객체지향 프로그래밍
파일 활용

1. 사전 Dictionary

지금까지 공부한 문자열, 튜플, 리스트와 같은 시퀀스는 0부터 순서대로 매겨진 정수 인덱스index를 키key로 사용하여 내부 정보를 참조한다. 그런데 정수 인덱스 대신, 담겨있는 내부 정보와 관련있는 문자열 또는 수를 키로 지정하여 사용하면 편리해진다. 예를 들어,

- 영어사전은 단어별로 뜻이 모여있는데, 단어 문자열을 인덱스로 하여 그에 해당하는 뜻을 찾을 수 있도록 하면 자연스럽다.
- 학생기록부는 학생의 신상인 이름, 생년월일, 학번, 소속학과 등을 모여있는 데이터구조이다. (`'홍길동', '980915', 2017001234, '소프트웨어학부'`) 같이 튜플로 모아서 사용할 수도 있겠지만, `'이름', '생년월일', '학번', '소속학과'`와 같은 문자열을 인덱스로 사용하여 참조할 수 있으면 편리할 것이다.

이 절에서는 Python으로 원소마다 인덱스를 지정할 수 있는 데이터구조인 사전을 어떻게 만들고 사용하는지 공부해보자.

사전 만들기

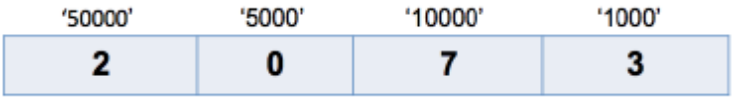
사전Dictionary은 원소를 모아놓은 데이터 구조인데 각 원소마다 인덱스 역할을 하는 키key를 배정하여 구별한다. 사전의 원소는 키와 표현식의 쌍으로 〈키〉 : 〈표현식〉으로 표현하며, 중괄호brace로 둘러싸고 쉼표comma로 구분하여 다음과 같이 표현한다.

{ 〈키〉 : 〈표현식〉, …, 〈키〉 : 〈표현식〉 }

리스트, 튜플과 같은 시퀀스는 0부터 시작하여 순서대로 지정된 정수를 키로 사용하는데 비해서, 사전은 사용자가 키를 문자열 또는 정수로 배정한다. 사전은 배정한 키로 해당 원소를 찾을 수 있어서 리스트 또는 튜플과 달리 나열된 순서는 전혀 중요하지 않다. 예를 들어, 내 지갑에 들어있는 지폐의 장수를 종류별로 다음과 같이 사전으로 표현할 수 있다.

```
>>> cash = {'50000':2, '10000':7, '5000':0, '1000':3}
>>> cash
{'50000': 2, '5000': 0, '10000': 7, '1000': 3}
```

이 사전에서 키는 지폐의 종류를 나타내는 문자열이고, 원소는 지폐의 장수를 나타내는 정수이다. 즉, 50,000원 짜리 지폐 2장, 10,000원 짜리 지폐 7장, 5,000원 짜리 지폐 0장, 1,000원 짜리 지폐 3장이 지갑에 있음을 표현한 것이다. 이를 그림으로 다음과 같이 이해할 수 있다.



여느 시퀀스와 마찬가지 구조이지만 정수 인덱스 대신 배정한 문자열을 키로 하여 내부 구성원소를 구분한다. 그림을 잘 살펴보면 표현식에 나열한 순서와는 다름을 관찰할 수 있다. 이유는 사전을 내부적으로

표현하는 방식과 관계가 있는데 어짜피 순서는 중요하지 않으므로 문제 될 것 없다. 사전에서 키는 유일무이unique하도록 엄격히 제한하고 있다. 즉, 사전 내부에서 키를 중복 사용할 수 없다.

연습 : 자신의 학생신상 데이터를 실행기에서 사전으로 표현하여 확인해보자.

사전에서 값 꺼내기

지폐 별로 장 수를 알아보고 싶으면 다음과 같이 키로 꺼내볼 수 있다.

```
>>> cash['10000']
7
```

만약 사전에 없는 키로 꺼내려고 시도하는 경우, 다음과 같이 실행오류가 발생한다.

```
>>> cash['20000']
Traceback (most recent call last):
File "<pyshell#3>", line 1, in <module>
cash['20000']
KeyError: '20000'
```

실행오류를 근본적으로 방지하기 위해서 다음과 같은 형식으로 키가 사전에 있는지 미리 검사해야 한다. '20000' 키가 cash에 없어서 if 문의 몸체를 실행하지 않으므로 실행오류가 발생하지 않는다.

```
if '20000' in cash:
    cash['20000']
```

사전 고치기

새로운 종류의 지폐가 발행되어 사전에 추가하고 싶으면 다음과 같이 한다.

```
>>> cash
{'5000': 0, '50000': 2, '1000': 3, '10000': 7}
>>> cash['100000'] = 3
>>> cash
{'5000': 0, '100000': 3, '50000': 2, '1000': 3, '10000': 7}
```

특정 지폐의 장 수를 고치려면 해당 키로 새로운 값을 지정하면 된다.

```
>>> cash['5000'] = 1
>>> cash
{'5000': 1, '100000': 3, '50000': 2, '1000': 3, '10000': 7}
```

특정 키와 값을 통채로 지우려면 del 명령을 사용한다.

```
>>> del cash['1000']
>>> cash
{'5000': 1, '100000': 3, '50000': 2, '10000': 7}
```

만약 사전에 없는 키로 del 명령을 시도하면 실행오류가 발생한다.

```
>>> del cash['1000']
Traceback (most recent call last):
File "<pyshell#17>", line 1, in <module>
del cash['1000']
KeyError: '1000'
```

따라서 이 경우에도 실행오류를 방지하려면 사전에 삭제 대상이 있는지 확인해야 한다. '1000' 키가 cash에 없어서 if 문의 몸체는 실행하지 않으므로 실행오류가 발생하지 않는다.

```
if '1000' in cash:
    del cash['1000']
```

사전 전체를 훑어서 키를 하나씩 꺼내 쓰려면 다음과 같은 형식으로 for 문을 사용한다.

```
for key in cash:
    print(key, '짜리가', cash[key], '장 있음')
```

cash 변수가 갖고 있는 사전의 키를 하나씩 참고하여 사전 전체의 키와 원소를 모두 프린트한다.

키

사전에서 키 자체를 임의로 수정할 수 있다면 혼란을 야기할 수 있으므로, 고칠 수 없는 값만 쓰도록 제한하고 있다. 키로 사용할 수 있는 값은 수정불가능한 값인 문자열, 정수, 불값, 튜플 등이다. 정수를 다음과 같이 키로 사용할 수도 있다.

```
>>> cash = {50000:2, 10000:7, 5000:0, 1000:3}
>>> cash
{50000: 2, 10000: 7, 5000: 0, 1000: 3}
>>> cash[10000]
7
>>> cash[100000]
Traceback (most recent call last):
File "<pyshell#4>", line 1, in <module>
cash[100000]
KeyError: 100000
>>> if 100000 in cash :
cash[100000]
```

사전 메소드

표준 라이브러리에 사전을 관리하는 메소드 다양하게 마련되어 있다. 메소드를 호출하여 사용하는 방식은 다음과 같은데 리스크 메소드 호출방식과 같다.

〈사전〉.〈메소드이름〉(〈인수〉,…,〈인수〉)

연습 : 표준 라이브러리 '4.10. Mapping Types — dict'에서 아래의 사전관리용 메소드를 찾아서 의미를 이해하고 실행기에서 예제를 만들어 이해한 대로 작동하는지 확인해보자.

- get(key)
- keys()
- values()
- items()

2. 블랙잭 확장

이전에 만들었던 블랙잭 카드게임은 실행을 끝내면 기록이 모두 사라진다. 왜냐하면 프로그램의 변수에 저장되어 있는 정보는 프로그램의 실행을 종료하는 순간 모두 사라지기 때문이다. 몇 번 게임을 했는지, 그 중에 몇 번이나 이겼는지, 칩을 얼마나 땄는지 또는 잃었는지 기록해두면 좋겠다. 영구 저장할 수 있는 매체는 하드디스크, USB와 같은 외부 저장장치이다. 가장 원시적인 데이터 저장방법은 데이터를 외부 저장장치에 텍스트 파일의 형태로 저장해두는 것이다. 언제라도 데이터가 필요한 경우 그 파일에서 데이터를 가져와서 사용할 수 있다. 블랙잭 게임에 플레이어 별로 게임 횟수, 승리 횟수, 취득한 칩의 개수와 같은 게임 기록을 파일에 체계적으로 저장해두어, 게임을 종료한 후에도 기록이 영구히 남아있게 하고 추후에 다시 게임을 할 때 지속적으로 사용할 수 있도록 블랙잭 게임을 개선해보자.

추가 요구사항

- 게임을 시작하기 전에 지금까지 게임을 몇 번 하여 몇 번 이겼는지 누적 승률이 얼마인지 보여 준다. 누적승률은 이긴 횟수 나누기 게임 횟수를 백분율로 환산하여 표현한다. 비긴 경우 0.5회 이긴 걸로 간주한다. 게임에서 이기면 받게 되는 칩도 누적하여 기록해두고 게임을 시작하면서 칩이 몇 개 있는지도 알려 준다. 화면에 보여주는 형식은 다음과 같다.

```
You played 37 games and won 20.5 of them.
Your all-time winning percentage is 55.4 %
You have 5 chips.
-----
```

- 칩을 잃어서 부채가 있는 경우 다음과 같은 형식으로 알려 준다.

```
You have -3 chips.
```

- 게임이 끝나면 해당 세션 동안의 기록을 다음과 같은 형식으로 보여 준다.

```
-----
You played 5 games and won 4 of them
-----
All-time Top 5
doh : 135 chips
didi : 36 chips
hy : 23 chips
who : 3 chips
dr : 2 chips
```

텍스트 파일에 저장하는 데이터의 형식

다음의 정보를 텍스트 파일 `members.txt`에 멤버 1인당 1줄 씩 저장한다.

- 이름 (`name`)
- 게임시도 횟수 (`tries`)
- 이긴 횟수 (`wins`)
- 칩 보유 개수 (`chips`)

각 정보는 사이에 쉼표를 두어 구분한다. 예를 들어, 다음과 같은 형식으로 정보를 저장한다.

```
doh,993,550,35
didi,130,55,10
hy,35,18,2
dr,18,8,0
who,34,18,0
```

이제 이 텍스트 파일에서 데이터를 읽어들이고, 저장하는 함수를 공부해보자.

load_members()

텍스트 파일 "members.txt"에서 한 줄씩 읽어서 다음과 같이 이름을 키로하는 사전을 만들어 내준다

{"doh":(993,550,35),"didi":(130,55,10),"hy":(35,18,2),"dr":(18,8,0),"who":(34,18,0)}

파일은 읽기 용으로 열고 다 읽은 후 반드시 닫아야 한다.

```
1 def load_members():
2     file = open("members.txt", "r")
3     members = {}
4     for line in file:
5         name, tries, wins, chips = line.strip('\n').split(',')
6         members[name] = (int(tries), float(wins), int(chips))
7     file.close()
8     return members
```

store_members(members)

이름을 키로 하는 사전 members를 텍스트 파일 "members.txt"에 위와 같은 텍스트 파일 형식으로 쓴다. 이전에 이 파일에 저장되어 있던 정보는 모두 지워진다. 파일은 쓰기 용으로 열고 다 쓴 후 반드시 닫아야 한다.

```
1 def store_members(members):
2     file = open("members.txt", "w")
3     names = members.keys()
4     for name in names:
5         tries, wins, chips = members[name]
6         line = name + ',' + str(tries) + ',' + \
7                str(wins) + "," + str(chips) + '\n'
8         file.write(line)
9     file.close()
```

로그인 함수

login(members)

멤버 사전 members을 가지고 로그인과 최초 출력을 처리하고, 멤버의 정보를 튜플로 모아서 내주는 함수이다. 완성한 login 함수 코드는 다음과 같다.

```
 1 def login():
 2     """gets player's name and returns it (string)"""
 3     name = input("Enter your name : (4 letters max) ")
 4     while len(name) > 4:
 5         name = input("Enter your name : (4 letters max) ")
 6     members = load_members()
 7     if name in members.keys():
 8         tries = members[name][0]
 9         wins = members[name][1]
10         print("Your played", tries, "games and won", wins, "of them")
11         winrate = 100 * wins / tries if tries > 0 else 0
12         print("Your all-time winning rate is", "{0:.1f}".format(winrate), "%")
13         chips = members[name][2]
14         print("You have", chips, "chips.")
15         return name, tries, wins, chips
16     else:
17         members[name] = (0,0,0)
18         return name, 0, 0, 0
```

필요 지식: 0으로 나누기 오류 방지하기

줄 11의 나눗셈은 분모가 0이 될 수 있음을 염두에 두고 코딩하였다. 실행 중 나눗셈 연산에서 분모 값이 0이 되면 다음과 같이 실행오류 ZeroDivisionError가 발생한다.

```
>>> def divide(x,y): return x / y
>>> divide(3,0)
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    divide(3,0)
  File "<pyshell#3>", line 2, in divide
    return x / y
ZeroDivisionError: division by zero
```

따라서 나눗셈은 다음과 같이 분모가 0인지 미리 검사하도록 작성해야 실행 중 오류를 방지할 수 있다.

```
>>> def divide(x,y): return x/y if y > 0 else 0
>>> divide(3,0)
0
```

필요 지식: 실수 출력 포맷

줄 12에서 winrate의 값은 실수이다. 실수를 출력할 때 소수점 이하 자리수를 지정하여 문자열로 바꾸고 싶으면 format 메소드를 다음과 같이 사용한다.

```
>>> 0.246
0.246
>>> "{0:.2f}".format(0.246)
'0.25'
```

Top 5 보여주기

show_top5(members)

멤버 사전 members를 인수로 받아 칩의 보유개수가 가장 많은 순으로 다음과 같은 형식으로 보여주는 함수이다. 칩의 개수가 0 이하인 경우는 보여주지 않도록 한다.

```
———
All-time Top 5 based on the number of chips earned
1. doh : 35
2. didi : 10
3. hy : 2
```

이 프로시저는 다음과 같은 틀에 맞추어 작성할 수 있다.

```
def show_top5(members):
    print("-----")
    sorted_members = # 칩의 개수 역순으로 정렬
    print("All-time Top 5")
    # sorted_members[:5]의 원소를 차례대로 참고하여 보여주되 0이하는 무시한다.
```

필요 지식: 사전 정렬하기 — sorted 함수 사용

멤버 사전을 칩의 개수 역순으로 정렬해야 하는데, 라이브러리 함수 sorted를 사용하면 사전을 다양한 형태의 키 또는 원소 정보를 정렬된 리스트로 얻을 수 있다. 먼저 sorted 함수를 어떻게 활용할 수 있는지, 불후의 명반 사전 사례를 가지고 공부해보자.

```
>>> dict = {}
>>> dict["Pink Floyd"] = ("Dark Side of the Moon",1973)
>>> dict["The Beatles"] = ("Abbey Road",1969)
>>> dict["Neil Young"] = ("Harvest",1972)
>>> dict
{'Pink Floyd': ('Dark Side of the Moon', 1973), 'Neil Young': ('Harvest', 1972),
'The Beatles': ('Abbey Road', 1969)}
```

다음과 같이 사전을 인수로 sorted 함수를 호출하면 키를 기준으로 오름차순으로 정렬한 키의 리스트를 내준다.

```
>>> sorted(dict)
['Neil Young', 'Pink Floyd', 'The Beatles' ]
```

다음과 같이 dict.items()를 인수로 sorted 함수를 호출하면 키를 기준으로 오름차순으로 정렬한 키와 원소의 튜플 쌍의 리스트를 내준다.

```
>>> sorted(dict.items())
[('Neil Young', ('Harvest', 1972)), ('Pink Floyd', ('Dark Side of the Moon', 1973)),
('The Beatles', ('Abbey Road', 1969))]
```

정렬할 기준을 지정할 수도 있다. 음반의 발매년도를 기준으로 오름차순으로 정렬하고 싶으면 다음과 같이 key를 추가인수로 지정하면 된다. 여기서 lambda x: x[1][1] 함수 표현식이며, 앞 부분의 lambda x: 에서 x는 함수의 형식파라미터이며, 뒷 부분 x[1][1]은 함수의 몸체이다. 예를 들어, ('The Beatles', ('Abbey Road', 1969))가 인수로 이 함수에 전달되면, x[1]은 ('Abbey Road', 1969)이고, x[1][1]은 1969이다. 즉, key 속성은 정렬의 기준으로 삼을 데이터를 지정해주는 역할을 한다.

```
>>> sorted(dict.items(), key=lambda x: x[1][1])
[('The Beatles', ('Abbey Road', 1969)), ('Neil Young', ('Harvest', 1972)), ('Pink
Floyd', ('Dark Side of the Moon', 1973))]
```

역순인 내림차순으로 정렬하고 싶으면 다음과 같이 reverse 속성을 True로 지정해주면 된다.

```
>>> sorted(dict.items(),key=lambda x: x[1][1],reverse=True)
[('Pink Floyd', ('Dark Side of the Moon', 1973)), ('Neil Young', ('Harvest', 1972)),
('The Beatles', ('Abbey Road', 1969))]
```

완성한 show_top5 프로시저 코드는 다음과 같다.

```
 1 def show_top5():
 2     members = load_members()
 3     print('-----')
 4     sorted_members = sorted(members.items(),\
 5                             key=lambda x: x[1][2],\
 6                             reverse=True)
 7     print("All-time Top 5")
 8     rank = 1
 9     for member in sorted_members[:5]:
10         chips = member[1][2]
11         if chips <= 0:
12             break
13         print(rank, '.', member[0], ':', chips)
14         rank += 1
```

블랙잭 알고리즘 (확장)

A. 환영인사를 프린트 한다.

```
print("Welcome to SMaSH Casino!")
```

B. **members.txt 파일에서 멤버 데이터를 읽고 로그인 절차를 통해서 사용자이름, 게임시도 횟수, 이긴 횟수, 칩 보유개수, 전체멤버 사전 정보를 수집한다.**

C. 잘 섞은 카드 1벌을 준비한다.

```
deck = fresh_deck()
```

D. 손님이 원하면 다음을 반복한다.

1. 카드를 1장씩 손님, 딜러, 손님, 딜러 순으로 배분한다.

```
dealer = []
player = []
card, deck = hit(deck)   # 1장 뽑아서
player.append(card)      # 손님에게 주고
card, deck = hit(deck)   # 1장 뽑아서
dealer.append(card)      # 딜러에게 주고
card, deck = hit(deck)   # 1장 뽑아서
player.append(card)      # 손님에게 주고
card, deck = hit(deck)   # 1장 뽑아서
dealer.append(card)      # 딜러에게 준다.
```

2. 딜러의 첫 카드를 제외하고 모두 보여준다.

```
print("My cards are:")
print(" ", "****", "**")
print(" ", dealer[1]["suit"], dealer[1]["rank"])
```

3. 손님의 카드를 보여준다.

```
show_cards(player, "Your cards are:")
```

4. 손님과 딜러의 카드 두 장의 합을 각각 계산한다.

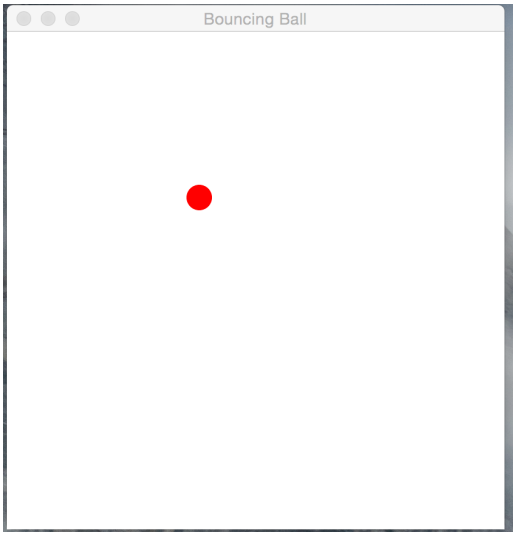
```
score_player = count_score(player)
score_dealer = count_score(player)
```

5. 손님의 카드 두 장의 합 score_player가 21이면 블랙잭으로 손님이 이긴다. 점수 chips를 2 만큼 더한다.
6. 손님의 카드 합이 21이 넘지 않는 한 손님이 원하면 카드를 더 준다. 21이 넘으면 손님이 버스트가 되어 딜러가 이기고 점수를 1 뺀다. A는 1 또는 11을 유리한 쪽으로 사용할 수 있어야 한다.
7. 손님이 21이 넘지 않았으면, 딜러의 카드 합을 계산하여 16 이하이면 16이 넘을때까지 무조건 카드를 받는다.
8. 딜러가 21이 넘으면 딜러가 버스트가 되어 손님이 이기고 1점을 더한다.
9. 둘 다 21이 넘지 않으면 합이 큰 쪽이 이긴다. 손님이 이기면 1점을 더하고, 딜러가 이기면 1점을 빼고, 비기면 점수 변동은 없다.
10. 더 할지 손님에게 물어봐서 그만하길 원하면 끝낸다.

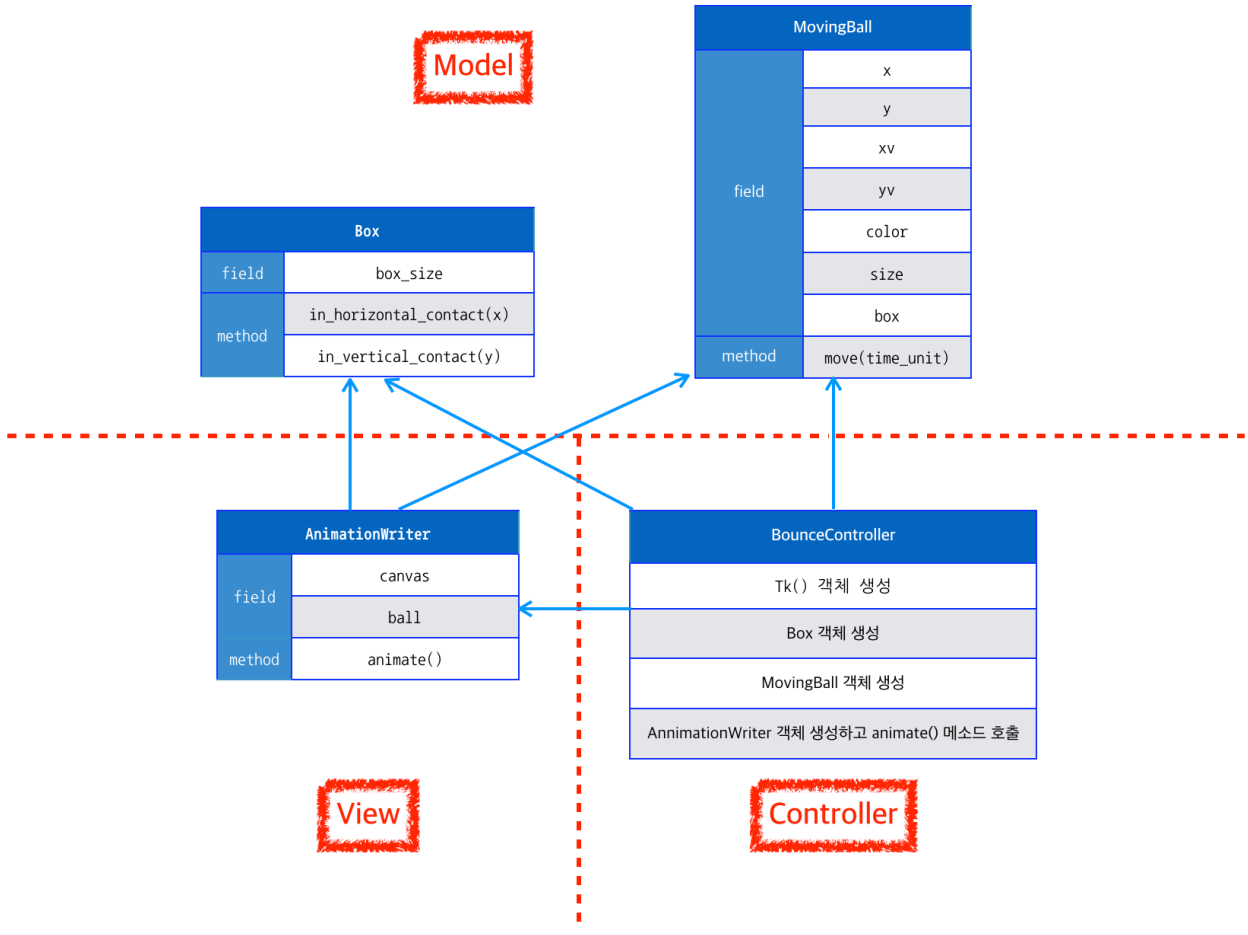
11. 게임이 진행되는 동안 승패 횟수와 칩의 획득 개수를 추적하여, 게임이 끝난 뒤 결과를 멤버 사전에 적용하여 수정하고, **members.txt** 파일에 저장한다.
12. 해당 세션의 게임 결과를 다음과 같이 요약하여 보여준다.
    You played 21 games and won 11 of them.
13. 지금까지의 칩 최다 보유 멤버 5명을 보여준다.
    show_top5(members)

객체지향 프로그래밍 실습

사례 1 : 공 튀기기 애니메이션



클래스 다이어그램



클래스 세부 명세서

Box		
field	box_size	정사각형 놀이터 변의 길이
method	in_horizontal_contact(x)	x 축의 범위 안에 있으면 True, 그렇지 않으면 False를 내줌
	in_vertical_contact(y)	y 축의 범위 안에 있으면 True, 그렇지 않으면 False를 내줌

MovingBall		
field	x	공의 위치 x 좌표
	y	공의 위치 y 좌표
	xv	공이 x 축으로 움직이는 속도
	yv	공이 y 축으로 움직이는 속도
	color	공의 색깔
	size	공의 반지름
	box	공이 굴러다니는 정사각형 놀이터 객체
method	move(time_unit)	time_unit 만큼 위치 변화

AnimationWriter		
field	canvas	Canvas 객체
	ball	공 객체
method	animate()	ball을 box 안에서 움직이게 하고 원모양으로 그림

BounceController
놀이터의 크기, 공의 크기와 색깔, 공의 움직임 방향 및 속도 지정
Tk() 객체 생성 (title은 Bouncing Ball)
Box 객체 생성
MovingBall 객체 생성
AnnimationWriter 객체 생성하고 animate() 메소드 호출

실습 : 공 튀기기 애니메이션 (변형)

1. 상자 정 가운데 정사각형의 파란색 기둥(stake)을 하나 만들고, 공이 이 기둥에 부딪치면 튕겨나가게 애니메이션 애플리케이션을 변형하시오. 기둥은 한 변이 공의 지름의 2배가 되는 정사각형으로 한다.

2. 1번이 완성되면, 녹색 공을 하나 더 만들어 돌아다니게 만든다. 시작 지점과 방향은 빨간 공과 다르게 잡되 속도는 같다.

3. 2번이 완성되면, 빨간 공과 녹색 공이 서로 부딪치면 둘다 진행 방향을 180도 바꿔서 계속 움직이도록 한다.

