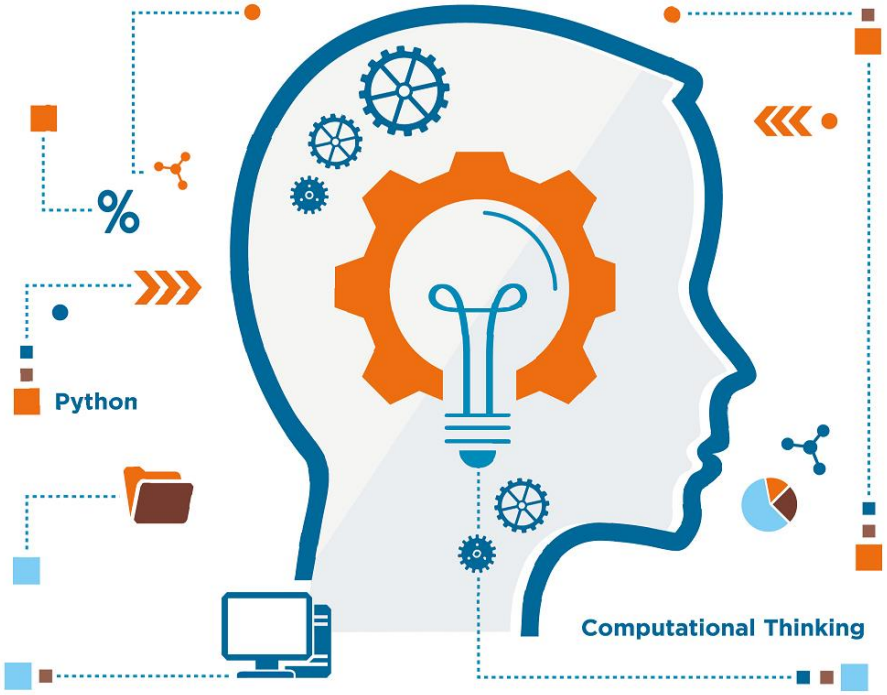


[Python]

Python으로 배우는

소프트웨어 원리

Chapter 10. 함수

목차

1. 분해와 함수
2. 함수의 사용
3. 함수의 활용

01

분해와 함수

01. 분해와 함수

[분해의 중요성]

- 컴퓨팅 사고력에 있어 중요한 구성 요소 중 하나는 분해이다.
- 여러 기능이 필요한 프로그램도 작은 단위로 분해하여 서로 연결하면 하나의 복잡한 문제를 해결할 수 있다.
- 복잡한 내용일수록 구조화가 필요하다.
- 코드에서의 구조화는 각 기능을 함수로 독립시키고
- 코드의 기본 기능을 함수 호출로 구성하면 구조적이 된다.

```
########## Main ####################################
read_menu_file()     #Menu.txt 파일로부터 menu 딕셔너리 생성하기
new_menugroup()      #대분류 메뉴 리스트(menugroup) 생성
selmlist = []        #주문 받을 준비 (빈 리스트 생성)
while True:          #작업 선택
    selno = sel_task()  #작업 번호 선택
    if selno == MENUGROUP:       #(1)대분류 메뉴 보기
        prt_menugroup(menugroup)
    elif selno == MENU:          #(2)소분류 메뉴 보기
        prt_menu(menu)
    elif selno == ORDER:         #(3)주문하기
        selmlist = []                          #주문 준비
        process_order(selmlist)                #주문 처리
        dseq = append_order_file(selmlist)  #주문내역 파일로 저장
        set_order_dseq(dseq)                   #일일 주문 일련번호 갱신
    elif selno == ORDERLIST:     #(4)현재 주문내역 보기
        prt_result(selmlist)
    elif selno == TOTORDER:      #(5)전체 주문내역 파일 보기
        prt_order_file()
    elif selno == END:           #(x)끝내기
        break
    else:
        break
####################################################
```

01. 분해와 함수

I. 분해의 개념

- 복잡한 문제를 해결 가능한 수준의 작은 문제로 나누는 과정이 분해
- 하나의 문제를 여러 개의 작은 부분으로 나누어 해결해 나가는 방법으로 분할 정복(Divide and Conquer)이라고도 표현

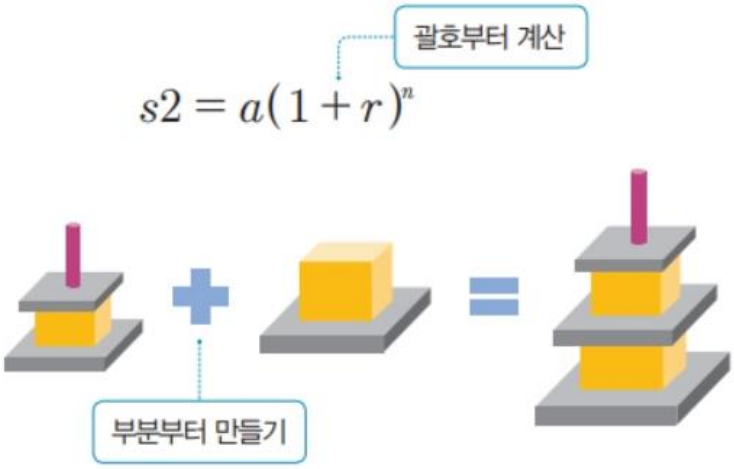


그림 10-1 분해의 활용

1. 컴퓨팅적 사고
1-1. 컴퓨팅적 사고의 개념
1-1-1. 컴퓨터의 특징
1-1-2. 컴퓨팅적 사고의 필요성
1-1-3. 컴퓨팅적 사고의 구성 요소
2. 프로그래밍
2-1. 프로그래밍의 개념
2-1-1. 프로그래밍 과정
2-1-2. 프로그래밍 언어

목차

01. 분해와 함수

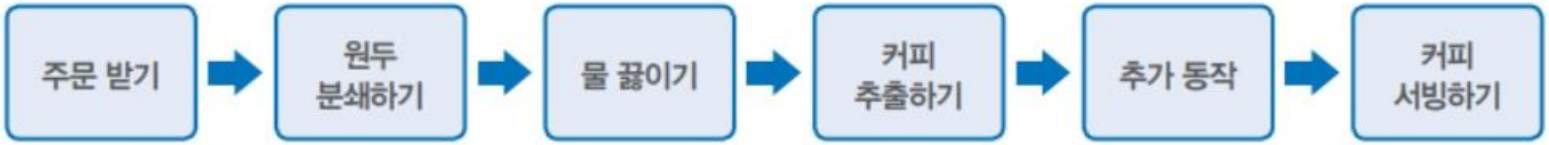
I. 분해의 개념

실습 10-1 무인판매기의 동작을 분해하기

① 필요한 작업을 나누어 순서대로 나열

그림 10-3 전체 작업을 동작별로 분해

② 전체 작업을 동작별로 분해하면 아이스커피나 라떼 등 다른 종류의 커피를 만들 때에도 필요한 동작을 가져와 **재사용**



아이스커피

그림 10-4 분해한 동작을 다른 작업에 재사용

01. 분해와 함수

I. 분해의 개념

실습 10-1 무인판매기의 동작을 분해하기

③ 여러 잔의 커피를 주문하는 경우 고려

그림 10-5 작업의 반복

④ 여러 잔의 커피를 각각 만드는 것보다 더 좋은 방법은 원두와 물을 주문한 수량에 맞게 더 많이 준비하고 추출한 커피를 3개의 잔에 나누어 서빙하는 방법

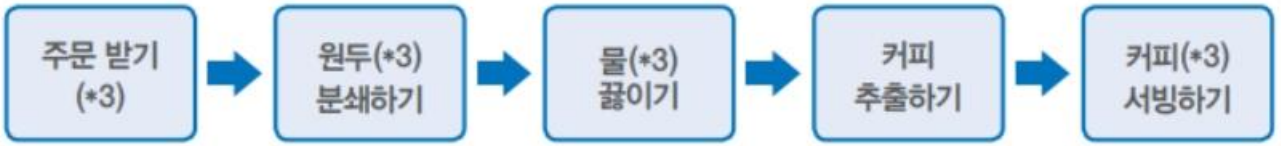


그림 10-6 효율적인 작업의 반복

01. 분해와 함수

I. 분해의 개념

실습 10-2 탑 쌓기에 필요한 블록의 개수 구하기

- 1cm 크기의 정육면체 블록으로 다음과 같은 모양의 삼층탑 만들기
- 탑을 완성하기 위 한 작업을 분해해서, 전체 탑을 쌓기 위해 사용되는 블록의 개수를 계산

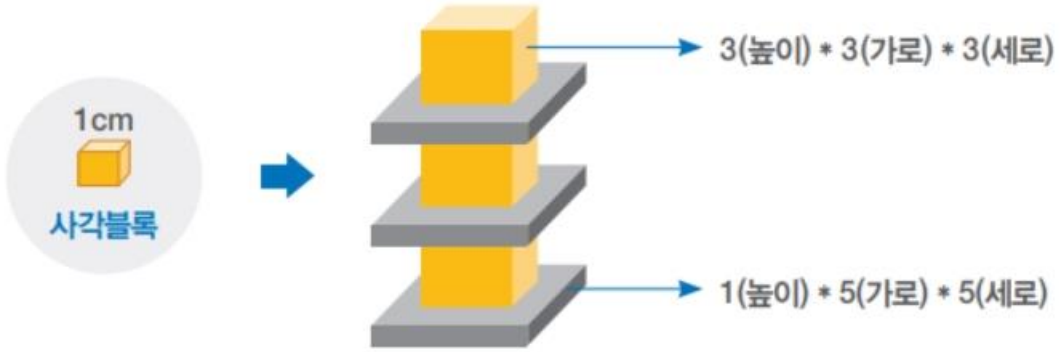


그림 10-7 탑의 구조

01. 분해와 함수

I. 분해의 개념

실습 10-2 탑 쌓기에 필요한 블록의 개수 구하기

① 탑을 완성하기 위해 판과 기둥으로 작업을 분해하고, 각 작업에 필요한 블록 개수를 계산

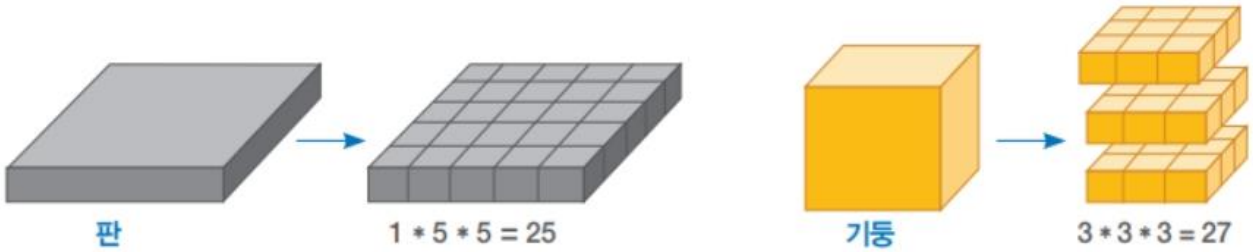


그림 10-8 판과 기둥에 필요한 블록의 개수

② 삼층탑을 만들기 위한 전체 블록의 개수는 다음과 같이 구할 수 있음

블록 합계 = (판 + 기둥) * 3 = (25 + 27) * 3 = 156

그림 10-9 전체 블록의 개수 계산

01. 분해와 함수

II. 함수의 개념

- 프로그램 내에서 분해된 작은 단위의 작업이 **함수**
- 원하는 목적을 달성하기 위한 기능들을 하나의 처리 단위로 묶어놓은 것
- **필요할 때마다 이름으로 호출하여 사용**
- 함수를 사용 하면 문제를 분해함으로써 해결이 쉬워질 뿐만 아니라, 공통으로 사용하는 기능이나 반복되는 **코드를 재사용**
- 프로그램을 **구조화**시킬 수 있어서 **가독성**을 높이는 효과도 있음

그림 10-11 함수의 재사용

02

함수의 사용

02. 함수의 사용

I. 함수의 정의와 호출

- **사용자 정의 함수** : 필요한 동작을 수행하는 함수를 사용자가 직접 생성
- 파이썬에서 함수를 만들 때는 다음의 함수 정의 형식처럼 def를 사용하여 함수의 이름과 필요한 명령을 나열
- 함수가 만들어지면, 프로그램 내 어떤 위치에서든 필요할 때마다 함수 이름을 사용해 호출

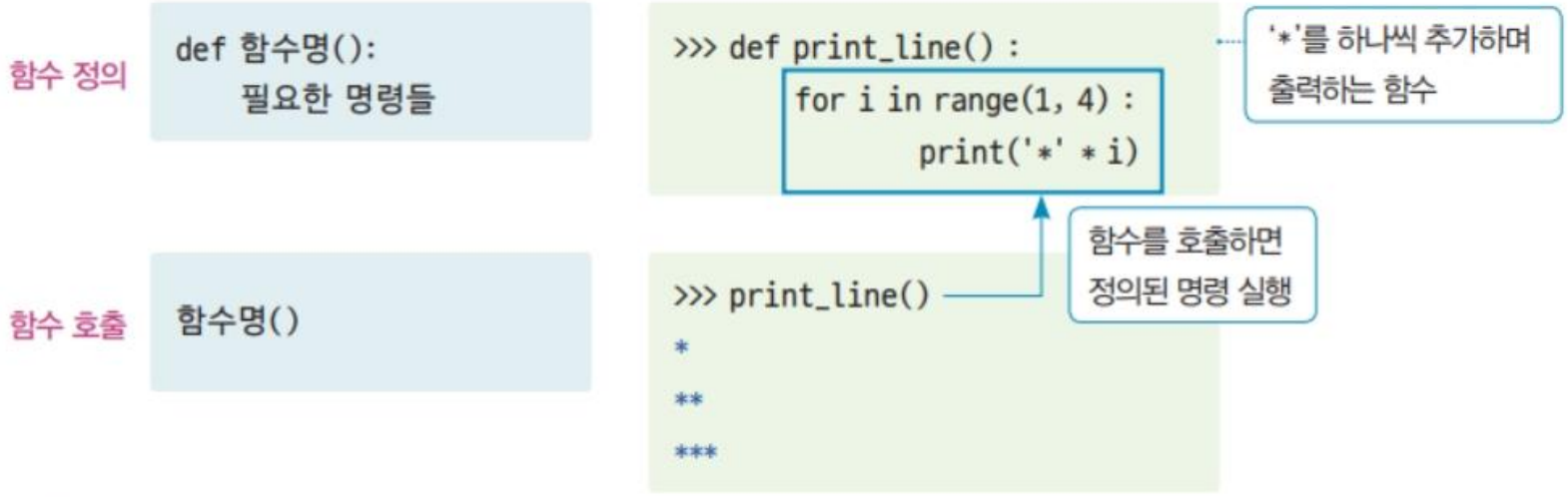


그림 10-12 함수의 정의와 호출 구조

02. 함수의 사용

I. 함수의 정의와 호출

실습 10-3 원 그리기 함수를 만들고 호출하기 code10-03.py

- 원을 그리는 함수를 정의하고, 원하는 위치에서 함수를 호출하면 그림을 그리는 프로그램을 작성
- 3개의 원을 위쪽으로 쌓아 원기둥 모양 만들기

① 원을 채워서 그리는 함수 draw_circle( )을 정의, 원의 반지름은 100으로 고정

```
def draw_circle() :                    # 함수 정의
    begin_fill()
    circle(100)
    end_fill()
```

② 정의한 draw_circle( ) 함수를 호출하는 부분을 추가하여 전체 프로그램을 작성

```
01  from turtle import *
02  color("yellow", "gray")
03
```

❖ from 모듈명 import * 는 모든 함수(*)를 **모듈명 없이 호출**하고자 할 때 사용

02. 함수의 사용

I. 함수의 정의와 호출

실습 10-3 원 그리기 함수를 만들고 호출하기 code10-03.py

② 정의한 draw_circle( ) 함수를 호출하는 부분을 추가하여 전체 프로그램을 작성

```
04  def draw_circle() :              # 함수 정의
05      begin_fill()
06      circle(100)
07      end_fill()
08
09  for i in range(3) :
10      goto(0, i * 20)              # 원을 그릴 위치로 이동
11      draw_circle()                # 함수 호출
```

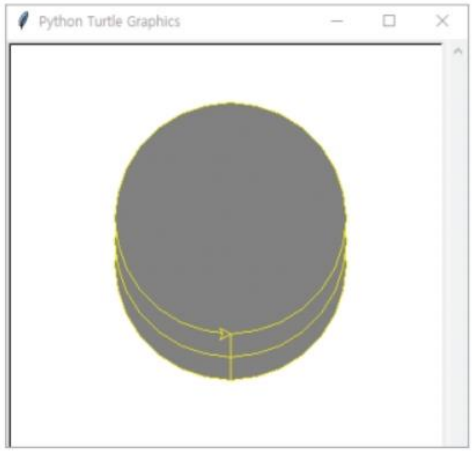


그림 10-13 실행 결과

02. 함수의 사용

II. 함수의 인수

- 함수는 인수Argument에 의한 값 전달을 통해 반복문처럼 같은 동작만 반복하는 것이 아니라 원하는 형태로 동작을 변형할 수 있음
- 함수를 호출할 때 괄호 안에 인수를 기술하면, 함수를 만들 때 정의된 매개변수 Parameter에 값이 전달

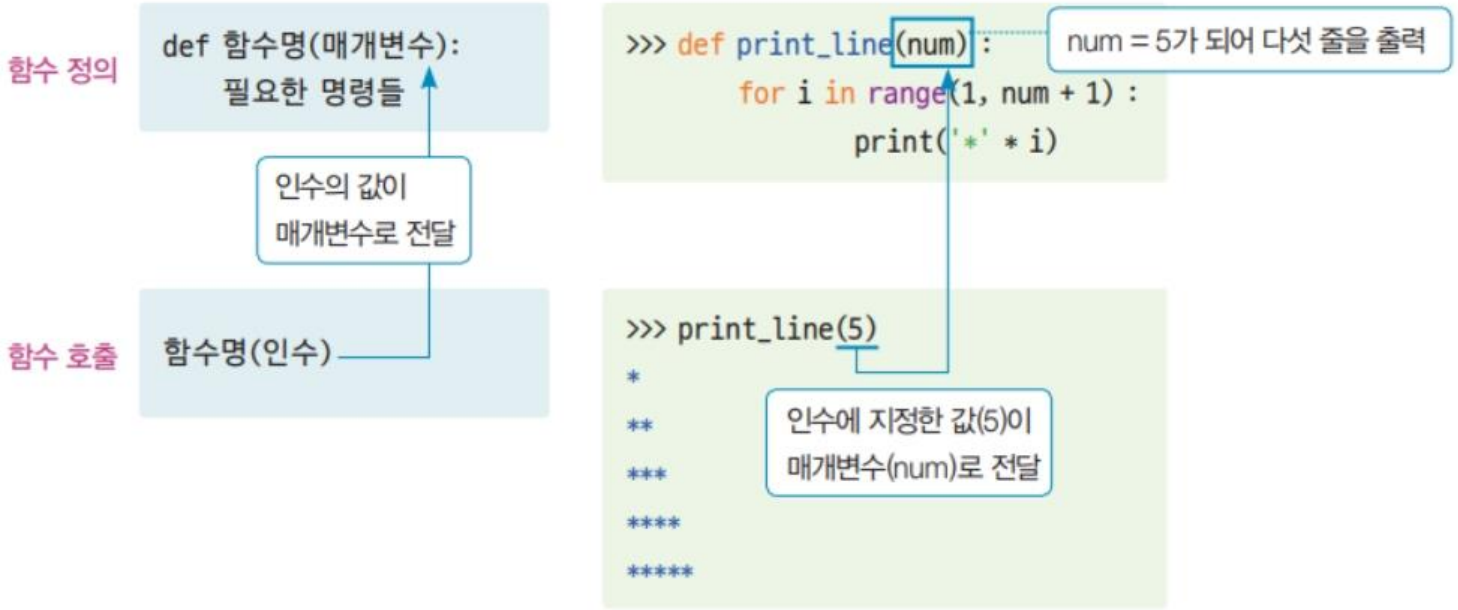
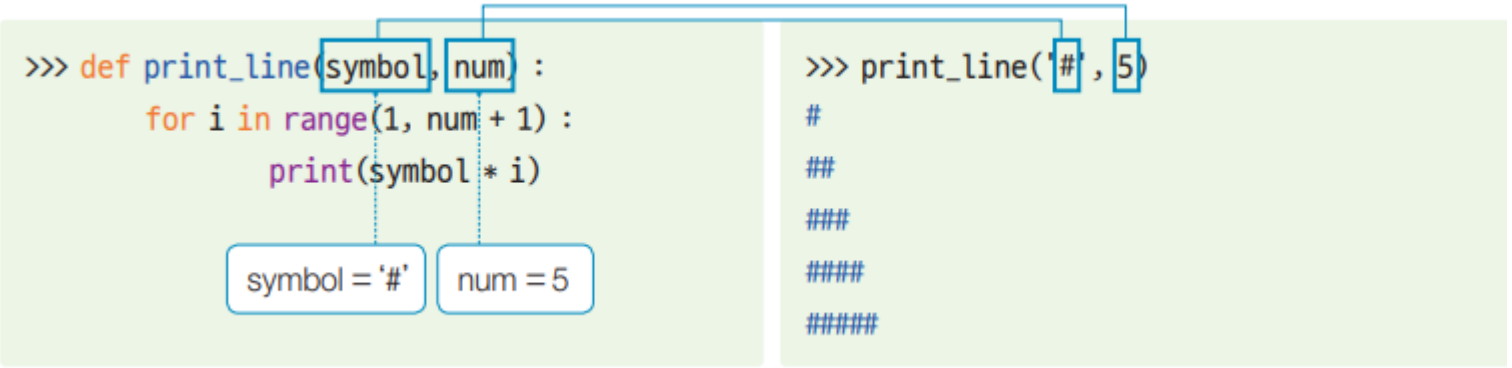


그림 10-15 매개변수와 인수의 값 전달

02. 함수의 사용

II. 함수의 인수

- 여러 개의 인수를 이용하면 함수의 동작을 더 다양하게 실행할 수 있음



실습 10-4 매개변수를 사용하여 원 그리기 함수 수정하기 code10-04.py

① draw_circle( ) 함수를 수정하여 매개변수를 가지도록 정의, 원을 그릴 위치(x, y 좌표), 원의 반지름(r)을 매개변수로 지정

```
def draw_circle(x, y, r) :          # 함수 정의
    goto(x, y)
    begin_fill()
    circle(r)
    end_fill()
```

02. 함수의 사용

I. 함수의 정의와 호출

여기서 잠깐 **매개변수와 인수의 기술 방법**

- 함수를 정의할 때 사용한 매개변수와 호출할 때의 인수는 개수와 순서에 맞게 기술해야 함

```
>>> print_line()
Traceback (most recent call last):
    File "<pyshell#9>", line 1, in <module>
        print_line()
TypeError: print_line() missing 1 required positional argument: 'num'
>>> print_line(5, '#')
Traceback (most recent call last):
    File "<pyshell#15>", line 1, in <module>
        print_line(5, '#')
    File "<pyshell#13>", line 2, in print_line
        for i in range(1, num + 1) :
TypeError: can only concatenate str (not "int") to str
```

인수의 개수가 다르므로 오류 발생

인수의 순서가 잘못 지정되어 오류 발생

II. 함수의 인수

실습 10-4 매개변수를 사용하여 원 그리기 함수 수정하기 code10-04.py

② 정의한 draw_circle( ) 함수를 호출하는 부분도 수정하여 프로그램을 완성

```
01  from turtle import *
02  color("yellow", "gray")
03
04  def draw_circle(x, y, r) :
05      goto(x, y)
06      begin_fill()
07      circle(r)
08      end_fill()
09
10  for i in range(3) :
11      draw_circle(0, i * 20, 100)          # 함수 호출
12
13  draw_circle(300, 0, 100)                 # 함수 호출
14  draw_circle(-300, 0, 100)                # 함수 호출
```

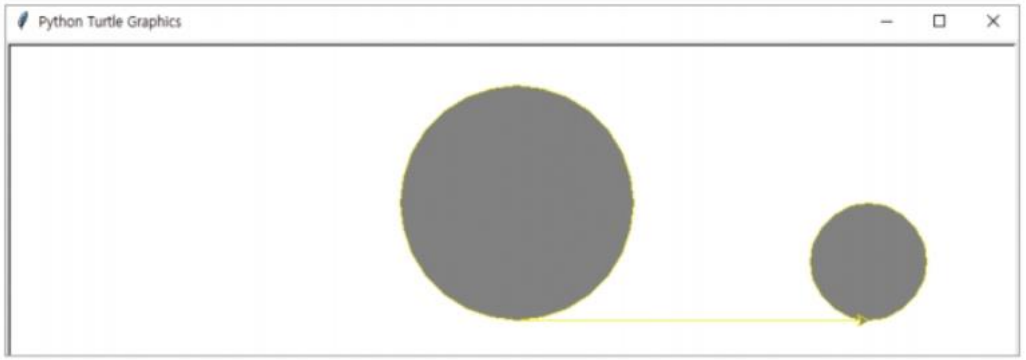


그림 10-20 실행 결과

02. 함수의 사용

III. 다양한 인수 설정

▪ 디폴트(default) 인수

- 함수를 정의할 때 매개변수의 기본값을 지정하면 함수를 호출할 때 인수 생략 가능
- 디폴트 인수를 가진 함수는 인수와 매개변수의 개수가 맞지 않아도 설정된 기본값을 사용하여 함수의 동작을 실행

```
>>> def print_line(symbol = '*', num = 3) :    # 매개변수의 기본값 지정
        for i in range(1, num + 1) :
            print(symbol * i)
```

인수를 생략해도 실행 가능

```
>>> print_line()
*
**
***
```

```
>>> print_line('=')
=
==
===
```

```
>>> print_line('#', 5)
#
##
###
####
#####
```

02. 함수의 사용

III. 다양한 인수 설정

▪ 키워드(keyword) 인수

- 인수가 순서에 맞지 않아도 함수의 동작이 실행되도록 하려면 함수를 호출할 때 인수의 이름을 지정
- 디폴트 값이 정의된 함수에서도 인수를 순서에 맞게 지정하지 않으면 원하는 동작을 실행할 수 없음 / 필요한 인수의 이름을 지정하는 키워드 인수를 사용하면 순서에 상관없이 매개변수에 값을 전달할 수 있음

```
>>> def print_line(symbol = '*', num = 3) :    # 함수 정의
        for i in range(1, num + 1) :
            print(symbol * i)
```

인수의 값(5)이 symbol에 전달

인수의 이름(num)을 지정하여 전달

```
>>> print_line(5)
5
10
15
```

```
>>> print_line(num = 5)
*
**
***
****
*****
```

02. 함수의 사용

III. 다양한 인수 설정

[실습] 함수의 디폴트 인수 처리 지정

```
def prints(symbol='', cnt=1, nl=''):
    print(symbol*cnt, end=nl)


prints('=', 10, '₩n')
prints("Test Mode", nl='₩n')
prints('#', 10, '₩n')
```

```
prints(cnt=10, symbol='*', nl='₩n')
print("Test Mode")
prints('*', 10)
```

02. 함수의 사용

III. 다양한 인수 설정

▪ 가변(variable length) 인수

- 함수의 매개변수를 '*'로 정의하면 가변 인수가 되어 인수의 개수에 상관없이 함수를 호출할 수 있음

```
>>> def add(*numbers) :                # 매개변수의 개수가 정해지지 않음 (tuple 변수로 처리)
        result = 0
        for x in numbers :
                result += x
        print(result)
```

인수의 개수에 상관없이 호출

```
>>> add()
0
```

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

```
>>> add(10, 20, 30, 40)
100
```

02. 함수의 사용

III. 다양한 인수 설정

실습 10-5 디폴트 인수를 사용하여 원 그리기 함수 수정하기 code10-05.py

- [실습 10-4]의 원을 그리는 함수를 수정, 함수를 호출할 때 반지름을 지정하지 않으면 반지름 100인 원을 그리도록 설정

① draw_circle( ) 함수의 반지름을 디폴트 인수로 수정하여 기본값을 100으로 설정

```
def draw_circle(x, y, r = 100) :        # 반지름을 디폴트 인수로 함수 정의
    goto(x, y)
    begin_fill()
    circle(r)
    end_fill()
```

② 수정한 draw_circle( ) 함수를 호출하는 부분을 추가하여 프로그램을 완성

```
01  from turtle import *
02  color("yellow", "gray")
03
04  def draw_circle(x, y, r = 100) :          # 반지름을 디폴트 인수로 함수 정의
05      goto(x, y)
```

III. 다양한 인수 설정

실습 10-5 디폴트 인수를 사용하여 원 그리기 함수 수정하기 code10-05.py

② 수정한 draw_circle( ) 함수를 호출하는 부분을 추가하여 프로그램을 완성

```
06      begin_fill()
07      circle(r)
08      end_fill()
09
10  for i in range(3) :
11      draw_circle(0, i * 20)          # 반지름을 지정하지 않고 함수 호출
12
13  draw_circle(300, 0, 50)
14  draw_circle(-300, 0, 80)
```

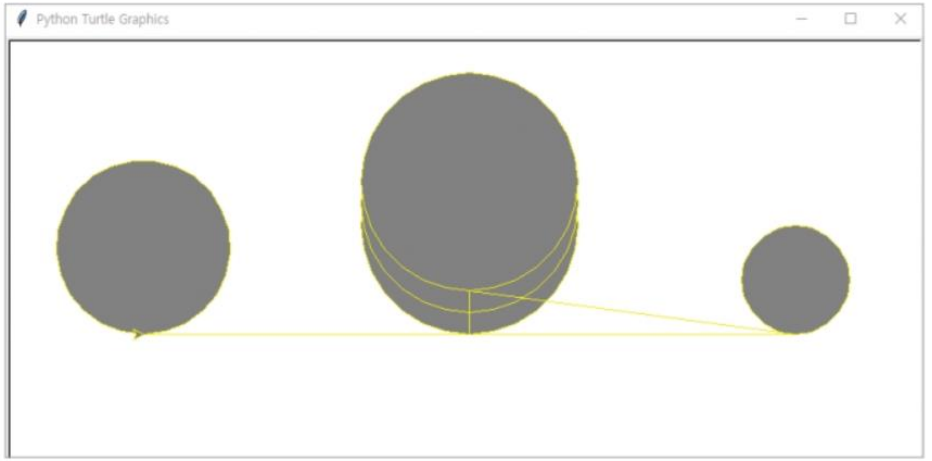


그림 10-17 실행 결과

02. 함수의 사용

IV. 함수의 반환

- 함수는 인수로 전달된 값을 이용해 필요한 동작을 수행하고 그 결과를 반환
- 반환값이 있는 경우 함수 정의 블록에 **return**과 함께 반환값을 기술
- return 문장이 실행될 때 함수의 동작은 끝나고, 함수를 호출한 부분으로 반환값이 전달

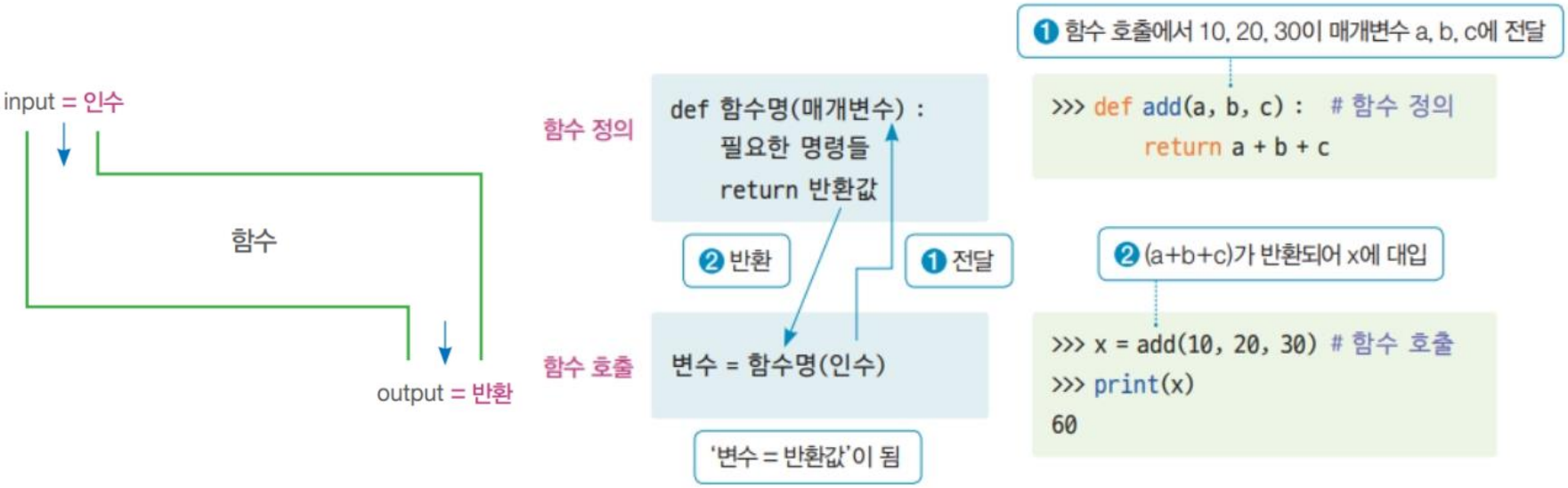


그림 10-19 값의 전달과 결과의 반환 구조

02. 함수의 사용

IV. 함수의 반환

여기서 잠깐 **반환값의 여러 가지 형태**

- 함수 정의 블록 안에서 사용할 수 있는 return은 값이나 변수, 수식을 반환값으로 사용하지만, 반환값을 생략 하거나 여러 개를 사용하는 경우도 있음
- 다음은 반환값을 생략하고 함수를 종료하기 위한 목적으로 사용한 코드

```
>>> def func1(x, y) :
        if y == 0 :
            return                  # y가 0이면 함수를 종료
        print(x / y)

>>> func1(5, 0)
>>> func1(5, 3)
1.6666666666666667
```

02. 함수의 사용

IV. 함수의 반환

반환값의 여러 가지 형태

- 반환값이 여러 개인 경우 **튜플 데이터형으로 반환**
- 함수 호출 문장에서 반환값을 대입할 변수를 여러 개 사용 할 수 있음

```
>>> def func2(x, y) :
        return (x + y), (x - y)          # 두 수의 합과 차를 반환
>>> a, b = func2(5, 3)   #함수가반환한 tuple 값들을 a, b로 언팩킹
>>> print(a, b)
8 2
```

```
def func2(x, y):
    return x+y, x-y

#함수가 반환한 튜플을 언팩킹하여 a와 b로 저장
a, b = func2(3, 4)
print(func2(3, 4))
```

```
(7, -1)
```

02. 함수의 사용

IV. 함수의 반환

실습 10-6 반환값이 있는 원 그리기 code10-06.py

① draw_circle( ) 함수에 원의 면적을 계산하고 결과를 반환하는 문장을 추가

```
def draw_circle(x, y, r = 100) :
    goto(x, y)
    begin_fill()
    circle(r)
    end_fill()
    return (3.14 * r ** 2)          # 원의 면적을 계산하고 반환
```

② 수정한 draw_circle( ) 함수를 호출하는 부분을 추가하여 프로그램을 완성

```
01  from turtle import *
02  color("yellow", "gray")
```

IV. 함수의 반환

실습 10-6 반환값이 있는 원 그리기 code10-06.py

② 수정한 draw_circle( ) 함수를 호출하는 부분을 추가하여 프로그램을 완성

```
03
04  def draw_circle(x, y, r = 100) :                # 함수 정의
05      goto(x, y)
06      begin_fill()
07      circle(r)
08      end_fill()
09      return = (3.14 * r ** 2)                    # 원의 면적을 계산하고 반환
10
11  print("원의 면적 = ", draw_circle(0, 0))        # 함수 호출
12  print("원의 면적 = ", draw_circle(300, 0, 50))  # 함수 호출
```

③ 작성한 프로그램을 실행

```
원의 면적 =  31400.0
원의 면적 =  7850.0
```

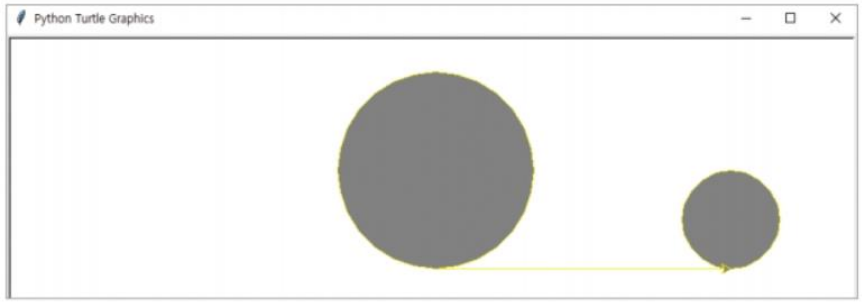


그림 10-20 실행 결과

02. 함수의 사용

V. 전역변수와 지역변수

- 함수는 독자적으로 처리되는 처리 단위이다.
- 함수 내에서 사용되는 변수, 값들은 함수 내에서만 유효하다.
- 변수명이 같아도 함수 안과 밖의 변수는 별개의 것이다.

```
def add(a, b):
    total = a + b
    return total

total = add(3, 4)
print(total)
```

```
def add(a, b) :
    total = a + b

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



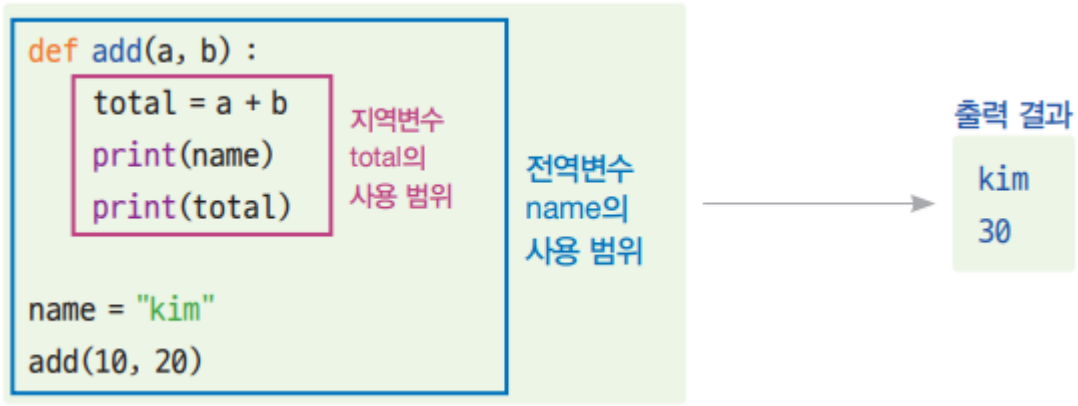
```
print(total)
NameError: name 'total' is not defined
```

- 변수는 함수 내에서만 사용되는 지역Local변수와 프로그램 전체 범위에서 사용할 수 있는 전역Global변수로 구분
- 함수 바깥에서 정의한 변수는 전역변수로 프로그램의 어느 위치에서든 사용할 수 있고, 프로그램 종료 시까지 유효하다.
- 반면 함수 내의 지역변수는 함수 내에서만 사용 가능하고 함수가 종료되면 사라짐

02. 함수의 사용

V. 전역변수와 지역변수

- 변수는 함수 내에서만 사용되는 지역변수와 프로그램 전체 범위에서 사용할 수 있는 전역변수로 구분
- 함수 바깥에서 정의한 변수 name은 전역변수로 프로그램의 어느 위치에서든 사용할 수 있지만, total은 함수 내에서만 사용 가능한 지역변수



02. 함수의 사용

V. 전역변수와 지역변수

- total 변수를 사용하면 2개의 total이 만들어져서 메모리의 다른 공간을 사용하게 되고, 합계를 올바르게 출력하지 못함

```
def add(a, b) :
    total = a + b        # 지역변수 total 생성

total = 0                # 전역변수 total 생성
add(10, 20)
print(total)             # 전역변수 total 출력
```

total 30

total 0

출력 결과

0

- 함수 안에서 '**global**' 키워드로 변수를 선언하면 함수 바깥에 정의된 전역변수를 사용할 수 있음

```
def add(a, b) :
    global total         # 전역변수 total 사용 선언
    total = a + b

total = 0                # 전역변수 total 생성
add(10, 20)
print(total)
```

total 30

출력 결과

30

02. 함수의 사용

VI. 재귀 함수

- 재귀 함수(Recursive Function)
  - 함수 내에서 자기 자신을 다시 호출(재귀 호출)하는 형태의 함수
  - 함수가 종료되기 전에 재귀 호출이 일어나므로 언제인가는 끝나고 복귀할 수 있는 조건으로 변경된 (매개변수)조건으로 재귀 호출을 해야한다.
- 재귀함수 조건
  - 재귀함수는 **변경된 조건으로 재귀 호출**되어야 한다.
  - 재귀함수는 **호출한 함수로의 복귀**도 반드시 포함되어야 한다.

```
##재귀함수 예제 : n!
def mul(num):  #재귀 호출 함수
    if num > 1: #재귀 호출 조건
        return num * mul(num-1)   #재귀 호출: 호출 조건 값(num)의 변화
    else:
        return 1  #재귀 호출 종료 후 복귀

n = int(input(">누적 곱할 끝 수? "))
result = mul(n)
print(result)
```

02. 함수의 사용

VI. 재귀 함수 호출

- **함수 호출 및 복귀 확인**

```
def mul(num):  #재귀 호출 함수
    print(">Call %d" %num)
    if num > 1: #재귀 호출 조건
        result = num * mul(num-1)  #재귀 호출: 호출 조건 값(num)의 변화 필수
        print(">Return %d" %result)
        return result
    else:
        return 1  #재귀 호출 종료 후 복귀

n = int(input(">누적 곱할 끝 수? "))
result = mul(n)
print(">>", result)
```

02. 함수의 사용

VI. 재귀 함수 호출

- **[실습] n의 m승 결과를 반환하는 재귀함수**

❖ Ch10-nPowerM.py

◆ n의 m 지수승을 계산하여 반환하는 재귀함수를 작성하시오.

```
def powers(n, m):  #재귀 호출 함수
    pass

inlist = [x for x in input(">(n의 m승)의 n m 입력? ").split()]
n, m = int(inlist[0]), int(inlist[1])
result = powers(n, m)
print(">>", result)
```

```
>(n의 m승)의 n m 입력? 2 10
>> 1024
```

02. 함수의 사용

VI. 재귀 함수 호출

- **[실습] 이진 검색: 회원 아이디 검색**

❖ [실습 8-6] code08-06.py 참조

◆ members 리스트에서 회원 아이디를 이진 검색 방법으로 찾는 코드이다.

➢ 몇 번만에 탐색이 끝났는지 출력하여 확인하여보시오.

```
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
           ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))
search = input("아이디 입력 : ")
number = -1      # 만족도 점수의 초기화
start = 0        # 범위의 시작과 마지막 설정
end = len(members)   #index 값이므로 -1
mid = (start + end) // 2  # 가운데 항목을 첫 번째 검색 위치로 설정

while start < end :
    if search == members[mid][0] :  # 찾는 아이디가 있으면 점수를 저장하고 반복 종료
        number = members[mid][1]
        break
    else :
        if start == (end - 1) : # 검색 범위를 줄일 수 없음(=찾는 값이 없음)
            break
        elif search > members[mid][0] :
            start = mid + 1   # 검색 범위를 뒤쪽 반으로 줄이기
            mid = (start + end) // 2
        else :
            end = mid  # 검색 범위를 앞쪽 반으로 줄이기
            mid = (start + end) // 2
## <이하 생략>
```

02. 함수의 사용

VI. 재귀 함수 호출

- ❖ Ch08-BinarySearch.py
- ❖ Ch10-BinarySearch00.py

▪ [과제] 회원 아이디 검색을 재귀 호출로 해결

◆ members 리스트에서 회원 아이디를 이진 검색 방법으로 찾는 코드이다.

- ➢ 이진 검색 부분을 재귀함수 호출로 해결하도록 코드를 수정하여보시오.
- ➢ 검색 결과를 다음과 같이 출력
  - ➢ 못 찾으면 : "찾는 회원이 없습니다." 또는 "Not found!"
  - ➢ 찾으면 : 찾은 index 값, 찾은 회원 아이디, 찾기 시도(함수 호출) 횟수

```
members = (('choi', 93), ('han', 50), ('jung', 92), ('kang', 68), ('kim', 80),
          ('lee', 90), ('moon', 65), ('na', 100), ('park', 75), ('song', 75))

scnt = 0  #search counter
def Bi_search(search, members, start, end):  #재귀 호출 함수
    return -1  #찾기 실패 시

search = input(">아이디 입력 : ")
result = -1      #찾은 위치 값의 초기화(number 변수를 대체)
start = 0        # 범위의 시작과 마지막 설정
end = len(members)   #index 값이므로 -1
result = Bi_search(search, members, start, end)  #찾은 값에 대한 index 값을 반환 받는다.
if result == -1:
    print("Not found!")
else:
    print(">>Index : %3d, Value : %s" %(result, members[result][0]))
    print(">>Count of search : %d" %scnt)
```

```
>아이디 입력 : kim
>>Index :   4, Value : kim
>>Count of search : 3
```

VI. 재귀 함수 호출

- **[실습] 스티브 잡스의 스탠퍼드 대학 연설문** ❖ Ch10-Speech_jobs.py

◆ 스티브 잡스의 스탠퍼드대학 졸업식 연설문으로 이진 검색 적용

- http://github.com/dndxor/BCPy/Ch10 에서 speech_jobs_KOR.txt 을 다운 받아 사용
- 2진 검색 복잡도(Complexity)
  - O(log n) #base가 2인 log
  - n = 1116이면, n이 2**10에 가까우므로 log2**10 = 10

```
import os
fpath = os.getcwd()
infname = 'speech_jobs_KOR.txt'
fname = fpath + '₩₩' + infname.strip()
fdata = []
try:
    fd = open(fname, 'r', encoding = 'UTF-8')
except FileNotFoundError:
    print("파일이 존재하지 않습니다.₩n%s" %fname)
else:
    for line in fd.readlines():
        for word in line.split():
            fdata.append(word)

words = sorted(set(fdata))
print(words)
print(">Words count: %d" %len(words))
```

```
['10년', '10년동안', '10년이', '17년',
>Words count: 1116
>찾을 단어 값? 10년
>>Index :   0, Value : 10년
>>Count of search : 11
```

02. 함수의 사용

VII. 다양한 함수 호출 방법

▪ 함수형 변수의 사용 (1)

◆ 변수에 함수명을 저장하면 함수형 변수가 된다.

- 함수 변수를 함수명 대신에 사용 가능
- 함수 변수에 인자를 주어 변수 내에 저장된 함수로 인자 전달 호출

```
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

op = add    #변수에 함수명 저장
op(3, 4)    #변수로 함수 호출
op = sub
op(3, 4)
type(op)    #type 'function'
```

```
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

op = add
op(3, 4)
7
type(op)
<class 'function'>
op = sub
op(3, 4)
-1
```

02. 함수의 사용

VII. 다양한 함수 호출 방법

▪ 함수형 변수의 사용 (2)

◆ 리스트에 함수명을 저장하여 함수 호출

- 리스트의 데이터 항목을 함수처럼 사용

```
def add(a, b):
    return a+b

def sub(a, b):
    return a-b

flist = [add, sub]

flist[0](3, 4)    #add(3, 4) 함수 호출
flist[1](3, 4)    #sub(3, 4) 함수 호출
type(flist[0])    #type 'function'
```

```
def add(a, b):
    return a+b

def add(a, b):
    return a+b

def sub(a, b):
    return a-b

flist = [add, sub]
flist[0](3, 4)
7
flist[1](3, 4)
-1
type(flist[0])
<class 'function'>
```

02. 함수의 사용

VII. 다양한 함수 호출 방법

▪ 함수형 변수의 사용 (3)

◆ 함수 호출 함수에 함수명을 전달하여 간접적으로 함수 호출 가능

- 함수명만 전달하는 방법
- 함수명()로 전달하는 방법

```
def prt_star():
    print('*'*10)
def prt_shap():
    print('#'*10)
def func1(fname):
    fname()
def func2(fname):
    fname

func1(prt_star)     #함수 호출 함수 func() 이용 함수 호출
func2(prt_shap())
```

VII. 다양한 함수 호출 방법

- **[실습] 학생 성적 관리 메뉴 호출(1)**

❖ Ch10-FunctionCall.py

◆ 학생 성적관리 메뉴에서 작업 선택에 대한 함수 호출이 변수로 처리되도록 구성하시오.

```
def find_student():      ##TASKS_fn[1][1] : 학번으로 student 딕셔너리 검색
    pass
def find_dept():         ##TASKS_fn[1][2] : 부서코드로 dept 딕셔너리 검색
    pass
def find_prof():         ##TASKS_fn[1][3] : 교수아이디로 prof 딕셔너리 검색
    pass
def find_subject():        ##TASKS_fn[1][4] : 과목코드로 subject 딕셔너리 검색
    pass
def find_stscore():     ##TASKS_fn[1][5] : 학번으로 stscore 리스트 검색
    pass
def list_stscore_st():     ##TASKS_fn[1][1] : 학번으로 stscore 출력
    pass
def list_stscore_sub():     ##TASKS_fn[1][2] : 과목코드로 stscore 출력
    pass
def in_stscore_list():       ##TASKS_fn[4][5] : stscore 리스트 추가 및 성적.txt 파일 갱신
    pass
def fpass():    #없는 메뉴 pass 처리
    pass
```

02. 함수의 사용

VII. 다양한 함수 호출 방법

- **[실습] 학생 성적 관리 메뉴 호출(2)**

❖ Ch10-FunctionCall.py

```
def sel_tasks(idx):      ##(최상위) 작업 메뉴 선택
    while True:
        tlen = 0
        for x in TASKS[idx]:
            tlen += len(x)*2+5    #타이틀 문자 수 계산
        print("_"*tlen)
        seq = 0
        print("[%s]" %MENUS.get(idx), end = ' ')  #메뉴 그룹 표시(Top, 검색, 현황, ...)
        for mname in TASKS[idx]:  #선택 가능 차상위 메뉴 표시
            print("%d:%s" %(seq, mname), end = '    ')
            seq += 1
        selno = input("₩n[작업 선택]> ")
        if len(selno) == 0:
            selno = '0'
        if int(selno) >= len(TASKS[idx]):  #선택 범위 초과
            continue
        return int(selno)

def task_fn_call(idx):      ##(차상위) 작업 메뉴 선택 및 관련 함수 호출
    while True:
        selno = sel_tasks(idx)  #선택된 최상위 메뉴에 대한 차상위 메뉴 선택 호출
        if selno != 0:
            print("[[%s: %s]]" % (MENUS.get(idx), TASKS[idx][selno]))
        if selno == 0:
            break
        else:
            TASKS_fn[idx][selno]()      #선택 작업 관련 함수 호출
```

VII. 다양한 함수 호출 방법

- **[실습] 학생 성적 관리 메뉴 호출(2)**

❖ Ch10-FunctionCall.py

◆ 학생 성적관리 메뉴에서 작업 선택에 대한 함수 호출이 변수로 처리되도록 구성하시오.

```
MENUS = {0:'Top:', 1:'검색', 2:'현황', 3:'통계',  4:'데이터관리'} #최상위 메뉴
TASKS = [['종료', '검색', '현황', '통계', '데이터관리'],              #최상위 메뉴 구성
         ['복귀', '학생', '학과', '교수', '과목', '성적'],            #차상위 메뉴(검색)
         ['복귀', '성적_통지서', '과목별_성적'],                      #차상위 메뉴(현황)
         ['복귀', '과목_성적', '교수_과목', '학과_과목', '학과_교수'], #차상위 메뉴(통계)
         ['복귀', '학생', '학과', '교수', '과목', '성적']]          #차상위 메뉴(데이터관리)
TASKS_fn = [[fpass, fpass, fpass, fpass, fpass],              #차상위 메뉴 호출 함수명(TASKS와 대응)
            [fpass, find_student, find_dept, find_prof, find_subject, find_stscore], #차상위 메뉴 함수(검색)
            [fpass, list_stscore_st, list_stscore_sub],       #차상위 메뉴 함수(현황)
            [fpass, fpass, fpass, fpass, fpass],              #차상위 메뉴 함수(통계)
            [fpass, in_stscore_list, fpass, fpass, fpass, fpass]] #차상위 메뉴 함수(데이터관리)

########## Main ####################################
while True:          #작업 선택
    selno = sel_tasks(0)      #상위 작업 번호 선택
    if selno == 0:
        break
    else:
        task_fn_call(selno)    #하위 작업 선택 및 관련 함수 호출
####################################################
```

03

함수의 활용

03. 함수의 활용

실습 10-7 함수와 딕셔너리를 이용한 환율 계산기 만들기 code10-07.py

① 국가명을 딕셔너리의 키로 하고, 환율과 단위를 튜플로 묶어서 값으로 저장

표 10-1 딕셔너리에 저장할 환율 정보

국가	환율(원)	단위
미국	1188.50	달러
영국	1570.13	파운드
러시아	15.01	루블
중국	173.93	위안

```
moneyInfo = {"미국":(1188.50, "달러"), "영국":(1570.13, "파운드"),
             "러시아":(15.01, "루블"), "중국":(173.93, "위안")}
```

03. 함수의 활용

실습 10-7 함수와 딕셔너리를 이용한 환율 계산기 만들기 code10-07.py

② 국가명과 금액을 매개변수 n, m에 전달받아 환전 금액을 계산하는 함수를 정의

```
def calc_money(n, m) :
    moneyInfo = {"미국":(1188.50, "달러"), "영국":(1570.13, "파운드"),
                 "러시아":(15.01, "루블"), "중국":(173.93, "위안")}
    if n in moneyInfo:
        rate, unit = moneyInfo.get(n)                    # 검색한 값(튜플)을 두 변수에 나누어 담기
        print("%d(원)을 %.2f(%s)로 교환합니다." % (m, m/rate, unit))
    else :
        print("국가를 잘못 입력했습니다.")
```

③ 사용자로부터 환전할 금액과 국가를 입력받은 후, 정의한 함수를 호출하는 순서로 전체 프로그램을 작성

```
01  def calc_money(n, m) :                          # 함수 정의
02      moneyInfo = {"미국":(1188.50, "달러"), "영국":(1570.13, "파운드"),
03                   "러시아":(15.01, "루블"), "중국":(173.93, "위안")}
04      if n in moneyInfo:
05          rate, unit = moneyInfo.get(n)    # 검색한 값(튜플)을 두 변수에 나누어 담기
06          print("%d(원)을 %.2f(%s)로 교환합니다." % (m, m/rate, unit))
```

❖ 딕셔너리.get(key); 딕셔너리의 값을 키로 접근하는 방법

03. 함수의 활용

실습 10-7 함수와 딕셔너리를 이용한 환율 계산기 만들기 code10-07.py

③ 사용자로부터 환전할 금액과 국가를 입력받은 후, 정의한 함수를 호출하는 순서로 전체 프로그램을 작성

```
07      else :
08          print("국가를 잘못 입력했습니다.")
09
10  money = 0    # 금액
11  nation = 0   # 국가
12
13  try:
14      money = int(input("환전하려는 금액(원) : "))
15  except:
16      print("금액은 숫자로 입력하세요.")
17  else:
18      nation = input("국가(미국/영국/러시아/중국) : ")
19      calc_money(nation, money)          # 함수 호출
```

④ 작성한 프로그램을 다양하게 실행

```
환전하려는 금액(원) : 200000
국가(미국/영국/러시아/중국) : 중국
200000(원)을 1149.89(위안)로 교환합니다.
```

03. 함수의 활용

실습 10-8 공의 포물선 그리기 code10-08.py

- 지면에서 하늘로 쏘아올린 공의 x초 후의 높이를 y라고 하고, 40*x-2*(x**2)의 식으로 계산된다고 가정
- 공이 다시 지면에 떨어질 때까지의 시간이 몇 초가 걸리는지 출력하고, 공의 움직임을 포물선으로 그리기
- 터틀 그래픽스를 이용하여 x 좌표는 초의 값으로, y 좌표는 계산한 높이 값으로 하여 공의 이동 경로 그리기

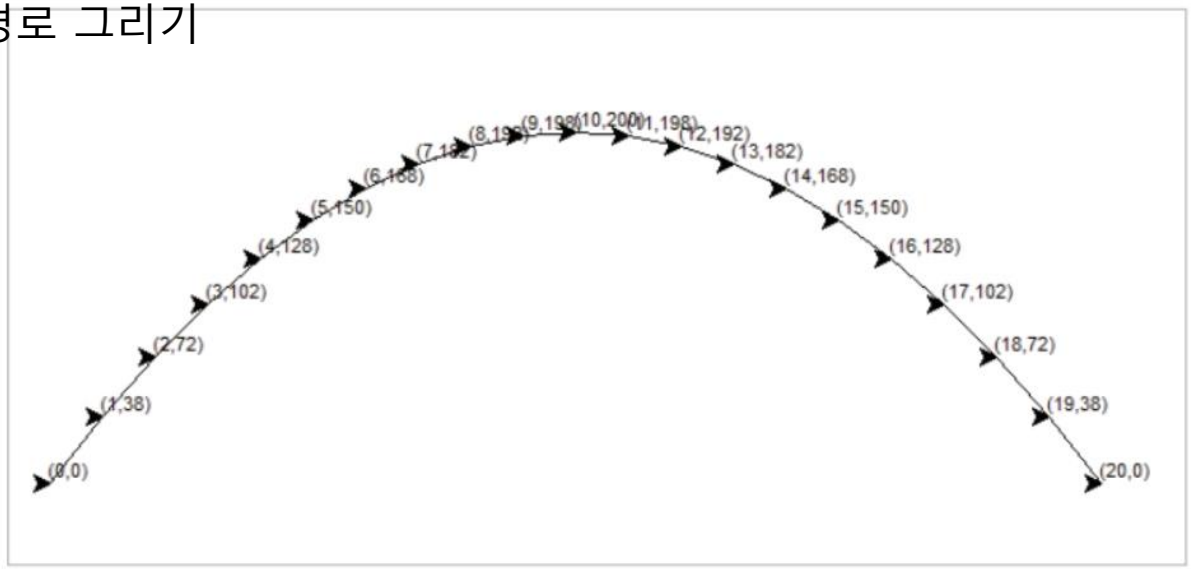


그림 10-21 공의 포물선 그리기

03. 함수의 활용

실습 10-8 공의 포물선 그리기 code10-08.py

① 초 단위 **시간(x)**에 따라 공의 **높이(y)**를 계산하고 해당 좌표(x, y)를 표시(stamp( ))하는 함수 draw_arc( )를 정의

```
from turtle import *

def draw_arc(x) :
    y = 40 * x - 2 * (x ** 2)          # 공의 높이를 계산하는 수식
    if y >= 0 :                        # 공이 지면 위에 있을 때만 그림 그리기
        goto(x, y)
        write("(%d,%d)" %(x, y))       # (x,y) 좌표 값을 터틀 윈도우에 쓰기
        stamp()
    return y                           # 공의 높이를 반환
```

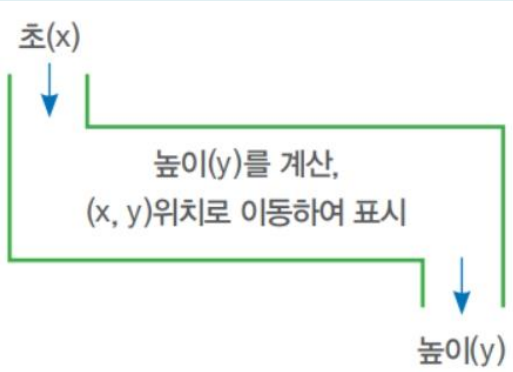


그림 10-22 함수의 정의

03. 함수의 활용

실습 10-8 공의 포물선 그리기 code10-08.py

② 1초부터 시작해서 공의 높이가 지면 위에 있는 동안 그리기 동작을 계속하도록 함수를 반복 호출

```
01  from turtle import *
02
03  def draw_arc(x) :                          # 함수 정의
04      y = 40 * x - 2 * (x ** 2)
05      if y >= 0 :
06          goto(x, y)
07          write("(%d,%d)" %(x, y))
08          stamp()
09      return y
10
11  shape('classic')                  # 터틀 모양 설정
12  sec = 1                           # 시간(초)의 초기화
13  while draw_arc(sec) > 0 :         # 공이 지면 위에 있는 동안 반복
14      sec += 1                      # 1초 증가
15
16  print("공이 떨어지는 데 %d초 걸렸네요." % sec)
```

03. 함수의 활용

실습 10-8 공의 포물선 그리기 code10-08.py

③ 초당 간격을 30픽셀로 설정하고 다시 그려 보자. 06행의 x를 x * 30으로 변경

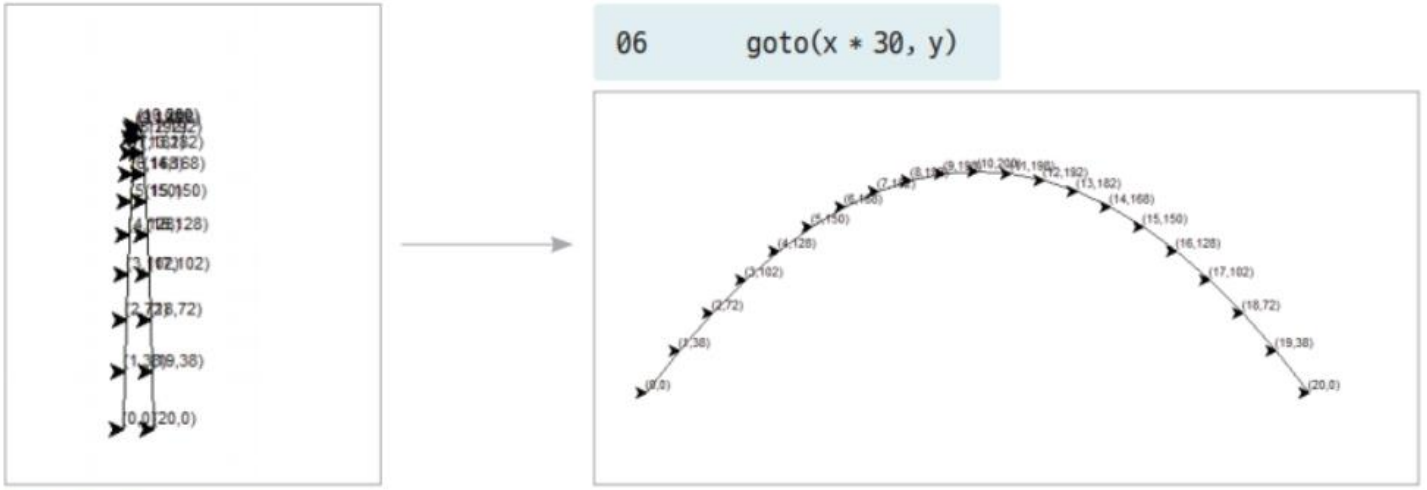


그림 10-23 실행 결과

공이 떨어지는 데 20초 걸렸네요.

➢ 발사 각도를 다양하게 적용하여 보시오.

goto(x*(90-t), y) #(90-t)로 조정, t는 발사각도(0~90)

03. 함수의 활용

실습 10-9 함수를 이용한 사칙연산기 만들기 code10-09.py

- 연산자별로 함수를 정의하고, 두 개의 정수를 인수로 전달받아 계산한 결과를 반환

그림 10-24 사칙연산을 수행하는 프로그램 동작 순서

① 4개의 함수는 두 정수를 인수로 전달받아 계산한 결과를 반환하는 형태로 동일하게 구성

```
def f_add(x, y) :                    # 덧셈 함수
    return x + y
```

```
def f_sub(x, y) :                    # 뺄셈 함수
    return x - y
```

```
def f_mul(x, y) :                    # 곱셈 함수
    return x * y
```

```
def f_div(x, y) :                    # 나눗셈 함수
    return x / y
```

03. 함수의 활용

실습 10-9 함수를 이용한 사칙연산기 만들기 code10-09.py

② 프로그램의 동작 순서를 고려하여 입력과 함수 호출, 출력을 위한 코드를 함수 정의에 추가하여 다음과 같이 전체 프로그램을 작성

```
01  def f_add(x, y) :
02      return x + y
03
04  def f_sub(x, y) :
05      return x - y
06
07  def f_mul(x, y) :
08      return x * y
09
10  def f_div(x, y) :
11      return x / y
12
13  try :
14      num1 = int(input("정수 입력 : "))
15      num2 = int(input("정수 입력 : "))
```

03. 함수의 활용

실습 10-9 함수를 이용한 사칙연산기 만들기 code10-09.py

② 프로그램의 동작 순서를 고려하여 입력과 함수 호출, 출력을 위한 코드를 함수 정의에 추가하여 다음과 같이 전체 프로그램을 작성

```
16      operator = int(input("연산 종류(1:덧셈 2:뺄셈 3:곱셈 4:나눗셈) : "))
17  except :
18      print("정수를 입력하세요.")
19  else :
20      if 1 <= operator <= 4 :
21          if operator == 1 :
22              result = f_add(num1, num2)
23          elif operator == 2 :
24              result = f_sub(num1, num2)
25          elif operator == 3 :
26              result = f_mul(num1, num2)
27          else :
28              result = f_div(num1, num2)
29          print("계산 결과 = ", result)
30      else :
31          print("잘못된 연산자입니다.")
```

03. 함수의 활용

실습 10-9 함수를 이용한 사칙연산기 만들기 code10-09.py

③ 작성한 프로그램을 다양하게 실행

```
정수 입력 : 5
정수 입력 : 8
연산 종류(1:덧셈 2:뺄셈 3:곱셈: 4:나눗셈) : 3
계산 결과 = 40
```

```
정수 입력 : 5
정수 입력 : 8
연산 종류(1:덧셈 2:뺄셈 3:곱셈: 4:나눗셈) : 4
계산 결과 = 0.625
```

```
정수 입력 : 10.5
정수를 입력하세요.
```

```
정수 입력 : 5
정수 입력 : 8
연산 종류(1:덧셈 2:뺄셈 3:곱셈: 4:나눗셈) : 0
잘못된 연산자입니다.
```

03. 함수의 활용

실습 10-10 터틀 그림판 만들기 code10-10.py

- 터틀 그래픽스의 메소드를 이용하여 단계별로 코드를 작성

표 10-2 그림판 만들기에 필요한 터틀 그래픽스의 메소드

메소드명	동작	예시	인수 설명
onscreenclick()	마우스를 클릭하면 지정한 함수를 호출	turtle.onscreenclick(func)	함수명
onkeypress()	특정 키를 클릭하면 지정한 함수를 호출	turtle.onkeypress(func,"Up")	함수명과 키 이름
listen()	키 입력 대기 모드	turtle.listen()	-
setheading(), seth()	이동 방향을 지정한 각도로 설정	turtle.setheading(90), turtle.seth(180)	동쪽부터 시계 반대 방향으로 각도 증가 (동:0, 북:90, 서:180, 남:270)

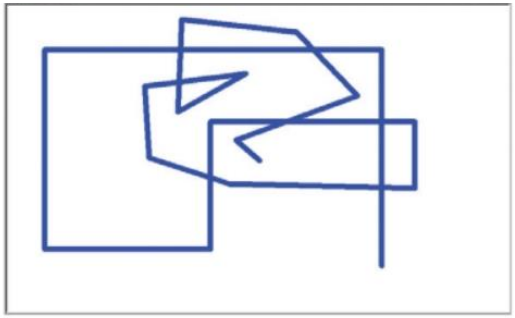


그림 10-25 실행 결과

03. 함수의 활용

실습 10-10 터틀 그림판 만들기 code10-10.py

① 그림을 그리기 위한 터틀 윈도우를 먼저 생성

```
from turtle import *

hideturtle()                # 커서를 숨김
pensize(10)
pencolor("blue")
```

② 마우스를 클릭한 위치로 이동시켜 선을 그리는 함수 go( )를 정의하고, 마우스 이벤트 처리를 위한 onscreenclick( ) 메소드의 인수로 사용

```
def go(x, y):               # 마우스 클릭 때의 동작을 정의, 현재 좌표를 인수로 사용
    goto(x, y)

onscreenclick(go)           # 마우스를 클릭하면 go() 함수를 실행
```

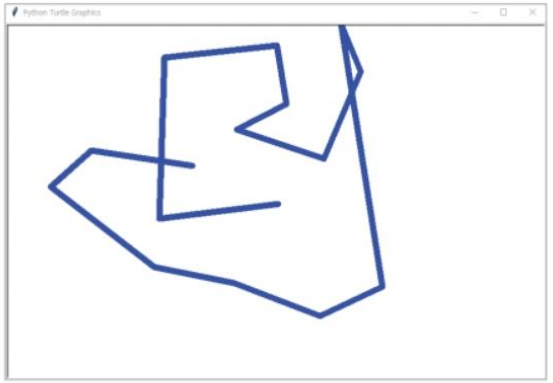


그림 10-28 마우스 클릭으로 그림 그리기

03. 함수의 활용

실습 10-10 터틀 그림판 만들기 code10-10.py

③ 각 키에 해당하는 네 개의 함수를 정의하고 키 이벤트 처리를 위한 onkeypress( ) 메소드의 인수로 지정, 코드의 마지막 에는 listen( ) 메소드를 사용하여 키 입력을 기다리도록 대기 모드로 설정

```
def east() :                    # 오른쪽 방향키가 눌렸을 때의 동작
    setheading(0)               # 이동 방향 설정
    forward(10)
def north() :                   # 위쪽 방향키가 눌렸을 때의 동작
    setheading(90)
    forward(10)

def west() :                    # 왼쪽 방향키가 눌렸을 때의 동작
    seth(180)
    forward(10)

def south() :                   # 아래쪽 방향키가 눌렸을 때의 동작
    seth(270)
    forward(10)
```

03. 함수의 활용

실습 10-10 터틀 그림판 만들기 code10-10.py

③ 각 키에 해당하는 네 개의 함수를 정의하고 키 이벤트 처리를 위한 onkeypress( ) 메소드의 인수로 지정, 코드의 마지막 에는 listen( ) 메소드를 사용하여 키 입력을 기다리도록 대기 모드로 설정

```
onkeypress(north, "Up")          # 키 이벤트 처리
onkeypress(south, "Down")
onkeypress(west, "Left")
onkeypress(east, "Right")

listen()                         # 키 입력 대기
```

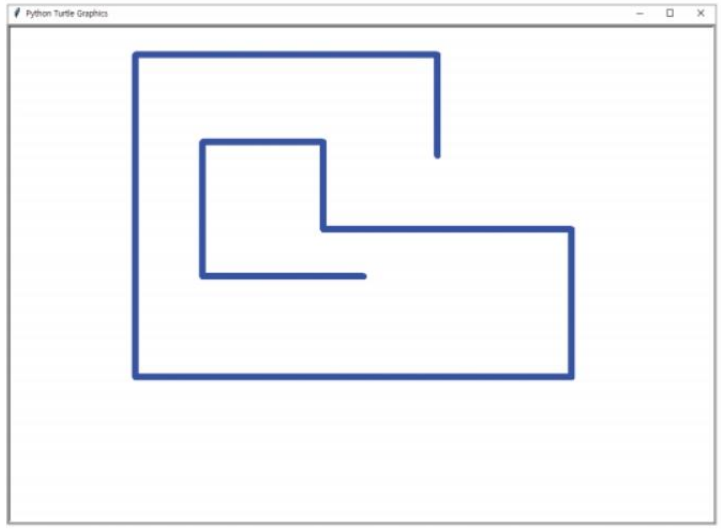


그림 10-29 키보드 방향키로 그림 그리기

03. 함수의 활용

실습 10-10 터틀 그림판 만들기 code10-10.py

③ 함수의 정의는 가급적 전체 프로그램 중 앞 부분에 기술하여 가독성을 높일 것

```
01 from turtle import *
02
03 def go(x, y):              # 마우스 클릭 때의 동작을 정의, 현재 좌표를 인수로 사용
04     goto(x, y)
05
06 def east() :               # 오른쪽 방향키가 눌렸을 때의 동작
07     setheading(0)          # 이동 방향 설정
08     forward(10)
09
10 def north() :              # 위쪽 방향키가 눌렸을 때의 동작
11     setheading(90)
12     forward(10)
13
14 def west() :               # 왼쪽 방향키가 눌렸을 때의 동작
15     seth(180)
```

03. 함수의 활용

실습 10-10 터틀 그림판 만들기 code10-10.py

③ 함수의 정의는 가급적 전체 프로그램 중 앞 부분에 기술하여 가독성을 높일 것

```
16      forward(10)
17
18  def south() :               # 아래쪽 방향키가 눌렸을 때의 동작
19      seth(270)
20      forward(10)
21
22  hideturtle()                # 터틀을 화면에서 숨김
23  pensize(10)
24  pencolor("blue")
25  onscreenclick(go)           # 마우스를 클릭하면 go() 함수를 실행
26  onkeypress(north, "Up")     # 키 이벤트 처리
27  onkeypress(south, "Down")
28  onkeypress(west, "Left")
29  onkeypress(east, "Right")
30  listen()                    # 키 입력 대기
31  done()
```

Thank You !