

Osnova

Review test Intro

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [REVIEW TEST INTRO](#)

So you've made it :) You should have a very solid Python knowledge base. You're almost ready to get to the project part again.

However, before we set you off to dive into it, you should really go through the review test that is about to follow. It will test your knowledge from the previous 3 lessons. Depending on your score, you should consider revision of the particular lesson.

Good luck with the test as well as the project!

Review test 13-15

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [REVIEW TEST 13-15](#)

1/12

[seznam otázek](#)

What are the benefits of comprehensions?

☐ They are always more readable

☐ Faster than for loops

94% z Lekce 20

Osnova



ONE LINE OF CODE

PROJECT

Project description

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [PROJECT](#) / [PROJECT DESCRIPTION](#)

Our aim is to build a blackjack game. Here are some basic rules we will want to translate into Python code. We have also [a youtube link](#), where the game principles are demonstrated.:

— • • — —

94% z Lekce 20

- The player's first two cards (called a "blackjack" or "natural"), without a

Osnova

ja

- Reach a final score higher than the dealer without exceeding 21; or
- Let the dealer draw additional cards until their hand exceeds 21.

Each player that beats the dealer earns the amount of the bet.

Any player, who goes over 21, including the dealer, loses. This is called **bust**. Also if the player and the dealer did not go over 21, but the dealer reached more points, the dealer wins.

In case the player and the dealer have the same amount of points lower or equal to 21, the player can keep the bet. This is called **push**.

How the game runs?

Let's limit the number of players at one table to 6 + the dealer.

1. Game starts by dealing 1 card to each player and the dealer. Then one more card is dealt to each player excluding the dealer. Players will have 2 cards, meanwhile the dealer only one.
2. Now the dealer asks each player, whether they want to draw more cards from the deck. The player can decide to **split** or **double down** (see below for more information). Of course, if the player has already 21 after dealing the first two cards, the dealer does not ask there. This is called **blackjack**.
3. Once all the players have completed their hands, it is the dealer's turn. The dealer must hit until the cards total 17 or more points.

To sum up, players win by not busting and having a total higher than the dealer, or getting a blackjack without the dealer getting a blackjack. If the player and dealer have the same total, this is called a "push", and the player typically does not win or lose money on that hand. Otherwise, the dealer wins.

Card values

One deck consists of 52 or 6 x 52 cards. There are 4 card **suits**: clubs (♣), diamonds (♦), hearts

- Cards with numbers have the value corresponding to the number depicted on it.

Osnova

- King, queen, jack have value of 10.
- Aces can have value of 1 or 11. If the hand value would exceed 21 due to aces, the player can choose value 1 for them. We will automatically expect that if the value would exceed 21 and player has aces on hand, their value will be changed to 1.

Splitting

If the first two cards of a hand have the same value, the player can split them into two hands, by moving in a second bet equal to the first. The player then plays out the two separate hands in turn.

Doubling down

After the first two cards are dealt, the player is allowed to increase the initial bet by up to 100% in exchange for committing to stand after receiving exactly one more card. So the player will have 3 cards at the end.

How will we proceed?

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [PROJECT](#) / [HOW WILL WE PROCEED?](#)

If we haven't played blackjack before, it can seem to us that there are **quite a lot of rules** to be incorporated into the program. **It can seem even overwhelming and discouraging** to continue with the program.

The best way, how to overcome a complicated task is to **divide it into smaller doable tasks**. And here we will speak about how to do that.

So let's try to list actions or tasks, that our program will have to perform.

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [PROJECT](#) / [PROGRAM TASKS](#)

A possible list of tasks, that our program will have to perform.

1. create deck
2. shuffle deck
3. register players + setup a table
4. putting a bet
5. serve players
6. play game
7. check hand - how many points are there?
8. get the card value
9. draw from the deck
10. show the current situation at the table

Designing our program

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [PROJECT](#) / [DESIGNING OUR PROGRAM](#)

Besides being able to rerun the same code again and again, functions allow for **better abstraction and program structuring**.

So we could plan our program writing down the function names and inside the definition insert **pass** keyword.

Click to see our plan below if you want to see, how we wrote the code.

Osnova

Click to see our plan



```
1 def create_deck():  
2     pass
```

```
1 def shuffle_deck():  
2     pass
```

```
1 def register_players():  
2     pass
```

```
1 def put_bets():  
2     pass
```

```
1 def serve_players():  
2     pass
```

```
1 def play():  
2     pass
```

```
1 def check_hand():  
2     pass
```

```
1 def card_value():  
2     pass
```

```
1 def draw():  
2     pass
```

```
1 def show_table():  
2     pass
```

Osnova

SOLUTION

Creating card deck

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [CREATING CARD DECK](#)

One deck consists of 52 cards. We distinguish 4 card **suits**:

- clubs (♣),
- diamonds (♦),
- hearts (♥) and
- spades (♠)

Each suit includes an ace, king, queen and jack, each depicted with a symbol of its suit and ranks two through ten.

We first need to generate the deck of cards, in order we can play the game.

94% z Lekce 20

Below if you want to see, how we wrote the code.

Osnova

Click to see our solution

```
1 SUITS = ['♣', '♦', '♥', '♠']
2 FACES = ['10', 'jack', 'queen', 'king', 'ace']
3 RANKS = list(range(2,10)) + FACES
4
5 deck = []
6
7 for suit in SUITS:
8     for rank in RANKS:
9         deck.append(suit+str(rank))
```

- What if one day we decided to create another card game program?
- Or what if we needed to create the card deck somewhere else in our blackjack program?

It would be nice if we could package the code above in a **function**. If we put the code above into a function, **we will be able to run it repeatedly** from different parts of our program.

Let's return the result of our action

```
1 def create_deck():
2     SUITS = ['♣', '♦', '♥', '♠']
3     FACES = ['10', 'jack', 'queen', 'king', 'ace']
4     RANKS = list(range(2,10)) + FACES
5
6     deck = []
7
8     for suit in SUITS:
9         for rank in RANKS:
10             deck.append(suit+str(rank))
11
12     return deck
```


To finish the function definition, we need to **call the function**:
Osnova

```
1 create_deck()
```

Shuffling the deck

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [SHUFFLING THE DECK](#)

Our task is to create a function that will shuffle a card deck. Also, if the deck contains 6 x 52 cards, we would like to cut it after shuffling and introduce a blank card into the place of the cut.

- Will this function need some inputs?
- Will this function return any value?

The Shuffle and Cut

The dealer thoroughly shuffles portions of the pack until all the cards have been mixed and combined. He designates one of the players to cut, and the plastic insert card is placed so that the last 60 to 75 cards or so will not be used. (Not dealing to the bottom of all the cards makes it more difficult for professional card counters to operate effectively.)

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

So, we need to shuffle our card deck. But how would we write the shuffling algorithm? Does not matter, because somebody has already written the function for us. This function resides

The first thing we need to do is to load the `random.py` module and we load its content into our program using `import random`.

```
1 import random
```

Once the `random` module is imported, we can call its `shuffle()` function passing it an input in form of the deck:

```
1 deck = create_deck()
2 random.shuffle(deck)
```

So why don't we store the result returned by call to `shuffle()` function?

Besides simulating random shuffling, we need to simulate random selection of the place, where the deck will be cut. The cut has to appear somewhere 75 to 60 cards from the top of the deck. So we need to randomly choose an integer from a range:

```
1 def shuffle_deck(deck):
2     random.shuffle(deck)
3
4     # cut
5     if ' ' in deck:
6         deck.remove(' ')
7
8     if len(deck) > 52:
9
10        cut_start = len(deck) - 75
11        cut_end   = len(deck) - 60
12        cut_num = random.randint(cut_start, cut_end)
13        # insert blank card
14        deck = deck[cut_num:] + [' '] + deck[:cut_num]
15
16    return deck
```

Registering the players

... first ... Osnova ... playing. First we need to serve the cards to the players. But

- who are the players?
- what information do we need to keep about them?
- do we need to keep order in which we will interact with the players?
- max how many players can there be at one table?
- can players play more than one game at one table?

So let's try to create a function that registers the players asking for their names. Each player should get the default amount of money equal to 50. The function should return an object containing all the registered players, that will sit behind the table.

- do we want to keep **house** among the players?
- what will be the amount of money the house will have at its disposal?

We could run the function `register_players()` as follows:

```
1 players = register_players(2)
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

We want to keep requesting a player name until we have not reached the limit of players for one table. We can optionally include another call to `input()` function asking, whether we want to continue adding new players and thus allowing the user to list less than the limit number of players.

```
1 def register_players(limit_players=6):  
2     print('-'*40)
```

```
Osnova def register_players(limit_players):  
  
    8     player = input('ENTER PLAYER NAME: ')  
    9     players.append([player, 50])  
    10  
    11     if 'n' == input('Enter another? [y/n]: '):  
    12         break  
    13  
    14     players.append(['house', 10**3])  
    15     print('-'*40)  
    16  
    17     return players
```

Default parameters

In the function above we have used so called **default parameter**. Why is it called default? Because, if when calling the function, the input for this parameter is not provided, function will use the default value from the function definition. In our case value 6.

That means we could call the function above

- with argument:

```
1 players = register_players(2)
```

- without argument:

```
1 players = register_players()
```

Bet before being served

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [BET BEFORE BEING SERVED](#)

• ... we need the information about what cards has each player on his/her hand?

Osnova

- where will we store the information about the bet for the given game?
- how much has each player bet?

Maybe we could first ask the players to put their bets. That way we will know, who plays this game and who not. So let's create the function `put_bets()`.

Function `put_bet()` aim is to collect the bets, relate them to the player betting and create place for cards, that a player will have on hand. Once this is collected, the function will return a collection of players, their hands and bets.

Few questions to answer:

- how the function will know, who sits at the table?
- what will be the amount of min. bet amount?
- will we allow to bet for people, who do not have enough money to bet?
- what will we do with the people, who do not have enough money?
- how will the bet be reflected in player's personal bank?

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

We decided to represent all the players, that play the current game using a list of lists. Each inner list contains the following information about a player:

And why to keep them in a list?

1. The dealer will always refer to players in the same order.
2. Moreover, we will need to add new cards drawn from the deck. Thus we will need to

Furthermore, we know who to ask for bets based on the list of players, that we pass into the function. Osnova

```

1  def put_bets(players):
2      game = []
3
4      for i, (player, money) in enumerate(players[:-1]):
5
6          if players[i][1] < MIN_BET:
7              print(player + ", sorry, don't have enough money - "
8                    + str(money))
9              players.pop(i)
10             continue
11
12         while True:
13             bet = int(input(player + ', how much do you want to bet
14                           ('+ str(money) +')?: '))
15
16             if 10 <= bet <= players[i][1] :
17                 players[i][1] = players[i][1] - bet
18                 break
19
20         game.append([player, [], bet])
21
22     game.append(['house', [], '-'])
23     return game

```

So if we now ran this function in conjunction with player registration:

```

>>> players = blackjack.register_players()
-----
ENTER PLAYER NAME: Bob
Enter another? [y/n]: y
ENTER PLAYER NAME: Ann
Enter another? [y/n]: n
-----
>>> players
[['Bob', 50], ['Ann', 50], ['house', 1000]]
>>> game = blackjack.put_bets(players)

```

```
>>> game
      Osnova .      , ['Ann', [], 10], ['house', [], '-']]
>>>
```

Now we can serve

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [NOW WE CAN SERVE](#)

Once our players have their hands open and ready to collect cards in them, we can begin to deal the cards. The rule is that 2 cards are dealt to players and one to house in two rounds. In the first round everybody including the house will receive a card (house receives as the last one). Second card is then dealt only to players.

So questions concerning the function `serve()` are:

- what will be the function inputs?
- what will be the function outputs?

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution ▲

We want our function to serve **to players** from **the deck**. So these are the **inputs** the function will need to receive.

```
1 def serve(players,deck):
2     for i in range(2):
3         # second round of cards should not go to the dealer
```

Mutable data types

Actually the function does not have to return anything. It performs changes on the list of those currently playing the game. Changes on lists are performed **in place**. That means that if two variables (in our case the variable `game` sending the players into the function `serve` and the parameter `players` of the function `serve`) refer to the same object that is being changed, both variables will reflect the changes. Both variables see the same object. Therefore the outer variable does not have to be assigned the list from the `serve()` function. It already sees it.

Data types that can be changed in place are called **mutable**. Examples are lists, dictionaries, sets.

So now we can connect our function calls:

```
>>> players = blackjack.register_players()
-----
ENTER PLAYER NAME: Bob
Enter another? [y/n]: y
ENTER PLAYER NAME: Ann
Enter another? [y/n]: n
-----
>>> players
[['Bob', 50], ['Ann', 50], ['house', 1000]]
>>> game = blackjack.put_bets(players)
Bob, how much do you want to bet (50)?: 10
Ann, how much do you want to bet (50)?: 10
>>> game
[['Bob', [], 10], ['Ann', [], 10], ['house', [], '-']]
>>> deck = blackjack.shuffle_deck(blackjack.generate_deck())
>>> blackjack.serve(game, deck)
>>> game
[['Bob', ['♦king', '♦ace'], 10], ['Ann', ['♥2', '♣8'], 10], ['house',
['♠9'], '-']]]
```


Osnova do draw cards?

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [DO WE WANT TO DRAW CARDS?](#)

To answer this question we need to know:

- how much do I already have on my hand?
- for that we would need to know, what is the value of each card?

Dealer does not want to ask me if I want another card, if I just got a blackjack. Also it would be nice for other players, if the computer calculated the points each player has. Also, if the player wants to split or double down, this is the time to tell the dealer.

Dealer needs to ask every player the same things and that means that we will need a loop during which we will:

1. check our hand
2. decide, whether we want to split or double down or draw or stand
3. if we want another card, dealer has to deal

Let's put this loop into a function called `play()`. This function will perform all the actions to be done during one blackjack game:

1. put bets
2. serve cards
3. ask all the players, whether they want more cards
4. evaluate the game

We have already created functions that cover the first two points, now we will turn our attention to the point number three.

Also note, that we do not count with registering the players that are currently at the table. It is because players at one table can play multiple games. Therefore their registration should take place outside the game itself.

... and ...
Osnova ... below if you want to see, how we wrote the code.

Click to see our solution

Our **play** function has been filled with first three correct lines of code. Then follow lines, we still need to adjust. Therefore we first create a draft of the function.

Our first task will be to **check player's hand**.

```
1 def play(players):
2     game = put_bets(players)
3     deck = shuffle_deck(generate_deck())
4     serve(game,deck)
5
6     for player in game:
7
8         check_hand()
9
10        stand
11
12        split
13
14        double down
15
16        draw()
```

Checking the hand

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [CHECKING THE HAND](#)

In order a computer can calculate the amount of points on our hand, it first needs to be able to

1. Our first task is to create a function `card_value()` that will translate the card rank into a value. This function is called **Osnova**.
2. This function can then be used by the function `check_hand()` that will sum all the values returned by `card_value()`.

The function `check_hand()` should also adjust the total amount for cases, when the player has one or more aces on the hand and the total sum would trespass the value of 21. In that case ace value should be adjusted to 1 instead of 11.

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

Function `card_value()` is so called **helper function**. It is used inside the `check_hand()` function helping with calculating the result. We have decided to isolate code into `card_value()` for better readability. Also, we could need to use the same functionality somewhere else in the program.

```
1 def card_value(card):
2
3     value = card[1:]
4     if value in FACES[:-1]:
5         return 10
6     elif value == FACES[-1]:
7         return 11
8     else:
9         return int(value)
```

We have made use of the fact that variable `FACES` contains ranks which value is 10 or 11 (aces). But we could **refactor** our code to make it shorter and more **pythonic**.

- change content of the variable `FACES` to a dictionary
- the `card_value` function will then just request the value associated to the rank isolate

```
1  ----- = {'king':10, 'queen':10, 'jack':10, 'ace':11}
2  Osnova = [(range(2,11)) + list(FACES)]
3
4  def card_value(card):
5
6      value = card[1:]
7      return int(FACES.get(value,value))
```

Checking the hand

We just need to inspect each card on the hand. By default, for aces we count with the value of 11. Then we just check, whether the total amount on hand trespasses 21 and at the same time, the player has at least on ace.

```
1  def check_hand(hand):
2      total = 0
3      count_aces = 0
4
5      for card in hand:
6          val = card_value(card)
7          total = total + val
8
9          if val == 11:
10             count_aces = count_aces + 1
11
12     if total > 21 and count_aces > 0:
13         total = total - 10 * count_aces
14
15     return total
```

Asking the player

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [ASKING THE PLAYER](#)

```
1 def main(players):
2     Osnova = init_bets(players)
3     deck = shuffle_deck(generate_deck())
4     serve(game, deck)
5
6     for player in game:
7
8         on_hand = check_hand()
9
10        stand
11
12        split
13
14        double down
15
16        draw()
```

Now we need to ask the player few questions or check few facts before even asking:

1. whose turn is it? If it is the dealer, then just cards can be draw until 17 or more.
2. does the player have a blackjack?
3. does the player want to split?
4. or deos the player want to double down?
5. or just draw until bust or satisfied?
6. what information will we have about each player?

We will now need to set up few conditions...

Few notes

Player can split only if both cards on player's hands have the same rank. If the player decides to split, new hand and bet have to be created in the game. The newly created hand receives one of the two cards from the original hand. The bet has to be the same as the player's original bet. Do not forget to discount it from the player's bank.

to double down, the player's bet has to be doubled and only one more card can be drawn. Double down is permitted only if the player has less than 12 points on the hand.

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

So currently we have the following code:

```
1 def play(players):
2     game = put_bets(players)
3     deck = shuffle_deck(generate_deck())
4     serve(game,deck)
5     show_table(game)
6
7     for i,(player,hand,bet) in enumerate(game):
8         on_hand = check_hand(hand)
```

We want to check, whether it is the dealer's turn to draw. In that case we draw cards from the deck until the dealer's hand achieve's the limit of 17 or higher:

```
1 if player == 'house':
2     while check_hand(game[i][1]) < 17:
3         game[i][1].append(deck.pop())
```

If the player had blackjack, we do not ask for any further actions

```
1 elif on_hand == 21: # blackjack
2     pass
```

If none of the previous is true, we can ask the player, whether to split. The split is permitted only if the player has enough money in the bank as well as the two cards on the hand are of the same rank:

Osnova

```

~
'y' == input(player + ', do you want to split? [y/n]: '):

4         game.insert(i+1,(player,[hand.pop()],bet))
5         players[i][-1] = players[i][-1] - bet
6
7         for _, hand, __ in game[i: i+2]:
8             draw(deck, player, hand)

```

If not splitting, we can ask whether the player wants to double down. Again enough credit is a prerequisite as well as the number of points on hand cannot exceed 11:

```

1 elif on_hand <= 11 and players[i][-1] >= bet and \
2     'y' == input(player + ', do you want to double down?
[y/n]: '):
3
4     game[i][-1] = game[i][-1] * 2
5     players[i][-1] = players[i][-1] - bet
6     game[i][1].append(deck.pop())
7     print(player,hand,bet)

```

Or if the player has less than 21 points, more cards can be drawn:

```

1 elif on_hand < 21:
2
3     draw(deck, player, hand)

```

We should now implement the `draw()` function.

Drawing a card

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [DRAWING A CARD](#)

Dealer repeatedly asks the player, if another card should be drawn from the deck. The loop should terminate if the player can decide to stand (stop drawing) or the amount of points on hand has exceeded 21

When a card is taken from the deck, the current amount of points as well as the cards on hand are passed to the player.

Osnova

- what will be the inputs?
- will there be any outputs?

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

We use infinite loop and check for user's input as well as for the amount of points on hand. There are two scenarios, when we want to interrupt drawing based on the amount of points:

- when the player exceeds 21 points
- when the player reaches 21 points

Inputs

As we want our questions to be a bit personalized, we use `player`'s name. Then we keep popping cards from the `deck` and pass it to `hand` list.

Outputs

This function does not return anything. This is due to the fact that it modifies a list `hand` that is also referenced by the variable in the function from where `draw()` has been called.

```
1 def draw(deck, player, hand):
2     while 'y' == input(player + ': you have ' +
3                         str(check_hand(hand))
4                         + ', want another card? [y/n]: '):
```



```
7
Osnova      it('-'*40)
9          print('Cards:', hand, ': ', on_hand)
10
11          if on_hand > 21:
12              print(player, 'bust!')
13              break
14          elif on_hand == 21:
15              break
16
17          print('-'*40)
```

Game evaluation

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [GAME EVALUATION](#)

Once all the players have taken their cards from the deck, it is time to evaluate wins and losses.

So when does the player win and how much does the player earn?

1. if got blackjack and the dealer did not get blackjack - player earns 1.5 of the original bet
2. if the dealer busted and the player's points do not exceed 21 - player earns the amount of the original bet
3. if the player has more points than the dealer not exceeding 21 - player earns the amount of the original bet

When does the player loose?

1. when player's cards exceeded 21
2. when the dealer has more on hand than the player and none of them exceed 21

No gain nor loss if the player and the dealer have the same amount of points not exceeding 21.

```

1     Osnova 'players):
2
3     ....
4
5     return evaluate_game(game, players)

```

The function `evaluate_game()` should return the list of players at the table with their current balances:

```
[[ 'Bob', 60], [ 'Ann', 40], [ 'house', 1000]]
```

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution 

```

1  def evaluate_game(game, players):
2
3      show_table(game)
4
5      house = game.pop()
6      house_on_hand = check_hand(house[1])
7
8      print('House had ' + str(check_hand(house[1])))
9
10
11     for i,(player,hand,bet) in enumerate(game):
12         on_hand = check_hand(hand)
13         if (on_hand == 21 and len(hand)==2) and \
14             not(house_on_hand == 21 and len(house[1])==2): #
15             blackjack - x1.5
16
17         players[i][-1] = players[i][-1] + bet + bet * 1.5

```

Osnova

```

20         elif on_hand > 21 or on_hand < house_on_hand <= 21: # bust
           loose money
21
22         # already discounted from the player
23         players[-1][-1] = players[-1][-1] + bet
24         print(player.upper() + ", you've LOST! Your current
bank: " + str(players[i][-1]))
25
26
27         elif on_hand > house_on_hand or on_hand <= 21 <
house_on_hand: # house discount add 2:1 to player
28
29         players[i][-1] = players[i][-1] + bet + bet
30         print(player.upper() + ", HOUSE BEATEN! Your current
bank: " + str(players[i][-1]))
31
32
33         elif on_hand == house_on_hand: # push, the bet stays
34
35         players[i][-1] = players[i][-1] + bet
36         print(player.upper() + ", TIE! Your current bank: " +
str(players[i][-1]))
37
38
39     return players

```

Allowing to play multiple games

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [ALLOWING TO PLAY MULTIPLE GAMES](#)

In order the program runs until no player has money to bet or until the player decides to terminate the game we need to put the `play()` function into a loop. So games will be played

There's one more thing we'd like to add to this function to be created `table()`. This will be the function we will call to start the program can be run.

Code Solution

Use dropdown feature below if you want to see, how we wrote the code.

Click to see our solution

```
1 def table():
2
3     players = register_players()
4
5     while len(players) > 1:
6         play(players)
7         print('-' * 40)
8
9     if len(players) <= 1 and 'n' == input('Another game? [y/n]:
10 ):
11         print('Good bye')
12         break
13
14     print('-' * 40)
```

Entire Code

[PYTHON ACADEMY](#) / [PROJECT 5: BLACK JACK \[H\]](#) / [SOLUTION](#) / [ENTIRE CODE](#)

So you'd like to see the whole thing, huh? Alright :) Below you can find:

1 def main():

3. ... starts the game.
Osnova

Individual tasks

Game set up

```
1 import random
2
3 SUITS = ['♣', '♦', '♥', '♠']
4 FACES = ['10', 'jack', 'queen', 'king', 'ace']
5 RANKS = list(range(2,10)) + FACES
6 MIN_BET = 10
```

Create deck

```
1 def generate_deck(): # Later add default deck = 1
2     # clubs (♣), diamonds (♦), hearts (♥) and spades (♠)
3     deck = []
4
5     for suit in SUITS:
6         for rank in RANKS:
7             deck.append(suit+str(rank))
8
9     return deck
```

Shuffle deck

```
~
Osnova
1     random.shuffle(deck)
2
3
4     # cut
5     if ' ' in deck:
6         deck.remove(' ')
7
8     if len(deck) > 52:
9
10        cut_start = len(deck) - 75
11        cut_end   = len(deck) - 60
12        cut_num = random.randint(cut_start, cut_end)
13        # insert blank card
14        deck = deck[cut_num:] + [' '] + deck[:cut_num]
15
16    return deck
```

Register players

```
1 def register_players(limit_players=6):
2     print('-'*40)
3     players = []
4
5     while len(players) <= limit_players:
6
7         player = input('ENTER PLAYER NAME: ')
8         players.append([player, 50])
9         if 'n' == input('Enter another? [y/n]: '):
10             break
11
12     players.append(['house', 10**3])
13     print('-'*40)
14
15     return players
```

Osnova

```
1 def put_bets(players):
2     game = []
3
4     for i, (player, money) in enumerate(players[:-1]):
5         if players[i][1] < MIN_BET:
6             print(player + ", sorry, don't have enough money - "
7 + str(money))
8             players.pop(i)
9             continue
10
11         while True:
12             bet = int(input(player + ', how much do you want to bet
13 ('+ str(money) +')?: '))
14
15             if 10 <= bet <= players[i][1] :
16                 players[i][1] = players[i][1] - bet
17                 break
18
19         game.append([player,[], bet])
20
21     game.append(['house',[], '-'])
22     return game
```

Serve players

```
1 def serve(players,deck):
2     for i in range(2):
3         # second round of cards should not go to the dealer
4         p = players if i == 0 else players[:-1]
```

Osnova

Get card value and Check hand

```
1  def card_value(card):
2
3      value = card[1:]
4      if value in FACES[:-1]:
5          return 10
6      elif value == FACES[-1]:
7          return 11
8      else:
9          return int(value)
10 # =====
11 FACES = {'king':10,'queen':10,'jack':10, 'ace':11}
12 RANKS = list(range(2,11)) + list(FACES)
13
14 def card_value(card):
15
16     value = card[1:]
17     return int(FACES.get(value,value))
18 # =====
19
20 def check_hand(hand):
21     total = 0
22     count_aces = 0
23
24     for card in hand:
25         val = card_value(card)
26         total = total + val
27
28         if val == 11:
29             count_aces = count_aces + 1
30
31     if total > 21 and count_aces > 0:
```


total
Osnova

Draw

```
1 def draw(deck, player, hand):
2     while 'y' == input(player + ': you have ' +
3       str(check_hand(hand))
4       + ', want another card? [y/n]: '):
5         hand.append(deck.pop())
6         on_hand = check_hand(hand)
7
8         print('-'*40)
9         print('Cards:', hand, ': ', on_hand)
10
11        if on_hand > 21:
12            print(player, 'bust!')
13            break
14        elif on_hand == 21:
15            break
16
17    print('-'*40)
```

Show table

```
1 def show_table(game):
2
3     print('-' * 40)
4
```

Osnova

```
8 print('-' * 40)
```

Play

```
1 def play(players):
2     game = put_bets(players)
3     deck = shuffle_deck(generate_deck())
4     serve(game, deck)
5     show_table(game)
6
7     for i, (player, hand, bet) in enumerate(game):
8         on_hand = check_hand(hand)
9
10        if player == 'house':
11            while check_hand(game[i][1]) < 17:
12                game[i][1].append(deck.pop())
13
14        elif on_hand == 21:
15            pass
16
17        elif hand[0] == hand[1] and players[i][-1] >= bet and \
18            'y' == input(player + ', do you want to split? [y/n]: '):
19
20            game.insert(i+1, (player, [hand.pop()], bet))
21            players[i][-1] = players[i][-1] - bet
22
23            for _, hand, __ in game[i: i+2]:
24                draw(deck, player, hand)
25
26
27        elif on_hand <= 11 and players[i][-1] >= bet and \
28            'y' == input(player + ', do you want to double down? [y/n]: '):
```

Osnova

```

30         game[i][-1] = game[i][-1] * 2
31         players[i][-1] = players[i][-1] - bet
32         game[i][1].append(deck.pop())
33         print(player, hand, bet)
34
35     elif on_hand < 21:
36
37         draw(deck, player, hand)
38
39     return evaluate_game(game, players)

```

Evaluate the game

```

1  def evaluate_game(game, players):
2      show_table(game)
3      house = game.pop()
4      house_on_hand = check_hand(house[1])
5      print('House had ' + str(check_hand(house[1])))
6
7
8      for i, (player, hand, bet) in enumerate(game):
9          on_hand = check_hand(hand)
10
11         if (on_hand == 21 and len(hand)==2) and \
12             not(house_on_hand == 21 and len(house[1])==2): #
13             blackjack - x1.5
14
15             players[i][-1] = players[i][-1] + bet + bet * 1.5
16             print(player.upper() + ', got BLACKJACK! Your current
17             bank: ' + str(players[i][-1]))
18
19         elif on_hand > 21 or on_hand < house_on_hand <=21: # bust
20             loose money

```

```

22         players[-1][-1] = players[-1][-1] + bet
Osnova     print(player.upper() + ", you've LOST! Your current
23         bank: " + str(players[i][-1]))
24
25         elif on_hand > house_on_hand or on_hand <= 21 <
house_on_hand: # house discount, add 2:1 to player
26
27             players[i][-1] = players[i][-1] + bet + bet
28             print(player.upper() + ", HOUSE BEATEN! Your current
bank: " + str(players[i][-1]))
29
30
31         elif on_hand == house_on_hand: # push, the bet stays
32
33             players[i][-1] = players[i][-1] + bet
34             print(player.upper() + ", TIE! Your current bank: " +
str(players[i][-1]))
35
36
37     return players

```

Set up table for multiple games

```

1  def table():
2
3      players = register_players()
4
5      while len(players) > 1:
6          play(players)
7          print('-' * 40)
8
9      if len(players) <= 1 and 'n' == input('Another game? [y/n]:
'\.

```

Osnova

it(' - ' * 40)

Entire code & Game start

Entire code

```
1 import random
2
3 SUITS = ['♣', '♦', '♥', '♠']
4 FACES = ['10', 'jack', 'queen', 'king', 'ace']
5 RANKS = list(range(2,10)) + FACES
6 MIN_BET = 10
```

```
1 def generate_deck(): # later add default deck = 1
2     # clubs (♣), diamonds (♦), hearts (♥) and spades (♠)
3     deck = []
4
5     for suit in SUITS:
6         for rank in RANKS:
7             deck.append(suit+str(rank))
8
9     return deck
```

```
1 def shuffle_deck(deck):
2     random.shuffle(deck)
3
4     # cut
5     if ' ' in deck:
6         deck.remove(' ')
7
8     if len(deck) > 52:
9
10         cut_start = len(deck) - 75
```

94% z Lekce 20

```

13         # insert blank card
    Osnova      : = deck[cut_num:] + [' '] + deck[:cut_num]
15
16     return deck

```

```

1  def register_players(limit_players=6):
2      print('-'*40)
3      players = []
4
5      while len(players) <= limit_players:
6
7          player = input('ENTER PLAYER NAME: ')
8          players.append([player, 50])
9          if 'n' == input('Enter another? [y/n]: '):
10             break
11
12     players.append(['house', 10**3])
13     print('-'*40)
14
15     return players

```

```

1  def put_bets(players):
2      game = []
3
4      for i, (player, money) in enumerate(players[:-1]):
5          if players[i][1] < MIN_BET:
6              print(player + ", sorry, don't have enough money - "
7                    + str(money))
8              players.pop(i)
9              continue
10
11         while True:
12             bet = int(input(player + ', how much do you want to bet
13             ('+ str(money) +')?: '))
14
15             if 10 <= bet <= players[i][1] :
16                 players[i][1] = players[i][1] - bet
17                 break
18
19

```

```
19 game.append(['house',[],'-'])
```

Osnova game

```
1 def serve(players,deck):
2     for i in range(2):
3         # second round of cards should not go to the dealer
4         p = players if i == 0 else players[: -1]
5
6         for _, hand, __ in p:
7             hand.append(deck.pop())
```

```
1 def card_value(card):
2
3     value = card[1:]
4     if value in FACES[: -1]:
5         return 10
6     elif value == FACES[-1]:
7         return 11
8     else:
9         return int(value)
10 # =====
11 FACES = {'king':10,'queen':10,'jack':10, 'ace':11}
12 RANKS = list(range(2,11)) + list(FACES)
13
14 def card_value(card):
15
16     value = card[1:]
17     return int(FACES.get(value,value))
18 # =====
19
20
21 def check_hand(hand):
22     total = 0
23     count_aces = 0
24
25     for card in hand:
26         val = card_value(card)
27         total = total + val
28
```

```
Osnova ' ' > 21 and count_aces > 0:
```

```
33         total = total - 10 * count_aces
34
35     return total
```

```
1  def draw(deck, player, hand):
2      while 'y' == input(player + ': you have ' +
   str(check_hand(hand))
3          + ', want another card? [y/n]: '):
4
5          hand.append(deck.pop())
6          on_hand = check_hand(hand)
7
8          print('-'*40)
9          print('Cards:', hand, ':', on_hand)
10
11         if on_hand > 21:
12             print(player, 'bust!')
13             break
14         elif on_hand == 21:
15             break
16
17     print('-'*40)
```

```
1  def show_table(game):
2
3      print('-' * 40)
4
5      for player, hand, money in game:
6          print(player.upper(), '|', hand, '|', check_hand(hand), '| $',
   money)
7
8      print('-' * 40)
```

```
1  def play(players):
2      game = put_bets(players)
3      deck = shuffle deck(generate deck())
```



```

6
Osnova    (player,hand,bet) in enumerate(game):
8        on_hand = check_hand(hand)
9
10        if player == 'house':
11            while check_hand(game[i][1]) < 17:
12                game[i][1].append(deck.pop())
13
14        elif on_hand == 21:
15            pass
16
17        elif hand[0] == hand[1] and players[i][-1] >= bet and \
18            'y' == input(player + ', do you want to split? [y/n]: '):
19
20            game.insert(i+1,(player,[hand.pop()],bet))
21            players[i][-1] = players[i][-1] - bet
22
23        for _, hand, __ in game[i: i+2]:
24            draw(deck, player, hand)
25
26
27        elif on_hand <= 11 and players[i][-1] >= bet and \
28            'y' == input(player + ', do you want to double down?
[y/n]: '):
29
30            game[i][-1] = game[i][-1] * 2
31            players[i][-1] = players[i][-1] - bet
32            game[i][1].append(deck.pop())
33            print(player,hand,bet)
34
35        elif on_hand < 21:
36
37            draw(deck, player, hand)
38
39    return evaluate_game(game, players)

1 def evaluate_game(game, players):
2     show_table(game)
```

```
print('HOUSE HAND: ' + str(check_hand(house[1])))
```

Osnova

```

8     for i,(player,hand,bet) in enumerate(game):
9         on_hand = check_hand(hand)
10
11         if (on_hand == 21 and len(hand)==2) and \
12             not(house_on_hand == 21 and len(house[1])==2): #
13             blackjack - x1.5
14             players[i][-1] = players[i][-1] + bet + bet * 1.5
15             print(player.upper() + ', got BLACKJACK! Your current
16             bank: ' + str(players[i][-1]))
17
18         elif on_hand > 21 or on_hand < house_on_hand <=21: # bust
19             Loose money
20             # already discounted from the player
21             players[-1][-1] = players[-1][-1] + bet
22             print(player.upper() + ", you've LOST! Your current
23             bank: " + str(players[i][-1]))
24
25         elif on_hand > house_on_hand or on_hand <= 21 <
26             house_on_hand: # house discount, add 2:1 to player
27             players[i][-1] = players[i][-1] + bet + bet
28             print(player.upper() + ", HOUSE BEATEN! Your current
29             bank: " + str(players[i][-1]))
30
31         elif on_hand == house_on_hand: # push, the bet stays
32
33             players[i][-1] = players[i][-1] + bet
34             print(player.upper() + ", TIE! Your current bank: " +
35             str(players[i][-1]))
36

```

Osnova

```
3     players = register_players()
4
5     while len(players) > 1:
6         play(players)
7         print('-' * 40)
8
9         if len(players) <= 1 and 'n' == input('Another game? [y/n]:
10         '):
11             print('Good bye')
12             break
13     print('-' * 40)
```

START THE GAME 

```
1 table()
```

DALŠÍ LEKCE