**Git**

[**https://github.com/Asist525/Source\_Code/tree/main/python/Assignment/blackjack/module**](https://github.com/Asist525/Source_Code/tree/main/python/Assignment/blackjack/module)

**개발 블로그**

**https://velog.io/@asist525/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%ED%94%84%EB%A1%9C%EC%A0%9D%ED%8A%B81**

**1. 블랙잭이란?**

**1) 게임 규칙**

블랙잭은 palayer과 dealer가 각각 카드를 받아 합이 21에 가장 가까운 user가 이기는 게임입니다. 숫자를 초과하게 되면 패배하며, 숫자가 더 크되 21을 넘지 않는 쪽이 승리합니다.  
게임의 규칙은 다음과 같습니다.

* 숫자 카드(2 ~ 10)은 그대로 점수로 계산합니다.
* J, Q, K는 모두 10점입니다.
* 에이스의 경우 상황에 따라 1점 혹은 11점으로 계산합니다.
* 처음 받은 두 장의 카드로 합이 21이라면 블랙잭으로 즉시 승리합니다.

**2) player의 행동**

* Hit: 카드를 한장 더 받습니다.
* Stand: 현재 점수로 턴을 종료합니다.

**3) dealer의 행동**

* 17이상이 될 때까지 자동으로 카드를 뽑고, 그 이후엔 멈춥니다.

**4) betting**

* 일반적인 승리의 경우 2배입니다.
* 블랙잭을 통한 승리의 경우 2.5배입니다.

**2. 코드 설명**

**1) Card**

class Card:

def \_\_init\_\_(self, suit, value):

self.suit = suit

self.value = value

def \_\_str\_\_(self):

return f'{self.suit} {self.value}'

def get\_value(self):

if self.value in ['J', 'Q', 'K']:

return 10

elif self.value == 'A':

return 11

else:

return int(self.value)

* 카드의 무늬(Heart, Spade, Diamond, clober)와 숫자를 표현합니다.
* get\_value 매소드를 통해 각 카드에 대한 점수 역시 관리합니다.
  + J, Q, K인 경우 10점
  + 에이스인 경우 11점(향후 Hand클래스의 score 매소드에서 1점 처리를 합니다)
  + J, Q, K, A가 아닌 경우는 숫자 뿐이므로, 이 경우 각 숫자에 맞는 value를 반환합니다.

**2) Deck**

class Deck:

def \_\_init\_\_(self):

self.cards = []

suits = ['Hearts', 'Diamonds', 'Clubs', 'Spades']

values = ['2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A']

for suit in suits:

for value in values:

self.cards.append(Card(suit,value))

def shuffle(self):

random.shuffle(self.cards)

def deal(self):

if len(self.cards) > 0:

return self.cards.pop()

else:

return None

* self.cards.append(Card(suit, value))을 통해 Card에서 리턴값을 가져와 self.cards 리스트에 삽입합니다. 이 과정에서 52장의 Deck이 만들어집니다.
* shuffle매소드는 인스턴스 변수로 만들어진 52장의 self.cards를 섞는 역할을 합니다.
* deal의 경우 카드를 하나씩 뽑는(pop으로 구현) 역할을 합니다. 만약 다 뽑은 경우라면 카드가 없으므로 None을 반환해 처리합니다.

**3) Game**

class Game:

'''

전체 게임 흐름 제어

'''

def \_\_init\_\_(self):

'''

Deck에서 전체 카드 덱 리스트를 가져와서 self.deck에 저장

'''

self.deck = Deck()

self.deck.shuffle()

def pop(self):

'''

1. 전체 카드 덱에서 하나씩 pop => 리턴

2. 리턴되는 값은 Hands에서 받음

'''

return self.deck.deal()

* Game클래스는 전체 게임 흐름을 제어하는 클래스입니다.
* 기본적으로 Deck()에 참조하여, Deck클래스의 메소드를 호출, 사용할 수 있습니다. 생성자에서 self.deck = Deck()을 통해 Deck클래스를 참조하며, deck클래스의 shuffle매소드를 통해 섞는것까지 Game클래스에서 담당합니다.
* pop 역시 deck클래스의 deal매소드를 가져오고 리턴합니다. 이렇게 리턴하는 값은 향후 Hand클래스가 Game클래스를 참조하며 한장씩 받는 역할을 합니다.

**4) Hand**

class Hand:

'''

플레이어나 딜러가 보유한 카드 관리, 점수 계산 기능 구현

Game의 카드를 받음 =>

'''

def \_\_init\_\_(self, game ):

self.dealer = []

self. player = []

self.game = game

for \_ in range(2):

self.dealer.append(self.game.pop())

for \_ in range(2):

self.player.append(self.game.pop())

def score(self, who):

cards = self.player if who == 'player' else self.dealer

total = 0

A = 0

for card in cards:

score = card.get\_value()

total += score

if card.value == 'A':

A += 1

while total > 21 and A > 0:

total -= 10

A -= 1

return total

* Hand 클래스에서는 플레이어나 딜러가 보유한 카드를 관리해야 하며, 점수 계산 역시 담당해야합니다.
* 먼저 게임의 규칙은 초기에 플레이어, 딜러가 2장의 카드를 받는 것입니다. 이를 위해 인스턴스 변수에서 game클래스를 포함하며, pop을 for문을 통해 2개의 카드를 각각 받습니다.
* score매소드에서는 점수를 측정하는 역할을 합니다. 위의 첫번째 클래스였던 card클래스의 get\_value매소드는 각 카드에 맞는 점수를 기록하고 있습니다. 이 점수를 가져오고, 에이스를 더할 경우 21을 넘는다면 에이스는 1로 변경됨을 처리합니다.

**5) Player**

class Player:

'''

플레이어 정보 및 게임 결정(히트 또는 스탠드) 관리

'''

def \_\_init\_\_(self, game):

self.game = game

self.hand = Hand(game) # 플레이어 초기 카드 2장 받음

def Hit(self):

card = self.game.pop()

self.hand.player.append(card) # 추가된 카드 저장

return self.hand.score('player')

def Stand(self):

print("턴 종료")

* Player 클래스에서도 역시 game에 참조합니다. 또한 Hand역시 참조하는데, 이는 Hand클래스를 통해 2장의 플레이어 카드를 가져오기 위함입니다.
* Hit은 플레이어가 카드를 한장 더 받는 것입니다. 이를 구현하기 위해서는 pop을 하는 game클래스와 hand의 점수를 측정하는 클래스, 두 클래스를 모두 가져와야 합니다. 먼저 game.pop을 통해 카드를 한장 가져오고, 이를 player에 맞게 점수를 측정하고, 리턴합니다.
* stand는 플레이어가 턴을 종료하는 매소드입니다. 멈출 경우 단순히 턴이 종료되는 매소드이므로 print로 처리하였습니다.

**6) Dealer**

class Dealer:

'''

딜러 행동 규칙 구현 (17 이상이면 스탠드)

'''

def \_\_init\_\_(self, game, hand):

self.game = game

self.hand = hand

def play(self):

while self.hand.score('dealer') < 17:

card = self.game.pop()

self.hand.dealer.append(card)

print(f"딜러가 카드를 뽑았습니다: {card}")

print("딜러 현재 점수:", self.hand.score('dealer'))

print("\n딜러는 스탠드합니다.")

print("딜러 최종 카드:", ', '.join(str(c) for c in self.hand.dealer))

print("딜러 최종 점수:", self.hand.score('dealer'))

* Player클래스와 마찬가지로 game과 hand 클래스를 참조하여 pop매소드와 score매소드를 사용합니다.
* Dealer의 경우 score가 17을 넘을때까지 카드를 뽑아야 합니다. 이는 while self.hand.score('dealer') < 17을 통해 구현할 수 있습니다.  
  이렇게 가져오는 값에 따라 점수를 기록, 17이 넘을때까지 반복합니다.

**7) main함수**

def main():

cib = 100

print("블랙잭 게임에 오신 것을 환영합니다!")

name = input("이름을 입력해주세요: ")

while True:

if cib <= 0:

print("칩이 모두 소진되었습니다. 게임 종료!")

break

print("\n────────── 새 게임 ──────────")

print(f"보유 칩: {cib}")

try:

betting = int(input("배팅할 금액을 입력해 주세요: "))

except ValueError:

print("숫자로 입력해 주세요.")

continue

if betting > cib or betting <= 0:

print("잘못된 금액입니다. 다시 입력해주세요.")

continue

cib -= betting

game = Game()

player = Player(game)

dealer = Dealer(game, player.hand)

# 출력: 딜러 카드 한 장만 공개

visible\_card = player.hand.dealer[0]

dealer\_str = f"{visible\_card}, hidden"

player\_str = ', '.join(str(card) for card in player.hand.player)

print(f"\n{name}의 카드: [{player\_str}]")

print(f"{name}의 초기 점수:", player.hand.score('player'))

print(f"\n딜러의 카드: [{dealer\_str}]")

print("딜러 초기 점수:", visible\_card.get\_value())

# 블랙잭 판정

if player.hand.score('player') == 21:

print("블랙잭! 이겼습니다!")

cib += int(betting \* 2.5)

else:

# 플레이어 턴

while True:

move = input("입력 (hit / stand): ").strip().lower()

if move == 'hit':

score = player.Hit()

player\_str = ', '.join(str(c) for c in player.hand.player)

print(f"{name}의 카드: {player\_str}")

print("현재 점수:", score)

if score > 21:

print("버스트! 당신은 졌습니다.")

break

elif move == 'stand':

print("턴 종료")

break

else:

print("잘못된 입력입니다. hit 또는 stand를 입력하세요.")

# 딜러 턴

dealer.play()

# 승패 판정

player\_score = player.hand.score('player')

dealer\_score = player.hand.score('dealer')

print("\n[결과]")

if player\_score > 21:

print("버스트! 패배")

elif dealer\_score > 21 or player\_score > dealer\_score:

print("이겼습니다!")

cib += betting \* 2

elif player\_score < dealer\_score:

print("딜러 승리!")

# 칩 차감은 이미 됐음

else:

print("무승부!")

cib += betting # 환불

cont = input("\n계속 하시겠습니까? (1: 계속 / 기타: 종료): ").strip()

if cont != "1":

print(f"게임 종료! 최종 칩: {cib}")

break

if \_\_name\_\_ == "\_\_main\_\_":

main()

* 메인함수는 전체 실행 흐름을 관리하는 진입점입니다. 플레이어로부터 이름을 받고, 반복적으로 게임을 실행하면서 칩의 상태, 배팅, 입력 처리, 승패 판정 등을 담당합니다.
* cib의 경우 기본 수치 100으로 설정하였습니다. while문에서 cib이 0보다 작은 경우 종료하는 예외 처리기능을 넣었습니다.
* 사용자에게 정수로된 배팅 금액을 입력받습니다. 잘못 입력하는 경우의 예외처리를 해두었습니다.
* 객체 생성 및 초기 카드 분배의 경우 다음과 같이 처리됩니다.
  + 덱 셔플 -> 카드 분배 -> 상태 초기화
  + Player와 Dealer의 경우 공통의 Hand 객체를 공유하며, 각각의 카드 리스트만 분리해 사용하도록 구성하였습니다.
* 게임 규칙에서 딜러의 경우 첫 카드만 공개하며, 나머지는 숨깁니다. 이를 visible\_card = player.hand.dealer[0]을 통해 처리해 두었습니다.
* 승패 판정의 경우 if-else를 통해 처리하였습니다. 버스트, 딜러 버스트, 점수 비교, 무승부의 4가지 경우를 처리합니다. 결과에 따라 배팅금액을 처리합니다.

**3. 모듈 처리**

현재 코드가 class와 main코드로 잘 처리가 되어있으므로 class별로 파일을 만들고, 참조관계에 맞게 라이브러리로 호출하면 됩니다.  
현재 참조 및 포함 관계는 다음과 같습니다.

* Dealer클래스는 Hand클래스와 Game클래스를 참조합니다.
* Deck클래스는 Card클래스를 참조합니다. 추가로 random 라이브러리를 사용합니다.
* Game클래스는 Deck클래스를 참조합니다.
* Hand클래스는 Card클래스를 참조합니다.
* Player클래스는 Game클래스와 Hand클래스를 참조합니다.
* main함수는 모든 클래스를 사용합니다.

.

├── Card.py

├── Dealer.py

├── Deck.py

├── Game.py

├── Hand.py

├── Player.py

├── \_\_pycache\_\_

│ ├── Card.cpython-312.pyc

│ ├── Dealer.cpython-312.pyc

│ ├── Deck.cpython-312.pyc

│ ├── Game.cpython-312.pyc

│ ├── Hand.cpython-312.pyc

│ └── Player.cpython-312.pyc

└── main.py

* tree 구조를 다음과 같이 설정하고 main.py를 실행하면 됩니다.

(python\_venv) ubuntu@DESKTOP-QIE759D:~/Source\_Code/python/Assignment/blackjack/module$ python main.py

블랙잭 게임에 오신 것을 환영합니다!

이름을 입력해주세요: