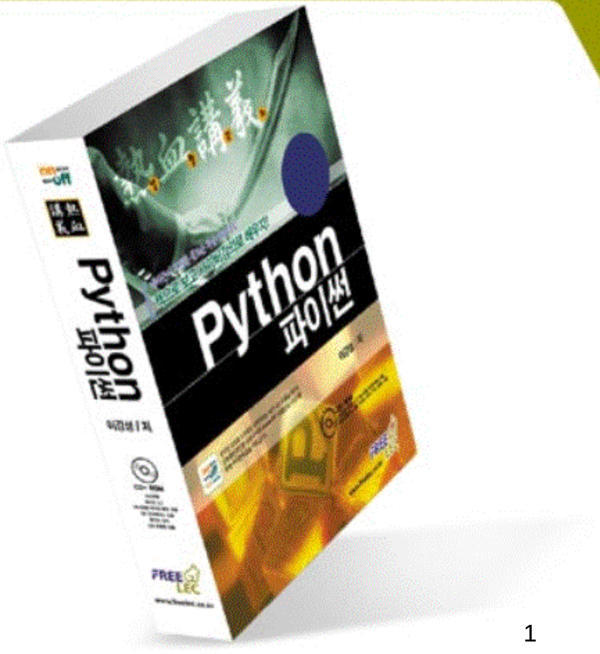
熱血講義

프리렉의 열혈강의 시리즈

Python
파이썬



열혈강의 Python 파이썬

❖제 10 강

주제 : 함수

이강성(gslee@mail.gwu.ac.kr)

❖강의 목차

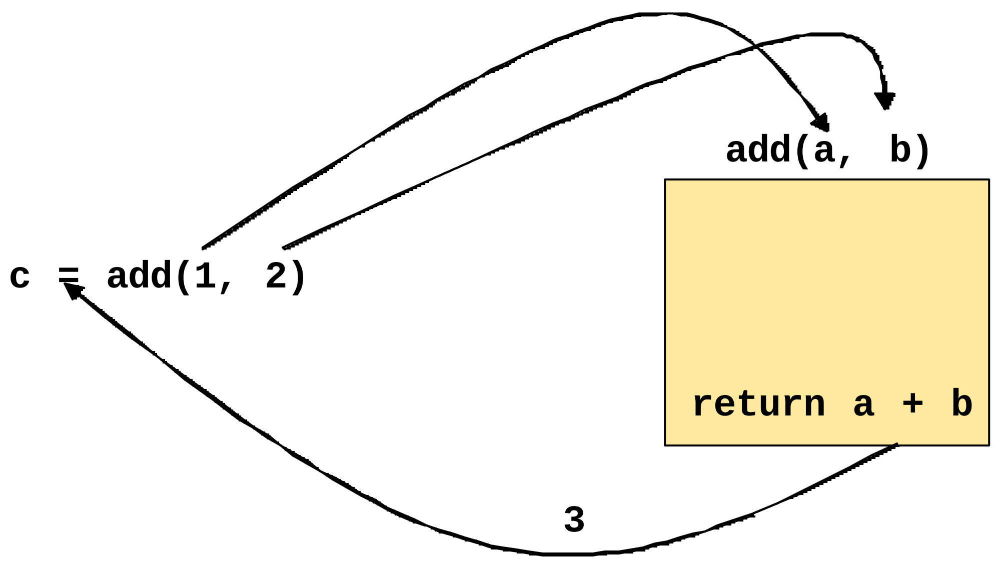
10-1 함수란?

- 함수란?
  - 제어문들을 하나로 묶은 단위
  - 반복적인 수행이 가능
  - 캡슐화
  - 새로운 연산의 생성
  - 코드의 부분을 작고 큰 논리적 단위로 분리하고 계층적으로 프로그램을 설계

10-2 함수의 정의와 호출

- 함수의 정의

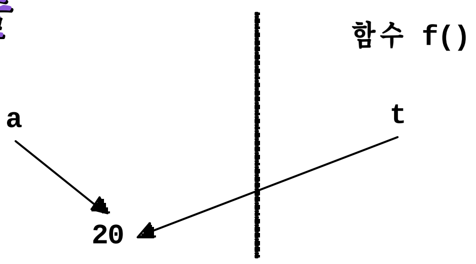
```
def 함수명 (인수들..):
    문들(statements)
    return <값>

>>> def add(a, b):  # 더하기 함수의 정의
        return a+b

>>> add
<function add at 80ca0d8>

>>> c = add(1,2) # a에 1, b에 2전달된다.
>>> c
3
>>> f = add
>>> f(4,5)
9
```



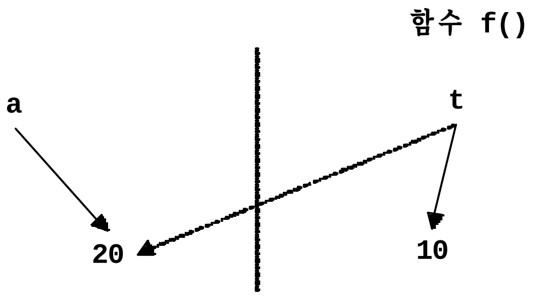
10-3 인수 전달 방법

- 참조에 의한 호출

```
>>> def f(t):
        t = 10

>>> a = 20
>>> f(a)
>>> print a
20
```



(a) 함수 f 를 막 호출 했을 때



(b) t=10 을 수행하고 나서 리턴하기 직전

10-4 return 문에 대하여

- 인수 없이 리턴하기

```
>>> def nothing():
        return

>>> nothing()
>>> a = nothing()
>>> print a
None

>>>
```

- return 문 없이 리턴하기

```
>>> def simple():
      pass

>>> simple()
>>> print simple()
None
>>>
```

10-4 return 문에 대하여

- 한 개의 값 리턴하기

```
# abs.py
def abs(x):
   if x < 0: return -x

   return x
```

- 두 개 이상의 값 리턴하기
  - 튜플을 이용한 방법

```
# swap.py
def swap(x, y):
    return y, x          # 튜플로 리턴된다.

a, b = swap(b, a)        # a, b = b, a와 동일
x = swap(a, b)
```

  - 리스트를 이용한 방법

10-5 동적인 자료형 결정

- 객체의 연산은 실행될 때 결정된다
  - a.__add__(b)를 호출

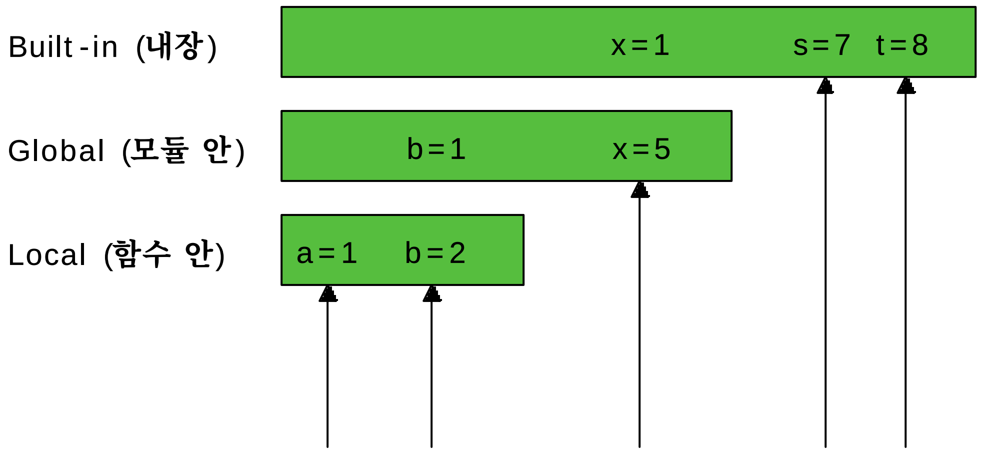
```
>>> def add(a, b):
    return a+b

>>> c = add(1, 3.4)
>>> d = add('dynamic', 'typing')
>>> e = add(['list'], ['and', 'list'])
```

10-6 스코핑 룰

- 이름 공간(name space)
  - 이름들(변수, 함수, 클래스, 인스턴스등)이 정의되는 공간
  - 종류
    - 지역(local)
      - 함수 내부 영역
    - 전역(global)
      - 모듈(파일) 영역
    - 내장(built-in)

10-6 스코핑 룰

- LGB rule (2.0까지)



10-6 스코핑 룰

- 변수는 정의되는 이름 공간에 생긴다

```
# g, h는 전역 이름
g = 10
h = 5

def f(a):          # a는 지역
    h = a + 10     # h는 지역, 새로 정의했으므로
    b = a + g      # b도 지역, g는 전역
    return  b
```

10-6 스코핑 룰

- 지역 영역에서 전역 변수 사용시 문제점

```
g = 10

def  f():
   a = g          # 1) g 지역? 전역?
   g = 20         # 2) g 지역? 전역?
   return  a

f()
```

10-6 스코핑 룰

- global의 사용

```
def  f():
   global g      # 1) 전역
   a = g         # 2) 전역
   g = 20        # 3) 전역
   return  a
```

- 내장 영역 이름 알아보기

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

10-6 스코핑 룰

- 중첩 영역(nested scopes)

```
#  nested01.py
x = 2                # global
def  F():
    x = 1            # ???
    def  G():
        print  x
    G()

F()
```

10-6 스코핑 룰

- 중첩 영역

```
# nested04.py
def  bank_account1(initial_balance):
     balance  =  initial_balance
     def  deposit(amount):
          balance  =  balance  +  amount
          return  balance
     def  withdraw(amount):
          balance  =  balance  -  amount
          return  balance
     return  deposit,  withdraw

d,  w  =  bank_account1(100)
print  d(100)
```

10-6 스코핑 룰

- 중첩 영역

```
# nested05.py
def  bank_account2(initial_balance):
     balance  =  [initial_balance]
     def  deposit(amount):
          balance[0]  =  balance[0]  +  amount
          return  balance[0]
     def  withdraw(amount):
          balance[0]  =  balance[0]  -  amount
          return  balance[0]
     return  deposit,  withdraw

d,  w  =  bank_account2(100)
print  d(100)
```

10-7 함수 인수

- 기본 인수 값
  - 함수를 호출 할 때 인수를 넘겨주지 않는 경우 기본 값을 취하는 것

```
>>> def incr(a,  step=1):
    return a + step

>>> b = 1
>>> b = incr(b)  # 1 증가
>>> b = incr(b,  10)      # 10 증가
>>> b
12
```

10-7 함수 인수

- 기본 인수 값 사용시 주의할 사항
  - 함수 정의시 기본 값 없는 인수가 기본 값 있는 인수 다음에 나올 수 없다

```
def decr(step=1,  b):
    pass
```

10-7 함수 인수

- 키워드 인수
  - 인수 이름으로 값을 전달하는 방식

```
# arg01.py
def area(height, width):
    return height * width

a = area(width=20, height=10)
b = area(height='height strng ', width=3)
```

  - 함수 호출 시에 키워드 인수 이후에 순서에 의한 이수를 지정할 수 없다.

```
area(width=5, 20)
```

10-7 함수 인수

- 가변 인수 리스트
  - 가변 개수의 인수를 함수로 전달하는 방법
  - 고정 인수들을 우선 받고 가변 인수들은 튜플 형식으로 한꺼번에 받는다

```
>>> def varg(a, *arg):
        print a, arg

>>> varg(1)
1 ()
>>> varg(2, 3)
2 (3,)
>>> varg(2,3,4,5,6)
2 (3, 4, 5, 6)
```

10-7 함수 인수

- 가변 인수 리스트
  - C언어의 printf 문 흉내내기

```
def printf(format, *args):
    print format % args


printf("I've spent %d days and %d night to
   do this", 6, 5)
```

10-7 함수 인수

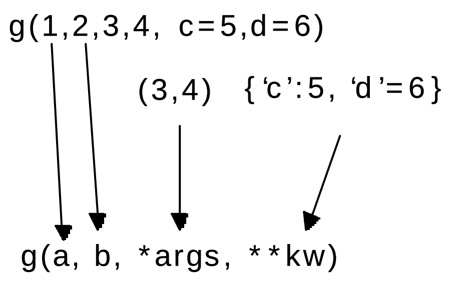
- 정의되지 않은 키워드 인수 처리하기
  - 키워드 인수로 함수를 호출할 때, 미리 정의되어 있지 않은 키워드 인수를 받는 방법
  - 함수의 가인수 목록의 제일 마지막에 나와야 한다

```
>>> def f(width, height, **kw):
        print width, height
        print kw


>>> f(width=10, height=5, depth=10,
   dimension=3)
10 5
{'depth': 10, 'dimension': 3}
```

10-7 함수 인수

- 함수의 가인수 나열 순서
  1. 순서에 의한 인수
  2. 가변 인수
  3. 사전 키워드 인수

```
>>> def g(a, b, *args, **kw):
        print a, b
        print args
        print kw


>>> g(1,2,3,4, c=5, d=6)
1 2
(3, 4)
{'c': 5, 'd': 6}
```



10-7 함수 인수

- 튜플과 사전 인수로 함수 호출하기
  - 튜플을 이용한 함수 호출

```
>>> def h(a,b,c):
      print a, b, c

>>> args = (1,2,3)
>>> h(*args)
1 2 3
```

  - 사전을 이용한 함수 호출

```
>>> dargs = {'a':1, 'b':2, 'c':3}
>>> h(**dargs)
1 2 3
```

10-7 함수 인수

- 튜플과 사전 인수로 함수 호출하기
  - 튜플과 사전을 이용한 함수 호출

```
>>> args = (1,2)
>>> dargs = {'c':3}
>>> h(*args, **dargs)
1 2 3
```

10-8 람다(lambda) 함수

- 한 줄 짜리 이름 없는 함수

  lambda 콤마로 구분된 인수들: 식
- 사용 예:

```
>>> lambda:1
<function <lambda> at 1206850>

>>> f = lambda:1
>>> f()
1

>>> g = lambda x, y: x+y
>>>  g(1,2)
3
```

10-8 람다(lambda) 함수

- 기본 값 인수

```
>>> incr = lambda x,inc=1: x+inc
>>> incr(10)
11
```

- 가변 개수 인수

```
>>> vargs = lambda x, *args: args
>>> vargs(1,2,3,4,5)
(2, 3, 4, 5)
```

- 키워드 인수

```
>>> kwords = lambda x, *args, **kw: kw
>>> kwords(1, 2, 3, a=4, b=6)
{'b': 6, 'a': 4}
```

10-8 람다(lambda) 함수

- 람다 함수 사용하기
  - 함수를 인수로 넘겨야 하는 경우에 편리

```
>>> def g(func):
        res = []
        for x in range(-10, 10):
                res.append(func(x))
        return res

>>> g(lambda x:x*x + 3*x - 10)
[60, 44, 30, 18, 8, 0, -6, -10, -12, -12, -
   10, -6, 0, 8, 18, 30, 44, 60, 78, 98]
>>> g(lambda x:x*x*x)
…출력 생략…
```

10-8 람다(lambda) 함수

- 일반 함수와 람다 함수의 차이

구분	def로 정의되는 함수	lambda 함수
문/식	문(statement)	식(expression)
함수의 이름	def 다음에 지정된 이름으로 생성한 함수 객체를 치환한다	함수 객체만을 생성한다
몸체	한 개 이상의 문을 포함한다	하나의 식만 온다
리턴	return 문에 의해 명시적으로 리턴 값이 지정된다	식의 결과가 리턴된다
내부 변수 선언	지역 영역에 변수를 생성하고 사용하는 것이 가능하다	지역 영역에 변수를 생성하는 것이 가능하지 않다

10-8 람다(lambda) 함수

- 정적 중첩 영역의 지원(2.2)

```
>>> def  h(k):
        a  =  1
        f  =  lambda  x:  x+a
        return  f(k)

>>>  h(4)
5
```

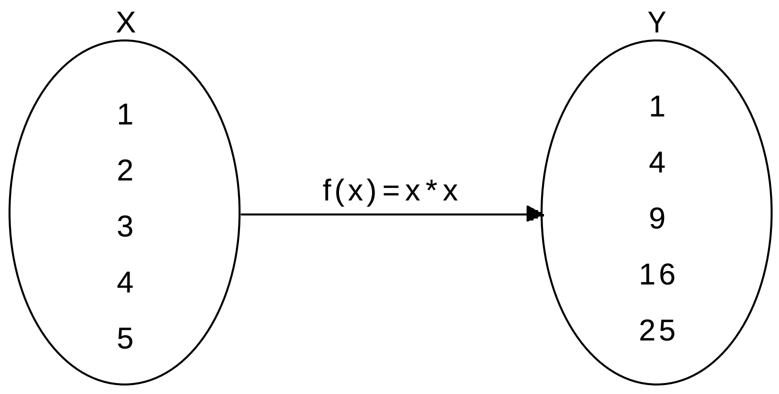
10-9 함수적 프로그래밍

- 함수적 프로그래밍이란?
  - Haskell에서 영향을 받음
  - 함수를 중심으로 한 프로그래밍
  - 명령적, 절차적이기 보다는 표현식(expression)에 중심을 둔 프로그래밍
  - 함수는 first-class 함수로, 다른 함수의 인수로 전달될 수 있고, 자료구조에 저장될 수도 있다.
- 함수적 프로그래밍에 사용되는 함수들
  - map, zip, filter, reduce, apply

10-9 함수적 프로그래밍

- map 내장 함수

```
X = [1,2,3,4,5]
Y = map(lambda a:a*a, X)
```

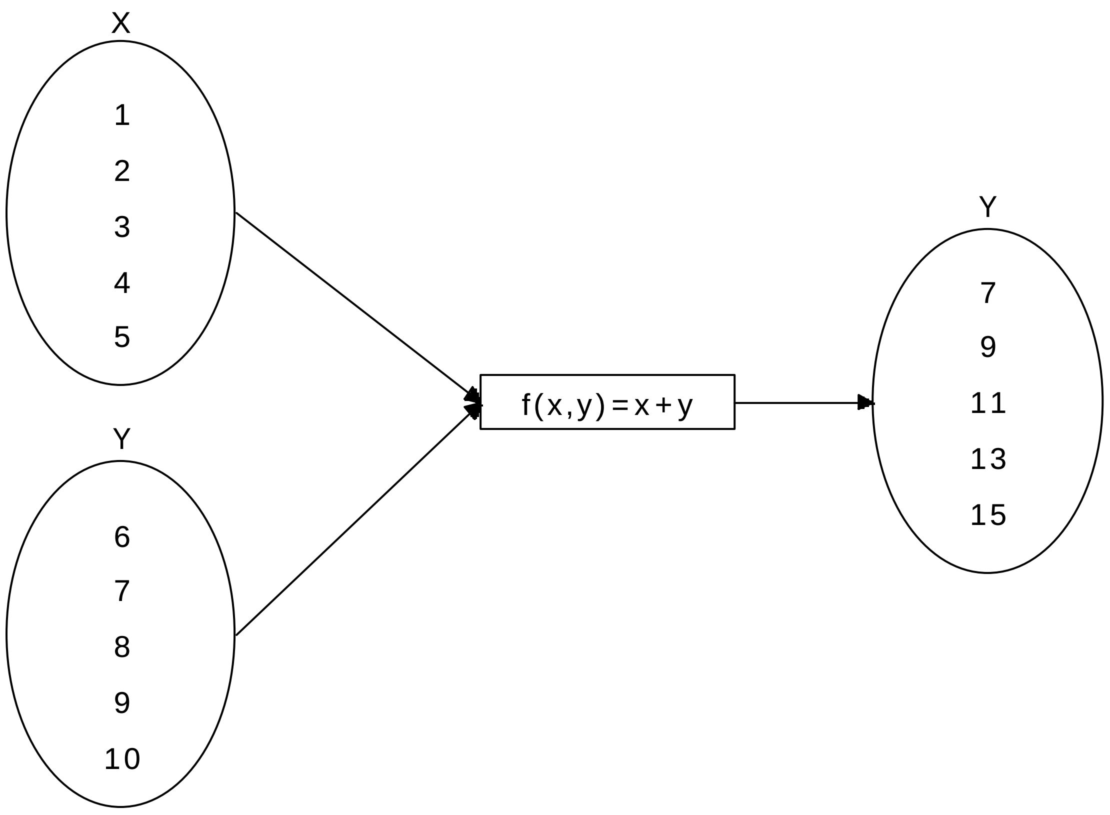


10-9 함수적 프로그래밍

- 두 개의 입력에 대한 map 내장 함수

```
>>> X = [1,2,3,4,5]
>>> Y = [6,7,8,9,10]
>>> Z = map(lambda x, y:x+y, X, Y)
>>> Z
[7, 9, 11, 13, 15]


>>> import operator
>>> Z = map(operator.add, X, Y)
```

10-9 함수적 프로그래밍



10-9 함수적 프로그래밍

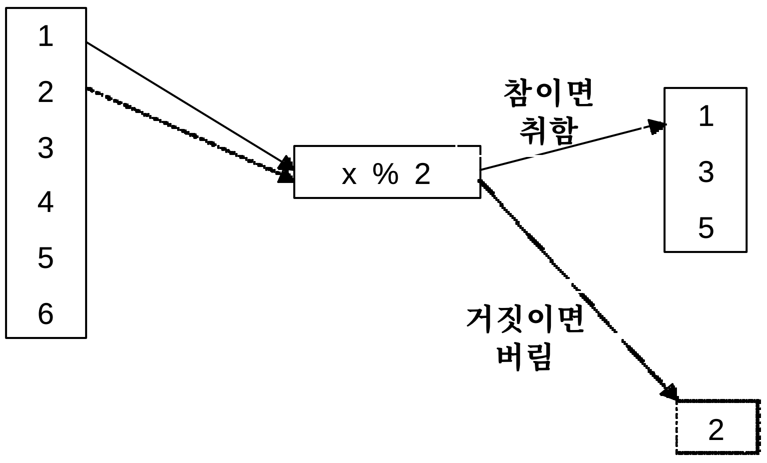
- 순서쌍 만들기(map, zip)

```
>>> a = [1, 2, 3, 4]
>>> b = [10, 20, 30, 40]
>>> map(None, a, b)
[(1, 10), (2, 20), (3, 30), (4, 40)]

>>> map(None, [1,2,3], [4,5,6,7,8])
[(1, 4), (2, 5), (3, 6), (None, 7), (None, 8)]

>>> zip([1,2,3], [4,5,6,7,8])
[(1, 4), (2, 5), (3, 6)]

>>> zip([1,2,3], [4,5,6], [7,8,9], [10, 11, 12])
[(1, 4, 7, 10), (2, 5, 8, 11), (3, 6, 9, 12)]
```

10-9 함수적 프로그래밍

- filter 내장 함수
  - 필터 함수를 통하여 참인 요소만 모은다

```
>>> filter(lambda x:x%2, [1,2,3,4,5,6])
[1, 3, 5]
```



10-9 함수적 프로그래밍

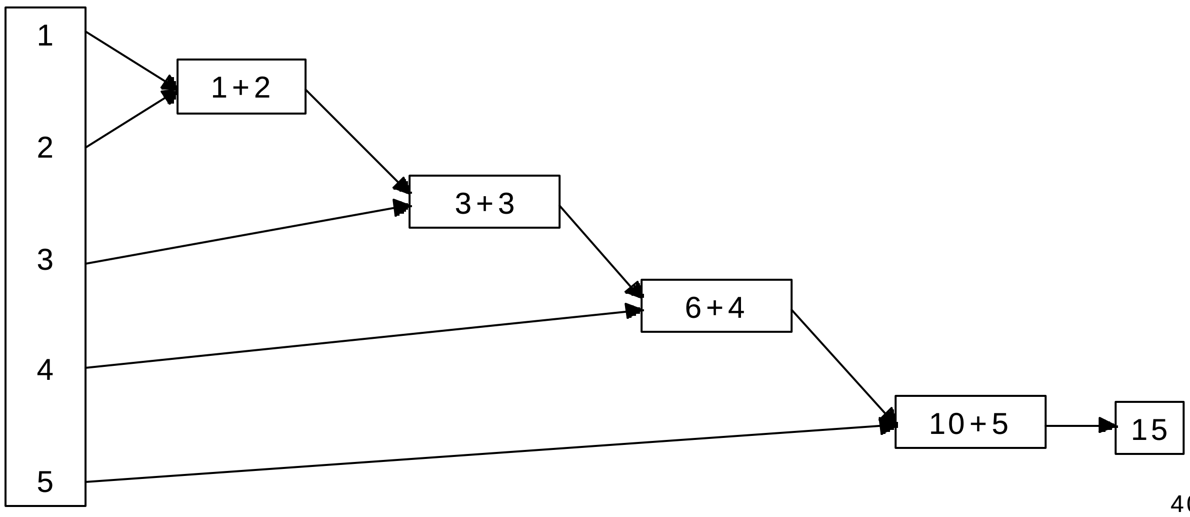
- filter 예제

```
>>> filter(lambda x:x>2, (1,2,3,34))   # 튜플 입출력
    (3, 34)
>>> filter(lambda x:x>2, [1,2,3,34])   # 리스트
[3, 34]
>>> filter(lambda x:x<'a', 'abcABCdefDEF') # 문자열

'ABCDEF'


>>> L = ['high', 'level', '', 'built-
    in', '', 'function']
>>> L = filter(None, L)
>>> L
['high', 'level', 'built-in', 'function']
```

10-9 함수적 프로그래밍

- reduce 내장 함수

```
>>> reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])

15
```



10-9 함수적 프로그래밍

- reduce 세 번째 인수는 초기 값

```
>>> reduce(lambda x, y: x+y, [1, 2, 3, 4, 5], 0)
15

>>> reduce(lambda x, y: x + y*y, range(1, 11), 0)
385
```

10-9 함수적 프로그래밍

- apply 내장 함수

```
>>> def  f(a,b,c):
        print  a,b,c

>>> args = (1,2,3)
>>> apply(f, args)
1 2 3
```

- 2.0 이상에서의 사용 방법

```
>>> f(*args)
1 2 3
```

10-11 재귀적 함수

- 자기 자신을 호출 하는 함수

```
>>> def  sum(N):
        if N == 1: return 1
        return N + sum(N-1)

>>>  sum(10)
55
```