**6강. 함수와 모듈 활용하기**

**6.1 파이썬 기본 용어 정의하기**

함수(function)

함수는 어떤 특정한 일을 수행하도록 만든 코드의 덩어리, 명령어 덩어리이다. 마치 집짓기를 하는데 사용되는 레고 조각 같은 것이 함수라 할 수 있다. 모양이나 색깔이 다른 여러 레고 조각을 이용하여 집을 지어나가듯이, 파이썬 프로그램도 명령어 뿐만 아니라 함수들을 이용하여 좀 더 빠르고 편리하게 작성해갈 수 있다.

또한, 함수는 하나의 프로그램에만 사용되는 것이 아니라 여러 프로그램에서 두루 사용할 수 있다. 하나의 레고 조각을 A 완성체를 만들 때 사용하기도 하고, B 완성체를 만들 때 사용하기도 하는 것과 같다.

함수는 def 키워드를 사용하여 정의하고(define), 함수명을 이용하여 호출(call)하여 사용한다.

모듈(module) -함수보다 큰개념

모듈이란 어떤 크고 복잡한 완성체를 만들기 위해 필요한 일부분을 의미한다. 부분들을 모아 하나의 완성체를 조립할 수 있고, 완성체를 분해하여 손상되지 않는 부분들로 손쉽게 분리할 수 있다면 이 상태를 모듈화(modular)되어 있다고 말하며, 각 부분들을 모듈(module)이라고 부른다.

파이썬으로 프로그램을 작성할 때 통으로 하나로 작성하지 않고 작은 모듈들로 나누어 작성하고 모듈들을 조합하고 이어붙여 큰 프로그램을 만들어 간다. 즉, 모듈은 크고 복잡한 기능의 프로그램을 만들어 갈 때 사용할 수 있는 작은 프로그램 덩어리라고 볼 수 있다. 모듈이 하나의 파일로 구성된다는 점을 기억해두기 바란다.

파이썬 프로그래밍에서 모듈화가 필수는 아니지만 모듈화로 얻을 수 있는 이점은 다음과 같다.

첫째, 모듈이라는 단위를 만들어냄으로써 프로그램 파일의 단위가 작아져서 특정 코드를 찾아내기가 용이해진다. 큰 파일 하나로 프로그램을 작성해 놓으면 수정하고자 할 때 그 부분을 찾아내기가 쉽지 않다. 모듈은 하나의 독립된 파일로 만들게 되며, 모듈의 이름만으로도 찾는 부분을 구별할 수 있어 코드관리가 편해진다.

둘째, 하나의 모듈은 하나의 프로그램에만 사용될 수 있는 것이 아니라 또 다른 프로그램을 만들 때도 사용될 수 있다는 점이 중요하다. 독립적인 모듈은 쉽게 재사용(reuse)될 수 있다.

그러면 모듈과 함수는 어떤 관계일까?

앞에서 함수를 비유하면서 레고 조각과 같다고 했다. 레고를 보면 색깔과 모양이 다양하다. 비슷한 모양이나 비슷한 색깔의 레고 조각들을 한데 모아 두면 조립할 때 빠르고 쉽게 원하는 조각을 찾을 수 있다. 이처럼, 모듈은 함수들 중에 비슷한 종류를 모아 하나의 파일로 만든 것이다.

**6.2 함수 만들고 호출하기**

우리가 직접 함수를 정의(define)한 후 호출(call)하여 사용할 수 있다.

함수를 만들 때는 def 이라는 키워드를 이용한다.

실습1) 학교와 학과를 출력하는 함수를 만든 후 이를 호출해 보세요

def printMyDepartment():

print "Sejong University"

print "English Literature"

print

printMyDepartment()

print "Finished!"

Sejong University

English Literature

Finished!

실습1을 수행하면 파이썬에서는 def 키워드 이후의 블록에 대해 새로운 함수로 인식을 하고넘어간다. 따라서 첫 실행문은 함수 호출문이 된다. 함수가 호출이 되면 함수의 블록에 해당하는 명령어들이 차례로 실행된다. 함수의 실행을 마치게 되면 함수 호출문 이후의 명령어들이 실행된다.

실습2) 지금 바로 다음을 실행하여 결과를 확인하세요

printMyDepartment()

printMyDepartment()

printMyDepartment()

Sejong University

English Literature

Sejong University

English Literature

Sejong University

English Literature

한 번 함수를 만들어 두면 필요할 때 반복해서 여러 번 호출할 수 있다. 따라서, 코드의 재사용을 높이는 효과적인 방법이 된다.

또한, 크고 복잡한 프로그램에 코드들이 길게 있는 것에 비해 기능별로 함수를 만들어 함수 호출문들이 나열된다면 프로그램의 전체적인 논리흐름을 파악하는데 효과적이 된다.

**6.3 함수에 인수 전달하기**

함수이름 뒤에 붙은 괄호()에는 인수(argument)를 전달할 수 있다. 프로그램에서 함수 호출시 인수는 값을 전달하기 위한 용도로 존재한다. 인수를 잘 활용하면 논리적 흐름은 동일하되 호출할 때마다 다른 범위를 처리를 하는 범용적인 함수를 만들 수 있다.

앞 강의에서 다루었던 구구단 프로그램을 범용적인 함수로 만들어 보자.

for multiplier in range(5,8):

for i in range(1,10):

print multiplier, " x ", i, " = ", multiplier\*i

print "===================="

위의 프로그램은 5단부터 7단까지의 구구단 표를 만들어준다. 원하는 단의 범위를 주면 구구단 표를 만들어주는 함수로 전환해 보자.

(1)함수의 이름을 먼저 정해보자. 함수명은 “intMultiplier”로 정하자.

(2)다음은 함수를 정의하는 명령어를 기억해보자. 함수 정의 명령어는 def 였다.

(3)다음은 함수의 인수를 정해보자. 시작하는 단을 initNum, 끝나는 단을 lastNum으로 명명하자.

그러면, 함수 정의문의 첫 번째 줄은 다음과 같이 만들게 된다.

def intMultiplier(initNum, lastNum):

함수 정의문의 첫 줄 끝에는 명령어들의 블록이 와야 하므로 블록의 시작을 의미하는 콜론(:)을 붙여준다.

(4)다음은 블록에 명령어들을 적어준다. 위에 제시한 프로그램을 인수를 이용하여 조금만 수정하면 된다. 첫 번째 for문장의 range(5,8)이라고 한 부분을 인수를 이용한 수식으로 바꿔주는 것이다. range()함수의 특성상 range(5,8)이라고 하면 [5,6,7]까지의 값이 반환된다는 점을 고려해서 다음과 같이 프로그램을 만들어주면 된다.

for multiplier in range(initNum,lastNum+1):

실습2) 다음 구구단 함수를 호출한 결과를 확인해 보세요.

def intMultiplier(initNum, lastNum):

for multiplier in range(initNum,lastNum+1):

for i in range(1,10):

print multiplier, " x ", i, " = ", multiplier\*i

print "==================== "

intMultiplier(3,5)

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

====================

4 x 1 = 4

4 x 2 = 8

4 x 3 = 12

4 x 4 = 16

4 x 5 = 20

4 x 6 = 24

4 x 7 = 28

4 x 8 = 32

4 x 9 = 36

====================

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

====================

퀴즈) 11단부터 15단까지의 구구단 표를 출력하려면 함수를 어떻게 호출하면 될까요?

def intMultiplier(initNum, lastNum):

for multiplier in range(initNum,lastNum+1):

for i in range(1,10):

print multiplier, " x ", i, " = ", multiplier\*i

print "==================== "

intMultiplier(11,15)

11 x 1 = 11

11 x 2 = 22

11 x 3 = 33

11 x 4 = 44

11 x 5 = 55

11 x 6 = 66

11 x 7 = 77

11 x 8 = 88

11 x 9 = 99

====================

12 x 1 = 12

12 x 2 = 24

12 x 3 = 36

12 x 4 = 48

12 x 5 = 60

12 x 6 = 72

12 x 7 = 84

12 x 8 = 96

12 x 9 = 108

====================

13 x 1 = 13

13 x 2 = 26

13 x 3 = 39

13 x 4 = 52

13 x 5 = 65

13 x 6 = 78

13 x 7 = 91

13 x 8 = 104

13 x 9 = 117

====================

14 x 1 = 14

14 x 2 = 28

14 x 3 = 42

14 x 4 = 56

14 x 5 = 70

14 x 6 = 84

14 x 7 = 98

14 x 8 = 112

14 x 9 = 126

====================

15 x 1 = 15

15 x 2 = 30

15 x 3 = 45

15 x 4 = 60

15 x 5 = 75

15 x 6 = 90

15 x 7 = 105

15 x 8 = 120

15 x 9 = 135

====================

**6.4 함수명과 인수명 지어주기**

함수명이나 인수명을 지을 때는 나름의 명명규칙을 정해 일관되게 지켜가는게 좋다.

첫째는 함수명의 경우 기억하기 좋아야 한다. 호출하여 사용해야하기 때문이다. 둘째로 함수명이나 인수명은 의미가 명확히 전달되도록 짓는게 좋다. 그래야 프로그램에 대한 이해도가 높아지고 쉽게 이용할 수 있다.

퀴즈) 위의 실습 프로그램에서 함수명을 xx로 변경하고, 인수명을 a, b로 변경하여 실행하면 결과가 어떨까요?

3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

====================

4 x 1 = 4

4 x 2 = 8

4 x 3 = 12

4 x 4 = 16

4 x 5 = 20

4 x 6 = 24

4 x 7 = 28

4 x 8 = 32

4 x 9 = 36

====================

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

====================

실습2)에서 인수명을 a, b로 바꾸어도 동일하게 실행된다. 함수명을 xx로 바꾸어도 역시 동일하게 실행된다. 그러나, 프로그램을 재사용하게 될 때는 명령어를 일일이 살펴보아야만 함수의 기능과 인수의 역할을 이해할 수 있어 가독력이 떨어지게 된다. 프로그램도 독자를 고려하여 글을 쓰듯 좋은 습관을 들여 작성하도록 하자.

**6.5 함수의 결과값 반환하기**

함수를 호출했을 때 수행결과값(result)을 전달받을 수 있다. 이를 반환값(return value)라고 부른다. 반환값이 있는 함수를 호출할 때는 명령어에 변수를 사용하거나, 변수가 사용될 수 있는 자리에서 호출해야 한다.

실습3) 다음은 패스트푸드점에서 일하는 직원들의 일급여를 계산하는 프로그램입니다. 반환값을 받기 위해 어떤 조치를 취했는지 살펴보세요

def calculateWage(role, time, overtime, night):

base\_wage = [6500, 8000, 10000]

increase\_rate = [15, 30]

base = 0

if role == 'Mate':

base = base\_wage[0]

elif role == 'Manager':

base = base\_wage[1]

elif role == 'Chief':

base = base\_wage[2]

total\_wage = base \* time + base \* (1 + increase\_rate[0] \* 0.01) + overtime \* base \* (1 + increase\_rate[1] \* 0.01) \* night

return total\_wage

RetVal = calculateWage('Mate', 6, 2, 3)

print int(RetVal), 'won'

calculateWage함수는 role과 하루 근무시간을 입력하면 그 날의 임금을 계산해서 반환해주는 프로그램이다.

실습3)에서 calculateWage함수를 호출할 때 3개의 인수값을 사용하고 있다. 첫 번째 인수값은 role이라는 인수에 할당되고, 두 번째 6이라는 인수값은 time이라는 인수에 할당되며, 세 번째 2라는 인수값은 overtime에 네 번째 3이라는 인수값은 night에 할당된다.

calculateWage함수는 role에 따라서 기본시급을 달리 적용한다.

정규시간에는 기본시급을 적용하고, 오버타임에는 15% 시급을 늘리고, 밤근무시간에는 30% 시급을 늘려준다. 이러한 규칙을 적용하여 totalWage를 계산한 후 return 해준다.

반환값을 갖는 함수는 블록 내에서 반드시 return이라는 명령어를 사용하게 된다. return 문장이 실행되면 해당 함수는 수행이 종료되고 함수 호출문으로 제어가 넘어가게 된다.

**6.6 전역 변수 이용하여 함수 호출하기**

프로그램에서 값을 저장하기 위해 변수가 사용된다. 이 변수들은 컴퓨터의 메모리를 할당받아 그 위치에 이름을 붙인 것이다.

프로그램이 만들어질 때마다 변수들을 계속 만들기만 한다면 컴퓨터는 메모리 부족으로 제대로 작동되지 않을 것이다. 따라서, 필요할 때 메모리를 할당받아 사용한 후 불필요해지면 즉시 메모리를 반납하도록 운영된다. 그런 원리에 따라 변수들은 유효한 범위를 갖게 된다.

실습4) 다음 variableTest함수를 정의한 후 변수들을 print해보세요

def variableTest(arg1, arg2):

print arg1

print arg2

val1 = arg1 \* 2

val2 = arg2 \* 2

print val1

print val2

return val1+val2

>>>RetVal = variableTest(10, 20)

10

20

20

40

>>>print RetVal

10

20

20

40

60

>>>print arg1

Traceback (most recent call last):

File "C:/Python27/4.25.py", line 13, in <module>

print arg1

NameError: name 'arg1' is not defined

>>>print arg2

10

20

20

40

60

Traceback (most recent call last):

File "C:/Python27/4.25.py", line 13, in <module>

print arg1

NameError: name 'arg1' is not defined

>>>print val1

10

20

20

40

60

Traceback (most recent call last):

File "C:/Python27/4.25.py", line 13, in <module>

print arg1

NameError: name 'arg1' is not defined

>>>print val2

10

20

20

40

60

Traceback (most recent call last):

File "C:/Python27/4.25.py", line 13, in <module>

print arg1

NameError: name 'arg1' is not defined

RetVal은 정상적으로 print되지만 arg1, arg2, val1, val2는 print 실행 중 에러가 발생할 것이다. arg1, arg2, val1, val2 변수들은 variableTest 함수의 블록 내에서만 의미있게 사용되는 변수인데, 함수 밖에서 사용하려 했기 때문에 알 수 없는 이름이 된 것이다.

arg1, arg2, val1, val2와 같이 특정 함수 내에서만 통용되는 변수를 지역변수(local variable)이라고 한다.

파이썬에서는 필요 시 전역변수(global variable)를 사용할 수 있다. 이름 그대로 하나의 함수나 프로그램 내에서만 통용되는게 아니라 좀 더 넓은 범위에서 통용되는 변수를 정의하여 사용할 수 있다는 것이다.

실습5) 다음은 variableTest 함수 안에서 RetVal 변수를 사용하고 있습니다. 이 변수는 외부에 정의된 변수임에도 에러가 나지 않고 잘 사용됨을 확인하세요.

def variableTest(arg1, arg2):

print arg1

print arg2

val1 = arg1 \* 2

val2 = arg2 \* 2

print val1

print val2

print RetVal

return val1+val2

>>>RetVal = 0

>>>RetVal = variableTest(10, 20)

10

20

20

40

0

>>>print RetVal

10

20

20

40

0

60

파이썬의 메모리관리 방식에 의해 변수의 유효범위가 결정된다. 파이썬에서는 주 프로그램에 미리 정의된 변수명인 경우 호출되는 함수에서 사용할 수 있다. 다만, 함수에서 변수의 값을 변경하지 않는 경우에 한해서 이다.

실습6) 함수에 RetVal 값을 변경하는 명령어를 추가한 다음 프로그램을 실행하여 결과를 확인하세요.

def variableTest(arg1, arg2):

print arg1

print arg2

val1 = arg1 \* 2

val2 = arg2 \* 2

print val1

print val2

RetVal = 1

print RetVal

return val1+val2

>>>RetVal = 0

>>>RetVal = variableTest(10, 20)

10

20

20

40

1

>>>print RetVal

10

20

20

40

1

60

이 경우에는 함수 내에서 사용된 RetVal은 주 프로그램에서 사용한 RetVal과 이름은 같으나 전혀 다른 변수로 만들어진 것이다. 함수 내에서 주 프로그램의 변수와 동일한 변수명을 사용하면서 값의 할당 등이 일어나게 되면 바로 지역변수로 생성되는 것이다.

그렇다면, 본의아니게 주 프로그램과 함수에서 변수명이 겹쳐 원치않는 결과가 나올 수도 있게 된다. 따라서, 변수명은 겹치지 않도록 작명하는 규칙을 만드는 것이 필요하다, 또한, 전역변수를 써야할 때는 global 이라는 키워드를 사용하여 의도를 분명히 나타내 주는 코딩을 해야할 것이다.

실습7) global 키워드를 사용한 다음 프로그램을 실행하고, 함수 호출 방식의 변화를 살펴보세요.

def variableTest(arg1, arg2):

global RetVal

print arg1

print arg2

val1 = arg1 \* 2

val2 = arg2 \* 2

print val1

print val2

RetVal = val1 + val2

print RetVal, ' of variableTest'

>>>RetVal = 0

>>>print RetVal, ' of main program'

0 of main program

>>>variableTest(10, 20)

0 of main program

10

20

20

40

60 of variableTest

>>>print RetVal, ' of main program'

0 of main program

10

20

20

40

60 of variableTest

60 of main program

**6.7 모듈의 함수 사용하기**

함수를 모아둔게 모듈임 함수보다 더 큰개념

모듈을 독립적인 파이썬 파일로 만들어진다. 이 때, 파일의 명칭이 모듈명칭이 된다.

실습8) 다음 코드대로 my\_module.py 파일을 만들어 모듈을 생성해 보세요.

#"my\_module.py" file

# Other program may use this file

def sumOfDouble(arg1, arg2):

print “sumOfDouble function in my\_module”

val1 = arg1 \* 2

val2 = arg2 \* 2

return val1+val2

===================== RESTART: C:/Python27/my\_module.py =====================

>>>

def multipleOfDouble(arg1, arg2):

print “multipleOfDouble function in my\_module”

val1 = arg1 \* 2

val2 = arg2 \* 2

return val1 \* val2

===================== RESTART: C:/Python27/my\_module.py =====================

>>>

위의 프로그램을 저장할 때 my\_module.py라는 파일명을 주게 되면, ‘my\_module’이라는 모듈이 생성된다.

모듈을 주 프로그램에서 사용하고자 할 때는 import라는 명령으로 모듈명을 호출해주어야 한다.

파이썬은 import 명령을 수행하기 위해 주 프로그램과 같은 위치의 path나 파이썬 기본 위치의 path에서 모듈명과 같은 프로그램 파일을 찾게 된다. 만약, 모듈 프로그램을 찾지 못하게 되면 오류가 나게 된다.

실습9) 다음의 프로그램을 작성하여 모듈의 함수를 호출해 보세요. 오류가 나는 원인을 파악해 보세요.

앞에 모듈이름을 써줘야 하는대 써주지않아서 에러가 남

import my\_module

a = int(raw\_input("Enter first number : "))

b = int(raw\_input("Enter second number : "))

ret\_val = sumOfDouble(a, b)

print "Result Value : ", ret\_val

Traceback (most recent call last):

File "C:/Python27/minjiii.py", line 6, in <module>

ret\_val = sumOfDouble(a, b)

NameError: name 'sumOfDouble' is not defined

실습9) 실행 결과 오류가 나는 것을 확인할 수 있다. sumOfDouble() 함수가 정의되어 있지 않다는 메시지 내용을 볼 수 있다. raw\_input() 함수는 인식되는데 sumOfDouble() 함수는 인식되지 않고 있다.

앞에서 언급한 바대로 raw\_input() 함수는 파이썬의 내장함수이다. 따라서, 아무런 사전 조치 없이 바로 함수를 호출하여 사용할 수 있다.

반면에 sumOfDouble() 함수는 개발자가 임의로 필요한 기능을 모듈에 넣어 정의한 것이다. 파이썬이 동작할 때 이러한 사실을 잘 알도록 조치해 주지 않으면 안된다.

첫 번째 조치가 “import 모듈명”의 명령어를 사용하는 것이다.

두 번째 조치는 함수를 호출할 때 “모듈명.함수명”의 방식으로 명령어를 적는 것이다.

실습10) 다음과 같이 my\_module의 함수들을 호출해 보세요.

import my\_module

a = int(raw\_input("Enter first number : "))

b = int(raw\_input("Enter second number : "))

ret\_val = my\_module.sumOfDouble(a, b)

print "Result Value from sumOfDouble : ", ret\_val

ret\_val = my\_module.multipleOfDouble(a, b)

print "Result Value from multipleOfDouble: ", ret\_val

sumOfDouble function in my\_module

Result Value from sumOfDouble : 22

multipleOfDouble function in my\_module

Result Value from multipleOfDouble: 120

**6.8 표준 모듈 활용하기**

앞서 언급한 바와 같이 개발자가 자기에게 필요한 모든 함수나 모듈을 만들어 사용해야 하는 건 아니다. 파이썬에는 파일을 찾거나, 시간을 재거나, 수학적 계산을 하거나 하는 등 자주 사용될 만한 프로그램들이 표준 라이브러리(Python Standard Library)라는 것으로 이미 잘 만들어져 있기 때문이다.

파이썬은 표준 라이브러리를 제공함으로써 개발자들이 쉽고 빠르게 자기가 원하는 기능의 프로그램을 작성할 수 있도록 도와주는 것이다.

실습11) 표준 라이브러리에 속한 time 모듈을 사용한 프로그램을 다시 실행해 보세요.

import time

for i in range(10,0, -1):

print i

time.sleep(1)

print "Happy New Year~~!!! "

10

9

8

7

6

5

4

3

2

1

Happy New Year~~!!!

실습11)의 프로그램에서 time.sleep(1) 명령어를 sleep(1)로 적게 되면 오류가 발생할 것이다. 그런데, 매번 함수를 호출할 때마다 모듈명을 같이 표기하는 것이 귀찮기도 하고 프로그램의 가독성을 떨어뜨릴 수 있다.

실습12) 함수의 이름공간(namespace)을 미리 선언해주는 방법을 사용한 다음 프로그램을 실행해 보세요.

from time import sleep

for i in range(10,0, -1):

print i

sleep(1)

print "Happy New Year~~!!! "

10

9

8

7

6

5

4

3

2

1

Happy New Year~~!!!

첫 번째 줄에서 time이라는 모듈 안에서 sleep이라는 함수를 사용할 예정이라고 선언해 준다. 그러면, 파이썬은 time 모듈의 sleep 함수의 존재를 확인한 후, 이후에 모듈명 없이 사용된 함수명을 인식할 수 있게 된다.

실습13) 마지막으로 활용도가 높은 난수발생 함수를 확인해 보세요

>>>import random

>>>print random.randint(1, 100)

79