컴퓨팅사고와 파이썬 프로그래밍

Ch 5. 함수 (Function)

교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; E-mail : ytkim@yu.ac.kr)

Outline

- 파이썬 함수 개요
- 코드블록, 네임스페이스 (name space)와 유효범위 (scope)
- 파이썬 프로그램의 변수 (variable)
- 함수의 인수 (argument) 전달 형식
- 함수에서 인수로 전달된 값의 변경
- 파이썬 내장 함수 (embedded function)
- 람다함수 (lambda function)
- 1급함수 (first class function)
- 재귀함수 (recursive function)
- 제네레이터 함수 (generator function)

파이썬 함수 개요

모듈화 프로그래밍의 개념

◆ 모듈(module)

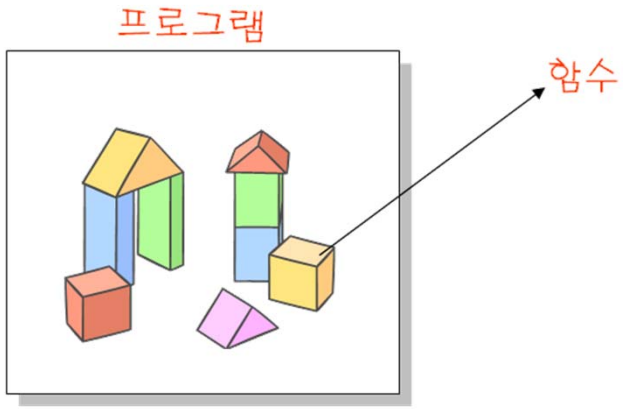
- 독립되어 있는 프로그램의 일부분

◆ 모듈화 프로그래밍

- 모듈 개념을 사용하는 프로그래밍 기법
- 프로그램에 포함되는 기능들을 모듈 별로 나누어 설계 및 구현
- 일부 모듈은 기존에 이미 개발되어 있는 모듈이나 라이브러리를 사용

◆ 모듈화 프로그래밍의 장점

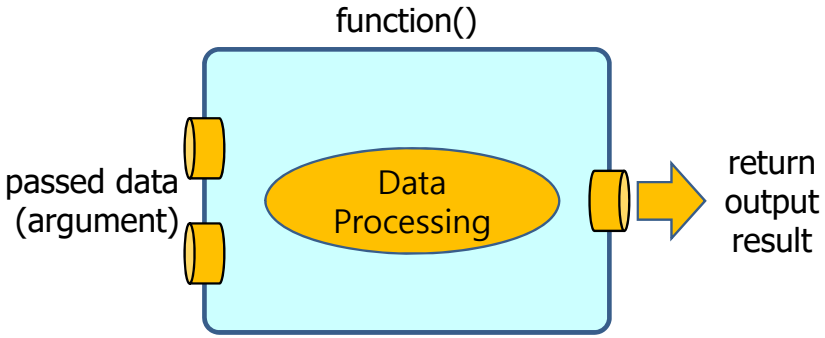
- 각 모듈들은 독자적으로 개발 가능
- 다른 모듈과 독립적으로 변경 가능
- 유지 보수가 쉬워진다.
- 모듈의 재사용 가능



함수의 개념

◆ 함수(function)

- 함수 : 특정한 작업을 수행하는 독립적인 모듈
- 함수 호출(function call) : 함수를 호출하여 사용하는 것
- 함수는 입력을 받으며 출력을 생성한다.



함수 사용의 장점과 단점

◆ 함수 사용의 장점

- 함수를 사용하면 코드가 중복되는 것을 막을 수 있다.
- 한번 작성된 함수 (모듈)는 여러 번 재사용할 수 있다.
- 함수를 사용하면 전체 프로그램을 모듈로 나눌 수 있어서 개발 과정이 쉬워지고 보다 체계적이 되면서 유지보수도 쉬워진다.

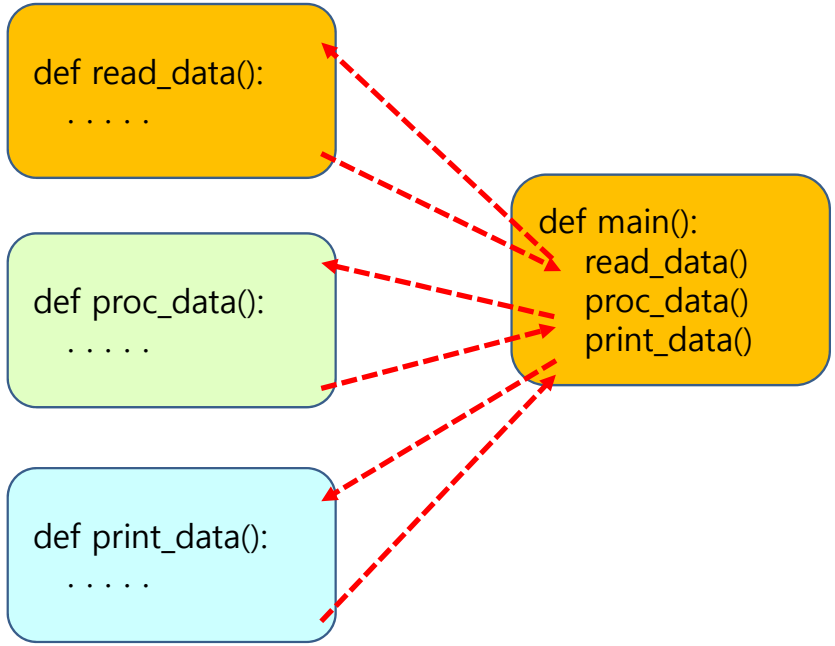
◆ 함수 사용의 단점

- 함수 호출을 할 때 마다 운영체제가 함수에서 사용되는 지역 변수들의 준비와 인수 (argument) 전달 등을 처리해야 하므로 부담이 발생한다.
- 따라서 너무 작은 단위의 함수를 구성하여 자주 호출하는 경우 성능에 문제가 발생할 수도 있다.

함수들의 연결

◆ 함수들의 연결

- 프로그램은 여러 개의 함수들로 이루어진다.
- 함수 호출을 통하여 서로 서로 연결된다.
- 함수들의 실행을 종합적으로 제어 및 관리하는 함수가 main()이다.



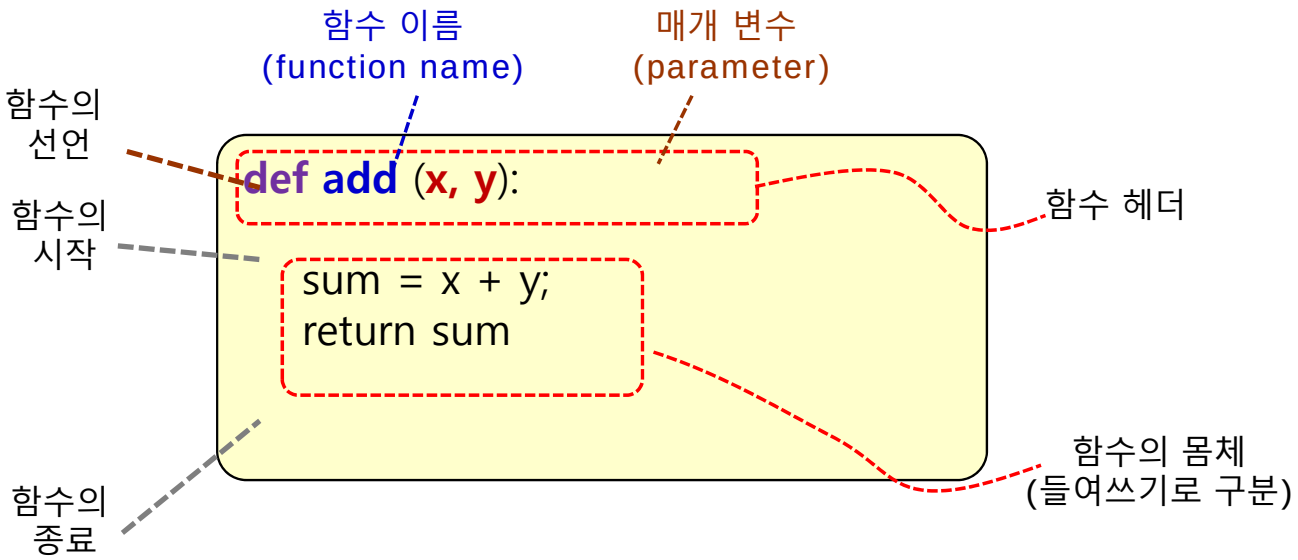
함수의 종류

◆ 사용자 정의함수 vs. Library 함수

함수 (function)	개발 및 제공	예
라이브러리 함수 (library function)	• 프로그래밍언어에서 기본적인 함수로 제공	• input() • print() • time() • rand()
사용자 정의 함수 (user defined function)	• 사용자가 직접 설계 및 구현 • 필요에 따라 필요한 함수를 구현	• mtrx_add() • mtrx_subtract() • mtrx_multiply() • print_mtrx() • print_list_sample()

함수의 구조

◆ 함수의 구조


함수 이름
(function name)
매개 변수
(parameter)
함수의
선언
함수의
시작
함수의
종료
def add (x, y):
sum = x + y;
return sum
함수 헤더
함수의 몸체
(들여쓰기로 구분)

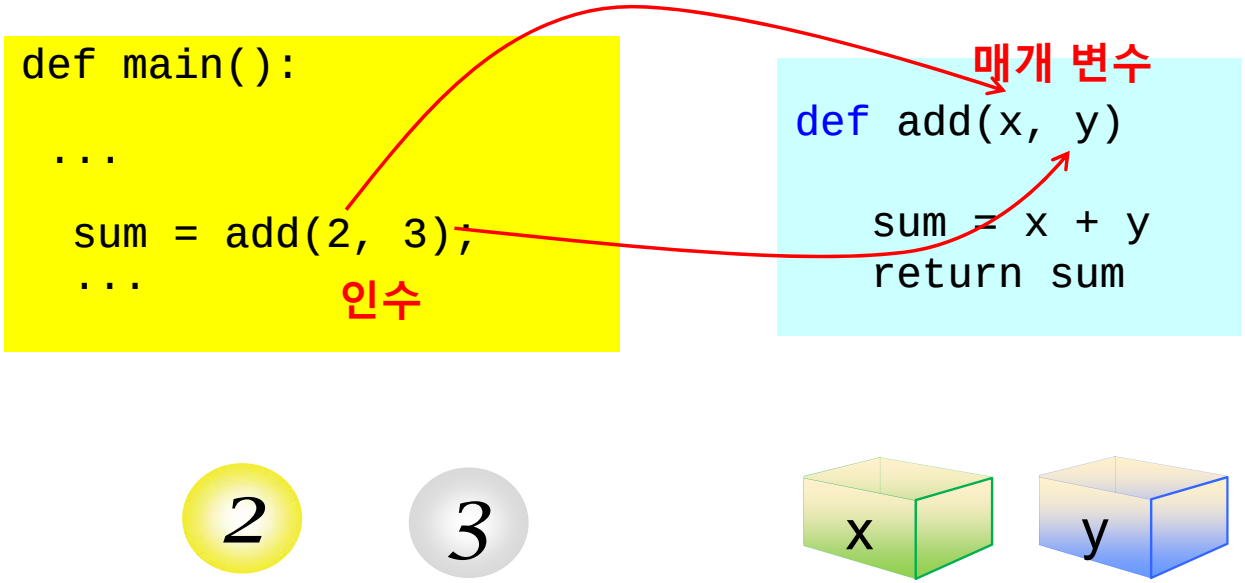
매개 변수 (parameter), 인수 (argument)

◆ **매개 변수(parameter): 형식 인수, 형식 매개 변수라고도 한다.**

- 함수 원형 (function prototype) 선언에서 형식 인수 (형식 매개변수) 데이터 유형 지정

◆ **인수(argument): 실인수, 실매개 변수라고도 한다.**

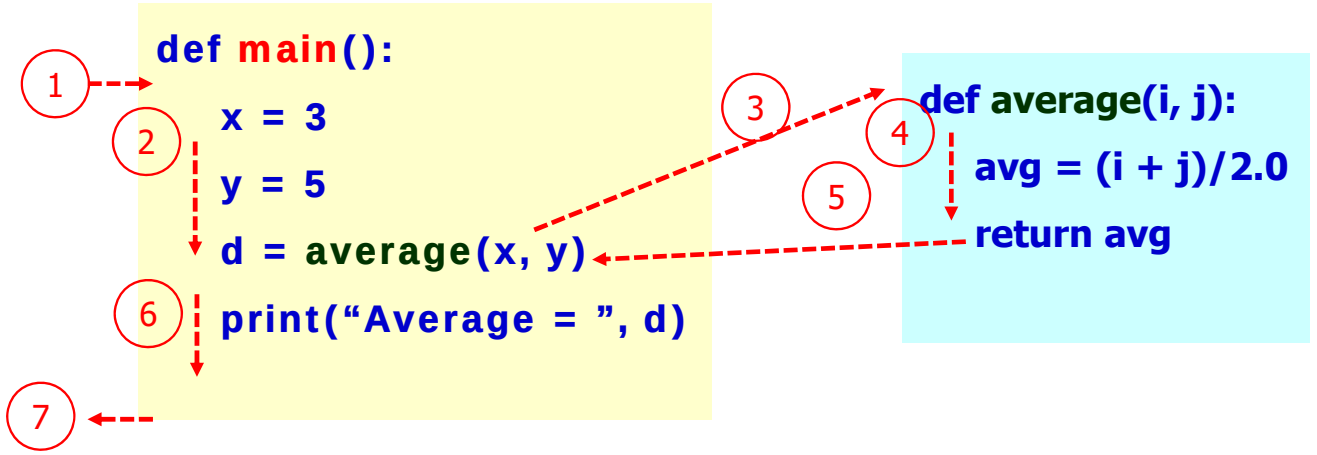
- 프로그램 실행단계에서 실제 함수 호출에 전달되는 데이터



함수 호출과 결과값의 반환

◆ 함수 호출(function call):

- 함수를 사용하기 위하여 함수의 이름을 적어주는 것
- 함수안의 문장들이 순차적으로 실행된다.
- 문장의 실행이 끝나면 호출한 위치로 되돌아 간다.
- 결과값을 전달할 수 있다.

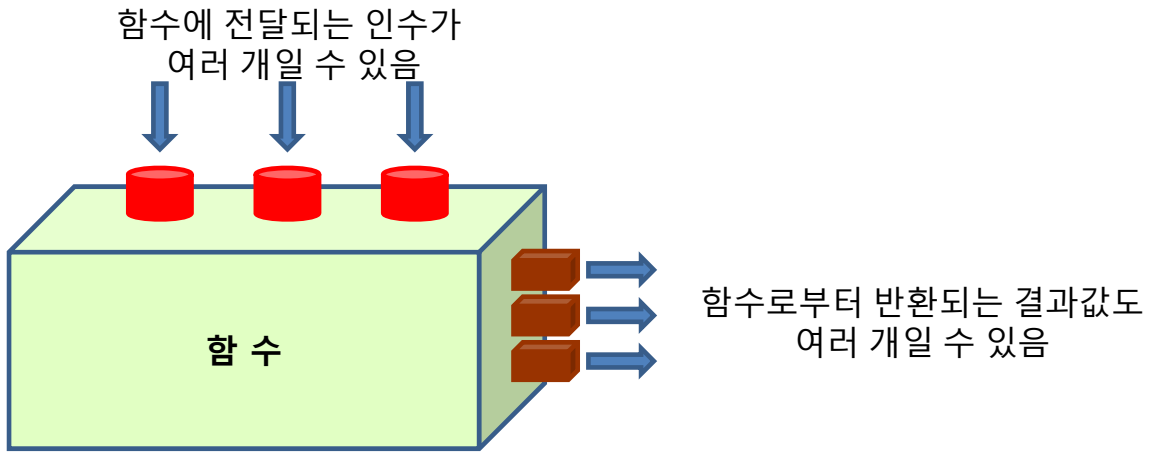


함수 실행 결과의 반환 (return)

◆ 함수 실행 결과의 반환

- **반환 값(return value)**: 호출된 함수가 호출한 곳으로 작업의 결과값을 전달하는 것
- 반환 값을 여러 개 보낼 수 있음

```
return 0
return x
return x, y, z
```



파이썬 함수

◆ 파이썬 함수의 기본 구조

```
def function_name(<parameter list>):
    "description of function"
    statement_1
    statement_2
    sub_block_1:
        sub_block_statement_1_1
        sub_block_statement_1_2
    statement_3
    return ret_val

def add(x, y):
    "add x and y"
    sum = x + y
    return sum

def multiply(x, y):
    "multiply x with y"
    result = x * y
    return result
```

```
# Example usage of function

def printList(L):
    "print list L"
    L_len =  len(L)
    for i in range(L_len):
        print("{:3}".format(L[i]), end=' ')
    print()

# main()
L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("list L = ", end=' ')
printList(L)
```

```
list L =   0   1   2   3   4   5   6   7   8   9
```

파이썬 함수 사용의 예 – find_min_max()

◆ Function find_min_max(List)

```
# Function call with list arguments and return tuples

def find_min_max(L):
    nMin = nMax = L[0]
    for e in L:
        if nMin > e:
            nMin = e
        if nMax < e:
            nMax = e
    return nMin, nMax

#
print('min, max = find_min_max([3, 1, 0, 5, 6, 9, 4, 2])')
min, max = find_min_max([3, 1, 0, 5, 6, 9, 4, 2])
print('min : ', min)
print('max : ', max)
#
print('min, max = find_min_max([3, 1, 0, 5, 6, 9, 4, 2, -1, 10])')
min, max = find_min_max([3, 1, 0, 5, 6, 9, 4, 2, -1, 10])
print('min : ', min)
print('max : ', max)
```

```
 RESTART: C:/YTK-Progs/2018 Book (Python)/ch 5  Function
rn tuple.py
min, max = minmax([3, 1, 0, 5, 6, 9, 4, 2])
min :  0
max :  9
min, max = minmax([3, 1, 0, 5, 6, 9, 4, 2, -1, 10])
min :  -1
max :  10
>>>
```

함수의 속성

◆ 함수의 속성

속성	설명
__doc__	함수를 설명하는 문서 또는 문자열
__name__	함수의 이름
__qualname__	전역 (global scope)에서 함수가 포함된 모듈의 경로 (path)을 나타내며 dot (.)을 포함하여 표시
__module__	함수가 선언되어 있는 모듈의 이름
__defaults__	매개 변수와 기본값 (default value) 의 튜플
__code__	함수를 컴파일하여 생성된 코드 객체
__globals__	함수가 포함된 모듈의 전역 이름 영역에 대한 딕셔너리
__closure__	함수의 변수 바인딩에 대한 셀 튜플

함수의 호출과 인수 전달

◆ Function definition and function call

```
# Function call with arguments

def add(x, y):
    return x + y

#
print('add(1, 2) : ', add(1, 2)) #addition of integers
print('add(1.0, 2.0) : ', add(1.0, 2.0)) #addition of float data
print('add("abc", "def") : ', add("abc", "def")) # addition of strings
print('add([1, 2, 3], [3, 4, 5]) : ', add([1, 2, 3], [3, 4, 5])) #addition of list
print('add((1, 2, 3), (3, 4, 5)) : ', add((1, 2, 3), (3, 4, 5))) #addition of tuples
print('add({1, 2, 3}, {3, 4, 5}) : ', add({1, 2, 3}, {3, 4, 5})) #addition of sets
```

```
add(1, 2) :  3
add(1.0, 2.0) :  3.0
add("abc", "def") :  abcdef
add([1, 2, 3], [3, 4, 5]) :  [1, 2, 3, 3, 4, 5]
add((1, 2, 3), (3, 4, 5)) :  (1, 2, 3, 3, 4, 5)
Traceback (most recent call last):
  File "C:\MyPyPackage\TextBook - 2019\ch 6  Function def, gen
asic Functions, Parameter, Argument\(2) function calls - add()
    print('add({1, 2, 3}, {3, 4, 5}) : ', add({1, 2, 3}, {3, 4
  File "C:\MyPyPackage\TextBook - 2019\ch 6  Function def, gen
asic Functions, Parameter, Argument\(2) function calls - add()
    return x + y
TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

함수 실행 결과 반환

```
# Python function that returns multiple data - circle_AreaCircum()

PI = 3.141592
def circle_AreaCircum(radius):
    ar = PI * radius * radius # calculation of area
    circum = 2.0 * PI * radius # calculation of circumference
    return ar, circum

r = 10.0
area, circumference = circle_AreaCircum(r)

print("Circle of radius ({}) :\n Area({}),\n Circumference({})"\
      .format(r, area, circumference))
```

```
Circle of radius (10.0) :
 Area(314.1592),
 Circumference(62.83184)
```

코드 블록

◆ 코드블록 정의

- 모듈
- 함수 (function)
- 클래스 (class)

```
#Block_0_Start
Header-line:
    #Block_1_Start
    Header-line:
        #Block_2_Start
        Header-line:
        . . . . .
        #Block_2_End
    #Block_1_End
#Block_0_End
```

```
# Example of code block - fibo()
def fibo(n):
    if n < 0:
        return None
    elif 0 <= n < 2:
        return n
    else:
        return fibo(n-2) + fibo(n-1)
# -----------
for n in range(10):
    fibo_n = fibo(n)
    print("fibo({:3}) = {:7}".format(n, fibo_n))
```

이름 공간 (Name Space)

◆ 이름공간 (네임스페이스, Name space)

- name: variable name, function name, class name, module name
- name space: dict of name and bound object

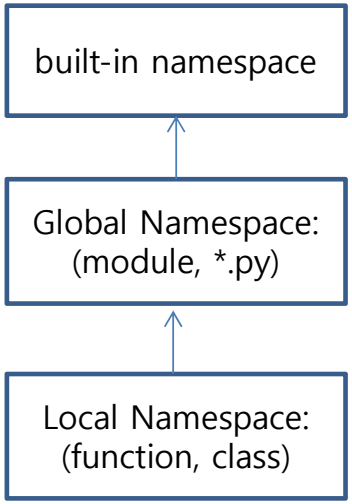
Name Space	설 명
지역 (local)	함수 또는 클래스 코드 블록의 내부에서 바인딩된 이름 locals() 함수로 확인 가능
전역 (global)	코드 블록이 속한 모듈의 최상위 (top-level) 네임 스페이스이며, 모듈 전체에서 사용가능한 이름으로 globals() 함수로 확인 가능
내장 (built-in)	파이썬에 내장되어 있는 객체: 내장 모듈 (__builtins__) 또는 dir(__builtins__)로 확인 가능

유효 범위 (Scope)

◆ 유효범위 (Scope)

- 이름이 어느 유효 범위에 속하는지 정의
- 유효 범위의 우선 규칙 (name scope rule)
  - 지역 이름 범위 (local namespace/scope)
  - 전역 이름 범위 (global namespace/scope)
  - 내장형 이름 범위 (built-in namespace/scope)

◆ 이름 범위의 상속 관계



이름 영역의 명시적 설정 – global, nonlocal

◆ 이름 영역 (name scope)의 명시적 설정 명령

Name scope 지정 선언	의미
global	지역 변수로 선언된 것을 사용하지 않고 전역 변수로 선언된 것을 사용
nonlocal	지역 변수로 선언된 것을 사용하지 않고, 그 코드 영역 바깥에서 선언한 변수를 사용

지역 변수(local variable)와 전역변수(global variable)

```
# Global vs. Local, Name space, scope

x = 1 # global variable
y = 3 # global variable
z = 5 # global variable
def func():
    x = 10 # local variable
    global y
    y = 30 # global variable
    z = 50 # local variable
    print("in func():: x = {0:2d}, y = {1:2d}, z = {2:2d}".format(x, y, z))
    def sub_func():
        nonlocal z
        z = 100
        print("in sub_func():: x = {0:2d}, y = {1:2d}, z = {2:2d}".format(x, y, z))
        print("in sub_func():: locals() : ", locals())
        print("in sub_func():: globals() : ", globals())
    sub_func()
    print("in func():: x = {0:2d}, y = {1:2d}, z = {2:2d}".format(x, y, z))
    print("in func():: locals() : ", locals())
    print("in func():: globals() : ", globals())

# -------------
func()
print("main:: x = {0:2d}, y = {1:2d}, z = {2:2d}".format(x, y, z))
```

```
in func():: x = 10, y = 30, z = 50
in sub_func():: x = 10, y = 30, z = 100
in sub_func():: locals() :  {'x': 10, 'z': 100}
in sub_func():: globals() :  {'__name__': '__main__', '__doc__': None
, '__package__': None, '__loader__': <class '_frozen_importlib.Builti
nImporter'>, '__spec__': None, '__annotations__': {}, '__builtins__':
<module 'builtins' (built-in)>, '__file__': 'C:\\MyPyPackage\\TextBoo
k - 2019\\ch 6 Function def, generator, trigonometric\\6.5 Variables
- local, global\\local vs global variables - func(), inner().py', 'x'
: 1, 'y': 30, 'z': 5, 'func': <function func at 0x032E6DB0>}
in func():: x = 10, y = 30, z = 100
in func():: locals() :  {'sub_func': <function func.<locals>.sub_func
at 0x032E6DF8>, 'x': 10, 'z': 100}
in func():: globals() :  {'__name__': '__main__', '__doc__': None, '_
_package__': None, '__loader__': <class '_frozen_importlib.BuiltinImp
orter'>, '__spec__': None, '__annotations__': {}, '__builtins__': <mo
dule 'builtins' (built-in)>, '__file__': 'C:\\MyPyPackage\\TextBook -
2019\\ch 6 Function def, generator, trigonometric\\6.5 Variables - lo
cal, global\\local vs global variables - func(), inner().py', 'x': 1,
'y': 30, 'z': 5, 'func': <function func at 0x032E6DB0>}
main:: x =  1, y = 30, z =  5
```

파이썬 함수 선언에서의
파라메타 (parameter) 형식 정의 및
함수호출에서의 인수 (argument) 전달

파이썬 함수의 인수 전달

형식지정 위치	Parameter/Argument Syntax	설명
함수 **선언** (**파라 메터 형식** 정의)	def func(arg)	보통 파라메타 (위치 파라메타) 선언
	def func(arg=**value**)	보통 파라메타 (위치 파라메타)에 기본값 설정
	def func(*varargs)	가변적 위치 파라메타 선언 (임의 개수의 위치 파라메터가 전달될 수 있음)
	def func(**kwargs)	가변적 키워드 파라메타 선언 (임의 개수의 키워드 파라메터가 전달될 수 있음)
	def func(arg, *other)	보통 파라메타와 가변적 위치 파라메타 사용 선언
	def func(*arg, **kwargs)	가변적 위치 파라메타와 가변적 키워드 파라메타 사용
	def func(*, name=value)	*표시 이후에는 키워드 파라메타만 사용하도록 선언하며, 기본값 설정
함수 **호출** (**인수** 정의)	func(arg1, arg2, , . . .)	보통 위치 인수 설정, 위치 인수로 전달
	func(arg=value)	키워드 인수, 이름으로 파라메타에 연계
	func(**iterable*)	열거 가능 *iterable* (e.g., string, list, tuple) 객체로 인수를 전달, iterable 객체의 각 개체들이 위치 인수로 연계됨
	func(**dict)	키워드:값으로 표현되는 항목들을 dict 자료형 인수로 전달되며, 인수에 포함된 키워드를 사용하여 키워드 파라메타에 전달됨

위치 인수 (Positional Argument)

```
# Python function with positional arguments

def func_posargs(x, y, z):
    print("passed arguments: x = {0}, y = {1}, z = {2}".format(x, y, z))
    return 3*x + 2*y + z

w = func_posargs(1, 2, 3) # positional arguments
print('w = func_posargs(1, 2, 3) ==> w : ', w)
```

```
passed arguments: x = 1, y = 2, z = 3
w = func_posargs(1, 2, 3) ==> w :  10
```

기본값 (default value)을 지정하는 위치인수

```
# Positional arguments with default values

def volume(width=1, length=1, height=1):
    print("passed arguments: width = {:3}, length = {:3}, height = {:3}"\
          .format(width, length, height))
    return width * length * height

print("volume() = ", volume())
print("volume() = ", volume(10))
print("volume() = ", volume(10, 20))
print("volume() = ", volume(10, 20, 30))
```

```
passed arguments: width =   1, length =   1, height =   1
volume() =  1
passed arguments: width =  10, length =   1, height =   1
volume() =  10
passed arguments: width =  10, length =  20, height =   1
volume() =  200
passed arguments: width =  10, length =  20, height =  30
volume() =  6000
```

키워드 인수 (Keyword argument)

```
# Function call with positional arguments vs keyword arguments

def div(x, y):
    return x / y

#
print('div(1, 2) : ', div(1, 2)) # function call with positional arguments
print('div(2, 1) : ', div(2, 1)) # function call with positional arguments
print('div(x = 3, y = 4) : ', div(x = 3, y = 4)) # function call with keyword arguments
print('div(y = 4, x = 3) : ', div(y = 4, x = 3)) # function call with keyword arguments
```

```
div(1, 2) :  0.5
div(2, 1) :  2.0
div(x = 3, y = 4) :  0.75
div(y = 4, x = 3) :  0.75
```

dict 자료형으로 키워드 인수를 전달

```
# Passing keyword arguments using dict

def func_dict_args(x, y, z):
    print("passed arguments: x = {0}, y = {1}, z = {2}"\
          .format(x, y, z))
    return 3*x + 2*y + z

w = func_dict_args(**{'x':1, 'y':2, 'z':3})
  # passing keyword arguments using dict
print('w = func_dict_args(**{'x':1, 'y':2, 'z':3}) ==> w : ', w)
```

```
passed arguments: x = 1, y = 2, z = 3
w = func_dict_args(**{'x':1, 'y':2, 'z':3}) ==> w :  10
```

* 표시 다음에 키워드 인수만을 사용하는 함수

```
# Keyword-only arguments

def keywordOnlyFunc(x, *, y, z):
    print("passed arguments: x = {0}, y = {1}, z = {2}"\
          .format(x, y, z))
    return 3*x + 2*y + z

keywordOnlyFunc(1, y=2, z=3)
keywordOnlyFunc(1, **{'z':3, 'y':2})
keywordOnlyFunc(1, 2, 3)
```

```
passed arguments: x = 1, y = 2, z = 3
passed arguments: x = 1, y = 2, z = 3
Traceback (most recent call last):
  File "C:\MyPyPackage\TextBook - 2019\ch 6  Function def, generator, trigon
ometric\6.2 Function Arguments\(5) keyWordOnly Arguments - keywordOnlyfunc()
.py", line 10, in <module>
    keywordOnlyFunc(1, 2, 3)
TypeError: keywordOnlyFunc() takes 1 positional argument but 3 were given
```

가변적 위치 인수
(variable positional argument)

```
# Variable positional arguments

def varPosArgsFunc(*args):
    print(type(args))
    print("args = ", args)
    s = 0
    for x in args:
      s += x
    return s

varPosArgsFunc(1)
varPosArgsFunc(1, 2)
varPosArgsFunc(1, 2, 3)
```

```
<class 'tuple'>
args =  (1,)
<class 'tuple'>
args =  (1, 2)
<class 'tuple'>
args =  (1, 2, 3)
```

가변적 위치 인수의 함수 호출에 다양한 자료형 인수를 전달

```
# Variable Positional arguments

def varPosArgsFunc(*args):
    print("args = <{}>, type= {}, len={}".format(args, type(args), len(args)))
    num_arg = len(args)
    for a in range(num_arg):
        t_arg = type(args[a])
        print("args[{}]: type = {}".format(a,t_arg))
        if t_arg == int:
            print("arg ({}) : ".format(t_arg), args[a])
        elif t_arg == float:
            print("arg ({}) : ".format(t_arg), args[a])
        elif (t_arg == list) or (t_arg == tuple):
            print("arg ({}) : ".format(t_arg), end='')
            len_list = len(args[a])
            for n in range(len_list):
                print(args[a][n], end=', ')
            print()
        else:
            print("Currently argument type {} cannot be processed !!".format(t_arg))
a, b = 1, 2.5
L = [1, 2, 3, 4, 5]
T = ('one', 'two', 'three')
print("L : ", L)
print("T : ", T)
varPosArgsFunc(a, b, L, T) # invoke function with variable positional arguments
```

```
L :  [1, 2, 3, 4, 5]
T :  ('one', 'two', 'three')
args = <(1, 2.5, [1, 2, 3, 4, 5], ('one', 'two'
, 'three'))>, type= <class 'tuple'>, len=4
args[0]: type = <class 'int'>
arg (<class 'int'>) :  1
args[1]: type = <class 'float'>
arg (<class 'float'>) :  2.5
args[2]: type = <class 'list'>
arg (<class 'list'>) : 1, 2, 3, 4, 5,
args[3]: type = <class 'tuple'>
arg (<class 'tuple'>) : one, two, three,
```

위치인수와 가변적 위치인수

```
# Positional arguments and variable positional arguments

def varPosArgsFunc(a, *args):
    print("a = ", a, "args = ", args)
    print(type(args))
    s = 0
    for x in args:
      s += x
    return s

varPosArgsFunc(1)
varPosArgsFunc(1, 2)
varPosArgsFunc(1, 2, 3)
varPosArgsFunc(1, 2, 3, 4, 5)
```

```
a =  1 args =  ()
<class 'tuple'>
a =  1 args =  (2,)
<class 'tuple'>
a =  1 args =  (2, 3)
<class 'tuple'>
a =  1 args =  (2, 3, 4, 5)
<class 'tuple'>
```

가변적 키워드 인수 (variable keyword argument)

```
# Variable keyword arguments (1)

def varKwArgsFunc(**kwargs):
    print("kwargs = <{}>, type= {}, len={}".format(kwargs, type(kwargs),  len(kwargs)))
    num_arg = len(kwargs)
    print("num_arg : ", num_arg)
    for kw in kwargs:
        arg_type = type(kwargs[kw])
        #print("kwargs({}): type = {}".format(kw, type(kwargs[kw])))
        if arg_type == int:
            print("kwarg ({}) : {}".format(kw, kwargs[kw]))
        elif arg_type == float:
            print("kwarg ({}) : {}".format(kw, kwargs[kw]))
        elif arg_type == str:
            print("kwarg ({}) : {}".format(kw, kwargs[kw]))
        elif (arg_type == list) or (arg_type == tuple):
            print("kwarg ({}) : {}".format(kw, kwargs[kw]), end=' ')
            kwarg = kwargs[kw]
            len_list = len(kwarg)
            print("; elements of {} : ".format(type(kwarg)), end=' ')
            for n in range(len_list):
               print(kwarg[n], end=', ')
            print()
        else:
             print("Currently argument type {} cannot be processed!!".format(kw_arg))
```

```
# Variable keyword arguments (2)

L = [1, 2, 3, 4, 5]
T = ('one', 'two', 'three')
print("L : ", L)
print("T : ", T)
varKwArgsFunc(a=1)  # invoke function with variable keywords arguments
varKwArgsFunc(a=1, b=2.5)  # invoke function with variable keywords arguments
varKwArgsFunc(a=1, b=2.5, c=L)  # invoke function with variable keywords arguments
varKwArgsFunc(a=1, b=2.5, c=L, d=T)  # invoke function with variable keywords arguments
```

```
L :  [1, 2, 3, 4, 5]
T :  ('one', 'two', 'three')
kwargs = <{'a': 1}>, type= <class 'dict'>, len=1
num_arg :  1
kwarg (a) : 1
kwargs = <{'a': 1, 'b': 2.5}>, type= <class 'dict'>, len=2
num_arg :  2
kwarg (a) : 1
kwarg (b) : 2.5
kwargs = <{'a': 1, 'b': 2.5, 'c': [1, 2, 3, 4, 5]}>, type= <class 'dict'>, len=3
num_arg :  3
kwarg (a) : 1
kwarg (b) : 2.5
kwarg (c) : [1, 2, 3, 4, 5] ; elements of <class 'list'> :  1, 2, 3, 4, 5,
kwargs = <{'a': 1, 'b': 2.5, 'c': [1, 2, 3, 4, 5], 'd': ('one', 'two', 'three')}>, type= <class 'dict'>, len=4
num_arg :  4
kwarg (a) : 1
kwarg (b) : 2.5
kwarg (c) : [1, 2, 3, 4, 5] ; elements of <class 'list'> :  1, 2, 3, 4, 5,
kwarg (d) : ('one', 'two', 'three') ; elements of <class 'tuple'> :  one, two, three,
```

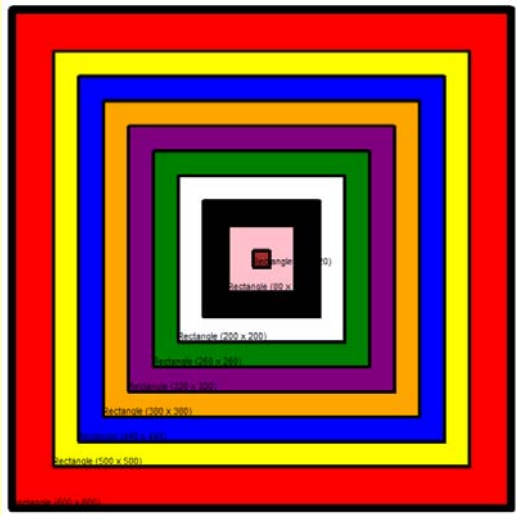
가변적 위치인수와 가변적 키워드 인수의 혼합

```
# Variable Positional arguments and variable Keyword arguments

def varPosVarKeywordFunc(*args, **kwargs):
    print("args = ", args)
    print("kwargs = ", kwargs)
    s = "var args: ( "
    for x in args:
        s += "{}, ".format(str(x))
    s += " ); var kwargs: { "
    for x in kwargs:
        s += "{}:{}, ".format(x, str(kwargs[x]))
    s += "}"
    return s

r1 = varPosVarKeywordFunc(1, A=1)
print('varPosVarKeywordFunc(1, A=1) ==> : \n', r1)

r2 = varPosVarKeywordFunc(1, 2, A=10, B=20)
print('varPosVarKeywordFunc(1, 2, A=10, B=20) ==> : \n', r2)

r3 = varPosVarKeywordFunc(1, 2, 3, A=10, B=20, C=30)
print('varPosVarKeywordFunc(1, 2, 3, A=10, B=20, C=30) ==> : \n', r3)
```

```
args =  (1,)
kwargs =  {'A': 1}
varPosVarKeywordFunc(1, A=1) ==> :
 var args: ( 1,  ); var kwargs: { A:1, }
args =  (1, 2)
kwargs =  {'A': 10, 'B': 20}
varPosVarKeywordFunc(1, 2, A=10, B=20) ==> :
 var args: ( 1, 2,  ); var kwargs: { A:10, B:20, }
args =  (1, 2, 3)
kwargs =  {'A': 10, 'B': 20, 'C': 30}
varPosVarKeywordFunc(1, 2, 3, A=1, B=2, C=30) ==> :
 var args: ( 1, 2, 3,  ); var kwargs: { A:10, B:20, C:30, }
```

파이썬 함수 사용의 예 – draw_rect()

```
# Function draw_rect() with arguments

import turtle
def draw_rect(pen, width, length, fill_color = "white", pen_color = "black", psize = 4):
    pen.pensize(psize)
    pen.color(pen_color, fill_color)
    pen.setheading(0)
    pen.up();  pen.setpos(-width//2, -length//2);   pen.down()

    pen.begin_fill()
    pen.setpos(width//2, -length//2)
    pen.setpos(width//2, length//2)
    pen.setpos(-width//2, length//2)
    pen.setpos(-width//2, -length//2)
    pen.end_fill()

#----------------------
t = turtle.Turtle()
t.ht() #hide turtle
colors = ["red", "yellow", "blue", "orange", "purple",\
              "green", "white", "black", "pink", "brown"]
c = 0
width, length = 600, 600
draw_rect(t, width, length, colors[c], "black", 10)
c += 1
t.write("Rectangle ({} x {})".format(width, length))
for i in range(250, 1, -30):
    width, length = i*2, i*2
    draw_rect(t, width, length, colors[c])
    c += 1
    t.write("Rectangle ({} x {})".format(width, length))
```



변경되는 인수 (Mutable Arguments)

```
# Mutable arguments

def funcA(x):
    print("at funcA(), before assignment :: x = {}, id = {}".format(x, id(x)))
    if isinstance(x, list): # list is mutable
        x[0] = 10 # 리스트 원소 변경
    if isinstance(x, dict): # dict is mutable
        x['color'] = 'red' # 딕셔너리 원소 변경
        x['price'] = 300 # 딕셔너리 원소 변경
    print("at funcA(), after assignment :: x = {}, id = {}".format(x, id(x)))

L = [1, 2, 3] # list
print("at main(), before function call :: L = {}, id = {}".format(L, id(L)))
funcA(L)
print("at main(), after function call :: L = {}, id = {}".format(L, id(L)))

D = {'color':'blue', 'price':100} # dictionary
print("at main(), before function call :: D = {}, id = {}".format(D, id(D)))
funcA(D)
print("at main(), after function call :: D = {}, id = {}".format(D, id(D)))
```

```
at main(), before function call :: L = [1, 2, 3], id = 66086920
at funcA(), before assignment :: x = [1, 2, 3], id = 66086920
at funcA(), after assignment :: x = [10, 2, 3], id = 66086920
at main(), after function call :: L = [10, 2, 3], id = 66086920
at main(), before function call :: D = {'color': 'blue', 'price': 100}, id = 63611680
at funcA(), before assignment :: x = {'color': 'blue', 'price': 100}, id = 63611680
at funcA(), after assignment :: x = {'color': 'red', 'price': 300}, id = 63611680
at main(), after function call :: D = {'color': 'red', 'price': 300}, id = 63611680
```

변경되지 않는 인수 (Immutable Arguments)

```
# Immutable arguments

def funcB(x):
    print("at funcB(), before assignment :: x = {}, id = {}".format(x, id(x)))
    if isinstance(x, int): # int is immutable
        x = 10 # 새로운 정수형 변수 생성
    if isinstance(x, tuple):
       x = (10, 20, 30) # 새로운 튜플 생성
    if isinstance(x, list):
        x = [10, 20, 30] # 새로운 리스트 생
    print("at funcB(), after assignment :: x = {}, id = {}".format(x, id(x)))

a = 1 # int variable
print("at main(), before function call :: a = {}, id = {}".format(a, id(a)))
funcB(a)
print("at main(), after function call :: a = {}, id = {}".format(a, id(a)))
print()

T = (1, 2, 3) # tuple
print("at main(), before function call :: T = {}, id = {}".format(T, id(T)))
funcB(T)
print("at main(), after function call :: T = {}, id = {}".format(T, id(T)))
print()

L = [1, 2, 3] # list
id(L)
print("at main(), before function call :: L = {}, id = {}".format(L, id(L)))
funcB(L)
print("at main(), after function call :: L = {}, id = {}".format(L, id(L)))
```

```
at main(), before function call :: a = 1, id = 1533536176
at funcB(), before assignment :: x = 1, id = 1533536176
at funcB(), after assignment :: x = 10, id = 1533536320
at main(), after function call :: a = 1, id = 1533536176

at main(), before function call :: T = (1, 2, 3), id = 62198216
at funcB(), before assignment :: x = (1, 2, 3), id = 62198216
at funcB(), after assignment :: x = (10, 20, 30), id = 62198344
at main(), after function call :: T = (1, 2, 3), id = 62198216

at main(), before function call :: L = [1, 2, 3], id = 62130248
at funcB(), before assignment :: x = [1, 2, 3], id = 62130248
at funcB(), after assignment :: x = [10, 20, 30], id = 62130824
at main(), after function call :: L = [1, 2, 3], id = 62130248
```

파이썬 내장 함수 (embedded functions)

파이썬 내장 함수 (1)

분류	파이썬 내장 함수	설명
자료형 변수 생성/변환	bool(x)	x를 bool 형태로 변환 출력
	int(i_str)	문자열 i_str을 정수 데이터로 변환하여 반환
	float(f_str)	문자열 f_str을 부동 소수점으로 변환하여 반환
	complex(c_str)	문자열 c_str을 복소수로 변환하여 반환
	complex(r, i)	실수 r과 i를 전달받아 실수부 (real part)가 r, 허수부(imaginary part)가 i인 복소수 객체를 생성
	bin(x)	정수값 x를 전달받아 2진수 bit (binary digit) 문자열로 변환하여 반환
	hex(x)	정수값 x를 전달받아 16진수 bytes 문자열로 변환하여 반환
	oct(x)	정수 형태의 데이터 x를 전달받아 8진수 bytes 문자열로 변환하여 반환
	str(object)	객체 object를 문자열 형태로 변환하여 반환
	list(s)	반복 가능한 자료형 데이터 s를 전달받아 리스트로 만들어 반환
	tuples(s)	반복 가능한 자료형 데이터 s를 전달받아 tuple 형태로 변환하여 반환
수치 데이터 연산	abs(x)	x의 절대값 (absolute value) 복소수 (x + yj)인 경우 절대값 sqrt(x**2 + y**2)
	divmod(a, b)	a를 b로 나눈 몫과 나머지 (모듈로 계산)를 tuple로 출력 divmod(7, 3) => (2, 1)
	max(L)	반복 가능한 자료형 데이터 L을 전달받아 최댓값을 찾아 반환
	min(L)	반복 가능한 자료형 데이터 L을 전달받아 최솟값을 찾아 반환
	pow(x, y)	x를 y 제곱한 지수승 결과값을 반환
	round(n, r)	숫자 n을 소수점 r자리까지 반올림하여 반환

파이썬 내장 함수 (2)

분류	파이썬 내장 함수	설명
문자/문자열 연산	chr(i)	ASCII 코드 값 i에 해당하는 문자를 출력
	ord(c)	문자 c의 UNICODE 코드 값을 출력
반복 가능 자료형 연산	all(x)	반복 (iterate) 가능한 자료형 x의 모든 값이 True이면 True, 그렇지 않으면 False
	any(x)	반복 (iterate) 가능한 자료형 x의 값 중 하나라도 True이면 True, 그렇지 않으면 (모두가 False 이면) False
	enumerate()	순서가 있는 자료형 데이터를 전달받아 차례로 출력
	filter(a, b)	반복가능한 자료형 데이터 b를 함수 a에 대입하였을 때 결과가 True인 것만 출력
	len(s)	전달받은 s의 길이를 반환
	map(func, x)	함수 func와 반복 가능한 자료형 데이터 x를 전달받고 전달 받은 데이터 x의 각 요소를 함수 func에 적용하여 실행된 결과를 반환
	sorted(L)	인수 L을 정렬하여 리스트로 반환
	sum(L)	리스트 L에 포함된 모든 항목들을 합산하여 반환
	zip(i)	동일한 개수로 이루어진 자료형 (list, tuple)을 차례대로 짝으로 묶어 줌
객체	dir()	객체가 자체적으로 가지고 있는 변수나 함수를 표시
	id(object)	객체 object의 고유 주소 (identifier) 값을 출력
	isinstance(o, c)	객체 o가 클래스 c의 인스탄스인가를 판단하여 참이면 True, 거짓이면 False를 반환
	type(object)	전달된 객체 object의 자료형을 반환

파이썬 내장 함수 (3)

분류	파이썬 내장 함수	설명
파일/ 디렉토리 관리	open(f, m)	파일이름 f를 모드 m으로 열고 그 파일 객체를 반환 fin = open("InputData.txt", 'r')
	compile(s, f, n)	파일 f로부터 소스 s의 문자열을 읽어 컴파일하고 Python code object를 생성하며, 생성된 객체는 exec() 또는 eval()를 사용하여 실행
실행 가능 문자열	eval(expression)	실행 가능한 문자열을 전달받아 이를 실행한 후 결과값을 반환
	exec()	문자열로 된 Python 실행문을 전달받아 실행
입출력	input(prompt)	화면에 prompt를 출력하고, 표준입력장치(키보드)로부터 데이터를 입력
	print()	화면으로 지정된 양식에 따라 출력
함수	lambda()	def와 같은 함수 생성 기능을 제공하며, 이름이 없는 함수를 생성

람다 함수 (Lambda function)

◆ 람다 함수 (Lambda function)

- 함수의 이름을 설정하지 않고 함수의 기능만을 설정한 함수
- 주로 함수의 호출에서 인수로 전달되어, 해당 함수에서 처리해야 하는 데이터들에 대하여 람다 함수의 기능이 수행되도록 함

```
# Lambda function - double, triple

double = lambda x : x * 2
triple = lambda x : x + x + x

print("double : ", double)
print("double(5) = ", double(5))
print("double('ABC') = ", double('ABC'))

print("triple : ", triple)
print("triple(5) = ", triple(5))
print("triple('ABC') = ", triple('ABC'))
```

```
double :  <function <lambda> at 0x03260348>
double(5) =  10
double('ABC') =  ABCABC
triple :  <function <lambda> at 0x006D2108>
triple(5) =  15
triple('ABC') =  ABCABCABC
```

람다함수를 인수로 전달

◆ Lambda function as a parameter

```
# Lambda function - used as parameter

from types import * # types.LamdaType, types.FunctionType
def G(f, n=0):
    f_type = type(f)
    print("G(f)::f_type = ", f_type)
    if f_type is LambdaType or f_type is FunctionType:
        return [f(x) for x in range(n)]
    else:
        return None

# -------------
print("G(10) => ", G(10))
print("G(lambda x: x**2, 10) => ", G(lambda x: x**2, 10))
print("G(lambda x: x**3, 5) => ", G(lambda x: x**3, 5))
```

```
G(f)::f_type =  <class 'int'>
G(10) =>  None
G(f)::f_type =  <class 'function'>
G(lambda x: x**2, 10) =>  [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
G(f)::f_type =  <class 'function'>
G(lambda x: x**3, 5) =>  [0, 1, 8, 27, 64]
```

반복가능 자료형의 연산을 위한 내장 함수 - map()

◆ map() function

```
# map() function

X = [1, 2, 3, 4, 5]
Sq_X = list(map(lambda x : x**2, X))
print("X    : ", X)
print("Sq_X : ", Sq_X)

Y = [1, 2, 3, 4, 5]
print("Y    : ", Y)
Pow_X_Y = list(map(pow, X, Y))
print("X^Y  : ", Pow_X_Y)
```

```
X    :  [1, 2, 3, 4, 5]
Sq_X :  [1, 4, 9, 16, 25]
Y    :  [1, 2, 3, 4, 5]
X^Y  :  [1, 4, 27, 256, 3125]
```

반복가능 자료형의 연산을 위한 내장 함수 - filter()

◆ filter()

```
# Python embedded function - filter()

X = list(range(-5, 6, 1))
print("X : ", X)
Negative_X = list(filter(lambda x : x<0, X))  # select only
negative element
print("X with filtering(x < 0)  : ", Negative_X)

Positive_X = list(filter(lambda x : x>=0, X))  # select only
positive element
print("X with filtering(x >= 0) : ", Positive_X)

Non_zero_X = list(filter(None, X)) # select non-zero element
print("X with filtering(None) : ", Non_zero_X)
```

```
X :  [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
X with filtering(x < 0)  :  [-5, -4, -3, -2, -1]
X with filtering(x >= 0) :  [0, 1, 2, 3, 4, 5]
X with filtering(None) :  [-5, -4, -3, -2, -1, 1, 2, 3, 4, 5]
```

반복가능 자료형의 연산을 위한 내장 함수 - zip()

◆ zip(), *zip()

```
# zip(), zip(*)

X = [1, 2, 3]
Y = ['a', 'b', 'c']
Z = [100, 200, 300]
print('X : ', X)
print('Y : ', Y)
print('Z : ', Z)

Zip_XY = list(zip(X, Y))
print('Zip_XY : ', Zip_XY)

A, B = zip(*Zip_XY)
print('A : ', A)
print('B : ', B)

D = list(zip(X, Y, Z))
print('D : ', D)

d1, d2, d3 = zip(*D)
print('d1 = ', d1)
print('d2 = ', d2)
print('d3 = ', d3)
```

```
X :  [1, 2, 3]
Y :  ['a', 'b', 'c']
Z :  [100, 200, 300]
Zip_XY :  [(1, 'a'), (2, 'b'), (3, 'c')]
A :  (1, 2, 3)
B :  ('a', 'b', 'c')
D :  [(1, 'a', 100), (2, 'b', 200), (3, 'c', 300)]
d1 =  (1, 2, 3)
d2 =  ('a', 'b', 'c')
d3 =  (100, 200, 300)
```

반복가능 자료형의 연산을 위한 내장 함수 – sorted()

```
# Python embedded function - sorted()

L = [5, 7, 3, 1, 9, 2, 4, 6, 8, 0] # list
print("L : ", L)
print("sorted(L) : ", sorted(L))
print("sorted(L, reverse=True) : ", sorted(L, reverse=True))

TL = [("Park", 26, 77.4), ("Kim", 25, 74.7), ("Lee", 23, 80.5), ("Ahn", 24, 82.2)]
print("TL : ", TL)
print("sorted(TL) : ", sorted(TL))
print("sorted(TL, reverse=True) : \n",  sorted(TL, reverse=True))
print("sorted(TL, key = lambda person : person[1]) : \n",\
      sorted(TL, key = lambda person : person[1]))
print("sorted(TL, key = lambda person : person[2]) : \n",\
      sorted(TL, reverse=True, key = lambda person : person[2]))
```

```
L :  [5, 7, 3, 1, 9, 2, 4, 6, 8, 0]
sorted(L) :  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sorted(L, reverse=True) :  [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
TL :  [('Park', 26, 77.4), ('Kim', 25, 74.7), ('Lee', 23, 80.5), ('Ahn', 24, 82.2)]
sorted(TL) :  [('Ahn', 24, 82.2), ('Kim', 25, 74.7), ('Lee', 23, 80.5), ('Park', 26, 77.4)]
sorted(TL, reverse=True) :
 [('Park', 26, 77.4), ('Lee', 23, 80.5), ('Kim', 25, 74.7), ('Ahn', 24, 82.2)]
sorted(TL, key = lambda person : person[1]) :
 [('Lee', 23, 80.5), ('Ahn', 24, 82.2), ('Kim', 25, 74.7), ('Park', 26, 77.4)]
sorted(TL, key = lambda person : person[2]) :
 [('Ahn', 24, 82.2), ('Lee', 23, 80.5), ('Park', 26, 77.4), ('Kim', 25, 74.7)]
```

파이썬 객체 (object) 관리에 관련된 내장 함수

◆ type(), dir()

```
# Python embedded function - type(), dir()

L = [0, 1, 2]
T = (0, 1, 2, 3, 4)
D = {'A':10, 'B':11, 'C': 12, 'D': 13}

print("L : ", L)
print("type(L) = ", type(L))
print("dir(L) = ", dir(L))

print("\nT : ", T)
print("type(T) = ", type(T))
print("dir(T) = ", dir(T))

print("\nD : ", D)
print("type(D) = ", type(D))
print("dir(D) = ", dir(D))
```

```
L :  [0, 1, 2]
type(L) =  <class 'list'>
dir(L) =  ['__add__', '__class__', '__contains__', '__delattr
__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format
__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '
__hash__', '__iadd__', '__imul__', '__init__', '__init_subcla
ss__', '__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', 'remo
ve', 'reverse', 'sort']

T :  (0, 1, 2, 3, 4)
type(T) =  <class 'tuple'>
dir(T) =  ['__add__', '__class__', '__contains__', '__delattr
__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',
'__getattribute__', '__getitem__', '__getnewargs__', '__gt__'
, '__hash__', '__init__', '__init_subclass__', '__iter__', '_
_le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__rmul__', '__set
attr__', '__sizeof__', '__str__', '__subclasshook__', 'count'
, 'index']

D :  {'A': 10, 'B': 11, 'C': 12, 'D': 13}
type(D) =  <class 'dict'>
dir(D) =  ['__class__', '__contains__', '__delattr__', '__del
item__', '__dir__', '__doc__', '__eq__', '__format__', '__ge_
_', '__getattribute__', '__getitem__', '__gt__', '__hash__',
'__init__', '__init_subclass__', '__iter__', '__le__', '__len
__', '__lt__', '__ne__', '__new__', '__reduce__', '__reduce_e
x__', '__repr__', '__setattr__', '__setitem__', '__sizeof__',
'__str__', '__subclasshook__', 'clear', 'copy', 'fromkeys', '
get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'updat
e', 'values']
```

파이썬 객체 (object) 관리에 관련된 내장 함수

◆ id(), isinstance()

```
# Python embedded function - id(), isinstance() (1)

def funcA(arg):
    print("funcA:: arg (type({}) = {}".format(type(arg), arg))
    if isinstance(arg, int):
        print("arg(id({})) is integer of bit_length ({})".format(id(arg), arg.bit_length()))
    elif isinstance(arg, list):
        print("arg(id({})) is list of len ({})".format(id(arg), len(arg)))
    elif isinstance(arg, tuple):
        print("arg(id({})) is tuple of len ({})".format(id(arg), len(arg)))
    elif isinstance(arg, dict):
        print("arg(id({})) is dict of len ({})".format(id(arg), len(arg)))
    else:
        print("arg(id({})) is unknown, yet".format(id(arg)))
    print()

d = 1234567
L = [0, 1, 2]
T = (0, 1, 2, 3, 4)
D = {'A':10, 'B':11, 'C': 12, 'D': 13}
```

◆ id(), isinstance() (2)

```
# Python embedded function - id(), ininstance() (2)
# -------------------
print("d = {}, id(d) = {}".format(d, id(d)))
print("L = {}, id(L) = {}".format(L, id(L)))
print("T = {}, id(T) = {}".format(T, id(T)))
print("D = {}, id(D) = {}".format(D, id(D)))
funcA(d)
funcA(L)
funcA(T)
funcA(D)
```

```
d = 1234567, id(d) = 50512608
L = [0, 1, 2], id(L) = 35398200
T = (0, 1, 2, 3, 4), id(T) = 53045264
D = {'A': 10, 'B': 11, 'C': 12, 'D': 13}, id(D) = 50466272
funcA:: arg (type(<class 'int'>) = 1234567
arg(id(50512608)) is integer of bit_length (21)

funcA:: arg (type(<class 'list'>) = [0, 1, 2]
arg(id(35398200)) is list of len (3)

funcA:: arg (type(<class 'tuple'>) = (0, 1, 2, 3, 4)
arg(id(53045264)) is tuple of len (5)

funcA:: arg (type(<class 'dict'>) = {'A': 10, 'B': 11, 'C': 12, 'D': 13}
arg(id(50466272)) is dict of len (4)
```

내장함수(sum(), max(), min())를 사용한 배열의 통계 분석

```
# Application of Python embedded functions - array statistics
import math
def statistics(L):
    sum_L = sum(L)
    max_L = max(L)
    min_L = min(L)
    len_L = len(L)
    avg_L = sum_L / len_L
    sq_diff_sum = 0
    for x in L:
        sq_diff_sum += pow((x - avg_L), 2)
    var = sq_diff_sum / len_L
    std_dev = math.sqrt(var)
    print("L (size: {}) : {}".format(len_L, L))
    print("Statistics of L : Max ({}), Min ({}), Avg({:7.2f})"\
          .format(max_L, min_L, avg_L))
    print("  var ({:10.2f}), std_dev ({:10.2f})".format(var, std_dev))

#--------------------
A = list(range(10))
statistics(A)
B = list(range(-5, 5))
statistics(B)
C = list(range(0, -10, -1))
statistics(C)
```

```
L (size: 10) : [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Statistics of L : Max (9), Min (0), Avg(   4.50)
  var (      8.25), std_dev (      2.87)
L (size: 10) : [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4]
Statistics of L : Max (4), Min (-5), Avg(  -0.50)
  var (      8.25), std_dev (      2.87)
L (size: 10) : [0, -1, -2, -3, -4, -5, -6, -7, -8, -9]
Statistics of L : Max (0), Min (-9), Avg(  -4.50)
  var (      8.25), std_dev (      2.87)
```

1급 함수 (First class function)
함수 수식자 (Function Decorator)

1급 함수 (first class function)

◆1급 함수

- 파이썬 함수를 변수 이름으로 설정할 수 있는 함수
- 파이썬 함수를 다른 함수에게 인수로 전달 할 수 있는 함수
- 다른 함수의 실행 결과로 반환되는 함수

```
# First class function - Greeting

def Greeting(name):
    return "Hello, " + name

sayHello = Greeting # binding first class function to a variable name
print("sayHello('Kim') ==> ", sayHello('Kim'))
print("dir(sayHello) : ", dir(sayHello))
```

```
sayHello('Kim') ==>  Hello, Kim
dir(sayHello) :  ['__annotations__', '__call__', '__class__', '__closur
e__', '__code__', '__defaults__', '__delattr__', '__dict__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__get__', '__getattribute
__', '__globals__', '__gt__', '__hash__', '__init__', '__init_subclass_
_', '__kwdefaults__', '__le__', '__lt__', '__module__', '__name__', '__
ne__', '__new__', '__qualname__', '__reduce__', '__reduce_ex__', '__rep
r__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
```

1급함수를 함수호출의 인수로 전달

```
# Function as Object/Argument
#    to Another Function
import copy

def cubic(n):
    return n * n * n

def rFibo(n):
    if n <= 1:
        return n
    return rFibo(n-1) + rFibo(n-2)

def rFact(n):
    if n <= 1:
        return 1
    else:
        d = rFact(n-1)
        return n * d
```

```
def ApplyFunctionToAnotherFunction(L, func):
    """ Assume L is a list, func a function.
        Apply each element in L into function func,
        generates the result list, and return """
    L_result = []
    L_result.clear()

    for i in range(len(L)):
        d = L[i]
        res = func(d)
        L_result.insert(i, res)
    return L_result
```

◆ 실행 결과

```
##################################
#  Application

L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print("L = ", L)

print("\nApply cubic() for each element of list L :")
L_res = ApplyFunctionToAnotherFunction(L, cubic)
print(L_res)

print("\nApply rFact() for each element of list L :")
L_res = ApplyFunctionToAnotherFunction(L, rFact)
print(L_res)

print("\nApply rFibo() for each element of list L :")
L_res = ApplyFunctionToAnotherFunction(L, rFibo)
print(L_res)
```

```
L =  [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Apply cubic() for each element of list L :
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]

Apply rFact() for each element of list L :
[1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]

Apply rFibo() for each element of list L :
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

함수의 실행 시간을 측정 – measureTime()

```
#First class function as argument to another function
import time
def measureTime(f):
    def wrapper(n): # n is argument
        t_before = time.time()
        ret = f(n)
        t_after = time.time()
        elapsed_time = t_after - t_before
        #print("Execution time : {0}".format(elapsed_time))
        return ret, elapsed_time
    return wrapper

def rFibo(n):
    if n <= 1:
        return n
    return rFibo(n-1) + rFibo(n-2)

measureTime_rFibo = measureTime(rFibo)
for n in range(1, 35, 1):
    fibo_n, t_elapsed = measureTime_rFibo(n)
    print("Fibo({:3d}) = {:8d}, took {:10.3f} milli-seconds"\
          .format(n, fibo_n, t_elapsed * 1000))
```

```
Fibo(  1) =        1, took      0.000 milli-seconds
Fibo(  2) =        1, took      0.000 milli-seconds
Fibo(  3) =        2, took      0.000 milli-seconds
Fibo(  4) =        3, took      0.000 milli-seconds
Fibo(  5) =        5, took      0.000 milli-seconds
Fibo(  6) =        8, took      0.000 milli-seconds
Fibo(  7) =       13, took      0.000 milli-seconds
Fibo(  8) =       21, took      0.000 milli-seconds
Fibo(  9) =       34, took      0.000 milli-seconds
Fibo( 10) =       55, took      0.000 milli-seconds
Fibo( 11) =       89, took      0.000 milli-seconds
Fibo( 12) =      144, took      0.000 milli-seconds
Fibo( 13) =      233, took      0.000 milli-seconds
Fibo( 14) =      377, took      0.000 milli-seconds
Fibo( 15) =      610, took     10.000 milli-seconds
Fibo( 16) =      987, took      0.000 milli-seconds
Fibo( 17) =     1597, took      0.000 milli-seconds
Fibo( 18) =     2584, took      0.000 milli-seconds
Fibo( 19) =     4181, took      0.000 milli-seconds
Fibo( 20) =     6765, took      0.000 milli-seconds
Fibo( 21) =    10946, took     10.000 milli-seconds
Fibo( 22) =    17711, took      0.000 milli-seconds
Fibo( 23) =    28657, took     10.000 milli-seconds
Fibo( 24) =    46368, took     20.000 milli-seconds
Fibo( 25) =    75025, took     40.000 milli-seconds
Fibo( 26) =   121393, took     50.000 milli-seconds
Fibo( 27) =   196418, took     90.000 milli-seconds
Fibo( 28) =   317811, took    140.000 milli-seconds
Fibo( 29) =   514229, took    240.000 milli-seconds
Fibo( 30) =   832040, took    370.000 milli-seconds
Fibo( 31) =  1346269, took    620.001 milli-seconds
Fibo( 32) =  2178309, took    990.001 milli-seconds
Fibo( 33) =  3524578, took   1590.002 milli-seconds
Fibo( 34) =  5702887, took   2580.004 milli-seconds
```

함수 수식자 (Function Decorator)

◆ 함수 수식자

- 함수의 실행 직전과 직후에 미리 설정되어 있는 기능을 수행하게 하는 기능 구현
- 다른 함수를 반환
- 함수 이름 앞에 '@' 를 붙여 함수 수식자로 설정
- 함수 수식자는 사용되기 이전에 미리 설정되어 있어야 함
- 함수 수식자는 1급 함수를 사용함

함수 수식자 (function decorator) 사용 예

◆ entryExit()

```
# First class function - decorator

def entry_exit(func):
    def wrapper():
        print("Entering {}".format(func.__name__))
        func()
        print("Returned from {}".format(func.__name__))
    return wrapper

@entry_exit
def testFunc():
    print("Processing at inside of testFunc() ....")

#-------------------
testFunc()
```

```
Entering testFunc
Processing at inside of testFunc() ....
Returned from testFunc
```

재귀함수
(Recursive function)

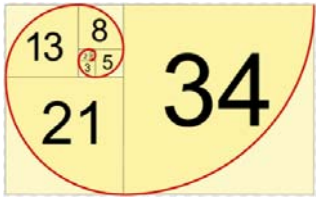
재귀함수 (Recursive Function)

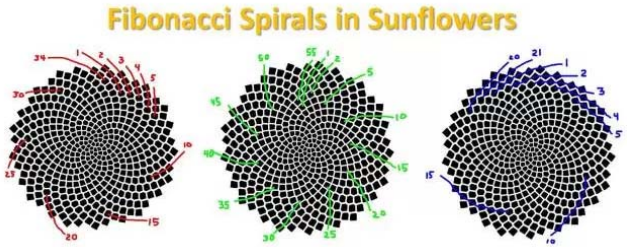
◆ 재귀함수 (recursive function)

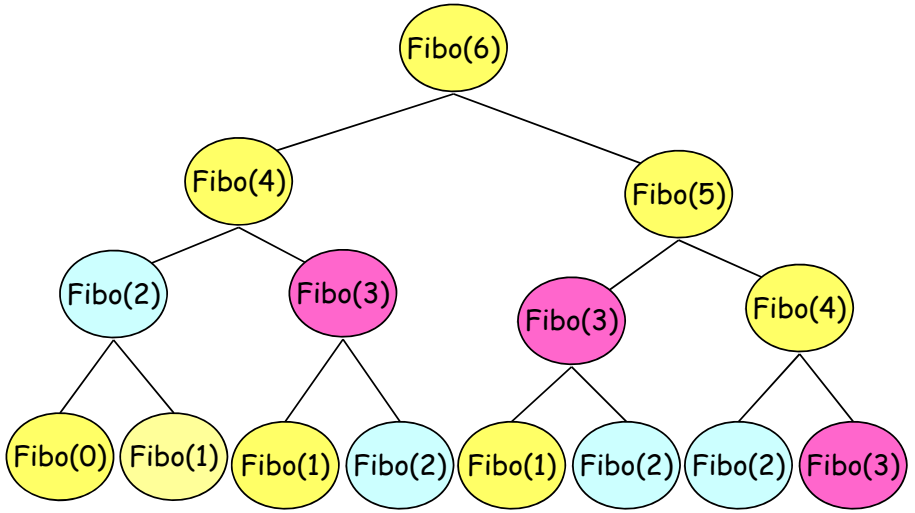
- 함수가 실행 중에 자신을 호출하는 함수 구조
- 분할 및 정복 (divide and conquer) 구조의 알고리즘 구현에 사용
- 대표적인 재귀함수 구조 : 피보나치 수열 (Fibonacci Series), 병합 정렬 (merge sort), 퀵 정렬 (quick sort)

◆ 피보나치 수열 (Fibonacci Series)

$F_0 = 0$
$F_1 = 1$
$F_n = F_{n-1} + F_{n-2}$







재귀함수 구조의 피보나치 수열 계산

◆ Fibonacci Series

```
# Dynamic Programming of Fibonacci with memoization

memo = dict()
def dynFibo(n):
    if n in memo:
        return memo[n]
    elif n <= 1:
        memo[n] = n
        return n
    else:
        fibo_n = dynFibo(n-1) + dynFibo(n-2)
        memo[n] = fibo_n
        return fibo_n


N = int(input("inpt N for fibo : "))
for n in range(0,N+1,5):
    print("dynFibo({:3d}) : {:25}".format(n, dynFibo(n)))
```

```
inpt N for fibo : 100
dynFibo(  0) :                         0
dynFibo(  5) :                         5
dynFibo( 10) :                        55
dynFibo( 15) :                       610
dynFibo( 20) :                      6765
dynFibo( 25) :                     75025
dynFibo( 30) :                    832040
dynFibo( 35) :                   9227465
dynFibo( 40) :                 102334155
dynFibo( 45) :                1134903170
dynFibo( 50) :               12586269025
dynFibo( 55) :              139583862445
dynFibo( 60) :             1548008755920
dynFibo( 65) :            17167680177565
dynFibo( 70) :           190392490709135
dynFibo( 75) :          2111485077978050
dynFibo( 80) :         23416728348467685
dynFibo( 85) :        259695496911122585
dynFibo( 90) :       2880067194370816120
dynFibo( 95) :      31940434634990099905
dynFibo(100) :     354224848179261915075
```

재귀함수 구조 - 정수 (int)의 16진수 문자열 변환

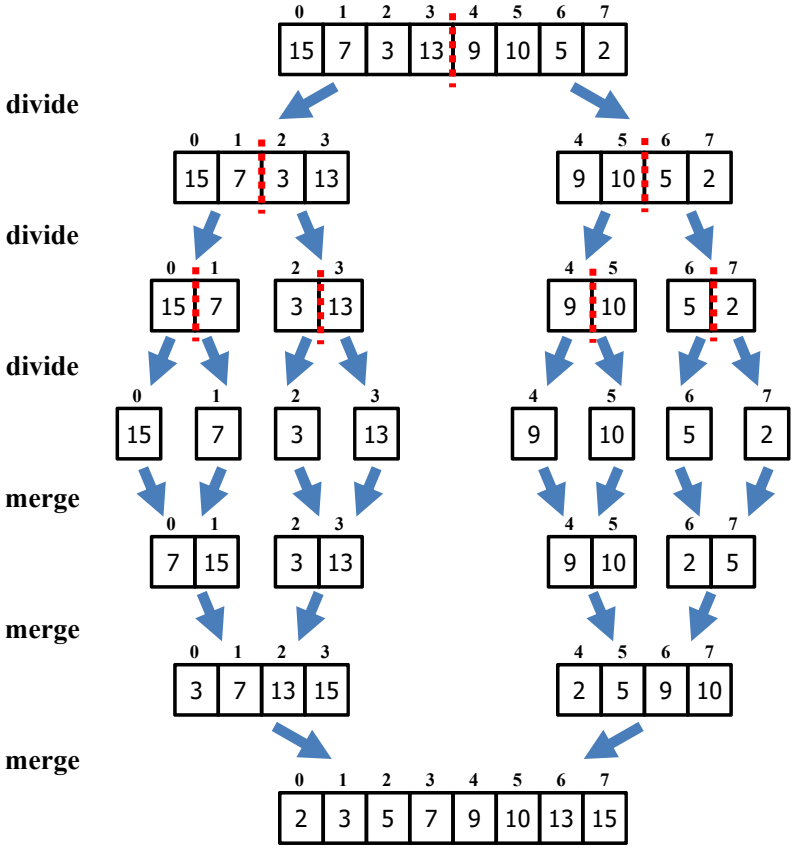
```
# Converting int into hexadecimal string

hexDeciChar = "0123456789ABCDEF"
def xtoa(hex_str, n, level):
    for i in range(level):
        print("  ", end='') # make indentation
    print("xtoa ({}, {:3}) :: ".format(hex_str, n))
    if n >= 16:
        hex_str = xtoa(hex_str, n//16, level+1)
    n = n % 16
    hex_str += str(hexDeciChar[n])
    for i in range(level):
        print("  ", end='')
    print("=> hex_str = ", hex_str)
    return hex_str

n = 0xABCD1234
level = 0
hexStr_n = "0x"
hexStr_n = xtoa(hexStr_n, n, level)
print("decimal ({}) => {}".format(n, hexStr_n))
```

```
xtoa (0x, 2882343476) ::
  xtoa (0x, 180146467) ::
    xtoa (0x, 11259154) ::
      xtoa (0x, 703697) ::
        xtoa (0x, 43981) ::
          xtoa (0x, 2748) ::
            xtoa (0x, 171) ::
              xtoa (0x,  10) ::
              => hex_str =  0xA
            => hex_str =  0xAB
          => hex_str =  0xABC
        => hex_str =  0xABCD
      => hex_str =  0xABCD1
    => hex_str =  0xABCD12
  => hex_str =  0xABCD123
=> hex_str =  0xABCD1234
decimal (2882343476) => 0xABCD1234
```

재귀함수 구조의 병합 정렬 (Merge Sort)

◆ 병합정렬 알고리즘

- 전체 알고리즘이 분할 (divide)과 병합(merge) 단계로 구성됨
- 분할 단계에서는 주어진 정렬 구간을 반복적으로 ½로 나누고, 최종적으로 2개씩의 원소가 남으면 이를 정렬
- 병합 단계에서는 정렬된 부분 구간들을 병합시키면서 정렬 기능을 수행
- 병합단계에서는 이미 부분 구간들이 정렬되어 있으므로 병합정렬이 신속하게 처리될 수 있음



Merge Sort with Recursive Function Calls

```
# merge sort
def merge(L_left, L_right):
    L_res = []
    i, j = 0, 0
    len_left, len_right = len(L_left), len(L_right)
    while i < len_left and j < len_right:
        if L_left[i] < L_right[j]:
            L_res.append(L_left[i])
            i += 1
        else:
            L_res.append(L_right[j])
            j += 1
    while (i < len_left):
        L_res.append(L_left[i])
        i += 1
    while (j < len_right):
        L_res.append(L_right[j])
        j += 1
    return L_res

def mergeSort(L): # merge_sort in increasing order
    if len(L) < 2:
        return L[:]
    else:
        middle = len(L) // 2
        L_left = mergeSort(L[:middle])
        L_right = mergeSort(L[middle:])
        return merge(L_left, L_right)
```

제네레이터 함수, 제네레이터 수식
(Generator function, Generator expression)

제네레이터와 코루틴

◆ 제네레이터 (Generator)

- 제네레이터는 데이터를 순차적으로 생성하여 전달
- 제네레이터는 제네레이터 함수 (generator function) 또는 제네레이터 수식 (generator expression)으로 구현
- 제네레이터 함수는 return 대신 yield를 사용하여 (중간) 결과값을 전달하거나, 전달 받음

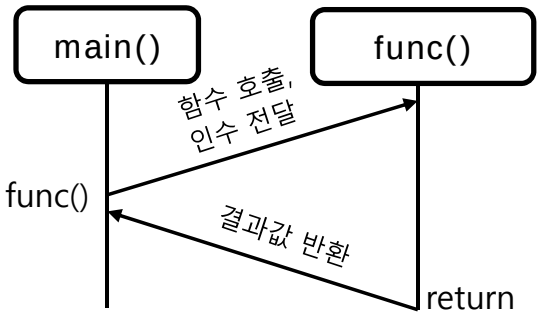
◆ 코루틴 (coroutine)

- 코루틴은 일반 함수와는 달리 여러 군데의 함수 진입점 (entry points)이 있으며, 실행 도중 잠시 정지하였다가 다시 실행을 재개 할 수 있음
- 코루틴은 대화형으로 실행할 수 있음
- 코루틴은 제네레이터 함수 또는 async를 사용하여 구현됨

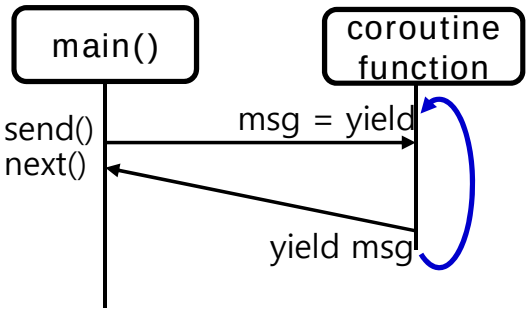
일반적인 서브루틴 호출 (subroutine call) vs. 코루틴 (coroutine)

◆ 일반적인 함수 호출 (subroutine)과 코루틴의 비교

- 서브루틴에서는 함수 호출에 대한 결과를 return으로 반환한 후, 함수가 종료됨
- 코루틴에서는 함수의 중간 결과를 yield로 전달한 후, 함수를 종료하지 않고 대기; 대입 연산자 오른쪽에 있는 yield를 사용하여 데이터를 전달 받음
- 코루틴을 호출하는 함수는 next()를 사용하여 데이터를 전달받으며, send()를 사용하여 데이터를 전달



(a) Subroutine



(b) Coroutine

제네레이터 함수 – myRange()

◆ Generator Function

```
# Generator function with yield

def myRange(n):
    i = 0
    while i < n:
        yield i #using yield instead of return
        i += 1

for i in myRange(5):
    print("returned value from myRange(n) : {:2d}"\
          .format(i))
```

```
returned value from myRange(n) :  0
returned value from myRange(n) :  1
returned value from myRange(n) :  2
returned value from myRange(n) :  3
returned value from myRange(n) :  4
```

제네레이터 함수 – myRange()

◆ myRange() with next()

```
# Generator function with yield and next

def myRange(n):
    i = 0
    while i < n:
        yield i #using yield instead of return
        i += 1
MAX_RANGE = 5
itr = myRange(MAX_RANGE)
for i in myRange(MAX_RANGE+1):
    d = next(itr)
    print("returned value from myRange(n) : {:2d}".format(d))
```

```
returned value from myRange(n) :  0
returned value from myRange(n) :  1
returned value from myRange(n) :  2
returned value from myRange(n) :  3
returned value from myRange(n) :  4
Traceback (most recent call last):
  File "C:\MyPyPackage\TextBook - 2019\ch 6
r function - myRange with yield and  next().
    d = next(itr)
StopIteration
```

제네레이터 함수 – yield를 사용한 데이터 전달

◆ 대입연산자 오른쪽의 yield를 사용한 데이터 수신

```
# Generator function that receives data from outside using (yield)

def echoMsg():
    print("Starting echoMsg() generator ....")
    try:
        while True:
            try:
                msg = yield # data received using (yield)
                print("Received message : ", msg)
            except Exception as e:
                print("Exception : ", e)
    finally:
        print("Generator function is closed now")

genMsg = echoMsg()
next(genMsg)

genMsg.send("First message")
genMsg.throw(TypeError, "Exception message !")
genMsg.send("Last message")
genMsg.close()
```

```
Starting echoMsg() generator ....
Received message :  First message
Exception :  Exception message !
Received message :  Last message
Generator function is closed now
```

Homework 5

Homework 5.1

5.1 튜플 리스트의 응용

- 학생 10명의 정보를 포함하는 튜플 (학생이름, 학과명, 학번, (연, 월, 일), 평균성적)들을 리스트에 포함시킨 후, 파이썬 내장함수 sum(), max(), min(), len()을 사용하여 학생들의 성적 분포 (최대, 최소, 평균)를 파악하여 출력하는 함수 statistics_student_GPA() 함수를 구현하라.
- (실행 예시)

```
# Homework 5.1 student tuple, statistics
import math
def statistics_student_GPA(L_GPAs):
    . . . . # 직접 구현

#  (Application) student tuple, statistics
    students = [
        . . . . # 학생정보 튜플
        ]
    print("students = ")
    for st in students:
        print(st)
    L_GPAs = []
    . . . . # L_GPAs 구성은 직접 구현
    statistics_student_GPA(L_GPAs)
```

```
students =
('Kim', 'EE', 12345, (2000, 12, 25), 4.3)
('Lee', 'ME', 11234, (2019, 9, 1), 4.2)
('Park', 'ICE', 10234, (2019, 3, 1), 3.5)
('Hong', 'CE', 13123, (2021, 1, 1), 4.1)
('Yoon', 'ICE', 11321, (2001, 8, 15), 4.4)
('ABC', 'ICE', 12321, (2000, 7, 31), 3.5)
('DEF', 'M', 24680, (2018, 8, 31), 4.2)
('Choi', 'A', 32165, (2020, 3, 25), 4.1)
('Hwang', 'K', 43210, (2020, 12, 25), 3.9)
('Jung', 'P', 56789, (2021, 1, 1), 4.1)
statistics_student_GPA ::
 - L_GPAs =  [4.3, 4.2, 3.5, 4.1, 4.4, 3.5, 4.2, 4.1, 3.9, 4.1]
 - num_students = 10
 - Statistics of GPAs : Max (4.4), Min (3.5), Avg (4.029999999999999),
    Var(0.0861), Std_dev(0.2934280150224242)
```

Homework 5.2

5.2 정수형 난수 리스트의 정렬 및 경과시간 측정

(1) 중복되지 않는 정수형 (int) 난수를 지정된 개수 (예: 10,000 ~ 1,000,000) 만큼 생성하는 함수 genBigRandList(L, n)을 구현하라. 이 함수에는 리스트 L과 리스트에 포함될 중복되지 않는 난수의 개수 n이 전달된다.

(2) 주어진 리스트의 첫 부분 2줄 (한 줄에 10개씩)과 마지막 2줄을 출력하는 함수 printListSample(L, per_line, sample_lines)를 구현하라.

(3) 주어진 리스트의 원소들을 오름차순으로 병합정렬 구조로 정렬하는 함수 mergeSort(L) 함수를 구현하라.

(4) 표준입력장치로 부터 100,000 이상 크기의 정수형 난수 개수를 입력 받은 후, genBigRandList() 함수를 사용하여 중복되지 않는 정수형 난수 리스트를 생성하고, 이를 printListSample() 함수를 사용하여 출력하라. 이 정수형 난수 리스트를 mergeSort() 함수를 사용하여 정렬하고, 정렬된 상태를 printListSample() 함수를 사용하여 출력하라.

(5) 정렬에서 걸린 시간을 time 모듈의 time() 메소드를 사용하여 측정하고, 출력하라.

(파이썬 프로그램 구성 예)

```
# Homework 5.2  (1)
import random, time

def genBigRandList(L, n):
   . . . . # 이 부분은 직접 구현
def printListSample(L, per_line = 10, sample_lines = 2):
   . . . . # 이 부분은 직접 구현
def _merge(L_left, L_right):
 . . . . # 이 부분은 직접 구현
def mergeSort(L):
 . . . . # 이 부분은 직접 구현
```

(응용 프로그램 및 실행 예시)

```
# Homework 5.2  (2)

# (Application) Performance test of merge sorting
while True:
    print("\nPerformance test of merge sorting algorithm")
    size = int(input("Input size of random list L (-1 to quit) = "))
    if size == -1:
        break
    L = []
    genBigRandList(L, size)

    # testing MergeSorting
    print("\nBefore mergeSort of L :")
    printListSample(L, 10, 2)
    t1 = time.time()
    sorted_L = mergeSort(L)
    t2 = time.time()
    print("After mergeSort of L :")
    printListSample(sorted_L, 10, 2)
    time_elapsed = t2 - t1
    print("Merge sorting took {} sec".format(time_elapsed))
```

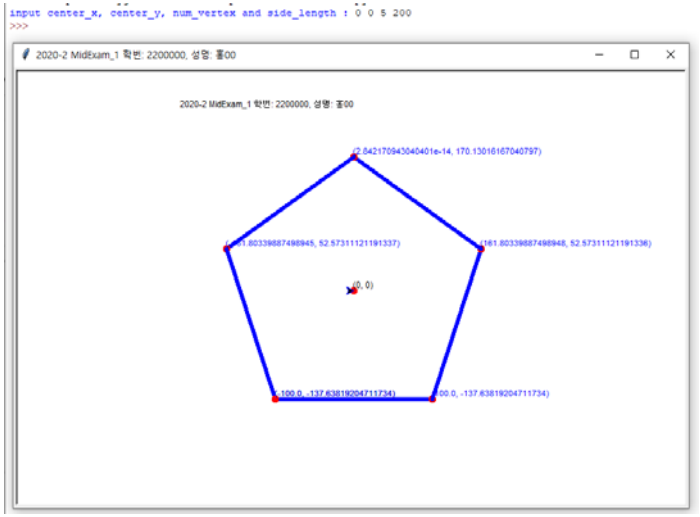
```
Performance test of merge sorting algorithm
Input size of random list L (-1 to quit) = 500

Before mergeSort of L :
     42     463     459     489     280     487     224     116     474     451
    356     304     399      17     371      84     334     447      66      52
     ......
    421     325     495     455      96     337     481     476     273     460
    435     252     348     442     207       1     430     278       4     261
After mergeSort of L :
      0       1       2       3       4       5       6       7       8       9
     10      11      12      13      14      15      16      17      18      19
     ......
    480     481     482     483     484     485     486     487     488     489
    490     491     492     493     494     495     496     497     498     499
Merge sorting took 0.0010266304016113281 sec

Performance test of merge sorting algorithm
Input size of random list L (-1 to quit) = 1000000

Before mergeSort of L :
 112545  578618  905723  129529  871202  734882  477491  301121  585592  348025
 199663  204752  504846  168448  730870  610181  804459  834841  754475  415706
     ......
 210476  903296  825211  926077  544762  266814  504230  443570  960208  303030
 322089   36568  716943  846200  748795  333868  706688  644337  715524  883863
After mergeSort of L :
      0       1       2       3       4       5       6       7       8       9
     10      11      12      13      14      15      16      17      18      19
     ......
 999980  999981  999982  999983  999984  999985  999986  999987  999988  999989
 999990  999991  999992  999993  999994  999995  999996  999997  999998  999999
Merge sorting took 2.7038002014160156 sec

Performance test of merge sorting algorithm
Input size of random list L (-1 to quit) = 5000000

Before mergeSort of L :
4998384 4695404 1649919 1582009 3553797  410353 2588769 1222399 2824561 1033069
2159073 3056034  657632 2323814  190032 2098497 2330273  245257 1245513  446468
     ......
2952214 1622225 1961559  597029  283801 3336693 1067215  829374 4893188  833137
 395994 4850244  428673 1730010 3878742  773035 4116693 1466934 2380217 4045822
After mergeSort of L :
      0       1       2       3       4       5       6       7       8       9
     10      11      12      13      14      15      16      17      18      19
     ......
4999980 4999981 4999982 4999983 4999984 4999985 4999986 4999987 4999988 4999989
4999990 4999991 4999992 4999993 4999994 4999995 4999996 4999997 4999998 4999999
Merge sorting took 16.690403699874878 sec

Performance test of merge sorting algorithm
Input size of random list L (-1 to quit) = -1
```

Homework 5.3

5.3 터틀 그래픽 - drawPolygon()

- 파이썬 터틀 그래픽 기능을 사용하여 지정된 위치에 지정된 크기 (한 변의 길이)의 다각형을 그리는 drawPolygon() 함수를 구현하라. 이 함수에 전달되는 인수로는 터틀 객체, 다각형의 중심 위치 (x좌표, y좌표), 다각형의 꼭지점 수, 다각형 한 변의 길이가 주어지도록 하라.
- 응용 프로그램 부분에서는 다각형의 중심 좌표 (x, y), 꼭지점 수, 한 변의 길이를 입력 받고, 입력된 중심 위치에 붉은색 점 (dot)을 표시하고 좌표값을 출력하라.
- 다각형의 각 꼭지점에 붉은색 점 (dot)을 표시하고, 각 꼭지점의 좌표값을 출력하라.
- (실행 예시)

```
# Homework 5.3 Turtle graphic - Polygon with given center position

import turtle, math

def drawPolygon(t, center_x, center_y, line_length, num_vertices):
    . . . .  // 직접 구현

# Application of drawPolygon
. . . . // turtle 기본 정보 설정
center_x, center_y, num_vertices, line_length  =\
 map(int, input("input center_x, center_y, num_vertices, and line_length : ").split(' '))
center_pos = (center_x, center_y)
line_width = 3
t.up(); t.goto(center_pos); t.down()
t.dot(10, "red"); t.write(center_pos)
t.width(line_width)
t.pencolor("blue")

drawPolygon(t, center_x, center_y, line_length, num_vertices)
```



Homework 5.4

5.4 재귀함수, 동적 프로그래밍

- Fibonacci 수열을 신속하게 계산할 수 있도록 memo 기능을 포함하는 재귀함수 dynFibo(n)를 구성하고, 시작 (start), 끝 (end), 간격 (stride)의 정수를 입력 받은 후 해당 순서의 피보나치 수열을 출력하는 프로그램을 작성하라. 각 단계별 피보나치 수열 계산에서 이미 메모에 포함된 피보나치 수열 값은 별도로 계산하지 않고 메모의 내용을 사용하며, 이전에 계산된 적이 없은 피보나치 수열 값은 재귀함수를 사용하여 계산한 후, 이를 메모에 기록하여 두고, 계산된 결과값을 반환하도록 하라.
  이 프로그램에서는 dynFibo(n)에서 걸린 경과 시간을 time() 함수를 사용하여 측정한 후, 계산된 피보나치 수열과 함께 출력하도록 하라. start = 50, end = 100, stride = 5로 설정하여 실행 결과를 출력하라.

- (프로그램 구성 및 실행 예시)

```
# Homework 5.4 Dynamic Programming for Fibonacci Series
import time

memo = dict()
def dynFibo(n): # recursive calculation with memoization
  . . . . # 직접 구현 할 것

# (Application)
(start, stop, step) = tuple(map(int, \
    input("input start, stop, step of Fibonacci series : ").split(' ')))

for n in range(start, stop+1, step):
    . . . . # dynFibo() 호출 및 경과시간 측정
    print("dynFibo({:3d}) = {:25d}, took {:10.2f}[micro_sec]".format(n, fibo_n, t_elapse_us))
```

```
input start, stop, step of Fibonacci series : 50 100 5
dynFibo( 50) =                12586269025, took       0.00[micro_sec]
dynFibo( 55) =               139583862445, took       0.00[micro_sec]
dynFibo( 60) =              1548008755920, took       0.00[micro_sec]
dynFibo( 65) =             17167680177565, took       0.00[micro_sec]
dynFibo( 70) =            190392490709135, took       0.00[micro_sec]
dynFibo( 75) =           2111485077978050, took       0.00[micro_sec]
dynFibo( 80) =          23416728348467685, took       0.00[micro_sec]
dynFibo( 85) =         259695496911122585, took       0.00[micro_sec]
dynFibo( 90) =        2880067194370816120, took       0.00[micro_sec]
dynFibo( 95) =       31940434634990099905, took       0.00[micro_sec]
dynFibo(100) =      354224848179261915075, took       0.00[micro_sec]
```

Homework 5.5

5.5 행렬의 출력, 연산 프로그램

(1) 행렬을 표현하는 2차원 리스트 1개를 행렬의 이름 (문자열)과 함께 전달받아 출력하는 함수 printMtrx(name, M)을 작성하라.

(2) 2차원 리스트 2개를 전달받아 행렬의 덧셈을 계산하여 결과를 반환하는 함수 mtrxAdd(A, B), 뺄셈을 계산하여 결과를 반환하는 함수 mtrxSub(A, B), 그리고 곱셈을 계산하여 결과를 반환하는 함수 mtrxMul(A, B)를 작성하라.

(3) 다음 기능 시험 프로그램을 사용하여 결과를 출력하라.

(실행 예시)

```
# Homework 5.5
def printMtrx(M):
  . . . # 직접 구현
def mtrxAdd(M1, M2):
 . . . # 직접 구현
def mtrxSub(M1, M2):
 . . . # 직접 구현
def mtrxMul(M1, M2):
 . . . # 직접 구현

#-------------------------
def main():
    A = [[1,2,3,4], [5,6,7,8], [9,10,0,1]]
    B = [[1,0,0,0], [0,1,0,0], [0,0,1,1]]
    C = [[1,0,0], [0,1,0], [0,0,1], [0,0,0]]
    print("A = "); printMtrx(A)
    print("B = "); printMtrx(B)
    print("C = "); printMtrx(C)

    D = mtrxAdd(A, B)
    print("A + B = "); printMtrx(D)

    E = mtrxSub(A, B)
    print("A - B = "); printMtrx(E)

    F = mtrxMul(A, C)
    print("A * B = "); printMtrx(F)

if __name__ == "__main__":
    main()
```

```
A =
  1  2  3  4
  5  6  7  8
  9 10  0  1
B =
  1  0  0  0
  0  1  0  0
  0  0  1  1
C =
  1  0  0
  0  1  0
  0  0  1
  0  0  0
```

```
A + B =
  2  2  3  4
  5  7  7  8
  9 10  1  2
A - B =
  0  2  3  4
  5  5  7  8
  9 10 -1  0
A * B =
  1  2  3
  5  6  7
  9 10  0
```