### Objective

### At this point, we really only have one complete player: larry with a health of 60. Our other players—curly and moe—haven't yet been assigned a health. In this exercise, we'll give them some life and also add a fourth player. The goal is to print out each player's name and health with a slightly different format, like so:

Larry has a health of 60.  
CURLY has a health of 125.  
\*\*\*\*\*\*\*\*\*\*\*\*\*Moe has a health of 100.\*\*\*\*\*\*\*\*\*\*\*\*\*  
Shemp........................... 90 health

Along the way we'll practice assigning objects to variables, calling methods on those objects, and finding documentation on methods in Ruby's standard library.

1. Player #1: Larry

As our game stands now, we have two variables holding onto the name and health of the first player:

name1 = "larry"  
health1 = 60

Our first formatting task is to capitalize the name.

1. We've seen a few string methods, but we've barely scratched the surface of what Ruby strings can do. Thankfully, you don't have to remember all the string-related methods. The ri command-line utility allows you to view information about Ruby classes, modules, and methods from the comfort of your command line. Use ri to view the RDoc documentation for the String class.

**ri String**

1. **Note:** If you don't have the RDoc documentation installed locally, you can use the [online documentation](http://ruby-doc.org/core) instead. Type "String" into the search field in the upper-right corner and then click on the first "Class: String" result.
2. The *instance methods* are the methods that you can call on a String object.
3. To exit the ri session, either press the q key to exit immediately or press the spacebar repeatedly to scroll through to the end.
4. **Windows Users:** If you see funny ANSI characters in the output generated by ri, a quick fix is to install ANSICON.
5. Now narrow it down to get more information about the capitalize method.

**ri string.capitalize**

1. The documentation usually includes example uses of each method. Reading them is one thing, but even better is to type an example into an irb session and run it for yourself. It's a great way to learn new methods... with immediate feedback!
2. **Tip:** Alternatively, you can start ri in *interactive mode* with the -i option, like so:
3. ri -i
4. You should see an interactive prompt where you can type the classes and methods you're looking for and use **Tab** for autocompletion. For example, type String.cap and then hit the **Tab** key to see the matching methods. To select a matching method, hit **Return**. When you're done reading the documentation for a particular class or method, you end up back at the prompt for more searching. To exit, simply enter a blank line.
5. Finally, returning to our objective, back in your studio\_game.rb file change how the first player is printed so his name is capitalized. Using the two variables above, here's the output you want:
6. Larry has a health of 60.

**puts “#{name1.capitalize} has a health of #{health1}”**

2. Player #2: Curly

For our second player, we want to print his information with the name in all uppercase letters.

1. Create variables for a second player with the name "curly" and an initial health of 125. Then use those variables to print his information like this:
2. CURLY has a health of 125.
3. If you don't remember the method to capitalize all the characters in a String, don't hesitate to use ri again to look it up.

**name2 = “curly”  
health2 = 125  
puts “#{name2.upcase} has a health of #{health2}”**

1. Now it's time to experiment a little. Add two lines to the bottom of the program file: change Curly's health variable to point to Larry's health variable and print out Curly's information again.

**health2 = health1  
puts “#{name2.upcase} has a health of #{health2}”**

1. What do you expect to see? Re-run the program.
2. Finally, add a few more lines to the bottom of the program. Reassign Larry's health variable to be 30. Then print out the name and health of both players again.

**health1 = 30  
puts “#{name1} has a health of #{health1}”  
puts “#{name2} has a health of #{health2}”**

1. What result do you expect for Larry and Curly now? Re-run the program.
2. The assignment of variables can be a bit tricky, especially when variables reference other variables. You should now have:

Larry has a health of 30.  
CURLY has a health of 60.

1. Make sure you understand why this works before moving on. Don't hesitate to go back and watch the part in the video about the variables my\_favorite\_movie and your\_favorite\_movie if you need help. Or grab a sharpie and a napkin and draw it out for yourself if that helps.

### 3. Player #3: Moe

### The third player's information needs to be printed so that it's all centered and padded with asterisks.

1. Use the documentation to find the method that centers a string. Note that the first parameter (the length) is required, but the second parameter for the character to use as padding is optional. Try the examples in irb to get a feel for how the method works.

**ri String.center**

1. Create variables for a third player with the name "moe" and an initial health of 100. Then use those variables to print his information like so:
2. \*\*\*\*\*\*\*\*\*\*\*\*Moe has a health of 100.\*\*\*\*\*\*\*\*\*\*\*\*
3. There are at least two ways to do this, so experiment a little before looking at the answer.

**name3 = moe  
health3 = 100  
puts “#{name3.capitalize} has a health of #{health3}.”.center(50, ‘\*’)**

### 4. Player #4: Shemp

1. Create a fourth player named "shemp" with an initial health of 90. Use variables to print out his information with the name capitalized and left-justified, like this: Shemp........................... 90 health
2. There are at least two ways to do this, so experiment a little before looking at the answer.

**name4 = shemp  
health4 = 90  
custom\_name = name4.capitalize.ljust(30, ‘.’)  
puts “#{custom\_name} #{health4} health”**

### 5. Reverse It

1. In the video, we used the reverse method to print a reversed version of the string "Goonies". What happens if you try to reverse a number, such as a player's health. Try it by opening an irb session and typing:
2. >> 123.reverse
3. Hmm, why doesn't that work?
4. Take a moment to study the error message:
5. **NoMethodError**: undefined method `reverse' for 123:Fixnum
6. Any time you see an error like this, take the time to read it. This is Ruby's way of trying to help us understand what went wrong. And in this case, the error is quite helpful. It's saying that it couldn't find a reverse method to call on the object 123, which is a Fixnum.
7. Indeed, if you look up the documentation for the reverse method you'll find that it's not defined on the Fixnum class. So even though you can call methods on objects, only certain methods are defined on certain classes of objects.
8. We know we can reverse a string, so how could we convert a number object to a string object? It turns out that all objects in Ruby have a to\_s method that converts the object to its string representation. Using this tip, try chaining methods together to reverse the number 123 again.

**123.to\_s.reverse**

1. How then would you convert the reversed string back to an integer?

**123.to\_s.reverse.to\_int**

### 6. Vocabulary Quiz

### Learning to program can feel a bit like learning a foreign language. Here are a couple sentences to help you practice your new language skills. (Fill In the Blanks)

1. In the following line of code, capitalize is a **method** called on the **object** named name and assigned to the **variable** named text.

text = "#{name.capitalize} has a health of #{health}."

1. In the case of the line of code below, center is a **method** that takes two **parameters.**

puts text.center(50, '\*')

### Bonus Round

### Time Method

### You've learned how to use the documentation and call methods on strings and numbers. The Ruby standard library has a host of other classes that you can use in your programs, as well.

### For example, the Time class is useful for working with dates and times. Unlike strings and numbers, which we created using literal values, you create a new Time object like this:

current\_time = Time.new

The Time class has a method called strftime that can format the time in a variety of ways. Check out the method's documentation and play around with it in irb until you've got the hang of it.

Then, back in your studio\_game.rb file, print the date and time the game started so that it's formatted like this:

The game started on Tuesday 03/06/2012 at 10:43AM

current\_time = Time.new  
formatted\_time = current\_time.strftime("%A %m/%d/%Y at %I:%M%p")  
puts "The game started on #{formatted\_time}"

Fundraising Program

If you're building the fundraising program, your next step is to assign a funding amount to each project. In other words, how much funding does each project have so far? Then, experiment with different formats for printing out each project's name and funding amount.

**project1 = 'Project ABC'**

**project2 = 'Project LMN'**

**project3 = 'Project XYZ'**

### funding1 = 1000 funding2 = 500 funding3 = 25

### puts "#{project1} has $#{funding1} in funding."

### puts "#{project1.capitalize} has $#{funding1} in funding."

### puts "#{project2.upcase} has $#{funding2} in funding."

### puts "#{project3.ljust(30, '.')} $#{funding3} in funding."

### puts "Projects: \n\t#{project1}\n\t#{project2}\n\t#{project3}"

### current\_time = Time.new

### formatted\_time = current\_time.strftime("%A %m/%d/%Y at %I:%M%p")

### puts "Funding amounts as of #{formatted\_time}."

### 

### Wrap Up

### Nicely done! Now you have all the game players introducing themselves. In this exercise you learned how to:

* assign variables
* call built-in methods
* pass parameters to methods
* chain together multiple method calls
* look up methods in the Ruby documentation

But what if you want to do something for which there isn't a built-in Ruby method? Well, then it's time to write your own method! We'll do that in the next section.