Player Class

Represents a player in the game.

```
class Player:
   """Represents a player in the game."""

   def __init__(self, symbol):
       """Initializes a Player object.

       Args:
           symbol: The player's symbol ('X' or 'O').
       """
       self.symbol = symbol
```

Game Class

Manages the Tic-Tac-Toe game logic.

```
class Game:
   """Manages the Tic-Tac-Toe game logic."""

   def __init__(self):
       """Initializes a Game object."""
       self.board = [[' ' for _ in range(3)] for _ in range(3)]
       self.current_player = Player('X')  # Start with 'X'

   def print_board(self):
       """Prints the current game board to the console."""
       print("-------------")
       for row in self.board:
           print("|", end=" ")
           for cell in row:
               print(cell, "|", end=" ")
           print()
           print("-------------")

   def get_player_move(self):
       """Gets and validates player input for their move."""
       while True:
           try:
               row, col = map(int, input(f"Player {self.current_player.symbol}, enter row and column (1-3, 1-3): ").split())
               row -= 1  # Adjust to 0-based indexing
               col -= 1
               if 0 <= row < 3 and 0 <= col < 3 and self.board[row][col] == ' ':
                   return row, col
               else:
                   print("Invalid move. Try again.")
           except ValueError:
               print("Invalid input. Please enter two numbers separated by a space.")

   def update_board(self, row, col):
       """Updates the game board with the current player's move."""
       self.board[row][col] = self.current_player.symbol

   def check_win(self):
       """Checks if the current player has won the game."""
       # Check rows
       for row in self.board:
           if all(cell == self.current_player.symbol for cell in row):
               return True

       # Check columns
       for col in range(3):
           if all(self.board[row][col] == self.current_player.symbol for row in range(3)):
               return True
```

```
        # Check diagonals
        if all(self.board[i][i] == self.current_player.symbol for i in range(3)):
            return True
        if all(self.board[i][2 - i] == self.current_player.symbol for i in range(3)):
            return True

        return False

    def check_draw(self):
        """Checks if the game is a draw."""
        return all(cell != ' ' for row in self.board for cell in row)

    def switch_player(self):
        """Switches the current player to the other player."""
        self.current_player = Player('O') if self.current_player.symbol == 'X' else Player('X')

    def play(self):
        """Runs the main game loop."""
        while True:
            self.print_board()
            row, col = self.get_player_move()
            self.update_board(row, col)
            if self.check_win():
                self.print_board()
                print(f"Player {self.current_player.symbol} wins!")
                break
            elif self.check_draw():
                self.print_board()
                print("It's a draw!")
                break
            self.switch_player()
```

Main Execution Block

Initializes the game and starts the game loop.

```
if __name__ == "__main__":
    game = Game()
    game.play()
```