### Objective

### Now that you know a bit more about numbers and strings, we'll start to add some players to the game. The goal of this exercise is to print out info about our initial player and a list of all the players. Here's the output we're shooting for:

larry's health is 60  
  
Players:   
 larry  
 curly  
 moe

### 1. Print Player Info

### Each player has a name (a string) and an initial health value (an integer number). We need a way to remember a player's name and health so we can refer to them later on in the game. To do that, we'll need to assign the name and health values to *variables*.

### Variables simply name things in our program. It's important to give the variables good names so it's easy to tell what they're referencing. Ruby has some variable naming rules: variable names can be a mix of letter and numbers (or an underscore), and the first character of a variable name must be a lowercase letter or an underscore.

### For example, the first player's name could be stored in the variable name1. Sometimes the best variable name has more than one word, in which case you separate the words with an underscore. For example, a good name for a player's high score would be high\_score.

### 

### With that in mind, let's create our first player...

1. Go ahead and delete the code currently in your studio\_game.rb file so you have a clean slate.
2. The name of our first player is "larry" (he prefers the lowercase pronunciation). Create a variable to remember that name. In other words, *assign* the player's name to the variable.

**name1 = larry**

1. Larry's initial health value is 60. Create a second variable and assign the health value to it.

**health1 = 60**

1. Next, using those two variables, create the single-quoted string "larry's health is 60" and print it out to the console. Keep in mind that in a single-quoted string, you'll need to use a backslash to escape the apostrophe in the possessive form of the player's name.

**puts name1 + ‘\’s’ + ‘health is’ + health1.to\_s**

Don't forget to run the program to check your work!

1. All that escaping and concatenation gets kinda tedious, doesn't it? This is a case where a double-quoted string would make things a lot easier. So change the single-quoted string to a double-quoted string and substitute (*interpolate*) the name and health values

**puts “#{name1}’s health is #{health1}”**

1. Run the program to make sure you're getting the same output as before.
2. Now that you're getting the hang of it, give this a try: How would you change the double-quoted string to triple Larry's health?

**puts “#{name1}’s health is #{health1 \* 3}”**

1. OK, now what if you wanted to divide Larry's health by 9 and have the answer printed out with decimals (a float) or with*out* decimals (an integer)?

**puts “#{name1}’s health is #{health1/9.0}” #Float  
puts “#{name1}’s health is #{health1/9}” #Integer**

1. Experimenting with code is a great way to learn, so play around a bit until you're comfortable.

### 2. Print a List of Players

### In addition to being able to substitute the value of any Ruby expression into a double-quoted string, double-quoted strings also support various *escape sequences* that start with a backslash character. For example, you use \n for a newline character and\t for a tab character. When a string is evaluated, Ruby looks for these escape sequences and replaces them with their equivalent character.

1. Print out a list of players in the game formatted like this
2. Players:   
    larry  
    curly  
    moe
3. Notice that each player's name is printed on a separate line and indented with a tab. There are at least two ways to do this. First try it without using any variables.

**puts “Players: \n\tlarry\n\tcurly\n\tmoe”**

1. Then change it to use variables for each player's name.

**name1 = larry  
name2 = curly  
name3 = moe**

**puts “Players: \n\t#{name1}\n\t#{name2}\n\t#{name3}”**

1. Make sure to run your program file to check the formatting.

### 

### Bonus Round

### Start Creating a Third Ruby Program

### For folks who want even more practice writing Ruby, here's an idea you could experiment with alongside the game program: a fundraising program. This program would track various projects (start-up projects, community projects, personal projects, etc.) and how much funding each project has received so far. For inspiration, check out [Kickstarter](http://www.kickstarter.com/), [Pledgie](http://pledgie.com/), [gofundme](http://www.gofundme.com/), or [indiegogo](http://indiegogo.com/).

### At the end of each module, we'll include a list of fundraising program requirements for you to implement. Please consider these requirements as non-exhaustive guidelines for building this program. If you think something else should be added, changed, or removed, go for it! There are no right answers. Be creative, experiment, don't be afraid of error messages, and have fun trying new things!

### To get started, create a new program (call it crowdfund, for example) and print out info about your initial project and a list of all your projects. Something like:

Project ABC has $1000 in funding.  
  
Projects:   
 Project ABC  
 Project LMN  
 Project XYZ

**project1 = 'Project ABC'**

**project2 = 'Project LMN'**

**project3 = 'Project XYZ'**

**funding1 = 1000**

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

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

### Wrap Up

### OK, now we have some players in the game. The game doesn't do much yet, but at least it's a start in the right direction. In the process, you got some practice and learned how to:

* use single and double-quoted strings
* concatenate strings
* mix strings and numbers
* use escape sequences
* evaluate (interpolate) Ruby code within a double-quoted string using #{}

In the next section we'll spiff up the game's intro by creating objects and calling their methods!