# Ch 4 Python Functions

- Function Examples
- Basic Rules of Python Functions
- Functions From Library
- Challenging Python Functions

# Taxonomy of Python Functions

- Python Built-in Functions:
  - input(), print(), len(), abs(), set(), …..

- User Defined Functions

- Functions belonging to Python Built-In Data Types

```
>>> L = [3, 5, 5]                 >>> S = {3, 5, 6}
>>> L.append(4)                   >>> S.remove(5)
```

- Functions belonging to User-Defined Classes

```
>>> class Box(object):                    >>> bbb = Box()
        def  calc_space(self):            >>> space_value = bbb.calc_space()
             ……
```

- Functions from Other Modules (consisting of Functions and Classes)

```
>>>  import sam_module
```

- Functions from Python Standard Library (consisting of Functions and Classes)

```
>>>  import math
```

# Happy Birth Day Song

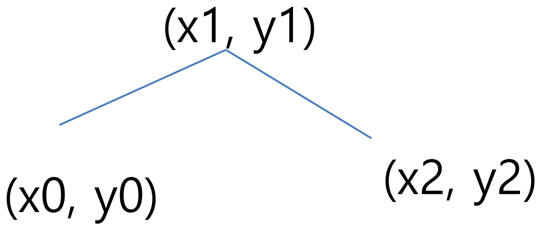
```
# sing("Kang Min")
#
# Happy birthday to you!
# Happy birthday to you!
# Happy birthday, dear Kang Min.
# Happy birthday to you!

def sing(name):
    print("Happy birthday to you! ")
    print("Happy birthday to you! ")
    print("Happy birthday, dear %s "    %name)
    print("Happy birthday to you! ")
```

# Compute Molecular Weight (분자 질량)

```
# Here are basic weights:  C → carbon: 12.011 ,   H → hydrogen: 1.0079 ,  O → oxygen: 15.9994
# use round(46.0688, 2) ==> 46.07  #  round() is built-in function

def molecular_wight():
    print("Please enter the number of each atom!!! ")
    C = input("carbon: ")
    H = input("hydrogen: ")
    O = input("oxygen: ")
    W = C*12.011 + H*1.0079 + O*15.9994
    print ("The molecular weight of C", C, "H", H, "O", O, "is: ",   round(W, 2) )

def molecular_wight_correct():
    print("Please enter the number of each atom!!! ")
    C = eval(input("carbon: "))
    H = eval(input("hydrogen: "))
    O = eval(input("oxygen: "))
    W = C*12.011 + H*1.0079 + O*15.9994
    print ("The molecular weight of C", C, "H", H, "O", O, "is: ",  round(W, 2) )
```

# Euclidean Distance Computation

```
#  Euclidean Distance:   직교 좌표계에서 두 점의 거리
#  예를 들어, 2차원 평면에서 두 점 (x1, y1), (x2, y2)의 거리는
#  math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2)) 로 계산
#  이와 같이 임의의 차원에서의 거리를 구하는 함수를 구현해보세요.

# 함수가 받을 parameter는 총 3개로,
# 첫번째 parameter n: point 갯수,  parameter X : list of x좌표, parameter Y : list of y좌표

import math

def eucDist(n, X, Y):
    distance = 0
    for i in range(n-1):
        distance = distance + math.sqrt( (X[i]-X[i+1])**2 + (Y[i]-Y[i+1])**2 )
    #
    return distance


>> xpt = [2.0  4.0  6.0]        #  점 3개의  X좌표
>> ypt = [1.5  4.5 10.2]        #  점 3개의  Y좌표
>> eucDist(3, xpt, ypt)         #
```



# Temperature Warning

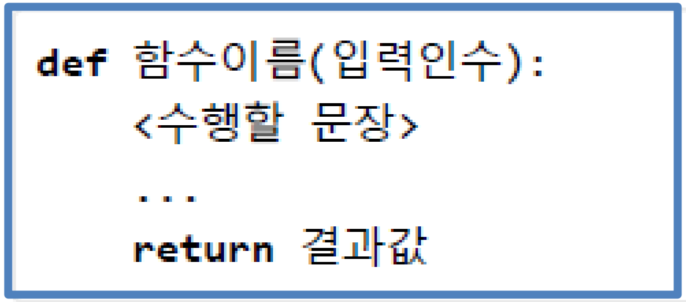
```
# input은  '20.3F'  '-10C'   '32.5C'  같은방식의 string으로 입력
# output은
#     물의 끓는 점 이상일 경우 Be careful!
#     물이 어는 점 이하일 경우 Don't get frozen!
#     섭씨 15도에서 20도 사이일 경우 You will be fine!

def FtoC(F):
    C = (F-32)*5/9
    return C

def TempOK(C):
    if C >= 100:                print ("Be careful!“)
    if C <= 0:                  print ("Don't get frozen!“)
    if C >= 15 and C <= 20:  print ("You will be fine“)

def WeatherMessage():
    temp = input("Type your temperature in string format:")
    if  temp[-1] == "C":
         Centi = float(temp[:-1])
         TempOK(Centi)
    elif  temp[-1] == "F":
         Fahren = float(temp[:-1])
         TempOK(FtoC(Fahren))
    else:   print("Pardon?")
```

# Ch 4 Python Functions

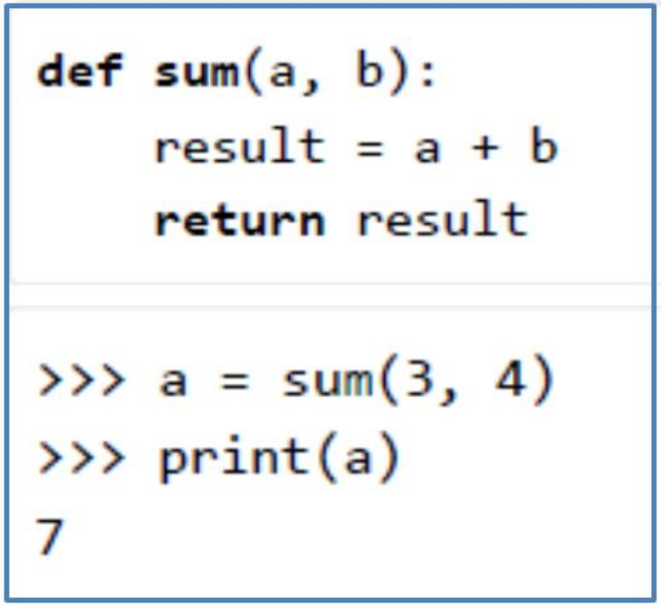
- Function Examples
- Basic Rules of Python Functions
- Functions From Library
- Challenging Python Functions

# 일반적인 함수

입력값이 있고 결과값이 있는 함수가 일반적인 함수이다.
함수는 대부분 다음과 비슷한 형태일 것이다.



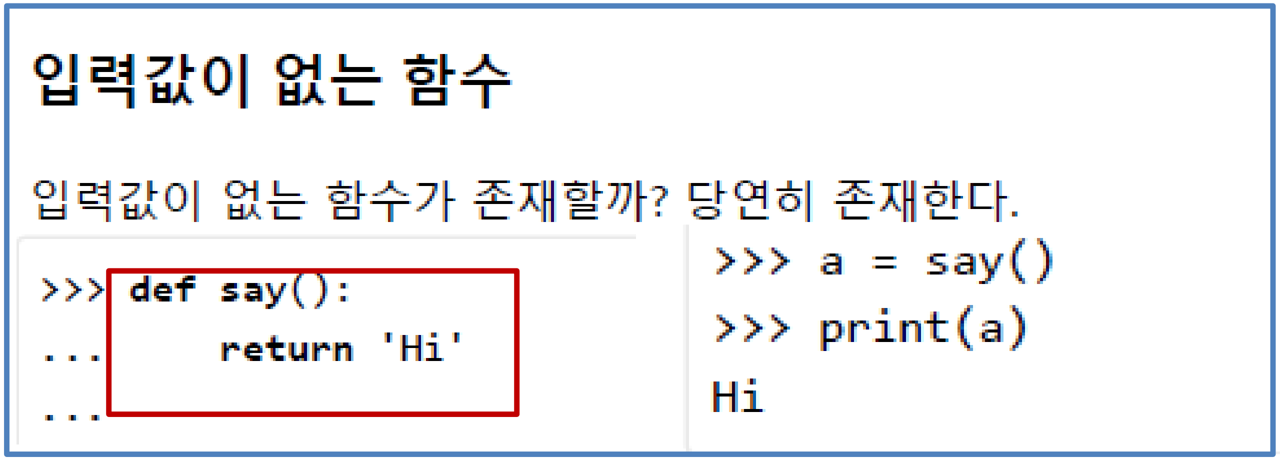
```
def 함수이름(입력인수):
    <수행할 문장>
    ...
    return 결과값
```



```
def sum(a, b):
    result = a + b
    return result
```

```
>>> a = sum(3, 4)
>>> print(a)
7
```

```
def sum(a, b):
    result = a +b
    print(result)

>>> a = sum(3, 4)
>>> print(a)
```

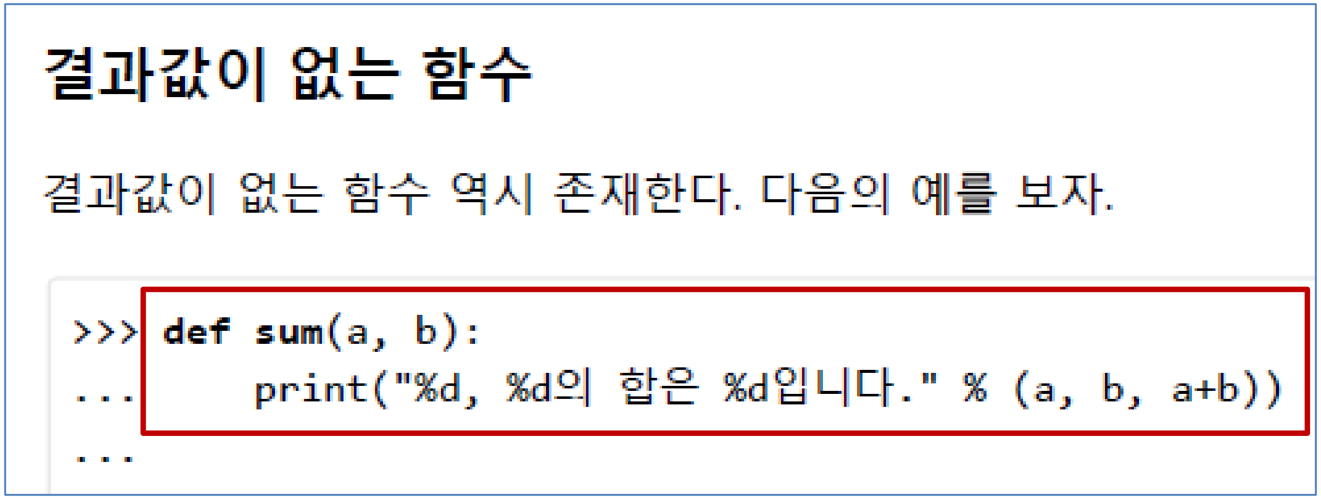


## 입력값이 없는 함수

입력값이 없는 함수가 존재할까? 당연히 존재한다.

```
>>> def say():
...     return 'Hi'
...
```

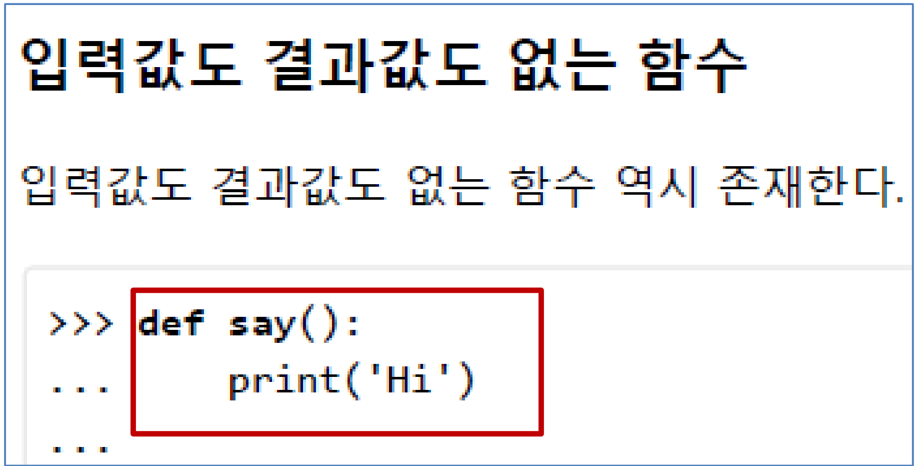
```
>>> a = say()
>>> print(a)
Hi
```



## 결과값이 없는 함수

결과값이 없는 함수 역시 존재한다. 다음의 예를 보자.

```
>>> def sum(a, b):
...     print("%d, %d의 합은 %d입니다." % (a, b, a+b))
...
```

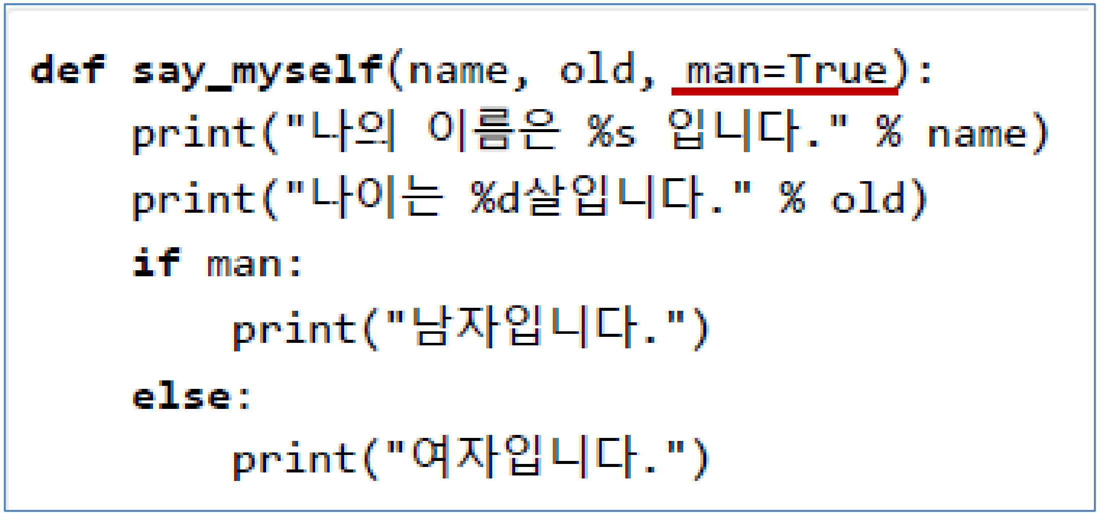


## 입력값도 결과값도 없는 함수

입력값도 결과값도 없는 함수 역시 존재한다.

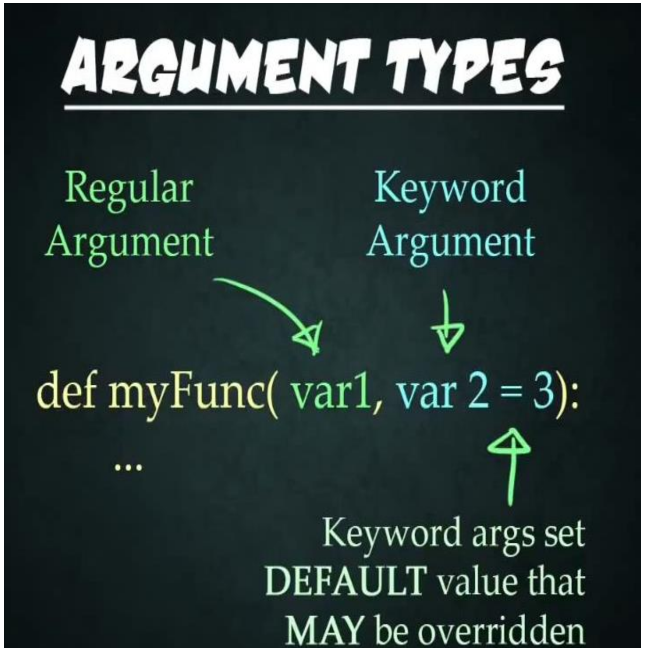
```
>>> def say():
...     print('Hi')
...
```

# Optional Parameter에  default value를 미리 설정



```
def say_myself(name, old, man=True):
    print("나의 이름은 %s 입니다." % name)
    print("나이는 %d살입니다." % old)
    if man:
        print("남자입니다.")
    else:
        print("여자입니다.")
```

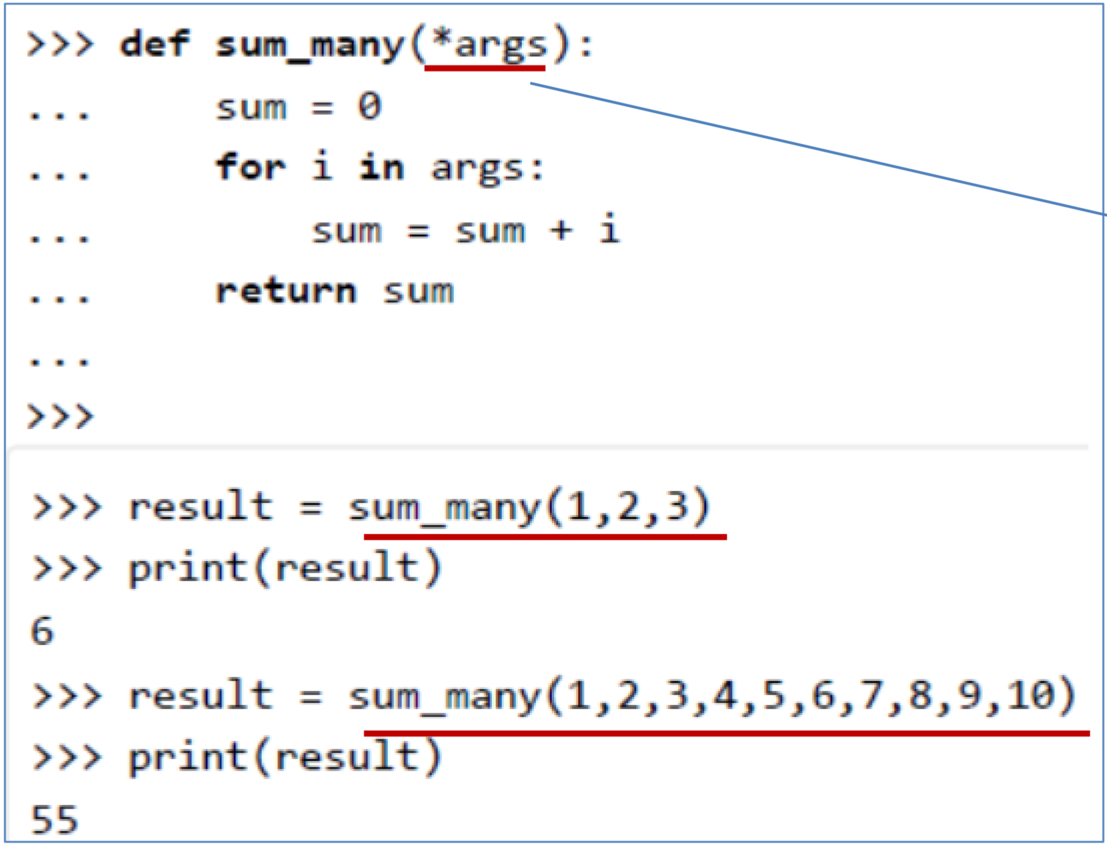
. 초기화시키고 싶은 입력 변수들을 항상 뒤쪽에 위치시키는 것을 잊지 말자.


ARGUMENT TYPES
Regular Argument
Keyword Argument
def myFunc( var1, var 2 = 3):
...
Keyword args set DEFAULT value that MAY be overridden

```
>>> def myFunc(var1, var2=3):
          return var1 + var2

>>> myFunc(10, 10)
20

>>> myFunc(10)
13
```

# 여러 개의 입력값을 받는 함수 [1/2]

- 여러 개의 입력값을 모두 더하는 함수를 만들려 한다.
- 예, sum_many(1, 2) returns 3  sum_many(1,2,3,4,5) returns 15

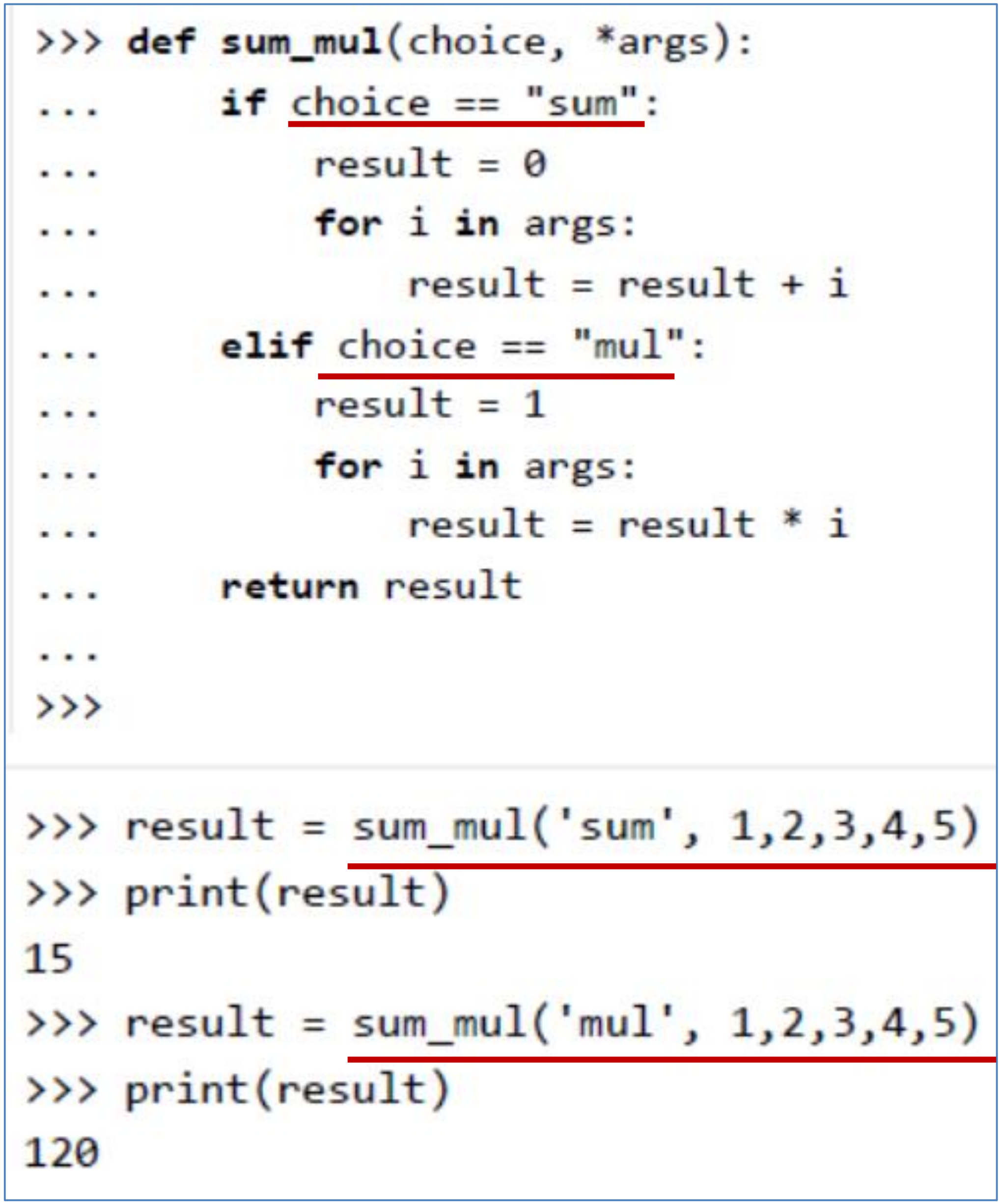


```
>>> def sum_many(*args):
...     sum = 0
...     for i in args:
...         sum = sum + i
...     return sum
...
>>>
```

```
>>> result = sum_many(1,2,3)
>>> print(result)
6
>>> result = sum_many(1,2,3,4,5,6,7,8,9,10)
>>> print(result)
55
```

여러개의 입력값이
들어오는것을 기대

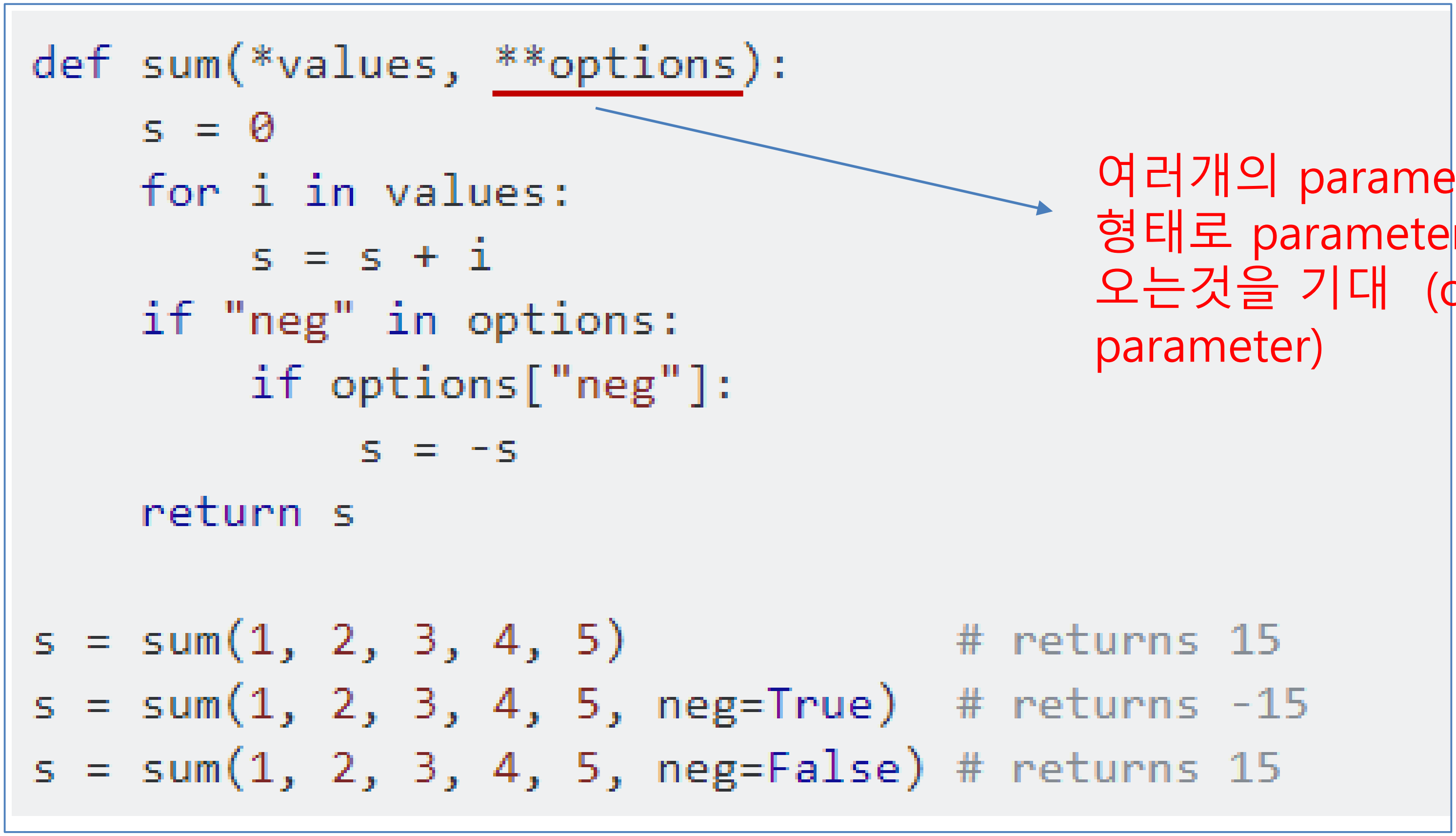
# 여러 개의 입력값을 받는 함수 [2/2]



```
>>> def sum_mul(choice, *args):
...     if choice == "sum":
...         result = 0
...         for i in args:
...             result = result + i
...     elif choice == "mul":
...         result = 1
...         for i in args:
...             result = result * i
...     return result
...
>>>
```

```
>>> result = sum_mul('sum', 1,2,3,4,5)
>>> print(result)
15
>>> result = sum_mul('mul', 1,2,3,4,5)
>>> print(result)
120
```

# 여러 개의 Optional Parameter 들을 받는 함수

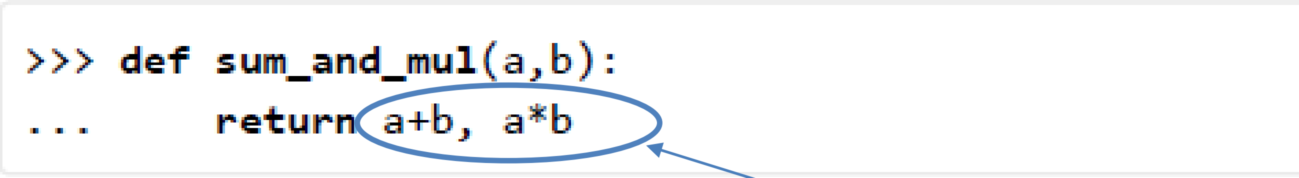


```
def sum(*values, **options):
    s = 0
    for i in values:
        s = s + i
    if "neg" in options:
        if options["neg"]:
            s = -s
    return s

s = sum(1, 2, 3, 4, 5)            # returns 15
s = sum(1, 2, 3, 4, 5, neg=True)  # returns -15
s = sum(1, 2, 3, 4, 5, neg=False) # returns 15
```

여러개의 parameter = value 형태로 parameter 값이 들어오는것을 기대 (optional parameter)

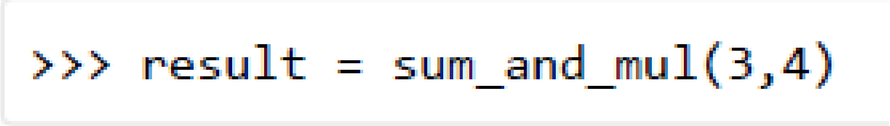
# 함수의 결과값은 언제나 하나이다

먼저 다음의 함수를 만들어 보자.

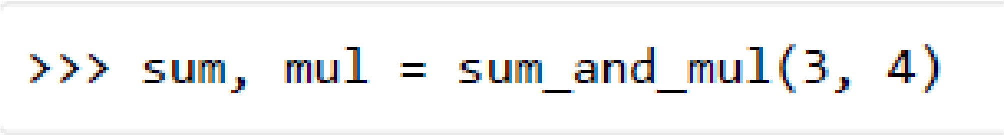


```
>>> def sum_and_mul(a,b):
...     return a+b, a*b
```

Tuple로 값을 return

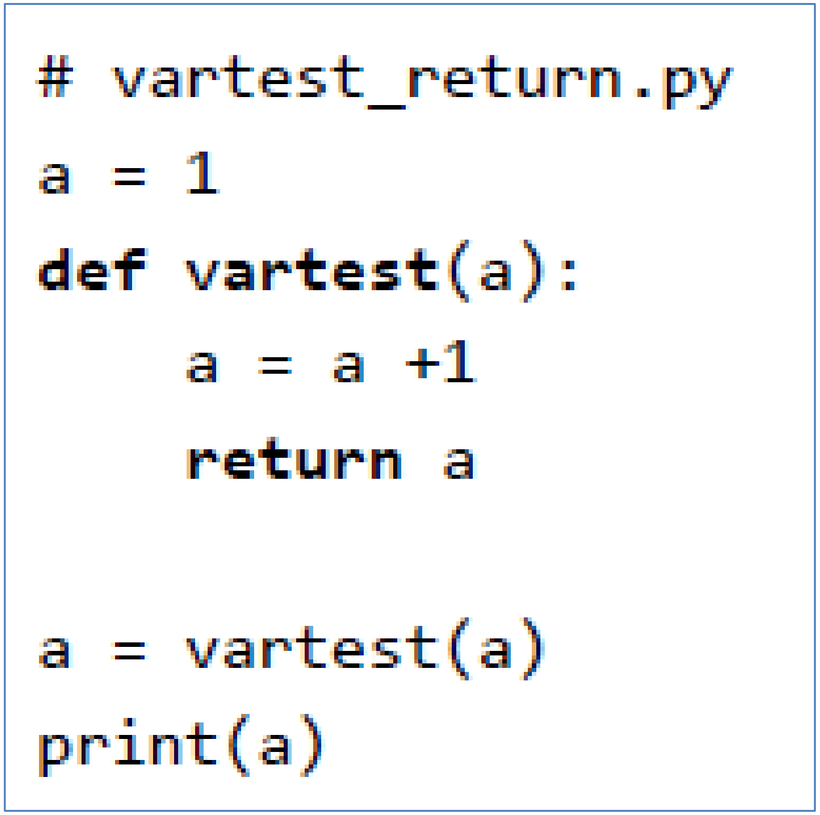


```
>>> result = sum_and_mul(3,4)
```



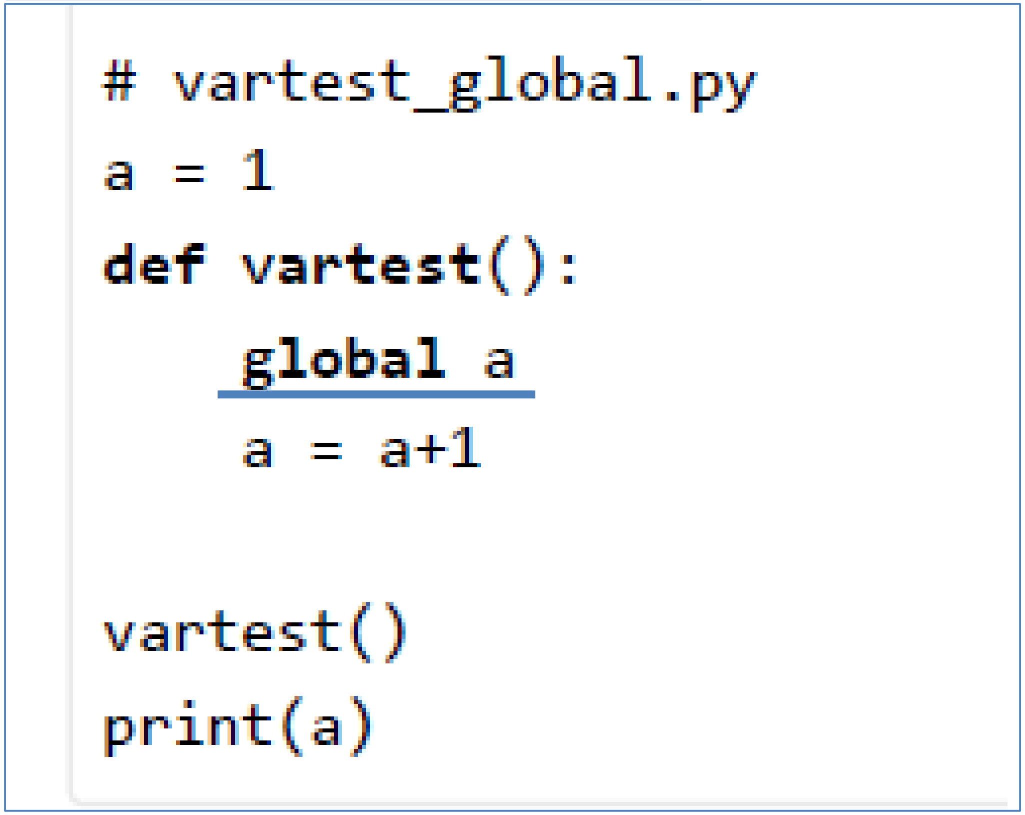
```
>>> sum, mul = sum_and_mul(3, 4)
```

# 함수 안에서 함수 밖의 변수를 변경하는 방법 [1/2]



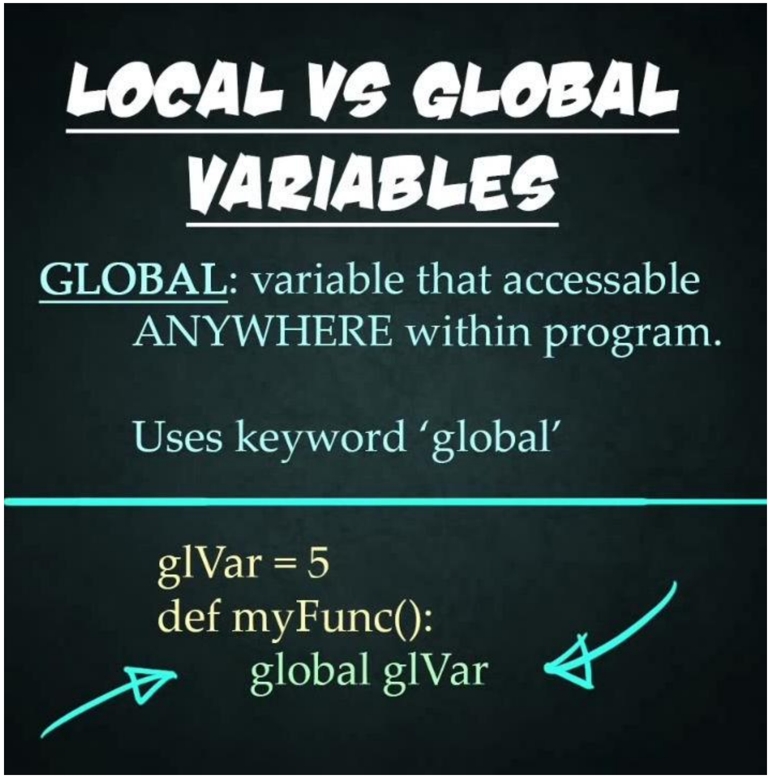
```python
# vartest_return.py
a = 1
def vartest(a):
    a = a +1
    return a

a = vartest(a)
print(a)
```

Global 명령어를 이용하기



```python
# vartest_global.py
a = 1
def vartest():
    global a
    a = a+1

vartest()
print(a)
```

# 함수 안에서 함수 밖의 변수를 변경하는 방법 [2/2]



LOCAL VS GLOBAL VARIABLES

GLOBAL: variable that accessable ANYWHERE within program.

Uses keyword ‘global’

```
glVar = 5
def myFunc():
        global glVar
```

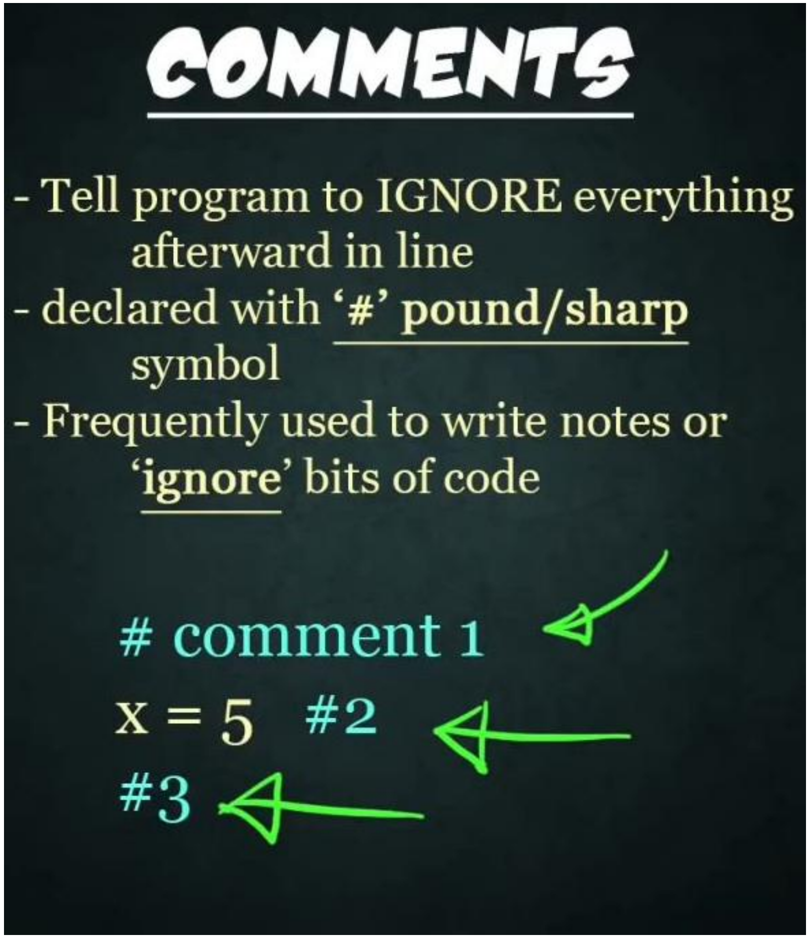
```
>>> glVar = 5

>>> def myFunc1():
>>>       global glVar
>>>       glVar = glVar - 10
>>>       print(“Current glVar: “, glVar)

>>> def myFunc2():
>>>       global glVar
>>>       glVar = glVar + 10
>>>       print(“Current glVar: “, glVar)


>>> myFunc1()
>>> myFunc2()
```

```
>>> x = 3
>>>
>>> def foo1(x)
         z = x + 2
         return z
>>>
>>> foo1(5)
```

```
>>> x = 3
>>>
>>> def foo1( )
         x = 1
         z = x + 2
         return z
>>>
>>> foo1( )
```

```
>>> x = 3
>>>
>>> def foo2( )
         z = x + 2
         return z
>>>
>>> foo2( )
```

# Python Special Variables



## COMMENTS

- Tell program to IGNORE everything afterward in line
- declared with ‘#’ **pound/sharp** symbol
- Frequently used to write notes or ‘**ignore**’ bits of code

```
# comment 1
x = 5   #2
#3
```

Document String

-- Text describing the function

-- comes immediately after function name

-- Use triple quotes to enclose

```
def myfunc( ):
‘” My description is more than one line
   and helps other people understand
   this function
‘’
```

```
>>> print( myFunc.__doc__ )
‘” My description is more than one line
   and helps other people understand
   this function
‘’


>>> print( myFunc.__name__ )
myFunc
```

Python Special Variables

__doc__

__name__

# Predefined Attributes in Python

- Called “special variables” or “magic variables”
  - They contain meta-data about script files / modules
  - The form of __<variable>__, which is enclosed by two underscores
- One important variable is **`__name__`**
  - it tells us the **name** of the module
  - currently running script file will have __name__ = “__main__”

```
>> import math
>> math.__name__
‘math’
>> __name__
‘__main__’
```

- The complete list of predefined attributes are listed in

  https://docs.python.org/2/reference/datamodel.html

  __name__, __dict__, __doc__, __code__, 등등

# if __name__ == '__main__':

```
# suppose we have testFile.py as follows

def testFile(dest):
      print(dest)

if __name__ == '__main__':      # Is this the main file?
      testFile('ham')
      print('done!!')
```

========================================

testFile.py를 Python interpreter에서 수행하면 (즉 python testFile.py 하면)
if __name__ = '__main__': 이 true가 되고 그 아래 문장들이 수행됨

반면에 import testFile 하면
if __name__ = '__main__': 이 false가 되고 그 아래 문장들이 수행이 안됨

** __name__ 은 python의 special variable로써 현재 수행되는 .py file의 상태정보가지고 있음 ➔ outside module을 수행하는지, main module을 수행하는지

# Ch 4   Python Functions

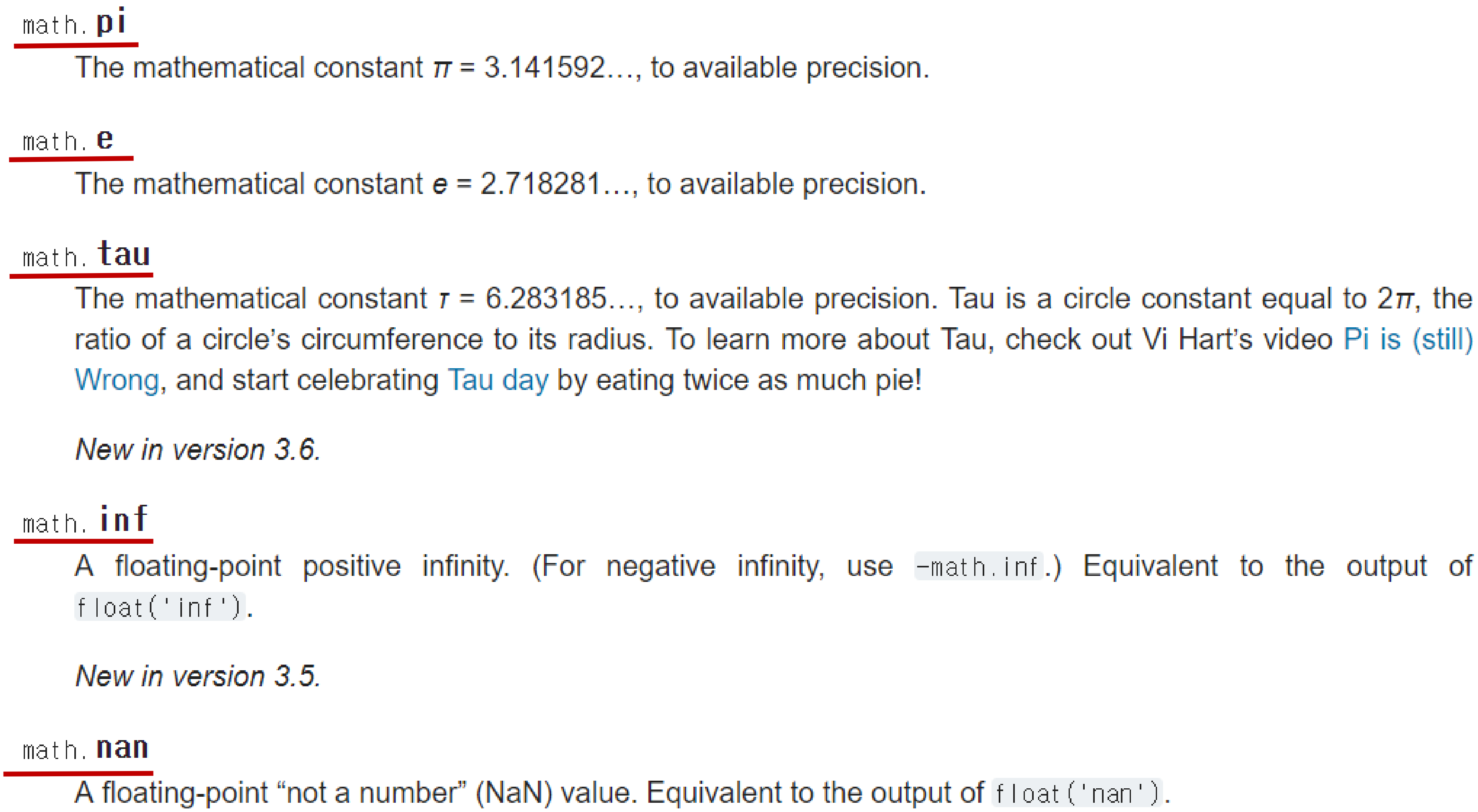
- Function Examples
- Basic Rules of Python Functions
- Functions From Library
- Challenging Python Functions

# “math” standard library module

- This module provides access to mathematical functions defined in C standard
  - These functions cannot be used with complex numbers (Use “cmath” module)

- These are the categories of the functions in “math” module
  - Number-theoretic functions
  - Power and logarithmic function (지수 로그 함수)
  - Trigonometric function (삼각함수)
  - Angular function (각도 함수)
  - Hyperbolic function (쌍곡선 함수)
  - Constant

# Mathematical Constants “math” Module



`math.`**`pi`**

The mathematical constant *π* = 3.141592…, to available precision.

`math.`**`e`**

The mathematical constant *e* = 2.718281…, to available precision.

`math.`**`tau`**

The mathematical constant *τ* = 6.283185…, to available precision. Tau is a circle constant equal to 2*π*, the ratio of a circle’s circumference to its radius. To learn more about Tau, check out Vi Hart’s video Pi is (still) Wrong, and start celebrating Tau day by eating twice as much pie!

*New in version 3.6.*

`math.`**`inf`**

A floating-point positive infinity. (For negative infinity, use `-math.inf`.) Equivalent to the output of `float('inf')`.

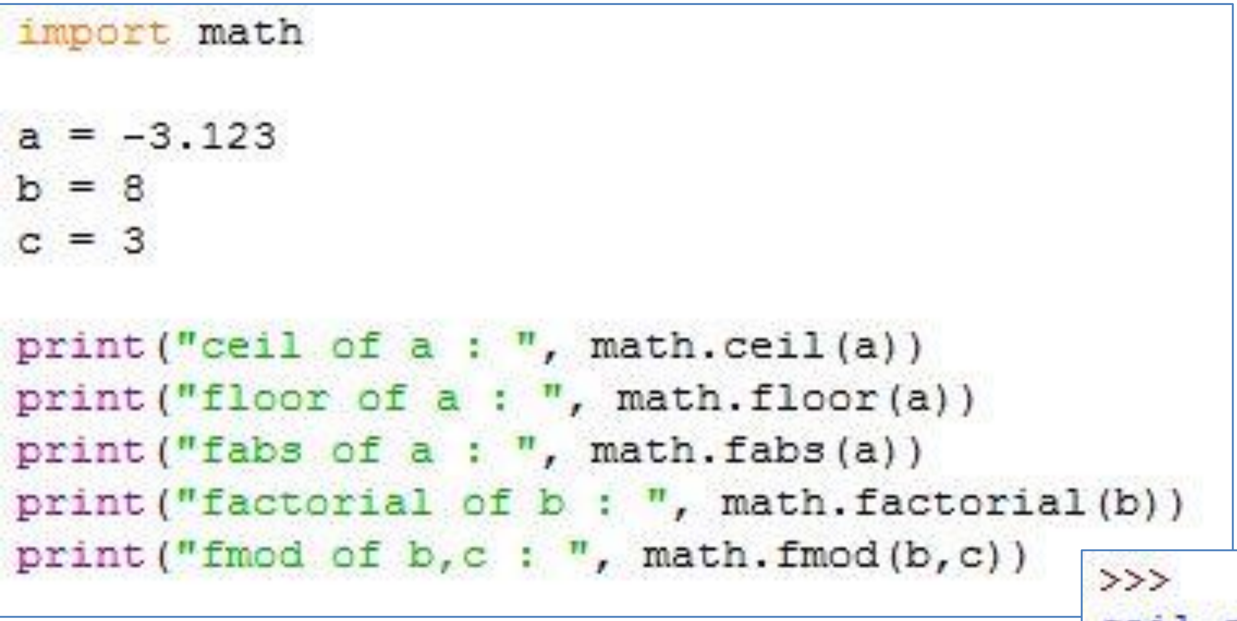
*New in version 3.5.*

`math.`**`nan`**

A floating-point “not a number” (NaN) value. Equivalent to the output of `float('nan')`.

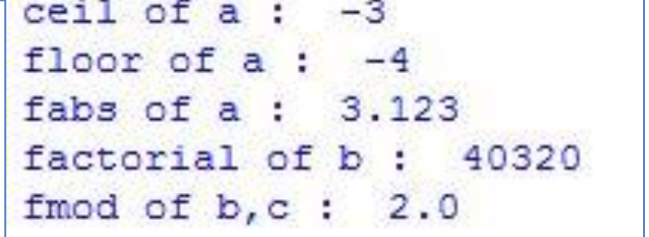
# Functions in Math module [1/2]

- math.**ceil**(x) : Return the smallest integer greater than or equal to x
- math.**floor**(x) : Return the largest integer less than or equal to x
- math.**fabs**(x) : Return the absolute value of x
- math.**factorial**(x) : Return the x factorial
- math.**fmod**(x,y) : Return the remainder of x divided by y



```python
import math

a = -3.123
b = 8
c = 3

print("ceil of a : ", math.ceil(a))
print("floor of a : ", math.floor(a))
print("fabs of a : ", math.fabs(a))
print("factorial of b : ", math.factorial(b))
print("fmod of b,c : ", math.fmod(b,c))
```

**Result**

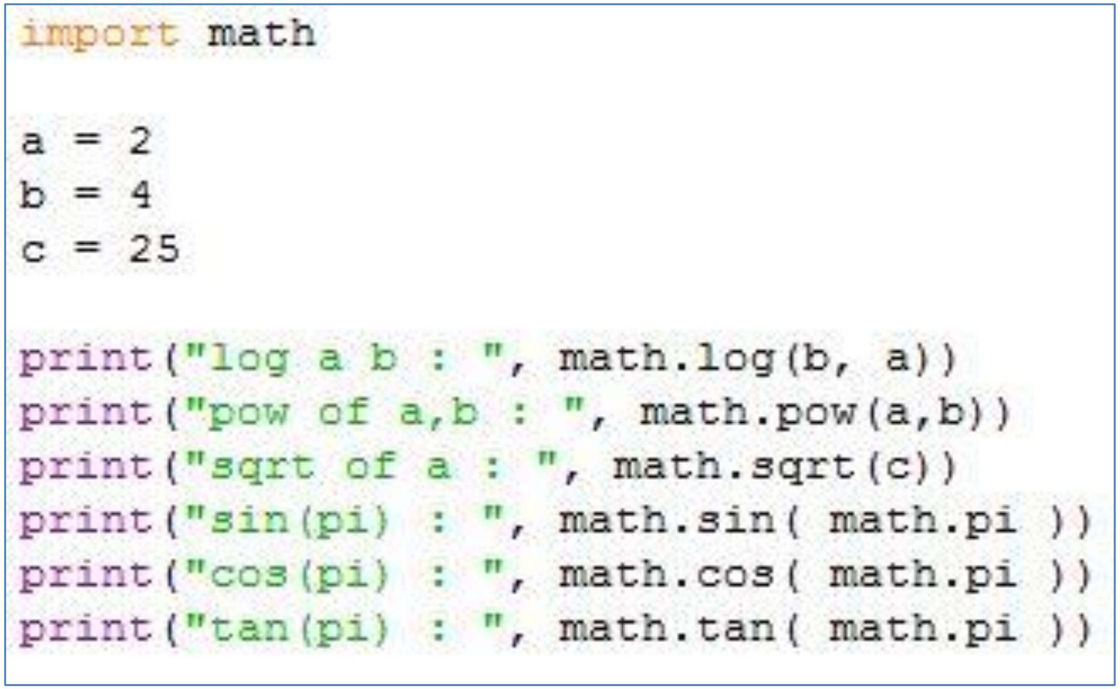


```
>>>
ceil of a :  -3
floor of a :  -4
fabs of a :  3.123
factorial of b :  40320
fmod of b,c :  2.0
```

# Functions in Math module [2/2]

- math.**log**(a, b) : Return the value of $\log_b a$
- math.**pow**(a, b) : Return the a raised to the power of b
- math.**sqrt**(a) : Return the square root of a

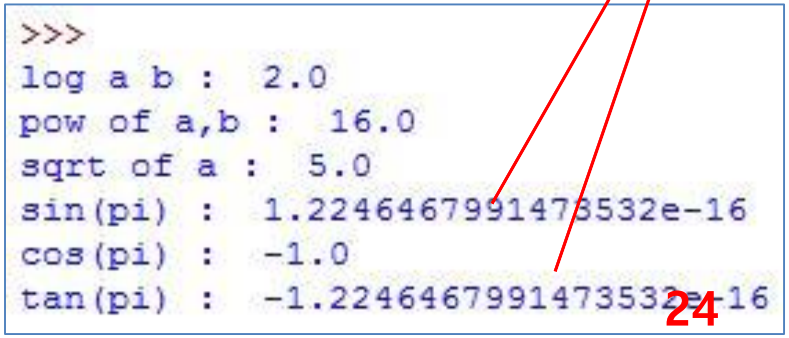
- math.**sin**(x) : Return the sine of x radians
- math.**cos**(x) : Return the cosine of x radians
- math.**tan**(x) : Return the tangent of x radians



```
import math

a = 2
b = 4
c = 25

print("log a b : ", math.log(b, a))
print("pow of a,b : ", math.pow(a,b))
print("sqrt of a : ", math.sqrt(c))
print("sin(pi) : ", math.sin( math.pi ))
print("cos(pi) : ", math.cos( math.pi ))
print("tan(pi) : ", math.tan( math.pi ))
```

**Result**

**Close to 0**



```
>>>
log a b :  2.0
pow of a,b :  16.0
sqrt of a :  5.0
sin(pi) :  1.2246467991473532e-16
cos(pi) :  -1.0
tan(pi) :  -1.2246467991473532e-16
```

# Ch 4   Python Functions

- Function Examples
- Basic Rules of Python Functions
- Functions From Library
- Challenging Python Functions

# Average Computation from User Interaction [1/2]

```
# avg_comp1.py
#    A program to average a set of numbers
#    while loop break using negative input

def avg_comp1():
    sum = 0.0
    count = 0
    x = eval(input("Enter a number (negative to quit) >> "))
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = eval(input("Enter a number (negative to quit) >> "))
    print("\n The average of the numbers is", sum / count)
```

- Assuming there are no negative numbers in the data

# Average Computation from User Interaction [2/2]

```
# avg_comp2.py
#    A program to average a set of numbers
#    While loop break using empty string

def avg_comp2():
    sum = 0.0
    count = 0
    xStr = input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = input("Enter a number (<Enter> to quit) >> ")

  print("\n The average of the numbers is", sum / count)
```

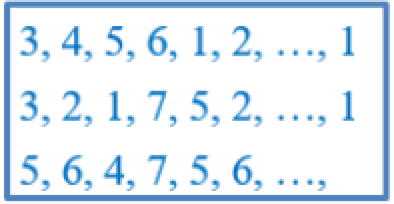
# Average Computation from Data File [1/2]

```
# avg_comp3.py
#     Computes the average of numbers listed in a file

def avg_comp3():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        sum = sum + eval(line)
        count = count + 1
        line = infile.readline()

    print("\n The average of the numbers is", sum / count)
```

```
3
4
7
1
12
```

# Average Computation from Data File [2/2]

- Data is given in CSV file (comma separated file)



3, 4, 5, 6, 1, 2, …, 1
3, 2, 1, 7, 5, 2, …, 1
5, 6, 4, 7, 5, 6, …,

- We use two loops:
  - The top-level loop loops through each line of the file
  - The second-level loop loops through each number of each line

```
# avg_comp4.py
#     Computes the average of numbers listed in a file.
#     Works with multiple numbers on a line.

import string
def avg_comp4():
    fileName = input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    line = infile.readline()
    while line != "":
        for xStr in line.split(","):
            sum = sum + eval(xStr)
            count = count + 1
        line = infile.readline()
    print("\n The average of the numbers is", sum / count)
```

# Palindrome Checker [1/2]

```
# Palindrome:  string을 뒤집어 놓아도 원래와 같은 string
# 부호와 빈칸을 제외하고 대소문자 구분없이 알파벳이 대칭을 이루는 문장
# 예를 들어, 'abcdcba'는 뒤집어도 똑같으므로 palindrome
#  Other Palindrome examples
#      ‘Are we not drawn onward, we few, drawn onward to new era’
#      ‘Do geese see God’
#      ‘Dennis and Edna sinned‘

# Step1:  User로 부터 string을 받아드린다
# Step2:  받아드린 string을 lower case로 바꾼다
# Step3:  string의 첫번째 character를 마지막 character를 p1, p2로 각각  setting
# Step4: p1 < p2인 상태에서는 계속 아래 substep을 수행
#         Step 3.1:  p1과 p2가 가르키는것이 alphabet이 아니면 전진한다
#         Step 3.2:  p1과 p2가 가르키는것이 같은 alphabet이면 Step4로 간다
```

# Palindrome Checker [2/2]

```
def palindrome_checker():
    P_candidate = input("Type your pallindrome candiate: ")
    print ("Here is your pallindrome candiate:", P_candidate)
    P_candidate = P_candidate.lower()
    print ("After lowering characters ==> ", P_candidate)
    #
    isPallindrome_candidate = True
    p1 = 0
    p2 = len(P_candidate) - 1
    #
    while isPallindrome_candidate and p1 < p2:
        if P_candidate[p1].isalpha():
            if P_candidate[p2].isalpha():
                if P_candidate[p1]==P_candidate[p2]:
                    p1 = p1 + 1
                    p2 = p2  - 1
                else:   isPallindrome_candidate = False
            else:  p2 = p2 -  1    # if not alphabet ==> move p2 to left
       else: p1 = p1 + 1           # if not alphabet ==> move p1 to right
#
    if isPallindrome_candidate:
           print ("Yes, your pallindrome candiate", P_candidate,  "is a real pallindrome!“)
    else:  print ("No, your pallindrome candiate", P_candidate,  "is not a real pallindrome!“)
```

# Leap Year Checker

```
# Leap Year (윤년): 1년이 366일
# The rule of leap year follows the definition of Wolfram.com
# Leap years were therefore 45 BC, 42 BC, 39 BC, 36 BC, 33 BC,
# 30 BC, 27 BC, 24 BC, 21 BC, 18 BC, 15 BC, 12 BC, 9 BC, 8 AD, 12 AD,
# and every fourth year thereafter (Tøndering), until the
# Gregorian calendar was introduced (resulting in skipping three out
# of every four centuries). 즉 100 AD, 200 AD, 300 AD은 평년, 400 AD은 윤년


def leap_year_checker():
    target_year = input("Please type your year:")
    leap_year = False
    #
    if target_year in [-45, -42, -39, -33, -30, -27, -24, -21, -18, -15, -12, -9, 8, 12]:
        leap_year = True
    #
    elif target_year > 12 and target_year % 4 == 0:
        leap_year = True
        if target_year % 100 == 0:    leap_year = False
        if target_year % 400 == 0:    leap_year = True
    #
    if leap_year: print ("Yes, the year", target_year, " is a leap year!“)
    else:         print ("No, the year", target_year, " is not a leap year!“)
```

# Valid Date Checker [1/3]

```
# 유효한 날짜는 달력 상 존재하는 날짜를 의미
# valid_date_checker( ) 는 입력된 날짜가 유효하면 valid 를 출력,
# 입력된 날짜가 없거나 입력된 값이 날짜 형태가 아닐 경우 invalid 를 출력
# 예를 들어, -5/12/17은 기원전 5년의 12월 17일을 의미하므로 유효. 하지만 0년은 존재하지 않음

def LeapYear(y):
    year = y
    leap_year = False
    if year in [-45,-42,-39,-33,-30,-27,-24,-21,-18,-15,-12,-9,8,12]:
        leap_year = True
    elif year > 12 and year%4==0:
        leap_year = True
        if year%100==0:   leap_year = False
        if year%400==0:   leap_year = True
    return leap_year

def Month_LastDate(y, m):            # Year와 Month가 주어지면 그 달의 마지막 날짜를 return
    if month in [1, 3, 5, 7 ,8, 10, 12]:
        return 31
    elif month == 2:
        if LeapYear(y):   return 29
        else:             return 28
    else:
        return 30
```

# Valid Date Checker [2/3]

예, 1960/02/29

```
def valid_date_checker():
    Target_Date = input("Type your date in yyyy/mm/dd string format:")
    print (“Your Target Date is:”,  Target_Date)

    try:
        year, month, date = Target_Date.split("/")
        year, month, date = int(year), int(month), int(date)
        print ("Your typed date is:", "Year", year, "Month", month, "Day", date)
        if year == 0:
            print ("Your typed date is invalid“)
        elif month in [1,2,3,4,5,6,7,8,9,10,11,12]:
            daylist = [ ]
            for i in range( Month_LastDate(year, month) ):
                daylist.append( i+1 )                # 그달에 속한 날짜를 List로 만듦
            if date in daylist:
                    print ("Your typed date is valid!“)
            else:   print ("Your typed date is invalid!“)

        else:                                        # month에 이상한값이 입력 되었다면
            print ("Your typed date is invalid“)

    except:                                          # 기타, 모든 비정상적인 입력 data에 대해서는
        print ("Your typed date is invalid“)
```

# Valid Date Checker [3/3]

예, 1960/02/29

```
def valid_date_checker():
    Target_Date = input("Type your date in yyyy/mm/dd string format:")
    print (“Your Target Date is:”,  Target_Date)

    try:
        year, month, date = Target_Date.split("/")
        year, month, date = int(year), int(month), int(date)
        print ("Your typed date is:", "Year", year, "Month", month, "Day", date)

        if (year != 0) and (1 <= month <= 12) and 1 <= date <= Month_LastDate(year, month) :
                    print ("Your typed date is valid!“)
        else:       print ("Your typed date is invalid!“)

     except:                                   # 기타, 모든 비정상적인 입력 data에 대해서는
        print ("Your typed date is invalid“)
```