Open in new tab

Arrays

Lesson Duration: 60 minutes

Learning Objectives

- Explain what arrays are
- Create an array
- Access an element from an array
- Add items to arrays
- Call array methods

What Are Arrays

We’ve been dealing a lot with single objects - a person, a number. Just as with real life, we don’t always have one of something - we could have a group of things. At home we don’t just have one piece of food, we have many. We don’t have one pair of shoes, we have many. So how do we represent this in our code?

This is where Arrays come in. An array is part of a subset of objects known as “collections”. It is essentially a storage container like a big basket, or bag, or box. In real life we might be storing fruit, shoes or sweets but in our code we can store any object we like of any of the datatypes we have seen - integers, strings, even booleans.

The key thing about arrays is that they are ‘indexed’, that is, you can access the things inside it by their index. It’s all very well having a shopping bag full of fruit, but how do we find an apple in there? We’d have to search through every other item until we found it.

It’s kind of like our lockers. They are ordered top to bottom and if we put items in them, even though we can’t see them anyone we know which locker they are in. As long as noone comes along and reorders our items without us knowing, we can always guarantee our items are in the same locker we left them.

What Are Arrays for in Programming

Given an array is a collection, we use them when we want to collect related items together to do useful things with them. Maybe we want to have a set of people’s names or a list of lottery numbers. We can pass this around our program and do interesting things with it - most notably use loops to do repeated actions to every item in the set. We call this “enumerating”.

This is such important, useful functionality that we’ll cover it in a whole dedicated lesson.

What Are They Not For

Arrays are pretty dumb, all they know is where something is in the list and no other information. It has no idea that the second item is a banana, or that bananas are yellow. It knows if it has an item at position 2 or not and that’s about it.

So if we wanted to find a certain person’s locker and we didn’t know which position it was in, we have no other information to help us. We’d need to go through each one until we found it. Later on you’ll see why using a hash would be a more efficient way to do it.

Accessing Arrays

All “keys” to an array are integers, we call this the “index”. The first element in an array is at index zero, and the amount of elements can generally go up as high as your computer/programming language will allow. Some languages require that you specify how big is each new array you use (to know how much space to allocate in memory), but Ruby is a bit more flexible, and will grow arrays automatically to fit as many items as you add to it.

So what does this look like?

First, let’s create a file in your working directory!

```
touch arrays.rb
```

```
# arrays.rb
fruits = ['apple', 'banana', 'grape', 'orange']
p fruits
```

Arrays are characterised by the square brackets around the elements.

Array indexes in most programming languages start at 0 (the reason for this is a discussion for another day).

```
# arrays.rb
p fruits[0]
p fruits[2]
p fruits[4] # what would you expect this to return?
```

In Ruby, if you’re trying to access an element with an index that is out of bounds, you’ll get back `nil`.

Unlike many other languages, Ruby lets us access elements from the end of the array using a negative index too

```
# arrays.rb
p fruits[-1]
p fruits[-2]
```

There are also some helpful utility “methods” (we will cover this later - just now let’s accept that arrays come with some helpful magic) to allow us to do this.

```
# arrays.rb
p fruits.first()
p fruits.first(2)
p fruits.last()
p fruits.last(2)
```

Creating an Empty Array

There will be times where you want to initialize an empty array. We can do this in several ways, including

```
# arrays.rb
my_array = []
my_array = Array.new()
```

Adding and Removing Items

We will so often need to add and remove items from arrays, that there are numerous ways to do it.

We can do assignment by index like we were doing to access the array

```
# arrays.rb
fruits[1] = 'mango'
p fruits
# notice that this overwrites whatever was already there -- any existing value is gone for ever (like re-assigning a variable)

fruits[100] = 'peach' # think about what the `fruits` variable might look like now... then see
p fruits
```

We can add/remove to the end of the array.

```
# arrays.rb
fruits.push('pear')
p fruits
fruits.pop()
p fruits
fruits << 'lemon' # referred to as the "shovel" operator
p fruits
```

And to the start of the array.

```
# arrays.rb
fruits.shift()
p fruits
fruits.unshift('apple')
p fruits
```

And of course, by index.

Array Elements

Elements can be *any* type of object; literal values or variables. Unlike other languages, Ruby allows us to put a mixture of different kind of objects in an array.

```
fruits_and_numbers = [ 'apple', 1, 'grape', 2 ]
p fruits_and_numbers
```

We can even put arrays inside of arrays.

```
what = [ 1, 2, 3, 4, [5, 6, 7] ]
p what
```

> [Task:] Go have a look at array methods in the docs and show some cool stuff you find

https://ruby-doc.org/core-2.4.1/Array.html

Published with GitHub Pages