**Meddling Medley**

This problem set will test your knowledge about the last few topics you explored. Like usual, the difficulty will ramp up as work through the set. Ready? Let's do it.

The solution is available [here](https://aao-alpha.s3-us-west-1.amazonaws.com/assets/topics/additional_problems/meddling_medley_solution.rb.zip), but please do not look at it until you have attempted all of the problems!

**Phase 1: Modest problems.**

**duos**

Write a method duos that accepts a string as an argument and returns the count of the number of times the same character appears consecutively in the given string.

Examples

p duos('bootcamp') # 1

p duos('wxxyzz') # 2

p duos('hoooraay') # 3

p duos('dinosaurs') # 0

p duos('e') # 0

**sentence\_swap**

Write a method sentence\_swap that accepts a sentence and a hash as arguments. The method should return a new sentence where every word is replaced with it's corresponding value in the hash. If a word does not exist as a key of the hash, then it should remain unchanged.

Examples

p sentence\_swap('anything you can do I can do',

'anything'=>'nothing', 'do'=>'drink', 'can'=>'shall'

) # 'nothing you shall drink I shall drink'

p sentence\_swap('what a sad ad',

'cat'=>'dog', 'sad'=>'happy', 'what'=>'make'

) # 'make a happy ad'

p sentence\_swap('keep coding okay',

'coding'=>'running', 'kay'=>'pen'

) # 'keep running okay'

**hash\_mapped**

Write a method hash\_mapped that accepts a hash, a proc, and a block. The method should return a new hash where each key is the result of the original key when given to the **block**. Each value of the new hash should be the result of the original values when passed into the **proc**.

Examples

double = Proc.new { |n| n \* 2 }

p hash\_mapped({'a'=>4, 'x'=>7, 'c'=>-3}, double) { |k| k.upcase + '!!' }

# {"A!!"=>8, "X!!"=>14, "C!!"=>-6}

first = Proc.new { |a| a[0] }

p hash\_mapped({-5=>['q', 'r', 's'], 6=>['w', 'x']}, first) { |n| n \* n }

# {25=>"q", 36=>"w"}

**counted\_characters**

Write a method counted\_characters that accepts a string as an argument. The method should return an array containing the characters of the string that appeared **more than twice**. The characters in the output array should appear in the same order they occur in the input string.

Examples

p counted\_characters("that's alright folks") # ["t"]

p counted\_characters("mississippi") # ["i", "s"]

p counted\_characters("hot potato soup please") # ["o", "t", " ", "p"]

p counted\_characters("runtime") # []

**triplet\_true?**

Write a method triplet\_true? that accepts a string as an argument and returns a boolean indicating whether or not the string contains three of the same character consecutively.

Examples

p triplet\_true('caaabb') # true

p triplet\_true('terrrrrible') # true

p triplet\_true('runninggg') # true

p triplet\_true('bootcamp') # false

p triplet\_true('e') # false

**energetic\_encoding**

Write a method energetic\_encoding that accepts a string and a hash as arguments. The method should return a new string where characters of the original string are replaced with the corresponding values in the hash. If a character is not a key of the hash, then it should be replaced with a question mark ('?'). Space characters (' ') should remain unchanged.

Examples

p energetic\_encoding('sent sea',

'e'=>'i', 's'=>'z', 'n'=>'m', 't'=>'p', 'a'=>'u'

) # 'zimp ziu'

p energetic\_encoding('cat',

'a'=>'o', 'c'=>'k'

) # 'ko?'

p energetic\_encoding('hello world',

'o'=>'i', 'l'=>'r', 'e'=>'a'

) # '?arri ?i?r?'

p energetic\_encoding('bike', {}) # '????'

**uncompress**

Write a method uncompress that accepts a string as an argument. The string will be formatted so every letter is followed by a number. The method should return an "uncompressed" version of the string where every letter is repeated multiple times given based on the number that appears directly after the letter.

Examples

uncompress('a2b4c1') # 'aabbbbc'

uncompress('b1o2t1') # 'boot'

uncompress('x3y1x2z4') # 'xxxyxxzzzz'

**Phase 2: More difficult, maybe?**

**conjunct\_select**

Write a method conjunct\_select that accepts an array and one or more procs as arguments. The method should return a new array containing the elements that return true when passed into all of the given procs.

Examples

is\_positive = Proc.new { |n| n > 0 }

is\_odd = Proc.new { |n| n.odd? }

less\_than\_ten = Proc.new { |n| n < 10 }

p conjunct\_select([4, 8, -2, 11, 7, -3, 13], is\_positive) # [4, 8, 11, 7, 13]

p conjunct\_select([4, 8, -2, 11, 7, -3, 13], is\_positive, is\_odd) # [11, 7, 13]

p conjunct\_select([4, 8, -2, 11, 7, -3, 13], is\_positive, is\_odd, less\_than\_ten) # [7]

**convert\_pig\_latin**

Write a method convert\_pig\_latin that accepts a sentence as an argument. The method should translate the sentence according to the following rules:

* words that are shorter than 3 characters are unchanged
* words that are 3 characters or longer are translated according to the following rules:
  + if the word begins with a vowel, simply add 'yay' to the end of the word (example: 'eat'->'eatyay')
  + if the word begins with a non-vowel, move all letters that come before the word's first vowel to the end of the word and add 'ay' (example: 'trash'->'ashtray')

Note that if words are capitalized in the original sentence, they should remain capitalized in the translated sentence. Vowels are the letters a, e, i, o, u.

Examples

p convert\_pig\_latin('We like to eat bananas') # "We ikelay to eatyay ananasbay"

p convert\_pig\_latin('I cannot find the trash') # "I annotcay indfay ethay ashtray"

p convert\_pig\_latin('What an interesting problem') # "Atwhay an interestingyay oblempray"

p convert\_pig\_latin('Her family flew to France') # "Erhay amilyfay ewflay to Ancefray"

p convert\_pig\_latin('Our family flew to France') # "Ouryay amilyfay ewflay to Ancefray"

**reverberate**

Write a method reverberate that accepts a sentence as an argument. The method should translate the sentence according to the following rules:

* words that are shorter than 3 characters are unchanged
* words that are 3 characters or longer are translated according to the following rules:
  + if the word ends with a vowel, simply repeat the word twice (example: 'like'->'likelike')
  + if the word ends with a non-vowel, repeat all letters that come after the word's last vowel, including the last vowel itself (example: 'trash'->'trashash')

Note that if words are capitalized in the original sentence, they should remain capitalized in the translated sentence. Vowels are the letters a, e, i, o, u.

Examples

p reverberate('We like to go running fast') # "We likelike to go runninging fastast"

p reverberate('He cannot find the trash') # "He cannotot findind thethe trashash"

p reverberate('Pasta is my favorite dish') # "Pastapasta is my favoritefavorite dishish"

p reverberate('Her family flew to France') # "Herer familyily flewew to Francefrance"

**disjunct\_select**

Write a method disjunct\_select that accepts an array and one or more procs as arguments. The method should return a new array containing the elements that return true when passed into at least one of the given procs.

Examples

longer\_four = Proc.new { |s| s.length > 4 }

contains\_o = Proc.new { |s| s.include?('o') }

starts\_a = Proc.new { |s| s[0] == 'a' }

p disjunct\_select(['ace', 'dog', 'apple', 'teeming', 'boot', 'zip'],

longer\_four,

) # ["apple", "teeming"]

p disjunct\_select(['ace', 'dog', 'apple', 'teeming', 'boot', 'zip'],

longer\_four,

contains\_o

) # ["dog", "apple", "teeming", "boot"]

p disjunct\_select(['ace', 'dog', 'apple', 'teeming', 'boot', 'zip'],

longer\_four,

contains\_o,

starts\_a

) # ["ace", "dog", "apple", "teeming", "boot"]

**alternating\_vowel**

Write a method alternating\_vowel that accepts a sentence as an argument. The method should return a new sentence where the words alternate between having their first or last vowel removed. For example:

* the 1st word should be missing its first vowel
* the 2nd word should be missing its last vowel
* the 3rd word should be missing its first vowel
* the 4th word should be missing its last vowel
* ... and so on

Note that words that contain no vowels should remain unchanged. Vowels are the letters a, e, i, o, u.

Examples

p alternating\_vowel('panthers are great animals') # "pnthers ar grat animls"

p alternating\_vowel('running panthers are epic') # "rnning panthrs re epc"

p alternating\_vowel('code properly please') # "cde proprly plase"

p alternating\_vowel('my forecast predicts rain today') # "my forecst prdicts ran tday"

**silly\_talk**

Write a method silly\_talk that accepts a sentence as an argument. The method should translate each word of the sentence according to the following rules:

* if the word ends with a vowel, simply repeat that vowel at the end of the word (example: 'code'->'codee')
* if the word ends with a non-vowel, every vowel of the word should be followed by 'b' and that same vowel (example: 'siren'->'sibireben')

Note that if words are capitalized in the original sentence, they should remain capitalized in the translated sentence. Vowels are the letters a, e, i, o, u.

Examples

p silly\_talk('Kids like cats and dogs') # "Kibids likee cabats aband dobogs"

p silly\_talk('Stop that scooter') # "Stobop thabat scobooboteber"

p silly\_talk('They can code') # "Thebey caban codee"

p silly\_talk('He flew to Italy') # "Hee flebew too Ibitabaly"

**compress**

Write a method compress that accepts a string as an argument. The method should return a "compressed" version of the string where streaks of consecutive letters are translated to a single appearance of the letter followed by the number of times it appears in the streak. If a letter does not form a streak (meaning that it appears alone), then do not add a number after it.

Examples

p compress('aabbbbc') # "a2b4c"

p compress('boot') # "bo2t"

p compress('xxxyxxzzzz')# "x3yx2z4"