RubyGems

Prerequisites

Command Line

What you’ll learn

- The gem command
- The ActiveSupport gem
- Monkey patching

RubyGems.org

RubyGems.org
your community gem host

RubyGems.org is a site the Ruby community has settled on as the place for people to upload their extensions to the Ruby standard library. These extensions are called gems. There are currently about ~~78,000~~ 89,000 gems hosted on RubyGems.org.

The Ruby community didn't always use RubyGems. RubyForge was popular for a long time. GitHub hosts many gems as well. But RubyGems is what Ruby uses by default.

gem command

```
man gem
```

RubyGems is not just a website. It’s also the name of a Ruby program.

The RubyGems program grew so popular that it is now included in the Ruby standard library by default as of Ruby version 1.9.

The RubyGems program does 2 things:

- it extends the Ruby language with new classes and methods, and
- it adds a new a command line command named `gem`.

gem subcommands

SYNOPSIS

```
gem command [arguments...] [options...]
```

Like the manpage says, the `gem` command supports a number of sub-commands. The `gem` command syntax *requires* the name of at least one subcommand, followed by optional arguments and options.

list

```
gem list

gem list --remote
```

The `list` subcommand returns a list of all the gems that are currently installed on your system. If you've never installed any gems, yet you still see items in this list, then those are the gems included by default in the Ruby standard library (aka the "library of libraries").

If you'd like to see a list of *all* the gems available on RubyGems.org, you can use the `--remote` flag. It'll take a while to run. There are 10s-of-1000s of gems available.

install / uninstall

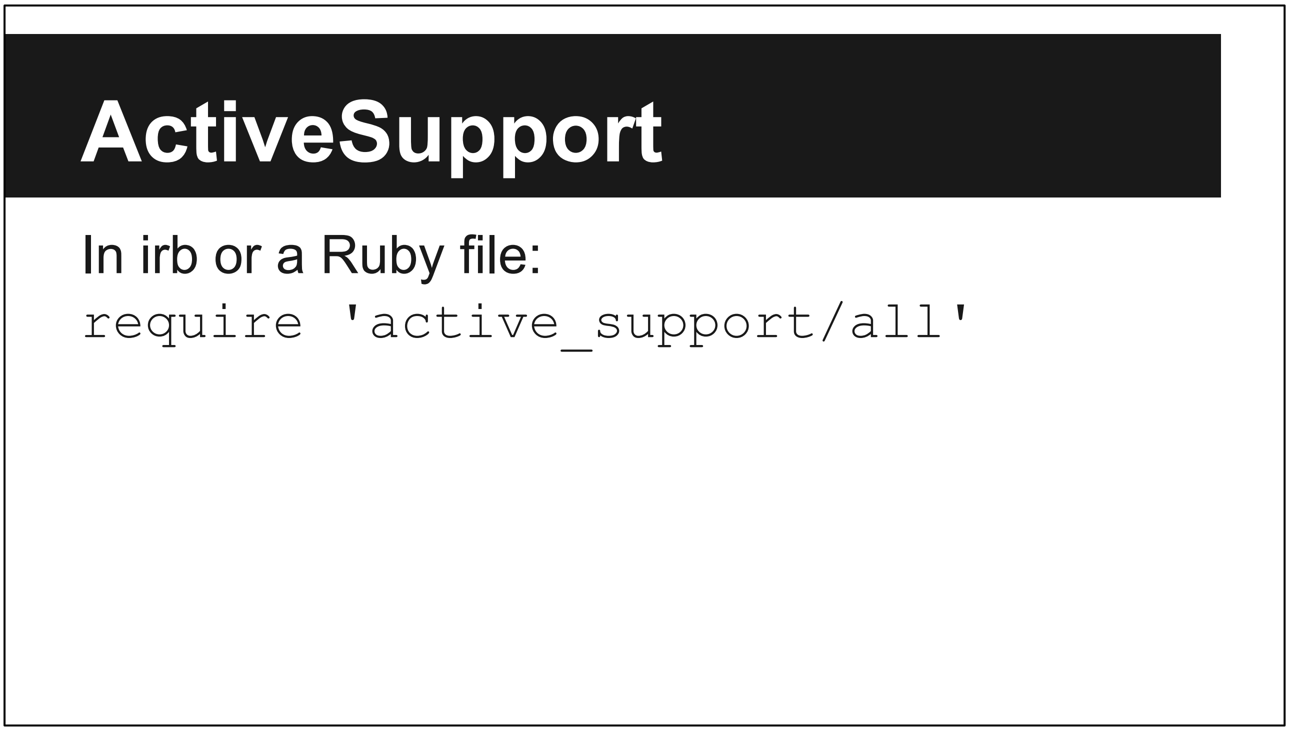
```
gem install activesupport

gem uninstall activesupport
```

To install a gem, use the `install` subcommand, followed by the gem name. `install` downloads the gem from RubyGems.org (requires an internet connection) and installs it into your local copy of Ruby. That’s *all* you have to do to *immediately* start using someone else’s code for free.

To uninstall a gem, use the `uninstall` subcommand.

Are you still clicking a link in a web page, downloading a file, and clicking on an icon to install your software? That is so passé. So 2000-late. This is how command line users install and uninstall software.



ActiveSupport

In irb or a Ruby file:

```
require 'active_support/all'
```

Let’s play around with the `activesupport` gem.

The ActiveSupport gem is a part of the Rails framework. It adds some new methods to the Ruby standard library.

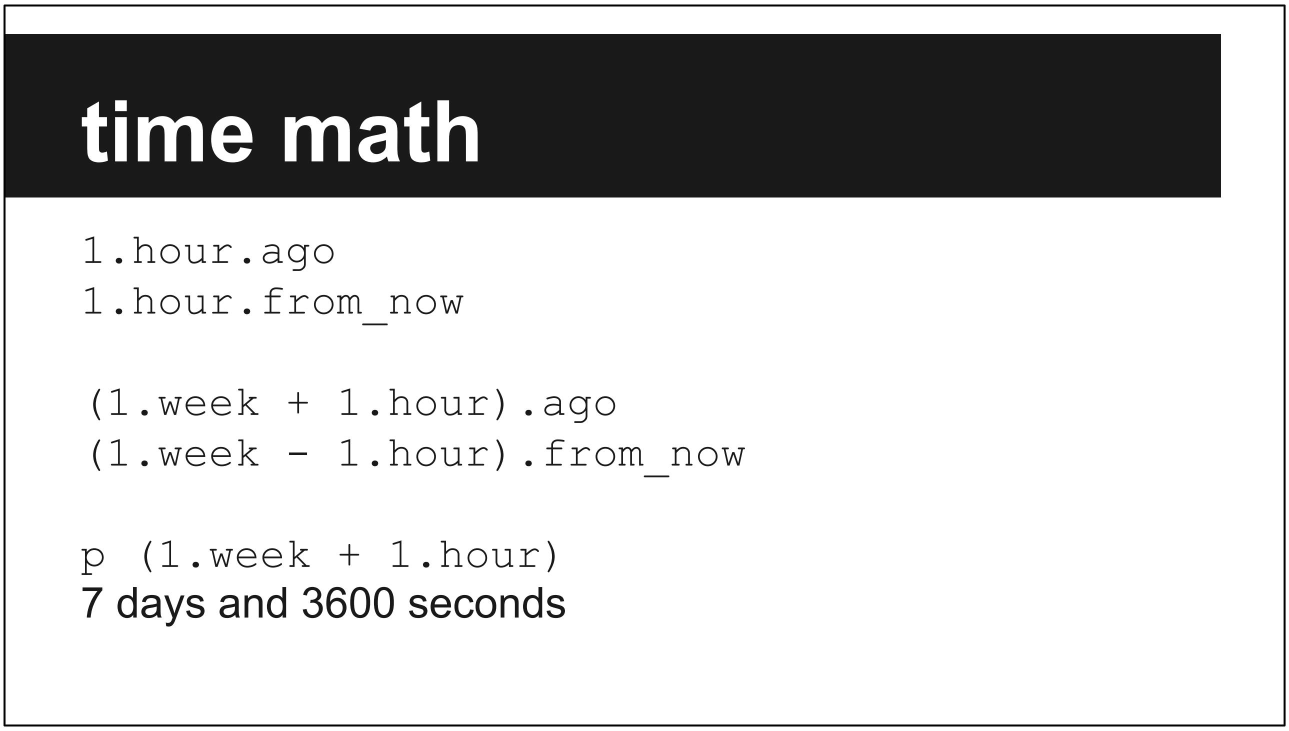
To enable these new features in a Ruby file or in `irb`, you need to `require` it first.

We’ve seen `require` before. It links *your* Ruby files together. But it also lets you add code from Ruby gems into your programs.

blank?

```
"".blank?
nil.blank?
false.blank?
"".blank?
{}.blank?
[].blank?
```

ActiveSupport adds a ton of new and useful methods to Ruby. The blank? method, for example, will return true for all of these objects. This is better than depending on Ruby’s concept of “truthiness”, which, can be confusing.



time math

```
1.hour.ago
1.hour.from_now

(1.week + 1.hour).ago
(1.week - 1.hour).from_now

p (1.week + 1.hour)
```

7 days and 3600 seconds

One of the best things the Rails ActiveSupport folks added to the Ruby language is the ability to easily express and do math with time.

If you’d like to get the time 1 hour ago, just type 1.hour.ago.
If you’d like to get the time 1 hour in the future, just type 1.hour.from_now

You can combine units of time as well using addition and subtraction. So you can get the time 1 week and 1 hour ago. Or you can get the time 1 week less 1 hour ago.

You can even output, in plain English, what these time objects really mean.

Monkey patching

How do these gems work? How did ActiveSupport add these methods to the standard library without introducing new Classes? The answer is the dreaded monkey patch.

Open Classes

```
class String
    def monkey?
        self.eql? "monkey"
    end
end
```

All Ruby classes are open. That means you can add whatever you want to existing classes: new instance variables, new methods, new modules, *anything*.

Here, for example, I’ve re-opened the Ruby String class and added a method called “monkey?” that returns true if the string contains the word “monkey”.

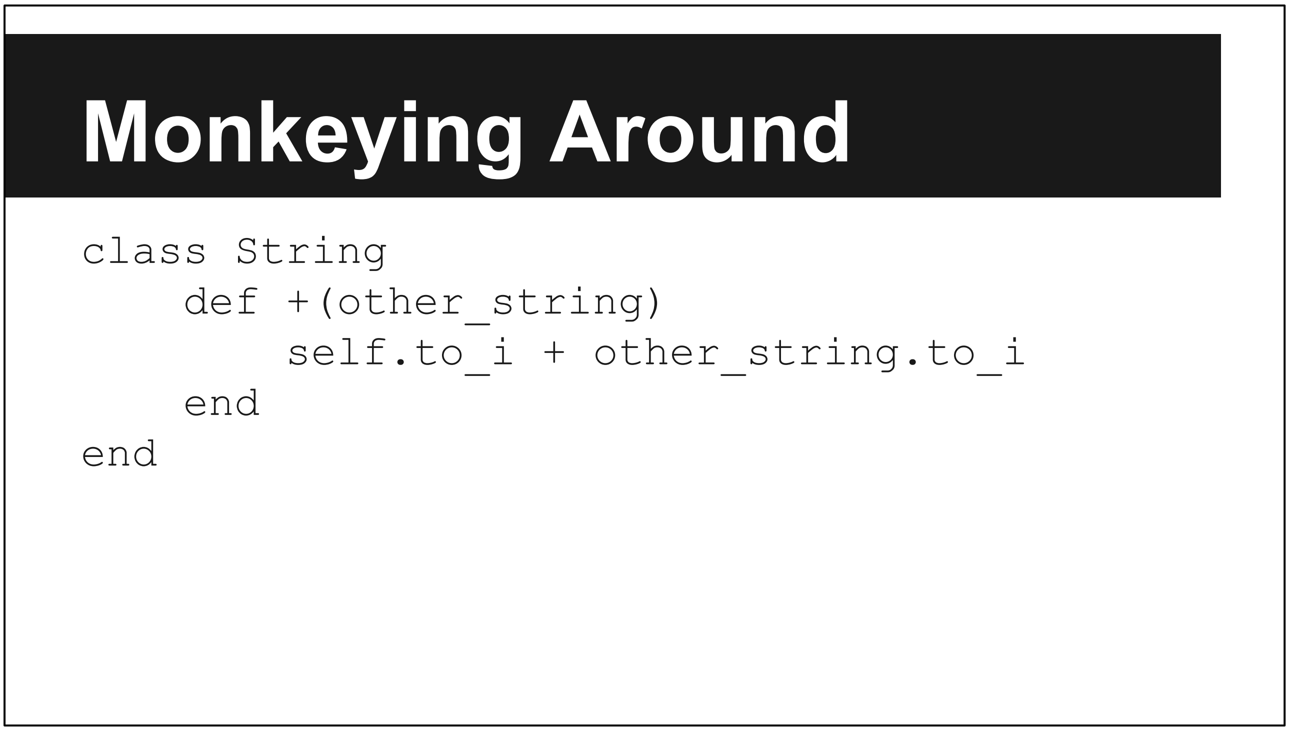
Open Classes

```
puts "".monkey?
puts "monkey".monkey?
```

Now I can call my “monkey?” method on Ruby strings and Ruby won’t raise an Error to complain that the method is missing.

This is called is called “monkey patching” for various mostly forgotten reasons. You’re “monkeying around” with the standard library, for one.

“Monkey patching” can be a term of endearment, but it’s also a warning. You don’t want to be a code monkey. Messing around with objects in the Ruby Standard Library like this is usually not a good idea. Remember that your user is not just the consumer. He can also be another coder. And other coders can be dumb or evil.



Monkeying Around

```
class String
    def +(other_string)
        self.to_i + other_string.to_i
    end
end
```

For example, a monkey patching coder may decide that he doesn’t like the fact that String addition concatenates instead of adding. So he may overwrite the + method to convert each string to a number and add them.

Monkeying Around

```
puts "1" + "2"
```

This is now 3 (Fixnum)

```
puts "a" + "b"
```

This is now 0 (Fixnum)

Now code that you thought you understood doesn’t work the same way anymore. “1” + “2” is the number 3. “a” + “b” is now the number 0. And anything you wrote assuming String addition worked a particular way is now broken. And it’s all because you installed and required some gem that you may not have realized would do this to you.

Rails gets away with monkey patching in ActiveSupport because they wrote Rails and they’re special. You shouldn’t do it - unless you have a very very good reason to.

Duck Punching

Another phrase for “monkey patching” is “duck punching”. This is a play on “duck typing”, the Ruby convention to care more about what an object does than what it is.

So, if you recall, “duck typing” meant our add_two method may assume it’s getting Fixnum argument, but what it really needs is just something that supports the addition method, so it can add two to whatever the argument is.

“Duck punching” comes from a conference talk in 2007 where a speaker said “if this duck is not giving you the noise that you want, you’ve got to just punch that duck until it returns what you expect.”

Whatever you call it, the fact that Ruby classes are open is one of the reasons why it’s called “extensible”. It’s very easy to extend the language and get it to do what you want (for good or evil). The easy extensibility of Ruby is the reason why there are 78,000 gems currently available.

TODO:
(Hipster) Gemitis