Reverse String

Javascript

```
const reverseInput = (input) => {
/* Find the type of the input and create an empty array to store the reversed items
*/

   let typeOf = typeof input;
   let reversed = [];

   const reverse = (input) => {
/* Loop over the input starting at the end and push into the reversed array, then
return it */
       for (let i = input.length - 1; i >= 0; i--) {
           reversed.push(input[i]);
       }
       return reversed;
   }

/* If the input is a number, convert it to a string so we can iterate over it, then
run the string through the reverse function. Convert the returned array from
reverse() into a string, then into a number and return. */

   if (typeOf === "number") {
       let stringifyNumber = input.toString();
       let reversed = reverse(stringifyNumber);
       let convertFromArray = parseInt(reversed.join(''));
       return convertFromArray;
   }



/* If the type is a string, run it through the reverse() function and join the
result to return a string */
   else if (typeOf === "string") {
       return reverse(input).join('');
   }
```

```
/* If the input is an array, run it through the reverse() function and return the
result */

   else if (Array.isArray(input)) {
       return reverse(input);
   }

/* We are not accepting other input types, so all other inputs should be discarded.
*/

   else {
       console.log("This input type is not supported");
   }
}

console.log(reverseInput("Reversing a string")); // "gnirts a gnisreveR"
console.log(reverseInput(12345)); // 54321
console.log(reverseInput(["a", "b", 1, "c", 2])); // [2, "c", 1, "b", "a"]
```

https://eddmann.com/posts/ten-ways-to-reverse-a-string-in-javascript/

Palindromes

Javascript

```
const isPalindrome = (input) => {
/* Edge case: if the input's length is an odd number, ignore the middle (pivot). If
it is even, compare all. */

   const compare = (input) => {
       for (let i = 0; i < Math.floor(input.length / 2); i ++) {
           if (input[i] !== input[input.length - 1 - i]) {
               return false;
           }
           else {
               continue;
           }
       }
       return true;
   };

   if (typeof input === "number") {
/* If the type is a number, we need to convert to string to iterate */
       let newString = input.toString();
       return compare(newString);
   }
   else if (typeof input === "string" || Array.isArray(input)) {
       return compare(input);
   }
   else {
       console.log("This input type is not supported");
   }
}
console.log(isPalindrome("tacocat")); // true
console.log(isPalindrome("Tacocat")); // false
console.log(isPalindrome(123456)); // false
console.log(isPalindrome([1, 2, 3, 4, "4", 3, 2, 1])); // false
```

https://stackoverflow.com/questions/14813369/palindrome-check-in-javascript

Outermost Parentheses

Javascript

```
var removeOuterParentheses = function(S) {
    let result = '';
    let open = 0
    for (let i = 0; i < S.length; i++) {
        if (S[i] === '(') {
            if (open > 0) {
                        result += '(';
                }
                open++;
        } else if (S[i] === ')') {
            if (open > 1) {
                        result += ')';
                }
                open--;
        }
    }
    return result;
};
```

https://leetcode.com/problems/remove-outermost-parentheses/discuss/301174/Javascript-beats-99.26-easy-to-understand

Robot Return to Origin

```
var judgeCircle = function(moves) {

    let horiz = 0;
    let vert = 0;

    for(let i = 0; i < moves.length; i++) {
        if(moves[i] == "L") {
            horiz -= 1;
        } else if(moves[i] == "R") {
            horiz += 1;
        } else if(moves[i] == "U") {
            vert += 1;
        } else {
            vert -=1;
        }
    }

    if(horiz == 0 && vert == 0) {
        return true;
    }
    return false;

};
```

https://leetcode.com/problems/robot-return-to-origin/discuss/278954/Javascript-beats-100

Construct Binary Tree

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val) {
 *     this.val = val;
 *     this.left = this.right = null;
 * }
 */

let pIndex;
var buildTree = function(preorder, inorder) {
    if (!preorder)
        return null;
    pIndex = 0;
    let res = build(preorder, inorder, 0, inorder.length -1);
    return res;
};

let build = function(preorder, inorder, istart, iend) {
    if (istart > iend)
        return null;

    let val = preorder[pIndex];
    let node = new TreeNode(val);
    pIndex++;

    if (istart === iend)
        return node;

    let index = searchInOrder(val, inorder, istart, iend);
    node.left = build(preorder, inorder, istart, index-1);
    node.right = build(preorder, inorder, index+1, iend);

    return node;
};

let searchInOrder = function(val, inOrder, start, end){
    let i = start;
    while(i <= end){
        if (inOrder[i] === val)
            break;
      i++;
    }
    return i;
};
```

Find Islands

```
var numIslands = function(grid) {
  // result we are returning
  let count = 0;

  // store given grid
  let g = grid;

  // rows of grid
  let x = g.length;
  if (g.length == 0) return 0;

  // columns of grid
  let y = g[0].length;

  // recursion helper function to check each point and its adjacent points
  // check current point g[i][j], if it's water, skip
  // if it's land, mark it as visited ('0') then check its adjacent point
  const dfs = function (i, j) {
   if (i < 0 || i >= x || j < 0 || j >= y || g[i][j] != '1') return;

    // mark it as visited
    g[i][j] = '0';

    // check its adjacent points
    dfs(i+1, j)
    dfs(i-1, j)
    dfs(i, j+1)
    dfs(i, j-1)
  }

  // iterate through each point in the grid
  for (let r = 0; r < x; r++) {
    for (let c = 0; c < y; c++) {
     if (g[r][c] == '1') {

        // run helper at current point see
        dfs(r, c)

        // count+1 when dfs helper finishes, meaning one piece of land is marked as
visited
        count++
      }
    }
  }
 return count;
};
```