

9주차

파이썬 기초 프로그래밍

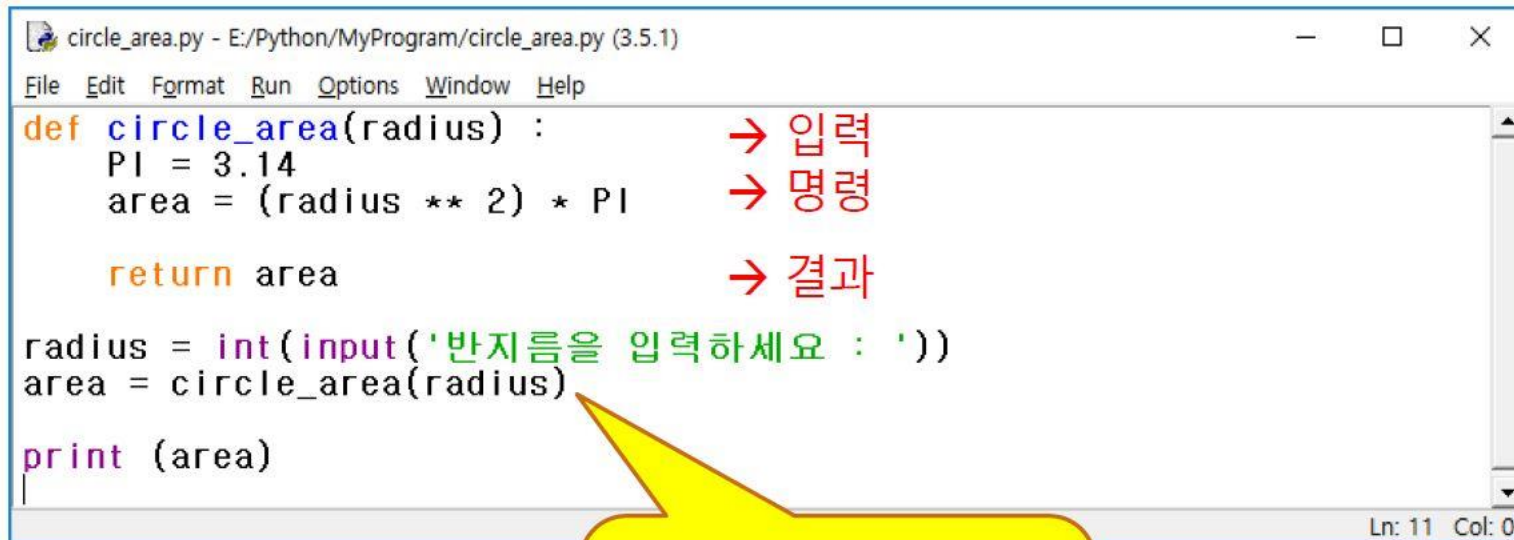


왜 함수를 써야하나?

- 프로그램 개발자 측면
 - 반복되는 코드를 줄일 수 있다.
- 함수 사용자 측면
 - 어떤 과정으로 함수가 작동되는지 몰라도, 함수가 어떤 역할을 하는지 알면 가져다 쓸 수 있다.

함수의 정의 : def()

def 함수명(인수들) :
 실행코드
 return 값



```
circle_area.py - E:/Python/MyProgram/circle_area.py (3.5.1)
File Edit Format Run Options Window Help
def circle_area(radius) :
    PI = 3.14
    area = (radius ** 2) * PI
    return area
radius = int(input('반지름을 입력하세요 : '))
area = circle_area(radius)
print (area)
```

→ 입력
→ 명령
→ 결과

Ln: 11 Col: 0

주의!!
함수 정의를 먼저하고
사용해야 함!!!!

함수의 유형

- 인자 값, 반환 값 없는 함수

```
>>> def myfn1():  
    result = 1 + 2
```

```
>>> print(myfn1())  
None
```

- 인자 값은 있지만, 반환 값은 없는 함수

```
>>> def myfn2(a, b, c):  
    result = a + b + c
```

```
>>> print(myfn2(1, 2, 3))  
None
```

- 인자 값은 없지만, 반환 값은 있는 함수

```
>>> def myfn3():  
    return 10
```

```
>>> print(myfn3())  
10
```

- 인자 값과 반환 값이 모두 있는 함수

```
>>> def myfn4(a, b, c):  
    result = a + b + c  
    return result
```

```
>>> print(myfn4(1, 2, 3))  
6
```

함수에서 여러 개의 값 반환하기(튜플)

```
def function(x, y, z):  
    return x + y, y + x, x + z
```

```
val = function(1, 3, 5)  
print(val)
```

output :
(4, 4, 6)

```
def function(x, y, z):  
    return x + y, y + x, x + z
```

```
val1, val2, val3 = function(1, 3, 5)  
print(val1)  
print(val2)  
print(val3)
```

output :
4
8
6

함수에서 여러 개의 값 반환하기(튜플)

```
def function(x, y, z):  
    return x + y, y + x, x + z
```

```
val1, val2 = function(1, 3, 5)  
print(val1)  
print(val2)
```

output:

```
val1, val2 = function(1, 3, 5)  
ValueError: too many values to unpack (expected 2)
```

▶ return 문 실행

return 문 사용 예

- 1. return sum → sum 값 전달
- 2. return a, b, c → (a, b, c) 전달
- 3. return → None 객체 전달

실습 01

- 두 수를 입력 받아서 둘 중에 작은 수를 반환하여 출력하는 `min()` 함수를 작성 하시오.
 - `def min(a, b) :`
- 두 수를 입력 받아서 둘 중에 큰 수를 반환하여 출력하는 `max()` 함수를 작성 하시오.
 - `def min(a, b) :`

실습 02 참고 : 함수를 사용하지 않는 다각형 그리기

```
from turtle import *

while True :
    n = int(input('다각형의 변을 입력하세요 : '))
    if n == 0 : break
    if n <= 2 :
        print('3 이상의 수를 입력하세요')
    else :
        clear()
        angle = 360 / n
        for i in range(n) :
            forward(100)
            left(angle)
```

실습 02

- 한 변의 길이와 다각형 각의 수 n 을 인수로 전달받아 지정한 크기의 다각형을 그리는 함수를 작성 하시오.
- `def polygon(side, n) :`

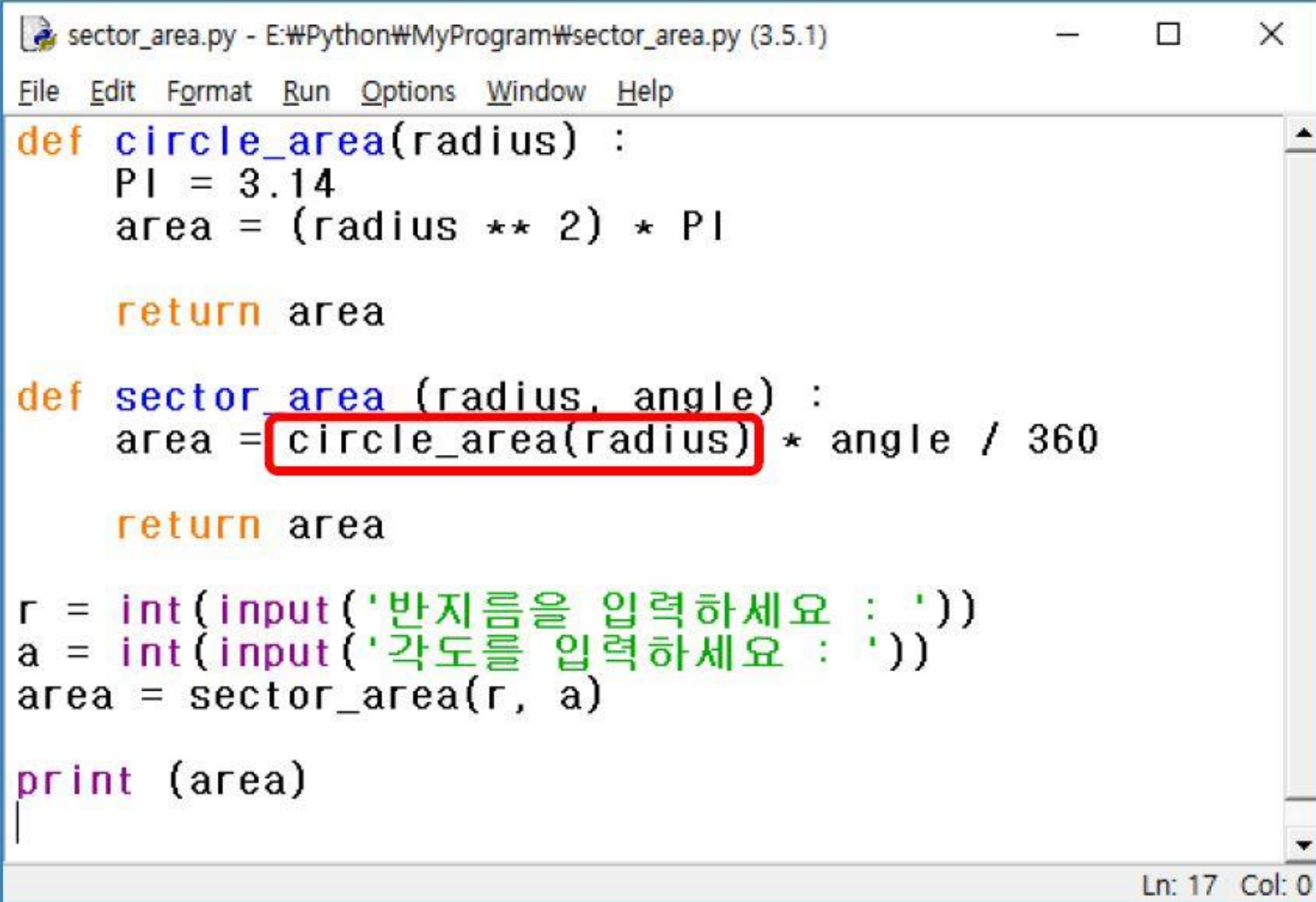
실습 03

- 변의 길이와 각의 수를 입력 받아 다각형을 그리는 `polygon()` 함수를 이용하여, 세번째 인수로 색상을 지정하는 경우, 지정된 색으로 다각형을 채우는 `polygon()` 함수를 작성하시오.
 - 예 : `polygon(100, 5)` -> 한변이 100인 5각형 그리기
`polygon(70, 6, 'red')` -> 한변의 길이가 70인 6각형을 그리고, 빨간색으로 내부 채우기
 - `def polygon(side, n, *args)` :

함수를 호출하는 함수

- 부채꼴의 넓이 구하기
 - 부채꼴의 넓이 = 원의 넓이 * 각도 / 360
- 이미 구현한 원의 넓이를 구하는 함수를 이용하여 부채꼴의 넓이를 구할 수 있다.

함수를 호출하는 함수



```
sector_area.py - E:\Python\MyProgram\sector_area.py (3.5.1)
File Edit Format Run Options Window Help
def circle_area(radius) :
    PI = 3.14
    area = (radius ** 2) * PI

    return area

def sector_area (radius, angle) :
    area = circle_area(radius) * angle / 360

    return area

r = int(input('반지름을 입력하세요 : '))
a = int(input('각도를 입력하세요 : '))
area = sector_area(r, a)

print (area)
|
```

Ln: 17 Col: 0

함수의 인수 : 키워드 인수

- 위치 기반으로 값을 전달하지 않고 인수의 이름으로 값을 전달

```
>>> def area(height, width) :  
        return height * width  
>>> area(width=20, height=10) # 순서가 아닌 이름으로 값을 전달  
200  
>>> area(20, width=5)  
100  
>> area(height=5, 10)  
SyntaxError: positional argument follows keyword argument  
>>>
```

함수의 인수 : 가변 인수

- 고정되지 않은 수의 인수를 함수에 전달하는 방법
- 함수 정의에서 반드시 넘겨야 하는 고정 인수를 먼저 나열하고, 나머지는 튜플 형식으로 한꺼번에 받는다.
- 가변 인수는 *변수명의 형식으로 인수 목록 마지막에 하나만 표현
- 예 : range() 함수

```
>>> def varg(a, *arg) :  
        print(a, arg)  
>>> varg(1)  
1 ()  
>>> varg(2,3)  
2 (3,)  
>>> varg(2,3,4,5)  
2 (3, 4, 5)  
>>>
```

Packing

- 인자의 갯수를 제한하지 않고, 다수의 인자를 받을 수 있음
- 다수의 Positional Arguments를 하나의 **tuple**로서 받을 수 있음 (**packing**)

```
def fn2(*colors): # 0개 이상의 인자를 받을 수 있음.  
    for color in colors:  
        print(color)
```

```
fn2()
```

```
fn2('white')
```

```
fn2('white', 'yellow')
```

```
fn2('white', 'yellow', 'black', 'pink')
```


Packing

```
def fn3(color1, color2, *other_colors):    # 2개 이상의 인자를 강요
    print('color1 :', color1)
    print('color2 :', color2)
    for color in other_colors:
        print(color)
```

```
fn3('brown', 'green')                    # 최소 2개의 인자 지정이 필요
fn3('brown', 'green', 'white')
fn3('brown', 'green', 'white', 'yellow')
```

Unpacking

- 인자를 넘길 때 **Sequence** Data Type (리스트/튜플 등) 을 다수의 인자인 것처럼 나눠서 전달 가능 (**unpacking**)

```
colors = ['white', 'yellow', 'black']  
fn2(*colors)  
fn2('brown', 'pink', *colors)
```

```
other_colors = ('violet', 'coral', 'cyan')  
fn2('brown', 'pink', *colors, *other_colors)
```

```
fn3('purple', *('aqua', 'beige', 'black'))  
fn3('purple', *['aqua', 'beige', 'black'])
```

Unpacking

- 가변인자없이 tuple/list 인자 1개로서 전달할 수도 있으나, 함수가 원하는 인자가 명확하게 드러나지 않음.

```
def fn1(colors): # 인자 1개로 받습니다.  
    for color in colors:  
        print(color)
```

```
fn1(['white', 'yellow', 'black'])  
fn1(['white', 'yellow', 'black', 'pink'])  
fn1(['white', 'yellow', 'black', 'pink', 'aqua'])
```

튜플, 딕셔너리 인수

- 튜플 인수
 - 함수 호출에 사용하는 인수들이 튜플에 있는 경우
 - *을 이용하여 함수 호출 가능
- 사전 인수
 - 함수 호출에 사용하는 인수들이 딕셔너리에 있는 경우
 - **을 이용하여 함수 호출 가능

튜플, 딕셔너리 인수 : Unpacking

```
>>> def h(a, b, c):  
    print(a, b, c)  
  
>>> args=(1, 2, 3)  
>>> h(*args)  
1 2 3  
>>> dargs={'a':1, 'b':2, 'c':3}  
>>> h(**dargs)  
1 2 3  
>>> args = (1, 2)  
>>> dargs={'c':3}  
>>> h(*args, **dargs)  
1 2 3  
>>>
```

실행해 보고 의미를 알아 봅시다

```
def sumOfListElements(list1) :  
    result = 0  
    for num in list1 :  
        result += num  
    return result
```

```
exList = [1,2,3,4,5]  
print(sumOfListElements(exList))
```

```
def sumOfDictElements(dic) :  
    result = 0  
    for key, value in dic.items() :  
        result += value  
    return result
```

```
exDic = {'a':1, 'b':2, 'c':3}  
print(sumOfDictElements(exDic))
```

```
def sumOfList(*list2) :  
    result = 0  
    for i in list1 :  
        result += i  
    return result
```

```
a = [1,2,3,4,5]  
print('sum of list', sumOfList(*a))  
print(sumOfList(1,2,3,4,5))
```

변수의 유효 범위(scope)

- ▶ 변수의 유효 범위 규칙
 - ▶ 변수가 유효하게 사용되는 범위(이름 공간:Naming Space)를 정하는 규칙
 - ▶ LEGB
 - ▶ L (Local) : 함수 안
 - ▶ E (Enclosing Function Local) : 함수를 내포하는 또 다른 함수 영역
 - ▶ G (Global) : 모듈 영역
 - ▶ B (Built-in) : 내장 영역

변수의 유효 범위 예

```
>>> x = 10                # G에 해당
>>> y = 11
>>> def foo():
    x = 20                # foo 함수의 L, bar 함수의 E에 해당
    def bar():
        a = 30            # L에 해당
        print(a, x, y)    # 각 변수는 L, E, G에 해당
    bar()                 # 30 20 11 출력
    x = 40                #
    bar()                 # 30 40 11 출력

>>> foo()
```


변수의 유효 범위 : global

- global 선언자 : 변수가 전역 변수임을 선언

```
>>> g = 10
>>> def f():
    global g          # g는 전역 변수
    a = g             # 전역 변수 g 참조
    g = 20            # 전역 변수 g 값 변경
    return a

>>> f()
10
```

변수의 유효 범위 : nonlocal

- nonlocal 선언자
 - E 영역에 변수 선언
 - 가장 가까운 이름 공간부터 변수를 찾음

```
>>> def outer() :  
    x = 1  
    def inner() :  
        nonlocal x # 함수 outer의 x 사용  
        x = 2      # 함수 inner의 지역변수 아님!!!  
        print('inner : ', x)  
    inner()  
    print('outer : ', x)  
  
>>> outer()  
inner : 2  
outer : 2
```

재귀적(Recursive) 프로그래밍

- 함수 내부에서 자기 자신을 호출하는 것
- 점화식

▶ $\text{sum}(N) = \text{sum}(N-1) + n$

▶ $\text{sum}(1) = 1$

```
>>> def sum(N) :  
        if N == 1 : return 1  
        return N + sum(N-1)
```

```
>>> sum(10)  
55
```

실습 04

- $n!$ 은 1부터 n 까지의 정수들의 곱을 나타낸다. n 을 인자로 받아 $n!$ 값을 계산하여 돌려주는 `factrial()`을 작성하시오.
- `def factrial(n) :`

결과

```
>>> factorial(3)
6
>>> factorial(7)
5040
>>> factorial(5)
120
```

람다(lambda) 함수

- 이름이 없는 한 줄짜리 함수
- lambda <인수들> : <반환할 식>

```
>>> f = lambda:1
>>> f()
1
>>> g = lambda x, y : x+y
>>> g(1,2)
3
>>> incr = lambda x, inc=1 : x + inc
>>> incr(10)                # inc 기본 인수값으로 1 사용
11
>>> incr(10, 5)
15
>>> vargs = lambda x, *args : args
>>> vargs(1, 2, 3, 4, 5)
(2, 3, 4, 5)
```

실습 05

- 인수를 전달받은 연도가 윤년이면 True를, 윤년이 아니면 False를 반환하는 함수를 작성 하시오.
 - 함수 안에서 별도의 키보드 입력을 받지 않습니다. 인수로 전달받은 year에 대해서 한번만 판단하는 함수를 작성하시요

```
print('[1번] isLeapYear 함수 테스트')
# 윤년 테스트
for year in [0, 4, 400]:
    if isLeapYear(year):
        print('%5d' % year, '{0:->30}'.format('success'))
    else:
        print('%5d' % year, '{0:->30}'.format('fail'))

# 평년 테스트
for year in [1, 100, 7]:
    if isLeapYear(year):
        print('%5d' % year, '{0:->30}'.format('fail'))
    else:
        print('%5d' % year, '{0:->30}'.format('success'))
```

과제

- 실습 01, 02, 03, 04. 05를 작성하시오.
 - lab01.py, lab02.py, lab03.py, lab04.py, lab05.py