Function Concepts

Function

최성철 교수
Director of TEAMLAB

프로그램을 여럿이

개발할 경우 코드를

어떻게 작성해야 할까?

[생각해보기]

프로그램을 여럿이 개발할 경우 코드를 어떻게 작성해야 할까?

① 다같이 모여서 토론하며 한 줄 한 줄

② 제일 잘하는 사람이 혼자 작성

③ 필요한 부분을 나눠서 작성한 후 합침

[생각해보기]

프로그램을 기능별로 나누는 방법

① 함수 ② 객체 ③ 모듈

함수

함수 (Function)

어떤 일을 수행하는 코드의 덩어리

```
# 사각형의 넓이를 구하는 함수
def calculate_rectangle_area (x , y):
    return x * y        # 가로, 세로를 곱해서 반환
```

- 반복적인 수행을 1회만 작성 후 호출
- 코드를 논리적인 단위로 분리
- 캡슐화: 인터페이스만 알면 타인의 코드 사용

함수 선언 문법

함수 이름, parameter, return value(optional)

```
def 함수 이름 (parmaeter #1 ….):
    수행문 #1(statements)
    수행문 #2(statements)
    return <반환값>
```

함수 선언 예시

```
def calculate_rectangle_area(x , y)
    return x * y

rectangle_x = 10
rectangle_y = 20
print ("사각형 x의 길이: ", rectangle_x)
print ("사각형 y의 길이: ", rectangle_y)

# 넓이를 구하는 함수 호출
print ("사각형의 넓이: ", calculate_rectangle_area(rectangle_x,
rectangle_y))
```

함수 수행 순서 (w/함수)

- 함수 부분 를 제외한 메인프로그램부터 시작
- 함수 호출 시 함수부분을 수행 후 되돌아옴

```
def calculate_rectangle_area(x , y)
    return x * y

rectangle_x = 10
rectangle_y = 20
print ("사각형 x의 길이: ", rectangle_x)
print ("사각형 y의 길이: ", rectangle_y)

# 넓이를 구하는 함수 호출
print ("사각형의 넓이: ", calculate_rectangle_area(rectangle_x, rectangle_y))
```

3 함수 수행

1 메인 프로그램 수행

4 메인 프로그램 수행

2 함수 호출

[알아두면 상식] 함수 vs 함수 (1/2)

- 프로그래밍의 함수와 수학의 함수는 유사함
- 모두 입력 값과 출력 값으로 이루어짐

f(x) = 2x + 7 , g(x) = x2 이고 x = 2일 때
f(x) + g(x) + f(g(x)) + g(f(x))의 값은?

f(2) = 11, g(2) = 4, f(g(x)) = 15, g(f(x))=121
11 + 4 + 15 + 121 = 151

이 공식을 파이썬으로 작성하면?

[알아두면 상식] 함수 vs 함수 (2/2)

```
def f(x):
    return 2 * x + 7
def g(x):
    return x ** 2
x = 2
print (f(x) + g(x) + f(g(x)) + g(f(x)))
```

f(x) 함수 선언

g(x) 함수 선언

Parameter vs. Argument

- Parameter : 함수의 입력 값 인터페이스

```
def f(x):
    return 2 * x + 7
```

- Argument: 실제 Parameter에 대입된 값

```
>>> print(f(2))
11
```

함수 형태

Parameter 유무, 반환 값(return value)
유무에 따라 함수의 형태가 다름

	Parameter 없음	Parameter 존재
반환 값 없음	함수 내의 수행문만 수행	인자를 사용, 수행문만 수행
반환 값 존재	인자없이, 수행문 수행 후 결과값 반환	인자를 사용하여 수행문 수행 후 결과값 반환

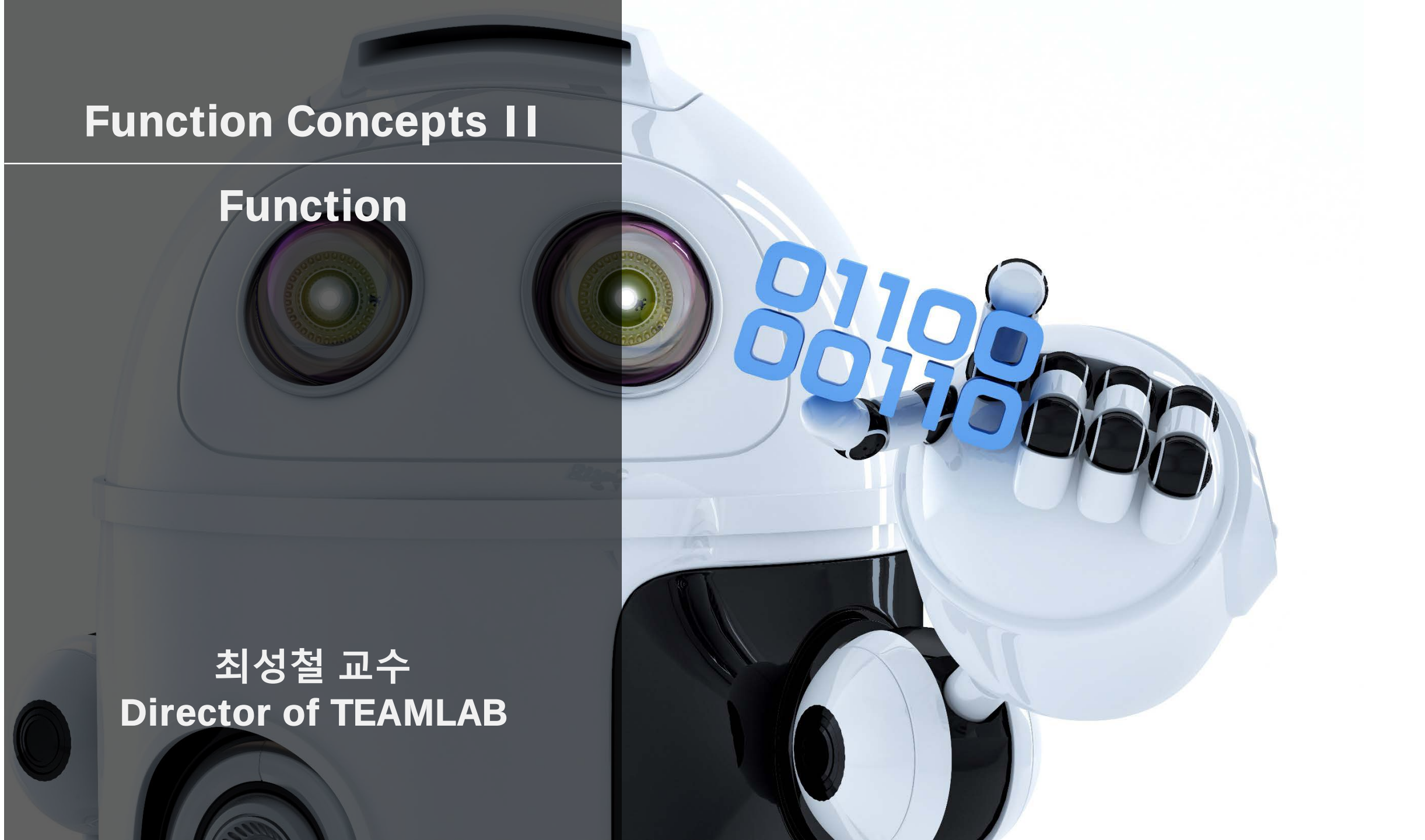
함수 형태 예제

```
def a_rectangle_area():# 인자 X , 리턴 값 X
    print (5 * 7)
def b_rectangle_area(x,y): # 인자 O , 리턴 값 X
    print (x * y)
def c_rectangle_area():    # 인자 X , 리턴 값 O
    return (5 * 7)
def d_rectangle_area(x ,y):# 인자 O , 리턴 값 O
    return (x * y)

a_rectangleArea()
b_rectangleArea(5,7)
print (c_rectangleArea())
print (d_rectangleArea(5,7))
```

Human knowledge belongs to the world.

Function Concepts II

Function



최성철 교수
Director of TEAMLAB

함수 호출 방식

함수 호출 방식 개요 (1/2)

함수의 인자를 전달하는 방식

값에 의한 호출(Call by Value)

참조의 의한 호출(Call by Reference)

함수 호출 방식 개요 (2/2)

Call by Value
함수에 인자를 넘길 때 값만 넘김.
함수 내에 인자 값 변경 시, 호출자에게 영향을 주지 않음

Call by Reference
함수에 인자를 넘길 때 메모리 주소를 넘김.
함수 내에 인자 값 변경 시, 호출자의 값도 변경됨

파이썬 함수 호출 방식(1/2)

파이썬은 객체의 주소가 함수로 전달되는 방식

전달된 객체를 참조하여 변경 시 호출자에게 영향을 주나,
새로운 객체를 만들 경우 호출자에게 영향을 주지 않음

파이썬 함수 호출 방식(2/2)

```
def spam(eggs):
    eggs.append(1) # 기존 객체의 주소값에 [1] 추가
    eggs = [2, 3]  # 새로운 객체 생성


ham = [0]
spam(ham)
print(ham)  # [0, 1]
```

Function Call Test

```
def test(t):
      t = 20
      print ("In Function :", t)

x = 10
print ("Before :", x)    # 10
test(x)                  # 함수 호출
print ("After :", x)     # 10 – 함수 내부의 t는 새로운 주소값을 가짐
```

변수의 범위

변수의 범위 (Scoping Rule)

- 변수가 사용되는 범위 (함수 또는 메인 프로그램)
- 지역변수(local variable) : 함수내에서만 사용
  전역변수(Global variable) : 프로그램전체에서 사용

```
def test(t):
    print(x)
    t = 20
    print ("In Function :", t)

x = 10
test(x)
print(t)
```

변수의 범위 (Scoping Rule)

- 전역변수는 함수에서 사용가능
- But, 함수 내에 전역 변수와 같은 이름의 변수를 선언하면 새로운 지역 변수가 생김

```
def f():
    s = "I love London!"
    print(s)

s = "I love Paris!"
f()
print(s)
```

http://goo.gl/O3vDwy

변수의 범위 (Scoping Rule)

- 함수 내에서 전역변수 사용 시 global 키워드 사용

```
def f():
    global s
    s = "I love London!"
    print(s)

s = "I love Paris!"
f()
print(s)
```

http://goo.gl/O3vDwy

변수의 범위 (Scoping Rule)

```
def calculate(x, y):
    total = x + y    # 새로운 값이 할당되어 함수 내 total은 지역변수가 됨
    print ("In Function")
    print ("a:", str(a), "b:", str(b), "a+b:", str(a+b), "total :", str(total))
    return total

a = 5           # a와 b는 전역변수
b = 7
total = 0       # 전역변수 total
print ("In Program - 1")
print ("a:", str(a), "b:", str(b), "a+b:", str(a+b))

sum = calculate (a,b)
print ("After Calculation")
print ("Total :", str(total), " Sum:", str(sum)) # 지역변수는 전역변수에 영향 X
```

Swap

- 함수를 통해 변수 간의 값을 교환(Swap)하는 함수
- Call By XXXX를 설명하기 위한 전통적인 함수 예시

```
Enter the value of x and y
4
5
Before Swapping
x = 4
y = 5
After Swapping
x = 5
y = 4
```

http://goo.gl/ZNht49

Swap

a = [1,2,3,4,5] 일 때 아래 함수 중 실제 swap이 일어나는 함수는?

```
def swap_value (x, y):
        temp = x
        x = y
        y = temp
```

```
def swap_offset (offset_x, offset_y):
        temp = a[offset_x]
        a[offset_x] = a[offset_y]
        a[offset_y] = temp
```

```
def swap_reference (list, offset_x, offset_y):
        temp = list[offset_x]
        list[offset_x] = list[offset_y]
        list[offset_y] = temp
```

Swap

swap_offset: a 리스트의 전역 변수 값을 직접 변경

swap_reference: a 리스트 객체의 주소 값을 받아 값을 변경

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

swap_value(a[1], a[2])
print (a)                    # [1,2,3,4,5]

swap_offset(1,2)
print (a)                    # [1,3,2,4,5]

swap_reference(a, 1, 2)
print (a)                  # [1,3,2,4,5]
```

재귀 함수

재귀함수 (Recursive Function)

- 자기자신을 호출하는 함수
- 점화식과 같은 재귀적 수학 모형을 표현할 때 사용
- 재귀 종료 조건 존재, 종료 조건까지 함수호출 반복

$$n! = n \cdot (n-1) \cdots 2 \cdot 1 = \prod_{i=1}^{n} i$$

$$1! = 1$$

$$2! = 2(1) = 2$$

$$3! = 3(2)(1) = 6$$

$$4! = 4(3)(2)(1) = 24$$

$$5! = 5(4)(3)(2)(1) = 120$$

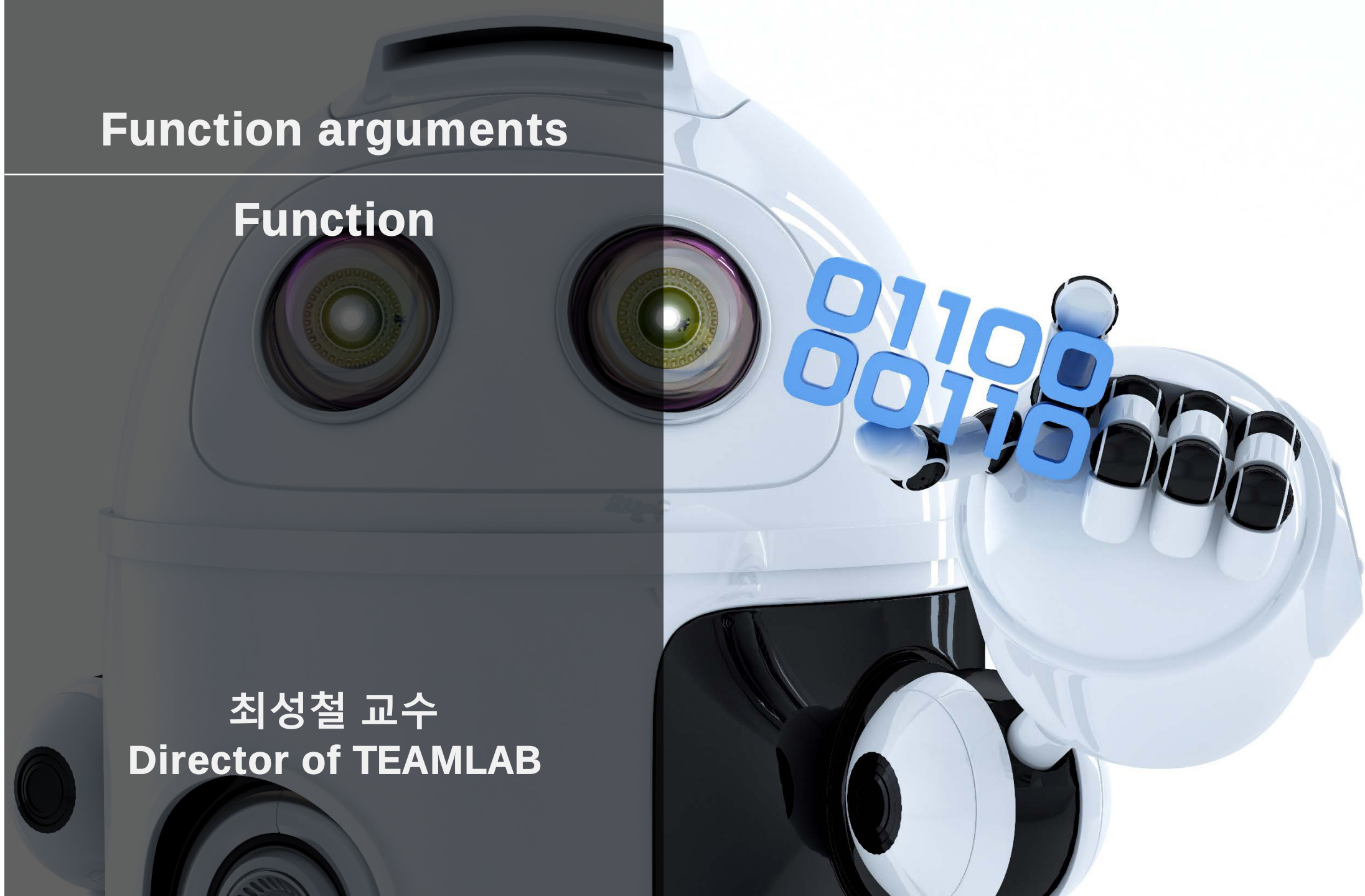
재귀함수 (Recursive Function)

```
def factorial(n):
      if n == 1:
            return 1
      else:
            return n + factorial(n-1)
print (factorial(int(input("Input Number for Factorial
Calculation: "))))
```

Human knowledge belongs to the world.

Function arguments

Function



최성철 교수
Director of TEAMLAB

Passing arguments

Passing arguments

- 함수에 입력되는 arguments는 다양한 형태를 가짐

  1) Keyword arguments
  2) Default arguments
  3) Variable-length arguments

Keyword arguments

- 함수에 입력되는 parameter의 변수명을 사용, arguments를 넘김

```
def print_somthing(my_name, your_name):
    print("Hello {0}, My name is {1}".format(your_name, my_name))


print_somthing("Sungchul", "TEAMLAB")
print_somthing(your_name="TEAMLAB", my_name="Sungchul")
```

Default arguments

- parameter의 기본 값을 사용, 입력하지 않을 경우 기본값 출력

```
def print_somthing_2(my_name, your_name="TEAMLAB"):
    print("Hello {0}, My name is {1}".format(your_name, my_name))


print_somthing_2("Sungchul", "TEAMLAB")
print_somthing_2("Sungchul")
```

Variable-length asterisk

함수의 parameter가 정해지지 않았다?

다항 방정식? 마트 물건 계산 함수?

가변인자 using Asterisk

가변인자 (Variable-length)

- 개수가 정해지지 않은 변수를 함수의 parameter로 사용하는 법
- Keyword arguments와 함께, argument 추가가 가능
- Asterisk(*) 기호를 사용하여 함수의 parameter를 표시함
- 입력된 값은 tuple type으로 사용할 수 있음
- 가변인자는 오직 한 개만 맨 마지막 parameter 위치에 사용가능

가변인자 (Variable-length)

- 가변인자는 일반적으로 *args를 변수명으로 사용
- 기존 parameter 이후에 나오는 값을 tuple로 저장함

```
def asterisk_test(a, b, *args):
    return a+b+sum(args)

print(asterisk_test(1, 2, 3, 4, 5))
```

가변인자 (Variable-length)

```
def asterisk_test_2(*args):
    x, y, z = args
    return x, y, z

print(asterisk_test_2(3, 4, 5))
```

키워드 가변인자 (Keyword variable-length)

- Parameter 이름을 따로 지정하지 않고 입력하는 방법
- Asterisk(*) 두개를 사용하여 함수의 parameter를 표시함
- 입력된 값은 dict type으로 사용할 수 있음
- 가변인자는 오직 한 개만 기존 가변인자 다음에 사용

키워드 가변인자 (Keyword variable-length)

```
def kwargs_test_1(**kwargs):
    print(kwargs)


def kwargs_test_2(**kwargs):
    print(kwargs)
    print("First value is {first}".format(**kwargs))
    print("Second value is {second}".format(**kwargs))
    print("Third value is {third}".format(**kwargs))
```

키워드 가변인자 (Keyword variable-length)

```
def kwargs_test_3(one,two, *args, **kwargs):
    print(one+two+sum(args))
    print(kwargs)

kwargs_test_3(3,4,5,6,7,8,9, first=3, second=4, third=5)
```

Human knowledge belongs to the world.

How to write GOOD code

Function

http://goo.gl/4OxqtO

프로그래밍 = 팀 플레이

좋은 팀을 위한 규칙

사람을 위한 코드

http://goo.gl/u21VvR

컴퓨터가 이해할 수 있는
코드는 어느 바보나 다 짤 수 있다.

좋은 프로그래머는 사람이
이해할 수 있는 코드를 짠다.

- 마틴 파울러 –

사람의 이해를 돕기 위해

규칙이 필요함

우리는 그 규칙을

코딩 컨벤션

이라고 함

파이썬 코딩 컨벤션

파이썬 코딩 컨벤션

- 명확한 규칙은 없음
- 때로는 팀마다, 프로젝트마다 따로
- 중요한 건 **일관성**!!!
- 읽기 좋은 코드가 좋은 코드

http://goo.gl/OCQJ35

파이썬 코딩 컨벤션의 예시

- 들여쓰기는 Tab or 4 Space 논쟁!

- 일반적으로 4 Space를 권장함

- 중요한 건 혼합하지 않으면 됨

http://goo.gl/2x3G51

PEP8 – 파이썬 코딩 컨벤션의 기준

- PEP (Python Enhance Proposal)
- 파이썬 개선을 위한 제안서
- PEP 8은 파이썬 코딩의 기준을 제시

http://goo.gl/2x3G51

PEP8 – 파이썬 코딩 컨벤션의 기준

- 들여쓰기 공백 4칸을 권장
- 한 줄은 최대 79자까지
- 불필요한 공백은 피함

```
def factorial( n ):
    if n == 1:
        return 1
```

PEP8 – 파이썬 코딩 컨벤션의 기준

- = 연산자는 1칸 이상 안 띄움

```
variable_example   =   12
variable_example = 12
```

- 주석은 항상 갱신, 불필요한 주석은 삭제

PEP8 – 파이썬 코딩 컨벤션의 기준

- 소문자 l, 대문자 O, 대문자 I 금지

```
lIO0 = "Hard to Understand"
```

- 함수명은 소문자로 구성, 필요하면 밑줄로 나눔

PEP8 – 파이썬 코딩 컨벤션의 기준

- "flake8" 모듈로 체크 – flake8 <파일명>
- conda install -c anaconda flake8

```
lL0O = "123"
for i in 10 :
    print ("Hello")
```

```
flake8 flake8_test.py
flake8_test.py:2:12: E203 whitespace before ':'
flake8_test.py:3:10: E211 whitespace before '('
```

함수 개발 가이드라인

함수 작성 가이드 라인

- 함수는 가능하면 짧게 작성할 것 (줄 수를 줄일 것)

- 함수 이름에 함수의 역할, 의도가 명확히 들어낼 것

```
def print_hello_world():
    print("Hello, World")
def get_hello_world():
    return "Hello, World")
```

함수 작성 가이드 라인

- 하나의 함수에는 유사한 역할을 하는 코드만 포함

```
def add_variables(x,y):
    return x + y

def add_variables(x,y):
    print (x, y)
    return x + y
```

함수 작성 가이드 라인

- 인자로 받은 값 자체를 바꾸진 말 것 (임시변수 선언)

```
def count_word(string_variable):
     string_variable = list(string_variable)
     return len(string_variable)
```

```
def count_word(string_variable):
     return len(string_variable)
```

함수는 언제 만드는가?

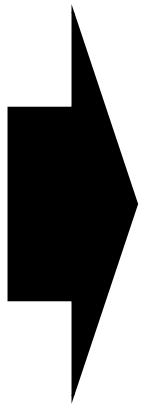
- 공통적으로 사용되는 코드는 함수로 변환

- 복잡한 수식 → 식별 가능한 이름의 함수로 변환

- 복잡한 조건 → 식별 가능한 이름의 함수로 변환

공통 코드는 함수로

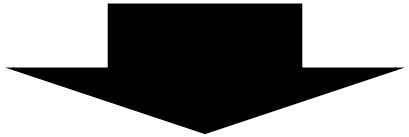
```
a = 5
if (a > 3):
      print "Hello World"
      print "Hello Gachon"
if (a > 4):
      print "Hello World"
      print "Hello Gachon"
if (a > 5):
      print "Hello World"
      print "Hello Gachon"
```



```
def print_hello():
      print "Hello World"
      print "Hello Gachon"

a = 5
if (a > 3):
      helloMethod()

if (a > 4):
      helloMethod()

if (a > 5):
      helloMethod()
```

복잡한 수식은 함수로

```
import math
a = 1; b = -2; c = 1

print ((-b + math.sqrt(b ** 2 - (4 * a * c)) )  / (2 * a))
print ((-b -  math.sqrt(b ** 2 - (4 * a * c)) )  / (2 * a))
```



```
import math

def get_result_quadratic_equation(a, b, c):
        values = []
        values.append((-b + math.sqrt(b ** 2 - (4 * a * c)) )  / (2 * a))
        values.append((-b - math.sqrt(b ** 2 - (4 * a * c)) )  / (2 * a))
        return values

print (get_result_quadratic_equation(1,-2,1))
```

Human knowledge belongs to the world.