UNISTUDY

파이썬

17강. 특수함수

1. 특수함수

- 특수함수는 특정한 문제의 해결 과정에서 필요로 하는 함수를 의미하는데, 여기서 특수 함수는 일반적 인 프로그래밍의 함수와는 다르게 파이썬에서만 특별하게 제공되는 함수와 특수 기능을 갖는 함수들에 대해서 알아본다.

2. 가변인수 함수

- 일반적인 함수에서는 하나의 매개변수를 이용하여 하나의 실인수를 받도록 되어있다. 하지만 파이썬에 서는 하나의 매개변수로 여러 개의 실인수를 받을 수 있는 가변인수를 제공한다.
- 가변인수를 갖는 함수 는 다음 형식과 같이 실인수를 받는 매개변수 앞부분에 * 또는 ** 기호를 붙인다. 여러 개의 실인수를 하나의 매개변수로 받을 때 '*매개변수' 형식은 튜플(tuple) 자료구조로 받고, '**매개변수' 형식은 딕 트(dict) 자료구조로 받는다.

```
def 함수명(매개변수, *매개변수, **매개변수) :↵
```

2. 가변인수 함수

chapter05.lecture.step03_special_func.py

```
# (1) 튜플형 가변인수
def Func1(name, *names):
    print(name)  # 실인수 : 홍길동
    print(names)  # 실인수 : ('이순신', '유관순')

Func1("홍길동", "이순신", "유관순")

# statistics 모듈 import
from statistics import mean, variance, stdev
```

Python Console

```
홍길동
('이순신', '유관순')
```

2. 가변인수 함수

```
# (2) 통계량 구하는 함수
def statis(func, *data):
    if func == 'avg':
        return mean(data)
    elif func == 'var':
        return variance(data)
    elif func == 'std':
        return stdev(data)
    else:
        return 'TypeError'

# statis 함수 호출
print('avg=', statis('avg', 1, 2, 3, 4, 5))
print('var=', statis('var', 1, 2, 3, 4, 5))
print('std=', statis('std', 1, 2, 3, 4, 5))

# (3) 딕트형 가변인수
def emp_func(name, age, **other):
    print(name)
    print(age)
    print(other)

# emp_func 함수 호출
emp_func('홍길동', 35, addr='서울시', height=175, weight=65)
```

```
avg= 3
var= 2.5
std= 1.5811388300841898

홍길동
35
{'addr': '서울시', 'height': 175, 'weight': 65}
```

2. 가변인수 함수

- 가변인수를 갖는 함수 예
- # (1) 튜플형 가변인수
- Func1() 함수는 name과 names의 매개변수를 갖는다. name은 일반 매개변수이고, names는 가변인수 의 역할을 하는 매개변수이다. 함수 호출 부분에서 3개의 실인수("홍길동", "이순신", "유관순")를 사용하기 때문에 맨 처음 실인수는 name 매개변수로 할당되고, 나머지 2개는 가변인수인 names 변수로 할당된다. 가변인수는 여러 개의 실인수를 튜플(tuple) 자료구조로 받는다.
- # (2) 통계량 구하는 함수
- 첫 번째 매개변수인 func는 함수명을 받는 역할이고, data는 가변인수 역할을 하는 매개변수이다. 함수명에 따라서 가변인수의 통계량을 구하는 함수이다. # statis 함수 호출 부분에서 첫 번째 실인수는 '함수명'이고, 두 번째 이후는 가변인수에 할당될 5개의 숫자이다. 함수의 호출 순서에 따라서 평균, 분산, 표준편차의 값이 출력된다.

2. 가변인수 함수

- # (3) 딕트형 가변인수
- 매개변수 앞부분에 '**' 기호를 붙이면 사전형 자료(key=value)를 받는 가변인수가 된다. #emp_func 함수 호출 부분에서 사전형 실인수 (addr='서울시', height= 175, weight=65)는 모두
- other 가변인수가 받는다. 딕트형 가변인수는 실인수를 딕트(dict) 자료 구조로 받는다.

3. 람다 함수

- 람다 함수(Lambda function)는 정의와 호출을 한 번에 하는 익명함수이다.
- 복잡한 함수 호출 과정 을 생략해서 처리 시간을 단축되고, 코드의 가독성을 제공하는 이점을 갖는다. 람다 함수의 형식은 다 음가 같다. 별도의 함수이름이나 인수를 나타내는 괄호, return 명령어를 볼 수 없다.

```
lambda 매개변수 : 실행문(반환값)
```

3. 람다 함수

- 람다 함수의 실행 과정은 lambda 명령어 다음에 오는 매개변수에 의해서 외부의 값을 받고, 콜론(:) 다음에는 실행문의 실행 결과가 반환된다. 한편 람다 함수를 인라인 함수라고도 부른다. 인라인 함수는 말 그대로 한 줄 문장 안으로 자연스럽게 삽입될 수 있는 함수를 말한다. 아래 예문은 몸무게를 'thin', 'normal', 'fat'으로 3개의 레이블(label)을 갖는 bmi 소속의 "label" 변수를 대상으로 'thin'이 람다 의 인수 x로 들어오면 [1,0,0], 'normal'이 들어오면 [0,1,0], 'fat'이 들어오면 [0,0,1]로 원 핫 인코 딩(one hot encoding)14)하는 과정이다.

- label_map = {"thin": [1,0,0], "normal": [0,1,0], "fat": [0,0,1]} bmi["label"] = bmi["label"].apply(lambda x : np.array(label_map[x]))

3. 람다 함수

chapter05.lecture.step03_special_func.py

```
# (1) 일반 함수
def Adder(x, y):
    add = x + y
    return add

print('add=', Adder(10, 20))

# (2) 람다 함수
print('add=', (lambda x, y: x + y)(10, 20))
```

Python Console

```
add= 30

add= 30
```

3. 람다 함수

- 해설 람다 함수 예
- # (1) 일반 함수
- 매개변수 x, y를 갖는 Adder() 함수는 두 인수를 더해서 덧셈 결과를 반환하는 일반함수 이다.
- # (2) 람다 함수
- lambda 명령어 다음에 매개변수 x, y를 넣고, 콜론 다음에 덧셈 명령문을 넣어서 덧셈 결과를 반환하 는 한 줄 함수이다. 람다 함수는 정의와 호출을 한 번에 가능하기 때문에 람다 함수 다음에 (10, 20) 형식으로 괄호 안에 실인수를 넣어서 호출한다.

4. 스코프(scope)

- 변수는 특정 지역에서만 사용되는 지역변수와 지역에 상관없이 전 지역에서 사용되는 전역변수로 분류 된다. 이처럼 변수가 사용되는 범위를 스코프라고 한다. 결국 스코프는 특정 지역에 의해서 구분되는 데, 여기서 특정 지역은 함수 또는 블록(if, for, while 등)을 의미한다. 다음은 함수 내부에서 전역 변수를 사용하기 위한 명령문 형식이다.

```
def 함수명(인수) :
        global 전역변수
```

4. 스코프(scope)

chapter05.lecture.step03_special_func.py

```
# (1) 지역변수 예
x = 50  # 전역변수
def local_func(x):
    x += 50  # 지역변수 -> 종료 시점
    소멸
local_func(x)
print('x=', x)

# (2) 전역변수 예
def global_func():
    global x  # 전역변수 x 사용
    x += 50  # x+50 = 100

global_func()
print('x=', x)
```

Python Console

```
x= 50
x= 100
```

4. 스코프(scope)

- 해설 함수 스코프 예
- # (1) 지역변수 예
- 함수 바깥쪽에서 선언된 전역변수 x를 실인수로 함수를 호출할 경우 함수 내부에서 x에 50을 더해서 x는 100이 된다. 그리고 함수가 종료된 이후 x를 출력하면 100이 아닌 50이 출력된다. 결국 함수 내부에서 사용한 x는 함수 내부에서만 사용되는 지역변수이다. 지역변수는 해당 지역(함수 또는 블록)을 벗어나면 자동으로 소멸된다.
- # (2) 전역변수 예
- global_func( ) 함수에서 'global x' 명령문은 함수 바깥쪽에서 선언된 전역변수 x를 사용한다는 의미이다. 따라서 함수 내부에서 x에 50을 더해진 값 100이 함수가 종료되어도 그대로 유지된다.

THANK YOU