PIXAR
ANIMATION STUDIOS

JAMF NATION user conference

OCTOBER 18-20, 2016 MINNEAPOLIS, MN

PIXAR
ANIMATION STUDIOS

Alleviate the Apprehension of Coding in Ruby

Chris Lasell

Apple Peeler

Pixar Animation Studios

PIXAR
ANIMATION STUDIOS

Install session materials

Includes some example code & libraries

- ruby-jss and ruby gem dependencies into
  - /Library/Ruby/Gems & /usr/local/bin
- two methods and a class into /Library/Ruby/Site
- an example script and data file into /Users/Shared

`http://bit.ly/2emM1VW`

P I X A R
Animation Studios

Goals

By the end we will:

- Learn some Ruby (& OOP) terminology
- Read (and write) basic Ruby code
- Use ruby-jss to access the REST API
- Find resources for further learning

Your Background

You should be familiar with one or more:

- Python, Perl
- Advanced Bash
- PHP, Javascript
- Swift, ObjC, Go, Lua, COBOL…

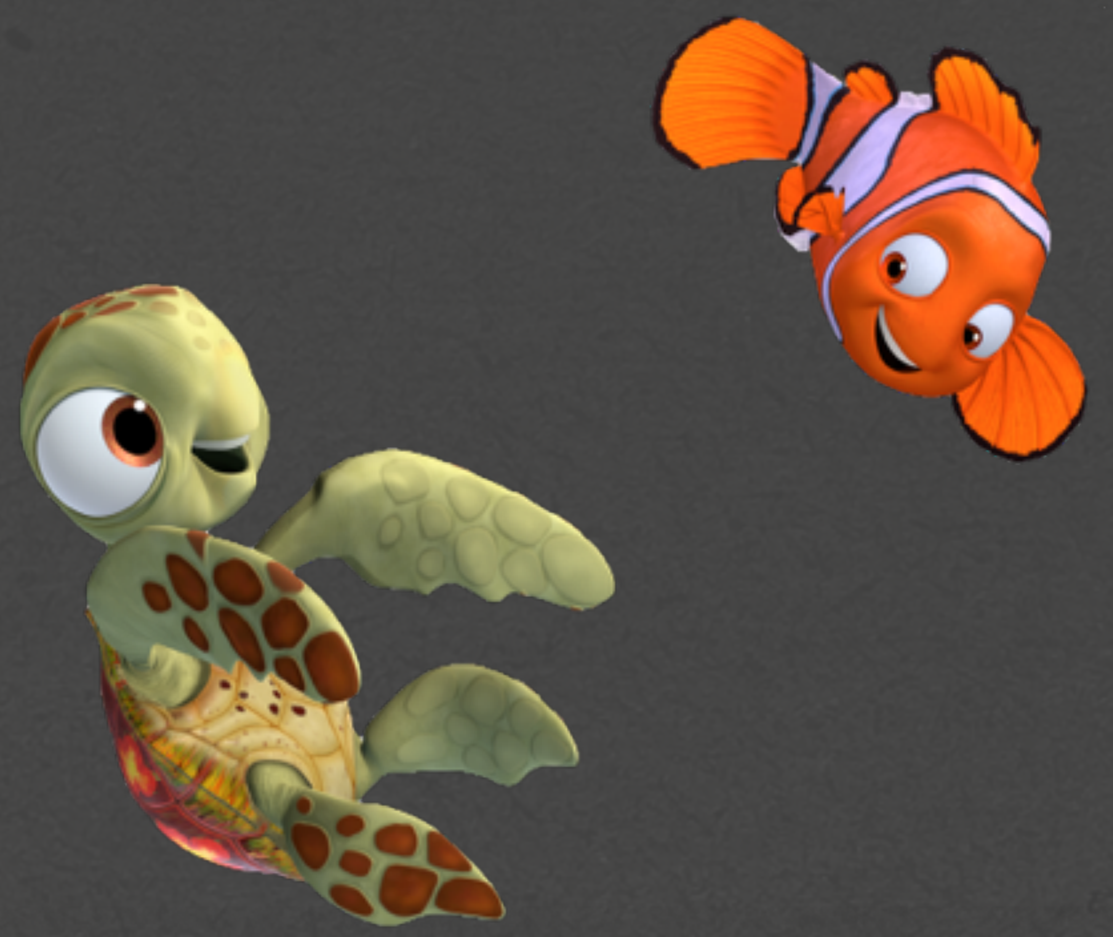
Your Understanding

You should already know about:

- Data Types (Strings, Numbers, Arrays, …)
- Variables, Constants & Functions
- Conditionals (if, unless)
- Loops (for, for each, repeat)

PIXAR
ANIMATION STUDIOS

The Ruby Philosophy



Programming in Ruby should be

- Adaptable & Contextual
- Expressive
- Fun!
- Unsurprising (once you know it)
- Computers are the slaves!

irb - Interactive Ruby

```
% irb
> puts 'hello world'
hello world
=> nil
> require 'jnuc-intro-ruby'
=> true


> welcome
Welcome to JNUC 2016
=> nil
```

Fire up a terminal and type 'irb'

irb is a Ruby 'shell' where you type and execute Ruby code in real-time

It's useful for testing your code as you write or for doing one-off tasks

'require' tells ruby to read and execute some pre-written code from disk

PIXAR
ANIMATION STUDIOS

Connect to the JSS REST API

JSS is a module that we load by requiring ruby-jss

```
> require 'ruby-jss'
=> true
> JSS::API.connect user: 'jssadmin', pw: :prompt, server: '10.0.1.2'
Enter the password for JSS user jssadmin@10.0.1.2: jnuc2016
=> "10.0.1.2"
```

PIXAR
ANIMATION STUDIOS

Every Thing is an Object

```
> 'hello world'
=> "hello world"
> 15
=> 15

> 'hello world'.class
=> String
> 15.class
=> Fixnum
> JSS::API.class
=> JSS::APIConnection
```

Objects = nouns
'things' stored in memory

There are many kinds
of objects, like…

Strings &

Integers

'kind of object' = 'Class'

P I X A R
Animation Studios

Every Action is a Method

Methods = verbs, "functions"
They make objects do things like...
retrieve attributes &
perform actions

```
> 'hello world'.length
=> 11
> 'hello world'.capitalize
=> "Hello World"
```

Methods always 'return' a value

```
> 15.capitalize
=> NoMethodError: undefined method `capitalize' for 15:Fixnum
```

Different classes have different methods

PIXAR
Animation Studios

Variables & Constants

Objects can be stored in Constants & Variables

```
> JNUC_CITY = 'Minneapolis'
=> "Minneapolis"

> my_string = 'this is my string'
=> "this is my string"

> JSS::API
=> #<JSS::APIConnection:0x007fde...
> JSS::Client::JAMF_SUPPORT_FOLDER
=> #<Pathname:/Library/Application Support/JAMF>
```

Constants start with a capital letter

Variables start with lower-case, possibly @

Modules and Classes often contain Constants

Classes & Instances

```
> my_string.class
=> String
```

A class defines a kind of object using constants, variables, and methods

```
> my_string
=> "this is my string"
```

An individual object is an 'instance' of its class

```
> Dog.class
=> Class
```

You can define your own classes

Class Methods

Class methods are used when
there is no context for an 'instance'
but the method is related to
the class as a whole

```
> Dog.top_ten_names
=> ["Bella", "Max", "Molly", "Buddy"...
> JSS::Computer.all_serial_numbers
=> ["5DBB27D6EB01", "5DBB27D6EB02"...
```

In documentation, they are sometimes
marked by a leading :: such as Dog::top_ten_names

Creating Instances

The usual way is with the 'new' class method

```
> my_dog = Dog.new(name: 'Colby')
=> #<Dog:0x007f9ba9c23930 @name="Colby"...
```

Some classes have shortcuts,
e.g. Strings use quotes

```
> my_string = "this is my string"
=> "this is my string"
```

This retrieves an instance of
JSS::Computer from the API

```
> a_comp = JSS::Computer.new(id: 72)
=> #<JSS::Computer:0x007fde250143b0 ...
```

P I X A R
Animation Studios

Instance Methods

```
> my_string.length
=> 17
> my_dog.bark
=> nil
> a_comp.department
=> "Sales"
> a_comp.managed?
=> true
```

Instance methods work on instances of a class

In documentation, they are often marked by a leading #

'String#length' or just '#bark' if we know the context

Quoting Strings

```
> my_string = 'this is my string'
=> "this is my string"

> A_CONSTANT = "This string shouldn't change"
=> "This string shouldn't change"
```

Single vs double quotes, generally work as in bash

```
> multiline = <<ENDQUOTE
  This is a
  multiline string
  ENDQUOTE
=> "This is a\nmultiline string"
```

As do HereDocs

String Interpolation

As with bash, double-quoted strings can embed values

Use #{}

```
> "the value of 'my_string' is: '#{my_string}'"
=> "the value of'my_string' is: 'this is my string'"
```

Any expression works

```
> "the value of 3 plus 4 is: #{3 + 4}"
=> "the value of 3 plus 4 is: 7"
> "the length of 'my_string' is: '#{my_string.length}'"
=> "the length of 'my_string' is: 17"
```

A common rookie error: interpolation with nothing

```
> "#{my_string}"
=> "this is my string"
> my_string
=> "this is my string"
```

PIXAR
ANIMATION STUDIOS

Symbols

Symbols: 'lightweight' Strings

```
> str1 = 'ruby'
=> "ruby"
> str2 = 'ruby'
=> "ruby"
> str1.object_id == str2.object_id
=> false
```

Two identical Strings are still different objects in memory

```
> sym1 = :ruby
=> :ruby
> sym2 = :ruby
=> :ruby
> sym1.object_id == sym2.object_id
=> true
```

Two identical Symbols are the same object

Usually used as labels

Arrays

Known as "lists" in some languages

```
> my_array = [ "hello world", 12, 3.1416, :foobar ]
=> ["hello world", 12, 3.1416, :foobar]

> my_array[0]
=> "hello world"

> my_array[2] = Math::PI
=> 3.141592653589793
> my_array
=> ["hello world", 12, 3.141592653589793, :foobar]
```

Ordered collections of objects indexed by zero-based numeric position

Arrays from the JSS

Lots of things in the JSS module return Arrays

```
> a_comp.computer_groups
=> ["All Managed Clients"]
```

```
> JSS::User.all_names
=> ["ijames", "bsingleton", "cschmidt"...
```

Many of which are full of Hashes

```
> JSS::MobileDevice.all
=> [{:id=>1, :name=>"Ismael's iPhone", :device_name=>"Ismael's iPhone"
```

PIXAR ANIMATION STUDIOS

Hashes

```
> my_hash = { :height =>  18,
     :width => 24,
     :depth => 15,
     :name => "my toy box",
     :unit => :inches }
=> {:height =>  18, :width => 24, :depth => 15, :name =>
    "my toy box", :unit => :inches}
> my_hash[:name]
=> "my toy box"
> my_hash["color"] = :blue
=> :blue
```

a.k.a. dictionaries, records, objects, associative arrays

Collections of key-value pairs

Indexed by their keys

Keys are often Symbols

Sometimes Strings, but can be any object

Modern Hashes

```
> my_hash = { :height =>  18,
    :width => 24,
    :depth => 15,
    :name => "my toy box",
    :unit => :inches }
=> {:height =>  18, :width => 24...
```

Normally, Hash items are defined with 'key => value'

```
> my_hash = { height: 18,
    width: 24,
    depth: 15,
    name: "my toy box",
    unit: :inches }
=> {:height =>  18, :width => 24...
```

But if keys are symbols, the new way is simpler

Looping over Arrays & Hashes

Ruby has 'for' loops,
but no one uses them

Instead, use methods called 'iterators'

```
> JSS::MobileDevice.all.each do |dev|
    puts "#{dev[:name]} is an #{dev[:model_display]}"
  end
  Ismael's iPhone is an iPhone 4S
  Bernard's iPad is an iPad 3rd Generation (Wi-Fi)
  Christina's iPhone is an iPhone 5S (GSM)
  [...]
=> [{:id=>1, :name=>"Ismael's iPhone"...
```

PIXAR
ANIMATION STUDIOS

Iterators & Code Blocks

```
> foo_string = 'foo'
=> "foo"
> 5.times { foo_string << 'bar' }
=> 5
> foo_string
=> "foobarbarbarbarbar"
> 5.upto 7 do |val|
    foo_string << val.to_s
  end
=> 5
> foo_string
=> "foobarbarbarbarbar567"
```

Iterators 'iterate' over collections

Code blocks are inside {...} or do...end

To pass values to the block, use |...|

Other kinds of methods can use code blocks too

The #each Iterator

```
> JSS::MobileDevice.all.each do |dev|
    puts "#{dev[:name]} is an #{dev[:model_display]}"
  end
 Ismael's iPhone is an iPhone 4S
 Bernard's iPad is an iPad 3rd Generation (Wi-Fi)
 Christina's iPhone is an iPhone 5S (GSM)
 [...]
=> [{:id=>1, :name=>"Ismael's iPhone"...
```

#each loops thru, handing each item to the block for processing

When finished, it returns the original Array or Hash

PIXAR
ANIMATION STUDIOS

The Array#map Iterator

```
> arr = [1, 2, 3]
=> [1, 2, 3]

> [1, 2, 3].map { |n| n * 2 }
=> [2, 4, 6 ]
```

#map returns a new Array, where each item is the result of executing the code block on the matching item in the original Array

```
> JSS::Category.all.map { |cat| cat[:name] }
=>  ["Graphics", "Music", "Operating System", "Text Editors"]
```

#select & #reject

```
> arr.select { |n| n.odd? }
=> [1, 3]
> arr.reject { |n| n == 3 }
=> [1, 2]
```

#select returns a new Array with only the items for which the code block was true

#reject does the opposite

Iterators can be chained like any other method

```
> JSS::Computer.all.select{|comp| comp[:managed]}.map{|c| c[:name]}
=> ["Ismael's MacBook Air", "Bernard's MacBook Pro"...
```

Nil

```
> empty_array = []
=> []
> empty_array[0]
=> nil
> empty_array[14].nil?
=> true
> my_dog.bark
=> nil
```

Nil is a non-object,
it's the lack of a value

Nil is the 'default' value for
Arrays and Hashes

Nil is often returned by methods
that don’t have a meaningful
return value

PIXAR
ANIMATION STUDIOS

Conditionals

```
>jss_size = if JSS::Computer.all.count > 4000
               :large
             else
               :small
             end
=> :small
```

"if" and "unless" evaluate the truth of an expression, and are expressions themselves

Conditionals can be 'modifiers'

```
> my_dog.speak if jss_size == :small
=> nil
```

Note: everything is true except false and nil

P I X A R
Animation Studios

Method Parameters

```
> 'hello world'.length
=> 11
> 'hello world'.delete 'lo'
=> "he wrd"
> 'hello world'.index 'l', 4
=> 9
> 'hello world'.index('l')
=> 2
> say_hello to: 'Chris', from: 'Alex'
=> "Alex says hello to Chris"
```

No parameters

Required parameters

Positional parameters

Optional parameters & parens

Named parameters

Docs are your friend

PIXAR
ANIMATION STUDIOS

Putting It All Together

In a text editor, open up the ruby script "group-sync" from /Users/Shared/intro-ruby

This script will synchronize a JSS static computer group with the contents of a file full of computer names

It does this, in an object-oriented way, in 40ish lines of (heavily commented) code

Lets have a look at through it.

Ruby Beyond the Code

Before we're done, a brief look at:

- Built-in Libraries
- Gems
- Resources

Built-in Libraries

- Core Library
  - Array, String, Symbol, Hash, Fixnum, Float, File, Dir…
    - No need to 'require'
- Standard Library
  - Pathname, Date, FileUtils, JSON, YAML, WebRick…
    - Must 'require'

Gems

- Ruby packages are called gems
  - Almost all 3rd party libraries are distributed as gems
  - Install and manage them with the 'gem' command
  - Thousands are out there
  - Most are at www.rubygems.org

Docs & Resources

There's tons out on the web, here's some:

http://ruby-doc.org/

 Core and Standard Library

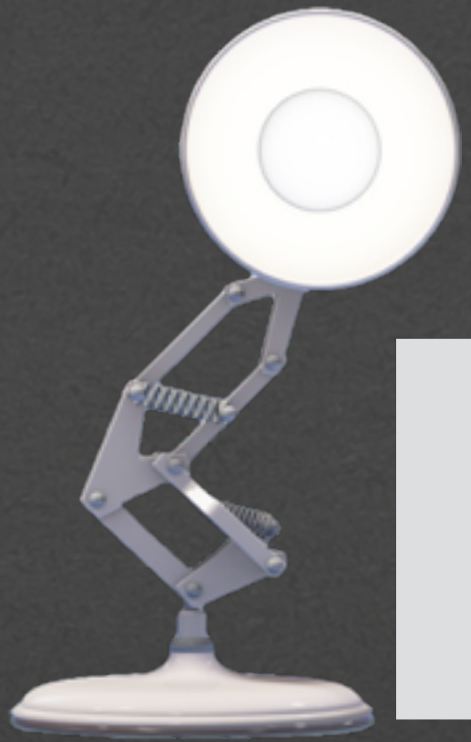
http://www.rubydoc.info/

 Auto-generated for all RubyGems & GitHub & StdLib

#ruby in Macadmins Slack

 @glenfarclas17

PIXAR
ANIMATION STUDIOS

Thnk you!