First Reverse

Have the function `FirstReverse(str)` take the **`str`** parameter being passed and return the string in reversed order. For example: if the input string is "Hello World and Coders" then your program should return the string **sredoC dna dlroW olleH**.

Examples

Input: "coderbyte"
Output: etybredoc

Input: "I Love Code"
Output: edoC evoL I

```
function FirstReverse(str) {

  var arr = str.split('');
  arr.reverse();
  return arr.join('');
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
FirstReverse(readline());
```

```
function FirstReverse(str) {

    let newString = '';
    for (let i = str.length-1; i >= 0; i--) {
        newString += str[i];
    }

    return newString;
}

// keep this function call here
FirstReverse(readline());
```

```
function FirstReverse(str) {
  // code goes here
  return str.split('').reverse().join('');

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
FirstReverse(readline());
```

First Factorial

Have the function `FirstFactorial(num)` take the `num` parameter being passed and return the factorial of it. For example: if `num` = 4, then your program should return **(4 * 3 * 2 * 1)** = 24. For the test cases, the range will be between 1 and 18 and the input will always be an integer.

Examples

Input: 4
Output: 24

Input: 8
Output: 40320

```
function FirstFactorial(num) {

    let factorial = 1;

    for (let i = 1; i <= num; i++) {
        factorial *= i;
    }


  return factorial;

}

// keep this function call here
FirstFactorial(readline());
```

```
function FirstFactorial(num) {

  // code goes here
  if (num < 0) return NaN;
  return (num > 1) ? FirstFactorial(num - 1) * num : 1;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
FirstFactorial(readline());
```

```
function FirstFactorial(num) {

  let answer = 1;
  for (let index = 1; index <= num; index++) {
      answer *= index;
  }
  return answer;

}

// keep this function call here
FirstFactorial(readline());
```

Longest Word

Have the function `LongestWord(sen)` take the **`sen`** parameter being passed and return the largest word in the string. If there are two or more words that are the same length, return the first word from the string with that length. Ignore punctuation and assume **`sen`** will not be empty.

Examples

Input: "fun&!! time"
Output: time

Input: "I love dogs"
Output: love

```
function LongestWord(sen) {

  sen = sen.trim();
  sen = sen.replace(/[^a-zA-Zsd]/g, '');

  var arr = sen.split(' ');

  arr.sort(function(a, b) {return b.length - a.length});

  return arr.shift();

  // code goes here
  return sen;

}
```

```
// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LongestWord(readline());



function LongestWord(sen) {

    let validCharacters =
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';

    let maxLength = 0;
    let longestWord = '';

    for (let i = 0, length = 0, word = ''; i < sen.length; i++) {
        let c = sen[i];
        if (validCharacters.includes(c)) {
            length++;
            word += c;
        } else {
            length = 0;
            word = '';
        }

        if (length > maxLength) {
            maxLength = length;
            longestWord = word;

        }
    }

  return longestWord;

}

// keep this function call here
LongestWord(readline());



function LongestWord(sen) {
  var trimmed = sen.replace(/[^\w]/g, ' ');
  var words = trimmed.split(/\s+/);
  var longestWord = words.sort(function(a, b) {return b.length - a.length;})[0];
  return longestWord;
}

// keep this function call here
LongestWord(readline());
```

Letter Changes

Have the function `LetterChanges(str)` take the `str` parameter being passed and modify it using the following algorithm. Replace every letter in the string with the letter following it in the alphabet (**ie.** c becomes d, z becomes a). Then capitalize every vowel in this new string (a, e, i, o, u) and finally return this modified string.

Examples

Input: "hello*3"
Output: Ifmmp*3

```
Input: "fun times!"
Output: gvO Ujnft!
```

```
function LetterChanges(str) {

  str = str.trim().toLowerCase();
  var len = str.length;
  var newStr = '';

  for (var i = 0; i < len; i++) {
    if (/[a-ce-gi-mo-su-y]/.test(str[i])) {
      newStr += String.fromCharCode(((str[i].charCodeAt(0) - 18) % 26) + 97)
    }
    else if (/[zdhnt]/.test(str[i])) {
        newStr += String.fromCharCode(((str[i].charCodeAt(0) - 18) % 26) + 65);
    }
    else {
     newStr += str[i];
    }
  }
  return newStr;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LetterChanges(readline());
```

```
function LetterChanges(str) {

    let validCharacters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';

    let newString = '';

    for (let i = 0; i < str.length; i++) {
        if (validCharacters.includes(str[i])) {
            let char = str.charCodeAt(i);
            newString += (str[i] === 'z' || str[i] === 'Z') ?
String.fromCharCode(char - 25) : String.fromCharCode(char + 1);
        } else {
            newString += str[i];
        }
    }

    let vowels = 'aeiou';

    let finalString = '';

    // capitlize vowels
    for (let i = 0; i < newString.length; i++) {
        if (vowels.includes(newString[i])) {
            finalString += newString[i].toUpperCase();
        } else {
            finalString += newString[i];
        }
    }

    return finalString;

}

// keep this function call here
LetterChanges(readline());



function LetterChanges(str) {
  str = str.replace(/[a-zA-Z]/g, function(ch) {
    if (ch === 'z') return 'a';
    else if (ch === 'Z') return 'A';
    else return String.fromCharCode(ch.charCodeAt(0) + 1);
  });

  return str.replace(/[aeiou]/g, function(ch) {
    return ch.toUpperCase();
  });
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LetterChanges(readline());
```

Simple Adding

Have the function `SimpleAdding(num)` add up all the numbers from 1 to **`num`**. For example: if the input is 4 then your program should return 10 because 1 + 2 + 3 + 4 = 10. For the test cases, the parameter **`num`** will be any number from 1 to 1000.

Examples

Input: 12
Output: 78

```
Input: 140
Output: 9870
```

```
function SimpleAdding(num) {
  if (num === 1) {
    return 1;
  }
  else {
    return num + SimpleAdding(num -1);
  }
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SimpleAdding(readline());
```

```
function SimpleAdding(num) {

    let sum = 0;
    for (i = 1; i <= num; i++) {
        sum += i;
    }
    return sum;

}

// keep this function call here
SimpleAdding(readline());
```

```
function SimpleAdding(num) {

  // code goes here
  if (num < 1) return NaN;
  else if (num === 1) return 1;
```

```
  return SimpleAdding(num - 1) + num;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SimpleAdding(readline());
```

Letter Capitalize

Have the function `LetterCapitalize(str)` take the `str` parameter being passed and capitalize the first letter of each word. Words will be separated by only one space.

Examples

Input: "hello world"
Output: Hello World

Input: "i ran there"
Output: I Ran There

```
function LetterCapitalize(str) {

  wordarr = str.split(" ");

  for (var i = 0, n = wordarr.length; i < n; i++) {
   wordarr[i] = wordarr[i][0].toUpperCase() + wordarr[i].slice(1);
  }
  str = wordarr.join(" ");

  return str;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LetterCapitalize(readline());
```

```
function LetterCapitalize(str) {
```

```
    let newString = '';
    for (let i = 0, newWord = true; i < str.length; i++) {
        if (newWord) {
            newString += str[i].toUpperCase();
        } else {
            newString += str[i];
        }

        newWord =  (str[i] === ' ') ? true : false;
    }

    return newString;
}

// keep this function call here
LetterCapitalize(readline());
```

```
function LetterCapitalize(str) {

  // code goes here
  return str.split(/s+/).map(function(word) {
    return word[0].toUpperCase() + word.slice(1);
  }).join(' ');
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LetterCapitalize(readline());
```

Simple Symbols

Have the function `SimpleSymbols(str)` take the **`str`** parameter being passed and determine if it is an acceptable sequence by either returning the string **true** or **false**. The **`str`** parameter will be composed of **+** and **=** symbols with several characters between them (**ie.** ++d+===+c++==a) and for the string to be true each letter must be surrounded by a **+** symbol. So the string to the left would be false. The string will not be empty and will have at least one letter.

Examples

Input: "+d+=3=+s+"
Output: true

Input: "f++d+"
Output: false

```
function SimpleSymbols(str) {

  if (/^[a-zA-Z]/.test(str) || /[a-zA-Z]$/.test(str)) {
      return false;
  }
  else if (/[^+][a-zA-Z]/.test(str) || /[a-zA-Z][^+]/.test(str)) {
      return false;
  }
  else {
   return true;
  }




}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SimpleSymbols(readline());



function SimpleSymbols(str) {

    if (str === '') {
        return 'false';
    }

    let result = str.split('').every(letterEval);

    return result ? 'true' : 'false';

    function letterEval(letter, index) {
        let letters = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
        if (letters.includes(letter)) {
            // Check for first or last character
            if (index === 0 || index === str.length - 1) {
                // Letter is first or last char, fail case
                return false;
            }

            if (str[index - 1] === '+' && str[index + 1] === '+') {
                return true;
            }
        } else {
            // Ignore non letters
            return true;
        }
    }
```

```
}

// keep this function call here
SimpleSymbols(readline());


function SimpleSymbols(str) {

  // code goes here
  if (str.match(/[^+][A-Za-z][^+]/g)) return false;
  else if (str.match(/[+][A-Za-z][^+]/g)) return false;
  else if (str.match(/[^+][A-Za-z][+]/g)) return false;
  return true;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SimpleSymbols(readline());
```

Check Nums

Have the function `CheckNums(num1,num2)` take both parameters being passed and return the string **true** if `num2` is greater than `num1`, otherwise return the string **false**. If the parameter values are equal to each other then return the string **-1**.

Examples

Input: 3 & num2 = 122
Output: true

Input: 67 & num2 = 67
Output: -1

```
VIEW CHALLENGE
function CheckNums(num1,num2) {

  if (num1 == num2) {
     return "-1";
  }
  else {
    return (num2 > num1);
  }
}

// keep this function call here
```

```
// to see how to enter arguments in JavaScript scroll down
CheckNums(readline());



function CheckNums(num1,num2) {

    if (num2 > num1) {
        return true;
    } else if (num1 === num2) {
        return '-1';
    } else {
        return false;
    }

}

// keep this function call here
CheckNums(readline());



function CheckNums(num1,num2) {

  // code goes here
  if ((num1 - num2) < 0) return true;
  else if ((num1 - num2) > 0) return false;
  else return -1;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
CheckNums(readline());
```

Time Convert

Have the function `TimeConvert(num)` take the `num` parameter being passed and return the number of **hours** and **minutes** the parameter converts to (**ie.** if `num` = 63 then the output should be 1:3). Separate the number of hours and minutes with a colon.

Examples

Input: 126
Output: 2:6

Input: 45
Output: 0:45

```
function TimeConvert(num) {
  var hours = Math.floor(num/60);
  var minutes = num % 60;
  var str = hours + ":" + minutes;
  return str;


}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
TimeConvert(readline());
```

```
function TimeConvert(num) {

    let hours = Math.floor(num/60);
    let minutes = num % 60;

    return hours + ':' + minutes;

}

// keep this function call here
TimeConvert(readline());
```

```
function TimeConvert(num) {

  // code goes here
  return Math.floor(num / 60) + ':' + (num % 60);

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
TimeConvert(readline());
```

Alphabet Soup

Have the function `AlphabetSoup(str)` take the **`str`** string parameter being passed and return the string with the letters in alphabetical order (**ie.** hello becomes ehllo). Assume numbers and punctuation symbols will not be included in the string.

Examples

Input: "coderbyte"

Output: bcdeeorty

Input: "hooplah"
Output: ahhloop

```
function AlphabetSoup(str) {

var arrays = str.split("");
var reversearrays = arrays.sort();
var result = reversearrays.join("");


  return result;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
AlphabetSoup(readline());
```

```
function AlphabetSoup(str) {

    let letters = str.split('');
    return letters.sort().join('');

}

// keep this function call here
AlphabetSoup(readline());
```

```
function AlphabetSoup(str) {

  // code goes here
  return str.split('').sort(function(ch1, ch2) {
    return ch1.charCodeAt(0) - ch2.charCodeAt(0);
  }).join('');
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
AlphabetSoup(readline());
```

AB Check

Have the function `ABCheck(str)` take the `str` parameter being passed and return the string true if the characters **a** and **b** are separated by exactly 3 places anywhere in the string at least

once (**ie.** "lane borrowed" would result in true because there is exactly three characters between **a** and **b**). Otherwise return the string false.

Examples

Input: "after badly"
Output: false

Input: "Laura sobs"
Output: true

```
function ABCheck(str) {

  patt = /(a...b|b...a)/

    return patt.test(str);

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ABCheck(readline());
```

```
function ABCheck(str) {
    if (str.length < 5) {
        return false;
    }
    console.log(str.length);

    for (let i = 0; i < str.length-4; i++) {
        // look for a
        if (str[i] === 'a') {
            if (str[i+4] === 'b') {
                return true;
            }
        }

        if (str[i] === 'b') {
            if (str[i+4] === 'a') {
                return true;
            }
        }
    }

    return false;

}
```

```
// keep this function call here
ABCheck(readline());



function ABCheck(str) {

  // code goes here

  return str.match(/[aA].{3}[bB]/) !== null;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ABCheck(readline());
```

Vowel Count

Have the function `VowelCount(str)` take the `str` string parameter being passed and return the number of vowels the string contains (**ie.** "All cows eat grass and moo" would return 8). Do not count **y** as a vowel for this challenge.

Examples

Input: "hello"
Output: 2

Input: "coderbyte"
Output: 3

```
function VowelCount(str) {

  var patt = /[aeiou]/gi;

  var arr = str.match(patt);

  if (arr == null) {
    return 0;
  }
  else {
    return arr.length;
  }
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
VowelCount(readline());
```

```
function VowelCount(str) {

    let vowels = 'aeiou';
    let count = 0;

    for (let i = 0; i < str.length; i++) {
        if (vowels.includes(str[i])) {
            count++;
        }
    }
    return count;
}

// keep this function call here
VowelCount(readline());
```

```
function VowelCount(str) {

  // code goes here
  return str.split('').filter(function(ch) {
    return ['a', 'e', 'i', 'o', 'u'].indexOf(ch.toLowerCase()) > -1;
  }).length;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
VowelCount(readline());
```

Word Count

Have the function `WordCount(str)` take the **`str`** string parameter being passed and return the number of words the string contains (**e.g.** "Never eat shredded wheat or cake" would return 6). Words will be separated by single spaces.

Examples

Input: "Hello World"
Output: 2

Input: "one 22 three"
Output: 3

```
function WordCount(str) {

  var arr = str.split(" ");

  var answer = arr.length;
```

```
  return answer;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
WordCount(readline());

function WordCount(str) {
    return str.split(' ').length;
}

// keep this function call here
WordCount(readline());


function WordCount(str) {

  // code goes here
  return str.split(' ').length;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
WordCount(readline());
```

Ex Oh

Have the function `ExOh(str)` take the **`str`** parameter being passed and return the string **true** if there is an equal number of **x**'s and **o**'s, otherwise return the string **false**. Only these two letters will be entered in the string, no punctuation or numbers. For example: if **`str`** is "xooxxxxooxo" then the output should return **false** because there are 6 x's and 5 o's.

Examples

Input: "xooxxo"
Output: true

Input: "x"
Output: false

```
function ExOh(str) {
    var regExPatternX = /x/gi;
    var regExPatternO = /o/gi;
```

```
        var arrayXLen = str && str.match(regExPatternX) ?
str.match(regExPatternX).length : 0;
        var arrayOLen = str && str.match(regExPatternO) ?
str.match(regExPatternO).length : 0;

        return arrayXLen === arrayOLen;
}

// keep this function call here
ExOh(readline());


function ExOh(str) {

    let xCount = 0;
    let oCount = 0;

    for (let i = 0; i < str.length; i++) {
        if (str[i] === 'x') {
            xCount++;
        }
        if (str[i] === 'o') {
            oCount++;
        }
    }

    return (xCount === oCount) ? true : false;

}

// keep this function call here
ExOh(readline());


function ExOh(str) {

  // code goes here
  var letters = str.split('');
  var numOfEx = letters.filter(function(ch) {
    return ch === 'x';
  }).length;
  var numOfOh = letters.filter(function(ch) {
    return ch === 'o';
  }).length;

  return numOfEx === numOfOh;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ExOh(readline());
```

Palindrome

Have the function `Palindrome(str)` take the **`str`** parameter being passed and return the string **true** if the parameter is a palindrome, (the string is the same forward as it is backward) otherwise return the string **false**. For example: "racecar" is also "racecar" backwards. Punctuation and numbers will not be part of the string.

Examples

Input: "never odd or even"
Output: true

Input: "eye"
Output: true

```
function Palindrome(str) {
  modified = str.replace(/W/g,"");

  var arr1 = modified.split("");
  var arr2 = arr1;
  arr2 = arr2.reverse();
  str2 = arr2.join("");

  return modified == str2;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
Palindrome(readline());
```

```
function Palindrome(str) {

    str = str.replace(/ /g, '');

    for (let i = 0, max = Math.floor(str.length/2); i < max; i++) {
        if (str[i] !== str[str.length-1-i]) {
            return false;
        }
    }
    return true;

}

// keep this function call here
Palindrome(readline());
```

```
function Palindrome(str) {

    str = str.replace(/ /g, '');

    for (let i = 0, max = Math.floor(str.length/2); i < max; i++) {
        if (str[i] !== str[str.length-1-i]) {
            return false;
        }
    }
    return true;

}

// keep this function call here
Palindrome(readline());
```

Arith Geo

Have the function `ArithGeo(arr)` take the array of numbers stored in **`arr`** and return the string "**Arithmetic**" if the sequence follows an arithmetic pattern or return "**Geometric**" if it follows a geometric pattern. If the sequence doesn't follow either pattern return **-1**. An arithmetic sequence is one where the difference between each of the numbers is consistent, where as in a geometric sequence, each term after the first is multiplied by some constant or common ratio. Arithmetic example: [2, 4, 6, 8] and Geometric example: [2, 6, 18, 54]. Negative numbers may be entered as parameters, 0 will not be entered, and no array will contain all the same elements.

Examples

Input: [5,10,15]
Output: Arithmetic

Input: [2,4,16,24]
Output: -1

```
function ArithGeo(arr) {
var len = arr.length;
var arithK = arr[1] - arr[0];
var geoK = arr[1] / arr[0];

  for (var i = 0; i < len - 1; i++) {
    if (arr[i+1] - arr [i] !== arithK) {
     break;
    }
    else if (i === len - 2) {
```

```
      return "Arithmetic";
    }
  }

  for (var i = 0; i < len - 1; i++) {
    if (arr[i+1] / arr [i] !== geoK) {
     break;
    }
    else if (i === len - 2) {
      return "Geometric";
    }
  }

  return -1;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ArithGeo(readline());



function ArithGeo(arr) {

    let arithmetic = true;
    // test arithmetic
    for (let i = 2, diff = arr[1] - arr[0]; i < arr.length; i++) {
        if (arr[i] - arr[i-1] !== diff) {
            arithmetic = false;
        }
    }

    if (arithmetic) {
        return 'Arithmetic';
    }

    let geometric = true;
    // geometric
    for (let i = 2, divisor = arr[1] / arr[0]; i < arr.length; i++) {
        if (arr[i] / arr[i-1] !== divisor) {
            geometric = false;
        }
    }

    if (geometric) {
        return 'Geometric';
    }

    return -1;


}

// keep this function call here
```

```
ArithGeo(readline());


function ArithGeo(arr) {

  // code goes here
  var isArith = function(arr) {
    if (arr.length < 2) return -1;
    var difference = arr[1] - arr[0];
    for (var i = 1; i < arr.length - 1; i++) {
      if (arr[i + 1] - arr[i] !== difference) return false;
    }
    return true;
  };

  var isGeo = function(arr) {
    if (arr.length < 2) return -1;
    var multiplier = arr[1] / arr[0];
    for (var i = 1; i < arr.length - 1; i++) {
      if (arr[i + 1] / arr[i] !== multiplier) return false;
    }
    return true;
  };

  if (isArith(arr)) return "Arithmetic";
  else if (isGeo(arr)) return "Geometric";
  else return -1;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ArithGeo(readline());
```

Array Addition I

Have the function `ArrayAdditionI(arr)` take the array of numbers stored in **`arr`** and return the string **true** if any combination of numbers in the array (excluding the largest number) can be added up to equal the **largest number** in the array, otherwise return the string **false**. For example: if **`arr`** contains [4, 6, 23, 10, 1, 3] the output should return **true** because 4 + 6 + 10 + 3 = 23. The array will not be empty, will not contain all the same elements, and may contain negative numbers.

Examples

Input: [5,7,16,1,2]
Output: false

Input: [3,5,-1,8,12]

Output: true

```
function ArrayAdditionI(arr) {
     var target;
    var addArr = arrayPrep(arr);
    var len = addArr.length;
    var permNum = Math.pow(2, len);
    console.log('test0 ', permNum, target, addArr);

    for(var i = 0; i <= permNum; i++) {
        permStr = (i).toString(2);
        strlen = permStr.length;
        var counter = 0;
        for(var j = 0; j < strlen; j++) {
            if(permStr[j] === '1') {
                counter += addArr[j];
            }
        }
        if (counter === target) {
            return true
        }
    }
    return false

    function arrayPrep(arr2) {
        arr.sort(function(a, b){
            return a - b
        });
        target = arr2.pop()
        return arr2
    }
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ArrayAdditionI(readline());


function ArrayAdditionI(arr) {

    // find largest number
    let largestIndex = 0;
    let largestNumber = 0;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] > largestNumber) {
            largestIndex = i;
            largestNumber = arr[i];
        }
    }
    arr.splice(largestIndex, 1);
```

```
    let combos = [];
    let size = arr.length;

    for (let i = 0, max = Math.pow(2, size); i < max; i++) {
        let num = i.toString(2);
        while (num.length < size) {
            num = '0' + num;
        }
        combos.push(num);
    }

    // iterate over all combos of numbers
    for (let i = 0; i < combos.length; i++) {

        let sum = 0;
        for (let j = 0; j < combos[i].length; j++) {
            if (combos[i][j] === '1') {
                sum += arr[j];
            }
        }

        if (sum === largestNumber) {
            return true;
        }
    }
    return false;
}

// keep this function call here
ArrayAdditionI(readline());

function ArrayAdditionI(arr) {

  // code goes here
  var sortedArr = arr.sort(function(a,b) {return a-b;});
  var largest = sortedArr.pop();
  var sums = [];

  for (i = 0; i < Math.pow(2, sortedArr.length); i++) {
    var sum = 0;
    var bitMask = i.toString(2).split('');

    while (bitMask.length < sortedArr.length) {
      bitMask.unshift('0');
    }

    for (j = 0; j < bitMask.length; j++) {
      if (bitMask[j] === '1') {
        sum += sortedArr[j];
      }
    }
    sums.push(sum);
  }
```

```
  return sums.indexOf(largest) !== -1;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ArrayAdditionI(readline());
```

Letter Count I

Have the function `LetterCountI(str)` take the **`str`** parameter being passed and return the first word with the greatest number of repeated letters. For example: "Today, is the greatest day ever!" should return **greatest** because it has 2 e's (and 2 t's) and it comes before **ever** which also has 2 e's. If there are no words with repeating letters return **-1**. Words will be separated by spaces.

Examples

Input: "Hello apple pie"
Output: Hello

Input: "No words"
Output: -1

```
function LetterCountI(str) {
  var arr = str.toLowerCase().split(" ");
  var n = arr.length;
  var counter = 1;
  var maxcount = 0;
  var newarr = [];

  for (var i = 0; i < n; i++) {
      wordarr = arr[i].split("");
      wordarr.sort();

      for (var j = 0; j < wordarr.length; j++) {
        if (wordarr[j] == wordarr[j+1]) {
          counter++;
        }
        else {
          if (counter > maxcount) {
            maxcount = counter;
            counter = 1;
          }
        }
      }
      newarr.push([arr[i], maxcount])
        maxcount = 0;
    }
```

```
  newarr.sort(function(a, b) {return b[1] - a[1]});

  if (newarr[0][1] == 1) {
    return -1;
  }
  else {
    return newarr[0][0];
  }
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LetterCountI(readline());


function LetterCountI(str) {

    let words = str.split(' ');

    let bestCount = 0;
    let bestWord = '';

    for (let i = 0; i < words.length; i++) {
        let letterCount = [];
        let largestCount = 0;
        for (let j = 0; j < words[i].length; j++) {
            letterCount[words[i][j]] = (letterCount[words[i][j]] === undefined) ? 1 :
++letterCount[words[i][j]];
            if (letterCount[words[i][j]] >= 2 && letterCount[words[i][j]] >
largestCount) {
                largestCount = letterCount[words[i][j]];
            }
        }

        if (largestCount > bestCount) {
            bestCount = largestCount;
            bestWord = words[i];
        }
    }
    return (bestWord === '') ? -1 : bestWord;

}

// keep this function call here
LetterCountI(readline());


function LetterCountI(str) {

  // code goes here
  var words = str.replace(/[^A-Za-zs]/g, '').split(/s+/);

  var counts = words.map(function(word) {
```

```
    word = word.toLowerCase();
    var letters = [];
    for (letter of word) {
      if (letters[letter]) {
        letters[letter] += 1;
      } else {
        letters[letter] = 1;
      }
    }
    return letters;
  });

  var greatestCounts = counts.map(function(word) {
    var greatest = 0;
    for (letter in word) {
      if (word[letter] > greatest) {
        greatest = word[letter];
      }
    }
    return greatest;
  });

  var greatest = Math.max.apply(null, greatestCounts);
  if (greatest <= 1) return -1;
  return words[greatestCounts.indexOf(greatest)];
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
LetterCountI(readline());
```

Second GreatLow

Have the function `SecondGreatLow(arr)` take the array of numbers stored in **`arr`** and return the second lowest and second greatest numbers, respectively, separated by a space. For example: if **`arr`** contains [7, 7, 12, 98, 106] the output should be **12 98**. The array will not be empty and will contain at least 2 numbers. It can get tricky if there's just two numbers!

Examples

Input: [1, 42, 42, 180]
Output: 42 42

Input: [4, 90]
Output: 90 4

```
function SecondGreatLow(arr) {
  arr.sort(function(a, b) {return a - b});
  arr = lonely(arr);
  var newlen = arr.length;

  return arr[1] + ' ' + arr[newlen - 2]

    function lonely(arr) {
      var len = arr.length;
      var testobj = {};
      var output = [];
      var count = 0;
      for (var i = 0; i < len; i++) {
        var holder = arr[i];
        if (!testobj[holder]) {
          testobj[holder] = true;
          output[count++] = holder;
        }
      }
      return output
    }
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SecondGreatLow(readline())


function SecondGreatLow(arr) {

    arr = arr.slice().sort((a, b) => a - b);

    let second = arr[1];
```

```
    for (let i = 1; i < arr.length; i++) {
        if (arr[i] !== arr[0]) {
            second = arr[i];
            break;
        }
    }

    let penultimate = arr[arr.length - 2];
    for (let i = arr.length - 2; i >= 0; i--) {
        if (arr[i] !== arr[arr.length - 1]) {
            penultimate = arr[i];
            break;
        }
    }

    return second + ' ' + penultimate;

}

// keep this function call here
SecondGreatLow(readline());


function SecondGreatLow(arr) {

  // code goes here
  var sorted = arr.sort(function(a,b){return a-b;});
  var greatest = sorted[sorted.length - 1];
  var lowest = sorted[0];
  var secondGreatest, secondLowest;

  for (var i = 0; i < sorted.length; i++) {
    if (sorted[i] !== lowest) {
      secondLowest = sorted[i];
      break;
    }
  }

  for (var i = sorted.length - 1; i >= 0; i--) {
    if (sorted[i] !== greatest) {
      secondGreatest = sorted[i];
      break;
    }
  }

  return secondLowest + " " + secondGreatest;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SecondGreatLow(readline());
```

Division Stringified

Have the function `DivisionStringified(num1,num2)` take both parameters being passed, divide **num1** by **num2**, and return the result as a string with properly formatted commas. If an answer is only 3 digits long, return the number with no commas (**ie.** 2 / 3 should output "1"). For example: if **num1** is **123456789** and **num2** is **10000** the output should be "**12,346**".

Examples

Input: 5 & num2 = 2
Output: 3

Input: 6874 & num2 = 67
Output: 103

```
VIEW CHALLENGE
function DivisionStringified(num1,num2) {

  var res = Math.round(num1/num2);
  var string = res.toString();
  var numleng = string.length;
  var newstring = string;

 if (numleng > 3) {
    var arr = string.split("");
   for (var i= numleng - 3; i > 0; i = i - 3) {
        arr.splice(i, 0, ",");
   }
    var newstring = arr.join("");
 }

  return newstring;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
DivisionStringified(readline());


function DivisionStringified(num1,num2) {

    let quotientString = Math.round(num1 / num2).toString();

    let temp = quotientString.split('').reverse();
    let newString = '';
    for (let i = 0; i < temp.length; i++) {
        if (i % 3 === 2 && i != temp.length-1) {  // need comma next
```

```
            newString = ',' + temp[i] + newString;
        } else {
            newString = temp[i] + newString;
        }
    }
    //console.log(newString);
    return newString;
}

// keep this function call here
DivisionStringified(readline());


function DivisionStringified(num1,num2) {

  // code goes here
  var format = function(num) {
    var result = '';
    var whole = Math.round(num);

    if (whole === 0) return 0;

    while (whole > 0) {
      result = (result === '' ? '' : ',') + result;
      result = (whole % 1000) + result;
      whole = Math.floor(whole / 1000);
    }

    return result;
  }

  return format(num1 / num2);
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
DivisionStringified(readline());
```

Counting Minutes I

Have the function `CountingMinutesI(str)` take the **`str`** parameter being passed which will be two times (each properly formatted with a colon and am or pm) separated by a hyphen and return the total number of minutes between the two times. The time will be in a 12 hour clock format. For example: if **`str`** is **9:00am-10:00am** then the output should be **60**.
If **`str`** is **1:00pm-11:00am** the output should be **1320**.

Examples

Input: "12:30pm-12:00am"

Output: 690

Input: "1:23am-1:08am"

Output: 1425

```
Solutions for Counting Minutes I
JAVASCRIPT
VIEW CHALLENGE
function CountingMinutesI(str) {
var seps = str.split("-");
var col1 = seps[0].indexOf(":");
var col2 = seps[1].indexOf(":");
var hour1 = parseInt(seps [0].slice(0, col1));
var hour2 = parseInt(seps[1].slice(0, col2));
var min1 = parseInt(seps[0].slice(col1+1, col1+3));
var min2 = parseInt(seps[1].slice(col2+1, col2+3));
var ampm1 = seps[0].slice(-2);
var ampm2 = seps[1].slice(-2);
  if (ampm1 == "pm" && hour1 != 12) {
    hour1 = hour1 + 12;
  }
  if (ampm2 == "pm" && hour2 != 12) {
    hour2 = hour2 + 12;
  }
  if (hour1 == 12 && ampm1 == "am") {
    hour1 = 0;
  }
  if (hour2 == 12 && ampm2 == "am") {
    hour2 = 0;
      }
var time1 = (hour1*60) + min1;
var time2 = (hour2*60) + min2;

  var diff = time2 - time1;

  if (diff < 0) {
    diff = diff + (60 * 24);
  }

  return diff;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
CountingMinutesI(readline());
```

```
function CountingMinutesI(str) {

    let times = str.split('-');

    times = times.map(function(currentValue, index, array){
        let pairs = currentValue.split(':');
        let time = ((pairs[1][2] === 'a') ? parseInt(pairs[0]) % 12 :
parseInt(pairs[0]) % 12 + 12) * 60 + parseInt(pairs[1][0] + pairs[1][1]);
        return time;
    });

   let diff = times[1] - times[0];
   return (diff < 0) ? diff + 1440 : diff;
}

// keep this function call here
CountingMinutesI(readline());

function CountingMinutesI(str) {

  // code goes here
  var times = str.split('-');
  var from = times[0], to = times[1];

  function parseMinutes(timeStr) {
    var time = timeStr.match(/d+/g);
    var hour = parseInt(time[0]), minute = parseInt(time[1]);
    var ampm = (timeStr.match(/[a|p]m/g)[0] === 'am') ? 0 : 1;
    if (hour === 12) hour = 0;

    return (hour + ampm * 12) * 60 + minute;
  }

  var fromMinutes = parseMinutes(from), toMinutes = parseMinutes(to);
  var difference = toMinutes - fromMinutes;
  var oneDayInMinutes = 24 * 60;

  return (difference < 0) ?  oneDayInMinutes + difference : difference;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
CountingMinutesI(readline());
```

Mean Mode

Have the function `MeanMode(arr)` take the array of numbers stored in **`arr`** and return **1** if the *mode* equals the *mean*, **0** if they don't equal each other (**ie.** [5, 3, 3, 3, 1] should return **1** because the mode (3) equals the mean (3)). The array will not be empty, will only contain positive integers, and will not contain more than one mode.

Examples

Input: [1, 2, 3]
Output: 0

Input: [4, 4, 4, 6, 2]
Output: 1

```
function MeanMode(arr) {
    len = arr.length
    //Step 1: determine the mean - self explanatory
    function mean(arr) {
        var count = 0;
        for (var i = 0; i < len; i++) {
            count += arr[i]
        }
        console.log (count / len)
        return count / len
    }
    //Step 2: determine the mode. We need to count how many of each type are in the
array.  One alternative is to keep a counter going, sort the array and then count
until an item changes, keeping track of the maximum count and the associate value.
The following is a much easier way, assuming that one has a basic familiarity with
javascript objects.  It makes each new entry a key, and keeps count of how many of
each there are, then makes an array of the key-value pairs created.

    function mode(arr) {
        var obj = {};
        for (var i = 0, len = arr.length; i < len; i++) {
            if (obj[arr[i]] === undefined) {
                obj[arr[i]] = 1;
            }
            else {
                obj[arr[i]]++;
            }
        }
        var objarr = [];
        for (x in obj) {
            objarr.push([x, obj[x]]);
        }
        objarr.sort(function(a, b) {return b[1] - a[1]});
        console.log(objarr[0][0]);
```

```
        return objarr[0][0];
    }
    //Compare the mean and the mode
    return mode(arr) == mean(arr)? 1: 0;
}

// keep this function call here
MeanMode(readline());
```

```
function MeanMode(arr) {


    // Find mode
    let mostOccurances = 0;
    let mode = 0;
    for (let i = 0; i < arr.length; i++) {
        let marco = arr[i];
        for (let j = i+1, count = 0; j < arr.length; j++) {
            if (marco === arr[j]) {
                // match!
                count++;
                if (count > mostOccurances) {
                    mostOccurances = count;
                    mode = arr[j];
                }
            }
        }
    }


    // Find mean
    let mean = 0;
    for (let i = 0; i < arr.length; i++) {
        mean += arr[i];
    }
    mean = Math.round(mean/arr.length);

    return (mean === mode) ? 1 : 0;

}

// keep this function call here
MeanMode(readline());
```

```
function MeanMode(arr) {

  // code goes here
  function mode(arr) {
```

```
    var counts = [], mode, maxCount = 0;
    for (num of arr) {
      num = num + ' ';
      if (counts[num]) counts[num]++;
      else counts[num] = 1;
    }

    for (num in counts) {
      if (counts[num] > 1 && counts[num] > maxCount) {
        mode = parseInt(num);
        maxCount = counts[num];
      }
    }
    return mode;
  }

  function mean(arr) {
    return arr.reduce(function(a,b) {return a+b;}, 0) / arr.length;
  }

  return mode(arr) === mean(arr) ? 1 : 0;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
MeanMode(readline());
```

Dash Insert

Have the function `DashInsert(str)` insert dashes ('-') between each two odd numbers in **`str`**. For example: if **`str`** is **454793** the output should be **4547-9-3**. Don't count zero as an odd number.

Examples

Input: 99946

Output: 9-9-946

Input: 56730

Output: 567-30

```
function DashInsert(num) {
var strnum = num.toString();
var arr = strnum.split("");

  for (i = 0; i < arr.length; i++) {
    arr[i] = parseInt(arr[i]);
  }
```

```
  for (i = 0; i < arr.length - 1; i++) {
    if (arr[i]%2 == 1 && arr[i + 1]%2 == 1) {
    arr.splice(i+1, 0, "-");
    }
  }
  x = arr.join("");

  return x;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
DashInsert(readline());


function DashInsert(str) {

    let newString = str[0];

    for (let i = 1, lastOdd = str[0] % 2; i < str.length; i++) {
       if (str[i] % 2 === 1 && lastOdd === 1) {
           newString += '-' + str[i];
       } else {
           newString += str[i];
       }
       lastOdd = str[i] % 2;
    }
    return newString;
}

// keep this function call here
DashInsert(readline());


function DashInsert(str) {

  // code goes here
  var digits = str.split('');

  function isOdd(n) {
    return parseInt(n) % 2 === 1;
  }

  for (var i = 0; i < digits.length - 1; i++) {
    if ( isOdd(digits[i]) && isOdd(digits[i+1]) ) {
      digits.splice(i + 1, 0, '-');
      i++;
    }
  }
  return digits.join('');
}

// keep this function call here
```

```
// to see how to enter arguments in JavaScript scroll down
DashInsert(readline());
```

Swap Case

Have the function `SwapCase(str)` take the **`str`** parameter and swap the case of each character. For example: if **`str`** is "Hello World" the output should be **hELLO wORLD**. Let numbers and symbols stay the way they are.

Examples

Input: "Hello-LOL"
Output: hELLO-lol

Input: "Sup DUDE!!?"
Output: sUP dude!!?

```
function SwapCase(str) {
arr = str.split("");
  for (var i = 0; i < arr.length; i++) {
    if (arr[i] == arr[i].toUpperCase()) {
     arr[i] = arr[i].toLowerCase();
    }
    else if (arr[i] == arr[i].toLowerCase()) {
     arr[i] = arr[i].toUpperCase();
    }
  }
    str = arr.join("");
  return str;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SwapCase(readline());
```

```
function SwapCase(str) {

    const LOWER = 'abcdefghijklmnopqrstuvwxyz';
    const UPPER = 'ABCDEFGHIJKLMNOPWRSTUVWXYZ';

    let newString = '';
    for (let i = 0; i < str.length; i++) {
        if (LOWER.includes(str[i])) {
            newString += str[i].toUpperCase();
        } else if (UPPER.includes(str[i])) {
```

```
            newString += str[i].toLowerCase();
        } else {
            newString += str[i];
        }
    }
    return newString;
}

// keep this function call here
SwapCase(readline());


function SwapCase(str) {

  // code goes here
  return str.split('').map(function(letter) {
    if (letter === letter.toLowerCase()) return letter.toUpperCase();
    else if (letter === letter.toUpperCase()) return letter.toLowerCase();
    else return letter;
  }).join('');
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
SwapCase(readline());
```

Number Addition

Have the function `NumberSearch(str)` take the **`str`** parameter, search for all the numbers in the string, add them together, then return that final number. For example: if **`str`** is "88Hello 3World!" the output should be **91**. You will have to differentiate between single digit numbers and multiple digit numbers like in the example above. So "55Hello" and "5Hello 5" should return two different answers. Each string will contain at least one letter or symbol.

Examples

Input: "75Number9"
Output: 84

Input: "10 2One Number*1*"
Output: 13

```
VIEW CHALLENGE
function NumberAddition(str) {
    var sum = 0;
    var count = 0;
    var strArr = function(str) {
        //take out the non-digits and replace with spaces to keep separated
        str = str.replace(/[^0-9]/g, ' ');
        //take off any preceding or final spaces
        str = str.trim();
        //turn any double spaces into single spaces, until no more doubles remain
        while (str.indexOf('  ') !== -1) {
            str = str.replace(/ss/g, ' ');
        }
        //return an array of number strings without spaces;
        return str.split(' ');
    }
    //call the strArr function on the user input and store as "prepped"
    var prepped = strArr(str);
    //turn each number string into a number
    prepped = prepped.map(function(val) {
        return val ? parseInt(val): 0;
    })
    console.log( 'preppedmap', prepped);
    //run the reduce method to add up all the numbers
    var ans = prepped.reduce(function(firstval, lastval){
        return firstval + lastval
    })

    return ans
}

// keep this function call here
```

```
// to see how to enter arguments in JavaScript scroll down
NumberAddition(readline());



function NumberAddition(str) {

    let DIGITS = '0123456789';
    let numbers = [];

    // First find numbers
    for (let i = 0, number = ''; i < str.length; i++) {

        if (!DIGITS.includes(str[i])) {
            if (number !== '') {
                numbers.push(number);
            }
            number = '';
        } else {
            number += str[i];

            // special case for last char
            if (i === str.length-1) {
                numbers.push(number);
            }
        }
    }

    let sum = 0;
    for (let i = 0; i < numbers.length; i++) {
        sum += parseInt(numbers[i]);
    }
    return sum;
}

// keep this function call here
NumberAddition(readline());


function NumberAddition(str) {
    var numbers = str.match(/\d+/g) || [];
    return numbers.reduce(function(sum, number) {return sum + Number(number)}, 0);
}

// keep this function call here
NumberAddition(readline());
```

Third Greatest

Have the function `ThirdGreatest(strArr)` take the array of strings stored in **`strArr`** and return the **third largest** word within it. So for example: if **`strArr`** is ["hello", "world", "before", "all"] your output should be **world** because "before" is 6 letters long, and "hello" and "world" are

both 5, but the output should be **world** because it appeared as the last 5 letter word in the array. If **`strArr`** was ["hello", "world", "after", "all"] the output should be **after** because the first three words are all 5 letters long, so return the last one. The array will have at least three strings and each string will only contain letters.

Examples

Input: ["coder","byte","code"]
Output: code

Input: ["abc","defg","z","hijk"]
Output: abc

```
function ThirdGreatest(strArr) {
  strArr.sort(function(a, b) {return b.length - a.length});
  return strArr[2];
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ThirdGreatest(readline());
```

```
function ThirdGreatest(strArr) {

    strArr.sort(function(a,b){return b.length - a.length});

    return strArr[2];

}

// keep this function call here
ThirdGreatest(readline());
```

```
function ThirdGreatest(strArr) {

  // code goes here
  return strArr.sort(function(a, b) {
    return (b.length - a.length);
  })[2];

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
ThirdGreatest(readline());
```

Powers of Two

Have the function `PowersofTwo(num)` take the `num` parameter being passed which will be an integer and return the string **true** if it's a power of two. If it's not return the string **false**. For example if the input is 16 then your program should return the string true but if the input is 22 then the output should be the string false.

Examples

Input: 4
Output: true

Input: 124
Output: false

```
function PowersofTwo(num) {

  if (num < 2){
   ans = "false";
  }

  else {
  while (num >= 2){
    var test = num%2;
    if (test == 0){
     num = num/2
       ans = "true";}
    else{
      num = 0;
      ans = "false"}
  }
    }
  // code goes here
  return ans;

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
PowersofTwo(readline());


function PowersofTwo(num) {
```

```
    return Number.isInteger(Math.log(num) / Math.log(2)) ? true : false;

}

// keep this function call here
PowersofTwo(readline());

function PowersofTwo(num) {

  // code goes here
  if (num === 1) return true;
  else {
    return (num % 2 === 0) ? PowersofTwo(num / 2) : false;
  }
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
PowersofTwo(readline());
```

Additive Persistence

Have the function `AdditivePersistence(num)` take the **num** parameter being passed which will always be a positive integer and return its additive persistence which is the number of times you must add the digits in **num** until you reach a single digit. For example:
if **num** is **2718** then your program should return **2** because 2 + 7 + 1 + 8 = 18 and 1 + 8 = 9 and you stop at 9.

Examples

Input: 4
Output: 0

Input: 19
Output: 2

```
VIEW CHALLENGE
function AdditivePersistence(num) {

  function arrprep(numb) {
    var numstr = numb.toString();
    var arr = numstr.split('');
    var numarr = arr.map(function(val) {
      return parseInt(val)
        })
    return numarr
```

```
  }

  function addup(arr) {
    var redux = arr.reduce(function(a, b){
      return a + b })
   return redux
     }

  var count = 0;

  while (num > 9) {
    num = addup(arrprep(num));
    count++;
  }

  return count

}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
AdditivePersistence(readline());




function AdditivePersistence(num) {

    num = num.toString();
    let count = 0;

    while (count < 10000) { // hard stop
        if (num.length === 1) {
            break;
        }
        count++;
        let sum = 0;
        for (let i = 0; i < num.length; i++) {
            sum += parseInt(num[i]);
        }
        num = sum.toString();

    }
    return count;
}

// keep this function call here
AdditivePersistence(readline());



function AdditivePersistence(num) {

  // code goes here
```

```
  function addDigits(num) {
    return num.toString().split('').map(function(num) {
      return parseInt(num);
    }).reduce(function(a, b) {return a + b;}, 0);
  }

  var result = num;
  var count = 0;
  while (result > 9) {
    result = addDigits(result);
    count++;
  }
  return count;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
AdditivePersistence(readline());
```

Multiplicative Persistence

Have the function `MultiplicativePersistence(num)` take the **`num`** parameter being passed which will always be a positive integer and return its multiplicative persistence which is the number of times you must multiply the digits in **`num`** until you reach a single digit. For example: if **`num`** is **39** then your program should return **3** because 3 * 9 = 27 then 2 * 7 = 14 and finally 1 * 4 = 4 and you stop at 4.

Examples

Input: 4

Output: 0

Input: 25

Output: 2

```
function MultiplicativePersistence(num) {

  function numprep(num) {
    var strnum = num.toString();
    var arr = strnum.split('');
    var numarr = arr.map(function(val) {
      return parseInt(val)});
    return numarr
  }

  function multnums(arr) {
    var newnum = arr.reduce(function(a, b) {
      return a * b})
```

```
        return newnum
  }
  // code goes here


  var count = 0;

  while (num > 9) {
    num = multnums(numprep(num));
    count++
      }
  return count
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
MultiplicativePersistence(readline());

function MultiplicativePersistence(num) {

    num = num.toString();
    let count = 0;
    while(1) {
        if (num.length === 1) {
            break;
        }

        count++;
        let sum = 1;
        for (let i = 0; i < num.length; i++) {
           sum *= parseInt(num[i]);
        }
        num = sum.toString();
    }
    return count;

}

// keep this function call here
MultiplicativePersistence(readline());

function MultiplicativePersistence(num) {

  // code goes here
  function multiplyDigits(num) {
    return num.toString().split('').map(function(num) {
      return parseInt(num);
    }).reduce(function(a, b) {return a * b;}, 1);
  }

  var result = num;
  var count = 0;
  while (result > 9) {
    result = multiplyDigits(result);
```

```
    count++;
  }
  return count;
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
MultiplicativePersistence(readline());
```

Off Line Minimum

Have the function `OffLineMinimum(strArr)` take the **`strArr`** parameter being passed which will be an array of integers ranging from **1...n** and the letter "E" and return the correct subset based on the following rules. The input will be in the following format: ["I","I","E","I",...,"E",...,"I"] where the I's stand for integers and the E means take out the smallest integer currently in the whole set. When finished, your program should return that new set with integers separated by commas. For example: if **`strArr`** is ["5","4","6","E","1","7","E","E","3","2"] then your program should return **4,1,5**.

Examples

Input: ["1","2","E","E","3"]
Output: 1,2
Input: ["4","E","1","E","2","E","3","E"]
Output: 4,1,2,3

```
function OffLineMinimum(strArr) {
    var len = strArr.length;
    var count = 0;
    var holdArr = [];
    var ans = [];

    for(var i = 0; i < len; i++) {
        if (strArr[count] === "E") {
            var headArr = strArr.splice(0, count);
            strArr.shift();
            holdArr = holdArr.concat(headArr);
            holdArr.sort(function(a, b) {return a - b});
            ans.push(holdArr.shift());
            count = 0;
        }
        else {
            count++;
        }
    }
    return ans.join(',');
}

// keep this function call here
```

```
// to see how to enter arguments in JavaScript scroll down
OffLineMinimum(readline());


function OffLineMinimum(strArr) {

    let result = [];
    for (let i = 0; i < strArr.length; i++) {
        if (strArr[i] === 'E') {
            result.push(removeSmallest(strArr, i));
        } else if (strArr[i] === 'R') {
            // do nothing
        } else {
            // number do nothing
        }
    }

    function removeSmallest(strArr, maxIndex) {

        let smallestIndex = 0;
        let smallestNumber = null;
        strArr.forEach(function(value,index){
            if (index > maxIndex) {
                return;
            }
            if (value !== 'E' && value !== 'R') {
                if (smallestNumber === null || parseInt(value) < smallestNumber) {
                    smallestIndex = index;
                    smallestNumber = parseInt(value);
                }
            }
        });

        strArr[smallestIndex] = 'R';
        return smallestNumber;
    }

    return result.join(',');

}

// keep this function call here
OffLineMinimum(readline());



function OffLineMinimum(strArr) {

  // code goes here
  var arr = [], result = [];
  var log = "";
  for (item of strArr) {
    if (item === 'E') {
      result.push(arr.shift());
```

```
    } else {
      arr.push(parseInt(item));
      arr = arr.sort(function(a, b) {return a - b});
    }
  }
  return result.join(',');
}

// keep this function call here
// to see how to enter arguments in JavaScript scroll down
OffLineMinimum(readline());
```

Changing Sequence

Have the function `ChangingSequence(arr)` take the array of numbers stored in **`arr`** and return the index at which the numbers stop increasing and begin decreasing or stop decreasing and begin increasing. For example: if **`arr`** is [1, 2, 4, 6, 4, 3, 1] then your program should return **3** because 6 is the last point in the array where the numbers were increasing and the next number begins a decreasing sequence. The array will contain at least 3 numbers and it may contains only a single sequence, increasing or decreasing. If there is only a single sequence in the array, then your program should return **-1**. Indexing should begin with 0.

Examples

Input: [-4, -2, 9, 10]
Output: -1

Input: [5, 4, 3, 2, 10, 11]
Output: 3

```
function ChangingSequence(arr) {
    //first, determine if it is an increasing or decreasing sequence
    var type = arr[1] - arr[0] > 0 ? 'increasing' : 'decreasing';

    //set a maximum index of the array, to keep from overflowing
    var maxInd = arr.length - 1;

    //code for an increasing array
    if (type === 'increasing') {
        //findIndex is an array iteration method that came in with ES6. It returns
        //the first value for which the callback returns false.
        var index = arr.findIndex(function(val, ind) {
            while (ind < maxInd) {
                return val > arr[ind + 1];
            }
            return false;
        });
```

```
        return index;
    }

    if (type === 'decreasing') {
        var index = arr.findIndex(function(val, ind) {
            while (ind < maxInd) {
                return val < arr[ind + 1]
            }
            return 0;
        });

        return index;
    }
}

// keep this function call here
ChangingSequence(readline());

function ChangingSequence(arr) {

    let index = null;
    let mode = (arr[1] - arr[0] > 0) ? true : false;
    for (let i = 2; i < arr.length; i++) {
        if (arr[i] - arr[i-1] > 0 !== mode) {
            index = i-1;
            break;
        }
    }
    return (index === null) ? -1 : index;
}

// keep this function call here
ChangingSequence(readline());

function ChangingSequence(arr) {
    if (arr.length < 2) return -1;
    var increasing = arr[0] < arr[1];

    for (var i = 1; i < arr.length - 1; i++) {
        if (increasing) {
            if (arr[i] > arr[i + 1]) return i;
        } else {
            if (arr[i] < arr[i + 1]) return i;
        }
    }

    return -1;
}

// keep this function call here
ChangingSequence(readline());
```

Overlapping Ranges

Have the function `OverlappingRanges(arr)` take the array of numbers stored in **`arr`** which will contain 5 positive integers, the first two representing a range of numbers (a to b), the next 2 also representing another range of integers (c to d), and a final 5th element (x) which will also be a positive integer, and return the string **true** if both sets of ranges overlap by at least x numbers. For example: if **`arr`** is [4, 10, 2, 6, 3] then your program should return the string **true**. The first range of numbers are 4, 5, 6, 7, 8, 9, 10 and the second range of numbers are 2, 3, 4, 5, 6. The last element in the array is 3, and there are 3 numbers that overlap between both ranges: 4, 5, and 6. If both ranges do not overlap by at least x numbers, then your program should return the string **false**.

Examples

Input: [5,11,1,5,1]
Output: true

Input: [1,8,2,4,4]
Output: false

```
function OverlappingRanges(arr) {
    var target = arr.pop();
    var MaxLowerBound = Math.max(arr[0], arr[2]);
    var MinUpperBound = Math.min(arr[1], arr[3]);

    var range = MinUpperBound - MaxLowerBound + 1;

    return range >= target ? 'true' : 'false';
}

// keep this function call here
OverlappingRanges(readline());
```

```
function OverlappingRanges(arr) {

    let count = 0;
    for (let i = arr[0]; i <= arr[1]; i++) {
        if (i >= arr[2] && i <= arr[3]) {
            count++;
        }
    }
    return (count >= arr[4]) ? true : false;
}

// keep this function call here
```

```
OverlappingRanges(readline());

function OverlappingRanges(arr) {
    var range1 = [arr[0], arr[1]]; // (a, b)
    var range2 = [arr[2], arr[3]]; // (c, d)
    var minimumOverlap = arr[4] - 1;

    // determines the overlap between two ranges
    // a negative number means there is no overlap
    function overlap(range1, range2) {
        if (range1[0] < range2[0]) {
            // a----b
            //    c----d
            return (range1[1] > range2[1] ? range2[1] : range1[1]) - range2[0];
        } else {
            //   a----b
            // c---d
            return (range2[1] > range1[1] ? range1[1] : range2[1]) - range1[0];
        }
    }

    return overlap(range1, range2) >= minimumOverlap;
}

// keep this function call here
OverlappingRanges(readline());
```

Superincreasing

Have the function `Superincreasing(arr)` take the array of numbers stored in **`arr`** and determine if the array forms a superincreasing sequence where each element in the array is greater than the sum of all previous elements. The array will only consist of positive integers. For example: if **`arr`** is [1, 3, 6, 13, 54] then your program should return the string "true" because it forms a superincreasing sequence. If a superincreasing sequence isn't formed, then your program should return the string "false"

Examples

Input: [1,2,3,4]
Output: false

Input: [1,2,5,10]
Output: true

```
function Superincreasing(arr) {
    var maxInd = arr.length - 1;
```

```
    var target;
    var sum;

    for (var i = maxInd; i > 0; i--) {
        target = arr.pop()
        sum = arr.reduce(function(val1, val2) {
            return val1 + val2;
        });
        if (sum >= target) {
            return 'false';
        }
    };
    return 'true';

}
Superincreasing(readline());


function Superincreasing(arr) {

    for (let i = 0, sum = 0; i < arr.length; i++) {
        if (arr[i] <= sum) {
            return false;
        }
        sum += arr[i];
    }
    return true;
}

// keep this function call here
Superincreasing(readline());

function Superincreasing(arr) {
    if (arr.length < 2) return false;
    if (arr.length === 2) return arr[0] < arr[1];
    if (arr[0] >= arr[1]) return false;

    var total = arr[0] + arr[1];

    for (var i = 2; i < arr.length; i++) {
        if (arr[i] <= total) return false;
        total += arr[i];
    }
    return true;
}

// keep this function call here
Superincreasing(readline());
```

Hamming Distance

Have the function `HammingDistance(strArr)` take the array of strings stored in **`strArr`**, which will only contain two strings of equal length and return the Hamming distance between

them. The Hamming distance is the number of positions where the corresponding characters are different. For example: if **strArr** is ["coder", "codec"] then your program should return **1**. The string will always be of equal length and will only contain lowercase characters from the alphabet and numbers.

Examples

Input: ["10011", "10100"]
Output: 3

Input: ["helloworld", "worldhello"]
Output: 8

```
function HammingDistance(strArr) {

    let count = 0;
    for (let i = 0; i < strArr[0].length; i++) {
        if (strArr[0][i] !== strArr[1][i]) {
            count++
        }
    }
    return count;
}

// keep this function call here
HammingDistance(readline());
```

```
function HammingDistance(strArr) {
    var word1 = strArr[0],
       word2 = strArr[1],
       len = word1.length,
              count = 0;

    for (var i = 0; i < len; i++) {
        if (word1[i] !== word2[i]) {
            count++;
        }
    }
    return count;
}

// keep this function call here
HammingDistance(readline());
```

```
function HammingDistance(strArr) {
    var hammingDistance = 0;
    for (var i = 0; i < strArr[0].length; i++) {
```

```
        if (strArr[0][i] !== strArr[1][i]) {
            hammingDistance++;
        }
    }
    return hammingDistance;
}

// keep this function call here
HammingDistance(readline());
```

Rectangle Area

Top Rated User Solution

```
function RectangleArea(strArr) {
    var obj = {};
    obj.x1 = parseInt(strArr[0].match(/((-*d+)/)[1], 10);
    obj.y1 = parseInt(strArr[0].match(/(-*d+))/)[1], 10);

    for (var i = 1; i < 3; i++) {
        if (obj.x1 !== parseInt(strArr[i].match(/((-*d+)/)[1], 10)) {
            obj.x2 = parseInt(strArr[i].match(/((-*d+)/)[1], 10);
        }
        if (obj.y1 !== parseInt(strArr[i].match(/(-*d+))/)[1], 10)) {
            obj.y2 = parseInt(strArr[i].match(/(-*d+))/)[1], 10);
        }
    }
    if (Object.keys(obj).length !== 4) {
        return 0;
    } else {
        return (Math.abs(obj.x1 - obj.x2) * Math.abs(obj.y1 - obj.y2));
    }
}

// keep this function call here
RectangleArea(readline());


function RectangleArea(strArr) {

    // Parse input into array
    let coords = strArr.map(function(val){
        let coords = val.split(' ');
        let x = parseInt(coords[0].substr(1, coords[0].length-1));
        let y = parseInt(coords[1].substr(0, coords[1].length-1));
        return [x,y];
    });

    let point = coords.shift();
    let x = point[0];
    let y = point[1];

    let deltaX = 0;
    let deltaY = 0;

    coords.forEach(function(val){
```

```
        if (x === val[0]) {
            deltaY = Math.abs(val[1] - y);
        }
        if (y === val[1]) {
            deltaX = Math.abs(val[0] - x);
        }
    });

    return deltaX * deltaY;

}

// keep this function call here
RectangleArea(readline());


function RectangleArea(strArr) {
    var points = strArr.map(str => str.match(/\d+/g));
    var minX = points.map(point => point[0]).sort()[0];
    var minY = points.map(point => point[1]).sort()[0];
    var maxX = points.map(point => point[0]).sort().reverse()[0];
    var maxY = points.map(point => point[1]).sort().reverse()[0];

    return (maxX - minX) * (maxY - minY);
}

// keep this function call here
RectangleArea(readline());
```

Bitwise One

Have the function `BitwiseOne(strArr)` take the array of strings stored in **`strArr`**, which will only contain two strings of equal length that represent binary numbers, and return a final binary string that performed the bitwise OR operation on both strings. A bitwise OR operation places a 0 in the new string where there are zeroes in both binary strings, otherwise it places a 1 in that spot. For example: if **`strArr`** is ["1001", "0100"] then your program should return the string "1101"

Examples

Input: ["100", "000"]
Output: 100

Input: ["00011", "01010"]
Output: 01011

```
VIEW CHALLENGE
function BitwiseOne(strArr) {
    var str1 = strArr[0];
    var str2 = strArr[1];
    var newStr = '';
    len = str1.length;

    for (var i = 0; i < len; i++) {
        if(str1.charAt(i) === '1' || str2.charAt(i) === '1') {
            newStr = newStr += '1';
        } else {
            newStr = newStr += '0';
        }
    }

    return newStr;
}

// keep this function call here
BitwiseOne(readline());

function BitwiseOne(strArr) {
    var output = '';
    for (var i = 0; i < strArr[0].length; i++) {
        if (strArr[0][i] === '1' || strArr[1][i] === '1') {
            output += '1';
        } else {
            output += '0';
        }
    }
    return output;
}

// keep this function call here
BitwiseOne(readline());

function BitwiseOne(strArr) {

    let newString = '';
    for (let i = 0; i < strArr[0].length; i++) {
        newString += parseInt(strArr[0][i]) || parseInt(strArr[1][i]);
    }
    return newString;
}

// keep this function call here
BitwiseOne(readline());
```

Other Products

Have the function `OtherProducts(arr)` take the array of numbers stored in **`arr`** and return a new list of the products of all the other numbers in the array for each element. For example:

if **`arr`** is [1, 2, 3, 4, 5] then the new array, where each location in the new array is the product of all *other* elements, is [120, 60, 40, 30, 24]. The following calculations were performed to get this answer: [(2*3*4*5), (1*3*4*5), (1*2*4*5), (1*2*3*5), (1*2*3*4)]. You should generate this new array and then return the numbers as a string joined by a hyphen: **120-60-40-30-24**. The array will contain at most 10 elements and at least 1 element of only positive integers.

Examples

Input: [1,4,3]
Output: 12-3-4

Input: [3,1,2,6]
Output: 12-36-18-6

```
function OtherProducts(arr) {
    let holdArray = [];

    arr.forEach((val, ind, theArray) => {
      newArray = Array.from(theArray)
      newArray.splice(ind,1)

      holdArray[ind] = newArray.reduce((val1, val2) => val1 * val2);
    })

    return holdArray.join('-');
}

// keep this function call here
OtherProducts(readline());
```

```
function OtherProducts(arr) {

    let results = [];
    for (let i = 0; i < arr.length; i++) {
        let product = 1;
        for (let j = 0; j < arr.length; j++) {
            if (i !== j) {
                product *= arr[j];
            }
        }
        results.push(product);
    }
    return results.join('-');
}

// keep this function call here
OtherProducts(readline());
```

```
function product(arr) {
    return arr.reduce((a,b) => a * b);
}

function OtherProducts(arr) {
    var products = [];
    for (var i = 0; i < arr.length; i++) {
        var other = arr.slice(0, i).concat(arr.slice(i + 1));
        products.push(product(other));
    }
    return products.join('-');
}

// keep this function call here
OtherProducts(readline());
```

Wave Sorting

Have the function `WaveSorting(arr)` take the array of positive integers stored in **`arr`** and return the string **true** if the numbers can be arranged in a wave pattern: **a1 > a2 < a3 > a4 < a5 > ...**, otherwise return the string **false**. For example, if **`arr`** is: [0, 1, 2, 4, 1, 4], then a possible wave ordering of the numbers is: [2, 0, 4, 1, 4, 1]. So for this input your program should return the string **true**. The input array will always contain at least 2 elements. More examples are given below as sample test cases.

Examples

Input: [0, 1, 2, 4, 1, 1, 1]
Output: false

Input: [0, 4, 22, 4, 14, 4, 2]
Output: true

```
function WaveSorting(arr) {
    //thinking about it, the desired result will be possible so long as we don't have
any one number
    //more times than other numbers to break it up

    //get the total number of numbers
    let length = arr.length;

    //get the number of entries for each number
    let countObj = {};
    arr.forEach(val => {
        if (!countObj[val]) {
```

```
            countObj[val] = 1;
        } else {
            countObj[val]++;
        }
    });

    //make an array of our results, so we can find the max
    let countArr = [];
    for (let key in countObj) {
        countArr.push(countObj[key]);
    }

    //find the max - don't need to use apply() any more!
    let maxCount = Math.max(...countArr);

    return maxCount > length/2 ? false : true;

}

// keep this function call here
WaveSorting(readline());


function WaveSorting(arr) {

    arr = arr.sort((a,b) => a - b).reverse();
    newArr = [];

    let halfLength = Math.floor(arr.length / 2);

    newArr = [];

    for (let i = 0, n = arr.length; i < n; i++) {
        if (i % 2 === 0) {
            newArr.push(arr.splice(0, 1));
        } else {
            // Look and take next element smaller than arr[i]
            for (let j = 1; j < arr.length; j++) {
                if (arr[j] < arr[0]) {
                    newArr.push(arr.splice(j, 1));
                    break;
                }
            }
        }

    }

    // Check if new Arr is wave sorted
    for (let i = 0; i < newArr.length-1; i++) {
        if (i % 2 === 0) {
            // i > i+1 = true
            if (parseInt(newArr[i]) <= parseInt(newArr[i+1])) {
                return false;
            }
```

```
        } else {
            // i < i+1 = true
            if (parseInt(newArr[i]) >= parseInt(newArr[i+1])) {
                return false;
            }
        }
    }
    return true;
}

// keep this function call here
WaveSorting(readline());

// find the count of the most frequent element, which is the mode
function mostFrequent(arr) {
    arr.sort();
    var most = 0;
    var frequency = 1;
    for (var i = 0; i < arr.length; i++) {
        if (arr[i] === arr[i + 1]) {
            frequency++;
            if (frequency > most) {
                most = frequency;
            }
        } else {
            frequency = 1;
        }
    }
    return most;
}

function WaveSorting(arr) {
    // as long as we can put some other number in between the same number, it works
    return mostFrequent(arr) < (arr.length / 2);
}

// keep this function call here
WaveSorting(readline());
```

Array Matching

Have the function `ArrayMatching(strArr)` read the array of strings stored in **`strArr`** which will contain only two elements, both of which will represent an array of positive integers. For example: if **`strArr`** is ["[1, 2, 5, 6]", "[5, 2, 8, 11]"], then both elements in the input represent two integer arrays, and your goal for this challenge is to add the elements in corresponding locations from both arrays. For the example input, your program should do the following additions: [(1 + 5), (2 + 2), (5 + 8), (6 + 11)] which then equals [6, 4, 13, 17]. Your program should finally return this resulting array in a string format with each element separated by a hyphen: **6-4-13-17**.

If the two arrays do not have the same amount of elements, then simply append the remaining elements onto the new array (example shown below). Both arrays will be in the format: [e1, e2, e3, ...] where at least one element will exist in each array.

Examples

Input: ["[5, 2, 3]", "[2, 2, 3, 10, 6]"]
Output: 7-4-6-10-6

Input: ["[1, 2, 1]", "[2, 1, 5, 2]"]
Output: 3-3-6-2

```
function ArrayMatching(strArr) {
    strArr = strArr.map(val => val.replace(/[[]]/g, '')
        .split(/s*,s*/).map(val1 => parseInt(val1, 10)));

    let resArr = [];
    let arr1 = strArr[0];
    let arr2 = strArr[1];
    let length = Math.max(arr1.length, arr2.length);

    for (let i = 0; i < length; i++) {
        if (arr1[i] && arr2[i]) {
            resArr[i] = arr1[i] + arr2[i];
        } else {
            resArr[i] = arr1[i] || arr2[i];
        }
    }
    return resArr.join('-');

}

// keep this function call here
ArrayMatching(readline());

function ArrayMatching(strArr) {

    let numbers1 = strArr[0].substr(1, strArr[0].length-1).split(', ');
    let numbers2 = strArr[1].substr(1, strArr[1].length-1).split(', ');

    let maxLength = (numbers1.length > numbers2.length) ? numbers1.length :
numbers2.length;

    let results = [];

    for (let i = 0; i < maxLength; i++) {
```

```
        let num1 = (i < numbers1.length) ? parseInt(numbers1[i]) : 0;
        let num2 = (i < numbers2.length) ? parseInt(numbers2[i]) : 0;
        results.push(num1 + num2);
    }

    return results.join('-');
}

// keep this function call here
ArrayMatching(readline());

function ArrayMatching(strArr) {
    arr1 = strArr[0].match(/\d+/g).map(Number);
    arr2 = strArr[1].match(/\d+/g).map(Number);

    if (arr1.length > arr2.length) {
        arr2 = arr2.concat(new Array(arr1.length - arr2.length).fill(0));
    } else if (arr1.length < arr2.length) {
        arr1 = arr1.concat(new Array(arr2.length - arr1.length).fill(0));
    }
    var sum = [];
    for (var i = 0; i < arr1.length; i++) {
        sum.push(arr1[i] + arr2[i]);
    }
    return sum.join('-');
}

// keep this function call here
ArrayMatching(readline());
```

Binary Reversal

Have the function `BinaryReversal(str)` take the **`str`** parameter being passed, which will be a positive integer, take its binary representation (padded to the nearest N * 8 bits), reverse that string of bits, and then finally return the new reversed string in decimal form. For example: if **`str`** is "47" then the binary version of this integer is **101111** but we pad it to be **00101111**. Your program should reverse this binary string which then becomes: **11110100** and then finally return the decimal version of this string, which is **244**.

Examples

Input: "213"
Output: 171

Input: "4567"
Output: 60296

```
function BinaryReversal(str) {

    let num = parseInt(str, 10)
        .toString(2);

    let length = num.length;

    //add leading zeroes to make the number an integral number of bytes
    let byteString = `${'0'.repeat(length % 8 === 0 ? 0 :  8 - length % 8)}${num}`;


    let stringByte = byteString.split('')
        .reverse()
        .join('');

    return parseInt(stringByte, 2).toString();

}

// keep this function call here
BinaryReversal(readline());

function BinaryReversal(str) {

    let binary = parseInt(str).toString(2);

    let size = Math.ceil(binary.length/8) * 8;
    while (binary.length < size) {
        binary = '0' + binary;
    }

    let newString = '';
    for (let i = 0; i < binary.length; i++) {
        newString = binary[i] + newString;
    }

    return parseInt(newString, 2);
}

// keep this function call here
BinaryReversal(readline());

function toBinary(str) {
    result = Number(str).toString(2);
    // pad left with leading 0's to make it a multiple of 8 digits...
    if (result.length % 8 !== 0)
        return new Array(8 - (result.length % 8)).fill(0).join('') + result;
    else return result;
}

function toDecimal(str) {
    return parseInt(str, 2);
}
```

```
function BinaryReversal(str) {
    //return toBinary(str);
    var reverse = toBinary(str).split('').reverse().join('');
    return toDecimal(reverse);
}

// keep this function call here
BinaryReversal(readline());
```

Longest Increasing Sequence

Have the function `LongestIncreasingSequence(arr)` take the array of positive integers stored in **`arr`** and return the length of the longest increasing subsequence (LIS). A LIS is a subset of the original list where the numbers are in sorted order, from lowest to highest, and are in increasing order. The sequence does not need to be contiguous or unique, and there can be several different subsequences. For example: if **`arr`** is [4, 3, 5, 1, 6] then a possible LIS is [3, 5, 6], and another is [1, 6]. For this input, your program should return **3** because that is the length of the longest increasing subsequence.

Examples

Input: [9, 9, 4, 2]
Output: 1

Input: [10, 22, 9, 33, 21, 50, 41, 60, 22, 68, 90]
Output: 7

```
function LongestIncreasingSequence(arr) {
    let len = arr.length;
    let arrHolder = [];

    //evaluate each possible combination of numbers
    for (let i = Math.pow(2, len); i < Math.pow(2, len + 1); i++) {

        //numArray is a binary digit, the 0s and 1s representing whether to include
        //a number or not in the combination array.  There will be 2 ^ n combinations
        //to get leading zeros, use numbers from 2^n to 2^n+1, then slice off the
leading 1
        let numArray = i.toString(2).slice(1).split('');

        //hold the selected numbers in the newSeq array
        newSeq = [];

        //populate the newSeq array
```

```
        arr.forEach((val, ind) => {
            if (numArray[ind] === '1') {
            newSeq.push(val);
            }
        });

        // include the newSeq array into arrHolder
        arrHolder.push(newSeq);
    }

    //take out all the arrays that are not ascending (use the ascend() to determine)
    arrHolder = arrHolder.filter(val => ascend(val));

    //replace each passing array with its length
    let arrLength = arrHolder.map(val => val.length);

    //return the largest length value
    return Math.max(...arrLength);
}


function ascend(inputArr) {
    let arrlen = inputArr.length;
    return inputArr.every((val, ind) => {
        if (ind < arrlen - 1) {
            return val < inputArr[ind + 1];
        }
        return true;
    });
}

// keep this function call here
LongestIncreasingSequence(readline());




// https://stackoverflow.com/questions/2631726/how-to-determine-the-longest-
increasing-subsequence-using-dynamic-programming
// Used algorithm from here
function LongestIncreasingSequence(arr) {

    let lis = [arr[0]];

    for (let i = 1; i < arr.length; i++) {
        if (arr[i] > lis[lis.length - 1]) {
            lis.push(arr[i]);
            continue;
        }

        for (let j = 0; j < lis.length; j++) {
            if (lis[j] >= arr[i]) {
                lis[j] = arr[i];
```

```
                break;
            }
        }
    }

    return lis.length;

}

// keep this function call here
LongestIncreasingSequence(readline());


// generates n true/false permutations
// this will be used to generate subsequences - to include this element or not
function permute(n) {
    if (n < 1) return null;
    if (n < 2) return [[true], [false]];
    var previous = permute(n - 1);
    var result = [];
    for (var permutation of previous) {
        result.push(permutation.concat([true]));
        result.push(permutation.concat([false]));
    }
    return result;
}

// get all possible subsequences
function getSubSequences(arr) {
    return permute(arr.length).map(function(permutation) {
        var result = [];
        for (var i = 0; i < permutation.length; i++) {
            if (permutation[i]) {
                result.push(arr[i]);
            }
        }
        return result;
    });
}

function increasing(arr) {
    return arr.every(function(value, index, arr) {
        var prev = (index === 0) ? 0: arr[index - 1];
       return prev < value;
    });
}

function LongestIncreasingSequence(arr) {
    var longest = 0;

    var subSequences = getSubSequences(arr);
    for (var subSequence of subSequences) {
        if(increasing(subSequence) && subSequence.length > longest) {
            longest = subSequence.length;
```

```
        }
    }
    return longest;
}

// keep this function call here
LongestIncreasingSequence(readline());
```

Even Pairs

Have the function `EvenPairs(str)` take the **`str`** parameter being passed and determine if a pair of adjacent even numbers exists anywhere in the string. If a pair exists, return the string **true**, otherwise return **false**. For example: if **`str`** is "f178svg3k19k46" then there are two even numbers at the end of the string, "46" so your program should return the string **true**. Another example: if **`str`** is "7r5gg812" then the pair is "812" (8 and 12) so your program should return the string **true**.

Examples

Input: "3gy41d216"
Output: true

Input: "f09r27i8e67"
Output: false

```
function EvenPairs(str) {

  var regEx = /[24680]d*[24680]/

  return regEx.test(str);

}

// keep this function call here
EvenPairs(readline());
```

```
function EvenPairs(str) {

    const DIGITS = '0123456789';

    let numberGroups = [];

    // Get groups of digits
    for (let i = 0, digitStr = ''; i < str.length; i++) {
        let isDigit = DIGITS.includes(str[i]);
```

```
        if (isDigit) {
            digitStr += str[i];
        }

        if (digitStr.length > 0 && (i === str.length - 1 || !isDigit)) {
            numberGroups.push(digitStr);
            digitStr = '';

        }
    }

    // Only care about group length > 1
    numberGroups = numberGroups.filter(v => v.length > 1);



    // Loop over all "groups"
    for (let i = 0; i < numberGroups.length; i++) {
        // Nested loops for each group
        for (let j = 0; j < numberGroups[i].length; j++) {
            for (let k = j + 1; k < numberGroups[i].length; k++) {
                let str1 = numberGroups[i].substr(0, j+1);
                let str2 = numberGroups[i].substr(j+1, k);
                if (parseInt(str1) % 2 === 0 && parseInt(str2) % 2 === 0) {
                    return true;
                }
            }
        }
    }
    return false;
}

// keep this function call here
EvenPairs(readline());



// see if there are more than two pairs of even numbers
function hasEvenPairs(number) {
    // non-greedy match of even numbers
    var result = number.toString().match(/\d*?[24680]/g);
    return (result === null) ? false : (result.length >= 2);
}

function EvenPairs(str) {
    var numbers = str.match(/\d+/g);
    for (var number of numbers) {
        if (hasEvenPairs(number)) return true;
    }
    return false;
}

// keep this function call here
EvenPairs(readline());
```

Next Palindrome

Have the function `NextPalindrome(num)` take the `num` parameter being passed and return the next largest palindromic number. The input can be any positive integer. For example: if `num` is 24, then your program should return **33** because that is the next largest number that is a palindrome.

Examples

Input: 2
Output: 3

Input: 180
Output: 181

```
function NextPalindrome(num) {

  let count = num + 1;

  while (true) {
    numString = count.toString();
    revString = numString.split('')
      .reverse()
      .join('');
    if (revString === numString) return parseInt(numString, 10);
    count++;
  }
}

// keep this function call here
NextPalindrome(readline());
```

```
function NextPalindrome(num) {

    let nextPalindrome = null;
    for (let i = num + 1; ; i++) {
        let string = i.toString();
        if (isPalindrome(string)) {
            nextPalindrome = i;
            break;
        }
    }
    return nextPalindrome;

    function isPalindrome(str) {
```

```
        for (let i = 0, max = Math.floor(str.length/2); i < max; i++) {
            if (str[i] !== str[str.length-1-i]) {
                return false;
            }
        }
        return true;
    }
}

// keep this function call here
NextPalindrome(readline());

function isPalindrome(num) {
    var numStr = num.toString();
    return numStr.split('').reverse().join('') === numStr;
}

function NextPalindrome(num) {
    var nextNum = num + 1;
    while (!isPalindrome(nextNum)) {
        nextNum++;
    }
    return nextNum;
}

// keep this function call here
NextPalindrome(readline());
```

Largest Pair

Have the function `LargestPair(num)` take the **num** parameter being passed and determine the largest double digit number within the whole number. For example: if **num** is **4759472** then your program should return **94** because that is the largest double digit number. The input will always contain at least two positive digits.

Examples

Input: 453857
Output: 85

Input: 363223311
Output: 63

```
function LargestPair(num) {

    let numStr = num.toString();

    let numArr = numStr.split('')
        .map(val => parseInt(val, 10));

    //get rid of the final digit, in case it is the largest;
    numArr.pop();

    let maxNum = Math.max(...numArr);

    let regEx = new RegExp(`${maxNum}\d`, 'g');

    let matches = numStr.match(regEx);

    return matches.sort((a, b) => a - b).pop();
}

// keep this function call here
LargestPair(readline());

function LargestPair(num) {

    num = num.toString();

    let largestNum = 0;
    for (let i = 1; i < num.length; i++) {
        let testNum = parseInt(num[i-1] + num[i]);
        if (testNum > largestNum) {
            largestNum = testNum;
        }
    }
    return largestNum;
```

```
}

// keep this function call here
LargestPair(readline());

function LargestPair(num) {
    var max = 0;
    var numStr = num.toString();

    // sliding window with size 2
    for (var i = 0; i < numStr.length - 1; i++) {
        var testNum = Number(numStr.slice(i, i + 2));
        if (testNum > max) {
            max = testNum;
        }
    }
    return max;
}

// keep this function call here
LargestPair(readline());
```

Nonrepeating Character

Have the function `NonrepeatingCharacter(str)` take the **`str`** parameter being passed, which will contain only alphabetic characters and spaces, and return the first non-repeating character. For example: if **`str`** is "agettkgaeee" then your program should return **k**. The string will always contain at least one character and there will always be at least one non-repeating character.

Examples

Input: "abcdef"
Output: a

Input: "hello world hi hey"
Output: w

```
function NonrepeatingCharacter(str) {
  let len = str.length;
  let countObj = {}

  for (let i = 0; i < len; i++) {
    if (countObj[str[i]]) {
      countObj[str[i]]++;
    }
    else {
```

```
      countObj[str[i]] = 1;
    }
  }

  for (let j = 0; j < len; j++) {
    if (countObj[str[j]] === 1) return str[j];
  }
}

// keep this function call here
NonrepeatingCharacter(readline());

function NonrepeatingCharacter(str) {

    let repeatingChars = '';
    let result = '';
    for (let i = 0; i < str.length; i++) {
        let repeating = false;
        for (let j = i+1; j < str.length; j++) {
            if (str[i] === str[j] || repeatingChars.includes(str[i])) {
                repeating = true;
                repeatingChars += str[i];
                break;
            }
        }
        if (!repeating) {
            result = str[i];
            break;
        }
    }
    return result;
}

// keep this function call here
NonrepeatingCharacter(readline());

function NonrepeatingCharacter(str) {
    str = str.replace(/\s+/g, '');
    var counts = {};

    // count each letter
    for (var i = 0; i < str.length; i++) {
        if (str[i] in counts) {
            counts[str[i]]++;
        } else {
            counts[str[i]] = 1;
        }
    }

    // return the first letter with count of 1
    for (i = 0; i < str.length; i++) {
        if (counts[str[i]] === 1) return str[i];
    }
}
```

```
// keep this function call here
NonrepeatingCharacter(readline());
```

Two Sum

Have the function `TwoSum(arr)` take the array of integers stored in **`arr`**, and determine if any two numbers (excluding the first element) in the array can sum up to the first element in the array. For example: if **`arr`** is [7, 3, 5, 2, -4, 8, 11], then there are actually two pairs that sum to the number 7: [5, 2] and [-4, 11]. Your program should return all pairs, with the numbers separated by a comma, in the order the first number appears in the array. Pairs should be separated by a space. So for the example above, your program would return: **5,2 -4,11**

If there are no two numbers that sum to the first element in the array, return **-1**

Examples

Input: [17, 4, 5, 6, 10, 11, 4, -3, -5, 3, 15, 2, 7]
Output: 6,11 10,7 15,2

Input: [7, 6, 4, 1, 7, -2, 3, 12]
Output: 6,1 4,3

```
function TwoSum(arr) {
  let target = arr.shift();
  let len = arr.length;
  let holdArr = [];

  for (let i = 0; i < len; i++) {
    for (let j = i + 1; j < len; j++) {
      if (arr[i] + arr[j] === target) {
        holdArr.push(`${arr[i].toString()},${arr[j].toString()}`);
        break;
      }
    }
  }
  return holdArr.length ? holdArr.join(' ') : -1;
}

// keep this function call here
TwoSum(readline());
```

```
function TwoSum(arr) {

    const answer = arr.shift(arr);
```

```
    const history = new Set();
    const matches = [];

    // Without reverse() here, the final results will be in the order
    // the second number appears in the array, but we want it to be
    // ordered by the first number.
    arr.reverse();

    arr.forEach(item => {
      const compliment = answer - item;
      if (history.has(compliment)) {
        matches.push([item, compliment]);
      } else {
        history.add(item);
      }
    });

    // The matches were pushed onto the array in reverse order, so
    // now we need to switch them back.
    matches.reverse();

    return (matches.length === 0) ? -1 : matches.map(m => m.join(',')).join(' ');
}

// keep this function call here
TwoSum(readline());

// find pairs that sums up to the given number
function findPairs(arr, sum) {
    var pairs = [];
    for (var i = 0; i < arr.length; i++) {
        for (var j = i + 1; j < arr.length; j++) {
            if (arr[i] + arr[j] === sum)
                pairs.push([arr[i], arr[j]]);
        }
    }
    return pairs;
}

function TwoSum(arr) {
    var pairs = [];
    var sum = arr[0];
    var rest = arr.slice(1);
    pairs = findPairs(rest, sum);

    return (pairs.length === 0) ? -1 : pairs.map(pair => pair.join(',')).join(' ');
}

// keep this function call here
TwoSum(readline());
```

Bitwise Two

Have the function `BitwiseTwo(strArr)` take the array of strings stored in **`strArr`**, which will only contain two strings of equal length that represent binary numbers, and return a final binary string that performed the bitwise AND operation on both strings. A bitwise AND operation places a 1 in the new string where there is a 1 in both locations in the binary strings, otherwise it places a 0 in that spot. For example: if **`strArr`** is ["10111", "01101"] then your program should return the string "00101"

Examples

Input: ["100", "000"]
Output: 000

Input: ["10100", "11100"]
Output: 10100

```
function BitwiseTwo(strArr) {
  let num1 = strArr[0];
  let num2 = strArr[1];
  let len = strArr[0].length;
  let resStr = '';

  for (let i = 0; i < len; i++) {
    if (num1[i] === '1' && num2[i] === '1') {
      resStr += '1';
      continue;
    }
    else {
      resStr += '0';
      continue;
    }
  }
  return resStr;
}

// keep this function call here
BitwiseTwo(readline());

function BitwiseTwo(strArr) {

    let newStr = '';
    for (let i = 0; i < strArr[0].length; i++) {
        newStr += (strArr[0][i] === strArr[1][i] && strArr[0][i] !== '0') ? '1' :
'0';
    }
    return newStr;
}
```

```
// keep this function call here
BitwiseTwo(readline());

function BitwiseTwo(strArr) {
    var result = '';
    for (var i = 0; i < strArr[0].length; i++) {
        if (strArr[0][i] === '1' && strArr[1][i] === '1') {
            result += '1';
        } else {
            result += '0';
        }
    }
    return result;
}

// keep this function call here
BitwiseTwo(readline());
```

Power Set Count

Have the function `PowerSetCount(arr)` take the array of integers stored in **`arr`**, and return the length of the power set (the number of all possible sets) that can be generated. For example: if **`arr`** is [1, 2, 3], then the following sets form the power set:

```
[]
[1]
[2]
[3]
[1, 2]
[1, 3]
[2, 3]
[1, 2, 3]
```

You can see above all possible sets, along with the empty set, are generated. Therefore, for this input, your program should return **8**.

Examples

Input: [1, 2, 3, 4]
Output: 16

Input: [5, 6]

Output: 4

```
function PowerSetCount(arr) {
  let len = arr.length;
  return Math.pow(2, len);
}

// keep this function call here
PowerSetCount(readline());
```

```
function PowerSetCount(arr) {
    return Math.pow(2, arr.length);
}

// keep this function call here
PowerSetCount(readline());
```

```
function PowerSetCount(arr) {
    return Math.pow(2, arr.length);
}

// keep this function call here
PowerSetCount(readline());
```

Product Digits

Have the function `ProductDigits(num)` take the **num** parameter being passed which will be a positive integer, and determine the least amount of digits you need to multiply to produce it. For example: if **num** is **24** then you can multiply 8 by 3 which produces 24, so your program should return **2** because there is a total of only 2 digits that are needed. Another example: if **num** is 90, you can multiply 10 * 9, so in this case your program should output **3** because you cannot reach 90 without using a total of 3 digits in your multiplication.

Examples

Input: 6

Output: 2

Input: 23

Output: 3

```
function ProductDigits(num) {
      let pivot = Math.sqrt(num);
      let value = num.toString().length + 1;
      for (let i = 1; i <= pivot; i++) {
```

```
            if (num % i === 0) {
                  let maxFactor = i;
                  maxCompFactor = num / maxFactor;
                  maxFactorString = maxFactor.toString();
                  maxCompFactorString = maxCompFactor.toString();
                  let totalLength = maxFactorString.length +
maxCompFactorString.length;
                  if (totalLength < value) {
                        value = totalLength;
                  }
            }
      }
      return value;
}

// keep this function call here
ProductDigits(readline());

function ProductDigits(num) {

    let pf = primeFactors(num);

    if (pf.length === 1) {
        return pf[0].toString().length + 1;
    }

    let factors = [];

    for (let divider = 0; divider < pf.length; divider ++) {
        let left = pf.slice(0, divider);
        let right = pf.slice(divider, pf.length);

        let leftProduct = left.reduce((product, value) => product *= value, 1);
        let rightProduct = right.reduce((product, value) => product *= value, 1);
        factors.push([leftProduct, rightProduct]);
    }

    return factors.map(factor => factor.join('').split('').length).reduce((max, val)
=> val < max ? val : max, Number.MAX_SAFE_INTEGER);

    function primeFactors(num) {
        if (num === 1) return 1;

        let pf = [];

        for (let i = 2; i <= num; i++) {
            if (num % i === 0) {
                pf.push(i);
                num /= i;
                i = 1;
            }
        }
        return pf;
    }
```

```
}

// keep this function call here
ProductDigits(readline());

function getDivisors(num) {
    var divisors = [1];
    for (var i = 2; i <= num / 2; i++) {
        if (num % i === 0) {
            divisors.push(i);
        }
    }
    divisors.push(num);
    return divisors;
}

function ProductDigits(num) {
    var divisors = getDivisors(num);
    var pairs = [];

    for (var i = 0; i < divisors.length / 2; i++) {
        pairs.push([divisors[i], divisors[(divisors.length - 1) - i]]);
    }

    return pairs.map(pair => ('' + pair[0] + pair[1]).length).sort()[0];
}

// keep this function call here
ProductDigits(readline());
```

Palindrome Creator

Have the function `PalindromeCreator(str)` take the **`str`** parameter being passed and determine if it is possible to create a palindromic string of **minimum length 3 characters** by removing 1 or 2 characters. For example: if **`str`** is "abjchba" then you can remove the characters **jc** to produce "abhba" which is a palindrome. For this example your program should return the two characters that were removed with no delimiter and in the order they appear in the string, so **jc**. If 1 or 2 characters cannot be removed to produce a palindrome, then return the string **not possible**. If the input string is already a palindrome, your program should return the string **palindrome**.

The input will only contain lowercase alphabetic characters. Your program should always attempt to create the longest palindromic substring by removing 1 or 2 characters (see second sample test case as an example). The 2 characters you remove do not have to be adjacent in the string.

Examples

Input: "mmop"
Output: not possible

Input: "kjjjhjjj"
Output: k

```
function PalindromeCreator(str) {
	let len = str.length;
	//test to see if it is a Palindrome already
	if (isPalindrome(str)) {
		return 'palindrome';
	}
	for (let i = 0; i < len; i++) {
		let testArray = str.split('');
		let res = testArray.splice(i, 1);
		let newString = testArray.join('');
		if (isPalindrome(newString)) {
			console.log('one');
			return res.join('');
		}
	}
	for (let i = 0; i < len; i++) {
		let res = [];
		for (let j = i; j < len - 1; j++) {
			let testArray = str.split('');
			res[0] = testArray.splice(i, 1);
			res[1] = testArray.splice(j, 1);
			let newString = testArray.join('');
			if(isPalindrome(newString)) {
				return res.join('');
			}
		}
	}
	return 'not possible';
}

//----------------helpers-------------------------

function isPalindrome(str) {
	let newStr = str.split('').reverse().join('');
	if (newStr === str) {
		return true;
	}
	return false;
}

// keep this function call here
PalindromeCreator(readline());
```

```
function PalindromeCreator(str) {

    if (isPalindrome(str)) {
        return 'palindrome';
    }

    // Create combos
    let combos = [];
    for (let i = 0, max = Math.pow(2, str.length); i < max; i++) {
        let combo = i.toString(2);

        // Filter for zero, one, or two 1s (1 represents removal)
        let count = 0;
        for (let j = 0; j < combo.length; j++) {
            if (combo[j] === '1') {
                count++;
            }
        }
        // Too big, ignore and go to next try!
        if (count > 2) {
            continue;
        }

        // Pad good combo
        while (combo.length < str.length) {
            combo = '0' + combo;
        }
        combos.push(combo);
    }

    let palindromeCombos = [];

    // Try removal combos
    for (let i = 0; i < combos.length; i++) {
        let tryString = '';
        for (let j = 0; j < combos[i].length; j++) {
            tryString += (combos[i][j] === '1') ? '' : str[j];
        }

        if (tryString.length >= 3 && isPalindrome(tryString)) {
            palindromeCombos.push(combos[i]);
        }
    }

    if (palindromeCombos.length === 0) {
        return 'not possible';
    }

    // Sort so first two letters found will be returned first for when
    // there are multiple values
    palindromeCombos.sort(function(a,b){
        return parseInt(a,2) < parseInt(b,2);
```

```
    });

    // Find and return shortest removal <=> longest palindrome
    let shortestCount = null;
    let shortestCombo = '';
    palindromeCombos.forEach(function(combo){
        let count = 0;
        for (let i = 0; i < combo.length; i++) {
            if (combo[i] === '1') {
                count++;
            }
        }
        if (shortestCount === null || count < shortestCount) {
            shortestCount = count;
            shortestCombo = combo;
        }
    });


    let result = '';
    for (let i = 0; i < str.length; i++) {
        if (shortestCombo[i] === '1') {
            result += str[i];
        }
    }

    return result;

    function isPalindrome(str) {
        return (str === str.split('').reverse().join('')) ? true : false;
    }

}

// keep this function call here
PalindromeCreator(readline());


function isPalindrome(str) {
    return str === str.split('').reverse().join('');
}

function PalindromeCreator(str) {
    if (isPalindrome(str)) return 'palindrome';

    for (var i = 0; i < str.length; i++) {
        // remove one character at position i
        var newStr = str.slice(0, i).concat(str.slice(i + 1));
        if (isPalindrome(newStr)) {
            return str[i];
        }
    }

    for (var i = 0; i < str.length; i++) {
```

```
        // remove two characters at position i and j, where (i < j)
        for (var j = i + 1; j < str.length; j++) {
            var newStr2 = str.slice(0, i).concat(str.slice(i + 1,
j)).concat(str.slice(j + 1));
            if (isPalindrome(newStr2)) {
                return str[i] + str[j];
            }
        }
    }
    return "not possible";
}

// keep this function call here
PalindromeCreator(readline());
```

Basic Roman Numerals

Have the function `BasicRomanNumerals(str)` read `str` which will be a string of Roman numerals. The numerals being used are: **I** for 1, **V** for 5, **X** for 10, **L** for 50, **C** for 100, **D** for 500 and **M** for 1000. In Roman numerals, to create a number like 11 you simply add a 1 after the 10, so you get **XI**. But to create a number like 19, you use the subtraction notation which is to add an I before an X or V (or add an X before an L or C). So 19 in Roman numerals is **XIX**.

The goal of your program is to return the decimal equivalent of the Roman numeral given. For example: if `str` is "XXIV" your program should return **24**

Examples

Input: "IV"
Output: 4

Input: "XLVI"
Output: 46

```
function BasicRomanNumerals(str) {
      let letterObj = {
            I: 1,
            V: 5,
            X: 10,
            L: 50,
            C: 100,
            D: 500,
            M: 1000
      }
      let res = 0;
      let len = str.length;
```

```
      for (let i = 0; i < len; i++) {
            if (!letterObj[str[i + 1]] || letterObj[str[i]] >= letterObj[str[i +
1]]) {
                  res += letterObj[str[i]];
            } else {
                  res += (letterObj[str[i + 1]] - letterObj[str[i]]);
                  i++;
            }
      }
      return res;
}

// keep this function call here
BasicRomanNumerals(readline());


function BasicRomanNumerals(str) {

    /*
        Symbol      I      V      X      L      C      D      M
        Value1      5      10     50     100    500    1,000
    */

    const ROMAN_I = 1;
    const ROMAN_V = 5;
    const ROMAN_X = 10;
    const ROMAN_L = 50;
    const ROMAN_C = 100;
    const ROMAN_D = 500;
    const ROMAN_M = 1000;

    let sum = 0;

    for (let i = 0; i < str.length; i++) {
        let symbol = str[i];
        let nextSymbol = (i+1 >= str.length) ? null : str[i+1];

        switch(symbol) {
            case 'I':
                if (nextSymbol === 'V') {
                    sum += ROMAN_V - ROMAN_I;
                    i++;
                } else if (nextSymbol === 'X') {
                    sum += ROMAN_X - ROMAN_I;
                    i++;
                } else {
                    sum += ROMAN_I;
                }
                break;
            case 'V':
                sum += ROMAN_V;
                break;
            case 'X':
                if (nextSymbol === 'L') {
```

```
                    sum += ROMAN_L - ROMAN_X;
                    i++;
                } else if (nextSymbol === 'C') {
                    sum += ROMAN_C - ROMAN_X;
                    i++;
                } else {
                    sum += ROMAN_X;
                }
                break;
            case 'L':
                sum += ROMAN_L;
                break;
            case 'C':
                if (nextSymbol === 'D') {
                    sum += ROMAN_D - ROMAN_C;
                    i++;
                } else if (nextSymbol === 'M') {
                    sum += ROMAN_M - ROMAN_C;
                    i++;
                } else {
                    sum += ROMAN_C;
                }
                break;
            case 'D':
                sum += ROMAN_D;
                break;
            case 'M':
                sum += ROMAN_M;
                break;
            default:
                // Illegal char or space
                break;
        }
    }

    return sum;
}

// keep this function call here
BasicRomanNumerals(readline());

function BasicRomanNumerals(str) {
    var decimals = [1000, 900, 500, 400, 100, 90, 50, 40, 10, 9, 5, 4, 1];
    var romans = ['M', 'CM', 'D', 'CD', 'C', 'XC', 'L', 'XL', 'X', 'IX', 'V', 'IV',
'I'];
    var output = 0;

    for (var i = 0; i < romans.length; i++) {
        // keep running while there is a match at the beginning
        while (str.indexOf(romans[i]) === 0) {
            output += decimals[i];
            str = str.replace(romans[i], '');
        }
    }
```

```
    return output;
}

// keep this function call here
BasicRomanNumerals(readline());
```

Food Distribution

Have the function `FoodDistribution(arr)` read the array of numbers stored in **`arr`** which will represent the hunger level of different people ranging from 0 to 5 (0 meaning not hungry at all, 5 meaning very hungry). You will also have N sandwiches to give out which will range from 1 to 20. The format of the array will be [N, h1, h2, h3, ...] where N represents the number of sandwiches you have and the rest of the array will represent the hunger levels of different people. Your goal is to minimize the hunger difference between each pair of people in the array using the sandwiches you have available.

For example: if **`arr`** is [5, 3, 1, 2, 1], this means you have 5 sandwiches to give out. You can distribute them in the following order to the people: 2, 0, 1, 0. Giving these sandwiches to the people their hunger levels now become: [1, 1, 1, 1]. The difference between each pair of people is now 0, the total is also 0, so your program should return **0**. Note: You may not have to give out all, or even any, of your sandwiches to produce a minimized difference.

Another example: if **`arr`** is [4, 5, 2, 3, 1, 0] then you can distribute the sandwiches in the following order: [3, 0, 1, 0, 0] which makes all the hunger levels the following: [2, 2, 2, 1, 0]. The differences between each pair of people is now: 0, 0, 1, 1 and so your program should return the final minimized difference of **2**.

Examples

Input: [5, 2, 3, 4, 5]
Output: 1

Input: [3, 2, 1, 0, 4, 1, 0]
Output: 4

```
function FoodDistribution(arr) {
       let treats = arr.shift();
       let myArray = arr.slice(0);
       let arrMin = arr.sort((val1, val2) => val2 - val1).pop()
       let len = myArray.length;
```

```
	//check to see if we have enough treats to get everybody to the current best
level
	let testCount = myArray.reduce((val1, val2) => {
		return val1 + val2 - arrMin;
	}, 0);
	if (testCount <= treats) {
		return 0;
	}

	let valQuantArr = objectify(myArray);

	for (let i = 1; i < 25; i++) {
		let arrayLen = valQuantArr.length;
		let resp = flattenMid(valQuantArr, treats, i);
		valQuantArr = resp[0];
		arrayLen = valQuantArr.length;
		treats = resp[1];
		while (valQuantArr[0].quant <= i && valQuantArr[0].value >
valQuantArr[1].value && treats >= i) {
			if (valQuantArr[0].quant <= treats) {
				valQuantArr[0].value--;
				treats -= valQuantArr[0].quant;
				valQuantArr = objectify(valQuantArr);
				arrayLen = valQuantArr.length;
			}
		}

		while (valQuantArr[arrayLen - 1].quant <= i && valQuantArr[arrayLen -
1].value > valQuantArr[arrayLen - 2].value && treats >= i) {
			if (valQuantArr[arrayLen - 1].quant <= treats) {
				valQuantArr[arrayLen - 1].value--;
				treats -= valQuantArr[arrayLen - 1].quant;
				valQuantArr = objectify(valQuantArr);
				arrayLen = valQuantArr.length;
			}
		}
	}

	let count = 0;
	for (let i = 0, len = valQuantArr.length; i < len - 1; i++) {
		count += Math.abs(valQuantArr[i].value - valQuantArr[i + 1].value);
	}
	return count;
}

//-----------------helpers----------------------
flattenMid = (arr, treats, q) => {
	let index = 0;
	while (treats > 0 && index > -1) {
		index = arr.findIndex((val, ind) => {
			return val.quant <= q && ind >= 1 && ind < arr.length - 1 &&
val.value > arr[ind - 1].value && val.value > arr[ind + 1].value;
		});
		if (index >= 0) {
			arr[index].value --;
```

```
                        treats -= q;
                }
        }
        return [objectify(arr), treats];
}

//turns an array into an object with the value equal to the number, and the
//quant being the number of times it occurs in a row
objectify = (array) => {
        //if it is the array of numbers
        if (typeof array[0] === 'number') {
                let target = [];
                let counter = 0;
                for (let i = 0, len = array.length; i < len; i++) {
                        let val = array[i];
                        counter++;
                        if (array[i] === array[i + 1]) {
                                continue;
                        } else {
                                target.push({
                                        value: array[i],
                                        quant: counter
                                });
                                counter = 0;
                        }
                }
                return target;
        } else {
                //if it is an array of objects, turn it into an array of numbers, and
                //then run it through the objectify method
                let targetArray = [];
                array.forEach (val => {
                        while (val.quant) {
                                targetArray.push (val.value);
                                val.quant--;
                        }
                });
                return objectify(targetArray);
        }
};


// keep this function call here
FoodDistribution(readline());


function FoodDistribution(arr) {

    let sandwiches = parseInt(arr.shift());

    // Generate combos, limitation of this method, max 32 sandwiches
    let combos = [];
    for (let i = 0, max = Math.pow(sandwiches+1, arr.length); i < max; i++) {
        let combo = i.toString(sandwiches+1);
```

```
        // Sum of digits (sandwiches) in combo
        let comboSum = parseInt(combo.split('').reduce((accumulator, currentValue) =>
accumulator + parseInt(currentValue, sandwiches+1), 0));

        // Too many sandwiches
        if (comboSum > sandwiches) {
            continue;
        }

        // Also can add a check here to remove sandwiches that would put hunger < 0

        // Pad combos of good combos
        while (combo.length < arr.length) {
            combo = '0' + combo;
        }

        if (comboSum <= sandwiches) {
            combos.push(combo);
        }
    }

    // Find lowest hunger difference
    let lowestHungerDifference = null;
    combos.forEach(function(combo){
        let testArr = arr.slice();
        for (let i = 0; i < combo.length; i++) {
            testArr[i] -= combo[i];
        }
        let diff = getHungerDifference(testArr);
        lowestHungerDifference = (lowestHungerDifference === null || diff <
lowestHungerDifference) ? diff : lowestHungerDifference;
    });

    return lowestHungerDifference;


    function getHungerDifference(arr){
        let diff = 0;
        for (let i = 1; i < arr.length; i++) {
            diff += Math.abs(arr[i] - arr[i-1]);
        }
        return diff;
    }

}

// keep this function call here
FoodDistribution(readline());


function FoodDistribution(arr) {
    var N = arr[0];
```

```
    var hungerLevels = arr.slice(1);

    while (N > 0) {
        var maxDifference = 0;
        // the index to be fed a sandwich next
        var mostNeeded = -1;
        for (var i = 0; i < hungerLevels.length - 1; i++) {
            var difference = Math.abs(hungerLevels[i + 1] - hungerLevels[i]);
            if (difference > maxDifference) {
                maxDifference = difference;
                mostNeeded = (hungerLevels[i + 1] > hungerLevels[i]) ? (i + 1) : i;
            }
        }
        // now we know who needs the sandwich so badly. give it away
        if (mostNeeded === -1) {
            // the adjacent differences are all 0's, so stop giving the sandwiches
away
            return 0;
        } else {
            hungerLevels[mostNeeded] -= 1;
            N--;
        }
    }

    // calculate the sum of adjacent differences
    var sumOfDifferences = 0;
    for (var i = 0; i < hungerLevels.length - 1; i++) {
        sumOfDifferences += Math.abs(hungerLevels[i + 1] - hungerLevels[i]);
    }

    return sumOfDifferences;
}

// keep this function call here
FoodDistribution(readline());
```

Three Sum

Have the function `ThreeSum(arr)` take the array of integers stored in **`arr`**, and determine if any three distinct numbers (excluding the first element) in the array can sum up to the first element in the array. For example: if **`arr`** is [8, 2, 1, 4, 10, 5, -1, -1] then there are actually three sets of triplets that sum to the number 8: [2, 1, 5], [4, 5, -1] and [10, -1, -1]. Your program should return the string **true** if 3 distinct elements sum to the first element, otherwise your program should return the string **false**. The input array will always contain at least 4 elements.

Examples

Input: [10, 2, 3, 1, 5, 3, 1, 4, -4, -3, -2]
Output: true

Input: [12, 3, 1, -5, -4, 7]

Output: false

```
function ThreeSum(arr) {
	let target = arr.shift();
	let len = arr.length;
	for (let i = 0; i < len - 2; i++) {
		for (let j = i + 1; j < len - 1; j++) {
			for (let k = i + 2; k < len; k++) {
				if (arr[i] + arr[j] + arr[k] === target) {
					return "true";
				}
			}
		}
	}
	return "false"
}

// keep this function call here
ThreeSum(readline());

function ThreeSum(arr) {

    let answer = parseInt(arr.shift());

    // Generate combos
    let combos = [];
    for (let i = 0, max = Math.pow(2, arr.length); i < max; i++) {
        let combo = i.toString(2);


        let digitsSum = parseInt(combo.split('').reduce((accumulator,value) =>
accumulator + parseInt(value), 0));

        if (digitsSum !== 3) {
            continue;
        }

        // Pad digits
        while (combo.length < arr.length) {
            combo = '0' + combo;
        }
        combos.push(combo);
    }



    // Test combos
    let goodCombos = [];
    combos.forEach(function(combo){
        let sum = 0;
        for (let i = 0; i < combo.length; i++) {
            if (combo[i] === '1') {
```

```
                sum += parseInt(arr[i]);
            }
        }
        if (sum === answer) {
            goodCombos.push(combo);
        }
    });

    return (goodCombos.length > 0) ? true : false;

}

// keep this function call here
ThreeSum(readline());

function ThreeSum(arr) {
    var sum = arr[0];
    var numbers = arr.slice(1);

    for (var i = 0; i < numbers.length; i++) {
        for (var j = i + 1; j < numbers.length; j++) {
            for (var k = j + 1; k < numbers.length; k++) {
                if (numbers[i] + numbers[j] + numbers[k] === sum) {
                    return 'true';
                }
            }
        }
    }
    return 'false';
}

// keep this function call here
ThreeSum(readline());
```

Correct Path

Have the function `CorrectPath(str)` read the **`str`** parameter being passed, which will represent the movements made in a 5x5 grid of cells starting from the top left position. The characters in the input string will be entirely composed of: **r, l, u, d, ?**. Each of the characters stand for the direction to take within the grid, for example: r = right, l = left, u = up, d = down. Your goal is to determine what characters the question marks should be in order for a path to be created to go from the top left of the grid all the way to the bottom right without touching previously travelled on cells in the grid.

For example: if **`str`** is "r?d?drdd" then your program should output the final correct string that will allow a path to be formed from the top left of a 5x5 grid to the bottom right. For this input,

your program should therefore return the string **rrdrdrdd**. There will only ever be one correct path and there will always be at least one question mark within the input string.

Examples

Input: "???rrurdr?"
Output: dddrrurdrd

Input: "drdr??rrddd?"
Output: drdruurrdddd

```
function CorrectPath(str) {
    //create an array to hold the positions of the question marks
       let blankArray = [];
       //put the position of the question marks into the array
       str.split('').forEach((val, ind) => {
              if (val === '?') {
                     blankArray.push(ind);
              }
       });

       let num = blankArray.length;

       //we are going to try each possibility until we find one that works,  This
will be 4^num permutations
       let total = Math.pow(4, num);

       for (let i = 0; i < total; i++) {
           //go through each permutation, first creating a representative number,
then making the path, then testing it
              let numString = (i + total).toString(4).slice(1);
              let currentPath = createPath(str, blankArray, numString);
              if (isPathGood(currentPath)) {
                     return currentPath;
              }
       }
}


isPathGood = (str) => {
       //create our empty array
       let testArray = []
       for (let i = 0; i < 5; i++) {
              testArray.push([0, 0, 0, 0, 0])
       }

       let len = str.length;
       let currentLoc = [0, 0];

       for (let i = 0; i < len; i++) {
```

```
            //mark our current square as visited
               testArray[currentLoc[0]][currentLoc[1]] = 1;
               //alter the position based on the next letter
               let newLoc = currentLoc.slice(0);
               switch (str[i]) {
                     case 'u':
                           newLoc[0]--;
                           break;
                     case 'd':
                           newLoc[0]++;
                           break;
                     case 'r':
                           newLoc[1]++;
                           break;
                     case 'l':
                           newLoc[1]--;
                           break;
               }
               //quit if we have gone off the board
               if (newLoc.includes (-1) || newLoc.includes (5)) {
                           return false;
               }
               //quit if we are on a previously visited space
               if (testArray[newLoc[0]][newLoc[1]] === 1) {
                     return false;
               }
               //return true if we are at the target square on our last go
               if (newLoc[0] === 4 && newLoc[1] === 4 && i === len - 1) {
                     return true;
               }
               //update our location for the next loop;
               currentLoc = newLoc;
      }
      return false;
};

createPath = (str, blanks, num) => {
      let moveArray = ['r', 'l', 'u', 'd'];
      strArr = str.split('');
      blanks.forEach((val, ind) => {
            strArr.splice(val, 1, moveArray[num[ind]]);
      });
      return strArr.join('');
};

// keep this function call here
CorrectPath(readline());

function CorrectPath(str) {


    let numQmarks = str.split('').reduce((a,v) => a + (v === '?' | 0), 0);

    // Generate combos, 0-r, 1-d, 2-l, 3-u
```

```
    let combos = [];
    for (let i = 0, max = Math.pow(4, numQmarks); i < max; i++) {
        let combo = i.toString(4);
        // Pad
        while (combo.length < numQmarks) {
            combo = '0' + combo;
        }
        combos.push(combo);
    }

    let goodPaths = [];

    // Try paths
    combos.forEach(function(combo){
        let comboArray = combo.split('');
        let tryPath = '';
        for (let i = 0; i < str.length; i++) {
            if (str[i] === '?') {
                let direction = comboArray.shift();

                switch (direction) {
                    case '0':  // right
                        tryPath += 'r';
                        break;
                    case '1':  // down
                        tryPath += 'd';
                        break;
                    case '2':  // left
                        tryPath += 'l';
                        break;
                    case '3':  // up
                        tryPath += 'u';
                        break;
                    default:
                        // Should never happen
                        break;
                }
            } else {
                tryPath += str[i];
            }
        }
        if (pathGood(tryPath)) {
            goodPaths.push(tryPath);
        }
    });

    // goodPaths according to the spec shold only ever be === 1, but this code can
handle more true cases
    return goodPaths[0];



    // Confirm/Deny good path (with Qmarks filled in)
    function pathGood(path) {
```

```
        let startX = 0;
        let startY = 0;

        let grid = [
            [1,0,0,0,0],
            [0,0,0,0,0],
            [0,0,0,0,0],
            [0,0,0,0,0],
            [0,0,0,0,0]
        ];  // goal 4,4

        for (let i = 0; i < path.length; i++) {
            switch (path[i]) {
                case 'r':
                    startX++;
                    break;
                case 'd':
                    startY++;
                    break;
                case 'l':
                    startX--;
                    break;
                case 'u':
                    startY--;
                    break;
                default:
                    // Should never happen
                    break;

            }

            if (startX < 0 || startX > 4 || startY < 0 || startY > 4 || grid[startY]
[startX] === 1) {
                // already traveled or out of bounds
                return false;
            }
            grid[startY][startX] = 1;
        }


        return (startX === 4 && startY === 4) ? true : false;


    }
}

// keep this function call here
CorrectPath(readline());

// check if the given directions can reach the finish line
// without touching previously visited positions
function canNavigate(str) {
    var position = [0, 0];
    var visited = {};
```

```
    for (var i = 0; i < str.length; i++) {
        switch(str[i]) {
            case 'u':
                position[0]--;
                if (position[0] < 0) return false;
                break;
            case 'd':
                position[0]++;
                if (position[0] > 4) return false;
                break;
            case 'l':
                position[1]--;
                if (position[1] < 0) return false;
                break;
            case 'r':
                position[1]++;
                if (position[1] > 4) return false;
                break;
            default:
                break;
        }
        if (visited[position[0] + '-' + position[1]]) {
            // already visited before
            return false;
        } else {
            // mark as visited
            visited[position[0] + '-' + position[1]] = i;
        }
    }

    return (position[0] === 4 && position[1] === 4);
}

function findMissingChars(str) {
    // first, generate all possible cases: replacing ? with directions
    var permutations = [''];
    for (var i = 0; i < str.length; i++) {
        if (str[i] === '?') {
            var newPermutations = [];
            permutations.forEach(function(permutation) {
                newPermutations.push(permutation + 'u');
                newPermutations.push(permutation + 'd');
                newPermutations.push(permutation + 'l');
                newPermutations.push(permutation + 'r');
            });
            permutations = newPermutations;
        } else {
            permutations = permutations.map(permutation => permutation + str[i]);
        }
    }

    // now filter out only valid ones
    // we need a net result of 4 downs and 4 rights
    return permutations.filter(function(permutation) {
```

```
        var rightCount = permutation.match(/[r]/g) === null ? 0 : permutation.match(/
[r]/g).length;
        var leftCount = permutation.match(/[l]/g) === null ? 0 :
permutation.match(/[l]/g).length;
        var upCount = permutation.match(/[u]/g) === null ? 0 :
permutation.match(/[u]/g).length;
        var downCount = permutation.match(/[d]/g) === null ? 0 :
permutation.match(/[d]/g).length;

        return (rightCount - leftCount === 4) && (downCount - upCount === 4);
    });
}

function CorrectPath(str) {
    var validPaths = findMissingChars(str);

    for (var validPath of validPaths) {
        if (canNavigate(validPath)) {
            return validPath;
        }
    }
}

// keep this function call here
CorrectPath(readline());
```

Scale Balancing

Have the function `ScaleBalancing(strArr)` read **`strArr`** which will contain two elements, the first being the two positive integer weights on a balance scale (left and right sides) and the second element being a list of available weights as positive integers. Your goal is to determine if you can balance the scale by using the least amount of weights from the list, but using at most only 2 weights. For example: if **`strArr`** is ["[5, 9]", "[1, 2, 6, 7]"] then this means there is a balance scale with a weight of 5 on the left side and 9 on the right side. It is in fact possible to balance this scale by adding a 6 to the left side from the list of weights and adding a 2 to the right side. Both scales will now equal 11 and they are perfectly balanced. Your program should return a comma separated string of the weights that were used from the list in ascending order, so for this example your program should return the string **2,6**

There will only ever be one unique solution and the list of available weights will not be empty. It is also possible to add two weights to only one side of the scale to balance it. If it is not possible to balance the scale then your program should return the string **not possible**.

Examples

Input: ["[3, 4]", "[1, 2, 7, 7]"]
Output: 1

Input: ["[13, 4]", "[1, 2, 3, 6, 14]"]
Output: 3,6

```
function ScaleBalancing(strArr) {
	//convert the array to something more workable
	let newArr = strArr.map(val => {
		return val.replace(/[\[\]]/g, "").split(',').map(val2 => {
			return parseInt(val2, 10);
		}).sort((a, b) => {
			return a - b;
		});
	});

	let diff = newArr[0][1] - newArr[0][0];
	let weights = newArr[1];

	//do the single-weight solution test
	if (weights.includes(diff)) {
		return diff.toString();
	}
    //do the two-weights, one-side test
	let weight1 = weights.find((val, ind) => {
		let newWeights = weights.slice(0);
		newWeights.splice(ind, 1);
		return newWeights.includes (diff - val)
	});
	if (weight1) {
		return `${weight1},${diff - weight1}`
	}
    //do the twp-weights, different sides, test
	weight1 = weights.find(val => {
		return weights.includes(diff + val);
	});
	if (weight1) {
		return `${weight1},${diff + weight1}`
	}
    //if nothing is returned yet . . .
	return `not possible`;

}

// keep this function call here
ScaleBalancing(readline());

function ScaleBalancing(strArr) {
```

```
    let objects = strArr[0].substr(1, strArr[0].length-2).split(', ').map(object =>
parseInt(object));
    let weights = strArr[1].substr(1, strArr[1].length-2).split(', ').map(weight =>
parseInt(weight));

    /*
        Generate all possible combinations of weights AND permutations of left/right
- 3^n time
        0 - Weight not in use
        1 - Weight on left side
        2 - Weight on right side
    */
    let combos = [];
    for (let i = 0, max = Math.pow(3, weights.length); i < max; i++) {
        let combo = i.toString(3);
        let numWeights = combo.split('').reduce((a,v) => a + (parseInt(v) > 0 | 0),
0);

        // Too many weights, discard this combo
        if (numWeights > 2) {
            continue;
        }

        // Pad
        while (combo.length < weights.length) {
            combo = '0' + combo;
        }

        combos.push(combo);
    }

    // console.log(combos);



    // Test combos
    let goodCombos = [];
    combos.forEach(function(combo){
        let left = objects[0];
        let right = objects[1];

        for (let i = 0; i < combo.length; i++) {
            if (combo[i] === '1') {  // Left
                left += weights[i];
            }
            if (combo[i] === '2') {  // Right
                right += weights[i];
            }
        }

        if (left === right) {
            goodCombos.push(combo);
        }
```

```
    });

    if (goodCombos.length === 0) {
        return 'not possible';
    }

    // Sort first by number of physical weights used, then by total weight if there
are multiple sets
    goodCombos.sort(function(a, b){
        let aCount = a.split('').reduce((ac,v) => ac + (parseInt(v) > 0 | 0), 0);
        let bCount = b.split('').reduce((ac,v) => ac + (parseInt(v) > 0 | 0), 0);

        if (aCount < bCount) {
            return -1;
        }
        if (aCount > bCount) {
            return 1;
        }
        // aCount === bCount -> must check weights and use lesser weight total
        let aTotal = 0;
        let bTotal = 0;
        for (let i = 0; i < a.length; i++) {
            if (a[i] !== '0') {
                aTotal += weights[i];
            }
            if (b[i] !== '0') {
                bTotal += weights[i];
            }
        }
        return aTotal - bTotal;
    });
    //console.log(goodCombos);

    let theCombo = goodCombos[0];
    let finalWeights = [];
    theCombo.split('').map(function(value, index) {
        if (value !== '0') {
            finalWeights.push(weights[index]);
        }
    });

    return finalWeights.sort((a,b) => a-b).join(',');
}

// keep this function call here
ScaleBalancing(readline());

function ScaleBalancing(strArr) {
    var weights = strArr[1].match(/\d+/g).map(Number);
    var weightsOnScale = strArr[0].match(/\d+/g).map(Number);
    var leftWeight = weightsOnScale[0];
    var rightWeight = weightsOnScale[1];

    // try only one weight
```

```
    for (var i = 0; i < weights.length; i++) {
        if (leftWeight < rightWeight) {
            if (leftWeight + weights[i] === rightWeight)
                return weights[i];
        } else {
            if (leftWeight === rightWeight + weights[i])
                return weights[i];
        }
    }

    // now try two weights
    for (var i = 0; i < weights.length; i++) {
        for (var j = i + 1; j < weights.length; j++) {
            // add one on each side
            if (leftWeight + weights[i] === rightWeight + weights[j]) {
                return weights[i] + ',' + weights[j];
            } else if (leftWeight + weights[j] === rightWeight + weights[i]) {
                return weights[i] + ',' + weights[j];
            }
            // add two on the lighter side
            if (leftWeight < rightWeight) {
                if (leftWeight + weights[i] + weights[j] === rightWeight) {
                    return weights[i] + ',' + weights[j];
                }
            } else {
                if (leftWeight === rightWeight + weights[i] + weights[j]) {
                    return weights[i] + ',' + weights[j];
                }
            }
        }
    }

    // no solution
    return 'not possible';
}

// keep this function call here
ScaleBalancing(readline());
```

Three Numbers

Have the function `ThreeNumbers(str)` take the **`str`** parameter being passed and determine if exactly three unique, single-digit integers occur within each word in the string. The integers can appear anywhere in the word, but they cannot be all adjacent to each other. If every word contains exactly 3 unique integers somewhere within it, then return the string **true**, otherwise return the string **false**. For example: if **`str`** is "2hell6o3 wor6l7d2" then your program should return "true" but if the string is "hell268o w6or2l4d" then your program should return "false" because all the integers are adjacent to each other in the first word.

Examples

Input: "2a3b5 w1o2rl3d g1gg92"
Output: true

Input: "21aa3a ggg4g4g6ggg"
Output: false

```
function ThreeNumbers(str) {

    const DIGITS = '0123456789';

    let words = str.split(' ');

    let result = null;

    words.forEach(function(word){
        let numbers = 0;
        let threeAdjacent = false;
        let usedDigits = '';
        for (let i = 0; i < word.length; i++) {
            // Check for used digits
            if (usedDigits.includes(word[i])) {
                // used! fail
                result = false;
            }

            if (DIGITS.includes(word[i])) {
                numbers++;
                usedDigits += word[i];
                if (numbers === 3) {
                    if (DIGITS.includes(word[i-1]) && DIGITS.includes(word[i-2])) {
                        threeAdjacent = true;
                    }
                }
            }
            // Check for 3 adjacent preceding digits
```

```
        }

        if (numbers === 3 && !threeAdjacent) {
            result = (result === null) ? true : result && true;

        } else {

            result = false;
        }
    });

    return result;
}

// keep this function call here
ThreeNumbers(readline());

function ThreeNumbers(str) {
    var words = str.split(' ');
    var numbersOfWords = words.map(word => word.match(/\d+/g));

    for (var numbers of numbersOfWords) {
        // check for exactly 3 numbers in each word
        var digits = numbers.join('');
        if (digits.length !== 3)
            return false;
        // no 3 numbers in a row allowed
        if (numbers.length !== numbers.filter(number => number.length < 3).length)
            return false;
        // no duplicating number in those 3 numbers
        if ((digits[0] === digits[1]) ||
            (digits[1] === digits[2]) ||
            (digits[2] === digits[0]))
            return false;
    }

    return true;
}

// keep this function call here
ThreeNumbers(readline());

function ThreeNumbers(str) {

  let array = str.split(' ');
  for (let i = 0; i < array.length; i++) {
      if (/[0-9]{3,}/.exec(array[i]) !== null) {
          return false;
      } else {
          let n = array[i].match(/[0-9]/g);
          if (n[0] === n[1] || n[1] === n[2] || n[0] === n[1]) {
              return false;
          }
```

```
      }
  }
  return true;

}

// keep this function call here
ThreeNumbers(readline());
```

Alphabet Searching

Have the function `AlphabetSearching(str)` take the **`str`** parameter being passed and return the string **true** if every single letter of the English alphabet exists in the string, otherwise return the string **false**. For example: if **`str`** is "zacxyjbbkfgtbhdaielqrm45pnsowtuv" then your program should return the string **true** because every character in the alphabet exists in this string even though some characters appear more than once.

Examples

Input: "abcdefghijklmnopqrstuvwxyyyy"
Output: false

Input: "abc123456kmo"
Output: false

```
function AlphabetSearching(str) {
       str = str.toLowerCase();
       for (let i = 97; i < 97 + 26; i++) {
              let char = String.fromCharCode(i);
              if (!str.includes(char)) {
                     return false;
              }
       }
       return true;
}

// keep this function call here
AlphabetSearching(readline());
```

```
function AlphabetSearching(str) {

    const LOWER_LETTERS = 'abcdefghijklmnopqrstuvwxyz';

    for (let i = 0; i < LOWER_LETTERS.length; i++) {
        if (!str.includes(LOWER_LETTERS[i])) {
            return false;
```

```
        }
    }
    return true;
}

// keep this function call here
AlphabetSearching(readline());

function isAlpha(char) {
    return /[A-Za-z]/.test(char);
}

function AlphabetSearching(str) {
    var foundLetters = [];
    for (var i = 0; i < str.length; i++) {
        if ( isAlpha(str[i]) && (foundLetters.indexOf(str[i]) === -1) ) {
            foundLetters.push(str[i]);
        }
    }
    return foundLetters.length === 26;
}

// keep this function call here
AlphabetSearching(readline());
```

Time Difference

Have the function `TimeDifference(strArr)` read the array of strings stored in `strArr` which will be an unsorted list of times in a twelve-hour format like so: HH:MM(am/pm). Your goal is to determine the smallest difference in minutes between two of the times in the list. For example: if `strArr` is ["2:10pm", "1:30pm", "10:30am", "4:42pm"] then your program should return **40** because the smallest difference is between 1:30pm and 2:10pm with a difference of 40 minutes. The input array will always contain at least two elements and all of the elements will be in the correct format and unique.

Examples

Input: ["1:10pm", "4:40am", "5:00pm"]
Output: 230

Input: ["10:00am", "11:45pm", "5:00am", "12:01am"]
Output: 16

```
function TimeDifference(strArr) {
    //times will hold our time differences
       let times = [];
       let newStrArr = strArr.map (val => {
              let matches = val.match(/^(\d+):(\d+)([ap]m)$/);
              let hours = parseInt(matches[1], 10);
              let minutes = parseInt(matches[2], 10);
              let half = matches[3];
              if (half === 'am' && hours === 12) {
                     hours = 0;
              }
              if (half === 'pm' && hours !== 12) {
                     hours += 12;
              }
              return (hours * 60 + minutes);
       })
              .sort((a, b) => { return a - b});
    //tricky - second example shows that we have to consider the earliest time both
on day 0 and day 1
       newStrArr.push(newStrArr[0] + 24 * 60);

       for (let i = 0, len = newStrArr.length; i < len - 1; i ++) {
              times.push(newStrArr[i + 1] - newStrArr[i]);
       }
       return Math.min.apply(null, times);
}

// keep this function call here
```

```
TimeDifference(readline());


function TimeDifference(strArr) {

    let timesInSeconds = [];
    strArr.forEach(function(strTime){
        let pair = strTime.split(':');
        let hours = (pair[1][2] === 'a') ? parseInt(pair[0]) % 12 : parseInt(pair[0])
% 12 + 12;
        let seconds = parseInt(pair[1].substr(0,2));
        let totalSeconds = hours * 60 + seconds;
        timesInSeconds.push(totalSeconds);
    });

    // Iterate over 2^n combos
    let smallestDifference = Number.MAX_VALUE;
    for (let i = 0, max = Math.pow(2,timesInSeconds.length); i < max; i++) {
        let combo = i.toString(2);
        let timesUsed = combo.split('').reduce((a,v) => a + (v === '1' | 0), 0);
        if (timesUsed === 2) {
            // Pad
            while (combo.length < timesInSeconds.length) {
                combo = '0' + combo;
            }

            // Iterate over each specific combo and test them
            let diff = 0;
            let pair = [];
            for (let j = 0; j < combo.length; j++) {
                if (combo[j] === '1') {
                    pair.push(timesInSeconds[j]);
                }
            }

            let t1 = Math.abs(pair[0] - pair[1]);
            let t2 = Math.abs(Math.min((1440 - pair[0]), (pair[0])) + Math.min((1440
- pair[1]), (pair[1])));

            diff = Math.min(t1, t2);

            if (diff < smallestDifference) {
                smallestDifference = diff;
            }
        }
    }

    return smallestDifference;
}

// keep this function call here
TimeDifference(readline());

function TimeDifference(strArr) {
```

```
    let array = strArr.map(value => {
        let matches = value.match(/^(\d+):(\d+)([ap]m)$/);
        let h = parseInt(matches[1], 10);
        let m = parseInt(matches[2], 10);
        if (matches[3] === 'am' && h === 12) {
            h = 0;
        } else if (matches[3] === 'pm' && h !== 12) {
            h += 12;
        }
        return (60 * h + m);
    }).sort(function(a, b) {
        return a - b;
    });
    array.push(array[0] + 24 * 60);
    let times = [];
    for (let i = 0; i < array.length - 1; i++) {
        times.push(array[i+1] - array[i]);
    }
    return Math.min(...times);

}
// keep this function call here
TimeDifference(readline());
```

Triangle Row

Have the function `TriangleRow(num)` take **`num`** which will be a positive integer representing some row from Pascal's triangle. Pascal's triangle starts with a [1] at the 0th row of the triangle. Then the first row is [1, 1] and the second row is [1, 2, 1]. The next row begins with 1 and ends with 1, and the inside of the row is determined by adding the k-1 and kth elements from the previous row. The next row in the triangle would then be [1, 3, 3, 1], and so on. The input will be some positive integer and your goal is to return the sum of that row. For example: if **`num`** is 4 then your program should return the sum of 1 + 4 + 6 + 4 + 1 which is **16**.

Examples

Input: 1
Output: 2

Input: 2
Output: 4

```
function TriangleRow(num) {
       return Math.pow(2, num);
}

console.log(TriangleRow(5));

// keep this function call here
TriangleRow(readline());


function TriangleRow(num) {

    let triangle = [];
    for (let row = 0; row <= num; row++) {
        let rowArr = [];
        for (let i = 0; i <= row; i++) {
            if (i === 0) {
                rowArr.push(1);
                continue;
            }
            let delta = (i < triangle[row-1].length) ? triangle[row-1][i] : 0;
            rowArr.push(triangle[row-1][i-1] + delta);
        }
        triangle.push(rowArr);
    }

    return triangle[num].reduce((a,v) => a + v,0);

}
```

```
// keep this function call here
TriangleRow(readline());


// get nth row and kth column of pascal's triangle
function pascalTriangle(n, k) {
    if (n === 0) return 1;
    if (k === 0 || k === n) return 1;
    return pascalTriangle(n - 1, k - 1) + pascalTriangle(n - 1, k);
}

function TriangleRow(num) {
    // the easy answer - it always sums up to n^2
    //return num * num;

    // the hard way
    var rowSum = 0;
    for (k = 0; k <= num; k++) {
        result = pascalTriangle(num, k);
        rowSum += result;
    }
    return rowSum;
}

// keep this function call here
TriangleRow(readline());
```

Vowel Square

Have the function `VowelSquare(strArr)` take the **`strArr`** parameter being passed which will be a 2D matrix of some arbitrary size filled with letters from the alphabet, and determine if a 2x2 square composed entirely of vowels exists in the matrix. For example: **`strArr`** is ["abcd", "eikr", "oufj"] then this matrix looks like the following:

a b c d
e i k r
o u f j

Within this matrix there is a 2x2 square of vowels starting in the second row and first column, namely, ei, ou. If a 2x2 square of vowels is found your program should return the top-left position (row-column) of the square, so for this example your program should return **1-0**. If no 2x2 square of vowels exists, then return the string **not found**. If there are multiple squares of vowels, return the one that is at the most top-left position in the whole matrix. The input matrix will at least be of size 2x2.

Examples

Input: ["aqrst", "ukaei", "ffooo"]
Output: 1-2

Input: ["gg", "ff"]
Output: not found

```
function VowelSquare(strArr) {
	strArr = strArr.map(val => {
		return val.toLowerCase().replace(/[aeiou]/g, "!");
	})
	for (let r = 0, len = strArr.length; r < len - 1; r++) {
		for (let c = 0, len = strArr[0].length; c < len - 1; c++) {
			if (checkPoint(strArr, [r, c])) {
				return `${r}-${c}`;
			}
		}
	}
	return 'not found'

}

function checkPoint(arr, point) {
	console.log('arr',  arr[point[0]][point[1]]);
	console.log('point', point);
	return (
		arr[point[0]][point[1]] === '!' &&
		arr[point[0] + 1][point[1]] === '!' &&
		arr[point[0]][point[1] + 1] === '!' &&
		arr[point[0] + 1][point[1] + 1] === '!'
	)
}

// keep this function call here
VowelSquare(readline());


function VowelSquare(strArr) {

    for (let row = 0; row < strArr.length-1; row++) {
        for (let col = 0; col < strArr[0].length-1; col++) {
            if (isVowels2x2(strArr, row, col)) {
                // Good!  Return first one because we
                // searche in a good order
                return row + '-' + col;
            }
        }
    }
    return 'not found';
```

```
    function isVowels2x2(strArr, row, col) {
        // Could do bounds checking here but it is done in main()

        if (!isVowel(strArr[row][col])) return false;
        if (!isVowel(strArr[row+1][col])) return false;
        if (!isVowel(strArr[row][col+1])) return false;
        if (!isVowel(strArr[row+1][col+1])) return false;
        return true;
    }

    function isVowel(letter) {
        const VOWELS = 'aeiou';
        return VOWELS.includes(letter);
    }
}

// keep this function call here
VowelSquare(readline());

// check if every letter in the given matrix is a vowel
function allVowels(matrix) {
    return matrix.every(row => row.every(letter => 'aeiou'.indexOf(letter) !== -1));
}

function VowelSquare(strArr) {
    var matrix = strArr.map(row => row.split(''));

    for (var row = 0; row < matrix.length - 1; row++) {
        for (var column = 0; column < matrix[0].length - 1; column++) {
            // test with sliding windows of 2x2 sub-matrix
            var subMatrix = matrix.slice(row, row + 2).map(rowArr =>
rowArr.slice(column, column + 2));
            if (allVowels(subMatrix)) {
                return row + '-' + column;
            }
        }
    }

    return 'not found';
}

// keep this function call here
VowelSquare(readline());
```

Closest Enemy

Have the function `ClosestEnemy(arr)` take the array of numbers stored in **`arr`** and from the position in the array where a 1 is, return the number of spaces either left or right you must move to reach an enemy which is represented by a 2. For example: if **`arr`** is [0, 0, 1, 0, 0, 2, 0, 2] then your program should return **3** because the closest enemy (2) is 3 spaces away from the 1.

The array will contain any number of 0's and 2's, but only a single 1. It may not contain any 2's at all as well, where in that case your program should return a 0.

Examples

Input: [1, 0, 0, 0, 2, 2, 2]
Output: 4

Input: [2, 0, 0, 0, 2, 2, 1, 0]
Output: 1

```
function ClosestEnemy(arr) {
	if (!arr.includes(2)) {
		return 0;
	}
	loc1 = arr.findIndex(val => {
		return val === 1;
	});

	for (let i = 1, len = arr.length; i < len; i++) {
		if (arr[loc1 + i] === 2 || arr[loc1 - i] === 2) {
			return i;
		}
	}
	return loc1;
}

// keep this function call here
ClosestEnemy(readline());

function ClosestEnemy(arr) {

    let hero = -1;
    for (let i = 0; i < arr.length; i++) {
        if (arr[i] === 1) {  // Found hero
            hero = i;
        }
    }

    // Search right
    let deltaRight = undefined;
    for (let i = hero+1; i < arr.length; i++) {
        if (arr[i] === 2) {  // Found enemy
            deltaRight = i - hero;
            break;
        }
    }

    // Search left
    let deltaLeft = undefined;
```

```
    for (let i = hero-1; i >= 0; i--) {
        if (arr[i] === 2) {  // Found enemy
            deltaLeft = hero - i;
            break;
        }
    }

    if (deltaLeft === undefined && deltaRight === undefined) {
        return 0;
    }

    if (deltaLeft === undefined) {
        return deltaRight;
    }

    if (deltaRight === undefined) {
        return deltaLeft;
    }

    return (deltaLeft < deltaRight) ? deltaLeft : deltaRight;
}

// keep this function call here
ClosestEnemy(readline());


function ClosestEnemy(arr) {
    // index of 1
    var me = arr.indexOf(1);
    var minimumDistance = arr.length;

    for (var i = 0; i < arr.length; i++) {
        if (arr[i] === 2) {
            var distance = Math.abs(i - me);
            if (distance < minimumDistance) {
                minimumDistance = distance;
            }
        }
    }

    return (minimumDistance === arr.length) ? 0 : minimumDistance;
}

// keep this function call here
ClosestEnemy(readline());
```

Closest Enemy II

Have the function `ClosestEnemyII(strArr)` read the matrix of numbers stored in **`strArr`** which will be a 2D matrix that contains only the integers 1, 0, or 2. Then from the position in the matrix where a 1 is, return the number of spaces either left, right, down, or up

you must move to reach an enemy which is represented by a 2. You are able to wrap around one side of the matrix to the other as well. For example: if **strArr** is ["0000", "1000", "0002", "0002"] then this looks like the following:

```
0 0 0 0
1 0 0 0
0 0 0 2
0 0 0 2
```

For this input your program should return **2** because the closest enemy (2) is 2 spaces away from the 1 by moving left to wrap to the other side and then moving down once. The array will contain any number of 0's and 2's, but only a single 1. It may not contain any 2's at all as well, where in that case your program should return a 0.

Examples

Input: ["000", "100", "200"]
Output: 1

Input: ["0000", "2010", "0000", "2002"]
Output: 2

```
function ClosestEnemyII(strArr) {
	//step one - return 0 if there is no 2 in the array
	let twosies = strArr.filter(val => {
		return val.includes("2");
	});
	if (!twosies.length) {
		return 0;
	}
	//step two - get the coordinates of the 1 (targetX, targetY)
	targetY = strArr.findIndex(val => {
		return val.includes('1');
	});
	targetX = strArr[targetY].search(/1/);
	//step three find the smallest path to a 2
	let res = strArr.length * strArr[0].length;

	for (let row = 0, len = strArr[0].length; row < len; row++) {
		for (col = 0, height = strArr.length; col < height; col++) {
			if (strArr[row][col] === '2') {
				xShift = rowDist(targetX, col, len);
```

```
                        yShift = rowDist(targetY, row, height);
                        res = Math.min(res, xShift + yShift);
                    }
                }
        }
        return res;

}

//--------------------helpers---------------------
//despite the name, use for column and row distance
function rowDist(y, x, len) {
        return Math.min(Math.abs(x - y), Math.abs(y - x + len));
}

// keep this function call here
ClosestEnemyII(readline());


function ClosestEnemyII(strArr) {

    // Find hero
    let heroY = -1;
    let heroX = -1;
    for (let i = 0; i < strArr.length; i++) {
        let result = strArr[i].indexOf(1);
        if (result > -1) {
            heroX = result;
            heroY = i;
        }
    }

    if (heroY === -1) {
        // No hero exists
        return -1;
    }

    //console.log(heroY + '-' + heroX);


    // Check for enemies
    let enemies = [];
    for (let i = 0; i < strArr.length; i++) {
        let result = strArr[i].indexOf(2);
        if (result > -1) {
            enemies.push([i,result]);
        }
    }

    if (enemies.length === 0) {
        return 0;
    }

    //console.log(enemies);
```

```
    let closestDistance = Number.MAX_SAFE_INTEGER;

    // Check enemy distances
    for (let i = 0; i < enemies.length; i++) {
        let enemyX = enemies[i][1];
        let enemyY = enemies[i][0];

        let deltaX = Math.abs(enemyX - heroX);
        let deltaY = Math.abs(enemyY - heroY);
        //console.log(deltaY + '-' + deltaX);

        // Now check wrap-around values
        // deltaX is max because if it is more then we don't care
        let wrapDeltaX = Number.MAX_SAFE_INTEGER;
        for (let i = 0; i < deltaX; i++) {
            if ((enemyX + i) % strArr[0].length === heroX) {
                // found hero delta
                wrapDeltaX = i;
                //console.log('wrap-aroundX: ' + i)
            }
        }

        let wrapDeltaY = Number.MAX_SAFE_INTEGER;
        for (let i = 0; i < deltaY; i++) {
            if ((enemyY + i) % strArr.length === heroY) {
                // found hero delta
                wrapDeltaY = i;
                //console.log('wrap-aroundY: ' + i)
            }
        }

        deltaX = (wrapDeltaX < deltaX) ? wrapDeltaX : deltaX;
        deltaY = (wrapDeltaY < deltaY) ? wrapDeltaY : deltaY;

        let distance = deltaX + deltaY;

        if (distance < closestDistance) {
            closestDistance = distance;
        }
    }

    return closestDistance;

}

// keep this function call here
ClosestEnemyII(readline());

// find the distance between two elements with given [row, column] indices
function getDistance(matrixSize, indices1, indices2) {
    var rowDistance = Math.min(Math.abs(indices2[0] - indices1[0]),
Math.abs(indices1[0] - (indices2[0] - matrixSize)));
```

```
    var columnDistance = Math.min(Math.abs(indices2[1] - indices1[1]),
Math.abs(indices1[1] - (indices2[1] - matrixSize)));
    return rowDistance + columnDistance;
}

function ClosestEnemyII(strArr) {
    var matrix = strArr.map(line => line.split('').map(Number));
    var minDistance = matrix.length * 2;

    // location of me: 1
    var meIndices;
    // locations of enemies: 2's
    var enemiesIndices = [];

    // identify the locations of 1 and 2's
    for (var row = 0; row < matrix.length; row++) {
        for (var column = 0; column < matrix[0].length; column++) {
            if (matrix[row][column] === 1) {
                meIndices = [row, column];
            } else if (matrix[row][column] === 2) {
                enemiesIndices.push([row, column]);
            }
        }
    }

    // check the distance from me for every enemy
    for (var enemyIndices of enemiesIndices) {
        var distance = getDistance(matrix.length, meIndices, enemyIndices);
        if (distance < minDistance) {
            minDistance = distance;
        }
    }

    // handles the case where there is no enemy
    return minDistance === matrix.length * 2 ? 0 : minDistance;
}

// keep this function call here
ClosestEnemyII(readline());
```

Number Stream

Have the function `NumberStream(str)` take the **`str`** parameter being passed which will contain the numbers 2 through 9, and determine if there is a consecutive stream of digits of at least N length where N is the actual digit value. If so, return the string **true**, otherwise return the string **false**. For example: if **`str`** is "6539923335" then your program should return the string **true** because there is a consecutive stream of 3's of length 3. The input string will always contain at least one digit.

Examples

Input: "5556293383563665"
Output: false

Input: "5788888888882339999"
Output: true

```
function NumberStream(str) {
    for (let i = 2; i < 10; i++) {
        let iChar = i.toString();
        let needle = iChar.repeat(i);
        if (str.indexOf(needle) !== -1) {
            return true;
        }
    }
  // code goes here
  return false;

}

// keep this function call here
NumberStream(readline());
```

```
function NumberStream(str) {

    for (let i = 0, lastDigit = -1, count = 0; i < str.length; i++) {
        if (str[i] === lastDigit) {
            // Stream continued
            count++;
            if (count >= Number(lastDigit)) {
                return true;
            }
        } else {
            // New stream
            lastDigit = str[i];
            count = 1;
```

```
        }
    }
    return false;
}

// keep this function call here
NumberStream(readline());

function NumberStream(str) {
    // generate patterns first
    var patterns = [];
    for (var i = 1; i < 10; i++) {
        patterns.push(String(i).repeat(i));
    }

    for (var pattern of patterns) {
        if (str.indexOf(pattern) !== -1) {
            // pattern found
            return true;
        }
    }
    return false;
}

// keep this function call here
NumberStream(readline());
```

Largest Four

Have the function `LargestFour(arr)` take the array of integers stored in **arr**, and find the four largest elements and return their sum. For example: if **arr** is [4, 5, -2, 3, 1, 2, 6, 6] then the four largest elements in this array are 6, 6, 4, and 5 and the total sum of these numbers is **21**, so your program should return **21**. If there are less than four numbers in the array your program should return the sum of all the numbers in the array.

Examples

Input: [1, 1, 1, -5]
Output: -2

Input: [0, 0, 2, 3, 7, 1]
Output: 13

```
function LargestFour(arr) {
      let newArr = arr.sort((val1, val2) => {
                  return val2 - val1;
```

```
            })
            .splice(0, 4);
      return newArr.reduce((val1, val2) => {
            return val1 + val2;
      }, 0);
}

// keep this function call here
LargestFour(readline());


function LargestFour(arr) {

    return arr.sort((a,b) => b-a).reduce((a,v,i) => a + ((i < 4) ? v : 0), 0);

}

// keep this function call here
LargestFour(readline());


function LargestFour(arr) {
    return arr.sort((a, b) => a < b).slice(0, 4).reduce((sum, v) => sum + v, 0);
}

// keep this function call here
LargestFour(readline());
```

Distinct Characters

Have the function `DistinctCharacters(str)` take the **`str`** parameter being passed and determine if it contains at least 10 distinct characters, if so, then your program should return the string **true**, otherwise it should return the string **false**. For example: if **`str`** is "abc123kkmmmm?" then your program should return the string **false** because this string contains only 9 distinct characters: a, b, c, 1, 2, 3, k, m, ? adds up to 9.

Examples

Input: "12334bbmma:=6"
Output: true

Input: "eeeemmmmmmmmm1000"
Output: false

```
function DistinctCharacters(str) {
      let mySet = new Set(str.split(''));
      return mySet.size >= 10 ? true :  false
```

```
}

// keep this function call here
DistinctCharacters(readline());

function DistinctCharacters(str) {

    let charCodes = [];
    for (let i = 0; i < str.length; i++) {
        if (!charCodes.includes(str.charCodeAt(i))) {
            charCodes.push(str.charCodeAt(i));
        }
    }
    return (charCodes.length >= 10) ? true : false;

}

// keep this function call here
DistinctCharacters(readline());

function DistinctCharacters(str) {
    var distincts = {};

    for (var i = 0; i < str.length; i++) {
        distincts[str[i]] = '';
    }

    return Object.keys(distincts).length >= 10;
}

// keep this function call here
DistinctCharacters(readline());
```

Questions Marks

Have the function `QuestionsMarks(str)` take the **`str`** string parameter, which will contain single digit numbers, letters, and question marks, and check if there are exactly 3 question marks between every pair of two numbers that add up to 10. If so, then your program should return the string **true**, otherwise it should return the string **false**. If there aren't any two numbers that add up to 10 in the string, then your program should return **false** as well.

For example: if **`str`** is "arrb6???4xxbl5???eee5" then your program should return **true** because there are exactly 3 question marks between 6 and 4, and 3 question marks between 5 and 5 at the end of the string.

Examples

Input: "aa6?9"

Output: false

Input: "acc?7??sss?3rr1??????5"

Output: true

```
function QuestionsMarks(str) {
	let numPlaces = [];
	//presumption of false, until flag is turned true
	let flag = false;

    //get an array of places in string holding integers
	for (let i = 0, len = str.length; i < len; i++) {
		if (/\d/.test(str[i])) {
			numPlaces.push(i);
		}
	}

	let numCount = numPlaces.length;

	for (let i = 0; i < numCount - 1; i++) {
		if (parseInt(str[numPlaces[i]], 10) + parseInt(str[numPlaces[i + 1]],
10) === 10) {
			flag = true;
			let strSeg = str.slice(numPlaces[i], numPlaces[i + 1]);

			strSeg = strSeg.replace(/[^\?]/g, '');
			if (strSeg !== '???') {
				return false;
			}
		}
	}
	return flag;
}

// keep this function call here
QuestionsMarks(readline());
```

Camel Case

Have the function `CamelCase(str)` take the `str` parameter being passed and return it in proper camel case format where the first letter of each word is capitalized (excluding the first letter). The string will only contain letters and some combination of delimiter punctuation characters separating each word.

For example: if **`str`** is "BOB loves-coding" then your program should return the string **bobLovesCoding**.

Examples

Input: "cats AND*Dogs-are Awesome"
Output: catsAndDogsAreAwesome

Input: "a b c d-e-f%g"
Output: aBCDEFG

```
function CamelCase(str) {
	let strArr = str.split(/[^a-zA-Z]/);
	strArr = strArr.map((val, ind) => {
		val = val.toLowerCase();
		if (ind) {
			valArr = val.split('');
			valArr[0] = valArr[0].toUpperCase();
			return valArr.join('');
		}
		return val;
	})

	return strArr.join('');
}

// keep this function call here
CamelCase(readline());
```

ASCII Conversion

Have the function `ASCIIConversion(str)` take the **`str`** parameter being passed and return a new string where every character, aside from the space character, is replaced with its corresponding decimal character code. For example: if **`str`** is "dog" then your program should return the string **100111103** because d = 100, o = 111, g = 103.

Examples

Input: "hello world"
Output: 104101108108111 119111114108100

Input: "abc **"
Output: 979899 4242

```
function ASCIIConversion(str) {
	let myArr = str.split(' ').map(val => {
		return val.split('').map(val2 => {
			return val2.charCodeAt(0);
		}).join('')
	}).join(' ');
	return myArr;
}

// keep this function call here
ASCIIConversion(readline());
```

Simple Evens

Have the function `SimpleEvens(num)` check whether every single number in the passed in parameter is even. If so, return the string **true**, otherwise return the string **false**. For example: if `num` is 4602225 your program should return the string **false** because 5 is not an even number.

Examples

Input: 2222220222
Output: true
Input: 20864646452
Output: false

```
function SimpleEvens(num) {
    return num.toString(10).search(/[13579]/) === -1 ? 'true' : 'false';
}
SimpleEvens(readline());
```

Snake Case

Have the function `SnakeCase(str)` take the `str` parameter being passed and return it in proper snake case format where each word is lowercased and separated from adjacent words via an underscore. The string will only contain letters and some combination of delimiter punctuation characters separating each word.

For example: if `str` is "BOB loves-coding" then your program should return the string **bob_loves_coding**.

Examples

Input: "cats AND*Dogs-are Awesome"
Output: cats_and_dogs_are_awesome

Input: "a b c d-e-f%g"
Output: a_b_c_d_e_f_g

```
function SnakeCase(str) {
      return str.split(/[^a-zA-Z]/).map(val => {return
val.toLowerCase()}).join('_');
}
```

```
SnakeCase(readline());
```

Find Intersection

Have the function `FindIntersection(strArr)` read the array of strings stored in **`strArr`** which will contain 2 elements: the first element will represent a list of comma-separated numbers sorted in ascending order, the second element will represent a second list of comma-separated numbers (also sorted). Your goal is to return a comma-separated string containing the numbers that occur in elements of **`strArr`** in sorted order. If there is no intersection, return the string **false**.

For example: if **`strArr`** contains ["1, 3, 4, 7, 13", "1, 2, 4, 13, 15"] the output should return "1,4,13" because those numbers appear in both strings. The array given will not be empty, and each string inside the array will be of numbers sorted in ascending order and may contain negative numbers.

Examples

Input: ["1, 3, 4, 7, 13", "1, 2, 4, 13, 15"]
Output: 1,4,13

Input: ["1, 3, 9, 10, 17, 18", "1, 4, 9, 10"]
Output: 1,9,10

```
function FindIntersection(input) {

  const [firstList, secondList] = input.map( s => s.split(", ") );

  const resultMap = {};
  const result = [];

  for ( const number of firstList ) {
    resultMap[ number ] = true;
  }

  for ( const number of secondList ) {
    if ( resultMap[number] ) {
      result.push( number );
    }
  }

  return result.length ? result.join(",") : false;
}
```

```
// keep this function call here
console.log(FindIntersection(readline()));
```

Sum Multiplier

Have the function `SumMultiplier(arr)` take the array of numbers stored in **`arr`** and return the string **true** if any two numbers can be multiplied so that the answer is greater than double the sum of all the elements in the array. If not, return the string **false**. For example: if **`arr`** is [2, 5, 6, -6, 16, 2, 3, 6, 5, 3] then the sum of all these elements is 42 and doubling it is 84. There are two elements in the array, 16 * 6 = 96 and 96 is greater than 84, so your program should return the string **true**.

Examples

Input: [2, 2, 2, 2, 4, 1]
Output: false

Input: [1, 1, 2, 10, 3, 1, 12]
Output: true

```
function SumMultiplier(arr) {
    const target = arr.reduce((val1, val2) => val1 + val2, 0) * 2;
       arr.sort((num1, num2) => num1 - num2);
       const len = arr.length;
       const checker = Math.max((arr[0] * arr[1]), (arr[len - 2] * arr[len - 1]));
       return checker > target;
}
SumMultiplier(readline());
```

String Merge

Have the function `StringMerge(str)` read the **`str`** parameter being passed which will contain a large string of alphanumeric characters with a single asterisk character splitting the string evenly into two separate strings. Your goal is to return a new string by pairing up the characters in the corresponding locations in both strings. For example: if **`str`** is "abc1*kyoo" then your program should return the string **akbyco1o** because a pairs with k, b pairs with y, etc. The string will always split evenly with the asterisk in the center.

Examples

Input: "aaa*bbb"
Output: abababab

Input: "123hg*aaabb"

Output: 1a2a3ahbgb

```
function StringMerge(str) {
	const cleanString = str.replace(/[^\w*]/g, '');
	const len = (cleanString.length - 1) / 2;
	let newString = '';
	const arr = str.split('*');

	for (let i = 0; i < len; i++) {
		newString += arr[0][i] + arr[1][i];
	}
	return newString;
}

// keep this function call here
StringMerge(readline());
```

One Decremented

Have the function `OneDecremented(str)` count how many times a digit appears that is exactly one less than the previous digit. For example: if `str` is "5655984" then your program should return **2** because 5 appears directly after 6 and 8 appears directly after 9. The input will always contain at least 1 digit.

Examples

Input: "56"

Output: 0

Input: "9876541110"

Output: 6

```
function OneDecremented(num) {
	let counter = 0;
	let arr = num.toString().split('');
	arr.forEach((val, ind) => {
		if (parseInt(val, 10) - parseInt(arr[ind + 1], 10) === 1) {
			counter++;
		}
	})
	return counter;
}

// keep this function call here
OneDecremented(readline());
```

Element Merger

Have the function `ElementMerger(arr)` take the array of positive integers stored in **`arr`** and perform the following algorithm: continuously get the difference of adjacent integers to create a new array of integers, then do the same for the new array until a single number is left and return that number. For example: if **`arr`** is [4, 5, 1, 2, 7] then taking the difference of each pair of elements produces the following new array: [1, 4, 1, 5]. Then do the same for this new array to produce [3, 3, 4] -> [0, 1] -> 1. So for this example your program should return the number **1** because that is what's left at the end.

Examples

Input: [5, 7, 16, 1, 2]
Output: 7

Input: [1, 1, 1, 2]
Output: 1

```
function ElementMerger(arr) {
	if (arr.length === 1) {
		return arr[0];
	} else {
		newArr = [];
		arr.forEach((val, ind) => {
			if (ind < arr.length - 1) {
				newArr.push(Math.abs(val - arr[ind + 1]));
			}
		})
		return ElementMerger(newArr);
	}
}
ElementMerger(readline());
```

GCF

Have the function `GCF(arr)` take the array of numbers stored in **`arr`** which will always contain only two positive integers, and return the greatest common factor of them. For example: if **`arr`** is [45, 12] then your program should return **3**. There will always be two elements in the array and they will be positive integers.

Examples

Input: [1, 6]
Output: 1

Input: [12, 28]
Output: 4

```
function GCF(arr) {
	let res = null;
	let max = Math.max(...arr);
	let min = Math.min(...arr);
	for (let i = 1; i <= min; i++) {
		if (max % i === 0 && min % i === 0) {
			res = i;
		}
	}
	return res;
}

// keep this function call here
GCF(readline());
```

Serial Number

Have the function `SerialNumber(str)` take the **`str`** parameter being passed and determine if it is a valid serial number with the following constraints:

1. It needs to contain three sets each with three digits (1 through 9) separated by a period.
2. The first set of digits must add up to an even number.
3. The second set of digits must add up to an odd number.
4. The last digit in each set must be larger than the two previous digits in the same set.

If all the above constraints are met within the string, the your program should return the string **true**, otherwise your program should return the string **false**. For example: if **`str`** is "224.315.218" then your program should return "true".

Examples

Input: "11.124.667"
Output: false

Input: "114.568.112"

Output: true

```
function SerialNumber(str) {
	//check the format of the string
	let goal = /^\d{3}\.\d{3}\.\d{3}$/
	if (!goal.test(str)) {
		return 'false';
	}

	//transform the string into three arrays of three digits each
	let arr = str.split(/\./).map(val => {
		return val.split('').map(val2 => {
			return parseInt(val2, 10);
		});
	});

	//check condition one
	if ((arr[0][0] + arr[0][1] + arr[0][2]) % 2) {
		return 'false';
	}

	//check condition two
	if (!((arr[1][0] + arr[1][1] + arr[1][2]) % 2)) {
		return 'false';
	}

	//check condition three
	for (let i = 0, len = arr.length; i < len; i++) {
		if (Math.max(...arr[i]) !== arr[i][2]) {
			return 'false';
		}
	}
	//if all conditions pass without a false, then return true
	return true;
}
SerialNumber(readline());
```

String Periods

Have the function `StringPeriods(str)` take the **`str`** parameter being passed and determine if there is some substring K that can be repeated N > 1 times to produce the input string exactly as it appears. Your program should return the longest substring K, and if there is none it should return the string **-1**.

For example: if **`str`** is "abcababcababcab" then your program should return **abcab** because that is the longest substring that is repeated 3 times to create the final string. Another example: if **`str`** is "abababababab" then your program should return **ababab** because it is the longest

substring. If the input string contains only a single character, your program should return the string **-1**.

Examples

Input: "abcxabc"
Output: -1

Input: "affedaaffed"
Output: -1

```
function StringPeriods(str) {
	// we will use only lengths of substrings that divide evenly into str
	const len = str.length;
	const pivot = Math.max(Math.trunc(Math.sqrt(len)), len);
	for (let i = 2; i <= pivot; i++) {
		if (len % i === 0) {
			const block = str.slice(0, len / i);
			if (block.repeat(i) === str) {
				return block;
			}
		}
	}
	return -1;
}

// keep this function call here
StringPeriods(readline());
```

Palindrome Swapper

Have the function `PalindromeSwapper(str)` take the **`str`** parameter being passed and determine if a palindrome can be created by swapping two adjacent characters in the string. If it is possible to create a palindrome, then your program should return the palindrome, if not then return the string **-1**. The input string will only contain alphabetic characters. For example: if **`str`** is "rcaecar" then you can create a palindrome by swapping the second and third characters, so your program should return the string **racecar** which is the final palindromic string.

Examples

Input: "anna"
Output: anna

Input: "kyaak"
Output: kayak

```
function PalindromeSwapper(str) {
  let inputArray = str.split('');
  let strLen = inputArray.length;
  let palTester = function(arr) {
    let len = arr.length;
    for (let i = 0; i < len; i++) {
      if (arr[i] !== arr[len - (1 + i)]) {
        return false;
      }
    }
    return true;
  }

  for (let i = 0; i < strLen - 1; i++) {
    let newArray = Array.from(inputArray);
    newArray.splice(i, 2, newArray[i + 1], newArray[i]);
    if (palTester(newArray)) {
      return newArray.join('');
    }
  }
  return -1;
}

// keep this function call here
PalindromeSwapper(readline());
```

Remove Brackets

Have the function `RemoveBrackets(str)` take the **`str`** string parameter being passed, which will contain only the characters "(" and ")", and determine the minimum number of brackets that need to be removed to create a string of correctly matched brackets. For example: if **`str`** is "(()))" then your program should return the number **1**. The answer could potentially be 0, and there will always be at least one set of matching brackets in the string.

Examples

Input: "(())((("
Output: 3

Input: "(()("
Output: 2

```
function RemoveBrackets(str) {
```

```
  let throwouts = 0;
  let counter = 0;
  let arr = str.split('');
  let len = arr.length;

  for (let i = 0; i < len; i++) {
    //a good example of the use of pre++ and pre-- as opposed to post++ and post--
    counter = arr[i] === '(' ? ++counter : --counter;
    if (counter < 0) {
      throwouts++;
      counter = 0;
    }
  }
  throwouts += counter;

  return throwouts;
}

// keep this function call here
RemoveBrackets(readline());
```

Command Line

Have the function `CommandLine(str)` take the **`str`** parameter being passed which represents the parameters given to a command in an old PDP system. The parameters are alphanumeric tokens (without spaces) followed by an equal sign and by their corresponding value. Multiple parameters/value pairs can be placed on the command line with a single space between each pair. Parameter **tokens** and **values** cannot contain equal signs but **values** can contain spaces. The purpose of the function is to isolate the parameters and values to return a list of parameter and value lengths. It must provide its result in the same format and in the same order by replacing each entry (tokens and values) by its corresponding length.

For example, if **`str`** is: "SampleNumber=3234 provider=Dr. M. Welby patient=John Smith priority=High" then your function should return the string "12=4 8=12 7=10 8=4" because "SampleNumber" is a 12 character token with a 4 character value ("3234") followed by "provider" which is an 8 character token followed by a 12 character value ("Dr. M. Welby"), etc.

Examples

Input: "letters=A B Z T numbers=1 2 26 20 combine=true"
Output: 7=7 7=9 7=4

Input: "a=3 b=4 a=23 b=a 4 23 c="
Output: 1=1 1=1 1=2 1=6 1=0

```
function CommandLine(str) {
  let splitter = /\s(?=\w+=)/
  let wordArr = str.split(splitter);

  wordArr = wordArr.map(pair => {
    let pairArr = pair.split('=');
    pairArr = pairArr.map(word => {
      return word.length.toString();
    });
    return pairArr.join('=');
  });

  return wordArr.join(' ');
}

// keep this function call here
CommandLine(readline());
```

Star Rating

Have the function `StarRating(str)` take the **`str`** parameter being passed which will be an average rating between 0.00 and 5.00, and convert this rating into a list of 5 image names to be displayed in a user interface to represent the rating as a list of stars and half stars. Ratings should be rounded up to the nearest half. There are 3 image file names available: "full.jpg", "half.jpg", "empty.jpg". The output will be the name of the 5 images (without the extension), from left to right, separated by spaces. For example: if **`str`** is "2.36" then this should be displayed by the following image:

So your program should return the string "full full half empty empty".

Examples

Input: "0.38"
Output: half empty empty empty empty

Input: "4.5"
Output: full full full full half

```
function StarRating(str) {
  let num = (Math.round(parseFloat(str) * 2)) / 2;
  let starString = '';
  let half = !(Math.trunc(num) === num);
```

```
  starString = starString + ' full'.repeat(num);
  if (half) {
    starString += ' half';
  }
  return (starString + ' empty'.repeat(5 - num)).trim();
}

// keep this function call here
StarRating(readline());
```