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

- **7/12 Game Logic - Part Three**
    - -> so we have the cards of the two players and one has been removed and placed on the table
    - **-> there are three solutions**
        - player 1>2
        - player 2<1
        - player 1==2 (a "war")
    - **-> writing this in an if / elif statements**
        - doing this under a while loop
        - -> stating "at war" equals False if the terms on the cards match
        - -> the rules
            - each player needs to draw 5 extra cards if there is a tie -> 5 of their cards are added to the deck
            - -> normally this is 3 -> but she's chosen 5 to make the game run faster
            - -> the player looses if they don't have at least 5 cards left to put in in the event that the two cards selected are matching (aka, it's a 'war')
            - **-> you can repeat the game again and again in a simulation -> and plot the result -> this can help you improve the game**
                - **some of the cases are where it gets stuck in a while loop**
    - **-> writing out the pseudocode / a flow diagram for the program**
        - **-> while there is a tie**
            - -> if / elif statements coding the different case scenarios -> either one players' hand is bigger or the other is
            - -> then if there is a war, either the players have enough cards left to draw 5 more, or they don't
            - -> so it's a series of ties (wars) or a single one
    - **-> in the .ipynb file**
        - under the code from the previous cell
        - -> at_war=True (in other words, it's a tie and the cards from each player match)
        - **-> while at_war:**
            - -> then she's checking that the last card in player1's deck is greater than that for player 2
            - **-> by the last card, it's [-1], the -1'th index of the array storing the player's cards**
            - -> it will always draw the last card
            - -> then adding cards according to which player had the higher ranking card
            - **-> essentially - coding the different possible scenarios for which combination of cards could be selected**
            - **-> all of these possible scenarios have been coded in an if, elif, else loop**
                - **-> if and elif require conditions (boolean in this case)**
                - **-> then breaking out of the loop**
                - **-> another common approach is to reuse the same block of code**
        - **-> so the thought process is**

    - **initialise everything**
        - create the players, deck, shuffle it, split it, put the game on, initialise counters, check are the players still eligible to play, start a new round, run a comparison check

    - **then coding the different possible outcomes for a player**
        - -> in a war, the comparison isn't happening (new cards are being added to the cards in the middle)

    - **running the code and checking it works -> this can be done through iterations (she's done this in a while loop in this example)**

- **-> OOP to create an application**
    - -> classes
    - -> deck classes
    - -> how the player class can hold instances of the deck class
    - -> and you can hold results in instances