파이썬 프로그래밍

람다 함수

한국기술교육대학교
온라인평생교육원

▣ 람다(lambda) 함수 정의

1. 람다 함수 정의 예

•람다(lambda) 함수 (or 축약 함수): https://wikidocs.net/64
- 일반적인 함수를 한 줄의 문(Statement)으로 정의할 수 있는 새로운 함수 정의 리터럴
  ◦함수 몸체에는 식(expression)만이 올 수 있다.
- 대부분의 경우 함수 이름을 정의하지 않으면서 일회성으로 활용할 함수를 정의할 때 활용
- 구문(syntax)
  ◦lambda 콤마로 구분된 인수들: 식(expression)

•인수가 한 개 있는 람다 함수

```
f = lambda x: x + 1
print f(1)
```

```
2
```

▪ lamda x: x + 1 → 하나의 statement
▪콜론(:) 뒤에는 expression(식)만 올 수 있음
▪f 식별자로 람다함수를 가리키게 할 수 있음 → 람다함수도 하나의 객체

▣ 람다(lambda) 함수 정의

1. 람다 함수 정의 예

•인수가 두 개 있는 람다 함수를 지니는 변수 지정 및 함수 호출

```
g = lambda x, y: x + y
print g(1, 2)
```

```
3
```

▪(1,2) → (x,y)

•기본 인수를 지니는 람다 함수 정의

```
incr = lambda x, inc = 1: x + inc
print incr(10) #inc 기본 인수 값으로 1 사용
print incr(10, 5)
```

```
11
15
```

▪람다함수도 기본 인수를 가질 수 있음

▪inc = 1 → 기본 인수

▣ 람다(lambda) 함수 정의

1. 람다 함수 정의 예

•가변 인수를 지니는 람다 함수 정의

```
vargs = lambda x, *args: args
print vargs(1,2,3,4,5)
```

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

▪가변 인수도 가질 수 있음
▪*args → 가변 인수
▪(1, 2, 3, 4, 5) → 1은 x, 나머지는 튜플형태로 args에 할당됨
▪args 변수가 리턴되는 것

▣ 람다(lambda) 함수 정의

2. 람다 함수 사용하기

```
def f1(x):
    return x*x + 3*x - 10

def f2(x):
    return x*x*x

def g(func):
    return [func(x) for x in range(-10, 10)]

print g(f1)
print g(f2)
```

```
[60, 44, 30, 18, 8, 0, -6, -10, -12, -12, -10, -6, 0, 8, 18, 30, 44, 60, 78, 98]
[-1000, -729, -512, -343, -216, -125, -64, -27, -8, -1, 0, 1, 8, 27, 64, 125, 216, 343,
512, 729]
```

▪람다 함수가 쓰이고 있지 않은 예제

▪함수 g 호출하는데 인자로 함수가 들어감 (함수도 객체이기 때문)

▪range(-10, 10) → [-10, -9, -8, ……, 7, 8, 9]

▪-10 * -10 + 3 * -10 -10 = 60

▪9 * 9 + 3 * 9 -9 = 98

▪TRUE

▪9 * 9 * 9 = 729

▣ 람다(lambda) 함수 정의

2. 람다 함수 사용하기

```
def g(func):
    return [func(x) for x in range(-10, 10)]

print g(lambda x: x*x + 3*x - 10)
print g(lambda x: x*x*x)
```

```
[60, 44, 30, 18, 8, 0, -6, -10, -12, -12, -10, -6, 0, 8, 18, 30, 44, 60, 78, 98]
[-1000, -729, -512, -343, -216, -125, -64, -27, -8, -1, 0, 1, 8, 27, 64, 125, 216, 343,
512, 729]
```

- 같은 예제를 람다함수로 간단히 수행 가능
- func에 한 줄짜리 람다 함수가 그대로 들어감
- 정상적인 함수를 미리 구현 X → 함수를 호출하는 순간 함수를 정의

▣ 람다(lambda) 함수 정의

2. 람다 함수 사용하기

•람다 함수를 사용하는 코드 예제

```
# 더하기, 빼기, 곱하기, 나누기에 해당하는 람다 함수 리스트 정의
func = [lambda x, y: x + y, lambda x, y: x - y, lambda x, y: x * y, lambda x, y: x / y]

def menu():
    print "0. add"
    print "1. sub"
    print "2. mul"
    print "3. div"
    print "4. quit"
    return input('Select menu:')

while 1:
    sel = menu()
    if sel < 0 or sel > len(func):
        continue
    if sel == len(func):
        break
    x = input('First operand:')
    y = input('Second operand:')
    print 'Result =', func[sel](x,y)
```

```
0. add
1. sub
2. mul
3. div
4. quit
Select menu : 0
First operand : 10
Second operand : 20
Result = 30
```

▣ 람다(lambda) 함수 정의

2. 람다 함수 사용하기

```
0. add
1. sub
2. mul
3. div
4. quit
Select menu : 1
First operand : 20
Second operand : 10
Result = 10

0. add
1. sub
2. mul
3. div
4. quit
Select menu : 2
First operand : 5
Second operand : 7
Result = 35

0. add
1. sub
2. mul
3. div
4. quit
Select menu : 3
First operand : 10
Second operand : 2
Result = 5

0. add
1. sub
2. mul
3. div
4. quit
Select menu : 4
```

▣ 람다(lambda) 함수 정의

2. 람다 함수 사용하기

- ▪리스트의 원소가 람다함수로 들어감
- ▪while 1 → 무한 루프
- ▪중간에 break 있어, 조건 미 충족 시 무한루프를 빠져나옴
- ▪0보다 작은 값이거나 4보다 큰 값(-1, -2, -3, 5, 6, 7, 8)은 수행 X
- ▪func의 길이가 4라서 4를 넣으면 break가 걸려 끝날 수 있음
- ▪select menu : 0 → sel에는 0이 들어가 있음
- ▪인덱스 0은 첫 번째 인자
- ▪인덱스 2는 세 번째 인자로 곱셈 수행
- ▪리스트의 원소에 람다함수가 들어가고, 검색도 가능

파이썬 프로그래밍

람다 함수

한국기술교육대학교
온라인평생교육원

▣ 람다 함수의 활용

1. map 내장 함수

•map(function, seq)
- seq 시퀀스 자료형이 지닌 각 원소값들에 대해 function에 적용한 결과를 동일 시퀀스 자료형으로 반환한다.

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

X = [1, 2, 3, 4, 5]
Y = map(f, X)
print Y
```

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

▪map(함수, 시퀀스 자료형)
▪시퀀스 자료형의 첫 번째 원소를 함수 수행
▪시퀀스 자료형의 유형에 따라 반환되는 유형 결정
▪map 함수는 쌍을 지어주는 것
▪x 원소가 5개이면, 수행 결과의 원소도 5개

▣ 람다 함수의 활용

1. map 내장 함수

•map 내장 함수를 사용하지 않을 때 코드

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

X = [1, 2, 3, 4, 5]
Y = []
for x in X:
    y = f(x)
    Y.append(y)
print Y
```

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

▪append(y) → 하나씩 x를 함수에 대응 후 결과 값을 추가함

▣ 람다 함수의 활용

1. map 내장 함수

•map과 람다 함수를 동시에 사용하는 코드 (가장 추천하는 코드)

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

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

▪map(람다 함수, 시퀀스 자료형)
▪map과 람다함수는 궁합이 잘 맞음
▪첫 번째 원소로 들어오는 함수가 map에서만 쓰이는 경우가 많음

•range(10)의 모든 값 x에 대해 f = x * x + 4 * x + 5의 계산 결과를 리스트로 구함

```
Y = map(lambda x: x * x + 4 * x + 5, range(10))
print Y
```

```
[5, 10, 17, 26, 37, 50, 65, 82, 101, 122]
```

▪range(10) → [0, 1, 2, ……, 7, 8, 9]

▣ 람다 함수의 활용

1. map 내장 함수

•각 단어들의 길이 리스트

```
y = map(lambda x: len(x), ["Hello", "Python", "Programming"])
print y
```

```
[5, 6, 11]
```

▪두 번째 인자의 자료형이 리스트이므로 결과 값도 리스트
▪리스트의 원소가 3개 이므로 결과 값의 원소도 3개
▪두 번째 인자의 자료형이 리스트이므로 결과 값도 리스트

▣ 람다 함수의 활용

2. filter 내장 함수

•filter(function, seq)
- seq 시퀀스 자료형이 지닌 각 원소값들에 대해 function에 적용한 결과가 참인 원소값들만을 동일 시퀀스 자료형으로 반환한다.

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

▪filter 함수 → 조건에 따라 값을 걸러냄
▪첫 번째 인자의 함수를 충족하는 값만 반환

•위 코드는 아래와 동일하다.

```
y = []
for x in [1, 2, 3, 34]:
    if x > 2:
        y.append(x)
print y
```

```
[3, 34]
```

▣ 람다 함수의 활용

2. filter 내장 함수

•주어진 시퀀스 내에 있는 정수중 홀수만 필터링

```
print filter(lambda x: x % 2, [1, 2, 3, 4, 5, 6])
```

▪시퀀스 내 존재하는 정수 중 홀수만 filtering 되는 내용

•주어진 시퀀스 내에 있는 정수중 짝수만 필터링

```
print filter(lambda x: x % 2 - 1, [1, 2, 3, 4, 5, 6])
```

▪1 % 2 = 나머지가 1 → true
▪2 % 2 = 나머지가 0 → false

•특정 범위에 있는 정수만 필터링

```
def F():
    x = 1
    print filter(lambda a: a > x, range(-5, 5))

F()
```

```
[2, 3, 4]
```

▪range(-5, 5) → [-5, -4, -3, ……, 2, 3, 4]

▣ 람다 함수의 활용

2. filter 내장 함수

•filter의 결과는 주어진 seq 자료형과 동일함

```
print filter(lambda x: x > 2, [1, 2, 3, 34])
print filter(lambda x: x > 2, (1, 2, 3, 34))
print filter(lambda x: x < 'a', 'abcABCdefDEF')
```

```
[3, 34]
(3, 34)
ABCDEF
```

▪주어진 시퀀스 자료형과 동일한 형태로 결과 반환

▣ 람다 함수의 활용

3. reduce 내장 함수

•reduce (function, seq[, initial])
- seq 시퀀스 자료형이 지닌 원소값들에 대해 function 함수를 적용하면서 하나의 값으로 매핑한다.
- 첫 번째 인자인 function 함수는 반드시 두 개의 인자 (예를 들어, x, y)를 받아야 한다.
  ◦seq 시퀀스 자료형의 각 원소값들은 각 단계별로 y에 순차적으로 들어간다.
  ◦함수가 수행된 값은 각 단계별로 x에 순차적으로 들어간다.
- 추가적으로 제공가능한 세번째 인자인 initial은 첫번째 단계에 x에 할당할 초기값으로 사용된다.

```
print reduce(lambda x, y: x + y, [1, 2, 3, 4, 5])
```

```
15
```

▣ 람다 함수의 활용

3. reduce 내장 함수

단계	x	y	reduce
1	0	1	1
2	1	2	3
3	3	3	6
4	6	4	10
5	10	5	15

▪reduce는 값이 하나로 반환됨
▪reduce(람다함수, 시퀀스자료형)
▪[, initial] → optional 한 부분
▪reduce는 람다함수의 인자가 반드시 2개여야 함
▪시퀀스 자료형의 내용이 들어가는 인자는 람다함수의 두 번째 인자
▪x + y 수행 시 맨 처음 y에 1 들어가면 x는 0 들어감
▪y에 2가 들어가면 x는 직전 계산 결과 1+0 =1이 들어감
▪1 + 2 = 3
▪y에 3이 들어가면 x는 직전 계산 결과 3이 들어감
▪y에 들어가는 값은 시퀀스 자료형의 순서대로 들어감
▪x에 들어가는 값은 초기에는 0, 그 다음부터는 중간 reduce 값 들어감
▪결과적으로 1 + 2 + 3 + 4 + 5에 해당하는 값 출력
▪reduce는 주어진 시퀀스 자료형을 하나의 값으로 축약시키는 것
▪축약시키는 룰을 람다함수로 정의

▣ 람다 함수의 활용

3. reduce 내장 함수

•initial 값 할당

```
print reduce(lambda x, y: x + y, [1, 2, 3, 4, 5], 100)
```

```
115
```

▪100라는 것은 첫 번째 x 값에 들어감
▪initial, 즉 초기값을 지정해 주는 것

•1부터 10까지 각 수에 대한 제곱값을 모두 더한 결과 구함

```
print reduce(lambda x, y: x + y * y, range(1, 11), 0)
```

```
385
```

▪range(1, 11) → [1, 2, 3, ..., 9, 10]
▪1부터 10까지 각 수에 대한 제곱값 = y * y
▪모두 더한 결과 → x +
▪1의 제곱 + 2의 제곱 + ... + 9의 제곱 + 10의 제곱 = 385

▣ 람다 함수의 활용

3. reduce 내장 함수

```
x = 0
for y in range(1, 11):
    x = x + y * y
print x
```

```
385
```

▪reduce와 lamda 함수 같이 사용하면 코드량 현저히 줄어듬

•문자열 순서 뒤집기

```
print reduce(lambda x, y: y + x, 'abcde')
```

```
edcba
```

단계	x	y	reduce
1	0	'a'	'a'
2	'a'	'b'	'ba'
3	'ba'	'c'	'cba'
4	'cba'	'd'	'dcba'
5	'dcba'	'e'	'edcba'

▪문자열의 각 문자가 람다함수의 두 번째 인자에 들어감
▪x에 들어가는 초기값은 공백문자
▪이 처음 값이 두 번째 수행의 x값에 들어감
▪reduce 함수는 시퀀스 자료형의 내용을 뒤집는 것도 가능