제 2 장 파이썬 기초 문법

1. 튜플(Tuple)

1.1 튜플이란?

- 리스트와 유사한 구조를 가진 자료구조
- 튜플은 리스트에 비해 접근 속도가 빠름
- 하지만 튜플은 직접 변경할 수 없으며 리스트와 같이 append, insert등의 함수 사용 불가

튜플 = (항목1 , 항목2 , 항목3, … , 항목n)

튜플은, 이전에 배운 리스트와 비슷한 자료구조라고 생각하시면 됩니다. 단지, 튜플은 한 번 값이 할당된 후에는 변경할 수 없습니다. 이게 리스트와 다른 점이죠. 5개의 과일을 하나의 튜플로 묶어 저장하는 예제를 살펴보겠습니다.

5개의 과일을 쉼표로 구분하여 data라는 변수에 대입하는 것을 보실 수 있습니다. 변수 data를 print문을 이용해 출력해보면 소괄호와 함께 튜플의 형태로 5개의 과일이 출력된 것을 확인할 수 있습니다. 또한, type 함수를 이용해 data 변수의 자료형을 출력하면 'tuple' 클래스가 출력되는 것을 볼 수 있습니다. 그림 1은 자료구조 튜플에 대한 내용입니다.

```
1 # 튜플의 기본 구조
2 data = ("사과", "배", "포도","토마토","딸기")
3
4 print(data)
5 print(type(data))
```

```
('사과', '배', '포도', '토마토', '딸기')
<class 'tuple'>
```

그림 1 자료구조 튜플

변수 data의 0번째 항목을 100이라는 값으로 변경하는 코드입니다. 출력된 결과가 같이 튜플은 한 번 값이 변경할 수 없기 때문에 에러가 나는 것을 확인할 수 있습니다. 그림 2는 튜플의 데이터 변경 에러에 대한 내용입니다.

```
data[0] = 100
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-dfb8473275cf> in <module>()
----> 1 data[0] = 100

TypeError: 'tuple' object does not support item assignment
```

그림 2 튜플의 데이터 변경 에러

다음은 튜플의 4가지 기본 연산에 대해 알아보겠습니다.

1.2 튜플의 기본 연산

- 튜플 전체 데이터 수 검색
- 튜플 슬라이싱
- 두 개의 튜플 데이터 병합
- 튜플 데이터 반복

먼저 튜플 안에 들어있는 특정 항목의 개수를 검색하는 방법과 튜플의 일부분을 자르는 방법, 두 개의 튜플 데이터를 병합하는 방법, 마지막으로 튜플 데이터 전체를 반복하여 출력하는 방법을 배워보겠습니다.

1.2.1 튜플 전체 데이터 수 검색

튜플 안에 들어있는 특정 항목의 개수는 튜플 클래스의 count라는 함수를 이용해 검색할 수 있습니다. 먼저 변수 data에 100, 200, 100, 300, 100, 500 이렇게 6개의 수를 넣어 튜플 형태로 저장합니다. 그리고 100이 data 튜플 안에 몇 개가 들어잇는지 count() 함수에 100을 넣어 print() 문으로 출력합니다. 결과적으로 3이 출력된 것을 확인할 수 있습니다. 그림 3은 튜플의 count 함수에 대한 내용입니다.

```
1 data = (100, 200, 100, 300, 100, 500)
2 print(type(data))
3 # 튜플은 Count함수를 이용해 해당 데이터의 개수를 확인가능함
4 print(data.count(100))
```

```
<class 'tuple'>
3
```

그림 3 튜플 count 함수

1.2.2 튜플 슬라이싱

이번에는 같은 변수 data의 일부분만 출력해보겠습니다. 튜플의 항목은 순서가 있으며 data 변수의 항목은 왼쪽부터 0,1,2,3,4,5라는 인덱스를 가지고 접근할 수 있습니다. 아래와 같이 data 변수의 두 번째 항목부터 네 번째 전인 세 번째 항목까지 print()문 안에 넣어 출력하면 100과 300이 튜플 형태로 출력됩니다. 그림 4는 튜플 슬라이싱(sclicing)에 대한 내용입니다.

```
data = (100, 200, 100, 300, 100, 500)
print(type(data))

# 튜플은 리스트와 같이 데이터의 슬라이싱이 가능
print(data[2:4])
```

```
<class 'tuple'>
(100, 300)
```

그림 4 튜플 슬라이싱(slicing)

1.2.3 두 개의 튜플 데이터 병합

두 개 이상의 튜플을 병합하여 하나로 만들기 위해서는 덧셈 연산자를 사용하면 됩니다. 예제 코드와 같이 data1과 data2 두 개의 튜플을 하나로 합쳐보겠습니다. 먼저, data1과 data2에 각각 5개씩 항목을 넣어 선언하고 새로운 data 변수에 data1과 data2를 덧셈 연산자로 연산하여 대입합니다. 결과적으로 print()문으로 변수 data를 출력하면 총 10개의 항목으로 이루어진 튜플이 출력됩니다. 그림 5는 두 튜플의 병합에 대한 내용입니다.

```
1 data1 = ("사과", "배", "포도","토마토","딸기")
2 data2 = (100, 200, 300, 400, 500)
3
4 # 두개의 튜플데이터를 덧셈연산자를 통해 하나로 합침
5 data = data1 + data2
6 print(data)
```

```
('사과', '배', '포도', '토마토', '딸기', 100, 200, 300, 400, 500)
```

그림 5 두 튜플의 병합

1.2.4 튜플 데이터 반복

튜플 안의 데이터를 반복하여 출력해보는 예제 코드를 실습해 보겠습니다. 사과, 배, 포도, 토마토, 딸기 5개의 항목으로 이루어진 변수 data를 선언한 후 곱셈 연산자를 이용해 'data' 곱하기 2를 print() 함수 안에 넣어 출력하면 기존 5개의 항목이 각각 2개씩 중복되어 총 10개의 과일명이 튜플 형태로 출력되는 것을 확인 할 수 있습니다. 그림 6은 튜플 안의 데이터 반복에 대한 내용입니다.

```
1 data = ("사과", "배", "포도","토마토","딸기")
2
3 # 튜플데이터는 곱셈연산자를 이용해 데이터를 반복할 수 있음
4 print(data*2)
```

```
('사과', '배', '포도', '토마토', '딸기', '사과', '배', '포도', '토마토', '딸기')
```

그림 6 튜플 안의 데이터 반복

2. 세트(Set)

2.1 세트란?

- 수학에서 배웠던 집합을 세트(set)라고 함
- 세트는 순서에 상관없고 중복을 허용하지 않는 자료구조
- 세트는 중괄호 기호 안에 항목들을 쉼표로 분리

세트 = {항목1 , 항목2 , 항목3 , … , 항목n}

세트는 우리가 흔히 아는 '집합'의 개념입니다. 세트는 앞에서 배웠던 튜플과 달리 항목의 순서가 없고 중복을 허용하지 않는 자료구조입니다. 세트는 중괄호와 함께 항목들을 쉼표로 구분하여 선언할 수 있습니다.

예제를 보면 변수 data는 100, 200, 300을 항목으로 갖는 세트형 변수입니다. type 변수를 사용하여 data의 자료형을 얻고 print() 함수에 넣어 출력하면 세트형 클래스가 출력되는 것을 확인할 수 있습니다. 세트는 튜플과 달리 한 번 값을 할당한 후에도 다른 값을 추가하거나 변경이 가능합니다. 그림 7은 자료구조 세트에 대한 내용입니다.

```
1 data = {100, 200, 300}
2 print(data)
3 print(type(data))
```

```
{200, 300, 100}
<class 'set'>
```

그림 7 자료구조 세트

리스트나 튜플과 달리 세트의 항목에는 순서가 없기 때문에 각 항목에 접근할 수 있는 인덱스 즉, 번호가 없습니다. 따라서 데이터의 인덱싱이나 일부분을 자르는 슬라이싱은 불가능합니다.

예제에서 변수 data는 100부터 700까지 7개의 정수를 항목으로 갖는 '세트' 형 변수입니다. 변수 data의 일부분을 자르기 위해 아래와 같이 인덱스로 세트에 접근한다면 출력된 결과와 같이 TypeError가 발생합니다. 그림 8은 자료구조 세트의 에러에 대한 내용입니다.

```
1 data = {100, 200, 300, 400, 500, 600, 700}
2 print(data)
3 print(type(data))
4
5 # set자료구조는 인덱싱과 슬라이싱이 불가능
6 data[0:4]
```

```
{100, 200, 300, 400, 500, 600, 700}
<class 'set'>

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-87-12dc64151c11> in <module>()
      4
      5 # set자료구조는 인덱싱과 슬라이싱이 불가능
----> 6 data[0:4]

TypeError: 'set' object is not subscriptable
```

그림 8 자료구조 세트의 에러

2.2 기본연산

- 데이터 추가
- 데이터 업데이트
- 데이터 삭제
- 데이터 초기화

세트형 변수의 네 가지 기본 함수에 대해 다뤄보겠습니다. 세트형 변수에 하나의 데이터를 추가하고 여러 개의 데이터를 업데이트 하고, 삭제 및 초기화 하는 방법을 실습해 보겠습니다.

2.2.1 데이터 추가

세트형 변수에 x라는 변수를 추가하기 위해서 세트 클래스의 add()라는 함수를 사용합니다. 예제의 변수 data는 100부터 600까지 6개의 정수로 이루어진 세트형 변수입니다. print() 함수 안에 변수 data를 넣어 출력하면 데이터가 잘 출력되는 것을 확인할 수 있습니다.

data.add를 호출하면 파라미터로 들어간 700이 data 변수에 추가되게 됩니다. 출력해보면 700이 추가되어 7개의 항목이 들어가 있는 것을 확인할 수 있습니다. print()문에 변수 data를 넣어 출력해보면 700이 추가되어 총 7개의 항목이 출력된 것을 확인할 수 있습니다. 그림 9는 자료구조 세트의 데이터 추가에 대한 내용입니다.

```
1 data = {100, 200, 300, 400, 500, 600}
2 print(data)
3
4 # 세트 자료구조는 add함수를 이용해 데이터를 추가할 수 있음
5 data.add(700)
6 print(data)
```

```
{100, 200, 300, 400, 500, 600}
None
{100, 200, 300, 400, 500, 600, 700}
```

그림 9 자료구조 세트의 데이터 추가

2.2.2 데이터 업데이트

update라는 함수를 사용해 데이터를 추가하는 방법을 배우도록 하겠습니다. 코드 첫 번째 줄에서 보여준 것과 같이, data라는 이름으로 set 자료형 변수를 만듭니다. 그리고 100 값을 넣어줍니다. 이때, 변수 data에는 100 값만 들어있습니다. 두 번째 줄에는 '여러 개의 데이터를 추가하기 전'이라는 문구를 출력하는 것을 보여주고 있습니다.

그리고 코드 세 번째 줄에는, 첫 번째 줄에서 정의한 변수 data 값을 출력합니다. 다섯 번째 줄에는 print함수를 사용했는데 이와 같이, print 함수를 사용할 때 괄호 안에 아무것도 안 넣어주면 빈 줄을 출력하여 다음 줄로 넘어갑니다. 이제 update ㅎ마수를 사용해 변수에 값을 추가하겠습니다.

코드 일곱 번째 줄에 보여준 것처럼, update 함수를 사용해 앞에서 data라는 이름으로 정의한 변수에, 200, 300, 400, 500, 600 다섯 개의 값을 넣어줍니다. 그리고 여덟 번째 줄에, '여러 개의 데이터를 추가한 후'라는 문구를 출력합니다. 마지막으로 아홉 번째 줄에 변수 data의 값을 출력합니다. 앞에서 update 함수를 사용해 데이터를 추가했기 때문에, 변수 data의 원래 값 100이라는 값과 update 함수로 추가한 값이 출력되는 것을 확인할 수 있습니다. 그림 10은 자료구조 세트의 데이터 업데이트에 대한 내용입니다.

```
1 data = {100}
2 print("여러개의 데이터를 추가하기 전")
3 print(data)
4
5 print()
6 # 세트 자료구조에서 여러 개의 데이터를 추가할 때는 update함수를 이용해 추가
7 data.update([200,300,400,500,600])
8 print("여러개의 데이터를 추가한 후")
9 print(data)
```

```
여러개의 데이터를 추가하기 전
{100}

여러개의 데이터를 추가한 후
{100, 200, 300, 400, 500, 600}
```

그림 10 자료구조 세트의 데이터 업데이트

2.2.3 데이터 삭제

세트 자료형 변수에서 값을 삭제하는 방법을 배우도록 하겠습니다. 코드 첫 번째 줄에서 보여준 것과 같이, 변수 data를 만들고, 100, 200, 300, 400, 500, 600 여섯 개의 값을 넣어줍니다. 그리고 '데이터를 삭제하기 전' 이라는 문구가 출력합니다. 코드 세 번째 줄과 같이 변수 data의 값을 출력하여, 값이 들어가 있는 것을 확인할 수 있습니다. 다섯 번째 줄과 같이, print문을 사용하되, 괄호 안에 아무 값도 넣어주지 않습니다.

이는 앞서 말씀 드렸다시피, 새로운 라인을 출력하기 위해 사용한다는 것을 기억하시죠. 여기까지는 변수 data의 값이 변하지 않습니다. 이제 remove 함수를 사용해 값을 삭제하는 방법을 배워보겠습니다. 코드 일곱 번째 줄과 같이, remove 함수를 사용해 삭제하려고 하는 값을 지울 수 있습니다. 괄호 안에 600 이라는 값을 넣어, 이를 삭제하도록 remove 함수를 사용합니다. '데이터를 삭제한 후'라는 문구를 출력되는 것을, 확인할 수가 있습니다. 그림 11은 자료구조 세트의 데이터 삭제에 대한 내용입니다.

```
1  data = {100, 200, 300, 400, 500, 600}
2  print("데이터를 삭제하기 전")
3  print(data)
4
5  print()
6  # 세트 자료구조에서는 remove를 이용해 데이터를 제거
7  data.remove(600)
8
9  print("데이터를 삭제한 후")
10 print(data)
```

```
데이터를 삭제하기 전
{100, 200, 300, 400, 500, 600}

데이터를 삭제한 후
{100, 200, 300, 400, 500}
```

그림 11 자료구조 세트의 데이터 삭제

2.2.4 데이터 초기화

세트 자료형 변수를 초기화 하는 방법을 배우도록 하겠습니다. data 변수를 정의하되, 여섯 개의 값을 넣어줍니다. '데이터를 삭제하기 전'이라는 문구를 출력하고, data라는 변수의 값을 출력합니다. 이때, 변수 data에서 여섯 개의 값이 출력되는 것을 확인할 수 있습니다.

그리고 코드 다섯 번째 줄에서, print()라는 함수를 사용해 빈 줄을 하나 출력합니다. 이는 enter와 같은 역할을 합니다. 다음, 코드 일곱 번째 줄에서 claear 함수를 사용해 봅니다.

변수 data의 값을 출력합니다. 이때, 변수 data에 앞에서 넣어 주었던 값이, 다 없어진 것을 확인 할 수 있습니다. 이와 같이 clear 함수를 사용하면, set 자료형 변수의 값을 초기화 할 수 있습니다. 그림 12는 자료구조 세트의 데이터 초기화에 대한 내용입니다.

```
1  data = {100, 200, 300, 400, 500, 600}
2  print("데이터를 삭제하기 전")
3  print(data)
4
5  print()
6  # 세트 자료구조에서 clear를 이용해 전체 데이터를 초기화
7  data.clear()
8  print(data)
```

```
데이터를 삭제하기 전
{100, 200, 300, 400, 500, 600}

set()
```

그림 12 자료구조 세트의 데이터 초기화

2.3 집합연산

- 교집합
- 차집합
- 합집합
- 합집합에서 교집합을 뺀 집합

여러 개의 세트 자료형 변수를 사용한 집합연산에 대해 배우겠습니다. 먼저, 교집합 연산에 대해 배우겠습니다.

2.3.1 교집합 연산

변수 A에 100, 200, 300, 400, 500 다섯 개의 값을 넣어 줍니다. 그리고 변수 B에 100, 150, 200, 250, 300 다섯 개의 값을 넣어줍니다. 코드 세 번째 줄에서는 변수 A의 값을 출력하고, 다섯 번째 줄에는 변수 B의 값을 출력하고 있습니다. 그리고 네 번째 줄에는 변수 A의 자료형 형태를 출력하고, 여섯 번째 줄에는 변수 B의 자료형 형태를 출력합니다. 그리고 아홉 번째 줄의 코드처럼 print()함수를 사용해 빈 줄을 출력합니다. 이제부터 교집합을 계산하는 방법을 알아보겠습니다.

코드 열한 번째 줄에서 보여준 것처럼, intersection이라는 함수를 사용해 교집합을 구할 수 있습니다. 출력하면 변수 A와 B의 값 중에서, 서로 중복되는 값을 출력하는 것을 확인할 수 있습니다. 그리고 intersection 함수를 사용할 때, A와 B의 순서를 바꿔도 값은 변함이 없다는 것을 알려드립니다. 그림 13은 세트의 교집합 연산에 대한 내용입니다.

```
1  A = {100, 200, 300, 400, 500}
2  B = {100, 150, 200, 250, 300}
3  print(A)
4  print(type(A))
5  print(B)
6  print(type(B))
7
8  # intersection함수는 교집합을 나타내는 함수
9  print()
10 print("intersection은 교집합을 나타내는 함수")
11 print(A.intersection(B))
```

```
{400, 200, 500, 300, 100}
<class 'set'>
{200, 250, 300, 100, 150}
<class 'set'>

intersection은 교집합을 나타내는 함수
{200, 300, 100}
```

그림 13 자료구조 세트의 교집합 연산

2.3.2 차집합 연산

세트의 차집합 연산을 해보겠습니다. 이전과 동일하게 변수 A와 B를 사용합니다. A, B 세트와 그 자료형을 type() 함수와 print() 함수를 이용하여 출력 확인할 수 있습니다. 세트의 차집합을 위해서 세트 클래스의 difference() 함수를 이용하면 되는데 빼려는 변수를 파라미터로 넣어주면 됩니다. 즉, A.difference(B) 함수를 호출하면 A 세트에서 A와 중복된 B의 항목이 제외된 400과 500이 출력되는 것을 확인할 수 있습니다.

마찬가지로 B.difference(A)를 하게 되면 B 세트에서 B와 중복된 A의 항목이 제외된 값이 출력되므로 현재 출력된 값과 다른 값이 출력됩니다. 그림 14는 세트의 차집합 연산에 대한 내용입니다.

```
1  A = {100, 200, 300, 400, 500}
2  B = {100, 150, 200, 250, 300}
3  print(A)
4  print(type(A))
5  print(B)
6  print(type(B))
7
8  # difference함수는 차집합을 나타내는 함수
9  print()
10 print("difference함수는 차집합을 나타내는 함수")
11 print(A.difference(B))
```

```
{400, 200, 500, 300, 100}
<class 'set'>
{200, 250, 300, 100, 150}
<class 'set'>

difference함수는 차집합을 나타내는 함수
{400, 500}
```

그림 14 자료구조 세트의 차집합 연산

2.3.3 합집합 연산

세트의 합집합을 구해보는 실습을 해보겠습니다. 이전과 동일하게 변수 A와 B를 선언하며, 합집합을 구하기 위해 세트 클래스의 union() 함수를 사용합니다. A 세트와 B 세트의 합집합을 구하기 위해 A.union(B) 함수를 호출하게 되면 A 세트와 B 세트를 모두 합한 10개의 항목에서 중복된 200, 300, 100이 한 번씩 빠진 총 7개의 항목이 출력되게 됩니다.

앞에서 배웠던 intersection() 함수, difference() 함수와 다르게 union() 함수는 A와 B의 순서가 변해도 합집합이기 때문에 출력되는 결과가 달라지지 않습니다. 그림 15는 세트의 합집합 연산에 대한 내용입니다.

```
A = {100, 200, 300, 400, 500}
B = {100, 150, 200, 250, 300}
print(A)
print(type(A))
print(B)
print(type(B))

# union함수는 합집합을 나타내는 함수
print()
print("union은 합집합을 나타내는 함수")
print(A.union(B))
```

```
{400, 200, 500, 300, 100}
<class 'set'>
{200, 250, 300, 100, 150}
<class 'set'>

union은 합집합을 나타내는 함수
{100, 200, 300, 400, 500, 150, 250}
```

그림 15 자료구조 세트의 합집합 연산

2.3.4 합집합에서 교집합을 뺀 집합

마지막으로 합집합에서 교집합을 빼는 연산 함수를 실습해 보겠습니다. 역시 동일한 변수 A, B를 사용합니다. 합집합에서 교집합을 뺀 집합 구하는 함수는 세트 클래스의 symmetric_difference() 함수입니다. 세트 A와 B의 합집합에서 둘의 교집합을 빼기 위해 A.symmetric_difference(B) 함수를 호출하면 앞에서 합집합으로 나온 7개의 항목에서 교집합인 200, 300, 100 3개의 항목이 제외된 나머지 4개의 항목이 출력됩니다. 그림 16은 세트의 합집합에서 교집합을 뺀 연산에 대한 내용입니다.

```
A = {100, 200, 300, 400, 500}
B = {100, 150, 200, 250, 300}
print(A)
print(type(A))
print(B)
print(type(B))

# symmetric_difference함수는 합집합에서 교집합을 뺀 나머지 집합을 나타내는 함수
print()
print("symmetric_difference함수는 합집합에서 교집합을 뺀 나머지 집합을 나타내는 수")
print(A.symmetric_difference(B))
```

```
{400, 200, 500, 300, 100}
<class 'set'>
{200, 250, 300, 100, 150}
<class 'set'>

symmetric_difference함수를 사용해 합집합에서 교집합을 뺀 나머지 집합을 나타내는 수
{400, 500, 150, 250}
```

그림 16 자료구조 세트의 합집합에서 교집합을 뺀 연산

3. 딕셔너리(Dictionary)

3.1 딕셔너리란?

- 사전이라는 의미
- 사전에는 단어와 설명이 있는데 파이썬의 딕셔너리에서는 이를 키(Key)와 값(Value)로 표현
- 딕셔너리는 중괄호 안에 항목을 쉼표로 분리시켜 나열

마지막으로 다루 자료구조는 딕셔너리 자료형입니다. 딕셔너리는 한글로 사전이라는 의미입니다. 사전에는 어떤 단어와 그 단어에 대한 설명들이 나열되어 있는 것처럼, 딕셔너리는 'key' 값과 'key'값에 대한 정보가 들어있는 'value'값의 집합으로 이루어져 있습니다.

딕셔너리는 중괄호 안에 'key'값과 콜론, 'value'값을 하나의 항목으로 쉼표로 분리시켜 정의할 수 있습니다. 예제에서 보면 dictionary 변수에는 'name', 'email', '주소' 세 개의 'key' 값이 잇고 그에 대한 'value'값이 있습니다.

Name은 '홍길동'이고, 'gdHong@namver.com'이라는 email 주소를 가지고 있고, 서울시 광진구에 사는 사람에 대한 데이터입니다. 변수 dictionary의 자료형을 type() 함수에 넣어 출력하면 'dict'가 찍히는 것을 볼 수 있습니다. 그림 17은 자료구조 딕셔너리에 대한 내용입니다.

	Key	Value
a	Name	홍길동
b	Email	gdHong@namver.com
c	주소	서울시 광진구

```
dictionary = {
                'Name' : '홍길동',
                'Email' : 'gdHong@naver.com',
                '주소' : '서울시 광진구'
             }
type(dictionary)
```

```
dict
```

그림 17 자료구조 딕셔너리

3.2 기본구조

딕셔너리의 기본 구조에 대해 알아보겠습니다. 변수 data는 각 과일에 대한

수량 데이터를 담고 있는 변수입니다. “사과”라는 key 값에 300 이라는 value 를 넣어주고, 배는 200, 포도는 500, 딸기는 700으로 key 값과 value 값을 각각 매칭시킵니다.

세 번째 줄에서 print() 함수 안에 변수 data를 넣어 출력하면 변수 data 안의 key, value 값들이 출력되는 것을 볼 수 있습니다. 그리고 type() 함수로 자료형을 print() 함수 안에 넣어 출력하면 dict 클래스가 출력됩니다. 그림 18은 딕셔너리의 기본 구조에 대한 내용입니다.

```
1 # 딕셔너리(dictionary)의 기본 구조는 다음과 같이 표현할 수 있음
2 data = {"사과":300, "배":200, "포도": 500, "딸기": 700}
3 print(data)
4 print(type(data))
```

```
{'포도': 500, '딸기': 700, '사과': 300, '배': 200}
<class 'dict'>
```

그림 18 자료구조 딕셔너리의 기본구조

3.3 딕셔너리를 이용해 연산하기

- 항목 검색
- 항목 추가
- 항목 삭제
- 항목 정렬

딕셔너리의 네 가지 연산을 배워보겠습니다. 딕셔너리 안에서 항목을 검색하고 추가, 삭제, 정렬하는 방법입니다.

3.3.1 항목 검색

먼저, 딕셔너리 안의 항목을 검색하는 방법입니다. 이전과 동일하게 변수 data를 선언합니다. 변수 data 안에서 ‘사과’의 수량을 알고 싶으면 data 오른쪽에 중괄호와 그 안에 ‘key’ 값인 ‘사과’를 넣고 출력하면 됩니다.

그러면 ‘300’이 정확하게 출력되는 것을 확인할 수 있습니다. 위와 같이 대괄호를 사용해도 되지만 딕셔너리 클래스의 get() 함수를 사용해서도 특정 ‘key’ 값의 ‘value’ 값을 찾을 수 있습니다. 아홉 번째 줄을 보면 data.get() 함수를 호출하고 안에 ‘key’ 값 ‘사과’를 넣어서 print()문 안에 넣어 출력해도 위와 동일하게 300이 출력되는 것을 확인할 수 있습니다. 그림 19는 자료구

조 딕셔너리의 항목 검색에 대한 내용입니다.

```
# 딕셔너리 데이터 예제를 data변수에 입력
data = {"사과":300, "배":200, "포도": 500, "딸기": 700}

# 항목 검색할 때 항목의 키를 사용하면 됨
# 위에서 키값 중 사과에 대한 값을 갖고 옴
print(data["사과"])

# 위와 같이 표현도 가능하지만 아래와 같이 get함수를 이용해 표현도 가능
print(data.get("사과"))
```

```
300
300
```

그림 19 자료구조 딕셔너리 항목 검색

3.3.2 항목 추가

딕셔너리 안의 항목을 추가하는 방법입니다. 동일하게 변수 data를 선언한 후 추가 전후를 비교하기 위해 '추가하기 전 딕셔너리 전체 데이터'라는 문구를 출력하고 변수 data를 print{() 함수 안에 넣어 출력합니다. 그리고 한 줄을 띄우기 위해서 print() 문을 파라미터 없이 출력합니다. 그리고 이번에는 새로운 과일인 메론의 수량을 항목으로 추가시켜보겠습니다.

추가하기 위해 왼쪽에는 딕셔너리의 'key' 값, 오른쪽에는 'value' 값을 적어줍니다. 즉, 왼쪽에는 변수 data와 중괄호, 그리고 그 안에 '메론'을 적어주고 오른쪽에 수량 1000을 적어 'key' 값과 'value' 값을 대입 연산자로 추가시켜줍니다. 이제 변수 data에 새로운 과일을 추가했으니 추가 전의 변수 data와 비교해보겠습니다.

print() 함수 안에 '추가한 후 딕셔너리 전체 데이터' 문자열을 넣어 출력하고 변수 data를 출력합니다. 출력된 결과를 보면 딕셔너리에 추가하기 전 데이터는 '포도, 딸기, 사과, 배' 기존 4개의 과일에 대한 정보만 있고, 추가한 후에는 '메론'이 추가된 5개의 과일이 출력된 것을 확인할 수 있습니다. 그림 20은 자료구조 딕셔너리의 항목 추가에 대한 내용입니다.

```
1  # 딕셔너리 데이터 예제를 data변수에 입력
2  data = {"사과":300, "배":200, "포도": 500, "딸기": 700}
3  print("추가하기 전 딕셔너리 전체 데이터")
4  print(data)
5
6  print()
7
8  # 딕셔너리에 데이터를 추가할 때는 아래와 같이 키를 설정하고 값을 입력함
9  data["메론"] = 1000
10 print("추가한 후 딕셔너리 전체 데이터")
11 print(data)
```

```
딕셔너리에 추가하기 전 전체 데이터
{'포도': 500, '딸기': 700, '사과': 300, '배': 200}

딕셔너리에 추가한 후 전체 데이터
{'포도': 500, '딸기': 700, '사과': 300, '배': 200, '메론': 1000}
```

그림 20 자료구조 딕셔너리 항목 추가

3.3.3 항목 삭제

딕셔너리 안의 항목을 삭제하는 방법입니다. 동일하게 변수 data를 선언하고 마찬가지로 전, 후를 비교해보겠습니다. print()함수로 '삭제하기 전 딕셔너리 전체 데이터' 문자열을 출력하고 변수 data를 출력합니다. 그리고 빈 파라미터로 print() 함수를 호출하고 '삭제한 후 딕셔너리 데이터'라는 문자열을 print() 함수를 출력합니다.

딕셔너리 안의 항목 삭제를 위해서는 pop() 함수를 사용하면 되는데 pop() 함수 안에 삭제하고자 하는 항목의 'key' 값을 파라미터로 넣어주면 됩니다. 열 번째 줄에서 '사과'에 대한 수량 항목을 삭제하기 위해 data.pop('사과') 함수를 호출합니다. 그리고 변수 data를 print()문에 넣어 출력합니다.

출력된 결과를 보면 삭제하기 전 데이터는 포도, 딸기, 사과, 배에 대한 수량 데이터가 출력되었는데, 삭제 후에는 '배'의 수량 데이터가 빠진 나머지 3개의 항목만 출력된 것을 확인할 수 있습니다. 그림 21은 딕셔너리의 항목 삭제에 대한 내용입니다.

```
1  # 딕셔너리 데이터 예제를 data변수에 입력
2  data = {"사과":300, "배":200, "포도": 500, "딸기": 700}
3  print(" 삭제하기 전 딕셔너리 전체 데이터")
4  print(data)
5
6  print()
7
8  # 딕셔너리는 pop함수를 이용해 데이터를 삭제 가능
9  print("삭제한 후 딕셔너리 데이터")
10 data.pop("사과")
11 print(data)
```

```
딕셔너리에 삭제하기 전 전체 데이터
{'포도': 500, '딸기': 700, '사과': 300, '배': 200}

딕셔너리에 삭제한 후 데이터
{'포도': 500, '딸기': 700, '배': 200}
```

그림 21 자료구조 딕셔너리 항목 삭제

3.3.4 항목 정렬

마지막으로 딕셔너리의 항목을 정렬해보겠습니다. 앞과 동일하게 '사과', '배', '포도', '딸기' 네 과일의 수량을 담고 있는 변수 data를 선언합니다. 마찬가지로 정렬 전 후를 비교하기 위해서 '정렬하기 전 딕셔너리 전체 데이터' 문자열과 변수 data를 각각 print() 문을 이용해 출력합니다.

그리고 5번째 줄에서 공백을 출력합니다. python에서 정렬을 위해 sorted() 함수를 제공하는데 sorted() 함수 안에 정렬하고자 하는 변수를 파라미터로 넣어주면 됩니다. 7번째 줄처럼 sorted() 함수 안에 변수 data를 넣고 print() 함수를 이용하여 출력합니다.

출력된 결과를 보면 딕셔너리 data의 'key' 값이 백과사전 순으로 정렬되어 출력된 것을 볼 수 있습니다. 만약 key 값이 아닌 value 값을 정렬시키고 싶으면 딕셔너리 클래스의 values() 함수를 이용해 sorted() 함수 안에 파라미터로 value 값들을 넣어주면 됩니다.

11번째 줄을 보면 sorted() 함수 안에 data.values() 함수로 호출된 value 값들을 넣어 출력합니다. 결과를 보면 value 값들이 오름차순으로 출력된 것을 볼 수 있습니다. 그림 22는 딕셔너리의 항목 정렬에 대한 내용입니다.

```python
data = {"사과":300, "배":200, "포도": 500, "딸기": 700}
print("정렬하기 전 딕셔너리 전체 데이터")
print(data)

print()
# sorted는 항목을 정렬시키는 내장함수
print(sorted(data))

# value값을 이용해 정렬하고 싶으면 아래와 같이 실행
print()
print(sorted(data.values()))
```

```
딕셔너리에 정렬하기 전 전체 데이터
{'포도': 500, '딸기': 700, '사과': 300, '배': 200}

['딸기', '배', '사과', '포도']

[200, 300, 500, 700]
```

그림 22 자료구조 딕셔너리 항목 정렬

4. 제어문 및 반복문

4.1 제어 및 반복문

- If문
- For문
- While문

파이썬의 제어문과 반복문에 대해 알아보겠습니다. 대표적인 제어문 if문과 반복문 for문, while문을 차례대로 살펴보겠습니다.

4.1.1 If 문

- If문은 데이터에서 조건이 일치하는 경우 실행되고 일치하지 않을 시 else문이 실행
- If문을 실행하기 위해서 아래 그림과 같이 블록을 생성함
- 해당 조건이 맞을 때 첫 번째 블록 '들여쓰기 한 명령문1'을 실행
- 맞지 않을 때 두 번째 블록 '들여쓰기 한 명령문2'를 실행
- If문을 구성할 때는 항상 4개의 공백 후 블록문을 들여쓰기 함

첫 번째로 if문은 데이터가 조건에 일치하는 경우 프로그램이 실행되고, 조건에 일치하지 않을 경우 else문이 실행되는 제어문입니다. if문 구조를 통해 자세히 살펴보겠습니다. 두 번째 줄에서 if문 뒤에 조건이 주어졌을 때 해당 조건이 맞으면 '명령문1'이 실행되며, 그렇지 않을 경우 else문에 해당하는 '명령문2'가 수행됩니다. 이 때, 각 조건에 해당할 때 수행되는 명령문은 들여쓰기를 통해 소속이 결정됩니다. 따라서 들여쓰기에 주의해야합니다. 그림 23은 if문 구조에 대한 내용입니다.

```
1 # if문은 조건에 따라 수행되는 명령이 달라짐
2 if 조건1 :
3     들여쓰기한 명령문1
```

그림 23 If문 구조

if문 예제를 실습하며 살펴보겠습니다. 먼저 변수 a에 초기 값으로 100을 저장합니다. 그리고 if문을 이용해 만약 a가 100이라는 조건이 맞을 경우 '일치'라는 단어를 출력하며, 그렇지 않을 경우 '불일치'라는 단어를 출력합니다. 예제는 변수 a의 값과 조건이 일치해 '일치'를 출력합니다.

두 번째 예제를 살펴보겠습니다. 첫 번째 줄에서 변수 a에 200을 저장했습니다. 그리고 네 번째 줄에서 위와 동일하게 위의 값이 100과 일치하는지를 확인하고 있습니다.

이번에는 위의 예제와 달리 a에 200이 저장되어 있기 때문에 if문의 조건에 맞지가 않습니다. 따라서 else문에 해당하는 '불일치'가 출력되는 것을 확인할 수가 있습니다. 그림 24는 if문 예제에 대한 내용입니다.

```
1 a = 100 #변수 a에 초기값으로 100을 저장
2
3 #변수 a가 100이라는 조건이 맞을 때 '일치' 출력, 아니면 '불일치' 출력
4 if a == 100:
5     print("일치")
6 else:
7     print("불일치")
```

일치

```
1 a = 200 #변수 a에 초기값으로 200을 저장
2
3 #변수 a가 100이라는 조건이 맞지 않으므로 '불일치' 출력
4 if a == 100:
5     print("일치")
6 else:
7     print("불일치")
```

불일치

그림 24 If문 예제

4.1.2 While문

- While문은 조건에 만족하는 값이 나올 때 까지 명령문들을 반복 수행
- While문이 시작한 행은 반복문의 헤더라 칭함
- 헤더의 조건을 비교해 while문의 몸통부분에 대해 코드를 조건에 만족하는 값이 나올 때 까지 반복적으로 수행
- 코드의 '들여쓰기한 명령문1' 블록을 몸통이라고 함

while 반복문에 대해서 살펴보겠습니다. while문은 주어진 조건이 부합하는 동안 명령문들을 반복 수행합니다. while문 구조를 통해 기능을 자세히 살펴보겠습니다. 세 번째 줄 while문이 시작하며 조건이 명시되어 있는 부분을 반복문의 헤더라고 부릅니다. 헤더의 조건에 데이터가 부합하는 동안에 while문에 속하는 명령문 네 번째 줄이 반복해서 수행이 됩니다.

이 때 수행되는 while문 내부를 몸통, body라고 합니다. 앞의 if문과 마찬가지로 명령문의 소속은 들여쓰기로 결정되므로, 명령문을 입력할 때 들여쓰기에 주의를 꼭 기울여야합니다. while문의 조건에 부합하는 동안 내부의 명령어는 반복적으로 수행이 되는데 특정 작업이 수행된 후나 조건에 따라 실행하던 작업을 중단해야할 경우가 발생합니다.

이 때, while문 내에서 명령을 반복 수행하던 작업을 중단하기 위해 break문을 사용할 수 있습니다. break문을 사용하면 반복하던 작업을 중단하고 while문을 탈출할 수 있습니다. 그림 25는 while문 구조에 대한 내용입니다.

```
1 # While문은 조건에 만족하는 값이 나올 때까지 명령문들을 반복 수행
2
3 while 조건문: #반복문의 헤더, 해당 반복문의 조건을 비교
4     들여쓰기한 명령문1
```

그림 25 While문 구조

예제 코드를 통해 살펴보겠습니다. 네 번째 줄 while문의 조건에 부합하는 동안 들여쓰기로 입력한 다섯 번째 줄의 명령문 1과 여섯 번째 줄의 명령문 2가 반복적으로 수행됩니다. 이때 일곱 번째 줄의 break문을 만나면 명령문들을 반복 수행하지 않고 while문 작업을 중단하게 됩니다. 이렇게 while문 내부에서 특정 작업이나 조건을 제시하고 break문을 통해 중단을 실행할 수 있습니다. 그림 26은 while문 구조에 대한 내용입니다.

```
1 # While문은 조건문에 만족할때까지 반복적으로 수행을 하지만
2 # 중간에 break문을 사용해 강제로 while문을 종료하기도 함
3
4 while 조건문: #반복문의 헤더, 해당 반복문의 조건을 비교
5     들여쓰기한 명령문1
6     들여쓰기한 명령문2
7     break     #반복문을 끝내는 부분
```

그림 26 While문 구조 2

while문 실습코드를 통해 살펴보겠습니다. 먼저 변수 a에 1을 저장합니다. 그리고 세 번째 줄에 while문에서 a의 값이 5이하인지 확인하는 조건문을 수행합니다. 이 조건에 변수의 값이 부합하는 동안 네 번째 줄과 다섯 번째 줄의 명령문이 실행이 됩니다.

네 번째 줄에서는 print() 함수를 사용해 변수 a를 출력하고 다섯 번째 줄

에서는 변수 a의 값에 1을 더하게 됩니다. 이 두 작업을 while문의 조건에 a의 값이 부합하는 동안 계속 반복을 하게 됩니다.

즉, a의 값을 출력을 하고 값을 1 증가시킨 뒤 그 값이 5 이하인지 확인하는 작업이 반복됩니다. 그렇게 진행하다가 계속 증가하던 a의 값이 5를 초과하면, while문의 조건에 부합하지 않으므로 while문을 종료합니다.

출력된 변수 a의 값은 1씩 더해지므로 차례로 출력이 되고 a의 값이 5가 되는 지점에서 while문을 종료하는 것을 확인할 수 있습니다. 그림 27은 while문 예제에 대한 내용입니다.

```
1 a = 1 #변수 a에 초기값 1 저장
2
3 while a <= 5: #while에 대한 조건문으로 변수 a의 값이 5이하가 될 때 까지 반복
4     print(a)  #변수 a 출력
5     a += 1    #변수 a에 1씩 더함
```

```
1
2
3
4
5
```

그림 27 While문 예제

4.1.3 For문

- For문은 리스트, 튜플, 문자열 등을 사용해 반복문 수행 가능
- While문과 비슷하지만 다양한 자료구조를 설정 후 명령문 블록이 실행

반복문 중 하나인 for문은 리스트, 튜플, 문자열 등의 값들을 변수로 사용해 명령문 반복 수행이 가능합니다. 앞서 살펴본 while 반복문과 비슷한 작업을 수행하지만, 다양한 자료구조를 변수로 설정 후 명령문 실행이 가능합니다. for문 구조를 통해서 다양한 자료구조를 변수로 사용하는 for문을 살펴보겠습니다.

네 번째 불의 for문 선언 부분을 살펴보겠습니다. in 명령어를 통해 리스트, 튜플 또는 문자열을 이루고 있는 각 요소에 대해 명령문을 실행합니다. 그림 28은 for문 구조에 대한 내용입니다.

```
# for문은 리스트, 튜플, 문자열 등을 반복적으로 실행
# for문도 while, if문과 같이 4칸의 들여쓰기를 포함하며
# 들여쓰기한 명령문을 실행
for [할당하는 변수] in [데이터의 범위(리스트, 튜플, 문자열) 또는 정수]:
    들여쓰기한 명령문1
```

그림 28 For문 구조

다양한 자료형을 사용해서 반복문을 수행하는 구조를 예제 코드를 통해 살펴보겠습니다. 먼저 리스트 구조 num_list에 1, 2, 3, 4, 5 값을 저장합니다. 그리고 세 번째 줄에서 for문을 이용해 num_list에 있는 값을 하나씩 변수 num에 입력하고 print() 함수를 이용해서 num값을 출력합니다. 그림 29는 for문 예제에 대한 내용입니다.

```
num_list = [1, 2, 3, 4, 5] #리스트 구조 변수 num_list에 1, 2, 3, 4, 5 값 저장

for num in num_list: #변수 num_list에 있는 값을 하나씩 변수 num에 입력
    print(num)       #변수 num 출력
```

```
1
2
3
4
5
```

그림 29 For문 예제

5. 함수

5.1 함수란?

- 함수는 재사용 가능한 프로그램을 의미
- def를 통해 함수를 정의할 수 있음
- Return을 사용해 최종적으로 사용할 값을 반환
- 함수도 이전의 조건문과 같이 들여쓰기 4칸을 이용해 함수 블록을 생성 후 안에서 작업이 가능

함수에 대해 알아보겠습니다. 함수란 객체를 이루기 위한 요소 중 하나로 재사용 가능한 프로그램을 의미합니다. 함수의 구조를 먼저 보도록 하겠습니다. 함수는 첫 번째 줄과 같이 def 키워드를 사용해 정의를 하게 됩니다. 괄호 안에는 이 함수가 받아들일 입력 값, 입력 변수를 지정하게 됩니다.

함수가 호출이 되면 입력 값을 사용해서 함수 내부 두 번째, 세 번째 줄에 명령문들이 실행이 됩니다. 함수 마지막 부분의 return은 함수를 사용했을 때 최종적으로 반환할 값을 명시하는 명령어입니다. 그림 30은 함수구조에 대한 내용입니다.

```
1 def 함수명 (입력값): #함수를 사용해 구조를 효율적으로 만듦
2     수행문1
3     수행문2
4     return 출력값
```

그림 30 함수 구조

함수 예제를 살펴보겠습니다. 첫 번째 줄에서 함수 f를 def 명령어를 사용해 정의합니다. 그리고 괄호 안에서 x의 값을 입력 값으로 지정을 하게 됩니다. 그리고 네 번째 줄에서 return문을 사용해 함수의 반환 값 x+10을 설정합니다.

따라서 함수 f를 실행했을 때 x에 10이 더해진 값이 반환이 됩니다. 다섯 번째 줄에서 함수 f에 입력 값 2를 주고 호출하면 2에 10을 더한 값인 12가 결과 값으로 반환이 되는 것을 확인할 수 있습니다. 그림 31은 함수 예제에 대한 내용입니다.

```
def f(x):
    #함수 f(x)를 생성하고 x의 값을 입력 받음
    #함수를 실행했을 때 결과 값으로 x에 10을 더한 결과가 나오게 함
    return x + 10
f(2)
```

```
12
```

그림 31 함수 예제

5.2 람다(Lambda)

- 함수를 만들되 익명으로 만드는 함수
- 복잡한 함수의 기능을 쓰지 않고 한 줄로 만들 수 있는 함수
- 복잡하게 def 및 return을 사용하지 않고 간단하게 구현이 가능

함수를 앞과 같이 def와 return문을 사용해 정의하고 사용하는 방법 외에 익명으로 만들어 간단하게 사용하는 방법도 있습니다. 예제를 살펴보겠습니다. 두 번째 줄에서는 앞에서 사용한 함수 f와 같은 함수를 선언하고 있습니다. 결과적으로 여섯 번째 줄에서 입력 값 10을 주고 함수 f를 호출하고 그 결과 값을 출력하면 10에 10이 더해진 20이 출력이 됩니다.

결과에서 이것을 확인할 수 있습니다. 함수는 해당 함수를 반복해서 사용하고자 할 때 유용한데, 어떤 연산을 한 번만 수행을 하고 더 이상 다른 곳에서 이용을 하지 않을 경우에는 람다라고 하는 함수를 통해서 일회성 함수를 구현해서 사용을 할 수가 있습니다. 예제 코드의 마지막 아홉 번째 줄을 살펴보겠습니다.

람다 명령어를 사용해서 함수를 선언하고 입력 변수 x에 대한 연산 x+10을 지정합니다. 그리고 입력 변수 x의 값을 10으로 제시합니다. 함수 연산의 결과 값을 print() 함수로 출력하면 20을 얻을 수 있습니다. 그림 32는 람다 예제에 대한 내용입니다.

```
# 함수와 lambda의 차이
def f(x):
    return x+10

# 일반적인 함수를 이용해 함수를 구현
print(f(10))

# lambda를 이용해 함수를 간단하게 구현
print((lambda x: x+10)(10))
```

```
20
20
```

그림 32 람다 예제

함수를 생성할 때 다음과 같은 에러를 주의해야 합니다. 예제를 통해 살펴보겠습니다. 첫 번째 줄에서 선언한 함수 f는 입력 변수 x의 값을 받아서 처리합니다. return 문에서 x가 아닌 y에 10을 더한 값을 반환하고 있습니다. 따라서 함수 f를 사용했을 때 변수가 없는 값인 y를 사용하기 때문에 다음과 같은 에러가 발생하게 됩니다. 그림 33은 함수 에러에 대한 내용입니다. 그림 33은 함수 에러 예제에 대한 내용입니다.

```
1 def f(x):
2     #하지만 변수가 없는 값을 입력했을 때는 에러 발생
3     return y + 10
4 f(2)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-74b33a7545fe> in <module>()
      2     #하지만 변수가 없는 값을 입력했을 때는 에러 발생
      3     return y + 10
----> 4 f(2)

<ipython-input-24-74b33a7545fe> in f(x)
      1 def f(x):
      2     #하지만 변수가 없는 값을 입력했을 때는 에러 발생
----> 3     return y + 10
      4 f(2)

NameError: name 'y' is not defined
```

그림 33 함수 에러 예제

6. 패키지와 라이브러리

6.1 라이브러리란?

- 서브루틴이나 함수들의 집합
- 일반적으로 라이브러리는 파이썬 내부에 있는 정적 라이브러리(내장 라이브러리)를 가리킴
- 다른 패키지의 코드를 가져올 수 있는 동적 라이브러리(외장 라이브러리)등을 패키지라 함
- 내장 라이브러리는 외부에서 패키지를 호출하지 않고 내부적으로 파이썬 자체에 포함되어 있는 라이브러 |
- 외장 라이브러리는 외부의 패키지를 파이썬 내부에서 설치해 사용할 수 있도록 하는 것을 뜻함

파이썬 패키지와 라이브러리에 대해 살펴보겠습니다. 이 강의에서는 패키지와 라이브러리를 같은 의미로 사용하겠습니다. 라이브러리란 서브루틴이나 함수들의 집합으로, 파이썬의 라이브러리는 크게 정적 라이브러리와 동적 라이브러리로 나눌 수 있습니다.

일반적으로 라이브러리는 파이썬 내부에 있는 정적 라이브러리를 의미합니다. 반면 다른 패키지의 코드를 가져올 수 있는 라이브러리를 동적 라이브러리라고 합니다. 내장 라이브러리는 외부 패키지를 호출하지 않고 파이썬 자체에 포함되어 있는 라이브러리입니다. 외장 라이브러리는 외부 패키지를 파이썬 내부에 설치해 사용할 수 있습니다.

아래에 있는 패키지 호출 구조를 살펴보겠습니다. 첫 번째 줄과 같이 패키지 전체는 import를 사용해서 호출이 가능합니다. 하지만 패키지 중 일부 함수만 호출하는 경우에는 두 번째 줄과 같이 from과 import를 사용하는 것이 더 편리합니다.

```
import [package] - 패키지 전체를 호출하는 경우
from [package] import [function] - 패키지 중 일부 함수만 호출하는 경우
```

6.2 내장 함수

- 내장(Built-in)이란 파이썬이 기본적으로 해당 함수를 포함하고 있다는

의미

- import 구문을 이용해 load 하지 않아도 사용 가능한 것들

파이썬의 내장 함수를 자세히 살펴보도록 하겠습니다. 내장이란 파이썬이 기본적으로 해당하는 함수를 포함하고 있다는 것을 의미합니다. 내장 함수는 앞에서와 같이 import 구문을 이용해 불러오지 않아도 사용이 가능합니다. 파이썬의 내장 함수들은 아래 표와 같으며 대표적으로 파일을 열 수 있는 open 함수, 코드를 출력하는 print() 함수 등이 있습니다. 그림 34는 내장 함수 종류를 나타냅니다.

Built-in Functions				
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

그림 34 내장 함수 종류

6.2.1 내장 함수 사용 방법

파이썬 내장 함수를 사용하는 방법에 대해 알아보겠습니다. 다양한 내장 함수들 중에서 input()과 print() 함수를 예제로 살펴보겠습니다. input() 함수는 값을 입력 받는 함수이며, print() 함수는 입력 받은 값을 출력하는 함수입니다. 두 번째 줄에 변수 text에 input()을 이용해 문장 또는 단어를 입력 받을 수 있습니다.

세 번째 줄에서 입력 받은 변수 text에 저장된 데이터를 print() 함수를 사용해서 출력을 해볼 수 있습니다. 따라서 text라는 단어를 입력하면 입력한 단어 text가 출력되는 것을 확인할 수 있습니다. 그림 35는 내장 함수 적용에 대한 내용입니다.

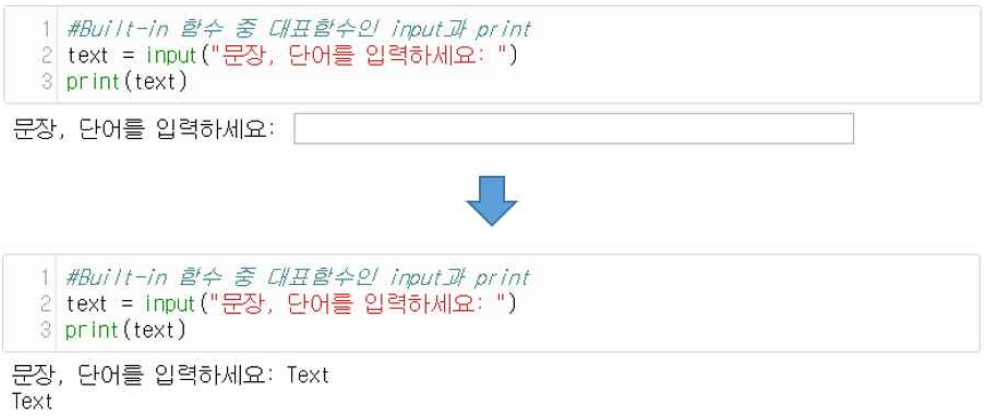

그림 35 내장 함수 적용

6.3 외장 패키지

- 아나콘다를 설치했을 때 아나콘다 파이썬 내부에 설치되어 있는 패키지들이 있음
- 대표적인 패키지는 NumPy, Pandas 등과 같은 데이터를 다루는 패키지
- 아나콘다 내부의 설치된 패키지 또는 설치 가능한 패키지들은 아래의 사이트에서 확인 가능 (https://docs.continuum.io/anaconda/pkg-docs)

내장 함수와 다르게 외장 패키지는 아나콘다를 사용해 파이썬을 설치할 경우 아나콘다 파이썬 내부에 설치되어 있는 패키지들을 예로 들 수 있습니다. 대표적인 외장 패키지 NumPy, Pandas와 같은 데이터를 다루는 패키지들이 있습니다. 이 밖에도 다양한 외장 패키지들이 있는데 아나콘다 내부의 설치된 패키지 또는 설치 가능한 패키지들은 다음의 사이트에서 확인 가능합니다.

6.4 이외의 패키지

- 아나콘다가 포함하지 않는 패키지의 경우 외장 패키지를 설치해 아나콘다 내부에서 사용 가능
- 아나콘다는 기본적으로 많은 패키지들을 보유하고 있지만 그 외의 패키지들이 필요한 경우가 있는데 그 때 다음과 같이 진행
- 외장 패키지를 설치
- 기본적으로 pip라는 패키지 관린 라이브러리를 사용해 외장 패키지들을 설치, 삭제, 수정 가능

- Linux, MacOS

$pip install [라이브러리]

- Windows

C:\Anaconda> pip install [라이브러리]

ex) pip install numpy

아나콘다는 기본적으로 많은 패키지들을 제공하고 있지만 그밖에 포함되지 않는 패키지도 많이 있습니다. 아나콘다가 포함하고 있지 않은 패키지의 경우 패키지 관리 라이브러리를 사용해 설치 후, 아나콘다 내부에서 사용이 가능합니다. 그런 경우에는 다음과 같이 외장 패키지를 설치할 수 있습니다. 외장 패키지를 설치할 때는 pip라는 패키지 관리 라이브러리를 사용해 설치할 수 있습니다.

이때 사용하는 pip를 통해 외장 패키지들을 설치, 삭제, 수정이 가능합니다. 각 운영체제에서의 pip 사용법은 다음과 같습니다. pip를 사용해 외장 패키지를 설치하는 명령문은 pip install numpy와 같습니다.

6.5 패키지 쓰임새

- 각 운영체제에서의 pip 사용법
- pip를 사용해서 외장 패키지를 설치하는 명령문은 pip install numpy와 같음
- 패키지는 각각의 목적에 따라 쓰임이 다름
- 수학적인 면에서는 math, scipy등을 많이 사용하며 컴퓨터의 로그를 분석할 시에는 logging등의 라이브러리를 사용

파이썬 패키지는 각각의 목적에 따라 쓰임이 다릅니다. 수학적 연산이 필요할 때는 math, scipy등의 라이브러리를 사용합니다. 또한 컴퓨터의 로그를 분석할 때는 logging 등의 라이브러리를 사용합니다. 이와 같이 쓰임에 따라 사용하는 파이썬 패키지가 다를 수 있습니다.

A. 웹 프로그래밍 분야 (Flask, Django 등)

- 파이썬에서 Flask, Django 등의 라이브러리를 사용해 웹 프로그래밍이 가능

B. 데이터베이스 분야 (SQLAlchemy, PyMySQL, SQL3, PyMongo 등)

- 파이썬에서 데이터베이스를 연결하는 라이브러리는 이 네 가지가 대표적으로 사용

C. 수학적 분야 (Math, SciPy 등)

- 파이썬에서 수학적 수식 및 계산하는 방법의 라이브러리가 포함되어진 내용

- 위에 설명한 것 이외에도 파이썬 라이브러리들은 다양하게 사용되어지고 있으며 동적으로 pip를 통해 패키지(라이브러리)들을 설치해 사용 가능

앞에서 설명한 파이썬 패키지의 쓰임새는 다음과 같이 구분할 수 있습니다. 웹 프로그래밍 분야에서 사용되는 패키지는 Flask, Django 등이 있습니다. 이와 같은 라이브러리들을 사용해 웹 프로그래밍이 가능합니다.

다음으로 데이터베이스 분야의 경우 SQLAlchemy, PyMySQL, SQL3, PyMongo 등 네 가지 라이브러리가 대표적입니다. 마지막으로 수학적 분야의 라이브러리는 Math, SciPy등의 라이브러리가 있습니다.

앞에서 설명한 파이썬 라이브러리 외에도 라이브러리들은 다양합니다. 이와 같이 다양한 패키지들을 pip를 사용해 동적으로 설치해 사용할 수 있습니다. 따라서 파이썬 패키지들을 분석 또는 개발 목적에 적합하게 선택해 사용할 수 있습니다.

7. NumPy

NumPy는 Numerical Python의 약자로 배열 또는 다양한 자료구조를 다룰 수 있는 클래스들을 포함하고 있는 패키지

7.1 List를 사용한 배열

- 다수의 List를 이용해 다차원의 배열을 쉽게 생성
- 다차원 배열은 다수의 List를 하나로 묶은 배열

7.1.1 List 자료형을 이용한 다차원 배열

리스트 자료형을 이용한 배열에 대해 설명 드리겠습니다. 리스트 자료형을 이용한 배열에서는 먼저 다차원 리스트에 대해 학습하겠습니다. 먼저 예제코드를 보겠습니다. 변수 np에 리스트를 넣기 위해 대괄호를 적고 1.0, 10.4, 3.2, 5.1, 7.5, 2.7의 값을 콤마로 구분해 넣습니다. 그리고 변수 flotV에 np 변수 3개를 리스트 형식으로 넣습니다. 대괄호를 열고 np 3개를 콤마로 구분해 넣습니다. flotV를 출력하면 3 x 6 크기의 2차원 리스트가 출력되는 것을 볼 수 있습니다. 그림 36은 리스트를 이용한 다차원 배열에 대한 내용입니다.

```
1 np = [1.0, 10.4, 3.2, 5.1, 7.5, 2.7] #변수 np에 실수 list 생성
2
3 flotV = [np, np, np] #변수 flotV에 np list를 3개 넣어 다차원 배열 생성
4 flotV
```

```
[[1.0, 10.4, 3.2, 5.1, 7.5, 2.7],
 [1.0, 10.4, 3.2, 5.1, 7.5, 2.7],
 [1.0, 10.4, 3.2, 5.1, 7.5, 2.7]]
```

그림 36 리스트 자료형을 이용한 다차원 배열

변수 np를 이용해 리스트를 생성했습니다. 이번에는 변수 flotV에 할당된 숫자들을 문자열로 변환해보겠습니다. 예제 코드 중 코드 그림 37에 있는 flotV[0]은 리스트의 첫 번째 요소 값을 말합니다. flotV[0]에는 np의 값이 저장되어 있으므로 1.0, 10.4, 3.2, 5.1, 7.5, 2.7의 값이 할당되어 있습니다.

그림 38에 있는 flotV[0]='Python'은 변수 flotV의 0번째 방의 숫자를 'Python'이라는 값으로 변경하게 됩니다. 변수 flotV의 대괄호 사이에 0을 입력 후 문자열 'Python'을 입력한 후 다시 flotV를 출력하면 0번째 값이 Python으로 변환된 것을 볼 수 있습니다. 그림 37과 38은 1차원 배열 값 추

출과 1차원 값을 변환해 다른 값을 대입했을 때의 다차원 배열에 대한 내용입니다.

```
#flotV는 다차원 배열
#flotV[n]을 통해 n차원 배열 값 출력
flotV[0]
```

```
[1.0, 10.4, 3.2, 5.1, 7.5, 2.7]
```

그림 37 1차원 배열 값 추출

```
flotV[0] = 'Python' #flotV[0]의 차원에 위의 값 대신 'Python'값을 입력해 값을 변환
flotV
```

```
['Python', [1.0, 10.4, 3.2, 5.1, 7.5, 2.7], [1.0, 10.4, 3.2, 5.1, 7.5, 2.7]]
```

그림 38 1차원 값을 변환해 다른 값을 대입했을 때 다차원 배열

7.2 NumPy 배열

- NumPy에서 배열 타입을 다루기 위한 클래스는 NumPy.ndarray
- NumPy.ndarray 클래스는 n-차원 배열을 쉽고 효율적으로 다루기 위한 목적

7.2.1 NumPy.ndarray를 이용한 배열 생성

numpy 라이브러리의 ndarray라는 모듈에 대해 설명 드리겠습니다. numpy 라이브러리에는 다양한 모듈이 있는데 그 중 ndarray는 가장 많이 사용되는 모듈 중하나입니다. numpy.ndarray 모듈은 n차원 배열을 쉽고 효율적으로 다루기 위한 목적으로 사용합니다. 다음의 코드는 numpy.ndarray를 이용해 나타낸 예시 코드입니다.

첫 번째 줄에서 numpy 라이브러리를 불러들이기 위해 import라는 명령어를 사용합니다. 그 다음에 import한 라이브러리의 별명을 붙이기 위해 as라는 것을 이용합니다. 그러므로 ‘import numpy as np는 numpy라는 라이브러리를 불러들이고, numpy 라이브러리를 np라는 축약어로 사용하겠다’라고 할 수 있습니다.

세 번째 줄에는 위에서 import한 numpy 라이브러리를 이용해 array 함수에 값을 입력하는 부분입니다. 코드와 같이 입력했을 경우 np.array에 입력한 값 1, 2.5, 4.0, 5.5, 7.0 등의 값을 콤마로 구분해 배열형태로 저장합니다. 네 번

째 줄의 a[3:0]은 변수 a에 저장된 값 중 네 번째 자리부터 끝가지 배열의 데이터를 출력하라는 코드입니다. 즉, 이 경우 5.5와 7.0이 출력됩니다. 이 때 array([5.5, 7.]) 형태로 출력됩니다.

array는 index 값을 통해 해당 주소로 접근하지만 리스트의 경우 데이터까지의 연결통로인 링크를 통해 접근하는 차이점이 있습니다. 배열의 경우 데이터 처리 시 리스트보다 빠른 장점을 갖고 있습니다. 그림 39는 numpy.ndarray 이용한 코드에 대한 내용입니다.

```
1 import numpy as np                    #numpy 패키지를 np라는 명칭으로 축약
2
3 a = np.array([1, 2.5, 4.0, 5.5, 7.0]) #np.array함수 사용한 값을 변수 a에 저장
4 a[3:]                                 #변수 a의 3번째 자리부터 출력
```

```
array([ 5.5,  7. ])
```

그림 39 NumPy.ndarray 코드

7.2.2 다양한 함수를 이용한 NumPy.ndarray 배열

numpy.ndarray 모듈은 다음과 같이 내장 함수 sum, std, cumsum등을 사용해 연산 작업을 할 수 있습니다. 그림 40은 sum 함수를 이용해 변수 a에 입력된 데이터들의 합을 출력하는 부분입니다. 즉, a.sum은 변수 a 안에 속해있는 데이터들을 전부 더하라는 의미입니다.

다음으로 std 함수입니다. std는 standard deviation의 약자로 데이터의 표준편차를 나타내는 함수입니다. 그림 41은 std 함수를 이용해 변수 a의 표준편차를 출력하는 부분입니다. a.std()를 하면 a의 표준편차를 출력할 수 있습니다.

마지막으로 cumsum 함수에 대해 알아보겠습니다. cumsum 함수는 누적합을 나타내는 함수입니다. cumsum 함수를 사용하는 방법은 a.cumsum()을 하면 변수 a에 있는 데이터들의 누적합을 출력할 수 있습니다. 그림 42는 cumsum 함수를 이용해 변수 a의 누적합을 출력하는 부분입니다.

```
1 a.sum() #np.array로 생성한 변수 a의 합을 출력
```

```
20.0
```

그림 40 sum 함수

```
1 a.std() #np.array로 생성한 변수 a의 표준편차를 출력
```

```
2.1213203435596424
```

그림 41 std 함수

```
1 a.cumsum() #np.array로 생성한 변수 a의 누적 합을 출력
```

```
array([ 1. ,  3.5,  7.5, 13. , 20. ])
```

그림 42 cumsum 함수

7.2.3 다양한 산술 연산 적용

numpy는 다양한 산술 연산도 적용할 수 있습니다. 산술 연산을 이용해 변수 a에 대해 연산을 하는 사례를 살펴보겠습니다. 첫 번째 줄에 있는 코드 print()라는 함수를 이용해 변수 a를 출력합니다. 그리고 print(a*2)라고 하면 변수 a의 두 배 값을 출력하게 됩니다.

세 번째 줄의 코드는 *가 두 개입니다. 이것은 변수 a를 제곱하는 의미입니다. 네 번째 줄은 변수 a에 제곱근 연산을 적용한 예입니다. 이 때 numpy 라이브러리의 sqrt 내장 함수를 이용합니다. sqrt를 사용했을 때 변수 a 안에 있는 각각의 데이터의 제곱근 값이 출력되는 것을 볼 수 있습니다. 그림 43은 numpy의 다양한 산술 연산에 대한 내용입니다.

```
1 print(a)          #변수 a의 값을 출력
2 print(a * 2)      #변수 a의 값에 2를 곱한 값을 출력
3 print(a ** 2)     #변수 a의 값에 2를 제곱한 값을 출력
4 print(np.sqrt(a)) #변수 a의 값에 루트를 적용한 값을 출력
```

```
[ 1.  2.5  4.  5.5  7. ]
[ 2.  5.  8.  11.  14.]
[ 1.    6.25  16.    30.25  49.  ]
[ 1.         1.58113883  2.         2.34520788  2.64575131]
```

그림 43 numpy의 다양한 산술 연산

7.2.4 axis 축을 기준으로 합계 표현

axis에 대해서 학습해보겠습니다. axis는 가로나 세로 중 하나를 선택하는 attribute입니다. 즉, axis = 0이라고 하면 세로축으로 연산을 하는 의미입니다.

axis = 1이라고 선언하면 가로축으로 연산을 수행하라는 의미입니다. 코드

를 통해 axis의 활용법에 대해 알아보겠습니다. axis의 활용 예시를 들기 위해 먼저 데이터를 구성해보겠습니다.

b = np.array([a, a**2])라는 문장으로, 변수 a와 a에 2를 제곱한 데이터를 변수 b에 저장하게 됩니다. 여기서 a는 앞서 생성한 변수 a입니다. 변수 a는 1, 2.5, 4, 5.5, 7 값들이 구성되어 있습니다. 결과적으로 b 값은 1, 6.25, 16, 30.25, 49로 구성된 리스트로 이루어진 배열 형태로 존재합니다. 그림 44는 numpy array의 제곱 계산에 대한 내용입니다.

```
#변수 a의 값을 변수 b에 변환해 저장
#a ** 2는 변수 a의 값을 제공
b = np.array([a, a ** 2])
b
```

```
array([[ 1.  ,  2.5 ,  4.  ,  5.5 ,  7.  ],
       [ 1.  ,  6.25, 16.  , 30.25, 49.  ]])
```

그림 44 numpy array 제곱 계산

b.sum(axis = 0)을 하게 되면, 세로축으로 연산을 하게 됩니다. axis가 0인 것은 세로축으로 연산을 하라는 의미입니다. array 안데 있는 각 열의 값을 더하게 됩니다. 즉, 1과 1을 더한 값, 2.5와 6.25를 더한 값, ... 등과 같은 값들이 콤마로 구분되어 출력되게 됩니다. 그림 45는 array의 열 별로 연산한 것에 대한 내용입니다.

```
b.sum(axis = 0) #축(axis)이 0일 경우 세로 축의 합 출력
```

```
array([ 2.  ,  8.75, 20.  , 35.75, 56.  ])
```

그림 45 array의 열 별 연산

b.sum(axis = 1)과 같이 하면 행 별 값이 합해집니다. array 내에 있는 첫 행의 값 1, 2.5, 4, 5.5, 7을 더해 20이라는 결과가 나옵니다. 또 다른 행의 값 1, 6.25, 16, 30.25, 49를 더해 102.5라는 결과가 출력됩니다. 그림 46은 array의 행 별로 연산한 것에 대한 내용입니다.

```
b.sum(axis = 1) #축(axis)이 1일 경우 가로 축의 합 출력
```

```
array([ 20. , 102.5])
```

그림 46 array의 행 별 연산

7.2.5 NumPy.array를 사용한 배열 생성

numpy를 이용해 데이터를 다양하게 표현하는 부분에 대해 학습해보겠습니다. np.array([[0,0,0], [0,0,0]])는 2차원 데이터에 0 값을 입력합니다. 결과적으로 코드에 출력된 것처럼 위에 0이 3개, 아래 0이 3개가 출력됩니다. 그림 47은 np.array()를 이용해 2차원 배열 생성에 대한 내용입니다.

```
1 np.array([[0,0,0],[0,0,0]])
```

```
array([[0, 0, 0],
       [0, 0, 0]])
```

그림 47 np.array 이용해 2차원 배열 생성

7.2.6 NumPy.zeros를 사용한 배열 생성

np.zeros(2,3)은 numpy의 zeros 함수를 이용한 코드입니다. 2행 3열로 0 값을 입력하는 작용을 합니다. 결과적으로 코드에 출력된 것처럼 위에 0이 3개, 아래 0이 3개가 출력됩니다.

np.array와 np.zeros를 실행한 결과는 같은 결과이지만 사용 방법에 따라 코드를 다르게 표현할 수 있습니다. 그림 48은 np.zeros()를 이용해 2차원 배열 생성에 대한 내용입니다.

```
1 # 위의 배열선언과 다른 방법으로 같은 모양 배열 생성
2 # np.zeros를 통해 값을 설정하고 type을 설정해 배열을 생성 가능
3
4 values = np.zeros((2,3), dtype='i') # dtype = i : 정수형, dtype = f : 복소수형
5 values
```

```
array([[0, 0, 0],
       [0, 0, 0]], dtype=int32)
```

그림 48 np.zeros 이용해 2차원 배열 생성

여기까지 numpy에 대해 알아보았습니다. 다음에는 데이터 분석에서 가장 중요한 것 중 하나인 pandas 라이브러리에 대해 알아보겠습니다.

8. Pandas

- Pnadas는 NumPy와 같이 데이터를 다루는데 있어 많이 사용되는 패키지
- 빠른 속도로 데이터 분석 가능
- CSV 파일 또는 데이터베이스로부터 데이터를 쉽게 읽고 쓸 수 있음
- Column을 통해 데이터를 조작, 새로운 Column 생성 가능
- Pandas의 자료구조로는 Series와 DataFrame이 있음

8.1 Series

- Series는 1차원으로 되어 있으며 index와 value 형태를 갖는 Pandas 내 자료구조

8.1.1 Series의 Index와 Value

예제 코드는 pd.series를 이용해 index와 value 값을 할당하여 나타냈습니다. 코드를 보시면 먼저 pandas와 numpy를 pd와 np라 축약해 라이브러리를 불러옵니다.

그리고 pandas_series 변수에 pd.series([3000, 3200, 2700], index = [‘2016-11-10’, ‘2016-11-11’, ‘2016-11-12’])와 같이 넣습니다. 여기서 3000, 3200, 2700은 데이터이며, ‘2016-11-10’, ‘2016-11-11’, ‘2016-11-12’는 index입니다. index ‘2016-11-10’에는 3000, ‘2016-11-11’에는 3200이 할당됩니다.

pandas_series 변수는 type 함수를 이용할 때 pandas.core.Series라는 모듈 type이 출력되는 것을 알 수 있습니다. 이것은 pandas series 자료구조 형태 type이라는 뜻입니다. 결과적으로 2016-11-10에는 3000이 할당되고, 2016-11-11에는 3200이, 2016-11-12에는 2700이 할당됩니다. 그림 49는 series의 index와 value에 대한 내용입니다.

```
import pandas as pd #pandas 패키지를 pd라는 명칭으로 축약해 호출
import numpy as np #numpy 패키지를 np라는 명칭으로 축약해 호출

#Series의 Value와 Index를 변수 pandas_series에 선언
pandas_series = pd.Series([3000, 3200, 2700],
                          index = ['2016-11-10', '2016-11-11', '2016-11-12'])

print(type(pandas_series)) #변수 pandas_series의 type 출력
pandas_series              #변수 pandas_series 실행
```

```
<class 'pandas.core.series.Series'>

2016-11-10    3000
2016-11-11    3200
2016-11-12    2700
dtype: int64
```

그림 49 Series의 index, value

8.1.2 Series를 통해 원하는 위치의 값 출력

series를 slicing하는 방법에 대해 알아보겠습니다. numpy처럼 series도 데이터를 slicing할 수 있습니다. 만약 변수 pandas_series 2번째 자리부터 값을 출력할 경우 pandas_series[1:]처럼 작성할 수 있습니다. 이럴 때 pandas_series 2번째 자리부터 series 제일 마지막까지의 값을 순차적으로 출력합니다. 예제에서 pandas_series 2번째 자리 2016-11-11은 3200이, 3번째 자리 2016-11-12는 2700을 출력합니다. 그림 50은 series slicing에 대한 내용입니다.

```
pandas_series[1:] #Series의 2번째 자리부터의 값 출력
```

```
2016-11-11    3200
2016-11-12    2700
dtype: int64
```

그림 50 Series slicing

8.2 DataFrame

- DataFrame은 index(행)와 label(열, column)으로 구분

데이터프레임은 index와 label로 구분할 수 있습니다. index는 행을 가리키며 label은 열을 가리킵니다. pandas 데이터프레임은 value, column 그리고 index를 이용해 구성할 수 있습니다.

	Label명 #0	Label명 #1
Index명 #0	Data[0, 0]	Data[1, 0]
Index명 #1	Data[0, 1]	Data[1, 1]

8.2.1 DataFrame 예

그림 51은 데이터프레임의 예제에 대한 내용입니다. 예제 코드를 보겠습니다. numpy와 pandas 라이브러리를 np와 pd로 축약해 라이브러리를 불러옵니다.

그리고 변수 df에 pd.DataFrame([100, 150, 200, 250, 300], columns = [‘numbers’], index = [‘a’, ‘b’, ‘c’, ‘d’, ‘e’])을 같이 할당합니다. 여기서 100, 150, 200, 250, 300은 데이터를 말하고 index가 a, b, c, d, e이므로 a는 100, b는 150, c는 200 등과 같이 할당합니다. 그리고 columns는 데이터프레임 컬럼의 이름을 말합니다. 코드에서는 numbers라는 이름을 붙였습니다.

```
1 import pandas as pd #pandas 패키지를 pd라는 명칭으로 축약해 호출
2 import numpy as np #numpy 패키지를 np라는 명칭으로 축약해 호출
3
4 #DataFrame의 Value, Column, Index를 변수 df에 선언
5 df = pd.DataFrame([100, 150, 200, 250, 300], columns = ['numbers'],
6                   index = ['a', 'b', 'c', 'd', 'e'])
7 df
```

	numbers
a	100
b	150
c	200
d	250
e	300

그림 51 DataFrame 예제

8.2.2 DataFrame 객체를 사용하는 예

데이터프레임 객체를 사용하는 예제에 대해 알아보겠습니다. df.index를 이용해 데이터프레임의 index 값이 무엇인지 확인할 수 있습니다. 결과적으로 a, b, c, d, e가 출력됩니다. 그림 52는 데이터프레임 index 객체에 대한 내용입니다.

```
1 df.index #DataFrame의 index를 표시
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

그림 52 DataFram index 객체

그리고 df.columns는 컬럼의 이름을 출력합니다. 앞선 코드에서 df 컬럼명에 numbers라는 값을 넣었으므로 numbers가 출력됩니다. 그림 53은 데이터프레임 column 객체에 대한 내용입니다.

```
1 df.columns #DataFrame의 column을 표시
```

```
Index(['numbers'], dtype='object')
```

그림 53 DataFrame columns 객체

마지막으로 df.ix를 이용해 index에 해당하는 값을 출력할 수 있습니다. df.ix[‘c’]를 출력하면 index 중 c에 해당하는 컬럼 값 200이 출력됩니다. 그림 54는 데이터프레임 ix 객체에 대한 내용입니다.

```
1 df.ix['c'] #DataFrame의 index 중 c에 해당하는 값 출력
```

```
numbers    200
Name: c, dtype: int64
```

그림 54 DataFrame ix 객체

8.2.3 DataFrame 연산

데이터프레임을 이용해 연산하는 방법에 대해 알아보겠습니다. df.sum()을 하게 되면 numbers 컬럼의 값을 모두 더하여 1000이라는 값을 출력합니다. 그림 55는 데이터프레임 sum 연산에 대한 내용입니다.

```
df.sum() #DataFrame의 Value를 모두 더함
```

```
numbers    1000
dtype: int64
```

그림 55 DataFrame sum 연산

다음으로 df.numbers ** 2를 하면 numbers 컬럼의 각각의 값을 제곱해 출력합니다. 결과적으로 a는 100을 제곱한 10000, b는 150을 제곱한 22500, c는 200을 제곱한 40000, d는 250을 제곱한 62500, e는 300을 제곱한 90000이 출

력됩니다. 그림 56은 데이터프레임 제곱 연산에 대한 내용입니다.

```
df.numbers ** 2 #DataFrame의 numbers 칼럼의 각 값을 제곱

a    10000
b    22500
c    40000
d    62500
e    90000
Name: numbers, dtype: int64
```

그림 56 DataFrame 제곱 연산

8.2.4 DataFrame의 Column 추가

데이터프레임에 컬럼을 추가하고 거기에 값을 넣는 방법에 대해 알아보겠습니다. 데이터프레임 구조를 가진 변수 df['values'] = (10, 50, 40, 30, 60)와 같이 하게 되면 values라고 하는 컬럼이 새로 추가되면서 그 컬럼에 10, 50, 40, 30, 60이라는 값이 배정됩니다.

여기서 주의할 점은 데이터프레임에 컬럼을 추가할 때 데이터 개수는 기존 데이터 개수와 동일해야 합니다. 즉, 데이터프레임 df의 경우 데이터 개수가 총 5개이므로 새로 추가하려는 컬럼의 데이터 개수도 동일하게 5개로 맞춰야 합니다. 그림 57은 데이터프레임에서 컬럼 추가에 대한 내용입니다.

```
1 #DataFrame에 values라는 칼럼을 추가하고 값을 입력
2 df['values'] = (10, 50, 40, 30, 60)
3 df
```

	numbers	values
a	100	10
b	150	50
c	200	40
d	250	30
e	300	60

그림 57 DataFrame 컬럼 추가

8.2.5 DataFrame에서 Column을 추가하는 방법

지금까지 데이터프레임에 컬럼명을 이용해 데이터를 추가하는 방법을 학습했습니다. 데이터프레임에 컬럼을 추가할 때 index를 이용해 데이터를 할당해보겠습니다. 예제 코드를 보시면 df[‘values’]는 values 컬럼에 값을 넣는다는 의미입니다. values 컬럼에 pd.DataFrame([Fourth, Second, First, Fifth, Third])와 같이 순서가 뒤섞인 데이터를 넣습니다. 이 때, index를 함께 넣어주도록 하겠습니다.

index = [‘d’, ‘b’, ‘a’, ‘e’, ‘c’]를 방금 전의 데이터 값에 이어 입력하여 주면 a, b, c, d, e index에는 First, Second, Third, Fourth, Fifth 값이 자동으로 할당됩니다. 지금까지 데이터프레임을 추가, 변경하는 방법에 대해 학습했습니다. 그림 58은 데이터프레임에서 index를 이용한 컬럼 추가에 대한 내용입니다.

```
#DataFrame의 index의 값에 일치하는 값을 넣음
df['values'] = pd.DataFrame(['Fourth', 'Second', 'First', 'Fifth', 'Third'],
                            index = ['d', 'b', 'a', 'e', 'c'])
df
```

	numbers	values
a	100	First
b	150	Second
c	200	Third
d	250	Fourth
e	300	Fifth

그림 58 DataFrame index를 이용한 컬럼 추가

8.2.6 DataFrame의 Column 삭제

del이라는 함수를 이용해 values 컬럼을 삭제해보겠습니다. 예제 코드를 보시면 del df[‘values’] 코드로 values 컬럼을 삭제할 수 있습니다. 이 문장을 실행하면 values 컬럼이 삭제된 것을 보실 수 있습니다. 그림 59는 데이터프레임의 컬럼 삭제에 대한 내용입니다.

```
#DataFrame의 values라는 칼럼을 삭제
del df['values']
df
```

	numbers
a	100
b	150
c	200
d	250
e	300

그림 59 DataFrame 컬럼 삭제

8.2.7 DataFrame에서 index를 맞추지 않았을 때 발생하는 오류

데이터 개수를 맞추는 부분에 대해 학습하겠습니다. 데이터 개수를 일치시키는 것은 pandas에서 중요합니다. 데이터프레임에 데이터를 추가할 때 index 또는 데이터 개수를 일치시키지 않을 경우 발생하는 오류를 예제 코드에서 확인 할 수 있습니다.

df의 numbers 값은 100, 150, 200, 250, 300 이렇게 5개입니다. df['values'] = (10, 50, 40) 같이 데이터를 3개만 넣으면 데이터 수가 기존 데이터 개수인 5개와 맞지 않으므로 에러가 발생합니다. 결과적으로 기존 데이터 길이와 현재 넣으려는 데이터의 길이를 일치시키도록 해야 합니다. 그림 60은 데이터프레임에서 index 오류에 대한 내용입니다.

```
df['values'] = (10,50,40) # DataFrame의 인덱스 수를 맞추지 않으면 에러 발생
df
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-53-8c6bf4219d31> in <module>()
----> 1 df['values'] = (10,50,40) # DataFrame의 인덱스 수를 맞추지 않으면 에러 발생
      2 df

C:\Users\Administrator\Anaconda2\envs\py34\lib\site-packages\pandas\core\frame.py in _
_setitem__(self, key, value)
   2297         else:
   2298             # set column
-> 2299             self._set_item(key, value)
   2300
   2301     def _setitem_slice(self, key, value):

C:\Users\Administrator\Anaconda2\envs\py34\lib\site-packages\pandas\core\frame.py in _
set_item(self, key, value)
```

그림 60 DataFrame index 오류

8.2.8 DataFrame에서 join 사용방법

서로 다른 데이터프레임을 하나의 데이터프레임으로 합치는 방법에 대해 알아보겠습니다. pandas에서 두 개의 데이터프레임을 합치기 위해 join, merge, concat 등의 함수를 사용할 수 있습니다. 예제 코드에서는 join을 이용해 서로 다른 두 데이터프레임을 합쳐보겠습니다.

먼저 변수 df_1에 pd.DataFrame([‘1’, ‘2’, ‘3’])을 넣고 컬럼 이름을 A로 할당합니다. df_2에는 pd.DataFrame([‘4’, ‘5’, ‘6’, ‘7’])을 넣고 컬럼 이름을 B로 할당했습니다. 서로 다른 두 데이터프레임 A와 B를 join을 이용해 합쳐보겠습니다.

df_1.join(df_2, how = ‘outer’)는 df_1의 옆에 df_2를 붙이겠다는 뜻입니다. 여기서 attribute how는 두 데이터프레임을 합치는 방법을 말합니다. 예제 코드에서는 outer를 이용해 데이터를 합치며 변수 df에 저장합니다. 결과적으로 A 컬럼에 1, 2, 3, NaN이 들어있으며 B에는 4, 5, 6, 7이 들어가 두 데이터프레임이 합쳐진 것을 볼 수 있습니다. 그림 61은 데이터프레임에서 join을 사용한 내용입니다.

```
#Column A에 1, 2, 3 값 넣음
df_1 = pd.DataFrame(['1', '2', '3'], columns = ['A'])

#Column B에 4, 5, 6, 7 값 넣음
df_2 = pd.DataFrame(['4', '5', '6', '7'], columns = ['B'])

df = df_1.join(df_2, how = 'outer') #두 개의 DataFrame을 outer join으로 합침
df #합친 DataFrame 출력
```

	A	B
0	1	4
1	2	5
2	3	6
3	NaN	7

그림 61 DataFrame join 사용

8.2.9 DataFrame에 난수를 이용한 임의의 값 생성

데이터프레임에서 임의의 난수를 생성해 최댓값, 최솟값을 연산하는 예제를 학습하겠습니다. 난수란 임의의 범위 내에 발생되는 무작위수 즉, 난수 데이터(random data)를 말합니다. numpy의 random 함수는 임의의 범위 내에서 데이터를 생성하는 모듈입니다.

np.rnadom.rand(5, 5)는 가로 5, 세로 5의 난수 데이터를 생성하라는 의미입니다. 이 때, 생성된 난수 데이터를 pd.DataFrame을 이용해 변수 df에 저장합니다. 그리고 변수 df의 컬럼명을 A, B, C, D, E로 넣습니다. 결과적으로 가로 5, 세로 5 크기의 A, B, C, D, E를 컬럼으로 갖는 데이터프레임이 만들어집니다. 그림 62는 데이터프레임의 난수 생성에 대한 내용입니다.

```python
import pandas as pd
import numpy as np

# random값을 넣은 DataFrame을 생성
# np.random.randn은 5행 5열의 랜덤 데이터를 생성
# 재호출 시 데이터가 변형되므로 주의
df=pd.DataFrame(np.random.randn(5,5))
df.columns=['A','B','C','D','E']

df
```

	A	B	C	D	E
0	-0.115540	-0.023043	0.994561	1.523005	0.623538
1	1.292809	-0.408988	-0.990792	-0.729662	-1.575598
2	0.944633	0.031914	0.492421	1.302999	0.527092
3	0.064526	-2.138368	-1.260443	-0.059801	0.911413
4	-0.094616	-1.721012	0.617084	0.722013	0.256642

그림 62 DataFrame 난수 생성

8.2.10 DataFrame을 이용한 연산 - 최댓값

데이터프레임은 최댓값을 구하기 위해 max 함수를 사용합니다. 따라서 df.max()라고 하면 각 컬럼의 최댓값을 출력하게 됩니다. 출력결과 A의 최댓값은 2.17, B의 최댓값은 2.71 등이 나오는 것을 볼 수 있습니다. 예제 코드를 각자 수행하는 경우 그 때 발생하는 난수는 달라서 출력 값은 다를 수 있습니다. 그림 63은 데이터프레임의 최댓값에 대한 내용입니다.

```python
# 각 Column별 최대값 출력
df.max()
```

```
A    2.170760
B    2.710702
C    1.175449
D    0.423712
E    0.897418
dtype: float64
```

그림 63 DataFrame 최댓값 연산

8.2.11 DataFrame을 이용한 연산 - 최솟값

최댓값과 반대로 최솟값을 구하는 방법으로 min 함수를 이용합니다. df.min()을 이용했을 때 각 컬럼의 최솟값을 구할 수 있습니다. 결과적으로 A의 최솟값 -2.74, B는 -1.09 등이 출력됩니다. 그림 64는 데이터프레임의 최솟값에 대한 내용입니다.

```
# 각 Column별 최솟값 출력
df.min()
```

```
A   -2.748275
B   -1.093980
C   -1.936413
D   -1.206294
E   -0.155345
dtype: float64
```

그림 64 DataFrame 최솟값 연산

8.2.12 DataFrame을 이용한 연산 - 평균값

데이터프레임을 저장한 변수 df의 평균을 구해보겠습니다. 평균값은 df.mean()을 이용해 각 컬럼의 평균값을 출력할 수 있습니다. 결과적으로 A의 평균은 0.24, B의 평균은 0.22, C의 평균은 -0.46과 같은 값이 나옵니다. 그림 65는 데이터프레임의 평균값에 대한 내용입니다.

```
# 각 Column별 평균값 출력
df.mean()
```

```
A    0.240514
B    0.225153
C   -0.462852
D   -0.457487
E    0.381158
dtype: float64
```

그림 65 DataFrame 평균값 연산

8.2.13 DataFrame을 이용한 연산 - 표준편차

df.std()와 같은 함수를 이용해 각 컬럼의 표준편차를 출력할 수 있습니다. 결과적으로 A는 1.97, B는 1.45, C는 1.22의 값이 나옵니다. 그림 66은 데이터프레임의 표준편차에 대한 내용입니다.

```
# 각 Column별 표준편차 값 출력
df.std()
```

```
A    1.976495
B    1.458724
C    1.223023
D    0.668688
E    0.482414
dtype: float64
```

그림 66 DataFrame 표준편차 연산

8.2.14 DataFrame을 이용한 연산 - 누적합

데이터프레임의 누적합 연산은 cumsum()이라는 함수를 이용해 누적합을 구합니다. df.cumsum()과 같이 하면 각 컬럼의 누적합을 구합니다. A의 0번째

에는 2.17, B의 0번째에는 -1.09 등의 값이 출력됩니다. 그림 67은 데이터프레임의 누적합에 대한 내용입니다.

```
# 각 Column별 누적합값 출력
df.cumsum()
```

	A	B	C	D	E
0	2.170760	-1.093980	-1.936413	-0.848546	0.897418
1	1.778379	-1.065675	-0.760965	-0.812715	1.345334
2	-0.969896	-1.119570	-0.698310	-2.019009	1.272538
3	-0.637447	-1.584939	-0.952504	-1.595297	2.061132
4	1.202568	1.125763	-2.314262	-2.287435	1.905788

그림 67 DataFrame 누적합 연산

8.3 Describe 함수와 Group by를 이용한 DataFrame

8.3.1 Describe 함수를 이용한

pandas 데이터프레임을 이용해 통계적 분포를 구해보겠습니다. 통계적 분포란 count, mean, std, min, 25%, 50%, 75%와 같이 통계적으로 계산한 값을 말합니다. pandas에는 describe 모듈을 이용해 통계적 분포를 확인할 수 있습니다. count는 변수에 저장된 각 컬럼의 데이터 수를 나타냅니다. df의 각 컬럼에 해당하는 데이터 수는 5개의 난수를 생성했으므로 5를 출력합니다.

그리고 mean은 평균값을 각 컬럼의 평균값을 나타냅니다. A는 0.24, B는 0.22 등의 값을 출력합니다. std는 각 컬럼의 표준편차를 나타냅니다. A는 1.97, B는 1.45 등의 값이 출력됩니다. 이외에도 describe를 이용해 25%, 50%, 75%의 값을 구할 수 있습니다.

25%는 컬럼에서의 25% 즉, $\frac{1}{4}$위치의 값을 말합니다. A는 -0.39, B는 -0.46이 출력됩니다. 50%는 컬럼에서의 50% 즉, $\frac{1}{2}$위치의 값을 말합니다. A는 0.44, B는 -0.05 등이 출력됩니다. 75%는 컬럼에서의 75% 즉, $\frac{3}{4}$위치의 값을 말합니다. 결과적으로 A는 1.84, B는 0.02 등이 출력됩니다. 그림 68은 데이터프레임의 통계적 분포에 대한 내용입니다.

```
# Describe함수를 이용해 DataFrame 데이터의 통계 요약값 표현
df.describe()
```

	A	B	C	D	E
count	5.000000	5.000000	5.000000	5.000000	5.000000
mean	0.240514	0.225153	-0.462852	-0.457487	0.381158
std	1.976495	1.458724	1.223023	0.668688	0.482414
min	-2.748275	-1.093980	-1.936413	-1.206294	-0.155345
25%	-0.392381	-0.465369	-1.361758	-0.848546	-0.072795
50%	0.332449	-0.053895	-0.254194	-0.692138	0.447916
75%	1.840015	0.028305	0.062654	0.035831	0.788594
max	2.170760	2.710702	1.175449	0.423712	0.897418

그림 68 DataFrame의 통계적 분포 확인

8.3.2 Group by를 이용해 DataFrame의 그룹화

group by를 이용해 데이터프레임의 그룹화에 대해 알아보겠습니다. 예제 코드와 같이 df[‘division’]에 [‘X’, ‘Y’, ‘X’, ‘Y’, ‘Z’]의 값을 입력합니다. division 컬럼은 데이터를 그룹별로 분석하는 기준이 됩니다. 그림 69는 데이터프레임의 그룹화에 사용할 기준을 삽입하는 내용입니다.

```
# Group by하기 전 그룹별로 구분하기 위해 division이라는 Column을 생성
df['division']=['X','Y','X','Y','Z']
df
```

	A	B	C	D	E	division
0	2.170760	-1.093980	-1.936413	-0.848546	0.897418	X
1	-0.392381	0.028305	1.175449	0.035831	0.447916	Y
2	-2.748275	-0.053895	0.062654	-1.206294	-0.072795	X
3	0.332449	-0.465369	-0.254194	0.423712	0.788594	Y
4	1.840015	2.710702	-1.361758	-0.692138	-0.155345	Z

그림 69 DataFrame 그룹화 기준 삽입

df.groupby([‘division’])은 division 컬럼을 기준으로 그룹화를 합니다. 앞에서 division 컬럼에 X, Y, Z를 넣었으므로 X, Y, Z를 기반으로 그룹화가 됩니다. 뒤의 mean()은 X, Y, Z의 값을 평균으로 출력한다는 의미입니다. A, B, C, D, E 컬럼이 division 컬럼의 X, Y, Z를 기준으로 평균값이 출력됩니다.

A 컬럼 X 의 평균값은 -0.28, B 컬럼 X의 평균값은 -0.57, C 컬럼 X의

평균값은 -0.93 등과 같이 출력됩니다. 그림 70은 데이터프레임의 그룹화가 진행되고 그룹화에 대한 평균값을 나타내는 내용입니다.

```
# DataFrame에 Group by를 사용해 Column 'division의 값에 따라 평균값을 산출
df.groupby(['division']).mean()
```

	A	B	C	D	E
division					
X	-0.288758	-0.573938	-0.936880	-1.027420	0.412311
Y	-0.029966	-0.218532	0.460627	0.229771	0.618255
Z	1.840015	2.710702	-1.361758	-0.692138	-0.155345

그림 70 DataFrame 그룹화 결과