```
class BJ_Card(cards.Card):
    ACE_VALUE = 1

    @property
    def value(self):

        if self.is_face_up:
            v = BJ_Card.RANKS.index(self.rank) + 1
            if v > 10:
                v = 10
        else:
            v = None
        return v
```

“””

Esta clase devuelve el valor de la tarjeta en función de su rango. Si la carta está boca arriba, devuelve el valor de la carta en función de su rango, con cartas con figuras (Jota, Reina, Rey) con un valor de 10. Si la carta está boca abajo, el valor se establece en 1

“””

```
class BJ_Deck(cards.Deck):
    def populate(self):
        for suit in BJ_Card.SUITS:
            for rank in BJ_Card.RANKS:
                self.cards.append(BJ_Card(rank, suit))
```

“””

Esta clase utiliza una función ‘populate’ que agrega una ‘BJ Card’ objeto para cada combinación ‘cards’ atributo ‘BJ Deck’ objeto ‘BJ_Card’ objeto.

“””

```
class BJ_Hand(cards.Hand):
    def __init__(self, name):
        super(BJ_Hand, self).__init__()
        self.name = name

    def __str__(self):
        rep = self.name + ":\t" + super(BJ_Hand, self).__str__()
        if self.total:
```

```
            rep += "(" + str(self.total) + ")"
        return rep
```

“””

Este código define una clase nombrada BJ_Hand que hereda de la Hand clase definida en el cards módulo. La BJ_Hand clase se utiliza para representar la mano de un jugador en el juego.

La función __init__se define para inicializar el BJ_Hand objeto con un name atributo que se pasa como argumento.

La función __str__está definida para devolver una representación de cadena de la mano del jugador.

“””

```
class BJ_Player(BJ_Hand):
    def is_hitting(self):
        response = games.ask_yes_no("\n" + self.name + ", Desea otra carta?:
")
        return response == "y"

    def bust(self):
        print(self.name, "SE PASO.")
        self.lose()

    def lose(self):
        print(self.name, "PERDIO.")

    def win(self):
        print(self.name, "GANO.")

    def push(self):
        print(self.name, "EMPATA.")
```

“””

La clase hereda de BJ_Hand, lo que sugiere que representa la mano de cartas de un jugador.

El is_hitting método le pide al jugador que elija si pedir o no pedir (solicitar otra carta) y regresa True si el jugador elige pedir y False si elige plantarse.

Los métodos bust, lose, win y push representan diferentes resultados del juego para el jugador. Si el jugador se pasa (excede el valor de la mano de 21), el método bust imprime un mensaje que indica que el jugador ha perdido y llama al lose método. Si el jugador gana, el método win imprime un mensaje que indica que el jugador ha ganado. Si el juego resulta en un empate, el método push imprime un mensaje que indica que el juego es un empate.

“””

```
class BJ_Dealer(BJ_Hand):

    def is_hitting(self):
        return self.total < 17

    def bust(self):
        print(self.name, "SE PASO.")

    def flip_first_card(self):
        first_card = self.cards[0]
        first_card.flip()
```

“””

Esta es una clase para el dealer. La clase también hereda de BJ_Hand, lo que sugiere que representa la mano de cartas del dealer.

El método is_hitting indica si el dealer debe o no continuar pidiendo (solicitando otra carta) en función del valor actual de su mano.

El método bust imprime un mensaje que indica que el dealer se ha pasado (excedió un valor de mano de 21).

El método flip_first_card voltea la primera carta del dealer para revelarla a los jugadores.

“””

```
class BJ_Game(object):

    def __init__(self, names):
        self.players = []
        for name in names:
            player = BJ_Player(name)
            self.players.append(player)

        self.dealer = BJ_Dealer("Dealer")

        self.deck = BJ_Deck()
        self.deck.populate()
        self.deck.shuffle()
```

“””

Esta clase inicializa un juego con una lista de nombres de jugadores pasados como argumento.

El método __init__ crea una lista de BJ_Player objetos para cada nombre de jugador proporcionado y los agrega al players atributo del BJ_Game objeto. También crea un BJ_Dealer objeto con el nombre "Dealer" y lo asigna al dealer atributo del BJ_Game objeto.

También se crea una instancia de la BJ_Deck clase y el método populate para agregar una baraja completa de 52 cartas a la baraja. Luego se llama al método para barajar la baraja (Shuffle).

“””

```
@property
    def still_playing(self):
        sp = []
        for player in self.players:
            if not player.is_busted():
                sp.append(player)
        return sp
```

“””

El método still_playing inicializa una lista vacía sp. Luego itera a través de cada jugador self.players y verifica si han fallado, llamando a el método is_busted. Si el jugador no ha pasado (es decir, el valor de su mano es menor o igual a 21), el jugador se agrega a la sp lista.

Finalmente, se devuelve la sp lista, que contiene solo aquellos jugadores que aún no han pasado y todavía están jugando activamente en el juego.

“””

```
def __additional_cards(self, player):
        while not player.is_busted() and player.is_hitting():
            self.deck.deal([player])
            print(player)
            if player.is_busted():
                player.bust()
```

“””

Este es un método privado de la BJ_Game clase que reparte cartas adicionales a un jugador siempre que aún no haya pasado y elija pedir. El método toma un player objeto como argumento.

El método inicia un ciclo que continúa mientras el jugador aún no haya pasado y elija pedir. Dentro del bucle, deck.deal se llama al método para repartir otra carta al jugador de la baraja. La mano del jugador se imprime usando la print(player).

Si el jugador se pasa como resultado de la carta adicional, el método player.bust() indica que el jugador ha perdido el juego.

“””

```
def main():


    names = []
    number = games.ask_number("Cuantos usuarios desean jugar? (1 - 2): ",
low = 1, high = 3)
    for i in range(number):
        name = input("Digit eel nombre o usuario del jugador: ")
        names.append(name)
    print()

    game = BJ_Game(names)
```

“””

Esta es una definición de la función main que inicia el juego de blackjack.

La función primero imprime un mensaje de bienvenida a la consola usando la print.

Luego le pide al usuario que ingrese la cantidad de jugadores usando la función games.ask_number, que limita la entrada a un número entre 1 y 2.

Luego, la función ingresa a un ciclo que se ejecuta (number), en el tiempo y solicita al usuario que ingrese el nombre de cada jugador mediante un input. Los nombres se agregan a una lista llamada names.

Una vez que se completa el ciclo, se imprime una nueva línea en la consola usando print.

Finalmente, BJ_Games crea un objeto con la lista de nombres de jugadores, usando el BJ_Game (names) constructor.

“””

```
 again = None
    while again != "n":
        game.play()
        again = games.ask_yes_no("\nDesea jugar otra vez?: ")
```

“””

Este código crea un bucle que se ejecuta mientras el jugador quiera seguir jugando.

Dentro del bucle, game.play() se llama al método para iniciar un nuevo juego de blackjack.

Una vez que finaliza el juego, games.ask_yes_no se llama a la función para preguntarle al jugador si quiere volver a jugar. La respuesta del usuario se almacena en la variable again.

El bucle continuará ejecutándose mientras el usuario responda "y" a la games.ask_yes_no, lo que indica que quiere jugar otro juego. Si el usuario responde "n", el ciclo se cerrará y el programa finalizará.

Se llama a la función main.

Una vez que finaliza el juego, con un input se le solicita al usuario que presione la tecla Intro para salir del programa.