Arrays

We are going to use IRB to have a look at **arrays**.

An array is an **ordered list of objects** .You can create an array literal by placing the objects inside square brackets separated by commas:

```
arr = [1,2,3]
```

You can have an array of strings as well:

```
arr = ['one', 'two', 'three']
```

There's also a shorthand way for creating arrays of strings:

```
%w('one two three')
```

The good thing about this, is you don't need to put the strings inside as well as you don't need to separate them by commas - a space will do.

You don't even have to use the same type of object inside an array:

```
arr = [1, 2, 'three']
```

Multi-dimensional Array

You can even have an array that has arrays inside it:

```
arr = [1, 2, ['a', 'b']]
```

This is called a **multi-dimensional array**, and it's useful when, for example, creating a type of coordinate system.

Arrays in IRB

Arrays are a powerful tool in the Ruby toolkit, and they have some very useful methods. To demonstrate these, I'm going to create an array that contains a list of items on my shopping list:

```
list = %w(apples bananas milk bread)
```

To access a specific value in an array, we write the position of where it is in the array, in square brackets (known as the **index**). So if I wanted to find the item at position one I would enter the name of the array, square brackets, and inside the square brackets I place the index:

```
list[1]
```

`bananas` will be returned, but notice that this isn't actually the first item in the array. It should have been `apples`, but it returned `bananas`. There's nothing wrong here. What's happening is that arrays start their numbering at `0`. So if you wanted to know what the first item in an array is, you need to pass an index of `0`:

```
list[0]
```

Negative Values

The numbering system also can use negative values to start counting from the back. So, if I wanted to find the last item in the list, I would enter an index as `-1`:

```
list[-1]
```

There are also some specific methods for finding first and last items in an array:

```
list.first
list.last
```

Subset Array

We can also return a subset of the array by specifying a second parameter with the index. This indicates the length of the sub-array that we want returned. So if I'm on to the first three items in my shopping list, I would enter

```
list[0,3]
```

3 means that I want to get three items.

Alternatively, we can supply range as the argument:

```
list[0..2]
```

Use `size` method to know how many items are there in the array:

```
list.size
```

Length Method

Another name of the `size` method is `length`:

```
list.length
```

If I want to check if I have apples on my shopping list, I would write

```
list.include?('apples')
```

`include?` is a boolean method that takes a parameter of what you are looking for.

If I want to add something to an array, I can use the push method:

```
list.push('cheese')
```

There is a shorthand operator for doing this that uses double angle brackets:

```
list << 'chocolate'
```

Push Method

What's quite nice about this method is those double angle brackets almost look like they're pointing towards the array saying "push this string into the array".

We can remove an item from an array using the pop method:

```
list.pop
```

I don't need any parameters to this method, because it will automatically just remove the last item in the array. Notice that it will return the string that you've removed, so it's useful for gaining access to the last item in the array.

Sort Method

We can sort an array into order using the `sort` method. This will be sorted alphabetically:

```
list.sort
```

However it does not change the original array, so we need to employ `sort!` method for this to happen:

```
list.sort!
```

Reverse Method

We can also use the `reverse` method to write an array backwards (it also has a bang method):

```
list.reverse
```

One last method we'll look at is the `join` method, which is very useful, because it can be used to create a string that joins all the elements of an array together. It takes an argument to indicate what you want to be used as a separator:

```
list.join(',')
```

You will see a string that has all the items from our array separated by commas.