Ejercicios Javascript

Validate PIN

ATM machines allow 4 or 6 digit PIN codes and PIN codes cannot contain anything but **exactly** 4 digits or exactly 6 digits.
If the function is passed a valid PIN string, return `true`, else return `false`.

Examples (Input --> Output)

```
"1234"   -->  true
"12345"  -->  false
"a234"   -->  false
```

Base function

```
function validatePIN (pin) {
  // evaluate and return true or false
}
```

Tests to run

```
describe("validatePIN", function() {
  it("should return False for pins with length other than 4 or 6", function()
{
    Test.assertEquals(validatePIN("1"),false, "Wrong output for '1'")
    Test.assertEquals(validatePIN("123"),false, "Wrong output for '123'")
    Test.assertEquals(validatePIN("1234567"),false, "Wrong output for
'1234567'")
    Test.assertEquals(validatePIN("00000000"),false, "Wrong output for
'00000000'")
  });

  it("should return False for pins which contain characters other than
digits", function() {
    Test.assertEquals(validatePIN("a234"),false, "Wrong output for 'a234'")
    Test.assertEquals(validatePIN(".234"),false, "Wrong output for '.234'")
  });

  it("should return True for valid pins", function() {
    Test.assertEquals(validatePIN("1234"),true, "Wrong output for '1234'");
    Test.assertEquals(validatePIN("1111"),true, "Wrong output for '1111'");
    Test.assertEquals(validatePIN("123456"),true, "Wrong output for
'123456'");
    Test.assertEquals(validatePIN("098765"),true, "Wrong output for
'098765'");
    Test.assertEquals(validatePIN("123456"),true, "Wrong output for
'123456'");
  });
});
```

Persistent Bugger

Write a function, `persistence`, that takes in a positive parameter `num` and returns its multiplicative persistence, which is the number of times you must multiply the digits in `num` until you reach a single digit.

Examples (Input --> Output)

```
 persistence(39) === 3 // because 3*9 = 27, 2*7 = 14, 1*4=4
                       // and 4 has only one digit

 persistence(999) === 4 // because 9*9*9 = 729, 7*2*9 = 126,
                        // 1*2*6 = 12, and finally 1*2 = 2

 persistence(4) === 0 // because 4 is already a one-digit number
```

Base function

```
function persistence(num) {
   //code me
}
```

Tests to run

```
describe('Initial Tests', function () {
  Test.assertEquals(persistence(39),3);
  Test.assertEquals(persistence(4),0);
  Test.assertEquals(persistence(25),2);
  Test.assertEquals(persistence(999),4);
});
```

Find the missing letter

Write a method that takes an array of consecutive (increasing) letters as input and that returns the missing letter in the array.
You will always get an valid array. And it will be always exactly one letter be missing. The length of the array will always be at least 2.
The array will always contain letters in only one case. (Use the English alphabet with 26 letters!)

Examples (Input --> Output)

```
['a','b','c','d','f'] -> 'e'
['O','Q','R','S'] -> 'P'
["a","b","c","d","f"] -> "e"
["O","Q","R","S"] -> "P"
```

Base function

```
function findMissingLetter(array)
{
  return ' ';
}
```

Tests to run

```
describe("FindMissingLetterTests", function(){
  it("exampleTests", function(){
    Test.assertEquals(findMissingLetter(['a','b','c','d','f']), 'e');
    Test.assertEquals(findMissingLetter(['O','Q','R','S']), 'P');
  });
});
```

Array.diff

Implement a difference function, which subtracts one list from another and returns the result. It should remove all values from list `a`, which are present in list `b` keeping their order.

Examples (Input --> Output)

```
arrayDiff([1,2],[1]) --> [2]
arrayDiff([1,2,2,2,3],[2]) --> [1,3]
```

Base function

```
function arrayDiff(a, b) {
  // your code goes here
}
```

Tests to run

```
describe("Sample tests", function() {
  it("Array diff Sample tests", function() {
    Test.assertDeepEquals(arrayDiff([], [4,5]), [], "a was [], b was [4,5]");
    Test.assertDeepEquals(arrayDiff([3,4], [3]), [4], "a was [3,4], b was 
[3]");
    Test.assertDeepEquals(arrayDiff([1,8,2], []), [1,8,2], "a was [1,8,2], b 
was []");
    Test.assertDeepEquals(arrayDiff([1,2,3], [1,2]), [3], "a was [1,2,3], b 
was [1,2]")
  });
});
```