

Python 06

김은진



함수 호출(Function call)

- 이미 작성된 함수를 프로그램에서 실행 시키는 방법

작업대상 함수 호출 법

```
>>> player_list.append('김하온')
```

작업대상.함수명(인자값)

특정 '작업대상'만 실행 경우

Built-in 함수 호출 법

```
>>> print ('Hello world')
```

함수명(인자값)

범용적인 작업인 경우



함수 호출 사용 예

pow() 함수를 호출 후 반환값을 화면에 출력하세요

1. 함수이름 : pow
2. 인자값 : 밑수 2, 지수 10
3. 반환값(함수결과값) : 계산 값

함수 결과값이 value에 저장됨

```
>>> value = pow(2, 10)
>>> print(value)
1024
```

함수 호출(실행)의 결과값이
이 위치로 돌아옴

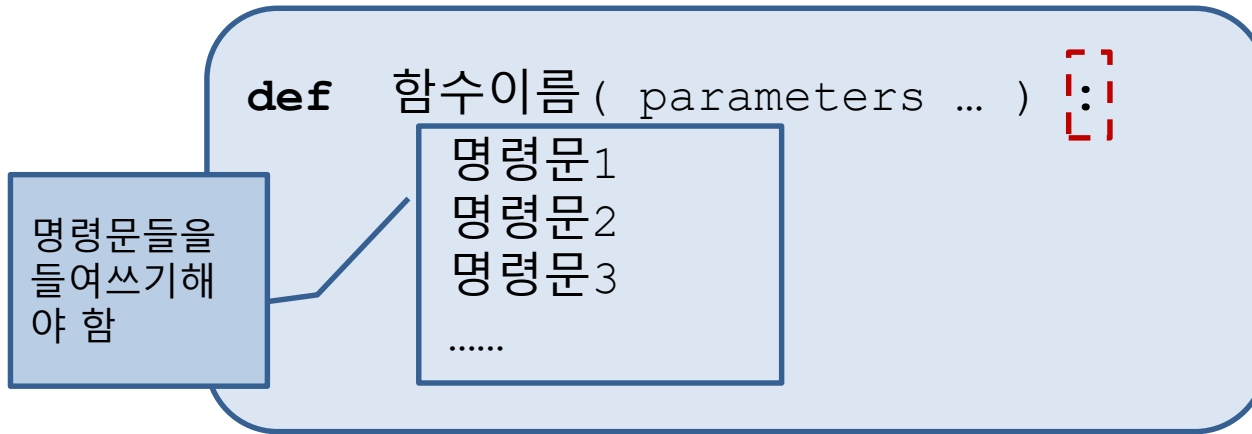
```
>>> print(pow(2, 10))
1024
```

② ①

함수 호출(실행)의 결과값(반환값)은 함수 호출 위치로 돌아옴!!!



함수 만들기



- 함수이름
 - 변수이름 만드는 규칙 따르기
- Parameters
 - 함수 호출 시의 인자값을 받아오는 변수들을 , 로 구분해서 나열
 - optional

함수 만들기 완료 위치는: 더 이상 들여쓰기 안하면
함수를 만든다고 실행되는 것이 아님, 함수 호출해야 실행됨!!!



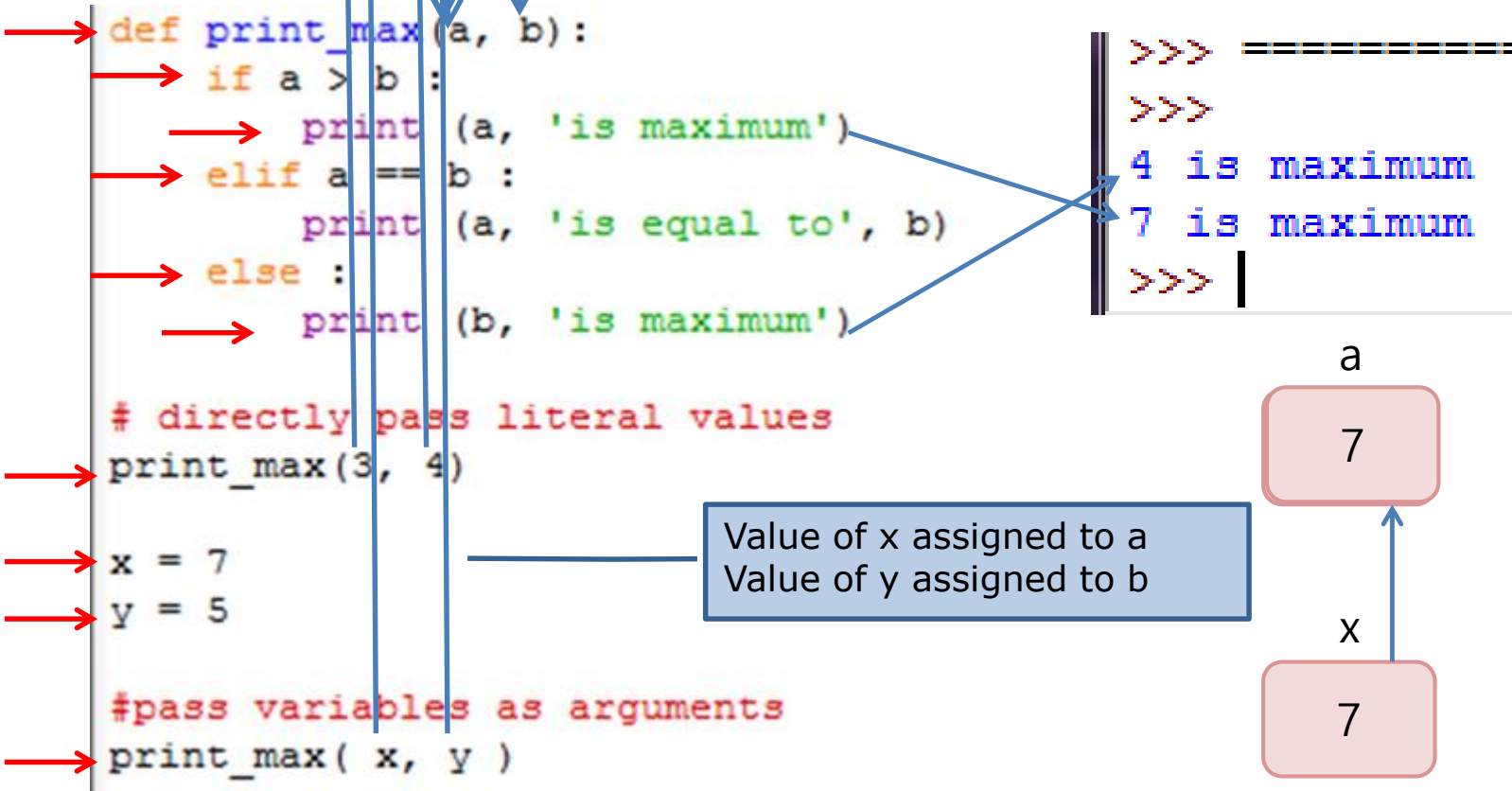
Parameter 2개인 함수 만들기

- 숫자 2개를 인자값으로 받아 둘 중 큰 수를 **출력**하는 함수

```
def print_max(a, b):  
    if a > b :  
        print (a, 'is maximum')  
    elif a == b :  
        print (a, 'is equal to', b)  
    else :  
        print (b, 'is maximum')  
  
# directly pass literal values  
print_max(3, 4)  
  
x = 7  
y = 5  
  
#pass variables as arguments  
print_max( x, y )
```

Value of x assigned to a
Value of y assigned to b

```
>>> =====  
>>>  
4 is maximum  
7 is maximum  
>>> |
```



return 문장

- return
 - 함수의 결과값을 반환
 - 함수 실행을 중단하고 함수 호출했던 자리로 돌아감
(return 문장 이후에 다른 문장 있어도 실행 안 됨)
- 숫자 2개를 인자값으로 받아 둘 중 큰 수를 **결과값으로 반환**하는 함수

```
def maximum( x, y ) :  
    if x > y :  
        return x  
    elif x == y :  
        return 'The numbers are equal'  
    else :  
        return y  
  
print ( maximum ( 2, 3 ) )  
print ( maximum ( 100, 100 ) )
```



return 문이 없는 함수의 반환값

```
def no_return_func( x ):  
    print( x**2 + x )
```

```
res = no_return_func( 4 )  
print( res )
```

- 눈에 보이지 않는 다음과 같은 `return` 문장이 마지막 명령문으로 자동 삽입됨
 `return None`



Parameter에 기본인자값

```
def say ( message, times = 1 ) :    Parameter이름 = 기본인자값  
    print( message * times )
```

say('Hello') say ('Hello', 1) 과 동일한 문장

say('Wow', 5) def 시 기본인자값 있으면 함수 호출시 인자값 안 줘도 됨

- 기본 인자값 지정하는 parameter는 parameter들 중 뒤에서부터 가능
 - def fun(a, b = 1, c) : X
 - def fun(a = 2, b=3, c) : X
 - def fun(a = 2, b=3, c=2) : O



인자 지정 호출

```
def fun ( num, p ) :  
    return num ** p
```

```
print( fun( p = 2, num = 5 ) )
```

```
print( fun( 5, p = 2, ) )
```

fun(p = 2, 5) 은 syntax error

```
print( fun( num = 5, p = 2 ) )
```

키워드인자 : parameter이름 = 인자값 으로 함수 호출

parameter이름을 알고 있어야 함



가변 인자 개수

다음과 같이 호출이 가능한 함수

```
test( 3, 5, 6 )
```

```
test( 3, 1, 8, 10 )
```

```
test( 2 )
```

```
test( )
```

빈 tuple, () 이 parameter인 x 값이 됨

인자값들의 합을 출력하는 함수 작성(인자의 개수는 가변적)

```
def test ( *x ) :  
    print( sum(x) )  
    #print( type(x) )  
    #print( x )
```

parameter 이름 앞의 * :
임의의 개수의 인자값들을 받아 tuple로 저장



함수 호출 인자 정보로 dict를

- 다음과 같이 호출할 때, 함수 fun에 a, bb, ccc 라는 parameter가 없으면?

```
fun ( a = 1, bb= 2, ccc = 3)
```

```
def price(**fruit) : parameter 이름 앞에 **  
    for x in fruit :  
        print( x, '는 ', fruit[x], '원')  
    #print( fruit )    다음과 같은 dict를 parameter에 전달  
                        { 'graph': 1000, 'apple': 700, 'orange': 600 }
```

```
price(graph = 1000, apple = 700, orange = 600)
```



Parameter 이름 앞에 *, ** 함께 사용

- 각 과일의 이름과 단가, 각 과일의 개수를 인자값으로 받아 과일 구매 총 비용을 계산, 출력

```
def totalcost(*count, **fruit) :  
    sum = 0                                *parameter1, **parameter2 순서 바뀌면 안됨  
    i = 0  
    for x in fruit :  
        sum += fruit[x]*count[i]  
        print( x, count[i], "개", end = ' ' )  
        i += 1  
    print( " ==> ", sum )
```

```
totalcost( 2, 3, 1, graph = 1000, apple = 700, orange = 600)
```



함수의 결과값으로 여러 개 값 반환

- data structure를 만들어서 반환

```
def min_max( t ) :  
    return min(t), max(t)
```

값1, 값2 → tuple

```
x = min_max( (3,5,2,1,9,10,0,6) )  
print( x )  
print( type( x ) )
```

```
print( min_max( [3,5,2,1,9,10,0,6] ) )  
print( min_max( 3,5,2,1,9,10,0,6 ) )
```

Error !!



함수 내의 변수, parameter의 유효범위

- 변수의 유효범위
 - 변수를 만든 위치에 따라 사용 가능한 문장의 범위가 결정됨
 - 함수 안에서 변수를 만들면, 해당 함수 안에서만 사용 가능

The diagram illustrates variable scope in Python. It shows a global variable `x = 50` and a function `func(x)` that creates a local variable `x = 2`. A call to `func(x)` is shown, and a terminal window displays the execution output. A table at the bottom summarizes the values of `x` in different scopes.

```
x = 50

def func(x):
    print("x is", x)
    x = 2
    print('Changed local x to', x)

func(x)
print('x is still', x)
```

local to **func**

```
>>> =====
>>>
>>> x is 50
>>> Changed local x to 2
>>> x is still 50
>>>
```

x	x in func
50	2



global 문장

```
x = 50
```

```
def func( ) : 함수 바깥에서 만든 x 사용  
    global x
```

```
    print ( 'x is', x )
```

```
    x = 2
```

```
    print ( 'Changed global x to ', x )
```

```
func( )
```

```
print( 'Value of x is', x )
```

```
>>> =====  
>>>  
x is 50  
Changed global x to 2  
Value of x is 2  
>>>
```



Module, library

- Module
 - 유용한 function 들의 정의(def)를 모아 놓은 것
- Library
 - 모듈들의 모임
- 표준 library
 - Python이 기본적으로 제공하는 library
 - <https://docs.python.org/ko/3/library/index.html>

Tip: 나만의 module 만드는 법

내가 만든 function 정의들을 하나의 .py 파일에 모아 놓으면 module



표준 library에 있는 함수 호출 방법

```
> > > import math
```

```
import module명
```

```
> > > math.sqrt(25)
```

```
module명.함수명(인자값)
```

module명 대신 library명 사용해도 됨



함수(Function)

- 함수 만들기
 - 기본 인자값
 - 인자 지정 호출
 - 가변 인자 개수
- 함수 parameter
- return 문장
 - 여러 개 값 반환
- 함수 내 변수의 유효범위
 - global 문장
- Module
- 표준 library
- 함수 호출 방법들

