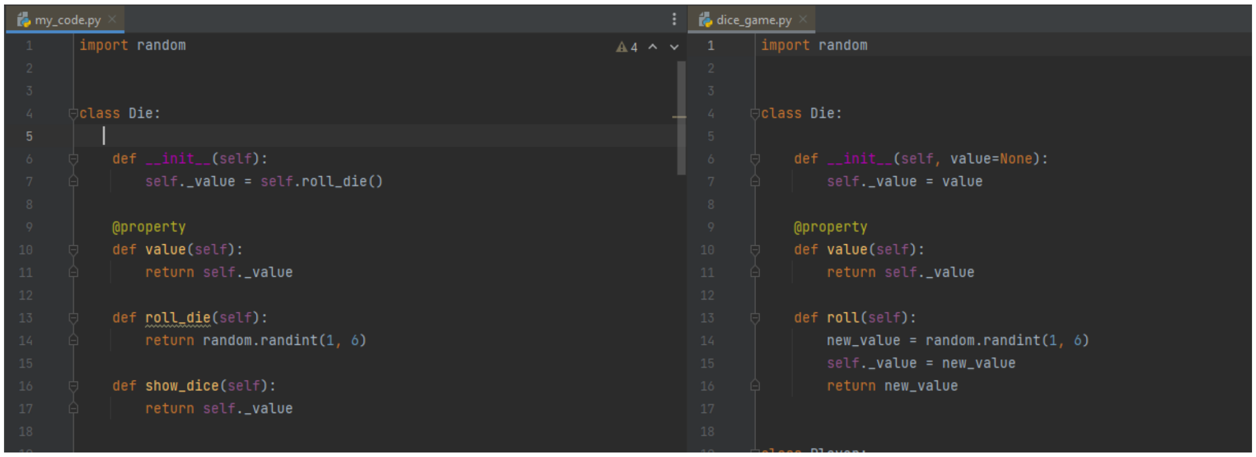


```python
import random


class Die:

    def __init__(self):
        self._value = self.roll_die()

    @property
    def value(self):
        return self._value

    def roll_die(self):
        return random.randint(1, 6)

    def show_dice(self):
        return self._value
```

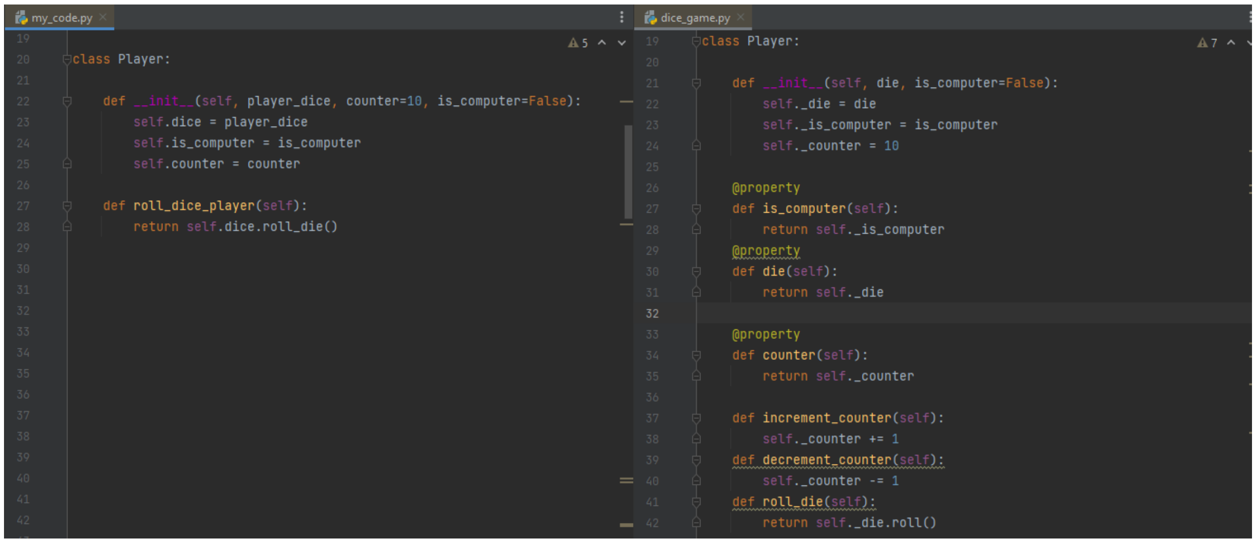
```python
import random


class Die:

    def __init__(self, value=None):
        self._value = value

    @property
    def value(self):
        return self._value

    def roll(self):
        new_value = random.randint(1, 6)
        self._value = new_value
        return new_value
```

Here as you can see, I didn't add the value attribute, because when introducing an instance we do npt give the value as a para,eter cause the value is something that is generated randomly.

You see how the instructor has introduced it by giving the input parameter a default value "None".

Secondly, instead of putting the vale attribute equal to the roll_dice methode, it puts it equal to the value param,eter. So here she do not give the value parameter its value inside the class itself, but it simply introduces the method and inside the method returns the new valuem but does not put this new_vale equal to the attribute itself.

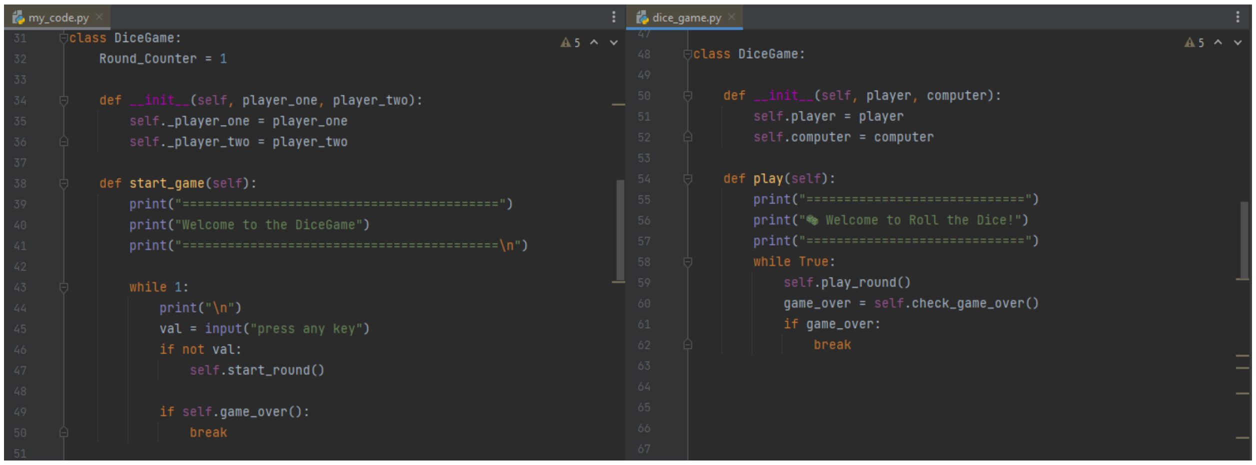
I also have an extra method.



```python
class Player:

    def __init__(self, player_dice, counter=10, is_computer=False):
        self.dice = player_dice
        self.is_computer = is_computer
        self.counter = counter

    def roll_dice_player(self):
        return self.dice.roll_die()
```

```python
class Player:

    def __init__(self, die, is_computer=False):
        self._die = die
        self._is_computer = is_computer
        self._counter = 10

    @property
    def is_computer(self):
        return self._is_computer
    @property
    def die(self):
        return self._die

    @property
    def counter(self):
        return self._counter

    def increment_counter(self):
        self._counter += 1
    def decrement_counter(self):
        self._counter -= 1
    def roll_die(self):
        return self._die.roll()
```

Here, I have two differences, firstly is that I have attributes that are public, so I should have defined getters, and secondly is the methods, that I didn't devised any methods for incrementing or decrementing, instead I directly defined them inside the Die_Game class.



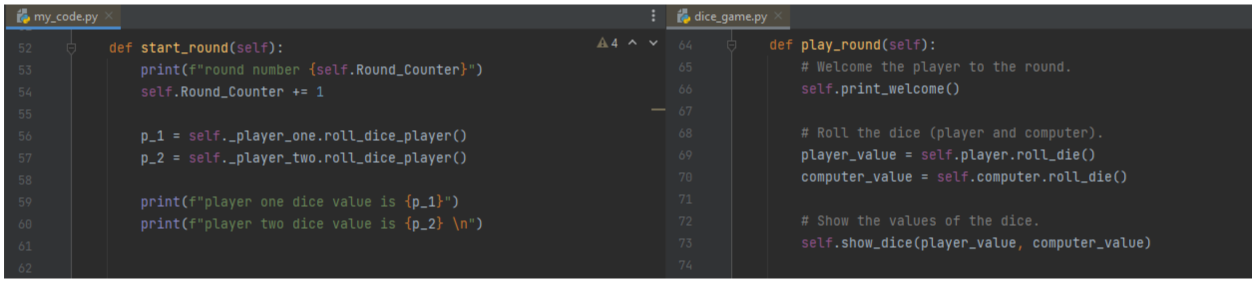
my_code.py

```python
class DiceGame:
    Round_Counter = 1

    def __init__(self, player_one, player_two):
        self._player_one = player_one
        self._player_two = player_two

    def start_game(self):
        print("==========================================")
        print("Welcome to the DiceGame")
        print("==========================================\n")

        while 1:
            print("\n")
            val = input("press any key")
            if not val:
                self.start_round()

            if self.game_over():
                break
```

dice_game.py

```python
class DiceGame:

    def __init__(self, player, computer):
        self.player = player
        self.computer = computer

    def play(self):
        print("============================")
        print("🎲 Welcome to Roll the Dice!")
        print("============================")
        while True:
            self.play_round()
            game_over = self.check_game_over()
            if game_over:
                break
```

Here, the the first part of the classes are generally the same, except for me having a redundant if condition. If not val, is always true, so whether we put the if or not it always happens. So I must had omit it!



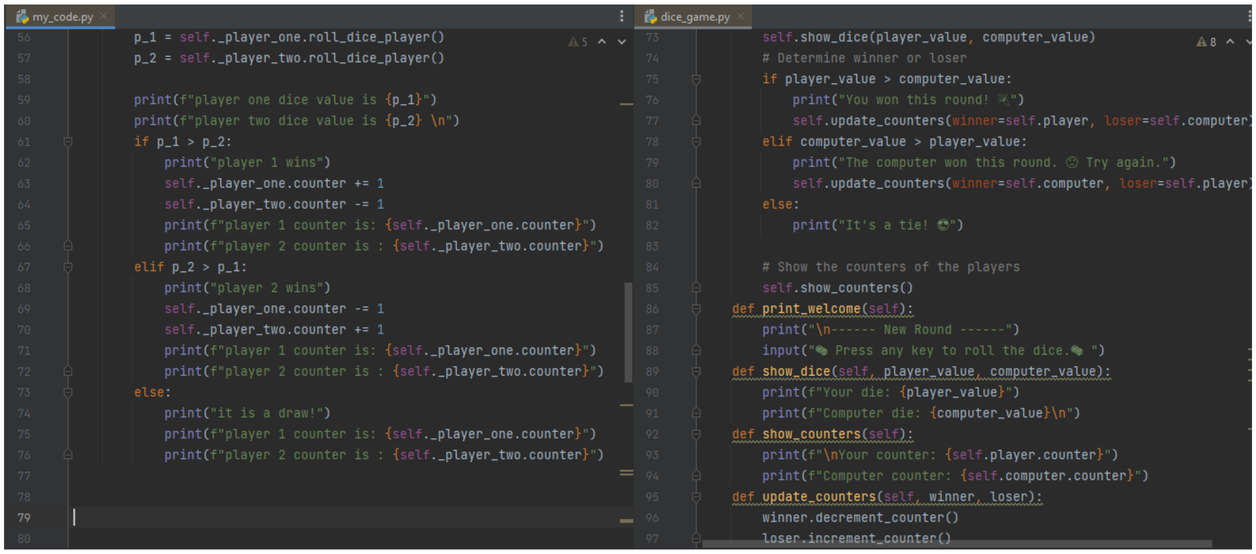
my_code.py

```python
    def start_round(self):
        print(f"round number {self.Round_Counter}")
        self.Round_Counter += 1

        p_1 = self._player_one.roll_dice_player()
        p_2 = self._player_two.roll_dice_player()

        print(f"player one dice value is {p_1}")
        print(f"player two dice value is {p_2} \n")
```

dice_game.py

```python
    def play_round(self):
        # Welcome the player to the round.
        self.print_welcome()

        # Roll the dice (player and computer).
        player_value = self.player.roll_die()
        computer_value = self.computer.roll_die()

        # Show the values of the dice.
        self.show_dice(player_value, computer_value)
```

In the second part of the DieGame class everything is the same, except the teacher has defined another method for showing the dice, in this case the code is much more readable.



my_code.py

```python
        p_1 = self._player_one.roll_dice_player()
        p_2 = self._player_two.roll_dice_player()

        print(f"player one dice value is {p_1}")
        print(f"player two dice value is {p_2} \n")
        if p_1 > p_2:
            print("player 1 wins")
            self._player_one.counter += 1
            self._player_two.counter -= 1
            print(f"player 1 counter is: {self._player_one.counter}")
            print(f"player 2 counter is : {self._player_two.counter}")
        elif p_2 > p_1:
            print("player 2 wins")
            self._player_one.counter -= 1
            self._player_two.counter += 1
            print(f"player 1 counter is: {self._player_one.counter}")
            print(f"player 2 counter is : {self._player_two.counter}")
        else:
            print("it is a draw!")
            print(f"player 1 counter is: {self._player_one.counter}")
            print(f"player 2 counter is : {self._player_two.counter}")
```

dice_game.py

```python
        self.show_dice(player_value, computer_value)
        # Determine winner or loser
        if player_value > computer_value:
            print("You won this round! 🎉")
            self.update_counters(winner=self.player, loser=self.computer
        elif computer_value > player_value:
            print("The computer won this round. 😞 Try again.")
            self.update_counters(winner=self.computer, loser=self.player
        else:
            print("It's a tie! 😎")

        # Show the counters of the players
        self.show_counters()
    def print_welcome(self):
        print("\n------ New Round ------")
        input("🎲 Press any key to roll the dice.🎲 ")
    def show_dice(self, player_value, computer_value):
        print(f"Your die: {player_value}")
        print(f"Computer die: {computer_value}\n")
    def show_counters(self):
        print(f"\nYour counter: {self.player.counter}")
        print(f"Computer counter: {self.computer.counter}")
    def update_counters(self, winner, loser):
        winner.decrement_counter()
        loser.increment_counter()
```

Here again the instructor has introduced many new methods to make the code more concise and readable.

First, she introduced method to show the value of the dice. I did it simply by writing the code directly. The logic is the same with the both of us.

```
def show_dice(self, player_value, computer_value):
    print(f"Your die: {player_value}")
    print(f"Computer die: {computer_value}\n")
```

Second, she updated the code by introducing a new method, which has another new method **from the player class** in it. I don't have these methods inside the player class.

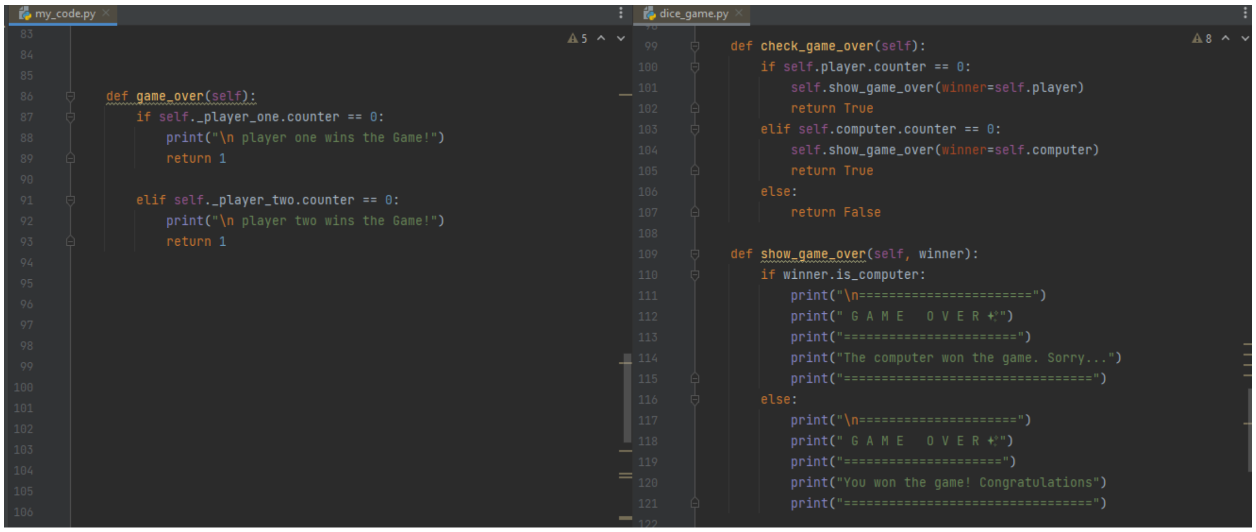
```
def update_counters(self, winner, loser): #new method
    winner.decrement_counter()#these two methods are coming from the player
class
    loser.increment_counter()
```

the other last thing is that I have a **repetition** in my code, which is the same for the three if clauses, which could be avoided, by bring it out the if clauses.

```
print(f"player 1 counter is: {self._player_one.counter}")
print(f"player 2 counter is : {self._player_two.counter}")
```

in this case also the instructor has defined a new method for it:

```
def show_counters(self):
    print(f"\nYour counter: {self.player.counter}")
    print(f"Computer counter: {self.computer.counter}")
```



my_code.py

```
    def game_over(self):
        if self._player_one.counter == 0:
            print("\n player one wins the Game!")
            return 1

        elif self._player_two.counter == 0:
            print("\n player two wins the Game!")
            return 1
```

dice_game.py

```
    def check_game_over(self):
        if self.player.counter == 0:
            self.show_game_over(winner=self.player)
            return True
        elif self.computer.counter == 0:
            self.show_game_over(winner=self.computer)
            return True
        else:
            return False

    def show_game_over(self, winner):
        if winner.is_computer:
            print("\n======================")
            print(" G A M E   O V E R ✨")
            print("======================")
            print("The computer won the game. Sorry...")
            print("================================")
        else:
            print("\n====================")
            print(" G A M E   O V E R ✨")
            print("====================")
            print("You won the game! Congratulations")
            print("================================")
```

In the last part, the game over method varies significantly from mine. She has a new method which is called in the game over method. In this new method, if the winner is the computer, (so winner.is_computer == True) the corresponding message would be called. While if not, the message related to the player would be called. The reason that I do not have these methods are that I have introduced the player number two as a general player and not the computer.