Given an array of numbers, return their product. Use `reduce` method

Example:

```
product([2, 4, 6]); // 48
product([-10, 10]); // -100
```

Write a function `anyPositive` which is given an array of numbers and returns true if any element is positive and false otherwise

Example:

```
anyPositive([1, 2, 3, 4, 5]); // true
anyPositive([1, 2, -3, 4, 5]); // true
anyPositive([0, 0, 1]); // true
anyPositive([-10, -10, -10]); // false
anyPositive([-10, -10, 1]); // true
```

Write a function `positives` which is given an array of numbers and returns a new array containing only the positive numbers within the given array. Use `filter` method

Examples:

```
positives([1, -3, 5, -3, 0]) // [1, 5]
positives([1, 2, 3]) // [1, 2, 3]
positives([-1, -2, -3]) // []
```

Write function `mean`, which takes an array of numbers and returns their mean. We use "mean" instead of average because "average" can mean many things — mean, median, or mode while mean only ever means one thing

The mean of three numbers a, b, c is (a + b + c) / 3. The mean of four numbers a, b, c, d is (a + b + c + d) / 4.

See https://en.wikipedia.org/wiki/Arithmetic_mean

Examples:

```
mean([30, 10, 20]); // 20)
mean([-10, 10]); // 0
```

Write a function `evens` which takes an array of numbers and returns a new array containing only the even numbers in the given array

Example:

```
evens([1, 2, 4, 3, 8]); // [2, 4, 8]
```

Write a function `odds` which takes an array of numbers and returns a new array containing only the odd numbers in the given array

Example:

```
odds([1, 2, 4, 3, 8]); // [1, 3]
```

Write a function `integers` which takes an array of numbers and returns a new array containing only the integers in the given array

Example:

```
integers([3.14, 2.4, 7, 8.1, 2]); // [7, 2]
```

Write a function `countEvens` which takes an array of integers and returns the count of even integers in the array

Example:

```
countEvens([1, 2, 3, 4, 5]); // 2
countEvens([10, 10, 10]); // 3
countEvens([1, 1, 1, 2]); // 2
```

Write function `countLessThan` which takes an array of numbers and a threshold number and returns the count of elements in the array strictly less than the threshold number

"Strictly less than" means we want numbers less than (<) and not less than or equal to (<=).

Example:

```
countLessThan([1, 2, 3, 4, 5], 2); // 1
countLessThan([1, 2, 3, 4, 5], 17); // 5
countLessThan([1, 2, 1, 2, 3, 4, 1, 2, 1], 1); // 0
countLessThan([10, 10, 10, -10, 15, 7], 10); // 2
```

Write a function `squareDance` which takes an array of numbers and returns a new array containing the result of squaring each of the numbers in the given array. Use `map` method

Example:

```
squareDance([1, 2, 3]) // [1, 4, 9]
```

Given two arrays, write a function `glueArrays` which returns a new array that is a concatenation of the two given arrays

Example:

```
glueArrays([1, 2, 3], [4, 5, 6]); // [1, 2, 3, 4, 5, 6]
glueArrays([-10, undefined], [true, 'waffles']); // [-10, undefined, true, 'waffles']
glueArrays([], []); // []
glueArrays([20, 104], []); // [20, 104]
glueArrays([], ['hello', 'world']); // ['hello', 'world']
```

Given an array and a value, write function `countValue` which returns the number of times that value is found in the array

Example:

```
countValue([1, 2, 3, 4, 5], 2); // 1
countValue([1, 2, 3, 4, 5], 17); // 0
countValue([1, 2, 1, 2, 3, 4, 1, 2, 1], 1); // 4
countValue([10, 10, 10, -10], 10); // 3
countValue(['hello', bananas', 'hello'], 'hello'); // 2
countValue(['hello', bananas', 'hello'], 'giraffe'); // 0
```

Given an array and a value, write function `findInHaystack` which returns true if the value is found in the array and false otherwise

The array doesn't need to contain a single type of data.

When searching an array for something, it's common to refer to the array as the "haystack" and the thing being searched for as the "needle", as in, "Looking for a needle in a haystack."

Example:

```
findInHaystack([1, 2, 30, -10], 480); // false
findInHaystack([1, 2, 30, -10], 30); // true
findInHaystack(['waffle', 'giraffe', 'banana'], 'giraffe'); // true
findInHaystack(['waffle', 'giraffe', 'banana'], 'lemons'); // false
```

Given an array and a value, write a function `firstIndexOf` which returns the index of the first occurence of the value. If the value is not found, returns -1

The array doesn't need to contain a single type of data.

Example:

```
firstIndexOf([10, 20, 30, 20], 20); // 1
firstIndexOf([10, 20, 30, 20], 17); // -1
firstIndexOf(['giraffe', giraffe', 'banana'], 'giraffe'); // 0
firstIndexOf(['giraffe', giraffe', 'banana'], 'banana'); // 2
```