Section 9 Worksheet

Ruby Practice

Practice with Arrays and blocks

1) Write a method `lengths` that takes an array and returns a new array that is the result of calling `length` on all of the elements in the given array.

2) Write a method `rev` that takes an array and returns a new array with all of the elements from the original array in the reverse order. You aren't allowed to use the Array class' `reverse` method (though in real life you probably should).

3) Write a method `num_even` that takes an array and returns the number of even elements in that array. There's a handy method `even?` that you may find useful.

4) Write a method `all_equal?` that returns `true` if all the elements in a given array are equal or if the given array is empty and `false` otherwise.

5) Write a method `prime` that takes in a positive integer greater than `1` and returns an array of all the prime numbers from `1` to the given number.

6) Write a method `trigger_sum` that takes in an array and a “trigger” number. `trigger_sum` creates a cumulative sum of all the values in the array until it finds the trigger number, at which point it begins subtracting future numbers (including the trigger) from the cumulative sum. The trigger only occurs once. If the trigger is not in the given array, then it will just end up returning the cumulative sum.

Practice with Hashes

1) Write a method `keys_and_values` that takes a hash and returns an array of elements that are both keys and values in the given hash. You might find the arrays set intersection operator `&` to be useful.

2) Write a method `flip_hash` that takes a hash and returns a new hash where every key-value pair in the original hash is flipped.

3) Write a method `intersect` that takes two hashes and returns a new hash that contains all of the key-value pairs that appear in both of the given hashes.

Practice using blocks

Implement the following functions using the arrays `each` method. Remember you can call a block given to a method using yield, and you can pass yield any necessary arguments.

1) Implement `our_map` which takes an array and expects a block and behaves as the built-in arrays `map` function would with the given block.

2) Implement `our_select` which takes an array and expects a block and behaves as the built-in arrays `select` function would with the given block.

3) Implement `our_inject` which takes an array and an initial value and expects a block and behaves as the built-in arrays `inject` function would with the given initial value and block.

Super brief Ruby cheat sheet *(really just for this handout)*

```
arr = [1, 2, 3, 4, 5]

# calls block on each value in arr, returns arr
arr.each { |x| block }

# returns a new array of block mapped across all elements in arr
arr.map { |x| block }

# returns a new array of elements in arr for which the block returns true
arr.select { |x| block }

# returns the number of elements for which the given block is true
arr.count { |x| block }

# behaves like fold over arr starting at init and accumulating the block
arr.inject(init) { |acc, x| block }

# true only if all the elements in arr return true for the given block
arr.all? { |x| block }

# true if any of the elements in arr return true for the given block
arr.any? { |x| block }

# hashes also have a version of many iterative functions the blocks
# typically expect two arguments instead of one (for key and value)
h = {1=>2, 3=>4, 5=>6}
# For example, each
h.each {|k, v| puts k + v }

# hashes also have a way to access the array of keys and the array of
values
h.keys
h.values
```