Ejercicios Python

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

```
def validate_pin(pin):
    #return true or false
```

Tests to run

```
def run_tests():
    @test.it("should return False for pins with length other than 4 or 6")
    def basic_test_cases():
        test.assert_equals(validate_pin("1"),False, "Wrong output for '1'")
        test.assert_equals(validate_pin("12"),False, "Wrong output for '12'")
        test.assert_equals(validate_pin("123"),False, "Wrong output for
'123'")
        test.assert_equals(validate_pin("12345"),False, "Wrong output for
'12345'")
        test.assert_equals(validate_pin("1234567"),False, "Wrong output for
'1234567'")
        test.assert_equals(validate_pin("-1234"),False, "Wrong output for '-
1234'")
        test.assert_equals(validate_pin("-12345"),False, "Wrong output for '-
12345'")
        test.assert_equals(validate_pin("1.234"),False, "Wrong output for
'1.234'")
        test.assert_equals(validate_pin("00000000"),False, "Wrong output for
'00000000'")

    @test.it("should return False for pins which contain characters other
than digits")
    def _():
        test.assert_equals(validate_pin("a234"),False, "Wrong output for
'a234'")
        test.assert_equals(validate_pin(".234"),False, "Wrong output for
'.234'")

    @test.it("should return True for valid pins")
    def _():
```

```
        test.assert_equals(validate_pin("1234"),True, "Wrong output for
'1234'")
        test.assert_equals(validate_pin("0000"),True, "Wrong output for
'0000'")
        test.assert_equals(validate_pin("1111"),True, "Wrong output for
'1111'")
        test.assert_equals(validate_pin("123456"),True, "Wrong output for
'123456'")
        test.assert_equals(validate_pin("098765"),True, "Wrong output for
'098765'")
        test.assert_equals(validate_pin("000000"),True, "Wrong output for
'000000'")
        test.assert_equals(validate_pin("123456"),True, "Wrong output for
'123456'")
        test.assert_equals(validate_pin("090909"),True, "Wrong output for
'090909'")
```

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)

```
39 --> 3 (because 3*9 = 27, 2*7 = 14, 1*4 = 4 and 4 has only one digit)
999 --> 4 (because 9*9*9 = 729, 7*2*9 = 126, 1*2*6 = 12, and finally 1*2 = 2)
4 --> 0 (because 4 is already a one-digit number)
```

Base function

```
def persistence(n):
    # your code
```

Tests to run

```
def fixed_tests():
    @test.it('Basic Test Cases')
    def basic_test_cases():
        test.assert_equals(persistence(39), 3)
        test.assert_equals(persistence(4), 0)
        test.assert_equals(persistence(25), 2)
        test.assert_equals(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"
```

Base function

```
def find_missing_letter(chars):
    return
```

Tests to run

```
test.describe("find_missing_letter tests")
test.assert_equals(find_missing_letter(['a','b','c','d','f']), 'e')
test.assert_equals(find_missing_letter(['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)

```
array_diff([1,2],[1]) == [2]
```

If a value is present in b, all of its occurrences must be removed from the other:

```
array_diff([1,2,2,2,3],[2]) == [1,3]
```

Base function

```
def array_diff(a, b):
    #your code here
```

Tests to run

```
@test.describe("Fixed Tests")
def fixed_tests():
    @test.it('Basic Test Cases')
    def basic_test_cases():
        test.assert_equals(array_diff([1,2], [1]), [2], "a was [1,2], b was
[1], expected [2]")
        test.assert_equals(array_diff([1,2,2], [1]), [2,2], "a was [1,2,2], b
was [1], expected [2,2]")
        test.assert_equals(array_diff([1,2,2], [2]), [1], "a was [1,2,2], b
was [2], expected [1]")
        test.assert_equals(array_diff([1,2,2], []), [1,2,2], "a was [1,2,2],
b was [], expected [1,2,2]")
        test.assert_equals(array_diff([], [1,2]), [], "a was [], b was [1,2],
expected []")
        test.assert_equals(array_diff([1,2,3], [1, 2]), [3], "a was [1,2,3],
b was [1, 2], expected [3]")
```