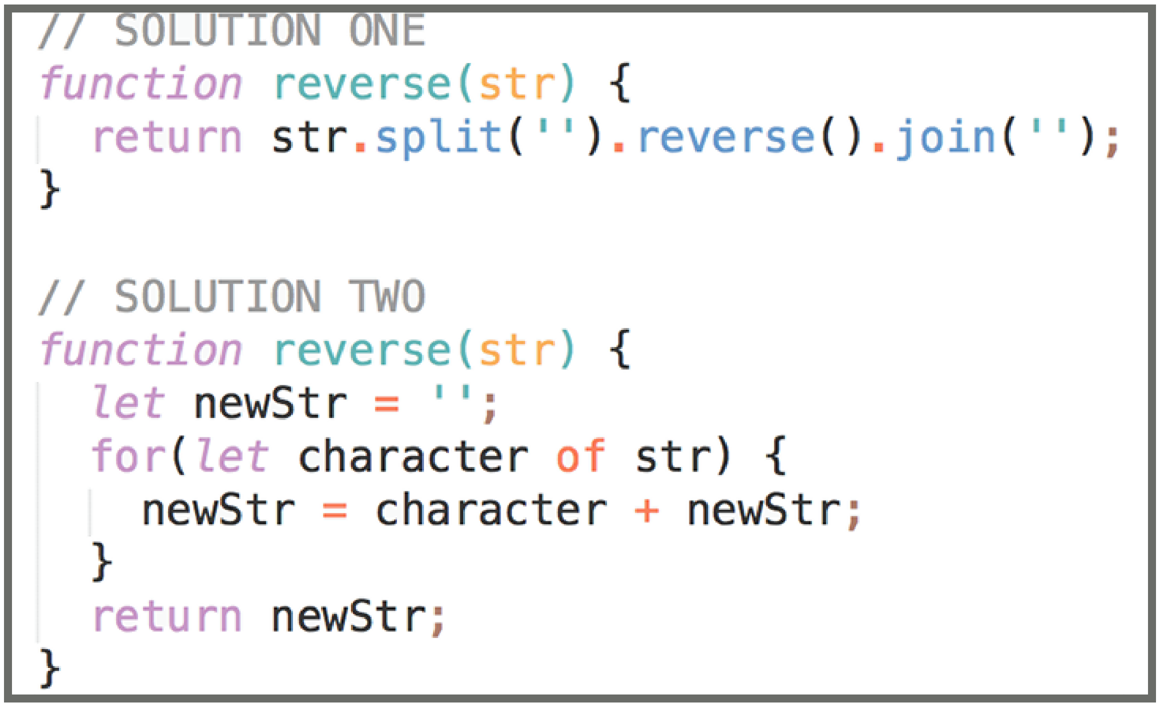
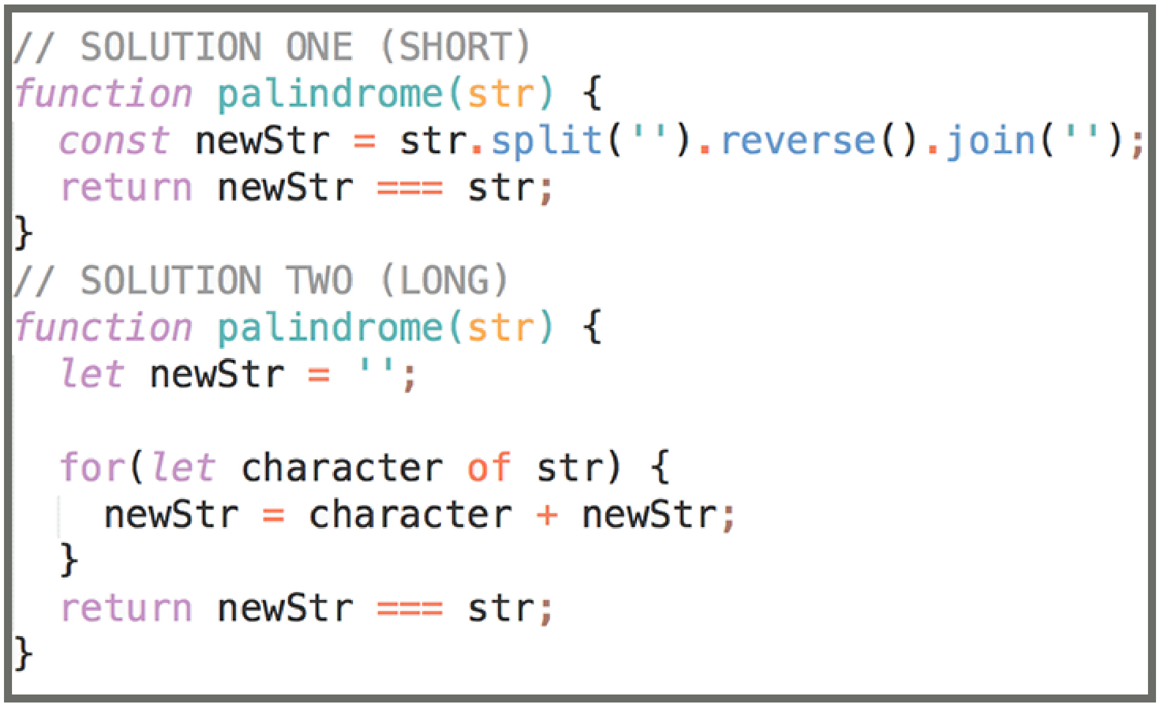
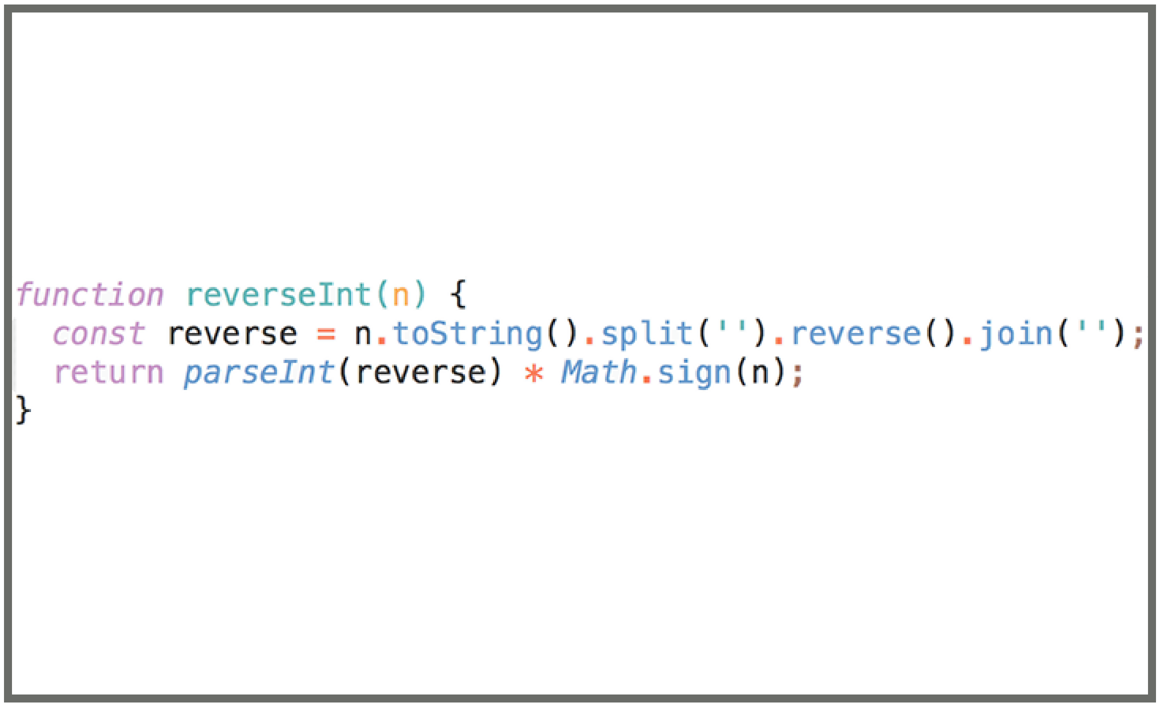


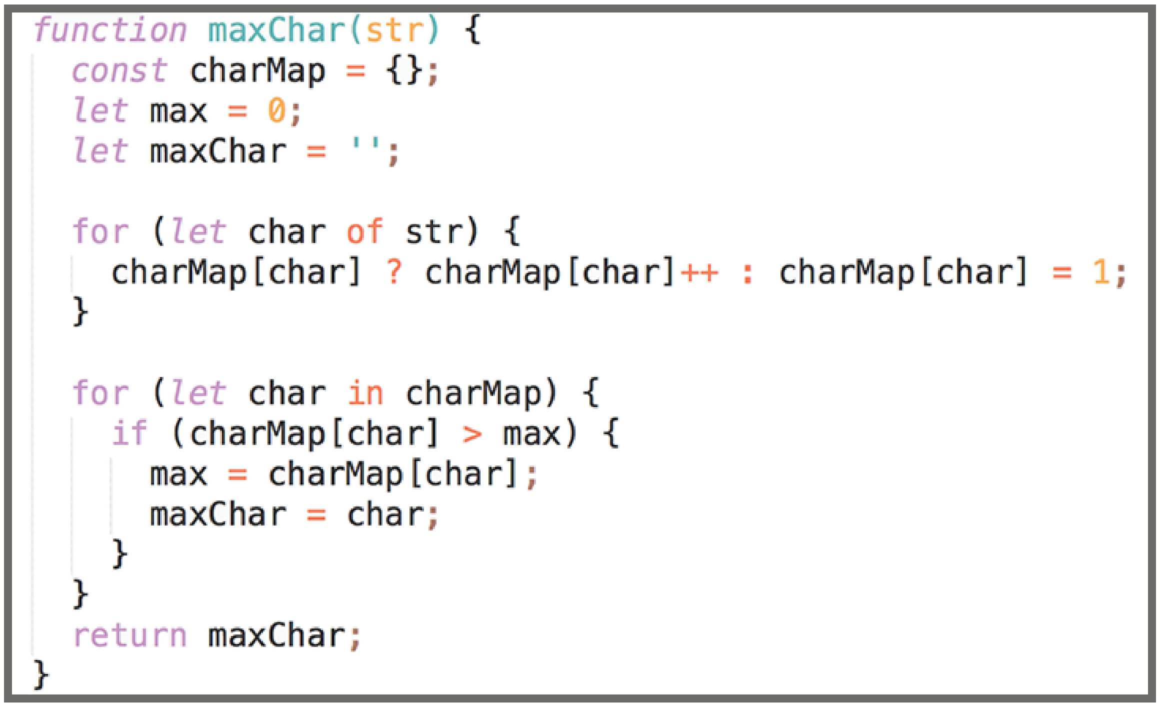
```
// SOLUTION ONE
function reverse(str) {
  return str.split('').reverse().join('');
}

// SOLUTION TWO
function reverse(str) {
  let newStr = '';
  for(let character of str) {
    newStr = character + newStr;
  }
  return newStr;
}
```



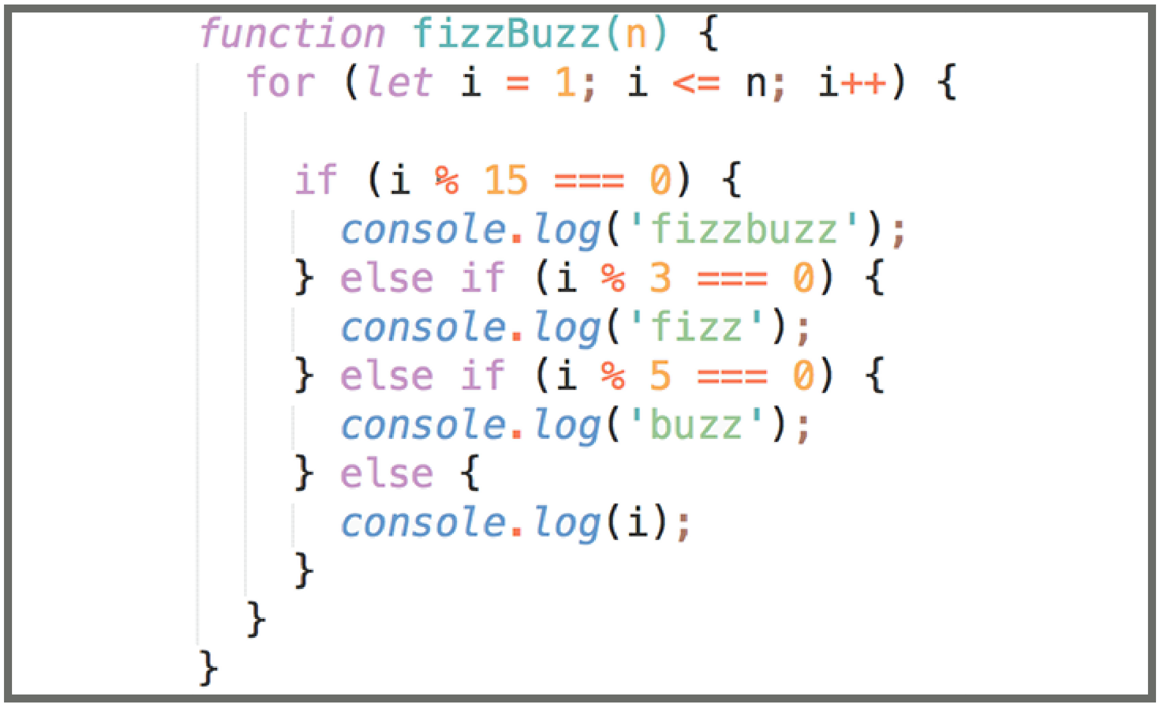
```
// SOLUTION ONE (SHORT)
function palindrome(str) {
  const newStr = str.split('').reverse().join('');
  return newStr === str;
}
// SOLUTION TWO (LONG)
function palindrome(str) {
  let newStr = '';

  for(let character of str) {
    newStr = character + newStr;
  }
  return newStr === str;
}
```



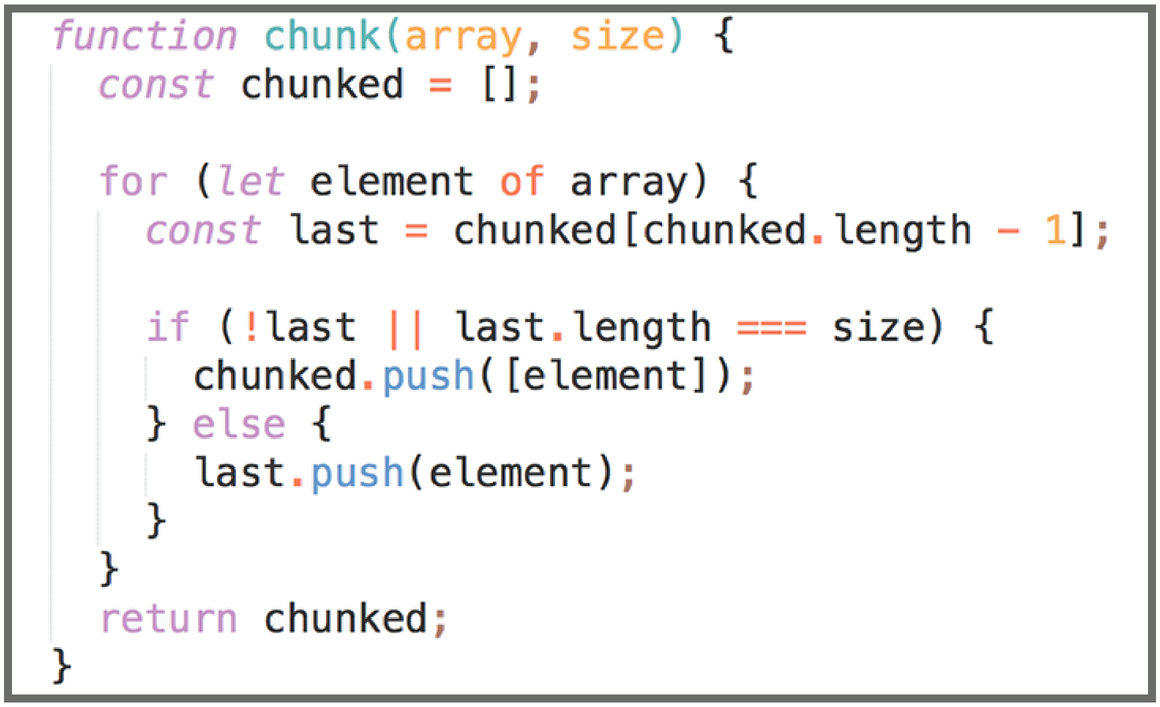
```
function reverseInt(n) {
  const reverse = n.toString().split('').reverse().join('');
  return parseInt(reverse) * Math.sign(n);
}
```



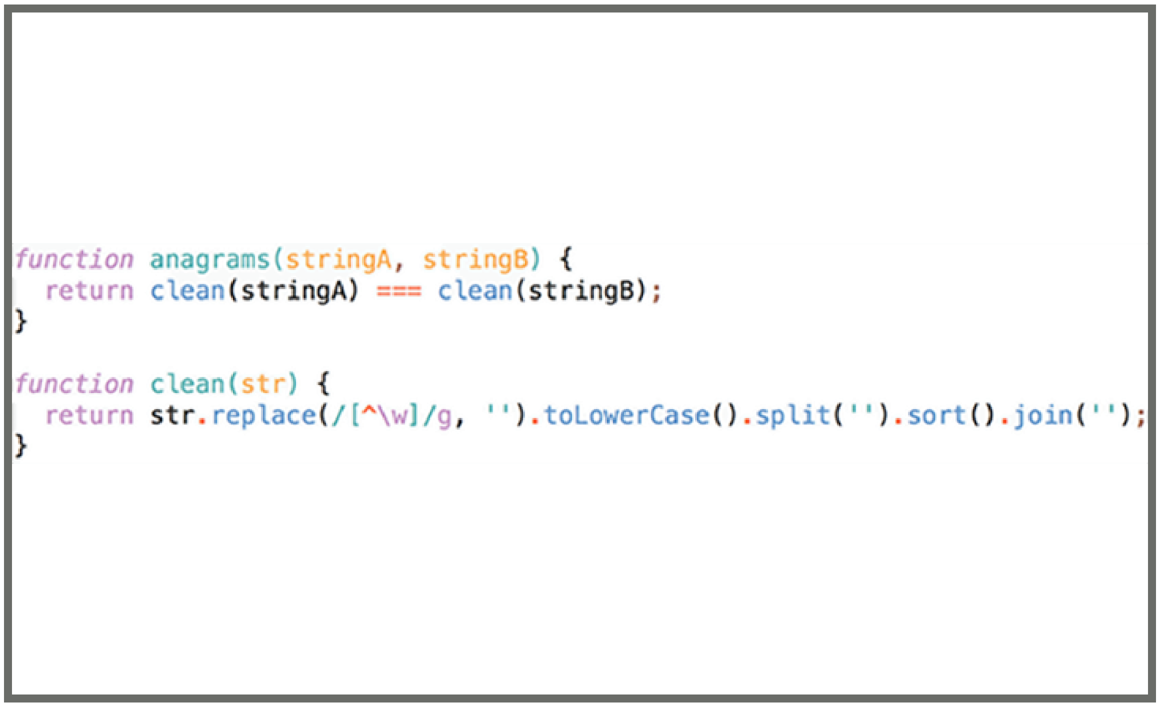
```
function maxChar(str) {
  const charMap = {};
  let max = 0;
  let maxChar = '';

  for (let char of str) {
    charMap[char] ? charMap[char]++ : charMap[char] = 1;
  }

  for (let char in charMap) {
    if (charMap[char] > max) {
      max = charMap[char];
      maxChar = char;
    }
  }
  return maxChar;
}
```



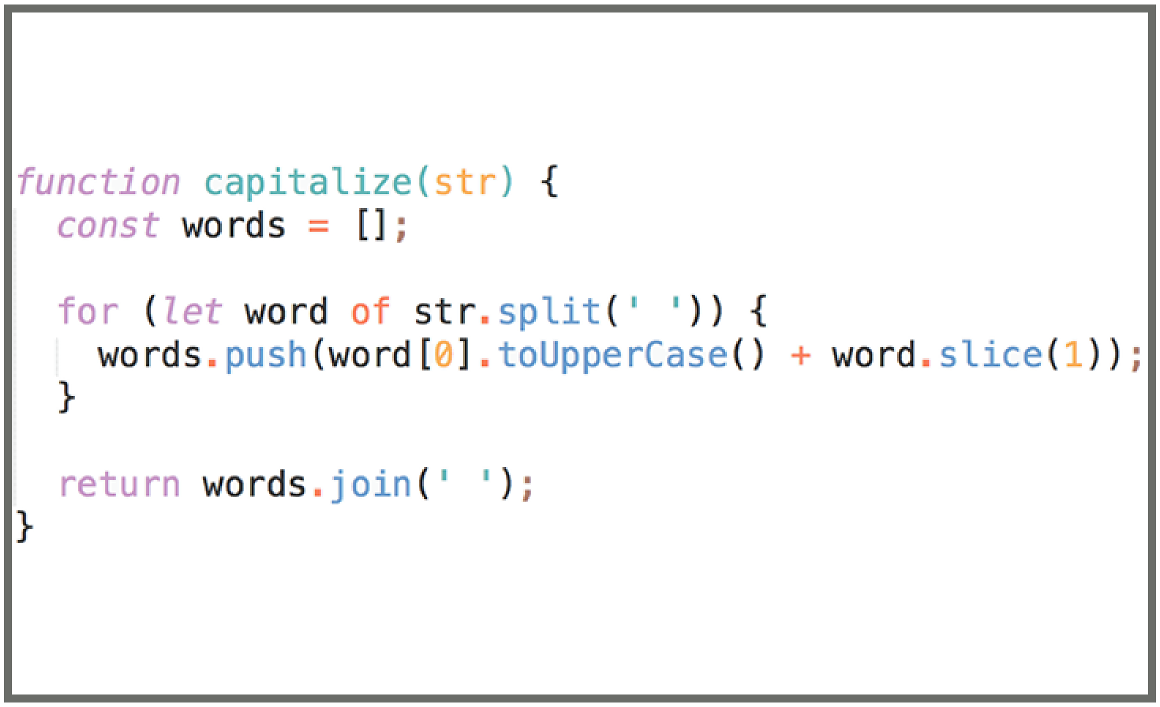
```
function fizzBuzz(n) {
  for (let i = 1; i <= n; i++) {

    if (i % 15 === 0) {
      console.log('fizzbuzz');
    } else if (i % 3 === 0) {
      console.log('fizz');
    } else if (i % 5 === 0) {
      console.log('buzz');
    } else {
      console.log(i);
    }
  }
}
```



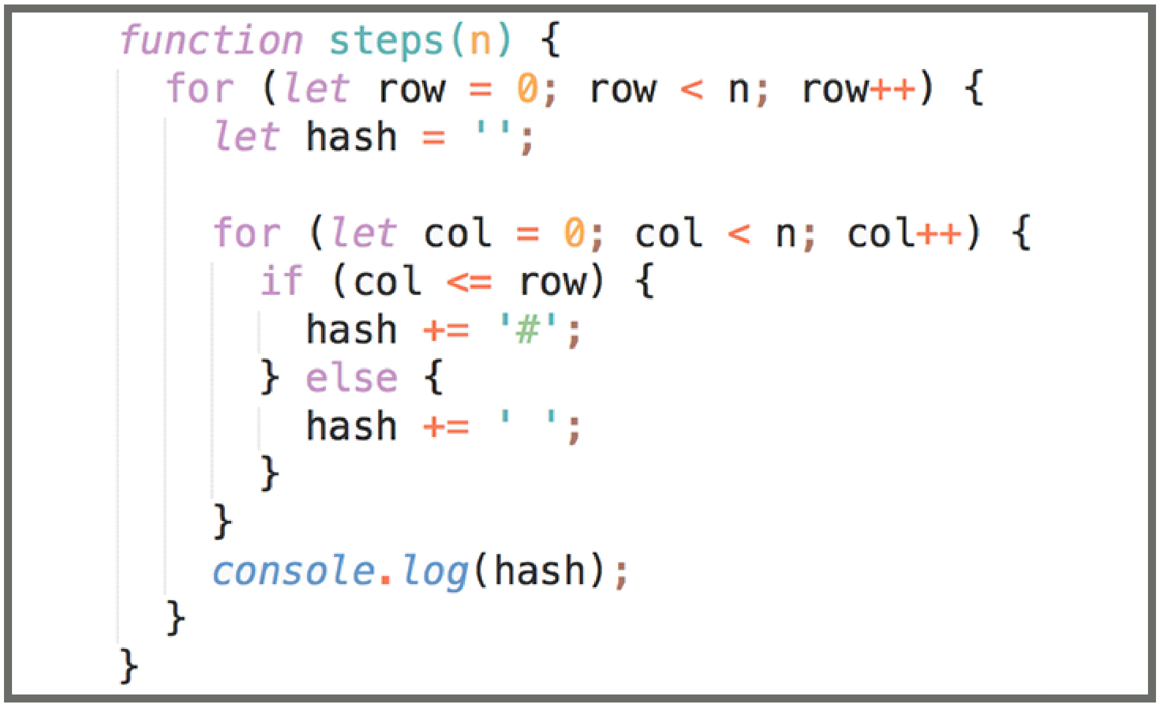
```
function chunk(array, size) {
  const chunked = [];

  for (let element of array) {
    const last = chunked[chunked.length - 1];

    if (!last || last.length === size) {
      chunked.push([element]);
    } else {
      last.push(element);
    }
  }
  return chunked;
}
```



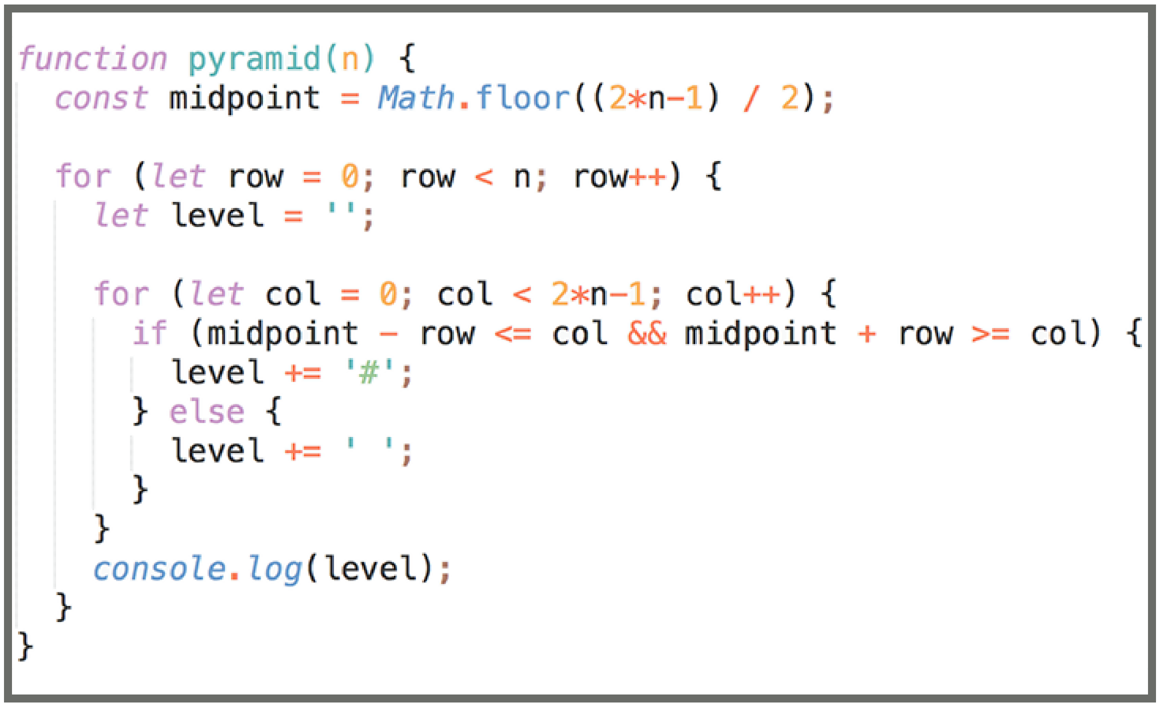
```
function anagrams(stringA, stringB) {
  return clean(stringA) === clean(stringB);
}

function clean(str) {
  return str.replace(/[^\w]/g, '').toLowerCase().split('').sort().join('');
}
```



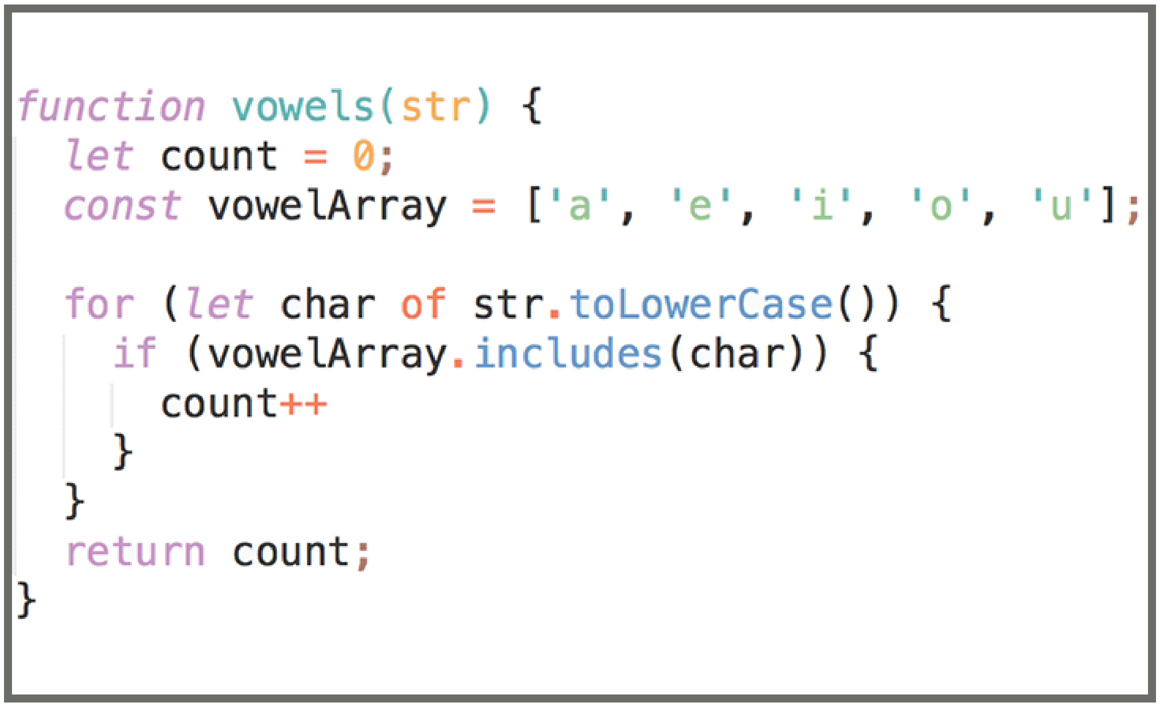
```
function capitalize(str) {
  const words = [];

  for (let word of str.split(' ')) {
    words.push(word[0].toUpperCase() + word.slice(1));
  }

  return words.join(' ');
}
```



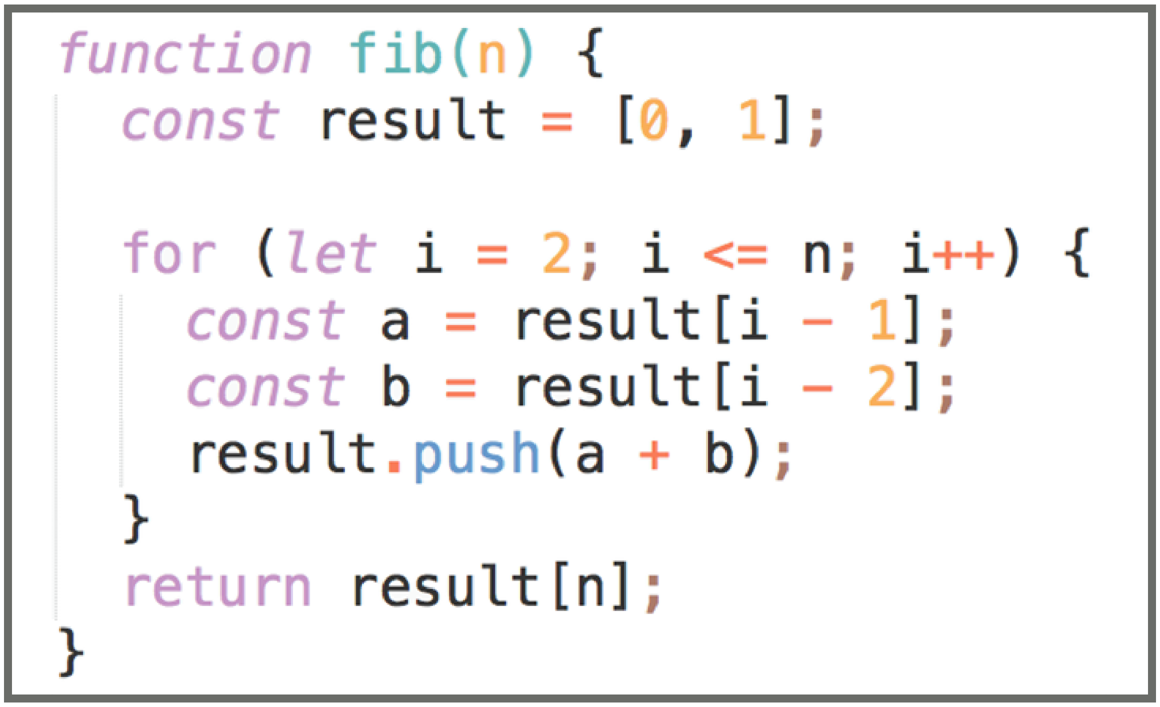
```
function steps(n) {
  for (let row = 0; row < n; row++) {
    let hash = '';

    for (let col = 0; col < n; col++) {
      if (col <= row) {
        hash += '#';
      } else {
        hash += ' ';
      }
    }
    console.log(hash);
  }
}
```



```
function pyramid(n) {
  const midpoint = Math.floor((2*n-1) / 2);

  for (let row = 0; row < n; row++) {
    let level = '';

    for (let col = 0; col < 2*n-1; col++) {
      if (midpoint - row <= col && midpoint + row >= col) {
        level += '#';
      } else {
        level += ' ';
      }
    }
    console.log(level);
  }
}
```



```javascript
function vowels(str) {
  let count = 0;
  const vowelArray = ['a', 'e', 'i', 'o', 'u'];

  for (let char of str.toLowerCase()) {
    if (vowelArray.includes(char)) {
      count++
    }
  }
  return count;
}
```



```javascript
function fib(n) {
  const result = [0, 1];

  for (let i = 2; i <= n; i++) {
    const a = result[i - 1];
    const b = result[i - 2];
    result.push(a + b);
  }
  return result[n];
}
```

Reverse String:

- Given a string, return a new string with the reversed order of characters
- *Examples*

```
//   reverse('apple') === 'leppa'
//   reverse('hello') === 'olleh'
//   reverse('Greetings!') === '!sgniteerG's
```

Palindrome:

- Given a string, return true if the string is a palindrome or false if it is not. Include spaces and punctuation in determining if the string is a palindrome.
- *Examples*

```
//   palindrome("abba") === true
//   palindrome("abcdefg") === false
```

Reverse Integer:

- Given an integer, return an integer that is the reverse ordering of numbers.
- *Examples*

```
//   reverseInt(981) === 189
//   reverseInt(500) === 5
//   reverseInt(-15) === -51
//   reverseInt(-90) === -9
```

Max Characters:

- Given a string, return the character that is most commonly used in the string.
- *Examples*

```
// maxChar("abccccccd") === "c"
// maxChar("apple 1231111") === "1"
```

FizzBuzz:

- Write a program that logs numbers from 1 to n.
  For multiples of three print “fizz” instead of the number
  and for the multiples of five print “buzz”. For numbers
  which are multiples of both three and five print “fizzbuzz”.
- *Example*

```
//   fizzBuzz(5);
//   1
//   2
//   fizz
//   4
//   buzz
```

Array Chunking:

- Given an array and chunk size, divide the array into many
  subarrays where each subarray is of length size
- *Examples*

```
// chunk([1, 2, 3, 4], 2) --> [[ 1, 2], [3, 4]]
// chunk([1, 2, 3, 4, 5], 2) --> [[ 1, 2], [3, 4], [5]]
// chunk([1, 2, 3, 4, 5, 6, 7, 8], 3) --> [[ 1, 2, 3], [4, 5, 6], [7, 8]]
// chunk([1, 2, 3, 4, 5], 4) --> [[ 1, 2, 3, 4], [5]]
// chunk([1, 2, 3, 4, 5], 10) --> [[ 1, 2, 3, 4, 5]]
```

Anagrams:

- Check to see if two provided strings are anagrams of
  each other. One string is an anagram of another if it uses
  the same characters in the same quantity. Only consider
  characters, not spaces or punctuation. Consider capital letters
  to be the same as lower case.
- *Examples*

```
//   anagrams('rail safety', 'fairy tales') --> True
//   anagrams('RAIL! SAFETY!', 'fairy tales') --> True
//   anagrams('Hi there', 'Bye there') --> False
```

Sentence Capitalization:

- Write a function that accepts a string. The function should capitalize the first letter of each word in the string then return the capitalized string.
- *Examples*

```
//   capitalize('a short sentence') --> 'A Short Sentence'
//   capitalize('a lazy fox') --> 'A Lazy Fox'
//   capitalize('look, it is working!') --> 'Look, It Is Working!'
```

Printing Steps:

- Write a function that accepts a positive number N. The function should console log a step shape with N levels using the # character. Make sure the step has spaces on the right hand side!

- Example

```
//   steps(3)
//       '#  '
//       '## '
//       '###'
```

Printing Pyramid:

- Write a function that accepts a positive number N. The function should console log a pyramid shape with N levels using the # character. Make sure the pyramid has spaces on both the left *and* right hand sides.

- Example

```
//   pyramid(3)
//       '  #  '
//       ' ### '
//       '#####'
```

Find the Vowels:

- Write a function that returns the number of vowels used in a String. Vowels are the characters 'a', 'e', 'i', 'o', and 'u'.
- *Examples*

```
//   vowels('Hi There!') --> 3
//   vowels('Why do you ask?') --> 4
//   vowels('Why?') --> 0
```

Fibonacci Series:

- Print out the n-th entry in the fibonacci series. The fibonacci series is an ordering of numbers where each number is the sum of the preceding two.
- *Example*

```
//  [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
// return 34
```