

Chapter 08

파이썬 스타일 코드 I



목차

1. 파이썬 스타일 코드의 이해 *Pythonic Code : 가독성 ↑, 간결성 ↑*
2. 문자열의 분리 및 결합 *split(), join()*
3. 리스트 컴프리헨션 *List comprehension : 기존 for 문보다 간결하게 코드 생성*
4. 다양한 방식의 리스트값 출력 *enumerate(), zip()*

학습목표

- 파이썬 스타일 코드를 사용하는 이유를 알아본다.
- 문자열의 분리 함수 `split()` 과 결합 함수 `join()` 에 대해 학습한다.
- 기존 리스트형을 사용하여 간단하게 새로운 리스트를 만드는 리스트 컴프리헨션에 대해 알아본다.
- 다양한 방식으로 리스트값을 출력하기 위해 `enumerate()` 함수와 `zip()` 함수에 대해 학습한다.

01

파이썬 스타일 코드의 이해

1. 파이썬 스타일 코드의 개념

- 파이썬 스타일 코드(**pythonic code**): 파이썬 스타일의 코드 작성 기법
- for문을 사용하여 단어들을 연결하여 출력하는 가장 일반적인 코드

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''
>>> for s in colors:
...     result += s
...
>>> print(result)
redbluegreenyellow
```

5

1. 파이썬 스타일 코드의 개념

- 파이썬에서의 코딩 방법

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''.join(colors)
>>> print(result)
redbluegreenyellow
```

- 리스트의 내부 함수인 join()을 활용하여 리스트의 각 요소를 빈칸 없이 연결함
- 특별한 문법이 아니라 파이썬에서 기본적으로 제공하는 문법들을 활용하여 코딩하는 것이 파이썬 스타일 코드임
- 딕셔너리나 collections 모듈도 대표적인 파이썬 스타일 코드의 하나

6

2. 파이썬 스타일 코드를 사용하는 이유

- 파이썬의 철학 : '인간의 시간이 컴퓨터의 시간보다 더 중요하다.'
- 다양한 파이썬 스타일 코드: split(), join(), 리스트 컴프리헨션(list comprehension), enumerate(), zip()과 같은 기본적인 개념부터 map()과 reduce()와 같은 상위 개념까지 포함함.
- 파이썬 스타일 코드는 왜 배워야 할까?
 - 파이썬으로 작성된 프로그램 대부분은 파이썬 스타일 코드의 특징을 포함하므로 파이썬 스타일 코드를 알아야 다른 사람이 작성 한 코드를 쉽게 이해할 수 있음
 - 효율적인 프로그래밍 측면: 코드 자체도 간결해지고 코드 작성 시간도 단축시킴

7

02 문자열의 분리 및 결합

1. 문자열의 분리: split() 함수

- split() 함수: ~~특정 값~~을 기준으로 문자열을 분리하여 리스트 형태로 변환
특정 구분자 (separator)

```
>>> items = 'zero one two three'.split() # 빈칸을 기준으로 문자열 분리하기
>>> print(items)
['zero', 'one', 'two', 'three']
```

default : " "

- 문자열 'zero one two three'를 split() 함수를 사용하여 리스트형의 변수로 변환함
- split() 함수 안에 매개변수로 아무것도 입력하지 않으면 빈칸을 기준으로 문자열을 분리하라는 뜻임

9

1. 문자열의 분리: split() 함수

```
>>> example = 'python,jquery,javascript'
>>> example.split(",") # comma 기준
['python', 'jquery', 'javascript']
>>> a, b, c = example.split(",")
>>> print(a, b, c)
python jquery javascript
>>> example = 'theteamlab.univ.edu'
>>> subdomain, domain, tld = example.split('.') # period (.) 기준
>>> print(subdomain, domain, tld)
theteamlab univ edu
```

- 첫 번째 줄에서 문자열이 콤마(,)를 기준으로 묶여 있음. split(",")를 사용하면 콤마를 기준으로 문자열을 분리할 수 있음.
- split(".",)을 사용하면 점(.)을 기준으로 분리한 split() 함수는 a, b, c = example.split(",")와 같이 결과값을 바로 언패킹하여 사용할 수도 있음.
- theteamlab.univ.edu와 같은 도메인 네임을 의미별로 분리할 때도 split() 함수 이용.

10

2. 문자열의 결합: join() 함수

- join() 함수: 문자열로 구성된 리스트를 합쳐 하나의 문자열로 만들 때 사용
- join() 함수 형태: 구분자.join(리스트형)

```
>>> colors = ['red', 'blue', 'green', 'yellow']
>>> result = ''.join(colors)
>>> result
'redbluegreenyellow'
```

11

2. 문자열의 결합: join() 함수

- 다양한 구분자를 삽입한 예

```
>>> result = ' '.join(colors) # 연결 시, 1칸을 띄고 연결
>>> result
'red blue green yellow'
>>> result = ', '.join(colors) # 연결 시 ", "로 연결
>>> result
'red, blue, green, yellow'
>>> result = '-'.join(colors) # 연결 시 "-"로 연결
>>> result
'red-blue-green-yellow'
```

12

03 리스트 컴프리헨션

1. 리스트 컴프리헨션 다루기

- **리스트 컴프리헨션(list comprehension):** 기존 리스트형을 사용하여 간단하게 새로운 리스트를 만드는 기법

- 일반적인 반복문 + 리스트

```
>>> result = [ ]
>>> for i in range(10):
...     result.append(i)
...
>>> result
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- 리스트 컴프리헨션

```
>>> result = [i for i in range(10)]
>>> result
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

(Handwritten notes: 'range(10)' is underlined, 'i' is circled, and 'range(10)' is labeled '값도 추가' and '값은 생성')

- 리스트 컴프리헨션 코드가 훨씬 더 간결하다는 것을 알 수 있음.

14

1. 리스트 컴프리헨션 다루기

여기서 잠깐! 리스트 컴프리헨션 용어

리스트 컴프리헨션(list comprehension)의 우리말 표기는 다양하다. 'comprehension' 자체가 '포괄하는'이라는 의미가 있어 '포괄형 리스트'나 '포함형 리스트'라고도 한다. 어떤 책에서는 '지능형 리스트'나 '축약형 리스트'라고도 한다. 하나의 리스트에 다른 리스트를 포함시켜 생성할 수 있다는 측면에서 '리스트를 포괄한다.'라는 의미로 이해하면 된다. 이 책에서는 적당한 번역 표현이 없다고 생각하여 원어 그대로 리스트 컴프리헨션이라고 부르도록 한다.

15

2. 리스트 컴프리헨션의 사용법

2.1 필터링

- if문과 함께 사용하는 리스트 컴프리헨션
- 짝수만 저장하기 위한 코드 작성

- 일반적인 반복문 + 리스트

```
>>> result = [ ]
>>> for i in range(10):
...     if i % 2 == 0:
...         result.append(i)
...
>>> result
[0, 2, 4, 6, 8]
```

(Handwritten notes: 'if i % 2 == 0:' is underlined, and 'i' is circled)

- 리스트 컴프리헨션

```
>>> result = [i for i in range(10) if i % 2 == 0]
>>> result
[0, 2, 4, 6, 8]
```

(Handwritten notes: 'range(10)' is underlined, 'i' is circled, and 'if i % 2 == 0' is labeled '필터링 (조건 추가)')

16

2. 리스트 컴프리헨션의 사용법

- else문과 함께 사용하여 해당 조건을 만족하지 않을 때는 다른 값을 할당할 수 있음.

```
>>> result = [i if i % 2 == 0 else 10 for i in range(10)]
>>> result
[0, 10, 2, 10, 4, 10, 6, 10, 8, 10]
```

- 이 코드처럼 if문을 앞으로 옮겨 else문과 함께 사용해도 되는데, 조건을 만족하지 않을 때 else 다음에 있는 i값을 할당하라는 코드임.

(if-else) 반드시 for 앞에 사용
(필터링) for 뒤에 사용

2. 리스트 컴프리헨션의 사용법

2.2 중첩 반복문(nested loop)

- 리스트 컴프리헨션에서도 리스트 2개를 섞어 사용할 수 있음.

```
>>> word_1 = "Hello"
>>> word_2 = "World"
>>> result = [i + j for i in word_1 for j in word_2] # 중첩 반복문
>>> result
['HW', 'Ho', 'Hr', 'Hl', 'Hd', 'eW', 'eo', 'er', 'el', 'ed', 'lW', 'lo', 'lr', 'll', 'ld', 'lW', 'lo', 'lr', 'll', 'ld', 'oW', 'oo', 'or', 'ol', 'od']
```

- word_1에 있는 값을 먼저 고정시킨 후, word_2의 값을 하나씩 가져와 결과를 생성

```
>>> result = [i for i in range(10) if i % 2 == 0]
>>> result
[0, 2, 4, 6, 8]
```

17

18

2. 리스트 컴프리헨션의 사용법

2.3 이차원 리스트(two-dimensional list)

- 하나의 정보를 열(row) 단위로 저장함.
- 이차원 리스트를 만드는 가장 간단한 방법은 대괄호 2개를 사용하는 것.

```
>>> words = 'The quick brown fox jumps over the lazy dog'.split()
>>> print(words)
['The', 'quick', 'brown', 'fox', 'jumps', 'over', 'the', 'lazy', 'dog']
>>> stuff = [[w.upper(), w.lower(), len(w)] for w in words]
# 리스트의 각 요소를 대문자, 소문자, 길이로 변환하여 이차원 리스트로 변환
```

- 이 코드는 기존 문장을 split() 함수로 분리하여 리스트로 변환한 후 각 단어의 대문자, 소문자, 길이를 하나의 리스트로 각각 저장하는 방식임.

[['THE', 'the', 3],
 ['QUICK', 'quick', 5],
 ['BROWN', 'brown', 5],
 ['FOX', 'fox', 3],
 ['JUMPS', 'jumps', 5],
 ['OVER', 'over', 4],
 ['THE', 'the', 3],
 ['LAZY', 'lazy', 4],
 ['DOG', 'dog', 3]]

19

2. 리스트 컴프리헨션의 사용법

- 저장한 후 결과를 출력하면 다음과 같이 이차원 리스트를 생성할 수 있음.

```
>>> for i in stuff:
...     print(i)
...
['THE', 'the', 3]
['QUICK', 'quick', 5]
['BROWN', 'brown', 5]
['FOX', 'fox', 3]
['JUMPS', 'jumps', 5]
['OVER', 'over', 4]
['THE', 'the', 3]
['LAZY', 'lazy', 4]
['DOG', 'dog', 3]
```

20

2. 리스트 컴프리헨션의 사용법

- for문 2개를 붙여 사용할 수도 있음

```
>>> case_1 = ["A", "B", "C"]
>>> case_2 = ["D", "E", "A"]
>>> result = [i + j for i in case_1 for j in case_2]
>>> result
['AD', 'AE', 'AA', 'BD', 'BE', 'BA', 'CD', 'CE', 'CA']
```

- 이전 코드와 달리 리스트 안에 `[i + j for i in case_1]`이 하나 더 존재함
- 따라서 먼저 나온 for문이 고정되는 것이 아니라 뒤의 for문 이 고정됨.
- A부터 고정되는 것이 아니라 case_2의 첫 번째 요소인 D가 고정되고 A, B, C가 차례로 D 앞에 붙어서 결과를 보면 이차원 리스트 형태로 출력됨..

```
>>> result = [[ i + j for i in case_1] for j in case_2]
>>> result
[['AD', 'BD', 'CD'], ['AE', 'BE', 'CE'], ['AA', 'BA', 'CA']]
```

21

2. 리스트 컴프리헨션의 사용법

- 정리! 다음 두 코드의 차이점을 꼭 구분하자.

```
① [i + j for i in case_1 for j in case_2]
② [[i + j for i in case_1] for j in case_2]
```

- ① 코드는 일차원 리스트를 만드는 코드로 앞의 for문이 먼저 실행됨.
- ② 코드는 이차원 리스트를 만드는 코드로 뒤의 for문이 먼저 실행됨.

22

3. 리스트 컴프리헨션의 성능

- 리스트 컴프리헨션은 문법적 간단함의 장점 외에도 성능이 뛰어남.
- [코드 8-1]과 [코드 8-2]를 실행하여 결과 비교하기

[코드 8-1] loop.py(일반적인 반복문 + 리스트)

```
1 def sclar_vector_product(scalar, vector):
2     result = [ ]
3     for value in vector:
4         result.append(scalar * value)
5     return result
6
7 iteration_max = 10000
8
9 vector = list(range(iteration_max))
10 scalar = 2
11
12 for _ in range(iteration_max):
13     sclar_vector_product(scalar, vector)
```

23

3. 리스트 컴프리헨션의 성능

[코드 8-2]

listcomprehension.py(리스트 컴프리헨션)

```
1 iteration_max = 10000
2
3 vector = list(range(iteration_max))
4 scalar = 2
5
6 for _ in range(iteration_max):
7     [scalar * value for value in range(iteration_max)]
```

- 두 개 모두 벡터(vector)와 스칼라(scalar)의 곱셈을 실행하는 코드.
- 벡터-스칼라 곱셈은 하나의 스칼라 값이 각 벡터의 값과 곱해져 최종값을 만들.
- 스칼라는 파이썬의 정수형 또는 실수형으로, 벡터는 리스트로 치환하여 생각하면 됨

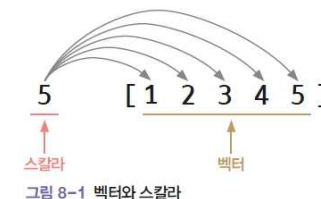


그림 8-1 벡터와 스칼라

24

3. 리스트 컴프리헨션의 성능

- [그림 8-2]는 두 코드의 성능을 비교하기 위해 리눅스 계열에서 사용하는 time 명령어의 결과를 캡처한 것

☞ 코드의 총 실행 시간을 알 수 있음.

| | | | |
|------|-----------|------|----------|
| real | 0m10.230s | real | 0m7.788s |
| user | 0m10.203s | user | 0m7.762s |
| sys | 0m0.023s | sys | 0m0.021s |

(a) 일반적인 반복문 + 리스트

(b) 리스트 컴프리헨션

그림 8-2 코드의 실행 시간을 통한 성능 비교

04 다양한 방식의 리스트값 출력

1. 리스트값에 인덱스를 붙여 출력: enumerate() 함수

- **enumerate() 함수**: 리스트의 값을 추출할 때 인덱스를 붙여 함께 출력하는 방법
인덱스 값을 같이 출력 *각각 저장한 배열*

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):  
...     print(i, v)  
...  
0 tic  
1 tac  
2 toe
```

- 리스트형의 ['tic', 'tac', 'toe']에 enumerate() 함수를 적용함.
- enumerate() 함수를 적용하면 인덱스와 리스트의 값이 연패킹되어 추출되는데, 위 코드에서는 'tic', 'tac', 'toe'에 각각 0, 1, 2의 인덱스가 붙어 출력되는 것을 알 수 있음.

1. 리스트값에 인덱스를 붙여 출력: enumerate() 함수

- 주로 딕셔너리형으로 인덱스를 키로, 단어를 값으로 하여 쌍으로 묶어 결과를 출력하는 방식을 사용함.

```
>>> {i:j for i,j in enumerate('TEAMLAB is an academic institute  
located in South Korea.'.split())}  
{0: 'TEAMLAB', 1: 'is', 2: 'an', 3: 'academic', 4: 'institute', 5:  
'located', 6: 'in', 7: 'South', 8: 'Korea.'}
```

Index ↔ 키
value ↔ 값

2. 리스트 값을 병렬로 묶어 출력: zip() 함수

- zip() 함수: 1개 이상의 리스트 값이 같은 인덱스에 있을 때 병렬로 묶는 함수

```
>>> alist = ['a1', 'a2', 'a3']
>>> blist = ['b1', 'b2', 'b3']
>>> for a, b in zip(alist, blist): # 병렬로 값을 추출
...     print(a, b)
...
a1 b1
a2 b2
a3 b3
```

정리: 두줄로 변함

- zip() 함수로 묶으면 다양한 추가 기능을 만들 수 있음. → 같은 인덱스에 있는 값끼리 더하는 것도 가능함.

```
>>> a, b, c = zip((1, 2, 3), (10, 20, 30), (100, 200, 300))
>>> print(a, b, c)
(1, 10, 100) (2, 20, 200) (3, 30, 300)
>>> [sum(x) for x in zip((1, 2, 3), (10, 20, 30), (100, 200, 300))]
[111, 222, 333]
```

29

2. 리스트 값을 병렬로 묶어 출력: zip() 함수

- enumerate() 함수와 zip() 함수를 같이 사용하는 방법

```
>>> alist = ['a1', 'a2', 'a3']
>>> blist = ['b1', 'b2', 'b3']
>>> for i, (a, b) in enumerate(zip(alist, blist)):
...     print(i, a, b) # (인덱스, alist[인덱스], blist[인덱스]) 표시
...
0 a1 b1
1 a2 b2
2 a3 b3
```

- alist와 blist를 zip() 함수로 묶고 enumerate() 함수를 적용하여 같은 인덱스의 값끼리 묶어 출력함.

30

Chapter 09 파이썬 스타일 코드 II



목차

1. 람다 함수
2. 맵리듀스
3. 별표의 활용
4. 선형대수학

학습목표

- 람다 함수를 사용하는 방식과 다양한 형태에 대해 알아본다.
- `map()` 함수와 `reduce()` 함수에 대해 학습한다.
- 별표를 사용하는 방법과 별표의 언패킹 기능에 대해 이해한다.
- 파이썬 스타일 코드로 벡터와 행렬을 표현한다

01 람다 함수

1. 람다 함수의 사용

- **람다(lambda) 함수:** 함수의 이름 없이 함수처럼 사용할 수 있는 익명의 함수

- [코드 9-1]과 [코드 9-2] 비교

*def 함수이름 ()
return _*

[코드 9-1]

function.py(일반적인 함수)

```
1 def f(x, y):  
2     return x + y  
3  
4 print(f(1, 4))
```

[실행결과]

5

1. 람다 함수의 사용

[코드 9-2]

lambda.py(람다 함수)

input *output*
1 f = lambda x, y: x + y
2 print(f(1, 4))

[실행결과]

5

- 두 코드 모두 입력된 x, y의 값을 더하여 그 결과를 반환하는 함수로 결과 값도 5로 같지만, 람다 함수는 별도의 def나 return을 입력하지 않음

- 람다 함수는 앞에는 매개변수의 이름을, 뒤에는 매개변수가 반환하는 결과값인 'x + y'를 입력함

☞ 기존의 f 함수와 구조는 같고 표현이 다름.

1. 람다 함수의 사용

여기서 잠깐! 람다 함수를 표현하는 다른 방식

람다 함수를 표현하는 또 다른 방식이 있다. 다음을 보자.

```
print((lambda x: x + 1)(5))
```

람다 함수는 함수를 선언하지 않고 익명함수로 작성 가능

람다 함수 자체는 위 코드처럼 이름 없이 사용할 수 있지만, 일반적으로 [코드 9-2]와 같이 어떤 변수에 람다 함수를 할당하여 함수와 비슷한 형태로 사용한다. 위 코드는 람다 함수에 별도의 이름을 지정하지는 않았지만, 괄호에 람다 함수를 넣고 인수(argument)로 5를 입력하였다. 이 코드의 결과는 6으로 출력된다.

37

2. 람다 함수의 다양한 형태

- 람다 함수는 다양한 형태로 사용할 수 있는데 기본적으로 함수를 선언하는 것과 완전히 같은 방식임.

```
>>> f = lambda x, y: x + y
>>> f(1, 4)
5
>>>
>>> f = lambda x: x ** 2
>>> f(3)
9
>>>
>>> f = lambda x: x / 2
>>> f(3)
1.5
>>> f(3, 5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: <lambda>() takes 1 positional argument but 2 were given
```

38

02 맵리듀스

1. map() 함수

- 맵리듀스(MapReduce):** 파이썬뿐만 아니라 빅데이터를 처리하기 위한 기본 알고리즘으로 많이 사용함.
- 맵리듀스의 종류:** map() 함수, reduce() 함수
- map() 함수:** 연속 데이터를 저장하는 시퀀스 자료형에서 요소마다 같은 기능을 적용할 때 주로 사용함.

```
>>> ex = [1, 2, 3, 4, 5]
>>> f = lambda x: x ** 2
>>> print(list(map(f, ex)))
[1, 4, 9, 16, 25]
```

함수 적용

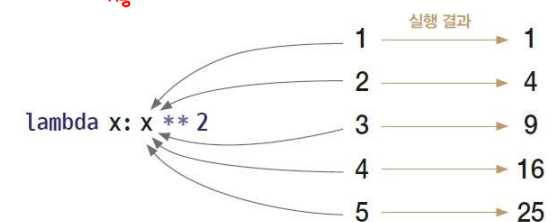


그림 9-1 map() 함수의 실행

40

1. map() 함수

1.1 제너레이터의 사용

- 파이썬 2.x와 3.x에서의 map() 함수 코드가 약간 다르다는 점 주의!

| | |
|---------|---|
| 파이썬 2.x | map(f, ex)라고만 입력해도 리스트로 반환 |
| 파이썬 3.x | <u>list(map(f, ex))</u> 처럼 <u>list</u> 를 붙여야 <u>리스트로 반환</u> |

☞ 제너레이터(generator)라는 개념이 강화되면서 생긴 추가 코드.

- 제너레이터:** 시퀀스 자료형의 데이터를 처리할 때 실행 시점의 값을 생성하여 효율적으로 메모리를 관리할 수 있다는 장점이 있음.

41

1. map() 함수

- list를 붙이지 않는다면 다음 코드처럼 코딩하는 것이 좋음.

```
>>> ex = [1, 2, 3, 4, 5]
>>> f = lambda x: x ** 2
>>> for value in map(f, ex):
...     print(value)
...
1
4
9
16
25
```

42

1. map() 함수

1.2 리스트 컴프리헨션과의 비교

- 최근에는 람다 함수나 map() 함수를 프로그램 개발에 사용하는 것을 권장하지 않음.

☞ 리스트 컴프리헨션 기법으로 얼마든지 같은 효과를 낼 수 있기 때문

- 리스트 컴프리헨션으로 변경한 코드

```
>>> ex = [1, 2, 3, 4, 5]
>>> [x ** 2 for x in ex]
[1, 4, 9, 16, 25]
```

43

1. map() 함수

1.3 한 개 이상의 시퀀스 자료형 데이터의 처리

- map() 함수는 2개 이상의 시퀀스 자료형 데이터를 처리하는 데도 문제가 없어, 여러 개의 시퀀스 자료형 데이터를 입력 값으로 사용할 수 있음

- 다음 코드를 보면 ex 변수와 같은 위치에 있는 값끼리 더한 결과가 출력됨.

```
>>> ex = [1, 2, 3, 4, 5]
>>> f = lambda x, y: x + y
>>> list(map(f, ex, ex))
[2, 4, 6, 8, 10]
```

- 리스트 컴프리헨션으로 변경한 코드

```
>>> [x + y for x, y in zip(ex, ex)]
[2, 4, 6, 8, 10]
```

44

1. map() 함수

1.4 필터링(filtering) 기능

- map() 함수는 리스트 컴프리헨션처럼 필터링 기능을 사용할 수 있음
- 리스트 컴프리헨션과 달리 else문을 반드시 작성해 해당 경우가 존재하지 않는 경우를 지정해 주어야 함.

- 짝수일 때는 각 수를 제공하고, 그렇지 않을 때는 해당 수를 그대로 출력하는 코드 작성.

```
>>> list(map(lambda x: x ** 2 if x % 2 == 0 else x, ex)) # map() 함수
[1, 4, 3, 16, 5]
>>> [x ** 2 if x % 2 == 0 else x for x in ex] # 리스트 컴프리헨션
[1, 4, 3, 16, 5]
```

45

2. reduce() 함수

- reduce() 함수: 리스트와 같은 시퀀스 자료형에 차례대로 함수를 적용한 다음 모든 값을 통합시켜 주는 함수
- map() 함수와 용법은 다르지만 형제처럼 함께 사용하는 함수임.

```
>>> from functools import reduce
>>> print(reduce(lambda x, y: x+y, [1, 2, 3, 4, 5]))
15
```

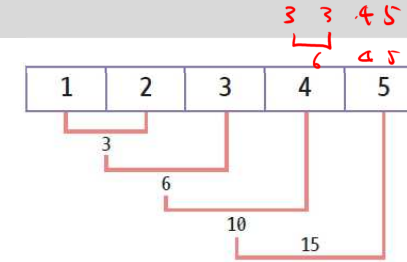


그림 9-2 reduce() 함수 실행

46

2. reduce() 함수

[코드 9-3]

```
1 x = 0
2 for y in [1, 2, 3, 4, 5]:
3     x += y
4
5 print(x)
```

[실행결과]

15

여기서 잠깐! 람다 함수와 맵리듀스의 사용

람다 함수와 맵리듀스는 파이썬 2.x 버전에서 매우 많이 사용하던 함수이다. 최근에는 그 문법의 복잡성 때문에 권장하지 않지만, 여전히 기존 코드와 새롭게 만들어지는 코드에서 많이 사용하고 있으므로 알아둘 필요가 있다.

47

03 별표의 활용

1. 별표의 사용

- 별표(asterisk): 곱하기 기호(*)를 뜻함.
- 별표는 기본 연산자로, 단순 곱셈이나 제곱 연산에 많이 사용됨.
- 별표를 사용하는 특별한 경우: 함수의 가변 인수(variable length arguments) ^{가변}를 사용할 때 변수명 앞에 별표를 붙임.

가변 인수

```
>>> 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'>
```

49

1. 별표의 사용

키워드 가변 인수

```
>>> def asterisk_test(a, **kwargs):
...     print(a, kwargs)
...     print(type(kwargs))
...
>>> asterisk_test(1, b=2, c=3, d=4, e=5, f=6)
1 {'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
<class 'dict'>
```

- 별표 한 개(*) 또는 두 개(**)를 변수명 앞에 붙여 여러 개의 변수가 함수에 한 번에 들어갈 수 있도록 처리함.
- 첫 번째 함수의 경우, 2, 3, 4, 5, 6이 변수 args에 할당됨.

50

2. 별표의 언패킹 기능

- 별표는 여러 개의 데이터를 담은 리스트, 튜플, 딕셔너리와 같은 자료형에 서는 해당 데이터를 언패킹(unpacking)하는 기능도 있음. ^{가변 인수로 사용}

```
>>> 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'>
```

51

2. 별표의 언패킹 기능

- 핵심은 `print(a, *args)` 코드.
- args는 함수에 하나의 변수로 들어갔기 때문에 일반적이라면 다음과 같이 출력되어야 함.

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

- BUT! 튜플의 값은 하나의 변수이므로 출력 시 괄호가 붙어 출력되지만 결과는 1 2 3 4 5 6 형태로 출력됨.
- 이것은 일반적으로 `print(a, b, c, d, e, f)`처럼 각각의 변수를 하나씩 따로 입력했을 때 출력되는 형식인데 이렇게 출력된 이유는 args 변수 앞에 별표가 붙었기 때문.

52

2. 별표의 언패킹 기능

```
>>> def asterisk_test(a, *args):  
...     print(a, args) #변인  
...     print(type(args))  
...  
>>> asterisk_test(1, *(2, 3, 4, 5, 6))  
1 (2, 3, 4, 5, 6) #튜플  
<class 'tuple'> 1, 2, 3, 4, 5, 6
```

- 함수 호출 시 별표가 붙은 asterisk_test(1, *(2, 3, 4, 5, 6))의 형태로 값이 입력됨. 즉, 입력 값은 뒤의 튜플 변수가 언패킹되어 다음처럼 입력된 것.

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

53

2. 별표의 언패킹 기능

- 두 코드의 형태는 다르지만 모두 기존의 튜플값을 언패킹하여 출력하는 것으로 결과는 같음.

```
>>> a, b, c = ([1, 2], [3, 4], [5, 6])  
>>> print(a, b, c)  
[1, 2] [3, 4] [5, 6]  
>>>  
>>> data = x[1, 2], [3, 4], [5, 6]x  
>>> print(*data)  
[1, 2] [3, 4] [5, 6]
```

- 별표의 언패킹 기능을 유용하게 사용하는 경우 중 하나가 zip() 함수와 함께 사용할 때임.

54

2. 별표의 언패킹 기능

- 만약 이차원 리스트에서 행(row)마다 한 학생의 수학·영어·국어 점수가 있고 이것에 대해 평균을 내고 싶다면, 2개의 for문을 사용하여 계산할 수 있지만 별표를 사용한다면 다음과 같이 하나의 for문으로도 원하는 결과를 얻을 수 있음.

```
>>> for data in zip(x[1, 2], [3, 4], [5, 6])x:  
...     print(data)  
...     print(type(data)) ↓  
  
(1, 3, 5) zip([1, 2], [3, 4], [5, 6])  
<class 'tuple'>  
(2, 4, 6)  
<class 'tuple'>
```

55

2. 별표의 언패킹 기능

- 키워드 가변 인수와 마찬가지로 두 개의 별표(**)를 사용할 경우 딕셔너리형을 언패킹함.
- 딕셔너리형인 data 변수를 언패킹하여 키워드 매개변수를 사용하는 함수에 넣는 예제

```
>>> def asterisk_test(a, b, c, d):  
...     print(a, b, c, d)  
  
>>> data = x{x"b":1, "c":2, "d":3x}  
>>> asterisk_test(10, **data)  
10 1 2 3
```

56

04 선형대수학

1. 벡터와 행렬의 개념

- 컴퓨터공학과에서의 벡터는 여러 개의 데이터를 하나의 정보로 표현한다는 관점에서 리스트와 비슷함.
- 수학에서의 벡터는 이차원 평면상에 정보를 나타내므로 값이 2개임.
- 벡터가 어떤 정보를 표현하는 방법이라는 관점에서 볼 때 3개 이상의 정보를 사용해야 하는 경우도 많음

$$\mathbb{R}^4 \ni [1, 2, -1.0, 3.14] \text{ 또는 } \mathbb{R}^4 \ni \begin{bmatrix} 1 \\ 2 \\ -1.0 \\ 3.14 \end{bmatrix}$$

그림 9-4 벡터의 표현: 3개 이상의 값

1. 벡터와 행렬의 개념

1.1 벡터(vector)

- 벡터는 '배달하다, 운반하다'의 뜻을 가진 라틴어에서 유래된 용어로, 고등학교 수학에서 어떤 정보를 표현할 때 크기와 방향을 모두 가지는 것을 벡터라고 하고, 크기만 가지는 것을 스칼라라고 부름 2개체
- 일반적으로 열 형태로 숫자를 표현하고 좌표평면에 나타냄.

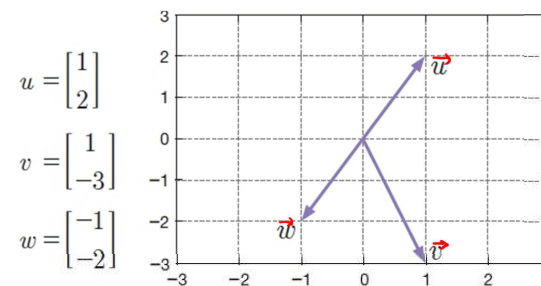


그림 9-3 벡터의 표현: 2개의 값

58

1. 벡터와 행렬의 개념

1.2 행렬(matrix)

- 행렬: 원래 격자를 뜻하는 말로, 수학에서는 사각형으로 된 수의 배열을 지칭함. 1개 이상의 벡터 모임
- 행렬에서 행 또는 열이 하나의 대상에 대한 정보를 표현한 것이며, 그 모음이 바로 행렬임. a_{ij} = i행 j열 값

$$A = \begin{matrix} \downarrow & \text{column (열)} \\ \begin{matrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{matrix} \\ \text{row (행)} \end{matrix} \quad A = \begin{bmatrix} \textcircled{1} & 2 & 3 \\ 4 & 5 & \textcircled{6} \\ \textcircled{7} & 8 & 9 \end{bmatrix} \quad \begin{matrix} a_{11} = 1 \\ a_{23} = 6 \\ a_{31} = 7 \end{matrix}$$

그림 9-5 행렬의 표현 m x n matrix 그림 9-6 행렬값의 표현

- 행렬의 구성: m개의 행과 n개의 열로 구성

2. 파이썬 스타일 코드로 표현한 벡터

- 벡터를 파이썬으로 표현하는 방법

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

- 가장 기본적인 방법은 리스트 형태로 표현하는 것. 튜플이나 딕셔너리 형태도 가능.
- 만약 각 데이터의 이름, 즉 x, y, z와 같은 정보(ex 키, 몸무게, 나이)를 함께 표현해야 한다면 딕셔너리로 표현하는 것도 좋은 방법임.
- 데이터의 위치나 순서가 바뀌지 않아야 한다면 튜플로 저장하는 것이 좋음.
- 벡터를 사용하는 목적에 따라 코드 표현은 다를 수 있으며 여기서는 기본적으로 리스트를 사용해 벡터의 연산을 실행함.

61

2. 파이썬 스타일 코드로 표현한 벡터

2.1 벡터의 연산

- 벡터의 가장 기본적인 연산: 같은 위치에 있는 값끼리 연산하는 것

$$[u_1, u_2, \dots, u_n] + [v_1, v_2, \dots, v_n] = [u_1 + v_1, u_2 + v_2, \dots, u_n + v_n]$$

(a) 공식

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

(b) 예시

그림 9-7 벡터의 연산

- [그림 9-7]의 수식을 코드로 작성

```
>>> u = [2, 2]
>>> v = [2, 3]
>>> z = [3, 5]
>>> result = [ ]
>>>
>>> for i in range(len(u)):
...     result.append(u[i] + v[i] + z[i])
...
>>> print(result)
[7, 10]
```

62

2. 파이썬 스타일 코드로 표현한 벡터

- 리스트 컴프리헨션 과 zip() 함수와 같은 파이썬 스타일 코드를 이용해 간단한 연산으로 나타낸 코드

```
>>> u = [2, 2]
>>> v = [2, 3]
>>> z = [3, 5]
>>>
>>> result = [sum(t) for t in zip(u, v, z)]
>>> print(result)
[7, 10]
```

- 코드가 훨씬 간단해짐. sum() 함수를 사용하여 zip() 함수로 묶인 튜플 t 변수의 합계를 구함. 변수 t에는 차례대로 (2, 2, 3), (2, 3, 5)가 들어감.
- 확인을 위해 다음 코드를 수행하면 t에 어떤 값이 할당되었는지 알 수 있음.

```
>>> [t for t in zip(u, v, z)]
[(2, 2, 3), (2, 3, 5)]
```

63

2. 파이썬 스타일 코드로 표현한 벡터

2.2 별표를 사용한 함수화

- 4개 이상의 변수를 사용해야 할 경우에는 어떻게 할까?

☞ 별표를 이용하여 다음과 같이 처리할 수 있음.

```
>>> def vector_addition(*args):
...     return [sum(t) for t in zip(*args)]
...
>>> vector_addition(u, v, z)
[7, 10]
```

(2, 2, 3) (2, 3, 5)

- 여전히 변수를 3개나 생성하는 문제는 어떻게 해결할 수 있을까?

☞ 이차원 리스트를 만든 후 별표의 언패킹으로 해결.

```
>>> row_vectors = [[2, 2], [2, 3], [3, 5]]
>>> vector_addition(*row_vectors)
[7, 10]
```

64

2. 파이썬 스타일 코드로 표현한 벡터

2.3 스칼라-벡터 연산

- 스칼라, 벡터: 숫자형 변수, 곱셈 연산이 가능하며 분배 법칙 적용 가능.

$$\alpha(u+v) = \alpha u + \alpha v$$

(a) 공식

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

(b) 예시

그림 9-8 스칼라-벡터 연산

- [그림 9-8]의 수식을 코드로 작성

```
>>> u = [1, 2, 3]
>>> v = [4, 4, 4]
>>> alpha = 2
>>>
>>> result = [alpha * sum(t) for t in zip(u, v)]
>>> result
[10, 12, 14]
```

65

3. 파이썬 스타일 코드로 표현한 행렬

- 행렬도 벡터와 마찬가지로 리스트, 튜플, 딕셔너리 등을 사용하여 파이썬 스타일 코드로 표현 할 수 있음.
- 행렬을 파이썬 스타일 코드로 표현하기

```
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} # 딕셔너리로 표현한 경우
```

↓
Sparse matrix
희소행렬

리스트 → 키

66

3. 파이썬 스타일 코드로 표현한 행렬

- 행렬: 이차원의 정보, 일차원의 벡터 정보를 모아 이차원 형태로 표현한 것
- 벡터의 정보 모아 표현하기

```
[[1번째 행], [2번째 행], [3번째 행]]
```

- 행렬은 딕셔너리로 표현할 때 많은 경우의 수를 나타냄.
- 행과 열의 좌표 정보를 넣을 수 있고, 이름 정보를 넣을 수도 있음.
- 여기서는 가장 일반적인 표현법인 리스트를 사용함.

67

3. 파이썬 스타일 코드로 표현한 행렬

3.1 행렬의 연산

- 행렬의 가장 기본적인 연산: 덧셈과 뺄셈
- 2개 이상의 행렬을 연산하기 위해 각 행렬의 크기는 같아야 함.
- 그 다음 인덱스가 같은 값끼리 연산이 일어남.

$$A = \begin{bmatrix} 3 & 6 \\ 4 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 8 \\ 6 & 7 \end{bmatrix}$$
$$C = A + B = \begin{bmatrix} 3 & 6 \\ 4 & 5 \end{bmatrix} + \begin{bmatrix} 5 & 8 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 8 & 14 \\ 10 & 12 \end{bmatrix}$$

그림 9-9 행렬의 연산

68

3. 파이썬 스타일 코드로 표현한 행렬

- 행렬의 연산을 파이썬 스타일 코드를 사용하여 표현하기

☞ 가장 쉬운 표현 방법은 별표(*)와 zip() 함수를 활용하는 것.

```
>>> 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]]
```

Handwritten notes: $\begin{pmatrix} 3, 5 \\ 4, 6 \end{pmatrix} \rightarrow \begin{pmatrix} 8 \\ 10 \end{pmatrix}$, $\begin{pmatrix} 6, 8 \\ 5, 7 \end{pmatrix} \rightarrow \begin{pmatrix} 14 \\ 12 \end{pmatrix}$

- 코드의 핵심: 'zip()' 함수를 어떻게 활용하는가'
 - 리스트 컴프리헨션 안에 2개의 for문이 있음.
 - 뒤에 있는 for문이 먼저 실행되어 matrix_a와 matrix_b에서 zip() 함수를 통해 같은 인덱스에 있는 값들이 추출됨. 즉, [3, 6]과 [5, 8]이 튜플로 묶여 ([3, 6], [5, 8])로 추출됨.

69

3. 파이썬 스타일 코드로 표현한 행렬

```
>>> [t for t in zip(matrix_a, matrix_b)]
[[([3, 6], [5, 8]), ([4, 5], [6, 7])]
```

- t: 2개의 리스트 값을 가진 하나의 튜플로 앞의 for문에 있는 리스트 컴프리헨션 구문에 들어감.
 - t는 1개의 튜플이므로 zip() 함수를 사용하기 위해 값을 언패킹해야 함.
 - [sum(row) for row in zip(*t)]와 같이 언패킹한 상태로 zip() 함수를 사용하면 ([3, 6], [5, 8])의 값에서 같은 인덱스에 있는 값들이 추출되어 (3, 5), (6, 8)의 형태로 row 변수에 할당됨.
 - [sum(row) for row in zip(*t)] 코드에서 같은 위치에 있는 값끼리 묶여 row라는 이름의 튜플이 생성된 후 sum() 함수가 적용됨.
 - 이로 인하여 같은 위치의 값끼리 더해져 [[8, 14], [10, 12]]라는 결과가 나옴.

70

3. 파이썬 스타일 코드로 표현한 행렬

3.2 행렬의 동치

- 2개의 행렬이 서로 같은지를 나타내는 표현
- 만약 행렬이 같다면 '2개의 행렬이 동치'라고 말함.
- [그림 9-10]은 두 행렬이 'A = B'이기 위한 조건을 나타낸 것.

$$A = \begin{bmatrix} 3 & 6 \\ 4 & 5 \end{bmatrix} \quad B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$b_{11} = 3, b_{12} = 6, b_{21} = 4, b_{22} = 5$$

그림 9-10 행렬의 동치

71

3. 파이썬 스타일 코드로 표현한 행렬

- 두 행렬이 동치임을 확인하는 코드

☞ '행렬의 연산'과 비슷한 코드를 작성하되 불린형을 활용.

```
>>> matrix_a = [[1, 1], [1, 1]]
>>> matrix_b = [[1, 1], [1, 1]]
>>> all([row[0] == value for t in zip(matrix_a, matrix_b) for row in
zip(*t) for value in row])
True
```

Handwritten notes: $\begin{pmatrix} 1, 1 \\ 1, 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1, 1 \\ 1, 1 \end{pmatrix}$, $\begin{pmatrix} 1, 1 \\ 1, 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1, 1 \\ 1, 1 \end{pmatrix}$

```
>>> matrix_b = [[5, 8], [6, 7]]
>>> all([all([row[0] == value for value in row]) for t in
zip(matrix_a, matrix_b)
for row in zip(*t)])
False
```

72

3. 파이썬 스타일 코드로 표현한 행렬

- **all() 함수:** 안에 있는 모든 값이 참일 경우에만 True를 반환함.
- **any() 함수:** 하나라도 참이 있으면 True를 반환 하고, 모두가 거짓일 때만 False를 반환함.

```
>>> any([False, False, False])
False
>>> any([False, True, False])
True
>>> all([False, True, True])
False
>>> all([True, True, True])
True
```

73

3. 파이썬 스타일 코드로 표현한 행렬

- 마지막에 같은 인덱스에 있는 값들을 row라는 튜플에 할당한 후, 마지막 for문인 for value in row 코드로 row 안의 값을 다시 value에 할당함.
- 그리고 모든 인덱스의 값이 같은지 확인하여 True 또는 False로 반환한 후, 마지막으로 all() 함수로 동치 여부를 확인함.
- 만약 all() 함수가 중간에 없다면 다음과 같은 결과가 출력됨.

```
>>> [[row[0] == value for value in row] for t in zip(matrix_a,
matrix_b) for row in zip(*t)]
[[True, False], [True, False], [True, False], [True, False]]
```

74

3. 파이썬 스타일 코드로 표현한 행렬

3.3 전치행렬(transpose matrix)

- 주어진 m × n의 행렬에서 행과 열을 바꾸어 만든 행렬

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}$$

(a) 공식

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$


(b) 예시

그림 9-11 전치행렬

- 전치행렬을 구현하기 위해 행과 열의 값을 변경해야 함.

75

3. 파이썬 스타일 코드로 표현한 행렬



```
>>> matrix_a = [[1, 2, 3], [4, 5, 6]]
>>> result = [[element for element in t] for t in zip(*matrix_a)]
>>> result
[[1, 4], [2, 5], [3, 6]]
```

- 위 코드의 핵심: for t in zip(*matrix_a)
- 별표 때문에 리스트를 다음과 같이 언패킹함.

```
zip([1, 2, 3], [4, 5, 6])
```

- 이렇게 언패킹한 상태에서 zip() 함수를 사용하면 같은 위치의 값들을 t로 할당할 수 있음. 즉, [1, 4], [2, 5], [3, 6]이 묶임.
- 이 값들이 그대로 리스트로 들어가면 전치행렬이 완성됨.

```
>>> [t for t in zip(*matrix_a)]
[(1, 4), (2, 5), (3, 6)]
```

76

3. 파이썬 스타일 코드로 표현한 행렬

3.4 행렬의 곱셈

- 앞 행렬의 행과 뒤 행렬의 열을 선형 결합하면 됨.
- [그림 9-12]와 같이 대응되는 값들끼리 곱셈 연산하면 됨.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \\ 139 & 154 \end{bmatrix}$$

$(1 \times 7) + (2 \times 9) + (3 \times 11)$

그림 9-12 행렬의 곱셈

- [그림 9-13]과 같이 행렬의 곱셈을 위한 조건을 만족하여야 연산이 됨.



그림 9-13 행렬의 곱셈을 하기 위한 조건

3. 파이썬 스타일 코드로 표현한 행렬

- 코드 구현 - 전치행렬의 코드 기법을 사용하여 한 행렬에서는 열의 값을, 다른 행렬에서는 행의 값을 추출하여 곱하는 코드로 구성해야 함.

```
>>> matrix_a = [[1, 1, 2], [2, 1, 1]]
>>> matrix_b = [[1, 1], [2, 1], [1, 3]]
>>> result = [[sum(a * b for a, b in zip(row_a, column_b)) for
column_b in zip(*matrix_b)] for row_a in matrix_a]
>>> result
[[5, 8], [5, 6]]
```