**Level #3 is Upon Us**

**Objective:**

You built a list. Then you sorted the list. And as you've seen, the ticket variable is now changed.

Now, let's look at how your second level went down:

* **Errors.** If you try to reverse a number or do anything fishy, Ruby will skip the prompt and tell you to straighten up.
* **Arrays** are lists of stored information.
* **Variables** are a place to save stuff you might need again, as well as give that stuff a name. You used the equals sign to do this, in a process called assignment.  
  Like: ticket = [14, 37, 18].

In all, there are just eight levels in this course. You are already two-eighths of the way to the end! This is simple stuff, don't you think? More good stuff up ahead.

Let's change directions for a moment. I've stuffed a bit of poetry for you in a certain variable. Take a look, by typing print poem

> print poem

=> "My toast has flown from my hand  
And my toast has gone to the moon.  
But when I saw it on television,  
Planting our flag on Halley's comet,  
More still did I want to eat it."

Success!

## Sadly, You Hate Toast Poetry

### Objective:

Look, it's okay. You don't have to like it. You may even want to hack it up. Welp, be my guest.

Instead of toast, maybe go for a melon or something. Try this one: poem['toast'] = 'honeydew'

> poem['toas­t'] = 'hone­ydew'

=> "honeydew"

Success!

## Sadly, You Hate Toast Poetry

### Objective:

Now type print poem once again to see the new poem.

See how you only changed the first toast? The joke's on you, bread hater.

When you want to move on, type next

> poem['toas­t'] = 'hone­ydew'

=> "honeydew"

Success!

> print poem

=> "My honeydew has flown from my hand  
And my toast has gone to the moon.  
But when I saw it on television,  
Planting our flag on Halley's comet,  
More still did I want to eat it."

> print poem[­'toast']

=> "toast"

> print poem[­"toast"]

=> "toast"

> poem["toas­t"]

=> "toast"

> poem['toas­t']

=> "toast"

> poem['toas­­t'] = 'hone­­ydew'

=> #<IndexError: string not matched>

> 'hone­ydew­'

=> "hone?ydew"

> 'toast'

=> "toast"

> toast

=> #<NameError: undefined local variable or method `toast' for main:Object>

## Sadly, You Hate Toast Poetry

### Objective:

Now type print poem once again to see the new poem.

See how you only changed the first toast? The joke's on you, bread hater.

When you want to move on, type next

## Ready, Aim

### Objective:

The square brackets you just used are very common in Ruby. Remember, you typed: poem['toast'] = 'honeydew'. That box that holds the word toast uses a square bracket on each side. See 'em?

The two brackets are like a crosshairs used to line up precisely on a target. These brackets mean, "I am looking for \_\_\_\_ somewhere in here." Ready...aim...**data**. Here, you were looking specifically for toast in order to swap it out with a fruit.

Let's see if your new experience can help you produce the answer to this question: what happens when we reverse this whole poem? poem.reverse

> poem.rever­se

=> ".ti tae ot tnaw I did llits eroM  
,temoc s'yellaH no galf ruo gnitnalP  
,noisivelet no ti was I nehw tuB  
.noom eht ot enog sah tsaot ym dnA  
dnah ym morf nwolf sah wedyenoh yM"

Success!

>

## Too Much Reversal

### Objective:

Okay, I suppose that was expected. The whole poem's been turned backwards, letter by letter. But say I really just wanted to reverse the lines only. In other words, move the last line up to first and the first line down to last. Backwards, yes, but not that backwards.

Ruby has a way. Try this: poem.lines.to\_a.reverse

> poem.lines­.to\_a.reve­rse

=> ["More still did I want to eat it.  
", "Planting our flag on Halley's comet,  
", "But when I saw it on television,  
", "And my toast has gone to the moon.  
", "My honeydew has flown from my hand  
"]

Success!

> poem.lines­

#<Enumerator: "My honeydew has flown from my hand\  
And my toast has gone to the moon.\  
But when I saw it on television,\  
Planting our flag on Halley's comet,\  
More still did I want to eat it.\  
":lines>

Make sure to print the reversed lines of the poem.

>

> poem.lines­.to\_s.reve­rse

">b3d43047x0:rotaremunE<#"

Make sure to print the reversed lines of the poem.

> poem.lines­.to\_a.reve­rse

["More still did I want to eat it.\  
", "Planting our flag on Halley's comet,\  
", "But when I saw it on television,\  
", "And my toast has gone to the moon.\  
", "My honeydew has flown from my hand\  
"]

Make sure to print the reversed lines of the poem.

>

> poem.lengt­h

173

## Ringlets of Chained Methods

### Objective:

So...what actually happened there? You typed poem.lines.to\_a.reverse and produced some Ruby magic.

First, you turned the poem into a list using lines.to\_a. The lines component decided the way the string should be split up, and then one of our "to" methods, to\_a, converted those splits into an Array. (**to\_a**rray.)

Different methods, such as bytes and chars can be used in place of lines. By using lines here, Ruby split the poem up according to each new line.

After that, you reverse'd your Array. You had each line prepared in advance. And then you reversed them. That's it!

And now, let's tack one more method on the end there, if you don't mind. Try: print poem.lines.to\_a.reverse.join.

> print poem.­lines.to\_a­.reverse

["More still did I want to eat it.\  
", "Planting our flag on Halley's comet,\  
", "But when I saw it on television,\  
", "And my toast has gone to the moon.\  
", "My honeydew has flown from my hand\  
"]

> print poem.­lines.to\_a­.reverse.j­oin

More still did I want to eat it.  
Planting our flag on Halley's comet,  
But when I saw it on television,  
And my toast has gone to the moon.  
My honeydew has flown from my hand

Success!

>