MILESTONE PROJECT

We will recreate the card game called “War”.

Let’s have a quick overview of the game.

wikipedia.org/wiki/War_(card_game)

Comprehensive Python Programming

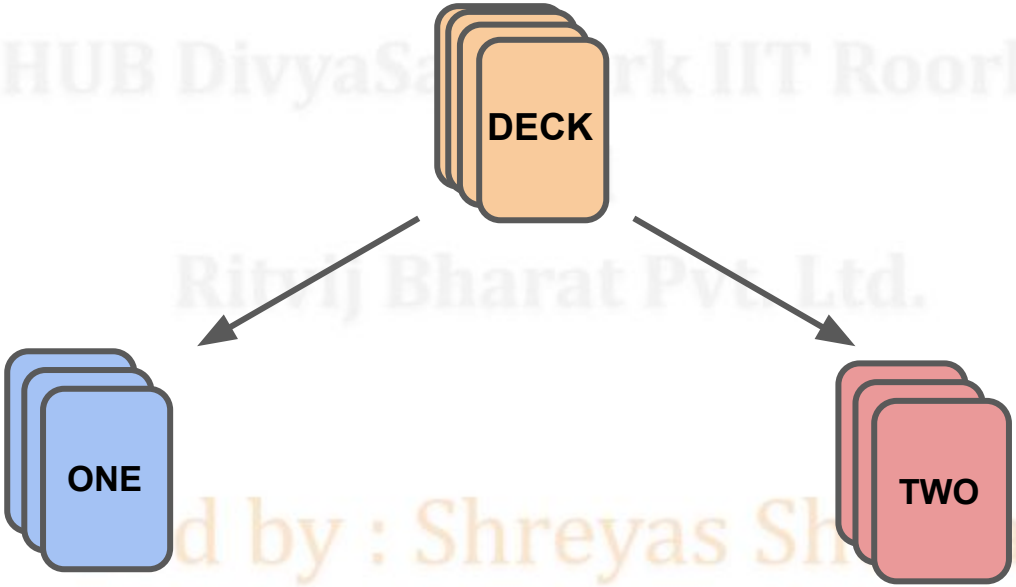
From Fundamentals to Advanced

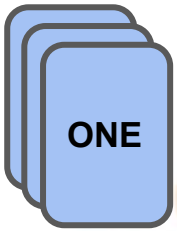
(21 Oct 2024 - 13 Nov 2024)

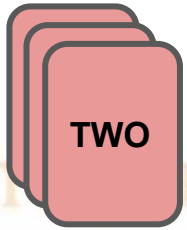
IHUB DivyaSampark IIT Roorkee

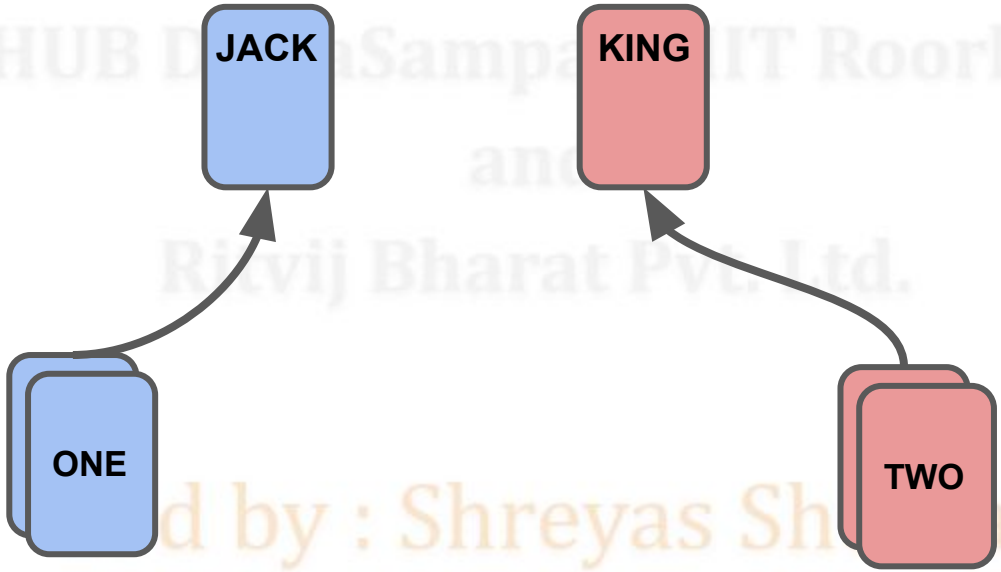
Ritvij Bharat Pvt. Ltd.

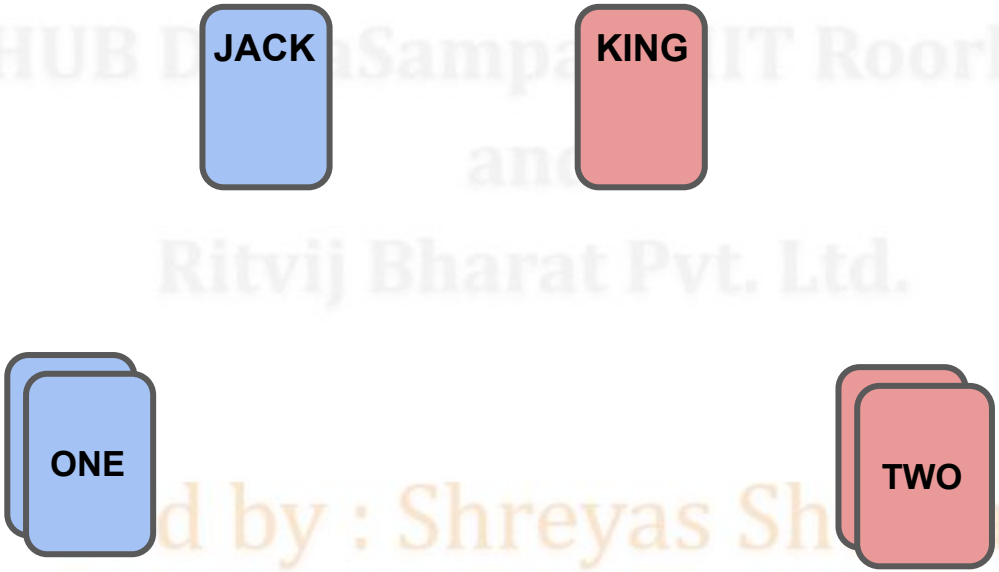
Led by : Shreyas Shukla

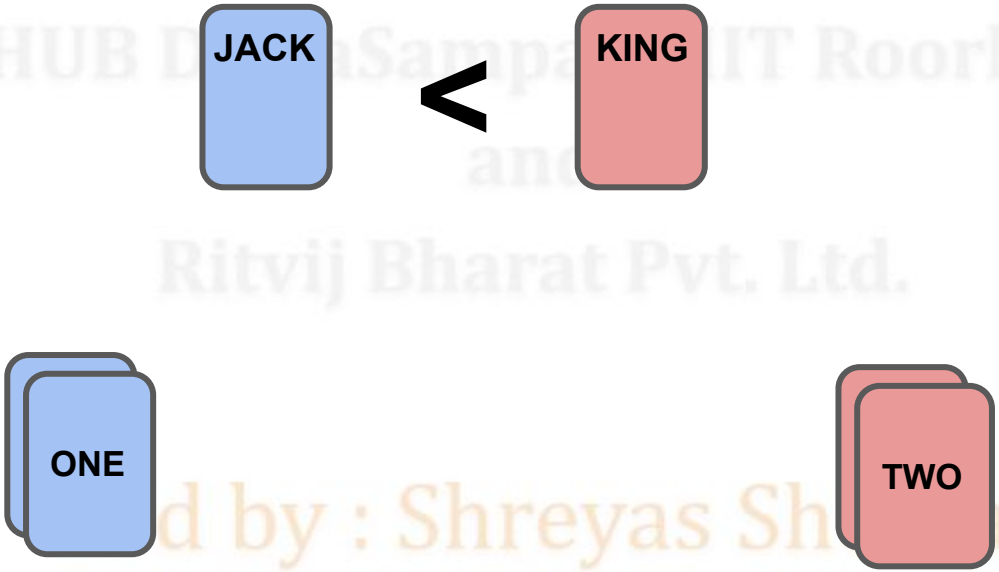


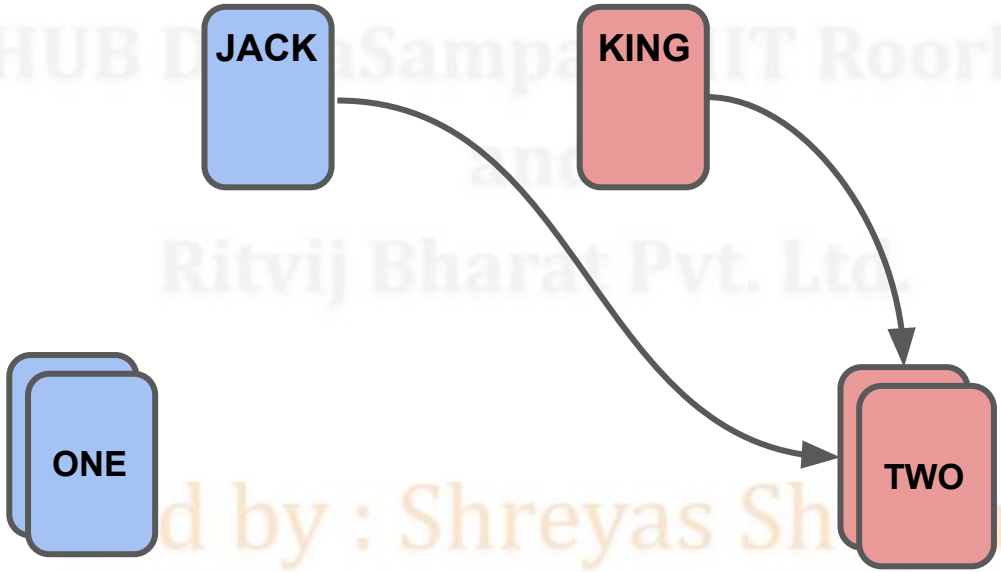












Comprehensive Python Programming

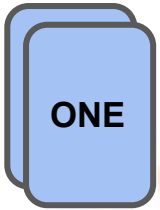
From Fundamentals to Advanced

(21 Oct 2024 - 13 Nov 2024)

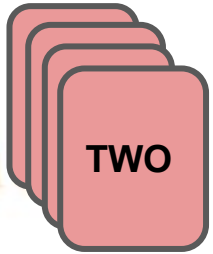
IHUB DivyaSampark IIT Roorkee

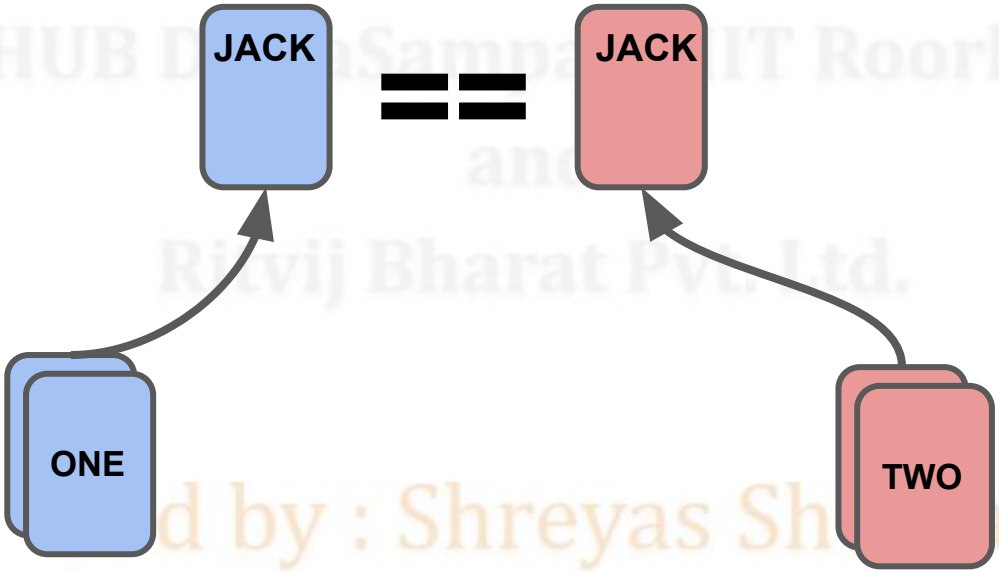
and

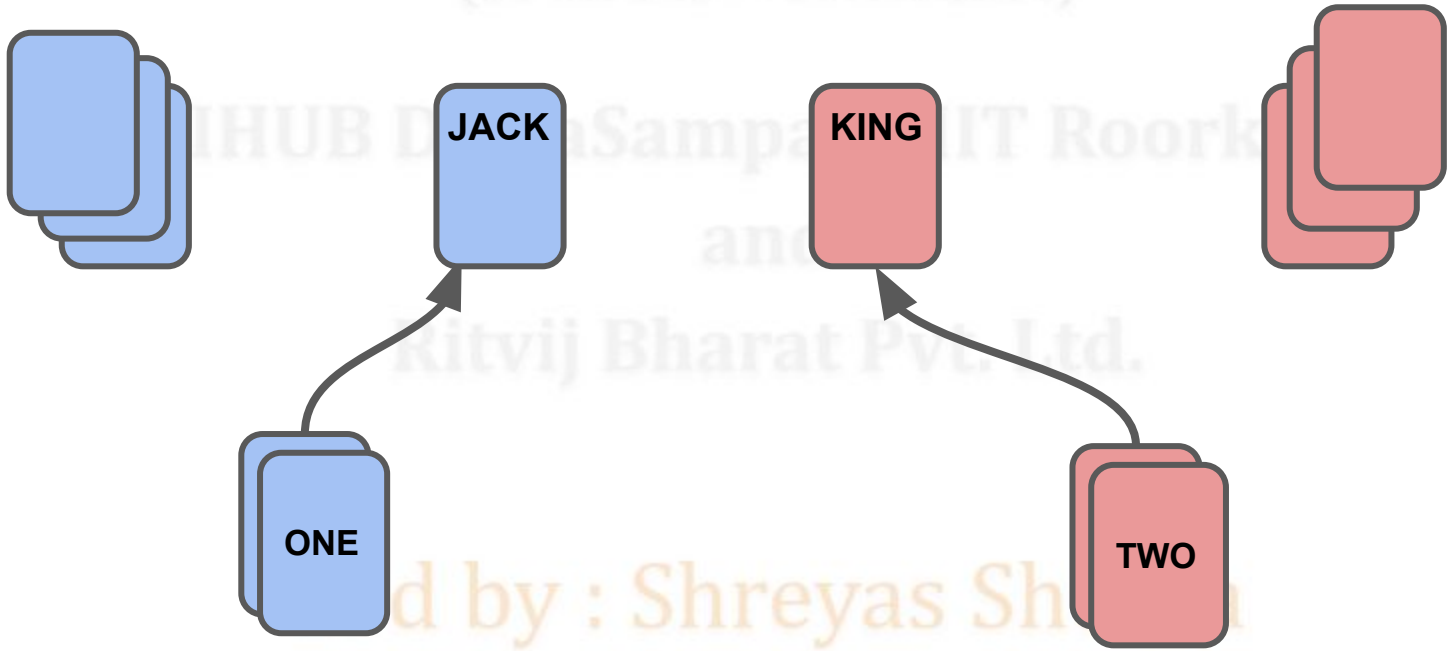
Ritvij Bharat Pvt. Ltd.

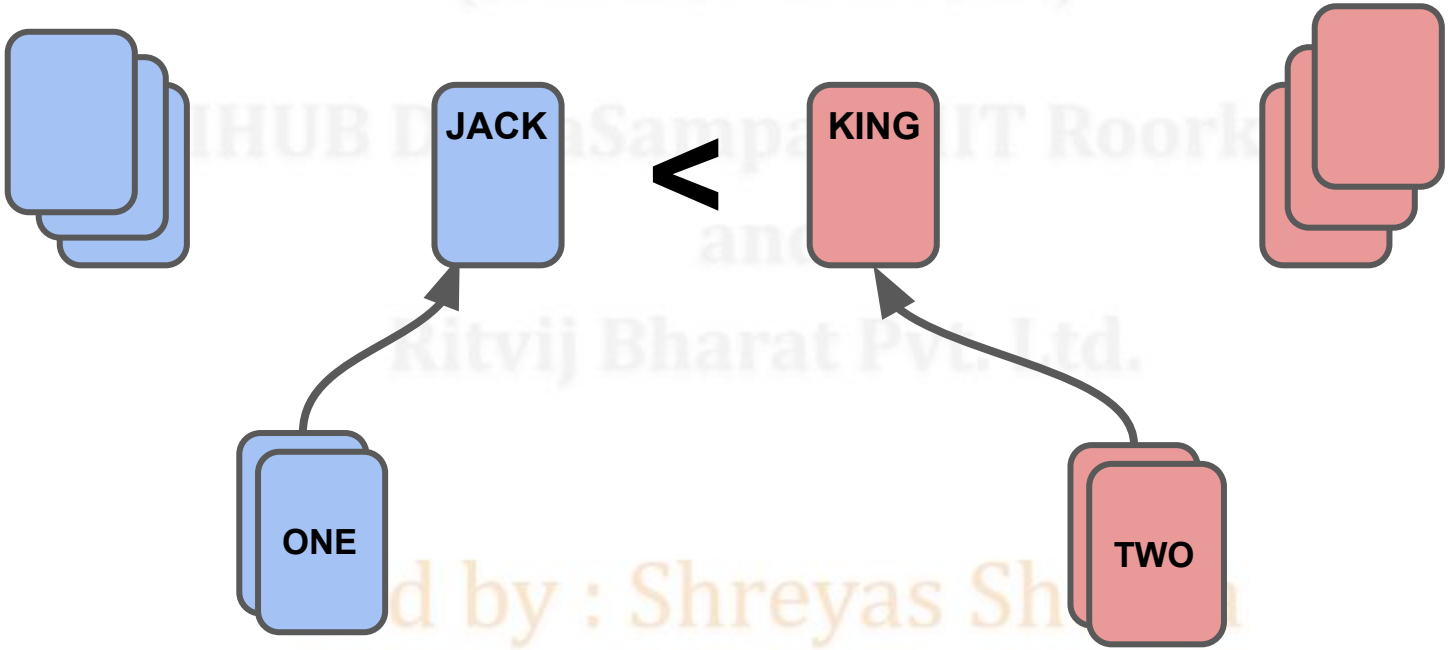


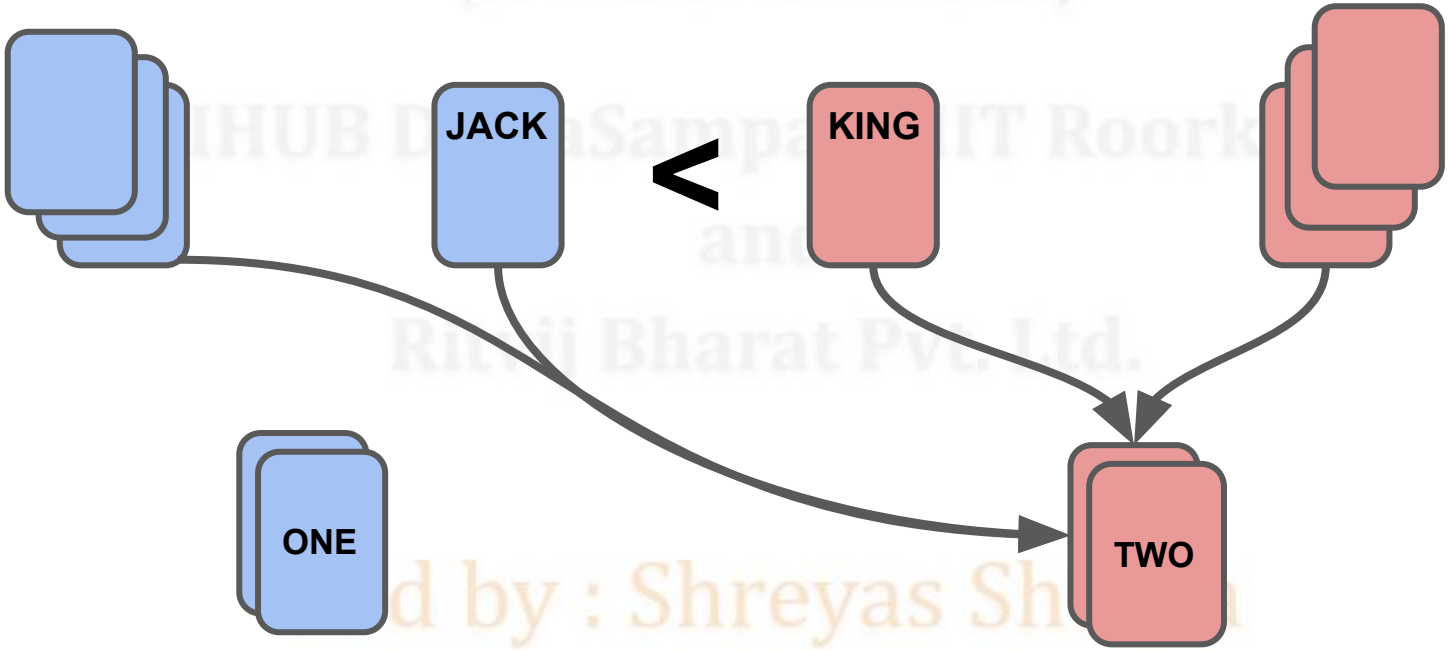
Created by : Shreyas Sh








JACK
<
KING
ONE
TWO



Comprehensive Python Programming

From Fundamentals to Advanced

(21 Oct 2024 - 13 Nov 2024)

IHUB DivyaSampark IIT Roorkee

and

Ritvij Bharat Pvt. Ltd.

d by : Shreyas Sh

Comprehensive Python Programming

From Fundamentals to Advanced

(21 Oct 2024 - 13 Nov 2024)

IHUB DivyaSampark IIT Roorkee

and

Ritvij Bharat Pvt. Ltd.

Led by : Shreyas Sh

To construct this game, we will create:

- Card Class
- Deck Class
- Player Class
- Game Logic

Let’s get started!

Card Class

Deck Class

Deck Class

- Instantiate a new deck
  - Create all 52 Card objects
  - Hold as a list of Card objects
- Shuffle a Deck through a method call
  - Random library shuffle() function
- Deal cards from the Deck object
  - Pop method from cards list

Deck Class

- Deck class holds a list of Card objects.
- This means the Deck class will return Card class object instances, not just normal python data types.

Let’s get started!

Player Class

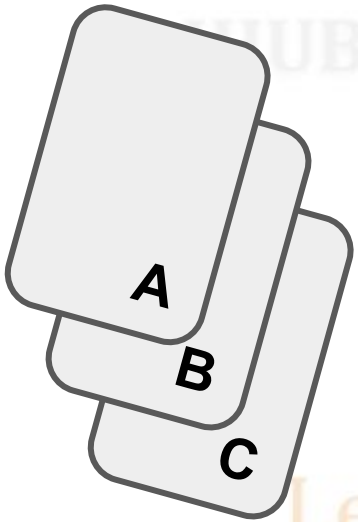
- This class will be used to hold a player's current list of cards.
- A player should be able to add or remove cards from their "hand" (list of Card objects).

We will want the player to be able to add a single card or multiple cards to their list, so we will also explore how to do this in one method call.

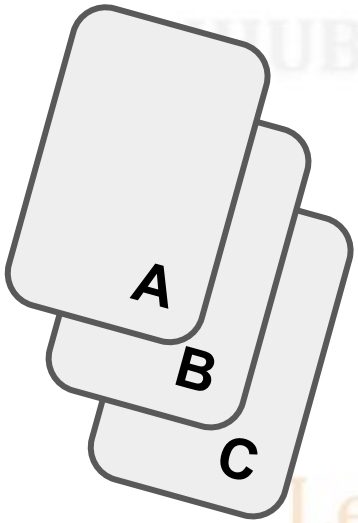
The last thing we need to think about is translating a Deck/Hand of cards with a top and bottom, to a Python list.

Let’s try to visualize this.

Player Class will have a self.all_cards list

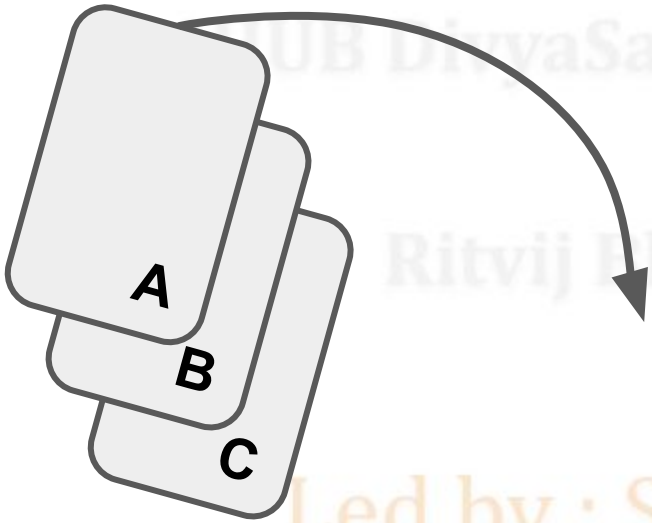


cards = [“A”, “B”, “C”]



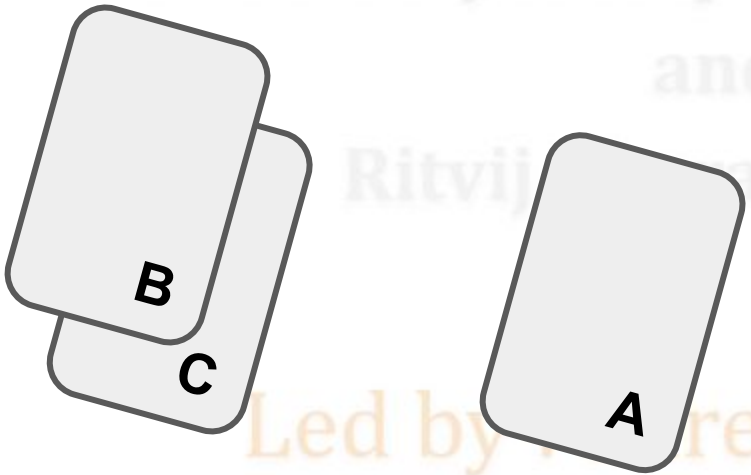
["A", "B", "C"]

A Player “plays” a card from the top



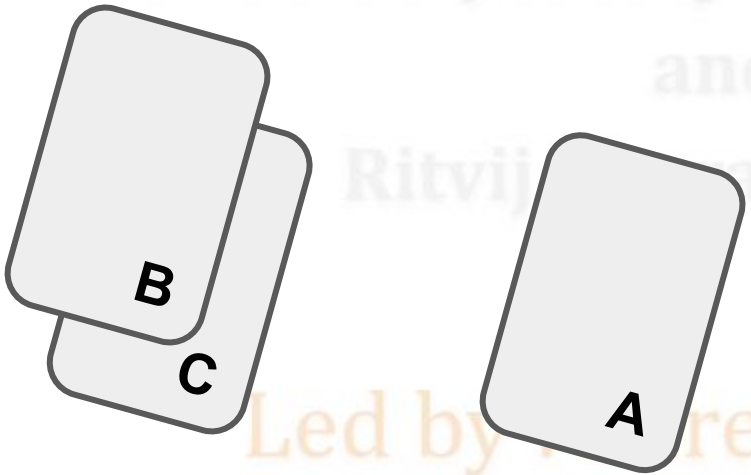
[“A”, “B”, “C”]

A Player “plays” a card from the top



[“A”, “B”, “C”]

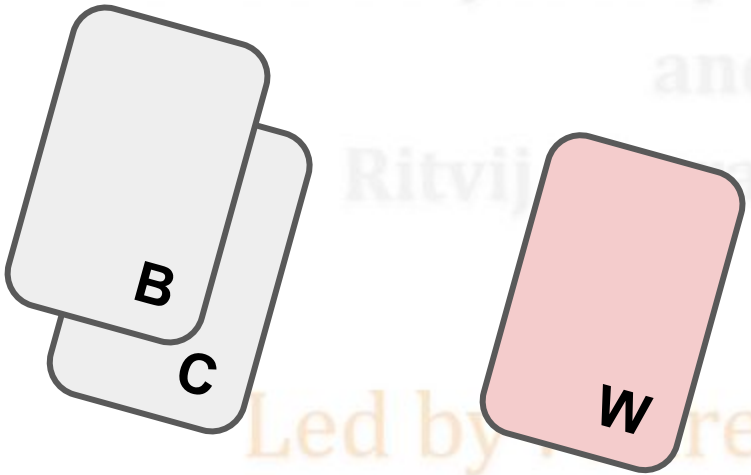
A Player “plays” a card from the top



[“B”, “C”]

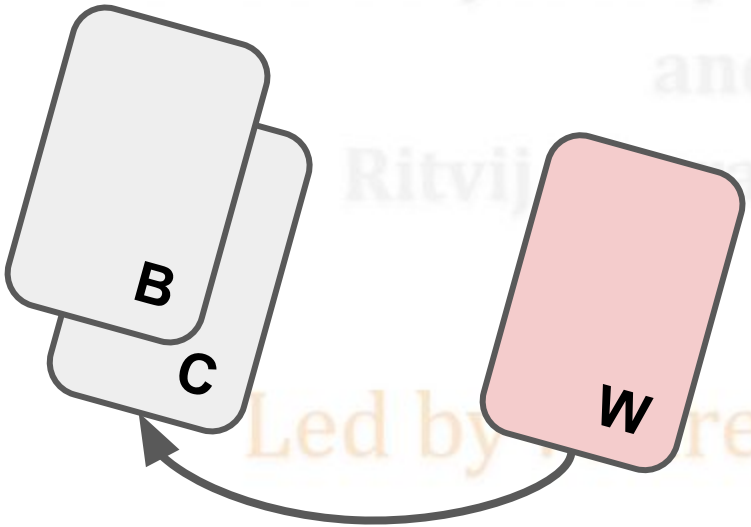
cards.pop(0)

Players will add cards to the “bottom”



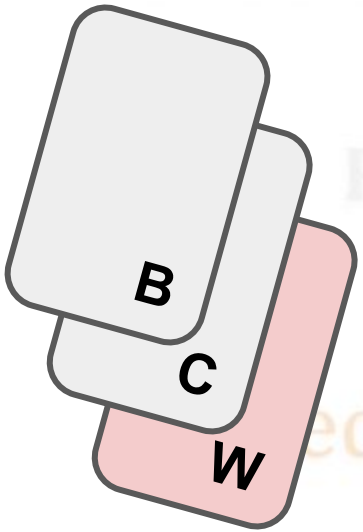
[“B”, “C”]

Players will add cards to the “bottom”



[“B”, “C”]

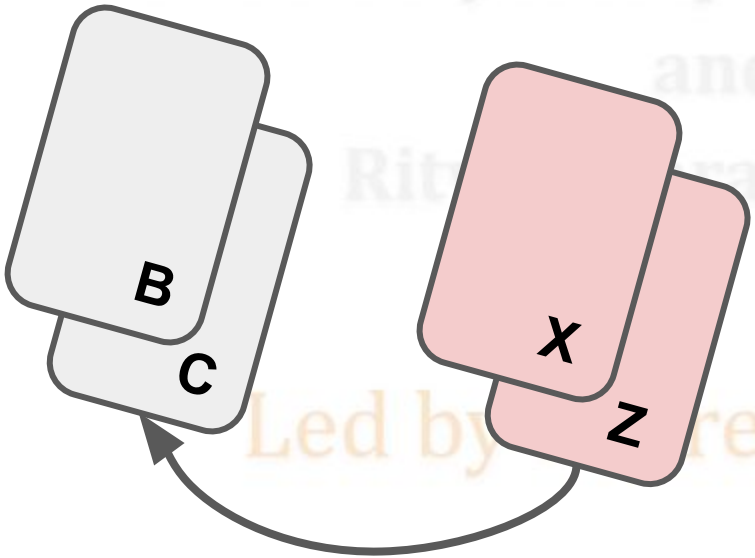
Players will add cards to the “bottom”



[“B”, “C”, “W”]

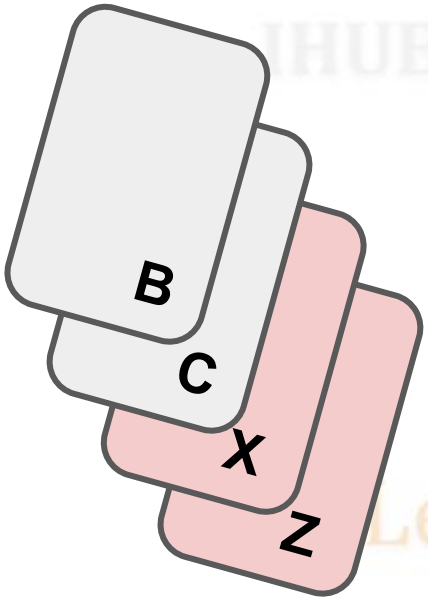
cards.append(“W”)

Player adding multiple cards uses extend()



cards = [“B”, “C”]
new = [“X”, “Z”]

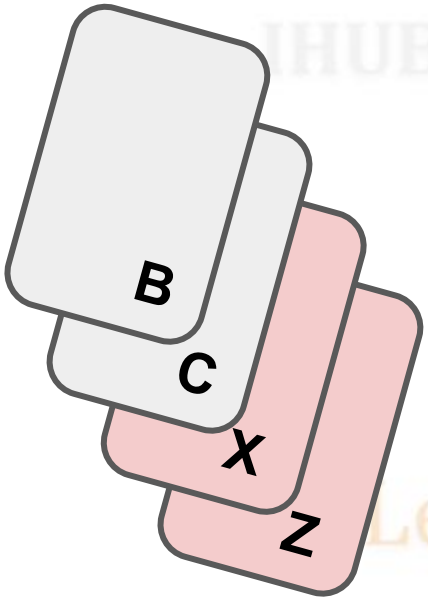
Player adding multiple cards uses extend()



cards = [“B”, “C”]
new = [“X”, “Z”]

cards.extend(new)

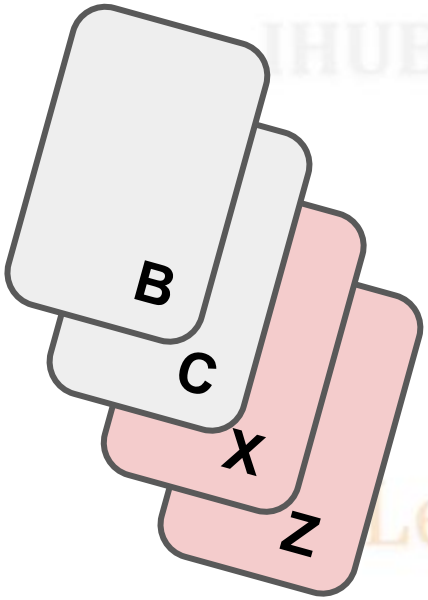
Player adding multiple cards uses extend()



cards = [“B”, “C” ,“X”, “Z”]

cards.extend(new)

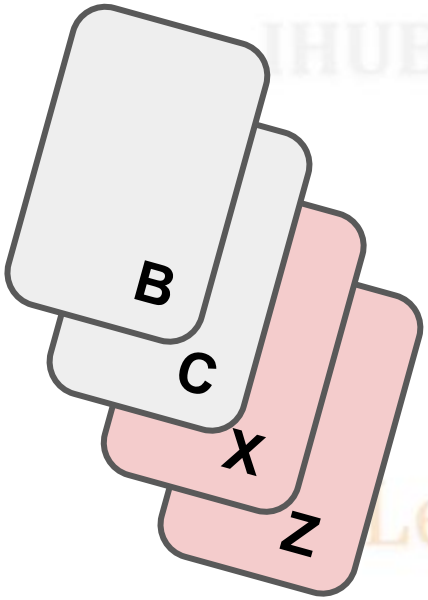
Don’t use append() or lists become nested!



cards = [“B”, “C” ,[“X”, “Z”]]

cards.**append**(new)

Don’t use append() or lists become nested!



cards = [“B”, “C” ,[“X”, “Z”]]

cards.append(new)

Let’s get started!

Game Logic

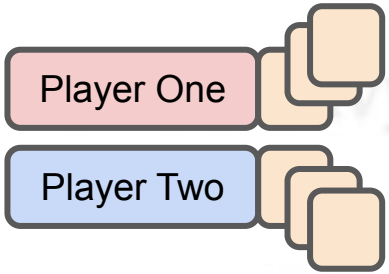
Creating the overall logic is often the hardest part of a project like this!

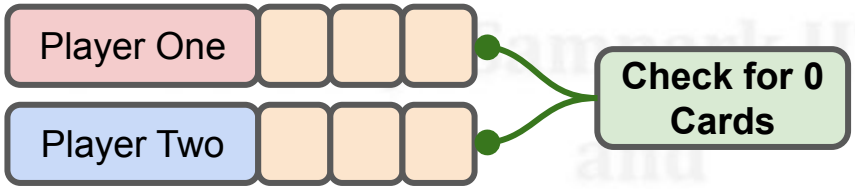
It is important to note, that we planned the classes around the upcoming logic, so in a real-world situation, you often think of both the logic and class structures simultaneously.

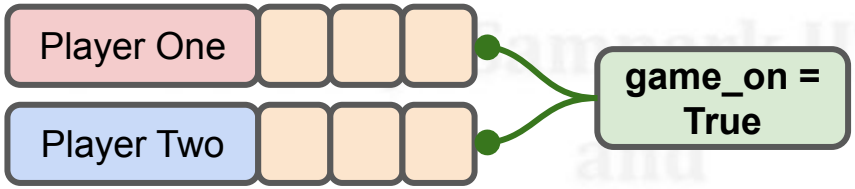
Let’s outline our logic for the game!

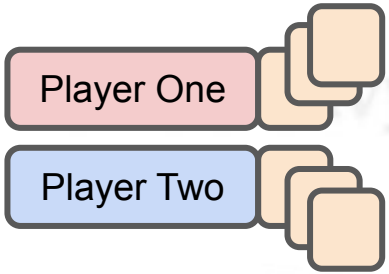
Player One

Player Two

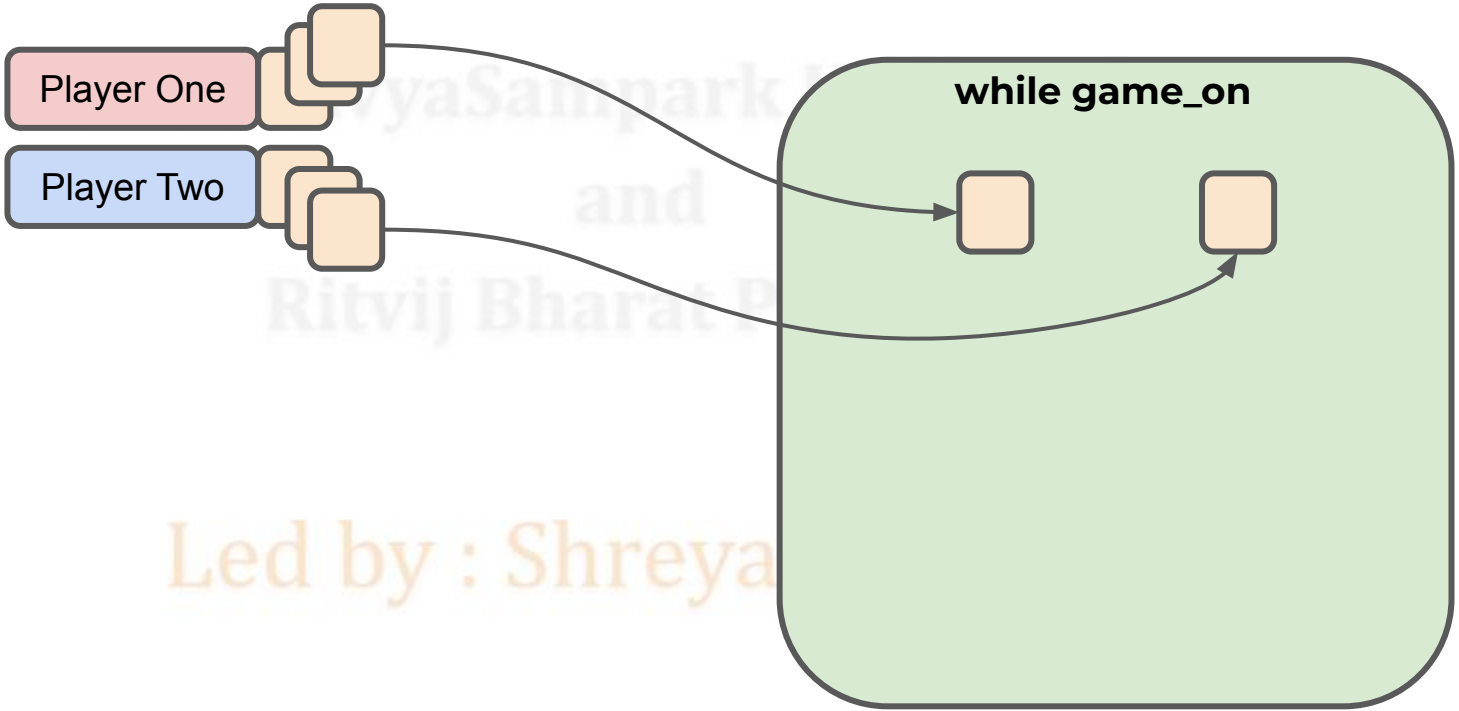


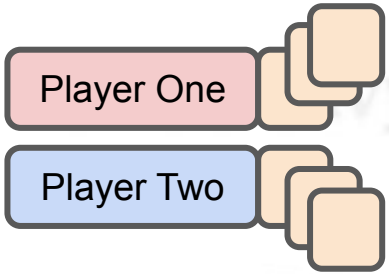


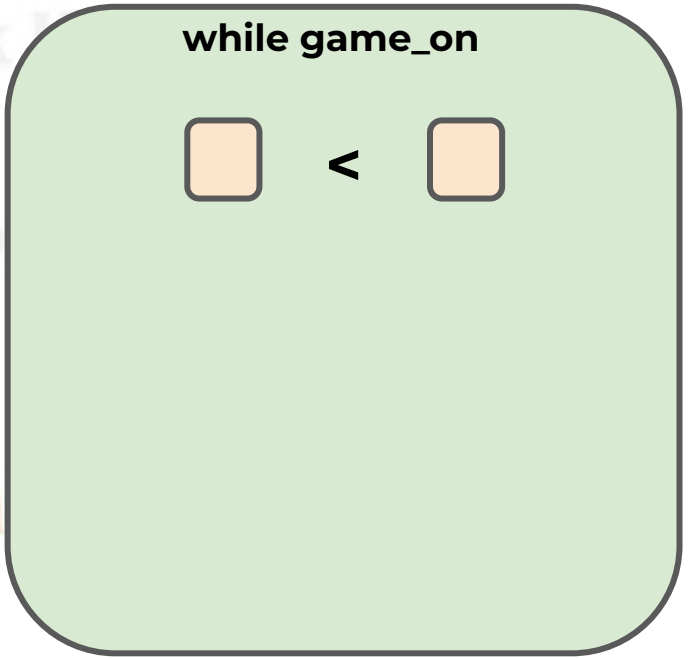


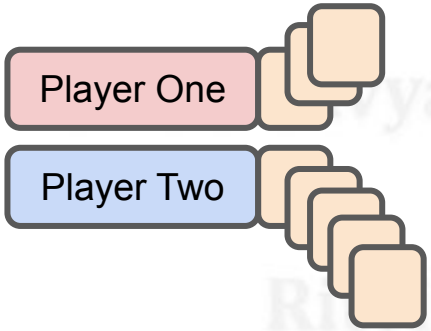


while game_on

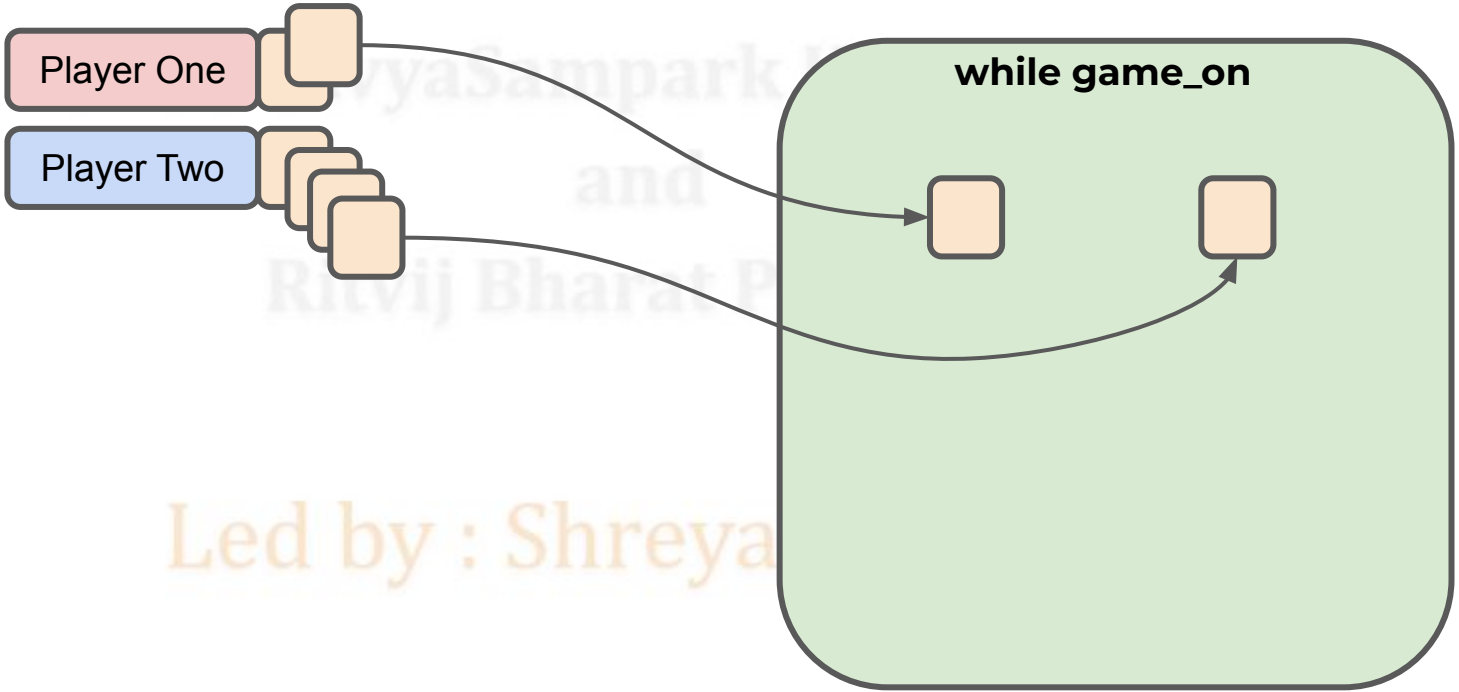


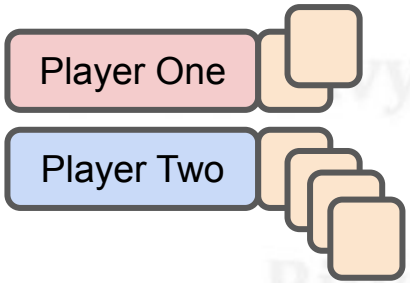


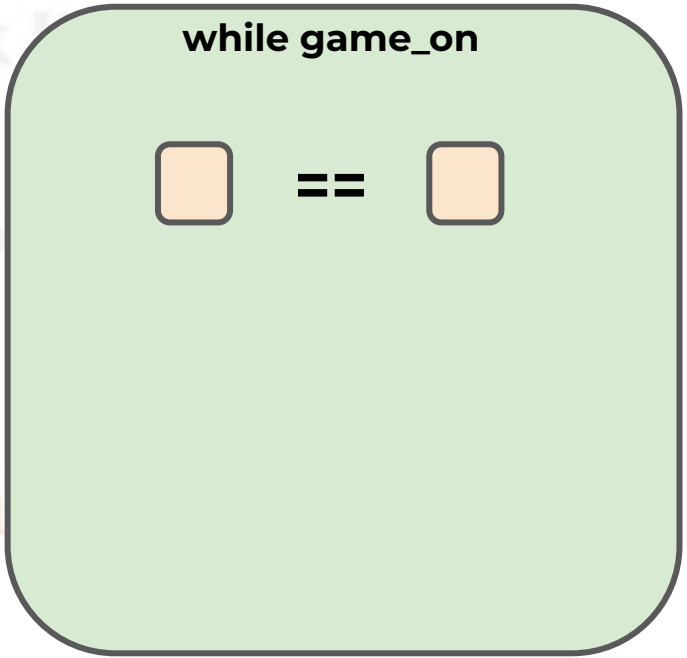


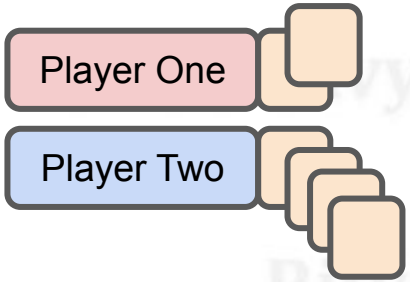


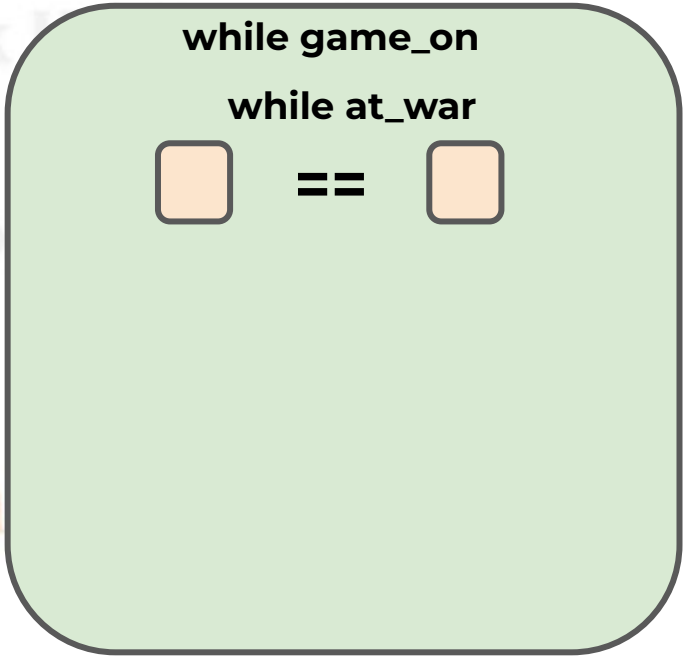
while game_on

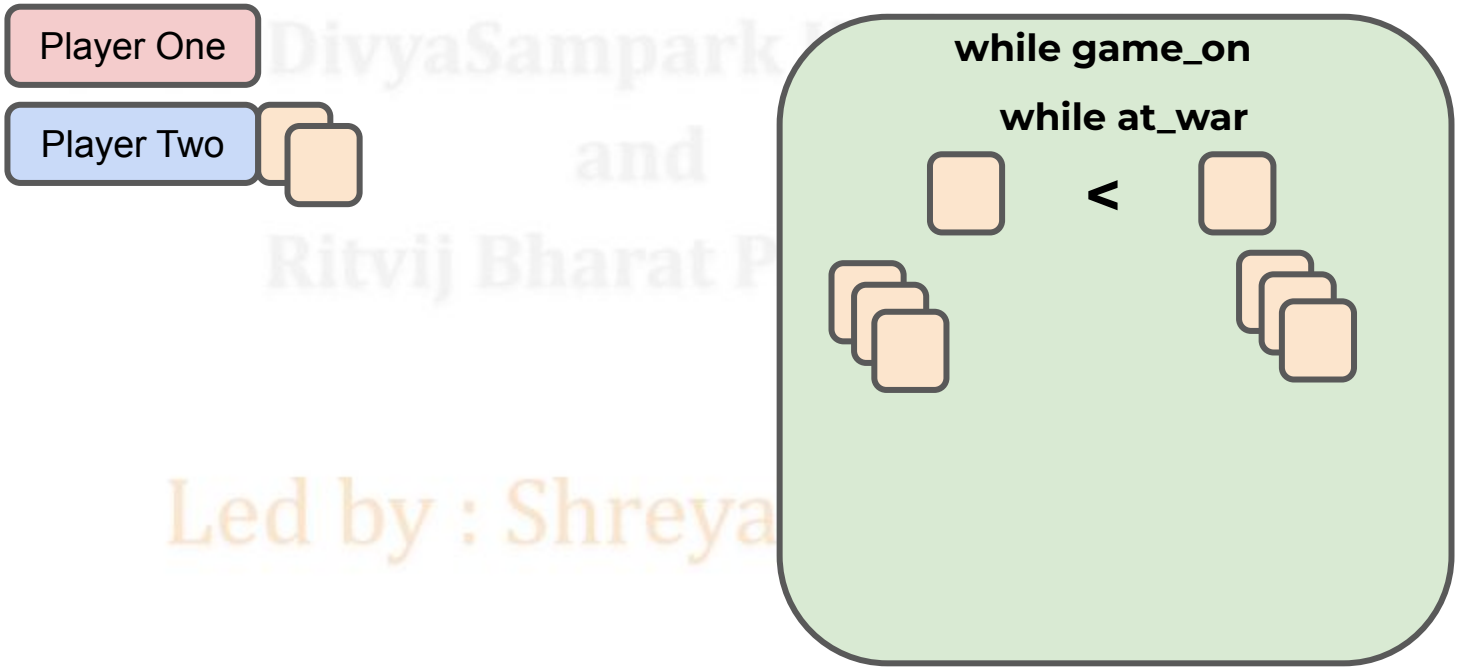


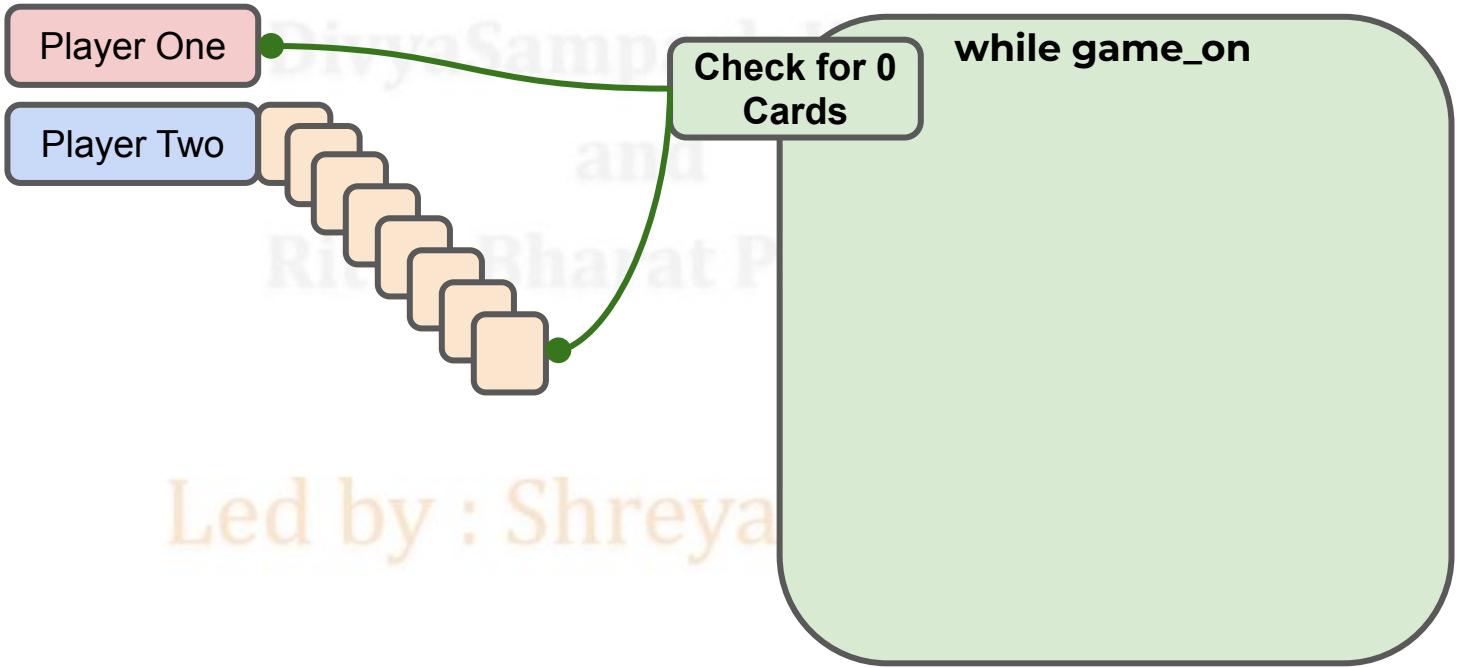


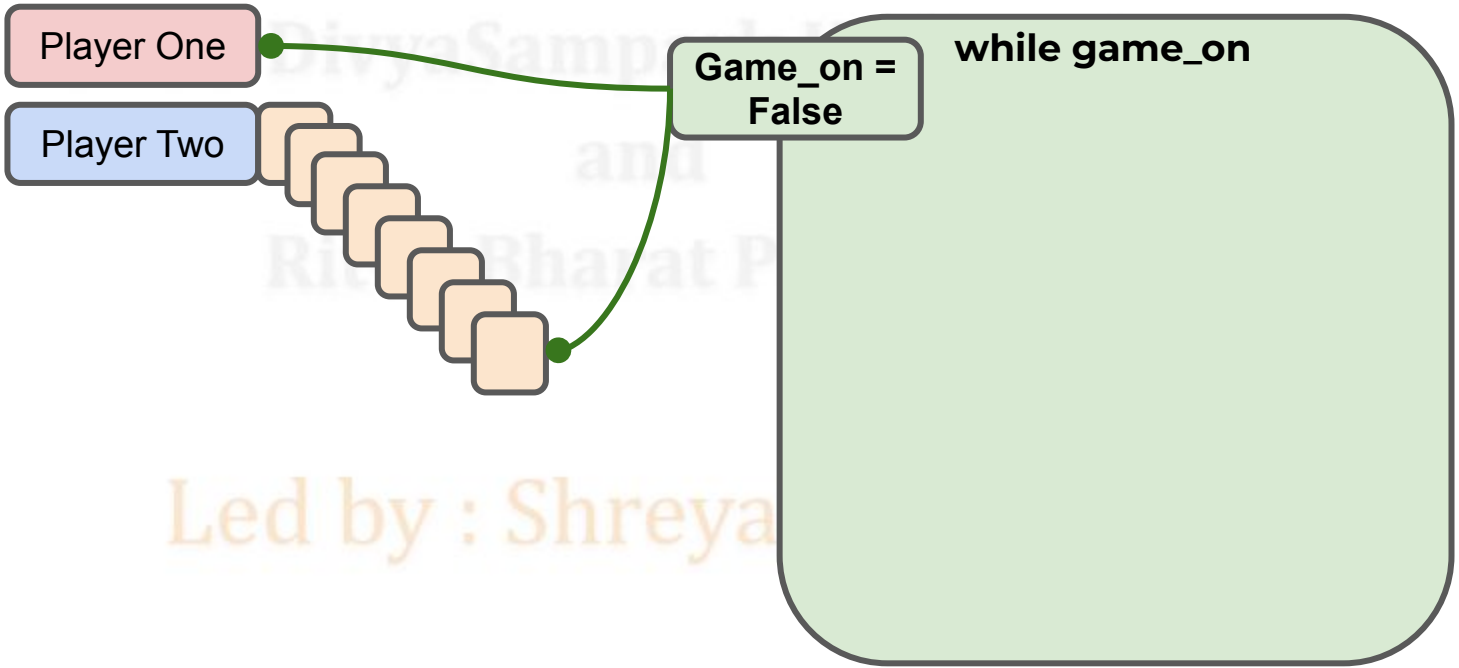


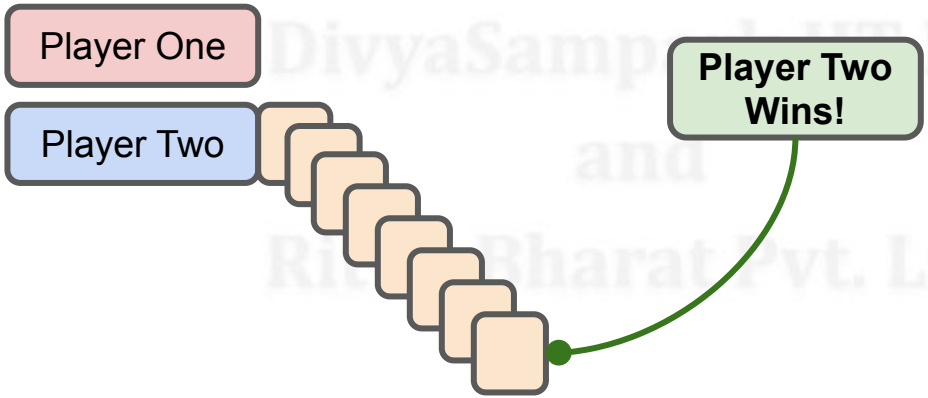












Game Logic

PART TWO

Game Logic

Now it’s time to check the player’s cards against each other.

We have 3 situations:

- Player One > Player Two
- Player One < Player Two
- Player One == Player Two

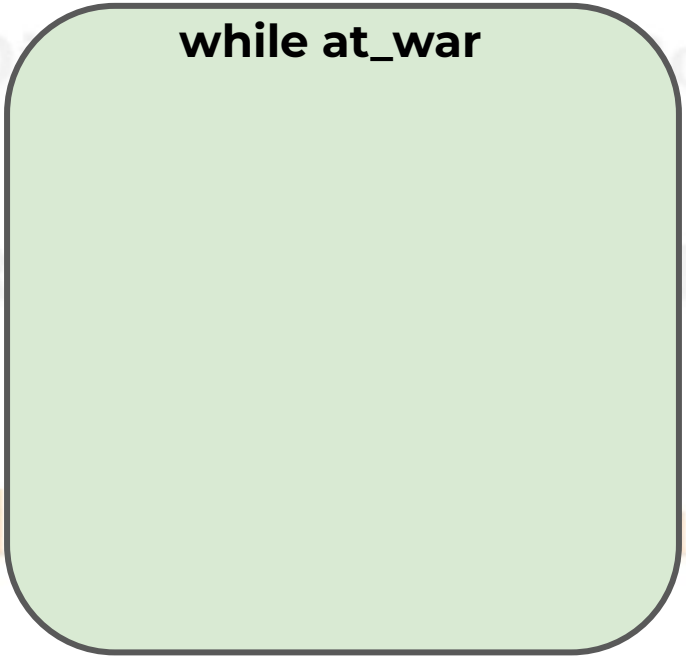
1. an if/elif/else within a while loop that assumes that a “war” has happened.

2. We will state at_war = False if the players resolve the match-up on the first drawn card, otherwise we will add cards to the current cards on the table.

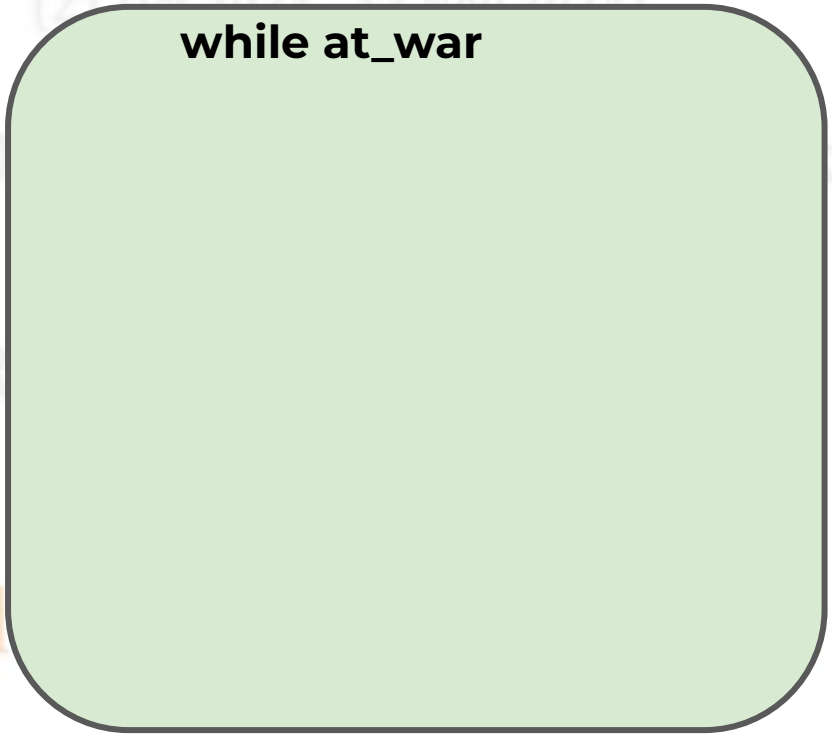
Few Rules:

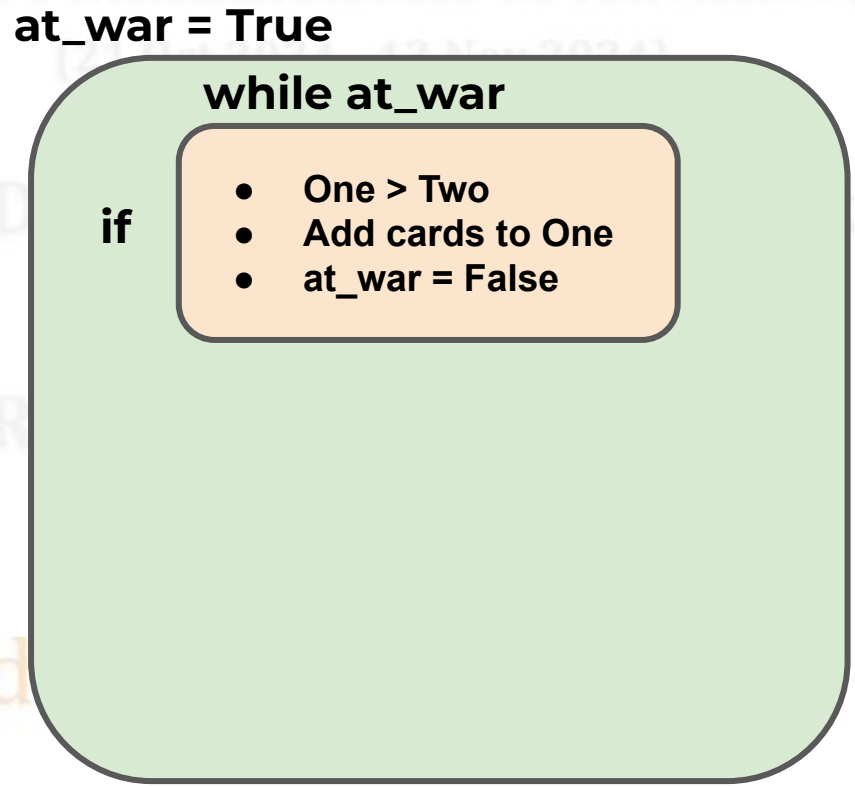
- If there is a tie, each player needs to draw 5 additional cards.
- A player loses if they don’t have at least 5 cards to play the war.
- This logic is easily edited to fit any rule structure you want.

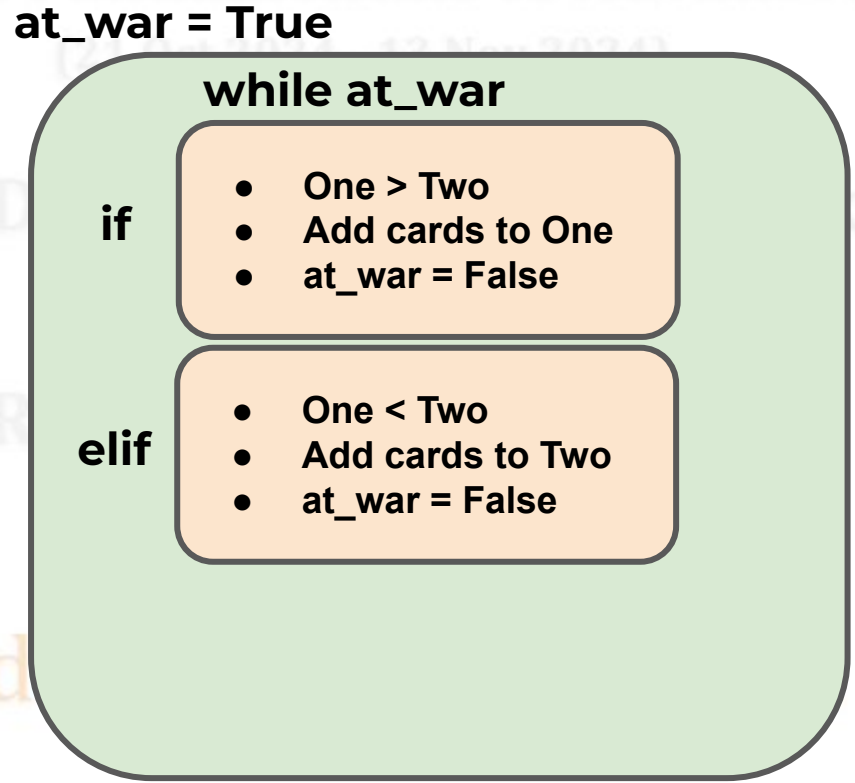
Let’s quickly explore this loop visually before we code it out!

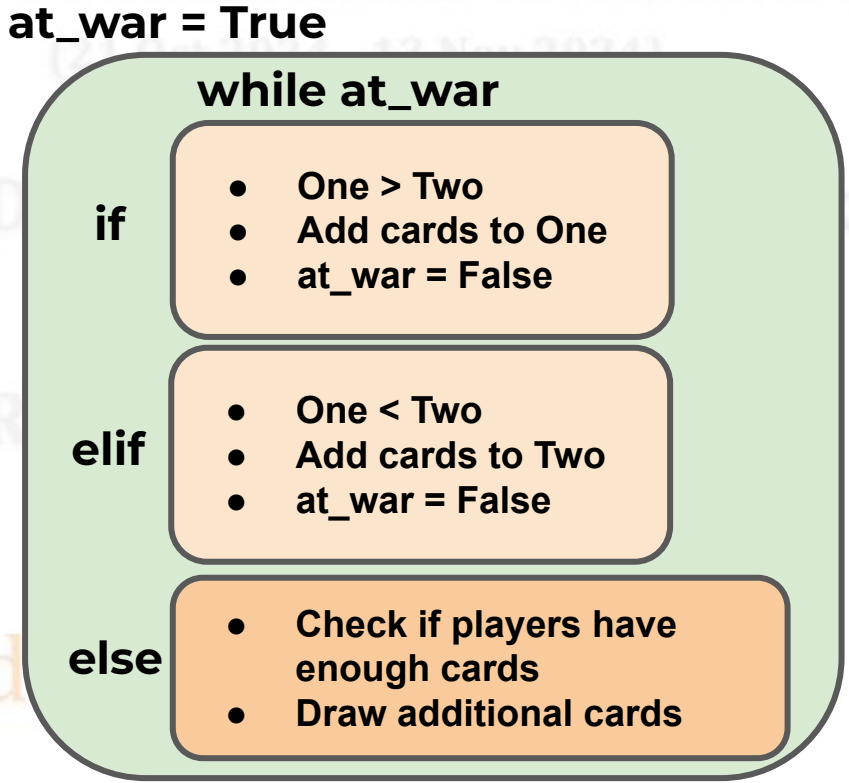


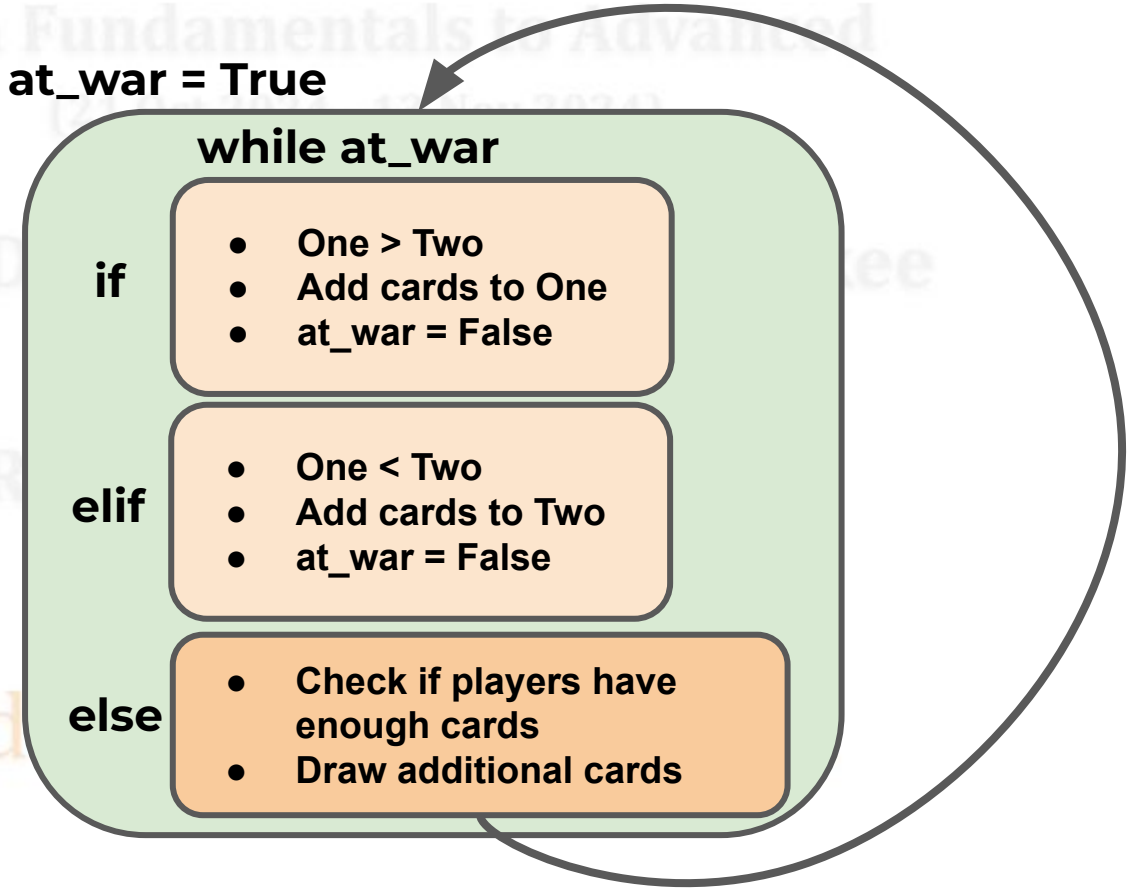
at_war = True




at_war = True
while at_war
if
One > Two
Add cards to One
at_war = False


at_war = True
while at_war
if
One > Two
Add cards to One
at_war = False
elif
One < Two
Add cards to Two
at_war = False


at_war = True
while at_war
if
One > Two
Add cards to One
at_war = False
elif
One < Two
Add cards to Two
at_war = False
else
Check if players have enough cards
Draw additional cards


at_war = True
while at_war
if
One > Two
Add cards to One
at_war = False
elif
One < Two
Add cards to Two
at_war = False
else
Check if players have enough cards
Draw additional cards