**Common Error Types**

As you've been progressing through this course, you've likely run into a few different error types by mistake. Let's take a look at these different errors and understand what caused them. If we are able to identify what exactly is causing an issue, we can act accordingly and ensure bug-free code. In this reading, the errors we refer to are those that prevent our code from completing execution. That is, Ruby will terminate execution of the program immediately when any of these errors are hit.

**Reading an Error Message**

Let's say we had this broken ruby code in a code.rb file:

def say\_hello

puts "hi!"

The code above is obviously incorrect because of a missing end. When we execute this code we'll receive an error message:

$ ruby code.rb

code.rb:2: syntax error, unexpected end-of-input, expecting keyword\_end

puts "hi!"

A ruby error message begins with the file name that we tried to execute (code.rb) as well as a line-number (2) for the error. Given this information, we ought to verify that we ran the correct file (it happens to the best of us :)) and scan line 2 for any obvious errors. It may be the case that the suggested error line is nowhere near the actual cause of the error, especially when we are dealing with a SyntaxError, so don't tunnel vision into checking that single line.

For example, let's say we had a different kind of typo in the code:

def check\_num(num) # 1

if num > 0 # 2

p "positive" # 3

else if num < 0 # 4

p "negative" # 5

else # 6

p "zero" # 7

end # 8

end # 9

The Ruby syntax coloring above makes this mistake obvious, but peek at the error message we would get when executing this code:

$ ruby code.rb

code.rb:9: syntax error, unexpected end-of-input, expecting keyword\_end

Ruby is complaining about line 9, but the real mistake is on line 4. The suggested line number isn't useful here, but the type of error, syntax error, will give us an idea of *what* kind of mistake we are looking for.

Let's break down common error types next. The following list is by no means comprehensive, but you will run into these errors most often.

**SyntaxError**

A SyntaxError results from incorrectly structuring our code. In proper English we must follow certain grammar rules. In the same way proper Ruby must follow particular syntax rules. Because we can violate the "grammar" of Ruby in many different ways, there is no hard and fast rule to spot a SyntaxError. In general, you should be scanning your code for structural mistakes:

# SyntaxError because we need to have a value to assign

my\_variable =

**NameError**

A NameError results from referring to a variable or method name that has not been defined:

my\_variable = 42

p my\_variabel # NameError because this name has not been defined previously

**NoMethodError**

A NoMethodError results from referring to an undefined method. It is considered a special instance of a NameError:

def say\_hello(name)

p "hi " + name

end

hello("world") # NoMethodError because `hello` is not a defined method

**ArgumentError**

A ArgumentError results from incorrectly specifying arguments for a method call. Typically this means passing too little or too many arguments:

def say\_hello(first\_name, last\_name)

p "hi " + first\_name + " " + last\_name

end

say\_hello("world") # ArgumentError because we gave 1 arg, but `say\_hello` expects 2 args

**TypeError**

A TypeError results from performing an operation with incompatible data types:

5 + "spaghetti" # TypeError because we cannot add a Integer and a String together