### Objective

### In the previous exercise, we experimented with a variety of ways to start our game using different formats for each player's name and health. In this exercise, we'll write our own Ruby method to establish a consistent way to format the information for all the players.

### At the end, we want to be able to run this code:

puts say\_hello("larry", 60)  
puts say\_hello("curly", 125)  
puts say\_hello("moe")  
puts say\_hello("shemp", 90)

and get this output:

I'm Larry with a health of 60.  
I'm Curly with a health of 125.  
I'm Moe with a health of 100.  
I'm Shemp with a health of 90.

### 1. Write a Method

### We need a single, authoritative way to print a player's information. To do that, we'll use a *method*. You can think of a method as a reusable chunk of code that you call by name. A method should do one thing well. The more focused a method, the better.

### You define a method with the keyword def, followed by the method name and the method's parameters between parentheses. The body of a method is generally indented two spaces, though white space is not significant in Ruby. You finish the method body with the keyword end.

### 

### Just like with variables, it's important to give your methods good names. (Naming things is one of the hardest parts of programming!) Method names should start with a lowercase letter, followed by a mix of letters, numbers, and underscores. As we saw with the reverse! and empty? methods, method names may end with a question mark (?) or exclamation point (!). There are a couple other variations, but the important thing to remember is to start a method name with a lowercase letter and separate words with an underscore.

1. Go ahead and delete the code currently in your studio\_game.rb file (or save it in a different file) so you have a clean slate. We'll create the players using methods in this exercise, and it's good practice to start from scratch again.
2. At the top of studio\_game.rb, define a say\_hello method that takes a name parameter and prints out the player's name. For example, if you call the method with the name "larry", it should print out "I'm Larry" with the name capitalized. Don't you dare look at the answer until you’ve tried something!

**def say\_hello(name)  
 puts “I’m #{name.capitalize}.”  
end**

1. After you've defined the say\_hello method, call it several times passing in a different player's name each time.

**say\_hello(“larry”)  
say\_hello(“curly”)  
say\_hello(“moe”)  
say\_hello(“shemp”)**

1. Don't forget to run the program to make sure it's doing what you expect!
2. Now suppose we don't want the say\_hello method to print the string to the console. Instead, we want the method to simply return a string.
3. Change the say\_hello method to return a string and also change how the say\_hello method is called so that the returned value is printed to the console.

**def say\_hello(name)  
 “I’m #{name.capitalize}.”  
end**  
**puts say\_hello(“moe”)**

1. The say\_hello method takes a name parameter. What do you suppose will happen if you try to run the following line of code *outside* of the say\_hello method? Go ahead, try it before peeking at the answer!
2. puts name

**Got an error “NameError: undefined local variable or method 'name' for main:Object”**

1. Why do you get that error?

**I think “name” is a local variable inside say\_hello method and it’s not accessible for outside that particular method.**

Make sure your program runs without errors before moving on.

### 2. Add a Health Parameter

### Our players also have a health value, so next we need to include their health in the greeting.

1. Change the say\_hello method so that you can call it with the player's name *and* health like so:
2. puts say\_hello("larry", 60)  
   puts say\_hello("curly", 125)  
   puts say\_hello("moe", 100)  
   puts say\_hello("shemp", 90)

**def say\_hello(name, health)   
 “I’m #{name.capitalize}.”  
end**

1. In the method body, change the player's information string to include the player's health. For example, Moe's information should be returned from the method as "I'm Moe with a health of 100."

**def say\_hello(name, health)**  
 **“I’m #{name.capitalize} with a health of #{health}.”**  
**end**

### 3. Set a Default Parameter Value

### Now suppose we'd like the player's health to default to a value of 100 if a specific health value isn't passed into the say\_hellomethod.

1. Change the method so that the health parameter has a default value of 100.

**def say\_hello(name, health=100)  
 “I’m #{name.capitalize} with a health of #{health}.”  
end**

1. Looking back at Moe's initial health value, notice that it's 100. Oh, isn't that convenient! That's the same as the default value of health parameter for the say\_hello method.
2. Change the call to the say\_hello method for Moe to use the default parameter value.

**puts say\_hello(“moe”)**

Run the program and make sure you get these results:

1. I'm Larry with a health of 60.  
   I'm Curly with a health of 125.  
   I'm Moe with a health of 100.  
   I'm Shemp with a health of 90.

### 4. Calling Other Methods

### Remember that a method should do *one thing* well. Then, to get more complex behavior, you compose your program of many small methods. Methods can invoke other methods by simply calling the method name.

### Change each player's information to include the current time, which is returned by another method. Here's an example of the output you're aiming for:

I'm Moe with a health of 100 as of 05:25:08.

1. In your studio\_game.rb file, define a new time method that returns a string representing the current hour of the day, minute of the hour, and second of the minute (e.g., "05:25:08").

**def time  
 current\_time = Time.new  
 current\_time.strftime(“%I:%M:%S”)  
end**

1. Then change the say\_hello method to call the time method so that the player's information includes the current time. (In the case, we'll interpolate the method call in a string, so we'll need to use #{ }.)

**def say\_hello(name, health=100)  
 “I’m #{name.capitalize} with a health of #{health} as of #{time}.”  
end**

1. Run your code and make sure each player's greeting includes the current time.

### Bonus Round

### Fundraising Program

### If you're building the fundraising program, now's the time to decide on a single format for introducing your projects. Pick whatever format suits your unique style and create a method to display the information consistently for all projects. For projects without a specified funding amount, set an initial default amount.

def time  
 current\_time = Time.new  
 current\_time.strftime("%A, %B %d, %Y")  
end

def project\_listing(name, funding=0)  
 "#{name} has $#{funding} in funding as of #{time}."  
end

puts project\_listing("Project ABC", 1000)  
puts project\_listing("Project LMN", 500)  
puts project\_listing("Project XYZ", 25)  
puts project\_listing("Project TBD")

### Wrap Up

### Knowing how to write and call methods is not only absolutely essential, it's also a great way to clean up duplication in your programs. We'll return to methods over and over in this course and we'll build on this in upcoming exercises.

### Before moving on, make sure you're comfortable with:

* defining your own methods with parameters
* calling those methods
* setting default parameter values
* calling one method from another method

As it stands now in the game, each player has two characteristics: a name and a health. And this will be true for all our players, even the ones who haven't yet heard about our game but will undoubtedly want to play it when we're finished. So next we'll learn about classes and how we can use them to easily and consistently create new player objects.