03. 파이썬 기본 자료 구조

2 장에서는 파이썬의 기본 데이터 타입인 정수형, 실수형, 문자열을 배웠습니다. 이번 장에서는 정수형, 실수형, 문자열 등의 데이터가 여러 개 있을 때 이를 효과적으로 관리하는 데 사용되는 자료구조에 대해 배우겠습니다. 파이썬에는 여러 가지 자료구조가 있는데 그중 가장 많이 사용되는 리스트(list), 튜플(tuple), 사전(dict)에 대해 다룹니다. 이러한 세 가지 자료구조는 파이썬으로 프로그래밍할 때 자주 사용되므로 반드시 잘 익혀두시기 바랍니다.

1) 리스트

유가증권(코스피)과 코스닥을 합치면 대략 2000 종목 정도가 있는데 그중에서 관심 있는 종목이 삼성전자, LG 전자, 네이버라고 가정해 봅시다. HTS(Home Trading System)에 관심 종목이 나오는 것처럼 관심 종목을 파이썬으로 관리해야 한다면 어떻게 하면 될까요? 아마도 2 장에서 배운 변수를 이용해 다음과 같이 관심 종목명을 문자열로 표현한 후 가리키게 할 수 있을 것입니다.

```
>>> interest1 = "삼성전자"
>>> interest2 = "LG 전자"
>>> interest3 = "네이버"
>>>
```

2 장에서 설명했듯이 파이썬의 변수는 C/C++ 같은 프로그래밍 언어와 달리 실제 데이터가 저장되는 공간 자체가 아니라 데이터가 메모리상에서 위치하는 주솟값이 저장되는 공간입니다. 파이썬에서는 그림 3.1 과 같이 변수가 실제 데이터가 존재하는 위치를 가리키고 있는 것을 바인딩(binding)한다고 표현합니다. 바인딩, 바인딩, 바인딩 ... 딱 10 번만 반복해봅시다. 저자가 지금은 '가리킨다'라는 표현과 '바인딩'이라는 표현을 의도적으로 섞어서 사용하고 있는데 이 책의 후반부로 갈수록 '바인딩'이라는 표현을 주로 사용할 것입니다.

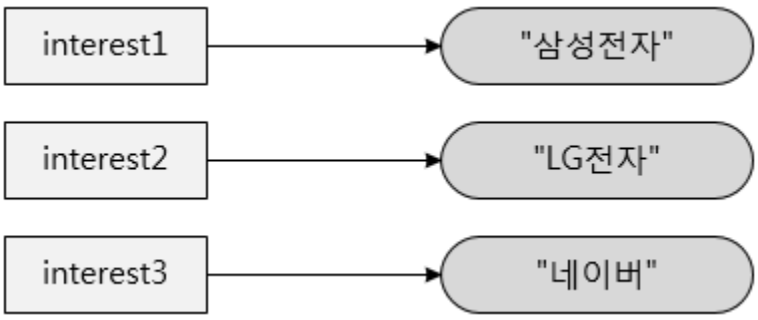

그림 3.1 파이썬 바인딩의 예

파이썬에서 변수가 어떤 값을 바인딩하고 있으면(가리키고 있으면), 프롬프트 상에서 변수명을 입력한 후 엔터 키를 누르면 해당 변수가 가리키는 값이 반환됩니다. 즉, 다음과 같이 interest1 을 입력한 후 엔터 키를 누르면 interest1 변수가 가리키고 있던 문자열 데이터인 '삼성전자'가 반환됩니다.

```
>>> interest1
'삼성전자'
>>>
```

위와 같이 관심 종목의 수가 적은 경우에는 각자 서로 다른 변수명을 사용해서 관심 종목을 바인딩하고 있을 수 있지만 그래도 뭔가 불편해 보입니다. 지금과 같은 방식에서는 관심 종목의 수가 100 개라면 이를

표현하기 위해 interest1, interest2, interest3, ..., interest100 과 같이 100 개의 변수명을 사용해야 할 것입니다.

이러한 불편을 해결하고자 파이썬에서는 리스트(list)라는 기본 자료구조를 제공합니다. 리스트를 이용하면 여러 데이터를 한 번에 저장할 수 있으므로 변수명을 여러 개 사용할 필요가 없습니다. 여러 개의 데이터를 리스트에 담으려면 '['와 ']' 사이에 데이터를 쉼표로 구분해서 넣기만 하면 됩니다. 예를 들어, 앞의 예제와 같이 삼성전자, LG 전자, 네이버를 관심 종목이라고 할 때 이를 리스트로 만들려면 다음과 같이 작성하면 됩니다.

```
>>> interest = ["삼성전자", "LG 전자", "네이버"]
```

파이썬의 리스트를 이용하면 여러 데이터를 한 번에 가리킬 수 있으므로 interest1, interest2, interest3 과 같이 각각 변수명을 사용하는 것이 아니라 interest 처럼 한 개의 변수만 있으면 됩니다. 파이썬에서 여러 데이터를 리스트를 통해 한 번에 저장하고 관리하는 것은 그림 3.2 처럼 출석부에 학생들의 이름을 적어 두는 것과 비슷한 개념입니다.

출석부 (코스피/코스닥)

번호	학년/반	성명	1	2	3	4	5	6	7
1		삼성전자							
2		LG전자							
3		네이버							
4									
5									

그림 3.2 파이썬 리스트와 출석부

학창 시절에 선생님께서 출석부를 보시고 1 번이라고 호명하면 1 번 학생이 대답했던 것처럼 파이썬에서도 리스트를 통해 바인딩하고 있는 데이터들은 리스트의 인덱스 값을 통해 접근할 수 있습니다. 단, 출석부의 번호가 1 번부터 시작했던 것과 달리 파이썬 리스트의 인덱스는 0 번부터 시작하기 때문에 이를 고려해야 합니다. 즉, 리스트의 첫 번째 데이터에 접근하려면 1 이 아니라 0 을 사용하면 됩니다.

다음 코드는 [0], [1], [2]라는 인덱스 값을 통해 interest 리스트에 있는 첫 번째, 두 번째, 세 번째 관심 종목 데이터를 차례대로 얻어옵니다.

```
>>> interest = ["삼성전자", "LG 전자", "네이버"]
>>> interest[0]
'삼성전자'
>>> interest[1]
```

```
'LG 전자'
>>> interest[2]
'네이버'
>>>
```

1-1) 리스트 생성

3.1 절에서 파이썬 기본 자료구조 중 하나인 리스트에 대해 간단히 살펴봤습니다. 그렇다면 리스트는 언제 사용하는 것이 좋을까요? 바로 여러 개의 데이터를 '순서대로' 저장하고 이를 관리해야 할 때입니다.

주식 분석 프로그램을 만드는데 '대신증권'이라는 종목의 최근 5 일 치의 종가인 9130, 9150, 9150, 9300, 9400 원을 저장해야 하는 경우를 생각해 봅시다. 이 경우 다음과 같이 파이썬의 리스트를 이용하면 종가를 '순서대로' 저장할 수 있습니다.

```
>>> daishin = [9130, 9150, 9150, 9300, 9400]
>>>
```

파이썬의 리스트에 정수 데이터만 넣을 수 있는 것은 아닙니다. 다음과 같이 실수 데이터로도 리스트를 생성할 수 있습니다.

```
>>> daishin = [9130.0, 9150.0, 9150.0, 9300.0, 9400.0]
>>>
```

물론 앞서 살펴본 것처럼 문자열로도 리스트를 만들 수 있습니다. 이뿐만 아니라 서로 다른 자료형을 동시에 넣을 수도 있습니다. 네이버 주식을 5,000 주 가지고 있어서 이를 리스트로 표현한다면 다음과 같이 작성할 수 있습니다.

```
>>> mystock = ['Naver', 5000]
>>>
```

파이썬의 리스트를 만들 때 항상 데이터가 있어야 하는 것은 아닙니다. 다음과 같이 리스트를 만들 때 '['와 ']' 기호 사이에 데이터를 넣지 않으면 아무것도 들어 있지 않은 빈(empty) 리스트가 만들어집니다.

```
>>> mystock = []
>>>
```

1-2) 리스트 인덱싱

파이썬 리스트의 인덱스 값을 통해 리스트가 가리키고 있는 값에 접근할 수 있습니다. 리스트의 인덱스 역시 문자열 인덱스처럼 0 부터 시작합니다.

다음 코드는 'Naver'라는 종목의 주식을 5,000 주 매수하기 전에 매수할 종목명과 매수할 주식 수를 리스트로 표현한 것입니다. 리스트는 데이터를 순서대로 저장하므로 리스트의 첫 번째 항목에는 종목명이 있고, 두 번째 항목에는 매수할 주식 수가 있습니다. 따라서 buy_list[0]이라는 표현을 통해 리스트의 첫 번째 항목으로 저장된 'Naver'라는 문자열 데이터를 가져올 수 있습니다.

```
>>> buy_list = ['Naver', 5000]
>>> buy_list[0]
'Naver'
>>> buy_list[1]
5000
>>>
```

마찬가지로 대신증권의 최근 5 일 치 종가가 담긴 리스트를 만든 후 인덱싱을 통해 각 종가를 가져올 수 있습니다. 파이썬의 인덱싱은 0 부터 시작하기 때문에 daishin 이라는 리스트에 있는 첫 번째 데이터인 9130 을 가져오려면 daishin[0]이라고 입력하면 됩니다.

```
>>> daishin = [9130, 9150, 9150, 9300, 9400]
>>> daishin[0]
9130
>>> daishin[1]
9150
>>> daishin[2]
9150
>>> daishin[3]
```

```
9300
>>> daishin[4]
9400
>>>
```

지금까지 살펴본 것처럼 파이썬 리스트의 인덱싱은 문자열 인덱싱과 동일합니다. 2 장의 문자열 인덱싱에서 배운 것처럼 인덱싱 값으로 음수를 사용하면 리스트의 뒤쪽부터 데이터에 접근할 수 있습니다. 단, 음수를 사용할 때는 -1 부터 사용할 수 있습니다. 왜냐하면, 0 은 이미 리스트의 맨 앞에 있는 데이터를 의미하기 때문입니다.

리스트를 인덱싱할 때 [-1]은 리스트의 데이터 중 맨 마지막에 있는 데이터를 의미합니다. 마찬가지로 [-2]는 리스트의 데이터 중 끝에서 두 번째 항목을 의미합니다. 다음 코드에서 대신증권 데이터를 예로 들어 설명하면 daishin[-1]의 값은 9400 으로 리스트에 맨 마지막에 위치하는 데이터임을 알 수 있습니다.

```
>>> daishin = [9130, 9150, 9150, 9300, 9400]
>>> daishin[-1]
9400
>>> daishin[-2]
9300
>>> daishin[-3]
9150
>>> daishin[-4]
9150
>>> daishin[-5]
9130
>>>
```

위 코드에서 daishin 리스트에 대한 인덱싱을 그림으로 표현하면 그림 3.3 과 같이 나타낼 수 있습니다. 인덱싱할 때 언제 양수 인덱싱을 사용하고 음수 인덱싱을 사용할지에 대한 규칙이나 제약은 없습니다. 프로그래밍할 때 필요에 따라 적절하게 사용하면 되는 것입니다.

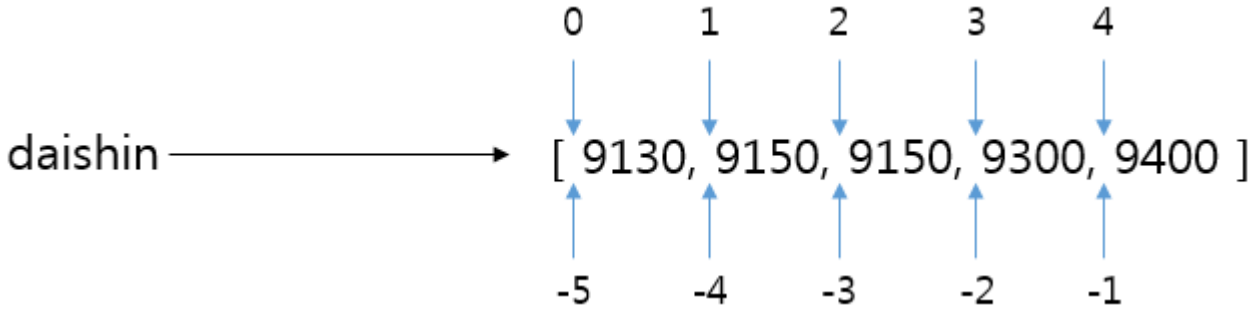

그림 3.3 파이썬 리스트 인덱싱

1-3) 리스트 슬라이싱

파이썬 리스트에 있는 데이터에 하나씩 접근할 때는 인덱싱을 사용하면 됩니다. 그런데 리스트에 있는 여러 개의 데이터에 동시에 접근하려면 어떻게 해야 할까요? 이럴 때 사용하는 것이 바로 파이썬 슬라이싱입니다. 파이썬 리스트의 슬라이싱은 문자열 슬라이싱과 동일합니다.

잠깐 주식 시장 이야기를 해보면, 2015 년 6 월 28 일 기준으로 코스피 시가총액 상위 10 종목은 표 3.1 과 같습니다. 코스피 시가총액 상위 10 종목은 순서가 있는 데이터이기 때문에 이를 파이썬으로 표현하려면 자료구조 중 리스트를 사용하는 것이 좋습니다. 다만 표 3.1 의 데이터는 순위와 종목명으로 구성돼 있지만 파이썬의 리스트에는 이미 각 인덱스 자체가 순서를 의미하므로 순위 값은 저장할 필요가 없이 1 순위부터 10 순위까지의 종목명을 리스트로 저장하면 됩니다.

표 3.1 코스피 시가총액 10 위 목록 (2015 년 6 월 28 일 기준)

순위	종목명
1	삼성전자
2	SK하이닉스
3	현대차
4	한국전력
5	아모레퍼시픽
6	제일모직
7	삼성전자우
8	삼성생명
9	NAVER
10	현대모비스

표 3.1 을 파이썬 리스트로 표현하면 다음과 같습니다.

```
>>> kospi_top10 = ['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽', '제일모직',
'삼성전자우', '삼성생명', 'NAVER', '현대모비스']
```

앞에서 배운 파이썬 인덱싱을 사용해 코스피 기준 시가총액 5 위를 출력하는 프로그램을 작성하면 다음과 같습니다. 여기서 한 가지 주의할 점은 파이썬의 인덱스는 0 부터 시작하기 때문에 kospi_top10[4]가 리스트 내에서 5 번째에 위치하는 데이터라는 것입니다.

```
>>> print("시가총액 5 위: ", kospi_top10[4])
시가총액 5 위:  아모레퍼시픽
>>>
```

현재 kospi_top10 이라는 리스트에는 10 개의 종목이 있는데, 만약 코스피 상위 5 종목만으로 구성된 새로운 리스트를 만들고 싶다면 어떻게 할까요? 다음과 같이 리스트의 슬라이싱을 사용하면 새로운 리스트를 손쉽게 만들 수 있습니다.

```
>>> kospi_top5 = kospi_top10[0:5]
>>> kospi_top5
['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽']
>>>
```

슬라이싱은 "[시작 인덱스: 끝 인덱스]"와 같은 형태로 표현하면 되는데, 위 코드에서는 0 이 시작 인덱스이고 5 가 끝 인덱스입니다. 따라서 kospi_top10[0:5]는 kospi_top10 이라는 리스트에서 0 번 위치에서부터 5 번 위치까지를 가져오라는 의미가 됩니다. 그림 3.4 와 같이 kospi_top10 이라는 이름의 리스트에 앞에서부터 인덱스를 붙여보면 [0:5]라는 슬라이싱의 의미를 더 쉽게 파악할 수 있습니다.

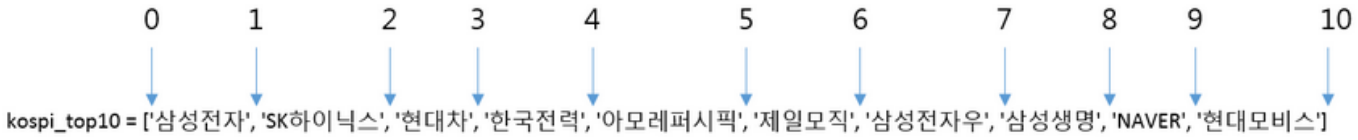

그림 3.4 리스트 인덱싱의 의미

지금부터는 배운 내용을 조금 응용해 보겠습니다. 이번에는 코스피 시가총액 기준으로 6 위부터 10 위에 대한 목록을 가져와 보겠습니다. 먼저 6 위를 가져올 때 파이썬 인덱스가 0 부터 시작하기 때문에 [5]라는 인덱싱을 사용해야 합니다. 그리고 "끝 인덱스" 값은 9 가 아니라 10 이어야 합니다. 그림 3.4 의 인덱스를 참고하면 9 를 사용하는 경우 'NAVER'까지만 가져오는 것을 확인할 수 있습니다.

```
>>> kospi_top10[5:10]
['제일모직', '삼성전자우', '삼성생명', 'NAVER', '현대모비스']
>>>
```

위 코드에서 한 가지 불편한 점은 슬라이싱할 때 리스트의 끝을 알려주기 위해서는 리스트 안에 있는 데이터의 개수를 미리 알고 있어야 한다는 점입니다. 위의 예제에서는 리스트 안에 총 10 개의 종목이 있었기 때문에 예제에서는 10 을 사용했습니다.

파이썬 리스트도 문자열과 마찬가지로 시작 인덱스나 끝 인덱스를 생략할 수 있습니다. 예를 들어, 다음 코드는 6 위부터 10 위에 대한 목록을 가져옵니다. 끝 인덱스에 해당하는 값에 아무 값도 넣지 않으면 자동으로 리스트의 끝을 알아서 인덱싱해줍니다.

```
>>> kospi_top10[5:]
['제일모직', '삼성전자우', '삼성생명', 'NAVER', '현대모비스']
>>>
```

이번에는 '시작 인덱스'에 해당하는 부분을 생략해 보겠습니다. 다음 코드를 참고하면 시작 인덱스에 해당하는 부분에 값을 생략하면 자동으로 리스트의 시작부터 값을 가져오는 것을 확인할 수 있습니다.

```
>>> kospi_top10[:5]
['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽']
>>>
```

앞에서 파이썬 리스트의 인덱싱에서 음수를 사용하는 경우를 살펴봤습니다. 마찬가지로 슬라이싱에서도 음수를 사용할 수 있습니다. 예를 들어, 6 위부터 9 위까지의 종목을 가져와야 하는 경우에는 다음과 같이 하면 됩니다. 다음 코드를 보면 [5:9]라고 슬라이싱하는 것과 [5:-1]이라고 슬라이싱하는 것이 동일한 결과를 보여주는 것을 알 수 있습니다.

```
>>> kospi_top10[5:9]
['제일모직', '삼성전자우', '삼성생명', 'NAVER']
>>> kospi_top10[5:-1]
['제일모직', '삼성전자우', '삼성생명', 'NAVER']
>>>
```

[5:-1] 슬라이싱의 의미가 잘 이해되지 않는 분들은 그림 3.5 를 참조하면 쉽게 이해할 수 있을 것입니다. 리스트 인덱싱에서 -1 이라는 값은 리스트에 있는 데이터 중 맨 마지막 데이터의 앞쪽을 가리키는 인덱스 값입니다. 따라서 kospi_top10[5:-1]은 ['제일모직', '삼성전자우', '삼성생명', 'NAVER']의 리스트를 반환합니다.

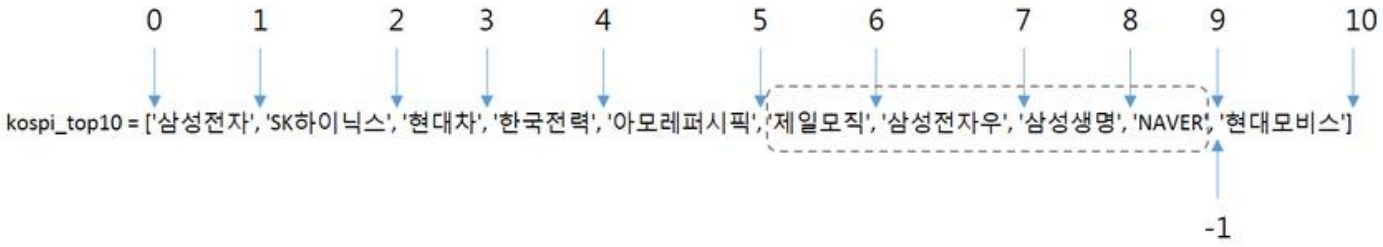

그림 3.5 리스트 인덱싱의 의미(2)

1-4) 리스트에 데이터 삽입하기

기존의 리스트에 새로운 데이터를 삽입하려면 append 메서드를 사용하면 됩니다. 여기서 메서드란 특정 목적을 위해 작성된 코드라고 이해하면 됩니다(자세한 내용은 6 장에서 다시 다루겠습니다). 다음 코드와 같이 코스피 상위 10 종목에 대한 리스트가 구성된 상태에서 해당 리스트에 코스피 11 번 순위인 'SK 텔레콤'을 추가해야 하는 경우를 생각해 봅시다. 이러한 상황에서 새로운 데이터를 리스트로 삽입할 때 사용하는 메서드가 바로 append 입니다.

```
>>> kospi_top10 = ['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽', '제일모직',
'삼성전자우', '삼성생명', 'NAVER', '현대모비스']
```

리스트에서 append 메서드를 사용하려면 다음 코드와 먼저 리스트 이름을 적은 후 마침표를 붙이고 그다음에 메서드 이름을 적으면 됩니다. 참고로 append 라는 영어 단어에 '~를 추가하다'라는 뜻이 있다는 점을 감안하면 append 메서드의 사용법을 기억하는 데 도움될 것입니다. 앞에서 append 의 뜻이 '~를 추가하다'라고 설명했는데, 이때 '~를'에 해당하는 부분을 괄호 사이에 넣어야 합니다. 다음 코드에서는 "SK 텔레콤"이라는 문자열 데이터를 리스트에 추가할 것이기 때문에 append("SK 텔레콤")이라고 한 것입니다.

```
>>> kospi_top10.append('SK 텔레콤')
>>>
```

append 메서드를 '호출'해 리스트에 데이터를 추가했으므로 리스트에 데이터가 잘 입력됐는지 확인해 보겠습니다. 앞서 여러 번 설명한 것처럼 리스트는 데이터를 순서대로 저장하고 있는 자료구조인데, append 메서드는 데이터를 리스트의 끝에 추가합니다. 따라서 다음과 같이 리스트의 이름을 파이썬 프롬프트에 입력해 보면 kospi_top10 리스트에 'SK 텔레콤'이라는 문자열이 추가되어 총 11 개의 문자열이 존재하는 것을 확인할 수 있습니다.

```
>>> kospi_top10
['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽', '제일모직', '삼성전자우', '삼성생
명', 'NAVER', '현대모비스', 'SK 텔레콤']
>>>
```

이제 kospi_top10 이라는 변수는 시가총액 상위 11 개 종목이 담긴 리스트를 바인딩하고 있으므로 kospi_top10 이라는 변수명은 적절하지 않습니다. 이럴 때는 간단히 리스트의 변수명을 변경할 수 있습니다.

```
>>> kospi_top11 = kospi_top10
>>> kospi_top11
['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽', '제일모직', '삼성전자우', '삼성생
명', 'NAVER', '현대모비스', 'SK 텔레콤']
>>>
```

파이썬 리스트에 데이터를 추가하는 두 번째 방법은 insert 라는 이름의 메서드를 사용하는 것입니다. 앞에서 배운 append 메서드를 사용하면 데이터를 리스트에 추가할 수 있는데, 왜 insert 라는 메서드도 있을까요? 왜냐하면 append 메서드는 데이터를 항상 리스트에 끝에 삽입하는 것과 달리 insert 메서드는 사용자가 원하는 위치에 데이터를 삽입할 수 있기 때문입니다.

예를 들어, 앞의 예제에서처럼 코스피 상위 10 개 종목이 있는데, 11 위였던 'SK 텔레콤'이 시가총액 기준으로 4 위로 갑자기 상승한 경우를 생각해봅시다. 이 경우 insert(3, 'SK 텔레콤')을 통해 4 위에 'SK 텔레콤'을 삽입할 수 있습니다. 참고로 insert 메서드에서 첫 번째 인자인 3 은 리스트에 데이터가 삽입될 위치를 의미하고, 두 번째 인자는 리스트에 삽입될 데이터를 의미합니다.

```
>>> kospi_top10 = ['삼성전자', 'SK 하이닉스', '현대차', '한국전력', '아모레퍼시픽', '제일모직',
'삼성전자우', '삼성생명', 'NAVER', '현대모비스']
>>> kospi_top10.insert(3, 'SK 텔레콤')
>>> kospi_top10
['삼성전자', 'SK 하이닉스', '현대차', 'SK 텔레콤', '한국전력', '아모레퍼시픽', '제일모직', '삼성전
자우', '삼성생명', 'NAVER', '현대모비스']
>>>
```

1-5) 리스트 데이터 삭제

3.1.4 절에서 insert 메서드를 이용해 'SK 텔레콤'을 kospi_top10 리스트의 4 번째에 삽입해 봤습니다. 그래서 kospi_top10 이라는 이름의 리스트에는 총 11 개의 종목이 들어 있게 됐습니다. 또한 'SK 텔레콤'이 4 순위로 입력됐기 때문에 '한국전력'부터는 자신의 순위가 하나씩 밀리게 됐고, 결국 '현대모비스'는 시가총액 상위 11 위가 됐습니다.

리스트의 이름은 kospi_top10 인데 현재 11 개의 종목이 리스트에 들어 있으므로 이번에는 시가총액 11 위인 '현대모비스'를 리스트로부터 제거하겠습니다. 이를 위해 먼저 리스트에 총 몇 개의 데이터가 있는지를 파이썬 코드를 통해 확인해 보겠습니다. 리스트에 입력된 데이터의 개수를 확인할 때는 len 이라는 내장 함수(built-in function)를 사용하면 됩니다.

```
>>> len(kospi_top10)
11
>>>
```

참고로 내장 함수라는 것은 파이썬에서 기본적으로 제공하는 함수라는 의미입니다. 일상생활에서 '빌트인'이라는 용어를 사용하곤 하는데, 그 '빌트인'이 여기서 사용하는 단어와 같은 의미입니다. len 이라는 내장 함수는 length 라는 영어 단어의 줄임말이며, 리스트뿐만 아니라 문자열의 길이를 확인할 때도 사용할 수 있습니다.

len 함수의 반환값을 확인해 보니 예상한 대로 kospi_top10 에는 현재 11 개의 데이터가 있습니다. 이제 마지막에 있는 데이터를 확인하고 이를 지워보겠습니다. 맨 마지막 데이터를 확인하기 위해서는 [-1]이라는 파이썬 인덱싱을 사용하면 됩니다. 그리고 리스트의 마지막 데이터를 지우기 위해서는 del 을 사용하면 됩니다. 이때 한 가지 주의할 점은 del 다음에 빈칸을 주고 그 다음에 지울 데이터를 입력해야 한다는 점입니다.

다음과 같이 리스트의 마지막 데이터인 '현대모비스'를 지운 후 kospi_top10 데이터를 다시 확인해보면 정상적으로 상위 10 종목만 남아 있는 것을 확인할 수 있습니다. 더불어 len 함수를 호출해보면 리스트에는 10 개의 데이터만 들어있는 것도 확인할 수 있습니다.

```
>>> kospi_top10[-1]
'현대모비스'
>>> del kospi_top10[-1]
>>> kospi_top10
```

```
['삼성전자', 'SK 하이닉스', '현대차', 'SK  텔레콤', '한국전력', '아모레퍼시픽', '제일모직', '삼성전
자우', '삼성생명', 'NAVER']
>>> len(kospi_top10)
10
>>>
```

2) 튜플

3.1 절에서는 파이썬의 리스트라는 자료구조를 배웠습니다. 이번 절에서 배울 튜플(tuple)도 여러 개의 데이터를 순서대로 담아두는 데 사용합니다. 단, 튜플과 리스트는 다음과 같은 두 가지 차이점이 있습니다.

1) 리스트는 '[' 와 ']'를 사용하는 반면 튜플은 '('와 ')'를 사용한다.

2) 리스트는 리스트 내의 원소를 변경할 수 있지만 튜플은 변경할 수 없다.

사실 파이썬의 튜플은 리스트에 있는 여러 기능이 빠져 있고, 반대로 리스트는 튜플이 지원하는 모든 기능을 포함하고 있으므로 튜플을 사용하지 않고 리스트라는 자료구조만 사용해도 프로그래밍하는 데 전혀 불편함이 없습니다. 다만 튜플은 리스트보다 속도가 빠르다는 장점이 있습니다. 따라서 한번 데이터를 저장해둔 후 추가하거나 삭제할 필요가 없는 경우라면 되도록 리스트보다는 튜플을 사용하는 것이 좋습니다. 먼저 튜플을 하나 만들어 봅시다.

```
>>> t = ('Samsung', 'LG', 'SK')
>>> t
('Samsung', 'LG', 'SK')
>>>
```

튜플은 리스트와 비슷하게 인덱싱을 통해 튜플의 원소에 접근할 수 있습니다. 먼저 len 내장 함수를 이용해 튜플의 원소의 개수를 확인해 보니 총 3 개가 있는 것을 확인했습니다. 다음과 같이 [0], [1], [2] 인덱싱을 통해 튜플의 각 원소에 접근하는 것을 확인할 수 있습니다. 그러나 튜플은 리스트와 달리 원소를 수정할 수 없으므로 원소의 내용을 바꾸려고 하면 에러가 발생합니다.

```
>>> len(t)
3
>>> t[0]
'Samsung'
>>> t[1]
'LG'
>>> t[2]
```

```
'SK'
>>> t[0] = "Naver"
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    t[0] = "Naver"
TypeError: 'tuple' object does not support item assignment
>>>
```

2-1) 튜플 슬라이싱

리스트에 저장된 데이터의 일부를 가져올 때 슬라이싱을 이용했습니다. 슬라이싱을 할 때는 [0:2]와 같은 형태로 데이터를 가져올 시작 인덱스와 끝 인덱스를 지정했습니다. 튜플도 리스트와 동일하게 슬라이싱이 가능합니다. 다음과 같이 t가 바인딩하는 튜플에서 'Samsung'과 'LG'만 가져오려면 다음과 같이 슬라이싱하면 됩니다.

```
>>> t
('Samsung', 'LG', 'SK')
>>> t[0:2]
('Samsung', 'LG')
>>>
```

여기서 한 가지 주의할 점은 튜플을 생성할 때는 '('와 ')' 기호를 사용하지만 데이터에 접근하는 인덱싱이나 슬라이싱에서는 데이터의 범위를 '['와 ']' 기호로 표현한다는 점입니다. 즉, 다음 코드는 파이썬 문법에 어긋나기 때문에 에러가 발생합니다.

```
>>> t(0:2)
SyntaxError: invalid syntax
>>>
```

3) 딕셔너리

파이썬에서 리스트, 튜플과 함께 자주 사용되는 자료구조로 딕셔너리(dictionary)가 있습니다. dictionary 는 여러분도 잘 알다시피 '사전'이라는 뜻을 가진 영어 단어입니다. 영어사전에는 영어 단어가 알파벳순으로 정렬돼 있어서 우리는 영어사전에서 영어 단어를 쉽게 찾을 수 있습니다. 파이썬의 딕셔너리도 영어사전과 유사하게 키(key)와 값(value)이라는 쌍으로 데이터를 구성해서 저장함으로써 더 쉽게 저장된 값을 찾을 수 있는 구조입니다. 말로는 딕셔너리가 무엇인지 잘 이해되지 않으니 직접 코드를 보면서 딕셔너리에 대해 배워보겠습니다.

여러 종목의 현재가를 저장하는 프로그램을 만든다고 해봅시다. 주식 종목별로 현재가를 저장해야 하는데, 리스트나 튜플로는 이를 쉽게 표현하기가 불가능합니다. 지금까지 배운 리스트나 튜플에는 값을 하나씩 저장했지 지금처럼 '종목'-'종목에 대한 현재가'와 같이 서로 연관된 두 개의 데이터를 저장하지는 못했습니다. 이러한 구조는 파이썬의 딕셔너리를 사용하면 아주 쉽게 표현할 수 있습니다.

딕셔너리는 '{'와 '}' 기호를 사용합니다. 리스트와 튜플을 복습해보면 리스트는 '['와 ']' 기호를 사용했고, 튜플은 '('와 ')' 기호를 사용했습니다. 다음과 같이 아무것도 들어 있지 않는 빈 딕셔너리를 하나 만들어 봅시다.

```
>>> cur_price = {}
>>>
```

위 코드를 실행하면 메모리의 어딘가에 딕셔너리가 하나 만들어지고, 이를 cur_price 라는 이름의 변수가 바인딩합니다. 정말로 딕셔너리가 잘 만들어졌는지 type 내장 함수를 이용해 확인해 봅시다. type 함수의 반환값을 보면 'dict'라는 문자열이 있는데, 이것은 dictionary 의 줄임말입니다.

```
>>> type(cur_price)
<class 'dict'>
>>>
``
```

원소가 들어 있지 않은 딕셔너리가 잘 만들어졌으니 딕셔너리에 키-값 쌍을 하나 추가해보겠습니다. 대신증권의 '현재가'가 30,000 원이라고 하면 다음과 작성하면 됩니다. 여기서 '**daeshin**'이 키고 30000 이 값이 됩니다.

```{.py}
>>> cur_price['daeshin'] = 30000
>>>
```

사전에 키와 값 쌍을 하나 추가했으니 해당 데이터가 잘 추가됐는지 딕셔너리를 확인해 봅시다. 파이썬 IDLE 의 프롬프트에 cur_price 라는 변수를 입력하고 엔터 키를 누르면 현재 딕셔너리에 저장된 값을 볼 수 있습니다.

```
>>> cur_price
{'daeshin': 30000}
>>>
```

딕셔너리에 값이 잘 추가됐으니 이번에는 'Daum KAKAO' 한 주의 현재가가 80,000 원이라는 것을 딕셔너리에 추가해 봅시다. 값을 추가한 후 cur_price 에 값이 잘 입력됐는지도 확인해보기 바랍니다.

```
>>> cur_price['Daum KAKAO'] = 80000
>>> cur_price
{'Daum KAKAO': 80000, 'daeshin': 30000}
>>>
```

딕셔너리에 들어 있는 데이터의 개수를 확인할 때도 리스트나 튜플과 동일하게 len 내장 함수를 사용하면 됩니다. 현재 cur_price 딕셔너리에는 두 개의 키-값 쌍이 입력돼 있으므로 len 내장 함수의 반환값이 2 가 됨을 확인할 수 있습니다.

```
>>> len(cur_price)
2
>>>
```

파이썬의 딕셔너리는 리스트와 튜플과 달리 인덱싱을 지원하지 않습니다. 딕셔너리는 리스트와 튜플과 달리 데이터를 순서대로 저장하는 것이 아니라 키와 값의 쌍이 서로 연결되도록만 저장하기 때문입니다. 따라서 다음과 같이 딕셔너리에 정숫값을 이용해 인덱싱하면 오류가 발생합니다.

```
>>> cur_price[0]
```

```
Traceback (most recent call last):
  File "<pyshell#18>", line 1, in <module>
    cur_price[0]
KeyError: 0
>>> cur_price[1]
Traceback (most recent call last):
  File "<pyshell#19>", line 1, in <module>
    cur_price[1]
KeyError: 1
>>>
```

딕셔너리는 리스트나 튜플과 달리 원소를 추가한 순서대로 데이터가 저장돼 있지 않기 때문에 딕셔너리에 추가한 데이터를 얻으려면 키 값을 사용해야 합니다. 앞서 'daeshin'이 키이고 30000 이 값이라고 했습니다. 따라서 cur_price 딕셔너리로부터 'daeshin'의 '현재가'를 구하려면 'daeshin'이라는 키 값을 사용하면 됩니다. 마찬가지로 'Daum KAKAO'라는 문자열이 키이므로 'Daum KAKAO'의 현재가도 'Daum KAKAO'라는 키를 통해 구할 수 있습니다.

```
>>> cur_price['daeshin']
30000
>>> cur_price['Daum KAKAO']
80000
>>>
```

3-1) 딕셔너리에 데이터 삽입 및 삭제

딕셔너리에 데이터를 삽입하는 것은 앞에서 해봤습니다. 딕셔너리에 데이터를 삽입하는 방법을 정리해보면 딕셔너리는 리스트의 append 나 insert 같은 메서드를 지원하지 않으며, 명시적으로 키-값 쌍을 넣어야 했습니다. 예제를 통해 한 번 더 복습해 보겠습니다. cur_price 라는 딕셔너리에 'naver'의 현재가가 80,000 원이라는 데이터를 추가해 보겠습니다.

```
>>> cur_price['naver'] = 800000
>>> cur_price
{'Daum KAKAO': 80000, 'naver': 800000, 'daeshin': 30000}
>>>
```

참고로 파이썬에서 딕셔너리를 생성할 때 다음과 같이 초깃값을 넣어서 딕셔너리를 생성할 수도 있습니다.

```
>>> cur_price = {'Daum KAKAO': 80000, 'naver':800000, 'daeshin':30000}
>>> cur_price
{'naver': 800000, 'Daum KAKAO': 80000, 'daeshin': 30000}
>>>
```

딕셔너리에 저장된 데이터를 삭제하려면 리스트와 마찬가지로 del 을 사용하면 됩니다. 참고로 튜플은 리스트와 딕셔너리와 달리 데이터를 삭제할 수 없었습니다. 주식 종목의 현재가를 저장하고 있는 cur_price 에서 'daeshin'이라는 종목의 현재가를 지우고 싶다면 다음과 같이 키 값을 이용하면 됩니다. 삭제한 후에는 cur_price 에서 키-값 쌍이 잘 지워졌는지 확인해보기 바랍니다.

```
>>> del cur_price['daeshin']
>>> cur_price
{'naver': 800000, 'Daum KAKAO': 80000}
>>>
```

3-2) 딕셔너리로부터 키-값 구하기

딕셔너리에는 키-값 쌍이 저장된다고 했습니다. 그렇다면 딕셔너리에 있는 키 값만 구하려면 어떻게 해야 할까요? 이를 위해 파이썬의 딕셔너리는 keys()라는 메서드를 제공합니다. 따라서 키 값만 구하고 싶을 때 "딕셔너리이름.keys()"라고 호출하면 됩니다. 그런데 언제 딕셔너리로부터 키 값만 구해야 할까요?

다음 코드의 cur_price 딕셔너리에는 세 종목에 대한 현재가가 저장돼 있습니다. cur_price 딕셔너리를 통해 어떤 주식 종목에 대한 현재가를 조회하고 싶다면 먼저 해당 종목명이 cur_price 라는 딕셔너리에 있는지 확인해야 합니다. 이를 위해서는 cur_price 라는 딕셔너리에서 키 값을 구해와야겠죠?

```
>>> cur_price = {'Daum KAKAO': 80000, 'naver':800000, 'daeshin':30000}
>>> cur_price.keys()
dict_keys(['naver', 'Daum KAKAO', 'daeshin'])
>>>
```

위 코드를 살펴보면 키 값을 구하기 위해 딕셔너리의 이름인 cur_price 다음에 마침표를 하나 붙이고 keys()라고 적는 것을 확인할 수 있습니다. 이러한 방식은 리스트에서 insert 나 append 같은 메서드를 호출하는 방식과 동일합니다.

keys() 메서드의 반환값을 보면 'Daum KAKAO'와 'naver'가 파이썬의 리스트로 표현된 것 같기도 한데 그 앞에 'dict_keys'라는 것이 있습니다. 사실 keys() 메서드의 반환값은 리스트는 아니며, 리스트로 만들려면 list 라는 키워드를 이용해 타입을 변환해줘야 합니다.

```
>>> list(cur_price.keys())
['naver', 'Daum KAKAO', 'daeshin']
>>>
```

그러나 이처럼 단순히 list 를 통해 반환값을 받으면 사용하기 어려울 것입니다. 항상 변수를 통해서 반환값을 바인딩하는 것이 좋습니다. 다음과 같이 stock_list 라는 이름의 변수로 list 의 반환값을 바인딩하도록 해줍니다. stock_list 가 바인딩하고 있는 값을 보면 파이썬의 리스트임을 확인할 수 있습니다.

```
>>> stock_list = list(cur_price.keys())
>>> stock_list
['naver', 'Daum KAKAO', 'daeshin']
```

```
>>>
```

딕셔너리에서 키 목록을 구하는 것과 비슷하게 값 목록을 구하려면 values()라는 메서드를 사용하면 됩니다. cur_price 딕셔너리에서 값 목록을 리스트로 만들어 바인딩해 봅시다.

```
>>> price_list = list(cur_price.values())
>>> price_list
[800000, 80000, 30000]
>>>
```

'Samsung'이라는 종목의 현재가를 cur_price 라는 딕셔너리로부터 구해야 한다고 해봅시다. 이를 위해서는 앞서 이야기한 것처럼 먼저 'Samsung'이라는 키 값이 cur_price 딕셔너리에 있는지 확인해야 합니다. 딕셔너리의 키 값은 keys() 메서드를 통해 얻을 수 있었죠? 다음과 같이 in 이라는 키워드와 keys() 메서드의 반환값을 이용하면 이를 쉽게 조회할 수 있습니다.

```
>>> 'Samsung' in cur_price.keys()
False
>>>
```

위 코드에 대해 한글로 해석해보면 'Samsung'이 keys() 안에 있는가? 정도가 됩니다. 참고로 in 이라는 영어 단어는 "안에", "내에"라는 뜻을 가지고 있습니다. 일단 반환값을 보면 False 라는 영어 단어를 볼 수 있는데 이 단어는 "거짓의", "틀린"이라는 의미가 있습니다. 즉, "Samsung"이 cur_price 딕셔너리의 키에 들어있지 않다는 것을 알 수 있습니다.

그렇다면 'naver'는 어떨까요? 다음 코드를 보면 True 가 반환된 것을 확인할 수 있습니다. True 는 '사실', '정말', '진짜의'라는 뜻이 있습니다. 즉, 'naver'라는 종목명은 cur_price 의 키 중 하나임을 알 수 있습니다.

```
>>> 'naver' in cur_price.keys()
True
>>>
```

'naver'는 cur_price 딕셔너리의 키 중 하나이기 때문에 키를 통해 현재가 데이터를 구할 수 있습니다. 그러나 'Samsung'은 키에 포함돼 있지 않기 때문에 'Samsung'이라는 키를 사용하면 오류가 발생합니다.

```
>>> cur_price['naver']
```

```
800000
>>> cur_price['Samsung']
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    cur_price['Samsung']
KeyError: 'Samsung'
>>>
```

1) 연습문제

문제 3-1

2015 년 9 월 초의 네이버 종가는 표 3.2 와 같습니다. 09/07 의 종가를 리스트의 첫 번째 항목으로 입력해서 naver_end_price 라는 이름의 리스트를 만들어보세요.

표 3.2 네이버 종가

날짜	요일	종가
09/11	금	488,500
09/10	목	500,500
09/09	수	501,000
09/08	화	461,500
09/07	월	474,500

문제 3-2

문제 3-1 에서 만든 naver_end_price 를 이용해 해당 주에 종가를 기준으로 가장 높았던 가격을 출력하세요. (힌트: 리스트에서 최댓값을 찾는 함수는 max()이고, 화면에 출력하는 함수는 print()입니다.)

문제 3-3

문제 3-1 에서 만든 naver_end_price 를 이용해 해당 주에 종가를 기준으로 가장 낮았던 가격을 출력하세요. (힌트: 리스트에서 최솟값을 찾는 함수는 min()이고, 화면에 출력하는 함수는 print()입니다.)

문제 3-4

문제 3-1 에서 만든 naver_end_price 를 이용해 해당 주에서 가장 종가가 높았던 요일과 가장 종가가 낮았던 요일의 가격 차를 화면에 출력하세요.

문제 3-5

문제 3-1 에서 만든 naver_end_price 를 이용해 수요일의 종가를 화면에 출력하세요.

문제 3-6

문제 3-1 의 표 3.2 를 이용해 날짜를 딕셔너리의 키 값으로, 종가를 딕셔너리의 값으로 사용해 naver_end_price2 라는 딕셔너리를 만드세요.

문제 3-7

문제 3-6 에서 만든 naver_end_price2 딕셔너리를 이용해 09/09 일의 종가를 출력하세요.

2) 연습문제 풀이

문제 3-1

```
>>> naver_end_price = [474500, 461500, 501000, 500500, 488500]
>>> naver_end_price
[474500, 461500, 501000, 500500, 488500]
>>>
```

문제 3-2

```
>>> max_price = max(naver_end_price)
>>> print(max_price)
501000
>>>
```

문제 3-3

```
>>> min_price = min(naver_end_price)
>>> print(min_price)
461500
>>>
```

문제 3-4

```
>>> naver_end_price = [474500, 461500, 501000, 500500, 488500]
>>> max_price = max(naver_end_price)
>>> min_price = min(naver_end_price)
>>> print("가격차: ", max_price - min_price)
```

```
가격차: 39500
>>>
```

문제 3-5

```
>>> print("수요일 종가: ", naver_end_price[2])
수요일 종가:  501000
>>>
```

문제 3-6

```
>>> naver_end_price2 = {'09/07':474500, '09/08':461500, '09/09':501000, '09/10': 500500,
'09/11':488500}
>>> naver_end_price2
{'09/09': 501000, '09/11': 488500, '09/08': 461500, '09/10': 500500, '09/07': 474500}
>>>
```

문제 3-7

```
>>> print(naver_end_price2['09/09'])
501000
>>>
```