Compare the letters of two words

A function that returns the number of letters two words have in common

- only works with words of the same length
- regardless of position & upper/lower case
- repeat letters will match the number of times in common

Examples:

- PEA vs EAT: 2
- TREE vs TRUE: 3
- APE vs pea: 3

Improving Compare

Remember:

- "working" is not the end
- You program for other coders, not the computer
- Skimmability means no need to READ all the code

This works...technically

```
function compare( word, guess ) {
  var count=0;
  var obj={};
  for(let i=0; i<word.length; i++) {
    if(isNaN(obj[word[i].toLowerCase()])) {
      obj[word[i].toLowerCase()]=1;
    } else {
      obj[word[i].toLowerCase()]++;
    }
  }
  for(let i=0; i<guess.length; i++) {
    if(obj[guess[i].toLowerCase()] > 0) {
      obj[guess[i].toLowerCase()]--;
      count++;
    }
  }
  return count;
}
```

NEVER USE VAR

- `var` is for old engines, not modern
- prefer `const`
- use `let` if you have to reassign the value

```
function compare( word, guess ) {
  let count=0;
  const obj={};
  for(let i=0; i<word.length; i++) {
    if(isNaN(obj[word[i].toLowerCase()])) {
      obj[word[i].toLowerCase()]=1;
    } else {
      obj[word[i].toLowerCase()]++;
    }
  }
  //...
}
```

Visual space makes it easier to skim

- Just like text, use space to make it easier to skim.
- Use "paragraphs" - blank lines between ideas
- There is no reward for tiny squished code

```
function compare( word, guess ) {
  let count = 0;
  const obj = {};

  for( let i = 0; i < word.length; i++ ) {
    if( isNaN(obj[word[i].toLowerCase()]) ) {
      obj[word[i].toLowerCase()] = 1;
    } else {
      obj[word[i].toLowerCase()]++;
    }
  }
//...
}
```

Variable names are huge

- Variable and function names: main source of info!
- Name for what it holds/represents, not how
- No need to take out a few letters - just hurts

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let i = 0; i < word.length; i++ ) {
    if( isNaN(letterCount[word[i].toLowerCase()]) ) {
      letterCount[word[i].toLowerCase()] = 1;
    } else {
      letterCount[word[i].toLowerCase()]++;
    }
  }
//...
}
```

Variable Names are HARD

Bad Names:

- `obj`, `ary`, `tmp`, `str`
- `map`, `dict`, `len`, `list`
- anything spleled wrong

Usually Bad Names:

- `data`, `result`, `retval`, `count`

Do you actually need that index value?

- use `for..of` to get the value you care about (letter)

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word ) {
    if(isNaN(letterCount[letter.toLowerCase()])) {
      letterCount[letter.toLowerCase()] = 1;
    } else {
      letterCount[letter.toLowerCase()]++;
    }
  }
//...
}
```

Pull out and name values

- Particularly if they are repeated
- Often you can move logic out to another function
- DRY - Don't Repeat Yourself
- DRY - Don't Repeat Yourself

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word ) {
    const lower = letter.toLowerCase();
    if(isNaN(letterCount[lower])) {
      letterCount[lower] = 1;
    } else {
      letterCount[lower]++;
    }
  }
//...
}
```

Remove unneeded focus

- NOT about being **shorter**
- IS about **focus** of the eye

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word.toLowerCase() ) {
    if(isNaN(letterCount[letter])) {
      letterCount[letter] = 1;
    } else {
      letterCount[letter]++;
    }
  }
//...
}
```

Use Truthy/Falsy

- Improve skimmability
- Draw eye to important parts
    - not `===` or `isSomething`
- Remember: 0 is **falsy** (good here, not always)

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word.toLowerCase() ) {
    if( !letterCount[letter] ) {
      letterCount[letter] = 1;
    } else {
      letterCount[letter]++;
    }
  }
//...
}
```

Cautious use of Ternary Operator

- When assigning a value, can reduce "visual noise"
- ...or INCREASE visual noise
- Remember: Shorter is NOT the exact goal
- ...I'll pass this time

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word.toLowerCase() ) {
    letterCount[letter] = letterCount[letter] ? letterCount[letter] + 1 : 1;
  }
//...
}
```

Pull out logic into more functions

- creates list of instructions instead of math
- Good to make the code DRYer
- ...I'll pass this time

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  const increment = count => count ? count + 1 : 1;

  for( let letter of word.toLowerCase() ) {
    letterCount[letter] = increment(letterCount[letter]);
  }
//...
}
```

Not always post-inc/decrement

- ++ and -- aren't the only way to increase/decrease
- += 1 and -= 1 work, and allow for other numbers
- draw focus to what you're actually doing

```
function compare( word, guess ) {
  //.. some code above

  for( let letter of guess.toLowerCase()) {
    if( letterCount[letter] ) {
      letterCount[letter] -= 1;
      matches += 1;
    }
  }

  return matches;
}
```

Defaulting and Short-Circuiting

- `&&` and `||` short circuit
- `&&` and `||` return a value
    - Not just boolean: `foo = foo || 'default';`
- Often when an `if` checks for truthyness, and assign to value either way

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word.toLowerCase() ) {
    letterCount[letter] = letterCount[letter] + 1 || 1;
  }

//...
}
```

Before...

```
function compare( word, guess ) {
  var count=0;
  var obj={};
  for(let i=0; i<word.length; i++) {
    if(isNaN(obj[word[i].toLowerCase()])) {
      obj[word[i].toLowerCase()]=1;
    } else {
      obj[word[i].toLowerCase()]++;
    }
  }
  for(let i=0; i<guess.length; i++) {
    if(obj[guess[i].toLowerCase()] > 0) {
      obj[guess[i].toLowerCase()]--;
      count++;
    }
  }
  return count;
}
```

...and After

```
function compare( word, guess ) {
  let matches = 0;
  const letterCount = {};

  for( let letter of word.toLowerCase() ) {
    letterCount[letter] = letterCount + 1 || 1;
  }

  for( let letter of guess.toLowerCase() ) {
    if( letterCount[letter] ) {
      letterCount[letter] -= 1;
      matches += 1;
    }
  }

  return matches;
}
```

The right answer?

"What is the right answer?"

That depends

- I know of at least 3 "good" algorithms
    - Count letters in one, reduce for matches in other
    - Sort letters, traverse, count where one is less than other
    - Count letters in both, take min from both
- What is O()?
- What is easy to understand? maintain?
- What is easy to test?

Example 1

```
/* Convert to uppercase and sort the characters */
word = word.toUpperCase().split('').sort().join('');
guess = guess.toUpperCase().split('').sort().join('');

/* Compare each letter */
let res = 0;
let lastFoundIndex = -1;
for(let i = 0; i < guess.length; i++){
  for(let j = 0; j < word.length; j++){
    if(
      j !== lastFoundIndex
        && guess.charAt(i) === word.charAt(j)
      ){
      res++;
      lastFoundIndex = j;
      break;
    }
  }
}

return res;
```

From too little to too much

```
/*
Compare each letter:
Use two index variables traversing the two sorted words.
Keep comparing word[i] and guess[j],
if they are matched, move both indexes forward by 1.
Otherwise, move the index of the smaller letter and
continue to compare until any index reach the word length.
*/
```

- Comments should talk about WHY more than WHAT
- Comments should not repeat code
- Comments should NOT BE STALE
- Comments shouldn't be required

Example 1 Issues

- Doesn't actually work (EVERGREEN vs OVERWHELM)
- Very visually noisy
- Logic is unclear (have to read every line and figure out why)
- Poor variable names (`i`? `j`? `res`?)
    - `lastFoundIndex` is great however

Example 2

```
word = word.toUpperCase().split('').sort().join('');
guess = guess.toUpperCase().split('').sort().join('');

/* Compare the sorted letters in turn */
let matchCount = 0;
let i = 0;
let j = 0;
while(i < word.length && j < guess.length){
  if(word[i] === guess[j]){
    matchCount++;
    i++;
    j++;
  } else if(word[i] < guess[j]){
    i++;
  } else{
    j++;
  }
}

return matchCount;
```

Review

- Better comments
- Logic is more clear
    - Emphasis on 'sort' as important for algorithm
    - moving forward logic not as clear
- `i` and `j` still poor
    - unclear
    - hide that we're comparing letters!
- What if `i` was `wordIndex` and `j` was `guessIndex`?
- Arrays could call `unshift()`
    - can use named functions to describe actions

Summary

- Functions should try to be 1-15 lines
- Names should be meaningful even by themselves
- Skimmability is about managing **focus**
    - Avoid visual noise
    - Avoid "squishing"
- People will argue about how best to do this
    - ...just like with human languages

Summary - Part 2

Impacts your grade:

- Meaningful Names (**useful** meaning!)
    - Not `i`, `obj`, `tmp`
- Aim for skimmability
- Never use `var`; prefer `const`
- Always use strict comparison
    - Unless using truthy/falsyness