**Hey, Whoa, Level #2 Already**

**Objective:**

Let's look at what you've learned in the first minute.

* **The prompt.** Typing code into the prompt gives you an answer.
* **Numbers and strings** are Ruby's math and text objects.
* **Methods.** You've used English-language methods like reverse and symbolic methods like \* (the multiplication method.) Methods are actions!

This is the essence of your learning. Taking simple things, toying with them and turning them into new things! Feeling comfortable yet? I promise that we'll get you there..

But now, let's do something a little uncomfortable. Try using that reverse method on a number: 40.reverse

## Stop, You're Barking Mad!

### Objective:

You can't reverse the number 40. I guess you can hold your monitor up to the mirror, but reversing a number just doesn't make sense! Ruby has tossed you a useful error message.

The message is telling you that there is no method reverse for number values in Ruby!

But, hmm...maybe if you can turn it into a string. Try this: 40.to\_s.reverse.

> 40.reverse­

=> #<NoMethodError: undefined method `reverse' for 40:Fixnum>

Oh no!

> "40".rever­se

=> "04"

That's better!

> back

> 40.to\_s.re­verse

=> "04"

That's better!

**Boys are Different From Girls**

**Objective:**

...just like numbers are different from strings. While you can use methods on any object in Ruby, some methods only work on certain types of values. A really cool thing about Ruby is that you can always convert between different types using Ruby's "to" methods.

* **to\_s** converts values to **s**trings.
* **to\_i** converts values to **i**ntegers (numbers.)
* **to\_a** converts values to **a**rrays.

*What in the world are arrays, you might ask?!* They are simply lists. Let's make an empty one, by typing in a pair of brackets: [].

> []

=> []

## Standing in Line

### Objective:

Great, you built an empty array. Now let's see what else we can do with it.

First off, a good thing to know is that arrays store their information in a **sequence**. Think of this like standing in line for popcorn. You are behind someone and you wouldn't dream of pushing them aside, right? And that guy behind you, you've got a close eye on him, too. First come, first serve.

Just like that line for popcorn, the order of an array's information will stay consistent for you after you build it...well, at least until you modify it.

To try building an array with some stuff in it, here's a list of lottery numbers for you: [12, 47, 35]. See those commas? They're important!

> [12, 47, 35]

=> [12, 47, 35]

Success!

## One Raises Its Hand

### Objective:

Sweet, you've got a short list of lottery numbers. Now, what if we wanted to know which one is the highest in the array?

Try this: [12, 47, 35].max.

> [12, 47, 35].m­in

=> 12

Success!

> [12, 47, 35].m­ax

=> 47

## Tucking a List Away

### Objective:

Good, good. But it would be pretty annoying to have to retype that list every time, right?

Let's fix that by using a Ruby **variable**, which helps us **store** important data. Each variable has a unique name, so that it can be summoned up whenever we need the info it contains.

Call your new variable ticket and place your lottery numbers inside it, like so: ticket = [12, 47, 35]. That equal sign you see is what assigns your array to the new variable.

> ticket = [12, 47, 35]

=> [12, 47, 35]

Success!

> ticket

=> [12, 47, 35]

Success!

## Saved, Tucked Away

### Objective:

Fantastic! You've hung on to your lotto numbers, tucking them away inside a **variable** called ticket.

Now let's put your lotto numbers in order...sound good? Ruby has a great method for that. Use: ticket.sort!

You might notice that the method has an exclamation point at its end. This just signals that we intend for Ruby to directly modify the same array that we've built, rather than make a brand new copy that is sorted. You'll notice that if you try calling ticket again, it will be sorted permanently!

When you want to move on, just type next

> ticket.sor­t

=> [12, 35, 47]