Introduction to the Intermediate Algorithm Scripting Challenges

This is a stub introduction

Table of Contents

Sum All Numbers in a Range

We'll pass you an array of two numbers. Return the sum of those two numbers plus the sum of all the numbers between them.

The lowest number will not always come first.

```
function sumAll(arr) {
  var sortedArr = arr.sort((a,b) => a-b);
  var firstNum = arr[0];
  var lastNum = arr[1];
```

```
  // Using Arithmetic Progression summing formula

  var sum = (lastNum - firstNum + 1) * (firstNum + lastNum) / 2;
  return sum;
}

sumAll([1, 4]);
```

or :

```
function sumAll(arr) {
  var sortedArr = arr.sort((a,b) => a-b);
  var firstNum = arr[0];
  var lastNum = arr[1];
  // Using Arithmetic Progression summing formula
  let temp=0;
  for (let i= firstNum; i<=lastNum;i++){

    temp=i+temp;
  }

  return temp;
}
```

Diff Two Arrays

Compare two arrays and return a new array with any items only found in one of the two given arrays, but not both. In other words, return the symmetric difference of the two arrays.

Remember to use Read-Search-Ask if you get stuck. Try to pair program. Write your own code.

Note
You can return the array with its elements in any order.

```
function diffArray(arr1, arr2) {
  var newArr = [];
  // Same, same; but different.
    for (let i=0; i<arr1.length; i++){
    if (arr2.includes(arr1[i])==false) {
      newArr.push(arr1[i]);
      }
    }
    for (let j=0; j<arr2.length; j++){
    if (arr1.includes(arr2[j])==false) {
      newArr.push(arr2[j]);
```

```
      }
     }
   return newArr;
}

diffArray([1, 2, 3, 5], [1, 2, 3, 4, 5]);
```

or :

```
   function diffArray(arr1, arr2) {
     return arr1
       .concat(arr2)
       .filter(
           item => !arr1.includes(item) || !arr2.includes(item)
       )
   }
```

Seek and Destroy

You will be provided with an initial array (the first argument in the destroyer function), followed by one or more arguments. Remove all elements from the initial array that are of the same value as these arguments.

Note

You have to use the `arguments` object.

```
function destroyer(arr) {
  // Remove all the values
  var args = Array.prototype.slice.call(arguments);

  for (var i = 0; i < arr.length; i++) {
    for (var j = 0; j < args.length; j++) {
      if (arr[i] === args[j]) {
        delete arr[i];
      }
    }
  }
  return arr.filter(Boolean);
}

destroyer([1, 2, 3, 1, 2, 3], 2, 3);
```

or:

```
function destroyer(arr) {
  var args = Array.from(arguments).slice(1);
```

```
  return arr.filter(function(val) {
    return !args.includes(val);
  });
}
```

Wherefore art thou

Make a function that looks through an array of objects (first argument) and returns an array of all objects that have matching name and value pairs (second argument). Each name and value pair of the source object has to be present in the object from the collection if it is to be included in the returned array.

For example, if the first argument is `[{ first: "Romeo", last: "Montague" }, { first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet" }]`, and the second argument is `{ last: "Capulet" }`, then you must return the third object from the array (the first argument), because it contains the name and its value, that was passed on as the second argument.

```
function whatIsInAName(collection, source) {
  // What's in a name?
  var arr = [];
  // Only change code below this line
    var srcKeys = Object.keys(source);

  // filter the collection
  return collection.filter(function (obj) {
    for(var i = 0; i < srcKeys.length; i++) {
      if(!obj.hasOwnProperty(srcKeys[i]) || obj[srcKeys[i]] !== source[srcKeys[i]]) {
        return false;
      }
    }
    return true;
  });

  // Only change code above this line
  return arr;
}

whatIsInAName([{ first: "Romeo", last: "Montague" }, { first: "Mercutio", last: null }, { first: "Tybalt", last: "Capulet" }], { last: "Capulet" });


or:
function whatIsInAName(collection, source) {
  // "What's in a name? that which we call a rose
```

```
  // By any other name would smell as sweet.”
  // -- by William Shakespeare, Romeo and Juliet
  var srcKeys = Object.keys(source);

  return collection.filter(function (obj) {
    return srcKeys.every(function (key) {
      return obj.hasOwnProperty(key) && obj[key] === source[key];
    });
  });
}
```

or:

```
function whatIsInAName(collection, source) {
  // "What's in a name? that which we call a rose
  // By any other name would smell as sweet.”
  // -- by William Shakespeare, Romeo and Juliet
  var srcKeys = Object.keys(source);

  // filter the collection
  return collection.filter(function (obj) {
    return srcKeys
      .map(function(key) {
        return obj.hasOwnProperty(key) && obj[key] === source[key];
      })
      .reduce(function(a, b) {
        return a && b;
      });
  });
}
```

Spinal Tap Case

Convert a string to spinal case. Spinal case is all-lowercase-words-joined-by-dashes.

```
function spinalCase(str) {
  // "It's such a fine line between stupid, and clever."
  // --David St. Hubbins
  // Create a variable for the white space and underscores.
  var regex = /\s+|_+/g;

  // Replace low-upper case to low-space-uppercase
  str = str.replace(/([a-z])([A-Z])/g, '$1 $2');

  // Replace space and underscore with -
```

```
  return str.replace(regex, '-').toLowerCase();
}

spinalCase('This Is Spinal Tap');
```

or:

```
function spinalCase(str) {
  // Replace low-upper case to low-space-uppercase
  str = str.replace(/([a-z])([A-Z])/g, '$1 $2');
  // Split on whitespace and underscores and join with dash
  return str.toLowerCase().split(/(?:_| )+/) .join('-');
}
```

or:

```
function spinalCase(str) {
  // "It's such a fine line between stupid, and clever."
  // --David St. Hubbins

  return str.split(/\s|_|(?=[A-Z])/).join('-').toLowerCase()
}
```

Pig Latin

Translate the provided string to pig latin.

Pig Latin takes the first consonant (or consonant cluster) of an English word, moves it to the end of the word and suffixes an "ay".

If a word begins with a vowel you just add "way" to the end.

Input strings are guaranteed to be English words in all lowercase.

```
function translatePigLatin(str) {
  // Create variables to be used
  var pigLatin = '';
  var regex = /[aeiou]/gi;

  // Check if the first character is a vowel
  if (str[0].match(regex)) {
    pigLatin = str + 'way';

  } else if(str.match(regex) === null) {
```

```
    // Check if the string contains only consonants
    pigLatin = str + 'ay';
  } else {

    // Find how many consonants before the first vowel.
    var vowelIndice = str.indexOf(str.match(regex)[0]);

    // Take the string from the first vowel to the last char
    // then add the consonants that were previously omitted and add the ending.
    pigLatin = str.substr(vowelIndice) + str.substr(0, vowelIndice) + 'ay';
  }

  return pigLatin;
}

translatePigLatin("consonant");


or:
function translatePigLatin(str) {
  function check(obj) {
      return ['a','i','u','e','o'].indexOf(str.charAt(obj)) == -1 ? check(obj + 1) : obj;
  }

  return str.substr(check(0)).concat((check(0) === 0 ? 'w' : str.substr(0, check(0))) + 'ay');
}

or:
function translatePigLatin(str) {
    var strArr = [];
    var tmpChar;

    // check if the char is consonant using RegEx
    function isConsonant(char) {
        return !/[aeiou]/.test(char);
    }

    // return initial str + "way" if it starts with vowel
    // if not - convert str to array
    if (!isConsonant(str.charAt(0)))
        return str + "way";
    else
        strArr = str.split("");

    // push all consonats to the end of the array
    while (isConsonant(strArr[0])) {
        tmpChar = strArr.shift();
```

```
      strArr.push(tmpChar);
    }
 // convert array to string and concatenate "ay" at the end
 return strArr.join("")+"ay";
}
```

Search and Replace

Perform a search and replace on the sentence using the arguments provided and return the new sentence.

First argument is the sentence to perform the search and replace on.

Second argument is the word that you will be replacing (before).

Third argument is what you will be replacing the second argument with (after).

Note
Preserve the case of the first character in the original word when you are replacing it. For example if you mean to replace the word "Book" with the word "dog", it should be replaced as "Dog"

```
function myReplace(str, before, after) {
  // Find index where before is on string
  var index = str.indexOf(before);
  // Check to see if the first letter is uppercase or not
  if (str[index] === str[index].toUpperCase()) {
    // Change the after word to be capitalized before we use it.
    after = after.charAt(0).toUpperCase() + after.slice(1);
  }
  // Now replace the original str with the edited one.
  str = str.replace(before, after);

  return str;
}

// test here
myReplace("A quick brown fox jumped over the lazy dog", "jumped", "leaped");
```

or:

```
function myReplace(str, before, after) {
//Create a regular expression object
  var re = new RegExp(before,"gi");
//Check whether the first letter is uppercase or not
  if(/[A-Z]/.test(before[0])){
```

```
  //Change the word to be capitalized
    after = after.charAt(0).toUpperCase()+after.slice(1);
     }
     //Replace the original word with new one
  var  newStr =  str.replace(re,after);

 return newStr;
}
```

or:

```
function myReplace(str, before, after) {

    // create a function that will change the casing of any number of letter in parameter "target"
    // matching parameter "source"
    function applyCasing(source, target) {
        // split the source and target strings to array of letters
        var targetArr = target.split("");
        var sourceArr = source.split("");
        // iterate through all the items of sourceArr and targetArr arrays till loop hits the end of shortest
array
        for (var i = 0; i < Math.min(targetArr.length, sourceArr.length); i++){
            // find out the casing of every letter from sourceArr using regular expression
            // if sourceArr[i] is upper case then convert targetArr[i] to upper case
            if (/[A-Z]/.test(sourceArr[i])) {
                targetArr[i] = targetArr[i].toUpperCase();
            }
            // if sourceArr[i] is not upper case then convert targetArr[i] to lower case
            else targetArr[i] = targetArr[i].toLowerCase();
        }
        // join modified targetArr to string and return
        return (targetArr.join(""));
    }

    // replace "before" with "after" with "before"-casing
    return str.replace(before, applyCasing(before, after));
}
```

DNA Pairing

The DNA strand is missing the pairing element. Take each character, get its pair, and return the results as a 2d array.

Base pairs are a pair of AT and CG. Match the missing element to the provided character.

Return the provided character as the first element in each array.

For example, for the input GCG, return [["G", "C"], ["C","G"],["G", "C"]]

The character and its pair are paired up in an array, and all the arrays are grouped into one encapsulating array.

```
function pairElement(str) {
  // Return each strand as an array of two elements, the original and the pair.
    var paired = [];

    // Function to check with strand to pair.
    var search = function(char) {
      switch (char) {
        case 'A':
          paired.push(['A', 'T']);
          break;
        case 'T':
          paired.push(['T', 'A']);
          break;
        case 'C':
          paired.push(['C', 'G']);
          break;
        case 'G':
          paired.push(['G', 'C']);
          break;
      }
    };

    // Loops through the input and pair.
    for (var i = 0; i < str.length; i++) {
      search(str[i]);
    }

    return paired;
}

pairElement("GCG");
```

or:

```
function pairElement(str) {
   //create object for pair lookup
   var pairs = {
     "A": "T",
     "T": "A",
     "C": "G",
     "G": "C"
```

```
  }
  //split string into array of characters
  var arr = str.split("");
  //map character to array of character and matching pair
  return arr.map(x => [x,pairs[x]]);
}

//test here
pairElement("GCG");
```

Missing letters

Find the missing letter in the passed letter range and return it.

If all letters are present in the range, return undefined.

```
function fearNotLetter(str) {
  for(var i = 0; i < str.length; i++) {
   /* code of current character */
   var code = str.charCodeAt(i);

   /* if code of current character is not equal to first character + no of iteration
   hence character has been escaped */
   if (code !== str.charCodeAt(0) + i) {

     /* if current character has escaped one character find previous char and return */
     return String.fromCharCode(code - 1);
   }
 }
 return undefined;
}

fearNotLetter("abce");
```

or:

```
function fearNotLetter(str) {
 var compare = str.charCodeAt(0), missing;

 str.split('').map(function(letter,index) {
   if (str.charCodeAt(index) == compare) {
     ++compare;
   } else {
     missing = String.fromCharCode(compare);
```

```
    }
  });

  return missing;
}
```

or:

```
function fearNotLetter(str) {
  for (let i = 1; i < str.length; ++i) {
    if (str.charCodeAt(i) - str.charCodeAt(i-1) > 1) {
      return String.fromCharCode(str.charCodeAt(i - 1) + 1);
    }
  }
}
```

Sorted Union

Write a function that takes two or more arrays and returns a new array of unique values in the order of the original provided arrays.

In other words, all values present from all arrays should be included in their original order, but with no duplicates in the final array.

The unique numbers should be sorted by their original order, but the final array should not be sorted in numerical order.

Check the assertion tests for examples.

```
function uniteUnique(arr) {
  // Creates an empty array to store our final result.
  var finalArray = [];

  // Loop through the arguments object to truly made the program work with two or more arrays
  // instead of 3.
  for (var i = 0; i < arguments.length; i++) {
    var arrayArguments = arguments[i];

    // Loops through the array at hand
    for (var j = 0; j < arrayArguments.length; j++) {
      var indexValue = arrayArguments[j];

      // Checks if the value is already on the final array.
      if (finalArray.indexOf(indexValue) < 0) {
        finalArray.push(indexValue);
      }
    }
```

```
  }

  return finalArray;
}

uniteUnique([1, 3, 2], [5, 2, 1, 4], [2, 1]);
```

or:

```
function uniteUnique(arr) {
  var args = [...arguments];
  var result = [];
  for(var i = 0; i < args.length; i++) {
    for(var j = 0; j < args[i].length; j++) {
      if(!result.includes(args[i][j])) {
        result.push(args[i][j]);
      }
    }
  }
  return result;
}

function uniteUnique(arr1, arr2, arr3) {
 var newArr;
 //Convert the arguments object into an array
  var args = Array.prototype.slice.call(arguments);
  //Use reduce function to flatten the array
  newArr = args.reduce(function(arrA,arrB){
  //Apply filter to remove the duplicate elements in the array
    return arrA.concat(arrB.filter(function(i){
      return arrA.indexOf(i) === -1;
    }));
  });

   return newArr;
}
```

Convert HTML Entities

Convert the characters &, <, >, " (double quote), and ' (apostrophe), in a string to their corresponding HTML entities.

```
function convertHTML(str) {
// Split by character to avoid problems.
```

```
    var temp = str.split('');

    // Since we are only checking for a few HTML elements I used a switch

    for (var i = 0; i < temp.length; i++) {
      switch (temp[i]) {
        case '<':
          temp[i] = '&lt;';
          break;
        case '&':
          temp[i] = '&amp;';
          break;
        case '>':
          temp[i] = '&gt;';
          break;
        case '"':
          temp[i] = '&quot;';
          break;
        case "'":
          temp[i] = "&apos;";
          break;
      }
    }

    temp = temp.join('');
    return temp;
}

convertHTML("Dolce & Gabbana");


or:
function convertHTML(str) {
//Chaining of replace method with different arguments
  str = str.replace(/&/g,'&amp;').replace(/</g,'&lt;').replace(/>/g,'&gt;').replace(/"/g,'&quot;').replace(/'/
g,"&apos;");
return str;
}

or:
function convertHTML(str) {
    // Use Object Lookup to declare as many HTML entities as needed.
    htmlEntities={
      '&':'&amp;',
      '<':'&lt;',
      '>':'&gt;',
      '"':'&quot;',
```

```
    '\":"&apos;"
  };
  //Use map function to return a filtered str with all entities changed automatically.
  return str.split('').map(entity => htmlEntities[entity] || entity).join('');
}
```

Sum All Odd Fibonacci Numbers

Given a positive integer `num`, return the sum of all odd Fibonacci numbers that are less than or equal to `num`.

The first two numbers in the Fibonacci sequence are 1 and 1. Every additional number in the sequence is the sum of the two previous numbers. The first six numbers of the Fibonacci sequence are 1, 1, 2, 3, 5 and 8.

For example, `sumFibs(10)` should return `10` because all odd Fibonacci numbers less than or equal to `10` are 1, 1, 3, and 5.

```
function sumFibs(num) {
  var prevNumber = 0;
  var currNumber = 1;
  var result = 0;
  while (currNumber <= num) {
    if (currNumber % 2 !== 0) {
      result += currNumber;
    }

    currNumber += prevNumber;
    prevNumber = currNumber - prevNumber;
  }

  return result;
}

sumFibs(4);
```

or:

```
function sumFibs(num) {
  // Perform checks for the validity of the input
  if (num < 0) return -1;
  if (num === 0 || num === 1) return 1;

  // Create an array of fib numbers till num
  const arrFib = [1, 1];
  let nextFib = 0;
```

```
  // We put the new Fibonacci numbers to the front so we
  // don't need to calculate the length of the array on each
  // iteration
  while((nextFib = arrFib[0] + arrFib[1]) <= num) {
    arrFib.unshift(nextFib);
  }

  // Sum only the odd numbers and return the value
  return arrFib.reduce((acc, curr) => {
    return acc + curr * (curr % 2);
  });
}
```

Sum All Primes

Sum all the prime numbers up to and including the provided number.

A prime number is defined as a number greater than one and having only two divisors, one and itself. For example, 2 is a prime number because it's only divisible by one and two.

The provided number may not be a prime.

```
function sumPrimes(num) {
 var res = 0;

 // Function to get the primes up to max in an array
 function getPrimes(max) {
  var sieve = [];
  var i;
  var j;
  var primes = [];
  for (i = 2; i <= max; ++i) {
   if (!sieve[i]) {
    // i has not been marked -- it is prime
    primes.push(i);
    for (j = i << 1; j <= max; j += i) {
     sieve[j] = true;
    }
   }
  }

  return primes;
 }

 // Add the primes
```

```
  var primes = getPrimes(num);
  for (var p = 0; p < primes.length; p++) {
    res += primes[p];
  }

  return res;
}

sumPrimes(10);

or:
function sumPrimes(num) {
  // function to check if the number presented is prime
  function isPrime(number){
      for (i = 2; i <= number; i++){
          if(number % i === 0 && number!= i){
          // return true if it is divisible by any number that is not itself.
             return false;
          }
       }
       // if it passes the for loops conditions it is a prime
      return true;
  }
  // 1 is not a prime, so return nothing, also stops the recursive calls.
  if (num === 1){
    return 0;
  }
  // Check if your number is not prime
  if(isPrime(num) === false){
  // for non primes check the next number down from your maximum number, do not add anything to
your answer
    return sumPrimes(num - 1);
  }

  // Check if your number is prime
  if(isPrime(num) === true){
  // for primes add that number to the next number in the sequence through a recursive call to our
sumPrimes function.
    return num + sumPrimes(num - 1);
  }
}
```

Smallest Common Multiple

Find the smallest common multiple of the provided parameters that can be evenly divided by both, as well as by all sequential numbers in the range between these parameters.

The range will be an array of two numbers that will not necessarily be in numerical order.

For example, if given 1 and 3, find the smallest common multiple of both 1 and 3 that is also evenly divisible by all numbers *between* 1 and 3. The answer here would be 6.

```
function smallestCommons(arr) {
  // Sort array from greater to lowest
  // This line of code was from Adam Doyle (http://github.com/Adoyle2014)
  arr.sort(function(a, b) {
    return b - a;
  });

  // Create new array and add all values from greater to smaller from the
  // original array.
  var newArr = [];
  for (var i = arr[0]; i >= arr[1]; i--) {
    newArr.push(i);
  }

  // Variables needed declared outside the loops.
  var quot = 0;
  var loop = 1;
  var n;

  // Run code while n is not the same as the array length.
  do {
    quot = newArr[0] * loop * newArr[1];
    for (n = 2; n < newArr.length; n++) {
      if (quot % newArr[n] !== 0) {
        break;
      }
    }

    loop++;
  } while (n !== newArr.length);

  return quot;
}


smallestCommons([1,5]);
```

or:
```
function smallestCommons(arr) {
   var range = [];
   for (var i = Math.max(arr[0], arr[1]); i >= Math.min(arr[0], arr[1]); i--) {
   range.push(i);
   }

   // can use reduce() in place of this block
   var lcm = range[0];
   for (i = 1; i < range.length; i++) {
   var GCD = gcd(lcm, range[i]);
   lcm = (lcm * range[i]) / GCD;
   }
   return lcm;

   function gcd(x, y) {    // Implements the Euclidean Algorithm
   if (y === 0)
      return x;
   else
      return gcd(y, x%y);
   }
}
```

or:
```
function smallestCommons(arr) {

  // range
  let min = Math.min.apply(null, arr);
  let max = Math.max.apply(null, arr);

  let smallestCommon = lcm(min, min + 1);

  while(min < max) {
    min++;
    smallestCommon = lcm(smallestCommon, min);
  }

  return smallestCommon;
}

/**
 * Calculates Greatest Common Divisor
 * of two nubers using Euclidean algorithm
 * https://en.wikipedia.org/wiki/Euclidean_algorithm
 */
function gcd(a, b) {
```

```
  while (b > 0) {
    let tmp = a;
    a = b;
    b = tmp % b;
  }
  return a;
}

/**
 * Calculates Least Common Multiple
 * for two numbers utilising GCD
 */
function lcm(a, b) {
  return (a * b / gcd(a, b));
}
```

Drop it

Given the array `arr`, iterate through and remove each element starting from the first element (the 0 index) until the function `func` returns `true` when the iterated element is passed through it.

Then return the rest of the array once the condition is satisfied, otherwise, `arr` should be returned as an empty array.

```
function dropElements(arr, func) {
  // Drop them elements.
  var times = arr.length;
  for (var i = 0; i < times; i++) {
    if (func(arr[0])) {
      break;
    } else {
      arr.shift();
    }
  }
  return arr;
}

dropElements([1, 2, 3], function(n) {return n < 3; });


or:
function dropElements(arr, func) {
  return arr.slice(arr.findIndex(func) >= 0 ? arr.findIndex(func): arr.length, arr.length);
}

or:
```

```
function dropElements(arr, func) {
  while(arr.length > 0 && !func(arr[0])) {
    arr.shift();
  }
  return arr;
}
```

Steamroller

Flatten a nested array. You must account for varying levels of nesting.

```
function steamrollArray(arr) {
  // I'm a steamroller, baby
  var flattenedArray = [];

  // Create function that adds an element if it is not an array.
  // If it is an array, then loops through it and uses recursion on that array.
  var flatten = function(arg) {
    if (!Array.isArray(arg)) {
      flattenedArray.push(arg);
    } else {
      for (var a in arg) {
        flatten(arg[a]);
      }
    }
  };

  // Call the function for each element in the array
  arr.forEach(flatten);
  return flattenedArray;
}

steamrollArray([1, [2], [3, [[4]]]]);
```

or:

```
function steamrollArray(arr) {
  let flat = [].concat(...arr);
  return flat.some(Array.isArray) ? steamrollArray(flat) : flat;
}
```

or:

```
function steamrollArray(arr) {
  return arr.toString()
    .replace(',,', ',')       // "1,2,,3" => "1,2,3"
```

```
    .split(',')             // ['1','2','3']
    .map(function(v) {
      if (v == '[object Object]') { // bring back empty objects
        return {};
      } else if (isNaN(v)) {        // if not a number (string)
        return v;
      } else {
        return parseInt(v);         // if a number in a string, convert it
      }
    });
}
```

Binary Agents

Return an English translated sentence of the passed binary string.

The binary string will be space separated.

```
function binaryAgent(str) {
  let biString = str.split(' ');
   let uniString = [];

   /*using the radix (or base) parameter in parseInt, we can convert the binary
     number to a decimal number while simultaneously converting to a char*/

     for(let i=0;i < biString.length;i++){
       uniString.push(String.fromCharCode(parseInt(biString[i], 2)));
     }

     // we then simply join the string
     return uniString.join('');
}

binaryAgent("01000001 01110010 01100101 01101110 00100111 01110100 00100000 01100010
01101111 01101110 01100110 01101001 01110010 01100101 01110011 00100000 01100110 01110101
01101110 00100001 00111111");
```

or:

```
function binaryAgent(str) {
     // Separate the binary code by space.
     str = str.split(' ');
     var power;
     var decValue = 0;
     var sentence = '';

     // Check each binary number from the array.
```

```
    for (var s = 0; s < str.length; s++) {
      // Check each bit from binary number
      for (var t = 0; t < str[s].length; t++) {
        // This only takes into consideration the active ones.
        if (str[s][t] == 1) {
          // This is quivalent to 2 ** position
          power = Math.pow(2, +str[s].length - t - 1);
          decValue += power;

          // Record the decimal value by adding the number to the previous one.
        }
      }

      // After the binary number is converted to decimal, convert it to string and store
      sentence += (String.fromCharCode(decValue));

      // Reset decimal value for next binary number.
      decValue = 0;
    }

    return sentence;
  }
```

or:

```
function binaryAgent(str) {
    return String.fromCharCode(...str.split(" ").map(function(char){ return parseInt(char, 2); }));
  }
```

Everything Be True

Check if the predicate (second argument) is truthy on all elements of a collection (first argument).

In other words, you are given an array collection of objects. The predicate `pre` will be an object property and you need to return `true` if its value is `truthy`. Otherwise, return `false`.

In JavaScript, `truthy` values are values that translate to `true` when evaluated in a Boolean context.

Remember, you can access object properties through either dot notation or `[]` notation.

```
function truthCheck(collection, pre) {
  // Is everyone being true?
  // Create a counter to check how many are true.
  var counter = 0;
  // Check for each object
  for (var c in collection) {
    // If it is has property and value is truthy
```

```
    if (collection[c].hasOwnProperty(pre) && Boolean(collection[c][pre])) {
      counter++;
    }
  }
  // Outside the loop, check to see if we got true for all of them and return true or false
  return counter == collection.length;
}
```

truthCheck([{"user": "Tinky-Winky", "sex": "male"}, {"user": "Dipsy", "sex": "male"}, {"user": "Laa-Laa", "sex": "female"}, {"user": "Po", "sex": "female"}], "sex");

or:

```
function truthCheck(collection, pre) {
  return collection.every(function (element) {
    return element.hasOwnProperty(pre) && Boolean(element[pre]);
  });
}
```

or:

```
function truthCheck(collection, pre) {
  // Is everyone being true?
  return collection.every(obj => obj[pre]);
}
```

Arguments Optional

Create a function that sums two arguments together. If only one argument is provided, then return a function that expects one argument and returns the sum.

For example, `addTogether(2, 3)` should return `5`, and `addTogether(2)` should return a function.

Calling this returned function with a single argument will then return the sum:

```
var sumTwoAnd = addTogether(2);
```

`sumTwoAnd(3)` returns `5`.

If either argument isn't a valid number, return undefined.

```
function addTogether() {
  // Function to check if a number is actually a number
      // and return undefined otherwise.
      var checkNum = function(num) {
        if (typeof num !== 'number') {
          return undefined;
        } else
```

```
    return num;
  };

  // Check if we have two parameters, check if they are numbers
  // handle the case where one is not
  // returns the addition.
  if (arguments.length > 1) {
    var a = checkNum(arguments[0]);
    var b = checkNum(arguments[1]);
    if (a === undefined || b === undefined) {
      return undefined;
    } else {
      return a + b;
    }
  } else {
    // If only one parameter was found, returns a new function that expects two
    // Store first argument before entering the new function scope
    var c = arguments[0];

    // Check the number again, must be outside the function to about returning an object
    // instead of undefined.
    if (checkNum(c)) {
      // Return function that expect a second argument.
      return function(arg2) {
        // Check for non-numbers
        if (c === undefined || checkNum(arg2) === undefined) {
          return undefined;
        } else {
          // if numbers then add them.
          return c + arg2;
        }
      };
    }
  }
}

addTogether(2,3);


or:
function addTogether() {
  var args = new Array(arguments.length);
  //Storing the arguments in an array
  for(var i = 0; i < args.length; ++i) {
    args[i] = arguments[i];
    }
 //Check for the arguments length
```

```
    if(args.length == 2){
      //If there are two arguments,check for the type of both arguments
      //Use typeof to check the type of the argument(both should be numbers)
      if(typeof args[0] !== 'number' || typeof args[1] !=='number' ){
        return undefined;
        }
      return args[0]+args[1];
     }
    //When only one argument is provided
    if(args.length == 1){
        a= args[0];
       //Check the  argument using typeof
      if(typeof a!=='number'){
          return undefined;
        }
      else{
        //Making use of closures
        return function(b){
        //Checking the second argument
          if(typeof b !=='number'){
            return undefined;
            }
          else
            return a+b;
          };
       }
      }
   }
```

or:

```
 //jshint esversion: 6
    function addTogether() {
      var args = Array.from(arguments);
      return args.some(n => typeof n !== 'number') ?
        undefined:
        args.length > 1 ?
          args.reduce((acc, n) => acc += n, 0):
          (n) => typeof n === "number" ?
            n + args[0]:
            undefined;
    }
```

Make a Person

Fill in the object constructor with the following methods below:

> getFirstName() getLastName() getFullName() setFirstName(first) setLastName(last) setFullName(firstAndLast)

Run the tests to see the expected output for each method.

The methods that take an argument must accept only one argument and it has to be a string.

These methods must be the only available means of interacting with the object.

```
var Person = function(firstAndLast) {
  // Complete the method below and implement the others similarly
  var fullName = firstAndLast;

  this.getFirstName = function() {
    return fullName.split(" ")[0];
  };

  this.getLastName = function() {
    return fullName.split(" ")[1];
  };

  this.getFullName = function() {
    return fullName;
  };

  this.setFirstName = function(name) {
    fullName = name + " " + fullName.split(" ")[1];
  };

  this.setLastName = function(name) {
    fullName = fullName.split(" ")[0] + " " + name;
  };

  this.setFullName = function(name) {
    fullName = name;
  };
};

var bob = new Person('Bob Ross');
bob.getFullName();
```

Map the Debris

Return a new array that transforms the elements' average altitude into their orbital periods (in seconds).

The array will contain objects in the format `{name: 'name', avgAlt: avgAlt}`.

You can read about orbital periods on Wikipedia.

The values should be rounded to the nearest whole number. The body being orbited is Earth.

The radius of the earth is 6367.4447 kilometers, and the GM value of earth is 398600.4418 km^3s^{-2}.

```
function orbitalPeriod(arr) {
  var GM = 398600.4418;
  var earthRadius = 6367.4447;
  var a = 2 * Math.PI;
  var newArr = [];
  var getOrbPeriod = function(obj) {
    var c = Math.pow(earthRadius + obj.avgAlt, 3);
    var b = Math.sqrt(c / GM);
    var orbPeriod = Math.round(a * b);
    delete obj.avgAlt;
    obj.orbitalPeriod = orbPeriod;
    return obj;
  };

  for (var elem in arr) {
    newArr.push(getOrbPeriod(arr[elem]));
  }

  return newArr;
}

orbitalPeriod([{name : "sputnik", avgAlt : 35873.5553}]);
```

or:

```
function orbitalPeriod(arr) {
  var GM = 398600.4418;
  var earthRadius = 6367.4447;

  //Looping through each key in arr object
  for(var prop in arr) {
    //Rounding off the orbital period value
    var orbitalPer = Math.round(2 * Math.PI * Math.sqrt(Math.pow(arr[prop].avgAlt + earthRadius, 3) /
GM));
    //deleting the avgAlt property
    delete arr[prop].avgAlt;
```

```
    //adding orbitalPeriod property
    arr[prop].orbitalPeriod = orbitalPer;
  }

  return arr;
}


or:
function orbitalPeriod(arr) {
  var GM = 398600.4418;
  var earthRadius = 6367.4447;

  // Loop through each item in the array arr
  arr.forEach(function(item) {
    // Calculate the Orbital period value
    var tmp = Math.round(2 * Math.PI * Math.sqrt(Math.pow(earthRadius + item.avgAlt, 3) / GM));
    //Delete the avgAlt property
    delete item.avgAlt;
    //Add orbitalPeriod property
    item.orbitalPeriod = tmp;
  });
  return arr;
}
```