고급프로그래밍및실습

/ 5. 함수

-

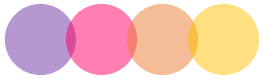
이 정 진

조교수, 숭실대 글로벌미디어학부

jungjinlee@ssu.ac.kr, 정보과학관 623호

학습 목표

- 함수의 개념을 학습합니다.
- 함수를 작성하는 방법을 학습합니다.
- 함수를 호출하여 사용하여 방법을 학습합니다.



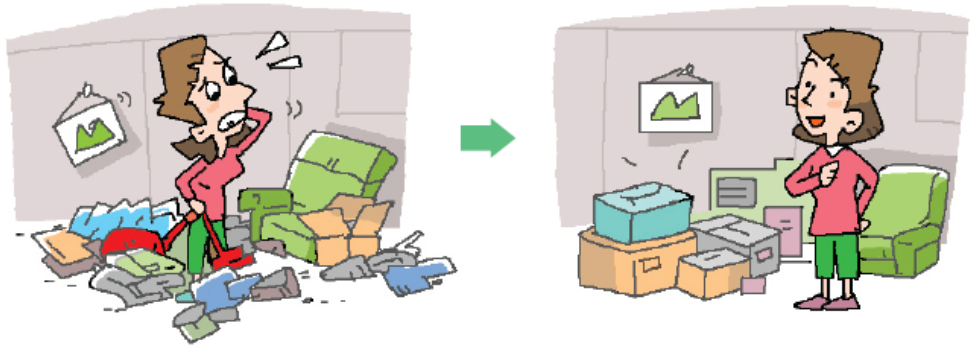
함수

함수란?

함수란?

코드를 묶는 방법

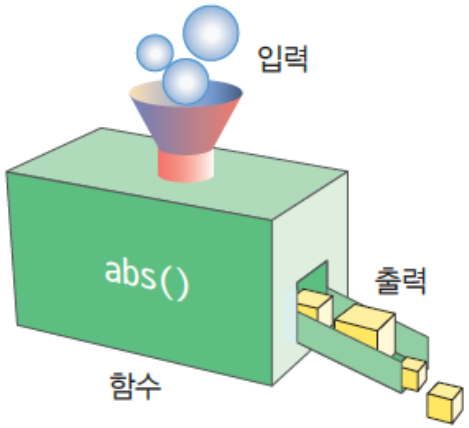
- 관련 있는 코드들을 묶어서 전체 프로그램을 조직화할 필요가 있다.
- 코드를 묶는 3가지의 방법
  - 함수(function)는 우리가 반복적으로 사용하는 코드를 묶은 것으로 프로그램의 빌딩 블록과 같다.
  - 객체(object)는 서로 관련 있는 변수와 함수를 묶는 방법이다.
  - 모듈(module)은 함수나 객체들을 소스 파일 안에 모은 것이다

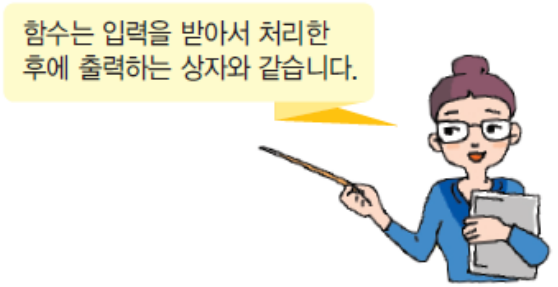


함수란?

함수

- 함수(function)는 특정 작업을 수행하는 명령어들의 모음에 이름을 붙인 것이다.
- 함수는 작업에 필요한 데이터를 전달받을 수 있으며, 작업이 완료된 후에는 작업의 결과를 호출자에게 반환할 수 있다.





함수의 필요성

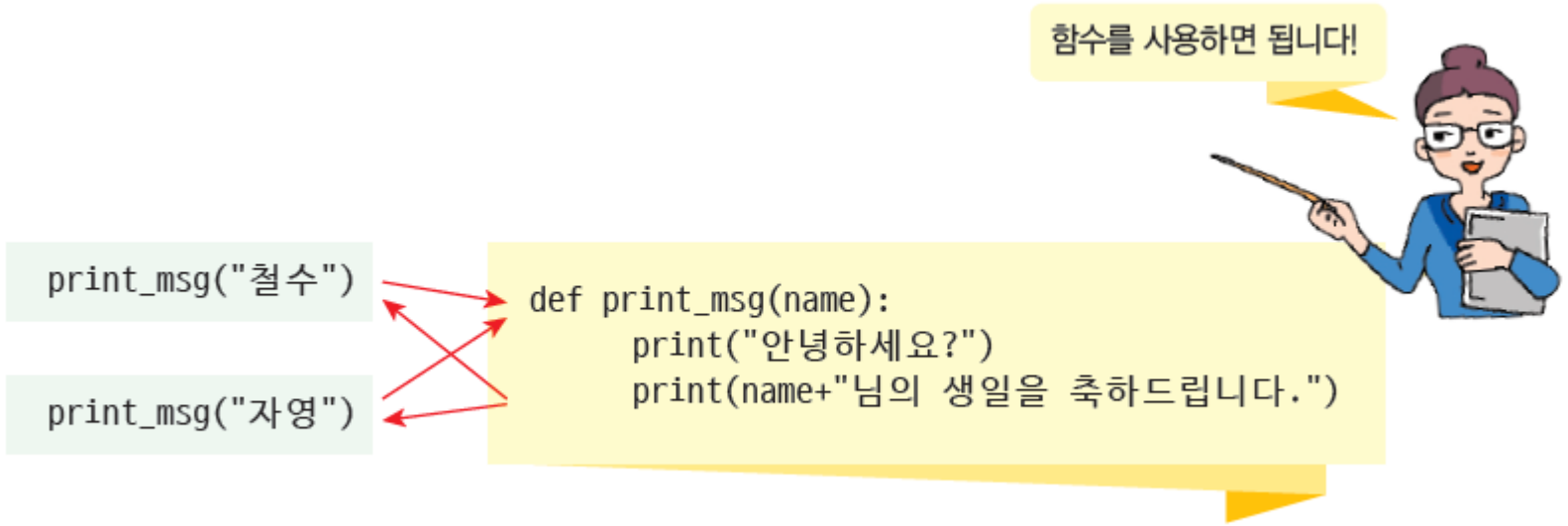
- 프로그램을 작성하다 보면 동일한 처리를 반복해야 하는 경우가 많이 발생한다.
- 이런 경우에는, 이미 작성한 코드를 재활용하여 사용할 수 있으면 정말 좋을 것이다.


```
print("안녕하세요?")
print("철수님의 생일을 축하드립니다.")
```

```
print("안녕하세요?")
print("자영님의 생일을 축하드립니다.")
```

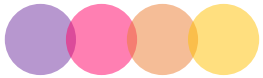
함수를 사용하는 경우

- 함수를 이용하면 우리가 여러 번 반복해야 되는 처리 단계를 하나로 모아서 필요할 때 언제든지 호출하여 사용할 수 있다.



중간점검

1. 함수란 무엇인가?
2. 함수를 사용하는 이유는 무엇인가?



함수

함수 작성하고 호출하기

함수 작성하고 호출하기

Syntax: 함수 정의

형식

```
def 함수이름(매개변수1, 매개변수2, ...) :
    명령문1
    명령문2
```

예

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

함수 이름

매개 변수

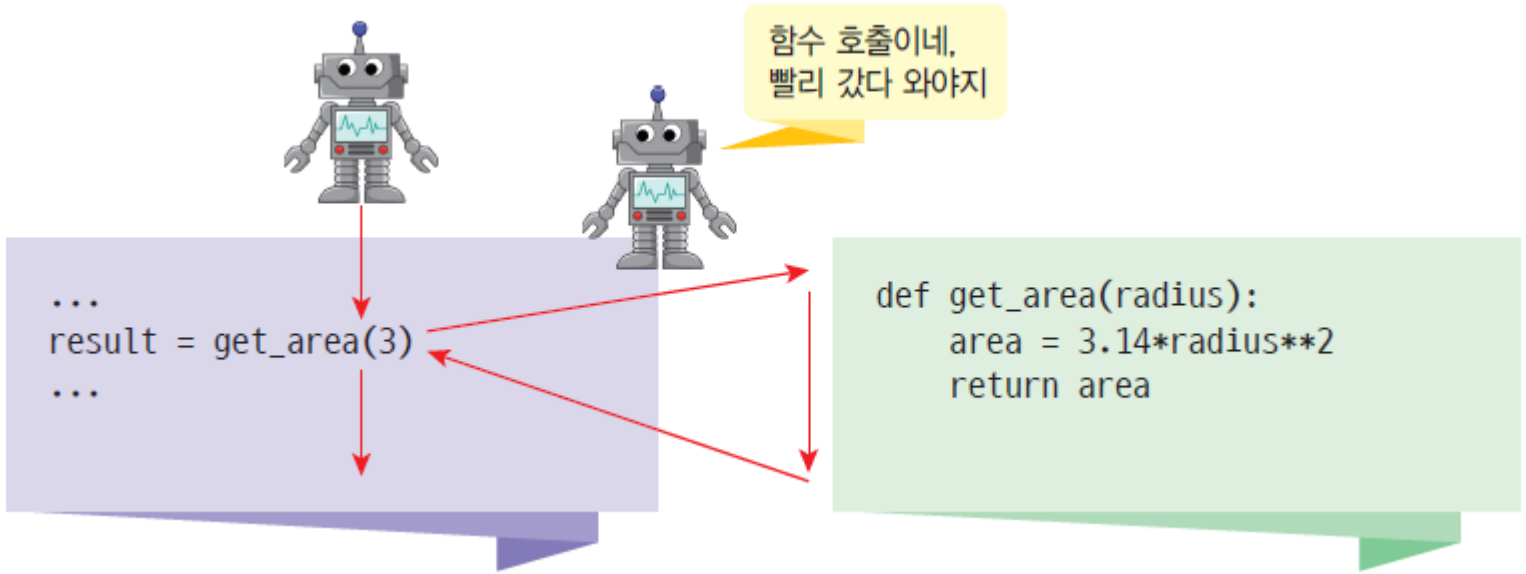
함수 헤더

함수 몸체

return 문장은 함수를 종료시키고 결과를 반환한다.

함수 호출

- 함수 호출(function call)이란 get_area()과 같이 함수의 이름을 써주는 것이다. 함수가 호출되면 함수 안에 있는 문장들이 실행되며 실행이 끝나면 호출한 위치로 되돌아간다.



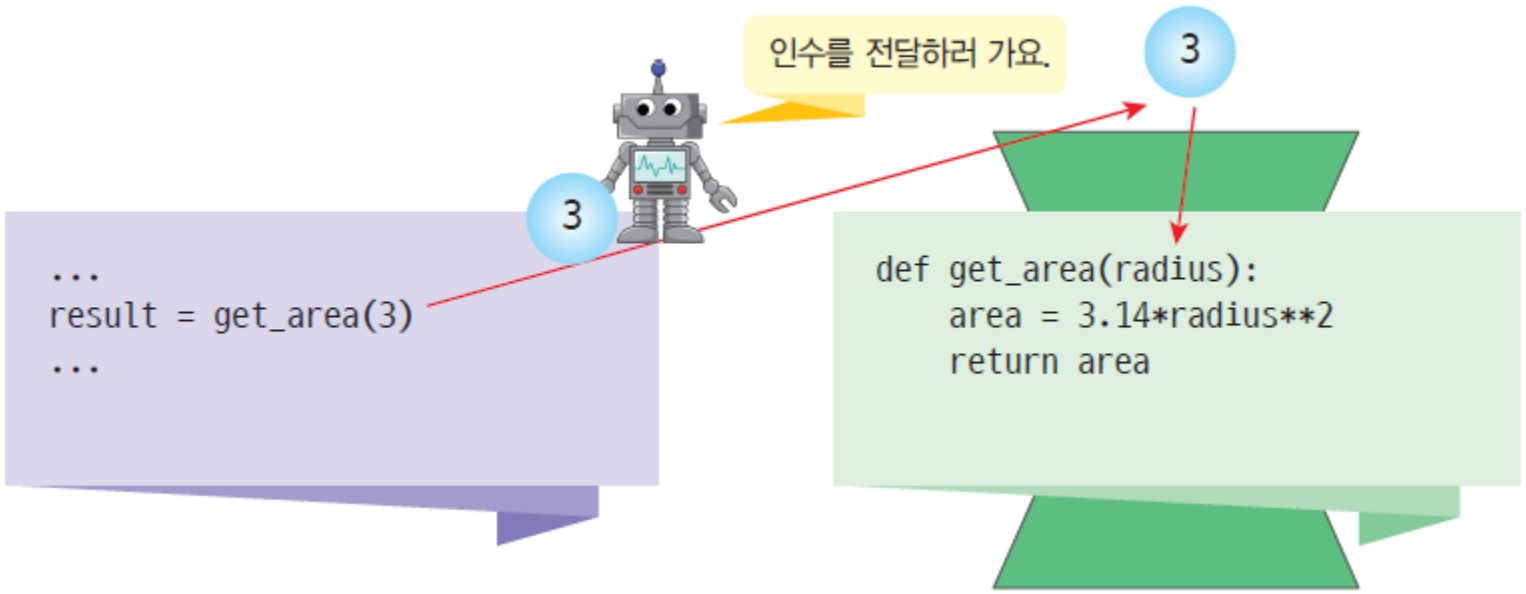
함수는 몇 번이고 호출될 수 있다.

- 함수는 일단 작성되면 몇 번이라도 호출이 가능하다.

```
...
x = get_area(3)
...
y = get_area(20)
```

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

인수


인수를 전달하러 가요.
3
3
...
result = get_area(3)
...
def get_area(radius):
area = 3.14*radius**2
return area

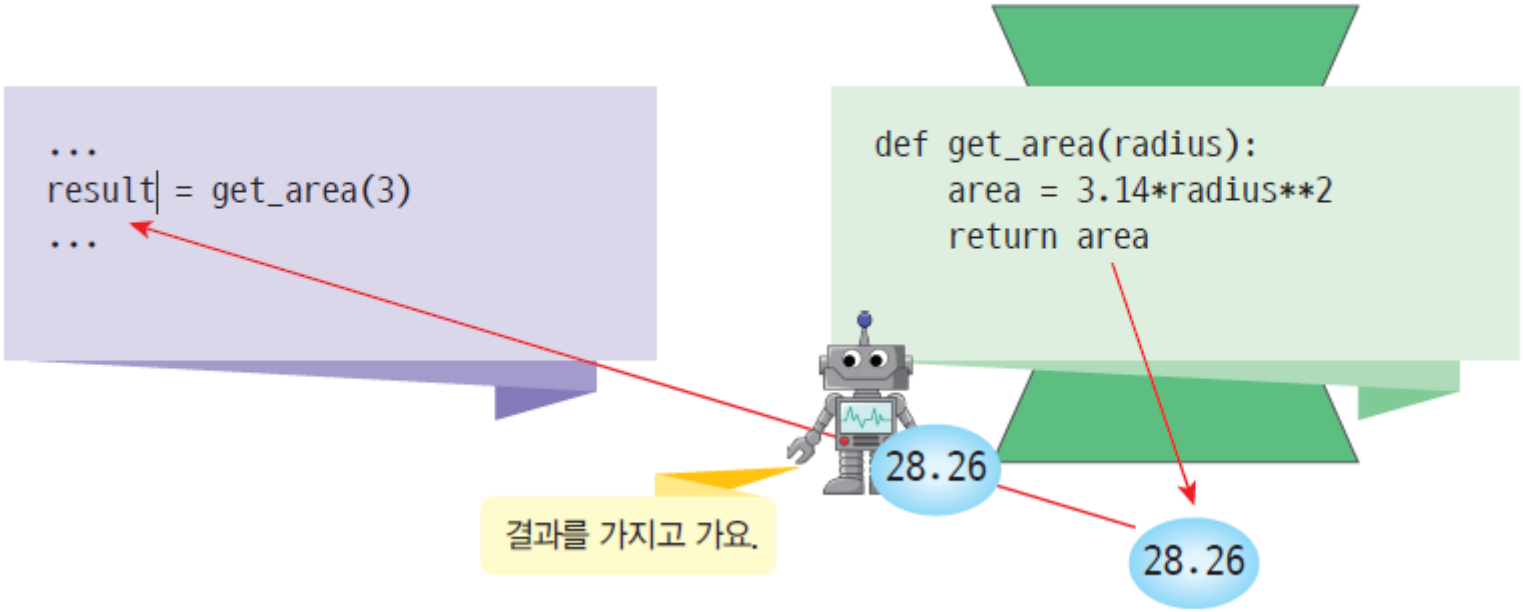
예제

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

result = get_area(3)
print("반지름이 3인 원의 면적=", result)
```

```
반지름이 3인 원의 면적= 28.26
```

함수 작성하고 호출하기

값 반환하기



서로 다른 인수로 호출될 수 있다.

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

result1 = get_area(3)
result2 = get_area(20)

print("반지름이 3인 원의 면적=", result1)
print("반지름이 20인 원의 면적=", result2)
```

```
반지름이 3인 원의 면적= 28.26
반지름이 20인 원의 면적= 1256.0
```

여러 개의 값 반환하기

- 파이썬에서는 함수가 여러 개의 값을 반환할 수 있다. 다음과 같은 형식을 사용한다. 이것은 7장에서 학습하는 튜플을 통하여 이루어진다.

```
def get_input():
        return 2, 3

x, y = get_input()          # x는 2이고 y는 3이다.
```

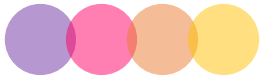
함수의 몸체는 나중에 작성할 수 있다.

- 파이썬에서 함수의 헤더만 결정하고 몸체는 나중에 작성하고 싶은 경우에는 pass 키워드를 사용할 수 있다.

```
def sub():
        pass
```

중간점검

1. 함수에 전달되는 값을 무엇이라고 하는가?
2. 함수 안에서 전달되는 값을 받는 변수를 무엇이라고 하는가?
3. 사용자로부터 2개의 정수를 받아서 반환하는 함수를 작성해보자.



함수

여러 개의 함수가 있는 프로그램

함수의 순서

- 파이썬은 인터프리트 언어이기 때문에 함수의 순서가 중요하다.

```
result = get_area(3)
print("반지름이 3인 원의 면적=", result)

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

정의되지 않은 함수를 사용하였으므로 오류!!

함수의 순서

- 그러나 함수 내에서는 아직 정의되지 않은 함수를 호출할 수는 있다.

```
def main() :
    result1 = get_area(3)
    print("반지름이 3인 원의 면적=", result1)

def get_area(radius):
    area = 3.14*radius**2
    return area

main()
```

함수 안에서는 정의되지 않은 다른 함수를 호출하여도 된다.

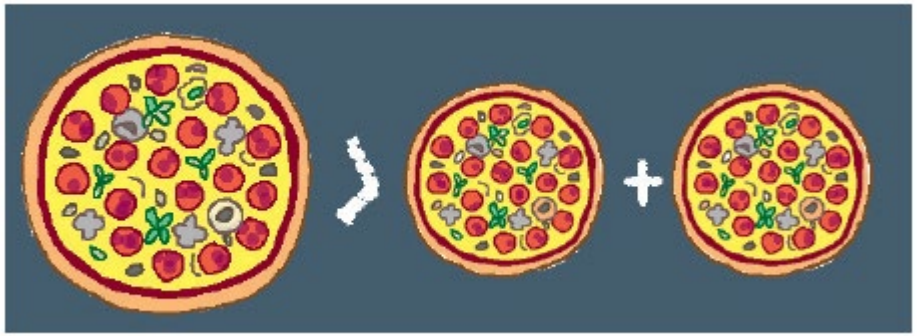
중간점검

1. 주어진 사각형의 면적을 계산하는 함수 get_rect_area(w, h)를 정의해보자. 여기서 w는 너비, h는 높이이다.

2. main() 함수를 정의하고 main() 함수 안에서 get_rect_area()를 호출해보자.

Lab: 피자 크기 비교

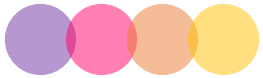
- 원의 반지름을 받아서 피자의 면적을 계산하는 함수를 작성해서 사용해 보자.



```
20cm 피자 2개의 면적: 2512.0
30cm 피자 1개의 면적: 2826.0
```

Solution:

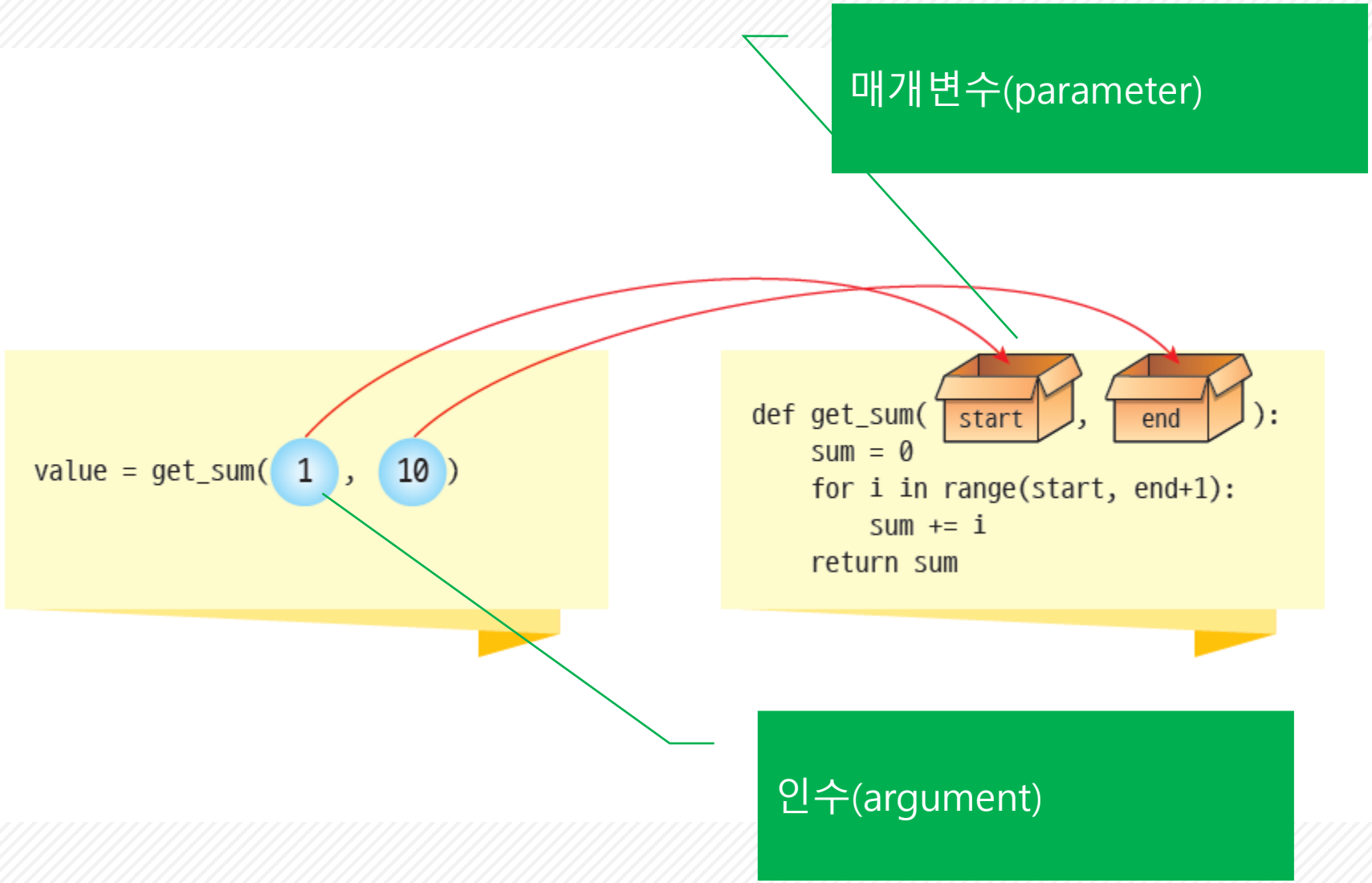
```
def main() :
    print("20cm 피자 2개의 면적:", get_area(20)+get_area(20))
    print("30cm 피자 1개의 면적:", get_area(30))

## 원의 면적을 계산한다.
# @param radius 원의 반지름
# @return 원의 면적
#
def get_area(radius) :
    if radius > 0 :
        area = 3.14*radius**2
    else :
        area = 0
    return area

main()
```



함수

매개 변수 전달

매개변수 전달


매개변수(parameter)
value = get_sum(1 , 10)
def get_sum(start , end):
sum = 0
for i in range(start, end+1):
sum += i
return sum
인수(argument)

매개 변수 전달

서로 다른 인수로 호출될 수 있다.

```
def get_sum(start, end):
        sum = 0
        for i in range(start, end+1):
                sum += i
        return sum



# 1과 10이 get_sum()의 인수가 된다.
x = get_sum(1, 10)

# 1과 20이 get_sum()의 인수가 된다.
y = get_sum(1, 20);
```

매개 변수를 변경한다고 해서 인수가 변경되지 않는다.

```
def set_radius(radius):
        radius = 100
        return

r = 20
set_radius(r)
print(r)
```

```
20
```

디폴트 인수

- 파이썬에서는 함수의 매개 변수가 기본값을 가질 수 있다. 이것을 디폴트 인수(default argument)라고 한다.

```
def greet(name, msg="별일없죠?"):
        print("안녕 ", name + ', ' + msg)

greet("영희")
```

```
안녕 영희, 별일없죠?
```

키워드 인수

- 키워드 인수(keyword argument)는 키워드 인수는 인수의 이름을 명시적으로 지정해서 값을 매개 변수로 전달하는 방법이다.

```
def sub(x, y, z):
        print("x=", x, "y=", y, "z=", z)

>>> sub(10, 20, 30)
x= 10 y= 20 z= 30

>>> sub(x=10, y=20, z=30)
x= 10 y= 20 z= 30

>>> sub(10, y=20, z=30)
x= 10 y= 20 z= 30

>>> sub(x=10, 20, 30)
sub(x=10, 20, 30)
^
SyntaxError: positional argument follows keyword argument
```

매개 변수 전달

가변 인수

- 파이썬에서는 가변 인수도 허용한다.

```
def varfunc(*args ):
        print (args)

print("하나의 값으로 호출")
varfunc(10)

print("여러 개의 값으로 호출")
varfunc(10, 20, 30)
```

```
하나의 값으로 호출
(10,)
여러 개의 값으로 호출
(10, 20, 30)
```

예제

```
def add(*numbers) :
        sum = 0
        for n in numbers:
                sum = sum + n
        return sum

print(add(10, 20))
print(add(10, 20, 30))
```

```
30
60
```

예제

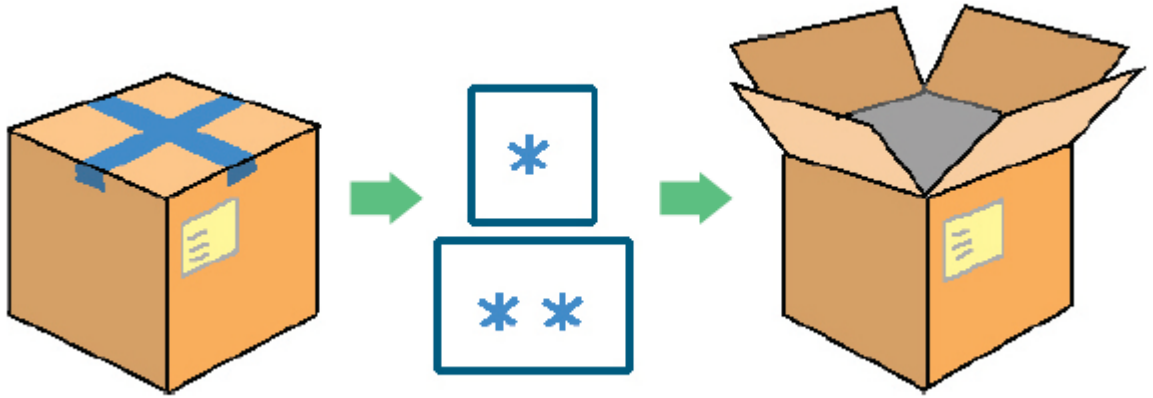
- 매개 변수 이름 앞에 이중 별표(**)를 사용하여 가변 길이 키워드 인수를 나타낸다.
- 인수는 딕셔너리 형태로 전달된다

```
def myfunc(**kwargs):
        result = ""
        for arg in kwargs.values():
                result += arg
        return result


print(myfunc(a="Hi!", b="Mr.", c="Kim"))
```

```
Hi!Mr.Kim
```

* 연산자로 언패킹하기

- 단일 별표 연산자 *는 파이썬이 제공하는 모든 반복 가능한 객체(iterable)을 언패킹할 수 있고 이중 별표 연산자 **는 딕셔너리 객체를 언패킹할 수 있다.



예제

```
>>> alist = [ 1 , 2 , 3 ]
>>> print(*alist)
1 2 3


>>> alist = [ 1 , 2 , 3 ]
>>> print(alist)
[1, 2, 3]
```

예제

```
def sum(a, b, c):
        print(a + b + c)


alist = [1, 2, 3]
sum(*alist)
```

```
6
```

중간점검

1. 인수와 매개 변수는 다시 한번 설명해보자.
2. 디폴트 인수란 무엇인가? 예를 들어보자.
3. 키워드 인수란 무엇인가? 예를 들어보자.
4. 매개 변수 앞에 * 기호가 있다면 무슨 의미인가?

Lab: 환영 문자열 출력 함수

- 전광판에 “환영합니다” 문자열을 여러 번 출력하는 함수 display(msg, count)를 작성해보자.

```
환영합니다.
환영합니다.
환영합니다.
환영합니다.
환영합니다.
```

Solution:

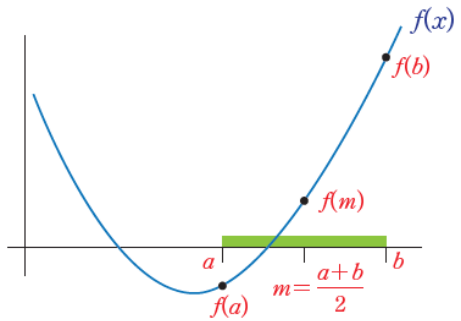
```
#           이 프로그램은 메시지를 반복하여 출력한다.

def display(msg, count=1) :
        for k in range(count) :
                print(msg)

display("환영합니다.", 5)
```

매개 변수 msg도 "Welcome"이라는 디폴트 값을 가지도록 함수를 정의해보자.

display() 함수를 가변 길이 인수로 다시 작성할 수 있는가?

Lab: 이분법

- 구간 [a, b]에서 f(a)f(b) < 0이라고 하자. f(a)f(b) < 0이면 함수 f는 반드시 구간 [a,b]에서 근을 가져야 한다. 계속해서 a와 b의 중간값 m=(a+b)/2을 계산하고, 함수 f(m)의 값을 계산한다.
- f(a)f(m) < 0이면 근은 [a, m] 사이에 있고, 그렇지 않으면 근은 [m, b] 구간에 있을 것이다




```
x**2-x-1의 근: 1.6180419921875
```

매개 변수 전

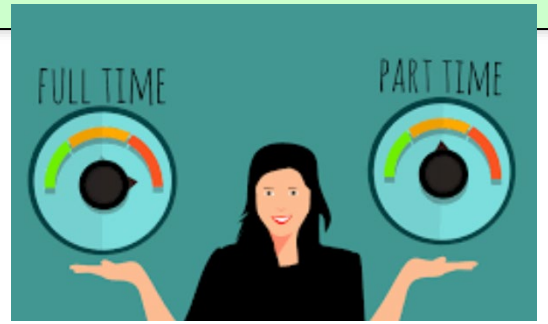
Solut

```
# 함수를 정의한다.
def f(x):
    return(x**2-x-1)

def bisection_method(a, b, error):
    if f(a)*f(b) > 0:
        print("구간에서 근을 찾을 수 없습니다.")
    else:
        while (b - a)/2.0 > error:          # 오차를 계산한다.
            midpoint = (a + b)/2.0          # 중점을 계산한다.
            if f(midpoint) == 0:
                return(midpoint)
            elif f(a)*f(midpoint) < 0:
                b = midpoint
            else:
                a = midpoint

        return(midpoint)

answer = bisection_method(1, 2, 0.0001)

print("x**2-x-1의 근:", answer)
```

Lab: 주급 계산 프로그램

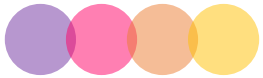
- 주단위로 봉급을 받는 알바생이 있다고 하자. 현재 시급과 일한 시간을 입력하면 주급을 계산해주는 함수 weeklyPay(rate, hour)를 만들고 이 함수를 호출하여 주급을 출력하는 프로그램을 작성해보자. 30시간이 넘는 근무 시간에 대해서는 시급의 1.5배를 지급한다고 하자.

```
시급을 입력하시오:10000
근무 시간을 입력하시오:38
주급은 420000.0
```



Solution:

```
#           이 프로그램은 주급을 계산한다.

def weeklyPay( rate, hour ):
    money = 0
    if (hour > 30):
        money = rate*30 + 1.5*rate*(hour-30)
    else:
        money = rate*hour
    return money

rate = int(input("시급을 입력하시오:"))
hour = int(input("근무 시간을  입력하시오:"))
print("주급은 " + str(weeklyPay(rate, hour)))
```



함수

값 반환하기

값 반환하기

- 모든 경우에 값을 반환하는 것이 좋다.

```
def get_area(radius) :
    if radius > 0 :
            return 3.14*radius**2       # radius가 음수일 때는 아무것도 반환되지 않는다.
```

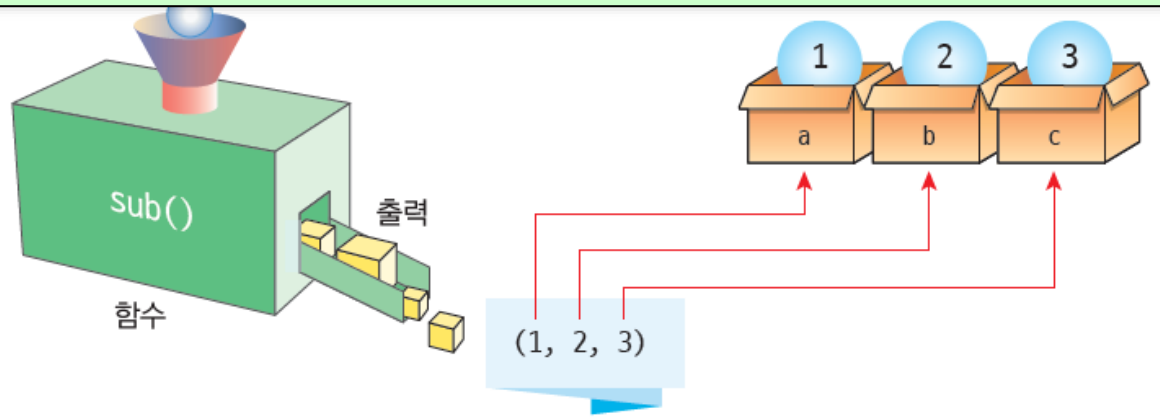
```
def get_area(radius) :
    if radius > 0 :
            return 3.14*radius**2
    else :
            return 0
```

여러 개의 값 반환하기

- 파이썬에서는 함수가 하나 이상의 값도 반환할 수 있다.

```
def sub():
        return 1, 2, 3

a, b, c = sub()
print(a, b, c)
```

```
1 2 3
```



중간점검

1. 값을 반환하는 키워드는 무엇인가?
2. x와 y를 받아서 x+y, x-y 값을 반환하는 함수 addsub(x, y)를 정의해 보자.

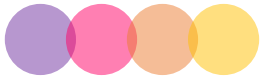
Lab: 여러 개의 값 반환

- 사용자로부터 이름과 나이를 입력받아서 동시에 반환하는 함수를 작성해보자.

```
이름:홍길동
나이:20
이름은 홍길동 이고 나이는 20 살입니다
```

Solution:

```
#           이 프로그램은 사용자로부터 이름과 나이를 받아서 다시 출력한다.

def get_info():
            name = input("이름:")
            age = int(input("나이:"))
            return name, age                # 2개의 값을 반환한다.

st_name, st_age = get_info()            # 반환된 값을 풀어서 변수에 저장한다.
print("이름은 ", st_name, "이고 나이는 ", st_age, "살입니다.")
```

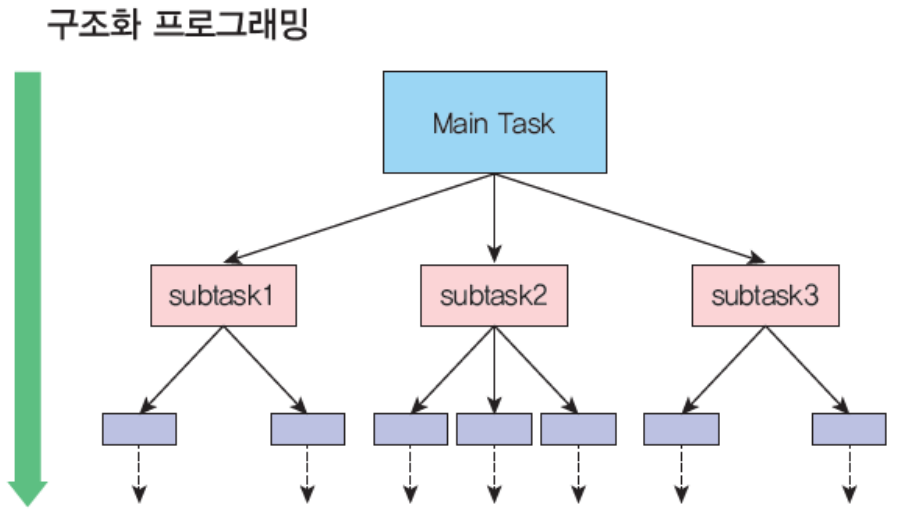


함수

함수를 사용하는 이유

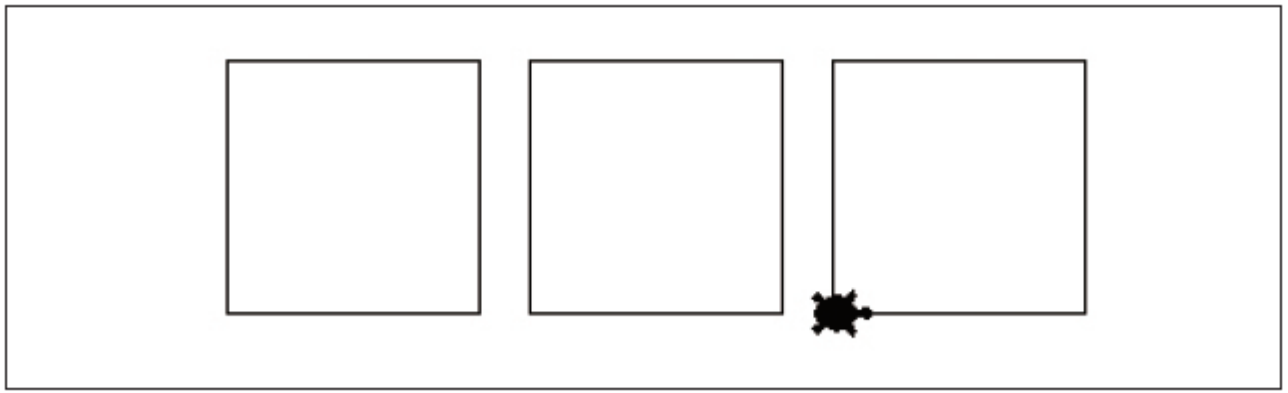
함수를 사용하는 이유

- 소스 코드의 중복성을 없애준다.
- 한번 제작된 함수는 다른 프로그램을 제작할 때도 사용이 가능하다.
- 복잡한 문제를 단순한 부분으로 분해할 수 있다.



Lab: 사각형을 그리는 함수 작성하기

- 터틀 그래픽에서 정사각형을 그리는 함수를 작성해보자.



Solution:

```
import turtle
t = turtle.Turtle()
t.shape("turtle")

def square(length):                    # length는 한변의 길이
    t.down()
    for i in range(4):
        t.forward(length)
        t.left(90)
    t.up()

square(100);                           # square() 함수를 호출한다.
t.forward(120)
square(100);
t.forward(120)
square(100);

turtle.mainloop()
turtle.bye()
```

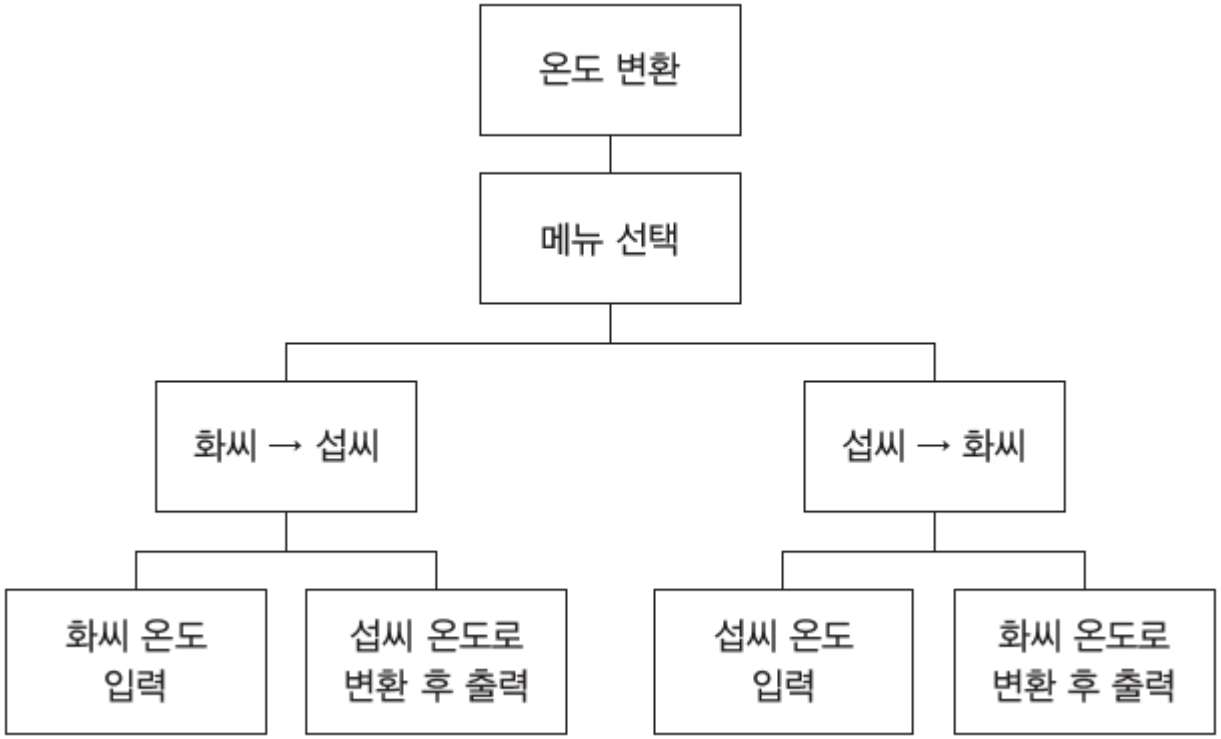
Lab: 구조화 프로그래밍 실습

- 온도를 변환해주는 프로그램을 작성해보자.

```
1. 섭씨 온도->화씨 온도
2. 화씨 온도->섭씨 온도
3. 종료
메뉴를 선택하세요: 1

섭씨 온도를 입력하시오: 37.0
화씨 온도 = 98.6

1. 섭씨 온도->화씨 온도
2. 화씨 온도->섭씨 온도
3. 종료
메뉴를 선택하세요: 3
```

함수를 사용하는 이유

Solution:

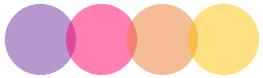


Solution:

```
def menu() :
    print("1. 섭씨 온도->화씨 온도")
    print("2. 화씨 온도->섭씨 온도")
    print("3. 종료")
    selection = int(input("메뉴를 선택하세요: "))
    return selection

def ctof(c) :
    temp = c*9.0/5.0 + 32
    return temp

def ftoc(f) :
    temp = (f-32.0)*5.0/9.0
    return temp

def input_f() :
    f = float(input("화씨 온도를 입력하시오: "))
    return f
```

Solution:

```
def input_c() :
    c = float(input("섭씨 온도를 입력하시오: "))
    return c

def main() :
    while True:
        index = menu()
        if index == 1 :
            t = input_c()
            t2 = ctof(t)
            print("화씨 온도 = ", t2, "₩n")
        elif index == 2 :
            t = input_f()
            t2 = ftoc(t)
            print("섭씨 온도 = ", t2, "₩n")
        else :
            break
main()
```

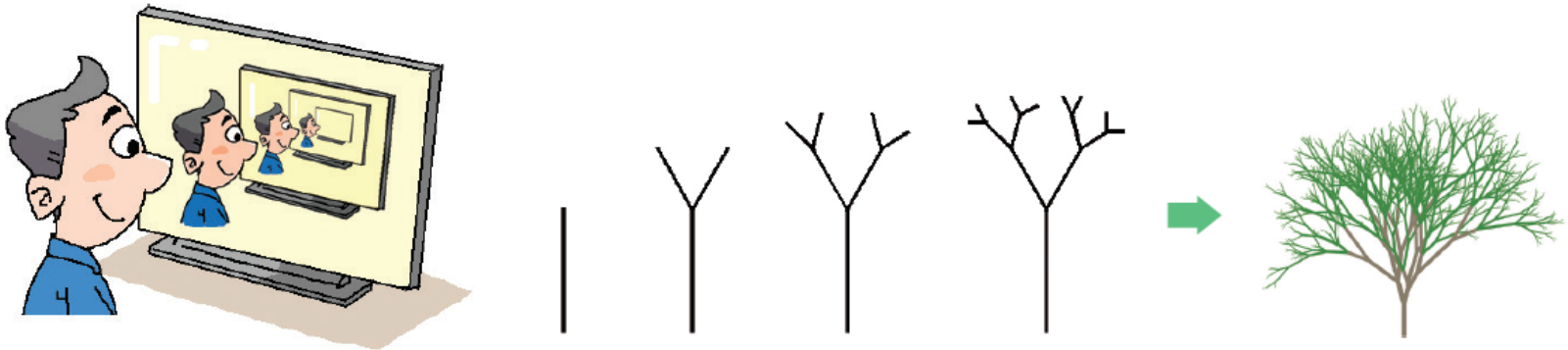


함수

순환호출

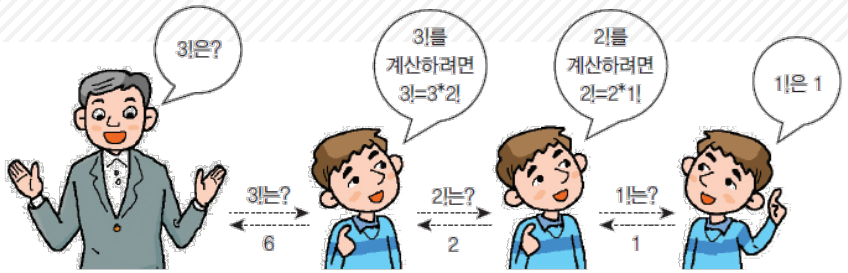
순환호출

- 순환(recursion)이란 어떤 알고리즘이나 함수가 자기 자신을 호출하여 문제를 해결하는 프로그래밍 기법이다.



순환호출의 예제

- 팩토리얼 계산 프로그램

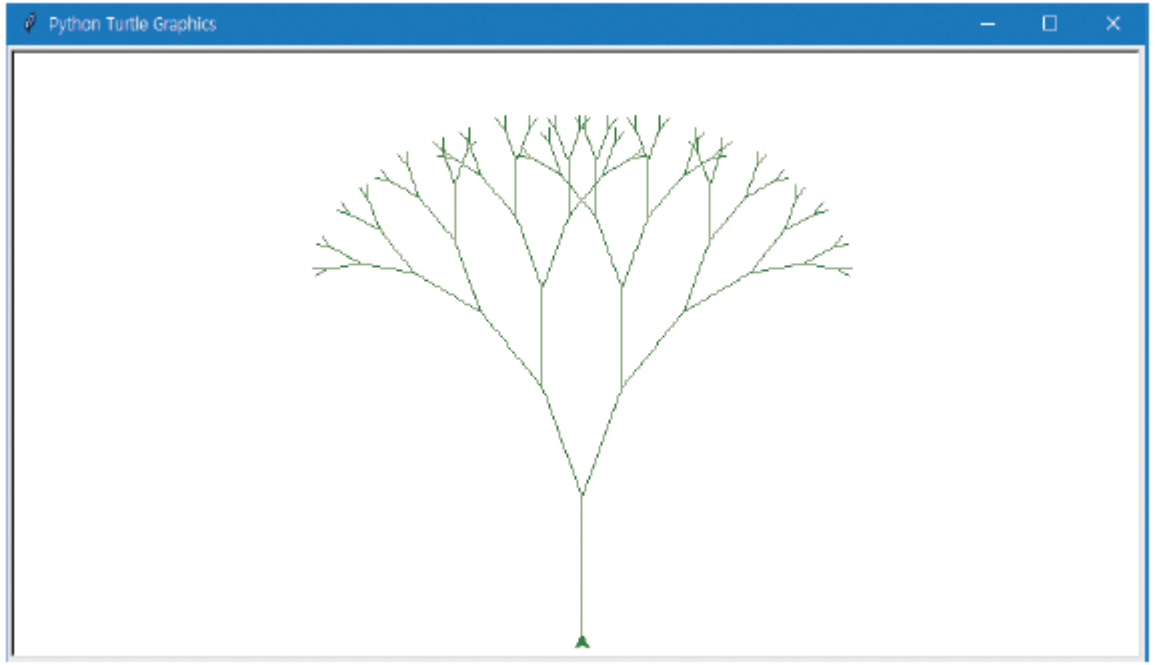



```
def  factorial(n):
    if n == 1 :
        return(1)
    else:
        return n * factorial(n-1)

n = eval(input("정수를 입력하시오:"))
print(n, "!= ", factorial(n))
```

```
정수를 입력하시오:10
10 != 3628800
```

Lab: 프랙탈 그래픽

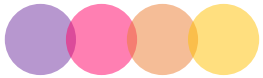
- 순환적으로 나무를 그리는 프랙탈(fractal) 프로그램을 작성해보자.



순환호출

Solut

```
import turtle

def drawTree(branch,t):
    if branch > 5:
        t.forward(branch)
        t.right(20)
        drawTree(branch-15,t)                    # 순환 호출
        t.left(40)
        drawTree(branch-15,t)                    # 순환 호출
        t.right(20)
        t.backward(branch)

def main():
    t = turtle.Turtle()
    window = turtle.Screen()
    t.left(90)
    t.up()
    t.backward(200)
    t.down()
    t.color("green")
    drawTree(100, t)
    window.exitonclick()

main()
```



함수

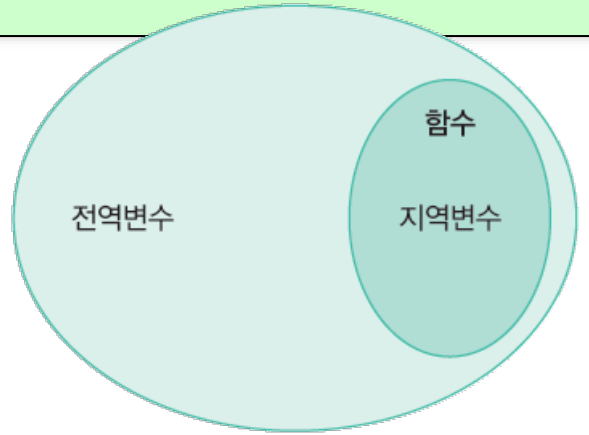
변수의 범위

변수의 범위

지역변수

```
def myfunc():
  x = 100
  print(x)

myfunc()
```

```
100
```



변수의 범위

전역 변수

```
gx = 100

def myfunc():
  print(gx)

myfunc()
print(gx)
```

```
100
100
```

지역 변수는 함수마다 동일한 이름을 사용할 수 있다.

```
def myfunc() :
    x = 200
    print(x)

def main() :
    x = 100
    print(x)

myfunc()
main()
```

변수의 범위

함수 안에서 전역 변수 변경하기

```
gx = 100

def myfunc() :
    gx = 200
    print(gx)

myfunc()
print(gx)
```

변경되지 않는다! -> 함수 안에서 변수에 값을 저장하면 새로운 지역 변수가 생성된다.

```
200
100
```

변수의 범위

함수 안에서 전역 변수 변경하기

```
gx = 100

def myfunc():
        global gx                    # 전역 변수 gx를 사용한다.
        gx = 200
        print(gx)

myfunc()
print(gx)
```

```
200
200
```

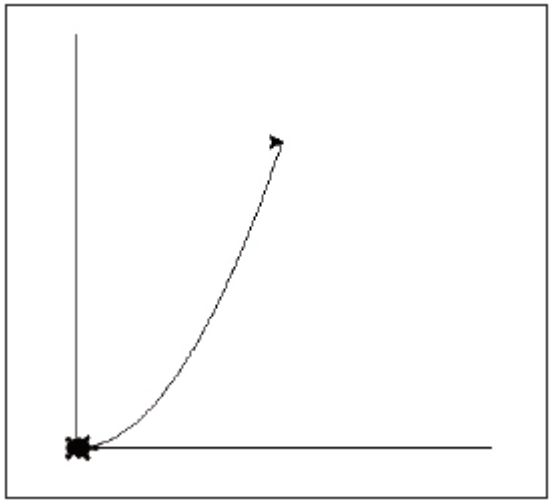
변수의 범위

중간점검

1. 지역 변수와 전역 변수는 어떻게 다른가?
2. 함수 안에서 전역 변수의 값을 읽을 수 있는가?
3. 함수 안에서 전역 변수의 값을 변경하면 어떻게 되는가?

Lab: 함수 그리기

- 함수 $f(x) = x^2 + 1$을 계산하는 함수를 작성하고 이 함수를 이용하여 화면에 f(x)를 그려보자.

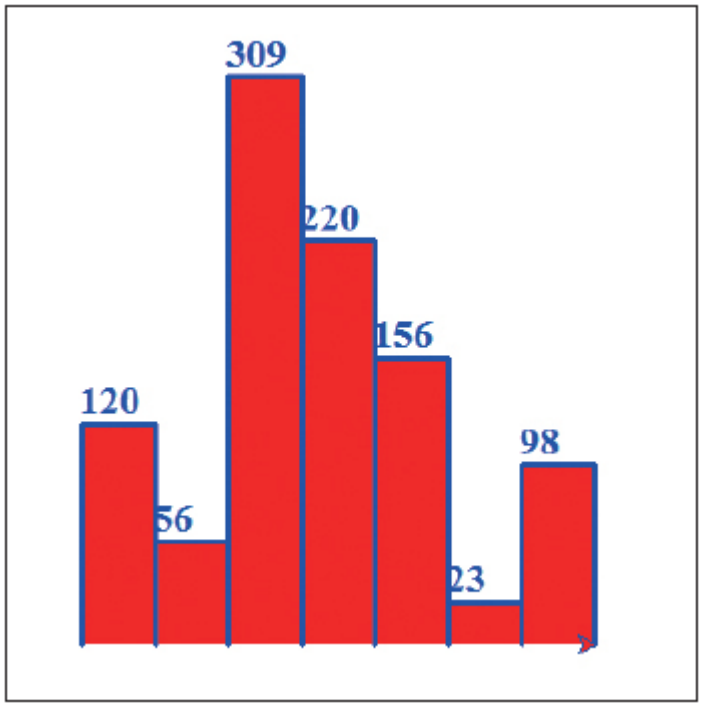


Solution:

```
import turtle
t = turtle.Turtle()
t.shape("turtle")
t.speed(0)

def f(x):
    return x**2+1

t.goto(200, 0)
t.goto(0, 0)
t.goto(0, 200)
t.goto(0, 0)

for x in range(150):
            t.goto(x, int(0.01*f(x)))

turtle.mainloop()
turtle.bye()
```

Lab: 막대 그래프 그리기

- 파이썬의 터틀 그래픽을 이용해서 막대 그래프를 그려보자.



Solution:

```
import turtle

def drawBar(height):
    t.begin_fill()
    t.left(90)
    t.forward(height)
    t.write(str(height), font = ('Times New Roman', 16, 'bold'))
    t.right(90)

    t.forward(40)
    t.right(90)
    t.forward(height)
    t.left(90)
    t.end_fill()
```

Solution:

```
data = [120, 56, 309, 220, 156, 23, 98]

t = turtle.Turtle()
t.color("blue")
t.fillcolor("red")

t.pensize(3)

for d in data:
    drawBar(d)

turtle.mainloop()
turtle.bye()
```

이번 장에서 배운 것

- 함수가 무엇인지를 학습하였다.
- 인수와 매개변수가 무엇인지를 학습하였다.
- 어떻게 함수로 인수를 전달할 수 있는지를 학습하였다.
- 여러 개의 인수를 함수로 전달하는 방법을 학습하였다.
- 함수가 값을 반환하는 방법을 학습하였다.
- 지역변수와 전역변수의 차이점에 대하여 학습하였다.
- global 키워드를 사용하여서 함수 안에서 전역변수를 사용하는 방법을 학습하였다.

함수

수고하셨습니다 ☺