Lesson 4: Arrays, Hashes, and Each Loops

[Overview](#h.9wkhsp5ykzac)

[Review Homework](#h.1fzyo0fjplgs)

[Arrays](#h.tqyfoc9m6zzf)

[Each Loops](#h.4yzlwkuydqhu)

[Hashes](#h.i308xrbb45zc)

[Homework](#h.tabmx6ugbfdj)

# Overview

1. Review homework
2. Arrays data type
3. Each loops
4. Hashes data type

# Review Homework

20 minutes

# Arrays

The arrays data type gives us a way to store objects inside another object in an ordered way. It is essentially an ordered list. This list can contain any other data type - integers, floats, strings, another array, and more!

Let’s jump into IRB to practice with arrays (I’ll provide arrays.rb which contains this IRB practice so they don’t have to worry about copying for notes). There are 2 different ways to initialize an array. We can set an empty array like when we set a variable, or we can explicitly call it a new array.

my\_first\_array = []

my\_second\_array = Array.new

As you can see, arrays are represented with square brackets. We can also put things inside of arrays in different ways. The first way is to set each item explicitly one-by-one.

my\_first\_array[0] = "first item"

(then type my\_first\_array after each so they see the effect)

my\_first\_array[1] = 2.0

my\_first\_array[2] = 3

What’s happening with the numbers? Arrays are zero-indexed. The index always start at zero, not one. This is a common source of errors for beginners. We call what normal refer to as the first item the **zeroth** item. Let’s go through numbering on the board (write on white board):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Array | apple | orange | banana | monkey |
| Index → | 0 | 1 | 2 | 3 |
| Backwards index ← | -4 | -3 | -2 | -1 |

Now let’s try calling terms both forwards and backwards in IRB. We can also put things in a new array all at once like so:

my\_array = ["apple", "orange", "banana", "monkey"]

my\_array[0]

my\_array[-1]

(practice more indices)

Remember how I said we can even add arrays to arrays? Let’s do that now:

my\_first\_array[3] = ["fourth", 4, 4.0]

Pretty cool, huh? How would we call the zeroth item of the third item of my\_first\_array? Test in IRB. (my\_first\_array[3][0])

Ruby has many methods for arrays (remind to Google “array methods ruby”). Let’s try a couple in IRB:

dogs = []

dogs.push("Priscilla")

dogs.push("Harry")

dogs << "foster pup" (this is the same as push)

Let’s say the foster pup got adopted:

dogs.pop

dogs << "cat"

dogs.reverse

dogs

dogs.reverse!

dogs

dogs.length

The bang (!) makes changes permanent.

Classroom Challenge (pair program): Create an array with at least 5 elements. Call the .reverse and .length methods on your array. Nest the .shuffle and .join methods to list your array objects in random order, separated by commas.

# Each Loops

Now that we know enough about arrays, we can learn about looping through arrays! We already know enough to perform a **while** loop on an array:

scores = [100, 80, 75, 93]

counter = 0

sum = 0

while counter < scores.length

sum += scores[counter]

counter += 1

end

puts "The sum of #{scores} is #{sum}."

A more efficient way of doing this loop is with an each loop.

scores.each do |score|

sum += score

end

puts "The sum of #{scores} is #{sum}."

We don’t need a counter anymore because Ruby automatically iterates through each element in the array when we do a “.each” on it. We put what we want to call each of these elements in the block - “score”. Then when we want to call just that element, we call “score” as if it were a variable.

Classroom challenge (individual): Modify the program to calculate the average also.

Remember inline loops? Well, we can do that with the each loop too:

scores.each { |score| sum += score }

Classroom challenge (individual or pair or all together): Create a trivia app that stores 3 questions in a questions array and the 3 simply answers in an answers array. The app will ask the user the question then check the answer.

# Pop Trivia App

questions = ["Who sang Material Girl?", "Which actor played Zoolander?", "Who loves tacos?"]

# We put all the answers in lowercase to make checking answers easier.

answers = ["madonna", "ben stiller", "gant"]

puts "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

puts " Welcome to Sia's Trivia App!!"

puts "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~"

puts "Let's get started...\n"

i = 0

while i < questions.length

puts questions[i]

answer = gets.chomp

if answer.downcase == answers[i]

puts "You are correct!"

else

puts "Sorry, Charlie, but that is incorrect."

end

i+=1

end

puts "\nThanks for playing trivia!"

# Hashes

Hashes, like arrays, are also collections of data. However, they are not ordered and the information is stored and accessed in pairs called keys and values.

You can think of an array as an ordered list and an hash as an unordered dictionary. Remember our trivia app? It would probably be a lot easier to just store an answer with a question rather than trying to make sure both arrays are perfectly aligned.

A hash is initialized with curly brackets, and, like arrays, can be created and populated a few different ways. Let’s jump into IRB and practice (share hashes.rb so that they can play without copying everything over):

my\_hash = {}

my\_hash = Hash.new

my\_hash["name"] = "Sia"

my\_hash["age"] = 30

my\_hash

grades = {"Priscilla" => 80, "Harry" => 70}

grades["Priscilla"]

grades2 = {Sia: 100, Gant: 50}

grades2[:Sia]

lunch = {:location => "Laurel Street Bakery", :attendees => ["Sia", "Harry"]}

lunch[:location]

grades3 = grades2.to\_a

We can also loop through hashes!

my\_hash.each { |key, value| puts "The key is #{key} and the value is #{value}" }

Now let’s play in our trivia app. Classroom challenge: Convert trivia.rb to use a hash for the questions and answers.

# Homework

See it [here](https://docs.google.com/document/d/19MAexqZriLKXXrsrldWavs0J_TCuQ3JrM1K0QKHmOSo/edit?usp=sharing).