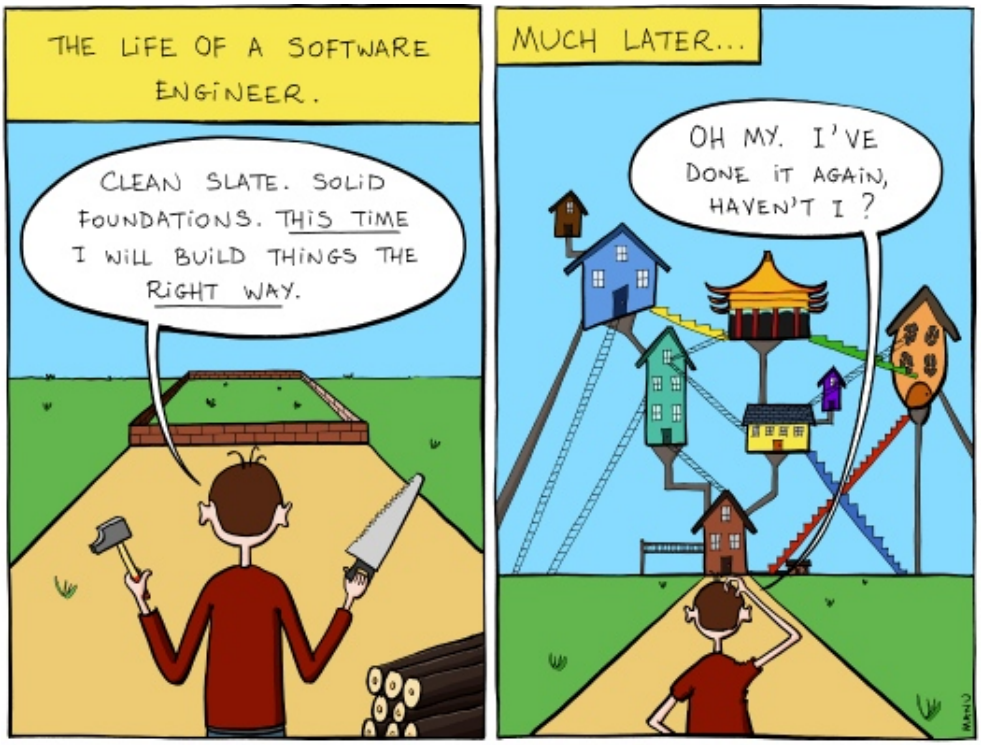
CS 312 - Assignment 2
Around the block with Ruby blocks

Due 11:59pm on Monday, February 27



http://www.bonkersworld.net/images/2011.11.15_life_of_a_swe.png

1 Blocks 101

Write the functions below in a file called `assign2_1.rb`. Make sure to follow the instructions exactly and use the specified names. You may **NOT** use a `for` loop for any of the functions (but consider using the methods we talked about on Tuesday including `each`, `each_with_index`, `each_char` and `sort_by`). You will be graded on both correctness as well as style and brevity.

- Write a function `hash_value_sum` that takes a hash whose values are numbers as a parameter and returns the sum of the values in the hash.
- Write a function `square_hash_values` that takes a hash whose values are numbers as a parameter and returns a *new* hash that has the same keys as the parameter but the values have been squared. You should use the `each` method.
- Write a function `every_other_sum` that takes an array of numbers as a parameter and returns the sum of every other number in the array starting with the first entry.

```
>> every_other_sum([1, 2, 3, 4, 5, 6])
9
```

- Write a function called `get_character_counts` that takes a string as a parameter and returns a hash with the keys as characters and the value being the number of times that character occurred in the string. Upper- and lower-case characters should be treated the same. (Hint: consider using `Hash.new` to create the hash to make your life easier. See the documentation and the examples in the book for more on this.)

```
>> character_counts("This is a string with some characters in it.")
{"t"=>5, "h"=>3, "i"=>6, "s"=>5, " "=>8, "a"=>3, "r"=>3, "n"=>2,
 "g"=>1, "w"=>1, "o"=>1, "m"=>1, "e"=>2, "c"=>2, "."=>1}
```

- Write a function called `longest_words` that takes a string as a parameter and returns a list of the words in the string in sorted order by the length of the word. A word is just anything separated by a space. The `split` method in the `String` class may be useful.

```
>> longest_words("this is a sentence with some short and some long words")
["sentence", "words", "short", "some", "some", "this", "with", "long",
 "and", "is", "a"]
```

2 Blocks 201

In a file called `assign2_2.rb` write functions where ***the body is a single statement***. Put a comment above each function with the corresponding problem number (along with your normal comments). In many situations, you wouldn't actually right these as separate functions, but to make grading easier, we will. (Hint: `select`, `select!`, `map` and `map!` may be useful for some of these.)

Some advice: Just get the functions working first, then work to try and compress them down to a single statement. If you find yourself a lot of complicated code, there is probably an easier way to do it using blocks.

1. Write a function called `invert_sign` that takes an array of numbers as a parameters and returns a new array with the sign of each number inverted.

2. Write a function called `invert_sign!` that does the same as above except modifies the input array rather than creating a new one.

3. Write a function called `filter_by_count` that takes a hash and a count threshold as parameters and returns a new hash where only entries with values greater than or equal to the count threshold exist.

4. Write a function called `filter_by_count!` that does the same as above, but changes the hash passed in (rather than constructing a new one).

5. Write a function called `filter_consonants!` that takes a hash as a parameter and removes any entries from the hash where the key does not start with a vowel.

6. Write a function called `print_sorted` that takes a hash as a parameter and prints the key/value pairs one per line formatted as shown below sorted by values from highest to lowest with the key.

```
>> print_sorted({"t"=>3, "h"=>1, "i"=>2, "s"=>3, " "=>3, "a"=>1, "e"=>1})
t:   3
 :   3
s:   3
i:   2
a:   1
h:   1
e:   1
```

7. Write a function called `array_transform!` that takes an array of numbers as a parameter and changes any entry in the array to 0 if the original value was less than the index where that value occurred. Hint: the ternary operator (i.e. the question mark: `?`) may be useful.

```
>> a = [1, 1, 5, 2, 1, 7]
>> array_transform!(a)
[1, 1, 5, 0, 0, 7]
>> a
[1, 1, 5, 0, 0, 7]
```

Blocks 301

In class we added `each`, `each_with_index` and `find` methods to our `LinkedList` class to support more Ruby-like traversal. Download this version of the `LinkedList` class from the course web pages and add the methods below. Where possible, avoid traversing the linked list yourself and instead utilize the `each` or `each_with_index` methods.

- Add a `select` method to the linked list class that is called with a block and returns an *array* with all of the values in the linked list where the input block returns true.[1]

```
l = LinkedList.new
```

[1] The better way to do this would probably be to have the function return a new linked list with the selected values, however, this is a bit more complicated than I'd expect you to handle. Feel free to try it out (name the function something else) if you're up for a challenge.

```
l.add(1)
l.add(2)
l.add(3)
l.add(4)

p l.select{|v| v%2 == 0}
# displays: [4, 2]
```

- Add a `select!` method that actually modifies the linked list and only keeps those items in the list where the block returns true. This can be a bit tricky since it's a singly linked list. Some pieces of advice:

  1. Make sure you think about when you need to update `@head`.
  2. As you work your way through the linked list if you keep a reference to the current node *and* a reference to the previous node you can splice out the current node using the previous reference.
  3. This isn't a lot of code (mine is <20 lines including `end`s), but make sure you think through the logic before you start implementing.

```
l = LinkedList.new
l.add(1)
l.add(2)
l.add(3)
l.add(4)

l.select!{|v| v\%2 == 0}
puts l
# displays:  4 2
```

Style/Comments

Make sure that your program is properly commented and that you've used good style.

Comments

- All classes and methods should have a block of comments above them.
- Each file should have a block of code at the top with a brief description, your name and the date.
- You should comment any tricky or confusing things in the code.

Style

- You should use appropriate `attr_` tags when defining class instance variables.
- You should appropriately use ? and ! in naming functions.
- You code should be succinct and follow the general ruby conventions, including variable, method and class naming schemes.
- Your code should be broken down into appropriate functions and blocks of code.

When you're done...

Make sure you've followed the specifications above.

To submit, follow the instructions on the course web page. You should have three files: `assign2_1.rb`, `assign2_2.rb` and `linked_list.rb`.

Grading

You will be graded based on the following criteria:

criterion		points
`assign2_1.rb`	3 per function	15
`assign2_2.rb`	2 per fuction	14
`linked_list.rb`		
	`select`	4
	`select!`	5
Commenting		2
Total		40