The provided Python code implements a binary search algorithm to find the index of a given element within a sorted list. Here's a step-by-step explanation of the code:-

```
def binary_search(list, element):
   middle = 0
   start = 0
   end = len(list)
   steps = 0
   while(start<=end):
      print("Step", steps, ":" ,str(list[start:end+1]))
      steps = steps+1
      middle = (start + end) // 2
      if element == list[middle] :
         return middle
      if element < list[middle]:
         end = middle -1
      else:
         start = middle + 1
   return -1
my_list = [1,2,3,4,5,6,7,8,9,10,11,12]
target = 12
binary_search(my_list, target)
```

1. **binary_search function:-**

   - The function takes two parameters: list (the sorted list to search in) and element (the target element to find).
   - It initializes several variables:
     - **middle: A variable to store the index of the middle element of the current search range.**
     - **start: The index of the first element of the current search range, initially set to 0.**
     - **end: The index of the last element of the current search range, initially set to the length of the list.**
     - **steps: A variable to count the number of steps taken in the binary search.**
2. The function enters a while loop with the condition start <= end, which means that it will keep searching as long as the start index is less than or equal to the end index.
3. Inside the loop, it prints the current step and the sublist of the list that is being considered for the search:-

```
print("Step", steps, ":" ,str(list[start:end+1]))
```

4. It increments the steps counter.

5. The middle index is calculated as the average of start and end using integer division:-

```
middle = (start + end) // 2
```

6. The code checks if the element is equal to the element at the middle index. If they are equal, it returns the middle index, indicating that the element has been found.

7. If the element is less than the element at the middle index, it updates the end index to middle - 1, effectively reducing the search range to the lower half of the current range.

8. If the element is greater than the element at the middle index, it updates the start index to middle + 1, effectively reducing the search range to the upper half of the current range.

9. If the element is not found after completing the loop (i.e., start exceeds end), the function returns -1 to indicate that the element is not in the list.

10. Finally, the function is called with a sample sorted list my_list and a target element target. In this case, it's searching for element 12 within the list. The result of the binary search is not stored or printed, so you may want to modify the code to display the result.