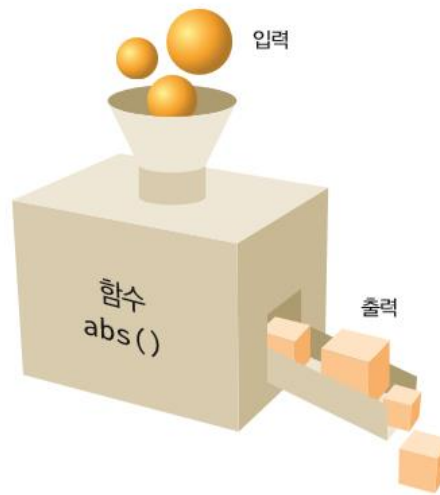


# 5장 함수(FUNCTION)

# 함수란?

- 함수(function)는 특정 작업을 수행하는 명령어들의 모음에 이름을 붙인 것
- 함수는 작업에 필요한 데이터를 전달받을 수 있으며, 작업이 완료된 후에는 작업의 결과를 호출자에게 반환할 수 있다.



# 함수의 예

`print()`

`input()`

`abs(), ...`

함수 안의 명령어들을 실행하려면 함수를 호출(call)하면 된다.

```
>>> value = abs(-100)
```

```
>>> value
```

```
100
```

# 함수는 왜 필요한가?

비슷한 코드인데 하나로 합칠 수 있을까?



```
sum = 0;
for i in range(1, 11)
    sum += i;
```

```
sum = 0;
for i in range(1, 21)
    sum += i;
```

```
get_sum(1, 10)
```

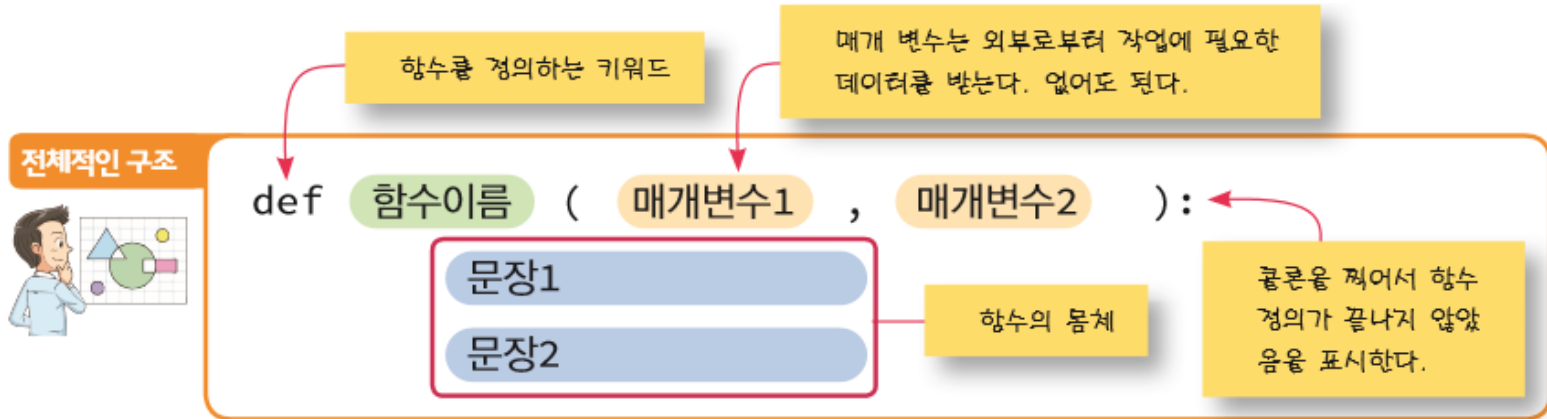
```
get_sum(1, 20)
```

```
def get_sum(start, end)
    sum = 0;
    for i in range(start, end+1)
        sum += i;
    return sum
```

함수를 사용하면 됩니다.

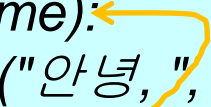


# 함수를 작성해보자.



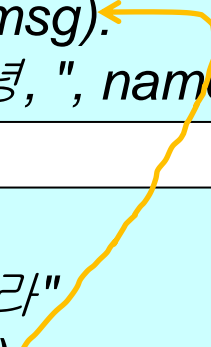
# 함수 작성의 예

```
>>>def say_hello(name):  
    print("안녕, ", name);
```



```
>>>say_hello("철수")  
안녕, 철수
```

```
>>>def say_hello(name, msg):  
    print("안녕, ", name, "야, ", msg);
```



```
>>>name="철수"  
>>>msg="어서 집에 오너라"  
>>>say_hello(name, msg)  
안녕, 철수야, 어서 집에 오너라
```

# 값을 반환하는 함수

```
>>>def get_sum(start, end) :  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

```
>>>value = get_sum(1, 10)  
>>>print(value)  
55
```

# 함수 사용의 장점

- 프로그램 안에서 중복된 코드를 제거한다.
- 복잡한 프로그래밍 작업을 더 간단한 작업들로 분해할 수 있다
- 함수는 한번 만들어지면 다른 프로그램에서도 재사용될 수 있다.
- 함수를 사용하면 가독성이 증대되고, 유지 관리도 쉬워진다.



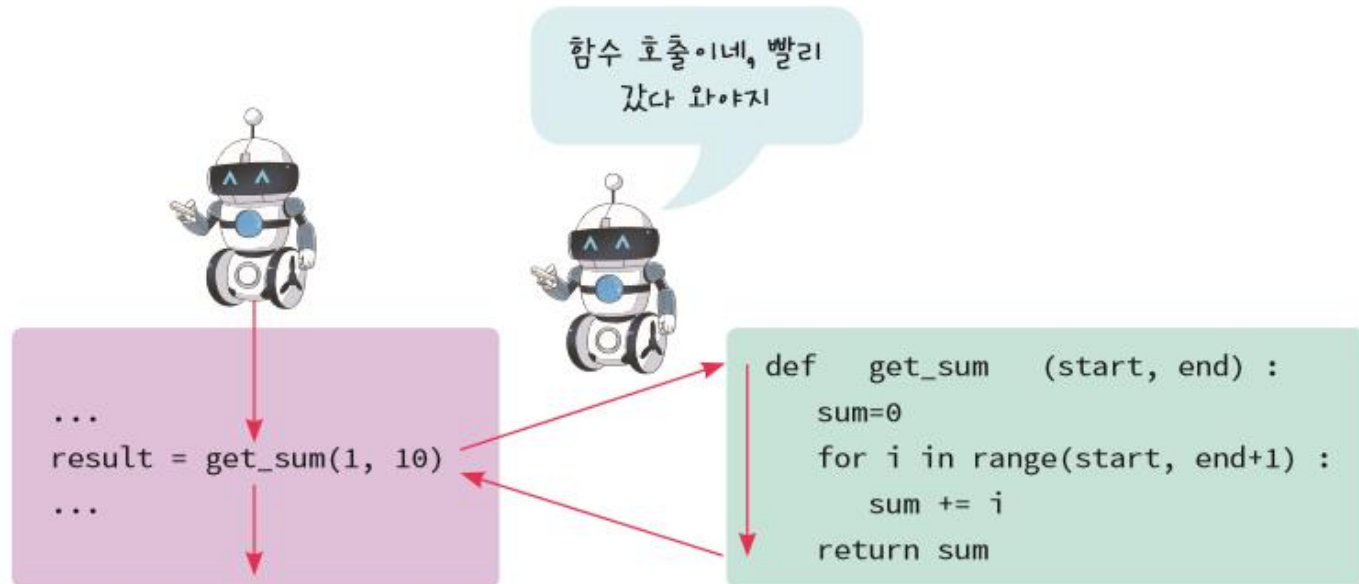
# 함수의 이름

함수의 목적을 설명하는 동사 또는 동사+명사를 사용

<code>square(side)</code>	// 정수를 제공하는 함수
<code>compute_average(list)</code>	// 평균을 구하는 함수
<code>set_cursor_type(c)</code>	// 커서의 타입을 설정하는 함수

# 함수 호출

함수를 사용하려면 함수를 호출(call)하여야 한다.



함수는 여러 번 호출할 수 있다.



```
...  
x = get_sum(1, 10)  
...  
y = get_sum(1, 20)
```

```
def get_sum (start, end) :  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

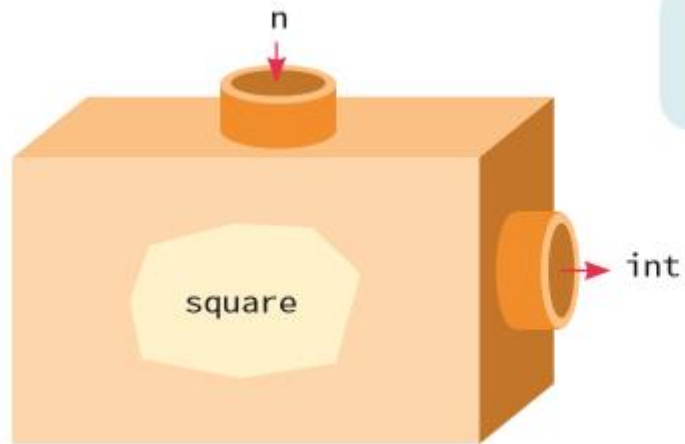
## 예제

```
def get_sum(start, end) :  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum  
  
print( get_sum(1, 10))  
print( get_sum(1, 20))
```

```
55  
210
```

# 함수 작성의 예 #1

정수를 입력받아서 제공한 값을 반환하는 함수를 만들어보자.



일단 함수 이름과 입력, 출력  
만 결정하자!



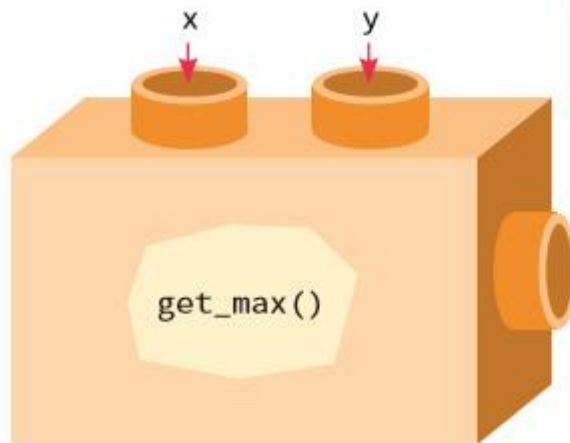
## 예제

```
def square(n):  
    return(n*n)  
print(square(10))
```

100

## 함수 작성의 예 #2

- 두개의 정수가 주어지면 두수 중에서 더 큰 수를 찾아서 이것을 반환하는 함수를 만들어보자.



일단 함수 이름과 매개변수  
만 결정하자!



## 예제

```
def get_max(x, y):  
    if( x > y ):  
        return x  
    else:  
        return y  
  
print(get_max(10, 20))
```

20



## 함수 작성의 예 #2

- 정수의 거듭제곱값을 계산하여 반환하는 함수를 작성하여 보자. (파이썬에는 \*\* 연산자가 있지만)



## 예제

```
def power(x, y):  
    result = 1  
    for i in range(y):  
        result = result * x  
    return result  
  
print(power(10, 2))
```

100

# 함수를 이용할 때 주의할 점

파이썬 인터프리터는 함수가 정의되면 함수 안의 문장들은 즉시 실행하지 않는다.  
함수 정의가 아닌 문장들은 즉시 실행하게 된다.

```
print(power(10, 2))
```

```
def power(x, y):  
    result = 1  
    for i in range(y):  
        result = result * x  
    return result
```

무엇이 문제인가?

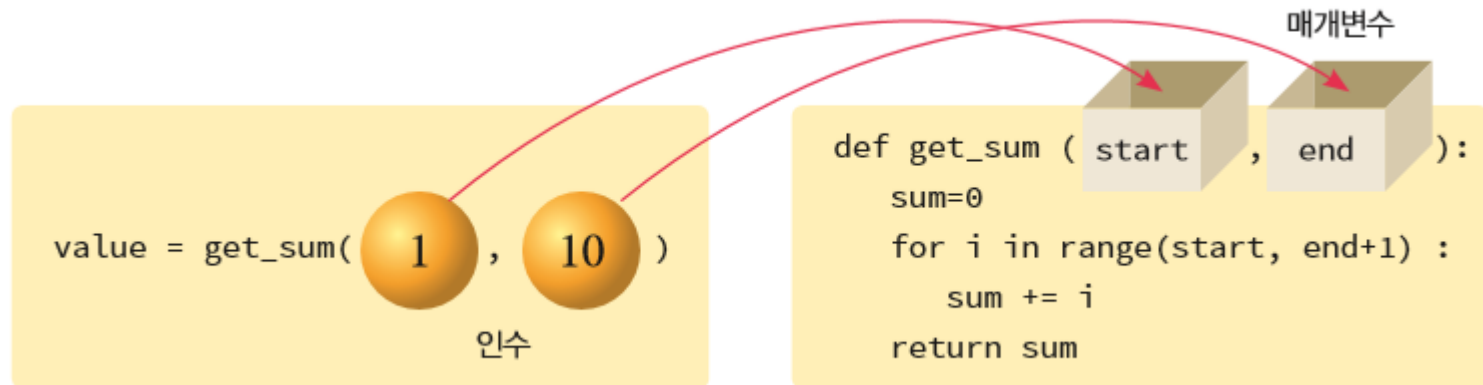
# 예제

```
def main():  
    print(power(10, 2))  
  
def power(x, y):  
    result = 1  
    for i in range(y):  
        result = result * x  
    return result  
  
main()
```

이런 형태는 가능하다!

# 인수와 매개 변수

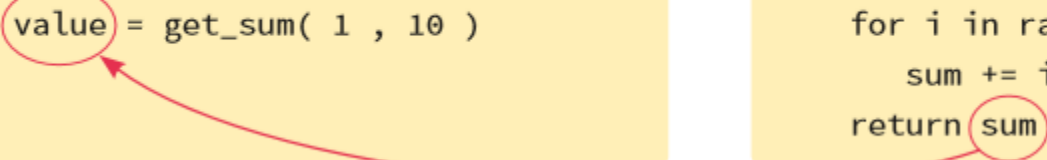
인수(argument)는 호출 프로그램에 의하여 함수에 실제로 전달되는 값이다.  
매개 변수(parameter)는 이 값을 전달받는 변수이다.



# 함수가 값을 반환

반환값(return value)은 함수가 호출한 곳으로 반환하는 작업의 결과값이다.

```
value = get_sum( 1 , 10 )
```



A red curved arrow originates from the word 'sum' in the 'return sum' line of the function definition on the right and points to the variable 'value' in the function call on the left, illustrating the return value being passed back to the caller.

```
def get_sum (    start , end    ):  
    sum=0  
    for i in range(start, end+1) :  
        sum += i  
    return sum
```

# 디폴트 인수

파이썬에서는 함수의 매개변수가 기본값을 가질 수 있다. 이것을 디폴트 인수 (default argument)라고 한다.

```
def greet(name, msg="별일없죠?"):
    print("안녕 ", name + ', ' + msg)

greet("영희")
```

안녕 영희, 별일없죠?

# 키워드 인수

인수들이 위치가 아니고 키워드에 의하여 함수로 전달되는 방식

```
>>> def calc(x, y, z):  
    return x+y+z  
  
>>> calc(10, 20, 30)  
60  
  
>>> calc(x=10, y=20, z=30)  
60  
  
>>> calc(y=20, x=10, z=30)  
60
```



# 참조값에 의한 인수 전달

함수를 호출할 때, 변수를 전달하는 2가지 방법

- 값에 의한 호출(call-by-value)
- 참조에 의한 호출

# 값에 의한 호출

```
def modify1(s):  
    s += "To You"
```

```
msg = "Happy Birthday"  
print("msg=", msg)  
modify1(msg)  
print("msg=", msg)
```

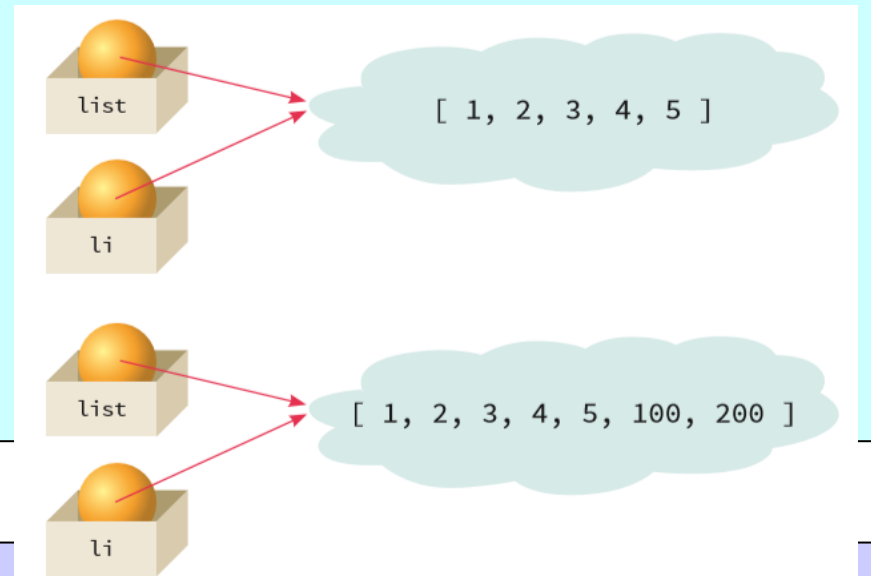


```
msg= Happy Birthday  
msg= Happy Birthday
```

# 참조에 의한 호출

```
def modify2(li):  
    li += [100, 200]
```

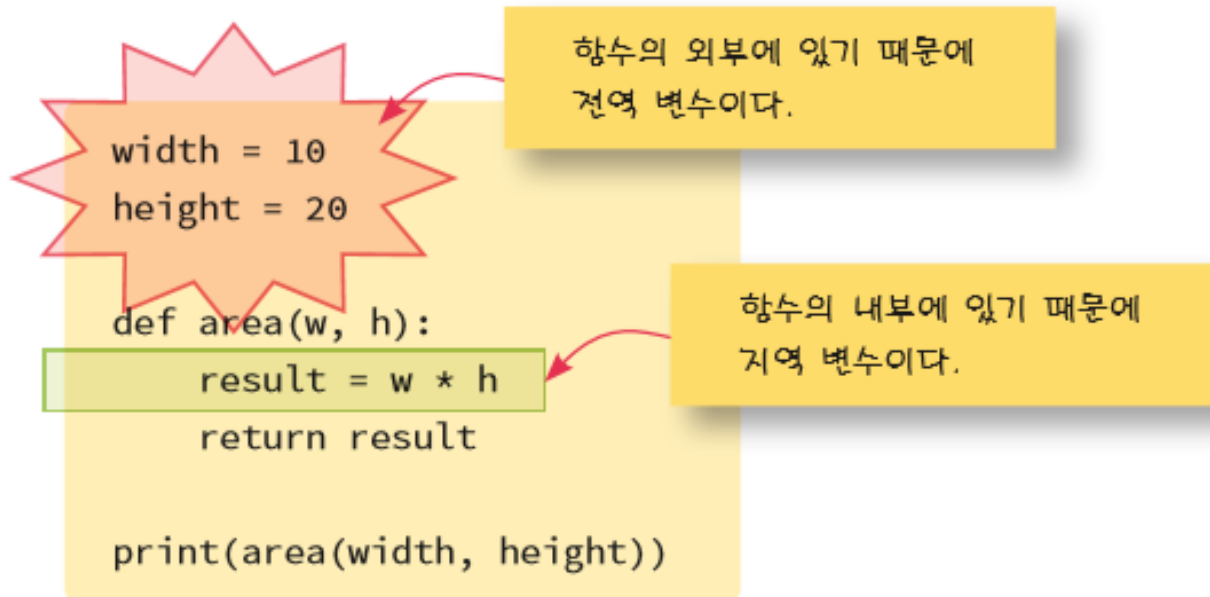
```
list = [1, 2, 3, 4, 5]  
print(list)  
modify2(list)  
print(list)
```



```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5, 100, 200]
```

# 지역 변수와 전역 변수



# 지역 변수

```
def sub():  
    s = "바나나가 좋음!"  
    print(s)
```

```
sub()
```

바나나가 좋음!

# 전역 변수

```
def sub():  
    print(s)  
  
s = "사과가 좋음!"  
sub()
```

사과가 좋음!

# 지역 변수와 전역 변수

```
def sub():
```

```
    s = "바나나가 좋음!"
```

```
    print(s)
```

```
s = "사과가 좋음!"
```

```
sub()
```

```
print(s)
```

전역 변수를 사용하는 것이 아님!



바나나가 좋음!

사과가 좋음!

# 전역 변수를 함수 안에서 사용하려면

```
def sub():  
    global s  
  
    print(s)  
    s = "바나나가 좋음!"  
    print(s)  
  
s = "사과가 좋음!"  
sub()  
print(s)
```

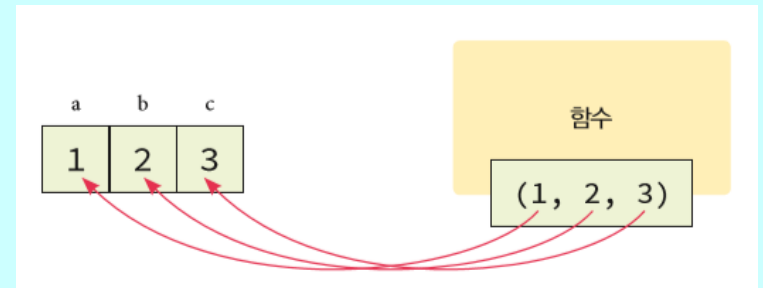
사과가 좋음!  
바나나가 좋음!  
바나나가 좋음!



# 여러 개의 값 반환하기

```
def sub():  
    return 1, 2, 3
```

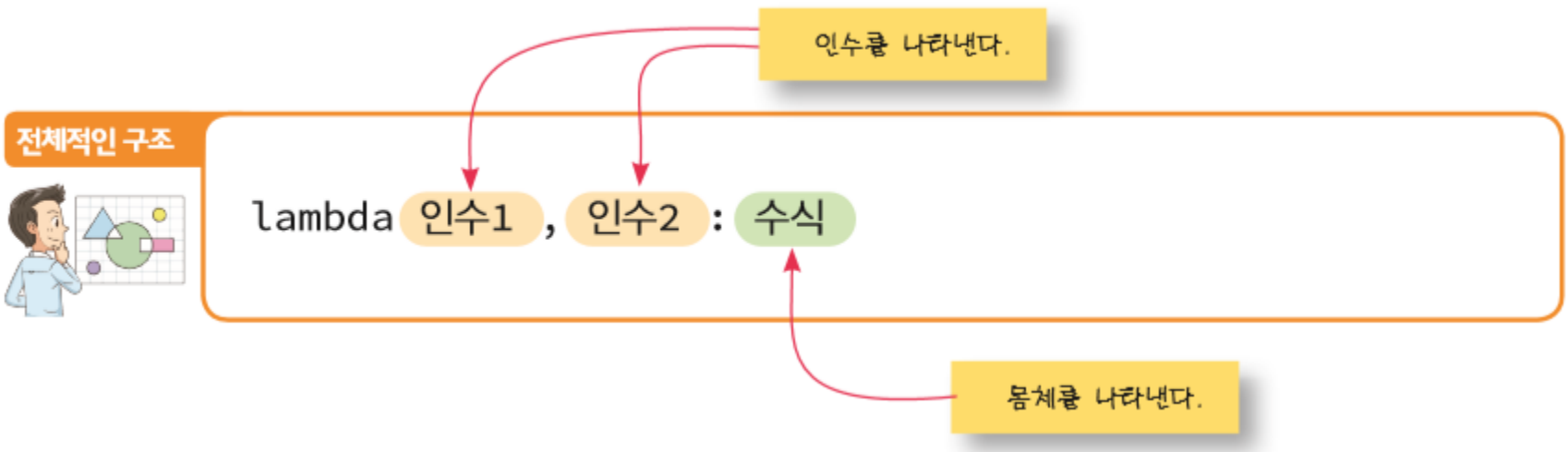
```
a, b, c = sub()  
print(a, b, c)
```



1 2 3

# 무명 함수(람다 함수)

- 무명 함수는 이름은 없고 몸체만 있는 함수이다. 파이썬에서 무명 함수는 lambda 키워드로 만들어진다.



# 무명 함수의 예

```
sum = lambda x, y: x+y;
```

```
print( "정수의 합 :", sum( 10, 20 ))
```

```
print( "정수의 합 :", sum( 20, 20 ))
```

```
정수의 합 : 30
```

```
정수의 합 : 40
```

## ● 재귀호출 알고리즘

```
def TOT(n):
```

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

```
        return 1
```

```
    else :
```

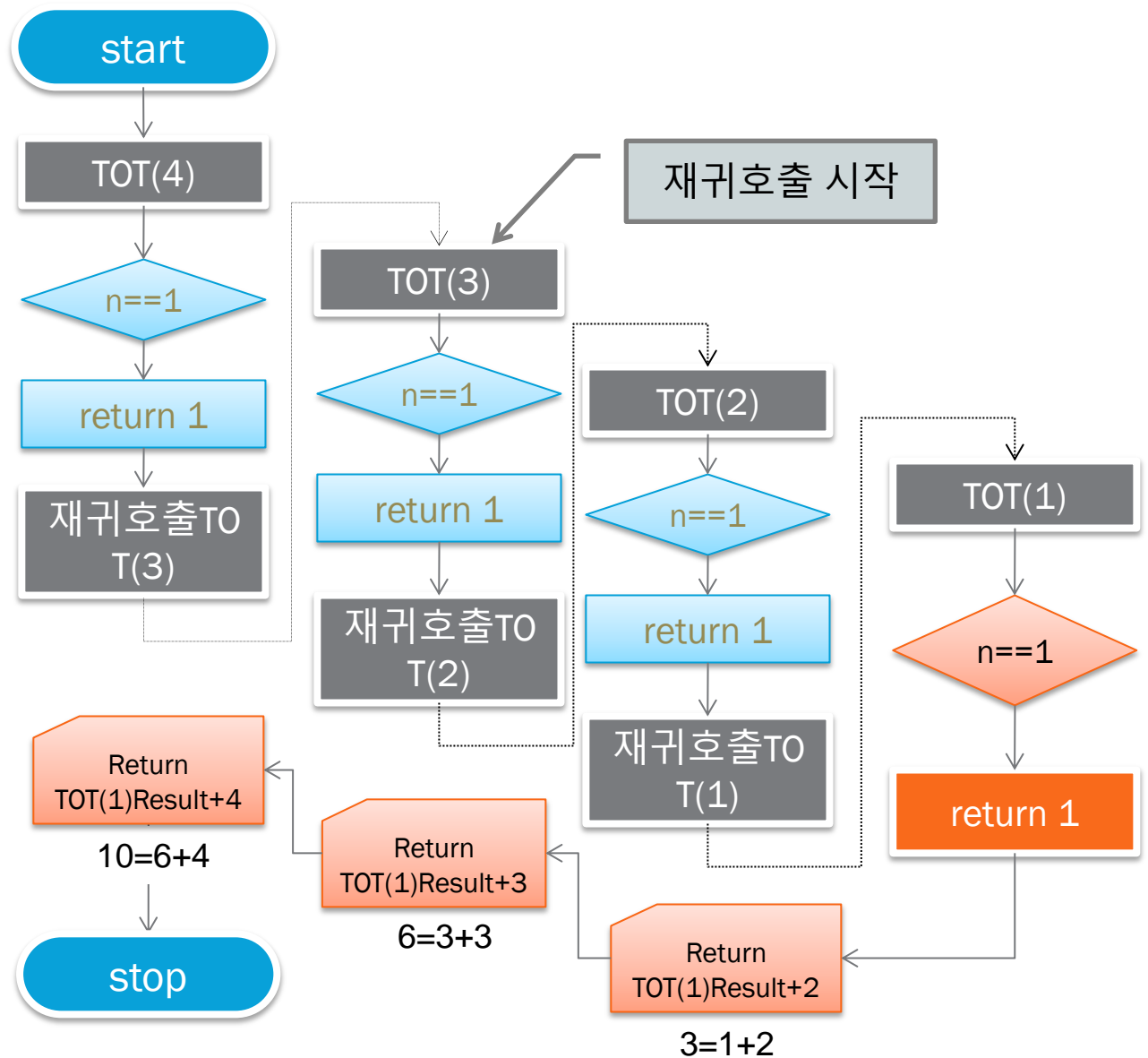
```
        tot = TOT(n-1) + n
```

```
    return tot
```

```
tot = TOT(4) # start
```

```
print('덧셈 결과 =', tot)
```

```
# 덧셈 결과 = 10
```



내장 함수

# 내장 함수

내장 함수는 대부분의 객체에 대해서 사용이 가능하다.

<b>abs()</b>	<b>dict()</b>	<b>help()</b>	<b>min()</b>	<b>setattr()</b>
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()	

# ABS(X) 함수

```
>>> abs(-3)
3
>>> abs(3+4j)
5.0
```

# CHR(I) 함수

```
>>> chr(65)
'A'
>>> ord('A')
65
```



# COMPILE(SOURCE, FILENAME, MODE) 함수

```
>>> with open('mymodule.py') as f:  
    lines = f.read()  
  
>>> code_obj = compile(lines, 'mymodule.py', 'exec')  
>>> exec(code_obj)  
...
```

# COMPLEX-REAL, IMAG) 함수

```
>>> x = complex(4, 2)
>>> x
(4+2j)
```

# DIR 함수

dir은 객체가 가지고 있는 변수나 함수를 보여 준다.(list 객체 변수, 함수 제공)

```
>>> dir([1, 2, 3])  
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',  
 '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',  
 '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__',  
 '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',  
 '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',  
 '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',  
 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

# EVAL() 함수

eval()은 파이썬의 수식을 문자열로 받아서 실행하고 그 결과를 반환한다.

```
>>> eval("1+2");  
3
```

```
>>> x = 1  
>>> y = 1  
>>> eval('x+y')  
2
```

```
>>> eval("print('Hi!')")  
Hi!
```

# EXEC() 함수

exec() 함수는 수식뿐만 아니라 모든 파이썬 문장을 받아서 실행한다.

```
>>> exec("y=2+3")
>>> y
5

>>> statements = """
import math
def area_of_circle(radius):
    return math.pi * radius * radius

"""

>>> exec(statements)
>>> area_of_circle(5)
78.53981633974483
```

# Float() 함수

float() 함수는 문자열을 부동소수점으로 변환하는 기능을 한다.

```
>>> str = input("실수를 입력하시오: ")
실수를 입력하시오: 12.345
>>> str
'12.345'
>>> value=float(str)
>>> value
12.345
```

# MAX() 함수

max() 함수는 리스트나 튜플, 문자열에서 가장 큰 항목을 반환한다.

```
>>> values = [ 1, 2, 3, 4, 5]
>>> max(values)
5
>>> min(values)
1
```

# MAX() 함수

max() 함수는 리스트나 튜플, 문자열에서 가장 큰 항목을 반환한다.

```
>>> values = [ 1, 2, 3, 4, 5]
>>> max(values)
5
>>> min(values)
1
```



# 파이썬에서 정렬하기

```
>>> sorted([4, 2, 3, 5, 1])  
[1, 2, 3, 4, 5]
```

```
>>> myList = [4, 2, 3, 5, 1]  
>>> myList.sort()  
>>> myList  
[1, 2, 3, 4, 5]
```

# KEY 매개변수

- 정렬을 하다보면 정렬에 사용되는 키를 개발자가 변경해주어야 하는 경우가 종종 있다.

```
>>> sorted("The health know not of their health, but only the sick".split(),  
key=str.lower)  
['but', 'health', 'health,', 'know', 'not', 'of', 'only', 'sick', 'The', 'the', 'their']
```

```
>>> students = [  
    ('홍길동', 3.9, 20160303),  
    ('김철수', 3.0, 20160302),  
    ('최자영', 4.3, 20160301),  
]  
>>> sorted(students, key=lambda student: student[2])  
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9,  
20160303)]
```

## 예제

```
>>> class Student:
    def __init__(self, name, grade, number):
        self.name = name
        self.grade = grade
        self.number = number
    def __repr__(self):
        return repr((self.name, self.grade, self.number))

>>> students = [
    Student('홍길동', 3.9, 20160303),
    Student('김철수', 3.0, 20160302),
    Student('최자영', 4.3, 20160301),
]
>>> sorted(students, key=lambda student: student.number)
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9, 20160303)]
```

# 오름차순 정렬과 내림차순 정렬

```
>>> sorted(students, key=lambda student: student.number, reverse=True)
[('홍길동', 3.9, 20160303), ('김철수', 3.0, 20160302), ('최자영', 4.3,
20160301)]
```

# LAB: 키를 이용한 정렬 예제

- 주소록을 작성한다고 하자. 간단하게 사람들의 이름과 나이만 저장하고자 한다. 사람을 Person 이라는 클래스로 나타낸다. Person 클래스는 다음과 같은 인스턴스 변수를 가진다.
  - name - 이름을 나타낸다.(문자열)
  - age - 나이를 나타낸다.(정수형)
- 나이순으로 정렬하여 보여주는 프로그램을 작성하자. 정렬의 기준의 사람의 나이이다.

[<이름: 홍길동, 나이: 20>, <이름: 김철수, 나이: 35>, <이름: 최자영, 나이: 38>]

# SOLUTION

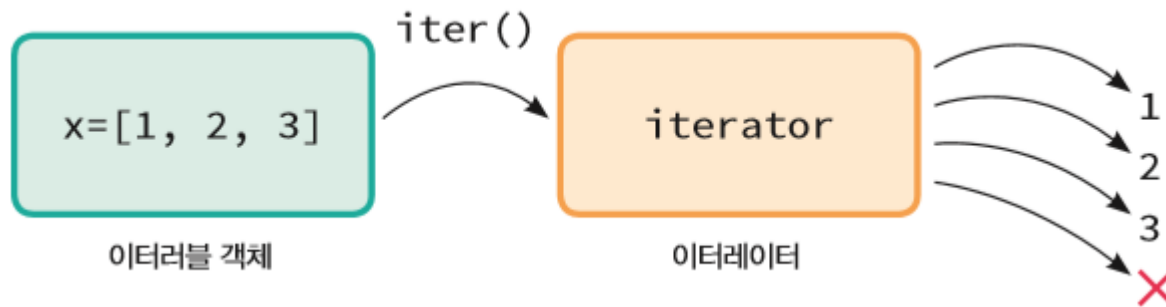
```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __repr__(self):
        return "<이름: %s, 나이: %s>" % (self.name, self.age)

def keyAge(person):
    return person.age

people = [Person("홍길동", 20), Person("김철수", 35), Person("최자영", 38)]
print(sorted(people, key = keyAge))
```

# 이터레이터

- 파이썬에서는 for 루프와 함께 사용할 수 있는 여러 종류의 객체가 있으며 이들 객체는 이터러블 객체 (iterable object)이라고 불린다.



# 제너레이터

제너레이터(generators)는 키워드 `yield`를 사용하여 함수로부터 이터레이터를 생성하는 하나의 방법이다.

```
def myGenerator():  
    yield 'first'  
    yield 'second'  
    yield 'third'
```



## 예제

```
def myGenerator():  
    yield 'first'  
    yield 'second'  
    yield 'third'  
  
for word in myGenerator():  
    print(word)
```

```
first  
second  
third
```

# 클로저

클로저(closure)는 함수에 의하여 반환되는 함수이다.

```
def addNumber(fixedNum):  
    def add(number):  
        return fixedNum + number  
    return add
```

```
func = addNumber(10)  
print(func(20))
```

30