CLASS 3: MORE RUBY CORE

AGENDA

- More Ruby Core Classes
  - Hash
  - Symbol
- Ruby syntax for making decisions
- Ruby syntax for loops

WHY NOT JUST STICK WITH ARRAYS?

- Arrays are good for storing collections of objects.
- Arrays are not good for continual searching for specific objects
- Array#include? works by looking through the whole array until it finds the object or reaches the end of the array.
  - The time to find an object is roughly proportional to the length of an array.
  - Worst case: the desired object is at the end of the array or not in the array at all.
- Big-O notation for a linear search is O(n) where n is the size of the array

RUBY SOURCE CODE FOR INCLUDE?

```
VALUE
rb_ary_includes(VALUE ary, VALUE item)
{
    long i;

    for (i=0; i<RARRAY_LEN(ary); i++) {
        if (rb_equal(RARRAY_PTR(ary)[i], item)) {
            return Qtrue;
        }
    }
    return Qfalse;
}
```

ENTER THE HASH

- A hash is an associative array or dictionary – a collection of keys and values. The keys can be any object – not just integers as in the case of arrays.
- To find an object, all we need to know is the key of the object.
  - The key goes directly to the value without having to search through the whole hash.
  - The size of the hash has no bearing on how fast we can find the value.
  - Big-O notation is O(1) or constant time.

HASH EXAMPLE

```
books = {}
books[:matz] = "The Ruby Language"
books[:black] = "The Well-Grounded Rubyist"
...
```

- **To find a book in the list, all we need is:**
  - **puts books[:matz]**

SIDE-TRIP: SYMBOLS

- A Ruby Symbol is like a string without quotes.
- Symbols are immutable - they can't change
  - "hello" is a string
    - a string has a location in memory
    - different "hello" strings have different locations in memory/
  - :hello is a symbol
    - a symbol has a single location in memory
    - repeated use of the symbol :hello uses the same object from the same location in memory.

SIDE-TRIP: SYMBOLS (CONT.)

From an irb session:

```
irb(main):007:0> puts "hello".object_id
70163697767580
=> nil
irb(main):008:0> puts "hello".object_id
70163697750860
=> nil
irb(main):009:0> puts "hello".object_id
70163697732160
=> nil

irb(main):010:0> puts :hello.object_id
524008
=> nil
irb(main):011:0> puts :hello.object_id
524008
=> nil
irb(main):013:0> puts :hello.object_id
524008
```

Symbols make excellent keys for hashes!

RUBY HASHES

- Use http://www.ruby-doc.org/core-2.1.2/Hash.html and irb to answer the following questions.
  - Create a new hash (3 ways)
    - h = Hash.new
    - h = {}
    - h = {:a => 3, :b => 7}
  - Create a new hash with a default value of 0.
  - Insert a key-value pair into an empty hash.

RUBY HASHES, CONT.

- Remove all key-value pairs from a hash.
- What is the difference between:
  - each
  - each_pair
  - each_key
  - each_value

MORE RUBY HASHES

- Given this hash: h = {:a => 100, :b => 200}, what does the hash look like after each of these statements:
  - h.store(:d, 42)
  - h[:d] = 420

- Each key must be unique!

DEFAULT VALUE EXAMPLE

- Hashes can have a default value

```
>> h = {'02138' => 'Harvard', '02139' => 'Central Square', '02141' => 'East
Cambridge'}
{
    "02138" => "Harvard",
    "02139" => "Central Square",
    "02141" => "East Cambridge"
}
>> h.default = 'No Idea'
"No Idea"
>> h['02138']
"Harvard"
>> h['02142']
"No Idea”
```

HASH EXERCISE

- ISO Country Codes can be found here: http://en.wikipedia.org/wiki/ISO_3166-1
- Create a hash consisting of all countries whose ISO codes are the keys (use symbols) and whose values are the corresponding English short names.
  - Just do those countries whose names start with the letter of your first name.
- Experiment with some of the hash methods found in the documentation at http://www.ruby-doc.org/core-2.1.2/Hash.html
  - #delete
  - #each
  - #invert
  - #length

DECISION STRUCTURES IN RUBY

tutorialspoint.com contains examples of Ruby decision structures.

Ruby *if...else* Statement:

Syntax:

```
if conditional [then]
           code...
[elsif conditional [then]
           code...]...
[else
           code...]
end
```

Example:

```
#!/usr/bin/ruby

x=1
if x > 2
   puts "x is greater than 2"
elsif x <= 2 and x!=0
   puts "x is 1"
else
   puts "I can't guess the number"
end
```

```
x is 1
```

CASE STATEMENT

Ruby *case* Statement

Syntax:

```
case expression
[when expression [, expression ...] [then]
   code ]...
[else
   code ]
end
```

```
age = 5
…
case age
when 0 .. 2
      puts “baby”
when 3 .. 6
      puts “little child”
when 7 .. 12
      puts “child”
when 13 .. 18
      puts “teen-ager”
else
      puts “adult
end
```

RUBY *IF* MODIFIER - “POSTSCRIPT”

Ruby *if* modifier:

Syntax:

```
code if condition
```

Executes *code* if the *conditional* is true.

```
a=[1,2,3,4]
...
puts “in the array" if a.include?(3)
```

RUBY *UNLESS*

Ruby *unless* modifier:

Syntax:

```
code unless conditional
```

Executes *code* if *conditional* is false.

a=[1,2,3,4]

...

```
puts “not in the array " unless a.include?(3)
```

LOOPS IN RUBY

- Most Ruby classes have their own methods for looping - each, each_pair, etc.
    - These methods are the preferred way to enumerate elements of a Ruby class
- tutorialspoint.com contains examples of all the possible kinds of loops.

ONE LOOP EXAMPLE: THE WHILE LOOP

```
i = 0
num = 5

while i < num do
      puts “Inside the loop, i = #{i}”
      i += 1
end
```

Result:

```
inside the loop, i=0
inside the loop, i=1
inside the loop, i=2
inside the loop, i=3
inside the loop, i=4
```

EXAMPLE PROBLEM USING IF AND WHILE

```
# Find the sum of all even-valued terms in a Fibonacci sequence, given the
upper limit.
#Test point:  The sum of all even terms (1,2,3,5,8,13,21,34,55,89) below 100 is
44

puts "Enter the upper limit: "
limit = gets.chomp.to_i

n1 = 1
n2 = 2
total = 2

while (next_term = n1 + n2) < limit
  total += next_term if next_term.even?
  n1 = n2
  n2 = next_term
end

puts "The sum of the even fibonacci terms less than #{limit} is #{total}."
```

SUMMARY

- Ruby hashes are:
  - similar to arrays, but with a non-integer key
  - useful for creating relationships between data values
- Useful methods from the Enumerable module include sort, map, inject and many others.
  - methods ending with a bang (!) change the data in place.
  - The inject method is useful for changing an object into a different object or value
- Ruby contains structures for loops and decisions.

NEXT WEEK

- Objects and Object Oriented Programming
- Messages and methods
- Create our own class using test driven development

HOMEWORK

Given a String *a* that is a sequence of characters, e.g., "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789" (which you can generate with: a = (('a'..'z').collect.to_a + ('A'..'Z').collect.to_a + ('0'..'9').collect.to_a).join, provide a one-liner that will create a password of a random sequence of characters, selected from *a*, of a given length *n*.

Example:

Given parameters *a* = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789", n = 10

One possible result: "j4mUndZuFO”

Another possible result: “wyANkR7W5u”

A third possible result: “qZz4lTJu6w”

Hint: write a Ruby program first, then see if you can condense it into one line of Ruby code that will execute in irb.

APPENDIX

Juicy Ruby Tidbits

SUDOKU.RB

So you can see a complete Ruby program that does something useful, download sudoku.rb from:

https://github.com/lgrains/Girl_Develop_It_Ruby_Intro

Launch *irb* from the same directory where you have saved sudoku.rb. Then run the program from inside *irb* as follows:

```
require './sudoku'
puts Sudoku.solve(Sudoku::Puzzle.new('.
578..3.14.9..1.5.2....3..4826..5.4.5.....2.1.9..5361..2....8.7.4..6.33.8..927.'))
```

Enter any Sudoku puzzle as input, using a dot (.) for empty squares.