CS142 Project #2: Ruby Calisthenics

Problem 1: everything is an object, including numbers (5 points)

Cut and paste the following code into a file named `squared.rb`:

```
... your code goes here ...

puts 9.squared
puts 16.squared
```

Then replace the line at the top with Ruby code that adds a new method `squared` to the `Numeric` class. The method should return the square of the object on which it is invoked. Once this method is defined you should be able to use expressions such as `9.squared`. Test the code by typing `ruby squared.rb` to your shell.

Problem 2: using blocks (5 points)

Create a file named `sort.rb` that defines a method `funny_sort`, which takes a single argument containing an array of strings and returns an array containing the same strings in sorted order. The sorting of the strings is determined by the first sequence of digits within each string. For example, the string "abc99.6" is sorted according to "99", and "-100x500" is sorted according to "100". If a string contains no decimal digits then it must be sorted according to "-1" (all such strings will appear first in the output, in any order). In your implementation you must use the `sort` function implemented by the `Enumerable` module, and you must provide a block that ensures the correct sorting order. In addition to defining `funny_sort` your file `sort.rb` must include code that invokes `funny_sort` a few times and prints the results to demonstrate that your code works.

Problem 3: defining an iterator (5 points)

Create a file named `starts_with.rb` that adds a method `each_starts_with`, which takes two arguments plus a block. The first argument is an array of strings and the second argument is a string. The method iterates over each of the strings in the array, invoking the block for each element whose first characters match the second argument exactly. The matching string is passed to the block as an argument. Here is an example of an `irb` session illustrating how the method should work:

```
$ irb --simple-prompt
>> load "starts_with.rb"
=> true
>> each_starts_with(["abcd", "axyz", "able", "xyzab", "qrst"], "ab") {|s| puts s}
abcd
able
=> ["abcd", "axyz", "able", "xyzab", "qrst"]
>>
```

Include code in your `starts_with.rb` file that invokes `starts_with` to demonstrate that it works correctly.

Problem 4: hashes and symbols (5 points)

Create a file named `filter.rb` that defines a method `filter`, which takes one or two arguments plus a block. The first argument is an array of numbers and the second argument, if specified, is a hash containing zero or more options. The method iterates over each of the numbers in the array, invoking the block with each number that matches the options. The `filter` method must support the following options:

`min`	Skip any numbers less than the value of this option.
`max`	Skip any numbers greater than the value of this option.
`odd`	If this option is defined, skip any even numbers.
`even`	If this option is defined, skip any odd numbers.
`scale`	If this option is defined, multiply each number by the value of the option before passing to the block.

Here is an example of an `irb` session illustrating how the method should work:

```
$ irb --simple-prompt
>> load "filter.rb"
=> true
>> nums = [6, -5, 319, 400, 18, 94, 11]
=> [6, -5, 319, 400, 18, 94, 11]
>> filter(nums, :min => 10, :max => 350, :odd => 1, :scale => 2) {|n| puts n}
   638
   22
   => nil
>> filter(nums, :max => 0) {|n| puts n}
-5
=> nil
>> filter(nums) {|n| puts n}
6
-5
319
400
18
94
11
=> nil
>>
```

Include code in your `filter.rb` file that invokes `filter` to demonstrate that it works correctly. Your solution must use symbols to index into hashes whereever that is practical.

Problem 5: extending a module (5 points)

Create a file named `group.rb` that adds a new method `each_group_by_first_letter` to the `Enumerable` module. This method is an iterator that yields two items at a time: a one-letter string and an array of all the input values starting with that letter. The order in which letters are yielded, and the order of values in the array, are not important. Once you have defined this iterator in the `Enumerable` module, you can use it for any enumerable collection. For example:

```
>> x = ["abcd", "axyz", "able", "xyzab", "qrst"]
=> ["abcd", "axyz", "able", "xyzab", "qrst"]
>> x.each_group_by_first_letter do |letter, words|
?>    printf("%s: %s\n", letter, words.join(" "));
?> end
a: abcd axyz able
x: xyzab
q: qrst
```

Include code in your `group.rb` file that invokes `each_group_by_first_letter` to demonstrate that it works correctly.

Problem 6: metaprogramming (5 points)

Create a file `adder.rb` that defines a class `Adder`. This class should provide a constructor that takes a single integer argument; other than this, the class should contain only a single method named `method_missing`. As discussed in class, this method will get invoked whenever a nonexistent method is invoked on objects of this class. If a method of the form `plus`*num* is invoked, where *num* is a positive integer, your code should add a new method by that name to the Adder class and invoke it; in the future, the method will already exist so `method_missing` will not be invoked. The method `plus`*num* should return *i*+*num*, where *i* was the argument passed to the object's constructor. If the method name doesn't have the form `plus`*num* then your `method_missing` method should invoke `method_missing` on the superclass so that a proper error gets generated. Note: you may find the methods `class_eval`, `eval`, and `super` useful in your solution. Here is an example `irb` session illustrating how the class should work:

```
$ irb --simple-prompt
>> load "adder.rb"
=> true
>> x = Adder.new(10)
>> x.plus20
=> 30
>> x.plus25
=> 35
>> x.minus2
NoMethodError: undefined method `minus2' for #<Adder:0x2cecca8 @i=10>
        from ./adder.rb:15:in `method_missing'
        from (irb):6
```

Style Points (5 points)

These points will be awarded if your Ruby code for the problems above is clean and readable.

Useful Hints

- There is reference documentation available for all the core Ruby classes such as `Array`, `Hash`, and `String` at http://ruby-doc.org/core-1.9.2/.
- You can use the Ruby range mechanism to extract substrings. For example, if `x` is the string "abcdef" then `x[1..4]` is "bcde".

Deliverables

Use the standard class submission mechanism to submit the code files you created for the problems, including `squared.rb`, `sort.rb`, `starts_with.rb`, `filter.rb`, `group.rb`, and `adder.rb`.