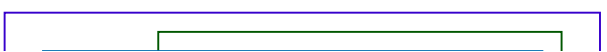


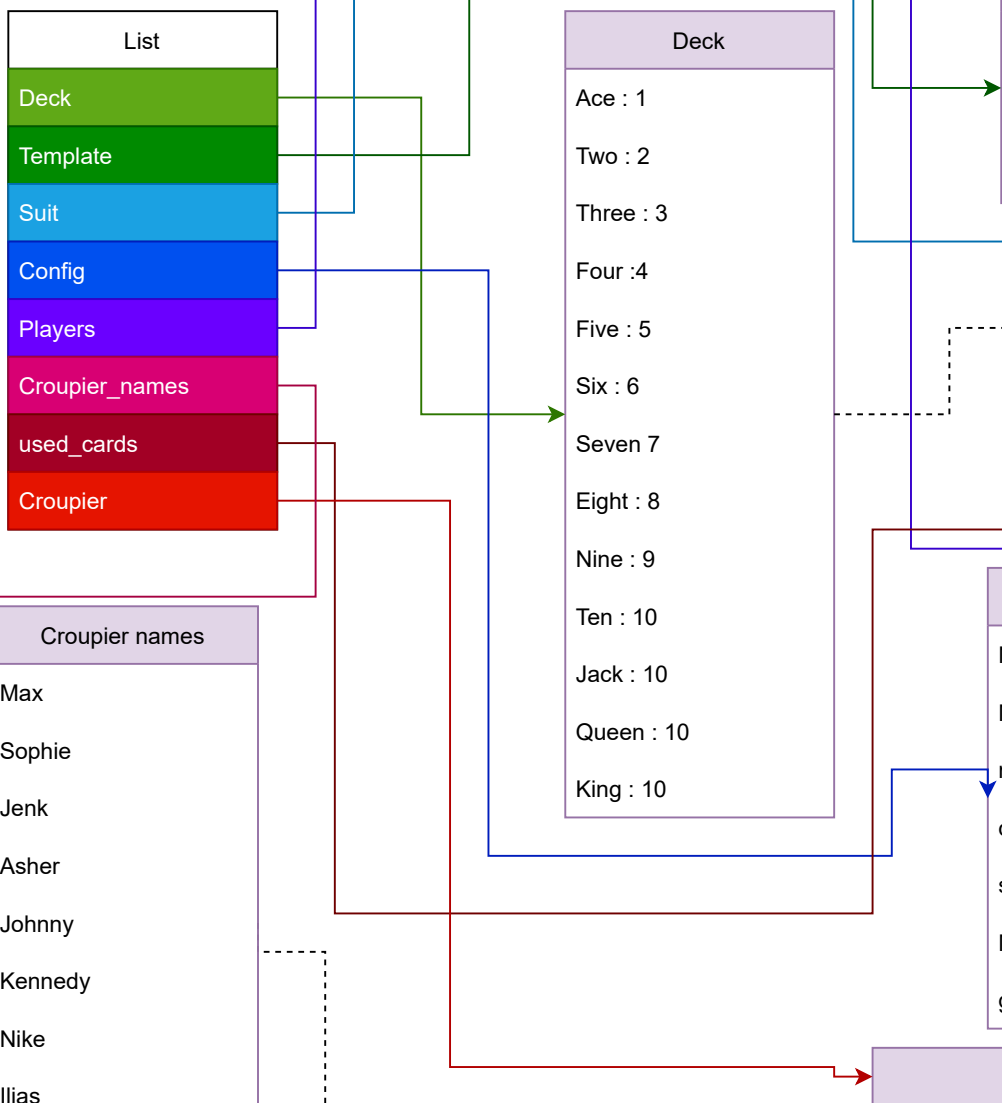
There is an element that is called Active. this indicates if the players has withdrawn or not.

Step 1

Set up initialisations. We are pre setting the lists and the dictionaries that will get updated by the functions later. Here are our initialisations:

Dictionaries

Lists



Step 2

Create the initialisation functions. These functions allow to update, change or modify the default initialisations that we presented above. For instance, we will personalise the game by giving to they players names via the initPlayers() function and we will save these details on the Players dictionary.

We decided to follow the object-oriented programming logic. This means that we were developing the functions the one after the other by thinking what we want the program to use on the computer's screen.

NOTE: On the functions below, the spaces from the program are not displayed correctly.

startGame()

```
Croupier['Name'] = random.choice(Croupier_names)
print("Hello, my name is {}. I'll be your Croupier for this session.".format(Croupier['Name']))
print('The prizes are:')
print('- Blackjack pays 3 to 2.')
print('- Wins pays 1 to 1.')
print('- Insurance pays 2 to 1.')
configuration()
```

It just introduces into the game and appears the croupier name that is selected randomly from the Croupier-names dictionary initialised above. Of course we have addd on the beginning of the file the random library via **import random**.

configuration()

```
print('\
print('D
while Tr
print('T
Config['
configur
if confi
print('P
elif con
numberPl
else:
while Tr
print('P
min_bet
if min_b
Config['
break
while Tr
print('P
```

This function corresponds to the UI that we created on the version 1 of the program on the main program. For version 2, we integrated this UI inside a function that let's the user to interact with it and once an action is completed according to rules that we have settled, then moves to the corresponding function due to our

A default template for every time that we are creating a new player in the players dictionary

of Clubs

of Hearts

of Spades

Used also to value cards. This is why instead of having a valueCard()function like we had on version 1 of the program, we have a valueCardSum() function that operates for the specified cards saved on local storage the current specific time

Used to print the card as mentioned on the project's document

Used cards

A dictionary to store the cards that have been used

Config

Number of players

Money

max number of decks

deck size

show

Minimum bet

game

Players

A blank dictionary. It will get updated from the initPlayers() function

Waits to be updated by the initPlayers() function

Croupier

mplate similar to players template to define croupier's properties

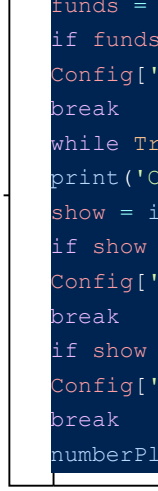
```
nLets personalize your game a little bit')
Do you prefer default rules, or would you rather have a custom set of rules?')
ue:
The default rules are: Minimum bet = {}¢; Funds available to each player = {}¢; Croupier show one card.'.format(
 Minimum_bet'], Config['Money']))
rationuration = input("default/custom: ").lower()
igurationuration not in ['default', 'custom']:
Please, enter a valid input.')
nfigurationuration == 'default':
layers()

ue:
How much should minimum bet be?')
 = input("#: ")
bet.isdigit() and 0 < int(min_bet) < 100:
 Minimum_bet'] = int(min_bet)

ue:
How much funds should each player have?')
```

objects-oriented logique as explained above.

Essential step for the game is set up the betting rules. The default rules (minimum 5 and maximum 100) are set inside the dictionary Config.

If the user selected default, then we is forced to the next function and action in the order which is to give players number -> and their names as well. Otherwise, he has to enter the rules as we can see, like: minimum bet, maximum bet and if the croupier will show his cards (show identifier inside the dictionary).

It is also interesting that here we are using an infinite loop to filter out all the incorrect responses in the different questions that are posed to the user. After the completion of the task, we move on to the numberPlayers() function.



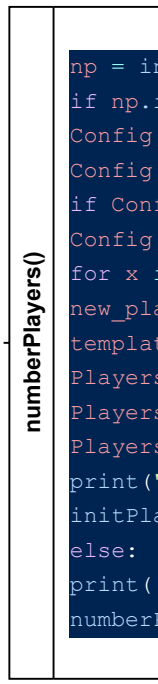
This function is similar to initPlayers() function from our version 1 of the program. It asks the number of players and it creates its profiles. Though, we decided to divide the personalised of players with names in a different function because dictionary (dict) and list are mutable, hence they are referenced and so we need to create a new one for each player via the Template dictionary.

Then, inside the big Players dictionary, we will personalise dictionary per dictionary the Name attribute

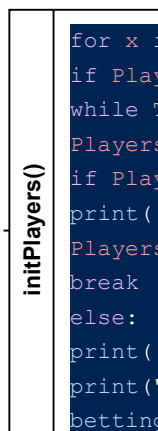
Here when we filter the response that needs to be a number, instead of having an infinite loop like in the previous function, we re-call the function in case that the user types something that do not correspond on the data type that we have set up in our filtration.

NOTE: The "`np.isdigit() and 1 <= int(np)`" prevents the program in debug mode to just stop running due to incorrect insertion of data from the keyboard.

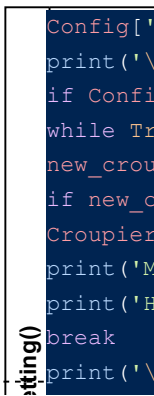
This function also includes the operation of initStack() from the version 1 of our program. On the first version, for every new player, we are using one more deck (+52 cards on the game). On version 2 of the game we are using the operation numberOfPlayers // 2 to make a smarter choice of how many decks we should use instead of adding many many cards on the game that will never get used since there is the possibility for the player/s to withdraw etc.



Same UI with the version 1 of the program. This function also updates the name from Player1, Player 2, Player 3 etc. that were setted previously to the name that the user now inserts via his keyboard.



It's time for every player to bet . Here we have a set a security valve for the croupier so that he can not play more than 10 games. This makes the game more balanced since we have not to forget that this function will repeat from now and on every time that we have a new game round.



```
input("#: ")
.isdigit() and Config['Minimum_bet'] * 2 < int(funds):
Money'] = int(funds)

ue:
Croupier should show you one card?')
nput("y/n: ").lower()
== 'y':
show'] = True

== 'n':
show'] = False

ayers()
```

```
nput("How many players ? ")
isdigit() and 1 <= int(np):
['Number of players'] = int(np)
['max_deck'] = Config['Number of players'] // 2
fig['max_deck'] == 0:
['max_deck'] = 1
in range(1, Config['Number of players'] + 1):
ayer = 'Player ' + str(x)
te_new = dict(Template)
s.setdefault(new_player, template_new)
s[new_player].setdefault('Deck1', [])
s[new_player].setdefault('c_values1', [])
"There will be {} deck(s) in play".format(Config['max_deck']))
ayers()

'Enter a valid input.')
Players()
```

```
in Players:
yers[x]['Active']:
True:
s[x]['Name'] = input("Name of player " + list(x)[7] + ": ")
yers[x]['Name'] != '':
'Nice to meet you {}!'.format(Players[x]['Name']))
s[x]['Money'] = Config['Money']


'Please, tell me your name.')
"We are ready. Let's start!")
g()
```

```
game'] += 1
nGame number {}'.format(Config['game']))
g['game'] % 10 == 0:
ue:
pier = random.choice(Croupier_names)
croupier != Croupier['Name']:
['Name'] = new_croupier
My turn is over. I introduce you to your new croupier, {}'.format(Croupier['Name']))
Have fun!\n')

nBetting round')
```

```
for x in
if Playe
while Tr
print('\
bet = in
if bet.i
Players[
print("Y
break
firstTur
```

firstTurn()

```
def firstTurn():
# round_number = 1 draw 2 cards and check for Blackjack, doubles, deck value
for x in Players:
if Players[x]['Active']:
drawCard(2, Players[x], 'Deck1', 'c_values1')
drawCard(2, Croupier, 'Deck1', 'c_values1')
for x in Players:
if Players[x]['Active']:
print(Players[x]['Name'] + ' these are your cards:')
print_deck(Players[x], 'Deck1')
if Config['show']:
print("The croupier, {}, has {} and one hidden card.".format(Croupier['Name'], Croupier['Deck1'][0]))
else:
print("The croupier, {}, has two hidden cards.".format(Croupier['Name']))
for x in Players:
if Players[x]['Active'] and Players[x]['Play']: # Check for Blackjack first on both sides
player = Players[x]
cards = player['c_values1']
if (cards[0] == 'Ace' and Deck.get(cards[1]) == 10) or (Deck.get(cards[0]) == 10 and cards[1] == 'Ace'):
player['BJ'] = True
print("Congratulations {}! You have Blackjack!".format(player['Name']))
player['Play'] = False
cards = Croupier['c_values1']
if (cards[0] == 'Ace' and Deck.get(cards[1]) == 10) or (Deck.get(cards[0]) == 10 and cards[1] == 'Ace'):
Croupier['BJ'] = True
print('I have Blackjack!')
print_deck(Croupier, 'Deck1')
winner() # if Croupier has blackjack, no need to look more
for x in Players:
if Players[x]['Active'] and Players[x]['Play']:
player = Players[x]
if player['c_values1'][0] == player['c_values1'][1] and valueCardSum(player['c_values1']) in [9, 10, 11]:
print(
'\n{}. It seems you can double your bet and split your hand. Be wary, you can only do one!'.format(
player['Name']))
if valueCardSum(player['c_values1']) in [9, 10, 11]:
double_bet(Players[x])
for n in Players:
player = Players[n]
if player['c_values1'][0] == player['c_values1'][1] and player['Play']: # and len(player['c_values1']) == 2
split_deck(player)
hit_stand(Players[n], 'Deck1', 'c_values1', 'Score1', 'Play')
if Players[n].get('Double'):
hit_stand(Players[n], 'Deck2', 'c_values2', 'Score2', 'Double')
croupier()
```

```
 Players:
rs[x]['Active']:
ue:
nPlease, {}. Place a bet! You can go up to {}'.format(Players[x]['Name'], Players[x]['Money']))
put("#: ")
sdigit() and Config['Minimum_bet'] <= int(bet) <= 500 and int(bet) <= Players[x]['Money']:
x]['Bet'] = int(bet)
our bet has been registered.")

n()
```

croupier()

```
print("\nIt's my turn.")
while Croupier['Play']:
val = valueCardSum(Croupier['c_values1'])
for card_v in Croupier['c_values1']:
if card_v == 'Ace':
Croupier['Ace'] = True
print('My cards are:')
print_deck(Croupier, 'Deck1')
if Croupier['Ace'] and (val + 10) <= 21:
print('Hard value: {}'.format(val))
print('Soft value: {}\n'.format((val + 10)))
else:
print("Its values is: {}\n".format(val))
if val > 21:
print('Bust! My hand is over 21.')
Croupier['Play'] = False
Croupier['Score1'] = val
elif Croupier['Ace'] and 17 <= (val + 10) <= 21:
print('I stand.')
Croupier['Score1'] = val + 10
Croupier['Play'] = False
elif 17 <= val <= 21:
Croupier['Score1'] = val
print('I stand.')
Croupier['Play'] = False
elif Croupier['Ace'] and (val + 10) < 17:
print('I hit for another card.')
drawCard(1, Croupier, 'Deck1', 'c_values1')
elif val < 17:
print('I hit for another card.')
drawCard(1, Croupier, 'Deck1', 'c_values1')
winner()
```

Once the players has played, it's time for our croupier to play. This is for what is the croupier() function all about.

This function also examines if the croupier is busted or not via the logique val > 21 and it has an algorithm to determine if he will stand or hit another card. This is the approach that was requested on the project's documentation.

It draws specific amount of cards anytime that is called this function as it does on version 1 of our program.

drawCard()

```
if Config['deck_size'] > (40 * Config['max_deck']):
left = (12 * Config['max_deck'])
print('\nOnly {} cards left in the deck!'.format(left))
print('Time for reshuffling!')
used_cards.clear()
Config['deck_size'] = 0
for x in range(quantity):
while True:
card_value = random.choice(list(Deck.keys()))
card = card_value + ' ' + random.choice(Suit)
if used_cards.get(card, 0) < Config['max_deck']:
used_cards.setdefault(card, 0)
used_cards[card] += 1
player[deck].append(card)
player[deck_value].append(card_value)
Config['deck_size'] += 1
break
```

winner()

```
for x in Players:
if Players[x]['Active']:
player = Players[x]
if Croupier['BJ']:
if player['BJ']:
```

This function is a combination of firstTurn() and playerTurn() from our version 1 of the program.

In the beginning it requests two cards for every single player by calling the drawCard() function and it does the same for croupiers cards successively. This means that it takes from the "Deck1" - the deck that we have defined - two cards for the player and then tow cards for the croupier for when it's time of this player to play with the croupier.

Then, as we do on our version 1, we print out the cards and the points of the player. While it selects the points based from the players cards, it is examining if by default if we have a blackjack. (NOTE: It examines if the croupier has a blackjack in the first place and if yes, we end the game since there is no need to look more.).

We are redirected to the winner function() from the firstTurn() function directly, if and only if the croupier has a blackjack.

Otherwise, we are moving to croupier() function and we let the croupier to decide if he will stand or hit. In case that he hits, we are calling again the drawCard() function. Then, in any case, we are moving to the winner() function to determine who wins and who loses in this round.

This function also checks if anyone has a blackjack (croupier and players) otherwise, it searches for the closest to 21.

```
for x in
if Player
print("Sc
Players[x
Players[x
print('Th
```

```
print('{}. You recover your bet of {}¢.'.format(player['Name'], player['Bet']))
if player['insurance'] and 0 < player.get('half_bet', 0):
print("{} your insurance covers your bet and you win {}¢".format(player['Name'],
player['half_bet'] * 2))
player['Money'] += player['half_bet'] * 2
else:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
else: # Croupier['BJ'] is False
if player['BJ']:
player['Money'] += (player['Bet'] * 3) // 2
print(
"{}. You got Blackjack and receive {}¢!".format(player['Name'], ((player['Bet'] * 3) // 2) +
player['Bet']))
if player['insurance'] and 0 < player.get('half_bet', 0):
print("{} You lost your insurance bet".format(player['Name']))
player['Money'] -= player['half_bet']
elif Croupier['Score1'] > 21 and player['BJ'] is False:
if player['Score1'] <= 21:
player['Money'] += player['Bet']
print('{}. You win! You get {}¢.'.format(player['Name'], player['Bet'] * 2))
if player.get('Score2', 22) <= 21:
player['Money'] += player['Bet']
print('{}. You win! You get {}¢ from hand #2.'.format(player['Name'], player['Bet'] * 2))
if player['Score1'] > 21:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
if player.get('Score2', 0) > 21:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢ from hand #2.".format(player['Name'], player['Bet'], ))
elif Croupier['Score1'] <= 21 and player['BJ'] is False:
if Croupier['Score1'] < player['Score1'] <= 21:
player['Money'] += player['Bet']
print('{}. You win! You get {}¢.'.format(player['Name'], player['Bet'] * 2))
if Croupier['Score1'] < player.get('Score2', 0) <= 21:
player['Money'] += player['Bet']
print('{}. You win! You get {}¢.'.format(player['Name'], player['Bet'] * 2))
if Croupier['Score1'] == player['Score1']:
print("{}. It's a tie, you recover your bet.".format(player['Name']))
if Croupier['Score1'] == player.get('Score2', 0):
print("{}. It's a tie, you recover your bet from hand #2.".format(player['Name']))
if player['Score1'] < Croupier['Score1']:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
if player.get('Score2', Croupier['Score1']) < Croupier['Score1']:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢ from hand #2.".format(player['Name'], player['Bet'], ))
if player['Score1'] > 21:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢.".format(player['Name'], player['Bet'], ))
if player.get('Score2', 0) > 21:
player['Money'] -= player['Bet']
print("Sorry, {}. You lost {}¢ from hand #2.".format(player['Name'], player['Bet'], ))
goodbye()
```

```
Players:
rs[x]['Money'] < Config['Minimum_bet'] and Players[x]['Active']:
orry, {}. You don't have enough funds to cover minimum bet. You only have left {}¢.".format(
x]['Name'], Players[x]['Money']))
x]['Active'] = False
hanks for playing! Come back another day!')
= 0
```

It's time to decide if any of the players has lost all their initial money and to determine how many left that they can use to play.

It gives the option to select if any player want to withdraw. In case that everyone wants to continue, then we type ' ' on the keyboard and we save the default values again onto our players sub-dictionary of each player.

If everyone withdraws, then the game ends.

goodBye()

```
inactive
for n in
if Player
inactive
if inacti
print('No
sys.exit
print('If
'next gam
out = inp
for x in
new_playe
if out ==
print('Fa
net = Pla
print('Ne
Players[n
goodbye()
if out ==
for n in
if Player
Players[n
Players[n
Players[n
Players[n
Players[n
Players[n
Players[n
Players[n
if Player
Players[n
Players[n
Players[n
Croupier
Croupier
Croupier
Croupier
Croupier
betting()
else:
print('No
goodbye()
```

Step 3

Time to introduce the 3 essential functions of the program. Their role determines that flow of the game since from them we are taking the points data, th
stand. While the filtration of their responses is done at the functions in step 2, without these essential functions, it is not possible to determine what we w

hit_stand()

```
while player['Active'] and player[state]:
val = valueCardSum(player[deck_value])
print('\n{}, it is your turn.'.format(player['Name']))
show_card_value(player, deck, deck_value)
if val > 21:
print("Bust! Your hand is over 21.")
player[score] = val
player[state] = False
elif val <= 21:
while True:
print("What do you want to do, hit or stand?")
choice = input("h/s: ")
if choice not in ['h', 's']:
```

print_deck() V1

```
for card in player[deck]:
print("- " + card)
```

valueCardsSum()

```
sum_card = 0
for y in card_values:
sum_card += Deck.get(y)
return sum_card
```

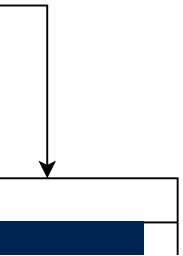
show_card_value()

```
print('Your cards are:')
```

```
= 0
 Players: # Reset all markers to default
rs[n]['Active'] is False:
 += 1
ive == Config['Number of players']:
o active players left. Thanks for playing!')
()
f any player would like to withdraw, please type your name. Leave it blank and we will continue with the '
me.')
out("Player: ")
 range(1, Config['Number of players'] + 1):
er = 'Player ' + str(x)
= Players[new_player].get('Name') and Players[new_player]['Active']:
arewell {}.'.format(Players[new_player]['Name']))
ayers[new_player]['Money'] - Config['Money']
et worth: {}¢'.format(net))
new_player]['Active'] = False
)
= '':
 Players: # Reset all markers to default
rs[n]['Active']: # If any player is Active, it will reset its markers
n]['Deck1'].clear()
n]['c_values1'].clear()
n]['Play'] = True
n]['Double'] = False
n]['Ace'] = False
n]['BJ'] = False
n]['insurance'] = False
rs[n].get('Deck2'):
n].pop('Deck2')
n].pop('c_values2')
n].pop('Score2')
['Deck1'].clear()
['c_values1'].clear()
['Play'] = True
['BJ'] = False
['Ace'] = False
)

o player found with the name {}.'.format(out))
)
```

e the value of cards in players' hands and the decision of the players to hit or
vant to do after the betting() function that we introduced above.

This function prints out the card of user. The V2 is modified to show card's body as mentioned on step 4.



```
print('Enter a valid input.')
if choice == 's':
player[state] = False
if player['Ace'] and (val + 10) <= 21:
player[score] = val + 10
elif player['Ace'] and (val + 10) > 21:
player[score] = val
else:
player[score] = val
break
elif choice == 'h':
drawCard(1, player, deck, deck_value)
break
```

```
print_deck(player, deck)
val = valueCardSum(player[deck_value
for card_v in player[deck_value]:
if card_v == 'Ace':
player['Ace'] = True
if player['Ace'] and (val + 10) <= 2
print('There is an Ace. Possible val
print('Hard value: ' + str(val))
print('Soft value: ' + str(val + 10)
else:
print('Its value is: {}'.format(val)
```

This functions is getting called first time on the firstTurn() function. According to the responses of the corresponding user for hit or stand ,it calculates the total value of cards (even we are on firstTurn or not) and if the user do not depose 21 points, then in case of hit, one new card is drawn for the user, otherwise, we are just updating the score of the user inside the Players dictionary value for key score.

Step 4

Making a more beautiful UI. Even if we are stick with command line for this blackjack, we can make it to be a little bit more beautiful by printing out a visu of card/s instead of just printing out something like "8 of Hearts".

In order to do it, we need to use a function that will translate the suite of the card into the corresponding symbol. This is what the cardVisualization() func about. I have also integrated the option to take the value of the card, to make it easier to print out the elements on the created list elem that we can see

cardVisualization()

```
cardOutput = card[0]
elem = []
elem.clear()
if "Hearts" in card:
elem.append("♥")
elif "Diamonds" in card:
elem.append("♦")
elif "Spades" in card:
elem.append("♠")
else:
elem.append("♣")
if "2" in cardOutput:
elem.append("2")
elif "3" in cardOutput:
elem.append("3")
elif "4" in cardOutput:
elem.append("4")
elif "5" in cardOutput:
elem.append("5")
elif "6" in cardOutput:
elem.append("6")
elif "7" in cardOutput:
elem.append("7")
elif "8" in cardOutput:
elem.append("8")
elif "9" in cardOutput:
elem.append("9")
elif "10" in cardOutput:
elem.append("10")
elif "Jack" in card:
elem.append("J")
```

print_deck()

```
for card in player[deck]:
identifiers = cardVisualization(card)
print('┌─────────┐')
print(f'| {identifiers[0]} |')
print('| |')
print(f'| {identifiers[1]} |')
print('| |')
print(f'| {identifiers[0]} |')
print('└─────────┘')
```

It's essential to give the else option while we are examining the value of the card, otherwise the program may give an error of a list out of range when we are running print_deck() function.

This is a workflow to depose the issue that appears on the debug mode of python, but we have never seen it to print out the "?" that we have defined.

This means that this workflow just completes the logique of the if operation for 'just in case'.

```
e])


21:
lues are:')

)

)
```

ualization

ction is all
below.

```
elif "Queen" in card:
elem.append("Q")
elif "Ace" in card:
elem.append("A")
elif "King" in card:
elem.append("K")
else:
elem.append("?")
return elem
```