Ruby From Scratch

A course for Rails Girls Wellington, March 2014.

Blurb:

> The goal of this course is for you to learn everything you need to know to write a simple Ruby program from scratch, and understand every line so you have the knowledge to go write similar (and bigger!) programs of your own.
>
> This course assumes no knowledge of programming, although if you've learnt by following tutorials on the 'net up until now, you might want to join us to learn the fundamentals of why things work the way they do.
>
> There's quite a bit of theory and terminology to cover, but we'll mix it up with lots of discussion and writing several text-based games as we go along. Our final project is the game "Hangman" which you'll write from scratch and know exactly how it works.

Required to teach the course:

- whiteboard
- computer with screen or projector so everyone can see
- Internet access for participants to download wordlist **or** wordlist on USB stick

The following is a rough proposed outline of the course. Please right-click/Comment to discuss a particular piece of the course, or add directly to the document if you'd like to flesh it out.

Objects, classes and methods in the real world

- Real world objects and classes
  - Brief explanation
  - Group identification: give me examples of objects
    - I imagine most people will say “a dog” is an object. It’s not; it’s a class. “My dog” is an object, “the pencil on my desk” is an object. See how long it takes for the group to work out the distinction!

- Methods
  - Questions and actions
    - Get the group to come up with some “question methods” and “action methods” for classes previously identified
  - Arguments
    - Argument type
      - e.g. for a TV, set the volume to 30 vs set the volume to chicken.
  - Return values
    - Return value type

- Robot game redux with classes, objects, methods and arguments

Ruby objects

- Numeric and String

- Entering stuff into irb
  - Using the up arrow
  - Encourage group to try stuff out as we're working through

- String methods: #reverse, #upcase, #length, #empty?, #index, #count
  - Tell group they will get a cheat sheet with these methods later and that docs are available online by googling "ruby" and the name of the class.
  - Ask for guesses as to what the method will return
  - All throughout the course, ask group to consider what type the arguments are and what type the return value is!

- false, true, and nil
  - The three special variables in Ruby
  - They are objects because everything is an object: e.g. true.class

- Chaining methods
  - e.g. "hello".upcase.reverse

- Everything is an object
  - Every statement has a return value, these are objects
  - Methods are called on objects and classes

- Numeric methods
  - Numbers are objects too and have methods: Fixnum#next, Float#round, #abs

- Sneaky number methods: #+ and friends
  - Demonstrate 3 + 2 == 3.+(2)

- Playing with Strings and Numerics: #+, #*, #next
  - What happens if you add/multiply a string and a number?
  - Exercises
    - How many "f"s are there in "finished files are the result of years of scientific study combined with the experience of years."
    - What's "z".next ?
    - (need more basic interesting stuff to do)

Variables

Variables are names for objects/classes

- Real world examples, e.g.
    - Roger = Mogest = me = [the object]
    - Powershop Offices = 43 Hanson St, Welly = [this building]
- At some stage you might have to explain that a class is an object of class Class… good luck with that!

The assignment operator

- Giving a name to object on the right of the =
- Methods return objects so if you call methods you can use this to name the results
- Example:
    - roger = “Roger”
    - reversed_roger = “Roger”.reverse

Using variables

- Example
    - name = “Roger”
    - “Hello “ + name + “, nice to meet you.”
    - “Hello “ + “Roger” + “, nice to meet you.”
    - name = “Madeleine”
    - “Hello “ + name + “, nice to meet you.”
    - “Hello “ + name.reverse + “, nice to meet you.”

Variables are not objects, they are “names of” or “references to” objects

- Explain computer memory roughly (visualise as bits of tape?)
- Explain roughly how objects and variables are represented

Mutating methods

- *Just spend 2 minutes on this; if the group doesn’t grok it, move on*
- String#concat
- Explain: greeting = “hello”; salutation = greeting; greeting + “ there”
- Explain: greeting = “hello”; salutation = greeting; greeting = greeting + “ there”
- Explain: greeting = “hello”; salutation = greeting; greeting.concat(“ there”)
- How do you know which methods mutate? (Rote learning :(.)

Play with assigning and using variables

Writing a Program

Entering stuff into a text editor and running it

Kernel.puts and Kernel.print
- Yeah, you can use puts, but reinforce calling methods vs using keywords like ‘if’

Kernel.gets

Comments and when you use them (explaining why, not what)

Putting everything together by writing mini programs
- “Hello, world”
- “What’s your name?”, “Hello, [name]”
    - Avoid string interpolation for now!
- Ask for age and display in seconds
    - String#to_i
    - “This is the only time we’re going to do maths this whole course, promise”

Flow control

Equality method ==
- Try comparing a few things with == to see what you get back
- Get group to speak out difference between “a = 1” and “a == 1” in English
- Esp compare 1 == “1”, 1 == 1.0

The if keyword
- Keywords are “special words” and there aren’t very many of them (36)
- Show how irb waits for more input until you type ‘end’
- Statements always return something, show how it returns the last thing in the if block, or nil if the ‘if’ was not matched
- Code inside the ‘if’ block is indented to visually differentiate it. It’s good coding practice but not required.
- Cover “else” if you have time/feel the group will understand it quickly

Practical
- Extend program written above to respond differently if name is a special string
- Make comment on person’s age
    - Introduce < and > operators

Still running methods on objects!
- 3.==(4), 3.<(4)

Number guessing game

Random.rand

    Demonstrate nothing and number as argument
    Show how Random.rand(10) + 1 gives us a number from 1 to 10

Fixnum#times

    Don't need to explain it at this point, just show it

Group activity: write a number guessing game that gives you 7 guesses to guess a number between 1 and 100, and tells you if you're low or high

    Write requirements on whiteboard
    Start by talking through structure of program, using comments to map out
    1-2 minute personal attempt to write it
    Group attempt at writing it

Maybe a break for stretches about here would be good.

Arrays

Constructing an array

    You can have any object as an *element*
    When would you use an array?

Methods

    #length, #at == #[], #+, #-, #push, #delete, #reverse, #member?, #index, #sample, #sort
    Show that [1,2,3][1] == [1,2,3].[](1) – keep reinforcing that every question/action on an object is done using a method
    Note again that some methods are destructive, some are idempotent

Arrays to Strings and back again

    String#chars, Array#join, String#split

Hand out cheat sheet at this point.

Playing with arrays

    Ask the participants to follow out your verbal instructions, manipulating an array you give them. (A list of colours maybe?) Hopefully they all end up with the same result!

Iterators and Blocks

Revisit Fixnum#times

Array#each

Show how blocks take arguments. Explain that #each “calls” the block, once per element, and sets the argument variable each time it calls it.
Note we haven’t even introduced defining methods yet, so this can be a difficult concept to grasp. Ask the group to write a few one-liners using Array#each.

Array#map, Array#select, Array#detect

Give examples of when you’d use each, e.g.
Array#each: printing each element to the screen
Array#map: converting all elements to uppercase
Array#select: finding words that have 5 letters
Array#detect: finding the first word that has a specified letter in it

Kernel.loop

Ctrl-C to manually break out of loop

break keyword

Show an example with Kernel.loop, Kernel.gets and break

Files

Get users to download list from http://seriousorange.com/words.txt or USB drive

File.read
Find most common word
Find first word sorted alphabetically
Find out how many words begin with “h”
Find words with 2 “z”s
Find single-word anagrams (if you have time)
  Challenge group to figure out how you could do that
  (It’s hard! .chars.sort)

Recap

What is an object, class, variable, method, argument, return value, if and break keywords, booleans, …

Go through basic methods again on String and Array
  “How do you X?”

Hangman

- Sample code: http://pastebin.com/J2Cr72iq

- Work through logic of program using comments
  - Get group to talk about the broad flow of entire app
  - Work through logic of each section using comments

- Personal attempt followed by group attempt at writing it

- Extension activities
  - Error checking on the user’s input
    - Only one character, between A and Z (use range?)
  - Restrict word length to 6-12 characters
    - Use Array#select on word list
  - Add a timer to the game
    - Play with Time.now and subtracting Time objects in the console
  - Draw ASCII art hangman
    - Might be a good time to introduce case...when...end

~ end of course ~

Basic things not covered (in order of what should come next):
- Writing your own methods (!)
- Hashes
- String interpolation
- Making your own simple class
- Symbols

Other project ideas:
- The “Memory” game (could implement with current knowledge)
- Library borrowing tracking system (needs hashes)
- Tuck shop point of sales system (needs basic classes to be neat)