**Debugging Using PRY**

As you develop and debug your programs, often times you'll find it useful to test ideas in a "sandbox" environment where you can get quick feedback on the state of your code. You'll use a Ruby **REPL** to this end. To review, **REPL** stands for **R**ead, **E**valuate, **P**rint, **L**oop.

At App Academy, the Ruby REPL of choice is [Pry](https://pry.github.io/). Pry is an open source project developed as an alternative to IRB, the standard Ruby REPL. Pry has more features than IRB, although IRB is catching up.

**Installing Pry**

If you haven't installed Pry already, run gem install pry.

**Using Pry**

You can begin a pry session simply by using the command pry in your command line. You'll be greeted by a seemingly unassuming REPL:

$ pry

[1] pry(main)>

Like any REPL, you can type Ruby code line by line to test ideas on the fly. Every line you write will have its evaluation displayed with =>:

[1] pry(main)> greens = ["kale", "spinach", "arugula"]

=> ["kale", "spinach", "arugula"]

[2] pry(main)> greens << "broccoli"

=> ["kale", "spinach", "arugula", "broccoli"]

[3] pry(main)> greens.length

=> 4

You can also copy and paste multiple lines of code into a Pry session.

So far, nothing groundbreaking, but what advanced tools does Pry offer you?

ls

You can use ls to list methods that you can use in a given context. For example, you can see what methods you have available to use on strings by entering ls String in Pry:

[1] pry(main)> ls String

String.methods: try\_convert

String#methods:

% each\_codepoint scan

\* each\_grapheme\_cluster scrub

+ each\_line scrub!

+@ empty? setbyte

-@ encode shellescape

<< encode! shellsplit

<=> encoding size

== end\_with? slice

=== eql? slice!

=~ force\_encoding split

[] freeze squeeze

[]= getbyte squeeze!

ascii\_only? grapheme\_clusters start\_with?

b gsub strip

bytes gsub! strip!

bytesize hash sub

byteslice hex sub!

capitalize include? succ

capitalize! index succ!

casecmp insert sum

casecmp? inspect swapcase

center intern swapcase!

chars length to\_c

chomp lines to\_f

chomp! ljust to\_i

chop lstrip to\_r

chop! lstrip! to\_s

chr match to\_str

clear match? to\_sym

codepoints next tr

concat next! tr!

count oct tr\_s

crypt ord tr\_s!

delete partition undump

delete! prepend unicode\_normalize

delete\_prefix pretty\_print unicode\_normalize!

delete\_prefix! replace unicode\_normalized?

delete\_suffix reverse unpack

delete\_suffix! reverse! unpack1

downcase rindex upcase

downcase! rjust upcase!

dump rpartition upto

each\_byte rstrip valid\_encoding?

each\_char rstrip!

Whoa! There are tons of methods that you haven't even learned. Don't be intimidated; documentation is your friend. From this point you can search the [Ruby Docs](https://ruby-doc.org/) in your web browser to gain more insight into the functionality of a method you are unfamiliar with. You can also look up documentation for a method in Pry using ri. Let's see that next.

ri

You can use [ri](https://ruby.github.io/rdoc/RI_rdoc.html), which is built into Ruby, to show the documentation for a given method. For example, if you want to learn more about the end\_with? method from the example above, you can run ri String#end\_with?. (Be sure to use Class#method for instance methods and Class::method for class methods.)

[1] pry(main)> ri String#end\_with?

String#end\_with?

(from ruby core)

------------------------------------------------------------------------

str.end\_with?([suffixes]+) -> true or false

------------------------------------------------------------------------

Returns true if str ends with one of the suffixes given.

"hello".end\_with?("ello") #=> true

# returns true if one of the +suffixes+ matches.

"hello".end\_with?("heaven", "ello") #=> true

"hello".end\_with?("heaven", "paradise") #=> false

You can also use ri in the command line outside of Pry! You can use the same ri Class#method syntax, or run ri by itself to enter an interactive session with autocomplete. From there, just type Class#method to retrieve documentation for a given method:

~/Desktop$ ri

Enter the method name you want to look up.

You can use tab to autocomplete.

Enter a blank line to exit.

>> String#end\_with?

load

[load](https://ruby-doc.org/core-3.0.2/Kernel.html#method-i-load) is another method built into Ruby that you'll use often in Pry. It allows you to load an entire **.rb** file into Pry and begin playing with it.

Let's say you start your Pry session in **my\_ruby\_code/**, which has the following structure:

my\_ruby\_code/

└── code.rb

└── nested\_folder/

└── other\_code.rb

You can load the files **code.rb** and **other\_code.rb** into Pry with load "path\_to\_file":

~/Desktop/my\_ruby\_code$ ls

code.rb nested\_folder

~/Desktop/my\_ruby\_code$ ls nested\_folder

other\_code.rb

~/Desktop/my\_ruby\_code$ cat code.rb

def is\_prime?(num)

(2...num).each do |factor|

return false if num % factor == 0

end

num > 1

end

~/Desktop/my\_ruby\_code$ cat nested\_folder/other\_code.rb

def every\_other\_case(str)

new\_chars = str.chars.map.with\_index do |char, index|

if index % 2 == 0

char.downcase

else

char.upcase

end

end

new\_chars.join

end

~/Desktop/my\_ruby\_code$ pry

[1] pry(main)> load "code.rb"

=> true

[2] pry(main)> is\_prime?(16)

=> false

[3] pry(main)> is\_prime?(7)

=> true

[4] pry(main)> load "nested\_folder/code.rb"

=> true

[5] pry(main)> every\_other\_case("hello world")

=> "hElLo wOrLd"

show-source

If you want to see the code that implements a method (i.e., the *source code*), you can use the show-source command. Let's say you have already run load "code.rb" from the previous example, and you want to see the definition of is\_prime?. Simply execute show-source is\_prime?:

[1] pry(main)> load "code.rb"

=> true

[2] pry(main)> show-source is\_prime?

From: code.rb:1:

Owner: Object

Visibility: private

Signature: is\_prime?(num)

Number of lines: 7

def is\_prime?(num)

(2...num).each do |factor|

return false if num % factor == 0

end

num > 1

end

**System Commands**

If you ever want to run a system command from within a Pry session, just begin the command with a dot (.). For example, if you started a session in **my\_ruby\_code/** but forgot the names of the files and directories within, you can run .ls:

~/Desktop/my\_ruby\_code$ pry

[1] pry(main)> .ls

code.rb nested\_folder

[2] pry(main)> .cd nested\_folder

[3] pry(main)> .ls

other\_code.rb

**Conclusion**

You'll find the few commands listed above the most useful as you code, so you should focus on learning those for now. However, there is much more you can do with Pry; feel free to read about it in the [Pry GitHub](https://github.com/pry/pry/blob/master/README.md).