온라인평생교육원

‘단계별 실습 과제’로 배우는 파이썬 프로그래밍 학습자료

2017.10.

- **Python 2.7과 3.6의 차이(고급 함수)**

e-koreatech.ac.kr

[파이썬 2.7의 xrange와 range 특징]

- **xrange(start, stop, step)**
  : 정해진 숫자 만큼 xrange object를 생성, 제너레이터와 같은 방식으로 동작하여 next()를 호출할 때 마다 다음 순서의 값을 반환함. 비교적 큰 범위의 값을 사용할 때는 xrange를 사용하는 것이 좋음.
- **range(start, stop, step)**
  : 등간격의 정수 리스트 자체가 필요할 사용하고 시퀀셜 자료를 인덱싱할 숫자들을 만들어낼 때도 쓸모가 있으며, 매번 list를 생성함으로써 overhead가 발생하여 메모리 낭비를 초래할 수 있음.

xrange는 필요한 숫자만 생성함으로써 메모리 낭비를 줄일 수 있고 데이터가 많을 경우 성능 차이는 크게 발생함.

Python 2.7	Python 3.6
xrange와 range가 있음	xrange가 없어졌음.

test

```
1  for x in range(10):
2      print x,
3  print
4  for x in xrange(10):
5      print x,
6  print
7  for x in xrange(3,10):
8      print x,
9  print
10 for x in xrange(3,10,2):
11     print x,
```

<terminated> test.py [C:₩Python27₩python.exe]

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
3 5 7 9
```

test

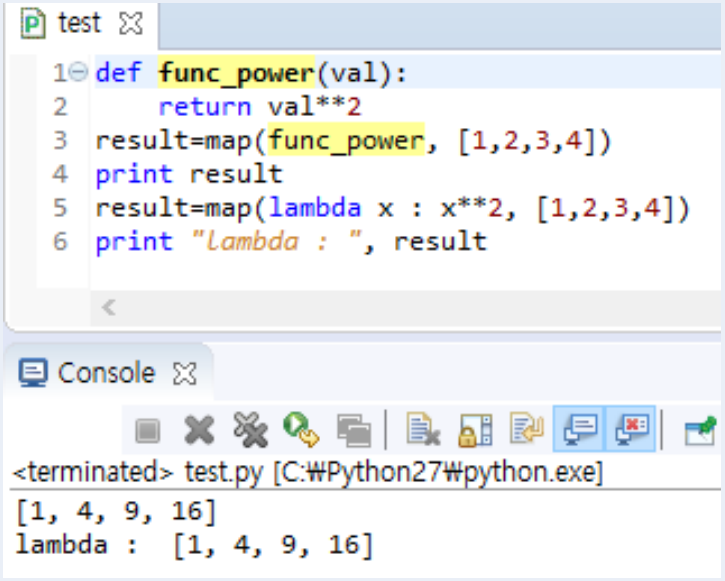
```
1  for x in list(range(10)):
2      print (x,end=" ")
3  print ("")
4  for x in range(10):
5      print (x,end=" ")
6  print ("")
7  for x in range(3,10):
8      print (x,end=" ")
9  print ("")
10 for x in range(3,10,2):
11     print (x,end=" ")
```

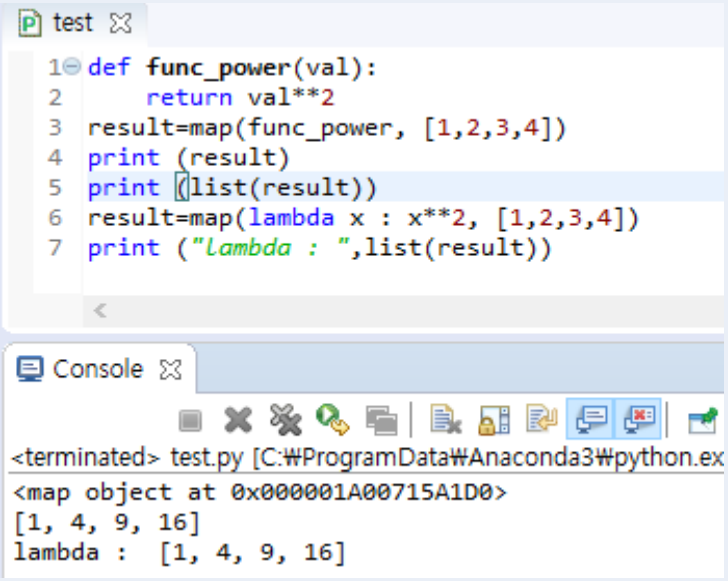
<terminated> test.py [C:₩ProgramData₩Anaconda3₩

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
3 4 5 6 7 8 9
3 5 7 9
```

- 변형된 값의_generator = map(값을 변환할 함수, iterable)
- 각 원소마다 지정함수가 호출되어, 리턴값으로 구성된 새로운 리스트를 리턴.

Python 2.7	Python 3.6
	Map의 결과값을 출력하기 위해서는 list함수를 사용하여야 함.



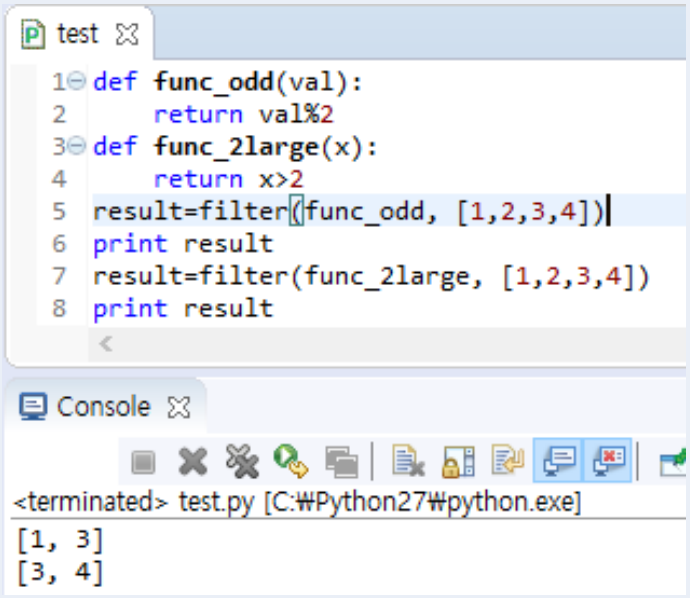


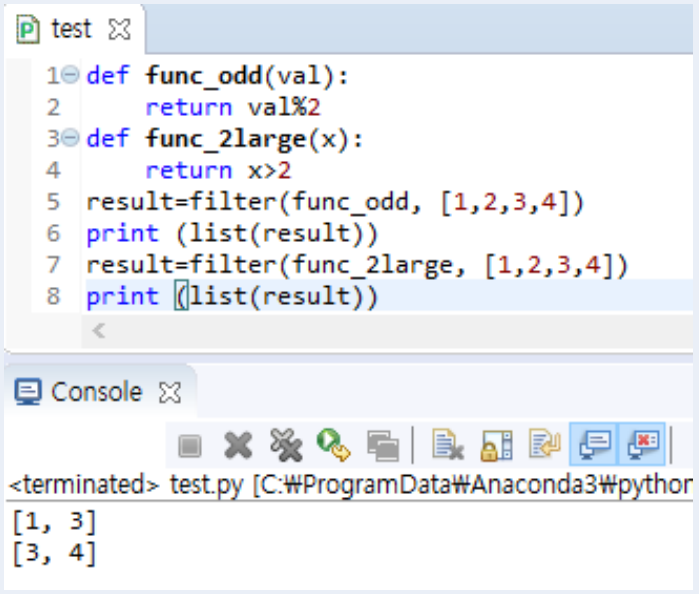
▣ lambda : 이름이 명시되지 않은 익명함수로, 함수를 한줄로 표현해 줌.
lambda 인자 : 표현식 ⇒ def 함수명(인자): return 표현식
[실행] (lambda 인자 : 표현식)(실인자) ⇒ 함수명(실인자)

Filter 함수

- 변형된 값의_generator = filter(값을 변환할 함수, iterable)
- 각 원소마다 지정함수가 호출되어, 리턴값이 참인 것으로 구성된 새로운 리스트를 리턴.
  (리턴값이 0이면 false, 그렇지않으면 true)

Python 2.7	Python 3.6
	filter의 결과값을 출력하기 위해서는 list함수를 사용하여야 함.





Exe 함수 : 문자열 수식 실행 함수

e-koreatech

Python 2.7	Python 3.6	실행 차이점

test

```
1 def execute(a, st):
2     b = 42
3     exec("b = {}\nprint 'b:', b".format(st))
4     print b                          # This prints
5 a = 1.
6 execute(a, "1.E6*a")
```

Console

```
<terminated> test.py [C:₩Python27₩python.exe]
b: 1000000.0
1000000.0
```

test

```
1 def execute(a, st):
2     b = 42
3     exec("b = {}\nprint('b:', b)".format(st))
4     print(b)                         # This prints
5 a = 1.
6 execute(a, "1.E6*a")
```

Console

```
<terminated> test.py [C:₩ProgramData₩Anaconda3₩python.exe]
b: 1000000.0
42
```

Python 2에서 Exec을 사용하면 컴파일러가 로컬 범위 최적화를 해제하므로 위와 같이 Python2.7과 Python3.6에서 다른 결과를 가져온다.

[Pythen 2.7에서 발생하는 문제를 해결하는 방법]
: exec 함수 호출 시 네임 스페이스를 사용함.

test

```
1 def execute(a, st):
2     b = 42
3     d = locals()
4     exec("b = {}\nprint 'b:', b".format(st), globals(), d)
5     print(b)                         # This prints 42
6     print(d['b'])                    # This prints 1000000.0
7     print(id(d) == id(locals()))     # This prints True
8 a = 1.
9 execute(a, "1.E6*a")
```

Console

```
<terminated> test.py [C:₩Python27₩python.exe]
b: 1000000.0
42
1000000.0
True
```