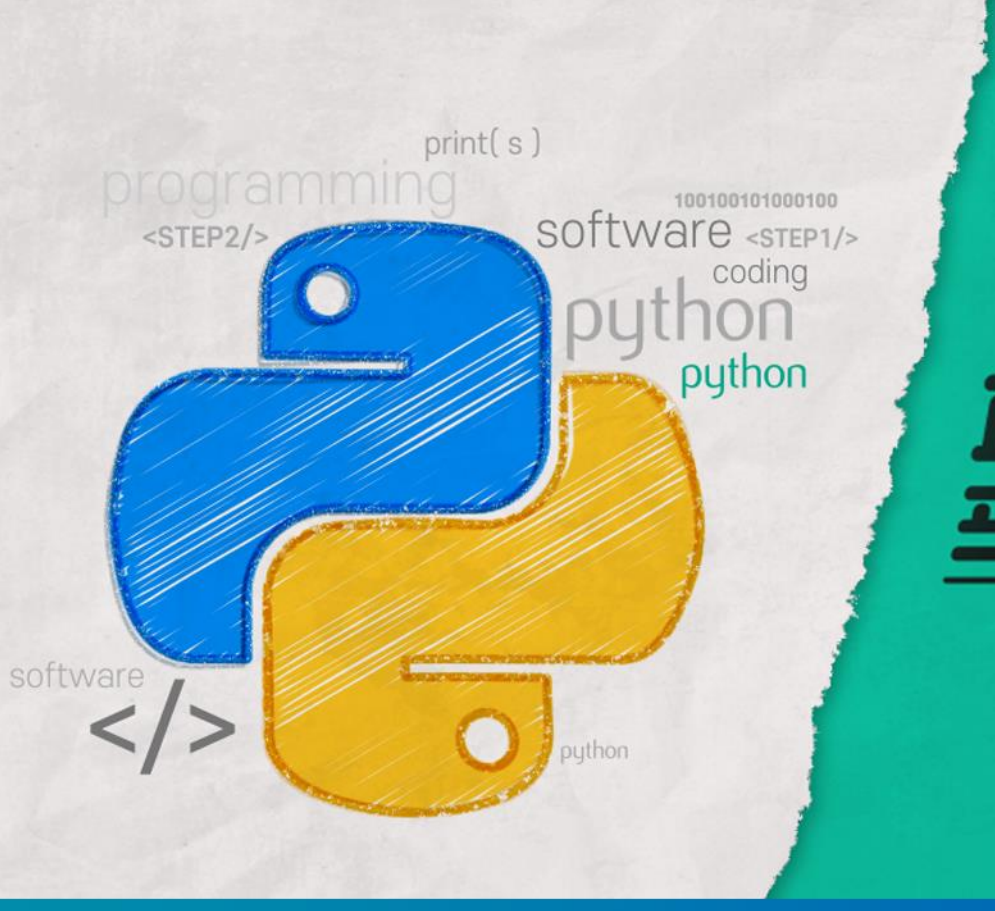


파이썬 </>
프로그래밍

파이썬 함수의 활용과 람다 함수

OLEI
한국기술교육대학교
온라인평생교육원

학습목표

- 람다 함수의 정의 및 활용 방법에 대해 이해하고, 코딩에 활용할 수 있다.
- 함수를 활용하는 예제들을 다뤄보고, 직접 코드를 작성할 수 있다.

학습내용

- 파이썬의 람다 함수
- 함수의 활용

파이썬의 람다 함수

1 람다 함수 정의

리스트 내포, 조건부 표현식 등과 같이
여러 줄의 코드를 간결하게 표현할 수 있도록 도와주는
새로운 함수 정의 방법

- lambda로 정의할 수 있고, def와 같은 역할
  - ➡ 함수의 이름을 정의하지 않고,
    일회성으로 간편하게 사용할 때 유용
  - ➡ def로 정의할 만큼 복잡하지 않을 때 활용
- lambda 매개변수1, 매개변수2, ... :
  매개변수를 이용한 표현식

파이썬의 람다 함수

1 람다 함수 정의

람다 함수를 add라는 변수에 할당해
일반 함수처럼 사용

람다 함수는 전통적인 함수와 다른 성격을 지님
(마크로 : macro)

```
def add(a, b):
    return a+b
result = add(3, 4)
print(result)
```

```
7
```

```
add = lambda a, b: a+b
result = add(3, 4)
print(result)
```

```
7
```

파이썬의 람다 함수

1 람다 함수 정의

따로 변수에 할당하지 않고 바로 한 줄로 표현 가능

```
print((lambda a, b: a+b)(3,4))
```

```
7
```

def 함수와 같이 기본 매개변수, 키워드 매개변수, 가변 매개변수 설정 가능

```
print((lambda a, b=1: a+b)(3))
```

```
4
```

```
print((lambda a, b: a+b)(b=1,a=3))
```

```
4
```

```
print((lambda a, *b: a*b)(3,3,4,5))
```

```
(3, 4, 5, 3, 4, 5, 3, 4, 5)
```

파이썬의 람다 함수

1 람다 함수 정의

조건문과 함께 사용 가능

```
print((lambda a,b: a if a%2==0 else b)(1,3))
```

```
3
```

```
def func(a,b):
    if a%2==0:
        return a
    else:
        return b
print(func(1,3))
```

```
3
```

파이썬의 람다 함수

2 람다 함수 활용

1 map 내장 함수와 함께 활용

- 시퀀스 자료형이 지닌 각 요소 값들에 대해 함수에 적용한 요소를 지닌 map 객체를 반환

```
def func(x):
    return x * x

a = [1, 2, 3, 4, 5]
b = map(func, a)
print(type(b))
print(list(b))
```

```
<class 'map'>
[1, 4, 9, 16, 25]
```

=

```
def func(x):
    return x * x

a = [1, 2, 3, 4, 5]
b = []
for x in a:
    y = func(x)
    b.append(y)
print(b)
```

```
[1, 4, 9, 16, 25]
```

```
print(list(map(lambda x: x * x, [1, 2, 3, 4, 5])))
```

```
[1, 4, 9, 16, 25]
```

파이썬의 람다 함수

2 람다 함수 활용

2 filter 내장 함수와 함께 활용

- 시퀀스 자료형이 지닌 각 요소의 값에 대해 함수에 적용한 결과가 참인 원소값만을 지닌 filter 객체를 반환

```
def func(x):
    return x>2

a = [1, 2, 3, 34]
b = []
print(list(filter(func,a)))
```

```
[3, 34]
```

=

```
def func(x):
    return x>2

a = [1, 2, 3, 34]
b = []
for x in a:
    if func(x):
        b.append(x)
print(b)
```

```
[3, 34]
```

파이썬의 람다 함수

2 람다 함수 활용

2 filter 내장 함수와 함께 활용

- 시퀀스 자료형이 지닌 각 요소의 값에 대해 함수에 적용한 결과가 참인 원소값만을 지닌 filter 객체를 반환

```
print(list(filter(lambda x: x > 2, [1, 2, 3, 34])))
```

```
[3, 34]
```

파이썬의 람다 함수

2 람다 함수 활용

3 복잡한 객체를 정렬할 때 활용

```
students = [
    ('영수', 'A', 15),
    ('철수', 'B', 16),
    ('영희', 'C', 10),
]
print(sorted(students))
print(sorted(students, key=lambda x: x[1]))
print(sorted(students, key=lambda x: x[2]))
```

```
[('영수', 'A', 15), ('영희', 'C', 10), ('철수', 'B', 16)]
[('영수', 'A', 15), ('철수', 'B', 16), ('영희', 'C', 10)]
[('영희', 'C', 10), ('영수', 'A', 15), ('철수', 'B', 16)]
```

4 문자열 포맷팅과 함께 활용

```
print((lambda x,y :'{} x {} = {}'.format(x, y, x * y))(3, 4))
```

```
3 x 4 = 12
```

함수의 활용

1 재귀 함수

재귀 함수는 자기 자신을 호출하는 함수로
파이썬은 자기 자신을 함수 내에서 호출할 수 있기 때문에
재귀 함수로 활용할 수 있음

파이썬은 끝없이 자기 자신을 호출해 무한루프에 빠지는 것을 방지하기 위해 **일정기간 반복하여 자기 자신을 호출할 경우 오류 발생**

- 종료 조건이 필요함

```
def recursive(num):
    print(num)
    num += 1
    recursive(num)

recursive(1)
```

함수 안에서 자기 자신을 계속 호출함

```
RecursionError: maximum recursion depth exceeded while calling a Python object
```

함수의 활용

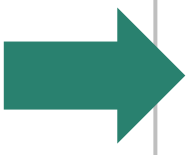
1 재귀 함수

팩토리얼 실습

팩토리얼 1부터 n까지의 곱셈의 결과

- 만약, n이 5이면 5 팩토리얼의 결과는
  5*4*3*2*1 = 120

15 팩토리얼을 구하는 함수를 작성하세요.
(재귀 함수 factorial() 사용)



```
factorial(15) = 15 * factorial(14)
              = 15 * 14 * factorial(13)
              = 15 * 14 * 13 ……
```

함수의 활용

1 재귀 함수

팩토리얼 실습

1 재귀 함수 작성

```
def factorial(n):
    return n * factorial(n-1)

print(factorial(15))
```

```
RecursionError: maximum recursion depth exceeded
```

함수의 활용

1 재귀 함수

팩토리얼 실습

2 종료 조건 작성

- 1부터 n까지이므로 n == 0 일 땐 1을 반환

```
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

print(factorial(15))
```

```
1307674368000
```

함수의 활용

1 재귀 함수

팩토리얼 실습

3 람다 함수로 변환

- 람다 함수는 함수명이 없기 때문에 변수에 저장하여 활용

```
fact = lambda x:(x == 0 and 1)or(x * fact(x-1))
print(fact(5))
```

```
120
```

함수의 활용

2 구구단 출력 실습

다음 코드를 람다 함수와 리스트 내포를 활용해 한 줄로 작성하세요.

```
l = []
for i in range(2,10):
    for j in range(1,10):
        l.append("{} X {} = {}".format(i,j,i*j))
print(l)
```

함수의 활용

2 구구단 출력 실습

1 2단부터 9단까지 구구단의 결과를 리스트로 출력하세요.

```
for i in range(2,10):
    for j in range(1,10):
        l.append(i*j)
print(l)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12,
5, 6, 12, 18, 24, 30, 36, 42, 48, 54, 7, 14,
3, 72, 81, 2, 4, 6, 8, 10, 12, 14, 16, 18, 3,
35, 40, 45, 6, 12, 18, 24, 30, 36, 42, 48, 54
5, 54, 63, 72, 81]
```

함수의 활용

2 구구단 출력 실습

2 리스트 내포를 활용해 2단부터 9단까지 구구단 출력문을 한 줄로 작성하세요.

```
print([x * y for x in range(2,10) for y in range(1,10)])
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12, 15, 18, 21, 24
5, 6, 12, 18, 24, 30, 36, 42, 48, 54, 7, 14, 21, 28, 35, 42,
3, 72, 81]
```

함수의 활용

2 구구단 출력 실습

3 두 개의 숫자를 넣으면 다음과 같이 출력하는 함수를 작성하세요.

- 출력 예시 : 3 X 4 = 12

```
def func(a,b):
    return "{} X {} = {}".format(a,b,a*b)
print(func(3,4))
```

```
3 X 4 = 12
```

```
print((lambda x,y :'{} x {} = {}'.format(x, y, x * y))(3, 4))
```

```
3 x 4 = 12
```

함수의 활용

2 구구단 출력 실습

4 람다 함수와 리스트 내포를 활용해 2단부터 9단까지 구구단 출력문을 한 줄로 작성하세요.

```
print([(lambda x,y :'{} x {} = {}'.format(x, y, x * y))(x, y) for x in range(2,10) for y in range(1,10)])
```

```
['2 x 1 = 2', '2 x 2 = 4', '2 x 3 = 6', '2 x 4 = 8', '2 x 5 = 10', '2 x 6 = 12', '2 x 7 = 14', '2 x 8 = 16', '
3', '3 x 2 = 6', '3 x 3 = 9', '3 x 4 = 12', '3 x 5 = 15', '3 x 6 = 18', '3 x 7 = 21', '3 x 8 = 24', '3 x 9 = 2
= 8', '4 x 3 = 12', '4 x 4 = 16', '4 x 5 = 20', '4 x 6 = 24', '4 x 7 = 28', '4 x 8 = 32', '4 x 9 = 36', '5 x 1
x 3 = 15', '5 x 4 = 20', '5 x 5 = 25', '5 x 6 = 30', '5 x 7 = 35', '5 x 8 = 40', '5 x 9 = 45', '6 x 1 = 6', '6
8', '6 x 4 = 24', '6 x 5 = 30', '6 x 6 = 36', '6 x 7 = 42', '6 x 8 = 48', '6 x 9 = 54', '7 x 1 = 7', '7 x 2 =
```

Run! 프로그래밍

Mission 1 | 한 줄의 람다 함수로 변경

```
def func(a):
    if a>10:
        return 'a가 10보다 크다.'
    else:
        return 'a가 10보다 작다.'func(14)
```

정답

```
print((lambda a : 'a가 10보다 크다' if a>10 else
'a가 10보다 작다')(14))
```

Run! 프로그래밍

Mission 2 | map 함수와 람다 함수 활용

```
def func(a):
    l=[]
    for i in range(a):
        l.append(i**2)
    return l
print(func(5))
```

정답

```
print(list(map(lambda x: x ** 2, range(5))))
```

학습정리

1. 파이썬의 람다 함수

람다 함수의 정의	• 리스트 내포, 조건부 표현식 등과 같이 여러 줄의 코드를 간결하게 표현할 수 있도록 도와주는 새로운 함수 정의 방법 • 함수는 lambda 매개변수1, 매개변수2, … : 매개변수를 이용한 표현식으로 정의할 수 있음
람다 함수의 활용	• 조건문, 내장 함수 map, filter와 함께 활용할 수 있음

2. 함수의 활용

재귀 함수	• 자기 자신을 함수 내에서 호출하는 함수 • 반복이 필요한 작업을 간결한 코드로 작성할 수 있음 • 무한루프에 빠지지 않도록 종료 조건 설정을 해주는 것이 필요
구구단 출력 실습	• 람다 함수, 리스트 내포 등을 활용하면 여러 줄의 복잡한 코드를 한 줄로 작성할 수 있음