Python (4)

자료형 - 리스트, 튜플

02-3 리스트 자료형

앞에서 우리는 숫자와 문자열에 대해서 알아보았다.
하지만 숫자와 문자열만으로 프로그래밍을 하기에는 부족한 점이 많다.

예를 들어 1부터 10까지의 숫자 중 홀수의 모음인 1, 3, 5, 7, 9의 집합을 생각해 보자.
이런 숫자의 모음을 숫자나 문자열로 표현하기는 어렵다.

파이썬에는 이러한 불편함을 해소할 수 있는 자료형이 존재한다. 이것이 바로 리스트(list)이다.

○ 리스트는 어떻게 만들고 사용할까?

리스트를 사용하면 1, 3, 5, 7, 9의 숫자 모음을 다음과 같이 간단하게 표현할 수 있다.

```
odd = [1, 3, 5, 7, 9]
```

리스트를 만들 때는 위에서 보는 것과 같이 대괄호([])로 감싸 주고 각 요솟값은 쉼표(,)로 구분해 준다.

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

여러 가지 리스트의 형태는 다음과 같다.

```
a = []
b = [1, 2, 3]
c = ['Life', 'is', 'too', 'short']
d = [1, 2, 'Life', 'is']
e = [1, 2, ['Life', 'is']]
```

비어 있는 리스트는 **a = list()**로도 생성할 수 있다.

위 예제에서,
a는 아무것도 포함하지 않아 비어 있는 리스트(**[]**)이다.
b는 숫자를 요소값으로 가지고 있다.
c는 문자열을 요소값으로 가지고 있다.
d는 숫자와 문자열을 함께 요소값으로 가지고 있다.
e는 다른 리스트를 요소값으로 가지고 있다.

즉, 리스트 안에는 파이썬의 어떠한 자료형도 포함할 수 있다.

○ 리스트의 인덱싱과 슬라이싱

리스트도 문자열처럼 인덱싱과 슬라이싱이 가능하다.

◆ 리스트의 인덱싱

리스트 역시 문자열처럼 인덱싱을 적용할 수 있다. 먼저 a 변수에 [1, 2, 3] 값을 설정한다.

아래와 같은 a 리스트가 있다고 하자.

```
a = [1, 2, 3]
```

리스트를 출력해보면 아래와 같다.

```
print(a)
--------------------
[1, 2, 3]
```

다음과 같이 a[0]은 리스트 a의 첫 번째 요소값을 말한다.

```
print(a[0])
--------------------
1
```

다음 예는 리스트의 첫 번째 요소인 a[0]과 세 번째 요소인 a[2]의 값을 더한 것이다.

```
print(a[0] + a[2])
--------------------
4
```

이것은 1 + 3으로 해석되어 값 4를 출력한다.

문자열에서 살펴봤듯이 파이썬은 숫자를 0부터 세기 때문에 a[0]이 리스트 a의 첫 번째 요소이다.
a[-1]은 문자열에서와 마찬가지로 리스트 a의 마지막 요소값을 말한다.

```
print(a[-1])
--------------------
3
```

이번에는 다음 예처럼 리스트 a를 숫자 1, 2, 3과 또 다른 리스트인 ['a', 'b', 'c']를 포함하도록 만들어 보자.

```
a = [1, 2, 3, ['a', 'b', 'c']]
```

그리고 다음과 같이 출력해 보자.

```
print(a[0])
print(a[-1])
print(a[3])
--------------------
1
['a', 'b', 'c']
['a', 'b', 'c']
```

예상한 대로 a[-1]은 마지막 요소값 ['a', 'b', 'c']를 나타낸다.
a[3]은 리스트 a의 네 번째 요소를 나타내기 때문에 마지막 요소를 나타내는 a[-1]과 동일한 결과값을 보여 준다.

그리고, 아래와 같이 리스트 a에 포함된 ['a', 'b', 'c'] 리스트에서 'a' 값을 인덱싱을 사용해 끄집어 낼 수 있다.

```
print(a[-1][0])
--------------------
a
```

위와 같이 하면 'a'를 끄집어 낼 수 있다.

다음 예도 마찬가지 경우이므로 어렵지 않게 이해할 수 있을 것이다.

```
print(a[-1][1])
print(a[-1][2])
--------------------
b
c
```

◆ 삼중 리스트에서 인덱싱하기

조금 복잡하지만, 리스트를 다음과 같이 작성할 수도 있다.

```
a = [1, 2, ['a', 'b', ['Life', 'is']]]
```

리스트 a는 삼중 구조의 리스트이다.

이 경우, 'Life' 문자열만 끄집어 내려면 다음과 같이 해야 한다.

```
print(a[2][2][0])
--------------------
Life
```

위 예는 리스트 a의 세 번째 요소인 리스트 ['a', 'b', ['Life', 'is']]에서 세 번째 요소인 리스트 ['Life', 'is']의 첫 번째 요소를 나타낸다.

◆ 리스트의 슬라이싱

문자열과 마찬가지로 리스트에서도 슬라이싱 기법을 적용할 수 있다.
슬라이싱은 '잘라 낸다'라는 뜻이라고 했다. 리스트의 슬라이싱에 대해서 살펴보자.

```
a = [1, 2, 3, 4, 5]
print(a[0:2])
--------------------
[1, 2]
```

앞의 예를 문자열에서 슬라이싱했던 예와 비교해 보자.

```
a = "12345"
```

```
print(a[0:2])
--------------------
12
```

리스트와 문자열이 거의 동일하게 사용되었다는 것을 알 수 있다. 문자열에서 했던 것과 사용법이 완전히 동일하다.

몇 가지 예를 더 살펴보면,

```
a = [1, 2, 3, 4, 5]
b = a[:2]
c = a[2:]

print(b)
print(c)
--------------------
[1, 2]
[3, 4, 5]
```

b 변수는 리스트 a의 첫 번째 요소부터 두 번째 요소인 a[1]까지 나타내는 리스트이다.
물론 a[2] 값인 3은 포함되지 않는다. c라는 변수는 리스트 a의 세 번째 요소부터 끝까지 나타내는 리스트이다.

◆ 중첩된 리스트에서 슬라이싱하기

리스트가 포함된 중첩 리스트 역시 슬라이싱 방법은 똑같이 적용된다.

```
a = [1, 2, 3, ['a', 'b', 'c'], 4, 5]

print(a[2:5])
print(a[3][:2])
--------------------
[3, ['a', 'b', 'c'], 4]
['a', 'b']
```

위 예에서 a[3]은 ['a', 'b', 'c']를 나타낸다.
따라서 a[3][:2]는 ['a', 'b', 'c']의 첫 번째 요소부터 세 번째 요소 직전까지의 값, 즉 ['a', 'b']를 나타내는 리스트가 된다.

◆ 리스트 슬라이싱에서 step 사용하기

리스트 슬라이싱에는 세 번째 값으로 step을 사용할 수 있습니다.

```
리스트[시작_인덱스:끝_인덱스:step]
```

```
a = ['a', 'b', 'c', 'd', 'e']

# 2칸씩 이동하면서 가져옵니다.
print(a[::2])
----------------------------
['a', 'c', 'e']
```

```
a = ['a', 'b', 'c', 'd', 'e']

# 3칸씩 이동하면서 가져옵니다.
print(a[::3])
----------------------------
['a', 'd']
```

근데, 여기서 한가지 중요한 사항이 있습니다.
step을 음수로 지정할 수도 있습니다.

슬라이스를 사용할 때 인덱스 증가폭을 음수로 지정하면 요소를 뒤에서부터 가져올 수 있습니다.
다음 예는 리스트 **a**에서 인덱스 **5**부터 **2**까지 **1**씩 감소시키면서 요소를 가져옵니다.

```
a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
print(a[5:1:-1])
--------------------
[50, 40, 30, 20]
```

여기서 주의할 점은 인덱스가 감소하므로 끝 인덱스보다 시작 인덱스를 더 크게 지정해야 한다는 점입니다.
즉, **a[5:1:-1]**과 같이 시작 인덱스부터 끝 인덱스까지 감소하도록 지정합니다.
그리고 끝 인덱스는 가져오려는 범위에 포함되지 않습니다.(즉, 끝 인덱스에 **+1**을 해주는 것과 같습니다.)

특히 다음과 같이 시작 인덱스와 끝 인덱스를 생략하면서 인덱스 증가폭을 **-1**로 지정하면 어떻게 될까요?
이때는 리스트 전체에서 인덱스를 **1**씩 감소시키면서 요소를 가져오므로 리스트를 반대로 뒤집습니다.

```
a = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90]
print(a[::-1])
---------------------------
[90, 80, 70, 60, 50, 40, 30, 20, 10, 0]
```

물론 이 방법은 리스트뿐만 아니라 모든 시퀀스 객체에 사용할 수 있습니다.

○ 리스트 연산하기

리스트 역시 +를 사용해서 더할 수 있고 *를 사용해서 반복할 수 있다.

◆ 리스트 더하기(+)

```
a = [1, 2, 3]
b = [4, 5, 6]
print(a + b)
--------------------
[1, 2, 3, 4, 5, 6]
```

리스트 사이에서 +는 2개의 리스트를 합치는 기능을 한다. 문자열에서 "abc" + "def" = "abcdef"가 되는 것과 같은 이치이다.

◆ 리스트 반복하기(*)

```
a = [1, 2, 3]
print(a * 3)
```

```
--------------------
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

위에서 볼 수 있듯이 **[1, 2, 3]** 리스트가 세 번 반복되어 새로운 리스트를 만들어 낸다.
문자열에서 **"abc" * 3 = "abcabcabc"** 가 되는 것과 같은 이치이다.

◆ 리스트 길이 구하기

리스트 길이를 구하기 위해서는 다음처럼 **len** 함수를 사용해야 한다.

```
a = [1, 2, 3]
print(len(a))
--------------------
3
```

len은 문자열, 리스트 외에 앞으로 배울 튜플과 딕셔너리에도 사용할 수 있는 함수이다. 실습에서 자주 사용하므로 잘 기억해 두자.

◆ 초보자가 범하기 쉬운 리스트 연산 오류

다음 소스 코드를 입력했을 때 결괏값은 어떻게 나올까?

```
a = [1, 2, 3]
print(a[2] + "hi")
```

a[2]의 값인 3과 문자열 hi가 더해져서 3hi가 출력될 것이라고 생각할 수 있다.
하지만 다음 결과를 살펴보면 오류가 발생한다.

이유는, a[2]에 저장된 값은 3이라는 정수인데 "hi"는 문자열이다.
정수와 문자열은 당연히 서로 더할 수 없기 때문에 오류가 발생한 것이다.

만약 숫자와 문자열을 더해서 '3hi'를 만들고 싶다면 다음처럼 숫자 3을 문자 '3'으로 바꾸어야 한다.

```
print(str(a[2]) + "hi")
--------------------
3hi
```

str은 정수나 실수를 문자열로 바꾸어 주는 파이썬의 내장 함수이다

○ 리스트의 수정과 삭제 도현

리스트는 각 요소값을 수정하거나 삭제할 수 있다.

◆ 리스트의 값 수정하기

```
a = [1, 2, 3]
a[2] = 4
print(a)
--------------------
[1, 2, 4]
```

결과를 보면, a[2]의 요솟값 3이 4로 바뀌었다.

◆ del 함수를 사용해 리스트 요소 삭제하기

```
a = [1, 2, 3]
del a[1]
print(a)
```

```
--------------------
[1, 3]
```

del a[x]는 **x**번째 인덱스의 요솟값을 삭제한다. 위에서는 **a** 리스트에서 **a[1]**을 삭제했다.
del 함수는 파이썬이 자체적으로 가지고 있는 삭제 함수이며 다음과 같이 사용한다.

```
del 객체
```

여기서, 객체란 파이썬에서 사용되는 모든 자료형을 말한다.

다음처럼 슬라이싱 기법을 사용하여 리스트의 요소 여러 개를 한꺼번에 삭제할 수도 있다.

```
a = [1, 2, 3, 4, 5]
del a[2:]
print(a)
--------------------
[1, 2]
```

a[2:]에 해당하는 리스트의 요소들이 삭제되었다.

○ 리스트 관련 함수

문자열과 마찬가지로 리스트 변수 이름 뒤에 '.'를 붙여 여러 가지 리스트 관련 함수를 사용할 수 있다.
유용하게 사용하는 리스트 관련 함수 몇 가지만 알아보자.

◆ 리스트에 요소 추가하기 - append

append의 사전적 의미는 '덧붙이다, 첨부하다'이다.
이 뜻을 안다면 다음 예가 바로 이해될 것이다. append(x)는 리스트의 맨 마지막에 x를 추가하는 함수이다.

```
a = [1, 2, 3]
a.append(4)
print(a)
--------------------
[1, 2, 3, 4]
```

리스트 안에는 어떤 자료형도 추가할 수 있다. 다음 예는 리스트에 다시 리스트를 추가한 결과이다.

```
a = [1, 2, 3]
a.append([5, 6])
```

```
print(a)
--------------------
[1, 2, 3, [5, 6]]
```

◆ 리스트 정렬 - sort

sort 함수는 리스트의 요소를 순서대로 정렬해 준다.

```
a = [1, 4, 3, 2]
a.sort()
print(a)
--------------------
[1, 2, 3, 4]
```

문자 역시 알파벳 순서로 정렬할 수 있다.

```
a = ['a', 'c', 'b']
a.sort()
print(a)
--------------------
```

```
['a', 'b', 'c']
```

◆ 리스트 순서 뒤집기 - reverse

reverse 함수는 리스트를 역순으로 뒤집어 준다.
이때 리스트 요소들을 순서대로 정렬한 다음 다시 역순으로 정렬하는 것이 아니라, 현재의 리스트를 그대로 거꾸로 뒤집는다.

```
a = ['a', 'c', 'b']
a.reverse()
print(a)
--------------------
['b', 'c', 'a']
```

◆ 인덱스 반환 - index

index(x) 함수는 리스트에 **x** 값이 있으면 **x**의 인덱스 값(위칫값)을 리턴한다.

```
a = [1, 2, 3]
```

```
print(a.index(3))
print(a.index(1))
--------------------
2
0
```

위 예에서 리스트 a에 있는 숫자 3의 위치는 a[2]이므로 2, 숫자 1의 위치는 a[0]이므로 0을 리턴한다.

만약 위의 a 리스트에 존재하지 않는 값을 찾으려 시도하면, 오류가 발생한다.

```
print(a.index(0))
```

◆ 리스트에 요소 삽입 – insert

insert(a, b)는 리스트의 a번째 위치에 b를 삽입하는 함수이다. 파이썬은 숫자를 0부터 센다는 것을 반드시 기억하자.

리스트 a가 아래와 같다고 하자

```
a = [1, 2, 3]
```

아래 예는 0번째 자리, 즉 첫 번째 요소인 a[0] 위치에 값 4를 삽입하라는 뜻이다.

```
a.insert(0, 4)
print(a)
--------------------
[4, 1, 2, 3]
```

아래 예는 리스트 a의 a[3], 즉 네 번째 요소 위치에 값 5를 삽입하라는 뜻이다.

```
a.insert(3, 5)
print(a)
--------------------
[4, 1, 2, 5, 3]
```

여기서 몇가지 주의할 것을 알아보자

- 리스트의 삽입은 반드시 하나씩 만 가능하다.
- 만약 삽입하려는 위치가 리스트의 길이 보다 큰 경우에는 항상 맨 마지막 위치에 삽입된다.

◆ 리스트 요소 제거 - remove

remove(x)는 리스트에서 첫 번째로 나오는 **x**의 값을 삭제하는 함수이다.
여기서 **x**는 인덱스가 아니고 삭제하고자 하는 리스트의 값이라는 것을을 주의해야 한다.

a가 아래와 같은 리스트라고 하면,

```
a = ['a', 'b', 'c', 'b', 'e']
```

```
a.remove('b')
print(a)
--------------------
['a', 'c', 'b', 'e']
```

a가 **'b'**라는 값을 **2**개 가지고 있을 경우, 첫 번째 **'b'**만 제거되는 것을 알 수 있다.

```
a.remove('a')
print(a)
--------------------
['c', 'b', 'e']
```

remove('a')을 한 번 더 실행하면 다시 'a'가 삭제된다.

◆ 리스트 요소 끄집어 내기 – pop

pop()은 리스트의 맨 마지막 요소를 반환하고, 그 요소는 삭제한다.

```
a = [1, 2, 3]

b = a.pop()
print(b)
print(a)
--------------------
3
[1, 2]
```

a 리스트를 다시 보면 [1, 2, 3]에서 3을 끄집어 내고 최종적으로 [1, 2]만 남는 것을 확인할 수 있다.

pop(x)는 리스트의 인덱스가 x인 요소를 반환하고, 그 요소는 삭제한다.

```
a = [1, 2, 3]
```

```
b = a.pop(1)
print(b)
print(a)
--------------------
2
[1, 3]
```

◆ 리스트에 포함된 요소 x의 개수 세기 – count

count(x)는 리스트 안에 값 **x**가 몇 개 있는지 조사하여 그 개수를 반환하는 함수이다.

```
a = ['a', 'b', 'c', 'b', 'e']
print(a.count('b'))
--------------------
2
```

'b'이라는 값이 리스트 **a**에 **2**개 들어 있으므로 **2**를 리턴한다.

◆ 리스트 확장 – extend

extend(x)에서 x에는 리스트만 올 수 있으며 원래의 a 리스트에 x 리스트를 더하게 된다.

```
a = [1, 2, 3]
a.extend([4, 5])
print(a)

b = [6, 7]
a.extend(b)
print(a)
--------------------
[1, 2, 3, 4, 5]
[1, 2, 3, 4, 5, 6, 7]
```

a.extend([4, 5])는 a += [4, 5]와 동일하다.

a += [4, 5]는 a = a + [4, 5]와 동일한 표현식이다.

02-4 튜플 자료형

튜플(tuple)은 몇 가지 점을 제외하곤 리스트와 거의 비슷하다.
리스트와 다른 점은 다음과 같다.

리스트는 [], 튜플은 ()으로 둘러싼다.
리스트는 요소값의 생성, 삭제, 수정이 가능하지만, 튜플은 요소값을 변경할 수 없다.

○ 튜플은 어떻게 만들까?

튜플은 다음과 같이 여러 가지 형태로 생성할 수 있다.

```
t1 = ()
t2 = (1,)
t3 = (1, 2, 3)
t4 = 1, 2, 3
t5 = ('a', 'b', ('ab', 'cd'))
```

모양은 리스트와 거의 비슷하지만, 튜플에서는 리스트와 다른 2가지 차이점을 찾아볼 수 있다.
t2 = (1,)처럼 단지 1개의 요소만을 가질 때는 요소 뒤에 쉼표(,)를 반드시 붙여야 한다는 것과,
t4 = 1, 2, 3처럼 소괄호(())를 생략해도 된다는 점이다.

얼핏 보면 튜플과 리스트는 비슷한 역할을 하지만, 프로그래밍을 할 때 튜플과 리스트는 구별해서 사용하는 것이 유리하다.

튜플과 리스트의 가장 큰 차이는 요소값을 변화시킬 수 있는지의 여부이다.
즉, 리스트의 요소값은 변경이 가능하고 튜플의 요소값은 변경이 불가능하다.

따라서 프로그램이 실행되는 동안 요소값이 항상 변하지 않아야 한다면 튜플을 사용해야 한다.
실제로 튜플의 요소값을 변경하거나 삭제하려고 하면, 오류가 발생한다.
이와 반대로 수시로 요소값을 변화시켜야할 경우라면 리스트를 사용해야 한다.

실제 프로그램에서는 값이 변경되는 형태의 변수가 훨씬 많기 때문에 평균적으로 튜플보다 리스트를 더 많이 사용한다.

○ 튜플 다르기

튜플은 요솟값을 변화시킬 수 없다는 점만 제외하면 리스트와 완전히 동일하므로 간단히 살펴보도록 한다.
다음에 나오는 예제는 서로 연관되어 있으므로 차례대로 실행해 보기 바란다.

◆ 인덱싱하기

```
t1 = (1, 2, 'a', 'b')
print(t1[0])
print(t1[3])
--------------------
1
b
```

문자열, 리스트와 마찬가지로 **t1[0], t1[3]**처럼 인덱싱이 가능하다.

◆ 슬라이싱하기

```
t1 = (1, 2, 'a', 'b')
print(t1[1:])
--------------------
(2, 'a', 'b')
```

t1[1]부터 튜플의 마지막 요소까지 슬라이싱하는 예이다.

◆ 튜플 더하기

```
t1 = (1, 2, 'a', 'b')
t2 = (3, 4)
t3 = t1 + t2
print(t3)
--------------------
(1, 2, 'a', 'b', 3, 4)
```

튜플을 더하는 방법을 보여 주는 예이다. 이때도 t1, t2 튜플의 요소값이 바뀌는 것은 아니다.
다만, t1, t2 튜플을 더하여 새로운 튜플 t3를 생성한 것이다.

◆ 튜플 곱하기

```
t2 = (3, 4)
t3 = t2 * 3
print(t3)
--------------------
(3, 4, 3, 4, 3, 4)
```

튜플의 곱하기(반복) 예를 보여 준다.

◆ 튜플 길이 구하기

```
t1 = (1, 2, 'a', 'b')
print(len(t1))
--------------------
4
```

튜플의 길이를 구하는 예이다.

튜플은 요소값을 변경할수 없기 때문에 sort, insert, remove, pop과 같은 내장 함수가 없다.