제7장
함수(function)-2

1. 참조값에 의한 인수 전달

1) 변수 전달

- 함수를 호출할 때, 변수를 전달하는 경우가 많다. 혹시 이런 의문을 가져보았는가? 내가 전달하는 변수의 무엇이 함수로 전달될까? 거의 모든 현대적인 프로그래밍 언어에서는 혼란을 막기 위하여 변수의 값만이 전달된다. 이것을 값에 의한 호출(call-by-value)라고 한다. 값에 의한 전달(pass-by-value)도 동일한 의미이다.

아래 간단한 코드를 보자.

```
def modify(n) :
        n=n+1

k = 10
print("k=", k}
modify(k)
print("k=", k)
```

```
k= 10
k= 10
```

- 위의 코드에서 변수 k가 modify() 함수로 전달되었지만 k의 값은 호출 후에도 변경되지 않았다. modify() 함수에서 매개변수 n의 값을 증가하였어도 말이다. 이것이 바로 값에 의한 호출(call-by-value)이다.

1. 참조값에 의한 인수 전달

2) 문자열 전달

- 문자열을 전달하여도 마찬가지이다.

```
def modifyl(s):
        s += "To You“

msg = "Happy Birthday"
print("msg=", msg)
modifyi(msg)
print("msg=", msg)
```

```
msg= Happy Birthday
msg= Happy Birthday
```

- 이것은 숫자나 문자열이 변경 불가능한 객체(immutable object)이기 때문이다. 숫자나 문자열을 변경하게 되면 새로운 객체가 생성된다.

1. 참조값에 의한 인수 전달

2) 문자열 전달

- 이전 슬라이드에서 살펴보았던 내용을 확인하기 위하여 참조값을 출력하는 id() 함수를 사용하여 문자열을 변경하면 문자열의 참조값이 어떻게 되는지를 확인하여 보자.

```
>>> msg = "Happy Birthday"
>>> id(msg)
50871576
>>> msg += "To You"
>>> id(msg)
50849984
```

- 기존의 문자열의 끝에 "To You"를 추가했을 뿐인데 객체의 주소는 완전히 변경되었다.

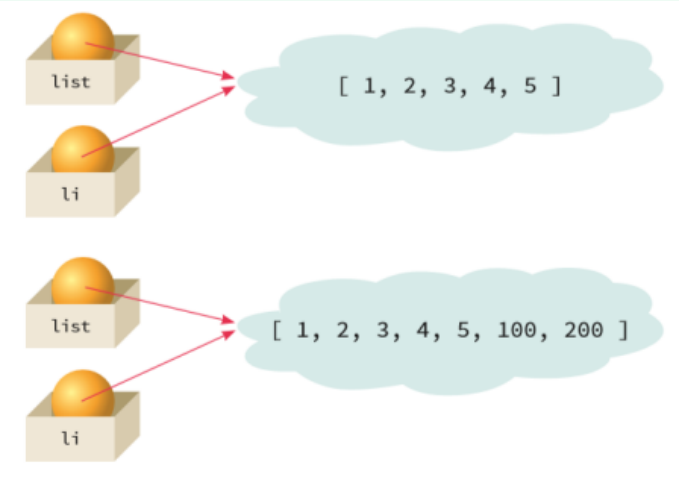
1. 참조값에 의한 인수 전달

3) 리스트 전달

- 리스트와 같은 변경 가능 객체(mutable object)를 전달하면 어떻게 될까? 그렇다면 아주 다른 상황이 된다.

```
def modify2(li):
        li += [100, 200]

list = [1, 2, 3, 4, 5]
print(list)
modify2(list)
print(list)
```

```
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 100, 200]
```

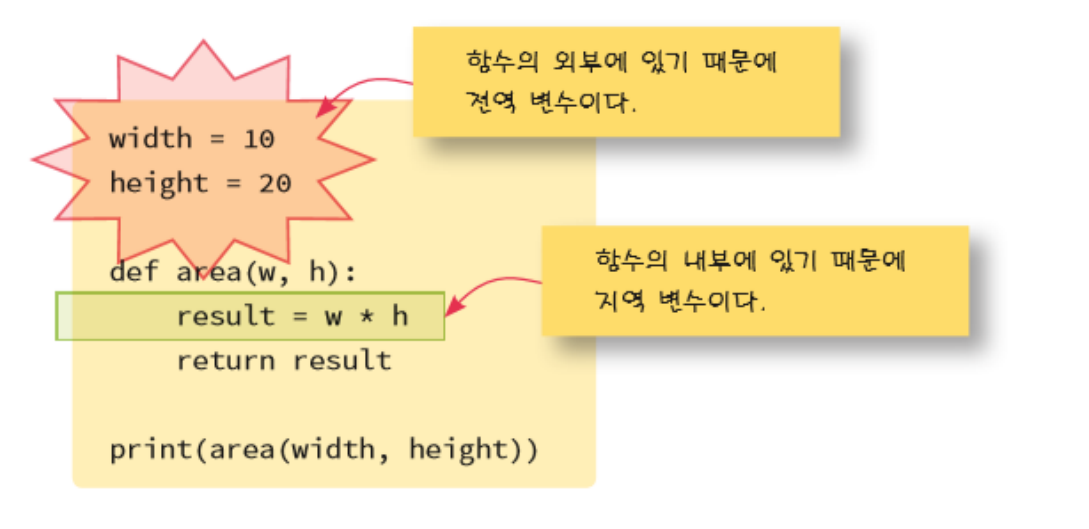


- 리스트는 변경 가능한 객체로서 modify2()에 리스트를 전달한 후에 리스트의 내용을 변경하면 원본에 영향을 끼친다. 이것은 다음과 같이 알아 두도록 한다. 리스트의 경우에 리스트의 참조값이 전달된다. 함수에서 참조값을 이용하여 리스트를 변경하면 리스트는 변경 기능하기 때문에 새로운 객체를 생성하지 않고 기존의 객체가 변경되는 것이다. 중요한 사항이니 반드시 기억해두기 바란다.

2. 지역 변수와 전역 변수

1) 지역 변수와 전역 변수

- 파이썬에서는 지역 변수와 전역 변수가 있다. 지역 변수는 함수 안에서 선언된 변수이고 전역 변수는 함수의 외부에서 선언된 변수이다. 주의할 점이 많으니 집중하도록 하자.



2) 지역 변수

- 함수 안에 정의된 변수는 지역 변수(local variable)라고 불리며 함수 안에서만 사용할 수 있다. 지역 변수는 함수가 호출될 때, 생성되고 함수가 종료되면 소멸되어서 더 이상 사용할 수 없다. 이것을 변수의 영역(scope)이라고 부른다.

2. 지역 변수와 전역 변수

2) 지역 변수

- 함수 sub를 작성하여 보자. 함수 sub() 안에서 지역 변수 s가 선언되었다.

```
def sub():
        s = "바나나가 좋음!"
        print(s)

sub()
```

```
바나나가 좋음!
```

- 변수 s는 sub() 함수 안에서만 사용할 수 있으며 함수 호출이 끝나면 사라진다. 지역 변수 s의 값을 함수 외부에서 출력할 수 있을까? 다음과 같이 print(s) 문장을 맨 끝에 추가하고 실행하여 보자.

```
def sub():
        s = "바나나가 좋음!"
        print(s)

sub()
print(s)
```

```
…
NameError: name 's' is not defined
```

오류가 발생하는 것을 알 수 있다. 지역 변수가 선언된 함수를 벗어나서 사용할 수는 없다. 지역 변수는 선언된 함수 안에서만 사용이 가능하다. 또 함수가 끝나면 자동으로 소멸된다.
상당히 중요한 개념이니 필히 기억하도록 한다.

2. 지역 변수와 전역 변수

3) 전역 변수

- 함수의 외부에 정의된 변수를 전역 변수(global variable)라고 한다. 파이썬이 전역 변수를 다루는 방식은 상당히 타언어에 비해서 좀 무식한 방식이다. 다른 언어와는 아주 다른 접근 방식을 취한다. 파이썬에서는 다른 이야기가 없으면 함수 안에서 선언된 변수들은 무조건 지역 변수이다. 하지만, 이러한 접근 방식은 좋은 프로그래밍 습관을 장려하기 위해서이다.

  일단 파이썬 함수 안에서 어떻게 전역 변수를 사용하는 지를 살펴보자.

```
def sub():
        print(s)

s = "사과가 좋음!"
sub()
```

```
사과가 좋음!
```

- 위의 코드에서 s는 함수의 외부에 정의된 전역 변수이다. sub()을 호출하기 전에 정의 되었다. 함수 sub()는 호출되기 전에는 실행되지 않는다. 함수 sub() 안에는 print() 호출 문장만 있다. 지역 변수를 선언하는 문장이 없으니 전역 변수 s가 사용된다.

2. 지역 변수와 전역 변수

3) 전역 변수

- 만약 함수 내부에서 전역 변수 s의 값을 변경하면 어떻게 될까? 전역 변수 s의 값이 변경될까? 아래와 같은 코드를 작성하여 실행해보자.

```
def sub():
        s = "바나나가 좋음!"
        print(s)

s = "사과가 좋음!"
sub()
print(s)
```

```
바나나가 좋음!
사과가 좋음!
```

- 다음의 말을 기억하도록 하자. “우리가 함수의 내부에서 변수 s에 값을 저장하면 파이썬은 우리가 지역 변수 s를 정의한 것으로 생각한다. 따라서 전역 변수 s가 아닌 것이다. 다시 한 번 파이썬의 원칙을 되새겨보자. 함수 안에서 변수에 어떤 값을 저장하면 무조건 지역 변수로 간주한다. 디폴트가 지역 변수인 것이다.

2. 지역 변수와 전역 변수

3) 전역 변수

- 아래와 같이 함수 안에서 변수 s의 값을 출력한 후에 변수 s의 값을 변경하면 어떻게 될까?

```
def sub():
        print(s)
        s = "바나나가 좋음!"
        print(s)
s = "사과가 좋음!!"
sub()
print(s)
```

```
UnboundLocalError: local variable 's' referenced before assignment
```

이때는 오류가 발생한다. 함수 안에서 하나의 변수가 전역 변수도 되었다가 지역 변수도 될 수는 없지 아니한가?

- 만약 함수 안에서 전역 변수의 값을 변경해야 되겠다면 global 키워드를 사용하여 전역 변수를 사용하겠다고 파이썬 인터프리터에게 알려주어야 한다.

```
def sub():
        global s    # 함수 안에서 전역 변수 s를 사용하겠다 라는 의미
        print(s)
        s = "바나나가 좋음!“    # 전역변수 값 변경
        print(s)
s = "사과가 좋음!!"
sub()
print(s)
```

```
출력 결과
사과가 좋음!
바나나가 좋음!
바나나가 좋음!
```

global 키워드를 사용하여서 전역 변수의 값을 함수 안에서 변경할 수 있었다. global을 사용하지 않고 파이썬의 함수 안에서 변수에 값을 저장하면 기본적으로 지역 변수가 된다. 이것을 잊지 말자. 또 변수는 선언 이후에만 사용할 수 있다. 파이썬에서는 변수에 값을 할당하는 순간 변수가 선언된다.

2. 지역 변수와 전역 변수

3) 전역 변수

- 지역 변수와 전역 변수를 섞어서 프로그램을 작성해보았다. 실행 결과를 추측할 수 있는가?

```
def sub(x, y):      # 함수의 매개 변수도 지역 변수의 일종이다.
        global a    #함수 안에서 전역 변수 a를 사용하겠다는 의미.
        a=7
        x, y = y, x
        b=3
        print(a, b, x, y)
a, b, x, y = 1, 2, 3, 4
sub(x, y)
print(a, b, x, y)
```

출력 결과

```
7343
7234
```

위의 결과가 왜 저렇게 나오는지에 대해서 반드시 이해할 필요가 있다. 지역변수, 전역변수가 어떻게 할당이 되고 변경이 되며 하는 것에 대하여 하나하나 따라가면서 이해할 필요가 있다는 것이다.

3. 매개변수 = 지역변수

1) 매개변수와 지역변수의 관계

- 함수가 외부로부터 값을 전달 받는데 사용하는 매개변수도 일종의 지역변수이다. 매개변수의 값을 함수 안에서 변경하면 어떻게 될까? 조금 착각할 수도 있을 것 같아서 아래 코드를 보고 확실하게 이해하도록 하자.

```
# 함수가 정의된다.
def sub(mylist):
   # 리스트가 함수로 전달된다.
   mylist = [1, 2, 3, 4]  # 새로운 리스트가 매개변수로 할당된다.
   print ("함수 내부에서의 mylist: ", mylist)
   return

# 여기서 sub() 함수를 호출한다.
mylist = [10, 20, 30, 40];
sub(mylist );
print ("함수 외부에서의 mylist: ", mylist)
```

```
함수 내부에서의 mylist: [1, 2, 3, 4]
함수 외부에서의 mylist: [10, 20, 30, 40]
```

- 함수의 매개변수와 외부의 전역변수의 이름이 모두 mylist이다. 매개변수도 지역변수의 일종임을 기억하자.
- pythontutor를 실행하여서 한 줄씩 실행하면서 메모리 내의 상황을 이해하기 바란다. 함수가 시작될 때는 전역변수 mylist와 매개변수 mylist는 동일한 리스트를 가리킨다. 하지만 함수 안에서 매개변수 mylist에 다른 리스트를 할당하면 전역변수와는 다른 리스트를 가리킨다.

4. 여러 개의 값 반환하기

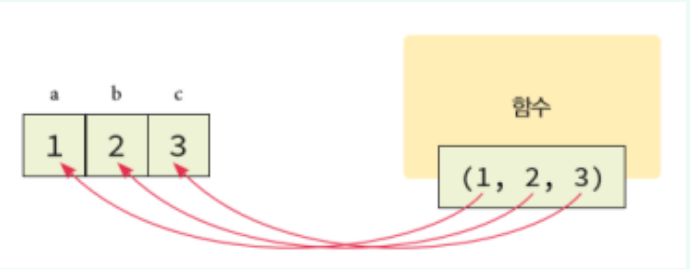
1) 여러 개의 값 반환하기

- 파이썬을 제외한 다른 프로그래밍 언어에서는 함수가 항상 하나의 값 만을 반환한다. 따라서 여러 개의 값을 반환하는 것이 필요한 함수는 다른 방법을 생각해야 했다. 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
```

- 위의 코드에서 보면 sub() 함수는 (1, 2, 3)을 반환한다. 즉 3개의 정수를 반환하고 있는 것이다. 반환된 3개의 정수 (1, 2, 3)는 문장 세 번째 줄에서 변수 a, b, c로 저장된다. 문장 네 번째 줄에서 변수 a, b, c의 값을 출력해보면 1 2 3이 출력되는 것을 확인할 수 있다.
- 파이썬에서는 어떻게 여러 개의 값을 동시에 반환할 수 있을까? return 1, 2, 3은 return (1, 2, 3)과 같다. 즉 1,2, 3은 튜플 (1, 2, 3)과 마찬가지이다. 즉 튜플 1개가 반환되고 이 튜플을 받아서 변수 a, b, c에 값을 풀어서 저장하는 것이다.

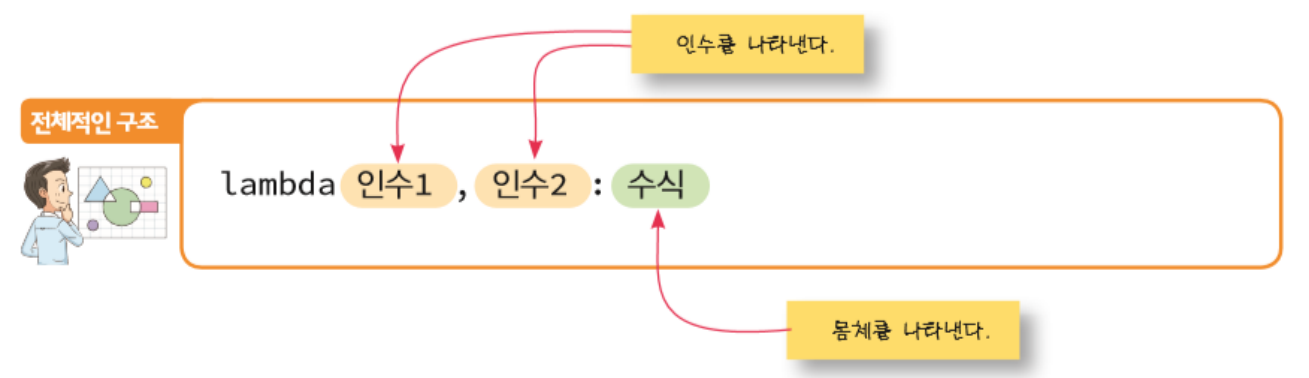


튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하며 리스트와 다른 점은 다음과 같다.(뒷 부분에서 자세하게 나온다)
- 리스트는 [ ]으로 둘러싸지만 튜플은 ( )으로 둘러싼다.
- 리스트는 그 값의 생성, 삭제, 수정이 가능하지만 튜플은 그 값을 바꿀 수 없다.

5. 무명 함수(람다식)

1) 무명 함수

- 무명 함수는 이름은 없고 몸체만 있는 함수이다. 파이썬에서 무명 함수는 lambda 키워드로 만들어진다. 무명 함수는 여러 개의 인수를 가질 수 있으나 반환값은 하나만 있어야 한다. 무명 함수 안에서는 print()를 호출할 수 없다. 계산만 가능하다. 또 자신만의 이름 공간을 가지고 있고 전역변수를 참조할 수 없다.



- 무명 함수를 이용하여 2개의 정수를 합하는 함수를 작성해보면 아래와 같다.

```
sum = lambda x, y: x + y;

print( "정수의 합  : ", sum( 10, 20 ))
print( "정수의 합  : ", sum( 20, 20 ))
```

```
정수의 합  : 30
정수의 합  : 40
```

[무명함수]

```
def get_sum(x, y) :
        return x + y
print( "정수의 합  : ", sum( 10, 20 ))
print( "정수의 합  : ", sum( 20, 20 ))
```

```
정수의 합  : 30
정수의 합  : 40
```

[일반함수]

5. 무명 함수(람다식)

2) 무명 함수의 용도

- 이전 슬라이드에서 확인할 수 있듯이, get_sum()과 sum()은 동일한 작업을 하며 동일한 방식으로 사용할 수 있다. 람다 함수에서는 return 키워드를 사용할 필요가 없다. 람다 함수에서는 항상 반환되는 수식만 써 주면 된다. 함수를 필요로 하는 곳에 람다 함수를 놓을 수 있으며 람다 함수를 반드시 변수에 할당할 필요도 없다.
- 그렇다면 람다 함수는 어디에 사용되는 것일까? 람다 함수는 코드 안에 함수를 포함하는 곳에서 어디든지 사용될 수 있다. 예를 들어서 GUI 프로그램에서 이벤트를 처리하는 콜백 함수(callback handler)에서 많이 사용된다. 콜백 함수를 간단하게 람다 함수로 작성하여서 포함시키는 것이다.

여기서는 함수 부분이므로 람다함수를 포함시킨 것이다. 하지만 후반부에 가면 람다에 대해서 나오니 람다함수의 개념과 사용 형태만 보고 넘어가도록 한다.

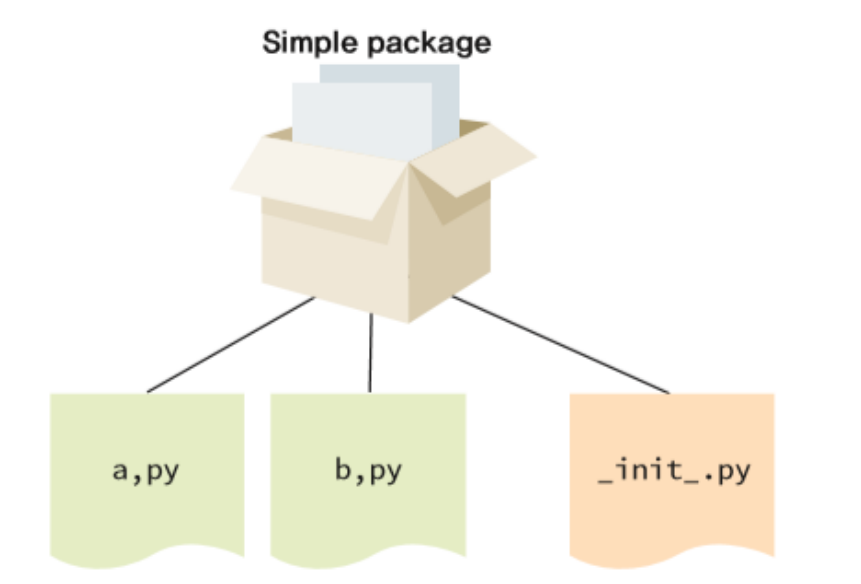
```
# 리스트안에 람다함수가 있는 형태
L = [  lambda x: x ** 2,
       lambda x: x ** 3,
       lambda x: x ** 4 ]

for f in L:
    print(f(3))
```

```
9
27
81
```

6. 모듈이란?

1) 모듈의 개념

- 우리가 파이썬 인터프리터를 종료하고 다시 들어가면, 이전에 정의해 놓았던 함수와 변수들의 정의는 전부 사라지게 된다. 따라서 우리가 어느 정도 이상의 복잡한 프로그램을 작성하려는 경우, 인터프리터를 사용하여서 한 줄씩 입력하여 실행하는 것보다 텍스트 편집기를 사용하여서 파일을 작성하고 인터프리터로 해당 파일을 실행하는 것이 더 낫다. 또 프로그램이 길어지면, 유지 보수를 쉽게 하기 위해 여러 개의 파일로 분할할 수 있다. 또한 파일을 사용하면 한번 작성한 편리한 함수를 복사하지 않고 여러 프로그램에서 사용할 수 있다.



이것을 지원하기 위해, 파이썬에서는 파일에 함수들을 저장하고 인터프리터에서 사용하는 방법을 제공한다. 이러한 함수나 변수들을 모아 놓은 파일을 모듈(module)이라고 한다. 모듈 안에 있는 함수들은 import 문장으로 다른 모듈로 포함될 수 있다. 모듈 중에서 main 모듈은 최상위 수준에서 실행되는 스크립트를 의미한다.

6. 모듈이란?

2) 모듈의 활용

- 파일 이름은 파이썬 모듈 이름에 .py 확장자를 붙이면 된다. 모듈 안에서는 모듈의 이름은 __name__의 값(문자열)으로 접근이 가능하다. 예를 들어 fibo.py 파일에 텍스트 편집기를 사용하여 다음과 같은 내용을 저장하였다고 하자.

fibo.py

```
# 피보나치 수열 모듈

def fib(n):    # 피보나치 수열을 화면에 출력한다.
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()
```

[fibo 모듈]

```
import fibo

fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

print(fibo.__name__)
'fibo'
```

__name__은 인터프리터가 실행 전에 만들어 둔 내장 전역변수에 해당한다

[모듈 사용법 1]

```
from fibo import *
fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

[모듈 사용법 2]

7. 함수를 사용한 프로그램 설계

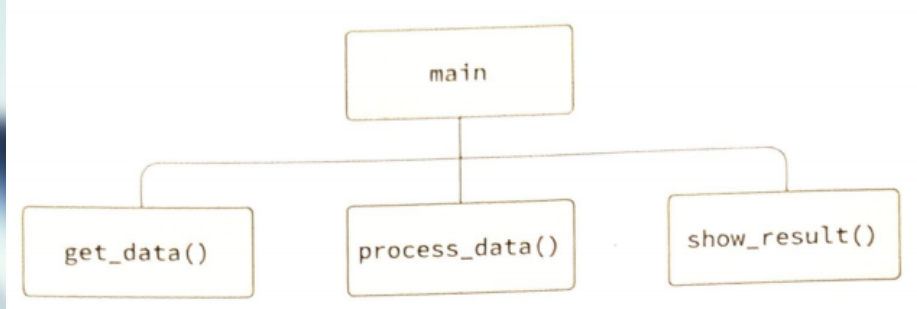
1) 함수를 사용한 프로그램 설계

- 한 대의 자동차를 만들기 위해서는 수백 개의 협력업체에서 부품을 공급해야 한다. 이들 부품을 자동차 공장에서 조립하면 자동차가 생산된다. 이와 같은 원리를 프로그램에 대해서도 적용할 수 있다. 지금까지는 간단한 프로그램이었기 때문에 하나의 함수로도 충분하였다. 하지만 윈도우나 한글과 같은 커다란 프로그램의 모든 코드가 하나의 함수 안에 들어 있다고 가정해보자. 흔히 대형 프로그램의 코드는 줄이 수만 라인이 넘는다. 이것이 하나의 함수 안에 들어 있다면 코드를 작성한 사람도 시간이 지나면 이해하거나 디버깅하기가 어려울 것이다.
- 그렇다면 어떻게 하여야 하는가? 정답은 작은 조각으로 분리하는 것이다. 파이썬에서는 작은 조각이 함수에 해당한다. 복잡하고 규모가 큰 프로그램은 여러 개의 함수로 나누어서 작성되어야 한다. 먼저 주어진 문제를 분석한 후에, 보다 단순하고 이해하기 쉬운 문제들로 나누게 된다. 문제가 충분히 작게 나누어지면 각 문제를 해결하는 절차를 함수로 작성한다.
- 문제를 한 번에 해결하려고 하지 말고 더 작은 크기의 문제들로 분해한다. 문제가 충분히 작아질 때까지 계속해서 분해한다.
- 문제가 충분히 작아졌으면 각각의 문제를 함수로 작성한다. 이들 함수들을 조립하면 최종 프로그램이 완성된다.

7. 함수를 사용한 프로그램 설계

2) 함수를 사용한 프로그램 설계 예제

- 파이썬 프로그램에서도 main() 함수를 작성하고 스크립트의 맨 아래에서 main()을 호출하도록 하는 것이 바람직하다고 생각하는 개발자들도 많다.



- C언어 또는 자바와 같은 프로그래밍 언어에서는 항상 main() 이라는 함수를 시작으로 프로그램을 실행시킨다. 하지만 파이썬은 main함수가 존재하지 않는다. 그렇다면 어떤 방식으로 코드를 실행시키는 것일까?

파이썬은 크게 두 가지 특징이 있다.

1. 들여쓰기를 통해 코드 실행의 레벨을 결정한다.
2. main이 존재하지 않는다.

- __name__는 현재 모듈의 이름을 담고 있는 내장 전역변수이다.이 변수는 직접 실행된 모듈의 경우 __main__이라
는 값을 가지게 되며 직접 실행되지 않은 import된 모듈은 모듈의 이름(파일명)을 가지게 된다. 결론을 짓자면 모듈에 if __name__=="__main__"이라는 조건문을 넣어주고 그 아래는 직접 실행시켰을 때만 실행되길 원하는 코드

7. 함수를 사용한 프로그램 설계

2) 함수를 사용한 프로그램 설계 예제

- 예제로 성적을 사용자로부터 읽어서 크기순으로 정렬하여서 화면에 출력하는 프로그램을 작성한다고 생각하여 보자. 사용자가 음수를 입력하면 입력을 종료한다.

```
def readList():
        nlist = []
        flag = True;
        while flag :
                number = int(input("숫자를 입력하시오: "))
                if number < 0:
                        flag = False
                else :
                        nlist.append(number)
        return nlist

def processList(nlist):
        nlist.sort()
        return nlist


def printList(nlist):
        for i in nlist:
                print("성적=", i)
```

```
def main():
        nlist = readList()
        processList(nlist)
        printList(nlist)

if __name__ == "__main__":     # 프로그램 시작점
        main()
```

```
숫자를 입력하시오: 30
숫자를 입력하시오: 50
숫자를 입력하시오: 10
숫자를 입력하시오: 90
숫자를 입력하시오: 60
숫자를 입력하시오: -1
성적= 10
성적= 30
성적= 50
성적= 60
성적= 90
```

함수들은 특징적인 한 가지 작업(기능)만을 맡아야 한다. 하나의 함수가 여러 가지 작업을 하면 안 된다. 다른 것과 구별되는 한가지의 작업만을 하여야 한다. 만약 함수 안에서 여러 작업들이 섞여 있다면 각각을 다른 함수들로 분리하여야 한다. 이런 식으로 함수를 사용하게 되면 함수들을 작업별로 분류할 수 있어서 소스 코드의 가독성이 높아진다.

감사합니다.