Ch10. 문자열 응용

- **문자열 조작하기**
  - 문자열은 문자열을 조작하거나 정보를 얻는 다양한 메서드(method)를 제공
  - 파이썬에서 제공하는 문자열 메서드는 여러 가지가 있지만
    여기서는 자주 쓰는 메서드를 다루자

- **문자열 바꾸기**
  - **replace('바꿀문자열', '새문자열') : 문자열 안의 문자열을 다른 문자열로 바꿈**

    (문자열 자체는 변경하지 않으며 바뀐 결과를 반환함)

```
>>> 'Hello, world!'.replace('world', 'Python')
'Hello, Python!'
```

  - 만약 바뀐 결과를 유지하고 싶다면 문자열이 저장된 변수에 replace를 사용한 뒤 다시 변수에 할당해주면 됨

```
>>> s = 'Hello, world!'
>>> s = s.replace('world!', 'Python')
>>> s
'Hello, Python'
```

- **문자 바꾸기**
  - **translate는 문자열 안의 문자를 다른 문자로 바꿈**
  - **먼저 str.maketrans('바꿀문자', '새문자')로 변환 테이블을 만듬**
  - **이후 translate(테이블)을 사용하면 문자를 바꾼 뒤 결과를 반환함**
  - 다음은 문자열 'apple'에서 a를 1, e를 2, i를 3, o를 4, u를 5로 바꿈

```
>>> table = str.maketrans('aeiou', '12345')
>>> 'apple'.translate(table)
'1ppl2'
```

- **문자열 분리하기**
  - **split() : 공백을 기준으로 문자열을 분리하여 리스트로 생성**
  - 지금까지 input으로 문자열을 입력 받은 뒤 리스트로 만든 메서드 : split

```
>>> 'apple pear grape pineapple orange'.split()
['apple', 'pear', 'grape', 'pineapple', 'orange']
```

  - split('기준문자열')과 같이 기준 문자열을 지정하면 기준 문자열로 문자열을 분리함
  - 즉, 문자열에서 각 단어가 ,(콤마)와 공백으로 구분되어 있을 때 ', '으로 문자열을 분리하면 단어만 리스트로 생성

```
>>> 'apple, pear, grape, pineapple, orange'.split(', ')
['apple', 'pear', 'grape', 'pineapple', 'orange']
```

- **구분자 문자열과 문자열 리스트 연결하기**
  - **join(리스트) : 구분자 문자열과 문자열 리스트의 요소를 연결하여 문자열로 생성**
  - 다음은 공백 ' '에 join을 사용하여 각 문자열 사이에 공백이 들어가도록 생성

```
>>> ' '.join(['apple', 'pear', 'grape', 'pineapple', 'orange'])
'apple pear grape pineapple orange'
```

  - 마이너스 '-'에 join을 사용하면 각 문자열 사이에 마이너스가 들어감

```
>>> '-'.join(['apple', 'pear', 'grape', 'pineapple', 'orange'])
'apple-pear-grape-pineapple-orange'
```

- **소문자를 대문자로 바꾸기**
  - **upper() : 문자열의 문자를 모두 대문자로 바꿈**
  - 만약 문자열 안에 대문자가 있다면 그대로 유지됨

```
>>> 'python'.upper()
'PYTHON'
```

- **대문자를 소문자로 바꾸기**
  - **lower() : 문자열의 문자를 모두 소문자로 바꿈**
  - 만약 문자열 안에 소문자가 있다면 그대로 유지됨

```
>>> 'PYTHON'.lower()
'python'
```

- **왼쪽 공백 삭제하기**
  - **lstrip() : 문자열에서 왼쪽에 있는 연속된 모든 공백을 삭제** (l은 왼쪽(left)을 의미)

```
>>> '   Python   '.lstrip()
'Python   '
```

- **오른쪽 공백 삭제하기**
  - **rstrip() : 문자열에서 오른쪽에 있는 연속된 모든 공백을 삭제**
    (r은 오른쪽(right)을 의미)

```
>>> '   Python   '.rstrip()
'   Python'
```

- **양쪽 공백 삭제하기**
  - **strip() : 문자열에서 양쪽에 있는 연속된 모든 공백을 삭제**

```
>>> '   Python   '.strip()
'Python'
```

- **왼쪽의 특정 문자 삭제하기**
  - **lstrip('삭제할문자들')**과 같이 삭제할 문자들을 문자열 형태로 넣어주면 문자열 왼쪽에 있는 해당 문자를 삭제
  - 다음은 문자열 왼쪽의 ,(콤마)와 .(점)을 삭제
  - 단, 여기서는 공백을 넣지 않았으므로 공백은 그대로 둠

```
>>> ', python.'.lstrip(',.')
' python.'
```

- **오른쪽의 특정 문자 삭제하기**
  - **rstrip('삭제할문자들')**과 같이 삭제할 문자들을 문자열 형태로 넣어주면 문자열 오른쪽에 있는 해당 문자를 삭제
  - 다음은 문자열 오른쪽의 ,(콤마)와 .(점)을 삭제
  - 마찬가지로 공백을 넣지 않았으므로 공백은 그대로 둠

```
>>> ', python.'.rstrip(',.')
', python'
```

- **양쪽의 특정 문자 삭제하기**
  - **strip('삭제할문자들')**과 같이 삭제할 문자들을 문자열 형태로 넣어주면 문자열 양쪽에 있는 해당 문자를 삭제
  - 다음은 문자열 양쪽의 ,(콤마)와 .(점)을 삭제
  - 여기서도 공백을 넣지 않았으므로 공백은 그대로 둠

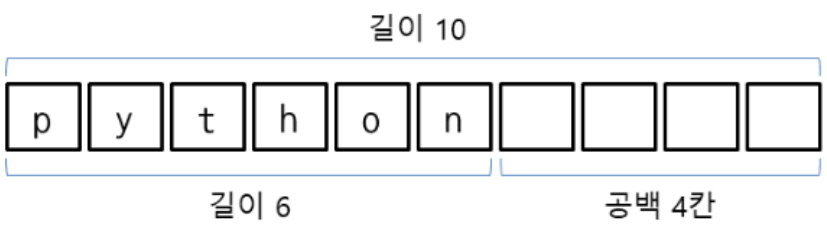
```
>>> ', python.'.strip(',.')
' python'
```

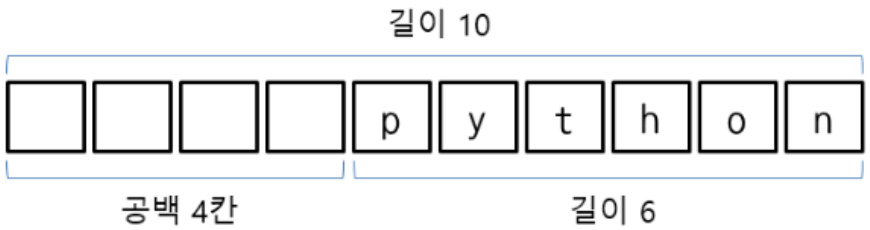
- **문자열을 왼쪽 정렬하기**
  - ljust(길이)는 문자열을 지정된 길이로 만든 뒤 왼쪽으로 정렬하며 남는 공간을 공백으로 채움(l은 왼쪽(left)을 의미)

```
>>> 'python'.ljust(10)
'python    '
```

- **문자열을 오른쪽 정렬하기**
  - rjust(길이)는 문자열을 지정된 길이로 만든 뒤 오른쪽으로 정렬하며 남는 공간을 공백으로 채움(r은 오른쪽(right)을 의미)

```
>>> 'python'.rjust(10)
'    python'
```



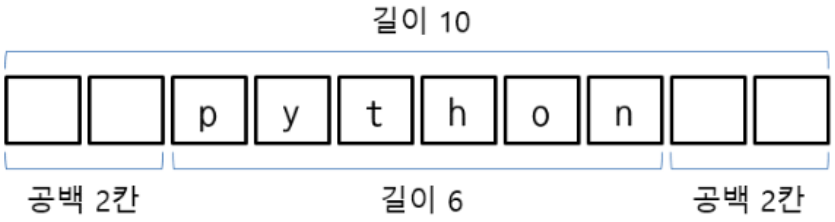


- **문자열을 가운데 정렬하기**
  - **center(길이) : 문자열을 지정된 길이로 만든 뒤 가운데로 정렬하며 남는 공간을 공백으로 채움**

```
>>> 'python'.center(10)
'  python  '
```

- 문자열을 가운데 정렬하기
  - 만약 가운데로 정렬했을 때 남는 공간이 홀수가 된다면 왼쪽에 공백이 한 칸 더 들어감

```
>>> 'python'.center(11)
'   python  '
```



- **메서드 체이닝**
  - **메서드 체이닝은 메서드를 줄줄이 연결한다고 해서 메서드 체이닝(method chaining)이라 부름**
  - 문자열을 오른쪽으로 정렬한 뒤 대문자로 바꿈

```
>>> 'python'.rjust(10).upper()
'    PYTHON'
```

  - 사실 문자열을 입력 받을 때 자주 사용했던 input().split()도 input()이 반환한 문자열에 split을 호출하는 메서드 체이닝임

- **문자열 왼쪽에 0 채우기**
  - **zfill(길이) : 지정된 길이에 맞춰서 문자열의 왼쪽에 0을 채움(zero fill을 의미)**
  - 단, 문자열의 길이보다 지정된 길이가 작다면 아무것도 채우지 않음

```
>>> '35'.zfill(4)        # 숫자 앞에 0을 채움
'0035'
>>> '3.5'.zfill(6)       # 숫자 앞에 0을 채움
'0003.5'
>>> 'hello'.zfill(10)    # 문자열 앞에 0을 채울 수도 있음
'00000hello'
```

- **문자열 위치 찾기**
  - **find('찾을문자열') : 문자열에서 특정 문자열을 찾아서 인덱스를 반환**하고, 문자열이 없으면 -1을 반환함
  - find는 왼쪽에서부터 문자열을 찾는데, 같은 문자열이 여러 개일 경우 처음 찾은 문자열의 인덱스를 반환함

```
>>> 'apple pineapple'.find('pl')
2
>>> 'apple pineapple'.find('xy')
-1
```

- **오른쪽에서부터 문자열 위치 찾기**
  - **rfind('찾을문자열') : 오른쪽에서부터 특정 문자열을 찾아서 인덱스를 반환**하고, 문자열이 없으면 -1을 반환함(r은 오른쪽(right)을 의미)
  - 같은 문자열이 여러 개일 경우 처음 찾은 문자열의 인덱스를 반환함

```
>>> 'apple pineapple'.rfind('pl')
12
>>> 'apple pineapple'.rfind('xy')
-1
```

- **문자열 위치 찾기**
  - **index('찾을문자열')은 왼쪽에서부터 특정 문자열을 찾아서 인덱스를 반환함**
  - 단, 문자열이 없으면 에러를 발생시킴
  - index도 같은 문자열이 여러 개일 경우 처음 찾은 문자열의 인덱스를 반환함

```
>>> 'apple pineapple'.index('pl')
2
```

- **오른쪽에서부터 문자열 위치 찾기**
  - **rindex('찾을문자열') : 오른쪽에서부터 특정 문자열을 찾아서 인덱스를 반환함** (r은 오른쪽(right)을 의미)
  - 마찬가지로 문자열이 없으면 에러를 발생시키며 같은 문자열이 여러 개일 경우 처음 찾은 문자열의 인덱스를 반환함

```
>>> 'apple pineapple'.rindex('pl')
12
```

- **문자열 개수 세기**
  - **count('문자열') : 현재 문자열에서 특정 문자열이 몇 번 나오는지 알아냄**

```
>>> 'apple pineapple'.count('pl')
2
```

- **문자열 서식 지정자와 formatting 사용하기**
  - 예를 들어 학생의 이름과 평균 점수를 출력한다고 하자

```
제임스의 평균 점수는 85.3점입니다.
```

  - 만약 학생이 바뀐다면 이름과 점수 부분도 바뀜

```
마리아의 평균 점수는 98.7점입니다.
```

  - 문자열 안에서 특정 부분을 원하는 값으로 바꿀 때 서식 지정자 또는 문자열 formatting을 사용함

- **서식 지정자로 문자열 넣기**

  - '%s' % '문자열'

```
>>> 'I am %s.' % 'james'
'I am james.'
```

  - **%s는 문자열이라는 뜻이며 string의 s임**
  - 문자열을 바로 지정하지 않고 변수를 지정할 수도 있음

```
>>> name = 'maria'
>>> 'I am %s.' % name
'I am maria.'
```

- **서식 지정자로 숫자 넣기**

  - '%d' % 숫자

```
>>> 'I am %d years old.' % 20
'I am 20 years old.'
```

  - 숫자는 %d를 넣고 % 뒤에 숫자를 지정하면 됨
  - %d는 10진 정수 decimal integer의 d임

- **서식 지정자로 소수점 표현하기**

  - '%f' % 숫자

```
>>> '%f' % 2.3
'2.300000'
```

  - **실수를 넣을 때는 %f를 사용하며 고정 소수점 fixed point의 f임**
  - 소수점 이하 자릿수를 지정하고 싶다면 다음과 같이 f 앞에 .(점)과 자릿수를 지정

  - '%.자릿수f' % 숫자

```
>>> '%.2f' % 2.3
'2.30'
>>> '%.3f' % 2.3
'2.300'
```

- **서식 지정자로 문자열 정렬하기**
  - 다음과 같이 % 뒤에 숫자를 붙이면 문자열을 지정된 길이로 만든 뒤 오른쪽으로 정렬하고 남는 공간을 공백으로 채움

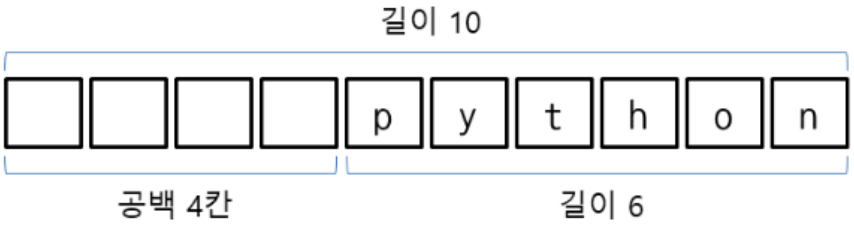
• %길이s

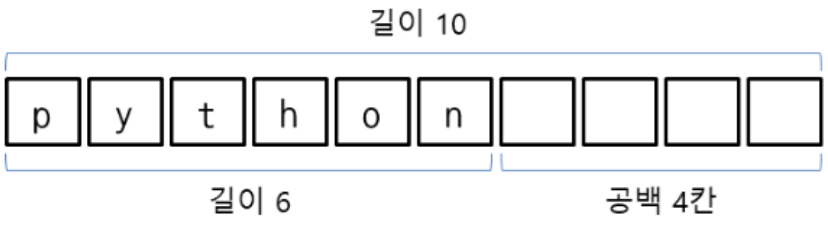
```
>>> '%10s' % 'python'
'    python'
```

- **서식 지정자로 문자열 정렬하기**
  - **왼쪽 정렬 : 문자열 길이에 -를 붙여주면 됨**

• %-길이s

```
>>> '%-10s' % 'python'
'python    '
```





- **서식 지정자로 문자열 안에 값 여러 개 넣기**
  - 문자열 안에 값을 두 개 이상 넣으려면 %를 붙이고,
    괄호 안에 값(변수)을 콤마로 구분해서 넣어주면 됨
  - **값을 괄호로 묶지 않으면 에러가 발생하므로 주의**해야 함

• '%d %s' % (숫자, '문자열')

```
>>> 'Today is %d %s.' % (3, 'April')
'Today is 3 April.'
```

  - 서식 지정자가 여러 개면 괄호 안의 값(변수) 개수도 서식 지정자 개수와
    똑같이 맞춰주어야 함
  - 만약 서식 지정자를 서로 붙이면 결과도 붙어서 나오므로 주의해야 함

```
>>> 'Today is %d%s.' % (3, 'April')
'Today is 3April.'
```

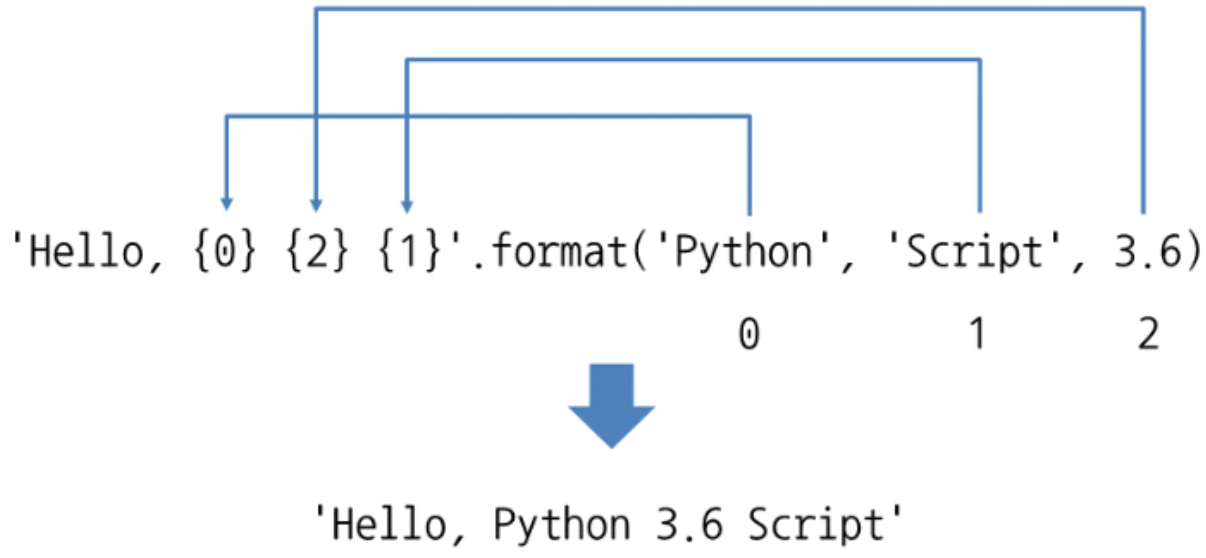
- **format 메서드 사용하기**
  - 파이썬은 문자열을 만들 때 서식 지정자 방식보다 더 간단한 문자열 포매팅(string formatting)을 제공함

• '{인덱스}'.format(값)

```
>>> 'Hello, {0}'.format('world!')
'Hello, world!'
>>> 'Hello, {0}'.format(100)
'Hello, 100'
```

- **format 메서드 사용하기**
  - 인덱스의 순서와 format에 지정된 값의 순서를 주목하자

```
>>> 'Hello, {0} {2} {1}'.format('Python', 'Script', 3.6)
'Hello, Python 3.6 Script'
```



- **format 메서드 사용하기**
  - 문자열에 'Python'이 두 개, 'Script'가 두 개 들어감

```
>>> '{0} {0} {1} {1}'.format('Python', 'Script')
'Python Python Script Script'
```

- **format 메서드에서 인덱스 생략하기**
  - 만약 { }에서 인덱스를 생략하면 format에 지정한 순서대로 값이 들어감

```
>>> 'Hello, {} {} {}'.format('Python', 'Script', 3.6)
'Hello, Python Script 3.6'
```

- **format 메서드에서 인덱스 대신 이름 지정하기**
  - { }에 인덱스 대신 이름을 지정할 수도 있음

```
>>> 'Hello, {language} {version}'.format(language='Python', version=3.6)
'Hello, Python 3.6'
```

- **문자열 포매팅에 변수를 그대로 사용하기**
  - 다음과 같이 변수에 값을 넣고 { }에 변수 이름을 지정하면 됨
  - **문자열 앞에 포매팅(formatting)이라는 뜻으로 f를 붙임**

```
>>> language = 'Python'
>>> version = 3.6
>>> f'Hello, {language} {version}'
'Hello, Python 3.6'
```

- **format 메서드로 문자열 정렬하기**
  - 다음과 같이 **인덱스 뒤에 :(콜론)을 붙이고 정렬할 방향과 길이를 지정**해주면 됨

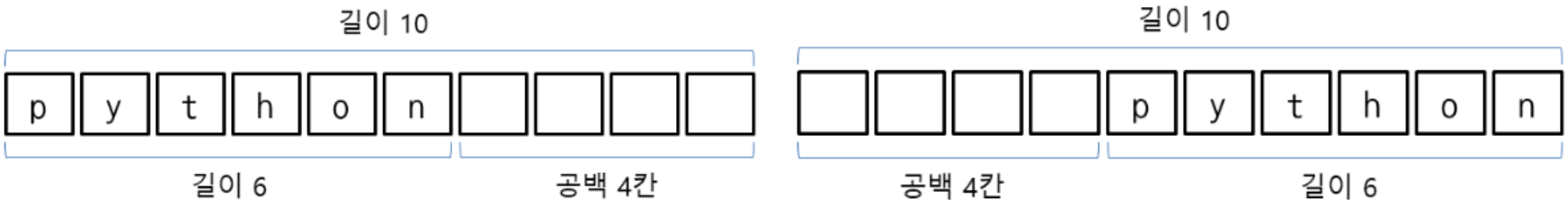
  - '{인덱스:<길이}'.format(값)

```
>>> '{0:<10}'.format('python')
'python    '
```

- **format 메서드로 문자열 정렬하기**
  - 다음과 같이 **>**을 넣어서 **오른쪽**을 가리키도록 만들면 문자열을 지정된 길이로 만든 뒤 오른쪽으로 정렬하고 남는 공간을 공백으로 채움

  - '{인덱스:>길이}'.format(값)

```
>>> '{0:>10}'.format('python')
'    python'
```



- **format 메서드로 문자열 정렬하기**
  - 참고로 인덱스를 사용하지 않는다면 :(콜론)과 정렬 방법만 지정해도 됨

```
>>> '{:>10}'.format('python')
'    python'
```

- **숫자 개수 맞추기**
  - { }를 사용할 때는 인덱스나 이름 뒤에 :(콜론)를 붙이고 03d처럼 0과 숫자 개수를 지정하면 됨

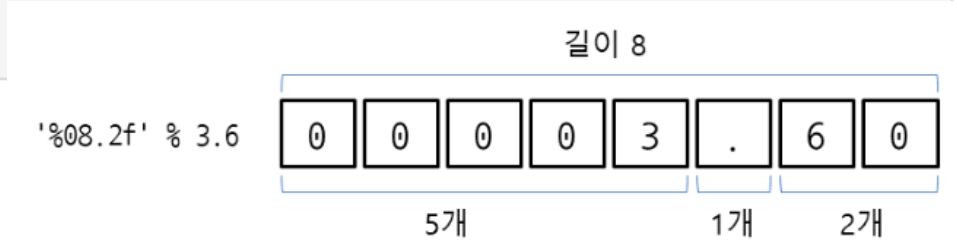
- '%0개수d' % 숫자
- '{인덱스:0개수d}'.format(숫자)

```
>>> '%03d' % 1
'001'
>>> '{0:03d}'.format(35)
'035'
```

  - 소수점 이하 자릿수를 지정하고 싶으면 %08.2f처럼 .(점) 뒤에 자릿수를 지정해줌

- '%0개수.자릿수f' % 숫자
- '{인덱스:0개수.자릿수f}'.format(숫자)

```
>>> '%08.2f' % 3.6
'00003.60'
>>> '{0:08.2f}'.format(150.37)
'00150.37'
```



- **채우기와 정렬을 조합해서 사용하기**

```
>>> '{0:0<10}'.format(15)    # 길이 10, 왼쪽으로 정렬하고 남는 공간은 0으로 채움
'1500000000'
>>> '{0:0>10}'.format(15)    # 길이 10, 오른쪽으로 정렬하고 남는 공간은 0으로 채움
'0000000015'
```

  - 소수점 자릿수와 실수 자료형 f를 지정

```
>>> '{0:0>10.2f}'.format(15)    # 길이 10, 오른쪽으로 정렬하고 소수점 자릿수는 2자리
'0000015.00'
```

  - 특히 채우기 부분에 0이 아닌 다른 문자를 넣어도 됨
  - 공백을 넣어도 되고, 문자를 넣어도 됨
  - 만약 채우기 부분을 생략하면 공백이 들어감

```
>>> '{0: >10}'.format(15)    # 남는 공간을 공백으로 채움
'        15'
>>> '{0:>10}'.format(15)     # 채우기 부분을 생략하면 공백이 들어감
'        15'
>>> '{0:x>10}'.format(15)    # 남는 공간을 문자 x로 채움
'xxxxxxxx15'
```

- **cf) 금액에서 천단위로 콤마 넣기**
  - format(숫자, ' , ')

```
>>> format(1493500, ' ,')
' 1,493,500'
```

  - format 함수는 서식 지정자와 사용 가능

```
>>> '%20s' % format(1493500,',')  #길이 20, 오른쪽 정렬
'          1,493,500'
```

  - fomatting에서 사용법 : 콜론 뒤에 ,

```
>>> '{0:,}'.format(1493500)
'1,493,500'

>>> '{0:>20,}'.format(1493500)    #길이 20, 오른쪽 정렬
'          1,493,500'

>>> '{0:0>20,}'.format(1493500)   #길이 20, 오른쪽 정렬, 나머지 빈칸 0으로 채움
'000000000001,493,500'
```