**More Problems!**

The following problems do not have specs and cover many different topics. You'll want to follow the prompt for each problem and make sure your implementation satisfies the behavior in the given examples. Create a .rb file to work in and test your code manually by running ruby <your-file-name>.rb!

**General Problems**

**no\_dupes?**

Write a method no\_dupes?(arr) that accepts an array as an arg and returns an new array containing the elements that were not repeated in the array.

# Examples

no\_dupes?([1, 1, 2, 1, 3, 2, 4]) # => [3, 4]

no\_dupes?(['x', 'x', 'y', 'z', 'z']) # => ['y']

no\_dupes?([true, true, true]) # => []

**no\_consecutive\_repeats?**

Write a method no\_consecutive\_repeats?(arr) that accepts an array as an arg. The method should return true if an element never appears consecutively in the array; it should return false otherwise.

# Examples

no\_consecutive\_repeats?(['cat', 'dog', 'mouse', 'dog']) # => true

no\_consecutive\_repeats?(['cat', 'dog', 'dog', 'mouse']) # => false

no\_consecutive\_repeats?([10, 42, 3, 7, 10, 3]) # => true

no\_consecutive\_repeats?([10, 42, 3, 3, 10, 3]) # => false

no\_consecutive\_repeats?(['x']) # => true

**char\_indices**

Write a method char\_indices(str) that takes in a string as an arg. The method should return a hash containing characters as keys. The value associated with each key should be an array containing the indices where that character is found.

# Examples

char\_indices('mississippi') # => {"m"=>[0], "i"=>[1, 4, 7, 10], "s"=>[2, 3, 5, 6], "p"=>[8, 9]}

char\_indices('classroom') # => {"c"=>[0], "l"=>[1], "a"=>[2], "s"=>[3, 4], "r"=>[5], "o"=>[6, 7], "m"=>[8]}

**longest\_streak**

Write a method longest\_streak(str) that accepts a string as an arg. The method should return the longest streak of consecutive characters in the string. If there are any ties, return the streak that occurs later in the string.

# Examples

longest\_streak('a') # => 'a'

longest\_streak('accccbbb') # => 'cccc'

longest\_streak('aaaxyyyyyzz') # => 'yyyyy

longest\_streak('aaabbb') # => 'bbb'

longest\_streak('abc') # => 'c'

**bi\_prime?**

Write a method bi\_prime?(num) that accepts a number as an arg and returns a boolean indicating whether or not the number is a bi-prime. A bi-prime is a positive integer that can be obtained by multiplying two prime numbers.

For Example:

* 14 is a bi-prime because 2 \* 7
* 22 is a bi-prime because 2 \* 11
* 25 is a bi-prime because 5 \* 5
* 24 is not a bi-prime because no two prime numbers have a product of 24

# Examples

bi\_prime?(14) # => true

bi\_prime?(22) # => true

bi\_prime?(25) # => true

bi\_prime?(94) # => true

bi\_prime?(24) # => false

bi\_prime?(64) # => false

**vigenere\_cipher**

A Caesar cipher takes a word and encrypts it by offsetting each letter in the word by a fixed number, called the key. Given a key of 3, for example: a -> d, p -> s, and y -> b.

A Vigenere Cipher is a Caesar cipher, but instead of a single key, a sequence of keys is used. For example, if we encrypt "bananasinpajamas" with the key sequence [1, 2, 3], then the result would be "ccqbpdtkqqcmbodt":

# Message: b a n a n a s i n p a j a m a s

# Keys: 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1

# Result: c c q b p d t k q q c m b o d t

Write a method vigenere\_cipher(message, keys) that accepts a string and a key-sequence as args, returning the encrypted message. Assume that the message consists of only lowercase alphabetic characters.

# Examples

vigenere\_cipher("toerrishuman", [1]) # => "upfssjtivnbo"

vigenere\_cipher("toerrishuman", [1, 2]) # => "uqftsktjvobp"

vigenere\_cipher("toerrishuman", [1, 2, 3]) # => "uqhstltjxncq"

vigenere\_cipher("zebra", [3, 0]) # => "ceerd"

vigenere\_cipher("yawn", [5, 1]) # => "dbbo"

**vowel\_rotate**

Write a method vowel\_rotate(str) that accepts a string as an arg and returns the string where every vowel is replaced with the vowel that appears before it sequentially in the original string. The first vowel of the string should be replaced with the last vowel.

# Examples

vowel\_rotate('computer') # => "cempotur"

vowel\_rotate('oranges') # => "erongas"

vowel\_rotate('headphones') # => "heedphanos"

vowel\_rotate('bootcamp') # => "baotcomp"

vowel\_rotate('awesome') # => "ewasemo"

**Proc Problems**

**String#select**

Extend the string class by defining a String#select method that accepts a block. The method should return a new string containing characters of the original string that return true when passed into the block. If no block is passed, then return the empty string. Do not use the built-in Array#select in your solution.

# Examples

"app academy".select { |ch| !"aeiou".include?(ch) } # => "pp cdmy"

"HELLOworld".select { |ch| ch == ch.upcase } # => "HELLO"

"HELLOworld".select # => ""

**String#map!**

Extend the string class by defining a String#map! method that accepts a block. The method should modify the existing string by replacing every character with the result of calling the block, passing in the original character and it's index. Do not use the built-in Array#map or Array#map! in your solution.

# Examples

word\_1 = "Lovelace"

word\_1.map! do |ch|

if ch == 'e'

'3'

elsif ch == 'a'

'4'

else

ch

end

end

p word\_1 # => "Lov3l4c3"

word\_2 = "Dijkstra"

word\_2.map! do |ch, i|

if i.even?

ch.upcase

else

ch.downcase

end

end

p word\_2 # => "DiJkStRa"

**Recursion Problems**

**multiply**

Write a method multiply(a, b) that takes in two numbers and returns their product.

* You must solve this recursively (no loops!)
* You must not use the multiplication (\*) operator

# Examples

multiply(3, 5) # => 15

multiply(5, 3) # => 15

multiply(2, 4) # => 8

multiply(0, 10) # => 0

multiply(-3, -6) # => 18

multiply(3, -6) # => -18

multiply(-3, 6) # => -18

**lucas\_sequence**

The Lucas Sequence is a sequence of numbers. The first number of the sequence is 2. The second number of the Lucas Sequence is 1. To generate the next number of the sequence, we add up the previous two numbers. For example, the first six numbers of the sequence are: 2, 1, 3, 4, 7, 11, ...

Write a method lucasSequence that accepts a number representing a length as an arg. The method should return an array containing the Lucas Sequence up to the given length. Solve this recursively.

# Examples

lucas\_sequence(0) # => []

lucas\_sequence(1) # => [2]

lucas\_sequence(2) # => [2, 1]

lucas\_sequence(3) # => [2, 1, 3]

lucas\_sequence(6) # => [2, 1, 3, 4, 7, 11]

lucas\_sequence(8) # => [2, 1, 3, 4, 7, 11, 18, 29]

**prime\_factorization**

[The Fundamental Theorem of Arithmetic](https://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic) states that every positive integer is either a prime or can be represented as a product of prime numbers.

Write a method prime\_factorization(num) that accepts a number and returns an array representing the prime factorization of the given number. This means that the array should contain only prime numbers that multiply together to the given num. The array returned should contain numbers in ascending order. Do this recursively.

# Examples

prime\_factorization(12) # => [2, 2, 3]

prime\_factorization(24) # => [2, 2, 2, 3]

prime\_factorization(25) # => [5, 5]

prime\_factorization(60) # => [2, 2, 3, 5]

prime\_factorization(7) # => [7]

prime\_factorization(11) # => [11]

prime\_factorization(2017) # => [2017]

**✔︎ Submit Project**