W1 – Methods and Other Tools
Mike Le

Strings
- str#length
- str#upcase
- str#downcase
- str#capitalize
- str[start_idx..ending_idx] - inclusive of last index
- str[start_idx...ending_idx] - exclusive of last index
- str#index(char) / str#index(?char) / below
- str#rindex(char) / str#rindex(/[substring, -num)
- str#include?(char)
- str#reverse / str#reverse**!**
- str#split("delimiter")
- str#chars
- str#count(char) ← number of instances
- "hello".gsub(/[aeiou]/, '*')   #=> "h*ll*"
- str#prepend(at least 1 str) ← puts args before str
- str#starts_with? / str#ends_with?

- Can't use any? / all? etc on str. Use #chars first!
- For numbers use → .to_s
- Num * str = strstrstr

Arrays
- Array#length
- Array#first
- Array#last
- Array#shift ← remove 1st ele
- Array#unshift
- Array#pop ← remove last ele
- Array#push
- Array << ele
- Array#include?(element)
- Array#index(element) ← nil if invalid index
- Array[start**...**end] / Array[start**..**end]
- array1.concat(array2)
- Array[index] = new value
- Array#delete(ele) / Array#delete_at(index)
- .to_a ← can use to turn range to arr!

- Array#reverse / Array#reverse!
- Array#flatten ← multi dim arr **to** one dim
- Array#sort / Array#sort! (small to largest)
- Array#uniq ← use for iterating + delt thr arr

- Array#map ←new arr, does something each ele
  - **New element is the result of the block!**
- Array#select ← creates new array of true_eles
- Array#reject ← creates new array of false_eles
- Array#filter ← (alias for select, same as)
- Array#partition ← new 2D array of [[true], [false]]

- Array#partition ← new 2D array of [[true], [false]]

Hash
- Hash#length ←number of key-value pairs
- Hash[key] = _____ ←create key-value pair
- Hash[key_n] ← access value, in key_n-/val pair
- h1.merge(h2) ← creates new combined hash

- Hash#keys / Hash#values ← gives arrays of keys/values
- Hash#hash_key? / Hash#has_value? ← Boolean

- **CAN USE →** .all? / .any? /.none? / .one?
- Hash#select / Hash#select! / Hash#reject / Hash#reject!

- **Hash w/ new default value, ie 0**
  - Counter = Hash.new(new_default_value)
- **Complex Default Value**
  - new_hash = Hash.new { |hash, key| hash[key] = new_default }
- 2D_arr = hash.sort_by{ |k, v| v } /
  = hash.sort_by{ |k, v| k }

- Hash#each { |k, v| <some_code> } / Hash#each { |k| <some_code> }

Math
- num#even?
- num#odd?
- num#abs
- num#ceil
- num#floor
- num#round
- Math#sqrt(num)
- Integer#sqrt(num)
- a ** b == a^b

Other Tools
- ("a".."z").to_a == array of alphabet
- *arr == equivalent to taking brackets [ ] off of arr. Allows methods to take in multiple args!
- **hash → similary takes off { }. {**hash_1, **hash_2, 10=>"a"}
- IF keys in hash are symbols, need :keys, to access them!
- Str[-1] == last char of str. To replicate slice in JS do → str[index, -1]
- Consider array of procs vs. method taking in multiple procs! Diff is user passes in array as arg vs. just listing multiple procs as diff args.
- How to swap elements in array or values of variables → arr[0], arr[1] = arr[1]. Arr[0]   var1, var2 = var2, val1
- Use HASH if need to keep track of multiple things or replace things!
- Use arr.uniq if iterating through an array and deleting stuff!
- If iterating through specifc values just do range! → (0...str.length). each { <some code> } (no parameters)
- ele = prc.call(ele) KEY is that RHS is evaluated first, then assigned to ele. So new ele each time.
- OppIndex = [-**i** – 1]
- For vowels always do "aeiou**AEIOU**"
- Result for block in arr.map is what ele becomes!