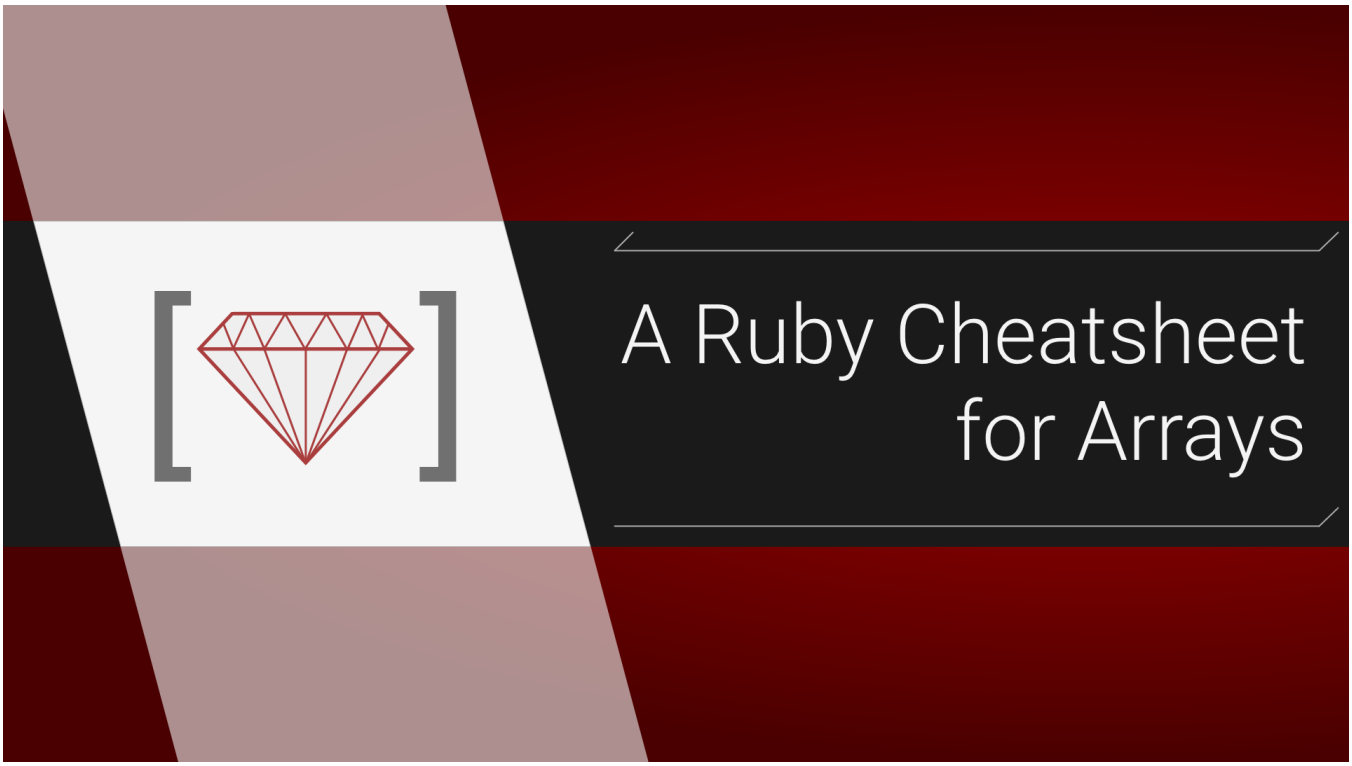
A Ruby Cheatsheet For Arrays

A reference for beginners and forgetful professionals

Mike Cronin
Aug 16 · 7 min read ★



Simply put, before you lies a metric *ton* of handy Ruby Array methods. It's long, but I tried to include all the really useful stuff. When a method is used, be sure to check the docs for more info. And to keep things shorter, I'll write return values in comments, so

Create a new array with values:

```
arr_with_stuff = ["value", "separated by comma"]
arr_with_stuff2 = Array.new(["a", "b", "c"])
range_to_arr = (1..9).to_a
```

create an array of duplicate values: Sometimes it's helpful to have an array filled with multiple, duplicates:

```
arr = Array.new(5, " ")
# -> [" ", " ", " ", " ", " "]
```

create an array of strings with a shortcut: use the `%w` shortcut to create an array of strings without quotes or commas:

```
arr = %w(cat dog mouse 1 2)
# -> ["cat", "dog", "mouse", "1", "2"]
```

convert a string into an array: #split will split a string into an array. It takes an argument of a string to decide where to split the array items, otherwise it splits by space. To get each character alone, specify "" as the argument:

```
arr = "this is a string".split
arr # -> ["this", "is", "a", "string"]

"hey".split("") # -> ["h","e","y"]
```

```
str2 = arr.join
str2 #-> "catdogmouse"
```

Reading/Getting Values

Get the length of the array: #length and #size are aliases

```
arr = %w(cat dog mouse cow pig)
arr.length # -> 5
arr.size # -> 5
```

Get the value at an index

```
arr[0] # -> "cat"
arr[1] # -> "dog"
arr[100] # -> nil
```

Get the index of a value: #index

```
arr.index("dog") # -> 1
arr.index("asdaf") # -> nil
```

Get the value at an index from the back: You can go from the back of the array with negative numbers:

```
arr[-1] # -> "pig"
arr[-2] # -> "cow"
arr[-100] # -> nil
```

```
arr.fetch(100) { |key| "nothing at index #{key}" }
# -> "nothing at index 100"
```

Get a slice of the array: you can use commas or ranges:

```
arr = ["cat", "dog", "mouse", "cow", "pig"]
arr[1,3] # -> ["dog", "mouse", "cow"]
arr[1..3] # -> ["dog", "mouse", "cow"]
arr[1...3] # -> ["dog", "mouse"]
arr[1..-1] # -> ["dog", "mouse", "cow", "pig"]
```

Check if a value is in the array: #include?

```
arr = %w(cat dog mouse)
arr.include?("cat") # -> true
arr.include?("zzz") #-> false
```

Add and update values

Change a value at an index:

```
arr = %w(cat dog mouse)
arr[0] = "lynx"
arr # -> ["lynx", "dog", "mouse"]
```

Add a value to the end of an array: The << is called a “shovel operator”, but you can also use #push. These methods modify the original array:

Add a value to the start of the array: #unshift

```
arr = %w(cat dog mouse)
arr.unshift("clam")
# returns changed array
# -> ["clam", "cat", "dog", "mouse"]

arr.unshift("cow", "bee")
# returns changed array
# -> ["cow", "bee", "clam", "cat", "dog", "mouse"]
```

add a value to the middle of the array: #insert

```
arr = %w(cat dog mouse)
arr.insert(1, "cow")
# returns changed array
# -> ["cat", "cow", "dog", "mouse"]

arr.insert(1, "bee", "pig")
# returns changed array
# -> ["cat", "bee", "pig", "cow", "dog", "mouse"]
```

Remove and delete values

delete by value: #delete

```
arr = %w(cat dog mouse)
arr.delete("cat")
# -> returns "cat"
arr # -> ["dog", "mouse"]
```

delete by any index: #delete_at

```
arr = %w(cat dog mouse)
arr.delete_at(0) # -> "cat"
arr # -> ["dog", "mouse"]
arr.delete_at(100) # -> nil
```

remove last value: #pop

```
arr = %w(cat dog mouse)
arr.pop # -> "mouse"
arr # -> ["cat", "dog"]
arr.pop # -> "dog"
arr.pop # -> "cat"
arr.pop # -> nil
```

remove first value: #shift

```
arr = %w(cat dog mouse)
arr.shift # -> "cat"
arr # -> ["dog", "mouse"]
arr.shift # -> "dog"
arr.shift # -> "mouse"
arr.shift # -> nil
```

Change the Order

sort the array: #sort

reverse the order: #reverse

```
arr = %w(first mid last)

arr.reverse # -> ["last", "mid", "first"]
arr # -> ["first", "mid", "last"]

# permanently change original arr
arr.reverse!

# useful for sorting in reverse order
arr.sort.reverse
```

randomize the order: #shuffle and #shuffle! randomize the order of elements, shuffle just a copy, and shuffle! on the array itself:

```
arr = [1,2,3,4]
arr.shuffle # -> [2, 1, 3, 4]
arr # -> [1, 2, 3, 4]

arr.shuffle! # -> [4, 2, 1, 3]
arr # -> [4, 2, 1, 3]
```

bonus: pick random value from an array: #sample returns a random value from an array:

```
[1,2,3,4].sample # -> 1
[1,2,3,4].sample # -> 3
```

#Each and #Map

```
arr = %w(a b c)
arr.each do |val|
  puts "value is #{x}"
end
# will put each value but return:
# ["a", "b", "c"]
```

Iterate through the array with an index: #each_with_index

```
arr = %w(a b c)
arr.each_with_index do |val, idx|
  puts “index #{idx}: #{val}”
end
# still returns:
# ["a", "b", "c"]
```

create a new object with each: #each_with_object is a useful method that lets you add to a given object during each iteration. At the end, instead of returning the original array, you return this object.

```
arr = %w(a b c)
arr.each_with_object({}) do |value, result|
  result[value] = value.upcase
end
# returns:
# {"a"=>"A", "b"=>"B", "c"=>"C"}
```

You can also use `#with_object` to chain together `#each_with_index`, for example to quickly stich arrays into objects:

```
#   "c"=>"I was index 2"
# }
```

iterate through array and create a new array: #map

```
arr = %w(a b c)
arr2 = arr.map do |value|
  “#{value}!!”
end

arr2 # -> [ “a!!”, “b!!”, “c!!”]
arr # ->[ “a”, “b”, “c” ]
```

As is common in ruby, the “!” means that instead of returning a copy, #map! will alter the original array:

```
arr = %w(a b c)
arr.map! do |value|
  “#{value}!!”
end

arr # -> [ “a!!”, “b!!”, “c!!”]
```

#map.with_index: by chaining #with_index, you can get access to the index like before. Note that these are #map and #with_index are two separate methods, unlike #each_with_index:

```
arr = %w(a b c)
arr.map.with_index do |value, idx|
  "index #{idx} is '#{value}'"
end
```

check if array is empty: #empty?

```
arr.empty? # returns true or false
```

concatenate (combine) arrays: note that the order in which the arrays are added matters

```
arr1 = [1, 2, 3]
arr2 = [4, 5, 6]
arr3 = [7, 8, 9]

# leaves originals alone
arr1 + arr3 + arr2
# returns:
# [1, 2, 3, 7, 8, 9, 4, 5, 6]

# if you want to permanently alter original
# use concat
arr1.concat(arr2)
arr1 # -> [1, 2, 3, 4, 5, 6]
```

see if any values match condition: #any? returns true or false, true if *at least* one of the items in an array meet a condition, false if not:

```
arr = [1,2,3,4]
arr.any? do |num|
  num > 2
end
# returns true
```

see if all or no values match condition: use #all to check that every single item fits

```
arr.none? do |num|
  num < 100
end
# returns false
```

find the first value to match a condition: #find

```
[1, 2, 3, 4].find do |num|
  num.even?
end
# returns 2
```

return all values to match a condition: #select

```
[1, 2, 3, 4].select do |num|
  num.even?
end
# returns [2, 4]
```

If you only care about the number of elements that meet a condition, then use #count. It just returns the number of values:

```
[1, 2, 3, 4, 5, 6].count do |num|
  num.even?
end
# returns 3
```

remove duplicate or nil values: #uniq knocks out duplicates and #copact takes out nil values. They both return copies of the arrays, use #uniq! or #compact! to

happy coding everyone,

mike

)

Programming Software Engineering Ruby Ruby on Rails Web Development

About Help Legal