RUBY CHEAT SHEET

SESSION 1 (ARRAYS)

ARRAY PRIMER

Create Array

- x = [ ]
- x = [1,2,3,4] # [1, 2, 3, 4]
- x = Array.new
- x = Array.new(3) # [nil, nil, nil]
- x = Array.new(3) { “hi” } # [“hi”, “hi”, “hi”]
- x = Array.new(3, “hi”) # [“hi”, “hi”, “hi”]

Accessing and Assigning elements (y = [1, 2, 3, 4, 5, 6])

- x = y[0] # 1
- x = y[-2] # 5
- x = y.values_at(0, 1, 4) # [1 , 2 , 5]
- y [1] = "hi" # [1, "hi", 3, 4, 5, 6]
- y [-1] = "hey" # [1, "hi" , 3, 4, 5, "hey"]

- x = y.slice(2) # 3
- x = y.slice!(2) # [1, "hi" , 4, 5, "hey"]

- "a b c".split # [“a”, “b”, “c”]
- y = %w/red blue pink yellow green/
  # ["red" , "blue" , "pink" , "yellow" , "green"]

Adding and Removing elements (y = [1, 2, 3])

- y.push "hi" # [1 , 2 , 3 , " hi "]
- y << "hi" # [1 , 2 , 3 , " hi " , "hi"]
- y.unshift "hey" # Adds an element in front of the array
- y.pop # Removes the last element from the array and returns it
- y.delete_at(0) # Removes the first element of the array
- users.shift # Removes the first element of the array and returns it

COMMON METHODS (Y = [1, 2, 3, 4, 5, 6])

- y.empty? # false
- y.include?(2) # true
- y.size # 6
- y.join # "123456"
- y.join(" ") # " 1 2 3 4 5 6 "
- y.reverse # [“6”, “5”, “4”,"3","2","1"]

Iterating over an array

- y.each { |item| puts item }

- y.each_with_index { |item, idx| puts "#{item} with index #{idx}" }

Sorting array

sort and sort_by

- array.sort
- array.sort_by { |object| object.attribute }
- arr.sort_by(&:attribute)

the above returns a sorted array . If an exclamation mark is added such as - sort! or sort_by! - modifies the original array.

Complex sorting can be done with the use of regex .

Selecting from an array by criteria (filter)

- array.select {|e| condition } # e is for element
- array.grep(/pattern/)
- array.find {|e| condition } # e is for element

The select method returns multiple elements

- x = [5, 8, 12, 9, 4, 30]
  x.select {|e| e % 2 == 0} # [8, 12, 4, 30]

The grep method In its simplest form, returns an array containing the matched elements.

- a = %w[January February March April May]
  a.grep(/ary/)
  # [“January, “February”]

- b = [1, 20, 5, 7, 13, 33, 15, 28]
  b.grep(12..24)
  # [20, 13, 15]

The find method iterates through an array and returns the first match for which the expression / condition is true.

- [1,2,3,4,5,6,7].detect { |x| x.between?(3,7) }
  # 3

ARRAY DEEP DIVE

SESSION 1 (ARRAYS)

MAP METHOD

Array.map { |x| x.function }
Array.map(&:function)   #shorthand

Applying map on an array returns a new array where each element is the result of the x.function (which can also be a block with x as the arguement)

- [1, 2, 3].map { |x| x * 2 }   # [2, 4, 6]

- [1, 2, 3].map { |n| n.even? }   # [false, true, false]

similarly ,

- [1, 2, 3].map { &: even?}   # [false, true, false]

maps can also be executed using the following syntax,

- ["Alex", "bianca", "TIM"].map do |name| name.upcase
  end
  # ["ALEX", "BIANCA", "TIM"]

using map with index ,

- ["a", "b", "c"].map.with_index { |x, i| x+ i.to_s }
  # ["a0", "b1", "c2"]

the ' map! ' syntax will transform the original data.

- [1, 2, 3].map! { |x| x * 2 }

- ["a", "b", "c"].map!.with_index { |x, i| x+ i.to_s }

Using ' map ' to iterate through an array while getting user input and assigning input_value to element.

- candidates = candidates.map.with_index
{|val, index| puts " Enter Candidate #{index}: " val =gets.chomp
}

or

- candidates.map!.with_index
{|val, index| puts " Enter Candidate #{index}: " val =gets.chomp
}

REDUCE METHOD

Array.reduce {|result, current| result.func(current) }
or
Array.reduce(value) {|result, current| result.func(current)
}
or
Array.reduce(:function)

when you want your array to be reduced to only 1 value , you use the reduce method. (reduce( ) returns 1 value).

- [5,2,3].reduce {|result, current| result += current }
  # 10
- [5,2,3].reduce(5) {|result, current| result += current }
  # 15
- [1, 2, 3].reduce(:+)
  # 6
- [1,2,3].reduce(100, :+)
  # 106

In an earlier statement , it is mentioned that ' reduce ( ) returns 1 value ' - only 1 value
This ' 1 value ' can also be an array.

- values = ["1", "2", "3"]
  integers = values.reduce([]) do |array, current|
  val = current
  array.push(val)
  end   # ["1" , "2" , "3"]

Using ' map ' and ' reduce ' in tandem

- x = [4, 2, 3]
  y = [4, 1, 6]
  z = [2, 8, 9]

  [x, y, z].map { |x| x.reduce(:+) }

  # [9, 11, 19]