Client-Side Scripting Exercises

For each exercise, create the function and implement it inside of exercises.js. The unit tests defined in tests.js will look for these functions and call them using the signature shown below.

Exercises

1. **SumDouble** Given two int values, return their sum. Unless the two values are the same, then return double their sum.

```
sumDouble(1, 2) → 3
sumDouble(3, 2) → 5
sumDouble(2, 2) → 8

function sumDouble(x, y) {
        // do logic here
        // return result;
        return x + y;
}
```

2. **HasTeen** We'll say that a number is "teen" if it is in the range 13..19 inclusive. Given 3 int values, return true if 1 or more of them are teen.

```
hasTeen(13, 20, 10) → true
hasTeen(20, 19, 10) → true
hasTeen(20, 10, 13) → true
```

3. **LastDigit** Given two non-negative int values, return true if they have the same last digit, such as with 27 and 57.

```
lastDigit(7, 17) → true
lastDigit(6, 17) → false
lastDigit(3, 113) → true
```

4. **SeeColor** Given a string, if the string begins with "red" or "blue" return that color string, otherwise return the empty string.

```
seeColor("redxx") → "red"
seeColor("xxred") → ""
seeColor("blueTimes") → "blue"
```

5. **MiddleThree** Given a string of odd length, return the string length 3 from its middle, so "Candy" yields "and". The string length will be at least 3.

```
middleThree("Candy") → "and"
middleThree("and") → "and"
middleThree("solving") → "lvi"
```

6. **FrontAgain** Given a string, return true if the first 2 chars in the string also appear at the end of the string, such as with "edited".

```
frontAgain("edited") → true
frontAgain("edit") → false
frontAgain("ed") → true
```

7. **AlarmClock** Write a function that given a day of the week encoded as 0=Sun, 1=Mon, 2=Tue, ...6=Sat, and a boolean indicating if we are on vacation, displays the weeday name, and the time in the form of "7:00" indicating when the alarm clock should ring. Weekdays, the alarm should be "7:00" and on the weekend it should be "10:00". Unless we are on vacation -- then on weekdays it should be "10:00" and weekends it should be "off".

```
alarmClock(1, false) → "Monday 7:00"
alarmClock(5, true) → "Frday 10:00"
alarmClock(0, false) → "Sunday 10:00"
```

8. **MakeMiddle** Write a function that given an array of ints of even length, returns a new array length 2 containing the middle two elements from the original array. If the original array length is not even, or at least 2 elements in length, return an empty array.

```
makeMiddle([1, 2, 3, 4]) → [2, 3]
makeMiddle([7, 1, 2, 3, 4, 9]) → [2, 3]
makeMiddle([1, 2]) → [1, 2]
```

9. **OddOnly** Write a function that given an array of integer of any length, filters out the even number, and returns a new array of just the the odd numbers.

```
oddOnly([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]) → [1, 3, 5, 7, 9, 11];
oddOnly([2, 4, 8, 32, 256]); → []
```

10. **Weave** Write a function that given two arrays, interleaves the two arrays one element from each array at a time to return a new array made up of the interwoven elements of the original two arrays.

```
weave([1, 3, 5], [2, 4]); → [1, 2, 3, 4, 5]
weave([1, 3, 5], [2, 4, 6, 8]); → [1, 2, 3, 4, 5, 6, 8]
```

11. **CigarParty** When squirrels get together for a party, they like to have cigars. A squirrel party is successful when the number of cigars is between 40 and 60, inclusive. Unless it is the weekend, in which case there is no upper bound on the number of cigars. Write a squirrel party function that return true if the party with the given values is successful, or false otherwise.

```
cigarParty(30, false) → false
cigarParty(50, false) → true
cigarParty(70, true) → true
```

12. **FizzBuzz** Because you know you can't live without it, FizzBuzz.

```
      fizzBuzz(3) → "Fizz"
      fizzBuzz(1) → 1
      fizzBuzz(10) → "Buzz"
      fizzBuzz(15) → "FizzBuzz"
      fizzBuzz(8) → 8
```

14. **CountValues** Write a function that accepts an array of integer values, count of the number of times each value is found in the array, and then display the values and their count.

```
countValues([1, 99, 43, 2, 55, 78, 99, 2345, 438, 2, 99, 107])  →
```

```
        1 : 1
        99 : 3
        43 : 1
        2 : 2
        55 : 1
        78 : 1
        2345 : 1
        438 : 1
        107 : 1
```

15. **Filter Evens** Write a function that filters an array to only include even numbers.

```
filterEvens([]) → []
filterEvens([1, 3, 5]) → []
filterEvens([2, 4, 6]) → [2, 4, 6]
filterEvens([100, 8, 21, 24, 62, 9, 7]) → [100, 8, 24, 62]
```

16. **Filter Numbers Greater than 100** Write a function that filters numbers greater than 100.

```
filterBigNumbers([7, 10, 121, 100, 24, 162, 200]) → [121, 100, 162, 200]
filterBigNumbers([3, 2, 7, 1, -100, -120]) → []
filterBigNumbers([]) → []
```

17. **Filter Numbers that are Multiples of X** Write a function to filter numbers that are a multiple of a paremeter, x passed in.

```
filterMultiplesOfX([3, 5, 1, 9, 18, 21, 42, 67], 3) → [3, 9, 18, 21, 42]
filterMultiplesOfX([3, 5, 10, 20, 18, 21, 42, 67], 5) → [5, 10, 20]
```

18. **Create Object** Write a function that creates an object with a property called firstName, lastName, and age. Populate the properties with your values.

```
createObject() →

{
        firstName,
        lastName,
        age
}
```

19. **Filter Inventors** Given an array of inventors, filter the list to include any inventors whose first or last name starts with a vowel.

```
Sample Inventors
[{ first: 'Albert', last: 'Einstein', year: 1879, passed: 1955 },
{ first: 'Isaac', last: 'Newton', year: 1643, passed: 1727 },
{ first: 'Galileo', last: 'Galilei', year: 1564, passed: 1642 }, ...]
```

```
filterInventors([...]) →

[{ first: 'Albert', last: 'Einstein', year: 1879, passed: 1955 },
 { first: 'Isaac', last: 'Newton', year: 1643, passed: 1727 }]
```

Challenge Exercises

1. **IQTest** Bob is preparing to pass an IQ test. The most frequent task in this test is to find out which one of the given numbers differs from the others. Bob observed that one number usually differs from the others in evenness. Help Bob — to check his answers, he needs a program that among the given numbers finds one that is different in evenness, and return the position of this number. *Keep in mind that your task is to help Bob solve a real IQ test, which means indexes of the elements start from 1 (not 0)*

```
        iqTest("2 4 7 8 10") → 3 //third number is odd, while the rest are even
        iqTest("1 2 1 1") → 2 // second number is even, while the rest are odd
        iqTest("") → 0 // there are no numbers in the given set
        iqTest("2 2 4 6") → 0 // all numbers are even, therefore there is no position of an odd
number
```

2. **TitleCase** Write a function that will convert a string into title case, given an optional list of exceptions (minor words). The list of minor words will be given as a string with each word separated by a space. Your function should ignore the case of the minor words string -- it should behave in the same way even if the case of the minor word string is changed.

- First argument (required): the original string to be converted.
- Second argument (optional): space-delimited list of minor words that must always be lowercase except for the first word in the string. The JavaScript tests will pass undefined when this argument is unused.

```
        titleCase('a clash of KINGS', 'a an the of') // should return: 'A Clash of Kings'
        titleCase('THE WIND IN THE WILLOWS', 'The In') // should return: 'The Wind in the Willows'
        titleCase('the quick brown fox') // should return: 'The Quick Brown Fox'
```

3. **PerfectSquare** Complete the findNextSquare method that finds the next integral perfect square after the one passed as a parameter. Recall that an integral perfect square is an integer n such that sqrt(n) is also an integer. If the parameter is itself not a perfect square, than -1 should be returned. You may assume the parameter is positive.

```
        findNextSquare(121) → 144
        findNextSquare(625) → 676
        findNextSquare(114) → -1 // 114 is not a perfect square
```