RUBY 101

Understanding the basics

MASTERY TOPICS

- *String*
- Numbers (*Integer* / *Float*)
- Scripts
- Variables
- Methods
- Ruby *gets* & *chomp*

STRING

Like Words

Ruby Strings are simply a sequence of characters. If this sentence were to be processed by Ruby, it would be a String.

Bookis — Nerd — ruby

```
> “Hello”
=> “Hello”

You can use + and * on Strings.
> “Kit” + “tens” + “!”
=> “Kittens!”

As well as other Methods
> “i’m mad”.upcase
=> “I’M MAD”
> “Bookis”.reverse
=> “sikooB”
> “I can contain numbers! 10. see!?”.length
=> 30
```

Check out the Documentation for String to see other things that you can do to a String

http://ruby-doc.org/core-2.0.0/String.html

or google `ruby string`

NUMBERS

Integer v. Float

Numbers are pretty obvious, but maybe not as simple as you might think. Numbers can be a few different datatypes, the most common are Integer and Float. The most common difference is the decimal point.

NUMBERS

Integer v. Float

Integers are whole numbers (1, 9999, -255) and are most commonly used.

Floats are numbers with a decimal point (3.14, 0.001) and are most often used for mathematical computations.

Bookis — Nerd — ruby

```
> 10 * 10
=> 100
> 2 / 3
=> 0
> 2.0 / 3.0
=> 0.66666666666
Notice Integers and Floats share many arithmetic methods, but they
won't always produce the same result.

> 2.4 * ((100/99.88) * 4.2**8) - 77
=> 232585.84332597523
Also notice that you can even mix and match Integers and Floats
(be careful, you may not always get the type that you expect back)
```

Check out the Documentation for Integers and Floats.

http://www.ruby-doc.org/core-2.0.0/Integer.html
http://www.ruby-doc.org/core-2.0.0/Float.html
or google `ruby integer` or `ruby float`

SCRIPTS

Ruby scripts are files with the .rb extension. Ruby scripts can be run from the Terminal without entering IRB.

Bookis — Nerd — ruby

```
ada-: subl cookie.rb
ada-: ruby cookie.rb

I want
369729637649726772657187905628805
440595668764281741102430259972423
552570455277523421410650010128232
727940978889548326540119429996769
494359451621570193644014418071060
667659301384999779999159200499899
Cookies

ada-:
```

cookie.rb — luna-sandals

cookie.rb

```
# I'm a script written in ruby.
# Save me in a .rb file
# run me from the Terminal

puts "I want #{99 ** 99} Cookies"
```

Using Terminal scripts are run with the `ruby` command followed by the filename.

Any thing that we did in IRB we could due in a script. The benefit is that we can save our work and if we make a mistake, we can just go edit the document and try again.

VARIABLES

& Assignment

To temporarily save some information like a String, we can assign it to a variable. A variable acts as a name of some stored data. This allows us to use the same data over and over again, and even modify the data.

Bookis — Nerd — ruby

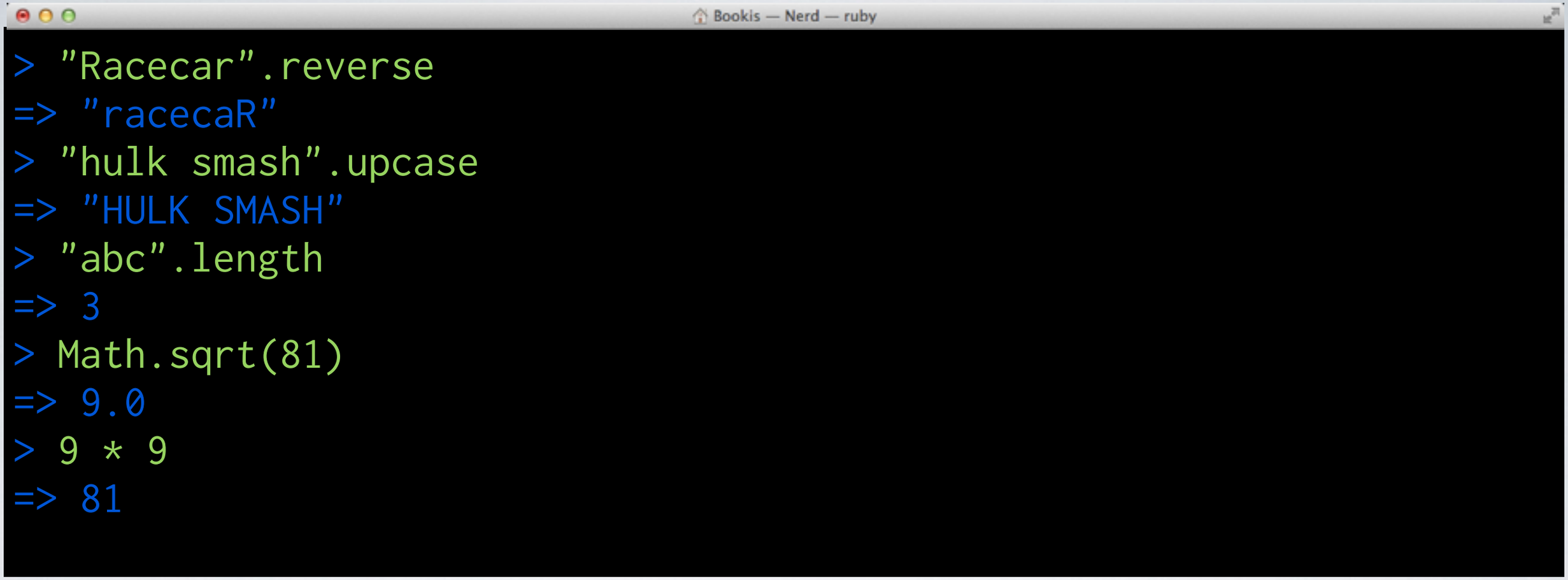
```
> name = "Susannah"
> name
=> "Susannah"
> name
=> "Susannah"
> name = name.reverse
=> "hannasuS"
> name
=> "hannasuS"
> name + " rocks!"
=> "hannasuS rocks!"
> x = 8
=> 8
> y = 10
=> 10
> x + y
=> 18
```

We can see that we can save the string "Susannah" to the variable name then we can continue to use and manipulate the string "Susannah" without having to type it again.

METHODS

Verbs to our Nouns

Think of methods as verbs. They are what objects (String, Integer, etc.) do. A String reverses, the string is the noun, reverse is the verb or method. Generally methods are called using the dot syntax ("Susannah".reverse), but we will learn later that this is not always the case.



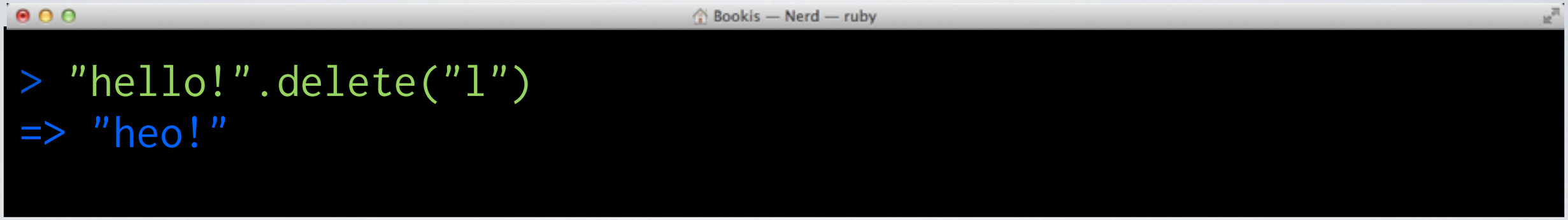
Some methods we've seen so far:

- .reverse
- .upcase
- .length
- .sqrt
- *
- **
- +
- /
- =

These are all actions that we do to some Object. We .reverse a String and we / an Integer or Float. Each type of Object can do different methods. Like real objects.

A NOTE ON ARGUMENTS:

Arguments are the values given in ( ) at the end of a method.



- "hello!" - Object (noun)
- delete - Method (verb)
- "l" - Argument

The delete method expects one argument, what happens if we use delete without passing in a argument?

Now that we know basic object types and methods lets identify a complex chain of methods.

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(make_angry(words).reverse.length * 10.4).to_s + " on the anger scale"

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(**make_angry**(words).reverse.length * 10.4).to_s + " on the anger scale"

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(make_angry(words).**reverse**.length * 10.4).to_s + " on the anger scale"

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(make_angry(words).reverse**.length** * 10.4).to_s + " on the anger scale"

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(make_angry(words).reverse.length * 10.4).to_s + " on the anger scale"

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(make_angry(words).reverse.length * 10.4).**to_s** + " on the anger scale"

Name that ruby:

```
def make_angry(word)
    word.upcase + "!!!!!!!"
end

words = "you wouldn't like me when I'm angry"
```

(make_angry(words).reverse.length * 10.4).to_s **+ " on the anger scale"**

GETS

& Chomp

Gets is a method that is used to get user input. This is typically used when running a script that requires a response from you.

Bookis — Nerd — ruby

```
ada-: subl gets.rb
ada-: ruby gets.rb
Hello there! How are you?
good
Oh good! That's just wonderful!
K Bye!
ada-:
```

cookie.rb — luna-sandals

cookie.rb

```ruby
# I'm a script written in ruby.
# Save me in a .rb file
# run me from the Terminal

puts "Hello there! How are you?"
response = gets.chomp
puts "Oh #{response} that's just
wonderful!"
puts "K Bye!"
```

You can see that we are assigning the gets method to the variable response, this allows us to use whatever you typed in our code.

When you press return↵, the return key gets submitted with your input, this is where chomp comes in. Chomp just cleans up any unintended input from gets.

MASTERY TOPICS

- *String*
- Numbers (*Integer* / *Float*)
- Scripts
- Variables
- Methods
- Ruby *gets* & *chomp*