데이터 과학을 위한
파이썬 프로그래밍

2판

Chapter 08
파이썬 스타일 코드 I

목차

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
```

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

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

02
문자열의 분리 및 결합

1. 문자열의 분리: split( ) 함수

- **split( ) 함수:** 특정 값을 기준으로 문자열을 분리하여 리스트 형태로 변환

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

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

1. 문자열의 분리: split( ) 함수

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

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'
```

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'
```

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]
```

- 리스트 컴프리헨션 코드가 훨씬 더 간결하다는 것을 알 수 있음.

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.

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]
```

➢ 리스트 컴프리헨션

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

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.

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]
```

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.

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]
```

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']]
```

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.

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)
```

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 만듦.
- 스칼라는 파이썬의 정수형 또는 실수형으로, 벡터는 리스트로 치환하여 생각하면 됨

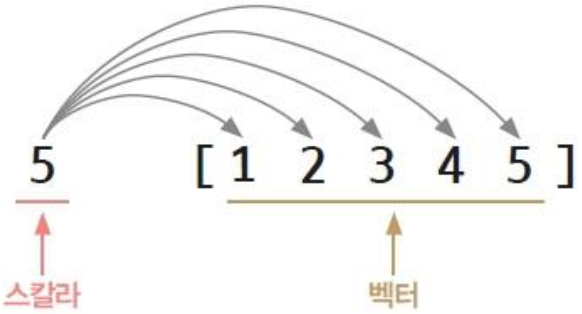

그림 8-1 벡터와 스칼라

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
user    0m10.203s
sys     0m0.023s
```

(a) 일반적인 반복문 + 리스트

```
real    0m7.788s
user    0m7.762s
sys     0m0.021s
```

(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

```
>>> 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 언패킹되어 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.'}
```

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
```

- 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]
```

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.

Thank you!

한빛아카데미 Hanbit Academy, Inc.