# Equality in Ruby

Here is a quick overview of the different equality operators in Ruby.

## #== (Value Equality)

This is the most fundamental check for equality, it checks if two objects have the same value.

In classes that you write, #== is inherited from the Object class. By default, it will return true if and only if the two objects are literally the same object (pointer equality). This default behavior is not particularly helpful, so you should override it.

This is your chance as the class author to decide what it means for two objects to have the same value.

## #eql? (Hash Equality)

Like #==, #eql? assesses general equality. Unlike most implementations of #==, #eql uses the objects' #hash method to assess equality. So a.eql? b is equivalent to a.hash == b.hash.

If you would like to provide a meaningful #eql? method for your classes, you will need to override the #hash method.

As an example of "meaningful" #eql? methods #== performs type conversion amongst numerics (e.g. Float to Integer), but #eql? does not.

3.0 == 3 #=> true

3.0.eql? 3 #=> false

This is because Float#hash and Integer#hash are not the same.

Now that you know that #eql? is used by Hash to check if an object is a key in a hash, you should not be surprised by this gotcha:

some\_hash = { 3 => 'the third' }

some\_hash[3.0] #=> nil

some\_hash[3] #=> 'the third'

As an exercise for the reader, I suggest playing with using Arrays and Hashes as the keys to a Hash, and seeing what happens when you modify the keys. See below:

some\_array = [1]

some\_hash = { some\_array => 'secrets' }

some\_array << 2

some\_hash[some\_array] #=> ???

## #equal? (Identity Equality)

#equal? does simple identity comparison (pointer comparison). i.e. a.equal? b if and only if a is the same object as b. This is identical to the default behavior of #== in the Object class.

class Dog

# ...

end

a = Dog.new

b = Dog.new

a = c

a.equal? b #=> false

a.equal? c #=> true

#equal? should never be overridden.

## #=== (Case Equality)

#=== has the same behavior as #== for most classes (and by default for classes that you write). This is the method that case uses to determine which block to execute.

case a

when b

# ...

when c

# ...

else

# ...

end

Is equivalent to:

if b === a # triple equals!

# ...

elsif c === a

# ...

else

# ...

end

I encourage you to override #=== if you want to add advanced case/when behavior to your class. As an example, Integer#=== checks to see if the argument is of type Integer ( Integer === 3 #=> true). So you can do this:

case number

when Integer

# ...

when Float

# ...

end

Also, check out what you can do with Regexp#=== (from this [StackOverflow post](http://stackoverflow.com/a/1735777))

tracking\_service = case number

when /^.Z/ then :ups

when /^Q/ then :dhl

when /^96.{20}$/ then :fedex

when /^[HK].{10}$/ then :ups

end

## Equality and Hash Keys

If you want to use instances of a class as hash keys, you need to know how a Hash uses the eql? equality method. When you give a hash a key to look up its associated value, the hash first looks for an existing key object whose hash method returns a value equal to that returned by the given key's hash method. Next, it checks if found\_key\_object.eql?(given\_key\_object), verifying that, in addition to having the same hash, the found key and the given key should be considered equal. Only if both these tests pass will the hash return the desired value instead of nil.

Here's what's going on. Say we have a cat class with a name, and we simply use the hash of the string name as our cat hash value:

class Cat

attr\_reader :name

def initialize(name)

@name = name

end

def hash

@name.hash

end

end

And say we create an entry in a hash using a Cat instance as a key:

hash = {}

cat1 = Cat.new('Fluffy')

hash[cat1] = 'is the best cat'

If we create another Cat instance with the same name, then try to use it to look up the value stored with the first instance, the hash won't be able to find it. This is a problem, as we want two Cat instances with the same data (name) to be treated as the same key by the hash:

hash[cat1] #=> 'is the best cat'

cat2 = Cat.new('Fluffy')

hash[cat2] #=> nil

This happens because our class inherits the default eql? method from Object, which simply tests for pointer equality. To get this working, we need to define Cat#eql? so it returns true if both cat instances have the same name:

class Cat

def eql?(other)

self.name == other.name

end

end

hash[cat2] #=> 'is the best cat'

The takeaway is that if you have created a class and you want to use it as a key in a hash, you should define #hash and #eql?.

## Further Reading

The interested student should read more about them in the [Object documentation](http://ruby-doc.org/core-2.1.2/Object.html), and in this awesome [Stack Overflow post](http://stackoverflow.com/a/7157051).