함수(Functions)

- ★ 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
  - ① 내장(Built-in) 함수 : 파이썬에서 제공하는 함수
    : 파이썬 설치 후 사용 가능(print(), input(), len(), type() .....)
  - ② 라이브러리 패키지 : 파이썬에서 제공하는 모듈(import 문)
    : 해당 모듈을 프로그램에 포함한 후에 사용할 수 있음
  - ③ 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"""      # optional(없어도 됨)  statemen
    ts                   # 함수 기능에 필요한 statements
    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 함수의 입력 값(콤마로 분리), 필요하지 않으면 괄호만 표시
  - ② 함수 호출시 전달하는 데이터는 함수의 매개변수로 전달됨
  - ③ 함수 호출시 전달하는 데이터를 인수(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 인수(argument)라고 함 (위치에 따른 인수 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 필요시 반복해서 호출할 수 있음
- 함수가 호출되는 시점에서 실행의 흐름은 함수로 넘어가며, 함수 종료 후에는 호출된 곳으로 돌아옴
- 파이썬 스크립트 코드에서는 들여쓰기 하지 않은 첫 명령어부터 실행되며 이를 main 함수로 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

func_name()

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

```
def  say():          # 함수 정의
     print('Hi')

say()  print         # Hi 출력
(say())              # None 출력
```

반환값이 없는 경우는 return 구문을 생략해도 됨(또는 return 이라고만 명시). 이런 경우 반환값을 할당한 변수나, 함수 실행 결과를 출력하면 None 이라고 표시.

★ 입력 값은 없고 반환 값만 있는 경우의 함수 호출

var_name = func_name()

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

```
def  say():          # 함수 정의
     return('Hi')

a = say()            # 함수 호출
print(a)             # Hi
```

함수 호출하기

★ 입력 값과 반환 값이 모두 있는 경우의 함수 호출

- 인수 전달방법 1 : 위치에 따라 인수가 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

- 인수 전달방법 2 : 인수의 이름을 명시적으로 지정해서 전달 (키워드 인수)

```
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):  to
     tal = 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  ret
    urn 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
```

함수 정의 및 호출 예제

★ Palindrome 판별(실습 예제) – 함수로 구현하여 동작 비교

```
n=int(input("Enter a number to be checked:"))

origin=n
reverse=0

while n>0 :
    tail=n%10
    reverse=reverse*10+tail
    n=n//10

if origin==reverse :
    print(origin, "is a palindrome!")
else:
    print(origin, "is NOT a palindrome")
```

```
def palindrome(number):  #판별 함수로 정의
    origin=number
    reverse=0

    while number>0 :        #숫자 역순으로 배열
        tail=number%10
        reverse=reverse*10+tail
        number=number//10

    if origin==reverse:     #역순이 동일한지 아닌지
        return True         #비교하는 부분
    else:
        return False        #두번째 return

number = int(input("Enter a number to be checked:"))

if palindrome(number):      #palindrome 판별 함수 호출
    print(number, "is a palindrome!")
else:
    print(number, "is NOT a palindrome")
```

출력(동일한 결과 출력)

```
Enter a number to be checked:121
121 is a palindrome!
```

```
Enter a number to be checked:1234
1234 is NOT a palindrome
```

실습

지역(local) 변수와 전역(global) 변수

- 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

지역변수와 전역변수

★ 전역변수의 값을 함수 안에서 수정하면 같은 이름의 지역변수로 새로 생성됨

같은 이름의 지역 변수가 생성,
해당 함수가 종료될 때 소멸됨

```
def classify_var():
    globalS = "I like only this!“    # 함수 안에서 전역 변수의 값을 수정하면
                                      # 새로운 지역 변수가 됨
    localS = "It's local variable!“  # 지역 변수 생성
    print(globalS)          # I like only this!  지역 변수 globalS의 값 출력
    print(localS)           # It's local variable!

globalS = "I like all everything!"      # 전역 변수 생성
classify_var()
print(globalS)    # I like all everything!  전역변수 globalS의 값을 출력
print(localS)     # NameError: name 'localS' is not defined
```

스코프(Scope, 영역)는 이름이 의미를 가지는 범위를 말하며, scope을 벗어난 변수의 접근은 오류를 일으킴(변수 localS는 함수 classify_var() 내에서만 유효).

지역변수와 전역변수

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

```
def classify_var():
    global globalS

    globalS = "I like only this!"       # 함수 안에서 전역 변수의 값을 수정
    print(globalS)      # I like only this!  수정된 전역 변수 globalS의 값 출력

globalS = "I like all everything!"      # 전역 변수 생성
print(globalS)    # I like all everything!  전역변수 globalS의 값을 출력
classify_var()
print(globalS)    # I like only this!  함수에서 수정된 전역 변수 globalS 값 출력
```

전역 변수로 사용한다는 선언

지역변수와 전역변수

★ 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 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
```

지역변수와 전역변수

```
def test():
    b = 20      # b 지역변수
    print(a,b)  # 100 20 : a 전역, b 지역변수

a = 100  # a 전역변수
print(a)  # 100
test()
print(a)  # 100
print(b)  # NameError!!!
```

```
def test():
    a = 20   # a 지역변수
    print(a)  # 20 : a 지역변수

a = 100  # a 전역변수
print(a)  # 100
test()
print(a)  # 100 : 전역 변수 a 값
```

```
def test():
    print(a)        # UnboundLocalError: local variable 'a' referenced
                    # before assignment
    a = 20
    print(a)

a = 100   # a 전역변수
print(a)  # 100
test()
print(a)
```

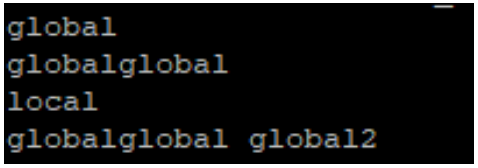
지역변수/전역변수 예제

```
x= "global“                    #변수 x 선언 – “global”
y= “global2”                   #변수 y 선언 – “global2”

def foo():
    global x                   #변수 x 전역변수로 사용 선언
    y= " local“                #변수 y 선언(지역 변수)
    x = x*2                    #x=“globalglobal”
    print(x)                   #globalglobal 출력
    print(y)                   #local 출력

print(x)                       #global 출력
foo()                          #foo()호출 ->전역변수로 사용되는 x의 값이 “globalglobal”로 변경
print(x, y)                    #x=“globalglobal” y=“global2” ->foo()함수에서 y는 지역변수로
                               #전역변수 y의 값은 바뀌지 않는다.
```

출력



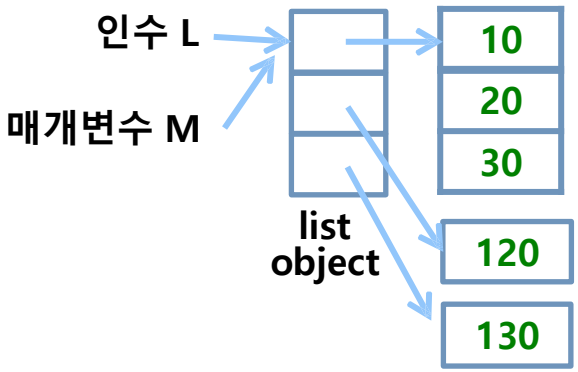
실습

mutable 객체의 인수 전달(참조 값에 의한 인수 전달)

- 파이썬에서 모든 데이터는 객체이며, 변수는 그 객체에 대한 참조(reference)임
- 리스트, 집합, 사전 등을 인수로 해서 함수를 호출하면, 인수인 객체의 참조 값이 매개변수로 전달
- 즉, mutable객체인 리스트, 집합, 사전 등이 인수로 지정된 경우는 매개변수와 인수가 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]
```

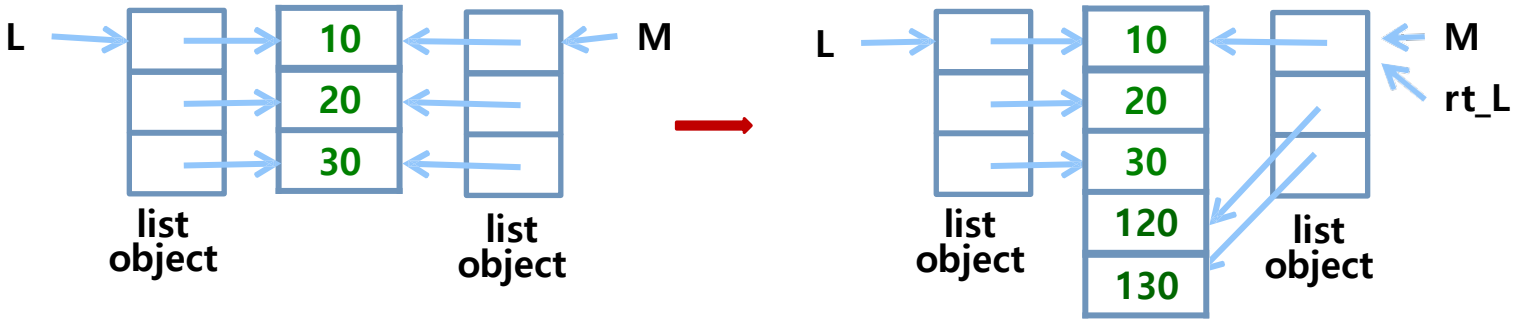


mutable 객체의 인수 전달(참조 값에 의한 인수 전달)

★ 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 인수
print(L)            #[10, 20, 30]
print(rt_L)         #[10, 120, 130]
```



mutable 객체 전달(list) 예제

```
def buy(book, n):                                   #buy 함수 정의
    new_book= " new_book " +n
    book.append(new_book)                           #인수로 받아온 리스트 book에 new_book 추가
    print(n,  " buy() :  " , book)                  #현재 리스트 book 출력

def booklist(book, n):                              #booklist 함수 출력
    new_book="new_book"+n
    book.append(new_book)                           #인수로 받아온 리스트 book에 new_book 추가
    n=int(n)+1                                      #n값에 1 추가 하여 mutable/immutable 객체 비교(정수형 변수는 immutable)

k=0
n=0
book=["book1","book2","book3"]                      #리스트 book  선언

while k!= " 5 " :                                   #new_book이 5개 추가 될때까지 반복
    n=n+1
    k=str(n)
    buy(book[:],k)                                  #buy함수에 book리스트의 참조값 전달(복사본)
    print(n, " * buy() outside " , book)            #메인부의 book리스트 출력(buy함수 바깥쪽) -> buy함수 내부와 비교하기 위해
                                                    #book리스트의 참조값을 인수로 전달했기 때문에 두 출력이 다르다
    booklist(book, k)                               #booklist함수에 book리스트 전달

Print(book)                                         #book리스트가 변경되었는지 확인(booklist()에서 변경되었음이 출력에서 확인)
```

출력

```
1 buy() :  ['book1', 'book2', 'book3', 'new_book1']
1 * buy() outside ['book1', 'book2', 'book3']
2 buy() :  ['book1', 'book2', 'book3', 'new_book1', 'new_book2']
2 * buy() outside ['book1', 'book2', 'book3', 'new_book1']
3 buy() :  ['book1', 'book2', 'book3', 'new_book1', 'new_book2', 'new_book3']
3 * buy() outside ['book1', 'book2', 'book3', 'new_book1', 'new_book2']
4 buy() :  ['book1', 'book2', 'book3', 'new_book1', 'new_book2', 'new_book3', 'new_book4']
4 * buy() outside ['book1', 'book2', 'book3', 'new_book1', 'new_book2', 'new_book3']
5 buy() :  ['book1', 'book2', 'book3', 'new_book1', 'new_book2', 'new_book3', 'new_book4', 'new_book5']
5 * buy() outside ['book1', 'book2', 'book3', 'new_book1', 'new_book2', 'new_book3', 'new_book4']
['book1', 'book2', 'book3', 'new_book1', 'new_book2', 'new_book3', 'new_book4', 'new_book5']
```

실습