

# 13. 함수

1. 함수 기초 이해하기
2. 함수 정의와 함수 호출
3. 함수의 매개변수와 인수의 기본 형태
4. 함수의 반환값 (Return Value)
5. 지역변수 (Local Variable)와 전역변수 (Global Variable)
6. 매개변수와 입력 인수의 다양한 형태
7. 람다 함수
8. 파이썬 내장 함수
9. 정리

# 13장. 함수

## ◆ 파이썬 함수

- 내장함수 - 파이썬이 미리 만들어서 제공하는 함수. 어떤 내장 함수들이 있고 각 내장 함수들은 어떻게 사용해야 하는지를 공부해야 함.
- 사용자 정의 함수 - 프로그래머가 직접 만드는 함수.

### 내장 함수 목록

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

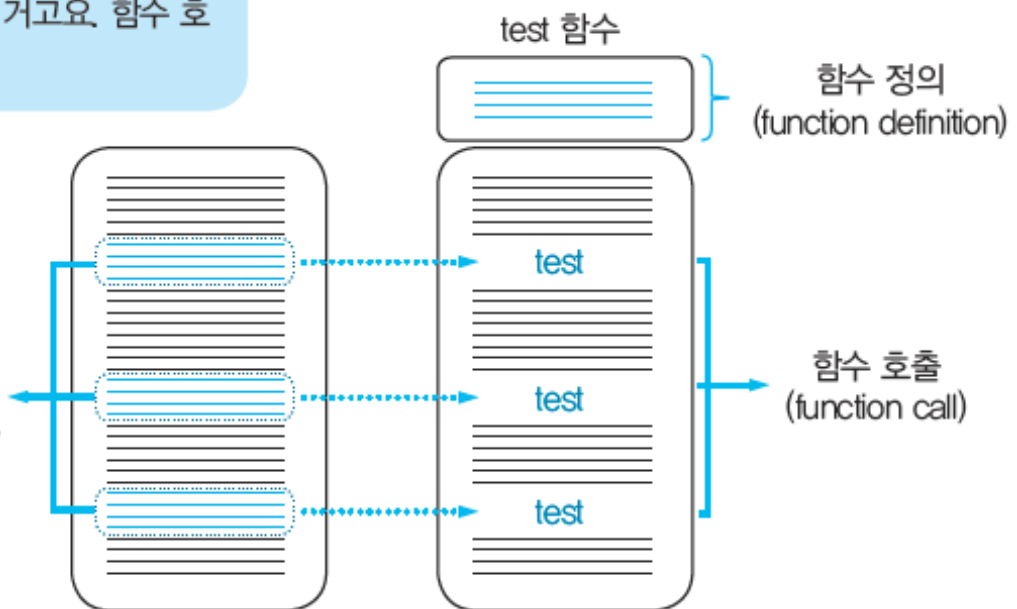
# 1. 함수 기초 이해하기

## ◆ 함수 정의와 호출

함수 정의는 함수를 만들어 놓는 거예요. 함수 호출은 만들어 놓은 함수를 불러서 사용하는 거고요. 함수 호출이 있어야 함수가 수행됩니다.

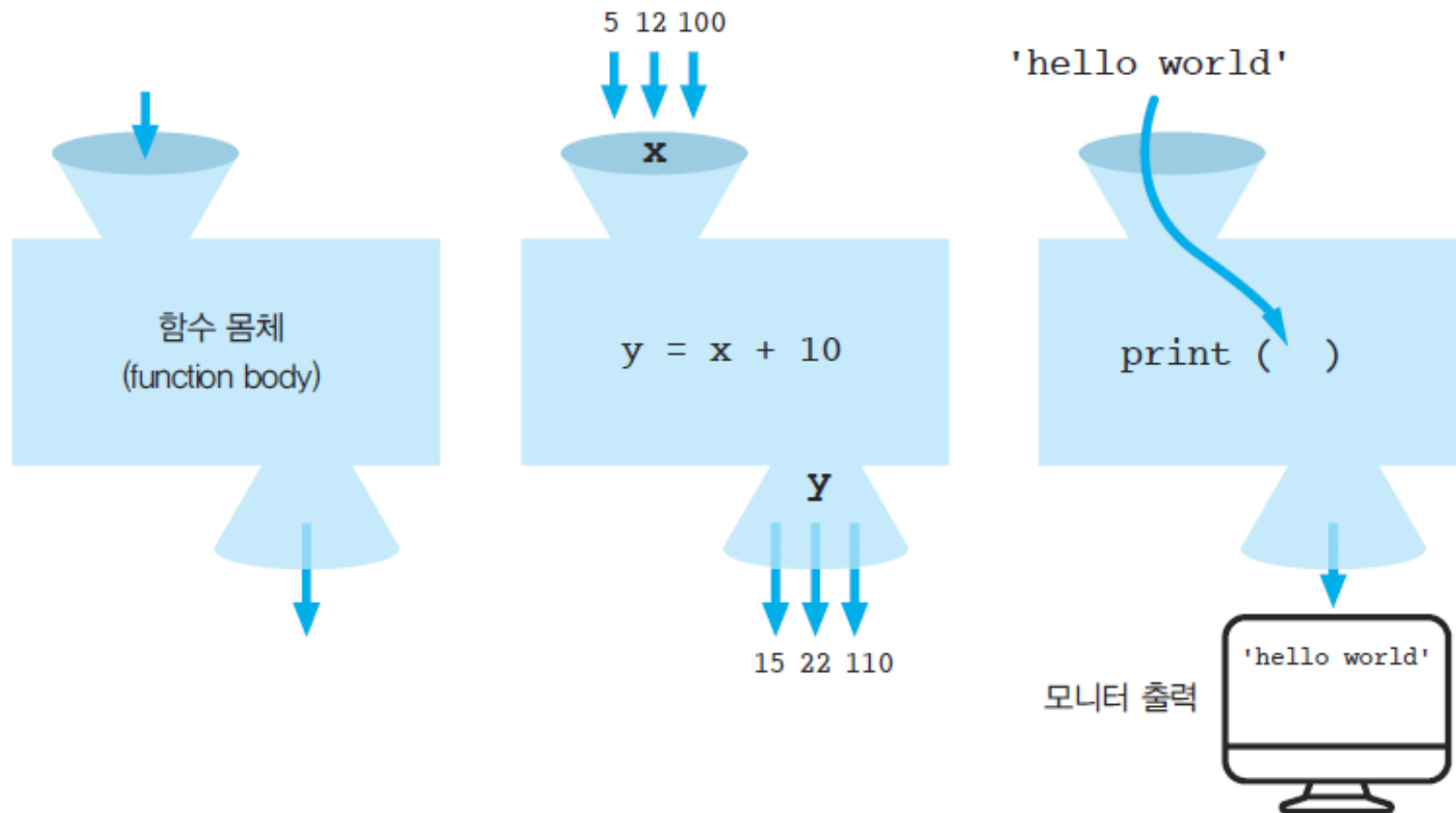


같은 코드가 반복됨.  
test 함수로 분리함.



# 1. 함수 기초 이해하기

- ◆ 함수는 블랙 박스라고 함.



## 2. 함수 정의와 함수 호출

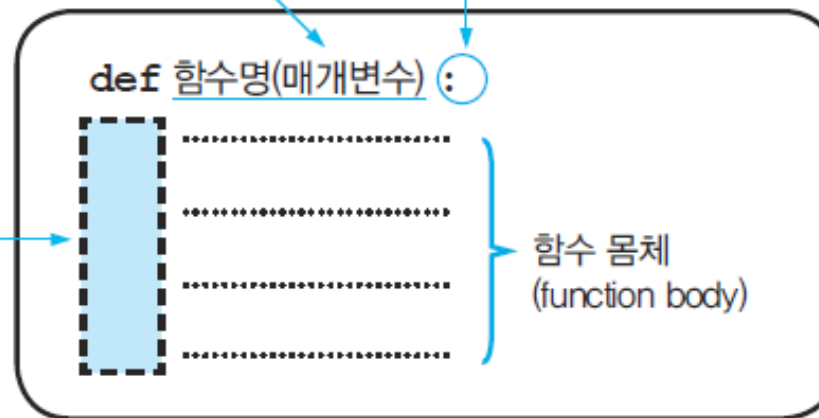
### ◆ 함수 정의하기

괄호 안은 비어 있기도 하고 변수가 오기도 합니다.  
이 변수는 '매개변수(parameter)'라고 부릅니다.

반드시 콜론으로 끝나야 합니다.

← 함수 정의  
(Function Definition)

반드시 인덴트되어야 합니다.  
인덴트되지 않았으면 [Tab] 키를  
이용하여 인덴트합니다.



간단한 함수 정의 예

```
def hello():
    print("Hi!")
    print("Nice to meet you")
```

## 2. 함수 정의와 함수 호출

### ◆ 함수 정의와 호출 예

코드	결과
<div>함수 정의</div> <pre> 1 def hello(): 2     print("Hi!") 3     print("Nice to meet you")  4 print("before function hello call") 5 hello()    # 함수 hello() 호출 6 print("after function hello call") 7 print("call one more time") 8 hello()    # 함수 hello() 호출 </pre> <div> <div>제일 먼저 수행</div> <div> <div>함수 호출할 때는 hello()라고 해야 합니다.</div> </div> </div>	<pre> before function hello call Hi! Nice to meet you after function hello call call one more time Hi! Nice to meet you </pre>

## 2. 함수 정의와 함수 호출

### ◆ 함수 정의와 호출 예

**코드****결과**

함수 호출하기 전에 호출하면 컴퓨터가 에러를 냅니다.

```
print('before hello')
hello()  # 함수 호출
print('after hello')
```

함수 정의

```
def hello():
    print("Hi!")
    print("Nice to meet you")
```

before hello  
Traceback (most recent call last):  
.....  
 hello()  
NameError: name 'hello' is not defined

## 2. 함수 정의와 함수 호출

### ◆ 함수 정의와 호출 예

코드

```
print('hello world')
a = 10
b = 20

def hello():    # 함수 정의
    print('Hi!')
    print('Nice to meet you')

c = a + b
print(a, b, c)
hello()        # 함수 호출
print('after hello')
```

결과

```
hello world
10 20 30
Hi!
Nice to meet you
after hello
```



## 2. 함수 정의와 함수 호출

### ◆ main - 코드에서 수행되는 첫 부분

일반적으로 프로그램에서 코드가 수행되는 첫 부분을 main 코드라고 명시합니다.



main  
시작

함수가 없는 코드

```
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____  
_____
```

main  
시작

함수가 맨 처음 나오는 코드

```
def 함수():  
    _____  
    _____  
    _____  
    _____  
    _____  
    _____  
    _____  
    _____  
    _____  
    _____
```

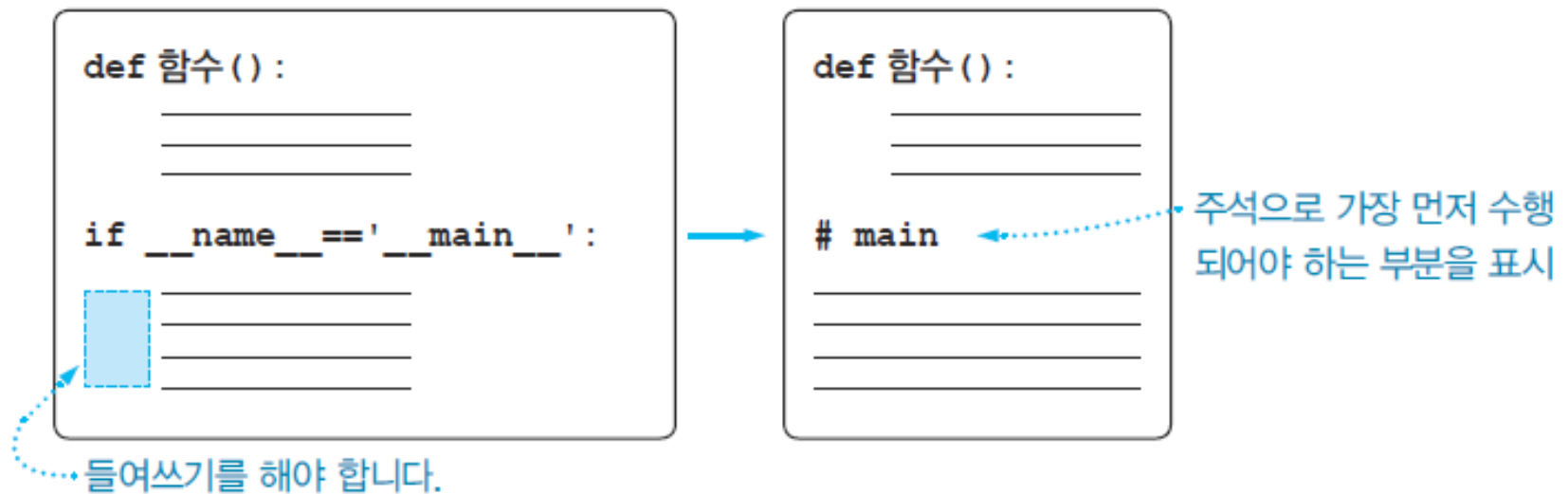
main  
시작

함수가 중간에 있는 코드

```
_____  
_____  
_____  
_____  
def 함수():  
    _____  
    _____  
    _____  
    _____  
    _____  
    _____
```

## 2. 함수 정의와 함수 호출

### ◆ if \_\_name\_\_ == '\_\_main\_\_':



### 3. 함수의 매개변수와 인수의 기본 형태

#### ◆ 매개변수 (parameter)

- 함수의 입력 부분
- 함수를 호출할 때 넘기는 값을 받는 변수

#### ◆ 인수 (argument)

- 함수를 호출할 때 넘기는 값

### 3. 함수의 매개변수와 인수의 기본 형태

#### 코드

```

1 def hello(name):
2     print("Hi!", name)
3     print("Nice to meet you")

# main  ←----- 여기에서 프로그램이 시작한다는 표현으로 주석을 넣었습니다.
4 print("Greet to Alice")
5 hello("Alice")
6 print()
7 print("Greet to David")
8 hello("David")

```

name은 매개변수입니다.

'Alice'가 hello() 함수의 인수로 넘어갑니다.

'David'가 hello() 함수의 인수로 넘어갑니다.

#### 결과

Greet to Alice  
Hi! Alice  
Nice to meet you

Greet to David  
Hi! David  
Nice to meet you



### 3. 함수의 매개변수와 인수의 기본 형태

**CODE 69** main 코드에서 세 개의 자연수 x, y, z를 입력받습니다. x, y, z를 함수 print\_largest( ) 에 넘겨서 x, y, z 중에서 가장 큰 값을 출력하는 프로그램을 작성합니다

```
def print_largest(a, b, c):
    if a > b: max_value = a
    else: max_value = b
    if c > max_value: max_value = c
    print('max value of ({}, {}, {}) - {}'.format(a, b, c, max_value))

# main
x = int(input('Enter x : '))
y = int(input('Enter y : '))
z = int(input('Enter z : '))
print_largest(x, y, z)
```

[결과 1]

```
Enter x : 1
Enter y : 3
Enter z : 5
max value of (1,3,5) - 5
```

[결과 2]

```
Enter x : 7
Enter y : 6
Enter z : 5
max value of (7,6,5) - 7
```

[결과 3]

```
Enter x : 3
Enter y : 5
Enter z : 4
max value of (3,5,4) - 5
```

### 3. 함수의 매개변수와 인수의 기본 형태

**CODE 70** main 코드에서 두 개의 자연수 x와 y를 입력받습니다. x, y를 함수 print\_star( )에 넘겨서 함수에서 x행 y열의 '\*'를 출력하는 프로그램을 작성해 볼게요.

```
def print_star(a, b):  
    for i in range(a):  
        print('*' * b)
```

```
# main  
x = int(input('Enter x : '))  
y = int(input('Enter y : '))  
print_star(x, y)
```

[결과 1]

```
Enter x : 2  
Enter y : 10  
*****  
*****
```

[결과 2]

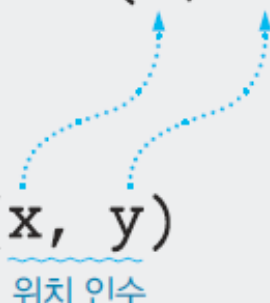
```
Enter x : 4  
Enter y : 5  
*****  
*****  
*****  
*****
```

### 3. 함수의 매개변수와 인수의 기본 형태

#### ◆ 위치 인수 (positional argument)

- 매개변수의 개수와 함수를 호출할 때 넘기는 인수의 개수는 같아야 함.
- 인수가 차례대로 매개변수의 값이 됨.
- 이런 인수를 ‘위치 인수’라고 함.

```
def print_star(a, b):  
  
# main  
print_star(x, y)
```



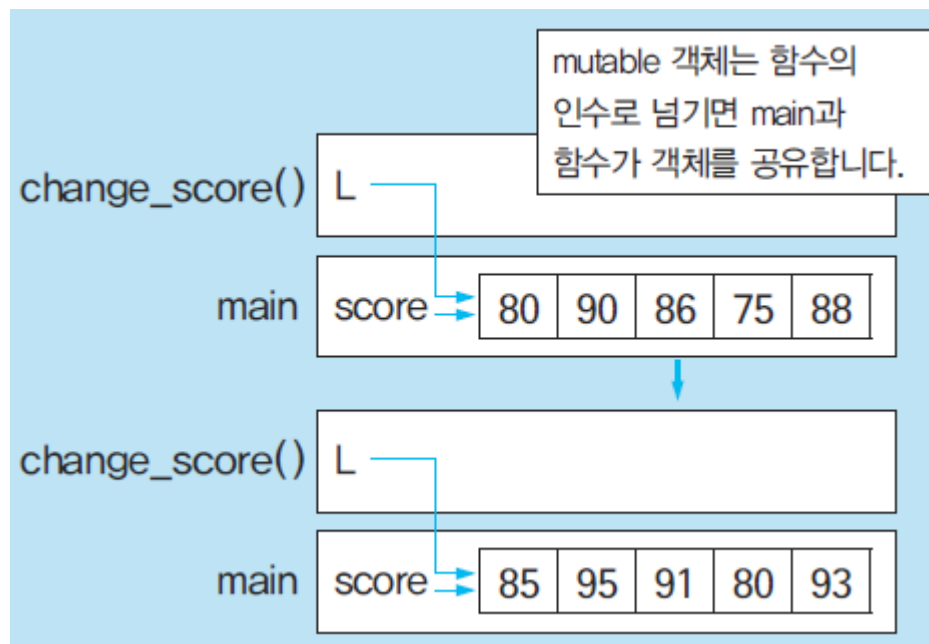
위치 인수

### 3. 함수의 매개변수와 인수의 기본 형태

#### ◆ mutable 객체를 함수의 인수로 넘기는 경우

- 리스트, 집합, 사전이 인수로 넘어가는 경우, main과 함수가 객체를 공유함.
- 함수에서 객체를 수정하면 main에서도 수정된 객체를 갖게 됨.

```
def change_score(L):  
    i = 0  
    while i < len(L):  
        L[i] += 5  
        i += 1  
  
# main  
score = [80, 90, 86, 75, 88]  
print(score) # [80, 90, 86, 75, 88]  
change_score(score)  
print(score) # [85, 95, 91, 80, 93]
```

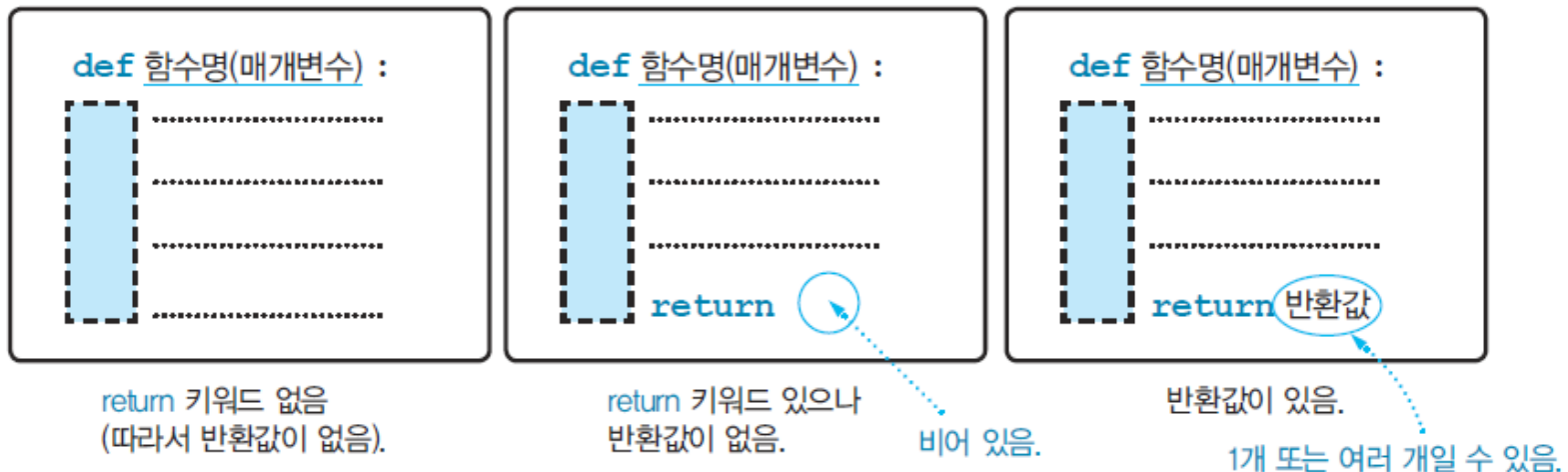




## 4. 함수의 반환값(Return Value)

### ◆ 함수의 반환값

- 함수를 수행한 후의 결과값
- return 키워드를 사용해서 함수의 결과값을 함수를 호출한 자리로 반환함.
- return 키워드는 **함수를 끝내고 함수를 호출한 자리로 돌아가라**는 의미임.
- return 키워드 옆에는 아무 것도 없기도 하고, 여러 값들이 오기도 함.



## 4. 함수의 반환값(Return Value)

return 키워드가 있는 경우	return 키워드가 없는 경우
<pre> 1 def with_return(): 2     print("inside function") 3     return  # 함수를 무조건 끝냄. 4     print("last line of function") # main 5 print('start of the program') 6 with_return() 7 print('after return') </pre> <p>이 줄은 출력될 수 없습니다.</p>	<pre> def without_return():     print("inside function")     print("last line of function")  # main print('start of the program') without_return() print('after return') </pre>
<p>&lt; 호출 결과 &gt;</p> <p>start of the program inside function after return</p>	<p>&lt; 호출 결과 &gt;</p> <p>start of the program inside function last line of function after return</p>

## 4. 함수의 반환값(Return Value)

### ◆ 반환값이 한 개인 경우

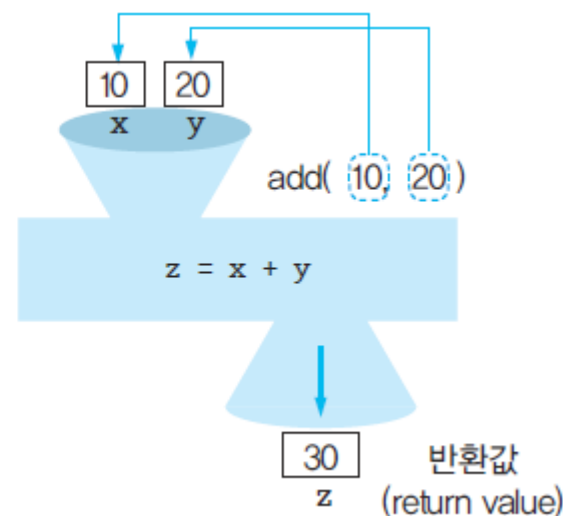
```

1 def add(x, y):
2     z = x + y
3     return z
    
```

반환값 z는 add()를 호출한 자리에 반환됩니다.

```

# main
4 a = add(10, 20) # a는 add의 반환값 받을 변수
5 print(a)
    
```



```

def add(x, y):
    z = x = y
    return z
    
```

a = add(10, 20)  
print(a)

(1)

```

def add(x, y):
    z = x = y
    return z
    
```

a = add(10, 20)  
print(a)

(2)

```

def add(x, y):
    z = x = y
    return z
    
```

30  
a = add(10, 20)  
print(a)

(3)

```

def add(x, y):
    z = x = y
    return z
    
```

30  
a = add(10, 20)  
print(a)

(4)

## 4. 함수의 반환값(Return Value)

**CODE 71** [CODE 69]를 반환값이 있는 예제로 바꾸어서 작성해 보겠습니다. main 코드에서 세 개의 자연수 x, y, z를 입력받습니다. x, y, z를 함수 print\_largest에 넘겨서 x, y, z 중에서 가장 큰 값을 반환합니다.

```
def print_largest(a, b, c):
    if a > b: max_value = a
    else: max_value = b
    if c > max_value: max_value = c
    return max_value

# main
x = int(input('Enter x : '))
y = int(input('Enter y : '))
z = int(input('Enter z : '))
result = print_largest(x, y, z)
print('max value of ({}, {}, {}) - {}'.format(x, y, z, result))
```

[결과 1]

```
Enter x : 1
Enter y : 3
Enter z : 5
max value of (1,3,5) - 5
```

[결과 2]

```
Enter x : 7
Enter y : 6
Enter z : 5
max value of (7,6,5) - 7
```

[결과 3]

```
Enter x : 3
Enter y : 5
Enter z : 4
max value of (3,5,4) - 5
```

## 4. 함수의 반환값(Return Value)

### ◆ 반환값이 두 개 이상인 경우

- return 키워드 옆에 반환할 값들을 콤마로 분리하여 적음.

```
def return_two(x, y):
    w = x + y
    z = x * y
    return w, z      # 여러 데이터가 콤마로 분리되어 있으면 튜플로 간주합니다.
```

위의 return\_two() 함수를 호출하는 코드

```
a, b = return_two(10, 20)    # a와 b의 자료형은 int입니다.
print(a,b)
t = return_two(30, 40)      # t의 자료형은 tuple입니다.
print(t)
```

[결과]

```
30 200
(70, 1200)
```

## 4. 함수의 반환값(Return Value)

```
def calc(x, y):
    a = x + y
    b = x - y
    c = x * y
    d = x // y
    e = x % y
    return a, b, c, d, e
```

다음과 같이 다양하게 호출할 수 있음

calc() 호출 형태	결과
<code>v = calc(30, 7)</code>	v는 튜플 자료형으로 (37, 23, 210, 4, 2)입니다.
<code>v1, v2, v3, v4, v5 = calc(30, 7)</code>	v1, v2, v3, v4, v5는 모두 정수 자료형입니다. 각각 37, 23, 210, 4, 2를 갖습니다.
<code>v, *w = calc(30, 7)</code>	*를 붙이면 나머지 모든 값을 의미하고 리스트 형으로 처리합니다. v는 정수형 변수로 37입니다. w는 리스트형 변수로 [23, 210, 4, 2]입니다.
<code>*v, w = calc(30, 7)</code>	v는 리스트형 변수로 [37, 23, 210, 4]입니다. w는 정수형 변수로 2입니다.
<code>u, *v, w = calc(30, 7)</code>	u는 정수형 변수로 37입니다. v는 리스트형 변수로 [23, 210, 4]입니다. w는 정수형 변수로 2입니다.

## 4. 함수의 반환값(Return Value)

\*를 사용할 때 애매모호하게 쓰면 안돼요!



`v = a, b, c, d, e`

`v1, v2, v3, v4, v5 = a, b, c, d, e`

`v, *w = a, b, c, d, e`

`u, *v, w = a, b, c, d, e`

## 4. 함수의 반환값(Return Value)

```
def calc(x, y):
    a = x + y
    b = x - y
    c = x * y
    d = x // y
    e = x % y
    return a, b, c, d, e
```

잘못 호출한 형태

calc() 호출 형태	에러인 이유
v1, v2, v3, v4 = calc(30, 7)	'=' 왼쪽에 하나씩 받을 때에는 반드시 반환하는 데이터의 개수와 같은 수의 변수가 있어야 합니다.
*v, *w = calc(30, 7)	'=' 왼쪽에 * 표현이 두 개 올 수가 없습니다.



## 4. 함수의 반환값(Return Value)

**CODE 72** 리스트 math에는 학생 여러 명의 수학 성적이 저장되어 있습니다. 리스트를 score\_info()라는 함수로 넘겨서, 학생 수, 최고 성적, 최저 성적, 평균을 반환하는 프로그램을 작성해 보겠습니다.

```
def score_info(M):  
    return len(M), max(M), min(M), sum(M)/len(M)
```

```
# main
```

```
math = [90, 80, 95, 83, 75, 99, 72, 90, 65, 88]  
count, mx, mn, avg = score_info(math)  
print('There {} scores in math list.'.format(count))  
print('Max score : {}'.format(mx))  
print('Min score : {}'.format(mn))  
print('Average : {:.2f}'.format(avg))
```

[결과]

There 10 scores in math list.  
Max score : 99  
Min score : 65  
Average : 83.70

## 4. 함수의 반환값(Return Value)

**CODE 73** 함수 count()는 학생들의 수학 성적이 저장된 math 리스트와 성적 x를 입력받습니다. 그리고 math() 리스트에서 x 이상의 성적이 몇 명인지를 찾아서 반환합니다. 코드를 어떻게 적고 있는지 잘 봐두기 바랍니다.

```
def count(M, x):  
    cnt = 0  
    for score in M:  
        if score >= x: cnt += 1  
    return cnt
```

# main

```
math = [90, 92, 85, 80, 77, 68, 73, 81, 65, 88]  
print('{}점 이상인 학생수 : {}'.format(85, count(math, 85)))
```

여기에서 함수를 호출하고 반환값을 반환함.

```
math = [90, 92, 85, 80, 77, 68, 73, 81, 65, 88]  
result = count(math, 85)  
print('{}점 이상인 학생수 : {}'.format(85, result))
```

## 4. 함수의 반환값(Return Value)

### ◆ 반환값이 없는 경우의 None 키워드

- 반환값이 없는 함수들은 특별한 값인 None 키워드를 반환함.
- print() 함수가 반환값이 없는 함수임.

```
>>> x = print('hello world')
hello world
>>> print(x)          # print( ) 함수는 반환값이 없기 때문에 None을 출력합니다.
None
>>> type(x)           # None은 'NoneType'이라는 특별한 타입입니다.
<class 'NoneType'>
```

## 4. 함수의 반환값(Return Value)

아래 두 코드의 결과는 같음.

<pre>def no_return():     print('inside no_return')     print('finish no_return') # main x = no_return() print(x) print(type(x))</pre>	<pre>def no_return():     print('inside no_return')     print('finish no_return')     return None # main x = no_return() print(x) print(type(x))</pre>	<p>[결과]</p> <pre>inside no_return finish no_return None &lt;class 'NoneType'&gt;</pre>
--	--	--

**CODE 74** 함수의 return 값이 있는지 없는지를 체크해 보는 간단한 프로그램을 작성해 볼게요

<pre>def test_return():     print('hello world')     print('have a nice day~') # main if test_return() == None: # 함수의 반환값이 없다면, 참     print('There is no return value')</pre>	<p>[결과]</p> <pre>hello world have a nice day~ There is no return value</pre>
---	--

## 4. 함수의 반환값(Return Value)

### ◆ 함수 안에 return 문이 여러 개인 경우

- return 키워드가 여러 개인 경우에는 가장 먼저 만나는 return에서 함수가 끝남.

```
def return_test():  
    print('return test')  
    return 100    # 여기에서 함수 끝남.  
    return 200
```

return\_test() 함수를 호출하면 100을 반환하고 함수를 끝냅니다.

```
def return_test(x, y):  
    if x >= y:  
        return x  
    else:  
        return y
```

return\_test(x,y) 함수를 호출하면 x와 y 중에서 큰 값을 반환하고 함수를 끝냅니다. 즉, 조건에 따라서 return x와 return y 중에 한 문장만 수행 됩니다.

## 5. 지역변수와 전역변수

### ◆ 지역변수 (local variable)

- 함수 안에서 만든 변수로 함수 내에서만 사용할 수 있음.
- 아래 코드 ⑥에서 NameError가 발생하는 이유는 지역변수 a를 외부에서 사용했기 때문임.

```
① def local_var():  
②     a = 10      # 지역변수 a  
③     print(a)  
    # main  
④ print("start line")  
⑤ local_var()  
⑥ print(a)        # 에러 발생
```

```
[결과]  
start line  
10  
Traceback (most recent call last):  
.....  
    print(a)  
NameError: name 'a' is not defined
```

## 5. 지역변수와 전역변수

### ◆ 지역변수 (local variable)

- 함수의 매개변수도 지역 변수임.
- 함수의 매개변수는 함수를 호출하여 인수를 넘겨받을 때 만들어 지고, 함수 내에서만 사용가능함.

```
def local_var(x):                # 매개변수 x도 함수의 지역변수입니다.
    print('x = {}'.format(x))
# main
print("start line")
a = 10                           # main에서 만든 변수는 전역변수입니다.
local_var(a)
print('a = {}'.format(a))
print('last line x = {}'.format(x)) # 여기에서 에러가 발생합니다(NameError).
```

## 5. 지역변수와 전역변수

### ◆ 지역변수 (local variable)

- 앞의 코드 결과

```
start line
x = 10
a = 10
Traceback (most recent call last):
.....
  print('last line x = {}'.format(x))
NameError: name 'x' is not defined
```

매개변수도 지역변수입니다.

```
def local-var(x):
```

```
# main
```

```
a = 10
```

```
local-var(a)
```

x는 여기에서만 사용할 수 있습니다.

매개변수 x가 만들어지고 x는 10을 갖습니다.



## 5. 지역변수와 전역변수

### ◆ 전역변수 (global variable)

- 어느 함수에도 속하지 않은 변수로 어디에서나 사용 가능함

<pre> ① def test(): ②     a = 10      # a는 지역변수입니다. ③     print(a, x)         # main ④ x = 100        # x는 전역변수로 어디에서나 사용 가능합니다. ⑤ test() ⑥ print(x) </pre>	<p>[결과]</p> <p>10 100 100</p>
---	-----------------------------------

라인 ④와 ⑤의 순서를 바꾼다면, NameError가 발생함.

<pre> def test():     a = 10     print(a, x)  # NameError 발생 # main test() x = 100 print(x) </pre>	<p>[결과]</p> <p>Traceback (most recent call last):</p> <p>.....</p> <p>print(a, x)</p> <p>NameError: name 'x' is not defined</p>
--	---

## 5. 지역변수와 전역변수

### ◆ 지역변수와 전역변수의 이름이 같은 경우

- 규칙 : 지역변수가 전역변수보다 우선 순위가 높음.

<pre> ① def test(): ②     a = 10    # 지역변수 a ③     print(a)  # 지역변수 a의 값 10이 출력됩니다.     # main ④ a = 20        # 전역변수 a ⑤ test() ⑥ print(a)      # 전역변수 a의 값 20이 출력됩니다. </pre>	<p>[결과]</p> <p>10</p> <p>20</p>
--	---------------------------------

- 만약에 위의 코드에서 라인 ②가 라인 ④에 있는 전역 변수 a의 값을 바꾸고자 하는 의도였다면, `global` 키워드를 이용하면 해결됨.

## 5. 지역변수와 전역변수

### ◆ global 키워드

<pre> ① def test(): ②     global a    # 변수 a가 전역변수 a라는 의미입니다. ③     a = 10      # 전역변수 a를 10으로 업데이트합니다. ④     print(a)    # 10이 출력됩니다. # main ⑤ a = 20 ⑥ test() ⑦ print(a)        # 10이 출력됩니다. </pre>	<p>[결과]</p> <p>10</p> <p>10</p>
---	---------------------------------

- 반드시 global 키워드를 선언하고 전역변수를 사용해야 함.

<pre> ① def test(): ②     a = 10 ③     global a    # 에러 발생 ④     print(a) </pre>	<p>변수 a가 global 키워드 전에 할당되었다는 에러 메시지가 나옵니다.</p> <p>(name 'a' is assigned to before global declaration)</p>
--	--

## 5. 지역변수와 전역변수

### ◆ global 키워드

- 함수에서 만든 지역변수를 전역변수로 사용하고자 할 때에도 global 키워드를 사용해야 함.

<pre> ① def test(): ②     a = 10 ③     print(a)    # main ④ test() ⑤ print(a) </pre>	<pre> 10 Traceback (most recent call last): ..... print(a) NameError: name 'a' is not defined </pre>
<pre> ① def test(): ②     global a ③     a = 10 ④     print(a)    # main ④ test() ⑤ print(a) </pre>	<pre> 10 10 </pre>

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 매개변수와 인수의 형태

- 기본값이 있는 매개변수
- 가변 개수의 인수를 받을 수 있는 매개변수
- 키워드 인수 (keyword arguments)
- 가변 개수의 키워드 인수

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 기본값이 있는 매개변수

- 주의점 - 매개변수가 기본값을 갖고 있어서 인수가 넘어 오지 않으면 기본값을 인수로 처리함.

width, length에는 항상 인수를 넣어야 함.

<pre>def volume(width, length, height=10):     return width * length * height # main ① v1 = volume(2, 3, 5) ② v2 = volume(2, 3) print(v1, v2)</pre>	<p>[결과]</p> <p>30 60</p>
---	--------------------------

2는 width, 3은 length로 넘어감. 세 번째 인수는 없기 때문에 height의 기본값인 10으로 처리함.

다음과 같이 인수를 한 개만 넘기면 positional argument가 필요하다는 에러가 발생함.

v = volume(5)	TypeError: volume() missing 1 required positional argument: 'length'
---------------	--

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 기본값이 있는 매개변수

- 주의점 - 함수를 정의할 때 기본값이 없는 매개변수가 기본값이 있는 매개변수보다 뒤에 나오면 안 됨.

기본값이 있는 매개변수  
(default argument)

기본값이 없는 매개변수  
(non-default argument)

```
def volume(width, length=10, height):  
    return width * length * height
```

[결과]  
non-default argument follows default  
argument

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 기본값이 있는 매개변수

- 매개변수가 세 개 있는 경우의 기본값이 있는 예제

모두 위치 인수인 경우	<pre>def volume(width, length, height):     return width * length * height</pre>
기본값이 있는 매개변수가 1개인 경우	<pre>def volume(width, length, height=10):     return width * length * height</pre>
기본값이 있는 매개변수가 2개인 경우	<pre>def volume(width, length=20, height=10):     return width * length * height</pre>
기본값이 있는 매개변수가 3개인 경우	<pre>def volume(width=15, length=20, height=10):     return width * length * height</pre>



## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 기본값이 있는 매개변수 예제

<pre>def volume(width, length=5, height=10):     return width * length * height # main print(volume(2))          # width=2, length=5, height=10 print(volume(2, 3))       # width=2, length=3, height=10 print(volume(2, 3, 4))    # width=2, length=3, height=4</pre>	<p>[결과]</p> <p>100</p> <p>60</p> <p>24</p>
--	--

<pre>def volume(width=1, length=1, height=10):     return width * length * height # main print(volume( ))          # width=1, length=1, height=10 print(volume(2))          # width=2, length=1, height=10 print(volume(2, 3))       # width=2, length=3, height=10 print(volume(2, 3, 4))    # width=2, length=3, height=4</pre>	<p>[결과]</p> <p>10</p> <p>20</p> <p>60</p> <p>24</p>
---	---

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 가변 개수의 인수를 받을 수 있는 매개변수

- 지금까지는 하나의 매개변수의 하나의 인수를 넣었음.
- 하나의 매개변수에 여러 개의 인수를 넣을 수도 있음. 이것이 가변개수의 인수를 받는 매개변수임 → 매개변수에 \* 기호를 붙임.
- 보통 \*args 라고 넣는데, args는 argument의 약자로 프로그래머들이 많이 사용하지만, 다른 이름을 사용해도 됨.

```
def print_all(*args):
    print(args)
# main
print_all()      # 인수를 넣지 않아도 됩니다.
print_all(1)
print_all(1,2)
print_all(1,3,5)
```

[결과]

()  
(1,)
 (1, 2)
 (1, 3, 5)

결과를 튜플로 처리함

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 가변 개수의 인수를 받을 수 있는 매개변수 예제

```
def print_all(*args):
    print(args.count('c'))
# main
print_all('a', 'b', 'c', 'd', 'c', 'a', 'c')
```

튜플 자료형의 count 메소드

[결과]

3

```
def find_larger(n, *args):
    count = 0
    for i in args:
        if i > n: count += 1
    return count
# main
w = find_larger(10)
x = find_larger(10, 1, 15, 20, 3)
y = find_larger(1, 2, 3, 4, 5, 6)
z = find_larger(10, 1, 2, 3)
print(w, x, y, z)
```

# 없음  
# 2개 : 15, 20  
# 5개 : 2, 3, 4, 5, 6  
# 없음  
# 0 2 5 0 출력

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 키워드 인수 (keyword arguments)

- 위치 인수 (positional arguments)

```
def introduce(name, age):  
    print('Hi, I am {} and {} years old'.format(name, age))  
    위치인수  
introduce('Alice', 10)      # 결과 : Hi, I am Alice and 10 years old.
```

- 키워드 인수 (keyword arguments)

- ‘매개변수 = 인수’의 형태로 인수를 넘김.

```
def introduce(name, age):  
    print('Hi, I am {} and {} years old'.format(name, age))  
  
introduce(name='Alice', age=10)    # 결과 : Hi, I am Alice and 10 years old  
introduce(age=7, name='Paul')      # 결과 : Hi, I am Paul and 7 years old
```

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 키워드 인수 (keyword arguments)

- 키워드 인수는 매개변수와 인수를 묶어서 적기 때문에 순서가 중요하지 않음.
- 키워드 인수와 위치 인수를 섞어서 사용할 수도 있음.

```
def introduce(name, age):  
    print('Hi, I am {} and {} years old'.format(name, age))  
  
introduce('Alice', age=10)    # 결과 : Hi, I am Alice and 10 years old
```

```
def introduce(name, age, address):  
  
introduce('Alice', address = 'seoul', age = 10)
```

함수 정의

위치 인수

키워드 인수는 매개변수를 적기 때문에 순서가 중요하지 않아요.

함수 호출

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 키워드 인수 사용시 주의점

- 주의점 1 - 위치 인수와 키워드 인수를 겹치게 사용하면 에러 발생함.

```
def introduce(name, age):  
    print('Hi, I am {} and {} years old'.format(name, age))  
  
introduce(10, name='Alice')
```

에러 메시지 :

`TypeError : introduce( ) got multiple values for argument 'name'`

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 키워드 인수 사용시 주의점

- 주의점 2 - 위치 인수와 키워드 인수를 섞어서 사용할 경우에는 반드시 위치 인수를 모두 적고 난 후에 키워드 인수를 적어야 함.

```
def introduce(name, age):  
    print('Hi, I am {} and {} years old'.format(name, age))
```

```
introduce(name = 'Alice', 10)    # 키워드 인수를 일반 인수보다 먼저 적은 경우
```

에러 메시지 :

SyntaxError : positional argument follows keyword argument

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 가변 개수의 키워드 인수

- 키워드 인수가 가변 개수로 있는 경우
- \*\* 기호를 사용함 (\*\*kwargs로 많이 사용함).

```
def 함수명(**kwargs) :
    함수 문장들
    .....
```

kwargs 말고 다른 이름을 써도 됩니다.

```
① def test(**kwargs):
②     print(kwargs)

# main
③ test(a = 10, b = 20)
④ test(name = "Alice", age = 10)
⑤ test(even = [2,4,6], odd = [1,3,5,7])
⑥ test(a = (1,), b = {4,5}, c={10, 20})
⑦ test(x = {1:'one', 2:'two'}, y = 'hello')
```

[결과] 사전으로 처리됨.

```
{'b': 20, 'a': 10}
{'age': 10, 'name': 'Alice'}
{'even': [2, 4, 6], 'odd': [1, 3, 5, 7]}
{'b': {4, 5}, 'a': (1,), 'c': {10, 20}}
{'y': 'hello', 'x': {1: 'one', 2: 'two'}} 48
```



## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 가변 개수의 키워드 인수

- `**kwargs`에 넘어가는 인수들이 사전 자료형으로 처리되면, `kwargs`에 사전 메소드를 적용할 수 있음.

```
def test_kwargs(**kwargs):  
    for key, value in kwargs.items():  
        print(key, ': ', value)  
  
test_kwargs(name='Alice', age=10, height=135)
```

[결과]

```
name : Alice  
age : 10  
height : 135
```

## 6. 매개변수와 입력 인수의 다양한 형태

### ◆ 가변 개수의 키워드 인수

- 사전에 키워드 인수를 미리 만들어 놓고 함수 호출에 사용할 수 있음.

<pre>def test( **kwargs ):     print(kwargs)</pre>	
<pre>D1 = {'a':10, 'b':20, 'c':30} test(**D1)</pre>	<pre>{'a': 10, 'b': 20, 'c': 30}</pre>
<pre>D2 = {'x':[5,7], 'y':{2,9}} test(**D2)</pre>	<pre>{'x': [5, 7], 'y': {9, 2}}</pre>
<pre>D3 = {'name':'Alice', 'age':10} test(**D3)</pre>	<pre>{'name': 'Alice', 'age': 10}</pre>

## 7. 람다 함수

### ◆ 람다 함수 (lambda)

- 이름 없는 함수
- 함수의 인수로 함수를 넣어야 하는 경우에 유용함.

```
lambda <인수들> : <반환할 식>
```

인수는 콜론 앞으로, 반환식은 콜론 뒤로 보냅니다.



```
def add( x, y ) :
```

```
    return x + y
```



```
lambda x, y : x + y
```

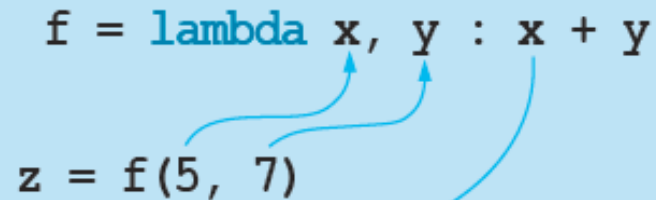
## 7. 람다 함수

### ◆ 람다 함수 (lambda)

람다 함수에 이름을 붙일 수도 있음

```
>>> f = lambda x,y: x+y  
>>> z = f(5,7)  
>>> print(z)  
12
```

```
f = lambda x, y : x + y  
z = f(5, 7)
```



```
def square(x):  
    return x ** 2
```

→

```
lambda x: x ** 2
```



```
>>> f = lambda x: x ** 2  
>>> f(10)  
100
```

# 7. 람다 함수

## ◆ 람다 함수 (lambda)

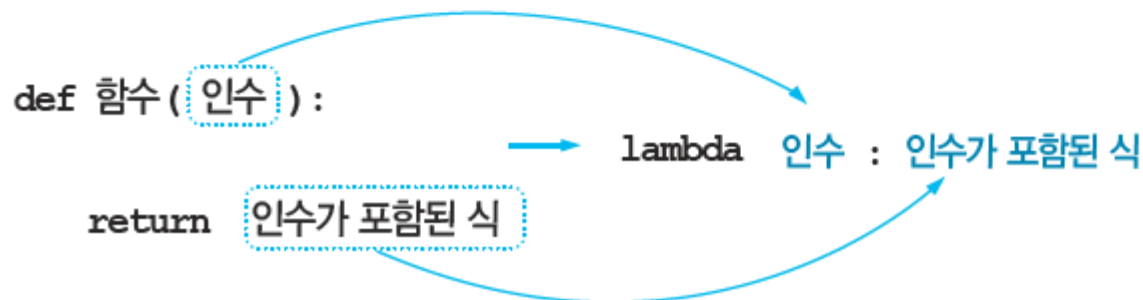
```
def check_even(x):
    return x % 2 == 0
```

→

```
lambda x: x % 2 == 0
```

```
>>> f = lambda x: x % 2 == 0
>>> f(10)
True
>>> f(5)
False
```

람다 함수는 간단한 함수를 좀 더 간단히 표현할 수 있도록 합니다.



## 8. 파이썬 내장 함수

### ◆ 파이썬 내장 함수 목록 보기

```
>>> dir(__builtins__)
```

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

## 8. 파이썬 내장 함수

### ◆ 자료형 변환 함수

- 각 자료형마다 같은 이름으로 함수가 존재함.
- `int()`, `float()`, `complex()`, `bool()`, `str()`, `list()`, `tuple()`, `set()`, `dict()`

## 8. 파이썬 내장 함수

### ◆ 수학 관련 함수

함수	설명	예
<code>abs(x)</code>	x에는 int 또는 float 자료형을 넣어야 하고 절대 값을 반환합니다.	<pre>&gt;&gt;&gt; abs(-4), abs(-5.5) (4, 5.5)</pre>
<code>divmod(x,y)</code>	<code>divmod()</code> 함수는 x와 y를 입력받아서 $(x // y, x \% y)$ 튜플을 반환합니다.	<pre>&gt;&gt;&gt; divmod(20, 7) (2, 6)</pre>
<code>pow(x,y)</code>	<code>pow()</code> 함수는 $x^y$ 를 계산하여 반환합니다.	<pre>&gt;&gt;&gt; pow(10,4) 10000</pre>
<code>round(x)</code>	<code>round()</code> 함수는 x를 반올림한 값을 반환합니다.	<pre>&gt;&gt;&gt; round(4.4), round(5.9) (4, 6) &gt;&gt;&gt; round(-1.5), round(-5.1) (-2, -5)</pre>
<code>bin(x)</code>	정수 x를 2진수로 변환하여 반환합니다. 결과는 0b로 시작하는 문자열입니다.	<pre>&gt;&gt;&gt; bin(20) '0b10100'</pre>
<code>oct(x)</code>	정수 x를 8진수로 변환하여 반환합니다. 결과는 0o로 시작하는 문자열입니다.	<pre>&gt;&gt;&gt; oct(20) '0o24'</pre>
<code>hex(x)</code>	정수 x를 16진수로 변환하여 반환합니다. 결과는 0x로 시작하는 문자열입니다.	<pre>&gt;&gt;&gt; hex(20) '0x14'</pre>



## 8. 파이썬 내장 함수

### ◆ iter(x) 함수

- x에 넣는 객체가 iterable 객체인지 판단하는 함수.
- 수치 자료형은 iterable 객체가 아님.

```
>>> a = 5; b = 3.5; c = True; d = 2 + 5j
>>> iter(a)      # 정수 객체는 iterable 객체가 아닙니다.
.....
TypeError: 'int' object is not iterable
>>> iter(b)      # 실수 객체는 iterable 객체가 아닙니다.
.....
TypeError: 'float' object is not iterable
>>> iter(c)      # 부울 객체는 iterable 객체가 아닙니다.
.....
TypeError: 'bool' object is not iterable
>>> iter(d)      # 복소수 객체는 iterable 객체가 아닙니다.
.....
TypeError: 'complex' object is not iterable
```

## 8. 파이썬 내장 함수

### ◆ iter(x) 함수

- 컨테이너 자료형은 iterable 객체임.

자료형	예제 코드
시퀀스 자료형	<pre> &gt;&gt;&gt; L = [1,2,3]; T = (5, 6); name = 'Alice' &gt;&gt;&gt; iter(L)           # 리스트는 iterable 객체입니다. &lt;list_iterator object at 0x023E2D50&gt; &gt;&gt;&gt; iter(T)           # 튜플은 iterable 객체입니다. &lt;tuple_iterator object at 0x023E2BF0&gt; &gt;&gt;&gt; iter(name)        # 문자열은 iterable 객체입니다. &lt;str_iterator object at 0x023E2DD0&gt; </pre>
set과 dict 자료형	<pre> &gt;&gt;&gt; S = {1,3,5}; D = {1:'one', 2:'two', 3:'three'} &gt;&gt;&gt; iter(S)           # 집합은 iterable 객체입니다. &lt;set_iterator object at 0x023DBDA0&gt; &gt;&gt;&gt; iter(D)           # 사전은 iterable 객체입니다. &lt;dict_keyiterator object at 0x023ED2A0&gt; </pre>

## 8. 파이썬 내장 함수

### ◆ iterable 객체를 반환하는 함수들

- range(), sorted, reversed(), enumerate(), map(), filter(), zip()
- 각 함수의 반환값에 iter() 함수를 적용하면 반환값이 iterable 객체인지 알 수 있음.

```
>>> a = range(5)
>>> iter(a)
<range_iterator object at 0x04137B60>
>>> b = sorted([2,4,3,1])
>>> iter(b)
<list_iterator object at 0x041567F0>
>>> c = reversed([2,4,1,5])
>>> iter(c)
<list_reverseiterator object at 0x04156E50>
```

## 8. 파이썬 내장 함수

### ◆ enumerate() 함수

- 인수가 1개 또는 2개임.
- 첫 번째 인수는 반드시 iterable 객체여야 함.
- 인수가 1개인 경우에는 iterable 객체에 0부터 번호를 붙여서 반환함.
- 인수가 2개인 경우에는 두 번째 인수로 숫자를 넣어야 함. Iterable 객체에 두 번째 인수부터 번호를 붙여서 반환함.

객체에 0부터 번호를 붙여서 반환함 ← `enumerate(iterable 객체)`  
객체에 정수부터 번호를 붙여서 반환함 ← `enumerate(iterable 객체, 정수)`

## 8. 파이썬 내장 함수

### ◆ enumerate() 함수

```
>>> fruits = ['apple', 'melon', 'kiwi', 'orange']
>>> enumerate(fruits)          # enumerate() 함수의 반환값은 enumerate 객체입니다.
<enumerate object at 0x03307C38>
>>> iter(enumerate(fruits))    # enumerate 객체는 iterable 자료형입니다.
<enumerate object at 0x023DBF08>
>>> list(enumerate(fruits))     # enumerate 객체를 리스트로 변환합니다.
[(0, 'apple'), (1, 'melon'), (2, 'kiwi'), (3, 'orange')]
>>> tuple(enumerate(fruits))    # enumerate 객체를 튜플로 변환합니다.
((0, 'apple'), (1, 'melon'), (2, 'kiwi'), (3, 'orange'))
>>> list(enumerate(fruits, 10))
[(10, 'apple'), (11, 'melon'), (12, 'kiwi'), (13, 'orange')]
>>> list(enumerate(fruits, 15))
[(15, 'apple'), (16, 'melon'), (17, 'kiwi'), (18, 'orange')]
```

## 8. 파이썬 내장 함수

### ◆ enumerate() 함수

- enumerate() 함수는 사전을 만드는 데 유용함. (사전의 '키'가 됨)

```
>>> dict(enumerate(['python', 'java', 'c', 'c++']))
{0: 'python', 1: 'java', 2: 'c', 3: 'c++'}
>>> dict(enumerate([100, 200, 300]))
{0: 100, 1: 200, 2: 300}
>>> dict(enumerate([x ** 2 for x in range(11)]))
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81, 10: 100}
```

```
>>> color = ['red', 'blue', 'white', 'black']
>>> for n, c in enumerate(color):          # for ... in iterable 객체
    print(n, c)
```

```
0 red
1 blue
2 white
3 black
```

enumerate(color) 반환값은 iterable 객체임

## 8. 파이썬 내장 함수

### ◆ enumerate() 함수

**CODE 75** 리스트 score에는 학생들의 성적이 저장되어 있습니다. 성적을 내림차순으로 정렬하여 등수를 매겨서 출력하는 코드를 작성해 볼게요(오른쪽 출력 결과). 이때, 각 사전으로 '키'는 등수이고 '값'에는 성적이 저장되도록 합니다(동점이 없다고 가정할게요).

```
score = [90, 88, 84, 77, 85, 97, 60, 66, 79, 93]
D = dict(enumerate(sorted(score, reverse=True), 1))
for rank, scr in D.items():
    print('{}등 - {}점'.format(rank, scr))
```

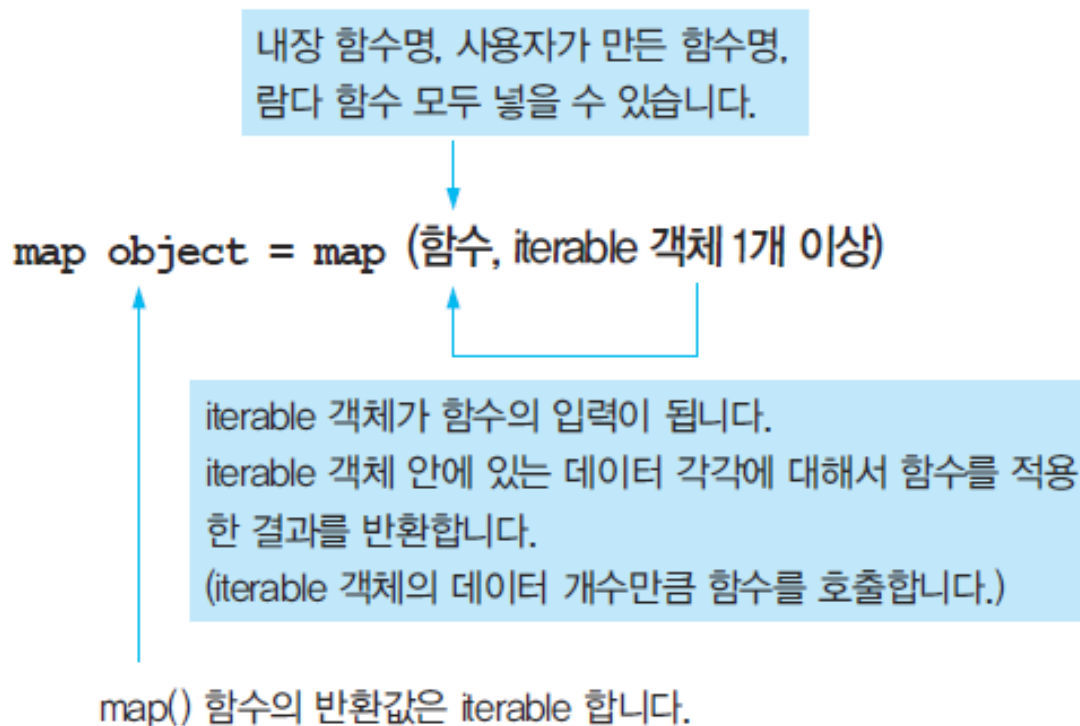
[결과]

1등 - 97점  
2등 - 93점  
3등 - 90점  
4등 - 88점  
5등 - 85점  
6등 - 84점  
7등 - 79점  
8등 - 77점  
9등 - 66점  
10등 - 60점

## 8. 파이썬 내장 함수

### ◆ map() 함수

- 첫 번째 인수 - 함수명
- 두 번째 인수 - 첫 번째 인수로 들어가는 함수의 입력으로 iterable 객체여야 함.





## 8. 파이썬 내장 함수

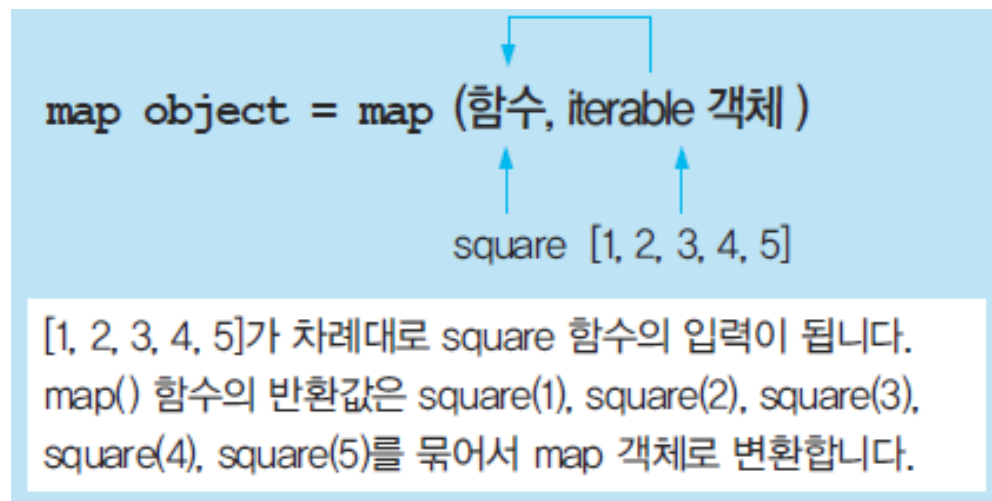
### ◆ map() 함수

```
>>> result = map(abs, [-3, -5, 0, 2, 7, -9])
>>> print(result)           # map() 함수의 결과는 map 객체입니다.
<map object at 0x023D3B70>
>>> list(result)           # map 객체에 list() 함수를 적용해서 리스트 결과를 봅니다.
[3, 5, 0, 2, 7, 9]
>>> iter(result)           # map() 함수의 결과는 iterable 객체입니다.
<map object at 0x023D3B70>
```

## 8. 파이썬 내장 함수

### ◆ map() 함수

```
>>> def square(x):
        return x ** 2
>>> X = [1, 2, 3, 4, 5]
>>> Y = map(square, X)
>>> print(Y)
<map object at 0x033B8330>
>>> type(Y)
<class 'map'>
>>> list(Y)
[1, 4, 9, 16, 25]
```



첫 번째 인수로 람다 함수를 넣어도 됨

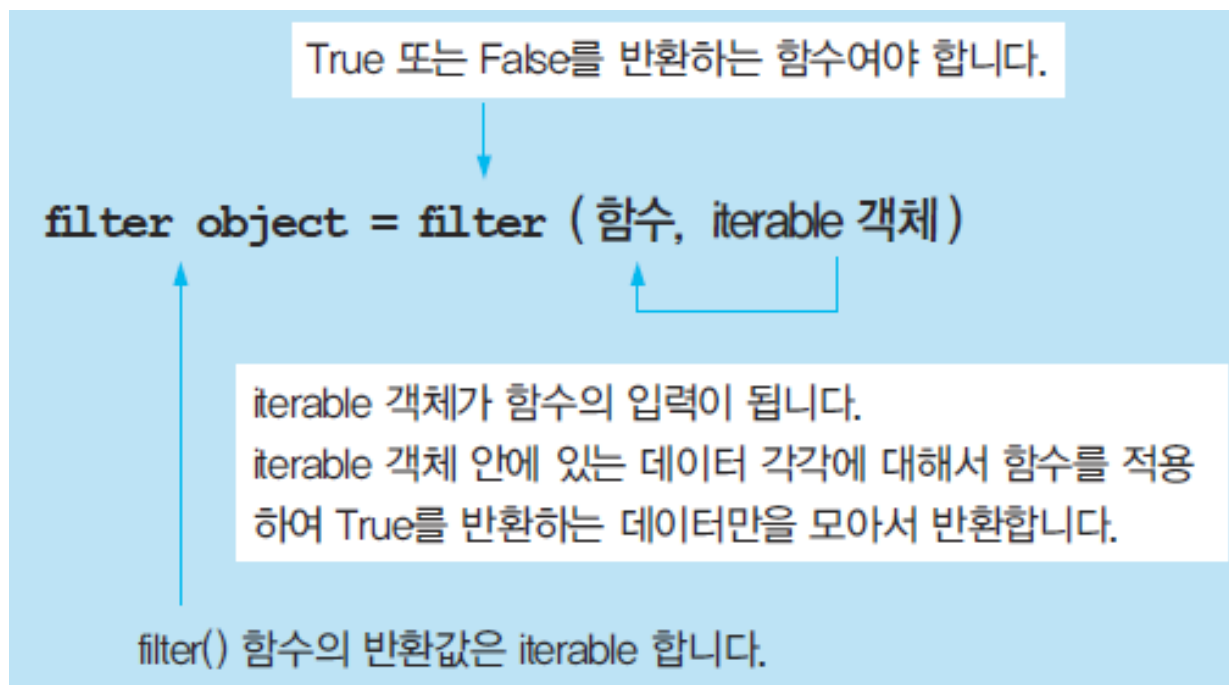
```
>>> L = [1, 2, 3, 4, 5]
>>> list(map(lambda x: x * x, L))
[1, 4, 9, 16, 25]
```

# 람다 함수는 이름없는 함수예요.

## 8. 파이썬 내장 함수

### ◆ filter() 함수

- map() 함수와 기본적인 함수 사용법이 같은데, 첫 번째 인수로 True 또는 False를 결과로 내는 함수를 넣어야 함.



## 8. 파이썬 내장 함수

### ◆ filter() 함수

```
>>> def even(x):                # x가 짝수이면 True, 홀수이면 False를 반환합니다.
    if x%2 == 0: return True
    else: return False
>>> L = [4, 9, 10, 11, 14, 2, 13, 15, 20, 9]
>>> result = filter(even,L)      # L에서 even( ) 함수를 True로 만드는 원소만 필터링합니다.
>>> print(result)               # filter 객체를 반환합니다.
<filter object at 0x023E6210>
>>> iter(result)                # filter 객체는 iterable 객체입니다.
<filter object at 0x023E6210>
>>> list(result)                # 리스트로 변환하여 결과를 확인합니다.
[4, 10, 14, 2, 20]
```

```
>>> L = [5, -1, -3, 9, 2, -7, 3]
>>> result = filter(lambda x : x > 0, L)    # L에서 양수만 필터링합니다.
>>> list(result)
[5, 9, 2, 3]
```

## 8. 파이썬 내장 함수

### ◆ filter() 함수

**CODE 76** 현재 리스트 data에는 많은 정수들이 저장되어 있어요. data에서 양의 짝수의 개수가 몇 개인지 알고 싶어요. 한 줄로 작성해 보세요.

```
>>> data = [4, 10, -22, 17, 23, -9, 3, -5, -6, -1, 2, 20, 33, -6]
>>> len(list(filter(lambda x : x > 0 and x % 2 == 0, data)))
4
```

## 8. 파이썬 내장 함수

**CODE 77** 현재 사전 temperature에는 12월 한 달 동안의 기온이 저장되어 있습니다. 사전의 ‘키’는 날짜이고 ‘값’은 해당 날짜의 기온입니다. 영하의 기온 중에서 가장 따뜻했던 날의 기온과 그 기온이었던 날짜를 출력하는 코드입니다(filter() 함수 이용 문제입니다).

```
temperature = {1:-5, 2:3, 3:2, 4:-2, 5:-8, 6:-6, 7:3, 8:4, 9:1, 10:-1,
               11:-2, 12:-3, 13:3, 14:10, 15:8, 16:7, 17:-3, 18:1, 19:-8, 20:-3,
               21:-1, 22:-10, 23:-9, 24:8, 25:-2, 26:-7, 27:4, 28:5, 29:-1,
               30:-4, 31:5}

# max_temp_under_zero 변수에는 영하의 온도 중에서 가장 높은 온도를 저장합니다.
max_temp_under_zero = max(filter(lambda x : x < 0, temperature.values()))
max_temp_days = []    # 영하의 온도 중에서 가장 높은 온도에 해당하는 날짜 저장 리스트
for day, temp in temperature.items():
    if temp == max_temp_under_zero:
        max_temp_days.append(day)
print('영하의 온도 중에서 가장 높은 온도 :', max_temp_under_zero)
print('영하의 온도 중에서 가장 높은 온도였던 날들 :', max_temp_days)
```

영하의 온도 중에서 가장 높은 온도 : -1

영하의 온도 중에서 가장 높은 온도였던 날들 : [10, 21, 29]

## 8. 파이썬 내장 함수

### ◆ zip() 함수

- 1개 이상의 iterable 객체를 인수로 받아서 사전을 만들어 반환함.

```
>>> A = [1, 2, 3]
>>> B = ['one', 'two', 'three']
>>> result = zip(A, B)           # zip() 함수의 인수는 모두 iterable 객체여야 합니다.
>>> print(result)                # zip() 함수의 반환값은 zip 객체입니다.
<zip object at 0x02173B70>
>>> iter(result)                 # zip 객체는 iterable 합니다.
<zip object at 0x02173B70>
>>> list(result)                 # zip 객체를 리스트로 변환하여 결과를 확인합니다.
[(1, 'one'), (2, 'two'), (3, 'three')]
>>> A = [1, 2, 3, 4, 5]; B = ['one', 'two', 'three']
>>> list(zip(A, B))              # 두 인수의 개수가 다르면 짧은 쪽에 맞춥니다.
[(1, 'one'), (2, 'two'), (3, 'three')]
```

## 8. 파이썬 내장 함수

### ◆ zip() 함수

```
>>> A = [1,2,3]; B = ('one', 'two', 'three'); C = [10, 20, 30]      # 인수 3개
>>> zip(A, B, C)
<zip object at 0x03D97F58>
>>> list(zip(A,B,C))
[(1, 'one', 10), (2, 'two', 20), (3, 'three', 30)]
```

```
>>> subject = ['math', 'english', 'science']
>>> score = [90, 95, 87]
>>> for sub, s in zip(subject, score):
        print("{}' score - {}".format(sub,s))
```

```
math' score - 90
english' score - 95
science' score - 87
```



## 9. 정리

- ◆ 파이썬 함수는 내장 함수와 사용자 정의 함수가 있음.
- ◆ 내장 함수는 파이썬이 만들어서 제공하는 함수임.
- ◆ 사용자 정의 함수는 사용자가 직접 만드는 함수임.
- ◆ 함수의 매개변수, 인수, 반환값을 문법에 맞게 사용해야 함.
- ◆ lambda 함수는 이름없는 함수임.