파이썬 프로그래밍

파이썬 함수

한국기술교육대학교
온라인평생교육원

▣ 함수의 정의와 호출

1. 간단한 함수의 정의

•함수: 여러 개의 Statement들을 하나로 묶은 단위
•함수 사용의 장점
- 반복적인 수행이 가능하다
- 코드를 논리적으로 이해하는 데 도움을 준다
- 코드의 일정 부분을 별도의 논리적 개념으로 독립화할 수 있음
◦수학에서 복잡한 개념을 하나의 단순한 기호로 대치하는 것과 비슷

•함수 정의시 사용하는 키워드: def

▪함수를 잘 정의해 두면 함수 호출을 통해 계속 수행 가능
▪함수를 잘 정의해 두면 전체 코드 논리적 이해 가능
▪반복적으로 수행하는 것은 함수로 묶어 코딩 하는 것이 이익
▪함수 정의 시 사용하는 키워드 : def (define의 약자)

▣ 함수의 정의와 호출

1. 간단한 함수의 정의

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

print add(1, 2)
print

def myabs(x):
    if x < 0 :
        x = -x
    return x

print abs(-4)
print myabs(-4)
```

```
3

4
4
```

- abs() → 내장되어 있는 기본 함수

▣ 함수의 정의와 호출

2. 함수 객체와 함수 호출

•함수의 이름 자체는 함수 객체의 레퍼런스(Reference)를 지니고 있다.

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

print add
```

```
<function add at 0x10d9ad6e0>
```

▪ 파이썬에서 함수는 객체 → add에는 함수객체의 레퍼런스가 있음

```
c = add(10, 30)
print c
```

```
40
```

•함수 이름에 저장된 레퍼런스를 다른 변수에 할당하여 그 변수를 이용한 함수 호출 가능

```
f = add
print f(4, 5)
```

```
9
```

▪ f = add → add의 참조값이 f에 복사

▣ 함수의 정의와 호출

2. 함수 객체와 함수 호출

```
print f


print f is add
```

```
<function add at 0x10b5ad758>
True
```

- add와 f가 가리키는 함수 객체가 동일
- 함수를 정의하게 되면 인자가 없어도 () 꼭 삽입

•함수의 몸체에는 최소한 한개 이상의 statement가 존재해야 함
 - 아무런 내용이 없는 몸체를 지닌 함수를 만들 때에는 pass 라는 키워드를
   몸체에 적어주어야 함

```
def simpleFunction():
    pass


simpleFunction()
```

- 함수의 내용이 없어도 'pass' 꼭 입력
- pass : 아무것도 수행하지 않고 지나가겠다는 의미

▣ 함수의 정의와 호출

2. 함수 객체와 함수 호출

•함수 사용 예

```
def addmember(members, newmember):
    if newmember not in members:   # 기존 멤버가 아니면
        members.append(newmember)  # 추가

members = ['hwang', 'lee', 'park', 'youn']  # 리스트에 초기 멤버 설정

addmember(members, 'jo')  # 새로운 멤버 추가

addmember(members, 'hwang') # (이미 존재하는) 새로운 멤버 추가

print members
```

```
<function add at 0x10b5ad758>
['hwang', 'lee', 'park', 'youn', 'jo']
```

▪members → 리스트 반환

▪newmember → 문자열 반환

▪append 메소드 : 추가

▪members와 addmembers는 동일한 객체(리스트)를 가리킴

▣ 함수의 정의와 호출

3. 함수 인수값 전달방법

•파이썬에서의 인수값 전달 방법
- 기본적으로 값에 의한 호출(Call-by-Value)
- 하지만 변수에 저장된 값이 참조값(Reference)이므로 실제로는 참조에 의한 호출(Call-by-Reference)로 실행됨

•함수 인자에 변경불가능(Immutable) 객체인 숫자값을 전달
- 함수 내에서 다른 숫자값으로 치환 --> 의미 없는 인자 전달

▪파이썬에서 기본적으로 인수값을 전달하는 방법 = 값에 의한 호출
▪members 변수의 값 자체가 전달되는 것
▪형식은 call by value 이나 내용은 call by reference

```
def f1(b):
    b = 100

a = 200
f1(a)
print a
```

```
200
```

▪a 변수는 200 객체의 레퍼런스 값을 가짐
▪함수 호출 시 레퍼런스 값이 b에 카피 → call by value
▪안의 내용 자체가 200을 가리키는 레퍼런스이므로 b도 200 가리킴

▣ 함수의 정의와 호출

3. 함수 인수값 전달방법

•함수 인자에 변경불가능(Immutable) 객체인 문자열을 전달
- 함수 내에서 다른 문자열로 치환 --> 의미 없는 인자 전달

```
def f2(b):
    b = "abc"

a = "def"
f2(a)
print a
```

```
def
```

▪a와 b 모두 문자열을 가리키며 레퍼런스 값 가짐
▪문자열은 변경할 수 없는 자료형
▪튜플도 같은 맥락으로 확인 가능

▣ 함수의 정의와 호출

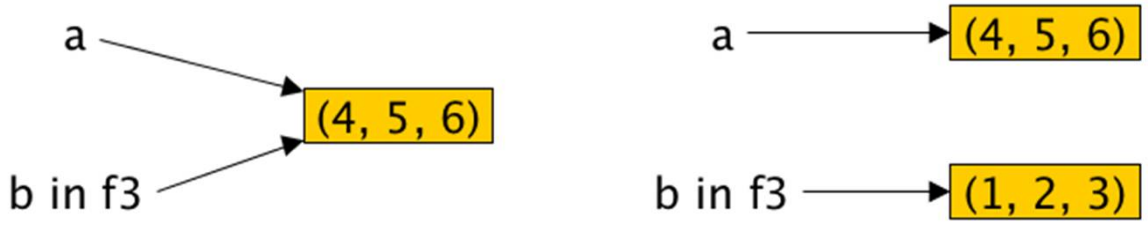
3. 함수 인수값 전달방법

•함수 인자에 변경불가능(Immutable) 객체인 튜플을 전달
 - 함수 내에서 다른 튜플로 치환 --> 의미 없는 인자 전달

```
def f3(b):
    b = (1,2,3)

a = (4,5,6)
f3(a)
print a
```

```
(4, 5, 6)
```



▪a의 레퍼런스 값이 b에 전달됨

▣ 함수의 정의와 호출

3. 함수 인수값 전달방법

•함수 인자에 변경가능한(Mutable)한 객체인 리스트 전달 및 내용 수정
- 전형적인 함수 인자 전달법 및 활용법

```
def f4(b):
    b[1] = 10

a = [4,5,6]
f4(a)
print a
```

```
[4, 10, 6]
```

▪리스트는 변경 가능한 자료형
▪함수에 인자 전달 시 리스트로 전달하는 경우 많음

▣ 함수의 정의와 호출

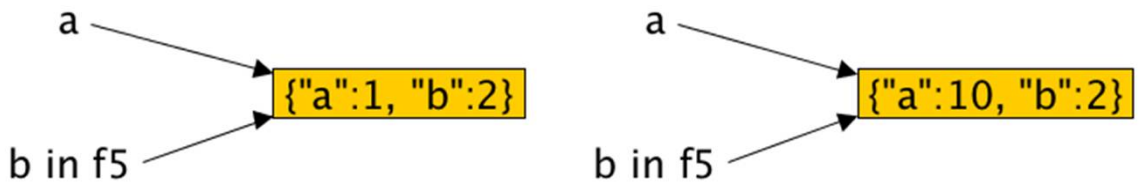
3. 함수 인수값 전달방법

•함수 인자에 변경가능한(Mutable)한 객체인 사전 전달 및 내용 수정
- 전형적인 함수 인자 전달법 및 활용법

```
def f5(b):
    b['a'] = 10

a = {"a":1, "b":2}
f5(a)
print a
```

```
{'a': 10, 'b': 2}
```



▪리스트나 사전처럼 변경 가능한 것이 함수 인자 전달 시 쓰임

▣ 함수의 정의와 호출

4. 반환(return)문

•인수 없이 return 문을 사용하면 실제로는 None 객체가 전달된다.
- None 객체: 파이썬 내장 객체로서 아무 값도 없음을 나타내기 위한 객체

```
def nothing():
    return

print nothing()
```

```
None
```

▪return 내용 없음 → 아무런 내용도 리턴 하지 않음 → none 출력
▪none 객체 : 파이썬 내장객체로 아무 값도 없음을 나타냄

•return문 없이 리턴하기

```
def print_menu():
    print '1. Snack'
    print '2. Snake'
    print '3. Snick'

print_menu()
```

```
1. Snack
2. Snake
3. Snick
```

▪return을 쓴 것 = 아무것도 안 쓴 것

▣ 함수의 정의와 호출

4. 반환(return)문

•return문 없는 함수라고 할 지라도 실제로는 None 객체가 전달됨

```
a = print_menu()

print a
```

```
1. Snack
2. Snake
3. Snick
None
```

•한 개의 값을 리턴할 때

```
def abs(x):
    if x < 0 : return -x
    return x

print abs(-10)
```

```
10
```

▪abs는 내장함수로 존재함으로 가급적 다른 이름 쓰는게 O
▪콜론(:) 뒤에는 들여쓰기
▪콜론 뒤 문장이 한 개인 경우는 바로 적어도 무방

▣ 함수의 정의와 호출

4. 반환(return)문

•두 개 이상의 값을 리턴할 때

```
def swap(x, y):
    return y, x   # 튜플로 리턴된다.

a = 10
b = 20
print a, b
print

a, b = swap(a, b) # 결과적으로 a, b = b, a와 동일
print a, b
print

a = 10
b = 20
x = swap(a, b)
print x[0], x[1]  # 하나의 이름으로 튜플을 받아서 처리할 수 도 있다.
```

```
10 20

20 10

20 10
```

▪튜플의 용도 → 2개 이상의 값을 리턴하는 함수
▪콤마(,)를 이용해서 하나 이상의 값을 동시에 반환 가능
▪즉, 튜플 기호가 숨어 있어 하나의 튜플을 반환하는 것
▪x = swap(a,b) → (y,x)

▣ 함수의 정의와 호출

4. 반환(return)문

•새로운 리스트를 리턴하는 함수 예
- 문자열 리스트를 받아서 각 문자열의 길이 정보를 지닌 리스트를 리턴

```
def length_list(l):
    res = []
    for el in l:
        res.append(len(el))
    return res

l = ['python', 'pyson', 'pythong', 'pydon']
print length_list(l)
```

```
[6, 5, 7, 5]
```

▪len(el) → 내장함수로 el에 들어 있는 각 문자열의 길이 반환

```
l = ['python', 'pyson', 'pythong', 'pydon']
print [len(s) for s in l]
```

```
[6, 5, 7, 5]
```

▪[len(s) for s in l] = 리스트 내포
▪[len(s) for s in l] → l 안에 있는 각 문자열을 가지고 오면서 len 실행

▣ 함수의 정의와 호출

5. 함수 인자에 대한 동적인 자료형 결정

•파이썬에서는 모든 객체는 동적으로 (실행시간에) 그 타입이 결정된다.
•그러므로, 함수 인자는 함수가 호출되는 순간 해당 인자에 전달되는 객체에 따라 그 타입이 결정된다.
- 함수 몸체 내에서 사용되는 여러가지 연산자들은 함수 호출시에 결정된 객체 타입에 맞게 실행된다.

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

c = add(1, 3.4)
d = add('dynamic', 'typing')
e = add(['list'], ['and', 'list'])
print c
print d
print e
```

```
4.4
dynamictyping
['list', 'and', 'list']
```

▪파이썬은 변수나 인자 상황에 type을 적지 않음
▪값이 실제 정의된 함수에 할당이 될 때 type 결정됨

파이썬 프로그래밍

파이썬 함수

한국기술교육대학교
온라인평생교육원

▣ 함수 인수 처리

1. 기본 인수 값

•기본 인수 값
  - 함수를 호출할 때 인수를 넘겨주지 않아도 인수가 기본적으로 가지는 값

```
def incr(a, step=1):
    return a + step

b = 1
b = incr(b)      # 1 증가
print b

b = incr(b, 10) # 10 증가
print b
```

```
2
12
```

▪step에는 기본적으로 인수값 1이 할당됨
▪incr(b)에서 b는 a에 들어감
▪incr(b, 10) → 10은 step에 들어감
▪기본인자 없으면 error 발생 → 정의한 함수의 인자가 2개기 때문

▣ 함수 인수 처리

1. 기본 인수 값

•[주의] 함수 정의를 할 때 기본 값을 지닌 인수 뒤에 일반적인 인수가 올 수 없음

```
def incr(step=1, a):
    return a + step
```

```
File "<ipython-input-82-67693752f310>", line 1
    def incr(step=1, a):
SyntaxError: non-default argument follows default argument
```

▪step = 1 이라고 정의한 것은 앞 인자에 쓰지 못함
▪기본인자는 맨 마지막 뒤에 와야 함
▪인자가 여러 개일 경우 기본인자 값이 없는 것이 맨 앞 위치

•함수 정의 시에 여러 개의 기본 인수 값 정의 가능

```
def incr(a, step=1, step2=10):
    return a + step + step2

print incr(10)
```

```
21
```

▪일반인자가 앞으로 위치해야 함

▣ 함수 인수 처리

2. 키워드 인수

• 인수 값 전달 시에 인수 이름과 함께 값을 전달하는 방식을 일컫는다.

```
def area(height, width):
    return height * width

#순서가 아닌 이름으로 값이 전달
a = area(height='height string ', width=3)
print a

b = area(width=20, height=10)
print b
```

```
height string height string height string
200
```

▪height 는 문자열, width에는 3, 이름으로 값이 들어감
▪이름으로 값이 들어가면 순서 상관 X

▣ 함수 인수 처리

2. 키워드 인수

•함수를 호출 할 때에 키워드 인수는 마지막에 놓여져야 한다.

```
print area(20, width=5)
```

```
100
```

▪width=5 → 키워드 인자 : 정의한 문자를 사용하여 가리킴
▪키워드 인자는 기본 인자와 마찬가지로 맨 뒤 위치

•[주의] 함수 호출시에 키워드 인수 뒤에 일반 인수 값이 올 수 없다.

```
area(width=5, 20)
```

```
  File "<ipython-input-80-32b5ca4bbf7f>", line 1
    area(width=5, 20)
SyntaxError: non-keyword arg after keyword arg
```

▣ 함수 인수 처리

2. 키워드 인수

•기본 인수값 및 키워드 인수의 혼용

```
def incr(a, step=1, step2=10, step3=100):
    return a + step + step2 + step3

print incr(10, 2, step2=100)
```

```
212
```

•함수 호출 시에 키워드 인수 뒤에 일반 인수 값이 오면 에러

```
def incr(a, step=1, step2=10, step3=100):
    return a + step + step2 + step3

print incr(10, 2, step2=100, 200)
```

```
  File "<ipython-input-89-3dbdc3e84e66>", line 4
    print incr(10, 2, step2=100, 200)
SyntaxError: non-keyword arg after keyword arg
```

▪키워드 인수는 중간에 위치할 수 없음

```
def incr(a, step=1, step2=10, step3=100):
    return a + step + step2 + step3

print incr(10, 2, step2=100, step3=200)
```

```
312
```

▣ 함수 인수 처리

3. 가변 인수 리스트

•함수 정의시에 일반적인 인수 선언 뒤에 *var 형식의 인수로 가변 인수를 선언할 수 있음
- var에는 함수 호출시 넣어주는 인수 값들 중 일반 인수에 할당되는 값을 제외한 나머지 값들을 지닌 튜플 객체가 할당된다.

```
def varg(a, *arg):
    print a, arg

varg(1)
varg(2,3)
varg(2,3,4,5,6)
```

```
1 ()
2 (3,)
2 (3, 4, 5, 6)
```

▪*arg → 가변 인수를 받겠다는 의미
▪*arg → 튜플 형태로 반환함
▪콤마(,)가 있어야지 원소가 하나인 튜플을 반영
▪arg[0] → 튜플의 첫 번째 원소

•C언어의 printf문과 유사한 형태의 printf 정의 방법

```
def printf(format, *args):
    print format % args

printf("I've spent %d days and %d night to do this", 6, 5)
```

```
I've spent 6 days and 5 night to do this
```

▪가변인수를 활용하면 C언어의 printf와 유사하게 사용 가능

▣ 함수 인수 처리

4. 튜플 인수와 사전 인수로 함수 호출하기

•함수 호출에 사용될 인수값들이 튜플에 있다면 "*튜플변수"를 이용하여
함수 호출이 가능

```
def h(a, b, c):
    print a,b,c

args = (1, 2, 3)
h(*args)
```

```
1 2 3
```

▪함수 정의 시 * 사용하게 되면 가변인수
▪함수 호출 시 * 사용하고 뒤에 튜플을 넣으면 튜플 전체 호출 가능

•함수 호출에 사용될 인수값들이 사전에 있다면 "*사전변수"를 이용하여
함수 호출이 가능

```
dargs = {'a':1, 'b':2, 'c':3}
h(**dargs)
```

```
1 2 3
```

▪함수 호출 시 ** 사용하고 뒤에 사전을 넣으면 사전 전체 호출 가능
▪인자의 이름과 식별자가 동일한 키 값을 가져야 함