기초 인공지능 프로그래밍

8장 함수, 모듈

함수(function)

- 함수(function)는 특정 작업을 수행하는 명령어들의 모임에 이름을 붙인 것
- 함수는 작업에 필요한 데이터를 전달받을 수 있으며, 작업이 완료된 후에는 작업의 결과를 호출자에게 반환할 수 있음
- 자주 사용되는 부분 문제를 함수로 작성하면 코드를 반복 작성할 필요가 없기 때문에 편리하며, 호출하여 재사용하면 됨
- 파이썬에서 지원하는 세 종류의 함수
  - 내장(Built-in) 함수 : 파이썬에서 제공하는 함수
    : 파이썬 설치 후 사용 가능(print(), input(), len(), type() .....)
  - 라이브러리 패키지 : 파이썬에서 제공하는 모듈(import 문)
    : 해당 모듈을 프로그램에 포함한(loading)후에 사용할 수 있음
  - 사용자 정의(User-defined) 함수
    : 사용자가 자신의 필요에 따라 특정 기능의 함수를 직접 작성

사용자 정의 함수

- 함수 작성 (함수 정의)

```
def function_name(parameters) :
    """docstring"""      # option(없어도 됨)
    statements           # 함수 기능에 필요한 명령어들
    return ret_values    # 반환할 것이 없으면 생략 가능
```

들여쓰기

- def : 함수의 시작을 알림
- 함수 이름(function_name) : identifier(변수) 규칙대로 이름 정의
- parameters(매개변수)
  - ① 함수 실행에 필요한 데이터를 받는 변수(여러 개인 경우 콤마로 구분). 필요하지 않으면 괄호만 표시
  - ② 함수 호출시 전달하는 데이터는 함수의 매개변수로 전달됨
  - ③ 함수 호출시 전달하는 데이터를 인자(arguments)라 함
- """docstring""" : 주석 (함수 설명), 생략 가능
- return : 실행 결과를 호출한 코드로 반환, 반환 값이 없으면 생략

함수 호출하기

- 프로그램에서 함수를 사용하려면 함수를 호출(call)하여야 함
- 함수 호출시 데이터(값,정보)를 지정하여 함수에 전달할 수 있으며, 이 데이터를 인자(argument)라고 함 (위치에 따른 인자 전달 방식)
- 인수와 매개변수는 함수 호출 시에 데이터를 주고받기 위하여 필요하며, 함수가 호출될 때마다 인수는 함수의 매개변수로 전달됨
  - 인수(argument) : 호출 프로그램에 의하여 함수에 전달되는 값
  - 매개 변수(parameter) : 함수에서 인수를 전달받는 변수
- 한번 정의된 함수는 필요할 때마다 반복해서 호출할 수 있음
- 함수가 호출되는 시점에서 실행의 흐름은 함수로 넘어가며, 함수 종료 후에는 호출된 곳으로 돌아옴
- 파이썬 script 코드는 들여쓰기 하지 않은 첫 명령어(함수 정의부분 제외하고)부터 실행되며 이를 main 함수의 시작으로 간주함

함수 호출하기

함수 정의

```
def isEven ( N ) :      # N은 매개변수
    """ (docstring)
     N 값이 짝수이면 True 반환, 아니면 False 반환
    """
    if N % 2 == 0 :
        return True     # 값을 반환하며 함수 종료
    else :
        return False    # return 문은 필요시 여러 번 사용 가능

print( isEven(10) ) # True
M = 11
rtn = isEven(M)
print( rtn ) # False
```

메인 함수
(isEven() 함수 두 번 호출)

① 함수는 호출되기 전에, 먼저 함수 정의가 되어 있어야 함
② 함수는 필요시 인수 값을 바꿔서 여러 번 호출 할 수 있음
③ 함수는 반환 값이 있는 경우, return 문을 사용해서 그 값을 반환해야 함

함수 호출하기

- 입력 값과 반환 값이 모두 없는 경우의 함수 호출 형식

  func_name()

  : 함수명 옆에 빈 괄호만 두며, 반환 값도 없기 때문에 함수 실행 후에 값을 반환 받을 변수에 할당할 필요가 없음

```
def  NoRtnNoArg():  # 함수 정의
     print("Sogang University")

NoRtnNoArg()   # Sogang University 출력
a = NoRtnNoArg() # 잘못된 사용
print(a)       # None 출력
```

반환값이 없는 경우는 return 구문을 생략해도 됨(또는 return 이라고만 명시). 반환값이 없는 함수의 실행 결과(반환값)를 변수에 할당하여 출력해보면 None 이라고 표시됨. 할당하지 않고 호출만 하여야 함

- 입력 값은 없고 반환 값만 있는 경우의 함수 호출 형식

  var_name = func_name()

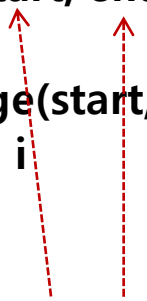
  : 함수가 실행 후에 값을 반환하므로 이를 코드에서 사용하기 위해서는 변수에 할당하여야 함

```
def  NoArg():   # 함수 정의
     return "Sogang University" # 문자열 Sogang University를 반환

a = NoArg()   # 변수 a는 반환 받은 문자열 "Sogang University" 를 저장
print(a)      # Sogang University 출력
```

함수 호출하기

- 입력 값과 반환 값이 모두 있는 경우의 함수 호출 형식
  - 인자 전달방법 1 : 위치에 따라 인자가 매개변수로 전달

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

sum1 = get_sum(1, 10)
sum2 = get_sum(20, 30)
print(sum1, sum2) # 55 275
```



```
def nPrint(message, n) :
    for i in range(0, n) :
        print(message)

nPrint("Hello", 3)  #정상적인 호출
nPrint(2, "Hello")  #오류 발생
#TypeError: 'str' object cannot be interpreted as an integer
```

함수 호출하기

- 입력 값과 반환 값이 모두 있는 경우의 함수 호출
  - 인자 전달방법 2 : 인자의 이름을 명시적으로 지정해서 전달

```
def nPrint(message, n) :
    for i in range(0, n) :
        print(message)

nPrint("Hello", 3)  #정상적인 호출
print()             #줄바꿈
nPrint(n = 2, message = "Hello")      #정상적인 호출
```

출력

```
Hello
Hello
Hello

Hello
Hello
```

함수 호출하기

- 함수의 반환값
  - return 키워드를 사용하여 값(반환값)을 호출자에게 반환
  - 함수 정의 안에 return 명령어가 없거나, 또는 return 예약어만 있는 경우는 반환값이 없는 함수이며 None을 기본적으로 반환

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

c_area = calculate_area(5.0)
print(c_area)          #78.5
```

```
def sum(number1, number2):
     total = number1 + number2

print(sum(1, 2))      # None : sum() 함수는 반환값이 없기 때문
t = sum(10, 20)
print(t)              # None : sum() 함수는 반환값이 없기 때문
```

함수 호출하기

- 함수에서 return 문이 여러 번 나오는 경우
  - return 문이 여러 번 나오더라도 먼저 실행되는 return 문에서 함수는 값을 반환하며 종료

```
def get_max(a,b) :
    if a > b :
        return  a   # a > b 경우이므로 a 값을 반환하면서 종료
    else:
        return  b   # a <= b 경우이므로 b 값을 반환하면서 종료

max = get_max(10, 20)
print(max)     # 20
```

함수 호출하기

- 반환값이 여러 개인 경우
  - 반환값이 2개 이상인 경우 튜플로 묶어서 반환

```
def add_multiply(x,y):
    sum = x + y
    mul = x * y
    return sum, mul        # 반환값 2개를 튜플로 반환

a = int(input('Enter a : '))
b = int(input('Enter b : '))
m, n = add_multiply(a,b) # 변수 m은 a+b의 값, 변수 n은 a*b의 값을 할당 받음
print(m,n)
rt = add_multiply(10, 20)
print(rt)
print(rt[0], rt[1])
```

출력

```
Enter a : 10
Enter b : -10
0 -100
(30, 200)
30 200
```

지역(local) 변수와 전역 변수(global)

- 스코프(scope)는 변수가 참조될 수 있는 프로그램의 영역을 일컫는 용어임
- 파이썬에서는 변수에 처음 값을 할당할 때 변수가 생성됨
- 스코프를 기준으로 변수를 다음과 같이 구분함
- 전역 변수(global variable)
  - 모든 함수의 외부에서 생성되며, 모든 함수에서 접근 가능
  - 즉, 프로그램 전체에서 사용 가능
  - 전역변수의 값을 함수 안에서 수정하면 같은 이름의 새로운 지역변수가 생성됨
- 지역 변수(local variable)
  - 함수 내에서 생성된 변수 및 매개변수는 지역 변수
  - 생성된 함수 내에서만 사용 가능
  - 함수 종료 후에는 소멸
- 프로그램에서 변수를 참조할 때 찾는 순서는 지역 변수→전역 변수 순서로 찾음

지역(local) 변수와 전역 변수(global)

- 전역변수의 값을 함수 안에서 수정하면 같은 이름의 지역변수로 새로 생성됨

지역 변수는 해당 함수가 종료될 때 소멸됨

```
def classify():
    globalV = "only"      # 함수 안에서 생성된 전역 변수와 같은 이름의 지역 변수 생성
    localV = "local"      # 지역 변수 생성
    print(globalV)        # only 출력
    print(localV)         # local 출력

globalV = "everything" # 전역 변수 생성
classify()
print(globalV)            # everything 출력
print(localV)             # NameError: name 'localV' is not defined
```

지역변수 globalV는 함수 종료시 소멸 되었기 때문에 전역변수 globalV의 값 출력

scope을 벗어난 변수의 접근은 오류를 일으킴 (변수 localV는 함수 classify() 내에서만 유효)

지역(local) 변수와 전역 변수(global)

- 전역 변수를 함수 안에서도 수정하면서 전역 변수로 사용하려면 global 예약어로 선언해야 함

```
def classify():                    전역 변수로 사용한다는 선언
    global globalV
    globalV = "only"   # 함수 안에서 전역 변수값 수정
    print(globalV)     # only 출력


globalV = "everything" # 전역 변수 생성
print(globalV)         # everything 출력
classify()
print(globalV)         # 함수에서 수정한 값 only 출력
```

지역(local) 변수와 전역 변수(global)

- 지역 변수 값은 return을 통하여 함수 밖에서 그 값은 사용이 가능

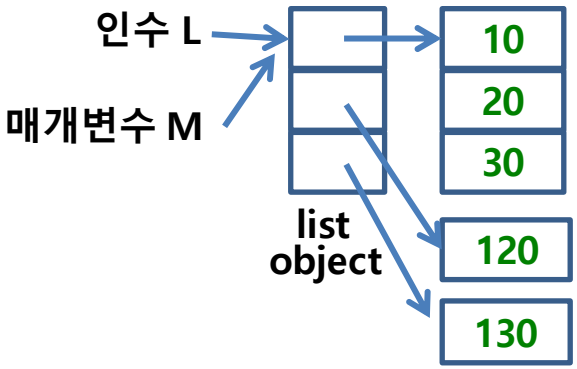
```
def swap( a, b ) : # 지역 변수 a, b는 전역 변수 a, b 의 값을 순서대로 전달 받음
    a, b = b, a      # 지역 변수 a, b 의 값을 교환
    return a, b      # 변경된 a, b 값을 반환

a = 10; b = 20       # 전역 변수 a, b 생성
a, b = swap( a, b ) # 함수가 반환한 값, 두개를 전역 변수 a, b에 차례로 할당
print(a, b)          # 20 10 출력
```

- 함수의 매개 변수는 해당 함수의 지역 변수임
- 전역 변수와 함수의 매개 변수는 변수명이 같아도 다른 변수임
- 함수의 매개 변수를 선언할 때 전역 변수와 같은 이름으로 할 필요는 없음
- 전역 변수를 인자로 함수를 호출할 때 readability를 위해 매개 변수를 같은 이름으로 하는 것임

mutable 객체의 인자 전달(참조 값에 의한 인자 전달)

- 파이썬에서 모든 데이터는 객체이며, 변수는 그 객체에 대한 참조(reference)임
- 리스트, 집합, 사전 등을 인자로 해서 함수를 호출하면, 인자인 객체의 참조 값이 매개변수로 전달
- 즉, mutable객체인 리스트, 집합, 사전 등이 인자로 지정된 경우는 매개변수와 인자가 동일한 객체를 가리키게 됨
- 매개변수는 지역 변수이기 때문에 함수 내에서 변수 값 수정을 해도 함수 밖에는 영향을 미치지 못함(함수 종료시 소멸되기 때문)
- 하지만, 리스트와 같은 mutable한 객체는 함수 안에서 값을 수정하면 함수 밖에서도 변경된 내용이 반영됨

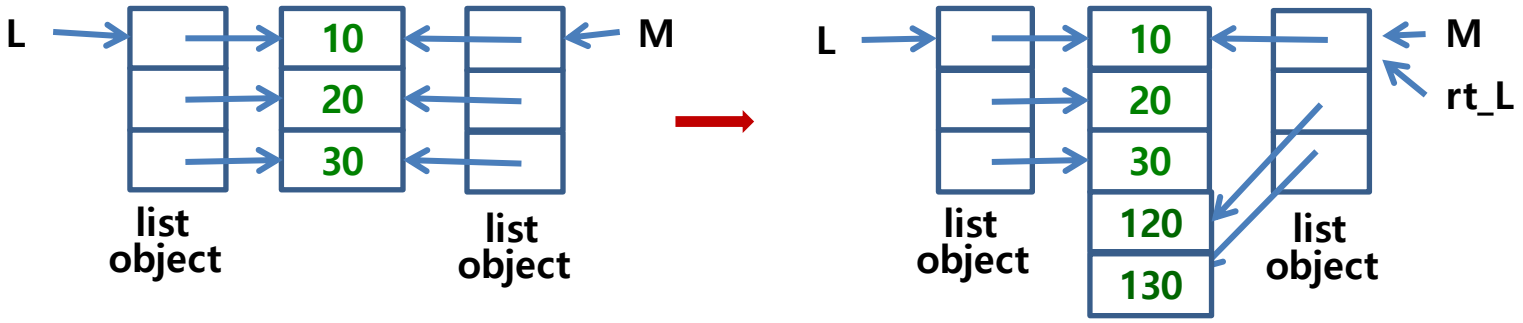
```
def  test(M) :
       M[1] += 100
       M[2] += 100

L = [10,20,30]
test(L)
print(L)    #[10, 120, 130]
```



mutable 객체의 인자 전달

- 리스트, 집합, 사전의 복사본을 매개변수로 전달하면 함수 안에서 값을 수정해도 원본은 그대로 유지할 수 있음

```
def  test(M) :
       M[1] += 100
       M[2] += 100
       return M

L = [10,20,30]
rt_L = test( L[:] )    # 리스트 L의 복사본(다른 객체)을 인자로 전달
print(L)               # [10, 20, 30]
print(rt_L)            # [10, 120, 130]
```



파이썬 모듈(라이브러리)

- 모듈(module)의 정의
  - 함수, 변수들을 정의해 둔 파일
  - 다른 파이썬 프로그램에서 호출해서 사용할 수 있게 만들어진 파이썬 프로그램 파일
  - 각각의 소스 파일을 일컬어 모듈이라 함
- 내장(built-in) 라이브러리는 인터프리터에 기본적으로 탑재되어 있어 import 문으로 load 하지 않아도 됨 (print(), input()….)
- 내장되어 있지 않은 표준 모듈(라이브러리)
  - 파이썬이 개발에 필요한 기능들을 모듈 형태로 제공
  - 파이썬 설치시에 함께 설치됨
  - 하지만, import 문으로 load 해서 사용하야 함
- 사용자 생성 모듈
  - 프로그래머가 작성한 모듈

파이썬 모듈

- 모듈(module) 탐색 순서
  - import 문으로 모듈명 지정하면 찾는 순서
    1. 현재 작업 폴더
    2. 파이썬 환경 변수에 정의되어 있는 디렉토리 (라이브러리가 저장된 디렉토리)
- import 문 형식

```
from math import *          # 이 경우 함수 사용시, 모듈 이름이 불필요
a = sqrt(4.0)               # sqrt() 함수를 함수명으로만 호출
print(a)                    # 2.0 출력
```

```
import math                 # 이 경우 math.을 붙여야 함
a = math.trunc(1.5)         # trunc() 함수 앞에 해당 모듈명을 명시해야 함
print(a)                    # 1 출력
```

```
import math as m            # 이 경우 m.을 붙여야 함
a = m.pow(81, 0.5)          # m은 math의 별칭에 해당
print(a)                    # 9.0 출력
```

sqrt(), trunc(), pow() 함수는 math 모듈에 정의되어 있음

파이썬 모듈

random() 함수 # 0.0 이상 1.0 미만의 임의의 실수 반환
randint(a, b) 함수 # a부터 b사이의 임의의 정수 반환

- 파이썬 표준 모듈 : random 모듈

```
import random

print( random.randint(1, 6) )  # 1~ 6 사이의 임의의 정수를 반환
print( random.randint(1, 6) )
```

출력

```
5
2
blue
```

- 파이썬 표준 모듈 : calendar 모듈

```
import   calendar

cal = calendar.month(2020, 12) # 인수받은 달력 반환
print(cal)
print(type(cal))
```

출력

```
 December 2020
Mo Tu We Th Fr Sa Su
    1  2  3  4  5  6
 7  8  9 10 11 12 13
14 15 16 17 18 19 20
21 22 23 24 25 26 27
28 29 30 31

<class 'str'>
```

사용자 정의 모듈

- 다음 프로그램을 사용자 정의 모듈로 구성

main.py 파일

```
def add_multiply(x,y):
    sum = x + y
    mul = x * y
    return sum, mul      # 반환값 2개를 튜플로 반환

a = int(input('Enter a : '))
b = int(input('Enter b : '))
m, n = add_multiply(a,b) # 변수 m은 a+b의 값, 변수 n은 a*b의 값을 할당 받음
print(m,n)
```

- 먼저, main.py 파일을 실행해 본 후 다음의 작업을 함
- userF.py 파일 코딩
  - add_multiply() 함수 정의 부분만 저장하는 모듈 파일
- test.py 파일 코딩
  - 모듈 userF를 import 한 후 add_multiply() 함수를 호출하는 파이썬 프로그램 파일

사용자 정의 모듈

test.py 파일

```
from userF import *

a = int(input('Enter a : '))
b = int(input('Enter b : '))
m, n = add_multiply(a,b)      # 변수 m은 a+b의 값, 변수 n은 a*b의 값을 할당 받음
print(m,n)
```

userF.py 파일

```
def add_multiply(x,y):
    sum = x + y
    mul = x * y
    return sum, mul           # 반환값 2개를 튜플로 반환
```

- 위와 같이 두 개의 파일을 코딩한 후, test.py 파일을 실행하면 main.py 파일을 실행 한 것과 같은 결과를 얻음

사용자 정의 모듈

- calculator 모듈 만들기
  - calculator.py 모듈 파일에 구현하고자 하는 함수들을 정의

calculator.py

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

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

def  multiply(a, b):
       return a*b;

def  divide(a, b):
       return a/b
```

test1.py

```
import  calculator

print( calculator.plus(10, 5) )
print( calculator.minus(10, 5) )
print( calculator.multiply(10, 5) )
print( calculator.divide(10, 5) )
```

test2.py

```
from calculator import *

print( plus(10, 5) )
print( minus(10, 5) )
print( multiply(10, 5) )
print( divide(10, 5) )
```