JavaScript

Approach 1

```javascript
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(nums, target) {
    const map = new Map(); // to store number and its index

    for (let i = 0; i < nums.length; i++) {
        const complement = target - nums[i];

        if (map.has(complement)) {
            return [map.get(complement), i]; // found the pair
        }

        map.set(nums[i], i); // store current number with its index
    }

    return [];
};
```

Approach 2

```javascript
/**
 * @param {number[]} nums
 * @param {number} target
 * @return {number[]}
 */
var twoSum = function(nums, target) {
    // let nums = [3,3]
    // let target = 6
    let [res, dic] =[[], {}]

    nums.forEach((ele, inx, arr) => {
        let rem = target - ele
```

```
        dic[ele] = inx
        let ix = nums.findIndex((el, ix) => el == rem && ix != inx)
        if (ix != -1) res = [ix, inx]
    })
    return res
};
```

Python

```python
class Solution:
    def twoSum(self, nums: List[int], target: int) -> List[int]:
        # nums = [3,3]
        # target = 6
        dic = {}
        for index, value in enumerate(nums):
            com = target - value
            if value not in dic.keys():
                dic[value] = index
            if com in dic.keys() and index != dic[com]:
                return [dic[com], index]
```

Java

```java
import java.util.HashMap;

class Solution {
    public int[] twoSum(int[] nums, int target) {
        HashMap<Integer, Integer> map = new HashMap<>(); // value -> index
        for (int i = 0; i < nums.length; i++) {
            int complement = target - nums[i];
            if (map.containsKey(complement)) {
                return new int[]{ map.get(complement), i };
            }
            map.put(nums[i], i);
        }
        return new int[]{}; // if no result is found
    }
}
```