

[CSE1017]

프로그래밍기초

카드게임 블랙잭

# #09. 프로젝트 기반 학습 II

김현하

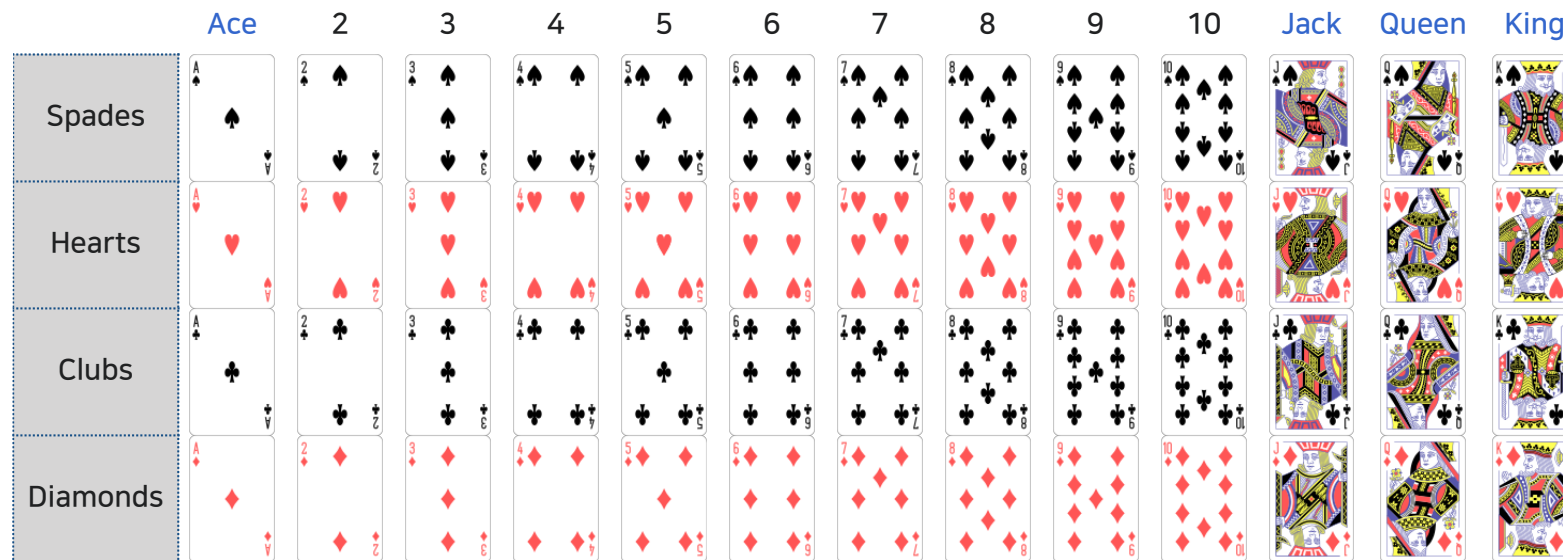
한양대학교 ERICA 소프트웨어학부

2024년도 1학기

# 블랙잭

# 블랙잭

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





























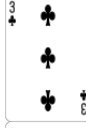
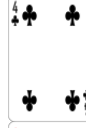
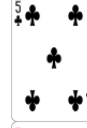
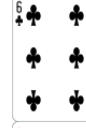






















출처 : [https://en.wikipedia.org/wiki/Standard\\_52-card\\_deck](https://en.wikipedia.org/wiki/Standard_52-card_deck)

# 블랙잭의 규칙

- 점수

- A에이스 Ace 카드는 1점 또는 11점 중 유리한 대로 선택할 수 있다.
- J, Q, K 카드의 점수는 각각 10점이다.
- 나머지 카드 2~10의 점수는 액면대로 매긴다.

point	1 or 11	2	3	4	5	6	7	8	9	10	10	10	10
	Ace	2	3	4	5	6	7	8	9	10	Jack	Queen	King
Spades													
Hearts													
Clubs													
Diamonds													

출처 : [https://en.wikipedia.org/wiki/Standard\\_52-card\\_deck](https://en.wikipedia.org/wiki/Standard_52-card_deck)

# 블랙잭의 규칙

- 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 버스트<sup>bust</sup>가 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

- 집합<sup>set</sup>
  - 순서와 중복 없이 데이터 값을 모을 수 있는 컬렉션<sup>collection</sup> 데이터 구조
  - 수정 가능함 (단, 모든 원소는 수정 불가능한 값만 가능)
  - 중괄호<sup>{}</sup> 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

- 집합<sup>set</sup>
  - 순서와 중복 없이 데이터 값을 모을 수 있는 컬렉션<sup>collection</sup> 데이터 구조
  - 수정 가능함 (단, 모든 원소는 수정 불가능한 값만 가능)
  - 중괄호<sup>{}</sup> 사이에 원소를 , 로 구분해서 나열

집합 메소드	의미
<b>s.add(n)</b>	집합 <b>s</b> 에 원소 <b>n</b> 을 추가한다.
<b>s.remove(n)</b>	집합 <b>s</b> 에서 원소 <b>n</b> 을 제거한다. 집합 <b>s</b> 에 원소 <b>n</b> 이 없으면 <b>KeyError</b> 오류가 발생한다.
<b>set()</b>	빈집합을 생성해서 리턴한다. ( {} 는 빈 딕셔너리 )
<b>set(iterable)</b>	<b>iterable</b> 의 내용을 담은 집합을 생성해서 리턴한다. <b>iterable</b> 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)
>>> s
{3}
>>> s.remove(3)
>>> s
set()
>>> s.remove(0)
...
KeyError: 0

```

# 집합

집합 메소드	연산자	의미
<code>s.update(iterable)</code>		집합 <b>s</b> 에 <b>iterable</b> 의 데이터를 추가한다.
<code>s1.union(s2)</code>	<code>s1   s2</code>	집합 <b>s1</b> 과 <b>s2</b> 의 합집합을 리턴한다. ( <b>s1</b> 과 <b>s2</b> 의 내용은 바뀌지 않음)
<code>s1.intersection(s2)</code>	<code>s1 &amp; s2</code>	집합 <b>s1</b> 과 <b>s2</b> 의 교집합을 리턴한다. ( <b>s1</b> 과 <b>s2</b> 의 내용은 바뀌지 않음)
<code>s1.difference(s2)</code>	<code>s1 - s2</code>	집합 <b>s1</b> 과 <b>s2</b> 의 차집합을 리턴한다. ( <b>s1</b> 과 <b>s2</b> 의 내용은 바뀌지 않음)
<code>s1.symmetric_difference(s2)</code>	<code>s1 ^ s2</code>	집합 <b>s1</b> 과 <b>s2</b> 의 대칭차집합을 리턴한다. ( <b>s1</b> 과 <b>s2</b> 의 내용은 바뀌지 않음)

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

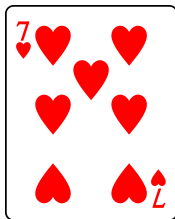
```
>>> s1 | s2
{1, 2, 3}
>>> s1 & s2
{2}
>>> s1 - s2
{1}
>>> s1 ^ s2
{1, 3}
```

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

# 카드게임 API 라이브러리 모듈

# 카드의 표현

- 무늬<sub>suit</sub> : 네가지 문자열로 표현
  - "Spade", "Club", "Heart", "Diamond"
- 숫자<sub>rank</sub> : J, Q, K, A 는 문자열로, 나머지 숫자는 정수로 표현
  - "A", 2, 3, 4, 5, 6, 7, 8, 9, 10, "J", "Q", "K"
- 카드 한 장 : 무늬와 숫자의 튜플로 표현



⇒ ("Heart", 7)

- 카드 덱 : 카드의 리스트로 표현



⇒

[ ("Heart", 7),  
("Club", 8),  
("Diamond", "A")]

# API 라이브러리

- API(Application Programming Interface) 라이브러리
  - 카드 1벌 만들어 무작위로 섞기<sup>실습</sup>
  - 카드 덱에서 카드 한 장 뽑아주기
  - 카드 점수 계산하기<sup>실습</sup>
  - 카드 프린트해서 보여주기
  - 카드를 더 받을지 물어보기

# 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 프린트해서 보여주기

- 인수 : 카드 리스트 cards와 문자열 message
- 메시지를 먼저 출력하고 다음 줄부터 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 `True`를 n이면 `False`를 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)
```

# 직접 모듈을 만들어 보기

## 직접 모듈을 만들어보기

1. IDLE 을 완전히 종료 후 새로 실행한다.
2. 새 파일을 열어 아래 내용을 입력한 후, mylib.py 파일을 저장한다.

```
mySID = 2024123456
myName = "홍길동"

def myFunc(n):
    return n + n
```

```
# 파일을 저장하는 위치
import os
os.getcwd()
```

## Python 인터프리터에서 모듈을 찾는 순서

1. sys.modules 에 등록된 모듈 (딕셔너리)
2. built-in 모듈
3. sys.path 에 나열된 순서대로 폴더를 탐색 (리스트)

3. 새 파일을 하나 더 열어서 아래 내용을 입력하고 적당한 이름으로 저장 후 실행해보자.

```
import mylib
print("내 학번은", mylib.mySID, "이고, 이름은",
mylib.myName, "이야.")
for n in range(0, 10):
    print(mylib.myFunc(n))
```

# 프로그래밍 프로젝트 1 단계 블랙잭

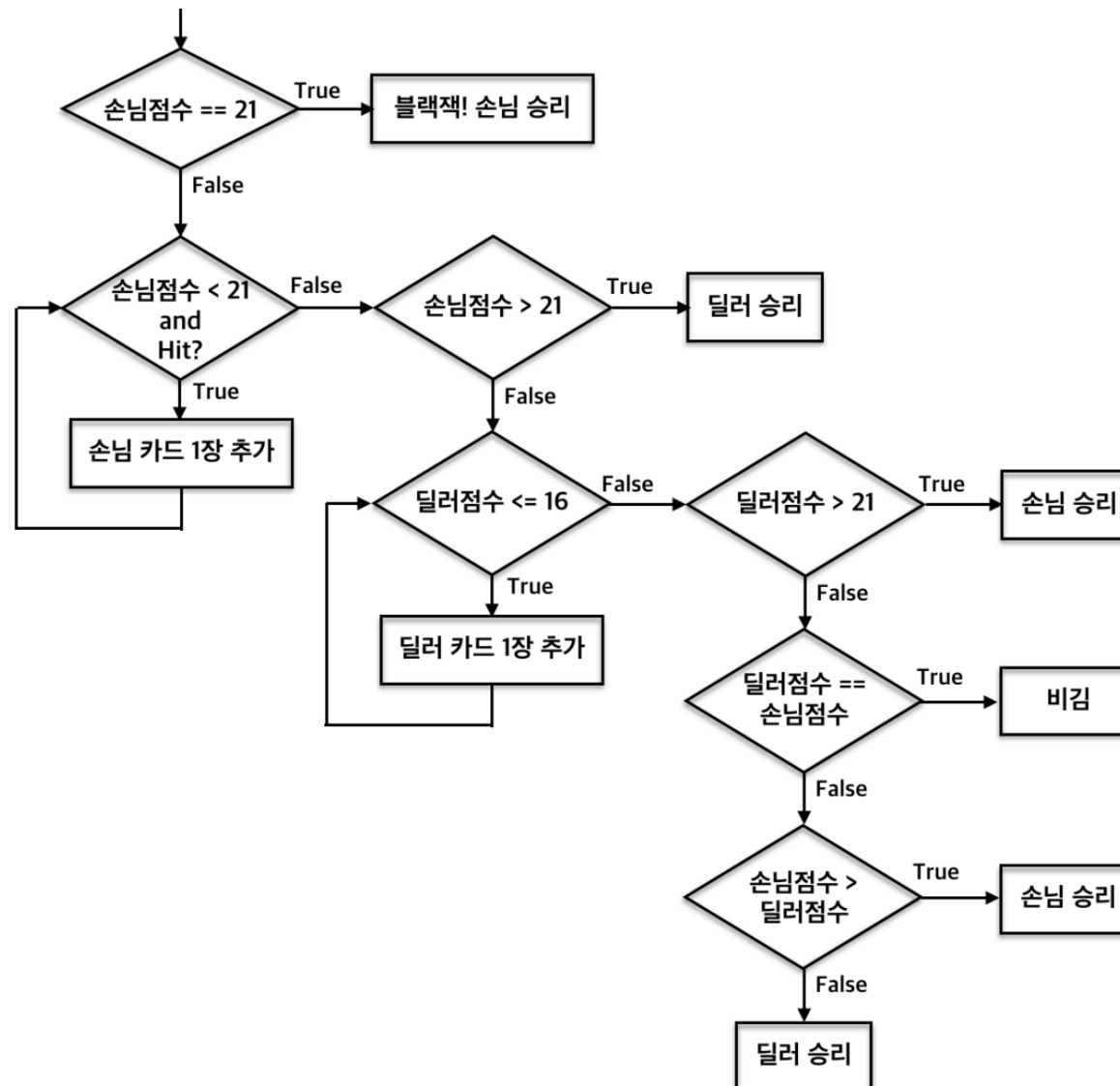
# 블랙잭: 요구사항

- 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 물어보아야 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 경우엔 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. (형식은 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_player`가 21이면 블랙잭으로 손님이 이긴다. `chips`에 2를 더한다.
10. 손님의 카드 합이 21을 넘지 않는 한 손님이 원하면 카드를 더 준다. 21을 넘으면 손님이 버스트되어 딜러가 이기고 `chips`에서 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 `chips`에 1을 더한다.
13. 둘 다 21이 넘지 않으면 큰 쪽이 이긴다. 손님이 이기면 `chips`에 1을 더하고, 딜러가 이기면 `chips`에서 1을 빼고, 비기면 변동 없다.
14. 더 할지 손님에게 물어봐서 그만하길 원하면 끝낸다.

# 블랙잭: 알고리즘



# 딕셔너리

# 딕셔너리

- 딕셔너리<sup>dict</sup>
  - 키와 값의 쌍을 모은 것
  - 중괄호`{}` 사이에 키와 값을 다음과 같이 , 로 구분해서 나열  
  
`{ <키> : <값>, ... , <키> : <값> }`
  - 시퀀스는 (저절로 매겨진) 정수 인덱스를 키로 사용하지만 딕셔너리는 자유자재로 키를 사용 가능
  - 수정불가능<sup>immutable</sup>한 값만 키로 사용 가능  
(집합은 <값> 없이 <키>로만 구성한 딕셔너리로 볼 수 있음)



# 딕셔너리

- 생성과 검색

```
>>> me = { "이름": "하냥이", "생년": 2018, "이메일": "어흥@hanyang.ac.kr" }
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> me["이름"]
'하냥이'
>>> me["생년월일"]
...
KeyError: '생년월일'
```

# 딕셔너리

- 빈 딕셔너리

```
>>> me = { "이름": "하냥이", "생년": 2018, "이메일": "어흥@hanyang.ac.kr" }
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> me["이름"]
'하냥이'
>>> me["생년월일"]
...
KeyError: '생년월일'
```

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

# 딕셔너리

- 안전하게 검색

```
>>> me = { "이름": "하냥이", "생년": 2018, "이메일": "어흥@hanyang.ac.kr" }
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> me["이름"]
'하냥이'
>>> me["생년월일"]
...
KeyError: '생년월일'
```

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

# 딕셔너리

- 수정과 추가

```
>>> me = { "이름": "하냥이", "생년": 2018, "이메일": "어흥@hanyang.ac.kr" }  
>>> me  
{ '이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr' }
```

```
>>> me["이메일"] = "어흥@ehanyang.ac.kr"  
>>> me  
{ '이름': '하냥이', '생년': 2018, '이메일': '어흥@ehanyang.ac.kr' }  
>>> me["친구"] = ["하이비비", "하이리온", "하이나리"]  
>>> me  
{ '이름': '하냥이', '생년': 2018, '이메일': '어흥@ehanyang.ac.kr',  
'친구': ['하이비비', '하이리온', '하이나리'] }
```

# 딕셔너리

- 삭제

```
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@ehanyang.ac.kr',
 '친구': ['하이비비', '하이리온', '하이나리']}
```

```
>>> del me['친구']
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@ehanyang.ac.kr'}
>>> del me['친구']
...
KeyError: '친구'
```

# 딕셔너리

- 안전하게 삭제

```
>>> me  
{ '이름': '하냥이', '생년': 2018, '이메일': '어흥@ehanyang.ac.kr' }
```

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

# 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 : 수정불가능<sup>immutable</sup>한 모든 식

종류	타입	설명	수정가능여부	기타
스칼라 <sup>scalar</sup> 타입	<b>int</b>	정수	수정불가능	
	<b>float</b>	실수	수정불가능	키로 사용할 경우, 실수 오차에 주의
	<b>NoneType</b>	<b>None</b>	수정불가능	
	<b>bool</b>	논리값	수정불가능	
시퀀스	<b>list</b>	리스트	수정가능	키로 사용 불가능
	<b>str</b>	문자열	수정불가능	
	<b>tuple</b>	튜플	수정불가능	내부 값도 수정불가능해야 키로 사용 가능
	<b>range</b>	정수범위	수정불가능	
집합 <sup>set</sup>	<b>set</b>	집합	수정가능	키로 사용 불가능
	<b>frozenset</b>	집합	수정불가능	내부 값도 수정불가능해야 키로 사용 가능
매핑 <sup>mapping</sup>	<b>dict</b>	딕셔너리	수정가능	키로 사용 불가능

# 딕셔너리

- 딕셔너리의 키와 값의 타입
  - 키 : 수정불가능<sup>immutable</sup>한 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
<b>d.keys()</b>	딕셔너리 <b>d</b> 의 키를 리스트로 모은 뷰 객체dict_keys를 리턴한다.
<b>d.values()</b>	딕셔너리 <b>d</b> 의 값을 리스트로 모은 뷰 객체dict_values를 리턴한다.
<b>d.items()</b>	딕셔너리 <b>d</b> 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': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> me.keys()
dict_keys(['이름', '생년', '이메일'])
>>> me.values()
dict_values(['하냥이', 2018, '어흥@hanyang.ac.kr'])
>>> me.items()
dict_items([('이름', '하냥이'), ('생년', 2018), ('이메일', '어흥@hanyang.ac.kr')])
```

# 딕셔너리

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

```
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> for x in me.keys() : print(x)

>>> for x in me.values() : print(x)

>>> for x in me.items() : print(x)
```

# 딕셔너리

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

```
>>> me
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> my_items = me.items()
>>> for (key, value) in my_items : print(key, ":\t", value)
```

# 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': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr'}
>>> my_items = me.items()
>>> me['친구'] = "하이비비"
{'이름': '하냥이', '생년': 2018, '이메일': '어흥@hanyang.ac.kr', '친구': '하이비비'}
>>> for (key, value) in my_items : print(key, ":\t", value)
```

# 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 등록 된 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 보여 준다.
- 게임에서 이기면 받는 칩도 누적하여 기록해 둔다.
- 게임을 시작하면 칩 보유 개수를 적절한 형식(교재 참조)으로 알려 준다.
- 게임이 끝나면 해당 세션 동안의 기록과 갱신된 랭킹을 적절한 형식(교재 참조)으로 보여 준다.

# 게임기록 저장하기

- CSV(comma-separated values) 파일에 저장하는 게임 기록 정보의 형식
  - 아이디name
  - 비밀번호password
  - 게임시도 횟수tries
  - 이긴 횟수wins
  - 칩 보유 개수chips

# 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!'
```

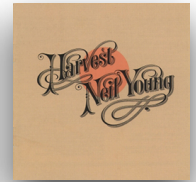
<https://docs.python.org/ko/3/library/stdtypes.html#str.format>

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

# 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)

>>> sorted(albums.items())

>>> sorted(albums.items(), key=lambda x: x[0])

>>> sorted(albums.items(), key=lambda x: x[1][1])

>>> sorted(albums.items(), key=lambda x: x[1][1], reverse=True)
```

# 블랙잭: 알고리즘

1. 환영인사를 프린트한다.
2. members.csv 파일에서 멤버 기록을 읽고 로그인 절차를 통해 사용자이름, 게임시도 횟수, 이긴 횟수, 칩 보유개수, 전체 멤버 디렉터리 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 디렉터리에 적용하여 수정하고, members.csv 파일에 저장한다.
16. 해당 세션의 게임 결과를 다음과 같이 요약하여 보여준다.
17. 지금까지의 칩 최다 보유 멤버를 5명까지 보여준다.