[COM1002]
프로그래밍1

카드게임 블랙잭

#09. 프로젝트 기반 학습 II

김현하

한양대학교 ERICA 소프트웨어학부

2021.11.23.
2021년도 2학기

블랙잭

블랙잭

- 블랙잭blackjack
  - 소유한 카드 점수의 합이 21을 넘지 않는 한도 내에서 딜러와 겨루어 점수가 높으면 이기는 카드 게임
  - 놀이카드playing card는 다음과 같이 총 52장의 카드로 구성
    - 네가지 무늬suit : 스페이드spade♠, 하트heart♥, 클럽club♣, 다이아몬드diamond♦
    - 13가지 끗수rank : A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K

블랙잭의 규칙

- 점수
  - A에이스 Ace 카드는 1점 또는 11점 중 유리한 방향으로 선택할 수 있다.
  - J, Q, K 카드의 점수는 각각 10점이다.
  - 나머지 카드 2~10의 점수는 액면대로 매긴다.
- 게임 규칙
  - 딜러가 카드 두 장을 손님과 자신에게 각각 한장씩 교대로 나누어 준다.
  - 딜러의 첫째 카드는 공개하지 않고, 나머지 카드는 모두 공개한다.
  - 처음 받은 카드 두장이 A와 10, J, Q, K 중 하나로 구성되어 합이 21이 되면 **블랙잭**으로 손님이 무조건 이긴다.
  - 손님은 먼저 받은 카드 두 장의 합이 21에 못 미치면, 카드의 합이 21을 초과하지 않는 한 원하는 만큼 카드를 한장씩 더 받을 수 있다. 만약 21을 초과하면 버스트bust가 되어 무조건 진다.
  - 딜러는 카드의 합이 16 이하이면 카드를 무조건 더 받아야 하고 17 이상이면 더이상 받을 수 없다.
  - 딜러 카드의 합과 손님 카드의 합이 같으면 비긴다.

목차

- 집합
- 카드게임 API 라이브러리 모듈
- 프로그래밍 프로젝트 1단계 : 블랙잭
- 딕셔너리
- 프로그래밍 프로젝트 2단계 : 블랙잭 기능 확장: 멤버십과 게임 기록 관리

집합

집합

- 집합set
  - 순서와 중복 없이 데이터 값을 모아놓을 수 있는 컬렉션collection 데이터 구조
  - 수정 가능함 (단, 모든 원소는 수정 불가능한 값만 가능)
  - 중괄호{} 사이에 원소를 , 로 구분해서 나열

```
>>> { 7 + 8, 8.15, "seven" }
{8.15, 'seven', 15}
>>> { 7 + 8, 8.15, "seven" } == { 8.15, "seven", 6 + 9 }
True
```

집합

- 집합set
  - 순서와 중복 없이 데이터 값을 모아놓을 수 있는 컬렉션collection 데이터 구조
  - 수정 가능함 (단, 모든 원소는 수정 불가능한 값만 가능)
  - 중괄호{} 사이에 원소를 , 로 구분해서 나열

집합 메소드	의미
s.add(n)	집합 **s** 에 원소 **n**을 추가한다.
s.remove(n)	집합 **s** 에서 원소 **n**을 제거한다. 집합 **s** 에 원소 **n**이 없으면 **KeyError** 오류가 발생한다.
set()	빈집합을 생성해서 리턴한다. ({} 는 빈 딕셔너리)
set(*iterable*)	***iterable*** 의 내용을 담은 집합을 생성해서 리턴한다. ***iterable*** 은 시퀀스나 집합과 같이 반복 가능한 객체를 의미한다.

```
>>> s = {1,2}
>>> s
{1,2}
>>> s.add(3)
>>> s
{1, 2, 3}
>>> s.remove(1)
>>> s
{2, 3}
>>> s.remove(2)
{3}
>>> s.remove(3)
>>> s
set()
>>> s.remove(0)
...
KeyError: 0
```

집합

집합 메소드	연산자	의미
s.update(*iterable*)		집합 **s** 에 ***iterable***의 데이터를 추가한다.
s1.union(s2)	**s1 | s2**	집합 **s1** 과 **s2**의 합집합을 리턴한다. (**s1**과 **s2**의 내용은 바뀌지 않음)
s1.intersection(s2)	**s1 & s2**	집합 **s1** 과 **s2**의 교집합을 리턴한다. (**s1**과 **s2**의 내용은 바뀌지 않음)
s1.difference(s2)	**s1 - s2**	집합 **s1** 과 **s2**의 차집합을 리턴한다. (**s1**과 **s2**의 내용은 바뀌지 않음)
s1.symmetric_difference(s2)	**s1 ^ s2**	집합 **s1** 과 **s2**의 대칭차집합을 리턴한다. (**s1**과 **s2**의 내용은 바뀌지 않음)

```
>>> s1 = {1,2}
>>> s2 = {2,3}
```

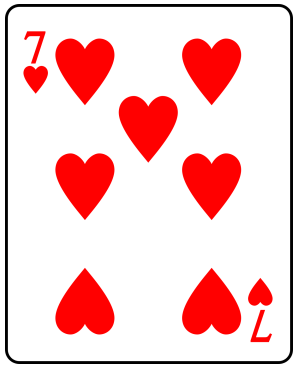
```
>>> s1 | s2
{1, 2, 3}
>>> s1 & s2
{2}
>>> s1 - s2
{1}
>>> s1 ^ s2
{1, 3}
```

```
>>> s1
{1, 2}
>>> s2
{2, 3}
```

카드게임 API
라이브러리 모듈

카드의 표현

- 무늬 : 네가지 문자열로 표현
  - `"Spade"`, `"Club"`, `"Heart"`, `"Diamond"`
- 끗수 : J, Q, K, A 는 문자열로, 나머지 숫자는 정수로 표현
- 카드 한장 : 무늬와 끗수의 튜플로 표현



⇒ `("Heart", 7)`

- 카드 덱 : 카드의 리스트로 표현

API 라이브러리

- API Application Programming Interface 라이브러리
  - 카드 1벌 만들어 무작위로 섞기
  - 카드 덱에서 카드 한 장 뽑아주기
  - 카드 점수 계산하기
  - 카드 프린트해서 보여주기
  - 카드를 더 받을지 물어보기

카드 1벌 만들어 무작위로 섞기

실습 9.1 fresh_deck 함수 만들기

- 새로운 카드 1벌(52장)을 리스트로 만들고 무작위로 섞어서 리턴

```
1   import random
2   def fresh_deck():
3       suits = { "Spade", "Heart", "Diamond", "Club" }
4       ranks = { 2, 3, 4, 5, 6, 7, 8, 9, 10, "J", "K", "Q", "A" }
5       deck = []
6       # Write your nested for-loop creating 52 cards in deck.
7
8
9
10      # shuffle deck using random.shuffle function
11      return deck
```

카드 덱에서 카드 한장 뽑아주기

- 카드 덱에서 맨 앞에 있는 카드를 선택, 선택한 카드는 카드 덱에서 제거
- 카드가 모두 소진된 경우 `fresh_deck()` 함수를 다시 호출해서 카드 1벌을 새로 만들어 게임을 지속

```
1  def hit(deck):
2      if deck == []:
3          deck = fresh_deck()
4      return (deck[0], deck[1:])
```

카드 점수 계산하기

실습 9.2 count_score 함수 만들기

```
1   def count_score(cards):
2       score = 0
3       number_of_ace = 0
4       for card in cards:
5           rank = card[1]
6           # accumulate score (counts A as 11)
7
8
9
10
11
12
13      # adjust score if score is over 21 and there is A
14      # there may be two or more A's
15
16      return score
```

카드 프린트해서 보여주기

- 인수 : 카드 리스트 cards와 문자열 message
- 메시지와 카드를 한줄에 하나씩 차례로 출력

```
1  def show_cards(cards, message):
2      print(message)
3      for card in cards:
4          print(' ', card[0], card[1])
```

카드를 더 받을지 물어보기

- 사용자에게 키보드로 y 또는 n을 입력받음
- 사용자입력이 y면 True를 n이면 False를 리턴

```
1  def more(message):
2      answer = input(message)
3      while not (answer in ['y', 'n']):
4          answer = input(message)
5      return answer == 'y'
```

모듈 cardgame

- 구현한 함수를 cardgame.py 파일로 저장 = 모듈 **cardgame** 생성
- `import` 키워드 뒤에 모듈이름을 붙여서 사용하면 해당 모듈에 정의된 함수(혹은 변수)를 사용가능
  - **cardgame.**<함수이름> 형식으로 사용
  - `from` 을 사용해서 일부 혹은 전체를 가져와서 사용

```
import cardgame
deck = cardgame.fresh_deck()
```

```
from cardgame import fresh_deck
deck = fresh_deck
```

```
from cardgame import *
deck = fresh_deck
```

```
from cardgame import fresh_deck, hit
deck = fresh_deck
a_card, left_deck = hit(deck)
```

프로그래밍 프로젝트 1단계
블랙잭

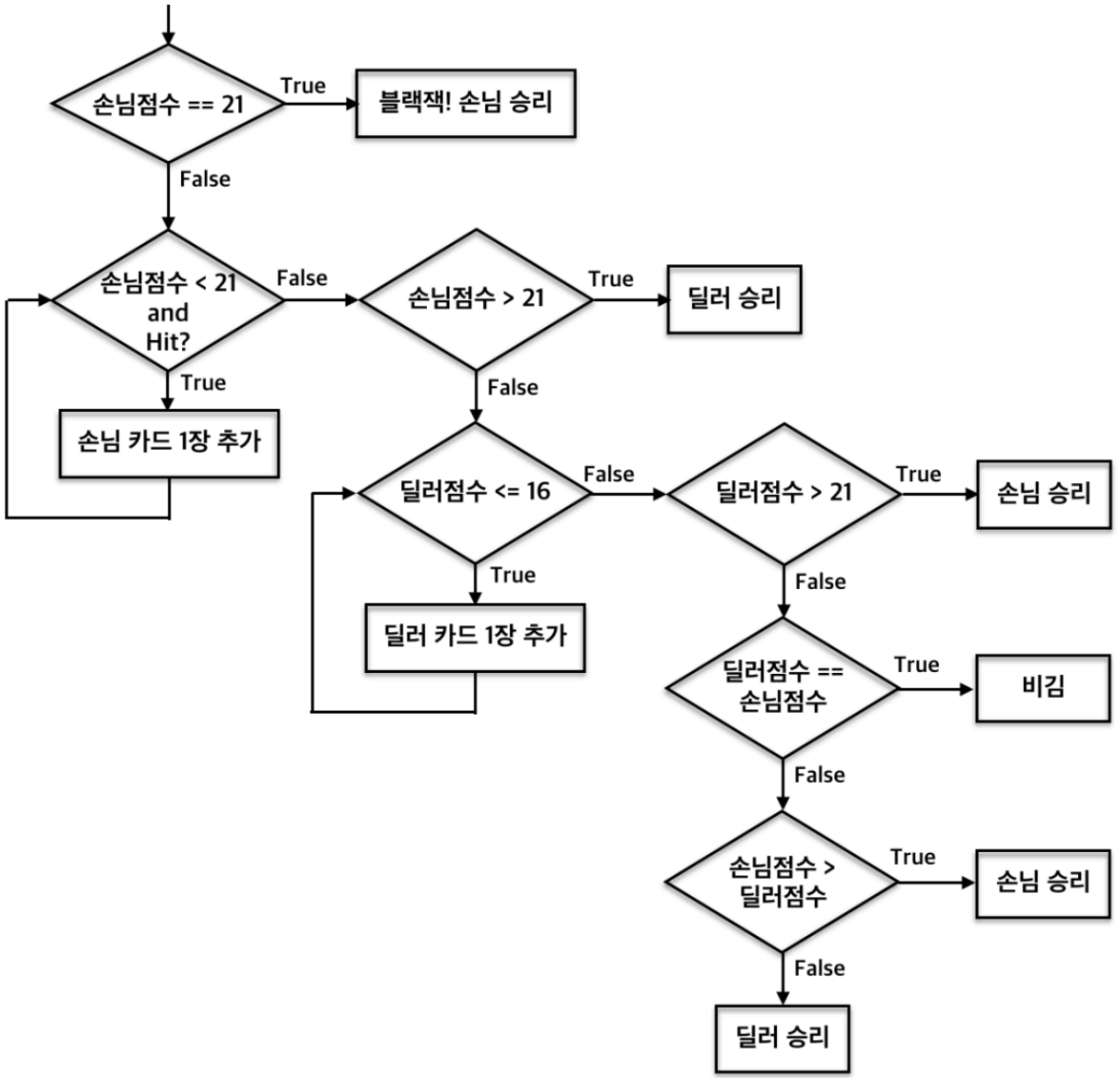
블랙잭: 요구사항

- 게임을 시작하면 환영 메시지를 (교재에 있는 대로) 실행창에 프린트한다.
- 카드는 1벌(52장)을 잘 섞어서 사용한다. 다 쓰면 1벌을 새로 만들어 다시 잘 섞어서 사용한다.
- 카드는 처음 2장씩 나누어 주되, 손님, 딜러, 손님, 딜러 순으로 나누어주고, 딜러의 첫 카드는 감춘다.
- 카드는 Spade J와 같은 형식으로 한 줄에 한 장씩 텍스트로 실행창에 프린트한다. 감추는 카드는 **** ** 로 프린트한다.
- 딜러의 카드를 (교재에 있는 형식에 맞춰) 먼저 보여 주고, 다음에 손님의 카드를 (교재에 있는 형식에 맞춰) 보여 준다.
- 손님은 점수가 21점 미만인 경우 카드를 추가로 요청할 수 있다.
- 손님에게 추가 카드를 원하는지 물어봐야 하며, 손님은 y(예) 또는 n(아니오)로 의사를 표시한다. (형식은 교재 참조)
- 받은 카드는 바로 보여준다. (형식은 교재 참조)
- 딜러는 카드의 합이 16점 이하이면 카드를 추가로 한 장 무조건 받아야 하며, 16점을 넘으면 더 이상 받을 수 없다.
- A(에이스)는 1점 또는 11점 중 하나를 유리한 쪽으로 선택할 수 있어야 한다. (자동으로 계산)
- 손님이 이기면 적절한 메시지를 프린트하고 다음 라운드로 넘어간다. 첫 두장의 합이 21이어서 블랙잭으로 이긴 경우와 딜러가 버스트bust인 경우, 메시지에 해당 정보를 추가해야 한다. (교재 참조)
- 딜러가 이기면 적절한 메시지를 프린트하는데, 손님이 버스트인 경우 해당 정보를 추가해야 한다. (교재 참조)
- 비긴 경우, 적절한 메시지를 출력한다. (교재 참조)
- 손님이 버스트로 진 경우를 제외하고 매 라운드가 종료할 때마다 딜러의 카드를 모두 보여준다.
- 손님이 소지한 칩의 개수를 매 라운드마다 알려줘야 한다. 칩은 최초 0개로 시작해서 이기면 1개를 획득하고 지면 1개를 잃는다. 블랙잭으로 이긴 겨우엔 추가로 1개를 획득한다. 딜러는 블랙잭으로 이겨도 보너스가 없다. (형식은 교재 참조)
- 매 라운드마다 게임을 계속할지 물어본다. 새 라운드를 시작할 경우엔 구분을 위한 내용을 프린트한다. (형식은 교재 참조)
- 게임을 마치면 이별 메시지를 프린트한다. (형색은 교재 참조)

블랙잭: 알고리즘

1. 환영인사를 프린트한다.
2. 잘 섞은 카드 1벌을 준비한다.
3. 칩의 개수를 0으로 초기화한다.
4. 손님이 원하는 한, 단계 5~14를 반복한다.
5. 카드를 1장씩 손님, 딜러, 손님, 딜러 순으로 배분한다.
6. 딜러의 첫 카드를 제외하고 모두 보여준다.
7. 손님의 카드를 보여준다.
8. 손님과 딜러의 카드 두 장의 합을 각각 계산한다.
9. 손님 카드의 합 score_player가 21이면 블랙잭으로 손님이 이긴다. chips에 2를 더한다.
10. 손님의 카드 합이 21을 넘지 않는 한 손님이 원하면 카드를 더 준다. 21을 넘으면 손님이 버스트되어 딜러가 이기고 chips에서 1을 뺀다.
11. 손님의 카드 합이 21을 넘지 않았으면, 딜러의 차례이다. 딜러의 카드 합을 계산하여 16 이하이면 16이 넘을 때까지 무조건 카드를 더 받고, 17 이상이 되는 순간 더 받지 않는다.
12. 딜러의 카드 합이 21을 넘으면 딜러가 버스트 되어 손님이 이기고 chips에 1을 더한다.
13. 둘 다 21이 넘지 않으면 큰 쪽이 이긴다. 손님이 이기면 chips에 1을 더하고, 딜러가 이기면 chips에서 1을 빼고, 비기면 변동 없다.
14. 더 할지 손님에게 물어봐서 그만하길 원하면 끝낸다.

블랙잭: 알고리즘


손님점수 == 21
True
블랙잭! 손님 승리
False
손님점수 < 21 and Hit?
False
손님점수 > 21
True
딜러 승리
True
손님 카드 1장 추가
False
딜러점수 <= 16
False
딜러점수 > 21
True
손님 승리
True
딜러 카드 1장 추가
False
딜러점수 == 손님점수
True
비김
False
손님점수 > 딜러점수
True
손님 승리
False
딜러 승리

딕셔너리

딕셔너리

- 딕셔너리dict
  - 키와 값의 쌍을 모아 놓은 것
  - 중괄호{} 사이에 키와 식을 다음과 같이 , 로 구분해서 나열

    { <키> : <식>, ... , <키> : <식> }

  - 시퀀스는 (저절로 매겨진) 정수 인덱스를 키로 사용하지만 딕셔너리는 자유자재로 키를 사용 가능
  - 수정불가능immutable한 값만 키로 사용 가능
    (집합은 <식> 없이 <키>로만 구성한 딕셔너리로 볼 수 있음)

딕셔너리

- 생성과 검색

```
>>> me = { "이름": "조상만", "생년": 2002, "이메일": "chosm@mail.com" }
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.com'}
>>> me["이름"]
'조상만'
>>> me["생년월일"]
...
KeyError: '생년월일'
```

딕셔너리

- 빈 딕셔너리

```
>>> me = { "이름": "조상만", "생년": 2002, "이메일": "chosm@mail.com" }
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.com'}
>>> me["이름"]
'조상만'
>>> me["생년월일"]
...
KeyError: '생년월일'
```

```
>>> dict()
{}
>>> {}
{}
>>> type(dict())
<class 'dict'>
>>> type({})
<class 'dict'>
```

딕셔너리

- 안전하게 검색

```
>>> me = { "이름": "조상만", "생년": 2002, "이메일": "chosm@mail.com" }
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.com'}
>>> me["이름"]
'조상만'
>>> me["생년월일"]
...
KeyError: '생년월일'
```

```
>>> me.get("이름")
'조상만'
>>> me.get("취미")
>>> me.get("취미", "취미가 없네요")
'취미가 없네요.'
```

딕셔너리

- 수정과 추가

```
>>> me = { "이름": "조상만", "생년": 2002, "이메일": "chosm@mail.com" }
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.com'}
```

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.com'}
>>> me["이메일"] = "chosm@mail.co.kr"
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
>>> me["취미"] = ["음악감상", "독서", "웹툰"]
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr',
'취미': ['음악감상']}
```

딕셔너리

- 삭제

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr',
'취미': ['음악감상']}
```

```
>>> del me['취미']
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
>>> del me['취미']
...
KeyError: '취미'
```

딕셔너리

- 안전하게 삭제

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
```

```
>>> del me['취미']
...
KeyError: '취미'
>>> if "취미" in me:
        del me['취미']
```

딕셔너리

- 딕셔너리의 키와 값의 타입
  - 키 : 수정불가능immutable한 모든 식
  - 값 : 모든 식

```
>>> x = 4
>>> { "1":1, 2:"two", "three": 1+2, x: 4.0 }
{'1':1, 2: 'two', 'three': 3, '4': 4.0}
```

```
>>> { (1,2):"(1,2)", "(2,3)": [2,3], range(0,3): {3,4} }
{(1,2): '(1,2)', '(2,3)': [2, 3], range(0, 3): {3, 4}}
>>> { (1,2):"(1,2)", "(2,3)": [2,3], [3,4]: {"3":3, 4:"4"} } # ???
```

딕셔너리

- 딕셔너리의 키와 값의 타입
  - 키 : 수정불가능immutable한 모든 식
  - 값 : 모든 식

```
>>> x = 4
>>> { "1":1, 2:"two", "three": 1+2, x: 4.0 }
{'1':1, 2: 'two', 'three': 3, '4': 4.0}
```

```
>>> { (1,2):"(1,2)", "(2,3)": [2,3], range(0,3): {3,4} }
{(1,2): '(1,2)', '(2,3)': [2, 3], range(0, 3): {3, 4}}
>>> { (1,2):"(1,2)", "(2,3)": [2,3], [3,4]: {"3":3, 4:"4"} } # ???
```

```
>>> { (1,2):"(1,2)", "(2,3)": [2,3], [3,4]: {"3":3, 4:"4"} }
...
TypeError: unhashable type: 'list'
>>> { (1,2):"(1,2)", "(2,3)": [2,3], {3,4}: {"3":3, 4:"4"} }
...
TypeError: unhashable type: 'set'
```

딕셔너리

- 딕셔너리의 키와 값의 타입
  - 키 : 수정불가능immutable한 모든 식

종류	타입	설명	수정가능여부	기타
스칼라scalar 타입	int	정수	수정불가능	
	float	실수	수정불가능	키로 사용할 경우, 실수 오차에 주의
	NoneType	None	수정불가능	
	bool	논리값	수정불가능	
시퀀스	list	리스트	수정가능	키로 사용 불가능
	str	문자열	수정불가능	
	tuple	튜플	수정불가능	내부 값도 수정불가능해야 키로 사용 가능
	range	정수범위	수정불가능	
집합set	set	집합	수정가능	키로 사용 불가능
	frozenset	집합	수정불가능	내부 값도 수정불가능해야 키로 사용 가능
매핑mapping	dict	딕셔너리	수정가능	키로 사용 불가능

딕셔너리

- 딕셔너리의 키와 값의 타입
  - 키 : 수정불가능immutable한 모든 식 : **정수 1과 실수 1.0은 같음에 주의!**
  - 값 : 모든 식

```
>>> numbers = { 1 : "one", 2 : "two" }
>>> numbers[1]
'one'
>>> numbers[1.0]
'one'
>>> numbers[1.0] = 'ONE'
>>> numbers[1]
'ONE'
```

딕셔너리

- 딕셔너리 훑기
  - 키, 값, 혹은 아이템 (키와 값)을 리스트로 모아서 `for` 루프로 훑기 가능

집합 메소드	의미
d.keys()	딕셔너리 **d** 의 키를 리스트로 모은 뷰 객체dict_keys를 리턴한다.
d.values()	딕셔너리 **d** 의 값을 리스트로 모은 뷰 객체dict_values를 리턴한다.
d.items()	딕셔너리 **d** 의 아이템을 리스트로 모은 뷰 객체dict_items를 리턴한다. 아이템은 키와 값의 튜플을 의미한다.

딕셔너리

- 딕셔너리 훑기
  - 키, 값, 혹은 아이템 (키와 값)을 리스트로 모아서 for 루프로 훑기 가능

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
>>> me.keys()
dict_keys(['이름', '생년', '이메일'])
>>> me.values()
dict_values(['조상만', 2002, 'chosm@mail.co.kr'])
>>> me.items()
dict_items([('이름', '조상만'), ('생년', 2002), ('이메일', 'chosm@mail.co.kr')])
```

딕셔너리

- 딕셔너리 훑기
  - 키, 값, 혹은 아이템 (키와 값)을 리스트로 모아서 for 루프로 훑기 가능

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
>>> for x in me.keys() : print(x)

이름
생년
이메일
>>> for x in me.values() : print(x)

조상만
2002
chosm@mail.co.kr
>>> for x in me.items() : print(x)

('이름', '조상만')
('생년', 2002)
('이메일', 'chosm@mail.co.kr')
```

딕셔너리

- 딕셔너리 훑기
  - 키, 값, 혹은 아이템 (키와 값)을 리스트로 모아서 for 루프로 훑기 가능

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
>>> my_items = me.items()
>>> for (key, value) in my_items : print(key, ":\t", value)

이름 :      조상만
생년 :      2002
이메일 :    chosm@mail.co.kr
```

딕셔너리

- 딕셔너리 훑기
  - 키, 값, 혹은 아이템 (키와 값)을 리스트로 모아서 for 루프로 훑기 가능
  - 한번 만들어 둔 뷰 객체는 딕셔너리와 연동 (사용할 때 마다 바뀐 내용이 반영)

```
>>> me
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr'}
>>> me['키'] = 181
{'이름': '조상만', '생년': 2002, '이메일': 'chosm@mail.co.kr', '키', 181}
>>> for (key, value) in my_items : print(key, ":\t", value)

이름 :      조상만
생년 :      2002
이메일 :    chosm@mail.co.kr
키 :        181
```

프로그래밍 프로젝트 2단계
블랙잭 확장

블랙잭: 추가 요구사항

- 게임을 시작하면 사용자의 아이디와 비밀번호를 확인하고 게임을 진행한다.
  - 신규 사용자는 최초로 입력한 아이디와 비밀번호로 등록된다.
  - 기존 사용자는 비밀번호가 맞는지 확인한다.
  - 입력 확인 조건(이름의 길이 제한)을 만족하지 못하거나 비밀번호가 등록된 것과 다를 경우, 다시 입력받는다.
- 게임에 들어가기 전에 지금까지 몇 번 게임을 하여 몇 번 이겼는지와 누적 승률(이긴횟수/게임횟수를 백분율로 표시, 비긴 게임은 0.5회 이긴 것으로 간주)이 얼마인지를 보여준다.
- 게임에서 이기면 받는 칩도 누적하여 기록해둔다.
- 게임을 시작하면 칩 보유 개수를 적절한 형식(교재 참조)으로 알려준다.
- 게임이 끝나면 해당 세션 동안의 기록과 갱신된 랭킹을 적절한 형식(교재 참조)으로 보여준다.

게임기록 저장하기

- CSVcomma-separated values 파일에 저장하는 게임 기록 정보의 형식
  - 아이디name
  - 비밀번호password
  - 게임시도 횟수tries
  - 이긴 횟수wins
  - 칩 보유 개수chips

파일에서 정보 읽기

실습 9.4 파일에서 게임 기록 정보 읽기 함수

- 같은 폴더에 있는 CSV 파일 members.csv 에서 텍스트 전체를 읽은 후, 아이디가 키인 멤버 딕셔너리를 만들어 리턴

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

문자열 메소드	실행 의미
s.strip(chars)	문자열 **s** 의 양쪽 끝에서 문자열 **chars**에 있는 문자를 모두 제거하고 리턴한다. 인수가 비어있으면, 빈칸을 모두 제거하고 리턴한다.
s.split(sep)	구분 문자열 **sep**을 중심으로 문자열 **s**를 분리하여 리스트로 모아 리턴한다. 인수가 비어있으면, 빈칸을 구분 문자열로 하여 분리한다.

파일에 정보 쓰기

실습 9.5 파일에 게임 기록 정보 쓰기 함수

- 멤버가 게임을 진행해서 갱신한 게임기록 딕셔너리를 CSV 형식으로 `members.csv` 에 저장

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

로그인하기

실습 9.6 로그인 함수

```
1  def login(members):
2      username = input("Enter your name: (4 letters max) ")
3      while len(username) > 4:
4          username = input("Enter your name: (4 letters max) ")
5      trypasswd = input("Enter your password: ")
6      if None: # username is in the key of members
7          if None: # trypasswd is the same as username's password
8
9              # Show the number of games played and the number of wins
10             # Example: "You played 101 games and won 54 of them."
11
12             # Show the winning percentage in percent
13             # Example: "Your all-time winning percentage is 53.5%"
14
15             # Show the number of chips the player has
16             # Example : if the number is 5, "You have 5 chips."
17             # Example : if the number is -5, "You owe 5 chips."
18             return username, tries, wins, chips, members
19         else:
20             return login(members)
21     else:
22         # Add username to members dictionary.
23         return username, 0, 0, 0, members
```

코딩 가이드 1: 나누기 0 오류 방지

- 신규 가입자는 게임시도 횟수가 0이므로, 승률을 계산하려 하면 나누기 0 오류가 발생
- 식의 옆에 적절한 조건(`if y >0`)과 대안(`else 0`)을 달아서 오류 예방

```
1  def safe_divide(x, y):
2      return x/y if y > 0 else 0
```

```
1  def safe_divide(x, y):
2      if y > 0:
3          return x/y
4      else:
5          return 0
```

코딩 가이드 2: 프린트 포맷

- 소수점 이하의 자리 수를 지정하여 프린트

```
>>> 0.246
0.246
>>> "{0:.2f}".format(0.246)
'0.25'
>>> "{0:.1f}".format(0.246)
'0.2'
>>> "{2:.2f} _ {0:.1f}".format(0.246, 0.264, 0.462)
'0.46 _ 0.2'
>>> f"{0.462:.2f} _ {0.246:.1f}"
'0.46 _ 0.2'
>>> "Hello {0}, Good {1}!".format("there", "morning")
'Hello there, Good morning!'
>>> f"Hello {'there'}, Good {'morning'}!"
'Hello there, Good morning!'
```

https://docs.python.org/ko/3/library/stdtypes.html#str.format

https://docs.python.org/ko/3/library/string.html#formatstrings

랭킹 보여주기

실습 9.7 Top 5 보여주기 함수

```
1  def show_top5(members):
2      print("---")
3      sorted_members = None # Sort the number of chips in reverse order.
4      print("All-time Top 5 based on the number of chips earned")
5      # Show the elements of sorted_members[:5] in order.
6      # Disregard the record if the number of chips is 0 or less
```

코딩 가이드: 딕셔너리 정렬하기 - sorted 함수 사용법

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

```
>>> sorted(albums)
['Neil Young', 'Pink Floyd', 'The Beatles']
>>> sorted(albums.items())
[('Neil Young', ('Harvest', 1972)), ('Pink Floyd', ('Dark Side of the Moon', 1973)), ('The
Beatles', ('Abbey Road', 1969))]
>>> sorted(albums.items(), key=lambda x: x[0])
[('Neil Young', ('Harvest', 1972)), ('Pink Floyd', ('Dark Side of the Moon', 1973)), ('The
Beatles', ('Abbey Road', 1969))]
>>> sorted(albums.items(), key=lambda x: x[1][1])
[('The Beatles', ('Abbey Road', 1969)), ('Neil Young', ('Harvest', 1972)), ('Pink Floyd',
('Dark Side of the Moon', 1973))]
>>> sorted(albums.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))]
```

블랙잭: 알고리즘

1. 환영인사를 프린트한다.
2. members.csv 파일에서 멤버 기록을 읽고 로그인 절차를 통해 사용자이름, 게임시도 횟수, 이긴 횟수, 칩 보유개수, 전체 멤버 딕셔너리 정보를 수집한다.
3. 잘 섞은 카드 1벌을 준비한다.

   ~~칩의 개수를 0으로 초기화한다.~~
4. 손님이 원하는 한, 단계 5~14를 반복한다.
5. 카드를 1장씩 손님, 딜러, 손님, 딜러 순으로 배분한다.
6. 딜러의 첫 카드를 제외하고 모두 보여준다.
7. 손님의 카드를 보여준다.
8. 손님과 딜러의 카드 두 장의 합을 각각 계산한다.
9. 손님 카드의 합 score_player가 21이면 블랙잭으로 손님이 이긴다. chips에 2를 더한다.
10. 손님의 카드 합이 21을 넘지 않는 한 손님이 원하면 카드를 더 준다. 21을 넘으면 손님이 버스트되어 딜러가 이기고 chips에서 1을 뺀다.
11. 손님의 카드 합이 21을 넘지 않았으면, 딜러의 차례이다. 딜러의 카드 합을 계산하여 16 이하이면 16이 넘을 때까지 무조건 카드를 더 받고, 17 이상이 되는 순간 더 받지 않는다.
12. 딜러의 카드 합이 21을 넘으면 딜러가 버스트 되어 손님이 이기고 chips에 1을 더한다.
13. 둘 다 21이 넘지 않으면 큰 쪽이 이긴다. 손님이 이기면 chips에 1을 더하고, 딜러가 이기면 chips에서 1을 빼고, 비기면 변동 없다.
14. 더 할지 손님에게 물어봐서 그만하길 원하면 15~17 단계를 수행한 다음 끝낸다.
15. 게임이 진행되는 동안 승패 횟수와 칩의 획득 개수를 추적하여, 게임이 끝난 뒤 결과를 멤버 딕셔너리에 적용하여 수정하고, members.csv 파일에 저장한다.
16. 해당 세션의 게임 결과를 다음과 같이 요약하여 보여준다.
17. 지금까지의 칩 최다 보유 멤버를 5명까지 보여준다.