SECTION 11: MILESTONE PROJECT - 2 hours 18 minutes, 12 sections

- **11/12 Solution Walkthrough - Functions for Game Play**
    - -> creating the functions which will be combined into the code for the game <- **we are combining several smaller functions into the larger program**

    - **-> a function called take_bet**
        - -> the argument of this is chips
        - -> there is a while loop
        - -> while it's True -> in other words, while the game is being played
            - -> it's asking for an integer input
            - -> then they try -> chips.bet <- betting and insuring that the input is in the right format
                - then we alter the logic according to the balance on the chip

    - **-> she's then defined a function called take_bet**
        - -> accepting an input from the user
        - -> the input needs to be in the right format -> she's made sure this happens with a while loop
        - -> and given certain outputs depending on what the user inputs

    - **-> then another function called hit**
        - -> the input is the deck and the hand
        - -> it adds a card to the hand and checks for an ace adjustment

    - **-> then another function hit_or_stand**
        - -> global playing
        - -> while True -> it's asking for an input
        - -> if the input is in a certain format -> it asks for the input again
        - -> **if the user inputs something e.g hit instead of h -> she is writing out the different things they could have entered -> and making sure that they lead to the same input in the computer (using a series of if / else loops)**

    - **-> then writing functions to display the cards**
        - -> the dealer's first card is hidden
        - -> there are two situations -> **defining functions for the different cases**

            - **showing some of the cards**
                - show one of the dealer's cards and showing all of the player's cards / hand
                - -> she's printing out statements -> the dealer's hand, that the first card is hidden
                - -> the dealer has two cards -> from an OOP perspective
                - -> dealer.cards[1] <- printing the second card of the dealer's hand

                - **-> then writing a for loop -> showing 2 cards of the player's hand**
                    - for card in player.cards:
                        - print(card) -> not sure why they didn't just print cards (they iterated through the entire list instead)

                - **-> then doing a similar process for the dealer's cards**
                    - printing the value of the dealer's cards -> **f string literals**
                    - **-> f"Text here: {name_of_variable}"**

        - **-> the same as the .format method in Python (it's a method i.e a function .format()**
        - -> looping through all of the cards in the dealer's hand

    - -> *****
        - **this is to print all of the cards in the collection, when you don't want to use a for loop**
        - **-> print("/n Dealer's hand: ",*dealer.cards) <- this loops through every item and prints it out (* is for every / all)**
            - **-> this is the second argument of the print function -> loop through all of the items in this list and print them all out**
        - **print("/n Dealer's hand: ",*dealer.cards,sep='\n') <- the sep is to add in a new line every time a new dealer card is printed out**
            - **-> you could also *items e.g where items is an array**

    - **-> writing functions which handle each of the game scenarios**
        - player_busts
        - -> player_wins
        - -> dealer_wins
        - -> dealer_busts
        - -> push
            - both of them tie <- **printing out different messages for each of the cases (the player or the computer wins e.g)**
    - -> the last thing is to run the logic on the cards (to combine the functions into the program)

- **then showing all of the cards**
    - -> showing all of the dealer's cards and all of the player's cards
    - -> who won the match
    - -> calculating and displaying the value (e.g a Jack + a King = 20)

- **-> overview**
    - creating the functions which will be called in the next lecture
    - -> a betting function, one which takes hits, a hit / stands one, then a few functions for the different cases / outcomes of the game