람다

def 키워드를 사용하지 않고, 식 형식으로 되어있다고 해서 람다(표현)식이라 부른다. 이름이 없으므로 익명함수라고 부르기도 한다. 재사용되지 않는 1회성 함수를 만들때 사용한다.

형식

```
#람다식 정의
변수명 = lambda 매개변수1, 매개변수2 : 실행문장

#람다식 호출
변수명(인자1, 인자2)
```

람다식 자체를 호출하기

람다식을 변수에 할당하지 않고 괄호를 이용해서 식 자체를 바로 호출할 수 있다.

형식

```
(lambda 매개변수1, 매개변수2 : 실행문장)(인자1, 인자2)
```

map

Input데이터를 동일 함수에 반복적으로 적용시켜 주는 역할을 한다.
for문과 같은 반복문을 사용하지 않아도 지정한 함수로 인수를 여러번 전달해서 그 결과를 list형태로 나타내는 유용한 함수이다.

형식

```
map(람다식, 파라미터)
```

filter

반복 가능한 객체에서 특정조건에 맞는 요소만 가져오는데, filter에 지정한 함수의 반환값이 True일 때만 해당 요소를 가져온다.

형식

```
filter(람다식, 반복가능한객체)
```

reduce

반복 가능한 객체의 각 요소를 지정된 함수로 처리한 뒤 이전 결과와 누적해서 반환한다.
파이썬 3부터 내장 함수가 아니므로, functools 모듈에서 reduce 함수를 가져와야 한다.

형식

> functools.reduce(람다식, 반복가능한객체)

예제] 12lambda.py

```
def two_sum(x, y):
    return x + y
print("함수를 통합 두수의 합=", two_sum(10, 20))

sum = lambda arg1, arg2: arg1 + arg2
print("람다식을 통한 합=", sum(10, 20))

power = lambda num : num**2
print("5의 제곱은=", power(5))

print("람다식 자체호출=", (lambda x, y: x + y)(100, 200))

```

여기까지 작성하세요.

```
print("### 람다식과 map함수1 ###")
multiLambda = lambda x: x*2
list_data = [1,2,3,4,-1,-2,-5,-10]
result_list = list(map(multiLambda, list_data))
print('result_list', result_list)

print("### 람다식과 map함수2 ###")
list_data2 = [1,2,3,4,5,6,7,8,9,10]
strNumLambda = lambda x: str(x) if x%3==0 else x
result_list2 = list(map(strNumLambda, list_data2))
print('result_list2', result_list2)

```

여기까지 작성하세요.

```
print("### 람다식과 filter함수 ###")
powLambda = lambda y : y**2
list_data3 = [1,4,9,16,25,46,64,81,100]
result_list3 = list(map(powLambda, list_data3))
print('result_list2', result_list3)

filter_result = list(filter(lambda z: z>50 and z<1000, result_list3))
print('filter_result', filter_result)
```

```
print("### 람다식과 reduce함수 ###")
import functools, operator

sum1 = functools.reduce(lambda i, j: i + j, range(1,11))
print("sum1=", sum1)

sum2 = functools.reduce(operator.add, range(1,11))
print("sum2=", sum2)
```