Hashes

Hashes

- Indexed collections of object references
- Created with either `{}` or `Hash.new`
- A.k.a. associative arrays
- Index(key) - any object not just an int as in the case of arrays
- Accessed using the `[]` operator
- Values set using either `=>` (creation) or `[]` (post creation) operators (methods)

Hashes

```
editor_props = { "font" => "Arial", "size" => 12, "color" => "red"}
# THE ABOVE IS NOT A BLOCK - IT'S A HASH
puts editor_props.length # => 3
puts editor_props["font"] # => Arial

editor_props["background"] = "Blue"
editor_props.each_pair do |key, value|
  puts "Key: #{key} value: #{value}"
end
# => Key: font value: Arial
# => Key: size value: 12
# => Key: color value: red
# => Key: background value: Blue
```

Hashes

- What if you try to access a value in the `Hash` for which an entry that does not exist?
  - `nil` is returned
- If a Hash is created with `Hash.new(0)`
  - 0 is returned instead
- Hashes API is also very important to master!

Hash created with Hash.new(0)

```
word_frequency = Hash.new(0)

sentence = "Chicka chicka boom boom"
sentence.split.each do |word|
  word_frequency[word.downcase] += 1
end

p word_frequency # => {"chicka" => 2, "boom" => 2}
```

Hashes

- As of Ruby 1.9
  - The order of putting things into `Hash` maintained
    - Similar to `LinkedHashMap` implementation in Java
  - If using symbols as keys – can use `symbol:` syntax
- If a `Hash` is the last argument to a method – the `{}` are optional
  - Last argument not including a block

Hashes

```
family_tree_b419 = {:oldest => "Jim", :older => "Joe"}
family_tree_b419[:younger] = "Jack"
p family_tree_b419 # => {:oldest=>"Jim", :older=>"Joe", :younger=>"Jack"}
family_tree_19 = {oldest: "Jim", older: "Joe", younger: "Jack"}
family_tree_19[:youngest] = "Jeremy"
p family_tree_19
# => {:oldest=>"Jim", :older=>"Joe", :younger=>"Jack", :youngest => "Jeremy"}

# Can a method have named parameters? Something like that
def adjust_colors (props = {foreground: "red", background: "white"})
  puts "Foreground: #{props[:foreground]}" if props[:foreground]
  puts "Background: #{props[:background]}" if props[:background]
end
adjust_colors # => foreground: red # => background: white
adjust_colors ({ :foreground => "green" }) # => foreground: green
adjust_colors background: "yella" # => background: yella
adjust_colors :background => "magenta" # => background: magenta
```

Hash and Block confusion

```ruby
# Let's say you have a Hash
a_hash = { :one => "one" }

# Then, you output it
puts a_hash # => {:one=>"one"}

# If you try to do it in one step - you get a SyntaxError
# puts { :one => "one" }

# Ruby gets confused and thinks {} is a block

# To get around this - you can use parens
puts ({ :one => "one" }) # => {:one=>"one"}

# Or drop the {} altogether...
puts one: "one"# => {:one=>"one"}
```

Time

```
today = Time.now
puts today # => 2011-07-21 00:34:19 -0400
p today.methods.grep /mon/ # => [:mon, :month, :monday?]
puts today.monday? # => false
puts today.month # => 7

one_day = 60 * 60 * 24
yesterday = today - one_day
puts yesterday # => 2011-07-20 00:34:19 -0400
tomorrow = today + one_day
puts tomorrow # => 2011-07-20 00:34:19 -0400
puts today.between?(yesterday, tomorrow) # => true
```

require_relative

- Usually, you will have multiple files you want to run – not just one
- To load one file from another – specify `require_relative` and provide a relative path to the file you want to load

require_relative

```
# Function defined inside file1.rb in some directory x
def hello_from_file1
  "hello from file1.rb"
end

# file2.rb in some directory x
require_relative "file1.rb"
puts hello_from_file1 # => hello from file1.rb
```

References

- “Eloquent Ruby”
- “Beginning Ruby: From Novice To Professional”
- “PickAxe”
- “Metaprogramming Ruby”