



Milestone Project 2

SECTION



Complete Python Bootcamp

- Now that you've learned about OOP, let's test your new skills with another Milestone Project.
- To warm-up, we will first guide you through creating a simple card game, then you will attempt the 2nd Milestone Project Exercise.



WARM UP PROJECT



Complete Python Bootcamp

- To warm up for the 2nd Milestone Project we will recreate the card game called “War”.
- Let’s have a quick overview of the game.



Complete Python Bootcamp

- To warm up for the 2nd Milestone Project we will recreate the card game called “War”.
- Let’s have a quick overview of the game.
 - [wikipedia.org/wiki/War_\(card_game\)](https://wikipedia.org/wiki/War_(card_game))

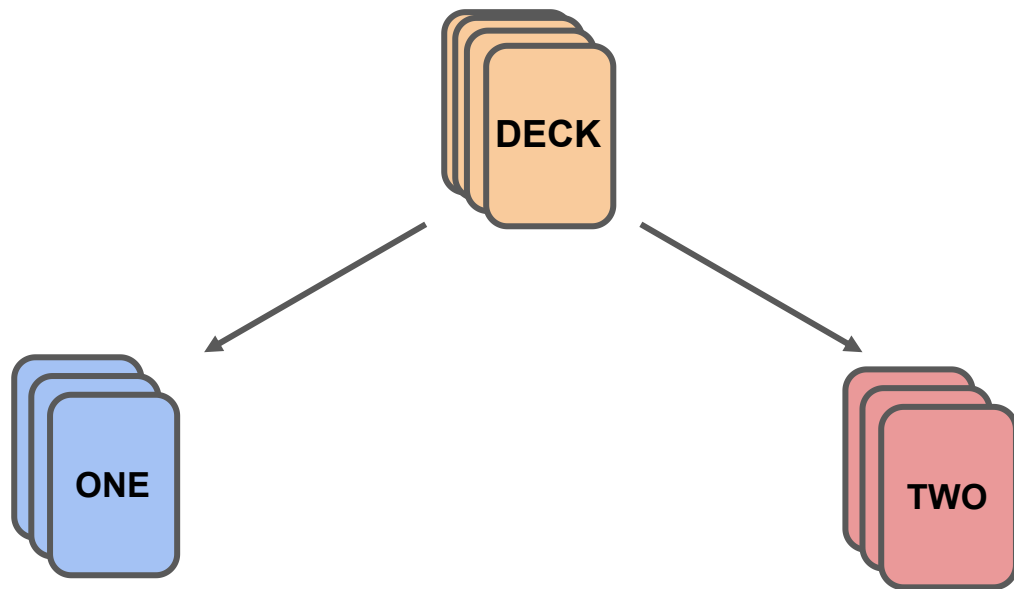


Complete Python Bootcamp



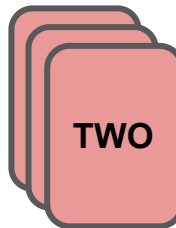
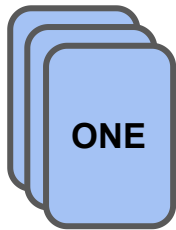


Complete Python Bootcamp



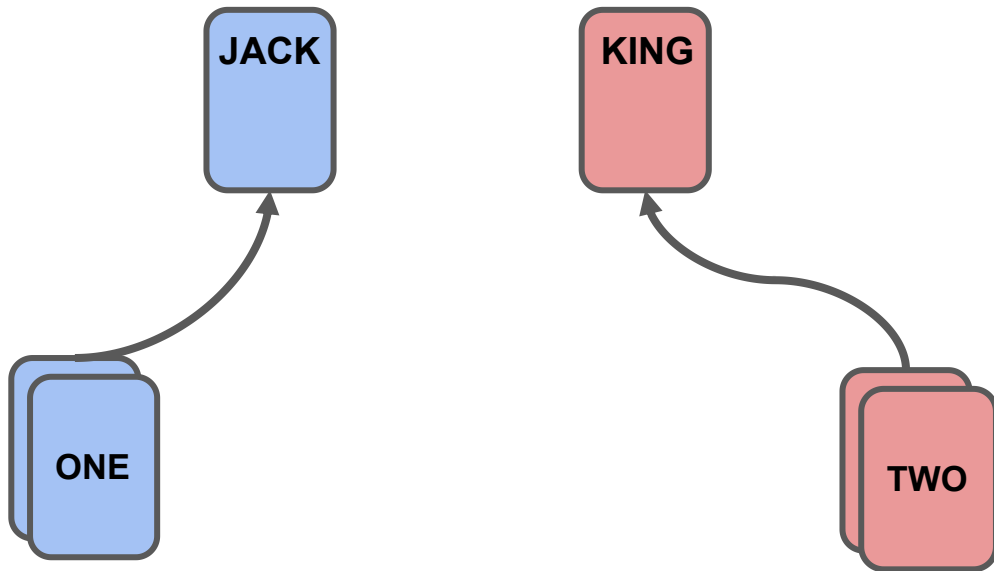


Complete Python Bootcamp



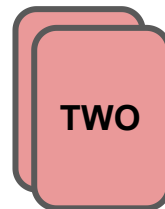
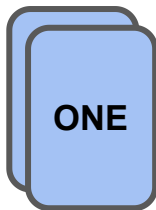


Complete Python Bootcamp





Complete Python Bootcamp

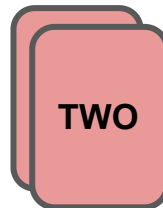
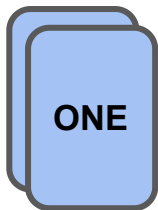
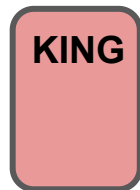




Complete Python Bootcamp

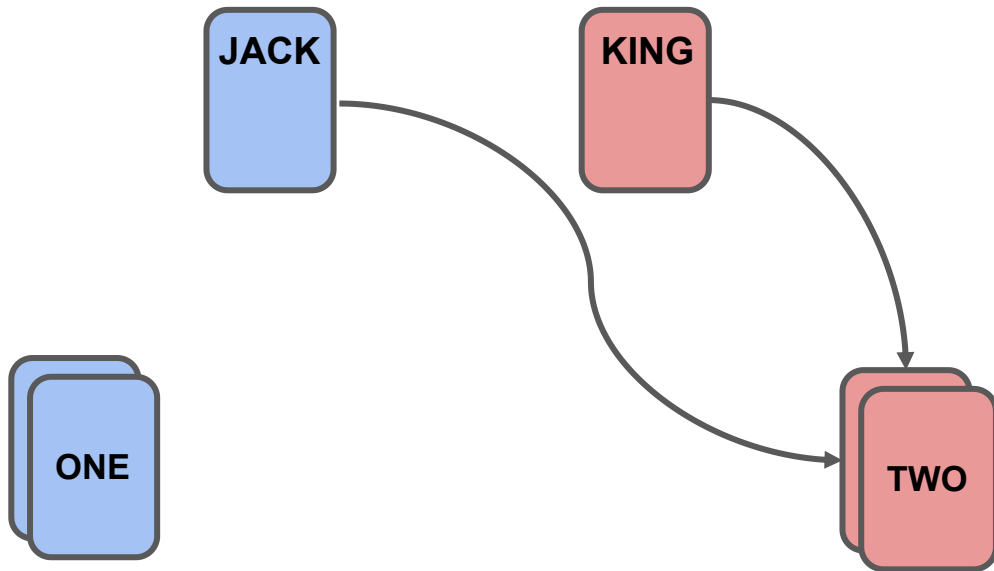


<



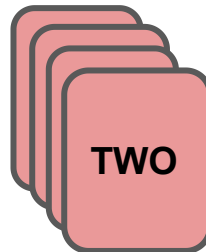


Complete Python Bootcamp



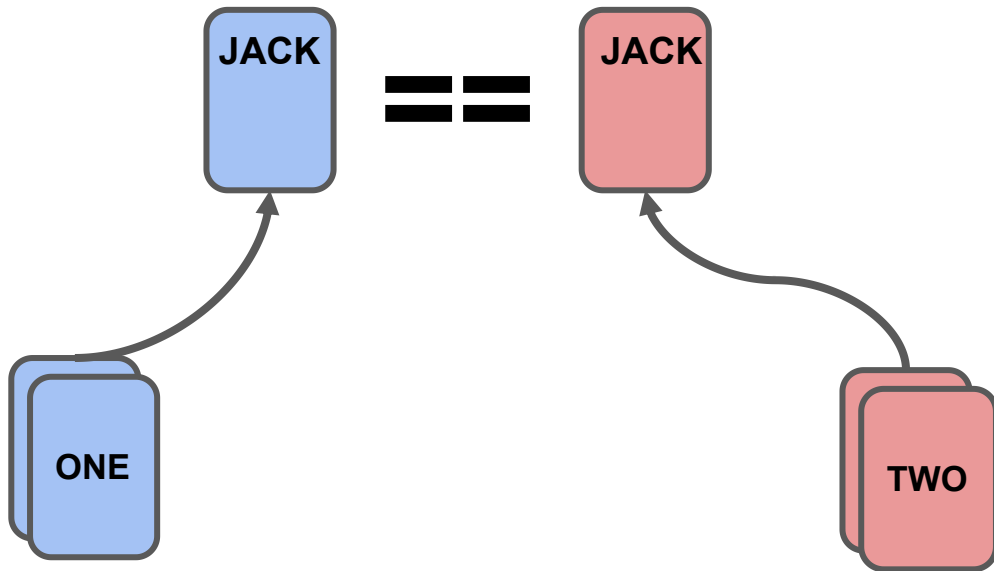


Complete Python Bootcamp



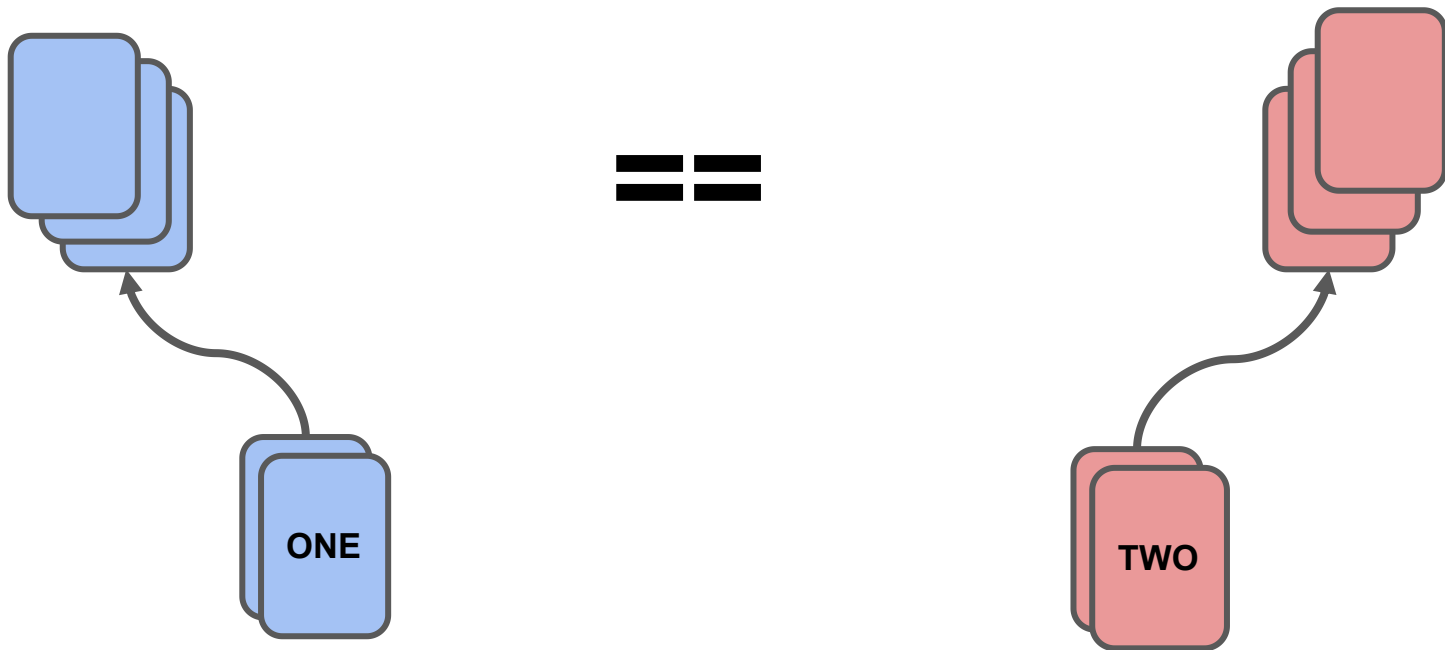


Complete Python Bootcamp



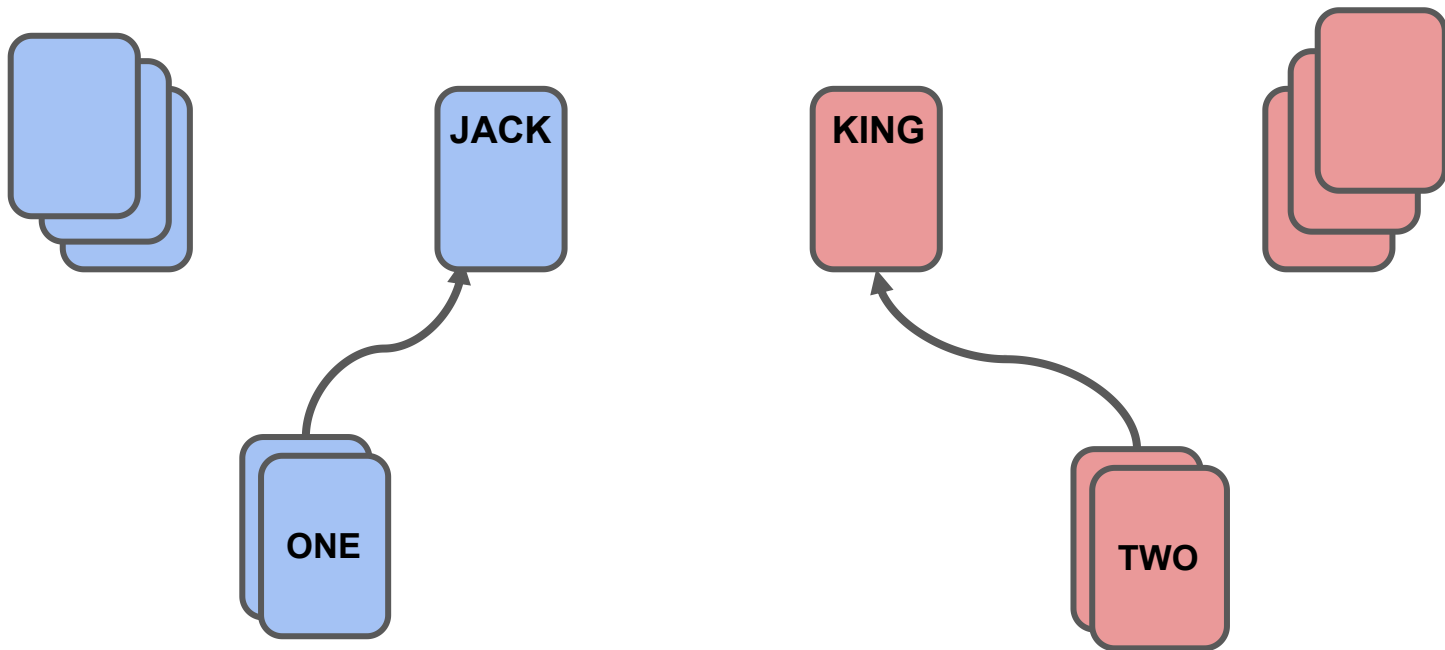


Complete Python Bootcamp



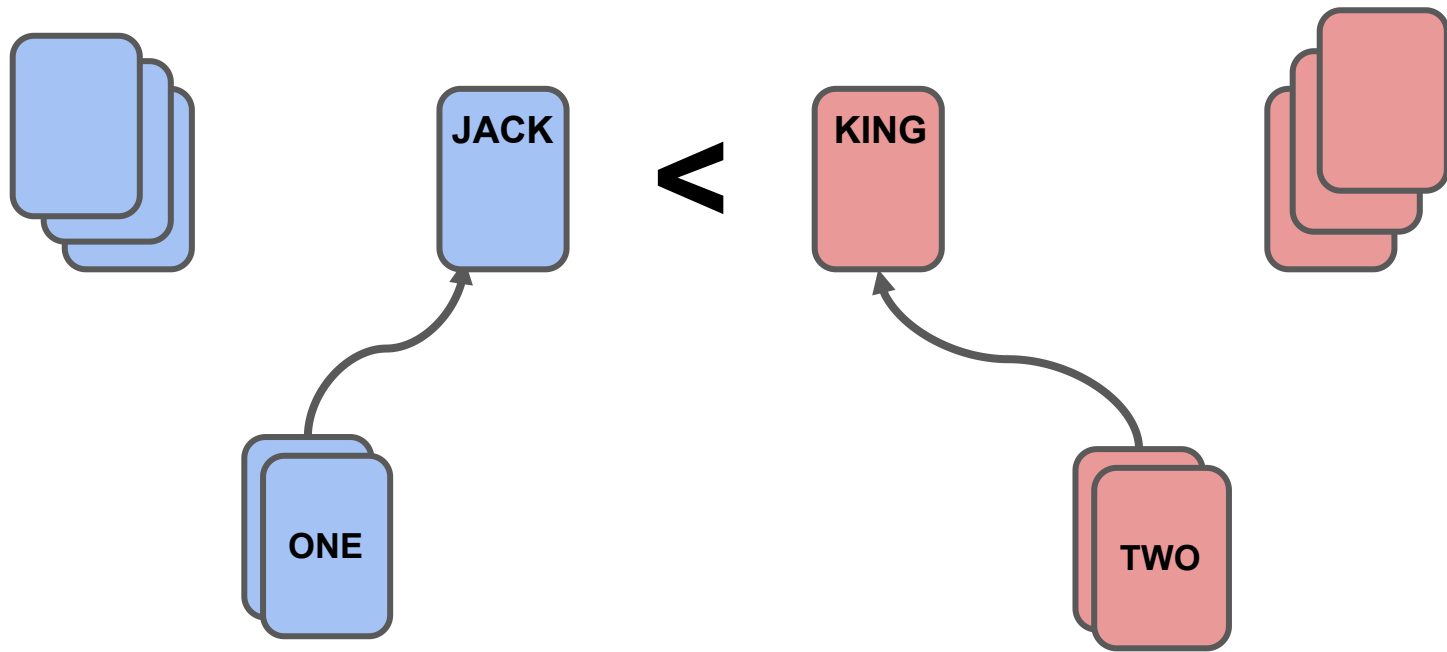


Complete Python Bootcamp



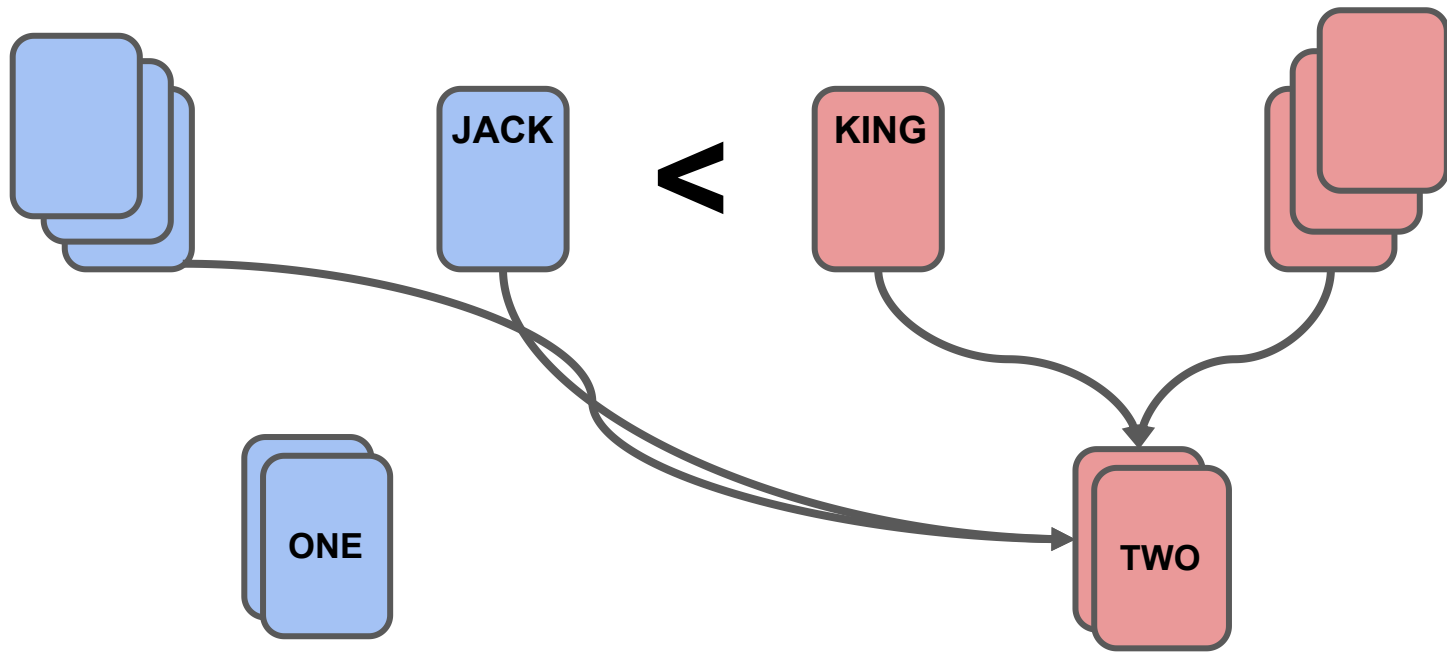


Complete Python Bootcamp



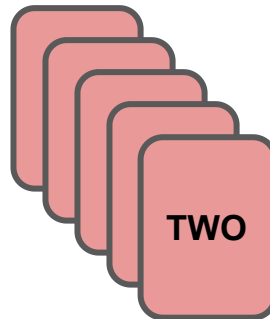


Complete Python Bootcamp



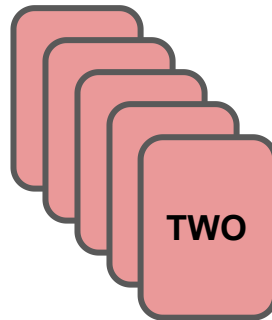
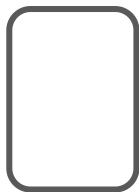


Complete Python Bootcamp





Complete Python Bootcamp





Complete Python Bootcamp

- The “war” process can be repeated in this case of back to back ties.
- To construct this game, we will create:
 - Card Class
 - Deck Class
 - Player Class
 - Game Logic



Let's get started!



Card Class



Deck Class



Complete Python Bootcamp

- 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



Complete Python Bootcamp

- Deck Class
 - We will see that the 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



Complete Python Bootcamp

- 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).



Complete Python Bootcamp

- Player Class
 - 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.



Complete Python Bootcamp

- Player Class
 - 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.



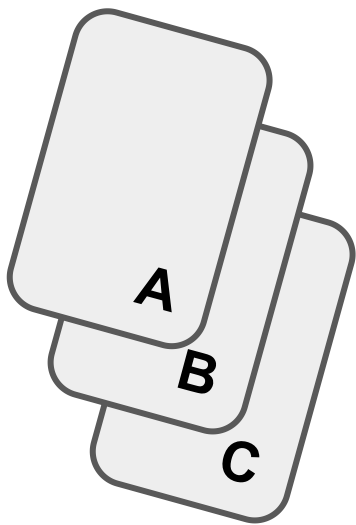
Complete Python Bootcamp

- Player Class will have a `self.all_cards` list



Complete Python Bootcamp

- Player Class will have a `self.all_cards` list

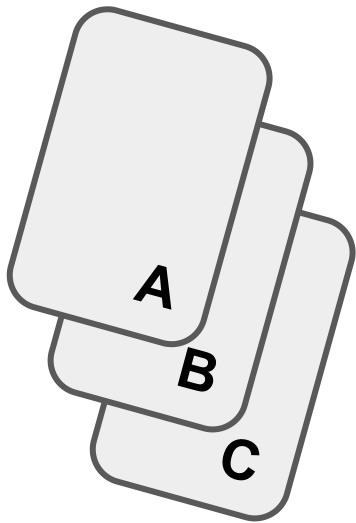


```
cards = ["A", "B", "C"]
```




Complete Python Bootcamp

- Player Class will have a `self.all_cards` list

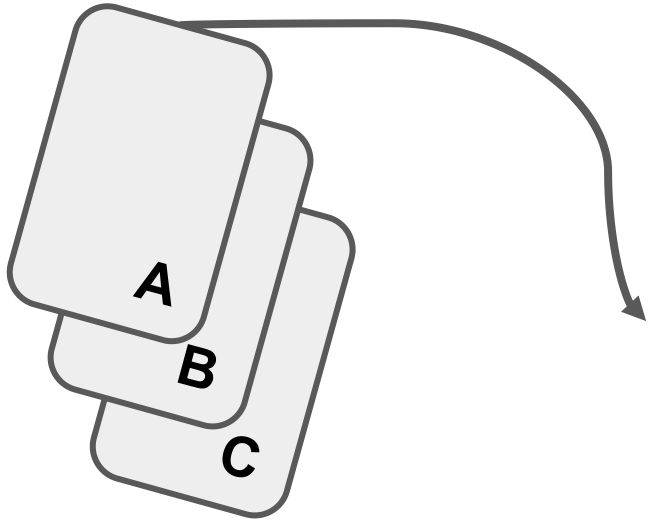


`["A", "B", "C"]`



Complete Python Bootcamp

- A Player “plays” a card from the top

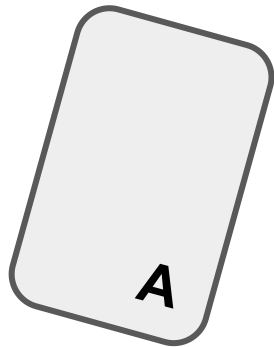
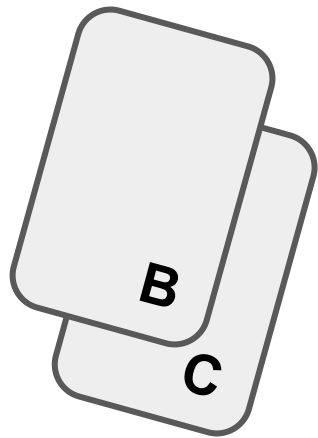


["A", "B", "C"]



Complete Python Bootcamp

- A Player “plays” a card from the top

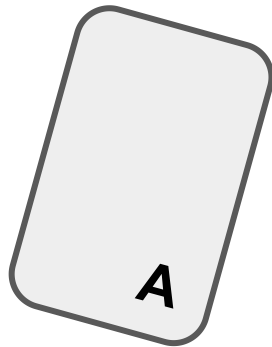
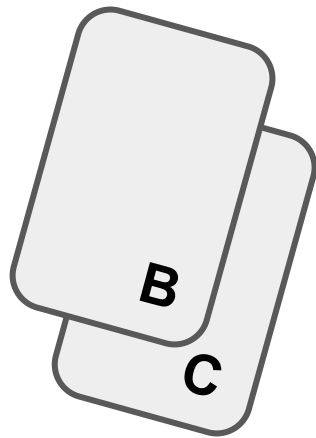


["A", "B", "C"]



Complete Python Bootcamp

- A Player “plays” a card from the top



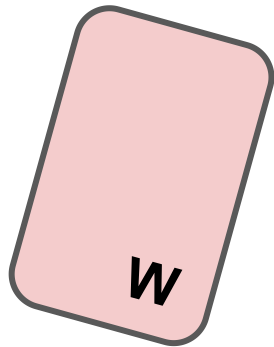
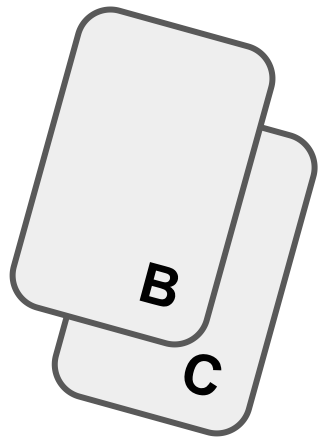
["B", "C"]

`cards.pop(0)`



Complete Python Bootcamp

- Players will add cards to the “bottom”

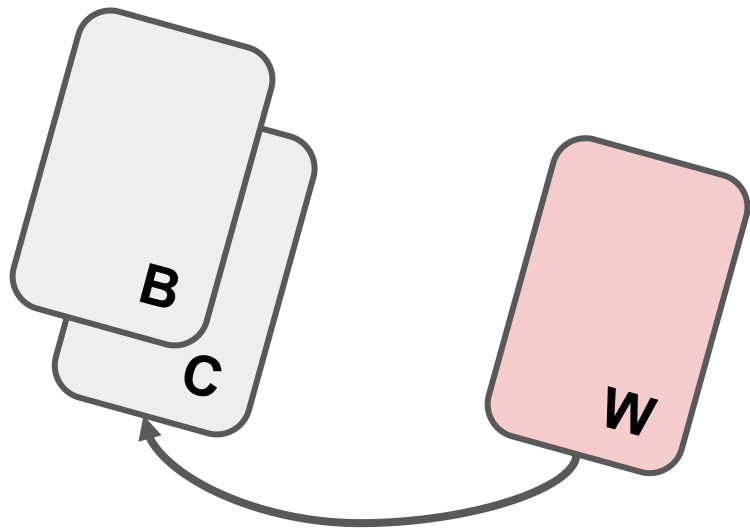


["B", "C"]



Complete Python Bootcamp

- Players will add cards to the “bottom”

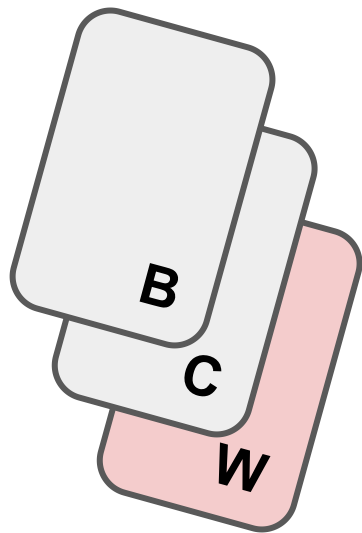


["B", "C"]



Complete Python Bootcamp

- Players will add cards to the “bottom”



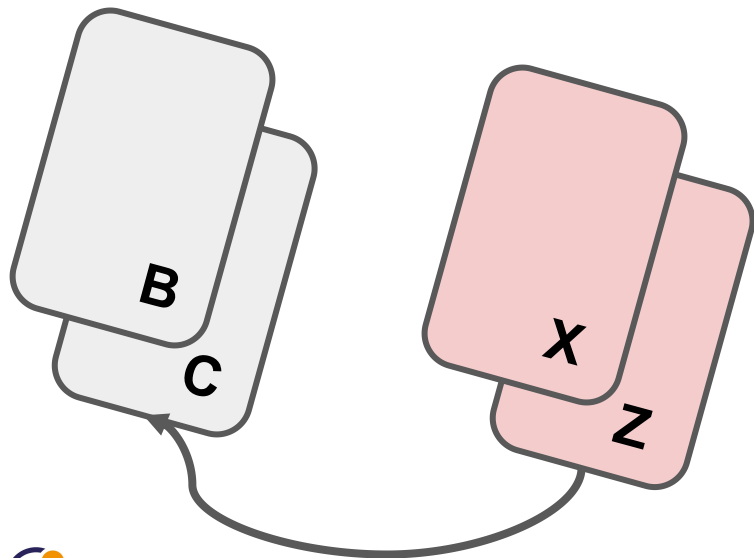
`["B", "C", "W"]`

`cards.append("W")`



Complete Python Bootcamp

- Player adding multiple cards uses `extend()`

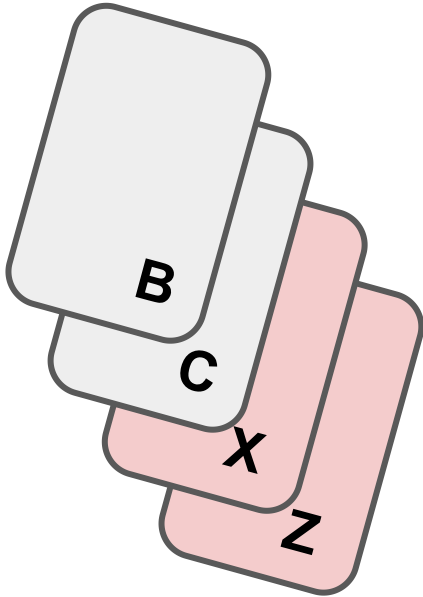


```
cards = [ "B", "C" ]  
new = [ "X", "Z" ]
```




Complete Python Bootcamp

- Player adding multiple cards uses `extend()`



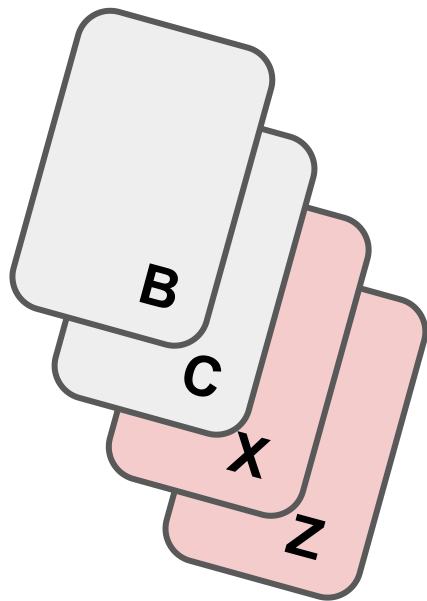
```
cards = [ "B", "C" ]  
new = [ "X", "Z" ]
```

```
cards.extend(new)
```



Complete Python Bootcamp

- Player adding multiple cards uses `extend()`



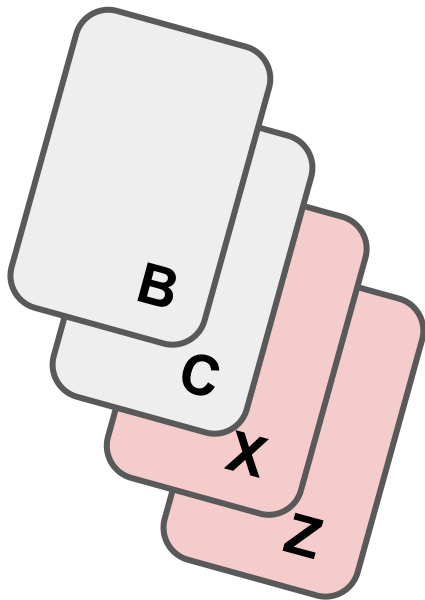
```
cards = [ "B", "C", "X", "Z" ]
```

```
cards.extend(new)
```



Complete Python Bootcamp

- Don't use `append()` or lists become nested!



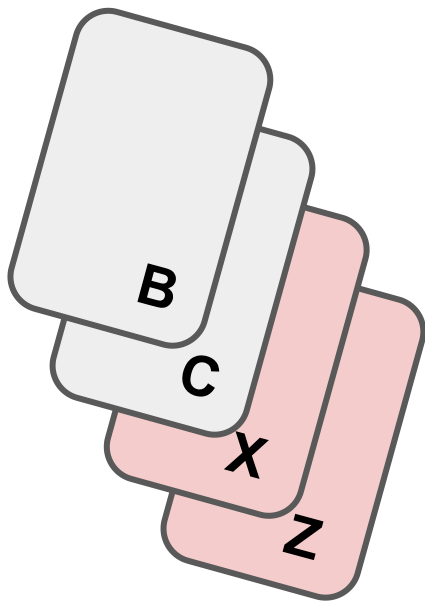
```
cards = [ "B", "C" ,["X", "Z"] ]
```

```
cards.append(new)
```



Complete Python Bootcamp

- Don't use `append()` or lists become nested!



```
cards = ["B", "C", ["X", "Z"]]
```

```
cards.append(new)
```



Let's get started!



Game Logic

PART ONE



Complete Python Bootcamp

- 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.



Complete Python Bootcamp

- Let's outline our logic for the game!



Complete Python Bootcamp

- Let's outline our logic for the game!

Player One

Player Two



Complete Python Bootcamp

- Let's outline our logic for the game!

Player One

Player Two

New Deck



Complete Python Bootcamp

- Let's outline our logic for the game!

Player One

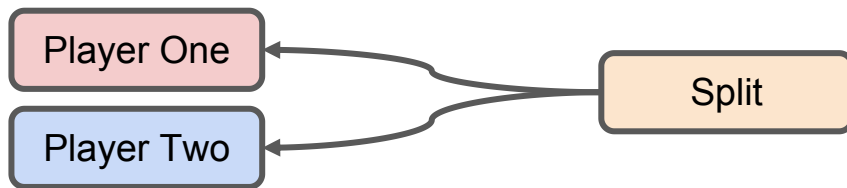
Player Two

Shuffle



Complete Python Bootcamp

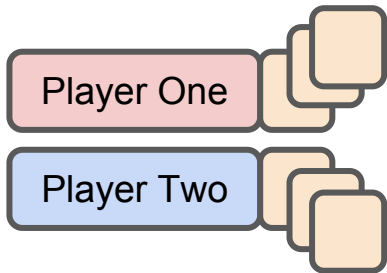
- Let's outline our logic for the game!





Complete Python Bootcamp

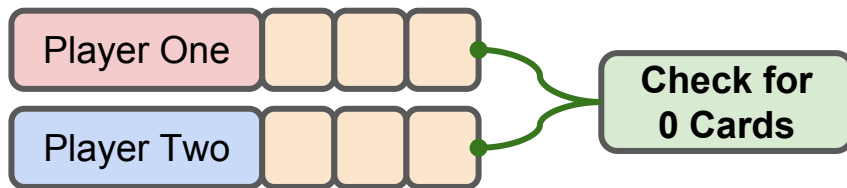
- Let's outline our logic for the game!





Complete Python Bootcamp

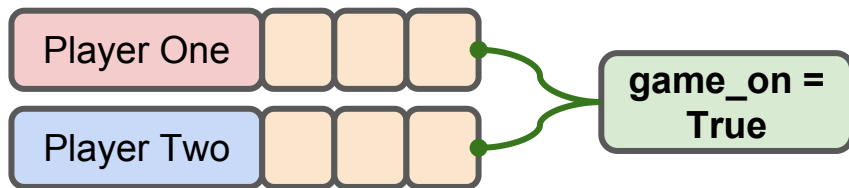
- Let's outline our logic for the game!





Complete Python Bootcamp

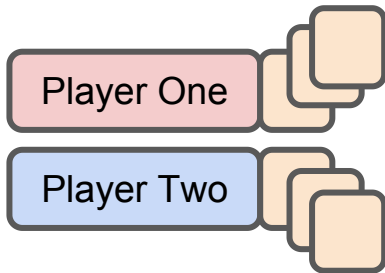
- Let's outline our logic for the game!





Complete Python Bootcamp

- Let's outline our logic for the game!

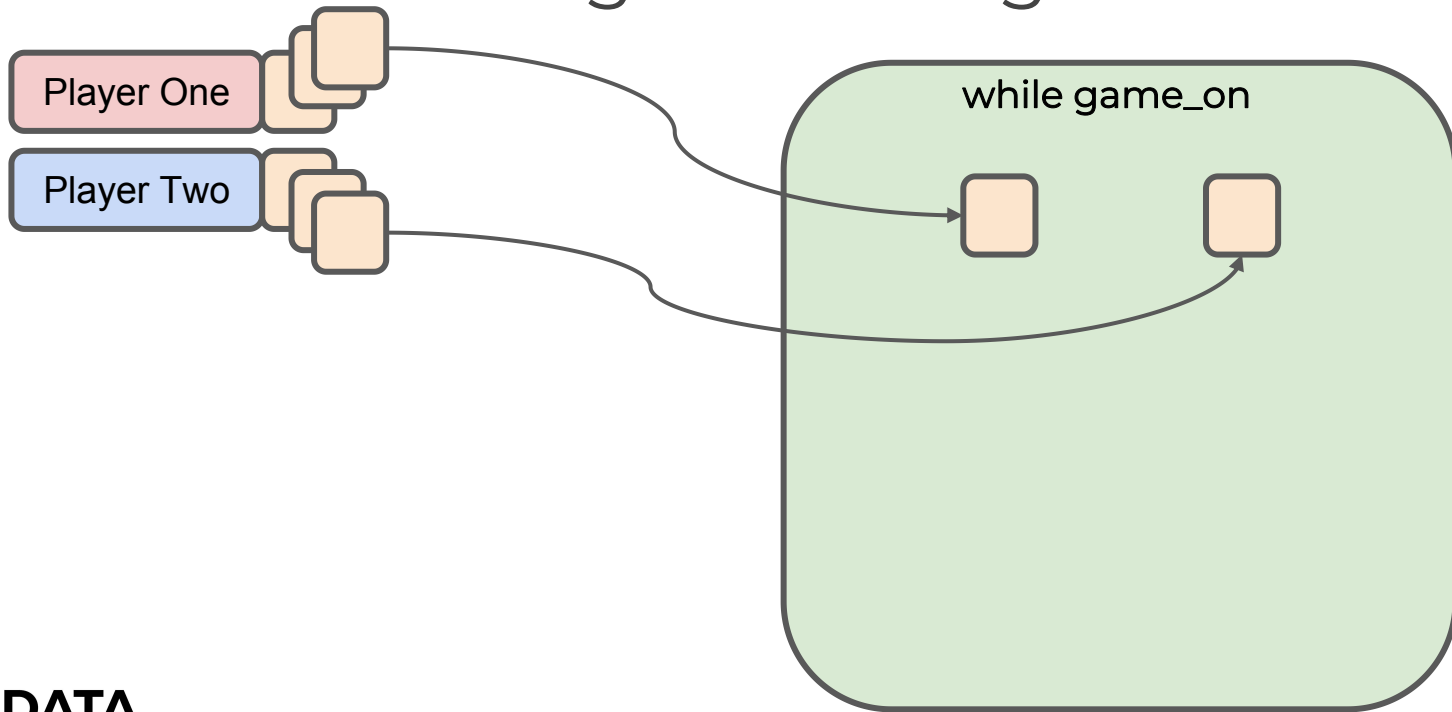


`while game_on`



Complete Python Bootcamp

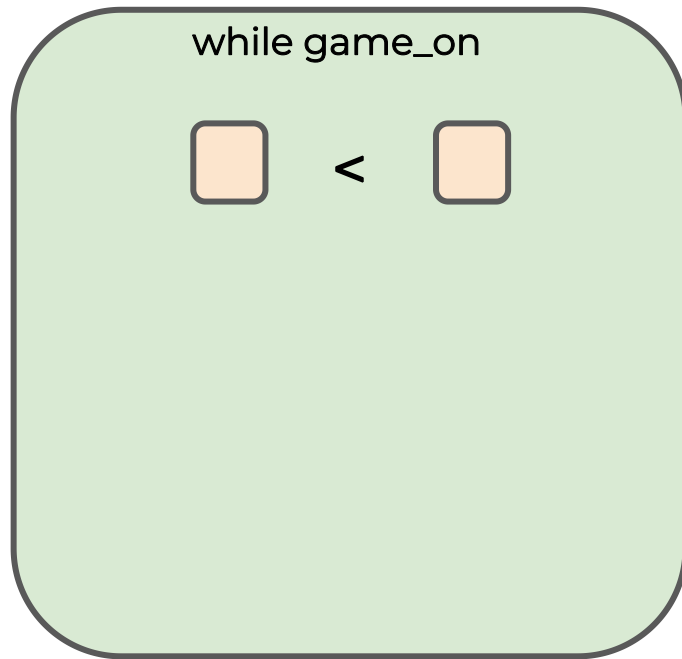
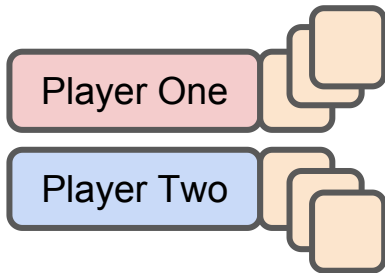
- Let's outline our logic for the game!





Complete Python Bootcamp

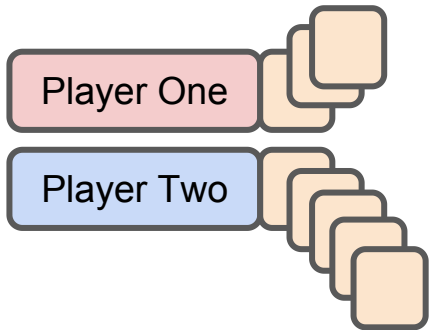
- Let's outline our logic for the game!





Complete Python Bootcamp

- Let's outline our logic for the game!

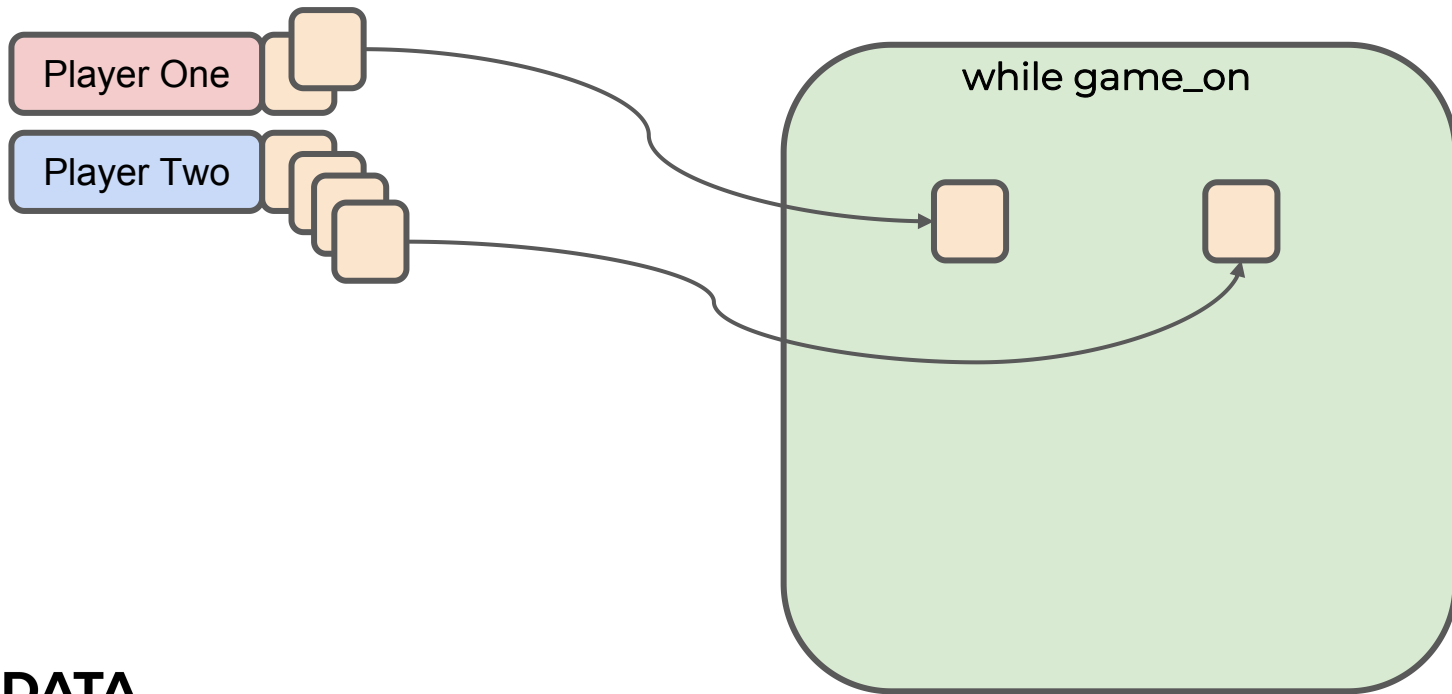


`while game_on`



Complete Python Bootcamp

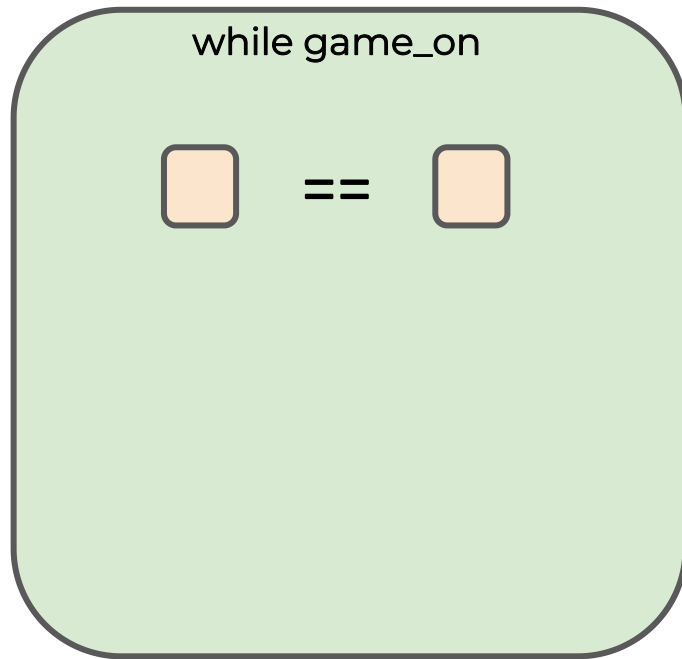
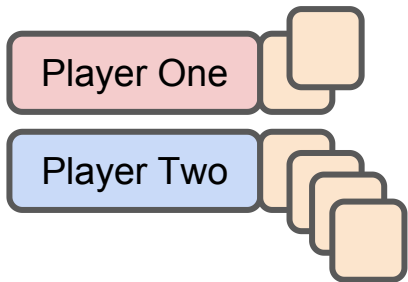
- Let's outline our logic for the game!





Complete Python Bootcamp

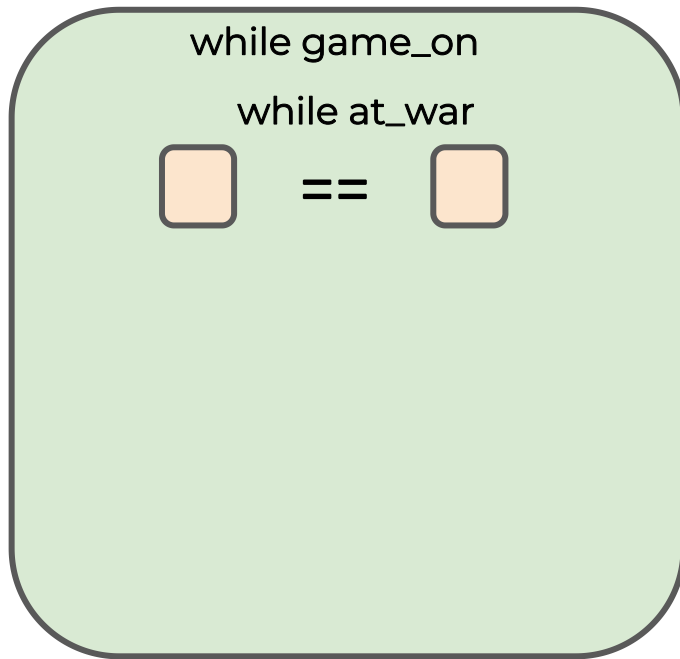
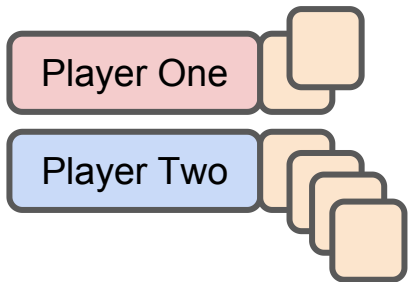
- Let's outline our logic for the game!





Complete Python Bootcamp

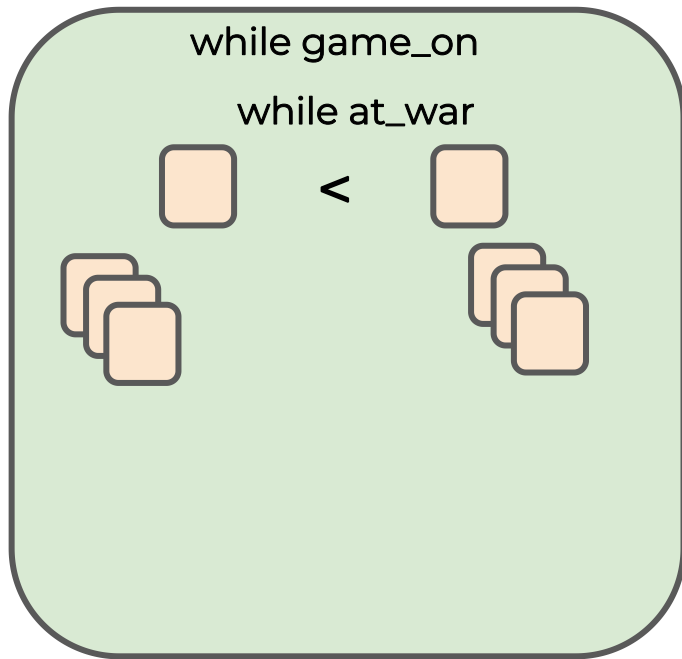
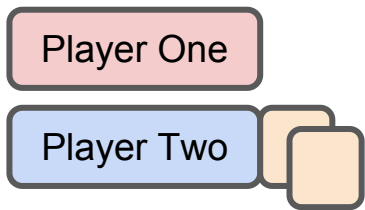
- Let's outline our logic for the game!





Complete Python Bootcamp

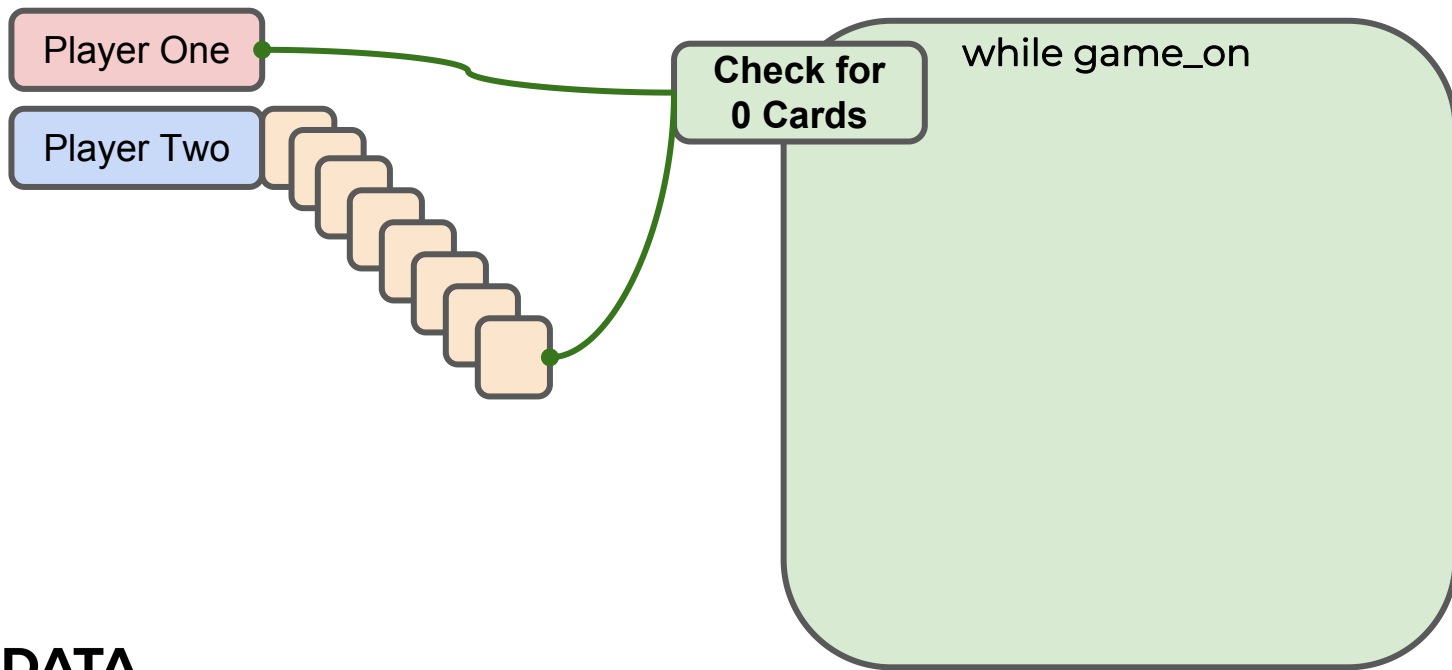
- Let's outline our logic for the game!





Complete Python Bootcamp

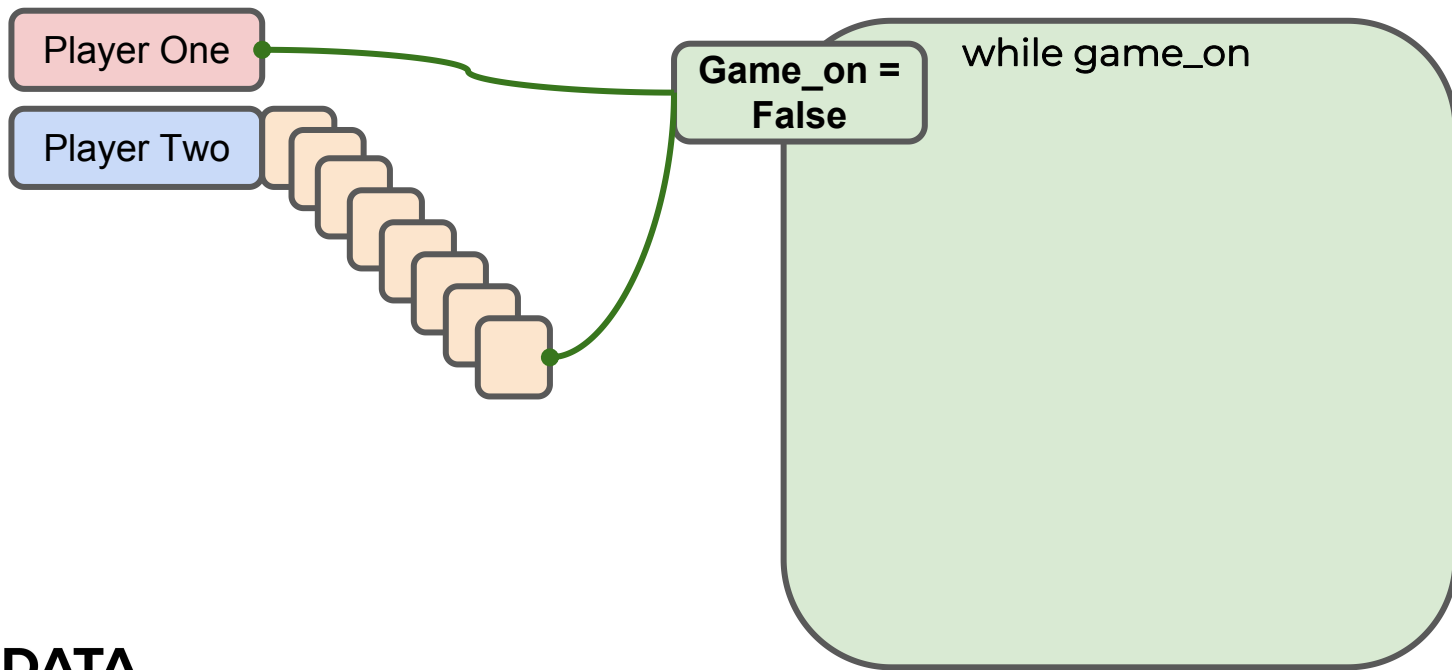
- Let's outline our logic for the game!





Complete Python Bootcamp

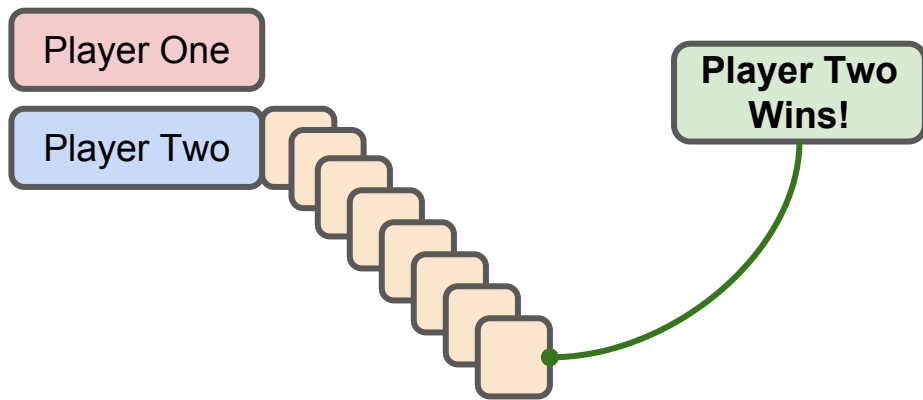
- Let's outline our logic for the game!





Complete Python Bootcamp

- Let's outline our logic for the game!





Game Logic

PART TWO



Game Logic

PART THREE



Complete Python Bootcamp

- Now it's time to check the player's cards against each other.
- There are lots of ways this can be done!
- We have 3 situations:
 - Player One > Player Two
 - Player One < Player Two
 - Player One == Player Two



Complete Python Bootcamp

- The way we will write this is with an if/elif/else within a while loop that assumes that a “war” has happened.
- 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.

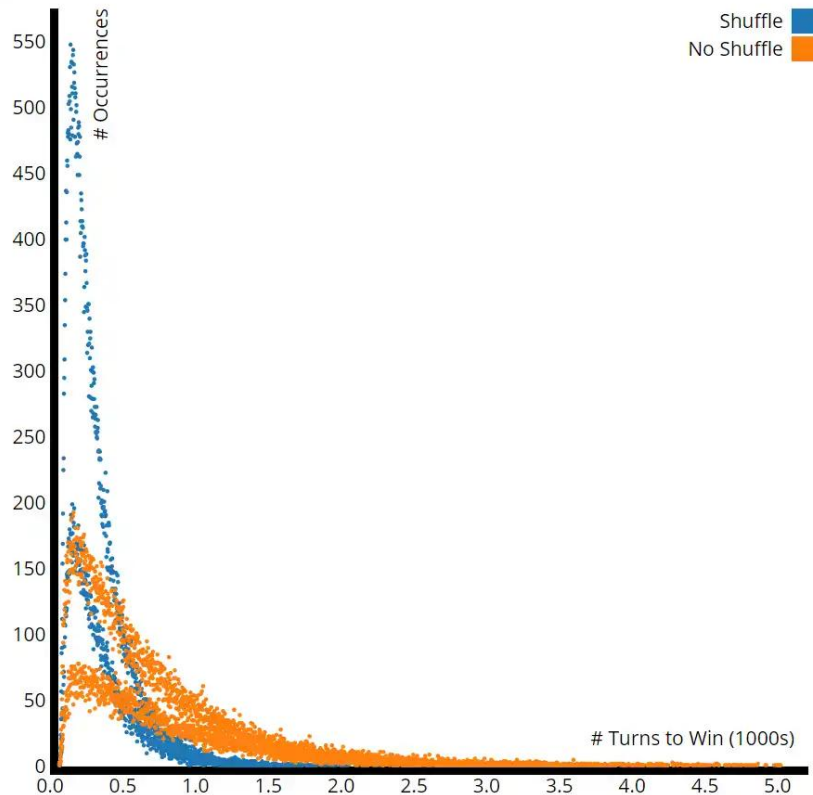


Complete Python Bootcamp

- The rules we'll use in this version is if there is a tie, each player needs to draw 5 additional cards.
- We'll also say that 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.



Complete Python Bootcamp





Complete Python Bootcamp

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



Complete Python Bootcamp

- Comparison Game Logic

```
while at_war
```



Complete Python Bootcamp

```
at_war = True
```

```
while at_war
```



Complete Python Bootcamp

at_war = True

while at_war

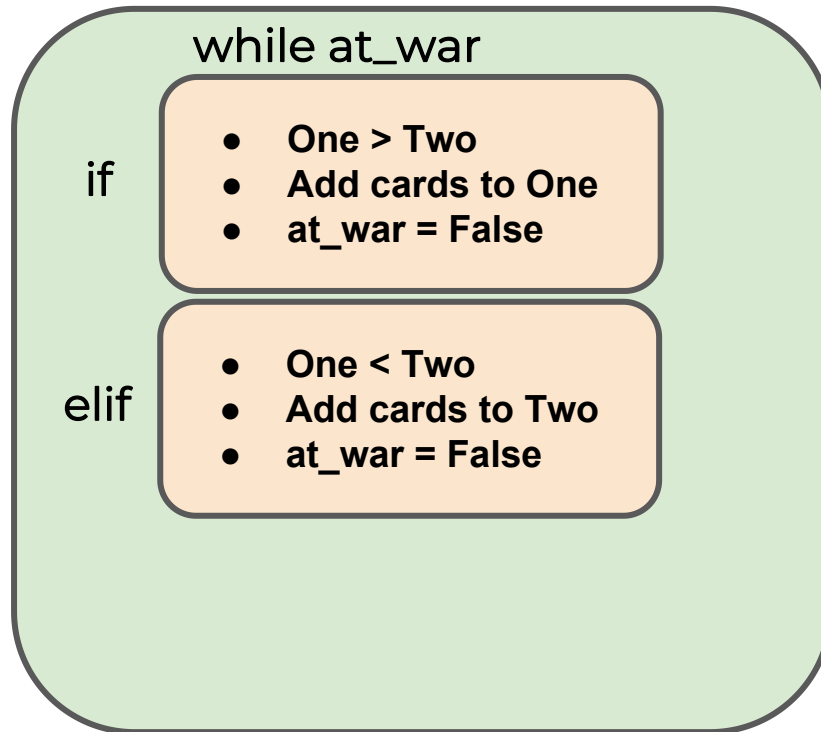
if

- One > Two
- Add cards to One
- at_war = False



Complete Python Bootcamp

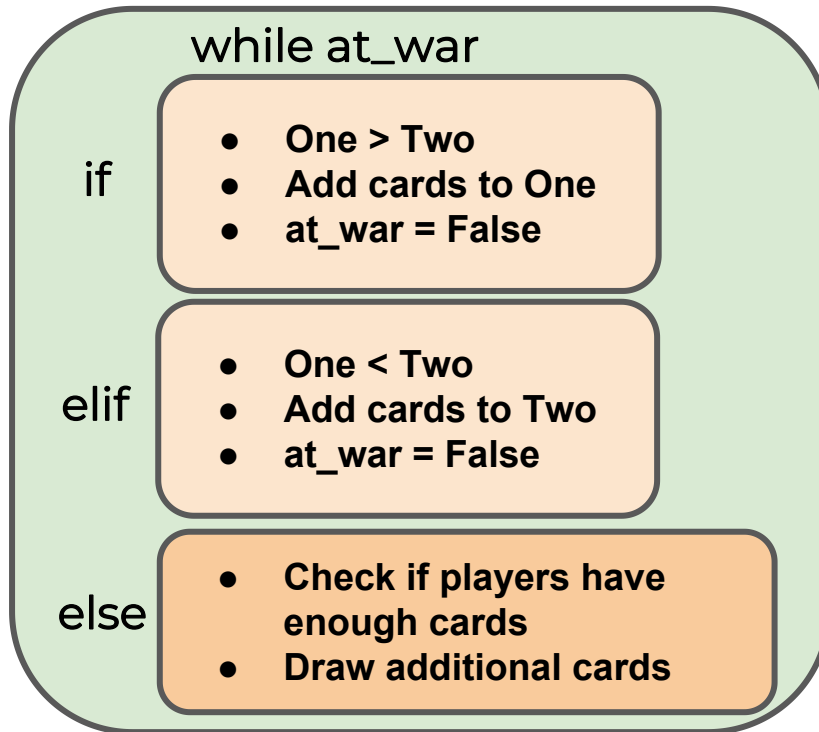
`at_war = True`





Complete Python Bootcamp

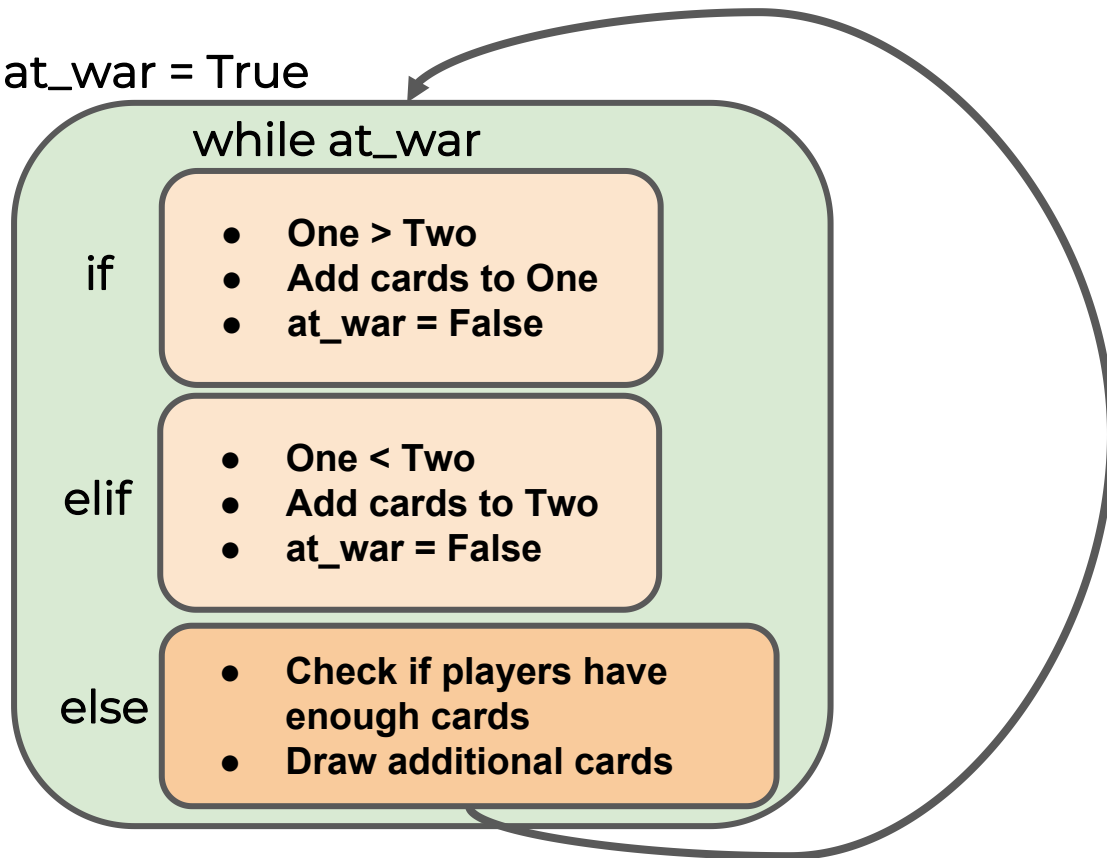
`at_war = True`





Complete Python Bootcamp

`at_war = True`





Milestone Project 2



Complete Python Bootcamp

- We've learned enough to start a second milestone project!
- You can treat this project a few ways:
 - Code along project with the solutions.
 - Attempt the project on your own.
 - Use the workbook as a guide for the project on your own.



Complete Python Bootcamp

- For this project you will use OOP to create a BlackJack Game with Python.
- Let's quickly go over the main idea of the game and discuss how OOP should be used for this project.



Complete Python Bootcamp

- For our version of the game we will only have a computer dealer and a human player.
- We start with a normal deck of cards, you will create a representation of a deck with Python.



Complete Python Bootcamp

COMPUTER DEALER



HUMAN PLAYER

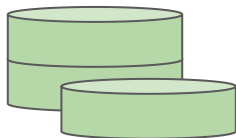


Complete Python Bootcamp

COMPUTER DEALER



PLAYER PLACES
A BET



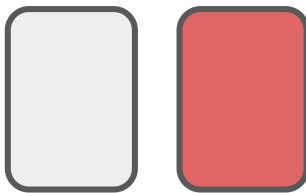
HUMAN PLAYER



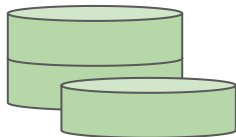
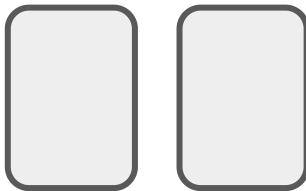
Complete Python Bootcamp

COMPUTER DEALER

Dealer starts with 1 card
face up and 1 card face
Down



Player starts with 2 cards
face up



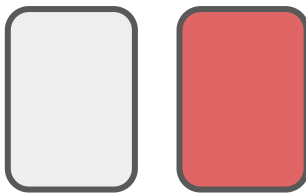
HUMAN PLAYER



Complete Python Bootcamp

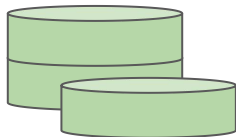
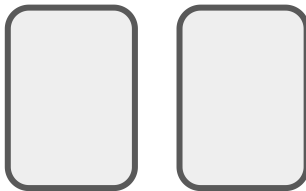
COMPUTER DEALER

Dealer starts with 1 card
face up and 1 card face
Down



PLAYER GOES
FIRST IN
GAMEPLAY

Player starts with 2 cards
face up

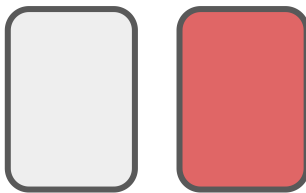


HUMAN PLAYER



Complete Python Bootcamp

COMPUTER DEALER



PLAYER GOAL: Get closer to a total value of 21 than the dealer does.

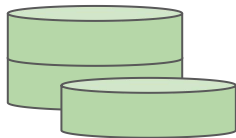
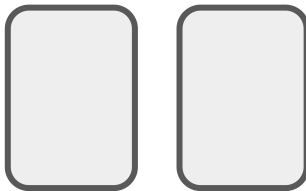
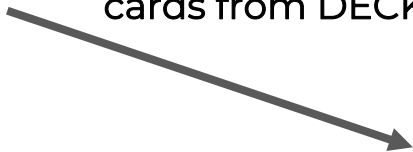
Possible Actions:

1. **Hit** (Receive another card)
2. **Stay** (Stop Receiving Cards)

We'll ignore actions like "Insurance", "Split", or "Double Down"



HIT for more
cards from DECK

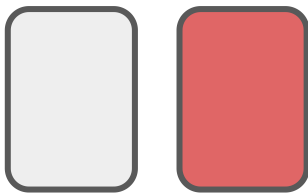


HUMAN PLAYER



Complete Python Bootcamp

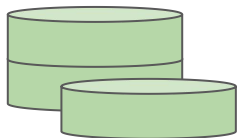
COMPUTER DEALER



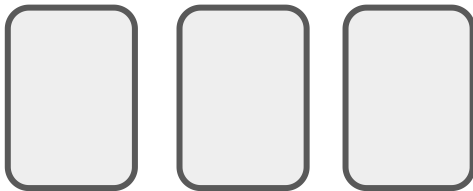
PLAYER GOAL: Get closer to a total value of 21 than the dealer does.



HIT for more
cards from DECK



HUMAN PLAYER



Possible Actions:

1. **Hit** (Receive another card)
2. **Stay** (Stop Receiving Cards)

We'll ignore actions like "Insurance", "Split", or "Double Down"



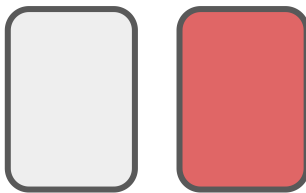


Complete Python Bootcamp

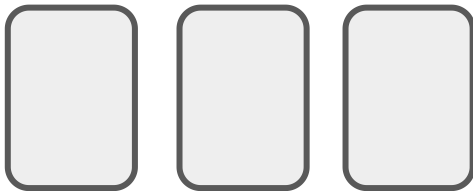
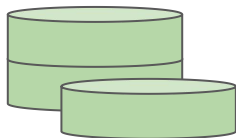
COMPUTER DEALER



Dealer HIT for
more cards from
DECK



AFTER PLAYER TURN:
2. If player is under 21,
dealer then hits until
they either beat the
player or the dealer
busts.

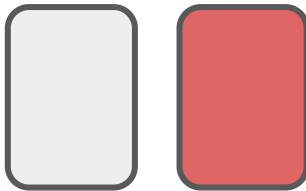


HUMAN PLAYER



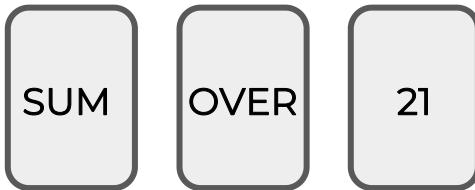
GAME END: PLAYER BUSTS

COMPUTER DEALER

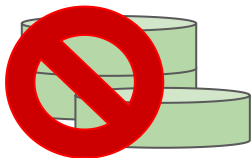


AFTER PLAYER TURN:

1. If player keeps hitting goes over 21, they bust and lost the bet!



The game is then over and dealer collects the money.



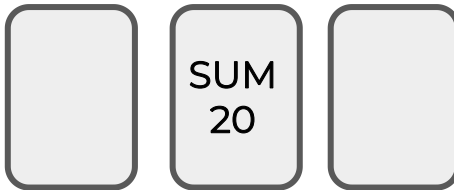
HUMAN PLAYER



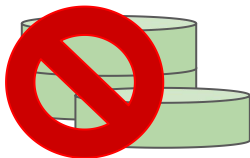
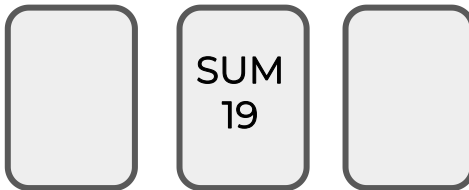
GAME END: Computer Beats Player

COMPUTER DEALER

Computer sum
higher than player
sum **and** still under
21.



AFTER PLAYER TURN:
2. If player is under 21,
dealer then hits until
they either beat the
player or the dealer
busts.

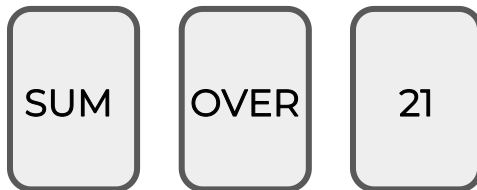


HUMAN PLAYER

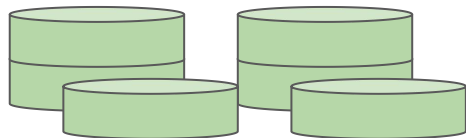
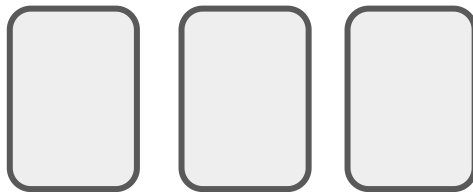


GAME END: PLAYER WINS

COMPUTER DEALER



AFTER PLAYER TURN:
2. If player is under 21, dealer then hits until they either beat the player or the dealer busts.



HUMAN PLAYER





Complete Python Bootcamp

- Special Rules:
 - Face Cards (Jack, Queen, King) count as a value of 10.
 - Aces can count as either 1 or 11 whichever value is preferable to the player.



Complete Python Bootcamp

- Check out the resource links for other explanations of BlackJack for more information.
- Let's now explore the project itself and the workbook!



Milestone Project 2

Example Solution