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

Array vs Object?

There are multiple approaches

First, let's review an approach using arrays

This works...technically

```
function compare(word,guess){
  var count=0;
  var ary1=word.toUpperCase().split('');
  var ary2=guess.toUpperCase().split('');

  for(var i=0;i<ary1.length;i++) {
    for(var j=0;j<ary2.length;j++) {
      if(ary1[i]===ary2[j]) {
        count++;
        ary2.splice(j,1); // Remove so it won't match again
      }
    }
  }
  return count;
}
```

Read the README

- `var` is for old engines, not modern
- prefer `const`, use `let` when you can't
- Notice how informative the `let` below becomes

```
function compare( word, guess ) {
  let count=0;
  const ary1=word.toUpperCase().split('');
  let ary2=guess.toUpperCase().split('');

  for(let i=0;i<ary1.length;i++){
    for(let j=0;j<ary2.length;j++){
      if(ary1[i]===ary2[j]){
        count++;
        ary2.splice(j,1); // Remove so it won't match again
      }
    }
  }
  return count;
}
```

Whitespace makes code better

- better without any logic changes

```
function compare( word, guess ) {
  let count = 0;
  const ary1 = word.toUpperCase().split('');
  let ary2 = guess.toUpperCase().split('');

  for (let i = 0; i < ary1.length; i++ ){
    for (let j = 0; j < ary2.length; j++ ){
      if( ary1[i] === ary2[j] ){
        count++;
        ary2.splice(j, 1); // Remove so it won't match again
      }
    }
  }
  return count;
}
```

Naming things is important

Good names let your code explain itself

- `count` is a count of what?
- `ary1`?
- `i`? `j`?

When these show up in logic, you have to mentally translate:

```
if ( ary1[i] === ary2[j] ) {
```

why not use the translated version right away?

```
if ( wordLetters[wordAt] === guessLetters[guessAt] ){
```

You will always be renaming

Each line is more clear, but now this looks messy

```
function compare( word, guess ) {
  let matched = 0;
  const wordLetters = word.toUpperCase().split('');
  let guessLetters = guess.toUpperCase().split('');

  for( let wordAt = 0; wordAt < wordLetters.length; wordAt++ ){
    for( let guessAt = 0 ; guessAt < guessLetters.length; guessAt++ ){
      if( wordLetters[wordAt] === guessLetters[guessAt] ){
        matched++;
        guessLetters.splice(guessAt,1);
      }
    }
  }
  return count;
}
```

Standard Array Methods are helpful!

There are solutions for common problems

- be sure to read up on the array methods on MDN

```
function compare( word, guess ) {
  let matched = 0;
  const wordLetters = word.toUpperCase().split('');
  let guessLetters = guess.toUpperCase().split('');

  for( let wordAt = 0; wordAt< wordLetters.length; wordAt++ ){
    const matchAt = guessLetters.indexOf(wordLetters[wordAt]);
    if( matchAt > -1 ) {
      matched++;
      guessLetters.splice(matchAt,1);
    }
  }
  return count;
}
```

For loops

A C-style for loop, but we don't care about the index!

```
function compare( word, guess ) {
  let matched = 0;
  const wordLetters = word.toUpperCase().split('');
  let guessLetters = guess.toUpperCase().split('');

  for( let letter of wordLetters ){
    const matchAt = guessLetters.indexOf(letter);
    if( matchAt > -1 ) {
      matched++;
      guessLetters.splice(matchAt,1);
    }
  }
  return count;
}
```

Comment: `for...in` for arrays is...unusual

Before

```
function compare(word,guess){
  var count=0;
  var ary1=word.toUpperCase().split('');
  var ary2=guess.toUpperCase().split('');

  for(var i=0;i<ary1.length;i++) {
    for(var j=0;j<ary2.length;j++) {
      if(ary1[i]===ary2[j]) {
        count++;
        ary2.splice(j,1); // Remove so it won't match again
      }
    }
  }
  return count;
}
```

After

```
function compare( word, guess ) {
  let matched = 0;
  const wordLetters = word.toUpperCase().split('');
  let guessLetters = guess.toUpperCase().split('');

  for( let letter of wordLetters ){
    const matchAt = guessLetters.indexOf(letter);
    if( matchAt > -1 ) {
      matched++;
      guessLetters.splice(matchAt,1);
    }
  }
  return count;
}
```

What it does is more clear

- Not about length

Common Concerns

- Do we worry about O()?
    - Performance?
    - Efficiency?
    - Time and Memory complexity?

Dev Complexity

Dirty Secret:

- School teaches performance
    - so it becomes habit
    - and we stop worrying
- Outside of truly restricted resources
    - We tend to not worry until n^2
    - Developer Efficiency often a bigger deal
        - More costly than hardware
        - Keep it easy to understand/change/write

But nested arrays ARE n^2 (or near)!

Object Approach

Different algorithm concept

- Not "compare each letter of one word to another"
- Use "count letters, then compare"

Counting letters of a word

```
const wordLetters = {};

for( let letter of word ) {
  if( wordLetters[letter] ) {
    wordLetters[letter] += 1;
  } else {
    wordLetters[letter] = 1;
  }
}
```

Example: `GEESE`

```
{ G: 1, E: 3, S, 1 }
```

- Object lets us read count of a letter in O(1) time
- Loop, but not nested

This works...technically

```
function compare( word, guess ) {
  let matched = 0;
  const obj = {};
  for(let i=0; i<word.length; i++) {
    if(obj[word[i].toLowerCase()]===undefined)) {
      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;
}
```

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( obj[word[i].toLowerCase()] === undefined ) {
      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( letterCount[word[i].toLowerCase()] === undefined ) {
      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
    - VERY COMMON!
- `i`, `j`
    - what is it?

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( letterCount[letter.toLowerCase()] === undefined ) {
      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( letterCount[lower]) === undefined ) {
      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( letterCount[letter] === undefined ) {
      letterCount[letter] = 1;
    } else {
      letterCount[letter]++;
    }
  }
//...
}
```

Use Truthy/Falsy

- Improve skimmability
- Draw eye to important parts
    - not `===` or `isSomething`
- Remember: 0 is **falsy** (fine 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 Conditional 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
    - Avoid Hasty Abstractions (AHA)
        - AHA opposes excessive DRYness

```
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?"

It depends :)

- What is easy to understand?
- ...maintain/change/adjust?
- ...test and confirm?
- What is O()?

Latest favorite

```
function compare( word, guess ) {
  function letterCountsOf( word ) {
    const letterCounts = {};

    word.toUpperCase().split('').forEach( letter => {
      letterCounts[letter] = letterCounts[letter] + 1 || 1;
    });

    return letterCounts;
  }

  const wordCounts = letterCountsOf(word);
  const guessCounts = letterCountsOf(guess);
  let matched = 0;

  for( let letter in guessCounts ){
    const wordCount = wordCounts[letter] || 0;
    const guessCount = guessCounts[letter] || 0;
    matched += Math.min( wordCount, guessCount );
  }
  return matched;
}
```

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