7장 함수

이번 장에서는 함수의 개념을 익히고, 함수를 작성하는 방법과 사용법을 익힌다. 또한 함수를 모아놓은 개념인 모듈을 이해한다.

7.1 함수(function) 일반

함수는 무엇을 입력하면 어떤 것을 돌려주는 상자라 생각할 수 있다. 지금까지 앞의 장들에서 몇 가지 함수를 이미 사용하였다. print(), range(), zip(), len(), input(), int() 등이 그것이다. print() 함수는 괄호 안에 문자열을 넣으면 문자열을 화면에 출력해주는 함수이고, len() 함수는 리스트 등을 넣으면 항목의 개수를 돌려주는 함수이다.

파이썬에서 자체에서도 함수를 제공하지만 사용자가 직접 만들어서 사용할 수도 있다. 함수는 다음 형식으로 사용한다.

```
함수명(입력값1, 입력값2, ...)
```

함수는 한 번 만들어 놓으면 여러 번 반복하여 사용할 수 있다. 따라서 반복적으로 코딩해야 할 부분을 함수로 만들어 놓으면 필요한 경우 불러 쓰기만 하면 되기 때문에 프로그래밍이 쉬워진다.

7.2 함수의 형식

함수는 매개변수를 입력받은 후 그 매개변수를 가공 및 처리해서 반환 값을 돌려준다. 다음의 예는 두 정수를 입력받아서 두 정수의 합계를 반환하는 sum() 함수를 정의하여 사용한다.

```
def sum(v1, v2):
    result = v1 + v2
    return result
```

```
# 메인 부분
hap = sum(10, 20)
print("sum() 함수의 결과는 %d이다." % hap)
```

```
<실행 결과>
sum() 함수의 결과는 30이다.
```

위의 예제에서 처음 세 줄은 sum() 이라는 함수의 선언 부분이다. 지금까지 작성한 프로그램은 위에서부터 차례대로 하나씩 문장들이 실행되었다. 그러나 함수는 위에 선언한다고 해서 그것부터 실행되는 것은 아니다. 함수는 어딘가에서 ____해야지만 함수 안의 내용이 차례대로 실행된다. 따라서 위의 코드에서 처음으로 실행되는 문장은 함수 부분은 건너뛰고 "hap = sum(10, 20)" 이 된다.

함수를 호출하는 방법은 '함수이름(값)' 형식이다. "hap = sum(10, 20)" 에서 우리는 sum() 함수를 호출한 것이다. 그리하면 비로소 sum() 함수가 실행된다. 위의 예제의 경우는 첫 번째 세 줄이 실행되고 'return 변수이름' 으로 값을 함수를 호출한 곳으로 넘겨주게 되고 이 값을 변수 hap에 대입하여 print() 함수에서 출력한다.

함수 선언 부분에서 괄호 안의 v1, v2를 _____라 한다. 매개 변수는 해당 함수의 외부와 내부를 매개하는 역할을 하는데, 함수의 입력 값이 매개변수를 통해 함수 내부로 전달이 되고 전달된 값을 잘 이용하여 명령을 실행한 후 원하는 값을 돌려주게 된다. 위의 예제에서는 함수 호출시 10과 20이 각각 매개 변수 v1과 v2에 대입되고, 함수 내부에서는 v1과 v2에 대입된 값을 더하여 변수 result에 대입한다. 그리고 최종적으로 result에 저장된 30을 함수 호출 부분으로 돌려준다. 즉 sum(10, 20)의 결과 값이 30이 되는 것이다. 그러면 30이 다시 변수 hap에 대입되어 그 다음 작업을 한다.

위 내용을 정리하면 다음과 같다.

1. 함수 호출
   sum(10, 20)로 함수를 호출한다.
   선언 부분 sum(v1, v2)에서 매개변수 v1에는 10이, v2에는 20이 대입 된다.
2. 함수 실행
   v1과 v2를 더한 결과를 변수 result에 대입한다.

result의 값을 호출한 곳에 돌려준다.

3. 반환값 이용

반환된 값을 변수 hap에 대입한다.

다음은 구구단의 단을 매개변수로 입력받아 해당 단의 구구단을 출력하는 함수를 정의하여 2단부터 9단까지 출력하는 예제이다.

```
def gugudan(dan):
    for i in range(1, 10, 1):
        print("%d x %d = %d" % (dan, i, dan * i))

#메인 부분
for i in range(2, 10, 1):
    gugudan(i)
```

<실행 결과>

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
・・・ (중간 생략) ・・・
9 x 8 = 72
9 x 9 = 81
```

위의 예제는 메인 부분에서 gugudan() 함수를 _번 호출하였다. 호출 할 때마다 매개변수인 ___에 2부터 9까지의 값을 대입하여 실행 한 결과이다. 이처럼 반복적인 작업을 수행할 때 함수로 만들어 놓으면 편하고 보기 좋게 프로그래밍을 할 수 있다. 그리고 한 가지 주목할 내용은 모든 함수가 결과 값을 반드시 반환해야하는 것은 아니란 것이다. 위의 예제에서처럼 함수의 마지막에 'return' 이 없고 함수 안에서 어떠한 일만하는 경우도 있다. (위 예제에서는 반환 없이 구구단만 출력하였다.)

● 지역 변수와 전역 변수

함수 안에서 변수를 새로 만들어 사용하면 그 변수는 _____에서만 사용이 가능하다. 이를 지역 변수라 부른다. 함수 선언 부분의 매개 변수 또한 해당 함수 안에서만 사용 가능한 ____ 변수이다.

이와 대비되는 개념이 전역 변수이다. _____는 함수 안과 밖을 가리지 않고 프로그램 전체에서 사용이 가능한 변수를 말한다. 주로 메인 부분 시작 전에 변수를 선언해 놓으면 그 변수는 함수가 실행되기 전에 선언이 되기 때문에 메인 부분과 함수 내부에서 모두 사용 가능한 전역 변수가 된다.

```
#함수 선언 부분
def func1():
    a = 100
    print("func1()에서 a의 값 : %d" % a)

def func2():
    print("func2()에서 a의 값 : %d" % a)

#전역 변수 부분
a = 200

#메인 부분
func1()
func2()
```

```
<실행 결과>
func1()에서 a의 값 : 100
func2()에서 a의 값 : 200
```

위의 코드를 실행하면 _____ 부분이 가장 먼저 실행된다. 그래서 변수 a가 선언 되면서 200이 대입된다. 그런 다음 메인 부분에서 func1()을 호출하였기 때문에 func1()이 실행된다. 여기서 또다시 변수 a가 선언 되면서 100이 대입되는데, 이는 전역 변수 a와는 이름이 같지만 같은 메모리 공간을 공유하지 않는 서로 다른 변수이다. 그래서 func1()에서는 지역 변수 a의 값인 100이 출력되었다. 그리고 함수 안에서 선언되었기 때문에 함수가 종료되면 해당 변수는 더 이상 사용할 수 없다. 그 다음 메인 부분에서 func2()를 호출하였는데 func2()에서는 선언 없이 변수 a를 출력에 사용하였다. 이미 a가 전역 변수로 선언되어 있었기 때문에 전역 변수 a의 값인 200이 출력된다.

만약 func1()에서 지역 변수 a가 아니라 이미 선언된 전역 변수 a의 값을 사용

하거나 변경시키고 싶다면 아래와 같이 변수 이름 앞에 'global' 이란 키워드를 사용하여야 한다. 그러면 이때의 a는 새로운 지역 변수가 아닌 전역 변수 a를 가리킨다. 그러면 func2()호출을 통해 전역 변수 a가 200에서 100으로 변경되었음을 확인할 수 있다.

```
#함수 선언 부분
def func1():
    global a
    a = 100
    print("func1()에서 a의 값 : %d" % a)

def func2():
    print("func2()에서 a의 값 : %d" % a)

#전역 변수 부분
a = 200

#메인 부분
func1()
func2()
```

```
<실행 결과>
func1()에서 a의 값 : 100
func2()에서 a의 값 : 100
```

7.3 함수의 반환 값과 매개 변수

앞에서 우리는 반환 값이 하나인 함수와 반환 값이 없는 함수를 살펴보았다. 반환 값이 하나이면 'return 반환값(또는 변수이름)' 으로 값을 반환하면 되고, 반환 값이 없는 함수는 'return' 을 생략하면 된다. 그러면 반환 값이 여러 개인 함수는 어떻게 해야 할까? 결론부터 말하면 반환 값이 여러 개인 함수는 ____에 반환할 값을 여러 개 담에 반환하면 된다.

아래는 숫자 두 개를 매개변수로 받아서 합과 차를 반환하는 함수를 보여준다.

```
#함수 선언 부분
def multi_return(v1, v2):
    retList = []
    retList.append(v1+v2)
    retList.append(v1-v2)
    return retList

#메인 부분
myList = multi_return(10, 20)
hap = myList[0]
cha = myList[1]
print("합: %d, 차: %d" % (hap, cha))
```

```
<실행 결과>
합: 30, 차: -10
```

7.4 매개 변수 전달

숫자를 매개변수로 입력 받아 합계를 구하는 함수를 생각해보자. 만약 입력할 매개변수의 개수가 2개인 경우와 3개인 경우가 있다고 할 때, 이러한 역할을 하는 함수는 아래와 같이 따로 정의해 주는 것이 일반적이다.

```
#함수 선언 부분
def sum2_func(v1, v2):
    result = v1 + v2
    return result

def sum3_func(v1, v2, v3):
    result = v1 + v2 + v3
    return result

#메인 부분
hap = sum2_func(100, 200)
print("매개 변수가 두 개인 함수 호출 결과 : %d" % hap)
hap = sum3_func(100, 200, 300)
print("매개 변수가 세 개인 함수 호출 결과 : %d" % hap)
```

```
<실행 결과>
매개 변수가 두 개인 함수 호출 결과 : 300
```

```
매개 변수가 세 개인 함수 호출 결과 : 600
```

그렇다면 매개 변수의 개수에 상관없이 값을 함수에 전달하여 합계를 구하게 할 수는 없을까? 매개 변수 개수에 따라 매번 함수를 선언해야한다면 상당히 힘든 작업이 될 것임은 틀림이 없을 것이다. 다행이 파이썬에서는 이러한 문제를 해결해주는 편리한 기능을 제공한다. 이를 가변 매개변수 방식이라 한다. 형식은 함수의 매개변수명 앞에 *를 붙이면 된다. *를 붙이면 매개변수가 ____ 형식으로 넘어와 튜플을 처리하는 방식으로 함수 안에서 사용할 수 있다.

```
#함수 선언 부분
def sum_func(*para):
    result = 0
    for num in para:
        result += num
    return result

#메인 부분
hap = sum_func(100, 200)
print("매개 변수가 두 개인 함수 호출 결과 : %d" % hap)
hap = sum_func(100, 200, 300)
print("매개 변수가 세 개인 함수 호출 결과 : %d" % hap)
hap = sum_func(100, 200, 300, 400, 500)
print("매개 변수가 여러 개인 함수 호출 결과 : %d" % hap)
```

```
<실행 결과>
매개 변수가 두 개인 함수 호출 결과 : 300
매개 변수가 세 개인 함수 호출 결과 : 600
매개 변수가 여러 개인 함수 호출 결과 : 1500
```

메인 부분에서 sum_func(100, 200)으로 호출하면 함수 부분에서 (100, 200) 형식의 이름이 para인 튜플로 전달되어 함수 내부에서 사용된다. 튜플 형식으로 함수에 전달되면 for 문을 이용하여 튜플 내의 항목을 하나씩 변수 num에 대입하여 result에 누적하고 result에 최종 누적된 값을 반환한다.

함수의 매개변수 앞에 **를 붙이면 딕셔너리 형식으로 전달받는다.

```
#함수 선언 부분
def dic_func(**para):
    for key in para.keys():
        print("%s은 %s입니다." % (key, para[key]))

#메인 부분
dic_func(apple='사과', orange='오렌지', mango='망고')
```

<실행 결과>

apple은 사과입니다.
orange은 오렌지입니다.
mango은 망고입니다.