

오늘의 강의 목표

- Function에 대한 이해
- Function argument에 대한 이해
- Variable scope에 대한 이해
- Function return value에 대한 이해

Python Built-in Functions

- Python이 미리 제공하는 함수들

```
>>> print("Hello World")    # print 함수  
Hello World
```

```
>>> r = pow(2, 10)          # pow 함수  
>>> print(r)  
1024
```

...

- 함수(function)란?
 - Named sequence of statements

User-Defined Functions

- 사용자가 직접 함수 정의

함수 정의 시작

함수 이름

```
>>> def print_hello_three_times():
```

```
    print("Hello")
```

```
    print("Hello")
```

```
    print("Hello")
```

} 함수 호출 시 실행할 statements

함수 호출

```
>>> print_hello_three_times()
```

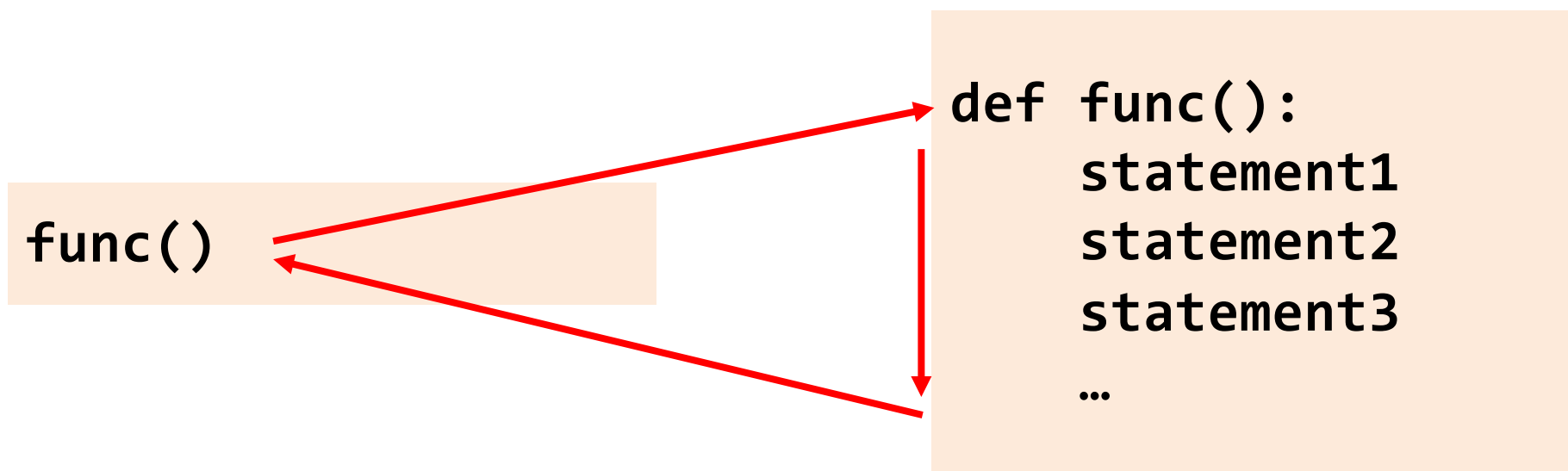
```
Hello
```

```
Hello
```

```
Hello
```

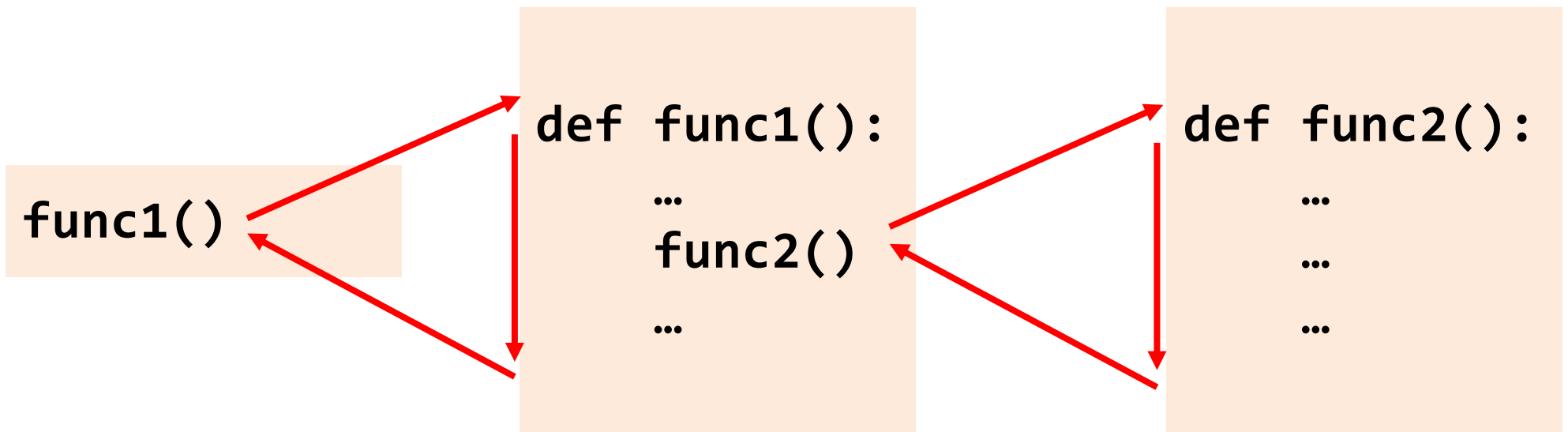
Control Flow

- 호출 시점에 해당 함수의 statement들이 실행



Nested Function Calls

- 함수 안에서 다시 함수 호출



Why we need functions?

- 여러 프로그램이 공통적으로 사용하는 기능 미리 제공
- (한 프로그램만 사용하더라도) 여러 번 사용되는 기능을 한번 정의하여 쉽게 사용
- (한번만 사용되더라도) 복잡한 문제를 함수로 쪼개서 (Divide and Conquer) 해결

Program with Functions

- 함수 정의 후 프로그램 시작

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

} 함수 정의1

```
def print_hello_three_times():  
    print_hello()  
    print_hello()  
    print_hello()
```

} 함수 정의2

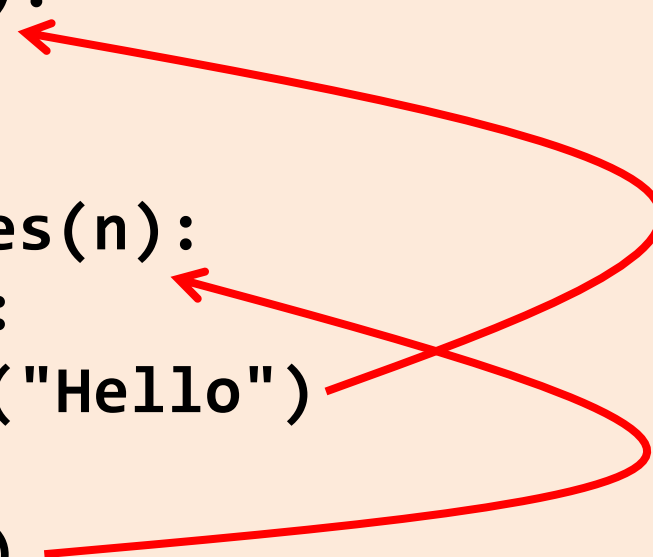
```
print_hello_three_times()
```

← 프로그램 시작

Function Parameters

- Information from caller to callee

```
def print_message(msg):  
    print(msg)  
  
def print_hello_n_times(n):  
    for _ in range(n):  
        print_message("Hello")  
  
print_hello_n_times(3)
```



For의 index variable 자리에 _ (underbar) → index variable을 사용하지 않음을 의미

Multiple Function Parameters

- Use commas

```
def print_message(msg):  
    print(msg)
```

```
def print_message_n_times(msg, n):  
    for _ in range(n):  
        print_message(msg)
```

Multiple Parameters (order matters)

```
print_message_n_times("Hello", 3)
```

Keyword-based Argument Passing


- Keyword = value

```
def print_message(msg):  
    print(msg)
```

```
def print_message_n_times(msg, n):  
    for _ in range(n):  
        print_message(msg)
```

Keyword-based
argument passing
(no orders)

```
print_message_n_times(msg = "Hello", n = 3)  
print_message_n_times(n = 3, msg = "Hello")
```



Default Arguments

- Used when an argument is not provided

```
def print_message(msg):  
    print(msg)
```

Default arguments



```
def print_message_n_times(msg = "Hello", n = 1):  
    for _ in range(n):  
        print_message(msg)
```

```
print_message_n_times(msg = "Hello")  
print_message_n_times(n = 3)  
print_message_n_times()
```

Variable Scope


- **Local variables** : defined within a function and thus accessible only within the function
- **Global variables** : defined outside of functions and accessible from anywhere

Which one is better?

Global Variable

- Global variable : defined outside a function

```
def print_msg():  
    print(msg)  
  
msg = "Hello World"  
print_msg()
```

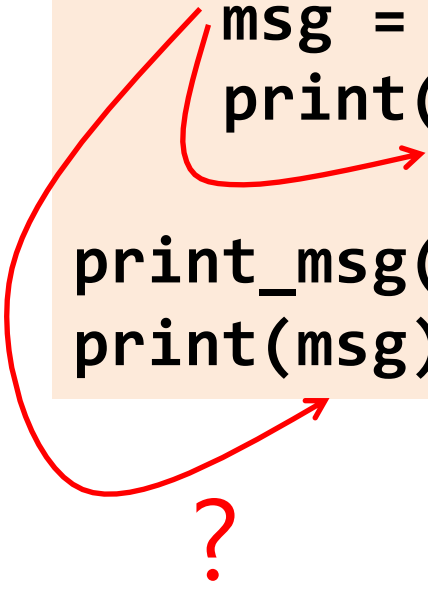


Local Variable

- Local variable : defined within a function

```
def print_msg():  
    msg = "Goodbye World"  
    print(msg)
```

```
print_msg()  
print(msg)
```

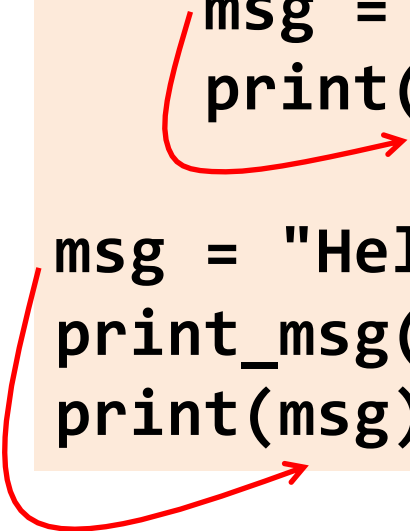


Global vs Local

- 같은 이름(msg)의 변수가 두 개 존재

```
def print_msg():  
    msg = "Goodbye World"  
    print(msg)
```

```
msg = "Hello World"  
print_msg()  
print(msg)
```

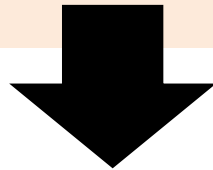


The diagram illustrates variable scope resolution. A red arrow originates from the `print(msg)` line within the `print_msg()` function and points to the `msg = "Goodbye World"` line, indicating that the function uses its local variable. Another red arrow originates from the `print(msg)` line at the bottom of the code block and points to the `msg = "Hello World"` line, indicating that the global variable is used when the variable is not found in the local scope.

Caution

```
def print_msg():  
    print(msg)  
    msg = "Goodbye World"  
    print(msg)
```

```
msg = "Hello World"  
print_msg()  
print(msg)
```



```
Traceback (most recent call last):  
  File "C:/Python34/a.py", line 7, in <module>  
    print_msg()  
  File "C:/Python34/a.py", line 2, in print_msg  
    print(msg)  
UnboundLocalError: local variable 'msg' referenced before assignment
```


Solution

- global 키워드를 이용

```
def print_msg():  
    global msg  
    print(msg)  
    msg = "Goodbye World"  
    print(msg)
```

```
msg = "Hello World"  
print_msg()  
print(msg)
```

Quiz

Correct the following code:

```
total = 0

def add_total(n):
    total += n

add_total(1)
add_total(10)
print(total)
```

General Rule

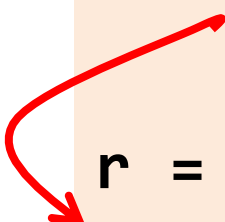
1. 가능한 한 Global Variable은 사용하지 않는다
2. 꼭 필요한 경우는 global을 사용한다

Function Return Value

- Use return

```
import math
```

```
def calc_circle_area(r):  
    return math.pi * (r ** 2)
```



```
r = float(input("Enter radius: "))  
area = calc_circle_area(r)  
print("Area = %f" % (area))
```

Function Return Value

- Return tuples and lists

```
def make_tuple(a, b, c):  
    return (a, b, c)
```

```
def make_list(a, b, c):  
    return [a, b, c]
```

```
t = make_tuple(3, 4, 5)  
l = make_list(3, 4, 5)  
print(t)  
print(l)
```

Advanced Topics

- Variable-length arguments
- Call by reference
- Lambda function

Questions

