

# 기초빅데이터프로그래밍

## 함수



# 목차

---

- 1 함수란 무엇인가?
- 2 기본 함수(Built-in 함수)
- 3 라이브러리(패키지) 함수
- 4 사용자 정의 함수
- 5 함수의 호출과 흐름
- 6 함수의 인자와 반환값
- 7 함수의 인자 전달과 가변 데이터
- 8 일반 인자, 기본 인자, 키워드 인자, 그리고 가변 인자
- 9 Scoping Rule
- 10 재귀 함수
- 11 람다 함수
- 12 함수를 인자로 전달하기
- 13 요일 구하기 예제를 통한 함수 알아보기
- 14 성적 처리 시스템 구현을 통한 함수 알아보기



# 1. 함수란 무엇인가?

---

- 함수
  - 특정의 기능을 수행하는 코드의 집합
- 파이썬에는 크게 세 가지 종류의 함수가 존재한다.
  - 기본 함수 혹은 **built-in 함수**
    - 기본 함수는 파이썬의 실행과 동시에 사용할 수 있는 함수들이다.
  - 라이브러리 혹은 패키지 함수
    - 해당 라이브러리를 포함한 후에 사용할 수 있다.
  - 사용자 정의 함수
    - 사용자가 자신이 필요로 하는 기능을 수행하는 함수를 작성한 함수

## 2 기본 함수(Built-in 함수)

- `dir(__builtins__)`



```
C:\Windows\system32\cmd.exe - python
>>> dir(__builtins__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '__build_class__', '__debug__', '__doc__', '__import__', '__loader__', '__name__', '__package__', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval', 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']
>>>
```

## • dir 함수

---

dir([대상체])

dir에 어떠한 대상체가 인자로 넘어오는가에 따라서 다양한 결과가 **리스트** 타입으로 산출된다. 여기서 대상체는 객체, 함수, 클래스, 모듈, 패키지등 다양한 요소가 될 수 있다.

**dir():** 현재 파이썬 인터프리터에 포함되어 있는 요소들의 이름을 구한다.

**dir(함수):** 함수가 가진 속성들을 구한다.


**dir(클래스):** 클래스가 가진 속성들을 구한다.

**dir(모듈):** 모듈이 포함하고 있는 요소들의 이름을 구한다.

**dir(패키지):** 패키지가 포함하고 있는 요소들의 이름을 구한다.

---

```
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__']
>>> import os
>>> dir()
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
>>> dir(os)
[...]
>>> dir(abs)
[...]
```



---

- **help 함수**

help(대상체)

대상체에 대한 설명을 표시한다. 대상체에 대해 알고 싶은 바가 있으면 help를 호출해보면 알 수 있다. 여기서 대상체는 객체, 함수, 클래스, 모듈, 패키지등 다양한 요소가 될 수 있다.



---

```
>>> import os
```

```
>>> dir(os)
```

```
['__builtins__', '__doc__', '__name__', '__package__', 'sys']
```

```
>>> help(os)
```

Help on module os:

NAME

os – OS routines for Mac, NT, or Posix depending on what system we're on.

DESCRIPTION

...

```
>>> help(abs)
```

Help on built-in function abs in module builtins:

abs(...)

abs(number) -> number

Return the absolute value of the argument.



## • type 함수

### type(대상체)

대상체에 대한 타입을 구한다. 대상체의 type을 알고 싶으면 type을 호출해보면 알 수 있다.

```
>>> import sys
>>> type(sys)
<class 'module'>
>>> type(dir)
<class 'builtin_function_or_method'>
>>> type(1)
<class 'int'>
>>> type(3.14)
<class 'float'>
>>> type("서강대학교")
<class 'str'>
```

```
In [8]: import sys
        type(sys)
```

```
Out[8]: module
```

```
In [9]: type(dir)
```

```
Out[9]: builtin_function_or_method
```

```
In [10]: type(1)
```

```
Out[10]: int
```

```
In [11]: type(3.14)
```

```
Out[11]: float
```

```
In [12]: type("서강대학교")
```

```
Out[12]: str
```

## • id 함수

---

### id(대상체)

대상체의 식별값을 구한다. 파이썬 인터프리터는 모든 대상체에 식별값을 부여한다. id 함수는 이 식별값을 구해준다.

```
>>> id(1)
505911008
>>> id(3.14)
20130464
>>> message = "Hello World"
>>> id(message)
31623072
```



- **sum 함수**

---

### sum(숫자\_컨테이너)

sum은 숫자 컨테이너 데이터의 합을 구하는 함수이다.

```
>>> sum([1, 2, 3, 4])
```

```
10
```

```
>>> sum([1, 3.14, 5, 7, 9])
```

```
25.14
```



## • sorted 함수

**sorted**(반복\_가능한\_데이터, key=None, reverse = False) -> 정렬된\_리스트

- sorted 함수는 반복\_가능한\_데이터를 입력 받아서 정렬된 리스트로 반환한다.
- key는 정렬한 기준을 넘겨준다. 만약 정렬기준이 없으면 기본 정렬 기준을 이용한다.
- reverse는 역순 정렬 시킬 것인지의 여부를 알려주는 인자이다.

```
>>> numbers = [2, 3, 1, 5, 4]
>>> sorted_numbers = sorted(numbers)
>>> sorted_numbers
[1, 2, 3, 4, 5]
>>> reverse_sorted_numbers = sorted(numbers, reverse=True)
>>> reverse_sorted_numbers
[5, 4, 3, 2, 1]
>>> numbers
[2, 3, 1, 5, 4]
```

---

```
>>> colors = ["red", "green", "blue", "yellow"]
>>> sorted_colors = sorted(colors)
>>> sorted_colors
["blue", "green", "red", "yellow"]
>>> len_sorted_colors = sorted(colors, key=len)
>>> len_sorted_colors
["red", "blue", "green", "yellow"]
>>> reverse_len_sorted_colors = sorted(colors, key=len, reverse=True)
>>> reverse_len_sorted_colors
["yellow", "green", "blue", "red"]
```



### 3 라이브러리(패키지) 함수

---

- 파이썬에서는 함수들을 기능별로 **패키지** 혹은 **모듈**에 포함시켜서 제공한다.
- **모듈**이란 파이썬 코드를 포함하고 있는 파일을 말한다.
- 모듈들을 묶어서 **패키지**라 한다.
- 파이썬을 설치하면 많은 **라이브러리(패키지)**와 모듈들이 설치되고 각 라이브러리 및 모듈마다 많은 수의 함수들이 포함되어 있다.
- 기본 라이브러리에서 제공되지 않는 것은 **추가의 라이브러리**(예를 들자면 **numpy, scipy, matplotlib** 등)를 설치해서 사용할 수 있다.
  - 라이브러리나 모듈 내에 포함된 함수를 사용하기 위해서는 이들을 포함(import) 시켜야 한다.
  - **import** 예약어를 사용한다.

- **import** 라이브러리명\_혹은\_모듈명
  - 라이브러리나 모듈을 포함시킨다.
  - 대상체를 사용할 경우에 라이브러리명\_혹은\_모듈명.대상체 형태로 사용해야 한다.

```
>>> import math
>>> math.sin(math.pi)
1.2246467991473532e-16
>>> math.log(math.e)
1.0
```



- **import** 라이브러리명\_혹은\_모듈명 **as** **축약이름**
  - 라이브러리나 모듈을 축약된 이름으로 바꾸어서 포함시킨다. 이 경우 모듈내의 대상을 사용할 경우에는 축약이름.대상체 형태로 사용해야 한다.

```
>>> import math as ma
>>> ma.sin(ma.pi)
1.2246467991473532e-16
>>> ma.log(ma.e)
1.0
```





- **from** 라이브러명\_혹은\_모듈명 **import** 대상체명
  - 라이브러리나 모듈로부터 몇몇 대상체 만을 포함시킨다.
  - 이 경우 대상체의 사용은 대상체명을 이용해서 사용한다.

```
>>> from math import sin, log, pi, e
```

```
>>> sin(pi)
```

```
1.2246467991473532e-16
```

```
>>> log(e)
```

```
1.0
```



- **from 라이브러리명\_혹은\_모듈명 import \***
  - 라이브러리나 모듈 내에 포함된 모든 대상을 포함시킨다.
  - 이 경우 대상체의 사용은 대상체명을 이용해서 사용한다.

```
>>> from math import *  
>>> sin(pi)  
1.2246467991473532e-16  
>>> log(e)  
1.0
```



---

- **math**

- math 라이브러리(패키지)는 수학 관련 함수 및 속성들을 포함하고 있다.

```
>>> import math
```

```
>>> dir(math)
```

```
[ ...
```

```
...
```

```
]
```



```
>>> from math import sin, cos, pi
```

```
>>> sin
```

```
<built-in function sin>
```

```
>>> sin(0)
```

```
0.0
```

```
>>> sin(pi/2)
```

```
1.0
```

```
>>> cos(0)
```

```
1.0
```

```
>>> cos(pi/2)
```

```
0.0
```

```
In [1]: from math import sin, cos, pi  
sin
```

```
Out[1]: <function math.sin>
```

```
In [2]: sin(0)
```

```
Out[2]: 0.0
```

```
In [3]: sin(pi/2)
```

```
Out[3]: 1.0
```

```
In [4]: cos(0)
```

```
Out[4]: 1.0
```

```
In [5]: cos(pi/2)
```

```
Out[5]: 6.123233995736766e-17
```

- **OS**

- os 라이브러리(패키지)는 운영체제 관련 속성, 함수, 및 클래스들을 포함하고 있다.

```
>>> import os
```

```
>>> dir(os)
```

- **sys**

- sys 라이브러리(패키지)는 파이썬 시스템(파이썬 인터프리터) 관련 속성, 함수, 및 클래스들을 포함하고 있다.

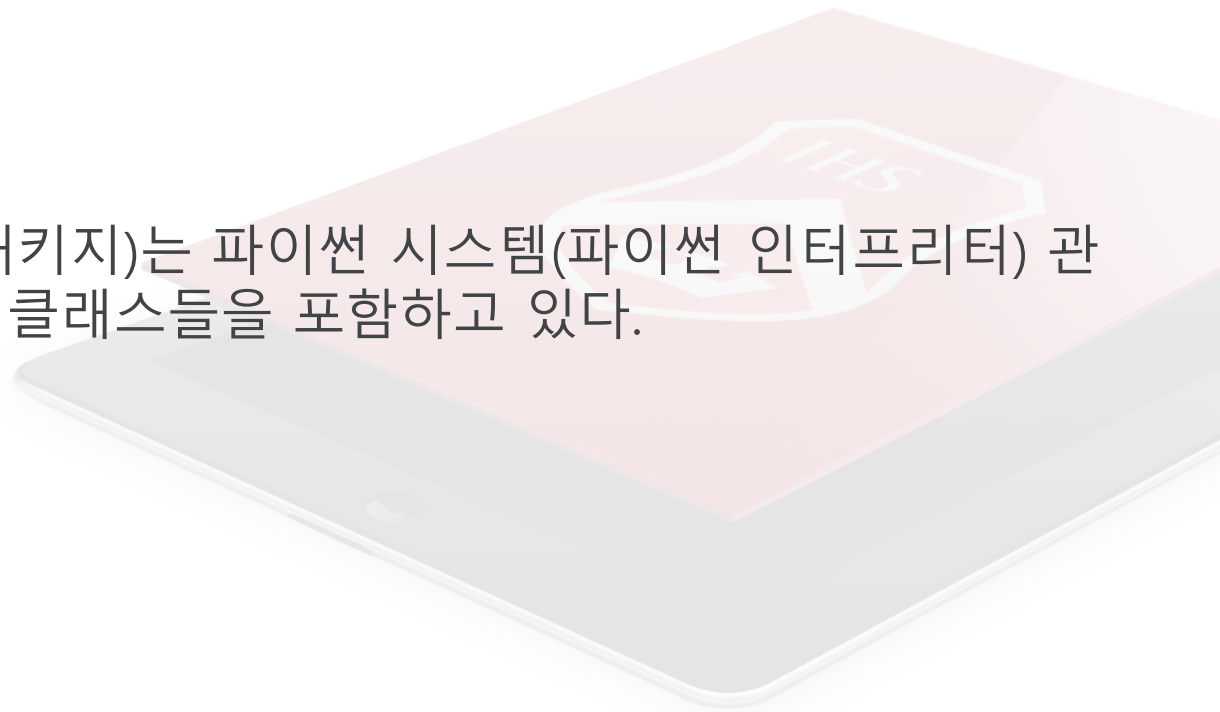
```
>>> import sys
```

```
>>> dir(sys)
```

```
[ ...
```

```
... 
```

```
]
```



```
In [5]: import time  
time.localtime()
```

```
Out[5]: time.struct_time(tm_year=2019, tm_mon=3, tm_mday=12, tm_hour=20, tm_min=35,  
tm_sec=59, tm_wday=1, tm_yday=71, tm_isdst=0)
```

```
In [6]: import calendar as ca  
ca.prmonth(2019,3)
```

```
      March 2019  
Mo Tu We Th Fr Sa Su  
          1  2  3  
 4  5  6  7  8  9 10  
11 12 13 14 15 16 17  
18 19 20 21 22 23 24  
25 26 27 28 29 30 31
```

```
In [7]: import os  
os.getcwd()
```

```
Out[7]: 'C:\\\\Users\\\\KyungHee'
```

Member	Type	Meaning	Range
tm_sec	int	seconds after the minute	0-61*
tm_min	int	minutes after the hour	0-59
tm_hour	int	hours since midnight	0-23
tm_mday	int	day of the month	1-31
tm_mon	int	months since January	0-11
tm_year	int	years since 1900	
tm_wday	int	days since Sunday	0-6
tm_yday	int	days since January 1	0-365
tm_isdst	int	Daylight Saving Time flag	

The Daylight Saving Time flag (tm\_isdst) is **greater than zero** if Daylight Saving Time is in effect, **zero** if Daylight Saving Time is not in effect, and **less than zero** if the information is not available.

\* tm\_sec is generally 0-59. The extra range is to accommodate for **leap seconds** in certain systems.

# library datetime 이용하여 요일구하기

---

```
In [23]: import datetime  
         date=datetime.date(2018,3,12)  
         date.strftime("%A")
```

```
Out[23]: 'Monday'
```

```
In [24]: date.strftime("%Y %B %A %X")
```

```
Out[24]: '2018 March Monday 00:00:00'
```

```
In [25]: date.strftime("%Y %B %A %x")
```

```
Out[25]: '2018 March Monday 03/12/18'
```

```
In [26]: date.strftime("%Y %B %A %c")
```

```
Out[26]: '2018 March Monday Mon Mar 12 00:00:00 2018'
```



```
In [9]: import datetime  
dt = datetime.datetime.now()
```

```
In [10]: dt.strftime("%y %b %A")
```

```
Out[10]: '19 Mar Tuesday'
```

```
In [11]: dt.weekday() #0: 日 1:: 月
```

```
Out[11]: 1
```

```
In [13]: dt
```

```
Out[13]: datetime.datetime(2019, 3, 12, 21, 45, 39, 589488)
```

```
In [14]: dt.strftime("%A %d. %B %Y")
```

```
Out[14]: 'Tuesday 12. March 2019'
```

```
In [19]: print(dt.strftime("%H/ %M /%S"))
```

```
21/ 45 /39
```

# strftime()

변환 문자열	의미	변환 문자열	의미
%a	요일 이름의 약자	%M	분(00-59)
%A	요일 이름	%p	AM 또는 PM
%b	월 이름의 약자	%S	초(00-59)
%B	월 이름	%w	요일(0-6)
%c	지역 날짜와 시간	%x	지역 날짜
%d	날짜( 01-31)	%X	지역 시간
%H	시간(00-23)	%y	연도(00-99)
%I	시간(01-12)	%Y	연도(예, 2003)
%j	1월 1일 이후의 날짜(001-366)	%%	퍼센트 기호(%)
%m	월(01-12)		

# 실습

---

- 현재 시간과 날짜를 오전 오후를 구분하여 출력하시오.
- 예) 2019/09/15 10:18:13 PM



# 실습

---

- 현재 시간과 날짜를 오전 오후를 구분하여 출력하시오.

```
import time  
time.localtime()
```

```
time.struct_time(tm_year=2019, tm_mon=9, tm_mday=15, tm_hour=22, tm_min=16, tm_sec=39, tm_wday=6, tm_yday=258, tm_isdst=0)
```

```
from time import localtime, strftime  
time_str= strftime("%Y-%m-%d %H:%M:%S", localtime())  
print(time_str)
```

```
2019-09-15 22:16:53
```

```
time_str= strftime("%Y/%m/%d %I:%M:%S %p", localtime())  
print(time_str)
```

```
2019/09/15 10:38:53 PM
```

## 4. 사용자 정의 함수

---

- 함수를 작성하는 이유는 크게 두 가지로 볼 수 있다.
  - **특정의 기능을 수행하는 코드들을 하나의 묶음으로** 사용하기 위해 함수를 사용한다.
  - **동일한 기능을 수행하는 코드들의 재사용성을** 높이고 코드의 통일된 관리를 하기 위해 함수를 작성한다.
- 함수 문법

```
def 함수명([인자1, 인자2, ...]):  
    수행할 문장들  
    [return 반환값]
```

# hello\_not\_function.py

---

```
print("Hello World")
print("Hello World")
print("Hello Word")
print("서강대학교")
print("Hello World")
print("Hello World")
print("Hello Word")
print("Hello Word")
print("서강대학교")
print("Hello World")
print("Hello World")
print("Hello Word")
print("Hello Word")
print("Hello Word")
print("서강대학교")
```

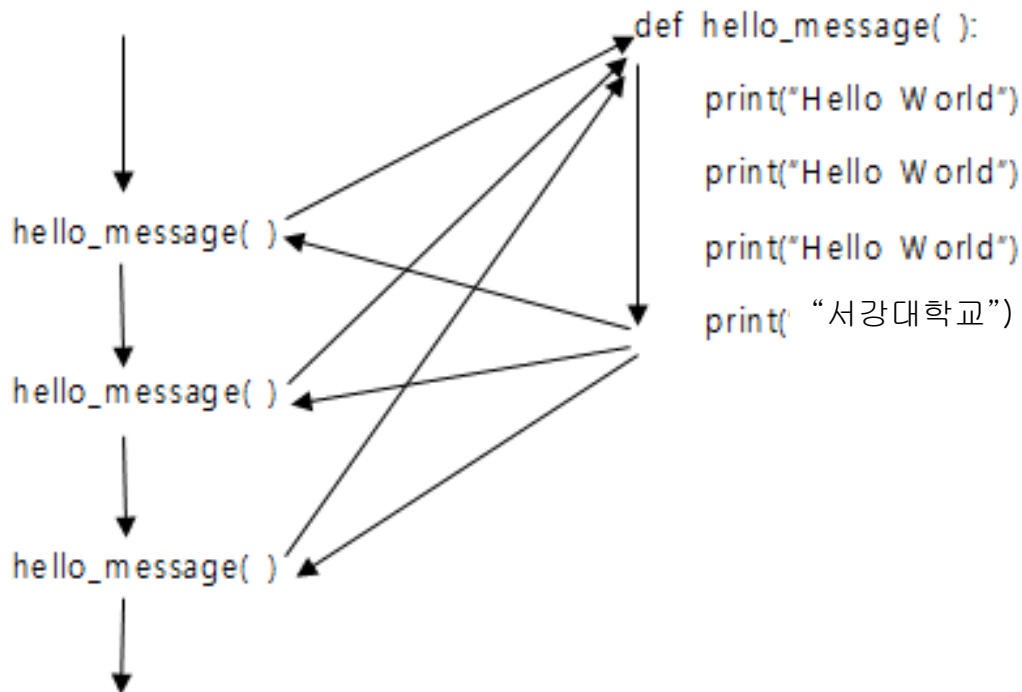
# hello\_function\_1.py

```
def hello_message():
    print("Hello World")
    print("Hello World")
    print("Hello Word")
    print("서강대학교")
```

```
hello_message( )
hello_message( )
hello_message( )
hello_message( )
```

## 5 함수의 호출과 흐름

- 함수가 호출되면 실행의 흐름은 어떻게 될까?
  - 함수가 호출되는 시점에 실행의 흐름은 함수로 넘어간다. 그리고 함수의 작업이 끝나면 실행의 흐름은 다시 호출된 곳으로 돌아온다.



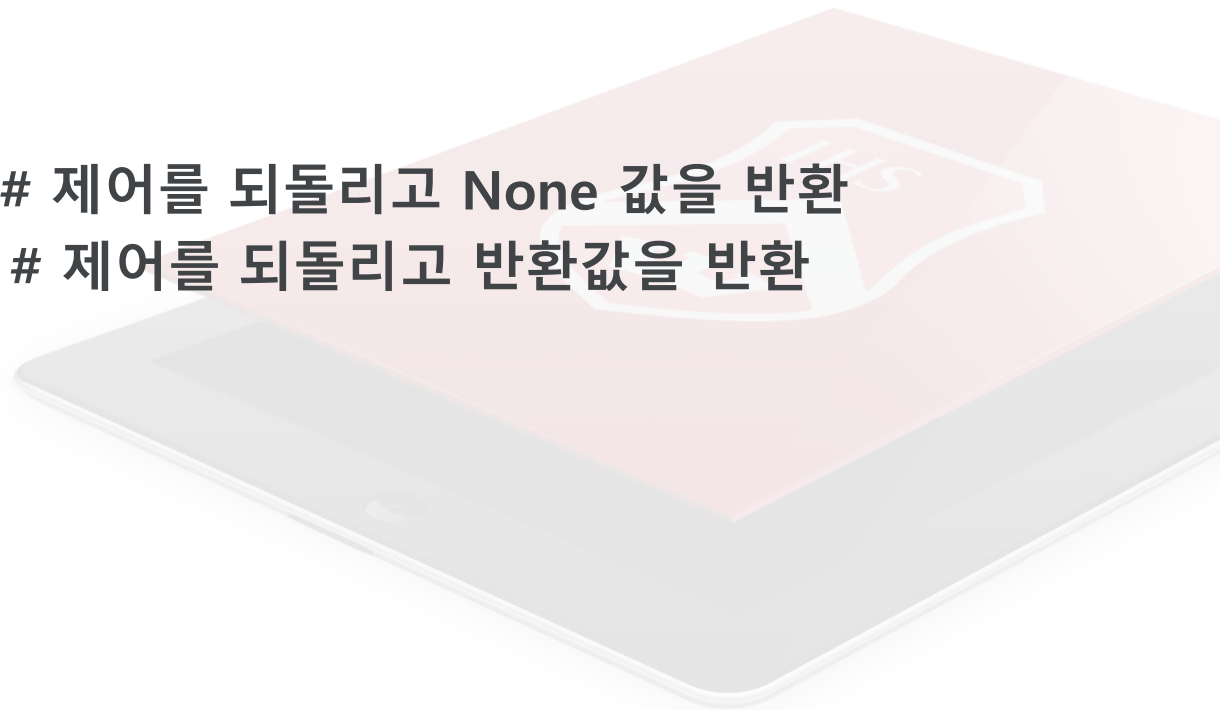
## 6 함수의 인자와 반환값

---

- 함수를 실행할 때 외부로부터 인자를 받아서 처리할 수 있다. 이렇게 외부로부터 넘어온 값은 함수 내부에서 자유롭게 사용이 가능하다. 그리고 함수는 작업을 마친 후 호출한 지점으로 돌아갈 때 반환값을 되돌려 줄 수 있다.

- **return 문**

- return                   # 제어를 되돌리고 None 값을 반환
  - return 반환값       # 제어를 되돌리고 반환값을 반환





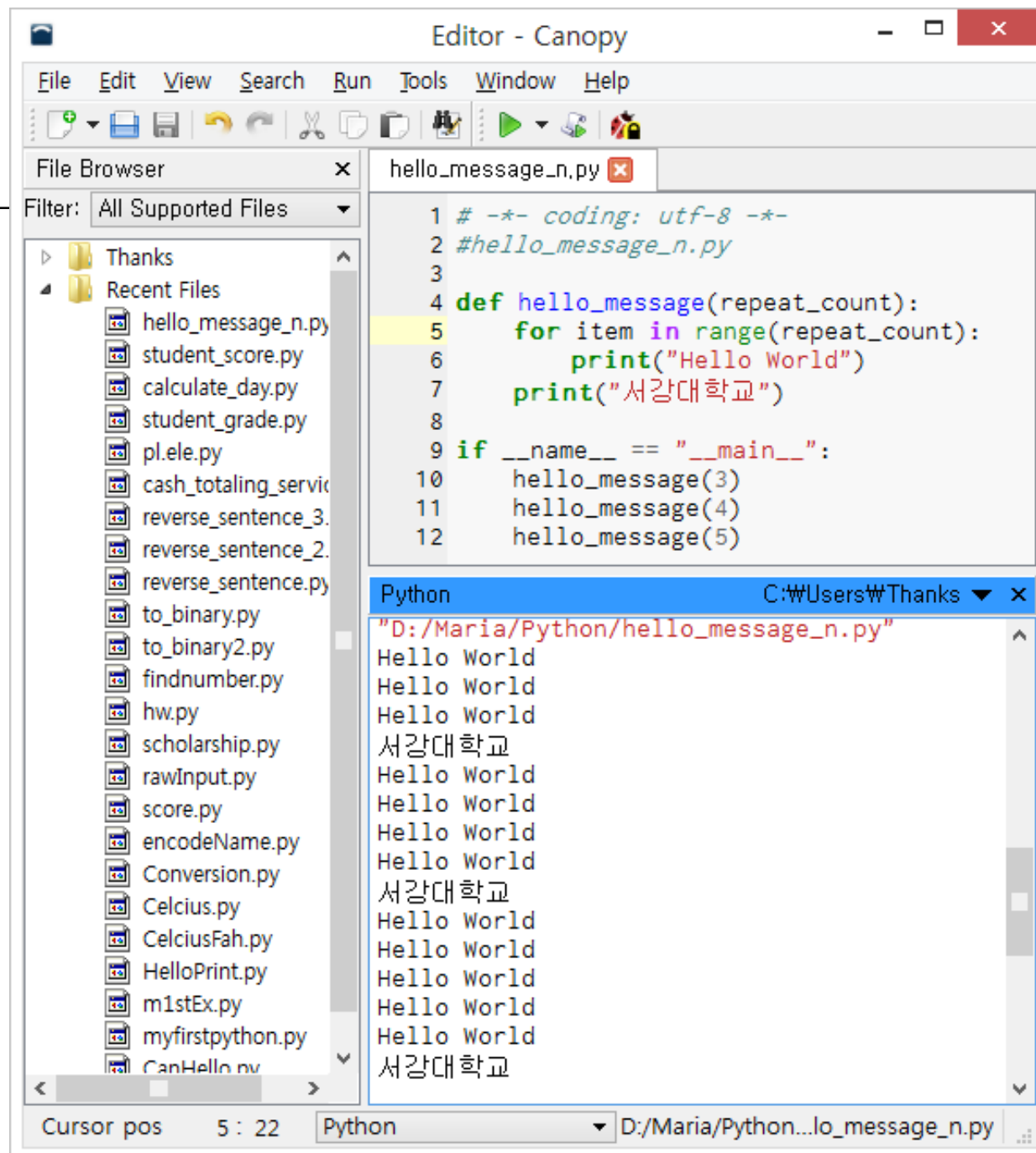
---

```
# hello_message_n.py
```

```
def hello_message( repeat_count ):  
    for item in range(repeat_count):  
        print("Hello World")  
    print("서강대학교")
```

```
if __name__ == "__main__":  
    hello_message(3)  
    hello_message(4)  
    hello_message(5)
```





# 인자 1개 사용하는 함수

```
%%writefile circle_function_1.py
def circle_area(radius):
    area = 3.14 * (radius **2)
    return area
if __name__ == "__main__":
    print("반지름: %d, 면적:%.2f" %(3, circle_area(3)))
    print("반지름: %d, 면적:%.2f" %(4, circle_area(4)))
```

Writing circle\_function\_1.py

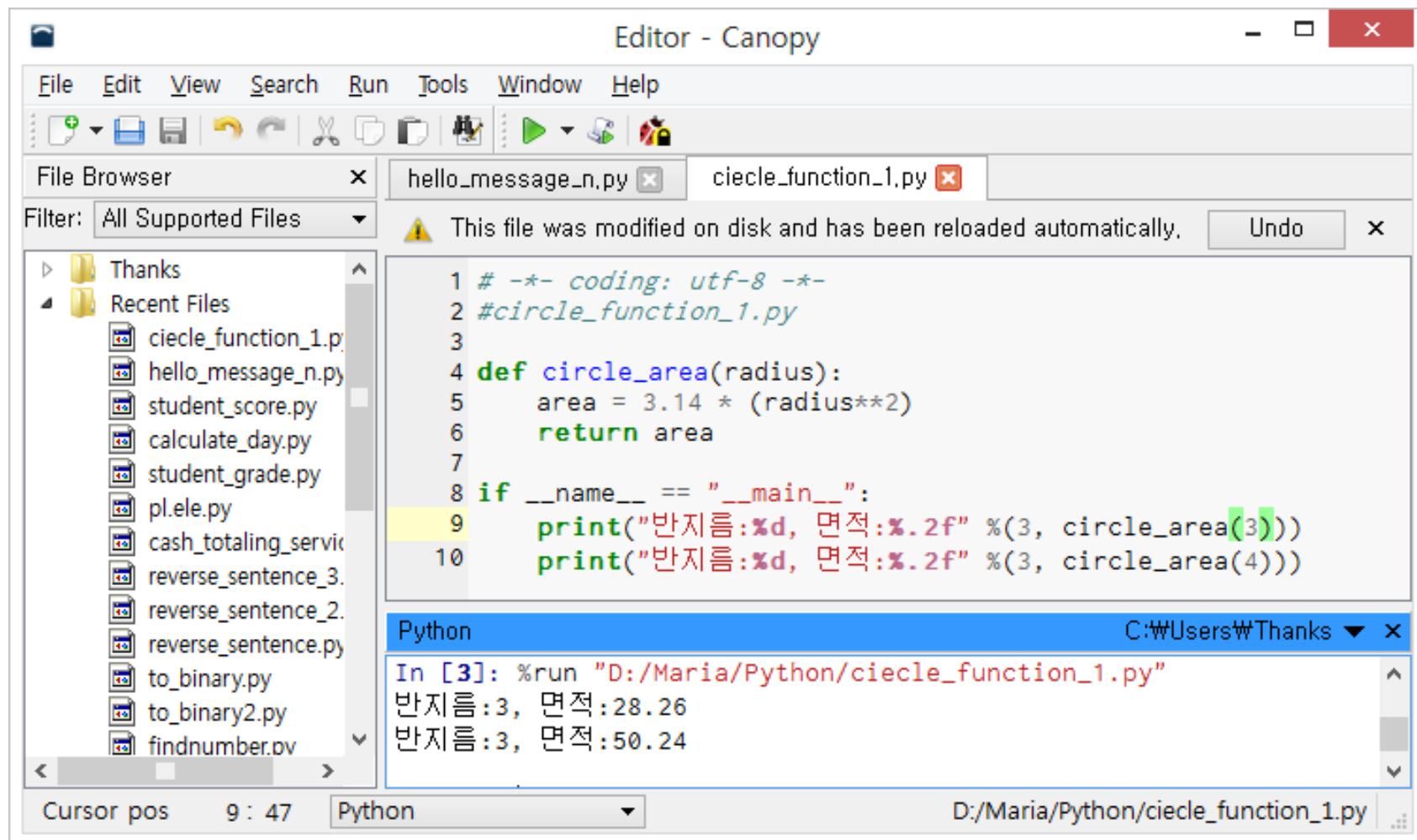
# circle\_function\_1.py

```
def circle_area(radius):
    area = 3.14 * (radius ** 2)
    return area
```

```
if __name__ == "__main__":
    print("반지름: %d, 면적: %.2f" % (3, circle_area(3)))
    print("반지름: %d, 면적: %.2f" % (4, circle_area(4)))
```

```
%run circle_function_1.py
```

```
반지름: 3, 면적:28.26
반지름: 4, 면적:50.24
```



## 인자 2개 사용하는 함수

```
def circle_area(radius, pi):
    area = pi * (radius ** 2)
    return area
```

```
if __name__ == "__main__":
    print("반지름: 3, PI: 3.14, 면적:", circle_area(3, 3.14))
    print("반지름: 3, PI: 3.1415, 면적:", circle_area(3, 3.1415))
```

```
%%writefile circle_fun_2.py
def circle_area(radius, pi):
    area = pi * (radius **2)
    return area
if __name__ == "__main__":
    print("반지름: 3, PI: 3.14, 면적:", circle_area(3, 3.14))
    print("반지름: 3, PI: 3.14,15, 면적:", circle_area(3, 3.1415))
```

Overwriting circle\_fun\_2.py

```
%run circle_fun_2.py
```

```
반지름: 3, PI: 3.14, 면적: 28.26
반지름: 3, PI: 3.14,15, 면적: 28.273500000000002
```

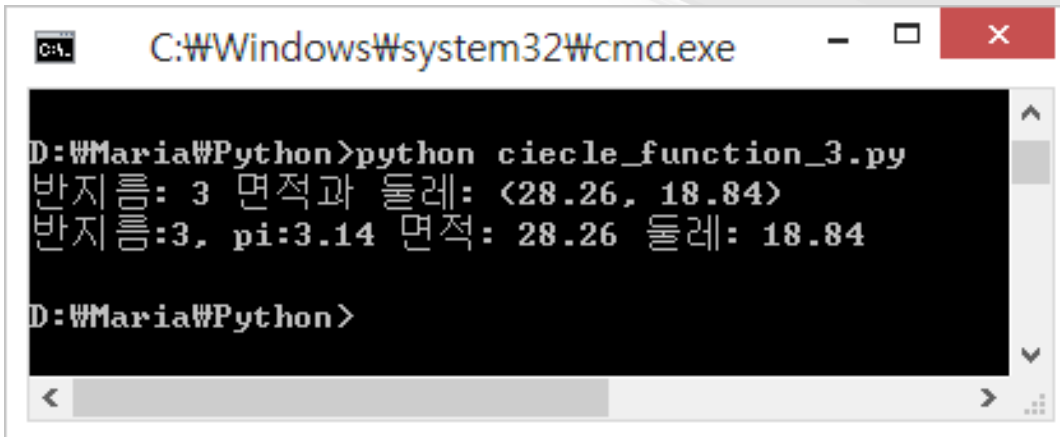
## # circle\_function\_3.py 두개의 반환값을 return하는 함수

```
def circle_area_circumference(radius, pi):  
    area = pi * (radius ** 2)  
    circumference = 2 * pi * radius  
    return area, circumference  
  
if __name__ == "__main__":  
    result = circle_area_circumference(3, 3.14)  
    print("반지름:", 3, "면적과 둘레:", result)  
    res1, res2 = circle_area_circumference(3, 3.1415)  
    print("반지름:", 3, "면적:", res1, "둘레: ", res2)
```

파이썬에서는 함수의 결과로 여러 개의 값을 반환할 수 있다.

return 다음에 적어주며, 반환값을 튜플의 형태로 돌려준다.

튜플을 그대로 이용할 수도 있고, **Unpacking**해서 각 요소를 구해 이용 가능하다



```
C:\Windows\system32\cmd.exe  
  
D:\Maria\Python>python ciacle_function_3.py  
반지름: 3 면적과 둘레: <28.26, 18.84>  
반지름:3, pi:3.14 면적: 28.26 둘레: 18.84  
  
D:\Maria\Python>
```

## 7. 함수의 인자 전달과 가변 데이터

- 불변데이터와 가변데이터의 인자 전달

# arguments1.py

```
def add_one(num1, lst1):      # 2
    num1 = num1 + 1
    lst1.append(1)            # 3
```

```
num = 1
lst = [1, 2, "Hello Sogang"]
print(num, lst)               # 1
```

```
add_one(num, lst)
print(num, lst)               #4
```

```
In [2]: %%writefile arguments1.py
def add_one(num1, lst1):
    num1 = num1 + 1
    lst1.append(1)

num = 1
lst = [1, 2, "Hello Sogang"]
print(num, lst)

add_one(num, lst)
print(num, lst)
```

Writing arguments1.py

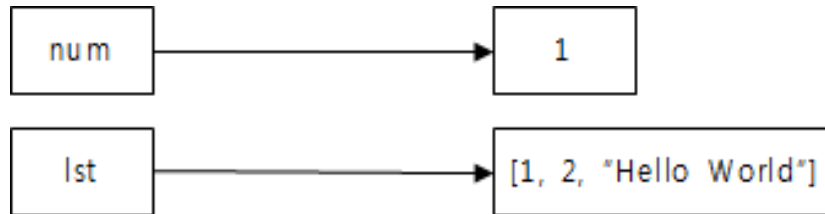
```
In [3]: %run arguments1.py

1 [1, 2, 'Hello Sogang']
1 [1, 2, 'Hello Sogang', 1]
```

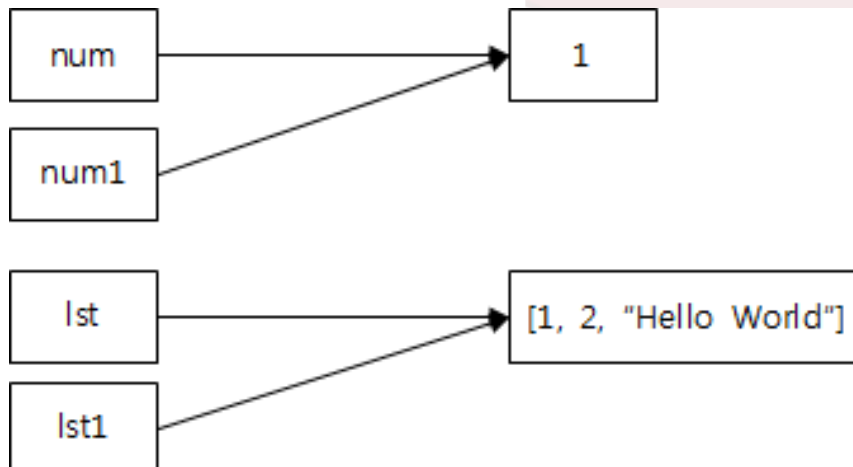
```
D:\Maria\Python>python arguments1.py
1 [1, 2, 'Hello Sogang']
1 [1, 2, 'Hello Sogang', 1]

D:\Maria\Python>
```

- #1: add\_one 호출 전

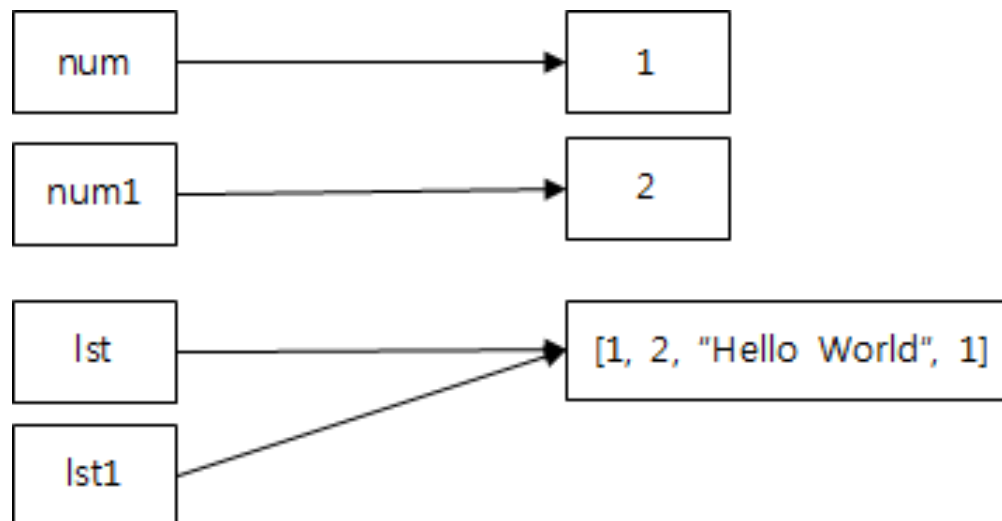


- #2: add\_one 함수 호출 중 (함수를 호출 했을 때)



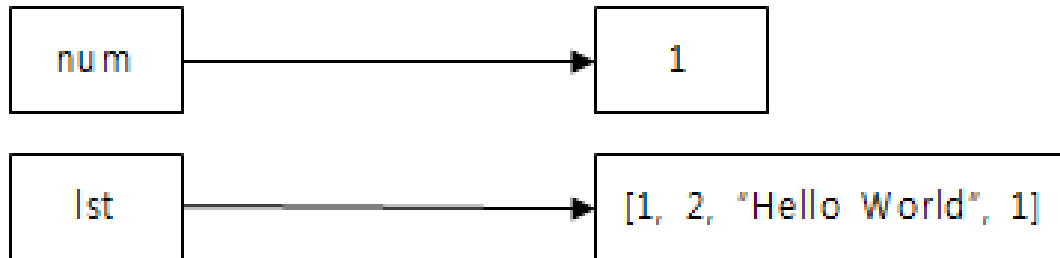


- #3: add\_one 함수 호출 중 (연산이 일어난 후)



불변 데이터인 정수의 경우 새 값을 생성 후 이를 `num1`이 가리키지만,  
가변 데이터인 리스트의 경우 내용이 바뀐다.

- 
- #4: add\_one 함수 호출로부터 돌아온 후



# 가변 데이터를 함수의 인자로 넘길 때, 원본 데이터를 보존하기 위해

....

```
# arguments2.py
```

```
import copy
```

```
def add_one(num1, lst1):
    num1 = num1 + 1
    temp_lst = copy.deepcopy(lst1) # 1
    temp_lst.append(1)
```

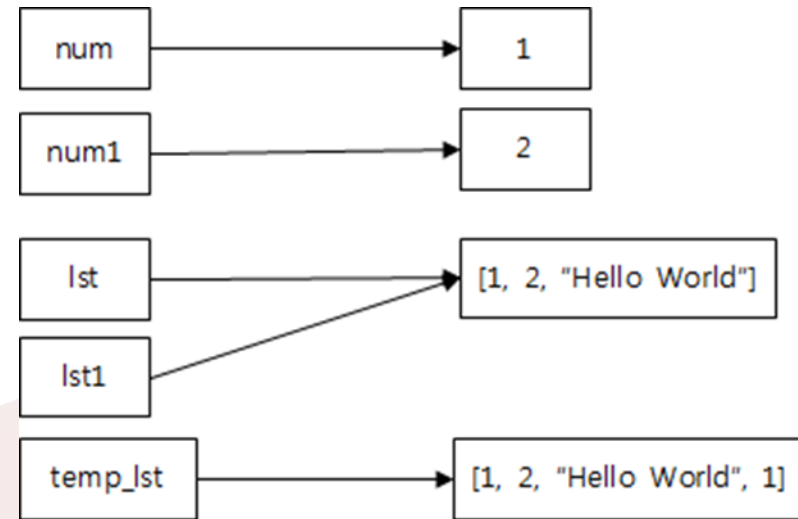
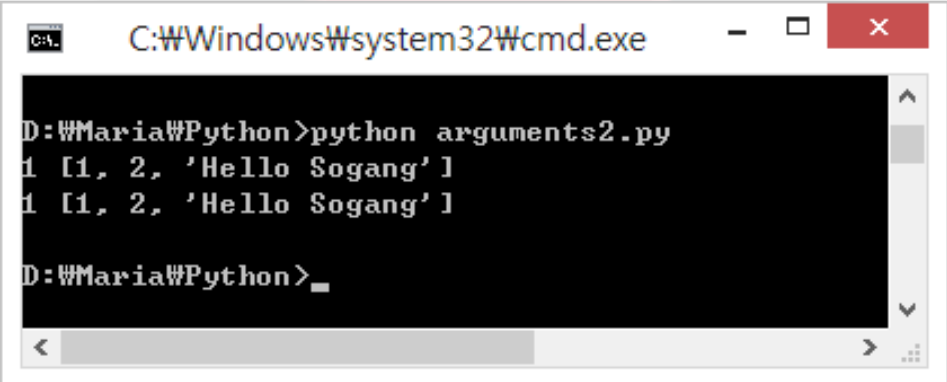
```
num = 1
```

```
lst = [1, 2, "Hello Sogang"]
```

```
print(num, lst)
```

```
add_one(num, lst)
```

```
print(num, lst)
```

```

C:\Windows\system32\cmd.exe

D:\Maria\Python>python arguments2.py
1 [1, 2, 'Hello Sogang']
1 [1, 2, 'Hello Sogang']

D:\Maria\Python>
  
```

In [4]: %%writefile arguments2.py

```
import copy
def add_one(num1, lst1):
    num1 = num1 + 1
    temp_lst = copy.deepcopy(lst1)
    temp_lst.append(1)

num = 1
lst = [1, 2, "Hello Sogang"]
print(num, lst)

add_one(num, lst)
print(num, lst)
```

Writing arguments2.py

In [5]: %run arguments2.py

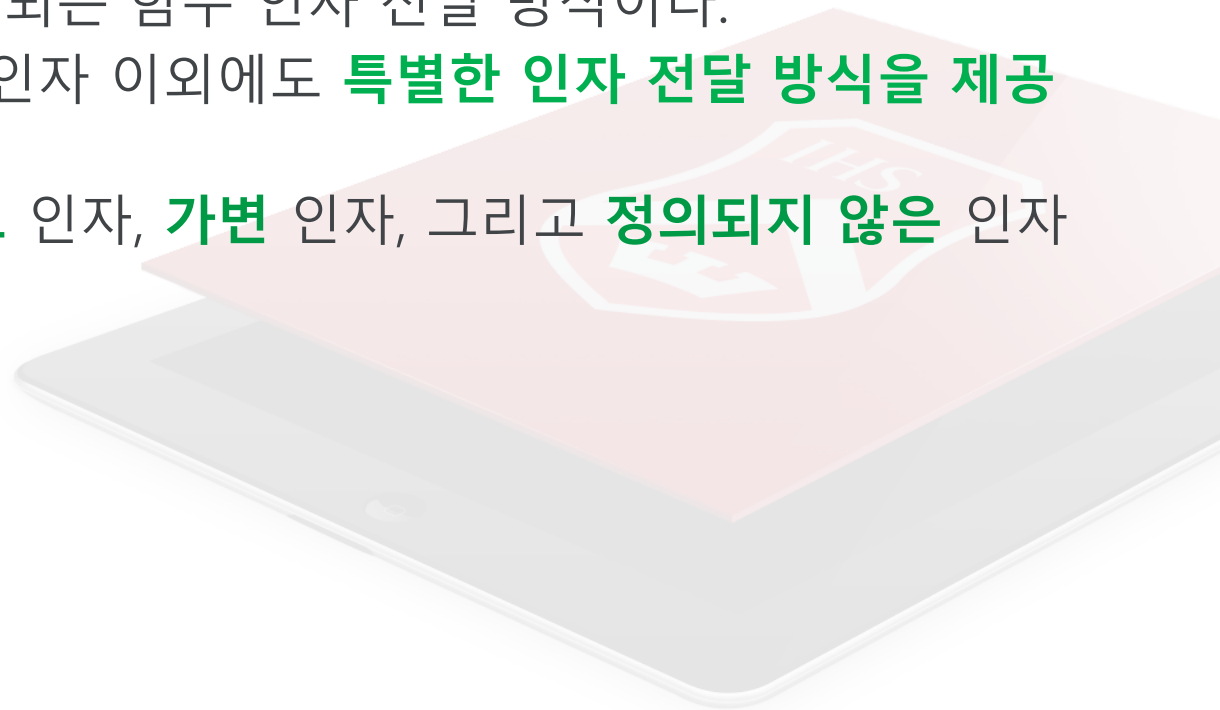
```
1 [1, 2, 'Hello Sogang']
1 [1, 2, 'Hello Sogang']
```



## 8. 일반 인자, 기본 인자, 키워드 인자 그리고 가변 인자

---

- **일반 인자**는 호출 시에 위치가 중요하다.
  - Positional argument
- 함수 호출 시에 넣어준 **인자 순서대로 인자값이 함수에 전달된다.** 이는 가장 많이 사용되는 함수 인자 전달 방식이다.
- 파이썬에서는 일반 인자 이외에도 **특별한 인자 전달 방식을 제공한다.**
  - **기본**인자, **키워드** 인자, **가변** 인자, 그리고 **정의되지 않은** 인자이다.



# Default argument

---

- **기본 인자**

- 함수를 호출할 때 **인자의 값을 설정해 주지 않아도 기본 값이 할당**되도록 하는 인자이다.
- 기본 인자는 일반 인자 뒤에 위치한다.

```
# circle_default_parameter.py

def circle_area(radius, pi=3.14):
    area = pi * (radius ** 2)
    return area

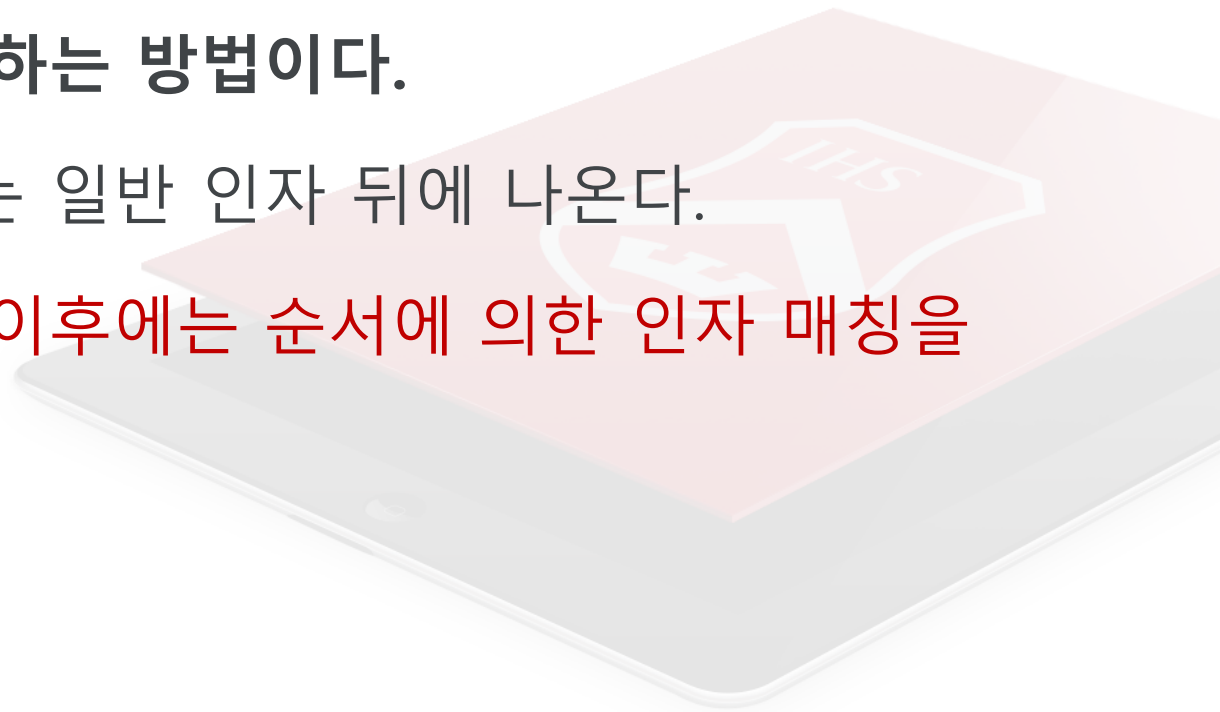
if __name__ == "__main__":
    print("면적: %.2f" % circle_area(3))
    print("면적: %.4f" % circle_area(3, 3.1415))
```

# Keyword argument

---

## • 키워드 인자

- 함수에 인자 값을 전달 할 때 **인자의 이름으로 인자 값을 전달하는 방법이다.**
- 키워드 인자는 일반 인자 뒤에 나온다.
- 키워드 인자 이후에는 순서에 의한 인자 매칭을 할 수 없다.



# 키워드인자의 예

---

```
# circle_keyword_parameter.py
```

```
def circle_area(radius, pi):  
    area = pi * (radius ** 2)  
    return area
```

```
if __name__ == "__main__":  
    print("반지름:", 3, "면적:", circle_area(3, 3.14))  
    print("반지름:", 3, "면적:", circle_area(3, pi=3.14))  
    print("반지름:", 3, "면적:", circle_area(radius=3, pi = 3.14))  
    print("반지름:", 3, "면적:", circle_area(pi = 3.14, radius=3))
```



## 키워드 인자로 radius를 전달하였기에, 이후에는 일반인자를 쓸 수 없는 예

---

```
# circle_wrong_keyword_parameter.py
```

```
def circle_area(radius, pi):  
    area = pi * (radius ** 2)  
    return area
```

```
if __name__ == "__main__":  
    print("반지름:", 3, "면적:", circle_area(radius = 3, 3.1415))
```

```
C:\W> python circle_wrong_keyword_parameter.py  
File ".\circle_wrong_keyword_parameter.py", line 6  
print("반지름:", 3, "면적:", circle_area(radius = 3, 3.1415))
```

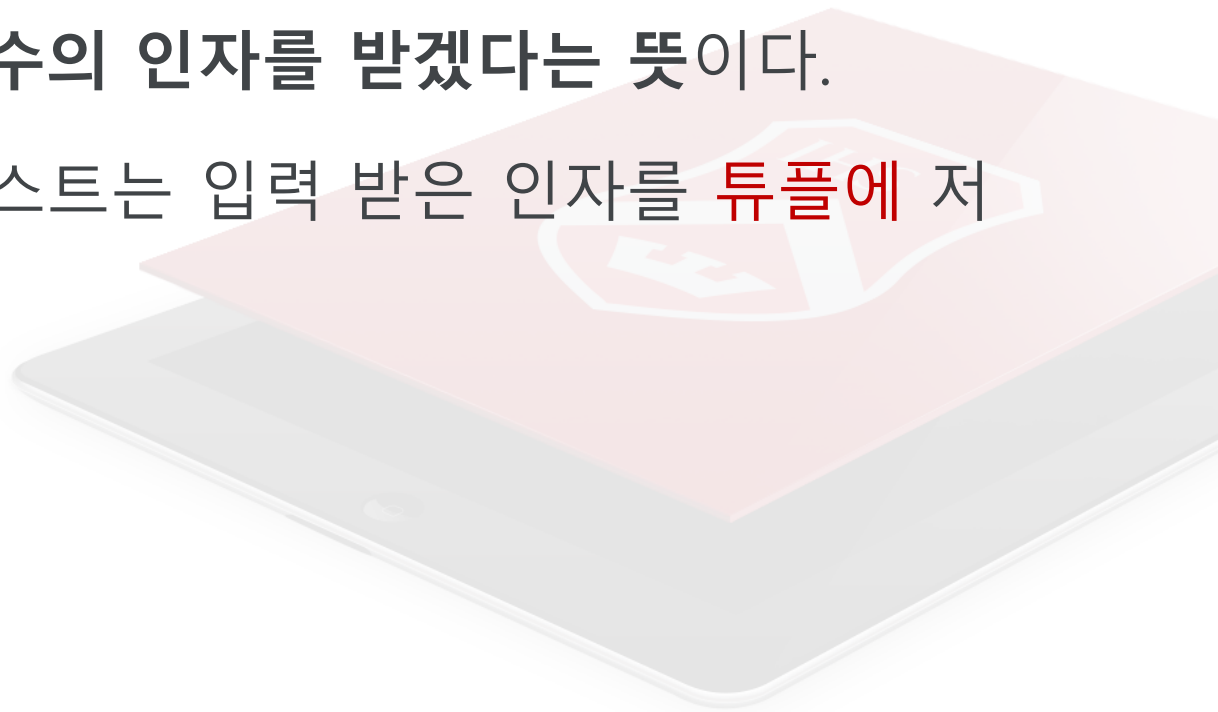
^

**SyntaxError:** non-keyword arg after keyword arg

---

## • 가변 인자

- 함수를 정의할 때, 함수 인자 앞에 **\***을 붙이면 **정해지지 않은 수의 인자를 받겠다는 뜻**이다.
- 가변 인자 리스트는 입력 받은 인자를 **튜플**에 저장한다.



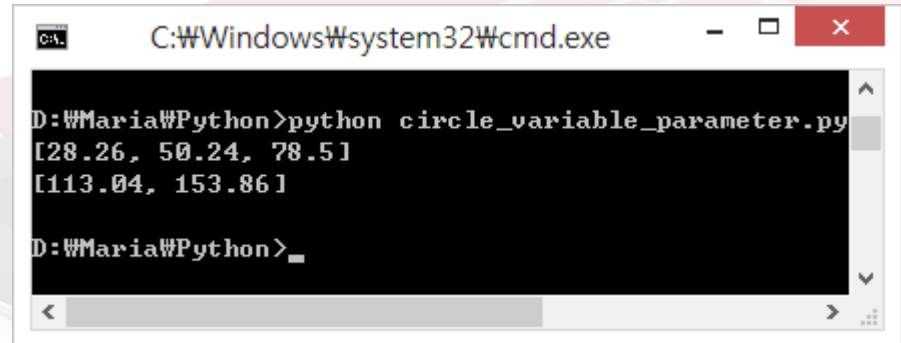
```
# circle_variable_parameter.py
```

```
def circle_area(pi, *radiuses):  
    areas = []
```

```
    for radius in radiuses:  
        area = pi * (radius ** 2)  
        areas.append(area)
```

```
    return areas
```

```
if __name__ == "__main__":  
    print(circle_area(3.14, 3, 4, 5))  
    print(circle_area(3.14, 6, 7))
```



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The prompt is at "D:\Maria\Python>". The user has entered the command "python circle\_variable\_parameter.py". The output of the script is displayed on two lines: "[28.26, 50.24, 78.5]" and "[113.04, 153.86]". The prompt is now at "D:\Maria\Python>\_".

```
%writefile addAddi.py
def add5(x,y, *args):
    print("local variables", locals())

    sum = x + y
    for i in args:
        sum += i
    return sum

print(add5(10,20,3,4,5))
print(add5(10,20))
```

Writing addAddi.py

```
%run addAddi.py

local variables {'args': (3, 4, 5), 'y': 20, 'x': 10}
42
local variables {'args': (), 'y': 20, 'x': 10}
30
```

locals()는 현재의 로컬 변수들을 사전형태로 돌려주는 내장함수

**직접 튜플 형태로 넘겨주면 더 이상 가변인자가 아니므로 \*를 붙이면 안된다.**

```
# circle_not_variable_parameter.py

def circle_area(pi, radiuses):
    areas = []

    for radius in radiuses:
        area = pi * (radius ** 2)
        areas.append(area)

    return areas

if __name__ == "__main__":
    print(circle_area(3.14, (3, 4, 5) ))
    print(circle_area(3.14, (6, 7) ))
```



## • 정의 되지 않은 인자

- 파이썬에서는 함수에 정의되지 않은 인자들을 함수에 전달할 수 있다.
- 정의되지 않은 인자는 **사전 형식으로 전달되며**, 함수에서 정의되지 않은 인자를 받을 때는 인자 앞에 **\*\***을 붙여서 받는다.
- 일반인자, 가변 인자, 정의되지 않은 인자가 있을 경우 함수에서는 **반드시 일반인자, 가변인자, 정의되지 않은 인자 순으로** 인자들을 전달하고 받아야 한다.

# circle\_undefined\_parameter.py

---

```
def circle_area(radius, *pi, **info):  
    for item in pi:  
        area = item * (radius ** 2)  
        print("반지름:", radius, "Pi: ", item, "면적:", round(area,2))  
  
    for key in info:  
        print(key, ":", info[key])  
  
if __name__ == "__main__":  
    circle_area(3, 3.14, 3.1415, line_color="파랑", area_color = "노랑")  
    print()  
    circle_area(5, 3.14, 3.1415, polygon_name="원", value = "면적")
```

```
def circle_area(radius, *pi, **info):
    for item in pi:
        area = item * (radius ** 2)
        print("반지름:", radius, "PI: ", item, "면적:", round(area,2))

    for key in info:
        print(key, ":", info[key])

if __name__ == "__main__":
    circle_area(3, 3.14, 3.1415, line_color="파랑", area_color = "노랑")
    print()
    circle_area(5, 3.14, 3.1415, polygon_name="원", value = "면적")
```

```
반지름: 3 PI: 3.14 면적: 28.26
반지름: 3 PI: 3.1415 면적: 28.27
line_color : 파랑
area_color : 노랑
```

```
반지름: 5 PI: 3.14 면적: 78.5
반지름: 5 PI: 3.1415 면적: 78.54
polygon_name : 원
value : 면적
```



```
%%writefile addAddiUn.py
def add5(x,y, *args, **kargs):
    print("local variables", locals())
    sum = sumj = 0
    sum = x + y
    for i in args:
        sum += i
    for k,j in kargs.items():
        sumj += j
    return sum + sumj

print(add5(10,20,3,4,5, k1=1, k2=2))
print(add5(10,20, *(3,4,5), **dict(k1=1, k2=2)))
print(add5(10,20))
```

Overwriting addAddiUn.py

```
%run addAddiUn.py
```

```
local variables {'kargs': {'k1': 1, 'k2': 2}, 'args': (3, 4, 5), 'y': 20, 'x': 10}
45
local variables {'kargs': {'k1': 1, 'k2': 2}, 'args': (3, 4, 5), 'y': 20, 'x': 10}
45
local variables {'kargs': {}, 'args': (), 'y': 20, 'x': 10}
30
```

## 9 스코핑 룰

- 스코프(Scope, 영역)란 이름이 의미를 가지는 범위이다.

```
# scoping_rule1.py
```

```
def circle_area(r):  
    result = 3.14 * (r ** 2)  
    return result
```

```
if __name__ == "__main__":  
    radius = 3  
    area = circle_area(radius)  
    print("반지름: %d, 면적: %.2f" % (radius, area))  
    print(r)
```

```
C:\W> python scoping_rule1.py
```

```
반지름: 3, 면적: 20.26
```

```
Traceback <most recent call last>:
```

```
File ".\Wscoping_rule1.py", line 9, in <module>
```

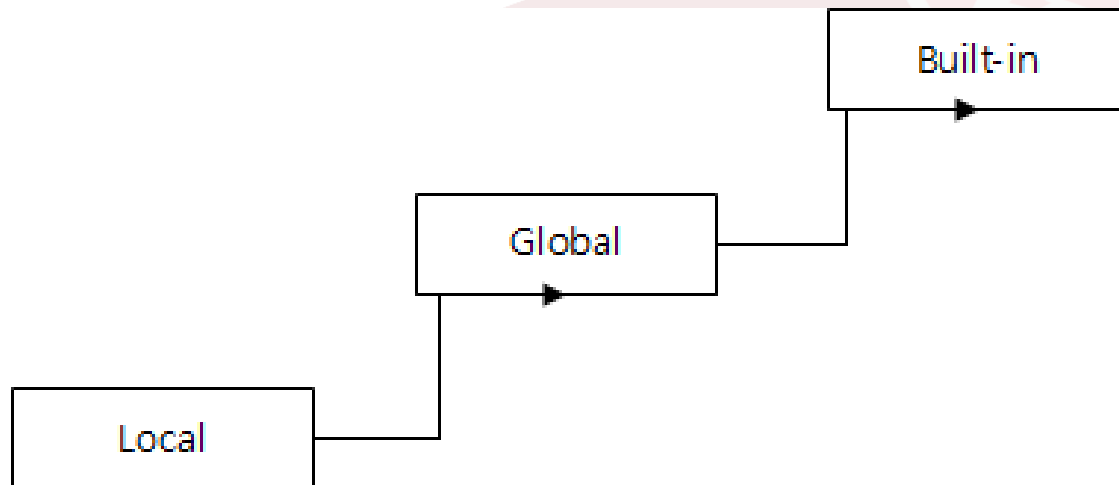
```
print(r)
```

```
NameError: name 'r' is not defined.
```

- 스코프를 벗어난 변수의 접근은 오류를 발생시킨다.

- LGB 규칙

- 이름을 찾는 순서는 Local -> Global -> Built-in 순이다.
  - Local이란 함수나 클래스 안을 의미한다.
  - Global은 프로그램이 수행되는 가장 높은 계층(파일)을 말한다.
  - Built-in은 파이썬이 특별히 예약해둔 이름들을 말한다.



```
# scoping_rule2.py
```

```
# scoping_rule2 global 영역
```

```
pi = 3.14159
```

```
def circle_area_with_pi(r):
    # circle_area_with_pi의 local 영역
    pi = 3.14
    result = pi * (r ** 2)
    return result
```

```
def circle_area_without_pi(r):
    # circle_area_without_pi의 local 영역
    result = pi * (r ** 2)
    return result
```

```
def sum_areas():
    results = [circle_area_with_pi(3), circle_area_without_pi(3)]
    return sum(results) # built-in의 sum 함수를 호출
```

```
if __name__ == "__main__":
    print("PI:", pi)
    print("반지름:", 3, "면적:", circle_area_with_pi(3))
    print("반지름:", 3, "면적:", circle_area_without_pi(3))
    print(sum_areas())
```

```
C:\Windows\system32\cmd.exe
D:\Maria\Python>python scoping_rule.py
PI: 3.14159
반지름: 3 면적: 28.26
반지름: 3 면적: 28.27431
56.534310000000005
D:\Maria\Python>
```

# Global 변수를 Local에서 접근은 되지만, 바로 연산에 이용하는 것은 오류 발생시킨다.

```
# scoping_rule3.py
```

```
# scoping_rule3 global 영역
```

```
pi = 3.14
```

```
def circle_area(r):
```

```
    # circle_area의 local 영역
```

```
    pi = pi + 0.0015
```

```
    result = pi * (r ** 2)
```

```
    return result
```

```
if __name__ == "__main__":
```

```
    print("PI:", pi)
```

```
    print("반지름:", 3, "면적:", circle_area(3))
```

```
    print("PI:", pi)
```

```
%run scoping_rule3.py
```

```
PI: 3.14
```

```
-----
UnboundLocalError                                Traceback (most recent call last)
```

```
C:\Users\KyungHee\scoping_rule3.py in <module>()
```

```
      8 if __name__ == "__main__":
```

```
      9     print('PI:', pi)
```

```
----> 10     print('반지름:', 3, '면적:', circle_area(3))
```

```
      11     print('PI:', pi)
```

```
C:\Users\KyungHee\scoping_rule3.py in circle_area(r)
```

```
      2 def circle_area(r):
```

```
      3     # circle_area의 local 영역
```

```
----> 4     pi = pi + 0.0015
```

```
      5     result = pi * (r ** 2)
```

```
      6     return result
```

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

# • global

## – 해당 변수가 global 영역의 변수임을 알린다

```
# scoping_rule4.py
```

```
# scoping_rule3 global 영역  
pi = 3.14
```

```
def circle_area(r):  
    # circle_area의 local 영역  
    global pi # 이 공간에서의 pi는 global의 pi임을 선언  
    pi = pi + 0.0015  
    result = pi * (r ** 2)  
    return result
```

```
if __name__ == "__main__":  
    print("PI:", pi)  
    print("반지름:", 3, "면적:", circle_area(3))  
    print("PI:", pi)
```

```
%run scoping_rule3.py
```

```
PI: 3.14
```

```
반지름: 3 면적: 28.273500000000002
```

```
PI: 3.1415
```

# 10 재귀 함수

- 재귀(recursive) 함수
  - 함수가 자신을 호출하는 경우의 함수를 말한다.

```
# recursive_sum.py
```

```
def sum(n):
```

```
    if n == 1:
```

```
        return 1
```

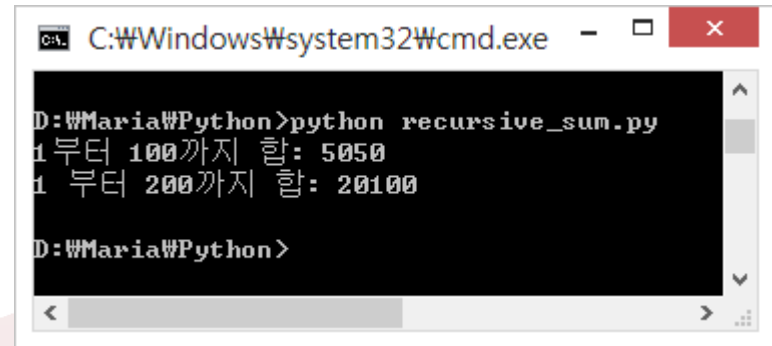
```
    else:
```

```
        return sum(n-1) + n    # 재귀적 호출이 일어나는 부분
```

```
if __name__ == "__main__":
```

```
    print("1부터 100까지 합: %d" % sum(100))
```

```
    print("1 부터 200까지 합: %d" % sum(200))
```



```
C:\Windows\system32\cmd.exe
D:\Maria\Python>python recursive_sum.py
1부터 100까지 합: 5050
1 부터 200까지 합: 20100
D:\Maria\Python>
```

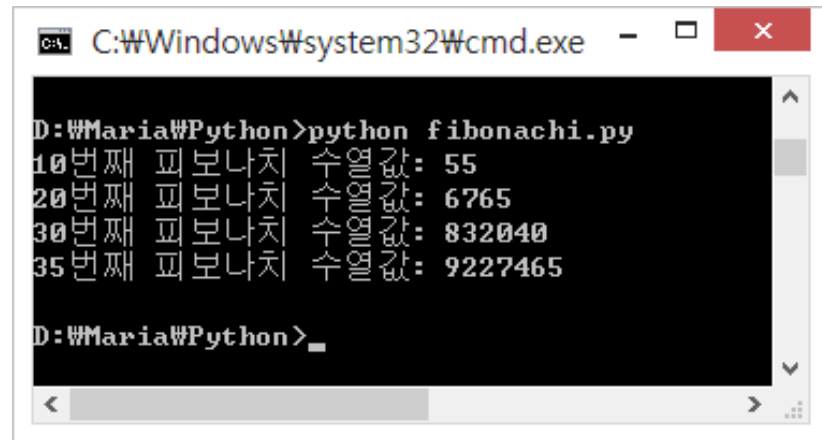




```
# fibonachi.py
```

```
def fibo(n):
    if n == 1:
        return 1
    elif n == 2:
        return 1
    else:
        return fibo(n-2) + fibo(n-1)
```

```
if __name__ == "__main__":
    print("10번째 피보나치 수열값: %d" % fibo(10))
    print("20번째 피보나치 수열값: %d" % fibo(20))
    print("30번째 피보나치 수열값: %d" % fibo(30))
    print("35번째 피보나치 수열값: %d" % fibo(35))
```



```
C:\Windows\system32\cmd.exe
D:\Maria\Python>python fibonachi.py
10번째 피보나치 수열값: 55
20번째 피보나치 수열값: 6765
30번째 피보나치 수열값: 832040
35번째 피보나치 수열값: 9227465
D:\Maria\Python>
```

# 재귀적 호출이 일어나는 부분

# Generator이용한 Fibonacci 수열 구현

```
def Fibonacci():  
    a,b =0, 1  
    while 1:  
        yield a  
        a, b = b, a+b  
  
for i, ret in enumerate(Fibonacci()):  
    if i < 20: print(i, ret)  
    else: break
```

```
0 0  
1 1  
2 1  
3 2  
4 3  
5 5  
6 8  
7 13  
8 21  
9 34  
10 55  
11 89  
12 144  
13 233  
14 377  
15 610  
16 987  
17 1597  
18 2584  
19 4181
```

- 함수 내에 yield가 포함될 경우 그 함수는 generator
- yield: 함수를 끝내지 않고 호출한 곳에 값을 전달함
- Generator는 for문과 짝을 이루어 사용됨

enumerate()는 순회 가능한 객체(순서가 있는 자료형, 리스트, 튜플, 문자열)에서 인덱스 값과 요소 값 둘 다 반환하는 내장함수

# yield와 return

---

```
def reverse(data):  
    for index in range(len(data)-1, -1, -1):  
        yield data[index]  
  
for char in reverse('golf'):  
    print(char)
```

f  
l  
o  
g

```
def reverse(data):  
    for index in range(len(data)-1, -1, -1):  
        return data[index]  
  
for char in reverse('golf'):  
    print(char)
```

f

```
def square_numbers(nums):  
    for i in nums:  
        yield i * i
```

```
my_nums = square_numbers([1, 2, 3, 4, 5])
```

```
print(my_nums)
```

```
<generator object square_numbers at 0x00000191F14B6F68>
```

```
def square_numbers(nums):  
    for i in nums:  
        yield i * i
```

```
my_nums = square_numbers([1, 2, 3, 4, 5])
```

```
print(next(my_nums))
```

```
def square_numbers(nums):  
    for i in nums:  
        yield i * i  
  
my_nums = square_numbers([1, 2, 3, 4, 5])  
  
for i in my_nums:  
    print(i)  
print(next(my_nums))
```

```
1  
4  
9  
16  
25
```

```
-----  
StopIteration                                Traceback (most recent call last):  
<ipython-input-3-fffb776348a4c> in <module>()  
      7 for i in my_nums:  
      8     print(i)  
----> 9 print(next(my_nums))
```

```
StopIteration:
```



```
def Test():
    print("start Test()")
    tmp = [x*x for x in range(5)]
    return tmp
a= Test()
print(a)
```

```
start Test()
[0, 2, 4, 6, 8]
```

```
def Test():
    print("start Test()")
    for x in range(5):
        yield x*x
a= Test()
for i in a:
    print(i)
```

```
start Test()
0
2
4
6
8
```

```
def Test():
    print("start Test()")
    for x in range(5):
        yield x*x
a= Test()
for i in range(0,5):
    print(next(a))
```

```
start Test()
0
2
4
6
8
```

```
def abc():
    data = "abc"
    for char in data:
        return char
it = iter(abc())
next(it)
```

```
'a'
```

```
next(it)
```

```
-----
StopIteration                                Traceback (most recent call last):
<ipython-input-7-2cdb14c0d4d6> in <module>()
----> 1 next(it)
```

```
StopIteration:
```

```
def abc():
    data = "abc"
    for char in data:
        yield char
it = iter(abc())
next(it)
```

```
'a'
```

```
next(it)
```

```
'b'
```

```
next(it)
```

```
'c'
```

```
In [19]: mytuple = ("apple", "banana", "cherry")
```

```
In [20]: iter(mytuple)
```

```
Out[20]: <tuple_iterator at 0x173b4859cf8>
```

```
In [21]: myit = iter(mytuple)
```

```
In [22]: next(myit)
```

```
Out[22]: 'apple'
```

```
In [23]: next(myit)
```

```
Out[23]: 'banana'
```

```
In [24]: next(myit)
```

```
Out[24]: 'cherry'
```

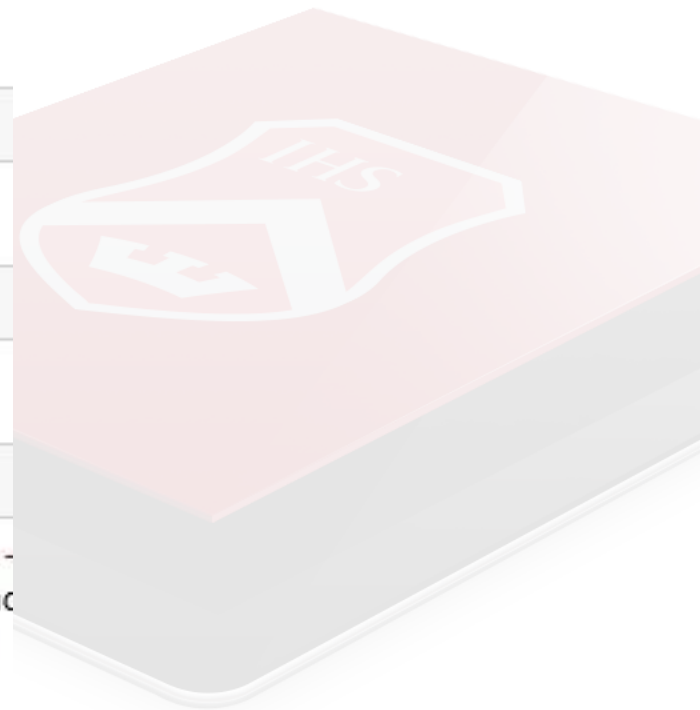
```
In [25]: next(myit)
```

```
-----  
StopIteration
```

```
<ipython-input-25-410d07518a3f> in <module>()
```

```
----> 1 next(myit)
```

```
StopIteration:
```



# 11 람다 함수

---

- 파이썬에서 run time에 생성해서 사용할 수 있는 이름 없는 **익명 함수**를 만들 수 있다.
- 익명 함수는 함수가 필요할 때 간단하게 만들어 쓸 수 있고
- 함수를 다른 함수의 인자로 넘겨줄 때 효과적으로 사용할 수 있다.
- 파이썬에서는 익명 함수를 만들기 위해 **람다 함수를 이용한다.**
- 문법

**lambda 인자들: 표현식**





```
# lambda_exam.py
```

```
if __name__ == "__main__":
```

```
    circle_area = lambda radius, pi: pi * (radius ** 2)
```

```
    print(circle_area(3, 3.14))
```

```
if __name__ == "__main__":  
    circle_area = lambda radius, pi: pi * (radius ** 2)  
  
    print(circle_area(3, 3.14))  
    print(circle_area(3, 3.14159))
```

```
28.26
```

```
28.27431
```

```
a=[1,2,3,4]  
b=[11,12,13,14]
```

```
list(map(lambda x,y :x+y, a,b))
```

```
[12, 14, 16, 18]
```

```
ListC =[1,2,3,4,5,6,7,8,9]
```

```
list(filter(lambda x: x%3 == 0, ListC))
```

```
[3, 6, 9]
```

```
from functools import reduce
```

```
reduce(lambda x,y:x+y, [1,2,3,4,5,6])
```

```
21
```

```
g=lambda x: x**2
```

```
g(8)
```

```
64
```

```
gg= lambda x, y: x+y
```

```
gg(8,9)
```

```
17
```



# 함수를 인자로 전달하기

- 파이썬에서는 함수의 인자로 **함수**를 전달할 수 있다.

```
def circle_area(radius, print_format):  
    area = 3.14 * (radius ** 2)  
    print_format(area)
```

```
def precise_low(value):  
    print("결과값:", round(value, 1))
```

```
def precise_high(value):  
    print("결과값:", round(value, 2))
```

```
if __name__ == "__main__":  
    circle_area(3, precise_low)  
    circle_area(3, precise_high)
```

간단하거나 일회용이라면 lambda로 처리 가능하다.

```
In [1]: def circle_area(radius, print_format):  
        area = 3.14 * (radius ** 2)  
        print_format(area)  
  
        if __name__ == "__main__":  
            circle_area(3, lambda x: print("결과값:", round(x, 1)))  
            circle_area(3, lambda x: print("결과값:", round(x, 2)))
```

결과값: 28.3

결과값: 28.26

## [실습] Lambda함수 이용하기

---

`iter(object[, sentinel])` 와 `lambda` 이용하여 아래와 같은 기능을 수행하시오.

```
import random
while True:
    i = random.randint(0,5)
    if i == 2:
        break
    print(i, end=" ")
```

5 4 1 4 4 1 1 1 5 0 1 4 3 4 0 1



# [실습] 요일 구하기 프로그램을 함수 이용해서 작성

- 년, 월, 일을 입력 받아서 요일을 구한다.
  - 서기 1년 1월 1일은 월요일이다.
  - 윤년을 구하는 공식은 다음과 같다.
    - ① 4로 나누어지는 해는 윤년이다.
    - ② 100으로 나누어지는 해는 윤년이 아니다.
    - ③ 400으로 나누어지는 해는 윤년이다.
  - 요일은 서기 1년 1월 1일 부터 입력된 날까지의 날수를 모두 더한 값을 7로 나누어 나머지를 이용하여 구한다.

```
if __name__ == "__main__":
    year, month, day = input_date()
    day_name = get_day_name(year, month, day)
    print(day_name)
    if is_leap(year) == True:
        print("입력하신 %s은 윤년입니다"% year)
```

\_\_년도를 입력하시오:2019  
\_\_월을 입력하시오:3  
\_\_일을 입력하시오:19  
화요일

```
if __name__ == "__main__":
    year, month, day = input_date()
    day_name = get_day_name(year, month, day)
    print(day_name)
    if is_leap(year) == True:
        print("입력하신 %s은 윤년입니다"% year)
```

\_\_년도를 입력하시오:1988  
\_\_월을 입력하시오:12  
\_\_일을 입력하시오:22  
목요일  
입력하신 1988은 윤년입니다