

Python Functions

- Basics of Python Functions
- Problem Solving using 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
```

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')  
...
```

여러 개의 입력값을 받는 함수 [1/2]

다음의 예를 통해 여러 개의 입력값을 모두 더하는 함수를 직접 만들어 보자. 예를 들어 `sum_many(1, 2)`이면 3을, `sum_many(1,2,3)`이면 6을, `sum_many(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)`이면 55를 돌려주는 함수를 만들어 보자.

```
>>> 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
```

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
```

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)
```

입력 인수에 초깃값 미리 설정하기

```
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)
```

```
myFunc(10)
```

함수 안에서 함수 밖의 변수를 변경하는 방법

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

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

Global 명령어를 이용하기

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

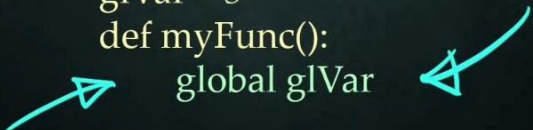
vartest()
print(a)
```

LOCAL VS GLOBAL VARIABLES

GLOBAL: variable that accessible
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()
```


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 creation
- Use triple quotes to enclose

```
def myFunc():
```

```
    """
```

```
    My description
```

```
    """
```

```
>>> print(myFunc.__doc__)
```

My description

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

Python Special Variables

`__doc__`

`__name__`

Predefined Attributes

- 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

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 `true`가 되고 그 아래 문장들이 수행됨

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

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

Python Functions

- Basics of Python Functions
- Problem Solving using Python Functions

Compute molecular weight

```
# Compute molecular weight
# Here are basic weights
# carbon(c): 12.011
# hydrogen(H): 1.0079
# oxygen(O): 15.9994
#
# use round(46.0688, 2) ==> 46.07
```

```
def molecular_weight():
    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_weight_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))
```

Palindrome Checker [1/2]

- # Palindrome이란 철자를 거꾸로 놓아도 원래와 같은 글귀를 말합니다.
- # 부호와 빈칸을 제외하고 대소문자 구분없이 알파벳이 대칭을 이루는 문장
- # 예를 들어, 'abcdcba'는 뒤집어도 똑같으므로 palindrome
- #
- # 조금 더 복잡한 palindrome 예제들
- # 'Are we not drawn onward, we few, drawn onward to new era'
- # 'Do geese see God'
- # 'Dennis and Edna sinned'

Palindrome Checker [2/2]

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

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! ")
```


Euclidean Distance Computation

Euclidean Distance란 직교 좌표계에서 두 점의 거리를 나타냅니다.
예를 들어, 2차원 평면에서 두 점 (x_1, y_1) , (x_2, y_2) 의 거리는
$\text{math.sqrt}((x_1 - x_2)^2 + (y_1 - y_2)^2)$ 로 계산
이와 같이 임의의 차원에서의 거리를 구하는 함수를 구현해보세요.

함수가 받을 parameter는 총 3개로,
첫번째 parameter n은 차원 수, parameter p1, parameter p2는 길이 가 n인 리스트

```
import math
```

```
def eucDist(n, p1, p2):  
    distance = 0  
    for i in range(n):  
        distance = distance + (p2[i]-p1[i])**2  
    #  
    return math.sqrt(distance)
```

```
>> p1 = [2.0 4.0 6.0]  
>> p2 = [1.5 4.5 10.2]  
>> eucDist(3, p1, p2)
```

Math module

- This module provides access to mathematical functions defined in C standard
 - These functions cannot be used with complex numbers (Use cmath)
- These are the categories of the functions
 - Number-theoretic
 - Power and logarithmic
 - Trigonometric
 - Angular
 - Hyperbolic
 - Constant

Math module – Functions [1/3]

- `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

```
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
```

Math module – Functions [2/3]

- `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

```
>>>  
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
```

Close to 0



Math module – Functions [3/3]

- `math.degrees(x)` : Convert angle x from radians to degrees
- `math.radians(x)` : Convert angle x from degrees to radians
- `math.cosh(x)` : Return the hyperbolic cosine of x
- `math.sinh(x)` : Return the hyperbolic sine of x
- `math.tanh(x)` : Return the hyperbolic tangent of x

Result

```
import math

print("degrees of pi : ", math.degrees(math.pi))
print("radians of 180 : ", math.radians(180))
print("cosh of pi : ", math.cosh( math.pi ))
print("sinh of pi : ", math.sinh( math.pi ))
print("tanh of pi : ", math.tanh( math.pi ))
```

```
>>>
degrees of pi : 180.0
radians of 180 : 3.141592653589793
cosh of pi : 11.548739357257746
sinh of pi : 11.591953275521519
tanh of pi : 0.99627207622075
```

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!
# 물 밀도가 가장 높은 점(섭씨 3도에서 5도 사이로 가정)일 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 >= 3 and C <= 5: 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?")
```

Leap Year Checker

```
# 윤년은 1년이 366일로 이루어져 있는 해인데
# 규칙은 다음 Wolfram.com에서 주어진 정의와 같습니다.
# 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).
```

```
def yun_year_checker():
    target_year = input("Please type your year:")
    yun_year = False
    #
    if target_year in [-45, -42, -39, -33, -30, -27, -24, -21, -18, -15, -12, -9, 8, 12]:
        yun_year = True
    #
    elif target_year > 12 and target_year % 4==0:
        yun_year = True
        if target_year % 100==0:
            yun_year = False
            if target_year % 400==0: yun_year = True
    #
    if yun_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 [1/2]

입력된 날짜가 유효할 경우 valid, 입력된 날짜가 유효하지 않거나 입력된 값이 날짜
형태가 아닐 경우 invalid을 출력합니다. 유효한 날짜는 달력 상 존재하는 날짜를
의미합니다. 예를 들어, -5/12/17은 기원전 5년의 12월 17일을 의미하므로 유효합니다.
하지만 0년은 존재하지 않습니다.

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

```
def MonthDate(y, m):  
    if m in [1, 3, 5, 7, 8, 10, 12]:  
        return 31  
    elif m == 2:  
        if LeapYear(y): return 29  
        else: return 28  
    else:  
        return 30
```


Valid Date Checker [2/2]

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

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

    except:
        print ("Your typed date is invalid")
```