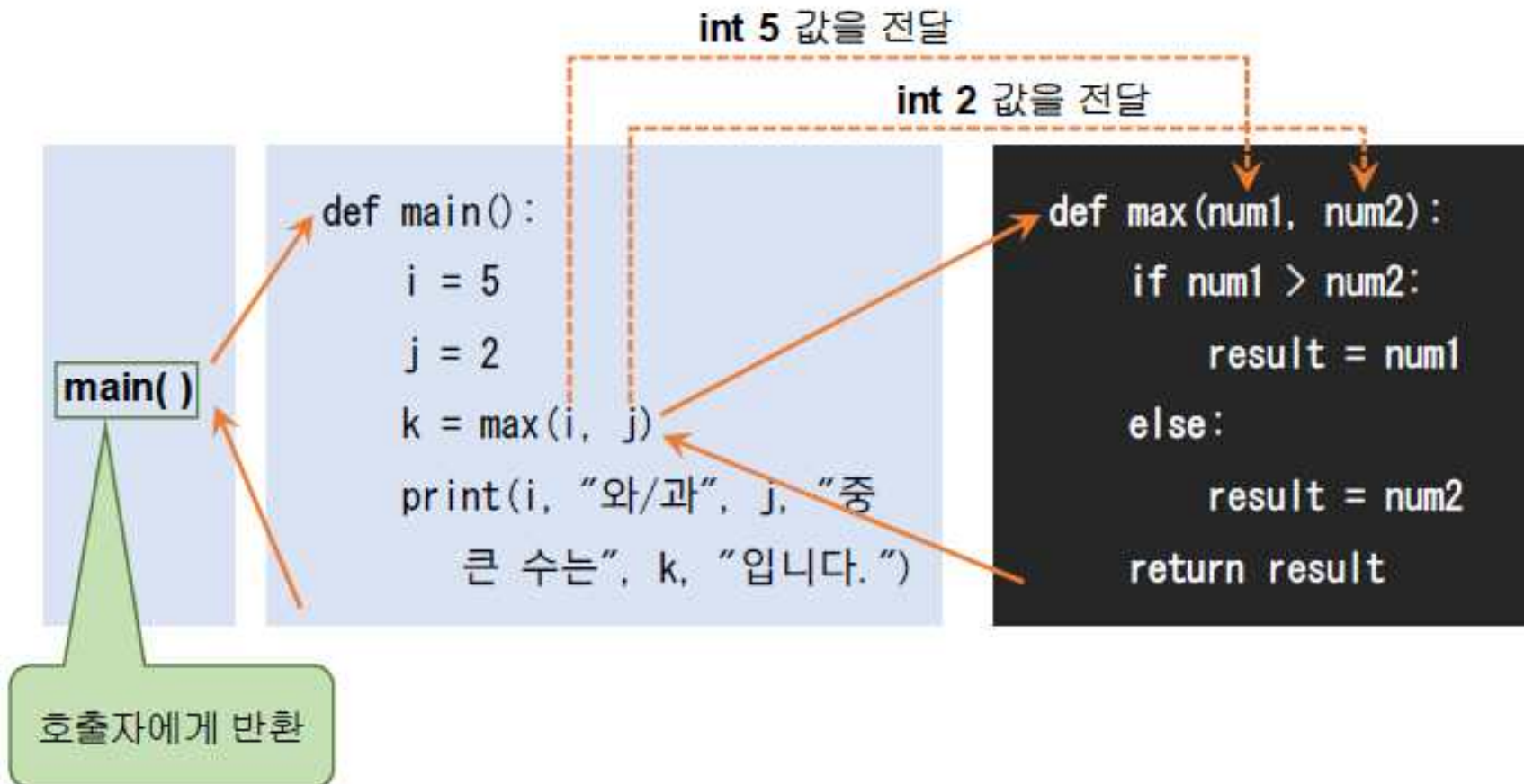
함수 호출 트레이스



함수 호출 트레이스



함수 호출 트레이스



함수의 반환값 유무

- void 함수
 - 값을 반환하지 않는 함수
- 파이썬에서는 모든 함수는 프로그램 상에 return 키워드를 사용하여 값을 반환하지 않는 경우 특별한 값인 None을 기본적으로 반환

```
def sum(number1, number2):  
    total = number1 + number2  
print(sum(1, 2))
```

sum 함수는 return으로 값을
반환하지 않는 함수

함수의 반환값 유무

- return 명령문은 어떤 값도 반환하지 않는 함수에서 제어의 정상 흐름을 우회하는 데 사용되기도 함

```
def printGrade(score):  
    if score < 0 or score > 100:  
        print("잘못된 점수입니다.")  
        return    # return None과 동일  
  
    if score >= 90.0:  
        print('A')  
  
    elif score >= 80.0:  
        print('B')  
  
    else:  
        print('C')
```

위치 인자와 키워드 인자

- 위치 인자: 위치에 따른 인자 전달
 - 함수 호출 시 호출되는 함수에 인자를 전달할 때 매개변수와 순서, 타입, 개수가 반드시 동일해야 함

```
def nPrintln(message, n):  
    for i in range(0, n):  
        print(message)
```

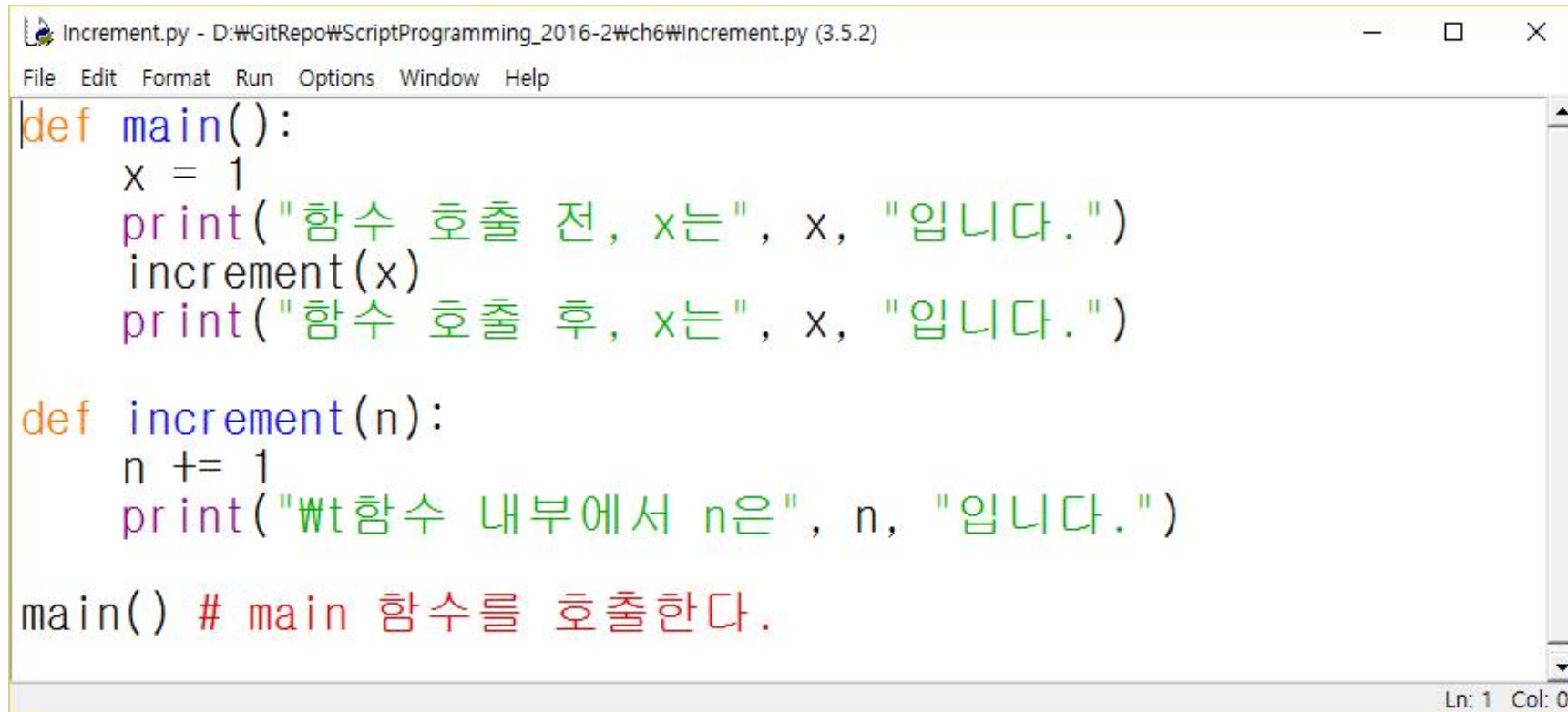
- 다음과 같은 구문으로 함수를 호출할 경우 출력값은 무엇일까?
 - nPrintln("파이썬에 오신것은 환영합니다", 5)
- 다음과 같은 구문으로 함수를 호출할 경우 출력값은 무엇일까?
 - nPrintln("파이썬에 오신것은 환영합니다", 15)
- 다음 구문은 무엇이 잘못되었는가?
 - nPrintln(4, "파이썬에 오신것은 환영합니다")

위치 인자와 키워드 인자

- 키워드 인자
 - `name = value` 형태로 인자를 전달
- 다음 구문은 가능할까?
 - `nPrintln(n = 4, message = “컴퓨터공학부”)`
- 위치 인자와 키워드 인자의 혼용
 - 위치 인자는 키워드 인자가 나온 이후에는 사용 불가

```
def func(p1, p2, p3):  
  
func(30, p2 = 4, p3 = 10)  
func(30, p2 = 4, 10)
```

참조값에 의한 인자 전달



```
Increment.py - D:\GitRepo\ScriptProgramming_2016-2\ch6\Increment.py (3.5.2)
File Edit Format Run Options Window Help

def main():
    x = 1
    print("함수 호출 전, x는", x, "입니다.")
    increment(x)
    print("함수 호출 후, x는", x, "입니다.")

def increment(n):
    n += 1
    print("함수 내부에서 n은", n, "입니다.")

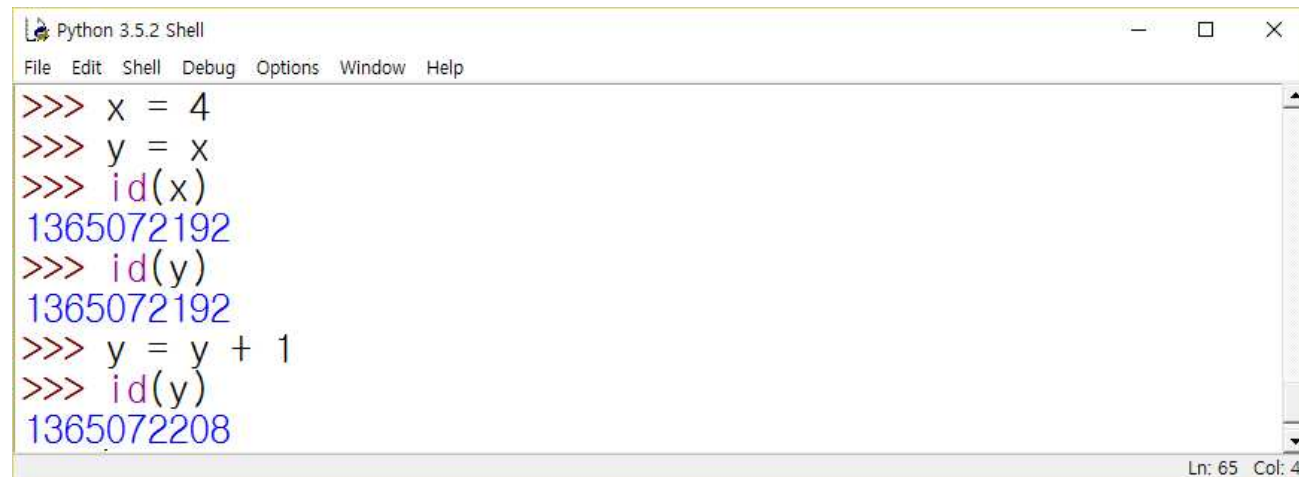
main() # main 함수를 호출한다.
```

Ln: 1 Col: 0

참조값에 의한 인자 전달

- 함수가 호출될 때 인자의 값이 매개변수로 전달
 - 인자의 값: 객체의 참조값
 - 파이썬에서 모든 데이터는 객체이기 때문에 객체에 대한 변수는 실제로 그 객체에 대한 참조값을 가지고 있음
 - 인자가 숫자나 문자열이면 함수 내부에서 매개변수의 값이 수정되더라도 인자에는 아무런 영향을 주지 않음

- 새로운 숫자가 변수에 할당될 때마다 파이썬은 그 숫자에 대한 새로운 객체를 생성하고 이 객체의 참조값을 변수에 할당



```
Python 3.5.2 Shell
File Edit Shell Debug Options Window Help
>>> x = 4
>>> y = x
>>> id(x)
1365072192
>>> id(y)
1365072192
>>> y = y + 1
>>> id(y)
1365072208
Ln: 65 Col: 4
```

코드 모듈화하기

- 함수

- 코드 중복을 줄이고 코드 재사용을 위한 목적으로 사용
- 코드를 모듈화하고 프로그램의 품질을 향상시키기 위해 사용

- 모듈화

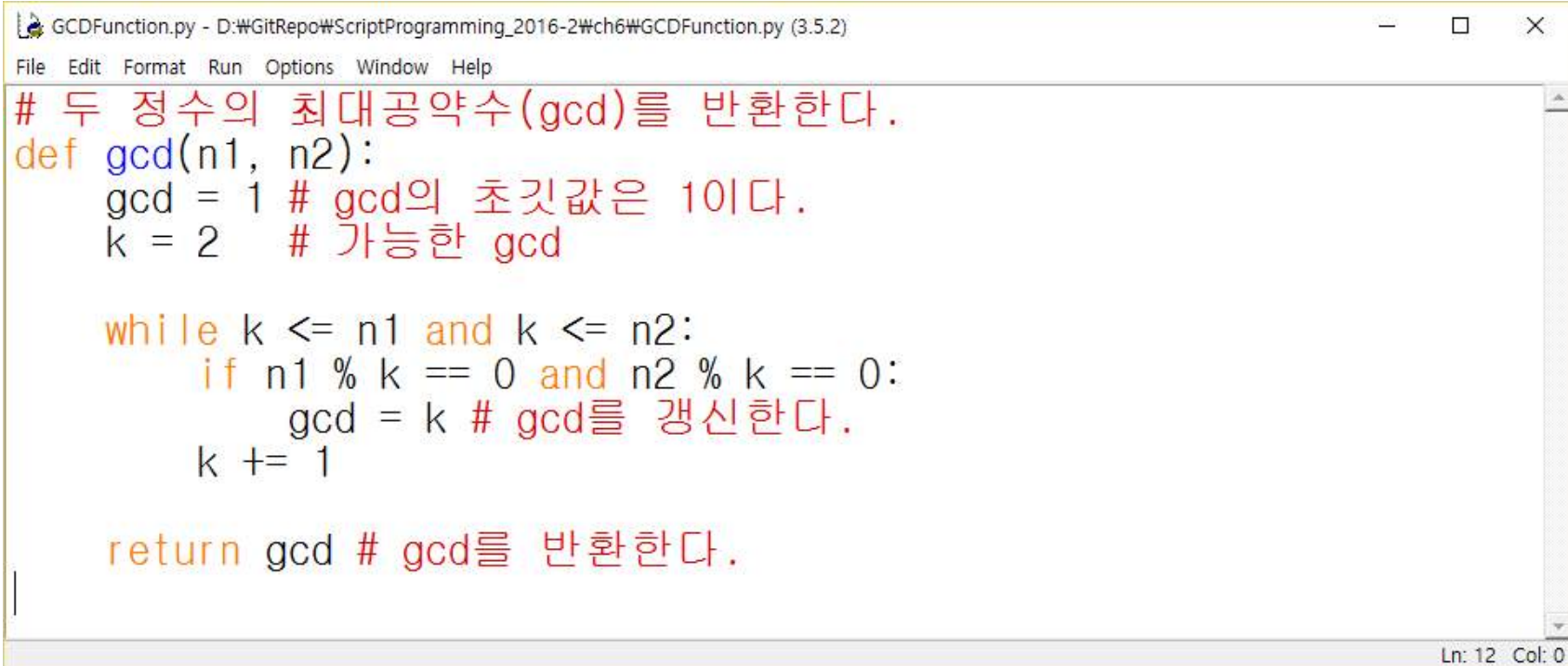
- 큰 프로그램을 독립된 작은 단위로 나누는 것이라고 볼 수 있음
- 코드 재사용 가능
- 코드 관리 및 디버깅을 용이하게 해줌

- 파이썬 모듈

- turtle, random, math, time
- 함수 정의를 파일 확장자 .py를 갖는 모듈 파일에 넣어 놓을 수 있음
- 이 모듈은 재사용을 위해 프로그램 내부로 임포트 할 수 있음
- 하나의 모듈에 하나 이상의 함수를 가질 수 있으며, 모듈 내부 함수는 서로 다른 이름을 가져야 함

코드 모듈화하기 예제

- 사용자로부터 2개 정수를 입력 받아 최대공약수를 구하는 프로그램
- 최대공약수를 구하는 함수를 모듈로 정의



```
GCDFunction.py - D:\GitRepo\ScriptProgramming_2016-2\ch6\GCDFunction.py (3.5.2)
File Edit Format Run Options Window Help
# 두 정수의 최대공약수(gcd)를 반환한다.
def gcd(n1, n2):
    gcd = 1 # gcd의 초깃값은 1이다.
    k = 2   # 가능한 gcd

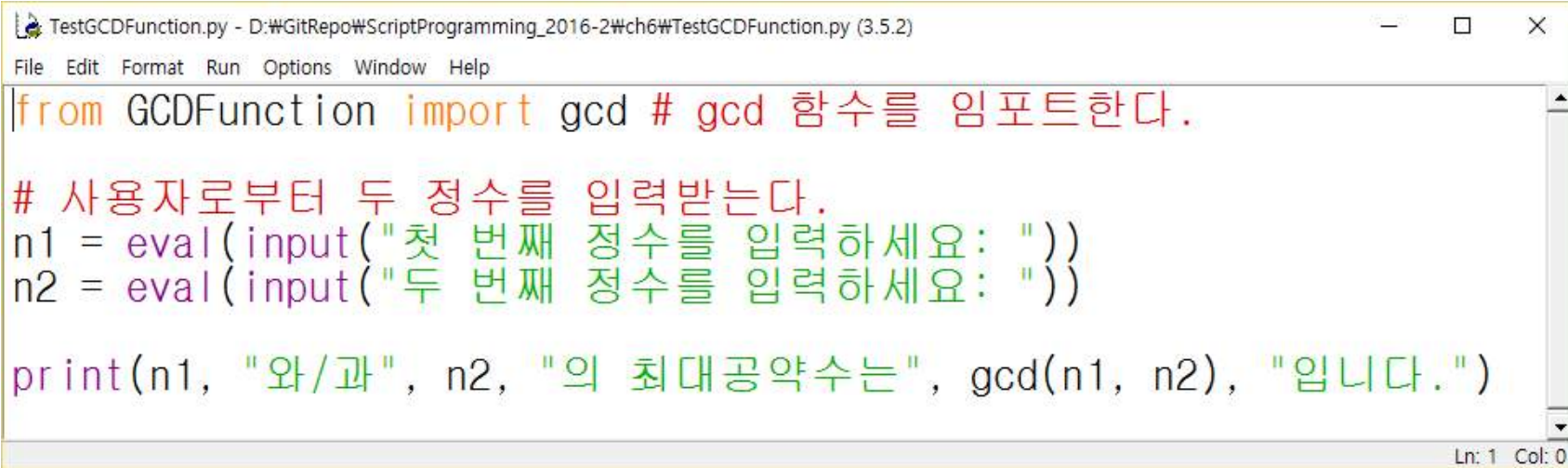
    while k <= n1 and k <= n2:
        if n1 % k == 0 and n2 % k == 0:
            gcd = k # gcd를 갱신한다.
            k += 1

    return gcd # gcd를 반환한다.

Ln: 12 Col: 0
```


코드 모듈화하기 예제

- 모듈화된 gcd 함수 사용



```
TestGCDFunction.py - D:\GitRepo\ScriptProgramming_2016-2\ch6\TestGCDFunction.py (3.5.2)
File Edit Format Run Options Window Help
from GCDFunction import gcd # gcd 함수를 임포트한다.

# 사용자로부터 두 정수를 입력받는다.
n1 = eval(input("첫 번째 정수를 입력하세요: "))
n2 = eval(input("두 번째 정수를 입력하세요: "))

print(n1, "와/과", n2, "의 최대공약수는", gcd(n1, n2), "입니다.")
```

- 앞의 GCDFunction.py 파일은 이 프로그램과 같은 디렉토리에 있어야 함

코드 모듈화하기 예제

- 장점

- 최대공약수 계산 문제를 프로그램 내의 나머지 코드와 분리할 수 있음. 그래서 논리가 명확해지고 프로그램이 읽기 쉬워짐
- 최대공약수 계산에서 발생하는 오류를 gcd 함수로 한정할 수 있음. 이는 디버깅 범위를 좁혀줌
- gcd 함수를 다른 함수/프로그램에서 재사용할 수 있음

변수의 스코프

- 스코프
 - 변수가 참조될 수 있는 프로그램의 영역
- 지역변수 (local variable)
 - 함수 내부에서 생성된 변수
 - 함수 내부에서만 접근될 수 있음
 - 지역변수의 스코프는 지역변수가 생성된 지점부터 그 변수를 포함하고 있는 함수의 끝까지임
- 전역변수 (global variable)
 - 전역변수는 모든 함수의 외부에 생성되며 모든 함수에서 접근 가능

변수 스코프 예제

```
globalVar = 1
def f1():
    localVar = 2
    print(globalVar)
    print(localVar)
f1()
print(globalVar)
print(localVar) # 스코프 밖이므로 오류가 발생한다.
```

```
x = 1
def f1():
    x = 2
    print(x) # 2가 출력된다.
f1()
print(x) # 1이 출력된다.
```

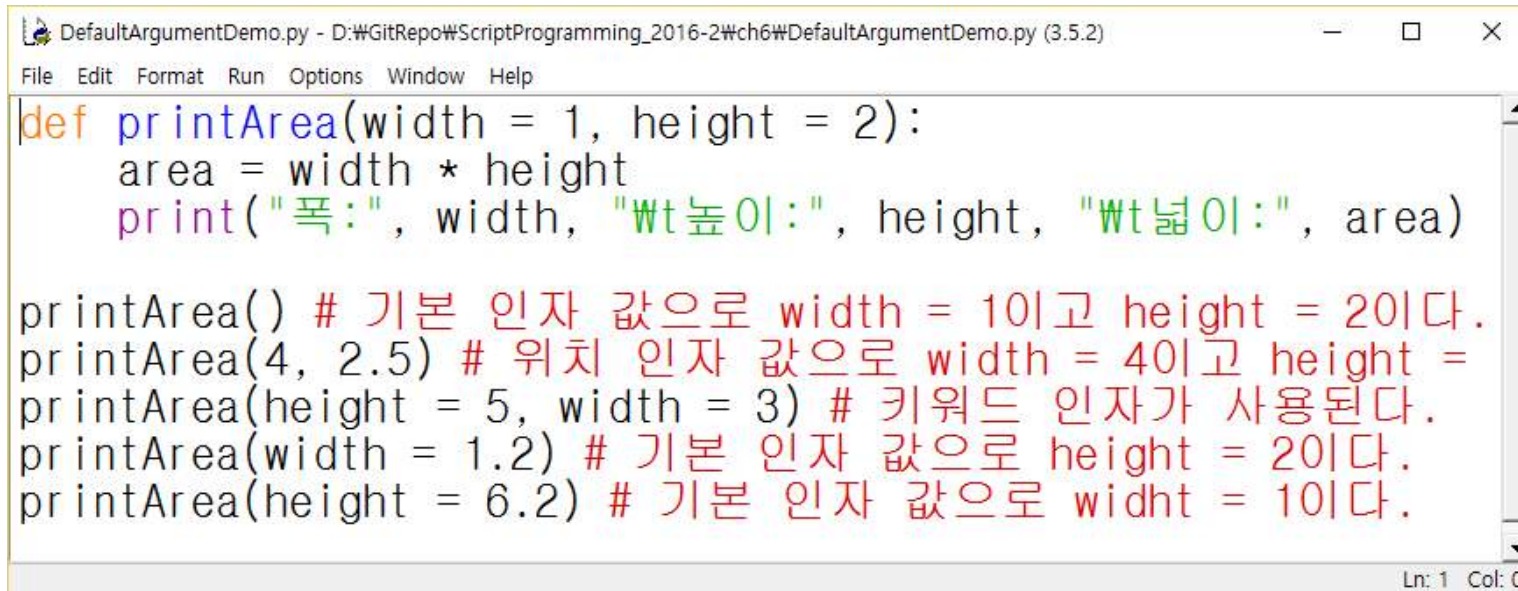
```
x = eval(input("숫자를 입력하세요: "))
if (x > 0):
    y = 4
print(y) # y가 생성되지 않으면, 오류가 발생한다.
```

```
sum = 0
for i in range(0, 5):
    sum += i
print(i)
```

```
x = 1
def increase():
    global x
    x = x + 1
    print(x) # 2가 출력된다.
increase()
print(x) # 2가 출력된다.
```

기본 인자

- 파이썬에서는 기본 인자 값을 가진 함수 정의를 허용
- 함수가 인자 없이 호출되는 경우 기본 값이 매개변수로 전달됨



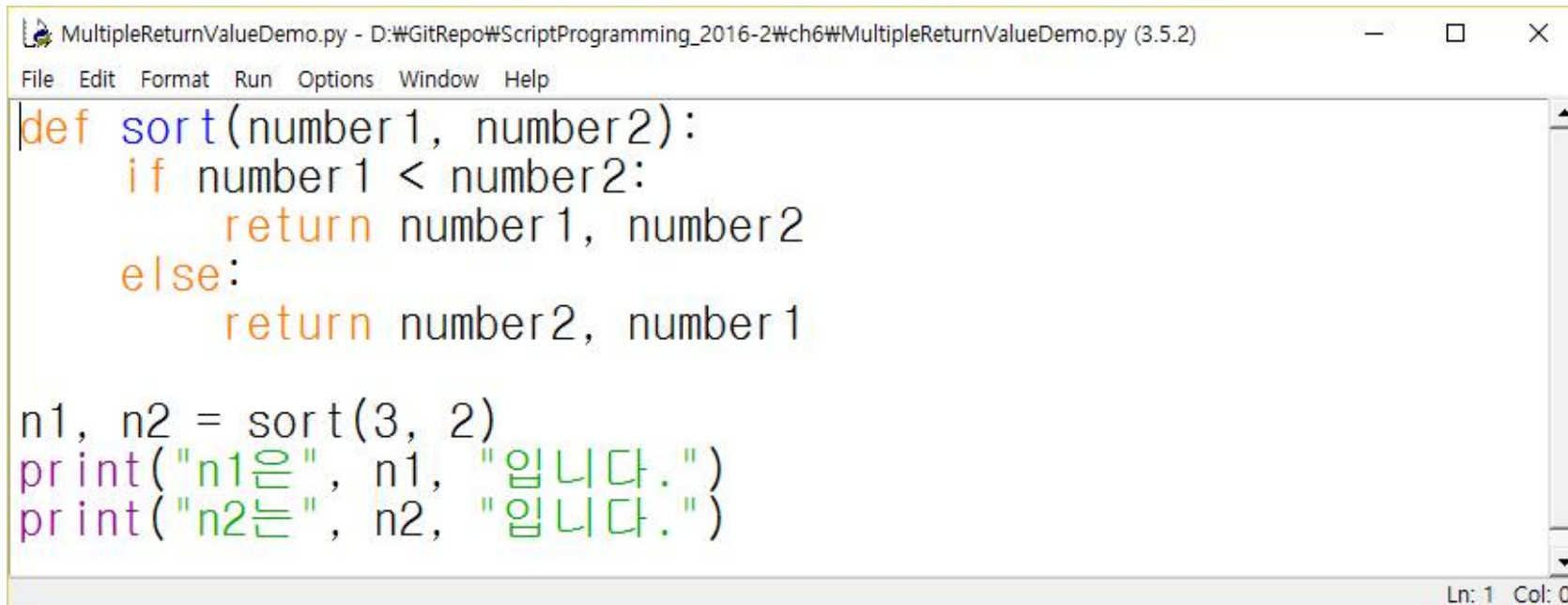
```
DefaultArgumentDemo.py - D:\GitRepo\ScriptProgramming_2016-2\ch6\DefaultArgumentDemo.py (3.5.2)
File Edit Format Run Options Window Help
def printArea(width = 1, height = 2):
    area = width * height
    print("폭:", width, "Wt높이:", height, "Wt넓이:", area)

printArea() # 기본 인자 값으로 width = 1이고 height = 2이다.
printArea(4, 2.5) # 위치 인자 값으로 width = 4이고 height = 2.5이다.
printArea(height = 5, width = 3) # 키워드 인자가 사용된다.
printArea(width = 1.2) # 기본 인자 값으로 height = 2이다.
printArea(height = 6.2) # 기본 인자 값으로 width = 1이다.
```

Ln: 1 Col: 0

다중값 반환하기

- 파이썬의 return 명령문은 다중값을 반환할 수 있음
 - 두개 이상의 값을 반환할 수 있음
 - 다중값을 반환하는 경우 동시 할당으로 전달되어야 함



```
MultipleReturnValueDemo.py - D:\GitRepo\ScriptProgramming_2016-2\ch6\MultipleReturnValueDemo.py (3.5.2)
File Edit Format Run Options Window Help
def sort(number1, number2):
    if number1 < number2:
        return number1, number2
    else:
        return number2, number1

n1, n2 = sort(3, 2)
print("n1은", n1, "입니다.")
print("n2는", n2, "입니다.")
Ln: 1 Col: 0
```

사례 연구: 랜덤 ASCII 문자 생성하기

- 모든 ASCII 문자는 0과 127 사이의 고유한 ASCII 코드를 가짐
- ASCII 문자를 랜덤하게 생성
 - 우선 0과 127 사이의 랜덤 정수를 생성
 - 그 정수에 대한 문자를 얻음 - chr 함수 사용
- ch1과 ch2 사이의 랜덤 문자 생성, 랜덤 소문자 생성, 랜덤 대문자 생성, 랜덤 숫자 문자 생성 등의 기능을 수행하는 함수를 만들어보자