Hashes are essentially a generalization of arrays: you can think of hashes as basically like arrays, but not limited to integer indices. (In fact, some languages, especially Perl, sometimes call hashes*associative arrays* for this reason.) Instead, hash indices, or *keys*, can be almost any object. For example, we can use strings as keys:

Hashes are indicated with curly braces containing key-value pairs; a pair of braces with no key-value pairs—i.e., **{}**—is an empty hash.

**It’s important to note that the curly braces for hashes have nothing to do with the curly braces for blocks.** (Yes, this can be confusing.)

Though hashes resemble arrays, one important difference is that hashes don’t generally guarantee keeping their elements in a particular order. If order matters, use an array.  
(Ruby 1.9 actually guarantees that hashes keep their elements in the same order entered, but it would be unwise ever to count on a particular ordering.)

Nested hashes:

In terms of symbols as hash keys, we can define a **user** hash as follows:

**>>** user = { :name => "Michael Hartl", :email => "michael@example.com" }

=> {:name=>"Michael Hartl", :email=>"michael@example.com"}

**>>** user[:name] *# Access the value corresponding to :name.*

=> "Michael Hartl"

**>>** user[:password] *# Access the value of an undefined key.*

=> nil

NESTED HASHES

Hash values can be virtually anything, even other hashes

**>>** params = {} *# Define a hash called 'params' (short for 'parameters').*

=> {}

**>>** params[:user] = { :name => "Michael Hartl", :email => "mhartl@example.com" }

=> {:name=>"Michael Hartl", :email=>"mhartl@example.com"}

**>>** params

=> {:user=>{:name=>"Michael Hartl", :email=>"mhartl@example.com"}}

**>>**  params[:user][:email]

=> "mhartl@example.com"

As with arrays and ranges, hashes respond to the **each** method. For example, consider a hash named **flash** with keys for two conditions, **:success** and **:error**:

**>>** flash = { :success => "It worked!", :error => "It failed. :-(" }

=> {:success=>"It worked!", :error=>"It failed. :-("}

**>>** flash.each **do** |key, value|

**?>**  puts "Key **#{**key.inspect**}** has value **#{**value.inspect**}**"

**>> end**

Key :success has value "It worked!"

Key :error has value "It failed. :-("

Note that, while the **each** method for arrays takes a block with only one variable, **each** for hashes takes two, a *key* and a *value*. Thus, the **each** method for a hash iterates through the hash one key-value *pair* at a time.

The last example uses the useful **inspect** method, which returns a string with a literal representation of the object it’s called on:

**>>** puts (1..5).to\_a *# Put an array as a string.*

1

2

3

4

5

**>>** puts (1..5).to\_a.inspect *# Put a literal array.*

[1, 2, 3, 4, 5]

**>>** puts :name, :name.inspect

name

:name

**>>** puts "It worked!", "It worked!".inspect

It worked!

"It worked!"

By the way, using **inspect** to print an object is common enough that there’s a shortcut for it, the **p**function:

**>>** p :name *# Same as 'puts :name.inspect'*

:name