1.

4 kyu **Vigenère Cipher Helper**

787 130 90% of 977 3,062 of 7,245 jacobb 7 Issues Reported

Instructions Output

The Vigenère cipher is a classic cipher originally developed by Italian cryptographer Giovan Battista Bellaso and published in 1553. It is named after a later French cryptographer Blaise de Vigenère, who had developed a stronger autokey cipher (a cipher that incorporates the message of the text into the key).

The cipher is easy to understand and implement, but survived three centuries of attempts to break it, earning it the nickname "le chiffre indéchiffrable" or "the indecipherable cipher."

From Wikipedia:

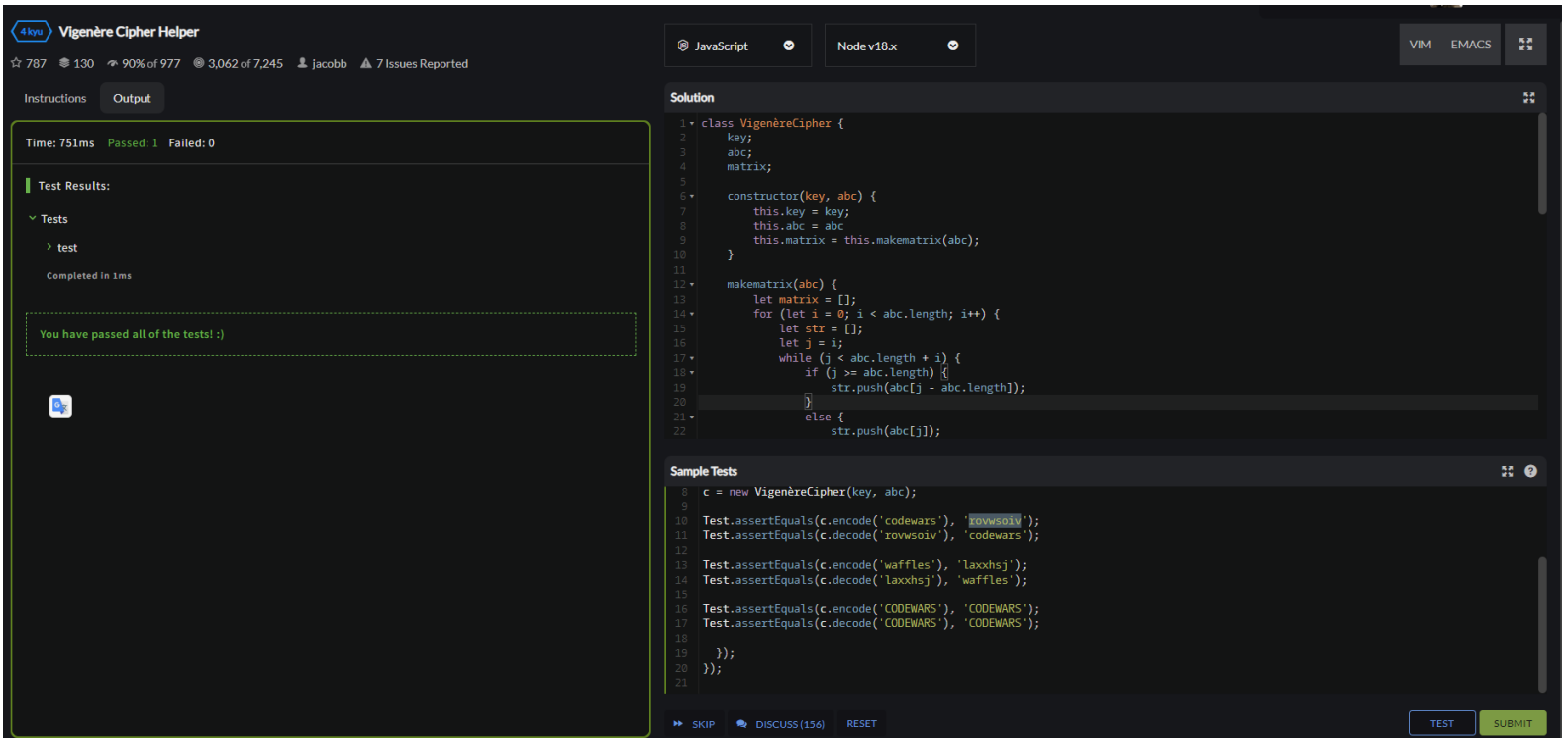
> *"The Vigenère cipher is a method of encrypting alphabetic text by using a series of different Caesar ciphers based on the letters of a keyword. It is a simple form of polyalphabetic substitution.*
>
> ...
>
> *In a Caesar cipher, each letter of the alphabet is shifted along some number of places; for example, in a Caesar cipher of shift `3`, `A` would become `D`, `B` would become `E`, `Y` would become `B` and so on. The Vigenère cipher consists of several Caesar ciphers in sequence with different shift values."*

Assume the key is repeated for the length of the text, character by character. Note that some implementations repeat the key over characters only if they are part of the alphabet -- **this is not the case here.**

The shift is derived by applying a Caesar shift to a character with the corresponding index of the key in the alphabet.

```
class VigenèreCipher {
  key;
  abc;
  matrix;

  constructor(key, abc) {
    this.key = key;
    this.abc = abc
    this.matrix = this.makematrix(abc);
  }

  makematrix(abc) {
    let matrix = [];
    for (let i = 0; i < abc.length; i++) {
      let str = [];
      let j = i;
      while (j < abc.length + i) {
        if (j >= abc.length) {
          str.push(abc[j - abc.length]);
        }
        else {
          str.push(abc[j]);
        }
        j++;
      }
      matrix.push(str)
    }
    return matrix;
  }

  makekeystream(str) {
    let keystream = "";
```

```
    let length = str.length;
    while (this.key.length < length) {
        keystream += this.key;
        length -= this.key.length;
    }
    keystream = keystream + this.key.slice(0, length);
    return keystream;
}


encode(str) {
    let keystream = this.makekeystream(str);
    let newStr = "";
    for (let i = 0; i < str.length; i++) {
        if (!this.abc.includes(str[i])) {
            newStr += str[i]
        }
        else {
            newStr += this.matrix[this.abc.indexOf(str[i])][this.abc.indexOf(keystream[i])];
        }
    }
    return newStr;
}

decode(str) {
    let keystream = this.makekeystream(str);
    let newStr = "";
    for (let i = 0; i < str.length; i++) {
        if (!this.abc.includes(str[i])) {
            newStr += str[i]
        }
        else {
            newStr += this.abc[this.matrix[this.abc.indexOf(keystream[i])].indexOf(str[i])];
        }
    }
    return newStr;
}
}
```



2.

5 kyu **Vector class**

562 | 122 | 88% of 820 | 2,966 of 7,237 | eugene-bulkin | 8 Issues Reported

Instructions | Output | Past Solutions

Create a Vector object that supports addition, subtraction, dot products, and norms. So, for example:

```
var a = new Vector([1, 2, 3]);
var b = new Vector([3, 4, 5]);
var c = new Vector([5, 6, 7, 8]);

a.add(b);      // should return a new Vector([4, 6, 8])
a.subtract(b); // should return a new Vector([-2, -2, -2])
a.dot(b);      // should return 1*3 + 2*4 + 3*5 = 26
a.norm();      // should return sqrt(1^2 + 2^2 + 3^2) = sqrt(14)
a.add(c);      // throws an error
```

If you try to add, subtract, or dot two vectors with different lengths, ***you must throw an error***!

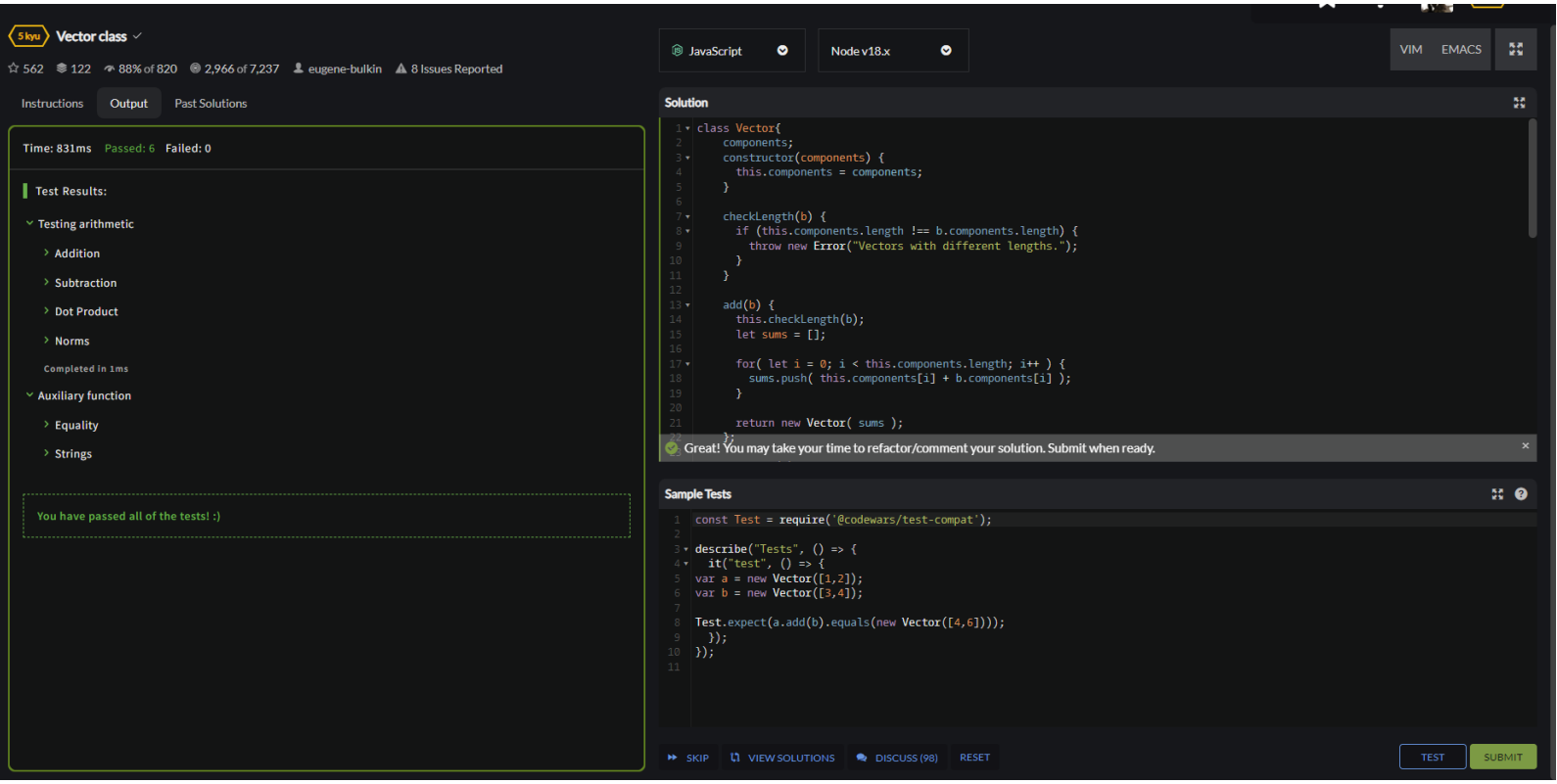
Also provide:

- a `toString` method, so that using the vectors from above, `a.toString() === '(1,2,3)'` (in Python, this is a `__str__` method, so that `str(a) == '(1,2,3)'`)
- an `equals` method, to check that two vectors that have the same components are equal

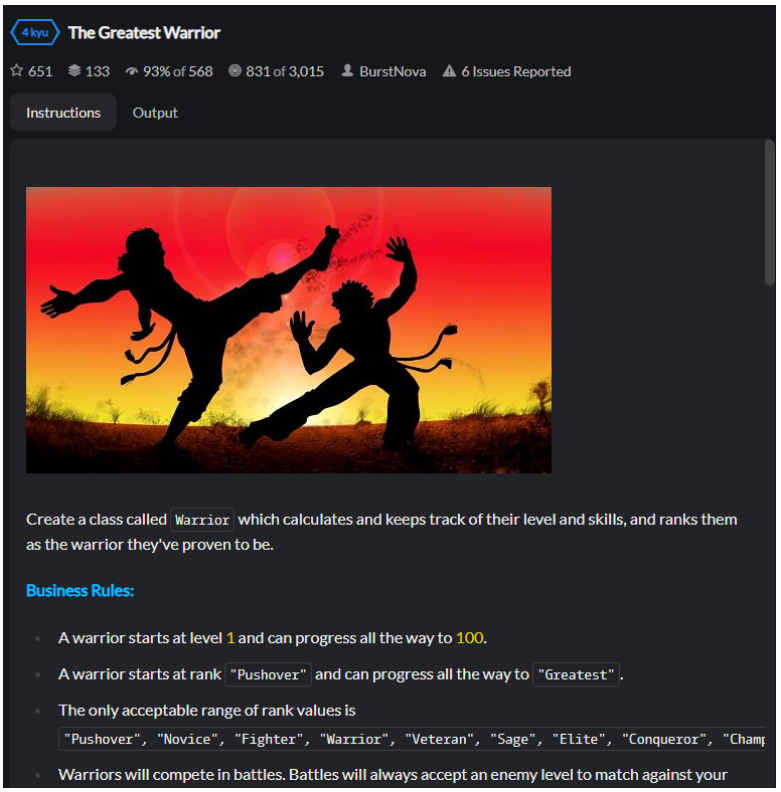
Note: the test cases will utilize the user-provided `equals` method.

```
class Vector{
  components;
  constructor(components) {
    this.components = components;
  }

  checkLength(b) {
    if (this.components.length !== b.components.length) {
      throw new Error("Vectors with different lengths.");
    }
  }

  add(b) {
    this.checkLength(b);
    let sums = [];

    for( let i = 0; i < this.components.length; i++ ) {
      sums.push( this.components[i] + b.components[i] );
    }

    return new Vector( sums );
  };

  subtract(b){
    this.checkLength(b);
    let differences = [];

    for( let i = 0;  i < this.components.length; i++ ) {
      differences.push( this.components[i] - b.components[i] );
```

```
  }

  return new Vector( differences );
 };

 dot(b) {
  this.checkLength(b);
  let products = 0;

  for( let i = 0; i < this.components.length; i++ ) {
    products += this.components[i] * b.components[i];
  }
   return products;
 };

 norm(){
  return Math.sqrt(this.dot(this));
 };

 equals(b){
  if ( this.components.length !== b.components.length ) {
  return false;
 }

  for ( let i = 0; i < this.components.length; i++ ) {
    if ( this.components[ i ] !== b.components[ i ] ) {
      return false;
    }
  }

  return true;
 };

 toString(b){
  return '(' + this.components.join(',') + ')';
 };

};
```

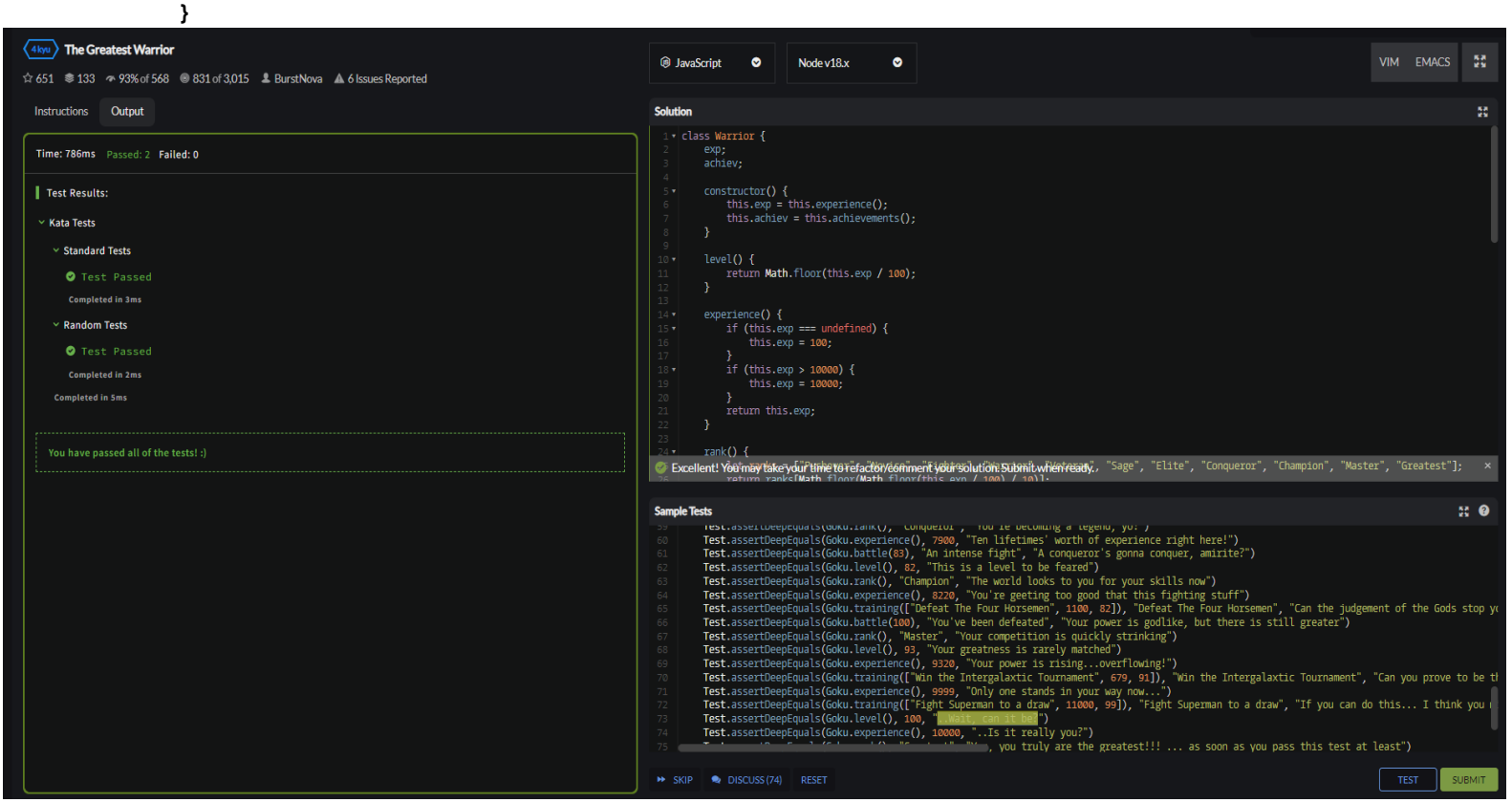


3.




```
class Warrior {
   exp;
   achiev;

   constructor() {
      this.exp = this.experience();
      this.achiev = this.achievements();
   }

   level() {
      return Math.floor(this.exp / 100);
   }

   experience() {
      if (this.exp === undefined) {
         this.exp = 100;
      }
      if (this.exp > 10000) {
         this.exp = 10000;
      }
      return this.exp;
   }

   rank() {
      let ranks = ["Pushover", "Novice", "Fighter", "Warrior", "Veteran", "Sage", "Elite", "Conqueror", "Champion",
"Master", "Greatest"];
      return ranks[Math.floor(Math.floor(this.exp / 100) / 10)];
   }

   achievements() {
      if (this.exp === 100) {
         this.achiev = [];
      }
      return this.achiev;
   }
```

```
    training([name, reward_exp, min_lvl]) {
        let lvl = Math.floor(this.exp / 100);
        if (lvl >= min_lvl) {
            this.exp += reward_exp;
            this.experience();
            this.achiev.push(name);
            return name
        } else {
            return "Not strong enough";
        }
    }

    battle(lvl_enemy) {
        let lvl = Math.floor(this.exp / 100);
        let msg;
        if (1 <= lvl_enemy && lvl_enemy <= 100) {
            if (lvl_enemy === lvl) {
                this.exp += 10;
                msg = "A good fight";
            } else if (lvl_enemy === lvl - 1) {
                this.exp += 5;
                msg = "A good fight";
            } else if (lvl_enemy <= lvl - 2) {
                this.exp += 0;
                msg = "Easy fight";
            } else if (lvl_enemy > lvl) {
                if (Math.floor(lvl / 10) < Math.floor(lvl_enemy / 10) && lvl_enemy >= lvl + 5) {
                    msg = "You've been defeated";
                }
                else {
                    let diff = lvl_enemy - lvl;
                    this.exp += 20 * diff * diff;
                    msg = "An intense fight";
                }
            }
        }
        else {
            msg = "Invalid level";
        }
        this.experience();
        return msg;
    }
}
```



4.

5 kyu **PaginationHelper**

1653 314 81% of 3,149 12,386 of 34,221 jhoffner 8 Issues Reported

Instructions Output Past Solutions

For this exercise you will be strengthening your page-fu mastery. You will complete the PaginationHelper class, which is a utility class helpful for querying paging information related to an array.

The class is designed to take in an array of values and an integer indicating how many items will be allowed per each page. The types of values contained within the collection/array are not relevant.

The following are some examples of how this class is used:

```
var helper = new PaginationHelper(['a','b','c','d','e','f'], 4);
helper.pageCount(); // should == 2
helper.itemCount(); // should == 6
helper.pageItemCount(0); // should == 4
helper.pageItemCount(1); // last page - should == 2
helper.pageItemCount(2); // should == -1 since the page is invalid

// pageIndex takes an item index and returns the page that it belongs on
helper.pageIndex(5); // should == 1 (zero based index)
helper.pageIndex(2); // should == 0
helper.pageIndex(20); // should == -1
helper.pageIndex(-10); // should == -1
```

```
class PaginationHelper {
  itemsPerPage;
  collection;

  constructor(collection, itemsPerPage) {
    this.collection = collection;
    this.itemsPerPage = itemsPerPage;
  }

  itemCount() {
    return this.collection.length;
  }

  pageCount() {
    let counter = 1;
    let length = this.itemCount();
    if (length === 0) {
      return 0;
    }
    while (length > this.itemsPerPage) {
      length -= this.itemsPerPage;
      counter++;
    }
    return counter;
  }

  pageItemCount(pageIndex) {
    let pg_count = this.pageCount(this.length)
    let length = this.itemCount();

    if (pageIndex + 1 > pg_count || pageIndex < 0) {
```

```
            return -1;
        } else if (pageIndex + 1 < pg_count) {
            return this.itemsPerPage;
        } else if (pageIndex + 1 === pg_count) {
            if (length % this.itemsPerPage === 0) {
                return this.itemsPerPage
            } else {
                return length % this.itemsPerPage;
            }
        }
    }

    pageIndex(itemIndex) {
        let length = this.itemCount();
        if (itemIndex >= 0 && itemIndex < length) {
            return Math.floor(itemIndex / this.itemsPerPage);
        } else {
            return -1;
        }
    }
}
```

5 kyu PaginationHelper

1653 314 81% of 3,149 12,386 of 34,221 jhoffner 8 Issues Reported

Instructions Output Past Solutions

Time: 835ms Passed: 102 Failed: 0 Show All

Test Results:

Tests suite

- sample test : 24 items with 10 per page
- empty collection
- random tests
  - for itemCount = 70 itemsPerPage = 123
    - Test Passed
  - for itemCount = 88 itemsPerPage = 176
  - for itemCount = 67 itemsPerPage = 6
  - for itemCount = 27 itemsPerPage = 1
  - for itemCount = 19 itemsPerPage = 32
  - for itemCount = 10 itemsPerPage = 4
  - for itemCount = 56 itemsPerPage = 39
  - for itemCount = 66 itemsPerPage = 12
  - for itemCount = 67 itemsPerPage = 117
  - for itemCount = 67 itemsPerPage = 10
  - for itemCount = 91 itemsPerPage = 73
  - for itemCount = 16 itemsPerPage = 10
  - for itemCount = 20 itemsPerPage = 32
  - for itemCount = 9 itemsPerPage = 7

JavaScript Node v18.x VIM EMACS

Solution

```javascript
class PaginationHelper {
    itemsPerPage;
    collection;

    constructor(collection, itemsPerPage) {
        this.collection = collection;
        this.itemsPerPage = itemsPerPage;
    }

    itemCount() {
        return this.collection.length;
    }

    pageCount() {
        let counter = 1;
        let length = this.itemCount();
        if (length === 0) {
            return 0;
        }
        while (length > this.itemsPerPage) {
            length -= this.itemsPerPage;
            counter++;
```

Impressive! You may take your time to refactor/comment your solution. Submit when ready.

Sample Tests

```javascript
    doTest(helper, 'pageIndex', -1, -1);
    doTest(helper, 'pageIndex', -1, -23);
    doTest(helper, 'pageIndex', -1, -15);
  });

  it ('empty collection', () => {
    const empty = new PaginationHelper([], 10);

    doTest(empty, 'pageCount', 0);
    doTest(empty, 'itemCount', 0);
    doTest(empty, 'pageIndex', -1, 0);
    doTest(empty, 'pageItemCount', -1, 0);
  });
});
```

SKIP VIEW SOLUTIONS DISCUSS (265) RESET TEST SUBMIT