HW 1-1: FUN WITH STRINGS

In this problem, you are asked to implement two functions that perform basic string processing. You can get the template file for this problem here (/static/content-berkeley-cs169x~2013_Spring/handouts/hw1.fa0dd758299a.zip) (the file for this problem is "part1.rb"). Please read the instructions carefully and submit your code via the "Browse" button at the bottom of the page.

Part A — Palindromes: Write a method `palindrome?(string)` that determines whether a given `string` (word or phrase) is a palindrome, that is, it reads the same backwards as forwards, ignoring case, punctuation, and nonword characters. (A "nonword character" is defined for our purposes as "a character that Ruby regexps would treat as a nonword character".)

The structure of your code should look as follows:

```
def palindrome?(string)
    # your code here
end
```

Your solution shouldn't use loops or iteration of any kind. Instead, you will find regular-expression syntax very useful; it's reviewed briefly in the book, and the website rubular.com (http://www.rubular.com) lets you try out Ruby regular expressions "live". Some methods that you might find useful (which you'll have to look up in Ruby documentation, ruby-doc.org (http://ruby-doc.org)) include: `String#downcase`, `String#gsub`, `String#reverse`.

Example test cases:

```
palindrome?("A man, a plan, a canal -- Panama")  # => true
palindrome?("Madam, I'm Adam!")                  # => true
palindrome?("Abracadabra")                       # => false (nil is also ok)
```

Part B — Word Count: Define a function `count_words(string)` that, given an input string, return a hash whose keys are words in the string and whose values are the number of times each word appears. Your code should look like:

```
def count_words(string)
    # your code here
end
```

Your solution shouldn't use for-loops, but iterators like `each` are permitted. As before, nonwords and case should be ignored. A word is defined as a string of characters between word boundaries. (Hint: the sequence "\b" in a Ruby

regexp means "word boundary".)

Example test cases:

```
count_words("A man, a plan, a canal -- Panama")
    # => {'a' => 3, 'man' => 1, 'canal' => 1, 'panama' => 1, 'plan' => 1}
count_words "Doo bee doo bee doo"
    # => {'doo' => 3, 'bee' => 2}
```

Choose Files No file chosen

```
On Time
#palindrome?
  should be defined
  recognizes palindromes correctly [25 points]
  recognizes non palindromes correctly [25 points]

#count_words
  should be defined
  should return a Hash
  counts the words properly [40 points]
  is not sensitive to case [10 points]

Finished in 0.00459 seconds
7 examples, 0 failures
```

Check

Show Discussion

New Post