파이썬 스타일 코드2 - 연습해보기

--

실습코드

1. 람다함수

- 람다(lambda) 함수는 함수의 이름 없이, 함수처럼 사용할 수 있는 익명의 함수를 말한다. 선형대수나 미적분 등의 과목을 수강하다 보면, 한 번쯤 람다 대수라는 표현을 들어 보았을 것이다. 람다 함수의 '람다'는 바로 이 람다 대수에서 유래하였다. 일반적으로 람다 함수는 이름을 지정하지 않아도 사용할 수 있다.

1.1. 기존 함수

In [30]:
```
def f(x,y): #x,y의 합을 구하는 함수 작성
    return x + y

print(f(1,4)) #x,y값을 입력하여 함수 출력
```

5

1.2. lambda 함수 할당

In [31]:
```
f=lambda x,y: x + y #x,y합을 구하는 함수작성
print(f(1,4))
```

5

1.3. 익명의 lambda 함수

In [32]:
```
print((lambda x, y:x + y)(1, 4)) #변수명이 나타나지 않지만 lamda함수를 사용하여 합을
```

5

2. 맵리듀스

2.1. map 함수

연속 데이터를 저장하는 시퀀스 자료형에서 요소마다 같은 기능을 적용할 때 사용한다. 일반적으로 리스트나 튜플처럼 요소가 있는 시퀀스 자료형에 사용된다. 다음의 사용 예제를 보자.

In [5]:
```
ex = [1,2,3,4,5]
f = lambda x:x**2
print(list(map(f, ex))) #각 리스트값을 제곱하는 함수 생성
```

[1, 4, 9, 16, 25]

- 코드 설명

- 위 코드에서는 먼저 ex라는 이름의 리스트를 만들고, 입력된 값을 제곱하는 람다함수 f를 생성하였다. 그리고 'map(함수이름, 리스트 데이터)'의 구조에서 map(f,ex) 코드를 실행한다. 이는 해당 코드로 함수 f를 ex의 각 요소에 매핑하라는 뜻이다.
- 파이썬 2.x와 3.x의 차이는 제너레이터의 사용인데 3.x 부터는 map()함수의 기본 반환이 제너레이터이므로 list() 함수를 사용해야 리스트로 반환된다.
- 제너레이터(generator)는 시퀀스 자료형의 데이터를 처리할 때, 실행 시점의 값을 생성하여 효율적으로 메모리를 관리할 수 있다는 장점이 있따.
- 만일 list를 붙이지 않는다면, 다음 코드처럼 코딩할 수도 있다. 여기서 함수는 반드시 람다함수일 필요는 없고, 일반 함수를 만들어 사용해도 문제 없다.

In [6]:
```
ex=[1,2,3,4,5]
f=lambda x:x**2
for value in map(f,ex):
    print(value)
```

```
1
4
9
16
25
```

- 리스트 컴프리헨션과 비교

최근에는 람다함수나 map() 함수를 프로그램 개발에 사용하는 것을 권장하지 않는다. 굳이 두 함수를 쓰지 않더라도 리스트 컴프리헨션 기법으로 얼마든지 같은 효과를 낼 수 있기 때문이다. 만약 위 코드를 리스트 컴프리헨션으로 변경하고자 한다면, 다음처럼 코딩하면 된다.

In [7]:
```
ex = [1, 2, 3, 4, 5]
[x**2 for x in ex]
```

Out[7]: [1, 4, 9, 16, 25]

- 한개 이상의 시퀀스 자료형 데이터의 처리

map()함수는 2개 이상의 시퀀스 자료형 데이터를 처리하는 데도 문제가 없다.

In [8]:
```
ex=[1,2,3,4,5]
f=lambda x,y:x+y
list(map(f,ex,ex))
```

Out[8]: [2, 4, 6, 8, 10]

In [9]:
```
[x+y for x,y in zip(ex,ex)] # 리스트 컴프리헨션 용법
```

Out[9]: [2, 4, 6, 8, 10]

2.2. reduce 함수

- map() 함수와 다르지만 형제처럼 사용하는 함수로 리스트와 같은 시쿼스 자료형에 차례대로 함수를 적용하여 모든 값을 통합하는 함수이다.

- lambda 함수와 함께 쓰여 좀 복잡해 보여 예전에는 많이 쓰였으나 최근 버전에서는 사용을 권장하지 않는다. 그러나 많은 코드들이 여전히 사용하고 있어 이해차원에서 배울 필요가 있다.

In [33]:
```
from functools import reduce #reduce를 사용하여 차례로 값을 구한다
print(reduce(lambda x,y:x+y, [1,2,3,4,5]))#lambda에 지정된 대로 합을 구한다.
```

```
15
```

- 비교 코드

In [34]:
```
x=0
for y in [1,2,3,4,5]: #x를 0으로 설정 후 y를 차례로 x에 할당하여 합을 구한다.
    x += y
print(x)
```

```
15
```

3. 별표의 활용

3.1. 가변 인수로 활용

- 가변 인수

In [36]:
```
def asterisk_test(a, *args): #여러개의 인자를 받기 위해 *args사용
    print(a,args)
    print(type(args)) #*args는 튜플형태로 출력됨

asterisk_test(1,2,3,4,5,6)
```

```
1 (2, 3, 4, 5, 6)
<class 'tuple'>
```

- 키워드 가변 인수

In [37]:
```
def asterisk_test(a,**kargs): #**kwargs는 키워드 형태로 함수를 호출할 수 있다.
    print(a,kargs)
    print(type(kargs))
asterisk_test(1,b=2,c=3,d=4,e=5,f=6) #*args와 다르게 딕셔너리 형태로 출력된다
```

```
1 {'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
<class 'dict'>
```

3.2. 별표의 언패킹 기능

- 함수에서의 사용

In [38]:
```
def asterisk_test(a,args): #함수에서 *를 사용해 언패킹 기능을 사용할 수 있다.
    print(a,*args)
    print(type(args))
asterisk_test(1,(2,3,4,5,6))
```

```
1 2 3 4 5 6
<class 'tuple'>
```

In [15]:
```
```

```
def asterisk_test(a,args):
    print(a,args)
    print(type(args))
asterisk_test(1,(2,3,4,5,6))
```

```
1 (2, 3, 4, 5, 6)
<class 'tuple'>
```

- 일반 자료형에서의 사용

In [39]:
```
a,b,c=([1,2], [3,4], [5,6])#*를 사용하여 a,b,c와 같은 값을 출력할 수 있다.
print(a,b,c)
data=([1,2], [3,4], [5,6])
print(*data)
```

```
[1, 2] [3, 4] [5, 6]
[1, 2] [3, 4] [5, 6]
```

- zip 함수와의 응용

In [40]:
```
for data in zip(*[[1,2],[3,4],[5,6]]):#변수 대신 리스트 앞에 *를 사용해도 같음
    print(data)
    print(type(data))
```

```
(1, 3, 5)
<class 'tuple'>
(2, 4, 6)
<class 'tuple'>
```

- 키워드 가변 인수 응용

In [41]:
```
def asterisk_test(a,b,c,d):
    print(a,b,c,d)
data={"b":1, "c":2, "d":3}
asterisk_test(10, **data) #가변인수에 사용할 수 있다.인수의 수가 정해지지 않은 경우 사
```

```
10 1 2 3
```

4. 선형대수학

4.1. 파이썬 스타일 코드로 표현한 벡터

In [20]:
```
vector_a=[1,2,10]  # 리스트로 표현한 경우
vector_b=(1,2,10)   # 튜플로 표현한 경우
vector_c={'x':1, 'y':2, 'z':10}  # 딕셔너리로 표현한 경우
```

- 벡터의 연산: 벡터합

[2,2]+[2,3]+[3,5]=[7,10]

In [42]:
```
u=[2,2]
v=[2,3]
z=[3,5]
result=[]

for i in range(len(u)):#각 벡터의 합을 구하는 식
```

```
    result.append(u[i]+v[i]+z[i])
print(result)
```

```
[7, 10]
```

In [22]:
```
u=[2,2]
v=[2,3]
z=[3,5]
result=[sum(t) for t in zip(u,v,z)]
print(result)
```

```
[7, 10]
```

- 별표를 사용한 함수화

In [23]:
```
def vector_addition(*args):
    return [sum(t) for t in zip(*args)]   # unpacking 통해 zip(u,v,z) 효과를 낼 수 있

vector_addition(u,v,z)
```

Out[23]: [7, 10]

- 간단한 두벡터의 합

In [24]:
```
a = [1, 1]
b = [2, 2]

[x + y for x, y in zip(a, b)]
```

Out[24]: [3, 3]

- 벡터의 연산: 스칼라곱

2([1,2,3]+[4,4,4])=2[5,6,7]=[10,12,14]

In [25]:
```
u=[1,2,3]
v=[4,4,4]

alpha=2

result=[alpha*sum(t) for t in zip(u,v)]
result
```

Out[25]: [10, 12, 14]

4.2. 파이썬 스타일코드로 표현한 행렬

- 딕셔너리로 표현하는 경우 좌표정보나 이름정보를 넣을 수 있으나 복잡함

In [26]:
```
matrix_a=[[3,6], [4,5]] #리스트로 표현한 경우
matrix_b=[(3,6), (4,5)] #튜플로 표현한 경우
matrix_c={(0,0):3, (0,1):6, (1,0):4, (1,1):5}  #디셔너리로 표현한경우
```

- 행렬의 연산: 행렬의 elemnet-wise 합

In [27]:
```
matrix_a=[[3,6], [4,5]]
matrix_b=[[5,8], [6,7]]

result=[[sum(row) for row in zip(*t)] for t in zip(matrix_a, matrix_b)]
print(result)
```

```
[[8, 14], [10, 12]]
```

일반문제

주민등록번호로 성별 찾기 with map

PR6에서 split을 활용하여 주민등록번호 뒷자리의 맨 첫 번째 숫자를 추출하여 성별을 알아내는 과정을 구현하였다. 이번에는 여러개의 요소를 가지는 다음과 같은 리스트에서 성별을 찾는 과정을 맵리듀스를 이용해 간단하게 구현해보자.

pins = ["891120-1234567", "931120-2335567", "911120-1234234", "951120-1234567"]

Q: lambda와 map을 사용하여 위의 리스트에서 출력결과 예시와 같이 성별을 나타내는 값을 추출하시오.

출력결과 예시

CODE

['1', '2', '1', '1']

HINT

1. lambda 함수로 주민등록번호 문자열에서 성별을 추출하는 과정을 구현한다.
2. map 함수에 해당 lambda함수와 주민등록번호 리스트를 입력한다.
3. 실습코드 2.1.에서 map 과 lambda를 어떻게 함께 사용하는지와 결과를 list로 출력하는 과정을 참고하세요.

In [43]:
```
pins = ["891120-1234567", "931120-2335567", "911120-1234234", "951120-1234567"]


list(map(lambda x: x.split("-")[1][0], pins))# "-"를 기준으로 문자열을 나눔,pins의 첫
```

Out[43]:
```
['1', '2', '1', '1']
```

도전문제

벡터의 내적

크기가 같은 두 벡터의 내적은 벡터의 각 성분별 곱한 값을 더해준 값이다.

Q: 크기가 같은 두 벡터 (list 형태)를 받으면 이를 내적한 값을 도출하는 함수 dot을 구현하고, 이를 활용하여 a=[1, 2], b=[3,4]를 내적한 값을 구하시오.

HINT

1. 실습코드 4.1.에서는 벡터의 합과 곱에 대한 연산만을 다루고 있습니다. 이중 간단한 벡터의 합에서 리스트 컴프리헨션을 사용한 방법에서 연산을 바꾸면 각 벡터별 곱을 간단히 구할 수 있습니다.

In [29]:
```
a = [1, 2]
b = [3, 4]

dot = lambda a,b : sum([x*y for x, y in zip(a, b)])

dot(a,b)
```

Out[29]: 11