연습문제 풀이

1장

1-1 ex1-1.py

사용자가 입력한 값이 짝수인지 홀수인지 계산해서 결과를 알려주는 프로그램을 의사코드로 작성하면 다음과 같다.

1. 사용자로부터 정수를 입력받는다.
2. 입력한 수를 2로 나눠서 나머지 값을 구한다.
3. 나머지 값이 1이면 홀수, 0이면 짝수로 분류한다.
4. 분류한 결괏값을 출력한다.

3장

3-1 ex3_1.py

```
season = input('좋아하는 계절은 무엇인가요? ') # input() 함수로 변수 season에 값을 할당한다.
print(season)                                  # 변수 season을 출력한다.

date = input('며칠에 태어났나요? ')             # input() 함수를 사용해 변수 date에 값을 할당한다.
date = int(date)                               # date의 값을 정수로 변환한 후 date에 재할당한다.
print(date)                                    # 변수 date의 자료형을 출력한다.

# 하나의 문자로 결합해서 결과를 출력한다.
print('좋아하는 계절은 ' + season + '이고, ' + str(date) + '일에 태어났습니다.')
```

[해설]

마지막 줄의 print() 함수는 문자열 리터럴과 변수들을 결합해서 문자열 리터럴 하나로 출력하기 때문에 정수 date 변수를 문자열로 변환한 후에 결합해야 한다. 그렇지 않으면 오류가 발생한다.

3-2 ex3_2.py

```
name = input('이름을 입력하세요...: ')      # input() 함수를 사용해 변수 name에 값을 할당한다.
print('안녕', name)                         # 인사말을 출력한다.

age = input('몇 살이세요? ')                # input() 함수를 사용해 변수 age에 값을 할당한다.
print('내년에 ' + str(int(age) + 1) + '살이 되시는군요.')    # 문자열을 결합해서 출력한다.
print('Bye~~~!')
```

[해설]

세 번째 줄의 변수 age는 문자열이다. input() 함수가 반환하는 값은 모두 문자열이란 것을 반드시 기억하자. 변수 age에 정수 1을 더하기 위해서는 먼저 int() 클래스 생성자를 통해 정수로 변환해야 한다. 1을 더한 결괏값을 다시 str() 클래스 생성자를 통해 문자열로 변환하여 나머지 문자열과 결합해서 출력하면 된다.

앞 마지막 세 줄을 다음 코드로 대신해도 된다. 즉, 애초에 input() 함수로 입력한 문자열을 정수로 변환한 후 age 변수에 할당하면 앞에서처럼 정수로 변환하는 과정 없이 1을 더한 후 문자열로 변환해서 나머지 문자와 결합해서 출력한다.

```
# input() 함수로 받아 정수로 변환한 후 변수 age에 값을 할당한다.
age = int(input('몇 살이세요? '))

# 문자열을 결합해서 출력한다.
print('내년에 ' + str(age + 1) + '살이 되시는군요.')
```

5-1 ex5_1.py

```
meal = 137.50
tax_rate = 8.875 / 100                    # 또는 tax_rate = 0.08875
tip = 15 / 100                            # 또는 tip = 0.15

meal = meal + (meal * tax_rate)           # 또는 meal += meal * tax
total = meal + (meal * tip)

print('{}'.format(round(total, 2)))       # 또는 print('{:.2f}'.format(total))
```

[해설]

우선 meal이란 변수에 음식의 총 가격인 $137.50를 할당한다. 주의할 점은 달러 부호($) 없이 숫자 137.50만 할당해야 한다는 점이다. 두 번째 줄 세율(tax_rate)은 백분율(percentage)로 표기했기 때문에 8.875%를 100으로 나눠야 원하는 결과인 0.08875를 구할 수 있다. tax_rate = 0.08875로 해도 무방하다. 봉사료도 백분율로 되어 있으니 100으로 나누거나, tip = 0.15로 해서 변수에 할당하면 된다.

지금부터 이 가족이 지불해야 하는 최종 음식 가격을 계산해보기로 하자. 세금은 음식 총액에 일정 비율을 곱하기 때문에 '음식 가격 × 세율'로 계산할 수 있다. 세금은 음식 가격에 포함되기 때문에 계산한 세금을 음식 가격에 합산하면 된다. 앞서 기존 변수에 새로운 값을 재할당할 수 있다는 것을 배웠다. 따라서 세금을 포함한 음식 가격은 meal = meal + (meal * tax_rate)과 같은 공식으로 계산할 수 있다.

이렇게 계산한 음식 가격에 봉사료 15%만 추가하면 저녁 식삿값으로 지불해야 할 총액을 구할 수 있다. 봉사료는 세금을 포함한 음식 가격의 15%이기 때문에 '세금 포함 음식 가격 × 봉사료 15%'를 적용하면 된다. 이미 봉사료 15%를 tip이란 변수에 할당했기 때문에 meal * tip처럼 계산할 수 있다.

이렇게 계산한 봉사료를 세금을 포함한 음식 가격과 합하면 이 가족이 지불해야 하는 최종 음식 가격을 산출할 수 있다. 공식은 total = meal + (meal * tip)과 같다.

그렇다면 정답 코드의 맨 마지막 줄인 `print('{}'.format(round(total, 2))`는 무엇을 뜻하는 것일까? `format()` 메소드를 통해 출력할 숫자를 `round()` 함수로 소수점 두 자리(소수점 두 자리 아래부터 반올림)까지만 문자열 형태로 보여주는 것이다.

이보다 더 나은 방식은 `print('{:.2f}'.format(total))`처럼 대체 필드 중괄호 안에 문자열 서식 설정을 하는 것이다. 중괄호 안에 있는 `{:.2f}`를 사용하면 `format()` 메소드를 통해 출력할 숫자를 소수점 두 자리(소수점 두 자리 아래부터 반올림)까지만 문자열 형태로 보여준다. `format()`은 결괏값을 문자열로 출력할 때 다양한 형식으로 출력 포맷을 설정할 수 있는 매우 유용한 메소드다.

5-2 ex5_2.py

최소한 세 가지 방법이 있을 것이다.

1. 삼중 따옴표를 사용해서 문자열 리터럴을 생성하는 방법이다.

```
print('''President Barack Obama said,
 "Don't just play on your phone,
\tprogram it."''')
```

[해설]

삼중 따옴표는 작은따옴표 3개(''' … ''') 또는 큰따옴표 3개(""" … """)로 표현할 수 있다. 이를 이용하면 따옴표 안에 따옴표를 넣어야 할 때 일일이 이스케이프 문자를 사용하지 않고 작은따옴표나 큰따옴표를 그대로 사용할 수 있다. 뿐만 아니라 여러 줄로 된 문자열 리터럴도 새줄 바꿈(\n)을 하지 않고 그대로 표현할 수 있다.

2. 작은따옴표를 사용해서 문자열 리터럴을 생성하는 방법이다.

```
print('President Barack Obama said,\n "Don\'t just play on your phone,\n\tprogram it."')
```

[해설]

작은따옴표로 문자열을 생성할 때 문자열 안에 있는 큰따옴표는 그대로 사용할 수 있지만, 작은따옴표는 이스케이프 문자를 사용해서 특수 문자가 아닌 일반 문자 작은따옴표로 처리하도록 한다.

3. 큰따옴표를 사용해서 문자열 리터럴을 생성하는 방법이다.

```
print("President Barack Obama said,\n \"Don't just play on your phone,\n\tprogram it.\"")
```

[해설]

마찬가지로 큰따옴표로 문자열을 생성할 때 문자열 안에 있는 작은따옴표는 그대로 사용할 수 있지만, 큰따옴표는 이스케이프 문자를 사용해서 특수 문자가 아닌 일반 문자 큰따옴표로 처리하도록 한다.

5-3 ex5_3.py

```
first = ''                                          # 암호화할 문자열의 처음에 오는 부분 문자열
second = ''                                         # 암호화할 문자열의 중간에 오는 부분 문자열
third = ''                                          # 암호화할 문자열의 마지막에 오는 부분 문자열

s = input('암호화하려는 문자열을 입력하세요 : ')

first = s[1::3]                     # 두 번째 문자부터 세 구간씩 이동하면서 추출한 문자열을 반환한다.
second = s[::3]                     # 첫 번째 문자부터 세 구간씩 이동하면서 추출한 문자열을 반환한다.
third = s[2::3]                     # 세 번째 문자부터 세 구간씩 이동하면서 추출한 문자열을 반환한다.

encrypted = first + second + third                  # 추출한 부분 문자열을 결합한다.
print(encrypted)                                    # 암호화한 문자열을 출력한다.
```

[해설]

문자열 분할 연산자를 사용하면 이 문제는 간단히 해결할 수 있다. 우선 입력한 문자열을 삼등분해서 담을 변수들을 초기화한다. first는 암호화할 문자열의 맨 앞에 위치할 문자들을 담는 변수다. second는 암호화할 문자열의 중간에 위치할 문자들을, third는 암호화할 문자열의 마지막에 올 문자들을 담는다.

즉, 입력한 문자열을 각각 분리 및 추출해서 앞서 설명한 변수로 담는다. 분할 연산자를 사용해서, 입력한 문자열 인덱스 1, 4, 7, … 순으로 세 칸씩 구간 이동(s[1::3])하면서 개별 문자를 추출한다. 그리고 암호화할 문자열의 첫 번째 부분 문자열인 first에 할당한다. 그 다음 문자열 인덱스 0, 3, 6, … 순으로 세 칸씩 구간 이동(s[0::3])하면서 개별 문자를 추출해서 두 번째 부분

문자열인 second에 할당한다. 마찬가지 방법으로 문자열 인덱스 2, 5, 8, … 순으로 세 칸씩 구간 이동(s[2::3])하면서 개별 문자를 추출해서 마지막 부분 문자열인 third에 할당한다. 이렇게 추출한 문자들의 조합을 순서대로 first, second, third로 결합하면 암호화 규칙대로 암호화한 문자열을 만든다. 마지막으로 암호화한 문자열을 출력한다.

6장

6-1 ex6_1.py

```
# 현재 책 목록
stock = ['마법천자문', '논어', '미움받을 용기', '자존감 수업', '채식주의자']
print('기존 책 목록:', stock)                    # 기존 책을 출력한다.

stock.remove('마법천자문')                       # 오래된 책을 삭제한다.
stock.remove('논어')                             # 오래된 책을 삭제한다.

new_books = ['골든아워', '바로 쓰는 파이썬']
stock.extend(new_books)                          # 새로운 책을 추가한다(stock += new_books와 같다).

stock.sort()                                     # 책을 '가나다' 순서로 정렬한다.

print('정리된 책 목록:', stock)                  # 정렬된 책을 출력한다.
print('남아있는 책 권수:', len(stock))           # 전체 책 권수를 출력한다.
```

[해설]

리스트 관련 메소드들을 사용하면 문제를 간단히 해결할 수 있다.

우선 현재 있는 책들을 리스트 stock에 저장하고, 현재 보유한 책의 리스트인 stock을 출력한다. 그리고 remove() 메소드를 이용해서 오래된 책들을 삭제한다. 오래된 책들을 정리한 후, 새로운 책들을 다시 stock 리스트에 추가할 때는 리스트에 객체를 추가하는 여러 가지 메소드 중 가장 흔히 사용하는 append() 메소드를 이용해서 다음처럼 책을 하나씩 추가할 수 있다.

```
new_books.append('골든아워')
new_books.append('바로 쓰는 파이썬')
```

하지만 새로운 책을 하나씩 추가하는 것이 오히려 코드를 복잡하게 만들 수 있다. 더 간단한 방법

으로는 '골든아워'와 '바로 쓰는 파이썬'이 있는 새로운 리스트를 만들어 new_books에 할당한 후, extend() 메소드를 사용해서 stock.extend(new_books)처럼 new_books의 모든 객체를 리스트 끝에 추가하는 방법이 있다.

이와 비슷한 방법으로 확장 연산자인 +=를 사용할 수도 있다. stock += new_books처럼 +=를 사용해 stock 리스트에 new_books 리스트를 그대로 더하면 된다.

책을 넣고 빼는 과정이 끝났다면, sort() 함수를 이용하여 new_books 리스트를 정렬한다. 그리고 print() 함수를 이용해 정렬한 리스트를 출력한다. 마지막으로 len() 함수를 사용해서 전체 책이 몇 권인지 출력한다.

6-2 ex6_2.py

```
# 튜플로 정의한다.
basket = ('바지', '치마', '니트', '책', '니트', '치마', '바지', '바지', '니트',
          '치마', '책', '필통')

basket_list = list(basket)                    # 내용을 수정하기 위해 튜플 bastket을 리스트로 변환한다.
del basket_list[-1]                           # 마지막 목록을 제거한다.

in_basket = input('어떤 물품을 확인하시겠습니까(있으면 True, 없으면 False)? ')
print(in_basket in basket_list)               # 불린으로 물품이 목록에 있는지 확인한다.
```

[해설]

우선 장바구니 목록을 튜플로 정의한 후 변수 basket에 할당한다. 튜플은 불변자료형이라 한 번 정하면 지우거나 변경할 수 없다는 것에 주의해야 한다. 따라서 마지막에 추가한 물품을 제거하기 위해서는 튜플을 리스트로 변환해야 한다. 리스트로 바꾼 것을 basket_list라는 새로운 변수에 할당한다. 그리고 마지막 물품의 인덱스가 -1인 것을 활용하여 삭제 연산자 del을 활용해서 제거한다.

여기서 pop() 메소드를 사용할 수 있는지 궁금할 것이다. pop() 메소드는 리스트의 맨 마지막 객체를 반환한 후 그 객체를 삭제하는 메소드다. 따라서 객체를 반환받을 필요가 없는 경우에는 굳이 pop() 메소드를 사용할 이유가 없다.

또한 '필통'이 한 개뿐이라 remove() 메소드를 사용할 수 있지만, 두 개 이상의 같은 물품이 있으면 먼저 나오는 물품부터 제거하기 때문에 원하지 않는 결과가 발생할 수 있다. 따라서 습관적으로 더 정확한 프로그램 구현을 위해, 즉 목록 리스트에서 삭제하고자 하는 목록을 정확히 삭제하기 위해, pop()이나 remove() 메소드를 사용하지 않고 del 연산자를 사용하도록 하자.

장바구니에서 맨 마지막 객체를 제거한 후 물품이 있는지 여부를 묻기 위해 input() 함수를 활용한다. 묻고자 하는 물품을 변수 in_basket에 할당하고 함수 멤버십 연산자인 in을 활용하여 묻고자 하는 물품이 basket_list에 들어있는지 확인한다. 결괏값은 불린으로 나올 것이다.

6-3 ex6_3.py

```
# 물품을 딕셔너리로 정의한다.
products = {'수박': 15000, '콩나물': 850, '초콜릿': 1500, '생선': 9000, '파리채': 1000}

del products['수박']                                          # 이전 물품을 제거한다.
del products['파리채']                                        # 이전 물품을 제거한다.

products['감'] = 800                                          # 신상품을 추가한다.
products['전기장판'] = 20000                                  # 신상품을 추가한다.

print('바뀐 물품 내역 :', products)                           # 바뀐 물품 내역을 고시한다.

purchase = input('어떤 물품을 구매하실 예정인가요? ')          # 구매할 물품을 입력받는다.
print('{}은 {:,}원입니다'.format(purchase, products[purchase]))  # 물품의 가격을 알려준다.
```

[해설]

이 문제는 딕셔너리 관련 연산자만으로도 쉽게 해결할 수 있다. 우선, 이전 물품의 딕셔너리를 변수 products에 정의한다. 계절이 바뀌면서 '수박'과 '파리채'는 더 이상 판매하지 않을 것이기 때문에 삭제 연산자 del을 이용해 삭제한다. 그리고 '감'과 '전기장판'이라는 새로운 물품을 추가하기 위해 딕셔너리 products에 키와 매핑값으로 물품 이름과 가격을 추가한다. print() 함수를 사용해 바뀐 물품 내역을 출력한다.

고객에게 구매할 물품을 묻기 위해 input() 함수를 사용한다. 입력한 물품을 purchase라는 변수에 저장한다. 가격을 알려주기 위해 products 딕셔너리에서 purchase에 해당하는 매핑값을 찾고, format() 메소드를 이용해서 쉼표로 숫자를 세 자리 단위로 구분({:,})해서 출력한다.

8-1 ex8_1.py

```
city = input('도시 이름을 입력하세요: ')

if city == '서울':                                  # city 값이 '서울'이면
    size = 605                                      # 변수 size에 605를 입력한다.
elif city == '파리':                                # city 값이 '파리'면
    size = 105                                      # 변수 size에 105를 입력한다.
elif city == '에든버러':                            # city 값이 '에딘버러'면
    size = 264                                      # 변수 size에 264를 입력한다.
else:                                               # 그 이외에는
    size = 'Unknown'                                # 변수 size에 'Unknown'을 입력한다.

print('{}의 면적은 {} 평방 킬로미터입니다.'.format(city, size))
```

[해설]

이 문제는 도시를 계속 추가한다는 점을 고려한다면 도시 이름을 키로, 면적을 매핑값으로 하는 딕셔너리로 구현하는 것이 좋다. 하지만 여기서는 간단하게 조건문을 사용해서 프로그램을 작성하기로 한다. input() 함수를 활용해 도시 이름을 변수 city에 저장한 후, 조건문인 if-eiif-else에서 city 변숫값이 표에 있는 세 도시에 해당하면 그 도시의 면적을 변수 size에 할당한다. 만약 표에 있는 세 도시가 아니면 변수 size에 'Unknown'을 할당한다. 그리고 도시 면적을 알려주기 위해 format() 메소드를 이용해서 결과를 출력한다.

8-2 ex8_2.py

```
# 판매 총액을 입력받아 변수 sales에 할당한다.
sales = int(input('회사의 판매 총액은 얼마입니까? '))

# 관리 비용을 입력받아 변수 expense에 할당한다.
expense = int(input('회사의 관리 비용은 얼마입니까? '))

operating_income = sales - expense                # 영업 이익을 계산한다(판매 총액 - 관리 비용).

if operating_income >= 0:                         # 영업 이익이 적자가 아니면 영업 이익을 출력한다.
    print('영업 이익은 {:,}원입니다.'.format(operating_income))
else:                                             # 영업 이익이 적자면 괄호로 묶어 출력한다.
    print('영업 이익은 ({:,})원입니다.'.format(abs(operating_income)))
```

[해설]

input() 함수를 통해 사용자로부터 총 판매 금액과 관리 비용을 입력받는다. 영업 이익은 변수 sales에, 관리 비용은 변수 expense에 각각 할당한다. 이때 입력한 값을 정수로 형변환을 한다. 영업 이익의 계산 공식은 다음과 같다.

영업 이익 = 총 판매 금액 – 관리 비용

따라서 입력한 변수 sales에서 expense 변숫값을 빼면 영업 이익은 간단히 계산할 수 있다. 이렇게 계산한 영업 이익은 변수 operating_income에 할당한다.

마지막으로 처리한 결과를 화면으로 보여줘야 한다. operating_income이 음수가 아니면(즉, 적자가 아니면) format() 메소드를 이용해서 쉼표로 숫자를 세 자리 단위로 구분({:,})해서 출력한다. operating_income이 0보다 작아 적자가 되면 마이너스 부호 대신 괄호로 표기해야 하기 때문에 abs() 함수를 통해 음수를 양수로 전환한 후 format() 메소드를 이용해서 대체 필드 양쪽 끝을 괄호로 묶어(({:,})) 출력한다.

8-3 ex8_3.py

```
import math                                          # math 모듈을 불러온다.
i = 0                                                # 변수 i를 정수로 선언하고 초깃값을 0으로 설정한다.
while i < 10:                                                # i 값을 0부터 9까지 순회한다.
    print('{}! = {}'.format(i, math.factorial(i)))           # i의 계승 값을 계산해 출력한다.
    i += 1                                           # i에 1을 더한다.
else:                                                # while문이 정상적으로 종료되면
    print('종료')                                     # '종료'를 출력하고 while문을 빠져나간다.
```

[해설]

계승 함수 factorial()을 사용하려면 먼저 math 모듈을 불러와야 한다. math 모듈을 불러온 후 math.factorial() 형식으로 사용하면 된다. 모듈에 관해서는 나중에 자세히 다룬다.

while문이 0부터 9까지 차례로 순회하면서 변수 i에 0, 1, 2, ..., 9가 할당된다. while문 바로 아래 코드에서 변수 i를 math.factorial(i)의 전달인자로 받아 i의 계승 값을 계산한 후 print() 함수에서 format() 메소드를 사용해 'n! = 계승 값' 형식으로 출력한다. while문이 정상적으로 종료된 후 else문을 사용해서 프로그램이 종료되었음을 알린다.

참고로 while문을 사용하여 0부터 9까지 각 정수까지 처리하는 코드를 구현하려면, 다음과 같은 while문 형식을 관용구처럼 사용하면 된다.

```
i = 0                                                # 센티널 값을 0으로 초기화한다.
while i < 10:                                        # 센티널 값을 0부터 9까지 순회한다.
    # ...                                            # 여기에 처리할 코드를 입력한다.
    i += 1                                           # 센티널 값을 1 증가시킨다.
else:
    print('종료')                                     # while문이 정상적으로 종료되면 else문이 실행된다.
```

8-4 ex8_4.py

```
import math                                                    # math 모듈을 불러온다.
for i in range(10):                          # for문과 range() 클래스를 이용해 0부터 9까지의 정수를 생성한다.
    print('{}! = {}'.format(i, math.factorial(i)))             # i의 계승 값을 계산해 출력한다.
else:                                                          # for문이 정상적으로 종료되면
    print('종료')                                               # '종료'를 출력하고 for문을 빠져나간다.
```

[해설]

계승 함수 factorial()을 사용하기 위해 math 모듈을 불러온다. for문에서 range() 클래스로 0부터 9까지 정수를 생성해서 0, 1, 2, ..., 9 순으로 순회할 때마다 각 정수를 1씩 증가시키면서 변수 i에 할당한다. for문 바로 다음 코드에서 변수 i를 math.factorial(i)의 전달인자로 받아 i의 계승 값을 계산한 후 print() 함수에서 format() 메소드를 사용해 'n! = 계승 값' 형식으로 출력한다. for문이 정상적으로 종료된 후 else문을 사용해서 프로그램이 종료되었음을 알린다.

참고로 for문을 사용하여 0부터 9까지 각 정수의 계승 값을 구하려면, 다음과 같은 for문 형식을 관용구처럼 사용하면 된다.

```
for i in range(10):                                    # 0부터 9까지 10회 순회한다.
    # ...                                              # 여기에 처리할 코드를 입력한다.
else:
    print('종료')                                       # for문이 정상적으로 종료되면 else문이 실행된다.
```

8-5 ex8_5.py

```
# 과일 목록을 리스트로 만들어 변수 fruits에 할당한다.
fruits = ['사과', '딸기', '블루베리', '바나나', '포도']

# 검색할 과일을 입력받아서 변수 target에 할당한다.
target = input('과일 이름을 입력하세요...: ')
index = 0                                   # 리스트의 인덱스로 사용할 센티널 값을 초기화해서 i에 할당한다.
while index < len(fruits):                  # 리스트의 인덱스 0부터 n - 1까지 전체 리스트를 순회한다.
    if fruits[index] == target:             # 검색하는 과일이 리스트에 있다면
        # 리스트에서의 위치를 출력한 후 while문을 빠져나간다.
        print('{}는 과일 목록의 {}번째에 존재합니다.'.format(target, index + 1))
        break
    index += 1                              # 인덱스를 1 증가시켜 다음 객체와 비교할 수 있도록 한다.
else:                                       # 리스트를 모두 순회한 후 검색하는 과일이 없다면
                                            # 리스트에 없다고 출력하고 while문을 종료한다.
    print('{}는 과일 목록에 존재하지 않습니다.'.format(target))
```

[해설]

찾고자 하는 객체가 리스트에 있는지 확인하려면 순환문을 사용해서 리스트의 첫 번째 객체부터 마지막까지 하나씩 비교하면 된다. 검색 대상이 있으면 검색 대상의 위치를 알려준 후 검색을 중단하고 순환문을 빠져나오고, 마지막까지 비교해도 검색 대상이 없다면 해당 객체가 없다고 알려주고 순환문을 종료하면 된다.

프로그램을 작성하기 위해 우선 검색 대상 과일을 input() 함수로 받아서 변수 target에 할당한다. 그리고 과일 목록에 해당하는 리스트를 생성해서 변수 fruits에 할당한다. while문으로 리스트를 순회할 때는 현재 위치를 파악하기 위해 리스트의 인덱스, 여기서는 변수 index를 센티널 값으로 활용하면 된다. 리스트의 인덱스 번호가 0부터 시작하기 때문에 변수 index를 0으로 초기화한다. 리스트의 모든 객체를 순회해야 하기 때문에 len() 함수로 리스트가 포함한 전체 객체 수, 즉 리스트의 길이를 확인하고, index 변숫값이 리스트의 길이보다 하나 적을 때까지 순회한다. 리스트에 포함된 객체 수가 n일 때 인덱스 번호는 0부터 n - 1라는 사실을 늘 기억하자.

fruits의 첫 번째 객체(인덱스 번호 0)를 target과 비교해서 일치한다면 format() 메소드를 통해 찾고자하는 객체가 첫 번째에 있다고 알려주고 break문을 통해 순환문을 빠져나간다. index에 1을 더하는(index += 1) 이유는 index가 0부터 시작하기 때문이다. 만약 첫 번째 객체가 target과 다르면 index를 1 증가시켜 fruits의 두 번째 객체와 비교를 한다.

이런 순서로 fruits의 객체를 차례대로 비교해서 해당 객체가 target과 일치하면 조건문이 '참(True)'이므로 그 객체의 위치를 출력한 후 break문을 통해 순환문을 빠져나간다. 리스트의 마지막까지 확인해도 target과 일치하는 객체를 찾지 못하면 순환문이 종료된다. 이때 순환문은 break문을 통하거나 예외가 발생하지 않고 정상적으로 종료되기 때문에 else문은 반드시 실행된다. 따라서 else문을 통해 target이 과일 목록에 존재하지 않는다는 메시지를 출력한 후 프로그램을 종료한다.

8-6 ex8_6.py

```
# 과일 목록을 리스트로 만들어 변수 fruits에 할당한다.
fruits = ['사과', '딸기', '블루베리', '바나나', '포도']

target = input('과일 이름을 입력하세요...: ')    # 검색할 과일을 입력받아서 변수 target에 할당한다.

for index, fruit in enumerate(fruits, start=1): # 시작 번호를 1로 설정해서 리스트 전체를 순회한다.
    if fruit == target:                         # 검색하는 과일이 리스트에 있다면
        # 리스트에서의 위치를 출력한 후 for문을 빠져 나간다.
        print('과일 목록의 {}번째에 존재합니다.'.format(index))
        break
else:                                           # 리스트를 모두 순회한 후 검색하는 과일이 없다면
    print('과일 목록에 존재하지 않습니다.')       # 리스트에 없다고 출력하고 for문을 종료한다.
```

[해설]

for문을 사용하면 while문보다 코드를 좀 더 간단히 작성할 수 있다. for문에서 enumerate() 함수를 활용하면 while문처럼 리스트의 인덱스를 추적할 필요가 없다. 과일 목록 리스트의 객체를 차례로 처리할 때 순번을 1부터 시작하면 된다. enumerate() 함수를 활용하여 fruits 안의 모든 객체를 순번 1부터 시작해서 차례대로 변수 index에 할당한다.

만약 fruit의 값이 target의 값과 같다면 리스트의 몇 번째에 있는지 위치를 알려준 후, break문을 통해 루프를 빠져나간다. fruits의 모든 객체가 target의 값과 일치하지 않는다면 순환문은 정상적으로 종료된다. 따라서 마지막으로 else문이 실행되면서 '과일 목록에 존재하지 않습니다.'라는 메시지를 출력하고 프로그램을 종료한다.

8-7 ex8_7.py

```
import random                                  # random 모듈을 불러온다.
i = 0                                          # 변수 i를 정수로 선언하고 초깃값을 0으로 설정한다.
while True:                                    # 무한 루프로 순회한다.
    i = random.randint(1, 9)                   # 1~9 사이의 임의의 정수를 생성해서 i에 할당한다.
    if i <= 5:                                 # i가 5 이하면
        print(i)                               # i를 출력하고
        continue                               # 다시 임의의 정수를 생성하기 위해 while문의 처음으로 돌아간다.
    print('{}: 무한 루프에서 빠져나갑니다.'.format(i))    # i가 5 이하가 아니면 i를 출력하고
    break                                      # 무한 루프를 종료한다.
```

[해설]

randint() 함수를 사용하기 위해서는 먼저 import문으로 random 모듈을 불러온다. 무한 루프인 while True: 안에서 randint() 함수를 사용해서 1부터 9 사이의 임의의 정수를 생성해서 변수 i에 할당한다.

조건문인 if문에서 i의 값을 확인하여 i가 1에서 5 사이의 숫자이면 i를 출력한 후 다시 while문의 처음으로 돌아간다. 다시 1~9 사이의 임의의 정수를 생성해서 변수 i에 할당한 후 i가 1에서 5 사이의 숫자인지 확인하는 작업을 반복한다.

만약 i의 값이 1~5가 아니면 if문이 실행되지 않고 if문 다음에 있는 print() 함수가 실행된다. print() 함수를 통해 randint() 함수가 생성한 숫자와 함께 무한 루프를 빠져나간다는 메시지를 출력한 후 break문을 실행해서 무한 루프를 빠져나간다. 이때 break문이 없으면 무한 루프가 계속되니 주의해야 한다.

9-1 ex9_1.py

```
# 과일 목록을 리스트로 만들어 변수 fruits에 할당한다.
fruits = ['사과', '딸기', '블루베리', '바나나', '포도']

# 검색할 과일을 입력받아서 변수 target에 할당한다.
target = input('과일 이름을 입력하세요...: ')

try:                                    # target이 fruits 안에 있는지 확인한다.
    index = fruits.index(target)
except ValueError:                      # target이 fruits 안에 없으면 예외를 발생시킨다.
    print('과일 목록에 존재하지 않습니다.')
else:                                   # target이 fruits 안에 있다면 인덱스 번호에 1을 더해서 출력한다.
    print('과일 목록의 {}번째에 존재합니다.'.format(index + 1))
```

[해설]

예외 처리를 사용해서 리스트의 내용을 검색할 경우, 검색 대상이 리스트에 없을 때 try문에서 예외를 발생시키면 된다. try문 안에서 리스트 메소드인 index()는 리스트에서 특정 객체가 있는지 검색한다. 검색하는 객체가 리스트에 없으면 ValueError를 발생시킨다. 검색하는 객체가 리스트에 있으면 처음 발견한 해당 객체의 인덱스를 반환한다.

try문에서 index() 메소드를 통해 target의 값이 과일 목록의 리스트인 fruits에 존재하는지를 확인한다. target의 값이 fruits에 없다면 ValueError가 발생할 것이다. 예외가 발생하면 except문에서 ValueError를 받아서 print() 함수를 통해 '과일 목록에 존재하지 않습니다.'라는 메시지를 출력하고 프로그램을 종료한다.

try문에서 예외가 발생하지 않는다면 target의 값이 fruits에 존재한다는 뜻이기 때문에, else문을 사용해서 target이 리스트의 몇 번째에 있는지 위치를 알려준다. 이는 try문 안의 코드가 예외가 발생하지 않고 정상적으로 종료하는 경우 else문이 반드시 실행된다는 점을 이용한 것이다.

10-1 ex10_1.py

```
def count_vowels(text):
    """영어 단어 또는 문장에 들어 있는 모음 개수를 반환하는 함수다.
    text.....: 영어 텍스트(문자열)
    Returns: 모음 개수(정수)
    """
    # 영어 모음(대소문자를 구분하지 않기 때문에 소문자로 이루어진 튜플)
    vowels = 'a', 'e', 'i', 'o', 'u'

    count = 0                                   # 영어 모음 개수를 세는 변수를 초기화한다.

    for char in text:
        if char.lower() in vowels:              # 추출한 문자를 소문자로 바꾼 후 영어 모음 중 하나면
            count += 1                          # count를 1 증가시킨다.

    return count                                # 영어 모음 개수를 반환한다.

# ----- 인터프리터 모드에서만 실행됨 ------------------------------------------------- #
if __name__ == '__main__':
    test_data = 'Apples', 'I love Python'       # 테스트용 데이터를 튜플로 만든다.
    for i in test_data:
        result = count_vowels(i)
        print('테스트 데이터:', i)
        print('결과.....:', result)
        print()
```

[해설]

count_vowels()는 전달인자로 입력된 영어 단어 또는 문장 안에 속한 모음 개수를 출력하는 함수다. 먼저 모음으로 이루어진 튜플을 생성하는데, 대소문자를 구분하지 않기 때문에 소문자로 이루어진 튜플 vowels를 생성한다.

생성 후 모음 개수를 세는 변수 count를 만들고 이를 0으로 초기화한다. 이제 입력한 단어 또는 문장에 모음이 있는지 판단하기 위해 for문을 사용하여 단어나 문장의 문자를 차례로 하나씩 검사한다. 대소문자의 구분이 없도록 문자열 메소드인 lower()를 사용해 모든 문자를 소문자로 변환한다. 만약 문자가 튜플 vowels에 있다면 count에 1을 증가시킨다. 검사가 끝나면 count의 값을 반환한다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행되도록 조건문인 if __name__ == '__main__': 안에 작성했다. 인터프리터 모드에서 프로그램을 실행하면 __name__의 값이 '__main__'으로 설정되기 때문에, if문의 조건을 만족해서 이 코드가 실행된다. 자세한 내용은 본문 414p를 참고하면 된다.

10-2 ex10_2.py

```
def no_number(seq):
    """숫자가 아닌 자료형을 반환하는 함수다.
    seq.....: 리스트나 튜플
    Returns: 숫자가 아닌 자료형을 담은 튜플
    """
    nonumeric = []                                  # 숫자가 아닌 자료형을 담을 리스트를 초기화한다.

    for i in seq:
        if type(i) == int or type(i) == float:                  # 정수나 실수면
            continue                                            # 건너뛴다.
        else:                                                   # 정수나 실수가 아니면
            nonumeric.append(i)                                 # 리스트에 추가한다.

    return tuple(nonumeric)                                     # 튜플로 변환해서 반환한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    test_data = [[1], {1}, 1, '1', 'one', -3.14, {'1': 1}, 0, '2.58', (5, 9.9)]
    result = no_number(test_data)
    print('테스트 데이터:', test_data)
    print('결과........:', result)
```

[해설]

no_number()는 전달인자로 입력된 리스트나 튜플에 속한 객체 중 정수나 실수가 아닌 자료형을 따로 추출해서 튜플로 반환하는 함수다. 먼저 숫자가 아닌 자료형을 담기 위해 빈 리스트를 만들어서 변수 nonumeric에 할당한다. 결괏값으로 반환하는 튜플은 불변자료형이기 때문에 숫자가 아닌 객체들을 선택해서 추가할 수 있는 가변자료형이 우선 필요하다. 따라서 가장 적절한 자료형인 리스트를 사용해서 필요한 객체들을 모두 추가한 후 이 리스트를 튜플로 변환해서 반환하면 된다.

for문을 활용해서 전달인자의 모든 객체를 순회하면서 개별 객체의 자료형이 정수나 또는 실수인지 확인한다. 정수 또는 실수면 따로 처리할 필요가 없기 때문에 continue문을 사용해서 순환문의 처음으로 돌아가 다음 객체를 추출해서 비교한다. 정수 또는 실수가 아닌 객체만 nonumeric 리스트에 추가한다. 객체를 모두 비교한 후 for문을 빠져나온 다음, nonumeric이 리스트이기 때문에 최종적으로 튜플로 변환해서 그 결과를 반환한다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

10-3 ex10_3.py

```
def switch_somecase(text):
    """
    특정 영어 문자(A, B, C, D)의 대문자를 소문자로, 소문자를 대문자로 바꾼 새로운 문자열을 반환하는 함수다.
    text.....: 알파벳 문자열
    Returns: 특정 대소문자를 반전한 알파벳 문자열
    """
    switch_case = set('ABCDabcd')              # 대소문자를 반전할 문자들을 세트로 선언한다.
    switched_text = ''                         # 대소문자를 반전한 문자열을 담을 변수를 초기화한다.

    for char in text:
        if char in switch_case:                # 문자가 'A', 'B', 'C', 'D', 'a', 'b', 'c', 'd' 중 하나면
            # 문자 char가 소문자면 대문자로, 대문자면 소문자로 바꾸어 switched_text에 결합해 재할당한다.
            switched_text += char.swapcase()
        else:                                  # 그 외 문자면
            switched_text += char              # switched_text에 그대로 결합해 재할당한다.

    return switched_text                       # A/a, B/b, C/c, D/d를 반전한 문자열을 반환한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    test_string = 'I am Python. I am not anaconda!!!'
    result = switch_somecase('I am Python. I am not anaconda!!!')
    print('테스트 데이터:', test_string)
    print('결과........:', result)
```

[해설]

먼저 대문자를 소문자로, 소문자를 대문자로 반전할 문자들을 세트로 만든다. 세트에 속한 단어들은 전달인자로 받은 영어 문자열의 단어들과 하나씩 비교할 때 사용된다. 세트로 만들면 다음처럼 각 단어가 순회할 수 있는 문자열로 구성된다.

```
>>> switch_case = set('ABCDabcd')
>>> switch_case                                    # 세트라 순서가 없다.
{'a', 'C', 'B', 'c', 'D', 'A', 'b', 'd'}
```

그리고 대소문자를 반전한 문자열을 담을 변수 switched_text를 빈 문자열로 초기화한다. 이 변수는 특정 문자열의 대소문자를 모두 바꾼 후 그 결과를 반환할 때 사용한다. for문에서 멤버십 연산자인 in을 사용해 전달받은 영어 문자열 text의 문자를 하나씩 변수 char에 할당한 후

이 문자를 차례대로 switch_case의 특정 문자('A', 'B', 'C', 'D', 'a', 'b', 'c', 'd')에 해당하는지 비교한다.

char의 문자가 이들 문자 중 하나면 문자열 메소드인 swapcase()를 사용해서 대문자를 소문자로, 소문자를 대문자로 바꾼 후, switched_text에 추가한다. char의 문자가 switch_case의 특정 문자에 해당하지 않으면 대소문자 변환없이 그대로 switched_text에 추가한다. 이때 문자열은 불변자료형이기 때문에 리스트처럼 직접 문자를 추가할 수는 없다. 따라서 결합한 문자열을 변수에 재할당해야 한다. 증강 할당 연산자 +=를 사용해서 기존 변수에 재할당하면 가장 편리하다. 이렇게 모든 문자를 처리한 후 반전한 결과를 문자열로 반환한다.

리스트를 사용해서 문자들을 추가하고 그 결과를 다시 문자열로 바꾸어 반환하는 프로그램을 작성할 수도 있다. 이 경우에 변수 switched_text를 다음처럼 빈 리스트로 선언해서 초기화한 후 처리된 문자 하나하나를 객체로 추가하고 join() 메소드를 활용해 리스트의 모든 문자 객체를 합해서 반환하면 된다.

```
switch_case = set('ABCDabcd')                    # 대소문자를 반전할 문자들을 세트로 선언한다.
switched_text = []                               # 대소문자를 반전한 문자들을 담을 리스트를 초기화한다.

for char in text:
    if char in switch_case:                      # 문자가 'A', 'B', 'C', 'D', 'a', 'b', 'c', 'd' 중 하나면
        # 문자 char가 소문자면 대문자로, 대문자면 소문자로 바꾸어 switched_text에 추가한다.
        switched_text.append(char.swapcase())
    else:                                        # 그 외 문자면
        switched_text.append(char)               # switched_text에 그대로 추가한다.

return ''.join(switched_text)    # switched_text에 있는 모든 문자열 객체를 빈칸으로 결합해서 반환한다.
```

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

10-4 ex10_4.py

```
def search_index(seq, target):
    """시퀀스형 객체 안의 특정 객체(항목)의 인덱스를 반환하는 함수다.
    seq.....: 검색 목록(시퀀스형)
    target...: 찾고자 하는 객체
    Returns: 시퀀스형의 인덱스 번호(만약 특정 객체가 존재하지 않으면 -1을 반환)
    """
    if target in seq:                                    # seq 안에 target이 있으면
        return seq.index(target)                         # target의 인덱스 번호를 반환한다.
    return -1                                            # seq 안에 target이 없으면 -1을 반환한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    test_data = (                                # 테스트용 데이터를 (seq, target) 형식의 튜플로 만든다.
        ('I love Python', 'p'),
        (['기타', '베이스', '키보드', '드럼'], '드럼'),
        ([1, 2, 3], 1)
    )
    for seq, target in test_data:
        result = search_index(seq, target)
        print('테스트 데이터:', seq)
        print('타깃.....:', target)
        print('결과.....:', result)
        print()
```

[해설]

search_index()는 검색 목록인 시퀀스형 객체(seq)와 찾고자 하는 객체(target)를 입력받아 시퀀스형 안에 그 객체가 있으면 찾은 객체의 인덱스 번호를 알려주는 함수다. 우선 if문을 활용해 시퀀스형 안에 찾으려는 객체가 있는지 판별한다. 만약 찾으려는 객체가 시퀀스형 안에 있으면, index() 메소드를 사용하여 그 인덱스 값을 반환한다. 만약 없다면 -1을 반환한다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

10-5 ex10_5.py

```
pair = [
    (2, 'apple'), (1, 'Banana'), (2, 'Banana'), (1, 'grape'),
    (3, 'apple'), (3, 'Orange'), (2, 'Pineapple')
]

# 튜플의 두 번째 객체를 대소문자 구분 없이 오름차순 정렬한 후, 첫 번째 객체를 오름차순으로 정렬한다.
sorted(pair, key=lambda e: (e[1].lower(), e[0]))

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    print('테스트 데이터:', pair)
    print('결과........:', sorted(pair, key=lambda e: (e[1].lower(), e[0])))
```

[해설]

우선 정렬 대상인 리스트 pair를 생성한다. 문제의 조건은 리스트 안 튜플의 두 번째 객체를 대소문자 구분 없이 먼저 정렬하는 것이다. 따라서 sorted() 함수의 key에 전달인자를 설정해야 한다. 여기서 람다 함수를 사용할 것이다. 이때 key란 정렬할 때의 기준이다.

key에 함수가 설정되면 정렬해야 할 리스트의 객체들을 하나씩 key에 설정한 함수에 전달해서, 이 함수를 실행한 결과를 기준으로 정렬이 진행된다. 리스트의 객체인 튜플을 하나씩 key의 람다 함수에 전달하는데, 튜플의 두 번째 객체가 첫 번째 정렬 기준이 된다. 이때 대소문자 구분 없이 적용해야 하므로 lower() 메소드를 사용해 모든 문자를 소문자로 변환한다. 그 후 두 번째 정렬 기준으로 튜플의 첫 번째 객체인 정수를 사용한다. key값에 따라 pair 리스트를 정렬할 것이다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

10-6 ex10_6.py

```
def sort_by_word_length(*words):
    """입력한 단어 중 단어의 길이가 긴 순서부터 내림차순으로 정렬하는 함수다.
    만약 단어의 길이가 같다면 그 단어들을 다시 내림차순으로 정렬한다.
    영어 단어라면 대소문자를 구분해서 내림차순으로 정렬한다.
    words..: 단어 문자열
    Returns: 단어 길이로 내림차순 정렬한 리스트
    """
    t = []                              # (길이, 단어) 형식의 튜플을 객체로 담을 리스트를 초기화한다.
    for w in words:                     # 전달인자로 패킹해서 받은 단어들을 차례로 하나씩 꺼내서
        t.append((len(w), w))           # (길이, 단어) 형식의 튜플 객체로 리스트에 추가한다.

    # 리스트의 sort() 메소드를 사용해서 (길이, 단어) 형식의 튜플 객체들을 길이 순으로 내림차순으로 정렬한다.
    # 만약 길이가 같다면 대소문자 등을 구분해서 내림차순으로 정렬한다.
    t.sort(reverse=True)

    sorted_words = []                   # 단어 길이 순으로 내림차순 정렬한 단어를 담을 리스트를 초기화한다.
    for length, word in t:              # (길이, 단어) 리스트에서 길이와 단어를 분리해서 각각 변수에 담는다.
        sorted_words.append(word)       # 단어만 순서대로 리스트에 추가한다.

    return sorted_words                 # 단어 길이 순으로 내림차순 정렬한 단어를 담을 리스트를 반환한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    result = sort_by_word_length('피아노', '기타', '베이스', '드럼')
    print('테스트 데이터:', "'피아노', '기타', '베이스', '드럼'")
    print('결과........:', result)
    print()
    result = sort_by_word_length('red', 'blue', 'green', 'brown', 'gray')
    print('테스트 데이터:', "'red', 'blue', 'green', 'brown', 'gray'")
    print('결과........:', result)
```

[해설]

sort_by_word_length()는 전달인자로 받은 한 개 이상의 단어 중 단어의 길이가 긴 순서부터 정렬해 리스트로 반환하는 함수다. 불특정 다수의 전달인자를 입력받기 때문에 이를 패킹하는 연산자인 *를 매개변수 앞에 붙여 사용한다.

우선 (단어 길이, 단어)의 튜플 객체를 담기 위한 리스트 t를 생성한다. 패킹한 시퀀스형 words 안의 단어들에 대해 각각 단어의 길이와 단어를 쌍으로 하는 튜플을 만드는 것이다. 그리고 튜플을 리스트 t에 추가한다.

리스트의 sort() 메소드를 이용해 리스트 t를 정렬하면, 단어의 길이가 첫 번째 정렬 기준이 될 것이다. 길이가 같다면 단어가 두 번째 정렬 기준이 될 것이다. 내림차순으로 정렬하기 때문에 reverse=True로 지정한다.

그 다음 단계는 순서대로 정렬한 단어를 담기 위해 리스트 sorted_words를 생성한다. 리스트 t가 담고 있는 튜플에서 단어의 길이와 단어를 분리하기 위해 length, word로 구분하여 for문을 실행한다. 리스트 sorted_words에는 단어인 word만을 추가한다. for문의 실행이 끝나면 리스트 sorted_words를 반환한다. sorted_words는 단어 길이로 내림차순 정렬한 단어들을 담고 있는 리스트다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

10-7 ex10_7.py

```
def installment(total_price, payment_amount, interest_rate):
    """매달 특정 할부금을 납입한다는 가정 아래 물건 가격을 전부 갚기 위해 필요한 전체 기간을
    월 단위로 계산해서 출력하는 함수다.
    total_price.....: 물건 가격(정수)
    payment_amount: 매달 납입할 할부금(정수)
    interest_rate.....: 이자율(실수)
    Returns.....: None
    """
    month = 0
    while total_price > 0:
        total_price = total_price * (1 + interest_rate) - payment_amount
        month += 1
    print('{}개월'.format(month))

while True:                                  # 대화형 모드에서 ex10_7을 import하면 while문이 바로 실행된다.
    msg_err = '오류: 잘못된 숫자입니다!'                  # 잘못 입력했을 때 출력하는 오류 메시지
    # 프로그램을 계속 실행할지 묻는 메시지
    msg_continue = '계속하시겠습니까? (1) 예 (2) 아니오 => '
    try:                                                # 모든 납입은 월말에 이행한다.
        price = int(input('\n구입한 물건의 가격은 얼마입니까(단위: 원)? '))
        payment_amount = int(input('매달 할부금을 얼마씩 납입할 예정입니까(단위: 원)? '))
        interest_rate = float(input('이자율은 얼마입니까(단위: %)? ')) / 100
        if price * interest_rate < payment_amount:      # 이자가 매달 납입하는 할부금보다 적다면
            installment(price, payment_amount, interest_rate)  # 할부 기간을 계산해서 출력한다.
            quit = int(input(msg_continue))             # 프로그램을 계속 실행할지 묻는다.
            if quit == 1:                               # 계속 실행할 때
                continue
            elif quit == 2:                             # 중단할 때
                print('Bye~~~!')
                break
            else:                                  # 기타
                print(msg_err)                     # 오류 메시지를 출력하고 다시 프로그램을 실행한다.
        else:                                      # 이자가 매달 납입하는 할부금보다 크다면
            print('이자가 할부금보다 크군요. 이자보다 더 큰 할부금을 매달 납입해야 합니
다!!!')
            quit = int(input(msg_continue))             # 프로그램을 계속 실행할지 묻는다.
            if quit == 1:                               # 계속 실행할 때
                continue
```

```
            elif quit == 2:                    # 중단할 때
                print('Bye~~~!')
                break
            else:                              # 기타
                print(msg_err)                 # 오류 메시지를 출력하고 다시 프로그램을 실행한다.
    except ValueError:
        print(msg_err)
```

해설

installment()는 구입한 물건의 전체 금액을 다 갚는 데 걸리는 기간을 계산하는 함수다. 이 함수는 전체 물건 금액, 매달 납입할 할부금, 이자율이라는 세 가지 전달인자를 입력받는다. 우선 전체 금액을 다 갚는데 걸리는 기간을 나타내는 변수 month를 0으로 초기화한다. while문을 이용해 물건 가격 total_price가 0보다 크다면(즉, 갚을 할부금이 남았다면), total_price에 이자율을 적용한 값에서 납입한 할부금을 뺀 값을 다시 total_price에 할당한다. 매달 남은 할부금을 업데이트하는 것이다. 그리고 month 변수에 1을 더한다. total_price가 0보다 작아지면(즉, 할부금을 다 갚았다면) 갚는 데 걸린 전체 기간인 month을 출력한다. 출력은 format() 메소드를 이용한다.

이제 샘플 데이터로 테스트하는 프로그램을 작성할 것이다. while문을 사용해 무한 루프를 실행한다. 잘못 입력했을 때 출력하는 msg_err와 프로그램을 계속 실행할지 묻는 msg_continue 변수를 지정한다.

예외 처리를 위해 try-except-else문을 이용한다. 우선 installment() 함수의 세 가지 전달인자를 input() 함수를 이용해 입력받는다. 물건 가격과 매달 납입할 할부금은 정수, 이자율은 실수이므로 각각 int()와 float()을 이용하여 형변환한다. 매달 납입할 할부금은 이자보다는 커야하기 때문에 if문을 사용해 이를 판단한다.

만약 매달 납입하는 할부금이 이자보다 크다면 installment() 함수를 실행한다. 그리고 프로그램을 계속 실행할 지 묻는다. 입력한 quit의 값이 '1'이라면 continue문을 사용해 프로그램을 다시 실행하고, '2'라면 break문을 사용해 프로그램을 중단한다. 만약 두 숫자가 아닌 값을 입력했다면 오류 메시지 msg_err를 출력한다.

만약 매달 납입하는 할부금이 이자보다 작다면, 이에 해당하는 오류 메시지를 출력한다. 그리고 프로그램을 계속 실행할지 묻는다. 이와 같은 방식으로 코드를 작성한다.

처음 세 가지 전달인자의 값이 잘못되었으면, except문에서 ValueError를 발생시키고 오류 메시지 msg_err를 출력한다.

10-8 ex10_8.py

```
def compare_texts(text1, text2):
    """두 문장을 비교하여 중복 단어의 개수를 반환하는 함수다.
    text1....: 첫 번째 문장이나 텍스트(문자열)
    text2....: 두 번째 문장이나 텍스트(문자열)
    Returns: 중복 단어를 키로, 중복 횟수를 매핑값으로 하는 딕셔너리
    """
    import string                   # 문장에 있는 부호들을 처리하기 위해 string 클래스를 불러온다.

    result = {}                     # 결과를 담을 딕셔너리를 초기화한다.

    # string 클래스의 punctuation에 있는 모든 부호를 문장과 하나씩 비교해서
    # 문장에 있는 부호들을 제거한다.
    for t in string.punctuation:
        text1 = text1.replace(t, '').lower()
        text2 = text2.replace(t, '').lower()

    # 문장 부호를 제거한 문자열을 단어별로 분리하여 리스트로 저장한다.
    text1 = text1.split(' ')
    text2 = text2.split(' ')

    # --- 같은 단어를 확인하는 순환문 ---------------------------------------------- #
    for index1, word1 in enumerate(text1):          # text1의 단어들을 하나씩 선택하고
        for index2, word2 in enumerate(text2):      # text2의 단어들을 하나씩 선택하여
            if word1 == word2:                      # 선택된 두 단어가 같으면
                # result 딕셔너리에 해당 키(key)가 있을 때는 중복 횟수만 1 증가시킨다.
                result[word1] = result.get(word1, 0) + 1
                text1[index1] = None                # 같은 단어의 중복 확인을 방지하는 코드
                text2[index2] = None                # 같은 단어의 중복 확인을 방지하는 코드
                break
            else:                                   # 선택한 두 단어가 다르면
                continue                            # 순환문을 계속 실행한다.

    return result                                   # 결과 딕셔너리를 반환한다.
```

```
# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    test_data = [
        ('apple is an apple!!!', 'this is not an apple!'),
        ('Python is the best computer language! but I do '
         'not prefer Python than Java for some purpose.',
         'Sometimes, Java is more powerful than Python '
         'language. but I recommend Python for scientific '
         'purpose.')
    ]

    for t1, t2 in test_data:
        result = compare_texts(t1, t2)
        print('테스트 데이터 1:', t1)
        print('테스트 데이터 2:', t2)
        print('결과........:', result)
        print()
```

[해설]

compare_texts()는 두 문장을 비교하여 중복 단어의 개수를 반환하는 함수다. 따라서 비교 대상인 두 문장이 전달인자가 된다. 문장 부호를 제거하기 위해 string 클래스를 불러온 후, 함수의 결괏값을 담을 빈 딕셔너리 result를 만든다. for문을 이용하여 string 클래스의 punctuation에 있는 문장 부호를 두 문장의 문장 부호와 비교하고, 존재하는 문장 부호는 문자열 메소드인 replace()를 사용해서 빈 문자열('')로 바꾼다. 이렇게 해서 문자를 제외한 불필요한 문장 부호들을 제거한다. 그리고 중복 단어를 대소문자 구분 없이 비교하기 위해 lower() 메소드를 이용해서 모든 문자를 소문자로 변환한다.

문장 부호를 제거했다면, 문자열 메소드인 split()을 이용하여 띄어쓰기(' ')를 기준으로 문장을 단어별로 분리한다. 즉, text1, text2를 단어별로 분리한 리스트를 생성하여 재할당한다.

이제 for문을 이용해 두 문장의 단어를 비교할 것이다. enumerate() 함수를 사용해 리스트 text1, text2의 인덱스와 단어를 각각 index1, index2와 word1, word2에 지정한다. 만약 word1과 word2가 같으면 딕셔너리 result에서 해당 단어를 키(key)로 하는 매핑값(value)에 중복 횟수인 1을 더한다. 이때 딕셔너리의 get() 메소드를 사용한다. get()을 사용할 때의 이점은 다음 [참고]에서 상세히 설명하도록 한다.

다음으로 이전에 확인한 단어를 중복 확인하지 않기 위해 None값으로 만든 후 break문을 통해 종료한다. 그런데 word1과 word2가 같지 않다면 continue문으로 순환문을 계속해서 실행한다. for문의 실행이 종료되면 결괏값을 담은 딕셔너리 result를 반환한다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

[참고]

이 프로그램에서 딕서너리 메소드인 get()을 사용하는 이유는 사용하지 않았을 때보다 훨씬 코드가 간결해지기 때문이다. get()은 찾고자 하는 키가 딕셔너리 안에 있으면, 해당 매핑값을 반환한다. 하지만 찾고자하는 키가 딕셔너리 안에 없으면, 함께 주어진 두 번째 전달인자의 값을 반환한다. 두 번째 전달인자를 지정하지 않으면 None을 반환한다. 따라서 찾고자 하는 키가 딕셔너리 안에 없는 경우를 따로 고민하지 않아도 되어 효율적이다.

다음 두 코드를 비교해보자.

```
if word1 == word2:                          # 선택된 두 단어가 같을 때
    # result 딕셔너리에 해당 키(key)가 있을 때는 중복 횟수만 1 증가시킨다.
    result[word1] = result.get(word1, 0) + 1
    text1[index1] = None                    # 같은 단어의 중복 확인을 방지하는 코드
    text2[index2] = None                    # 같은 단어의 중복 확인을 방지하는 코드
```

```
if word1 == word2:                          # 선택된 두 단어가 같을 때
    if word1 in result.keys():              # result 딕셔너리에 해당 키(key)가 있으면
        result[word1] += 1                  # 중복 횟수만 1 증가시킨다.
        text1[index1] = None                # 같은 단어의 중복 확인을 방지하는 코드
        text2[index2] = None                # 같은 단어의 중복 확인을 방지하는 코드
        break
    else:                                   # result 딕셔너리에 해당 키(key)가 없으면
        result[word1] = 1                   # 단어를 키(key)로 매핑값을 1로해서 딕셔너리에 추가한다.
        text1[index1] = None                # 같은 단어의 중복 확인을 방지하는 코드
        text2[index2] = None                # 같은 단어의 중복 확인을 방지하는 코드
        break
```

첫 번째 코드에서 word1은 키가 없더라도 함께 주어진 값 0에 1을 더해 단어를 키로 매핑값을 1로 해서 딕셔너리에 추가하고, 키가 있으면 중복 횟수 1을 더해 의도한 대로 코드가 실행된다. 반면 get() 메소드를 사용하지 않은 두 번째 코드에서는 키가 있는지 없는지 확인하는 절차가 필요하다. 만약 확인하지 않고 result[word1]에 바로 1을 더한다면 KeyError가 발생할 수 있기 때문이다. 따라서 키가 없으면 else에서 키에 해당하는 word1의 매핑값을 지정해주어야 하기 때문에 첫 번째 코드보다 비효율적인 코드라고 할 수 있다.

11장

11-1 ex11_1.py

```
lines = open('고향의봄.txt', encoding='utf-8').readlines()

count = 0                                   # count 값을 0으로 설정한다.
for line in lines:                          # 각 문장을 추출하기 위해 for문을 사용한다.
    count += line.count('꽃')               # 문장마다 '꽃'의 개수를 더해 재할당한다.

print(count)                                # count를 출력한다.
```

[해설]

파일 열기의 기본적인 활용법을 묻는 문제다. open() 함수로 '고향의봄.txt' 파일을 읽기 모드로 열고, 파일 메소드인 readlines()를 사용해서 파일의 각 줄을 문자열 객체로 갖는 리스트 lines를 생성한다.

이제 '꽃'이 고향의 봄에서 몇 번 등장하는지 알아볼 것이다. 우선 변수 count를 0으로 설정한다. 단어 '꽃'이 등장할 때마다 count에 1씩 추가할 것이다. 그리고 for문을 활용하여 lines의 각 줄을 추출하고 줄마다 '꽃'이라는 단어가 몇 개 들어있는지 count() 메소드를 이용해 계산한다. 계산한 결과를 count에 더하여 재할당한다. 마지막으로 결괏값인 count를 출력한다.

11-2 ex11_2.py

```
# zen.txt 파일을 '추가' 모드로 열어서 형식대로 작자와 제목을 입력하고 파일을 닫는다.
with open('zen.txt', mode='a', encoding='utf-8') as file:
    file.write('\n\n{} - {}'.format('Tim Peters', 'The Zen of Python'))

# 수정한 파일을 다시 열어 새줄바꿈 부호를 삭제한 각 줄을 문자열 객체로 갖는 리스트를 반환한다.
lines = open('zen.txt', mode='r', encoding='utf-8').read().splitlines()

for line in lines:                                  # for문을 이용해 리스트의 각 줄을 차례로 추출해서
    print(line)                                     # 출력한다.
```

[해설]

여러 가지 모드로 파일을 여는 법을 연습하는 문제다. 여기서는 파일의 마지막에 내용을 추가해야 하므로 추가 모드인 'a'를 사용한다. 'zen.txt' 텍스트 파일을 'a' 모드로 연 다음, write() 메소드를 사용해서 파일의 마지막에 작자와 제목을 추가한다. 이때 주어진 형식을 만족하기 위해 문자열의 format() 메소드를 사용한다. 또한 가져온 텍스트와 입력 정보 사이에 빈 줄을 넣어야 하므로 새줄바꿈(\n)을 두 번 사용한다. 파일을 열 때 with 구문을 사용하면 종료할 때 자동으로 파일이 닫히기 때문에 따로 close() 메소드를 호출할 필요는 없다.

파일의 마지막 부분에 내용을 제대로 추가했는지 확인하기 위해서 수정한 파일을 읽기 모드인 'r'로 연다. 파일 전체를 read() 메소드로 읽은 후 문자열 메소드인 splitlines()로 처리하면, 각 줄의 끝에 있는 새줄바꿈(\n)을 제거한 문자열을 객체로 갖는 리스트를 반환한다. 파일 전체를 출력하기 위해 for문을 활용해서 각 줄을 차례로 출력한다.

한편 read()와 splitlines() 메소드 대신 readlines() 메소드를 사용해서 lines = open('zen.txt', mode = 'r', encoding='utf-8').readlines()로 실행해도 된다. 이때는 각 줄의 마지막에 새줄바꿈이 있는 문자열을 객체로 갖는 리스트를 반환한다.

만약 readlines() 메소드를 사용해서 파일을 읽었다면, print() 함수는 출력 후 기본적으로 새줄바꿈을 하기 때문에 최종 결과는 줄과 줄 사이에 이중으로 줄바꿈이 일어난다. 따라서 문자열의 rstrip() 메소드를 사용하여 오른쪽 끝에 있는 새줄바꿈을 삭제한 후 출력해야 한다.

```
for line in lines:
    print(line.rstrip())
```

11-3 ex11_3.py

```
def hours_week(dataset, country):
    """평균 주당 노동 시간을 구하는 함수다.
    dataset: 전처리한 adult 데이터 세트(리스트)
    country: 출신 국가(문자열)
    Returns: 평균 주당 노동 시간을 소수점 두 자리에서 반올림한 실수
    """
    hours = []                                      # 주당 노동 시간을 담을 리스트를 초기화한다.
    for d in dataset:                               # dataset 안 딕셔너리를 차례로 추출한다.

        if d.get('native-country') == country:      # 'native-country'의 매핑값이 전달인자와 같으면
            # 'hours-per-week'의 매핑값을 가져와서 정수로 변환한다.
            hours.append(int(d.get('hours-per-week')))
    return round(sum(hours) / len(hours), 2)        # 평균을 구한 후 반올림해서 반환한다.

# adult_US.txt 파일을 열어 파일 안 각 줄을 문자열 객체로 갖는 리스트를 반환한다.
lines = open('adult_US.txt', encoding='utf-8').read().splitlines()

adult_data = []                                     # 데이터를 담을 리스트를 초기화한다.

header = [                                          # 키(key)로 사용할 열의 제목을 모아둔 리스트
    'age', 'workclass', 'fnlwgt', 'education', 'education-num',
    'martal-status', 'occupation', 'relationship', 'race', 'sex',
    'capital-gain', 'capital-loss', 'hours-per-week', 'native-country', 'salary'
]

for line in lines:                                  # 각 줄을 차례로 하나씩 추출하여
    row_list = line.split(',')                      # 줄마다 ',' 기준으로 분할한다.
    for index, item in enumerate(row_list):         # 그 항목들을 하나씩 추출하여
        row_list[index] = item.strip()              # 앞뒤 공백을 제거한다.

    row_dict = {}                                   # 각 줄의 정보를 담을 딕셔너리를 초기화한다.
    for col_name in header:                         # header의 키를 하나씩 꺼낸다.
        # 키와 매핑값을 row_dict로 저장한다.
        row_dict[col_name] = row_list[header.index(col_name)]
    adult_data.append(row_dict)                     # 해당 사전을 adult_data에 추가한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    test_data = 'United-States', 'Cuba', 'Mexico'  # 테스트용 데이터를 튜플로 만든다.
    for i in test_data:
        print('{} 출신 사람들의 평균 주당 노동 시간: {}'.format(i, hours_week(adult_data,
i)))
```

[해설]

[연습문제 11-3]부터는 텍스트 파일의 정보를 파싱하는 방법에 대한 문제다. 데이터를 어떻게 전처리하는지가 중요하다.

먼저 출신 국가별로 평균 주당 노동 시간을 계산하는 함수를 정의한다. 리스트로 전처리한 데이터(dataset)와 출신 국가(country)를 전달인자로 받아서 해당 국가의 평균 주당 노동 시간을 반환하는 함수 hours_week()를 만든다. 그리고 입력한 나라에 해당하는 주당 노동 시간을 담을 리스트를 초기화해서 변수 hours에 할당한다.

이제 전달받은 출신 국가에 해당하는 정보를 찾기 위해 adult_dataset에 있는 딕셔너리를 하나씩 추출한다. 비교할 매핑값은 'native-country'라는 키가 있으므로, 딕셔너리의 get() 메소드를 활용하여 이를 가져온다. 만약 'native-country'의 매핑값이 입력한 나라와 같은 값이면, 딕셔너리 안에서 키가 'hours-per-week'인 매핑값을 찾아 이를 리스트 hours에 추가한다.

이제 이 정보들로 평균을 구할 차례다. 리스트 hours 안 모든 수의 전체 합을 총 개수로 나눈다. 그리고 round() 함수를 사용해 소수점 둘째 자리까지만 구해서 반환한다.

이제부터 저장한 텍스트 파일을 전처리한 후 앞에서 정의한 함수를 사용해서 출신 국가별 평균 주당 노동 시간을 계산해보자.

먼저, 'adult_US.txt' 파일을 읽기 모드로 열고, 파일 전체를 read() 메소드로 읽은 후 문자열 메소드인 splitlines()로 처리한다. 각 줄의 맨 끝에 있는 새줄바꿈(\n)을 제거한 문자열을 객체로 갖는 리스트를 반환해준다. read()와 splitlines() 메소드를 사용하지 않고 대신 readlines() 메소드를 사용해서 lines = open('adult_US.txt', mode = 'r', encoding='utf-8').readlines()로 실행해도 된다. 이때는 각 줄의 행의 마지막에 새줄바꿈이 있는 문자열을 객체로 갖는 리스트를 반환한다.

파일에서 데이터를 읽은 후 각 줄의 정보를 담은 딕셔너리를 생성해서 빈 리스트로 초기화해서 변수 adult_data에 할당한다. 그리고 각 줄을 딕셔너리로 담을 때 사용할 키(key)를 모아둔 리스트 header를 생성한다. 키 정보는 문제에 있는 링크에서 찾아볼 수 있다.

이제 각 줄을 딕셔너리로 만들어보자. 우선 for문을 활용하여 리스트 lines에서 각 줄을 차례로

하나씩 추출한다. 이때 각 줄의 정보는 쉼표(,)로 구분했기 때문에 문자열 메소드인 split()를 사용하여 쉼표(,) 기준으로 분할한 객체를 갖는 리스트를 반환해서 변수 row_list에 할당한다. 이때 개별 문자열 객체 앞뒤에 있는 불필요한 공백을 없애기 위해 다시 for문을 이용해서 row_list에 있는 데이터를 차례로 추출하여 strip() 메소드를 적용한다. enumerate() 함수를 사용했기 때문에 문자열 객체 자신의 인덱스 위치에서 스스로를 공백을 제거한 문자열로 대체한다. 그럼 존재했던 공백이 전부 사라지고 깔끔하게 데이터를 가공할 수 있다.

각 줄의 정보를 키와 매핑값으로 담을 빈 딕셔너리를 만들어서 변수 row_dict에 할당한다. 이때 아까 만들어 두었던 리스트 header를 사용한다. for문을 이용해 row_dict의 키로 사용할 header의 각 문자열을 하나씩 추출한다. 키로 사용할 열의 제목들을 순서대로 header에 저장했기 때문에, header의 인덱스 값을 사용해 row_list의 데이터를 추출한 값이 바로 키에 대응하는 매핑값이다.

이렇게 각 줄을 키와 매핑값의 딕셔너리로 만들어 row_dict로 추가해서 완성한 딕셔너리는 전체 정보를 모아두는 리스트 adult_data에 추가한다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

11-4 ex11_4.py

```
def wines(dataset, alcohol, flavonoid):
    """와인 목록에서 지정한 알코올 도수보다는 낮고, 플라보노이드의 수치는 높은 와인 개수를 반환하는 함수다.
    dataset..: 전처리한 wine quality 데이터 세트(리스트)
    alcohol..: 알코올 도수(숫자형)
    flavonoid: 플라보노이드 수치(숫자형)
    Returns..: 추천 와인 개수(정수)
    """
    count = 0                                           # 추천할 와인 개수를 초기화한다.
    try:
        for d in dataset:                               # dataset 안 딕셔너리를 차례로 추출한다.
            # 전달인자와 dataset의 값을 비교해서 만족하면, count에 1 추가한다.
            if d.get('alcohol') < alcohol and d.get('flavonoids') > flavonoid:
                count += 1
        return count                                    # 조건을 충족하는 추천 와인 개수를 반환한다.
    except TypeError:                                   # 전달인자가 정수나 실수가 아니면
        print('오류입니다. 정수나 실수를 입력하세요!')       # 오류 메시지를 출력한다.
# wine_quality.csv 파일을 열어 파일 안 각 줄을 문자열 객체로 갖는 리스트를 반환한다.
lines = open('wine_quality.csv', mode='r', encoding= 'utf-8').read().splitlines()

wine_data = []                                          # 각 행의 데이터를 담을 리스트를 초기화한다.

header  = [                                             # 키(key)로 사용할 열 제목을 모아둔 리스트
    'classifier', 'alcohol', 'malic acid', 'ash', 'alcalinity of ash',
    'magnesium', 'total phenols', 'flavonoids', 'nonflavanoid phenols',
    'proanthocyanins', 'color intensity', 'hue', 'OD280/OD315 of diluted wines',
    'proline'
]

for line in lines:                                      # 행을 차례로 하나씩 추출하여
    row_list = line.split(',')                          # 해당 줄을 ',' 기준으로 분할한다.
    row_dict = {}                                       # 각 줄의 정보를 담을 딕셔너리를 초기화한다.
    for col_name in header:                             # header의 키를 하나씩 추출한다.
        # 키와 실수를 변환한 매핑값을 row_dict로 저장한다.
        row_dict[col_name] = float(row_list[header.index(col_name)])
    wine_data.append(row_dict)                          # 해당 딕셔너리를 wine_data에 추가한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    test_data = (15, 'a'), (13, 4), (15.5, 3)           # 테스트용 데이터를 튜플로 만든다.
    for i, j in test_data:
        print('테스트 전달인자: 알코올 도수 = {}, 플라보노이드 수치 = {}'.format(i, j))
        print('추천 와인 수..: {}'.format(wines(wine_data, i, j)))
        print()
```

[해설]

이번 문제에서는 텍스트 파일(txt)이 아닌 csv(comma-separated values) 파일을 다룬다. 하지만 파일을 읽는 방법은 거의 같다.

먼저 문제 조건을 만족하는 함수를 정의한다. 리스트로 전처리한 데이터(dataset), 알코올 도수(alcohol), 플라보노이드 수치(flavonoid)를 전달인자로 받아서 입력한 값의 알코올 도수보다는 낮고, 플라보노이드의 수치는 높은 와인 개수를 반환하는 함수 wines()를 만든다.

추천할 와인이 총 몇 개인지 담아둘 정숫값를 초기화해서 변수 count에 할당한다. 이때 dataset의 모든 매핑값이 실수이기 때문에 alcohol과 flavonoid가 정수 또는 실수가 아니면 두 값을 비교할 수 없다. 따라서 예외 처리 try-except문을 사용해서 알코올 도수와 플라보노이드 수치를 비교한다. 만약 둘 중에 하나라도 숫자가 아니라면 TypeError가 발생하고 오류 메시지를 출력한다.

만약 전달인자 값에 문제가 없다면 dataset 안에 있는 딕셔너리를 하나씩 검사해서 전달인자 알코올 도수보다 낮은 값이면서, 플라보노이드 수치보다 높은 값인 와인을 찾는다. 딕셔너리 get() 메소드로 키가 'alcohol'과 'flavonoids'인 객체의 매핑값을 추출한다. 그리고 두 조건을 모두 만족해야 하므로 논리 연산자 and를 사용한다. 이 조건을 충족하는 와인이 나타나면 count 변숫값을 1 증가시킨다. dataset에 있는 모든 와인의 탐색이 끝나면 count를 반환한다.

이제부터 저장한 텍스트 파일을 전처리한 후, 앞에서 정의한 함수를 사용해서 고객에게 맞는 와인이 몇 개 있는지 알아보자. 먼저 'wine_data.csv' 파일을 읽기 모드로 열고, 파일 전체를 read() 메소드로 읽은 후 문자열 메소드인 splitlines()로 처리한다. 각 줄의 맨 끝에 있는 새줄바꿈(\n)을 제거한 문자열을 객체로 갖는 리스트를 반환해준다. read()와 splitlines() 메소드를 사용하지 않고 대신 readlines() 메소드를 사용해서 lines = open('wine_data.csv', mode = 'r', encoding='utf-8').readlines()로 실행해도 된다. 이때는 각 줄의 마지막에 새줄바꿈이 있는 문자열을 객체로 갖는 리스트를 반환한다.

파일에서 데이터를 읽은 후 각 줄의 정보를 담은 딕셔너리를 생성해서 빈 리스트로 초기화해서 변수 wine_data에 할당한다. 그리고 각 줄을 딕셔너리로 담을 때 사용할 딕셔너리의 키(key)를 모아둔 리스트 header를 생성한다. 키 정보는 문제에 있는 링크에서 찾아볼 수 있다.

이제 각 줄을 딕셔너리로 만들어보자. 우선 리스트 lines에서 줄에 해당하는 정보를 추출한다. 각 줄의 정보는 쉼표(,)로 구분했다. 따라서 문자열 메소드인 split()를 사용하여 개별 데이터를 쉼표(,) 기준으로 분할한 리스트를 변수 row_list에 할당한다. 그리고 각 줄의 정보를 담을 빈 딕셔너리를 만들어서 변수 row_dict에 할당한다. 이 딕셔너리에 개별 줄의 정보를 키와 매핑값으로 저장한다. 이때 아까 만들어 두었던 header를 사용한다. for문을 이용해 row_dict의 키로 사용할 header의 각 문자열을 하나씩 추출한다.

다음으로 키에 대응하는 매핑값을 찾아야 한다. 정보 순서대로 키를 header에 저장했기 때문에 header의 인덱스 값을 사용해 row_list의 데이터를 추출한 값이 바로 키에 대응하는 매핑값이다. 그리고 이 값은 모두 실수로 변환하여, 각 줄을 키와 매핑값의 딕셔너리로 만들어 row_dict에 추가한다. 완성한 딕셔너리는 전체 정보를 모아두는 리스트 wine_data에 추가한다.

끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 if __name__ == '__main__': 안에 작성했다.

11-5 ex11_5.py

```
def customer_stat(dataset, job, avg_age=True):
    """고객의 직업에 따른 평균 나이 또는 평균 전화 지속 시간을 출력하는 함수다.
    dataset..: 전처리한 bank 데이터 세트(리스트)
    job.....: 고객의 직업(문자열)
    avg_age: True면 직업별 평균 나이(실수), False면 직업별 평균 전화 통화 시간(실수)
    """

    # dataset에 있는 직업군 튜플
    job_in_dataset = (
        'admin.', 'blue-collar', 'entrepreneur', 'housemaid', 'management',
        'retired', 'self-employed', 'services', 'student', 'technician',
        'unemployed','unknown'
    )
    # 전달인자 job이 해당 직업에 없다면 ValueError를 발생시킨다.
    if job not in job_in_dataset:
        raise ValueError('데이터 세트에 없는 직업입니다')

    result = []                          # 나이 또는 전화 통화 시간을 담을 리스트를 초기화한다.

    # 전달인자 job이 해당 직업에 있다면 dataset 안 딕셔너리를 차례로 추출한다.
    for d in dataset:
        if d['job'] == job:              # 딕셔너리의 직업이 전달인자 job과 일치하고
            if avg_age:                  # avg_age가 True면 해당 직업의 고객 나이를 추가한다.
                result.append(int(d.get('age')))
            else:                        # avg_age가 True면 해당 직업의 고객 통화 시간을 추가한다.
                result.append(int(d.get('duration')))

    if avg_age:                          # avg_age가 True면 해당 직업의 고객들 평균 나이를 출력한다.
        print('{} 직업의 평균 나이: {:,.2f}'.format(job, sum(result)/len(result)))
    else:                                # avg_age가 False면 해당 직업의 고객들 평균 나이를 출력한다.
        print('{} 직업의 평균 통화 시간: {:,.2f}'.format(job, sum(result)/len(result)))

# bank.csv 파일을 열어 파일 안 각 줄을 문자열 객체로 갖는 리스트를 반환한다.
lines = open('bank.csv', mode='r', encoding= 'utf-8').read().splitlines()

bank_data = []                           # 각 행의 데이터를 담을 리스트를 초기화한다.

# 파일 첫 번째 줄 값에서 양 끝의 큰따옴표를 제거한 후 키(key)로 사용할 열 제목을 리스트에 할당한다.
header = lines[0].replace('"', '').split(';')
```

```
for line in lines[1:]:                                  # 파일의 두 번째 줄부터 차례로 하나씩 추출하여
    # 해당 줄의 큰따옴표를 모두 제거한 후 ';' 기준으로 분할한다.
    row_list = line.replace('"', '').split(';')
    row_dict = {}                                       # 각 줄의 정보를 담을 딕셔너리를 초기화한다.
    for col_name in header:                             # header의 키 값을 하나씩 추출하여
        # 키와 매핑값을 row_dict로 저장한다.
        row_dict[col_name] = row_list[header.index(col_name)]
    bank_data.append(row_dict)                          # 해당 사전을 bank_data에 추가한다.

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    # 테스트용 데이터를 튜플로 만든다.
    test_data = ('admin', True), ('Govern', True), ('housemaid', False)
    for i, j in test_data:
        print('테스트 전달인자: job = {}, avg_age = {}'.format(i, j))
        try:
            customer_stat(bank_data, i, j)
        except ValueError as err:
            print(err)
        print()
```

[해설]

이번 문제에서도 csv 파일을 다룰 것이다. 먼저 문제 조건을 만족하는 함수를 정의한다. 전달인자로는 리스트로 전처리한 데이터(dataset), 직업(job), 직업별 평균 나이를 계산해서 출력할지, 직업별 평균 통화 시간을 계산해서 출력할지 결정하는 불린(avg_age) 값을 받아서 고객 직업에 따른 평균 나이 또는 평균 전화 지속 시간을 출력하는 함수 customer_stat()을 만든다.

직업에 따른 평균 나이나 통화 시간을 담아둘 리스트를 초기화해서 변수 result에 할당한다. dataset에 있는 모든 직업군을 변수 job_in_dataset에 튜플로 할당한 후, 전달인자로 지정한 job이 dataset에 없는 직업이라면 ValueError를 발생시킨다.

dataset에 있는 직업이라면 dataset 안에 있는 딕셔너리를 하나씩 추출해서, 딕셔너리의 'job'이라는 키가 전달인자의 직업(job)과 같은지 비교한다. 만약 같다면 세 번째 전달인자인 avg_age에 따라 찾고자 하는 정보를 다르게 한다. avg_age가 True면, 키 'age'를 통해 나이에 해당하는 매핑값을, False면 키 'duration'을 통해 전화 지속 시간에 해당하는 매핑값을 리스트 result에 추가한다.

정보 처리가 끝나면 평균을 구하기 위해 리스트 result에 담긴 모든 수의 총합을 전체 개수로 나눈다. 그리고 avg_age의 값에 따라 True면 직업별 고객의 평균 나이를, False면 직업별 평균 통화 지속 시간을 출력한다. 출력할 때 format() 메소드의 서식 설정({:,.2f})에서 쉼표(,)를 사용해 숫자를 세 자리 단위, 소수점 두 자리에서 반올림하는 형식으로 출력한다.

이제부터 저장한 텍스트 파일을 리스트로 전처리한 후, 앞에서 정의한 함수를 사용해서 고객들의 직업별 평균 나이 또는 평균 통화 지속 시간을 알아보자. 먼저 'bank.csv' 파일을 읽기 모드로 열고, 파일 전체를 read() 메소드로 읽은 후 문자열 메소드인 splitlines()로 처리한다. 각 줄의 맨 끝에 있는 새줄바꿈(\n)을 제거한 문자열을 객체로 갖는 리스트를 반환한다. read()와 splitlines() 메소드를 사용하지 않고 대신 readlines() 메소드를 사용해서 lines = open('bank.csv', mode = 'r', encoding='utf-8').readlines()로 실행해도 된다. 이때는 각 줄의 마지막에 새줄바꿈이 있는 문자열을 객체로 갖는 리스트를 반환한다.

파일에서 데이터를 읽은 후 각 줄의 정보를 담은 딕셔너리를 생성하고 빈 리스트로 초기화해서 변수 bank_data에 할당한다. 그리고 각 줄을 딕셔너리로 담을 때 사용할 딕셔너리의 키(key)를 담아둔 리스트 header를 생성한다. 키 정보는 해당 자료의 첫 줄에서 찾아볼 수 있기 때문에 리스트 lines의 첫 번째 객체를 추출한다. 여기서 한 가지 문제가 발생한다. 자료 안 정보에 큰따옴표(")가 있는 것이다. 이와 같은 경우 전달인자에 큰따옴표를 붙이지 않는 이상, 파싱한 정보와의 비교를 할 수 없다. 따라서 replace() 메소드를 사용해 먼저 큰따옴표를 빈 문자열('')로 바꾼 후, 문자열의 split() 메소드를 사용해서 세미콜론(;) 기준으로 분할한 리스트를 변수 header에 할당한다.

이제 각 줄을 딕셔너리로 만들어보자. 우선 리스트 lines에서 줄에 해당하는 정보들을 추출한다. 첫 줄은 header에 해당하기 때문에 이를 제외하기 위해 인덱스를 0이 아닌 1부터 시작한다. header 정보와 마찬가지로 replace() 메소드를 사용해서 먼저 큰따옴표를 제거한 후, split() 메소드를 사용해 개별 세미콜론(;) 기준으로 분할한 리스트를 변수 row_list에 할당한다. 그리고 각 줄의 정보를 담을 빈 딕셔너리를 만들어서 변수 row_dict에 할당한다. 이 딕셔너리에 개별 줄의 정보를 키와 매핑값으로 저장한다. 이때 아까 만들어 두었던 header를 사용한다.

for문을 이용해 row_dict의 키로 사용할 header의 각 문자열을 하나씩 추출한다. 앞서 정보의 순서대로 키를 header에 저장했다. 따라서 header의 인덱스 값을 사용해 row_list의 데이터를

추출한 값이 바로 키에 대응하는 매핑값이다. 이렇게 각 줄을 키와 매핑값의 딕셔너리로 만들어 row_dict로 추가한다. 완성한 딕셔너리는 전체 정보를 모아두는 리스트 bank_data에 추가한다. 끝으로 테스트 코드는 인터프리터 모드에서만 실행할 수 있도록 조건문인 `if __name__ == '__main__':` 안에 작성했다.

12-1 ex12_1.py

```
class Vector2D:
    """2차원 실수 평면 위 벡터(점)를 구현한 클래스다.
    속성...: __x_point, __y_point
    메소드: distance(), dot()
    사용법:
        >>> v1 = Vector2D('3', 4.0)
        >>> v1.distance()
        5.0
        >>> v2 = Vector2D(1, 2)
        >>> v2.dot(v1)
        11.0
    """
    # --- 초기화 메소드 ----------------------------------------------------------------#
    def __init__(self, x, y):
        """인스턴스를 생성할 때 x와 y 좌표를 전달받아 초기화하는 메소드다.
        x.....: x 좌표(실수)
        y.....: y 좌표(실수)
        Returns: None
        사용법..:
            >>> v1 = Vector2D('1', '2')
            >>> v2 = Vector2D(1, 2)
            >>> v3 = Vector2D(1.0, 2.0)
            >>> v4 = Vector2D(1, 2.3)
            >>> v5 = Vector2D(1.2, '2')
        """
        try:                                    # 전달인자를 실수로 변환해서 인스턴스 속성에 할당한다.
            self.__x_point = float(x)           # 인스턴스 속성: x 좌표
            self.__y_point = float(y)           # 인스턴스 속성: y 좌표
        except ValueError:
            print('ERROR: x 좌표와 y 좌표 모두 숫자일 때만 정의합니다.')
    # --- 일반 메소드 ------------------------------------------------------------------#
    def distance(self):
        """해당 벡터(점)와 원점 사이 거리를 반환하는 메소드다.
        Returns: 원점으로부터의 거리(실수)
        사용법..:
            >>> v1 = Vector2D(3, 4)
            >>> v1.distance()
            5.0
        """
        return (self.__x_point ** 2 + self.__y_point ** 2) ** (1/2)
```

```
    def dot(self, other):
        """Vector2D 클래스의 다른 인스턴스와의 내적을 반환하는 메소드다.
        other....: Vector2D 클래스의 인스턴스
        Returns: 두 Vector2D 클래스 인스턴스 사이의 내적(실수)
        사용법..:
            >>> v1 = Vector2D(3, 4)
            >>> v2 = Vector2D(1, 2)
            >>> v2.dot(v1)
            11.0
        """
        try:
            return self.__x_point * other.__x_point + self.__y_point * other.__y_point
        except Exception:
            print('ERROR: Vector2D 클래스 인스턴스가 아닙니다.')

# ----- 인터프리터 모드에서만 실행됨 ---------------------------------------------------- #
if __name__ == '__main__':
    print('Vector2D.__doc__')
    print(Vector2D.__doc__)              # Vector2D 클래스의 설명문자열을 출력한다.
    print("v1 = Vector2D('3', 4.0)")
    v1 = Vector2D('3', 4.0)              # Vector2D 인스턴스 v1을 생성한다.
    print('v1.distance()')
    print(v1.distance())                 # 원점으로부터의 v1 거리를 구한다.
    print('v2 = Vector2D(1, 2)')
    v2 = Vector2D(1, 2)                  # Vector2D 인스턴스 v2를 생성한다.
    print('v2.dot(v1)')
    print(v2.dot(v1))                    # v2와 v1의 내적을 구한다.
    print('v2.dot(3)')
    v2.dot(3)                            # v2와 정수 3의 내적을 구하기 때문에 오류가 난다.
    print("v3 = Vector2D('line', 2)")
    v3 = Vector2D('line', 2)             # Vector2D 인스턴스 v3의 좌표에 문자열이 있어 오류가 난다.
```

[해설]

우선 Vector2D 클래스의 인스턴스 속성을 정의해보자. 인스턴스 속성인 x 좌표와 y 좌표는 실수이기 때문에 인스턴스를 생성할 때 전달인자의 값은 실수로 변환 가능한 값이어야 한다. 예외가 발생할 때를 대비해서 try-except문에서 float()을 사용해 입력된 전달인자를 실수로 변환한다. 만약 변환이 불가능하다면 ValueError로 예외 처리한다.

벡터와 원점 사이의 거리를 구하는 distance() 메소드는 인스턴스 속성을 self.__x_point, self.__y_point로 불러와 점과 점 사이의 거리 공식을 사용하여 원점에서 인스턴스까지의 거리를 (self.__x_point ** 2 + self.__y_point ** 2) ** (1/2)로 계산해서 반환한다. math 모듈의 sqrt() 함수를 사용하여 math.sqrt(self.__x_point ** 2 + self.__y_point ** 2)로 처리해도 같은 결과를 가져온다.

두 인스턴스 사이의 내적을 구하는 dot() 메소드는 다른 인스턴스를 전달받아야 하기 때문에 other라는 전달인자를 추가했다. other가 Vector2D의 인스턴스가 아니면 예외 처리해야 하기 때문에 try-except문을 사용한다. other가 Vector2D의 인스턴스라면 other의 x 좌표, y 좌표를 각각 other.__x_point, other.__y_point로 불러온다. 그리고 내적 공식을 사용하여 self.__x_point * other.__x_point + self.__y_point * other.__y_point로 계산해서 반환한다.

other가 Vector2D의 인스턴스가 아니라면 예외가 발생할 것이기 때문에 예외 처리해서 메시지를 출력한다.

12-2 ex12_2.py

```
# ----- Dragon 클래스 정의 ----------------------------------------------------------#
class Dragon:
    """용 클래스다.
    속성...: __name, __hunger, __fatigability, __hygiene, __joy, __affection
    메소드: name(), hunger(), fatigability(), hygiene(), joy(), affection(), status(), change(), next_turn()
    """

    # --- 초기화 메소드 ---------------------------------------------------------------#
    def __init__(self, name):
        """용의 이름은 전달한 값으로 초기화하고 나머지 속성들은 0으로 초기화하는 메소드다.
        Returns: None
        """
        self.__name = name                                  # 인스턴스 속성: 용의 이름
        self.__hunger = 0                                   # 인스턴스 속성: 용의 배고픔 정도
        self.__fatigability = 0                             # 인스턴스 속성: 용의 피로도 정도
        self.__hygiene = 0                                  # 인스턴스 속성: 용의 위생상태
        self.__joy = 0                                      # 인스턴스 속성: 용의 행복지수
        self.__affection = 0                                # 인스턴스 속성: 용의 사랑지수

    # --- 접근자 메소드 ---------------------------------------------------------------#
    def name(self):
        """용의 이름을 반환하는 메소드다.
        Returns: 용의 이름(문자열)
        """
        return self.__name

    def hunger(self):
        """용의 배고픔 정도를 반환하는 메소드다.
        Returns: 용의 배고픔 정도(정수)"""
        return self.__hunger

    def fatigability(self):
        """용의 피로도를 반환하는 메소드다.
        Returns: 용의 피로도(정수)"""
        return self.__fatigability

    def hygiene(self):
        """용의 위생상태를 반환하는 메소드다.
        Returns: 용의 위생상태(정수)"""
        return self.__hygiene
```

```
    def joy(self):
        """용의 행복지수를 반환하는 메소드다.
        Returns: 용의 행복지수(정수)"""
        return self.__joy

    def affection(self):
        """용의 사랑지수를 반환하는 메소드다.
        Returns: 용의 사랑지수(정수)"""
        return self.__affection

    # --- 일반 메소드 ----------------------------------------------------------#
    def status(self):
        """용의 상태를 출력하는 메소드다.
        Returns: None
        """
        print('\n[ {}의 신체 상태 ]\n배고픔 = {} ¦ 피로도 = {} ¦ 위생상태 = {}'.format(
            self.__name, self.__hunger, self.__fatigability, self.__hygiene))

    def change(self, target):
        """용의 상태를 바꾸는 메소드다.
        target...: 용의 상태(문자열)
        Returns: None
        """
        if target == 'hunger':
            self.__hunger -= 3
        elif target == 'fatigability':
            self.__fatigability -= 3
        elif target == 'hygiene':
            self.__hygiene -= 3
        elif target == 'affection':
            self.__affection = 1 if self.__affection < 0 else self.__affection + 2
        elif target == 'joy':
            self.__joy = 1 if self.__joy < 0 else self.__joy + 1

    def next_turn(self):
        """게임이 다음 단계로 넘어가면서 용의 전체 상태를 업데이트하는 메소드다.
        Returns: None
        """
        self.__hunger += 1
        self.__fatigability += 1
        self.__hygiene += 1
        self.__joy -= 1
        self.__affection -= 1
```

```
# ----- 게임 시작 ---------------------------------------------------------------------- #
turn = 0                                                    # 게임의 최대 횟수를 초기화한다.

mn_game_rule = (                                            # 게임 시작 전에 게임 규칙을 설명한다.
    '\n##### 용 키우기 게임 #####\n게임 규칙:\n\t'
    '(1) 배고픔, 피로도, 위생상태가 5에 도달하면, 용은 죽는다.\n\t'
    '(2) 30단계 후, 용은 늙어 죽는다.'
)
mn_action = (                                               # 용에게 시킬 행동을 묻는다.
    '> {}에게 어떤 명령을 내릴까요?\n(1) 먹기, (2) 놀기, (3) 잠자기, '
    '(4) 데이트하기, (5) 씻기\n번호를 입력하세요 => '
)

print(mn_game_rule)                                         # 게임 규칙을 출력한다.

name = input('용의 이름을 입력하세요: ')                    # 용의 이름을 붙여준다.
dragon = Dragon(name)

while True:
    dragon.status()                                         # 용의 현재 상태를 표시한다.

    # 용이 죽을 수 있는 상태를 규정하여 용이 이 상태에 도달하면 게임을 끝낸다.
    if (dragon.hunger() >= 5 or dragon.fatigability() >= 5
        or dragon.hygiene() >= 5 or turn >= 30):
        if dragon.hunger() >= 5:
            print('!!! {}은(는) 배고파서 죽었습니다. !!!'.format(dragon.name()))
        elif dragon.fatigability() >= 5:
            print('!!! {}은(는) 수면 부족으로 죽었습니다. !!!'.format(dragon.name()))
        elif dragon.hygiene() >= 5:
            print('!!! {0}은(는) 씻지 못해서 죽었습니다. !!!'.format(dragon.name()))
        elif turn >=30:
            print('!!! {0}은(는) 늙어 죽었습니다. !!!'.format(dragon.name()))
        break

    # 사용자로부터 용에게 시킬 행동을 입력받는다.
    command = input(mn_action.format(dragon.name()))
```

```
    # 각 행동에 따라서 용의 상태를 변화시키고 결과를 출력한다.
    if command == '1':
        if 0 <= dragon.hunger() < 5:
            dragon.change('hunger')
            print('배고픔이 감소했습니다.')
        elif dragon.hunger() < 0:
            print('{}은(는) 배불러서 더 이상 먹을 수 없습니다.'.format(dragon.name()))
    elif command == '2':
        dragon.change('joy')
        if dragon.joy() > 0:
            print('{}은(는) 행복해보입니다.'.format(dragon.name()))
    elif command == '3':
        if 0 <= dragon.fatigability() < 5:
            dragon.change('fatigability')
            print('피로도가 감소했습니다.')
        elif dragon.fatigability() < 0:
            print('{}은(는) 너무 많이 자서 피곤하지 않습니다.'.format(dragon.name()))
    elif command == '4':
        dragon.change('affection')
        if dragon.affection() > 0:
            print('{}은(는) 사랑에 빠졌습니다.'.format(dragon.name()))
    elif command == '5':
        if 0 <= dragon.hygiene() < 5:
            dragon.change('hygiene')
            print('위생 상태가 나빠졌습니다.')
        elif dragon.hygiene() < 0:
            print('{}은(는) 너무 깨끗해서 더 이상 씻을 필요가 없습니다.'.format(
                dragon.name()))
    else:
        print('번호를 잘못 입력했습니다!!!!!')
        continue

    # 한 번 순회할 때마다 용의 상태를 갱신한다.
    dragon.next_turn()
    turn += 1
```

[해설]

이름이 Dragon인 클래스를 정의한다. 그리고 Dragon의 인스턴스 속성인 __name, __hunger, __fatigability, __hygiene, __joy, __affection을 초기화 함수에서 선언한다. Dragon의 인스턴스를 생성할 때 반드시 용의 이름을 지정해야 하기 때문에 __init__() 메소드의 매개변수로 name을 사용한다. 그리고 각 속성 값을 반환하는 여섯 개의 메소드를 구현한다.

용의 상태를 출력하는 status() 메소드를 정의한다. 문자열의 format() 메소드를 사용해서 각각 배고픔, 피로도, 위생 상태를 출력한다.

용에게 명령하면 용의 상태가 바뀌는데 이는 change() 메소드를 통해 구현한다. 용에게 내린 명령의 종류에 따라서 용의 상태 중 어떤 상태를 바꿀지 결정하는 매개변수로 target을 선언한다. 명령이 먹기, 잠자기, 씻기(target이 'hunger', 'fatigability', 'hygiene')면 각 인스턴스 속성을 3씩 감소시킨다. 명령이 데이트하기(target이 'affection')면 인스턴스 속성을 2씩, 놀기(target이 'joy')면 1씩 증가시킨다. 이때 데이트하기와 놀기 상태의 수치가 0보다 작으면 1로 초기화한다.

용이 명령을 실행해서 자신의 상태를 바꾼 후, 다음 명령을 내리기 전 자신의 전체 상태를 갱신한다. 이는 next_turn() 메소드를 통해 구현한다. 용에게 특정 명령을 하면 배고픔, 피로도, 위생 상태는 1씩 증가하고, 행복지수와 사랑지수는 1씩 감소한다.

이제 전략 게임을 구성해보자. 게임은 30단계가 지나면 종료하므로, 각 단계를 나타내는 변수 turn을 0으로 초기화한다. 게임의 규칙을 담은 변수 mn_game_rule와 게임을 실행할 때 명령을 나타내는 변수 mn_action를 생성한다. 게임을 본격적으로 시작하기 전 게임의 규칙을 출력한다. 또한 용의 이름을 입력받아 Dragon 클래스의 인스턴스를 생성해서 변수 dragon에 할당한다.

이제 본격적인 게임을 시작한다. while문을 사용해 게임을 반복 실행할 것이다. while문은 turn이 30이 되거나 배고픔, 피로도, 위생상태가 5에 도달할 때까지 반복적으로 실행한다. 용에게 명령을 내리기 전, status() 메소드를 호출해서 용의 현재 상태를 출력한다. 그리고 용이 죽을 수 있는 상태에 도달했는지 검사한다. turn이 30이 되거나 배고픔, 피로도, 위생상태 중 하나가 5에 도달하였다면, 이에 상응하는 메시지를 출력한다.

용이 죽을 수 있는 상태가 아니라면, 용에게 시킬 행동을 입력받는다. input() 함수를 이용해 용에게 시킬 행동을 입력받고, mn_action에서 정의한 명령을 입력받는다. 각 번호 가운데 1번은 배고픔, 2번은 행복지수, 3번은 피로도, 4번은 사랑지수, 5번은 위생상태를 변화시킬 것이다. 용의 인스턴스 속성을 변화시키기 위해 Dragon 클래스에서 정의한 change() 메소드를 사용한다. 배고픔, 피로도, 위생상태는 수치가 0 이상 5 미만일 때만 상태가 변화할 수 있다. 만약 배고픔, 피로도, 위생상태가 0보다 작으면, 명령이 필요없다는 내용을 출력한다. 행복지수와 사랑지수의 수치는 범위가 없으며 만약 행복지수와 사랑지수가 0보다 커지면 각각 상황에 어울리는 내용을

출력한다. 용의 상태가 바뀐 후에는 변화 내용을 출력한다. 1번에서 5번에 해당하는 번호를 입력하지 않았다면 오류 메시지를 출력한다.

명령을 한 번 실행한 후에는 용의 전체 상태를 업데이트해야 한다. 이때 `next_turn()` 메소드를 사용한다. 또한 한 단계가 지났으므로, `turn`을 1 증가시킨다.