PYTHON 3주차
FUNCTIONS (함수)

서울대학교 전기정보공학부
명예교수 성원용

아래 자료와 인터넷 참고 편집
MIT 6.0001 Introduction to Computer Science and Programming in Python

TODAY

- **structuring programs and hiding details**
- functions
- specifications
- keywords: `return` vs `print`
- **scope**

GOOD PROGRAMMING

- more code not necessarily a good thing
- measure good programmers by the amount of functionality
- introduce **functions**
- mechanism to achieve **decomposition** and **abstraction**

DECOMPOSITION

- in programming, divide code into **modules**
  - are **self-contained**
  - used to **break up** code
  - intended to be **reusable**
  - keep code **organized**
  - **keep code coherent**
- this lecture, achieve decomposition with **functions**
- tomorrow, achieve decomposition with **classes**

ABSTRACTION

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

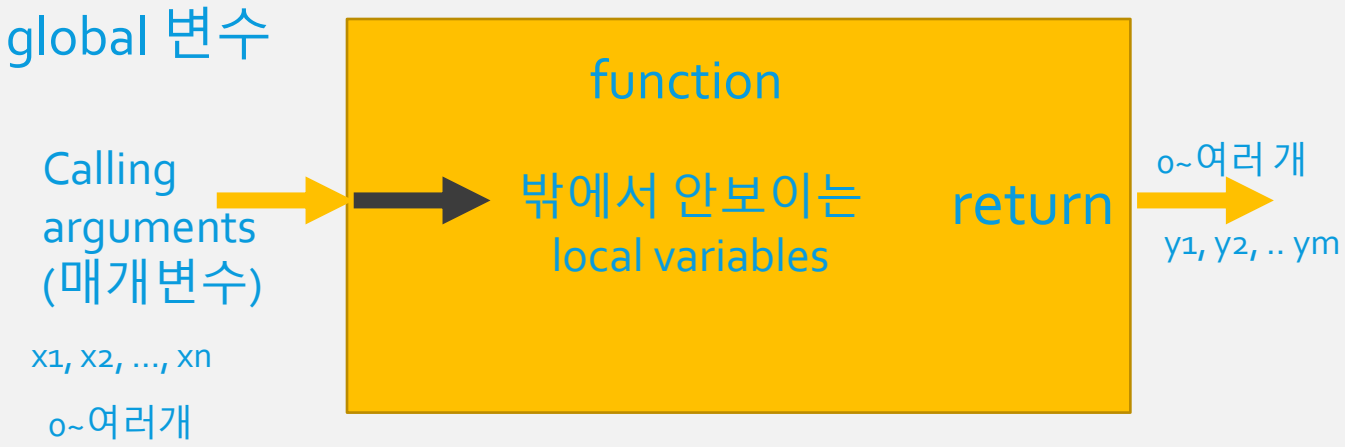
- In programming, think of a piece of code as a **black box**
  - cannot see details
  - do not need to see details
  - do not want to see details
  - hide tedious coding details
- achieve abstraction with **function specifications** or
- **docstrings**

FUNCTIONS

- write reusable pieces/chunks of code, called **functions**
- functions are not run in a program until they are "**called**" or "**invoked**" in a program
- function characteristics:
  - has a **name**
  - has **parameters** (0 or more) (arguments)
  - has a **docstring** (optional but recommended)
  - has a **body**
  - **returns** something

Function

y1, y2, .. ym = f(x1, x2, …, xn)



function은 불리워져서 실행되는 동안만
변수 등의 공간을 차지한다. Return이 되면
싹 없어진다. (function 은 object 이기 때문에 가지는 attribute 는 있다)

Function의 동작은 일반적으로 documentation을 보고 아는 것이지
내부 코드를 공부해서 알지 않는다. 어떻게 무엇으로 짜든 관여않는다.

HOW TO WRITE AND CALL/INVOKE A FUNCTION

keyword

name

parameters or arguments

specification, docstring

```
def is_even( i ):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

body

later in the code, you call the function using its name and values for parameters

```
is_even(3)
```

IN THE FUNCTION BODY

```
def is_even( i ):
    """
    Input: i, a positive int
    Returns True if i is even, otherwise False
    """
    print("inside is_even")
    return i%2 == 0
```

run some commands

keyword

expression to evaluate and return

함수 정의하기(2) : 인자(1)

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- 인자는 쉼표로 구분하며, 별도로 인자='default value' 식으로 기본값을 지정해 주지 않은 모든 인자는 함수를 사용할 때 입력해주어야 한다.

- 함수를 정의할 때 default값을 지정 해 놓은 인자는 사용할 때 생략이 가능하며, '인자3'='value'식으로 굳이 넣어줄 수도 있다.

```
def intro(name, age, sex='Male'):
    print('My name is {0}, and age is {1}'.format(name, age))
    if sex == 'Male':
        print('I am Male')
    else:
        print('I am a Female')


print(intro('Chulsoo', 33))
print(intro('Mary', 39, 'Female'))
--------------------
My name is Chulsoo, and age is 33
I am Male
None
My name is Mary, and age is 39
I am a Female
None
```

함수 정의하기(3) : 인자(2)

서울대학교 SEOUL NATIONAL UNIVERSITY

- 순서상의 문제로 def를 할 때 default값을 지정해주는 인자는 그렇지 않은 인자의 앞에 있을 수 없다.
- ex) def hello(a,b='hello',c): 와 같이 쓰면 에러가 난다.
-
- 함수의 인자에서 사용하는 임의의 변수명과 함수 내부에서 정의된 변수는 다른 함수나 본문의 변수명과 상관이 없다. 겹쳐도 상관 없으며 서로 영향을 주지 않는다. 다른 변수를 인자 자리에 넣어도 전혀 영향이 없다.
- def hello(a): 라고 해놓고 다른 함수나 본문에서 a를 사용해도 이 두 함수의 a사이에는 아무런 관계가 없다. 이 함수 내에서만 통용되는 local 변수인 셈. (단, 나중에 배울 class변수나 global변수같이 공유하는 변수를 이용하면 상관이 있다.)

- Multi-line comments
- Or, not quite as intended, you can use a multiline string.
- Since Python will ignore string literals that are not assigned to a variable, you can add a multiline string (triple quotes) in your code, and place your comment inside it:
- Example
- """
  This is a comment
  written in
  more than just one line
  """
  print("Hello, World!")

VARIABLE SCOPE (중요)

- **formal parameter** gets bound to the value of **actual parameter** when function is called

Parameter는 함수 혹은 메서드 정의에서 나열되는 변수 명입니다. 반면 Argument는 함수 혹은 메서드를 호출할 때, 전달 혹은 입력되는 실제 값입니다.

- new **scope/frame/environment** created when enter a function
- **scope** is mapping of names to objects

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```

formal parameter 매개변수

actual parameter 인수

Function definition

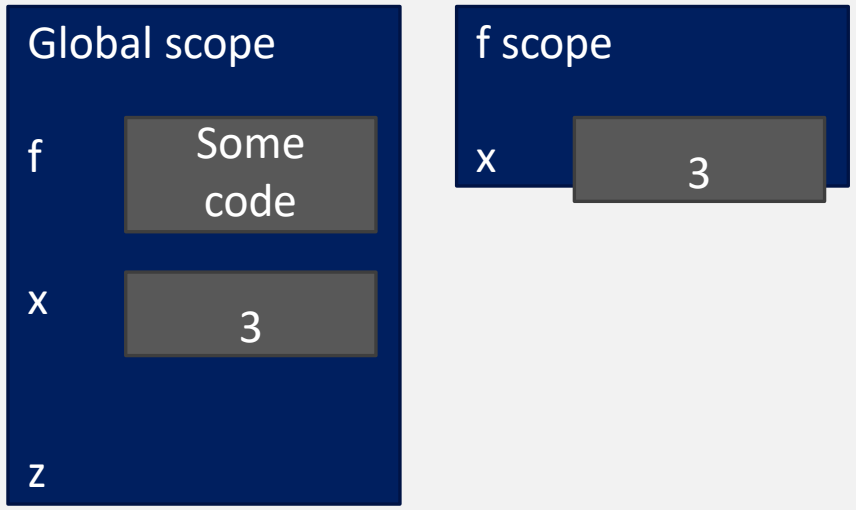
Main program code
* initializes a variable x
* makes a function call f(x)
* assigns return of function to variable z

VARIABLE SCOPE

```
y = y + 1
return y
```

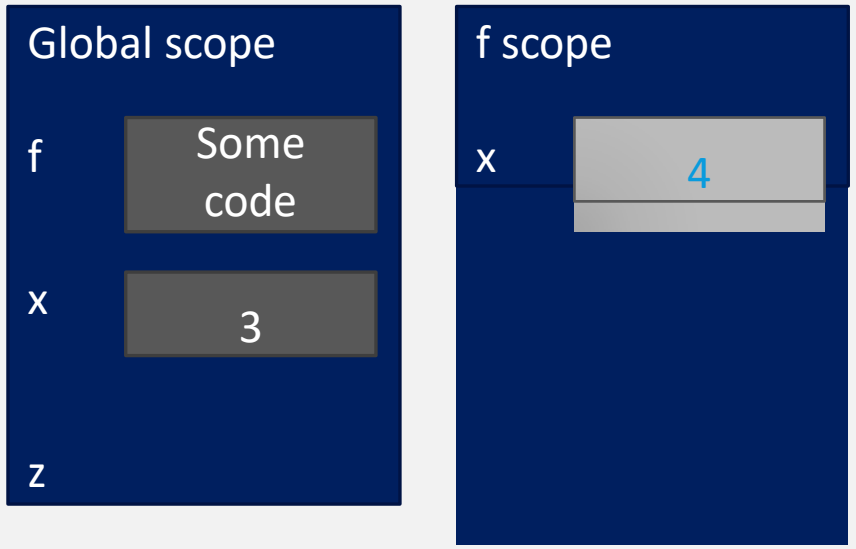
```
x = 3
z = f( x )
print(z)
```

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



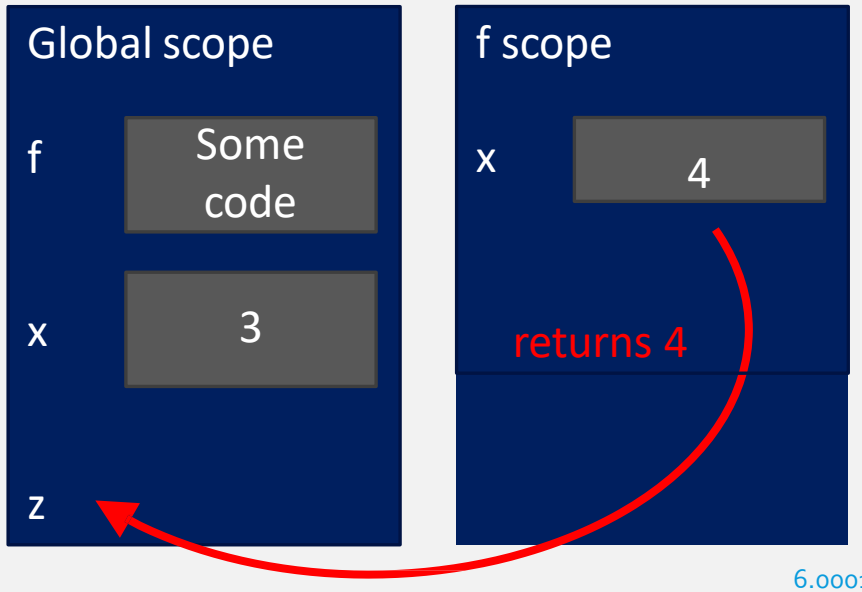
VARIABLE SCOPE

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



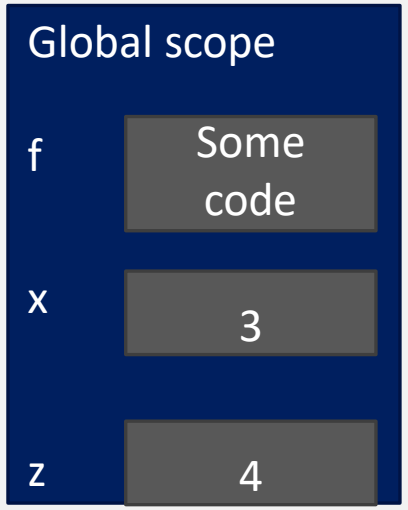
VARIABLE SCOPE

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



VARIABLE SCOPE

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



ONE WARNING IF NO `RETURN` TATEMENT

```
def is_even( i ):
    """
    Input: i, a positive int
    Does not return anything
    """
    i%2 == 0
```

without a return statement

- Python returns the value **None, if no `return` given**
- represents the absence of a value

FUNCTIONS AS ARGUMENTS

- arguments can take on any type, even functions

```
def func_a():
    print('inside func_a')

def func_b(y):
    print('inside func_b')
    return y

def func_c(z):
    print('inside func_c')
    return z()

print(func_a())
print(5 + func_b(2))
print(func_c(func_a))
```

```
inside func_a
None
inside func_b
7
inside func_c
inside func_a
None
```

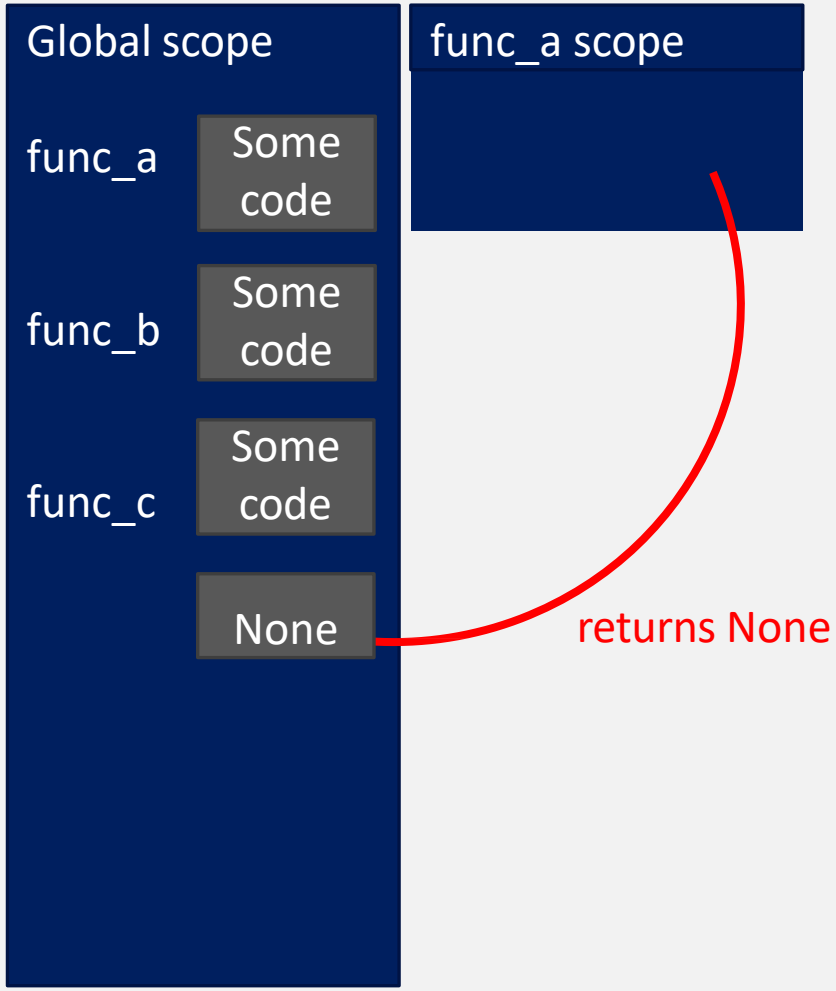
call `func_a`, takes no parameters

call `func_b`, takes one parameter

call `func_c`, takes one parameter,

FUNCTIONS AS ARGUMENTS

```
def func_a():
    print 'inside func_a'
def func_b(y):
    print 'inside func_b'
    return y
def func_c(z):
    print 'inside func_c'
    return z()
print func_a()
print 5 + func_b(2)
print func_c(func_a)
```



FUNC_A() 와 FUNC_A

- def func_a():
- return "in func_a"

- print(func_a())
- print(func_a)

- in func_a

- <function func_a at 0x7ff7455c3f28>

```
def func_a():
    return "in func_a"

print(func_a())
print(func_a)
print(func_a.__name__)

x = [3, 'a', 3.0]
print(x.__class__.__name__)
print(type(x))
```

```
in func_a
<function func_a at
0x0000025A12656E58>
func_a
list
<class 'list'>
```

FUNCTION IS AN OBJECT

서울대학교 SEOUL NATIONAL UNIVERSITY

- Yes, python functions are full objects. They can have attributes and methods like objects. The functions can have data variables and even functions written inside of them.

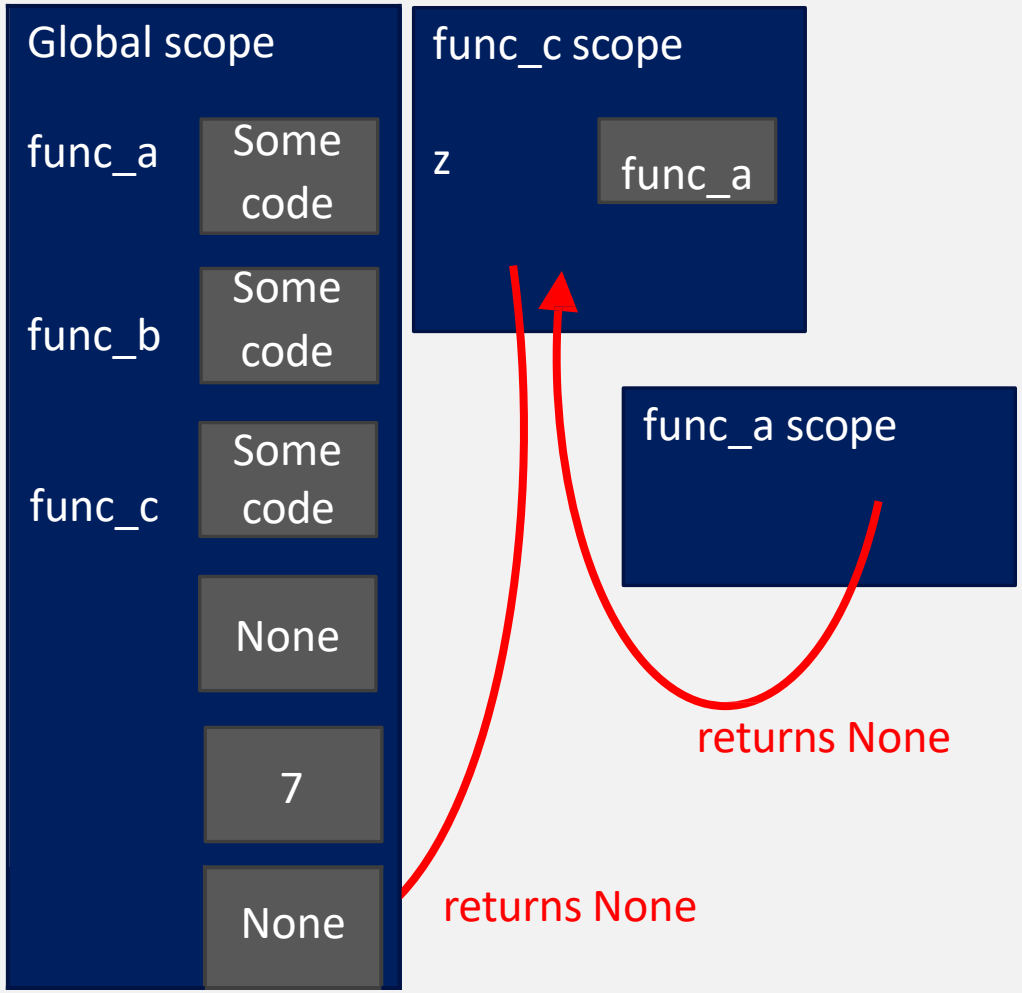
```
def foo():
    print ("inside function foo")
    print(foo.score1)
    foo.score2 = 10
    print(foo.score2)

foo.score1 = 20
foo()
print(type(foo))
print(foo.score1, foo.score2)

foo.score1 += 1
foo.score2 += 1
print(foo.score1, foo.score2)
```

```
inside function foo
20
10
<class 'function'>
20 10
21 11
```

FUNCTIONS AS ARGUMENTS

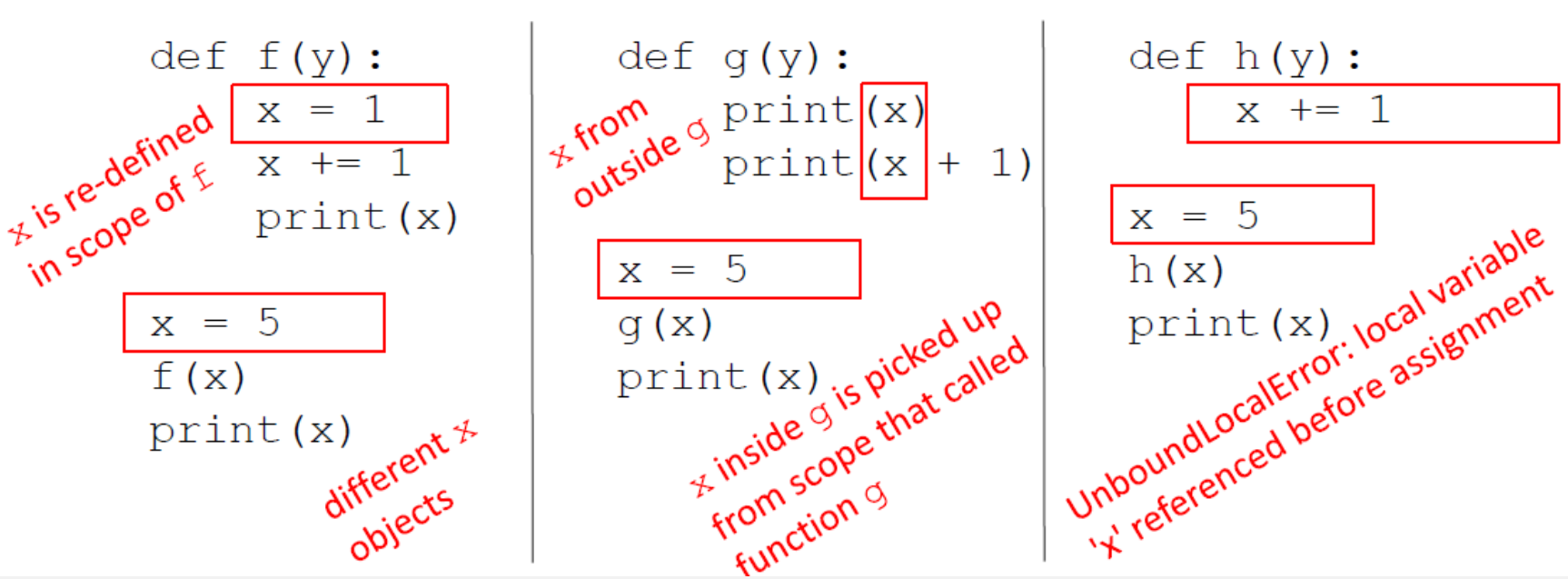
```
def func_a():
    print('inside func_a')

def func_b(y):
    print('inside func_b')
    return y

def func_c(z):
    print('inside func_c')
    return z()

print(func_a())
print(5+func_b(2))
print(func_c(func_a))
```

```
inside func_a
None
inside func_b
7
inside func_c
inside func_a
None
```



SCOPE EXAMPLE (중요)

- inside a function, **can access** a variable defined outside
- inside a function, **cannot modify** a variable defined outside -- can using **global variables**, but frowned upon



SCOPE EXAMPLE

- inside a function, **can access** a variable defined outside
- inside a function, **cannot modify** a variable defined outside -- can usir **global variables**, but frowned upon

```
def f(y):
    x = 1
    x += 1
    print(x)

x = 5
f(x)
print(x)
```

```
def g(y):
    print(x)


x = 5
g(x)
print(x)
```

```
def h(y):
    x += 1


x = 5
h(x)
print(x)
```

x from global/main program scope

```
def f(y):
    x =1
    x +=1
    print(x)   #2
x=5
f(x)
print(x)    #5
```

```
def g(y):
    y += 1
    print("y=", y)
    print("x in function=", x)
x = 5
g(x)
print(x)
```

```
y= 6
x in function= 5
5
```

```
def h(y):
    x += 1

x = 5
h(x)
print(x)
```

```
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-63-a87ca0542697> in <module>
      3
      4 x = 5
----> 5 h(x)
      6 print(x)

<ipython-input-63-a87ca0542697> in h(y)
      1 def h(y):
----> 2     x += 1
      3
      4 x = 5
      5 h(x)

UnboundLocalError: local variable 'x' referenced before assignment
```

```
def h(y):
    y=x+1
    print(y, "in function") #6
x = 5
h(x)
print(x)  #5
```

```
def f():
    def g():
        def h():
            print('x in h =', x)

        print('x in g=', x)
        h()

    print('x in f',x)
    g()

x=2
f()
print('complete')
```

```
x in f 2
x in g= 2
x in h = 2
complete
```

2 5

```
def f(y):
    x = 1
    x += 1
    print(x)

x=5
f(x)
print(x)
```

2
5

```
def g(y):
    y=y+1
    print("y",y)
    print("x",x)

x=5
g(x)
print("x",x)
```

y 6
x 5
x 5

```
def h(y):
   x +=1
x=5
h(x)
print(x)
```

*매개변수,
입력값이 여러 개이고 정해지지 않았을 때

```
def add_many_input(*things_to_add):
    sum = 0
    for i in things_to_add:
        if type(i) == int:
            sum += i
    return sum


print(add_many_input(3, 4, 5, 'some', 7, 8, 9))  #36
```

Yes. You can use *args as a non-keyword argument. You will then be able to pass any number of arguments.

```
def manyArgs(*arg):
  print "I was called with", len(arg),
"arguments:", arg

>>> manyArgs(1)
I was called with 1 arguments: (1,)
>>> manyArgs(1, 2, 3)
I was called with 3 arguments: (1, 2, 3)
```

GLOBAL VARIABLES

- In Python, a variable declared outside of the function or in global scope is known as a global variable. This means that a global variable can be accessed inside or outside of the function.
- Function 의 내부에서 global variable을 조작하고 싶으면 global 로 지정
- (**Function 내부에서 global variable 을 바꾸는 것은 안좋은 programming style**)

```
x = "global "

def foo():
    global x
    y = "local"
    x = x * 2
    print(x)   #global global
    print(y)   #local

foo()
print(x)      #global global
```

- 전역 이름공간에 정의되어, 프로그램 어디서든 부를 수 있는 이름을 **전역변수(global variable)**라고 한다. 함수 밖에서 변수를 정의하면 전역변수가 된다. 반면에 지역 이름공간에 정의되어, 그 문맥 속에서만 부를 수 있는 이름을 **지역변수(local variable)**라고 한다. 모든 함수는 자신만의 지역 이름공간을 가지며, 함수 속에서 작성한 변수는 그 함수의 지역변수가 된다.
- 3.4.1 지역변수는 함수만의 것
- 함수의 지역변수는 함수가 실행되는 동안에만 존재한다. 각 함수가 호출되어 실행될 때 만들어지고, 함수의 실행이 끝나면 모두 삭제된다. 그래서 지역변수는 그 변수가 속한 함수의 밖이나 다른 함수에서는 부를 수 없다. 매개변수도 함수 안에 정의되므로 지역변수다.

https://python.bakyeono.net/chapter-3-4.html

- 함수는 문제를 작은 문제로 나누어 해결하기 위해서 사용한다. 문제를 나누어 해결하려면, 다른 문제를 배제하고 지금 다루는 문제만을 생각할 수 있어야 한다. 함수는 프로그램 전체를 구성하는 일부이지만, 각각의 함수는 자기가 맡은 작은 문제만을 해결할 뿐이다. 함수는 자기 문제에 필요한 데이터만을 조작해야 하고 관련이 없는 프로그램 전체 데이터나 다른 함수의 데이터에 손을 대면 안 된다.
- 그러므로 함수 안에서는 지역변수만을 사용하는 것이 바람직.
- 전역변수는 프로그램 전체에서 공통적으로 사용되고 잘 변하지 않는 데이터를 담는 데만 써야 한다. 전역변수가 수시로 변하고 여러 함수에서 저마다 손을 댄다면 프로그램의 흐름을 파악하기 어렵다. 함수의 입력 통로와 출력 통로를 매개변수와 return 문으로 제한하는 것도 프로그램의 흐름을 파악하기 쉽게 하기 위한 것이다. 함수 안에서 함수 밖의 데이터를 건드리는 것은 물이 흐르는 파이프에 구멍을 뚫는 것과 같다. 구멍이 많으면 그 물이 어디로 흐를지 예측하기 어렵다.

- seconds_per_minute = 60 # 1분은 60초 ❶

- def minutes_to_seconds(minutes):
- """분을 입력받아 같은 시간만큼의 초를 반환한다."""
- seconds = minutes * seconds_per_minute # ❷
- return seconds

- print(minutes_to_seconds(3)) # 화면에 180이 출력된다
- print(seconds) # ❸ 오류! 함수 밖에서 지역변수를 불렀다

특징	전역변수	지역변수
함수 안에서 읽기	가능	가능
함수 안에서 수정	불가(*)	가능
함수 밖에서 읽기	가능	불가
함수 밖에서 수정	가능	불가

*함수안에 지역변수가 있으면 지역변수가 우선된다.

A global **variable** is a **variable** which is accessible in multiple scopes. In **Python**, it is better to use a single **module** to hold all the global **variables** you want to use and whenever you want to use them, just import this **module**, and then you **can** modify that and it **will** be visible in **other modules** that **do** the same.

Create a Global module

```
#global.py
current_value=0
```

Create a Python program file to access global variable

```
#updater.py
import global
def update_value():
  global.current_value = 100
```

Create another Python program to test or not

```
#display_value.py
import global
import updater
updater.update_value()
print(global.current_value)
```

config.py:

```
x = 0
```

mod.py:

```
import config
config.x = 1
```

main.py:

```
import config
import mod
print(config.x)
```

```
current_value = 0
def update_value():
   global current_value
   current_value= 100

update_value()
print(current_value)
```

EXERCISE

example

```
myGlobal = 5
def func1():
    myGlobal = 50
def func2():
    global myGlobal
    myGlobal = 50
def func3():
    print (myGlobal)
func1()
func3()
print("After using Global")
func2()
func3()
```

```
def func_a():
    def func_b():
        print("b", ten)
        return

    print("a", ten)
    func_b()
    return

ten = 10
func_a()
```

```
def func_a():
    def func_b():
        print("b", ten)
        return

    print("a", ten)
    ten = 20
    func_b()
    return

ten = 10
func_a()
```

**UnboundLocalError: local variable 'ten' referenced before assignment

GLOBAL – 지역변수에서 전역변수를 고친다 (사용 자제)

```
num_stamp = 0
# 쿠폰 스탬프가 찍힌 횟수 (전역변수)


def stamp():
    """쿠폰 스탬프가 찍힌 횟수를 증가시키고,
화면에 출력한다."""
    global num_stamp
# ❶ num_stamp는 전역변수다
    num_stamp = num_stamp + 1
# 이제 오류가 발생하지 않는다
    print(num_stamp)


stamp()  # 화면에 1이 출력된다
stamp()  # 화면에 2가 출력된다
```

```
#좋은 코드
num_stamp = 0
# ❶ 쿠폰 스탬프가 찍힌 횟수 (전역변수)


def stamp(num_stamp):
# ❷ 지역변수(매개변수) num_stamp
    """쿠폰 스탬프가 찍힌 횟수를 증가시키고,
화면에 출력한다."""
    num_stamp = num_stamp + 1
    print(num_stamp)
    return num_stamp


num_stamp = stamp(num_stamp)
# ❸ 전역변수에 함수의 반환값을 대입한다
num_stamp = stamp(num_stamp)
```

함수관련 MUTABLE, IMMUTABLE

- Mutable – 입력 parameter로 들어간 변수의 원본값을 함수내에서 바꿀 수 있는 것. List, dictionary, numpy (안 바꾸려면 함수내에서 deep copy 후 사용)
- Immutable – 원본값이 바뀌지 않는 것. 숫자, 문자(string), tuple

```
def mutable_immutable_check(xin, input_list):
    xin += 1
    input_list.append(20)

x=1
test_list = [1, 2, 3]
mutable_immutable_check(x, test_list)
print('x=', x, 'test_list =', test_list)
```

```
x= 1 test_list = [1, 2, 3, 20]
```

```
def mutable_test(xin):
    y=xin
    y[0] = 'a'

test = ['h', 'e', 'l']
mutable_test(test)
print(test)
```

```
['a', 'e', 'l']
```

왜 MUTABLE , IMMUTABLE 구분이 필요할까?

- List, dictionary, numpy array 는 실제적은 응용(toy 예제가 아닌 것)에서는 엄청 사이즈가 큰 경우가 많다. 다른 말로 복사 코스트가 매우 비싸다. 따라서, 가급적 원래 데이터를 변경해가면서 쓰는 것이 유리한 경우가 많다.
- Tuple은 원래 바꾸지 못하는 list 같은 경우이다. Mutable, immutable 의미가 없다.
- String, 숫자, 문자는 사이즈가 작다. 실제 응용에 나오는 것이나 지금 toy example에 나오는 것이나 차이가 크게 없다. 그리고 매번 assignment 에 data type 이 달라질 수 있다. 매번 다시 메모리를 assign하고 Immutable 로 사용한다.

매개변수에 기본값(DEFAULT)을 지정하기

- 함수를 정의할 때 매개변수를 옵션으로 지정하려면, 매개변수에 값이 전달되지 않았을 때 사용할 기본값(default value)을 정의해 두면 된다.

- round(3.1415) # 매개변수를 하나만 전달
- 3
- round(3.1415, 2) # 두 번째 매개변수로 반올림 자리 지정
- 3.14

- def 동전계산(오백원=0, 백원=0, 오십원=0, 십원=0):
- """동전의 개수를 입력받아 돈의 합계를 계산한다."""
- return 오백원 * 500 + 백원 * 100 + 오십원 * 50 + 십원 * 10

- print(동전계산()) # 기본값은 모두 0

- print(동전계산(1, 10, 0, 100)) # 값을 순서대로 전달

- print(동전계산(십원=220)) # 십원짜리만 220개
- print(동전계산(백원=3, 오십원=2)) # 백원 3개, 오십원 2개

```
def printName(first, last, reverse):
    if reverse:
        print(last+', '+first)
    else:
        print(first, last)

printName('Won', last='Sung', True)
```

```
  File "<ipython-input-13-d9edfe9a6414>", line 7
    printName('Won', last='Sung', True)
                                  ^
SyntaxError: positional argument follows keyword argument
```

```
def printName(first, last, reverse=False):
    if reverse:
        print(last+', '+first)
    else:
        print(first, last)

printName('Won', last='Sung')
```

```
Won Sung
```

```
def print_name(first, last, reverse=True):
    if reverse:
        print(last+', '+first)
    else:
        print(first, last)

print_name('Won', 'Sung', True)
print_name('Won', 'Sung', reverse=False)
print_name('Won', last = 'Sung', reverse=False)
print_name('Won', last = 'Sung')
```

```
Sung, Won
Won Sung
Won Sung
Sung, Won
```

```python
def f():
    print(x)

def g():
    print(x)
    x=1

x=3
f()
x=3
g()
```

```
3

---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-27-d1da7ff9c6b9> in <module>
      9 f()
     10 x=3
---> 11 g()

<ipython-input-27-d1da7ff9c6b9> in g()
      3
      4 def g():
----> 5     print(x)
      6     x=1
      7

UnboundLocalError: local variable 'x' referenced before assignment
```

```
def f(x):
    def g():
#         x = 'abc'
        print('x=', x)

    def h():
        z=x
        print('z=', z)

    x=x+1
    print('x=', x)
    h()
    print('x', x)
    return g

x=3
z=f(x)
print('x=', x)
print('z=', z)
z()
```

```
x= 4
z= 4
x 4
x= 3
z= <function f.<locals>.g at 0x000001554146BEE8>
x= 4
```

```
def f(x):
    def g():
#         x = 'abc'
        print('x=', x)

    def h():
        z=x
        print('z=', z)

    x=x+1
    print('x=', x)
    h()
    print('x', x)
    return g

x=3
z=g()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call
<ipython-input-26-2bbc5644fd2a> in <module>
     15
     16 x=3
---> 17 z=g()

NameError: name 'g' is not defined
```

MULTIPLE AND NONE RETURN

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- Python의 함수에서 여러 개의 값을 return 할 수 있다. 이 들이 tuple 로 묶여서 온다 (괄호로 묶인 값). 그런데 하나씩 받을 수도 있다.

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

def g():
    print("g")

x=2
a, b, c = f(x)
print(a, b, c)
print(f(x))
print(g())
```

```
3 9 27
(3, 9, 27)
g
None
```

HARDER SCOPE EXAMPLE

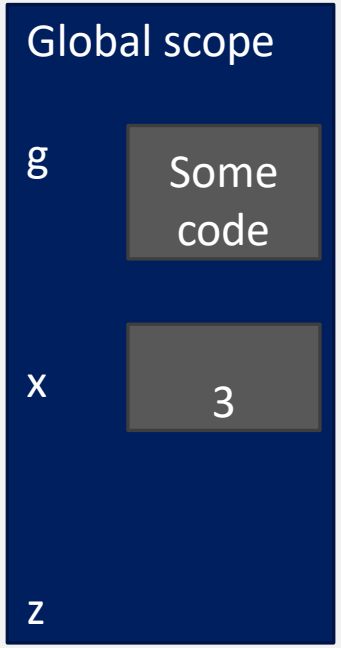
Python Tutor is your best friend to help sort this out!

http://www.pythontutor.com/

SCOPE DETAILS

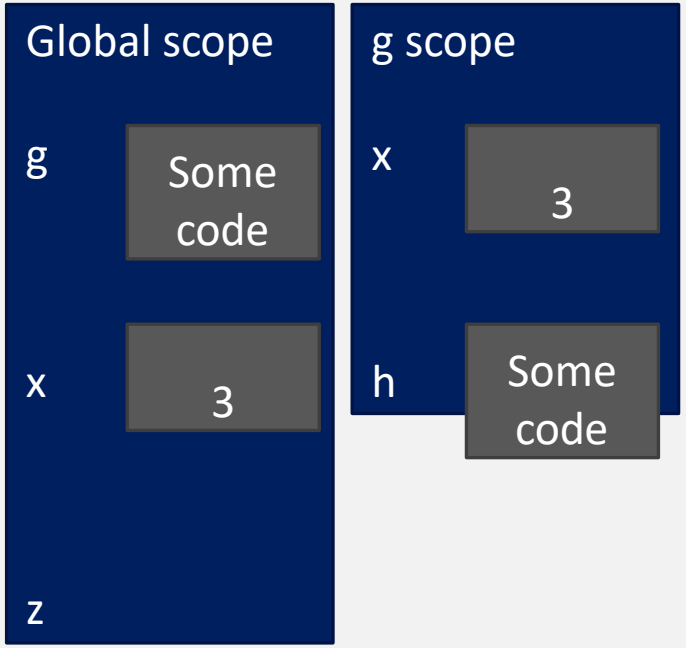
```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```

Some code



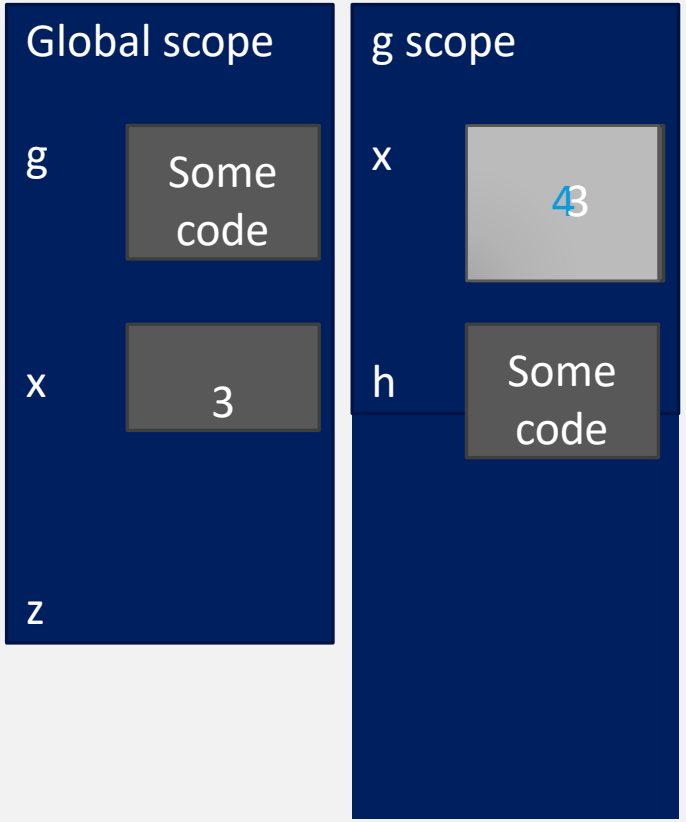
SCOPE DETAILS

```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



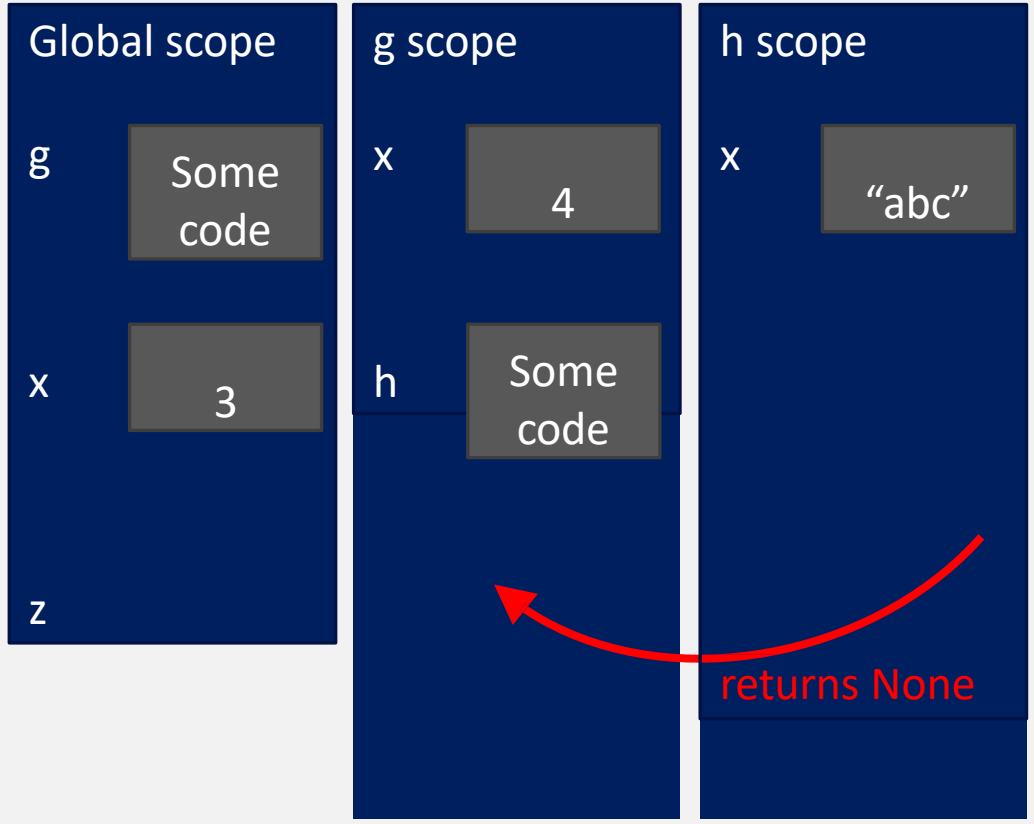
SCOPE DETAILS

```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



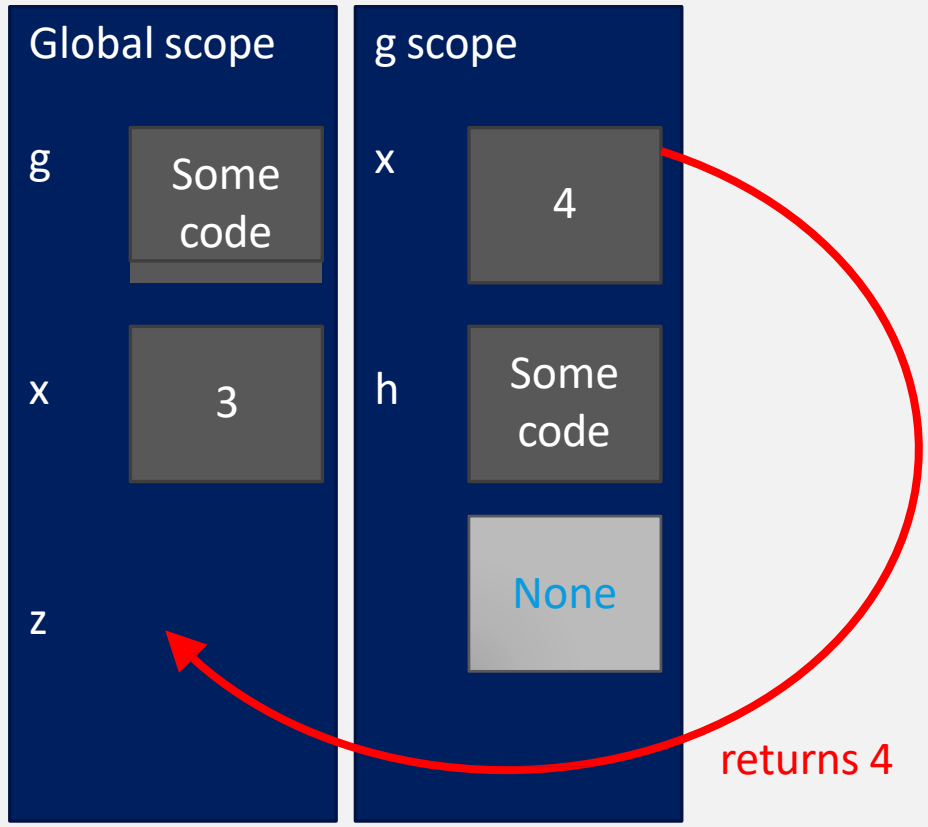
SCOPE DETAILS

```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



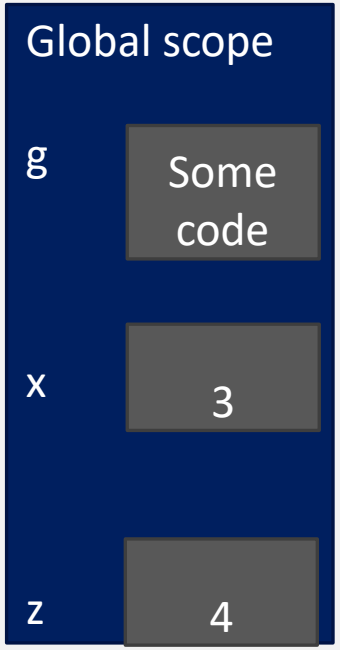
SCOPE DETAILS

```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



SCOPE DETAILS

```
def g(x):
    def h():
        x = 'abc'
    x = x + 1
    print('g: x =', x)
    h()
    return x

x = 3
z = g(x)
```



재귀함수 (RECURSION)

- def로 정의한 함수는 본문이나 다른 함수에서 이용할 수 있다.
- 함수에서 자기 자신을 불러오는 것도 가능한데 이를 재귀함수(recursion)라고 하며, loop처럼 반복적인 행위를 할 때 이용할 수 있다.

```
def hello(count):
    if count == 0:    # 종료 조건을 만듦.
        print("count=0")
        return
    print('Hello, world!', count)
    count -= 1      # count를 1감소
    hello(count)    # 다시 hello에
    print("coming here?", count)

hello(3)
```

```
Hello, world! 3
Hello, world! 2
Hello, world! 1
count=0
coming here? 0
coming here? 1
coming here? 2
```

http://www.pythontutor.com/visualize.html#mode=display

PYTHON 내장함수 : PRINT()

서울대학교 SEOUL NATIONAL UNIVERSITY

- f-문자열 : print(f"print내용 {변수명:표현식}")을 통해 변수를 원하는 표현식으로 표현할 수 있음. 표현식은 정수를 넣으면 최소 문자 폭, d는 정수형, nf는 소수점 n번째까지 표시하라는 뜻. 생략 가능.
- format() : string의 클래스 함수 가운데 format()을 이용하여 변수나 다른 값을 string에 할당할 수 있음. "{ } ... { }".format('asdf', 변수)
- -> {} 내에 정수 0부터 순서를 입력하는걸로 순서를 바꾸는것도 가능
- \, ', "을 출력하고 싶다면 앞에 \를 넣으면 된다.

PYTHON 내장함수 : PRINT()

서울대학교 SEOUL NATIONAL UNIVERSITY

- print의 인자 중 end는 기본적으로 '\n'으로 설정되어 있어 print할때마다 줄이 바뀐다. 이를 원하지 않으면 end 인자를 다르게 할당하면 된다. print("hello ", end='')
- print의 인자 중 sep은 쉼표로 나뉘는 부분에 들어가는 값. 기본적으로 공백이 들어가 있다.
- print의 인자 중 file은 출력 결과를 보내는 곳. 별도의 파일명을 넣으면 그곳에 문자열이 저장된다.

PYTHON 내장함수 : INPUT()

서울대학교 SEOUL NATIONAL UNIVERSITY

- input(A) : A를 출력하면서 유저에게서 입력을 받은 값을 string으로 반환하는 함수. int(), float(), eval() (int와 float를 자동으로 판단)함수를 이용해 다른 형으로 변환한다.
- s1, s2 = input().split() 과같은 형태를 사용하면 공백으로 분리된 두개의 입력을 받을 수 있다.(자세한 활용은 split을 검색)

PRINT WITH FORMAT

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

```
#Don't do this:

name = 'Tyler'
level = 15
rank = 'Supreme'
#instead of doing this:
print('Hello '+name+', you are on level '+str(level)+' and your rank is
'+rank+'.')

#rather do this:

print('Hello {}, you are on level {} and your rank is
{}.'.format(name,level,rank))
```

PYTHON 내장함수 : OPEN(1)

서울대학교 SEOUL NATIONAL UNIVERSITY

- 파일을 읽는 함수, open(읽을 파일 이름, 읽는 방법(default “r”))
- 읽는 방법은 “w” – 쓰기 모드, “r” – 읽기모드, “a” – 추가모드, “b” – 바이너리 모드(w,r,a중 하나의 뒤에 붙음. 010010101같이 된 파일.)
- 흔히 with open(파일명, “r”) as 지정할 파일이름: 처럼 with구문과 함께 이용된다. 이 구문을 이용하지 않으면 파일명.close()를 이용한다.
- ex) with open(“hello world.py”, “r”) as HW:
- 파일 데이터베이스 쪽으로는 Json, pickle등 더 강력한 외장모듈들이 있음.
- 인자 가운데 default가 “UTF-8”로 되어 있는 encoding이 있는데 다른 형식으로 저장된 파일의 경우 이를 변경할 수 있어 보인다.

PYTHON 내장함수 : OPEN(2)

서울대학교 SEOUL NATIONAL UNIVERSITY

- read모드에서 쓰는 형식들
- - 파일이름.readlines() : 읽어온 파일들을 전부 읽어와서 한줄한줄을 원소로 가지는 리스트 형태로 반환한다.
- - 파일이름.read() : 파일을 읽어 문자열로 반환함.
- - for line in 파일이름 : 읽어 온 파일의 각 line에 대해 for문을 돌리게 된다.

PYTHON 내장함수 : OPEN(3)

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- write/append 모드는 읽어온 파일명에 내용을 쓸 수 있다는 점에서 거의 유사하다. 차이점은 wrtte는 파일을 새로 작성, append는 기존 파일에 내용을 추가한다는 점.
- - 파일명.write(“새로 적을 내용”)을 통해 파일에 내용을 추가할 수 있다.

기타 내장함수

- zip(갯수제한 없이 반복) : 같은 개수로
- 이루어진 자료형들을 하나씩 묶어준다.

- eval(실행가능한 문자열) : 문자열 내부를 계산해서 반환한다.
- ex) eval(“1+2”) -> 3 eval(‘”hi”+”a”’) ->’hia’
- enumerate(어레이) : 어레이를 순서값과 이름을 붙여 반환한다.
- ex) for i, name in enumerate([list]): 를 이용하면 i를 통해 몇 번째인지, name을 통해 무슨 원소인지를 이용할 수 있다.

```
>>> list(zip([1, 2, 3], [4, 5, 6]))
[(1, 4), (2, 5), (3, 6)]
>>> list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
[(1, 4, 7), (2, 5, 8), (3, 6, 9)]
>>> list(zip("abc", "def"))
[('a', 'd'), ('b', 'e'), ('c', 'f')]
```

라이브러리와 외장함수(1)

- 현재 python파일에서 사용하기 위해 외부의 코드로부터 클래스와 함수를 가져와야 하는 경우가 있다. 이럴 때 공통적으로 사용하는 것들을 모듈화 시킨 기능들의 덩어리. (환경에 따라 유료인 것도 있으며, windows의 경우 dll확장자들이 런타임 라이브러리다)
- 파이썬에선 이들을 모듈이라고 칭한다. 기본적으로 외장 모듈과 내장 모듈이 있으며 필요한 것을 다운받아서 사용한다.
- tensorflow, numpy, pickle등 공식적이고 규모가 방대한 모듈부터 여러분들이 직접 사용하기 위해 커스텀으로 만드는 것까지 다양한 규모를 가진다.

라이브러리와 외장함수(2)

- from, import, as를 이용해 의도에 맞게 import할 수 있다.
- import : 모듈 및 파일을 불러옴
- from ~ import : 모듈 A에서 특정 함수나 클래스 ~만을 가져옴. 모듈의 함수나 클래스명이 겹칠 경우 둘 중 하나가 무시되거나 에러가 날 수 있으므로 사용할 함수만 가져오는데 사용
- as ~ : 가져온 모듈의 이름을 ~로 바꿈
- ex) from tensorflow import keras as KR : 텐서플로우 모듈에서 keras 모듈을 가져온 후 KR로 호출한다.

기본 외장함수(1) - SYS

- 파이썬 인터프리터의 내부변수와 함수를 제어하는 모듈
- sys.path : 파이썬 모듈이 저장된 위치, list형이므로 append로 새 경로 이름을 추가할 수 있음. 추가한 뒤로는 해당 폴더의 모듈을 불러올 수 있음.
- sys.exit() : 파이썬 스크립트 강제종료
- sys.argv : python파일 실행시 인자로 받은 것들. 공백을 기준으로 하나씩 구분한다. (argparse라는 모듈을 사용하기도 함)
- ex) python test.py a b c로 파일을 실행시
- print(sys.argv)를 하면 [‘a’,’b’,’c’]가 출력된다.

기본 외장함수(2) - RANDOM

- 난수를 발생시키는 모듈.
- random.random() : 0.0과 1.0사이의 float 난수를 생성한다
- random.randint(a,b) : a와 b사이의 정수 난수를 생성한다.
- random.suffle(어레이) : 어레이의 순서를 섞는다.

기본 외장함수(3) - TIME

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- time.time() : UTC의 1월 1일 0시 0분 0초를 기준으로 지난 시간을 초단위로 반환함.
- time.localtime(실수-일반적으로 time.time()) : 받은 시간값을 년월일시분초의 형태로 바꿔서 튜플의 형태로 return
- time.asctime(time.localtime) : time.localtime의 튜플값을 받아 알아보기 쉬운 string으로 반환함.
- time.ctime() : 현재 시간을 알아보기 쉽게 string으로 반환함.
- time.strftime('코드', time.localtime(실수)) : 원하는 표현형식으로 시간을 표현함(코드의 종류가 굉장히 많음)
- time.sleep(실수) : 실수에 해당하는 초 간격만큼 진행을 멈춘다.

기본 외장함수(4) - PICKLE

- 객체의 형태를 유지하면서 저장 및 불러오기를 할 수 있는 모듈. open(file, “wb”)을 먼저 해서 저장할 파일을 만들어 둔 뒤에 사용한다.
- pickle.dump(저장할 객체, 저장할 파일명) : 객체를 형태 그대로 파일에 저장한다.
- data = pickle.load(불러올 파일명) : 저장했던 객체를 불러온다.
- 객체를 그대로 저장한다는 것은 복잡한 형태의 객체를 저장할 때 단순히 텍스트나 별도의 형태로 변환하는 과정이 없다는 뜻. 이렇게 저장하지 않으면 객체가 단순한 텍스트로 저장 되어버린다.

MODULES

- A module is a file containing Python definitions and statements
- File name is the module name with the suffix “.py”

A module can contain executable statements as well as function definitions. These statements are intended to initialize the module. They are executed only the first time the module name is encountered in an import statement. 1 (They are also run if the file is executed as a script.)

PACKAGE

서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

- pip install tensorflow
- Import tensorflow

- Useful Python packages
  - Data sciences
    - Numpy
    - SciPy
    - Pandas: data manipulation and visualization
  - Visualization
    - Matplotlib
  - Machine learning
    - Tensorflow
    - Keras

A package is the form of a collection of tools which helps in the initiation of the code. A python package acts as a user-variable interface for any source code. This makes a python package work at a defined time for any functionable code in the runtime.

What actually makes a Python Package different from Modules?

This is a generally asked question, what makes a python package different from a module. A python package works on a library, defining the codes as a single unit of any particular function. While the modules are a separate library themselves, which have inbuilt functions. The reusability of packages makes it more preferable over modules.