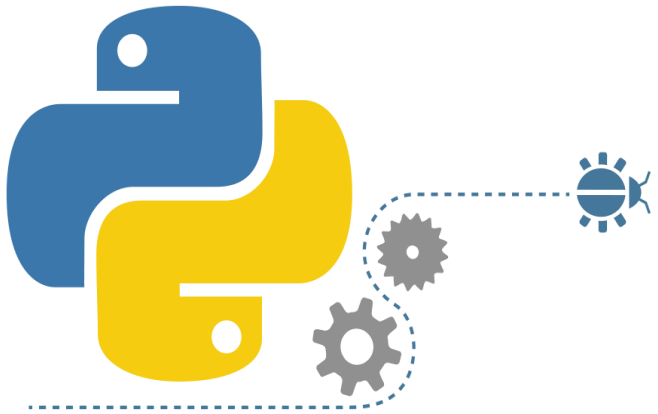


응용프로그래밍

python

수업내용

- 함수의 의미
- 함수의 사용이유
- 함수의 종류
- 함수의 동작 절차
- 함수 사용하기
- 전역변수와 지역변수
- 함수의 매개변수 및 반환값
- 함수에 시퀀스 객체 전달하기

python

함수의 의미

함수의 의미

프로그램

(예) 서든 어택 게임 → 하나의 "프로그램"

함수

함수

함수

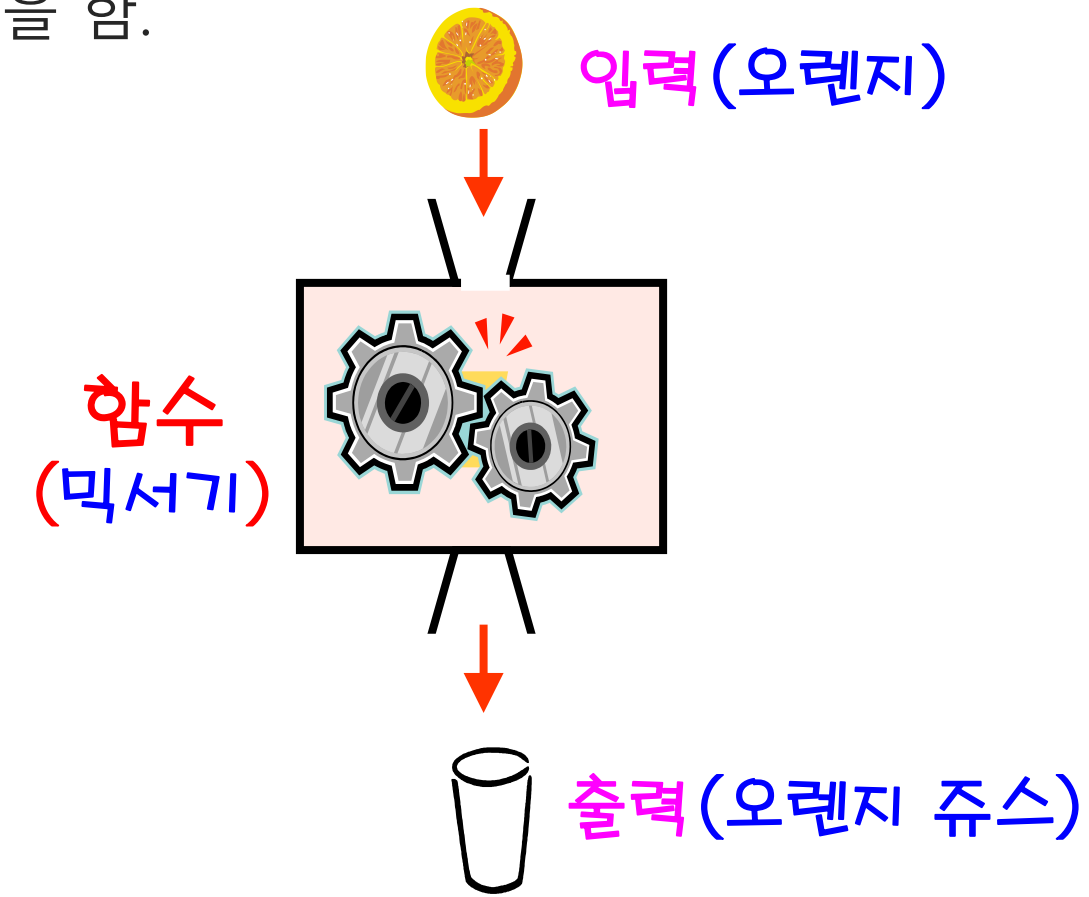
(예) "총 쏘는 장면" → 하나의 "함수"

(예) "폭파되는 장면" → 하나의 "함수"

(예) "칼 휘두르는 장면" → 하나의 "함수"

함수의 의미

- **'특정 기능을 수행'하도록 미리 만들어 놓은 '프로그램'을 "함수(Function)"**라고 함.
  - ↳ 일반적인 "**함수**"는 '**입력값**'과 '**출력값**'이 존재함.
  - ↳ 마치 "**함수**"는 '**믹서기**'와도 같은 역할을 함.



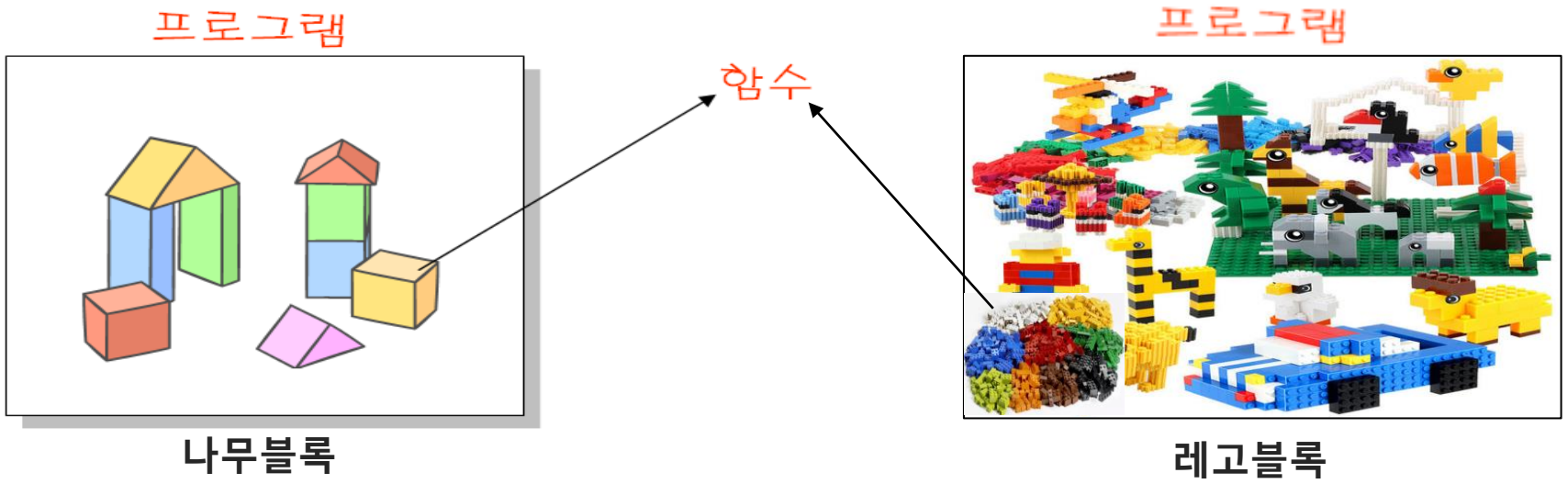
python

함수의 사용이유

함수의 사용이유

- '한번 작성된 함수'는 여러 번 재사용할 수 있음.
  - ↳ 따라서, 코드가 중복되는 것을 막을 수 있음.

- '전체 프로그램'을 작은 모듈(module)로 나눌 수 있음.
  - ↳ 개발 과정이 쉬워짐.
  - ↳ 보다 체계적으로 되어 유지보수가 쉬워짐.
  - ↳ 가독성이 높아져, '프로그램의 흐름을 쉽게 파악'할 수 있음.

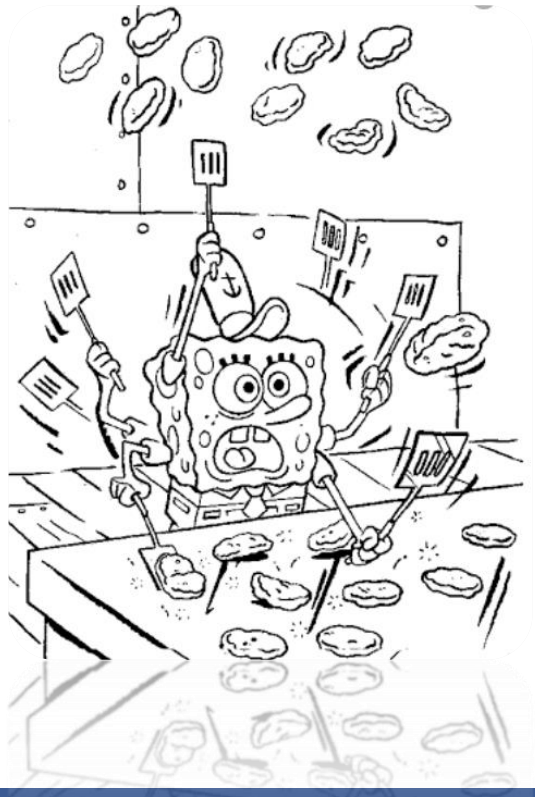


python

함수의 종류

함수의 종류

- **내장함수** : **파이썬 설치 후 바로 사용가능한 함수.**
  - **print( )와 같은 많은 내장 함수**를 가지고 있음.
  - 많은 프로그램에서 **유용함이 입증**되었음.
- **외장함수** : **전 세계 파이썬 사용자들이 만든 파이썬 라이브러리.**
  - **Import하여 사용** 가능.
  - 파이썬 설치 시 **자동으로 컴퓨터에 설치**됨.
- **사용자 정의 함수** : **사용자에 의해 정의된("user-defined") 함수.**
  - 코딩의 세계에는 **수억개, 수조개의 함수**가 있음.
  - 현존하는 함수의 수는 절대 셀 수 없음.
    (지금도 누군가는 열심히 함수를 만들어내고 있기 때문)

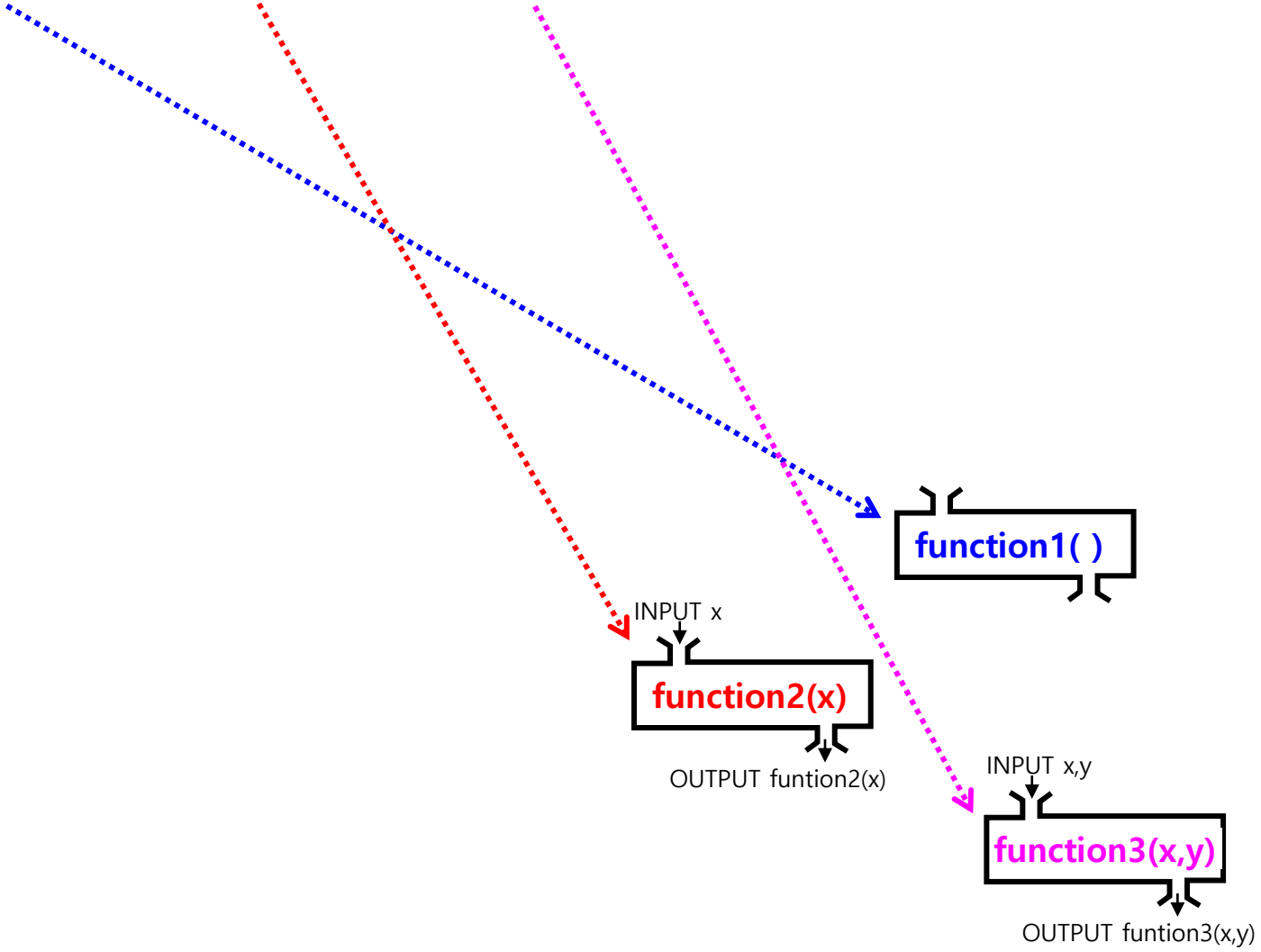


python

함수의 동작 절차

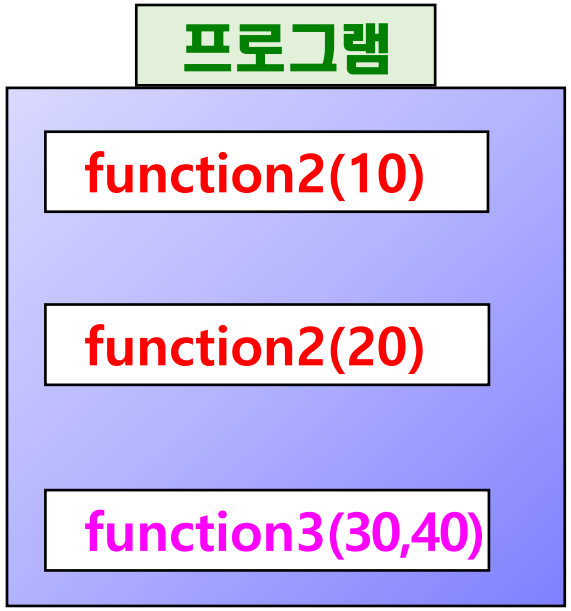
함수의 동작 절차

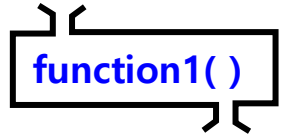
- 아래 그림과 같이 function1, function2, function3는 정의되어 있다고 가정.



함수의 동작 절차

- 아래 그림과 같이 function1, function2, function3는 정의되어 있다고 가정.
  - function1은 정의되어 있지만 호출되지 않음.



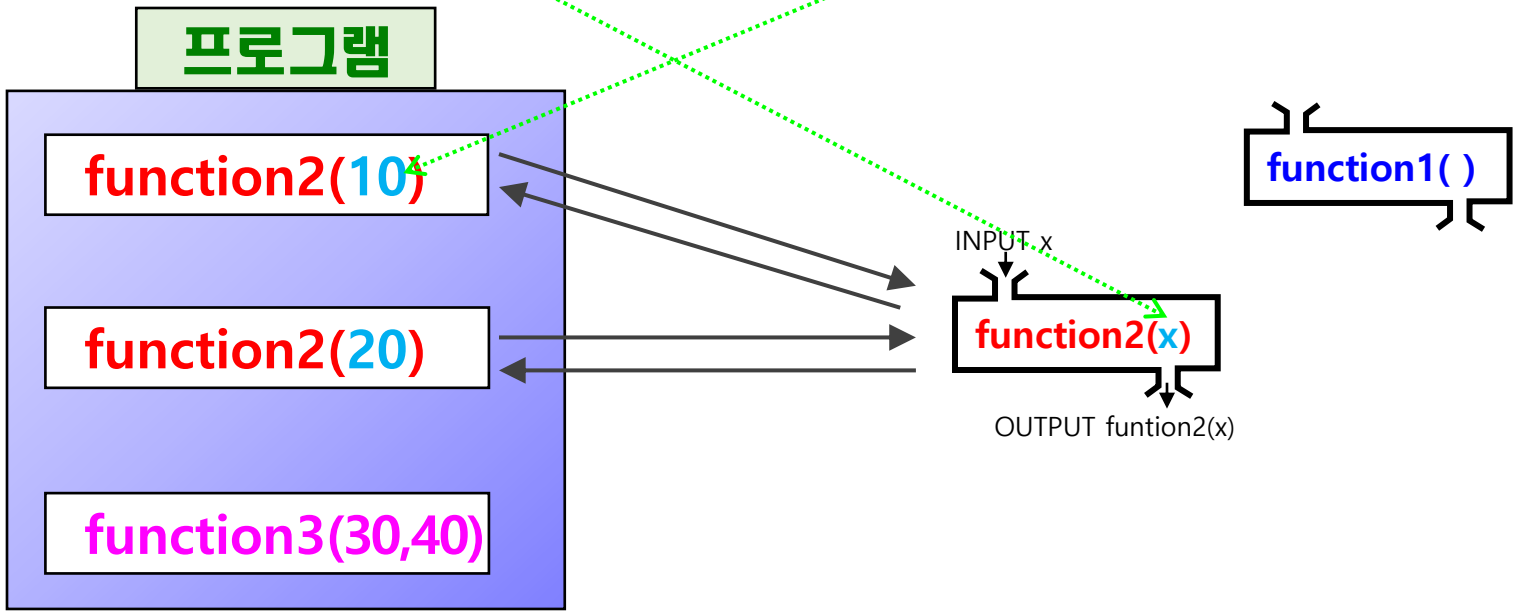


함수의 동작 절차

- 아래 그림과 같이 **function1**, **function2**, **function3**는 정의되어 있다고 가정.
  - **function1**은 정의되어 있지만 호출되지 않음.
  - **function2**는 **매개변수** x에 **인수로** 10과 20이라는 다른 값을 사용해서 **2번 호출**됨.

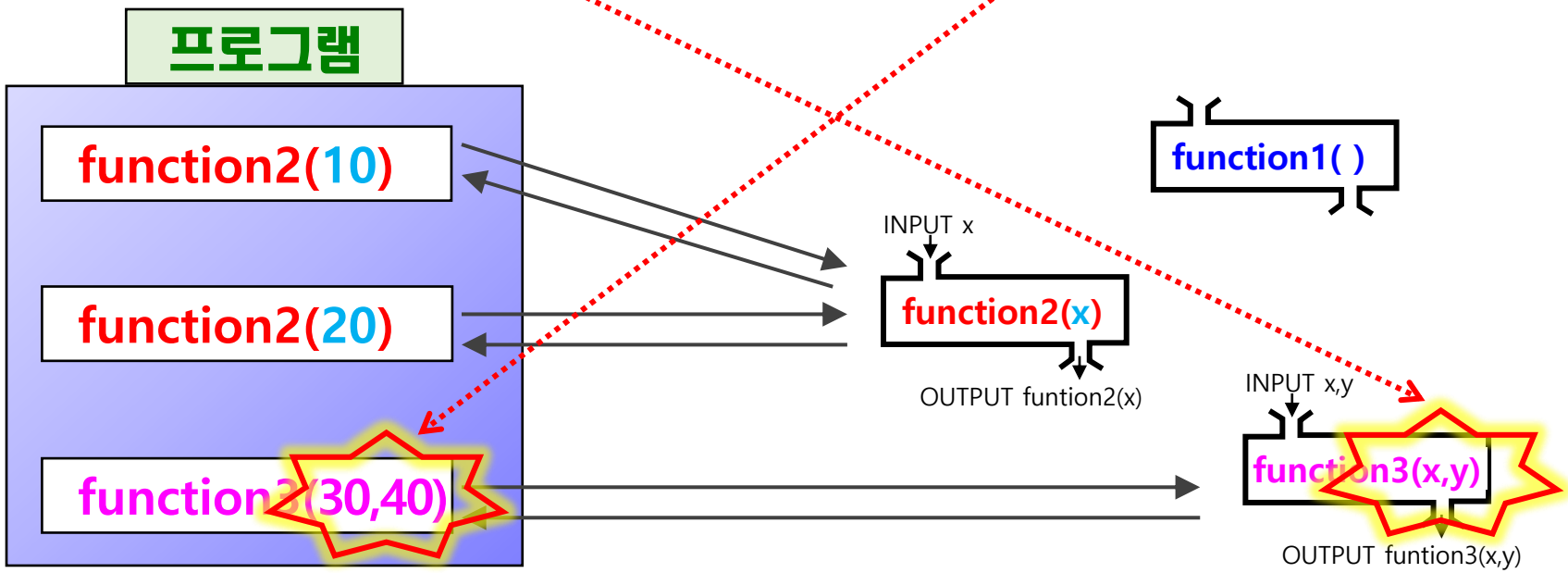
▶매개변수(parameter):
함수 정의 시 사용되는 변수!

▶인수(argument):
함수 호출 시 사용되는 값!



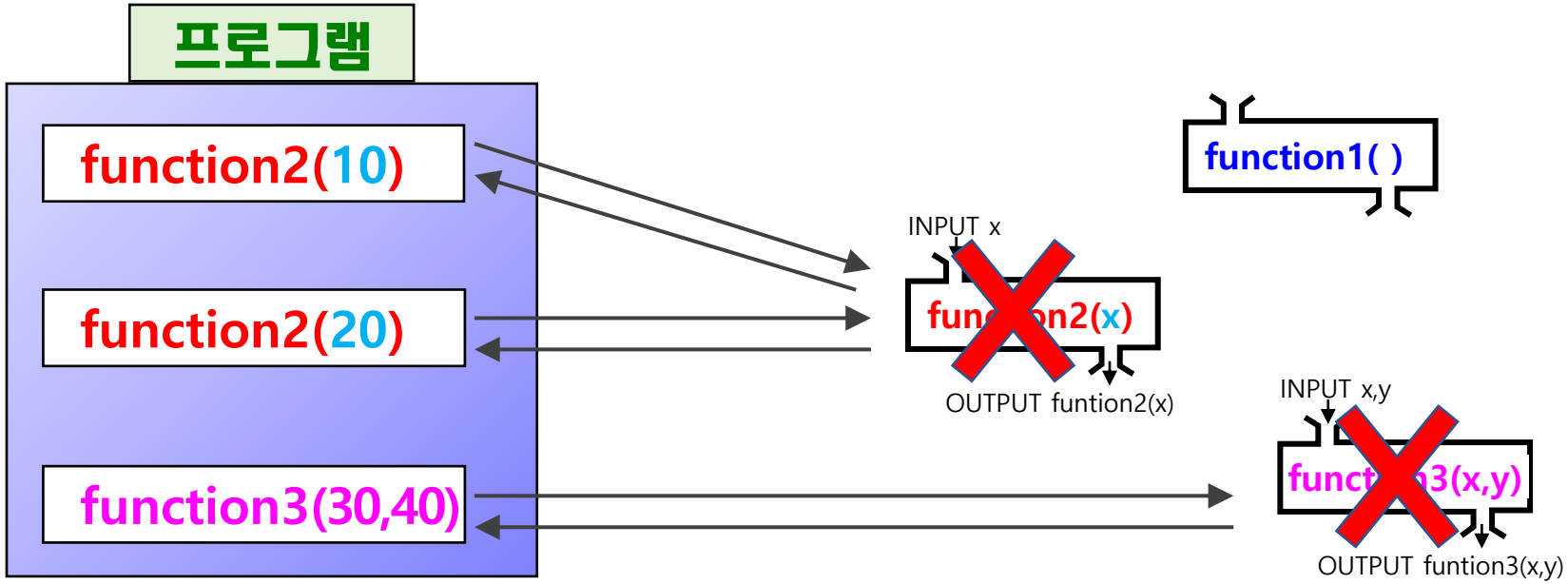
함수의 동작 절차

- 아래 그림과 같이 **function1**, **function2**, **function3**는 정의되어 있다고 가정.
  - **function1**은 정의되어 있지만 호출되지 않음.
  - **function2**는 **매개변수 x**에 **인수로 10**과 **20**이라는 다른 값을 사용해서 **2번 호출**됨.
  - **function3**은 **2개의 매개변수**를 가지고 있는데, **호출할 때 인수도 반드시 2개를 전달**해야만 함.



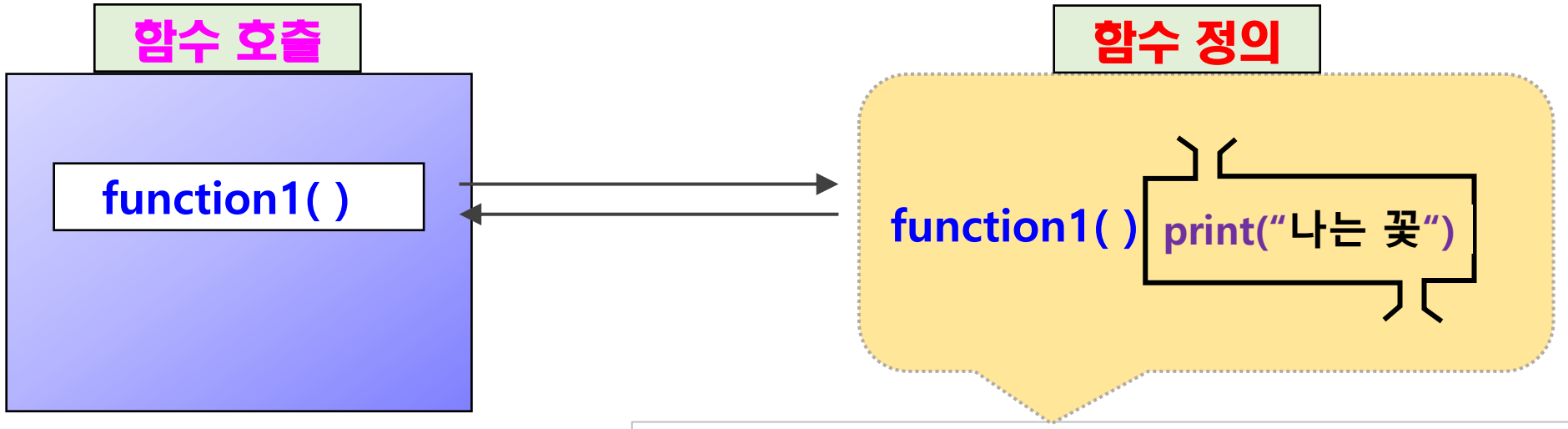
함수의 동작 절차

- 아래 그림과 같이 function1, function2, function3는 정의되어 있다고 가정.
  - function1은 정의되어 있지만 호출되지 않음.
  - function2는 **매개변수** x에 **인수로** 10과 20이라는 다른 값을 사용해서 **2번 호출**됨.
  - function3은 **2개의 매개변수**를 가지고 있는데, **호출할 때 인수도 반드시 2개를 전달**해야만 함.
- 함수는 **그 임무를 수행하라**는 **"호출(요청)"이 있을 때에만 "활성화"**됨.
  - ↳ **임무가 완료**되면 (메모리 및 리소스가 해제되어) "**못쓰게 됨**"



Example

- 네가 나의 이름을 불러줄 때... **나는 꽃**이 되리니...




```
>>> def  function1( ) :          # 함수 정의
    print("나는 꽃")

>>>function1( )                  # 함수 호출
나는 꽃
```

python

함수 사용하기

- **내장 함수**
  - 파이썬 인터프리터가 내장함수를 제공한다.
- **외장 함수**
  - Import로 사용되는 함수
- **사용자 정의 함수**
  - 자신만의 함수를 작성할 수 있다.

내장 함수들

가천대학교 Gachon University

- **파이썬 배포본에 함께 들어있는 활용빈도가 높은 내장함수들**

abs( )	all( )	any( )	ascii( )	bin( )
bool( )	breakpoint( )	bytearray( )	callable( )	chr( )
classmethod( )	compile( )	complex( )	delattr( )	dict( )
dir( )	divmod( )	enumerate( )	eval( )	exec( )
filter( )	float( )	format( )	frozenset( )	getattr( )
globals( )	hasattr( )	hash( )	help( )	id( )
input( )	int( )	isinstance( )	issubclass( )	iter( )
len( )	list( )	locals( )	**max( )**	map( )
memoryview( )	min( )	next( )	object( )	oct( )
ord( )	open( )	**pow( )**	print( )	repr( )
reversed( )	round( )	set( )	setattr( )	slice( )
sorted( )	staticmethod( )	str( )	sum( )	super( )
tuple( )	type( )	vars( )	zip( )	__import__( )

Example

- **abs( )** : 숫자의 **절대값을 반환**함.
- **max( )** : 두 개 또는 더 많은 매개변수 중에서 **가장 큰 값을 반환**함.
- **pow(x, y)** : **x의 y제곱(x^y)한 결과값을 반환**함. (**x**y** 와 동일)

```
>>> abs(-14)                          # 절대값 구하기
14
>>> max(70, 85, 57, 99, 88)           # 최대값 구하기
99
>>> pow(2, 3)                         # 2^3 결과값 구하기 (2**3과 같음)
8
```

외장 함수

- **전세계 파이썬 사용자들이 만든 유용한 프로그램을 모아 놓은 라이브러리**
  - ↳▶ **import를 사용**하여 **모듈**(또는 **라이브러리**)을 불러옴.
  - ↳▶ **가져온 모듈을 사용**할 때는 **모듈이름.함수이름( )**으로 사용함.

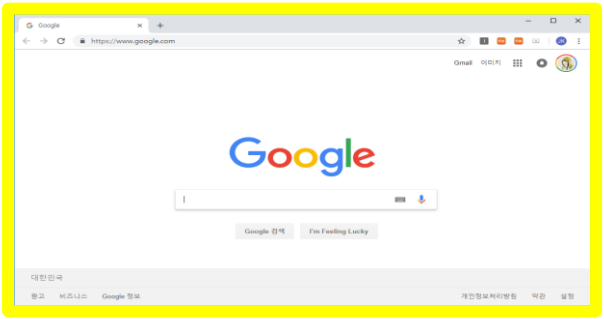
```
>>> import random
>>> dir(random)                                  # random 모듈 내의 '메소드' 와 '변수' 보기
['BPF', 'LOG4', 'NV_MAGICCONST', 'RECIP_BPF', 'Random', ......,
'_BuiltinMethodType', '_MethodType', '_Sequence', '_Set', '__all__'......,
'betavariate', 'choice', 'choices', 'expovariate', 'gammavariate', 'gauss',
'getrandbits', 'getstate', 'lognormvariate'......]

>>> random.random( )                             # 0부터1 사이의 임의의 수
0.910977259164385
```

Example

- random, time, webbrowser 모듈을 import 하고 사용해 보기

```
>>> help('modules')                    # 모듈 종류 보기
......          sys             pickle          os
shutil          glob            tempfile        calendar
random          time            webbrowser      ......
```

```
>>> import  random
>>> random.randint(1,10)              # 1부터10 사이의 임의의 정수 선택
6

>>> import  time
>>> time.ctime( )                     # 현재 '요일 월 일 시간 년' 보기
'Wed Jul 31 10:04:50 2020'

>>> import  webbrowser
>>> webbrowser.open("http://google.com")
```



사용자 정의 함수

▶다음 페이지 참조

1. **키워드 def** : **함수 헤더의 시작을 표시**한다.
2. **함수이름** : **함수를 고유하게 식별**한다. 함수 이름은 '**식별자와 동일한 규칙**'을 따른다.
3. **매개변수(parameter)** : **매개변수를 통해 함수에 값을 전달**한다. ***선택사항***
4. **콜론 (:)** : **함수 헤더의 끝을 표시**한다.
5. **함수 본문** : **작업을 수행하는 문장**. 동일한 들여쓰기(**대개 4칸**)를 가져야 한다.
6. **리턴문** : **호출자에게 값을 반환**한다. ***선택사항***
7. **함수호출** : **인수를 포함하여 함수를 호출**한다.

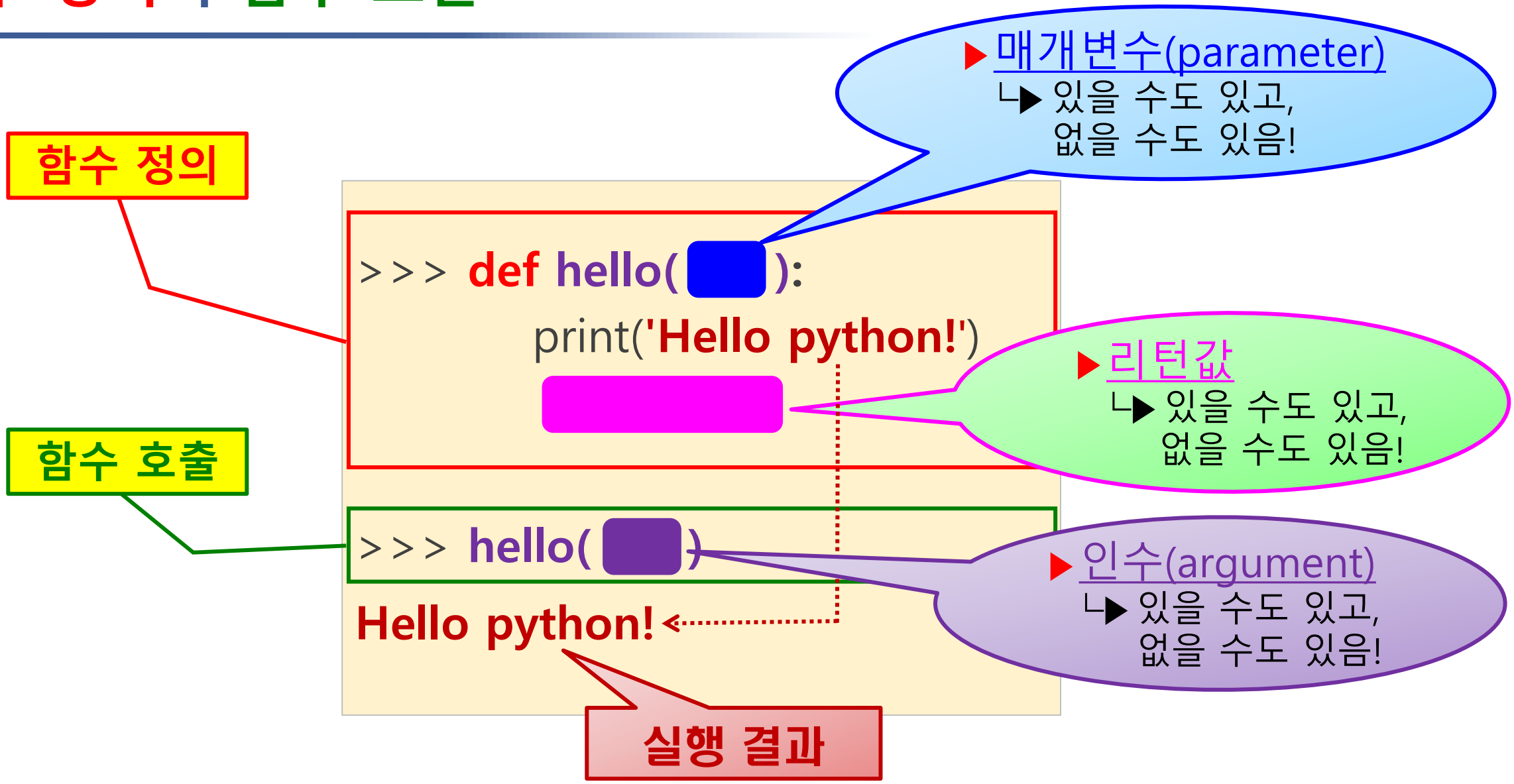
```
def hello(x) :
    y = 10 * x + 2
    return y

hello(10)
```

#102 (함수 실행결과)

식별자 작성 규칙(9가지)

- **식별자는 소문자(a ~ z), 대문자(A ~ Z), 숫자(0 ~ 9), 밑줄 (_)의 조합입니다.**
  - myClass, var_1, print_this_to_screen : 모두 유효한 예
- **식별자는 숫자로 시작될 수 없습니다.**
  - **1**variable : 불가, variable**1** : 가능
- **키워드는 식별자로 사용할 수 없습니다.**
- **특수 문자(!, @, #, $, % 등)는 식별자로 사용할 수 없습니다.**
- **공백(space)은 사용할 수 없습니다.**
- **대소문자를 구별합니다.**
  - **V**ariable과 **v**ariable는 전혀 다른 식별자
- **식별자의 길이는 제한이 없습니다.**
- **의미 있는 식별자의 이름을 지정하십시오***(권장 사항)*.
- **한글, 한자 등도 가능하지만 지양해 주십시요***(권장 사항)*.

함수 정의와 함수 호출


함수 정의
함수 호출
>>> def hello( ):
print('Hello python!')
>>> hello( )
Hello python!
실행 결과
매개변수(parameter)
있을 수도 있고,
없을 수도 있음!
리턴값
있을 수도 있고,
없을 수도 있음!
인수(argument)
있을 수도 있고,
없을 수도 있음!

매개변수가 없는 함수

(1) 매개변수가 없고, 리턴값도 없는 경우

```
>>> def function1():
        print('난, 함수야 안녕?')

>>> function1()
난, 함수야 안녕?
```

(2) 매개변수가 없고, 리턴값이 있는 경우

```
>>> def function2():
        print('나 불렀니?')
        return '끝'

>>> function2()
나 불렀니?
'끝'
```

매개변수가 있는 함수

가천대학교 Gachon University

(3) 매개변수가 있고, 리턴값이 없는 경우

```
>>> def function3(name):
        print('안녕하세요? ' + name + '님')

>>> function3('김가천')
안녕하세요? 김가천님

>>> function3('홍길동')
안녕하세요? 홍길동님
```

(4) 매개변수가 있고, 리턴값도 있는 경우

```
>>> def function4(x, y):
        sum = x + y
        return sum

>>> function4(13, 2)
15
```

매개변수가 있고, 리턴값이 여러 개인 함수

가천대학교 Gachon University

- 리턴값이 여러 개인 경우, 하나의 "튜플"을 생성하여 반환

```
>>> def three(x):
        print('3개의 값 리턴하기')
        return  x,  x*2,  x*3

>>> three(5)
3개의 값 리턴하기
(5, 10, 15)
```

매개변수 1개

리턴값 3개

튜플 1개

```
>>> def three(x, y, z):
        sum = x + y + z
        mul = x * y * z
        div = x / y
        return  sum, mul, div

>>> three(5, 2, 4)
(11, 40, 2.5)
```

매개변수 3개

리턴값 3개

튜플 1개

Example & Solution

잠시 영상을 멈추고 다음 예제를 풀어 보시기 바랍니다.

- **덧셈 함수** : **func_add( )**를 만드시오**(매개변수 2개).** **(파일명: function_add.py)**
- **뺄셈 함수** : **func_sub( )**를 만드시오**(매개변수 2개).**
- 다음과 같이 **'2개의 인수'를 보내서 "함수를 호출"**한 다음, **실행결과를 화면에 출력**하시오**(실행결과 참조).**
  - **func_add(10, 20)**
  - **func_sub(10, 20)**

Result>
덧셈 함수 호출 결과 = 30
뺄셈 함수 호출 결과 = -10

function_add.py

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

def func_sub(x, y):
    return x-y

print("덧셈 함수 호출 결과 = ", func_add(10, 20))
print("뺄셈 함수 호출 결과 = ", func_sub(10, 20))
```

Exercise1 & Solution

잠시 영상을 멈추고 다음 연습문제를 풀어 보시기 바랍니다.

- **곱셈 함수** : **func_mul( )**를 만드시오**(매개변수 2개).** **(파일명: function_4.py)**
- **나눗셈 함수** : **func_div( )**를 만드시오**(매개변수 2개).**
- 다음과 같이 **'2개의 인수'**를 보내서 **"함수를 호출"**한 다음, **실행결과를 화면에 출력**하시오**(실행결과 참조).**
  - **func_mul(30, 40)**
  - **func_div(30, 40)**

Result>
곱셈 함수 호출 결과 = 1200
나눗셈 함수 호출 결과 = 0.75

function_4.py

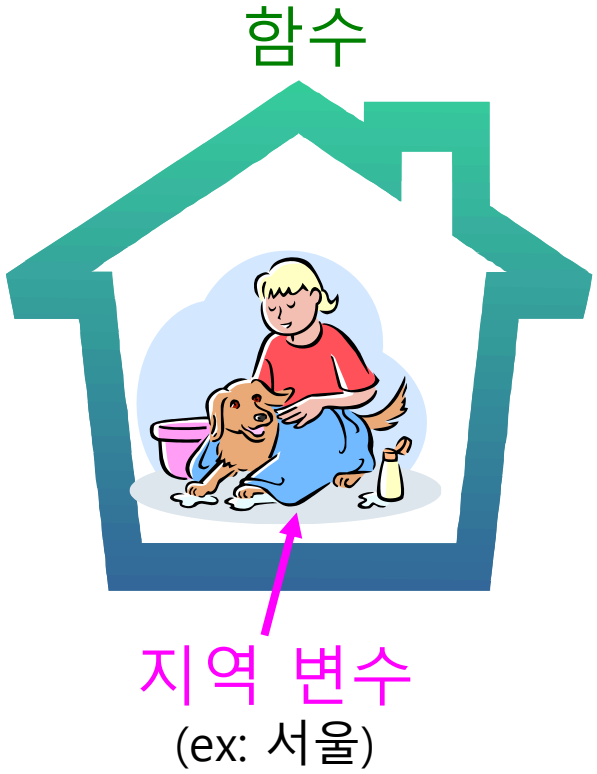
```
def func_mul(x, y):
    return x*y

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

print("곱셈 함수 호출 결과 = ", func_mul(30, 40))

print("나눗셈 함수 호출 결과 = ", func_div(30, 40))
```

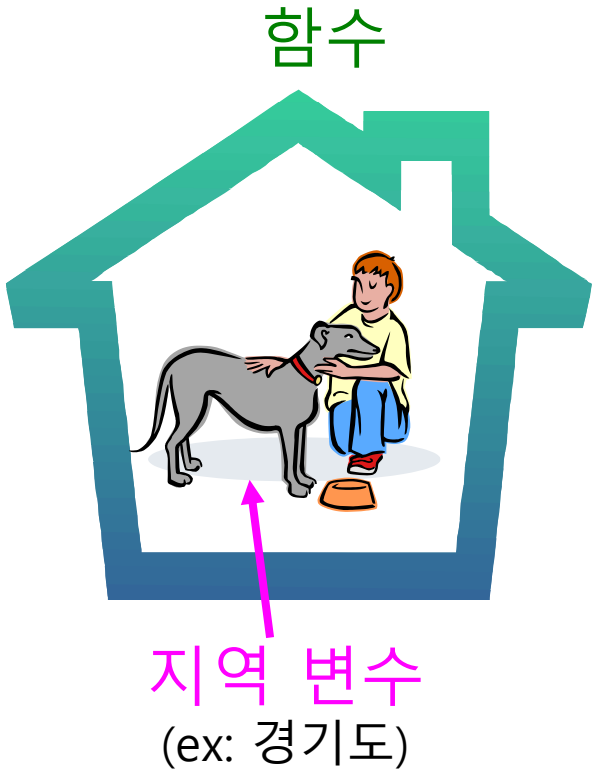
python

전역 변수와 지역 변수

- **전역 변수**
  - 전역 범위를 갖는 변수
- **지역 변수**
  - 지역 범위를 갖는 변수

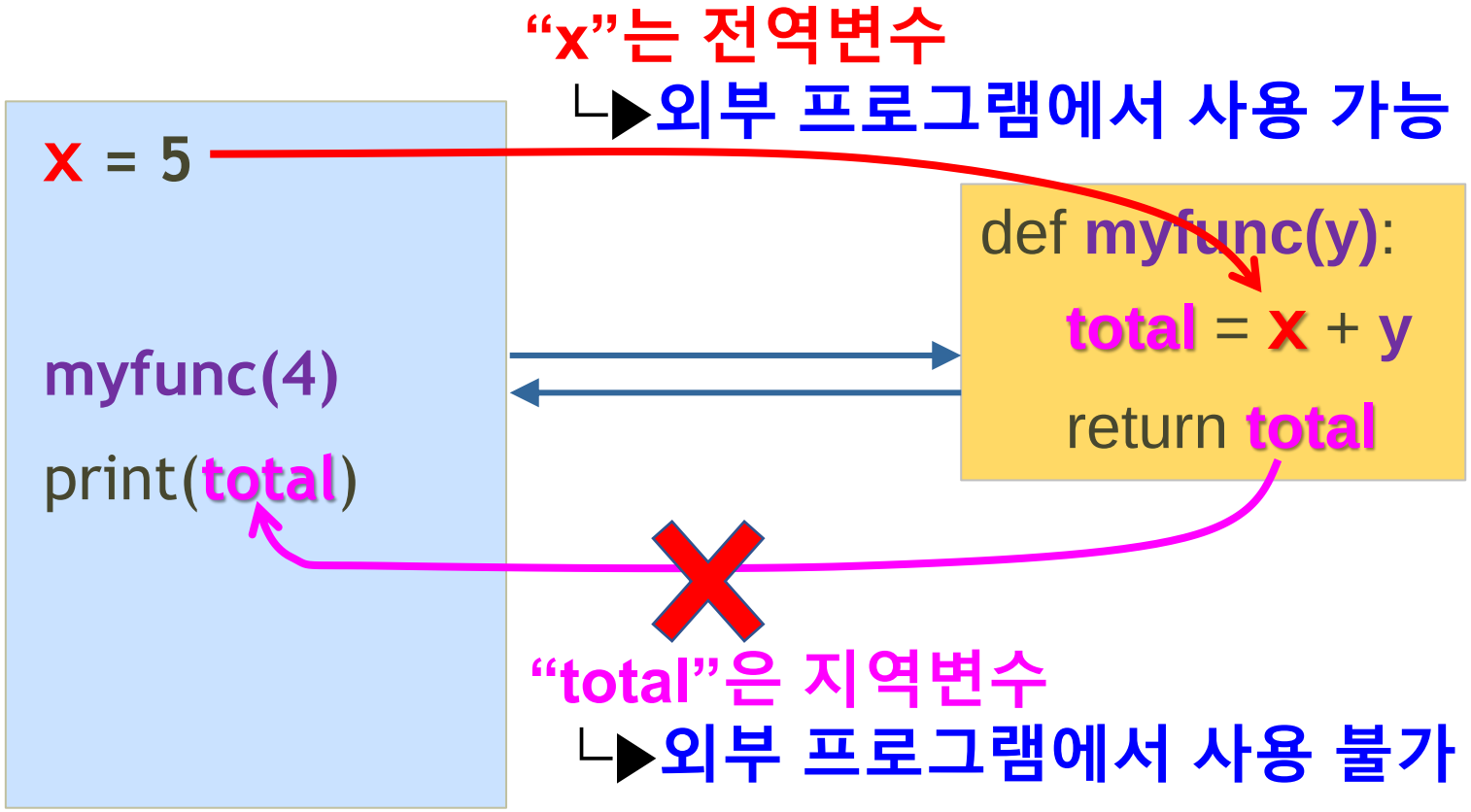
전역 변수와 지역 변수





전역 변수와 지역 변수

- 전역 변수(Global variables)는 함수 외부에서 정의되며 전역 범위를 가짐.
- 지역 변수(Local variables)는 함수 내부에서 정의되며 지역 범위를 가짐.



전역 변수와 지역 변수

가천대학교 Gachon University

- **전역 변수**와 **지역 변수** 사용하기

```
>>> x = 5                          # "x"는 전역변수
>>> def myfunc( y ):
        total = x + y              # "total"은 지역변수
        return total

>>> myfunc( 4 )
9
>>> print(x)
5
>>> print(total)
Traceback (most recent call last):
  File "<pyshell#278>", line 1, in <module>
    total
NameError: name 'total' is not defined
```

"total"은 지역변수
└▶"myfunc( )" 내부에서만 사용 가능

Exercise2 & Solution

잠시 영상을 멈추고 다음 연습문제를 풀어 보시기 바랍니다.

- **반지름을 구하는 함수 func_circum( )**를 만드시오(매개변수: radius).
  - ↳ **def func_circum(radius):**
- 전달받은 매개변수를 사용하여 **result**를 계산하시오. 공식은 **2πr**이다.
  - ↳ **result** = 2*3.14***radius**
- 소수점 이하 둘째자리까지 출력하도록 하시오.
  - ↳ print("원의 둘레는 **%5.2f** 입니다." **%result**)
- **반지름을 입력** 받아 변수 **radius**에 저장하시오.
  - ↳ **radius** = int(input("원의 반지름을 입력하세요: "))
- 매개변수를 **radius**로 하여 **func_circum( )를 호출**하시오.
- **다음 결과와 같이 인쇄**하시오.
  - ↳ **func_circum(radius)**

▶ **%** : 형식지정자
  ↳▶ **5** : 전체 자릿수(소수점 포함)
  2 : 소수점 이하 자릿수
  f : floating point(실수)

Result>
원의 반지름을 입력하세요: 5
원의 둘레는 31.40입니다.

Exercise2 & Solution1

circumference.py

```
def func_circum(radius):
    result = 2*3.14*radius
    print("원의 둘레는 %5.2f 입니다." %result)

radius = int(input("원의 반지름을 입력하세요: "))
func_circum(radius)
```

▶ % : 형식지정자
  ㄴ▶ 5 : 전체 자릿수(소수점 포함)
  2 : 소수점 이하 자릿수
  f : floating point(실수)

Result>
원의 반지름을 입력하세요: 5
원의 둘레는 31.40입니다.

Exercise2 & Solution2

circumference2.py

```
def func_circum(radius):
    result = 2*3.14*radius
    return result

radius = int(input("원의 반지름을 입력하세요: "))
result = func_circum(radius)
print("원의 둘레는 {0:%5.2f}입니다." .format(result))
```

▶ 이 부분만 조금 차이 남 !

▶ ❶ 결과를 리턴한 다음, ❷ 출력 !

Result>
원의 반지름을 입력하세요: 5
원의 둘레는 31.40입니다.

python

함수의 매개변수 및 반환값

기본 매개변수

- **함수 정의** 시, **기본값이 지정된 매개변수가 포함**될 수 있음.
  ↳▶ 이를 "**기본 매개변수**"라 함.

- **매개변수와 일치하는 인수 없이 함수가 호출되면** "**기본 매개변수**"가 사용됨.

- "**기본 매개변수**"는 **함수의 매개변수 목록에서 마지막부터 지정**해 주어야 함.

기본 매개변수

가천대학교 Gachon University

- 다음 "**default**" 함수에서 "**country 매개변수**"**의 기본값**은 "**한국**"임.

```
>>> def default(country = '한국'):
        print("나는 " + country + "에서 왔어요.")

>>> default('미국')
                                # 나는 미국에서 왔어요.
>>> default('브라질')
                                # 나는 브라질에서 왔어요.
>>> default( )
                                # 나는 한국에서 왔어요.
```

함수 호출 시, "**default( )**"에는 **인수**가 없으므로
기본값이 있는 "기본 매개변수"가 사용된다.

Example

함수"**default_score**" 에서, **매개변수 "mat"은 기본값 50**을 가진다. **(파일명: default.py)**

잠시 영상을 멈추고 입력해 주시기 바랍니다.

```
def default_score(name, kor, eng, mat=50):
    print('이름 : ', name)
    print('국어 : ', kor)
    print('영어 : ', eng)
    print('수학 : ', mat)

default_score('김가천', 90, 80, 70)
print( )                    #한 줄을 띄워주는 역할
default_score('홍길동', 85, 95 )
```

Result>

```
이름 :  김가천
국어 :  90
영어 :  80
수학 :  70

이름 :  홍길동
국어 :  85
영어 :  95
수학 :  50
```

*args Argument

가천대학교 Gachon University

- ***args**는 **함수 정의에서 '매개변수'로 사용**됨.
  - ↳ **args**는 **arguments**의 약어임

***args**는 **star(*) arguments**로 읽음

- **함수에 인수를 전달**하기 위해 **가변 인수**를 사용함.
  - ↳ **가변 인수**란 **전달되는 인수의 개수가 가변적**이라는 것을 의미함.

***args**는 **함수에 전달되는 "인수의 개수"가 확실하지 않은 경우**에 사용

  - ↳ **여러 개의 인수**를 받은 경우, 함수 내부에서는 **튜플(tuple)로 받은 것처럼 인식**함

**kwargs Argument

가천대학교 Gachon University

- **kwargs는 함수 정의에서 '키워드 매개변수'로 사용됨.
  - ↳▶ kwargs는 keyword arguments의 약어임.

**kwargs는 double star(**) keyword arguments로 읽음

- 매개 변수 kwargs는 딕셔너리 형태를 가짐.
  - ↳▶ 'key = value' 형태의 모든 입력 인수가 해당 딕셔너리에 저장됨.

- **kwargs는 가변적으로 인수들을 처리할 수 있음.
  - ↳▶ 함수 호출 시, 인수는 (키워드 = 특정 값) 형태를 취함.
  - ↳▶ 그것은 딕셔너리 형태로 {'키워드' : '특정 값'} 형태로 함수 내부 전달됨.

Example

- 전달받은 가변적인 인수를 출력하는 keyword( ) 함수를 작성하시오.

(파일명: function_kwarg.py)

잠시 영상을 멈추고 입력해 주시기 바랍니다.

function_kwarg.py

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

keyword( a=1 )
keyword( name='foo', age=3, addr='seoul' )
```

▶(키워드 = 특정 값) 형태로 keyword 함수를 호출!

Result>

```
{ 'a': 1 }
{ 'name': 'foo',  'age': 3,  'addr': 'seoul' }
```

▶{'키워드' : '특정 값'} 딕셔너리 형태로 출력됨!

python

함수에 시퀀스 객체 전달하기

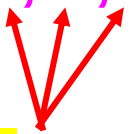
Unpacking a List

▶분해, 묶음 풀기

- list를 언패킹(unpacking)하려면, 리스트명 앞에 '*'를 붙임.

↳ function(*listname)

unpacking(a, b, c)



unpacking

unpacking(*mylist)

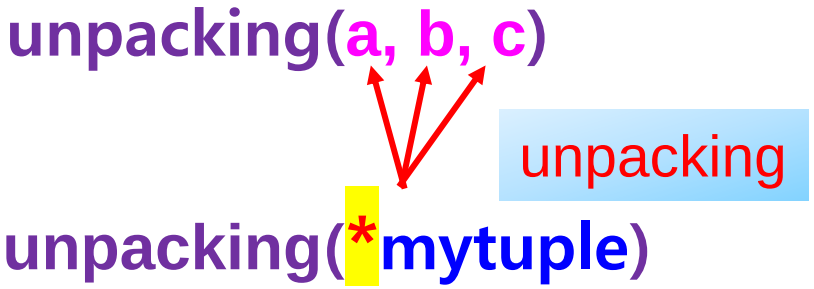
```
>>> def unpacking(a, b, c):
        print(a)
        print(b)
        print(c)
>>> mylist=[10, 20, 30]
>>> unpacking(*mylist)
```

Result>
10
20
30

Unpacking a Tuple

- **tuple을 언패킹(unpacking)**하려면, **튜플명 앞에 '*'를 붙임**.

↳ **function(*tuplename)**




```
>>> def unpacking(a, b, c):
        print(a)
        print(b)
        print(c)
>>> mytuple=('John', 'Rosa', 'Danny')
>>> unpacking(*mytuple)
```

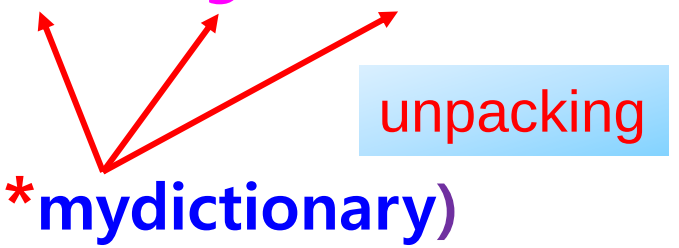
Result>
John
Rosa
Danny

Unpacking a Dictionary (2-1)

- **dictionary를 언패킹(unpacking)**하려면, **딕셔너리명 앞에 '**'를 붙임.**

  ↳ **function(**dictionaryname)**

unpacking(name, age, address)



unpacking(mydictionary)**

↳ 단, **dictionary의 key**는 **모두 문자열**이어야 함.

```
mydictionary= { 'name': 'kim',  'age': 22,  'address': 'Seoul' }
```

↳ **함수의 매개변수**는 **dictionary의 key**를 가져와서, **key**에 **할당된 value**을 가져옴.

Example

- 전달받은 key에 해당하는 value를 출력하는 unpacking 함수를 작성하시오.

잠시 영상을 멈추고 입력해 주시기 바랍니다.

```
>>> def unpacking(name, age, address):
        print('이름 : ', name)
        print('나이 : ', age)
        print('주소 : ', address)
>>> mydictionary= {'name': 'kim',  'age': 22,  'address': 'Seoul'}
>>> unpacking(**mydictionary)
```

```
이름: Kim
나이: 22
주소: Seoul
```

▶ key인 name, age, address 에 할당된 value인 kim, 22, Seoul 이 각각 출력됨!

Unpacking a Dictionary (2-2)

가천대학교 Gachon University

- "인수의 개수"를 모르는 경우, dictionary 언패킹 방법
  - 함수의 매개변수로 **kwargs를 사용
  - kwargs['key'] 형태로 key에 할당된 value를 가져옴

잠시 영상을 멈추고 입력해 주시기 바랍니다.

```
>>> def unpacking(**kwargs):
    print("학생 이름 : ", kwargs['name'])
    print("학생 폰번호 : ", kwargs['phone'])
    print("학생 주소 : ", kwargs['address'])
>>> mydict = {'name':'Sam', 'section':'A' , 'address':'London' , 'phone':'112' }
>>> unpacking(**mydict)
```

Result>

```
학생 이름 : Sam
학생 폰 번호 : 112
학생 주소 : London
```

▶ key인 name, phone, address 에 할당된 value인 Sam, 112, London 이 각각 출력됨!

Exercise3 & Solution

잠시 영상을 멈추고 다음 연습문제를 풀어 보시기 바랍니다.

- **두 개의 숫자를 받아들여 x, y에 저장**하시오.
  - `x = int(input('첫 번째 숫자를 입력하세요 : '))`
  - `y = int(input('두 번째 숫자를 입력하세요 : '))`

Result>

첫 번째 숫자를 입력하세요 : **77**

두 번째 숫자를 입력하세요 : **5**

몫 : **15**, 나머지 : **2**

- **몫과 나머지를 반환**하는 **함수 quotRemain을 작성**하시오.

```
def quotRemain(a, b):
    return  a // b,  a % b
```

- **a // b** 77 // 5 → 몫 : 15
- **a % b** 77 % 5 → 나머지 : 2

- **다음 결과와 같이 출력**되도록 **프로그램을 작성**하시오.

```
quotient, remain = quotRemain(x, y)
print('몫 : {0}, 나머지 : {1}' .format(quotient, remain))
```

Exercise3 & Solution

quotRemain.py

```
x = int(input('첫 번째 숫자를 입력하세요 : '))
y = int(input('두 번째 숫자를 입력하세요 : '))

def quotRemain(a, b):
    return  a // b,  a % b

quotient, remainder = quotRemain(x, y)
print('몫 : {0}, 나머지 : {1}' .format(quotient, remainder))
```

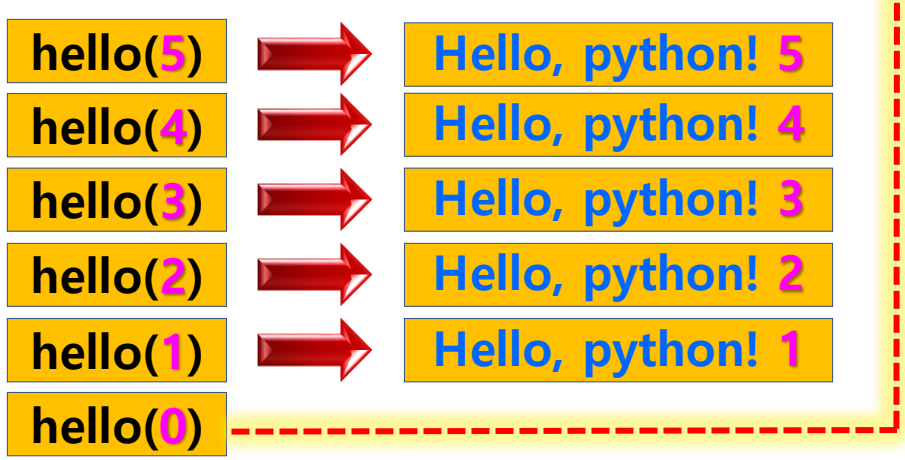
재귀(Recursion)

- 재귀는 '종료 조건'에 도달할 때까지 '자기자신을 계속 호출'함.

function_recursion.py

```
def hello(count):
    if count == 0:
        return
    print('Hello, python!', count)
    count -= 1
    hello(count)

hello(5)
```

▶ count == 0 이 참인 경우 return 이 실행됨!
└▶ return 이 실행되면 프로그램은 종료



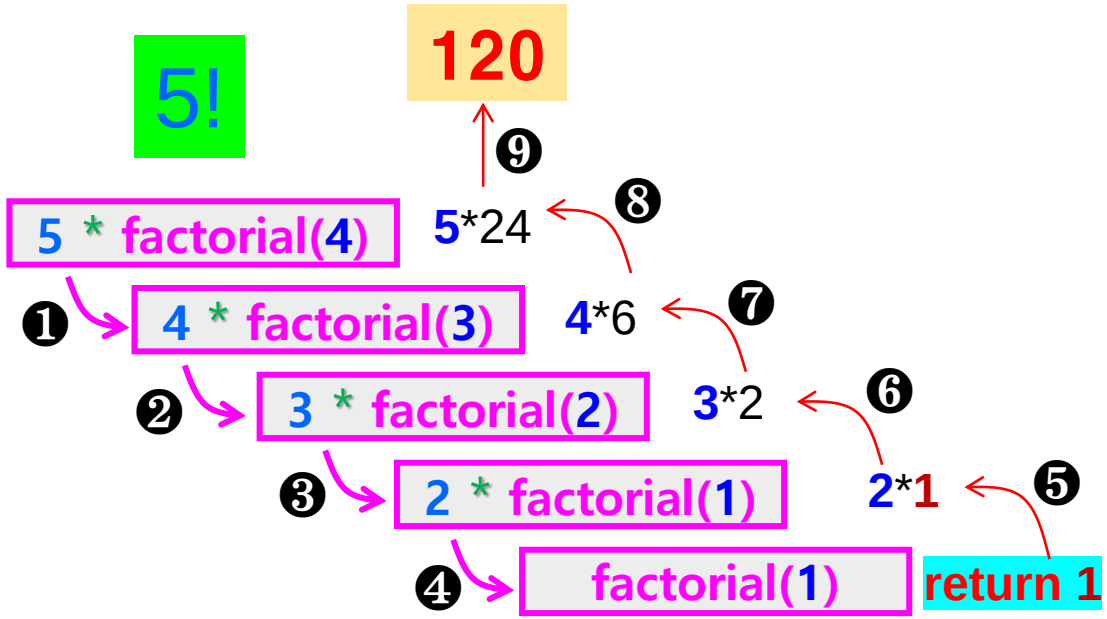
팩토리얼(Factorial)

재귀를 사용하여 팩토리얼(5!)을 계산하시오.

function_factorial.py

```
def factorial(n):
    if n == 1 :
        return 1
    print('n =  ', n)
    return  n * factorial(n-1)

factorial( 5 )
```

n = 5
n = 4
n = 3
n = 2



Q&A