문자열과 텍스트 파일 데이터 다루기

문자열과 텍스트 파일 데이터 다루기

문자열과 텍스트 파일 데이터 다루기

- 데이터가 포함된 텍스트 파일에서 문자 열을 읽어올 수 있어야 함
- 사용목적에 맞게 문자열을 처리할 수 있어야 함
- 파이썬에서는 문자열 처리를 위한 다양한 내장 문자열 메서드가 제공되므로 문자열을 처리하기 쉬움
- 파이썬에서 문자열을 처리하는 방법과 텍스트 파일의 내용을 읽어서 처리하는 방법

문자열 다루기

문자열 다루기

- 파이썬에서는 큰따옴표(")나 작은따옴표(') 안에 들어 있는 문자의 집합을 문자열
- 한 텍스트 파일의 내용을 읽어 오면 그것도 문자열임
- 텍스트 파일을 읽어서 가져온 문자열은 대부분 문자열 처리를 통해 원하는 형태의 데이터로 변환해서 이용
- 문자열을 처리하기 위해서 는 문자열 분리, 불필요한 문자열 제거, 문자열 연결 등을 할 수 있어야 함

문자열 다루기

문자열 분리하기

- 문자열을 부분 문자열로 나누고 싶을 때는 split() 메서드를 이용
- split() 메서드의 사용법

  str.split([sep])

- split() 메서드는 문자열(str)에서 구분자(separator)인 sep을 기준으로 문자열을 분리해 리스트로 반환
- 소괄호 안의 대괄호([ ]) 부분은 생략할 수 있음
- 구분자(sep)를 입력하지 않고 str.split()을 수행하면 문자열 사이의 모든 공백과 개행문자(₩n)를 없애고 분리된 문자열을 항목으로 담은 리스트를 반환

문자열 다루기

문자열 분리하기

- split()을 이용해 문자열을 분리
- 콤마(,)로 구분된 단어가 여러 개 적힌 문자열이 있을 때 구분자를 콤마(,)로 입력해 split() 메서드를 적용하면 콤마를 기준으로 단어를 분리

```
[ ] coffee_menu_str = "에스프레소,아메리카노,카페라테,카푸치노"
    coffee_menu_str.split(',')
```

- 문자열을 변수에 할당하지 않고 문자열에 직접 split() 메서드를 사용

```
[ ] "에스프레소,아메리카노,카페라테,카푸치노".split(',')
```

문자열 다루기

문자열 분리하기

- split()을 이용해 문자열을 분리
- 콤마(,)로 구분된 단어가 여러 개 적힌 문자열이 있을 때 구분자를 콤마(,)로 입력해 split() 메서드를 적용하면 콤마를 기준으로 단어를 분리

```
[ ] coffee_menu_str = "에스프레소,아메리카노,카페라테,카푸치노"
    coffee_menu_str.split(',')
```

- 문자열을 변수에 할당하지 않고 문자열에 직접 split() 메서드를 사용

```
[ ] "에스프레소,아메리카노,카페라테,카푸치노".split(',')
```

문자열 다루기

문자열 분리하기

- 하나의 공백으로 구분된 단어가 여러 개 적힌 문자열은 공백 구분자를 인자로 갖는 split()메서드로 분리

```
[ ] "에스프레소 아메리카노 카페라테 카푸치노".split(' ')
```

- 공백 구분자를 인자로 갖는 split() 메서드를 사용할 경우 인자 없이 split()를 적용할 수 있음

```
[ ] "에스프레소 아메리카노 카페라테 카푸치노".split()
```

문자열 다루기

문자열 분리하기

- 문자열에 인자 없이 split()를 사용하면 문자열 사이의 모든 공백과 개행 문자(₩n)를 없애고 분리된 문자열을 반환
- 단어 사이에 공백과 개행문자가 아무리 많더라도 split() 이용하면 공백과 개행문자를 모두 없애고 문자열을 분리

```
[ ] "   에스프레소  ₩n₩n  아메리카노  ₩n  카페라테   카푸치노 ₩n₩n".split()
```

문자열 다루기

문자열 분리하기

- 문자열을 분리할 때 인자 maxsplit을 추가하면 앞에서부터 원하는 횟수만큼만 문자열을 분리 할 수 있음

  str.split([sep ,] maxsplit=숫자)

- 문자열(str)을 구분자 sep(생략 가능)을 기준으로 maxsplit만큼 분리해 리스트로 반환

문자열 다루기

문자열 분리하기

- maxsplit을 지정해 문자열을 분리

```
[ ] "에스프레소 아메리카노 카페라테 카푸치노".split(maxsplit=2)
```

- 인자로 지정한 maxSplit=2로 인해 앞에서부터 2개의 공백(sep)까지만 문자열을 나눠 결과적으로 3개의 항목이 담긴 리스트를 반환

문자열 다루기

문자열 분리하기

- split()에서 구분자(sep)와 분할 횟수(maxsplit=숫자)를 모두 지정하여 국가번호까지 있는 전화번호에서 국가번호를 뺀 나머지 번호를 구하기

```
[ ] phone_number = "+82-01-2345-6789" # 국가 번호가 포함된 전화번호
    split_num = phone_number.split("-", 1) # 국가 번호와 나머지 번호 분리

    print(split_num)
    print("국내전화번호: {0}".format(split_num[1]))
```

문자열 다루기

필요 없는 문자열 삭제하기

- 문자열에서는 앞뒤 공백 혹은 개행문자와 같이 불필요한 부분을 지워야 할 때
- 사용할 수 있는 것이 strip() 메서드
- strip() 메서드의 사용법

```
str.strip([chars])
```

문자열 다루기

필요 없는 문자열 삭제하기

- strip()메서드는 문자열(str)의 앞과 뒤에서 시작해서 지정한 문자(chars) 외의 다른 문자를 만날 때까지 지정한 문자(chars)를 모두 삭제한 문자열을 반환
- 지정한 문자(chars)와 일치하는 것이 없으면 문자열(str)을 그대로 반환
- 지정한 문자(chars)가 여러 개일 경우 순서는 상관이 없음
- 지정한 문자(chars) 없이 str.strip()를 수행하면 문자열 앞과 뒤의 모든 공백과 개행문자(₩n)를 삭제한 후에 문자열을 반환

문자열 다루기

필요 없는 문자열 삭제하기

- strip()을 이용해 문자열에서 앞과 뒤의 모든 공백을 지우기

  str = “ Python “

  str.strip()

- 문자열 "aaaaPythonaa"에서 앞뒤의 모든 ‘a’를 제거하고 싶다면
- 없애고자 하는 문자 (chars)를 ’a’로 지정해서 strip() 메서드를 실행

```
[ ]  "aaaaPythonaaa".strip('a')
```

문자열 다루기

필요 없는 문자열 삭제하기

- 문자열에서 지우고자 하는 문자가 하나가 아니라 둘이면 strip() 메서드를 두 번 사용하면 됨
- 첫 번째로 지우려고 하는 문자를 지정해서 strip()메서드를 실행

```
[ ] test_str = "aaabbPythonbbbaa"
    temp1 = test_str.strip('a') # 문자열 앞뒤의 'a' 제거
    temp1
```

- 변수 test_str에 할당한 문자열의 앞과 뒤에서 'a'가 제거 됨
- 두 번째로 지우고자 하는 문자(여기서는 b)를 지정해서 strip() 메서드를 실행

```
[ ] temp1.strip('b') # 문자열 앞뒤의 'b' 제거
```

문자열 다루기

필요 없는 문자열 삭제하기

- strip()메서드의 경우에는 제거하고자 하는 문자를 하나만 지정해서 여러 번 수행 할 수도 있지만
- 지우고자 하는 문자를 모두 지정해서 한 번에 제거할 수도 있음

```
[ ] test_str.strip('ab') # 문자열 앞뒤의 'a'와 'b' 제거
```

- 지우고자 하는 문자를 여러 개 지정할 때 순서는 상관이 없음
- 지우고자 하는 문자의 순서를 바꿔서 지정해도 됨

```
[ ] test_str.strip('ba')
```

문자열 다루기

필요 없는 문자열 삭제하기

- 문자열의 앞과 뒤에서 좀 더 많은 문자를 삭제

```
[ ] test_str_multi = "##***!!!##....  Python is powerful.!...  %%!#..    "
    test_str_multi.strip('*.#! %')
```

- strip()메서드에 '*.#! %'를 인자로 지정해서 문자열의 앞과 뒤의 다양한 문자('*.#! %')를 모두 삭제
- 인자로 지정한 문자의 순서는 상관없기 때문에 인자를 지정해도 결과는 같음

```
[ ] test_str_multi.strip('%* !#.')
```

문자열 다루기

필요 없는 문자열 삭제하기

- strip()메서드를 이용해 문자열 앞뒤의 공백을 제거

```
[ ] "  Python   ".strip(' ')
```

- 문자열 앞뒤의 공백과 개행문자(₩n)를 지우고 싶을 때는 지우고자 하는 문자를 '₩n' 혹은 '₩n'로 지정해서 strip() 메서드를 실행

```
[ ] "₩n  Python  ₩n₩n".strip(' ₩n')
```

- 어떤 문자열에서 지우고자 하는 문자가 공백과 개행문자라면 인자를 지정하지 않고 strip()을 실행해도 됨

```
[ ] "₩n  Python  ₩n₩n".strip()
```

문자열 다루기

필요 없는 문자열 삭제하기

- strip()메서드는 문자열(str)의 앞뒤에서 지정한 문자(chars) 외 다른 문자를 만날 때까지만 지정한 문자(chars)를 모두 삭제
- "aaaBallaaa" 문자열에 strip('a')수행하면 'Ball'의 'a'는 지워지지 않음

```
[ ]  "aaaBallaaa".strip('a')
```

문자열 다루기

필요 없는 문자열 삭제하기

- 공백과 개행문자가 다른 문자들 사이에 있는 문자열에 인자 없이 strip() 메서드를 적용하면
- 문자열의 앞뒤 공백과 개행문자는 모두 삭제되지만 문자열 사이에 있는 공백과 개행문자는 삭제되지 않음

```
[ ]  "₩n This is very ₩n fast. ₩n₩n".strip()
```

문자열 다루기

필요 없는 문자열 삭제하기

- strip() 메서드는 문자열의 앞과 뒤 양쪽을 검색해 지정한 문자를 삭제하는 역할
- 앞이나 뒤 중에서 한쪽만 삭제하고 싶으면 lstrip()나 rstrip() 메서드를 사용
- 문자열 왼쪽 (즉,앞쪽) 부분만 삭제하려면 lstrip()메서드
- 문자열 오른쪽 (즉,뒤쪽) 부분만 삭제하려면 rstrip()메서드를 이용

문자열 다루기

필요 없는 문자열 삭제하기

```
[ ] str_lr = "000Python is easy to learn.000"
    print(str_lr.strip('0'))
    print(str_lr.lstrip('0'))
    print(str_lr.rstrip('0'))
```

- 문자열을 할당한 변수 str_lr에 strip(), lstrip() , rstrip() 메서드를 각각 적용
- strip()를 적용한 결과로 문자열에서 양쪽 모두 0이 삭제되고
- lstrip()이나 rstrip()을 적용한 결과로 문자열의 왼쪽과 오른쪽에서 각각 0이 삭제

문자열 다루기

필요 없는 문자열 삭제하기

- 콤마와 공백을 포함한 문자열에서 콤마를 기준으로 문자열을 분리하고 공백을 모두 제거
- coffee_menu 변수에는 콤마와 공백을 포함한 여러 커피 종류가 있음
- split(',')을 이용해 콤마를 구분자로 삼아 문자열을 리스트로 분리

```
[ ] coffee_menu = "  에스프레소, 아메리카노,    카페라테    , 카푸치노  "
    coffee_menu_list = coffee_menu.split(',')
    coffee_menu_list
```

문자열 다루기

필요 없는 문자열 삭제하기

- coffee_menu_list에는 공백을 포함한 문자열을 항목으로 갖는 리스트가 반환
- 리스트 변수 coffee_menu_list의 모든 항목에 공백을 제거하기 위해 항목마다 strip() 메서드를 적용

문자열 다루기

필요 없는 문자열 삭제하기

- 공백이 제거된 문자열은 append()를 이용해 리스트 변수 coffee_list에 하나씩 추가하면 최종적으로 원하는 결과를 얻을 수 있음

```
[ ] coffee_list = [] # 빈 리스트 생성
    for coffee in coffee_menu_list:
        temp = coffee.strip() # 문자열의 공백 제거
        coffee_list.append(temp) # 리스트 변수에 공백이 제거된 문자열 추가

    print(coffee_list) #최종 문자열 리스트 출력
```

문자열 다루기

문자열 연결하기

- 더하기 연산자(+)로 두 문자열을 연결하는 방법

```
[ ]  name1 = "철수"
     name2 = "영미"

     hello = "님, 주소와 전화 번호를 입력해 주세요."
     print(name1 + hello)
     print(name2 + hello)
```

- 더하기 연산자를 이용하면 문자열과 문자열을 연결

문자열 다루기

문자열 연결하기

- 문자열이 아니라 리스트의 모든 항목을 하나의 문자열로 만들려면?
- 리스트의 모든 항목을 하나의 문자열 join() 메서드를 사용
- join() 메서드를 사용하는 방법

  str.join(seq)

- join() 메서드는 문자열을 항목으로 갖는 시퀀스(seq)의 항목 사이에 구분자 문자열(str)을 모두 넣은 후에 문자열로 반환
- 시퀀스는 리스트나 튜플과 같이 여러 데이터를 순서대로 담고 있는 나열형 데이터

문자열 다루기

문자열 연결하기

- 문자열을 항목으로 갖는 문자열 리스트를 join() 메서드를 이용해 문자열로 변환하는 과정
- 문자열 리스트의 항목 사이에는 구분자 문자열(한 칸 공백)이 들어감

문자열 다루기

문자열 연결하기

- join() 메서드를 이용해 문자열 리스트를 문자열로 변환
- 문자열을 항목으로 갖는 리스트를 생성

```
[ ] address_list = ["서울시","서초구","반포대로","201(반포동)"]
    address_list
```

- 문자열 리스트(address_list)를 공백으로 연결해서 문자열을 생성
- 구분자 문자열은 한 칸의 공백이 됨

```
[ ] a = " "
    a.join(address_list)
```

문자열 다루기

문자열 연결하기

- 구분자 문자열을 변수에 할당하지 않고 직접 지정할 수도 있음

```
[ ]  " ".join(address_list)
```

- 문자열 리스트를 여러 문자로 이뤄진 구분자 문자열("*^-^*")로 연결해서 문자열로 변환

```
[ ]  "*^-^*".join(address_list)
```

문자열 다루기

문자열 찾기

- 문자열에서 원하는 단어를 찾을 때 사용할 수 있는 find() 메서드

```
str.find(search_str)
```

- find() 메서드는 문자열(str)에서 찾으려는 검색 문자열(search_str)과 첫번째로 일치하는 문자열(str)의 위치를 반환
- 문자열의 위치는 0부터 시작
- 문자열에서 검색 문자열을 찾을 수 없으면 -1을 반환

문자열 다루기

문자열 찾기

- find()를 이용해 문자열에서 특정 문자열의 위치를 찾음
- 특정 문자열을 찾이서 맨 처음 발견된 위치를 반환하고 찾으려는 문자열이 없으면 -1을 반환

```
[ ] str_f = "Python code."

    print("찾는 문자열의 위치:", str_f.find("Python"))
    print("찾는 문자열의 위치:", str_f.find("code"))
    print("찾는 문자열의 위치:", str_f.find("n"))
    print("찾는 문자열의 위치:", str_f.find("easy"))
```

문자열 다루기

문자열 찾기

- str.find(search_str)에 시작 위치(start)와 끝 위치(end)를 추가로 지정해서 검색 범위를 설정할 수도 있음
- str.find(search_str, start, end)
- start ~ end-1 범위에서 검색 문자열(search_str)을 검색해 일치하는 문자열(str)의 위치를 반환
- 지정된 범위에서 찾지 못하면 -1 을 반환

문자열 다루기

문자열 찾기

- 시작 위치만 지정해서 검색 범위를 설정할 수도 있음
- str.find(search_str, start)
- 검색 범위는 start부터 문자열(str)의 끝이 됨

문자열 다루기

문자열 찾기

- 시작 위치와 끝 위치를 지정해 문자열

```
[ ]  str_f_se = "Python is powerful. Python is easy to learn."

     print(str_f_se.find("Python", 10, 30)) # 시작 위치(start)와 끝 위치(end) 지정
     print(str_f_se.find("Python", 35)) # 찾기 위한 시작 위치(start) 지정
```

- find() 메서드는 찾으려는 문자열과 일치하는 첫 번째 위치를 반환

문자열 다루기

문자열 찾기

- 해당 문자열이 몇 번 나오는지 알고 싶다면 count() 메서드를 이용

```
str.count(search_str)

str.count(search_str, start)

str.count(search_str, start, end)
```

- count()메서드는 문자열(str) 에서 찾고자 하는 문자열(search_str)과 일치하는 횟수를 반환하고, 없으면 0을 반환
- find()와 마찬가지로 start와 end로 검색 범위를 지정할 수도 있음

문자열 다루기

문자열 찾기

- count() 메서드를 사용하여 찾는 문자열 횟수 반환

```
[ ] str_c = "Python is powerful. Python is easy to learn. Python is open."

    print("Python의 개수는?:", str_c.count("Python"))
    print("powerful의 개수는?:", str_c.count("powerful"))
    print("IPython의 개수는?:", str_c.count("IPython"))
```

문자열 다루기

문자열 찾기

- 다른 찾기 메서드로 startwith() 메서드와 entwith() 메서드
- 각각 문자열이 지정된 문자 열로 시작하는지 끝나는지를 검사할 때 사용

```
str.startswith(prefix)

str.startswith(prefix, start)

str.startswith(prefix, start, end)
```

- startswith() 메서드는 문자열(str)이 지정된 문자열(prefix)로 시작되면 True, 그렇지 않으면 False를 반환

문자열 다루기

문자열 찾기

- find()와 마찬가지로 start와 end로 범위를 지정할 수도 있음

```
str.endswith(suffix)

str.endswith(suffix, start)

str.endswith(suffix, start, end)
```

- endswith()메서드는 문자열(str)이 지정된 문자열(suffix)로 끝나면 True, 그렇지 않으면 False를 반환
- start와 end로 범위를 지정할 수도 있음

문자열 다루기

문자열 찾기

- startwith() 메서드와 endwith() 메서드를 사용

```
[ ]  str_se = "Python is powerful. Python is easy to learn."

     print("Python으로 시작?:", str_se.startswith("Python"))
     print("is로 시작?:", str_se.startswith("is"))
     print(".로 끝?:", str_se.endswith("."))
     print("learn으로 끝?:", str_se.endswith("learn"))
```

문자열 다루기

문자열 바꾸기

- 문자열에서 지정한 문자열을 찾아서 바꾸는 메서드로 replace()
- str.replace(old, new[, count])
- replace() 메서드는 문자열(str)에서 지정한 문자열(old)을 찾아서 새로운 문자열(new)로 바꿈
- count는 문자열(str)에서 지정된 문자열을 찾아서 바꾸는 횟수
- 횟수를 지정하지 않으면 문자열 전체에서 찾아서 바꿈

문자열 다루기

문자열 바꾸기

- 문자열에서 지정한 문자열을 찾아서 새로운 문자열로 바꿈

```
[ ] str_a = 'Python is fast. Python is friendly. Python is open.'
    print(str_a.replace('Python', 'IPython'))
    print(str_a.replace('Python', 'IPython', 2))
```

문자열 다루기

문자열 바꾸기

- 특정 문자열을 삭제할 때도 replace() 메서드를 이용할 수 있음
- 문자열에서 '['와 ']'를 제거
- replace() 메서드에는 문자열을 하나씩만 지정할 수 있으므로
- '['와 ']'를 모두 제거하려면 replace() 메서드를 두 번 사용

```
[ ] str_b = '[Python] [is] [fast]'
    str_b1 = str_b.replace('[', '') # 문자열에서 '['를 제거
    str_b2 = str_b1.replace(']', '') # 결과 문자열에서 다시 ']'를 제거

    print(str_b)
    print(str_b1)
    print(str_b2)
```

문자열 다루기

문자열의 구성 확인하기

- 문자열이 숫자만으로 이뤄졌는지,
- 문자로만 이뤄졌는지 아니면 숫자와 문자가 모두 포함돼 있는지,
- 로마자 알파벳 대문자로만 이뤄졌는지,
- 소문자로만 이뤄졌는지 등 문자열의 구성을 알아야 할 때가 있음

문자열 다루기

문자열의 구성 확인하기

- 문자열의 구성을 확인하기 위한 메서드

메서드	설명	사용예
isalpha()	문자열이 숫자, 특수 문자, 공백이 아닌 문자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isalpha()
isdigit()	문자열이 모두 숫자로 구성돼 있을 때만 True. 그 밖에는 False 반환	str.isdigit()
isalnum()	문자열이 특수 문자나 공백이 아닌 문자와 숫자로 구성돼 있을 때만 True. 그 밖 에는 False 반환	str.isalnum()
isspace()	문자열이 모두 공백 문자로 구성돼 있을 때만 True, 그 밖에는 False 반환	str.isspace()
isupper()	문자열이 모두 로마자 대문자로 구성돼 있을 때만 True. 그 밖에는 False 반환	str.isupper()
islower()	문자열이 모두 로마자 소문자로 구성돼 있을 때만 True. 그 밖에는 False 반환	str.islower()

문자열 다루기

문자열의 구성 확인하기

- 문자열이 숫자, 특수 문자, 공백이 아닌 문자로 구성돼 있는지 검사하는 isalpha()메서드의 사용

```
[ ] print('Python'.isalpha()) # 문자열에 공백, 특수 문자, 숫자가 없음
    print('Ver. 3.x'.isalpha())# 공백, 특수 문자, 숫자 중 하나가 있음
```

- 모든 문자가 숫자로 이뤄져 있는지를 검사하는 isdigit() 메서드의 사용

```
[ ] print('12345'.isdigit()) # 문자열이 모두 숫자로 구성됨
    print('12345abc'.isdigit())# 문자열이 숫자로만 구성되지 않음
```

문자열 다루기

문자열의 구성 확인하기

- 문자열이 특수 문자나 공백이 아닌 문자와 숫자로 구성돼 있는지 검사하는 isalnum() 메서드

```
[ ] print('abc1234'.isalnum()) # 특수 문자나 공백이 아닌 문자와 숫자로 구성됨
    print(' abc1234'.isalnum()) # 문자열에 공백이 있음
```

- 문자열이 공백 문자로만 구성돼 있는지를 검사하는 isspace() 메서드

```
[ ] print('   '.isspace()) # 문자열이 공백으로만 구성됨
    print(' 1 '.isspace()) # 문자열에 공백 외에 다른 문자가 있음
```

문자열 다루기

문자열의 구성 확인하기

- 문자열이 모두 로마자 알파벳 대문자로 구성돼 있는지, 소문자로 구성돼 있는지를 각각 알아보는 isupper()와 islower() 메서드의 사용

```
[ ] print('PYTHON'.isupper()) # 문자열이 모두 대문자로 구성됨
    print('Python'.isupper()) # 문자열에 대문자와 소문자가 있음
    print('python'.islower()) # 문자열이 모두 소문자로 구성됨
    print('Python'.islower()) # 문자열에 대문자와 소문자가 있음
```

문자열 다루기

대소문자로 변경하기

- 문자열에서 로마자 알파벳을 모두 대문자나 소문자로 변경하는 lower() 와 upper() 메서드

```
str.lower()

str.upper()
```

문자열 다루기

대소문자로 변경하기

- lower() 메서드는 문자열(str)에서 로마자 알파벳의 모든 문자를 소문자로 바꾸고 upper() 메서드는 대문자로 바꿈

```
[ ] string1 =  'Python is powerful. PYTHON IS EASY TO LEARN.'
    print(string1.lower())
    print(string1.upper())
```

문자열 다루기

대소문자로 변경하기

- 파이썬에서는 로마자 알파벳 대문자와 소문자를 구분하므로 같은 의미의 문자열을 비교하더라도 대소문자까지 같지 않으면 다른 문자열

```
[ ] 'Python' == 'python'
```

- ‘Python’과 ‘python’은 의미가 같은 문자열이지만 대소문자의 차이 때문에 다른 문자열로 인식

문자열 다루기

대소문자로 변경하기

- 문자열을 모두 대문자나 소문자로 바꾼 후에 비교하면 같은 문자열이라고 인식
- lowed()와 upper() 메서드를 이용해 문자열 비교

```
[ ] print('Python'.lower() == 'python'.lower())
    print('Python'.upper() == 'python'.upper())
```

  - 문자열을 비교할 때 lower()와 upper() 메서드를 이용해 문자열을 모두 소문자 혹은 대문자로 변경한 후에 비교

텍스트 파일의 데이터를 읽고 처리하기

텍스트 파일의 데이터를 읽고 처리하기

- 파일의 내용을 한 번에 읽어오는 것이 아니라 한 줄씩 읽어서 처리
- 파일에서 읽은 내용은 문자열 데이터가 되는데 이 문자열 데이터를 원하는 형태로 분리하고
- 연산이 필요한 부분은 숫자 데이터로 변환한 후에 처리하는 방법

텍스트 파일의 데이터를 읽고 처리하기

데이터 파일 준비 및 읽기

- 데이터가 저장된 텍스트 파일을 읽고 처리하기 위해 먼저 처리할 데이터와 원하는 작업

- 데이터: 어느 커피 전문점에서 나흘 동안 기록한 메뉴별 커피 판매량
- 원하는 직업: 4일 동안 메뉴당 전체 판매량과 하루 평균 판매량 구하기

텍스트 파일의 데이터를 읽고 처리하기

데이터 파일 준비 및 읽기

- 텍스트 파일('coffeeShopSales.txt')
- 첫 번째 줄에는 각 항목의 이름이 있고
- 두 번째 줄 이후로는 각 항목의 값

```
[2]  !cat coffeeShopSales.txt
```

```
날짜    에스프레소  아메리카노  카페라테  카푸치노
10.15       10          50          45          20
10.16       12          45          41          18
10.17       11          53          32          25
10.18       15          49          38          22
```

텍스트 파일의 데이터를 읽고 처리하기

데이터 파일 준비 및 읽기

- 텍스트 파일('coffeeShopSales.txt') 읽기

```
[ ]  # file_name = 'c:₩myPyCode₩data₩coffeeShopSales.txt'
     # file_name = 'c:/myPyCode/data/coffeeShopSales.txt'

     file_name = 'coffeeShopSales.txt'

     f = open(file_name)        # 파일 열기
     for line in f:             # 한 줄씩 읽기
         print(line, end='')    # 한 줄씩 출력
     f.close()                  # 파일 닫기
```

- 파일명을 경로와 함께 지정해 file_name 변수에 할당한 후
- 파일 열기로 해당 파일을 열고 한 줄씩 읽어서 line 변수에 할당하고 출력
- line 변수에는 문자열 한 줄 전체가 들어가 있음

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- 첫 번째 줄에 있는 항목 이름을 가져와 빈칸을 기준으로 나누고
- 두 번째 줄 이후의 항목 값을 처리
- 첫 번째 줄의 항목 이름을 가져오는 코드

```
[ ]  f = open(file_name)    # 파일 열기
     header = f.readline() # 데이터의 첫 번째 줄을 읽음
     f.close()             # 파일 닫기

     header
```

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- 줄의 문자열을 분리해 리스트로 변환하려고 하는데 단어 사이에 공백과 개행문자가 있음
- 인자 없이 split() 메서드를 호출해 첫 줄의 문자열 에서 항목 이름을 분리해 리스트로 만듬

```
[ ] header_list = header.split() # 첫 줄의 문자열을 분리후 리스트로 변환
    header_list
```

  - 첫번째 줄에 있는 항목 이름을 리스트 변수인 header_list에 할당

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- for문을 이용해 두 번째 줄부터 끝줄까지의 데이터를 문자열에서 공백과 개행문자를 제거하고 각 항목을 data_list에 넣는 코드를 추가

```
[ ] f = open(file_name)            # 파일 열기
    header = f.readline()          # 데이터의 첫 번째 줄을 읽음
    header_list = header.split() # 첫 줄의 문자열을 분리한 후 리스트로 변환

    for line in f:                 # 두 번째 줄부터 데이터를 읽어서 반복적으로 처리
        data_list = line.split() # 문자열을 분리해서 리스트로 변환
        print(data_list)           # 결과 확인을 위해 리스트 출력

    f.close()                      # 파일 닫기
```

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- 출력 결과를 보면 리스트 변수 data_list의 각 항목이 문자열로 되어 있음
- 전체 판매량과 평균 을 계산하려면 일일 판매량 데이터 문자열은 숫자로 바꿔야 함
- int()나 float()을 이용하면 문자열 타입의 데이터를 정수나 실수 타입으로 변환
- 정수인 것을 int()를 이용해 판매량 데이터를 숫자로 변환

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- 커피 종류별로 생성한 빈 리스트에 항목을 추가하는 append()를 이용해 커피 종류별로 판매량 데이터를 분류

```
[ ] f = open(file_name)           # 파일 열기
    header = f.readline()         # 데이터의 첫 번째 줄을 읽음
    headerList = header.split() # 첫 줄의 문자열을 분리한 후 리스트로 변환

    espresso = []                 # 커피 종류별로 빈 리스트 생성
    americano = []
    cafelatte = []
    cappucino = []
```

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

```
for line in f:                  # 두 번째 줄부터 데이터를 읽어서 반복적으로 처리
    dataList = line.split() # 문자열에서 공백을 제거해서 문자열 리스트로 변환

    # 커피 종류별로 정수로 변환한 후, 리스트의 항목으로 추가
    espresso.append(int(dataList[1]))
    americano.append(int(dataList[2]))
    cafelatte.append(int(dataList[3]))
    cappucino.append(int(dataList[4]))

f.close() # 파일 닫기
```

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

```
print("{0}: {1}".format(headerList[1], espresso)) # 변수에 할당된 값을 출력
print("{0}: {1}".format(headerList[2], americano))
print("{0}: {1}".format(headerList[3], cafelatte))
print("{0}: {1}".format(headerList[4], cappucino))
```

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- 리스트를 이용해 4일 메뉴별 전체 판매량과 하루 평균 판매량을 구함
- 리스트, 튜플, 세트 데이터에서 항목의 합을 구하는 내장 함수 sum()과 항목의 개수(길이)를 구하는 내장 함수 len()을 이용

```
[ ]  total_sum = [sum(espresso), sum(americano), sum(cafelatte), sum(cappucino)]
     total_mean = [sum(espresso)/len(espresso), sum(americano)/len(americano),
                   sum(cafelatte)/len(cafelatte), sum(cappucino)/len(cappucino) ]

     for k in range(len(total_sum)):
         print('[{0}] 판매량'.format(headerList[k+1]))
         print('- 4일 전체: {0}, 하루 평균: {1}'.format(total_sum[k], total_mean[k]))
```

텍스트 파일의 데이터를 읽고 처리하기

파일에서 읽은 문자열 데이터 처리

- 날짜별로 커피 판매량 데이터가 저장된 텍스트 파일에서 메뉴별로 판매량 데이터를 읽어와서
- 숫자로 변환한 후 리스트의 합과 길이를 구하는 내장 함수를 이용해
- 메뉴별 전체 판매량과 평균 판매량을 구함

정리

정리

- 문자열의 다양한 처리 방법
- 문자열 분리, 삭제, 연결, 찾기 및 바꾸기를 위한 메서드와 사용법
- 문자열의 구성 요소를 확인하는 방법과 대소문자로 바꾸는 방법
- 문자열 데이터를 읽어서 원하는 정보로 가공하는 방법