# [ 함수의 고급 ]

- 다양한 자료구조를 사용하는 함수의 작성
- 지역변수와 전역변수
- 디폴트 인수
- 키워드 인수
- 람다 함수

# 다양한 자료구조를 사용하는 함수의 작성

- 앞에서 배운 다양한 자료구조를 함수와 함께 사용 가능
- 작성하는 함수의 매개변수로 넘기고, 함수의 반환값으로도 사용 가능

# 문자열 사용 함수

- 임의의 문자열 s를 함수에 넘기면 그 문자열의 순서를 뒤집어 새로운 문자열 s2를 반환받는 함수 reverseStr()과 그것을 이용하는 예 :

```
def reverseStr(s):
    s2 = s[::-1] # s의 순서를 뒤집어 s2를 만듬
    return s2

s = input('Type any string: ')
s2 = reverseStr(s)
print(s, s2)
```

# 리스트 사용 함수

- 임의의 리스트 L을 넘겨 받으면 그 내용을 출력하는 함수 prtList(L)와 사용 예 :

```
def prtList(L):
    print(L)

L = [1,2,3]
prtList(L)
```

- 임의의 숫자 리스트를 넘겨 받으면 그 리스트의 요소를 제곱해 만든 새로운 리스트를 반환하는 함수 sqList()와 사용 예 :

```
def ListSq(L):
    L2 = [ i**2 for i in L]
    return L2

L = [1,2,3]
print(ListSq(L))
```

# 리스트 사용 함수(2)

- 임의의 정수로 이루어진 리스트 L을 함수에 넘기면 그 리스트 중 짝수만을 추출해 만든 새로운 리스트 L2를 반환하는 함수 even()과 그것을 이용하는 예 :

```
def even(L):
    L2 = _________________________ # 넘겨받은 정수 리스트 L의 요소 중 짝수만으로 새로운 리스트 L을 만듬
    return L2

L = [1, 2, 3, 4, 5, 6, 7, 8, 9]
_________________
print(L2)
```

# 튜플을 사용하는 함수의 예

- 임의의 두 튜플 T1, T2를 넘겨받아 그 두 튜플을 합쳐 새로운 튜플 T를 반환받는 함수 funcTuple()과 그것을 이용하는 예 :

```
def addTuple(T1, T2):
    T = T1 + T2
    return T

T1 = (1,2,3)
T2 = (4,5)
T = addTuple(T1, T2)
print(T)
```

- 튜플을 이용하면 다음과 같이 정사각형의 넓이와 둘레를 한번에 튜플로 반환하는 함수를 작성할 수 있다.

```
def square(s) :
    area = s*s
    circumference = 4*s
    return (area, circumference)

s = float(input("정사각형 한변의 길이 : "))
(a, c) = square(s)
print("정사각형 넓이="+str(a)+"둘레="+str(c))
```

# 딕셔너리 사용 함수

- 임의의 주어진 딕셔너리 D에 임의의 내용을 추가해 새로운 딕셔너리 D2를 반환받는 함수 addDic()과 그것을 이용하는 예 :
- 리스트의 경우와 유사하게 깊은 복사를 사용해야 제대로 동작함을 유의

```
def addDic(D, key, val):
    newD = D.copy()
    newD[key] = val
    return newD

D = {'a':1, 'b':2}
D2 = addDic(D, 'c', 3)
print(D)
print(D2)
print(id(D))
print(id(D2))
```

# 세트 사용 함수

- 임의의 두 집합 S1, S2를 넘겨받아 그 두 집합의 합집합 S를 반환받는 함수 setUnion()과 그것을 이용하는 예 :

```
def setUnion(S1, S2):
    S = S1.union(S2)
    return S

S1 = {1,2,3}
S2 = {3,4,5}
S = setUnion(S1, S2)
print(S1)
print(S2)
print(S)
```

# 함수의 반환값은 그전처럼 하나인가? 아니면 여럿도 가능한가?

- 함수의 리턴값은 언제나 하나이다 (교재 163쪽)
- 함수는 항상 한 종류의 데이터를 반환함 _

```
def test(L):
    L2 = [n for n in L if n%2 == 0]
    T = (1,2,3)
    D = {1:1, 2:2}
    return L2, T, D  # possible
    # return [L2, T, D] # possible
    # return {L2, T, D} # impossible

L = [1, 2, 3, 4, 5, 6, 7, 8, 9]
R = test(L)
print(R)
print(type(R))
```

# 매개변수를 지정해 호출

- 함수 호출시 매개변수를 지정해 호출할 수 있음(교재 159쪽 참조)

# 여러 개의 매개변수 전달: *이 붙은 매개 변수

- 때로는 함수에 매개변수의 개수를 임의로 지정해주고 싶을 때가 있다. 이것을 가변 매개 변수라 하는 데, 다음 예제와 같이
- 매개변수의 이름 앞에 별표( `*` )를 사용하여 인수(arguments)가 변할 수 있음을 표시
- 별(*) 기호가 붙은 매개 변수(예: *param)를 지정해 주면 함수에 넘겨진 모든 인수들이 `param` 이라는 이름의 튜플로 묶여 넘어와 처리됨 :

```
def total(*numbers):
    sum = 0
    for n in numbers:
        sum += n
    return sum

print (total(1,2,3))
print (total(1,2,3,4,5))
```

## 또 다른 예 :

```
def calc(choice, *args):
    if choice == "add":    # 매개변수 choice에 "add"를 입력받았을 때
        result = 0
        for i in args:
            result = result + i
    elif choice == "mult":    # 매개변수 choice에 "mult"을 입력받았을 때
        result = 1
        for i in args:
            result = result * i
    return result

print(calc('add', 1,2,3,4))
print(calc('mult', 1,2,3,4,5))
```

# 키워드 매개변수 : **이 붙은 매개 변수

- 매개변수를 **param과 같이 별 2개를 붙여 지정하면 함수에 넘겨진 모든 인수들이 `param` 이라는 이름의 딕셔너리로 전달됨
- 따라서 함수를 호출할 때 매개변수를 딕셔너리 `키=값` 형식으로 사용해야 함

```
def print2star(**kwargs):
    print(kwargs)

print2star(a=1)
print2star(b=2, c=3)
```

- 함수의 입력값으로 a=1이 사용되면 kwargs는 {'a': 1}이라는 딕셔너리 임
- 입력값으로 b=2, c=3이 사용되면 kwargs는 {'b': 2, 'c': 3}라는 딕셔너리 임

## 또 다른 예

```
def dicPresident(**keywords):
    for i in keywords.keys():
        print("%s : %d-th president" %(i, keywords[i]))

dicPresident(Kennedy=35, Obama=44, Trump=45)
```

# 디폴트 인수(default argument) :

- 함수의 매개변수가 기본값을 가질 수 있게 하는 것
- 함수를 호출할 때 인수를 선택적으로 넘겨주게 해 사용자가 값을 넘겨주지 않으면 자동으로 기본값을 사용하도록 하는 것이 편함
- 함수를 선언할 때 원하는 매개 변수 뒤에 대입 연산자(=)와 기본값을 입력해 기본 인수값을 상수로 지정
- 다음 예의 실행 결과를 확인해 동작원리를 이해

```
def prtMesg(message, count=1):
    print (message * count)

prtMesg('Hi! ', 5)
prtMesg('default')
```

# 키워드 인수를 이용한 매개변수 전달 :

- 함수 선언시 지정된 매개 변수의 순서대로 값을 넘겨주는 것은 물론
- 매개 변수들의 위치 대신 이름(키워드)을 사용해 각 매개 변수에 인수를 넘기도록 지정하는 것도 가능
- 키워드 인수를 사용하면 인수의 순서를 신경쓰지 않고 함수를 쉽게 호출할 수 있는 점과 특정한 매개 변수에만 값을 넘기도록 하여 나머지는 주어진 기본 인수값으로 채울 수 있다는 장점

```
def func(a, b=2, c=3):
    print('a=', a, 'b=', b, 'c=', c)
    return a + b + c

print(func(1))
print(func(4, 5))
print(func(4, 5, 6))
print(func(4, c=7, b=3))
print(func(5, b=5, c=7))
# print(func(5, x=5, y=7))
```

# 지역변수(local variable)와 전역 변수(global variable)

- 변수는 함수 안에서만 사용하는 지역변수와 함수 밖에서도 사용할 수 있는 전역 변수로 나눈다.
- 지역변수(local variable)
  - 함수의 안에서 정의되고 사용하는 변수
  - 함수에 인수(argument)로 전달되는 매개변수(parameter)도 지역변수와 같은 특성
- 전역변수(global variable)
  - 함수의 밖에서 정의되는 변수
  - 함수의 안과 밖에서 사용 가능
  - 전역변수임을 쉽게 알 수 있도록 하는 것이 권장됨 (예: 변수 이름을 길게 또는 G로 시작 등)

# 지역변수만 사용 :

```
# 함수 선언부
def func1() :
    v = 10    # 함수 func1()안의 지역 변수. func1()안에서만 유효
    print("func1()의 v = %d" %v)

def func2() :
    v = 20    # 함수 func2()안의 지역 변수. func2()안에서만 유효
    print("func2()의 v = %d" %v)

# 메인 코드 부분
func1() # 지역 변수 값인 10을 출력
func2() # 전역 변수 값인 20을 출력
```

# 지역변수와 전역변수 사용 :

```
# 함수 선언부
def func1() :
    v = 10    # 함수 func1()안의 지역 변수. func1()안에서만 유효
    print("func1()의 v = %d" %v)

def func2() :
    print("func2()의 Gv = %d" %Gv) # 함수 func2()안의 전역 변수
    # Gv = 100    # 전역변수 값 변경?

# 전역변수 선언부
v = 20      # 전역 변수이므로 프로그램 내에서 유효

# 메인 코드 부분
func1() # 지역 변수값 출력
func2() # 전역 변수값 출력
```

# 함수 안에서 전역변수의 값을 변경 :

- 함수 안에서 전역 변수의 값을 변경하려면 global 키워드로 전역변수임을 알려야 함

```
# 함수 선언부
def func1() :
    v = 10    # 함수 func1()안의 지역 변수 v. func1()안에서만 유효
    print("func1()의 v = %d" %v)

def func2() :
    global Gv
    Gv = 200    # 함수 func2()안에서 Gv의 값 변경
    print("func2()의 Gv = %d" %Gv)

# 전역변수 선언부
Gv = 20   # 전역 변수이므로 프로그램 내에서 유효
# 메인 코드 부분
func1() # 지역 변수값 출력
func2() # 전역 변수값 출력
print("Gv = %d" %Gv)
```

# 람다(lambda, 무명) 함수

- 이름은 없고 몸체만 있는 함수를 한 줄로 간단히 만들어 사용
- 람다 함수는 `lambda` 키워드로 생성. return 명령어 없음
- lambda 함수의 기본 구조와 사용 예

```
lambda parameters : expr # parameters는 매개변수, expr는 결과 반환식
print('10+20=', (lambda x, y: x+y)(10,20))
```

- 위의 예는 간단한 한번 작업에 유용
- 여러번 사용이 필요하면 지정문을 사용해 함수이름을 붙인 후 재사용도 가능

```
fn_name = lambda parameters : expr # fn_name은 함수이름, parameters는 매개변수, expr는 결과 반환식
sum = lambda x, y: x+y # lambda는 계산값 x+y를 반환
print( "정수의 합 : ", sum( 20, 200 ))
```

# 여러 사용 예

- 다음은 람다함수의 계산값을 res에 지정 후 사용

```
res = (lambda x, y: x+y)(10,20)
print(res)
```

- 다음은 람다함수의 이름을 sum으로 지정 후 사용

```
sum = lambda x, y: x+y
sum(10,20)
```

```
myList = [1, 2, 3, 4, 5, 6, 7, 8, 9]
result = map(lambda x: x**2, myList)    # [1, 4, 9, 16, 25, 36, 49, 64, 81]

command=lambda t=button_text: click(t)).grid(row=row_index, column=col_index)
```