CSC236 Problem Set 2

Question 5.

(a) I'll call the player who takes the first turn player 1.

Define $P(n)$: 'there is a strategy for player 1 to win the game when the game starting with s_1, s_2 pebbles in two piles where $s_1, s_2 \in \mathbb{N}$, $s_1 \neq s_2$., and $n = \min(s_1, s_2)$. where $n \in \mathbb{N}$.

WTP: $\forall n \in \mathbb{N}, P(n)$.

Let $s_1, s_2 \in \mathbb{N}$. $s_1 \neq s_2$. Let $n \in \mathbb{N}$, $n = \min(s_1, s_2)$.

Base Case: $n = 0$.

when $n=0$, which $n = 0 = \min(s_1, s_2)$. Since $s_1, s_2 \in \mathbb{N}$ and $s_1 \neq s_2$, one of the piles must have more than 0 pebbles. Since it's player 1's turn to play and according to the rule, it just needs to remove every pebbles from the pile that is not 0, which player 1 will always win.

I've proved the base case is true.

Induction Step. Let $n \in \mathbb{N}, n \geq 1$

Induction Hypothesis. $\forall k \in \mathbb{N}, 0 \leq k < n \Rightarrow P(k)$.

WTP: $P(n)$.

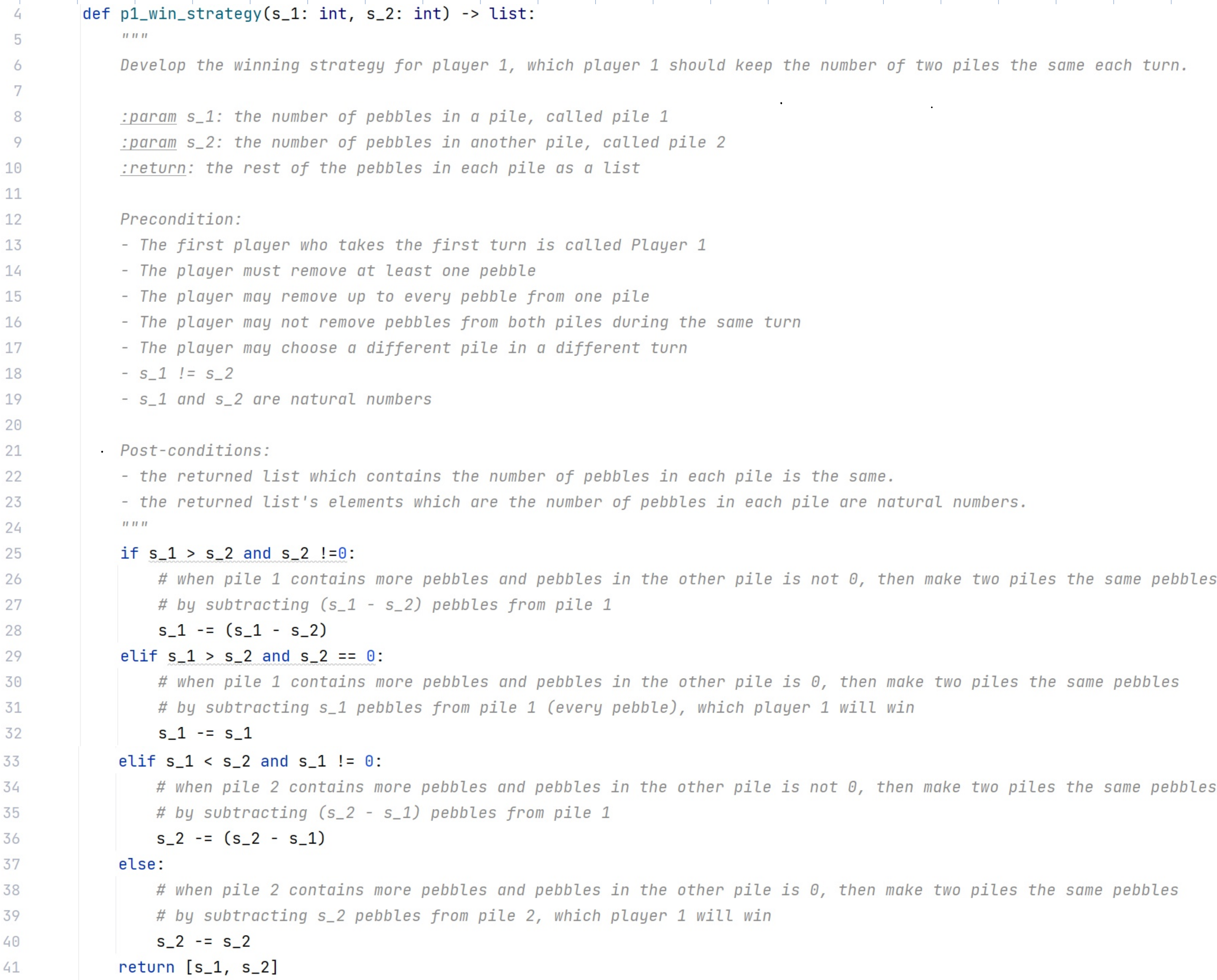
Player 1 just need to remove $|s_1 - s_2|$ pebbles from the pile contains more pebbles, making both piles contain n pebbles.. No matter which pile player 2 chooses, the pile player 2 choose will less then the other pile, which the number of pebbles in the pile player 2 removed is k', which $0 \leq k' < n$. Correspondingly, it's time for player 1 to play. according to the I.H., $P(k')$ holds, as $k' = \min(n, k')$.

I've proved the induction step.

Therefore, If the game starts with two piles having different numbers of piles, then there's a strategy that guarantees a win for the player. who takes the first turn

■

(b)

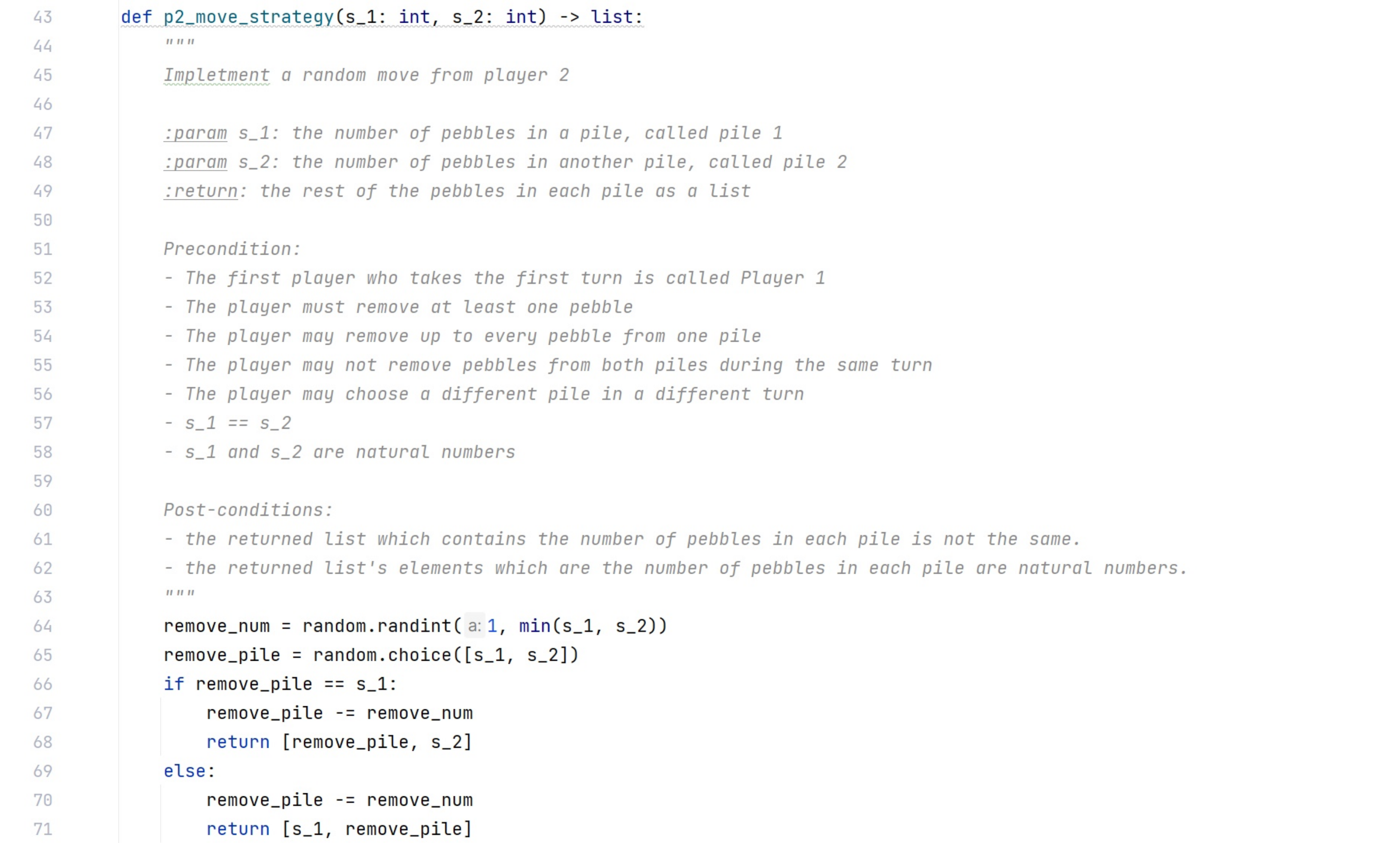


```python
def p1_win_strategy(s_1: int, s_2: int) -> list:
    """
    Develop the winning strategy for player 1, which player 1 should keep the number of two piles the same each turn.

    :param s_1: the number of pebbles in a pile, called pile 1
    :param s_2: the number of pebbles in another pile, called pile 2
    :return: the rest of the pebbles in each pile as a list

    Precondition:
    - The first player who takes the first turn is called Player 1
    - The player must remove at least one pebble
    - The player may remove up to every pebble from one pile
    - The player may not remove pebbles from both piles during the same turn
    - The player may choose a different pile in a different turn
    - s_1 != s_2
    - s_1 and s_2 are natural numbers

    Post-conditions:
    - the returned list which contains the number of pebbles in each pile is the same.
    - the returned list's elements which are the number of pebbles in each pile are natural numbers.
    """
    if s_1 > s_2 and s_2 !=0:
        # when pile 1 contains more pebbles and pebbles in the other pile is not 0, then make two piles the same pebbles
        # by subtracting (s_1 - s_2) pebbles from pile 1
        s_1 -= (s_1 - s_2)
    elif s_1 > s_2 and s_2 == 0:
        # when pile 1 contains more pebbles and pebbles in the other pile is 0, then make two piles the same pebbles
        # by subtracting s_1 pebbles from pile 1 (every pebble), which player 1 will win
        s_1 -= s_1
    elif s_1 < s_2 and s_1 != 0:
        # when pile 2 contains more pebbles and pebbles in the other pile is not 0, then make two piles the same pebbles
        # by subtracting (s_2 - s_1) pebbles from pile 1
        s_2 -= (s_2 - s_1)
    else:
        # when pile 2 contains more pebbles and pebbles in the other pile is 0, then make two piles the same pebbles
        # by subtracting s_2 pebbles from pile 2, which player 1 will win
        s_2 -= s_2
    return [s_1, s_2]
```

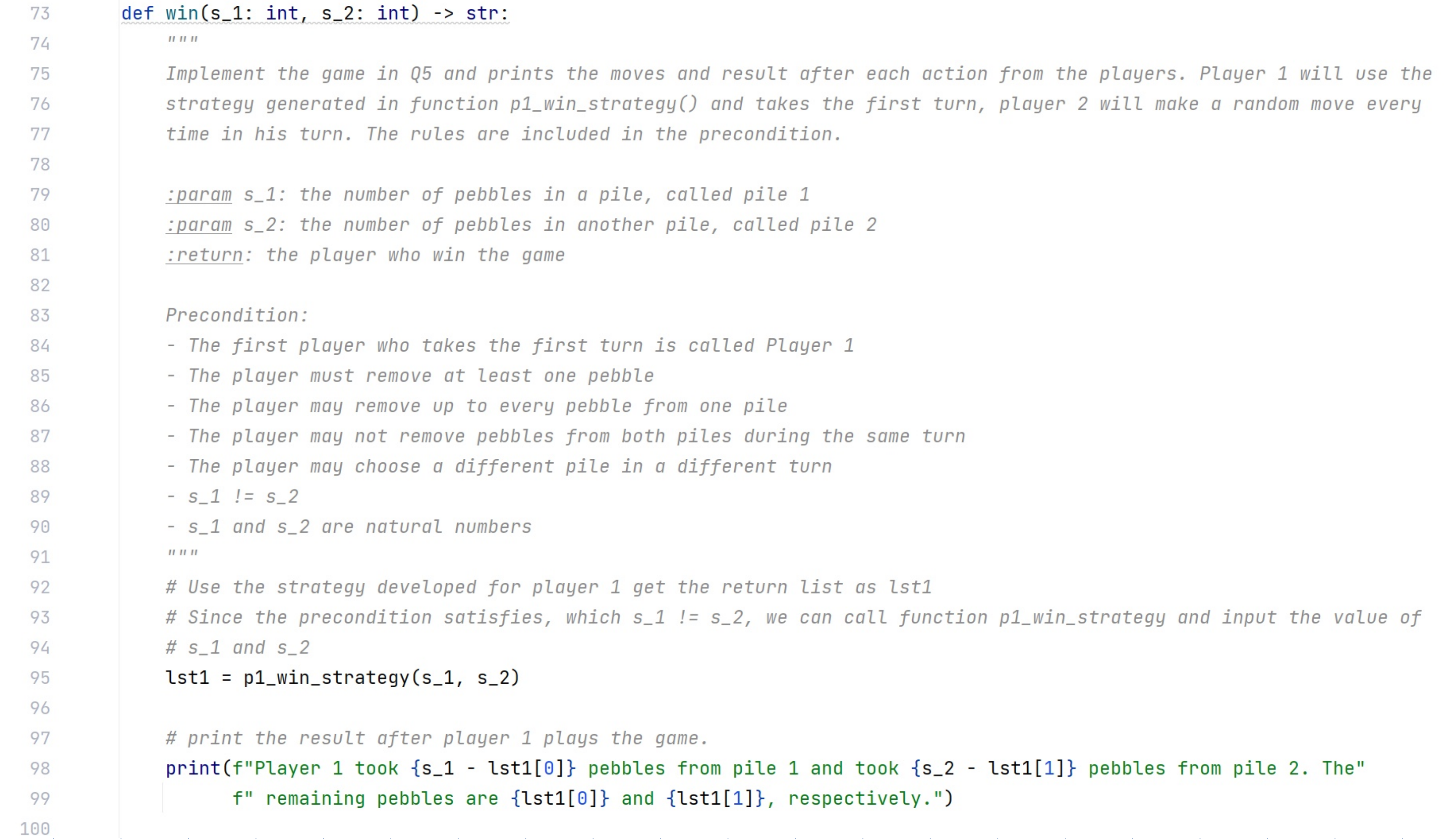


```python
def p2_move_strategy(s_1: int, s_2: int) -> list:
    """
    Impletment a random move from player 2

    :param s_1: the number of pebbles in a pile, called pile 1
    :param s_2: the number of pebbles in another pile, called pile 2
    :return: the rest of the pebbles in each pile as a list

    Precondition:
    - The first player who takes the first turn is called Player 1
    - The player must remove at least one pebble
    - The player may remove up to every pebble from one pile
    - The player may not remove pebbles from both piles during the same turn
    - The player may choose a different pile in a different turn
    - s_1 == s_2
    - s_1 and s_2 are natural numbers

    Post-conditions:
    - the returned list which contains the number of pebbles in each pile is not the same.
    - the returned list's elements which are the number of pebbles in each pile are natural numbers.
    """
    remove_num = random.randint(a: 1, min(s_1, s_2))
    remove_pile = random.choice([s_1, s_2])
    if remove_pile == s_1:
        remove_pile -= remove_num
        return [remove_pile, s_2]
    else:
        remove_pile -= remove_num
        return [s_1, remove_pile]
```

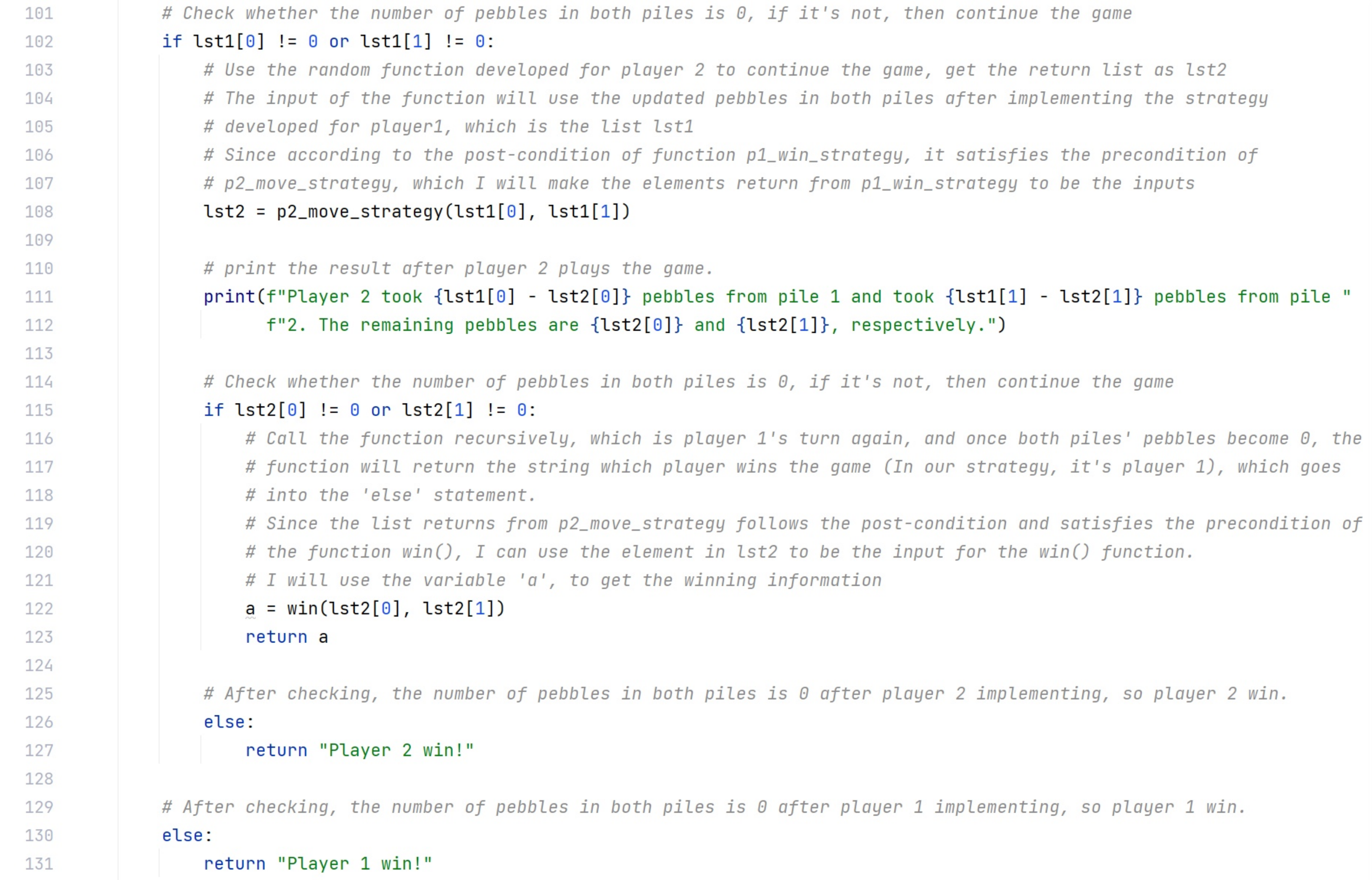


```python
def win(s_1: int, s_2: int) -> str:
    """
    Implement the game in Q5 and prints the moves and result after each action from the players. Player 1 will use the
    strategy generated in function p1_win_strategy() and takes the first turn, player 2 will make a random move every
    time in his turn. The rules are included in the precondition.

    :param s_1: the number of pebbles in a pile, called pile 1
    :param s_2: the number of pebbles in another pile, called pile 2
    :return: the player who win the game

    Precondition:
    - The first player who takes the first turn is called Player 1
    - The player must remove at least one pebble
    - The player may remove up to every pebble from one pile
    - The player may not remove pebbles from both piles during the same turn
    - The player may choose a different pile in a different turn
    - s_1 != s_2
    - s_1 and s_2 are natural numbers
    """
    # Use the strategy developed for player 1 get the return list as lst1
    # Since the precondition satisfies, which s_1 != s_2, we can call function p1_win_strategy and input the value of
    # s_1 and s_2
    lst1 = p1_win_strategy(s_1, s_2)

    # print the result after player 1 plays the game.
    print(f"Player 1 took {s_1 - lst1[0]} pebbles from pile 1 and took {s_2 - lst1[1]} pebbles from pile 2. The"
          f" remaining pebbles are {lst1[0]} and {lst1[1]}, respectively.")
```



```
    # Check whether the number of pebbles in both piles is 0, if it's not, then continue the game
    if lst1[0] != 0 or lst1[1] != 0:
        # Use the random function developed for player 2 to continue the game, get the return list as lst2
        # The input of the function will use the updated pebbles in both piles after implementing the strategy
        # developed for player1, which is the list lst1
        # Since according to the post-condition of function p1_win_strategy, it satisfies the precondition of
        # p2_move_strategy, which I will make the elements return from p1_win_strategy to be the inputs
        lst2 = p2_move_strategy(lst1[0], lst1[1])

        # print the result after player 2 plays the game.
        print(f"Player 2 took {lst1[0] - lst2[0]} pebbles from pile 1 and took {lst1[1] - lst2[1]} pebbles from pile "
              f"2. The remaining pebbles are {lst2[0]} and {lst2[1]}, respectively.")

        # Check whether the number of pebbles in both piles is 0, if it's not, then continue the game
        if lst2[0] != 0 or lst2[1] != 0:
            # Call the function recursively, which is player 1's turn again, and once both piles' pebbles become 0, the
            # function will return the string which player wins the game (In our strategy, it's player 1), which goes
            # into the 'else' statement.
            # Since the list returns from p2_move_strategy follows the post-condition and satisfies the precondition of
            # the function win(), I can use the element in lst2 to be the input for the win() function.
            # I will use the variable 'a', to get the winning information
            a = win(lst2[0], lst2[1])
            return a

        # After checking, the number of pebbles in both piles is 0 after player 2 implementing, so player 2 win.
        else:
            return "Player 2 win!"

    # After checking, the number of pebbles in both piles is 0 after player 1 implementing, so player 1 win.
    else:
        return "Player 1 win!"
```