The Ruby Language FAQ

Originally by: *Shugo Maeda* <mailto:shugo@netlab.co.jp>. Now maintained by *Dave Thomas* <mailto:dave@pragmaticprogrammer.com> with help from *Andy Hunt* <mailto:andy@pragmaticprogrammer.com>. Thanks to *gotoken* <mailto:gotoken@math.sci.hokudai.ac.jp> for the original English translation.

Many thanks to Arima Yasuhiro, John Dell'Aquila, Robert Gustavsson Clemens Hintze, Josh Huber, H Morita, Aleksi Niemelä, Hugh Sasse, Conrad Schneiker, Larry W. Virden, Jim Weirich, and of course, Matz.

Contents

1 General questions

1.1 What is Ruby?

Ruby is a *simple and powerful object-oriented programming language*, created by Yukihiro Matsumoto (who goes by the handle "matz" in this document and on the mailing lists).

Like Perl, Ruby is good at text processing. Like Smalltalk, everything in Ruby is an object, and Ruby has blocks, iterators, meta-classes and other good stuff.

You can use Ruby to write servers, experiment with prototypes, and for everyday programming tasks. As a fully-integrated object-oriented language, Ruby scales well.

Ruby features:

- Simple syntax,
- Basic OO features (classes, methods, objects, and so on),
- Special OO features (Mix-ins, singleton methods, renaming, ...),
- Operator overloading,
- Exception handling,
- Iterators and closures,
- Garbage collection,
- Dynamic loading (depending on the architecture),
- High transportability (runs on various Unices, Windows, DOS, OSX, OS/2, Amiga, and so on)

1.2 Show me some Ruby code.

Let’s define a class called Person, with a name and an age. We’ll test our code by creating a few people and examining them.

```
class Person
  attr_accessor :name, :age
  def initialize(name, age)
    @name = name
    @age  = age.to_i
  end
  def inspect
    "#@name (#@age)"
  end
end

p1 = Person.new(’elmo’, 4)
p2 = Person.new(’zoe’, 7)

p1              # -> elmo (4)
p2              # -> zoe (7)
```

Now let’s populate an array of people by reading their names and ages from a file containing lines like:

```
bert:    8
cookie: 11
elmo:    4
ernie:   8
zoe:     7
```

The code uses regular expressions to parse successive lines from the input file, creating a new `Person` object for each match and pushing it on to the end of the array `people`.

```
people = Array.new

File.foreach("ages") { |l|
  people « Person.new($1, $2) if l =~ /(.*):\s+(\d+)/
}

people           # -> [bert (8), cookie (11), elmo (4), ernie (8), zoe (7)]
```

Now, let's sort the result based on the person's age. There are many ways to do this. We can define a sort block, which tells Ruby how to do the comparison of two people:

```
sorted = people.sort do |a,b| a.age <=> b.age end
sorted           # -> [elmo (4), zoe (7), bert (8), ernie (8), cookie (11)]
```

Another way would be to change the comparison method for class `Person`:

```
class Person
  def <=>(other)
    @age <=> other.age
  end
end
people.sort      # -> [elmo (4), zoe (7), bert (8), ernie (8), cookie (11)]
```

1.3 Why the name 'Ruby'?

Influenced by Perl, Matz wanted to use a jewel name for his new language, so he named Ruby after a colleague's birthstone.

Later, he realized that Ruby comes right after Perl in several situations. In birthstones, pearl is June, ruby is July. When measuring font sizes, pearl is 5pt, ruby is 5.5pt. He thought Ruby was a good name for a programming language newer (and hopefully better) than Perl.

(Based on an explanation from matz in [ruby-talk:00394] on June 11, 1999.)

1.4 What is the history of Ruby?

The following a summary of a posting made by Matz in [ruby-talk:00382] on June 4, 1999. (The birthday of Ruby is corrected in [ruby-list:15977]).

> Well, Ruby was born on February 24 1993. I was talking with my colleague about the possibility of an object-oriented scripting language. I knew Perl (Perl4, not Perl5), but I didn't like it really, because it had smell of toy language (it still has). The object-oriented scripting language seemed very promising.

I knew Python then. But I didn't like it, because I didn't think it was a true object-oriented language—OO features appeared to be add-on to the language. As a language manic and OO fan for 15 years, I really wanted a genuine object-oriented, easy-to-use scripting language. I looked for, but couldn't find one.

So, I decided to make it. It took several months to make the interpreter run. I put it the features I love to have in my language, such as iterators, exception handling, garbage collection.

Then, I reorganized the features of Perl into a class library, and implemented them. I posted Ruby 0.95 to the Japanese domestic newsgroups in Dec. 1995.

Since then, highly active mailing lists have been established and web pages formed.

1.5 Where is the Ruby Home Page?

The official Ruby Home Page is *http://www.ruby-lang.org* <http://www.ruby-lang.org> (in English) and *http://www.ruby-lang.org/ja/* <http://www.ruby-lang.org/ja> (in Japanese).

You can also find Ruby information at *http://www.rubycentral.com* <http://dev.rubycentral.com>. In particular, there is a complete *online reference* <http://www.dev.rubycentral.com/ref/index.html> to Ruby's built-in classes and methods.

1.6 Is there a Ruby newsgroup?

comp.lang.ruby was established in May, 2000 (thanks to the efforts of *Conrad Schneiker* <mailto:schneiker@jump.net>).

1.7 Is there a Ruby mailing list?

There are five mailing lists now talking about Ruby. The first is in English, the last four in Japanese:

- ruby-talk: English language discussion of Ruby.
- ruby-list: Japanese language discussion of Ruby.
- ruby-dev: List for Ruby developers.
- ruby-ext: List for people writing extensions for or with Ruby.
- ruby-math: Ruby in mathematics.

See *joining the mailing list* <http://www.ruby-lang.org/en/ml.html>.

You can search the mailing list archives using *http://blade.nagaokaut.ac.jp/ruby/ruby-talk/index.shtml* <http://blade.nagaokaut.ac.jp/ruby/ruby-talk/index.shtml>. (This is the url for the ruby-talk list: munge as required for the others).

1.8 **Changed** How can I thread the mailing list in mutt?

The Ruby mailing list software adds a prefix to the subject lines, for example [ruby-talk:1234]. This can confuse the threading in some mail user agents.

In mutt, you can get threading to work using the following variable setting.

```
# reply regexp, to support MLs like ruby-talk.
set reply_regexp="^(\[[a-z0-9:-]+\][[:space:]]*)?(re([\[0-9\]+])*|aw):[[:space:]]*"
```

1.9 Which is correct, Ruby or ruby?

Officially, the language is called "Ruby". On most systems, it is invoked using the command "`ruby`". It's OK to use ruby instead of Ruby.

Please *don't* use RUBY as the language name.

Originally, or historically, it was called "ruby".

1.10 Are there any Ruby books?

Programming Ruby: The Pragmatic Programmer's Guide <http://www.awlonline.com/product/0,2627,0201710897,00.html>, (the Pickaxe Book) by David Thomas and Andrew Hunt: ISBN 0-20171-089-7, Addison-Wesley, October 2000.

A Japanese language Ruby reference book by matz, et al and published by ASCII is available in Japan (ISBN 4-7561-3254-5). An English translation, "*The Ruby Programming Language* <http://www.awlonline.com/product/0,2627,020171096X,00.html>," is in the works from Addison-Wesley (ISBN 020171096X).

>

A Japanese language "Ruby Pocket Reference" is published by O'Reilly Japan (ISBN 4-87311-023-8). Let O'Reilly in the US know if you'd like to see a translation.

>

In addition, "*Mastering Regular Expressions* <http://www.oreilly.com/catalog/regex/>," by Jeffrey Friedl, (the Hip Owl Book): ISBN 1-56592-257-3 from O'Reilly & Associates, is a reference work that covers the art and implementation of regular expressions in various programming languages. Most of it is highly relevant to Ruby regular expressions.

1.11 Which editors provide support for Ruby.

- ***Emacs*** <http://www.gnu.org/software/emacs/emacs.html> or ***XEmacs*** <http://www.xemacs.org>: ruby-mode.el is supplied in the Ruby distribution. With some versions of XEmacs, you may need to add (load "font-lock") to your .emacs file to allow ruby-mode.el to detect the syntax highlighting package you're using.
- ***Vim*** <http://www.vim.org>: Vim 5.7 and later have Ruby syntax files as standard in the runtime package. For prior versions, a syntax file for Ruby is available at *http://www.xs4all.nl/~hipster/lib/ruby/ruby.vim* <http://www.xs4all.nl/~hipster/lib/ruby/ruby.vim>.
- ***Jedit*** <http://jedit.sourceforge.net>: A portable editor written in Java, comes with support for Ruby.
- ***Jed*** <http://space.mit.edu/~davis/jed.html>: An s-lang file supporting Ruby is available at *http://kondara.sdri.co.jp/~g/slang/ruby.sl* <http://kondara.sdri.co.jp/~g/slang/ruby.sl>.
- **Nedit** (*http://www.nedit.org* <http://www.nedit.org>): Eric Santonacci has written Ruby support for Nedit, available from *ftp://ftp.talc.fr/pub/ruby/ruby.nedit-0.1.tar.gz* <ftp://ftp.talc.fr/pub/ruby/ruby.nedit-0.1.tar.gz>.

- Barry Shultz has written a Ruby definition file for TextPad, available at *http://www.textpad.com/add-ons/ntsyn.html* <http://www.textpad.com/add-ons/ntsyn.html>.

1.12 How can I annotate Ruby code with its results?

People commonly annotate Ruby code by showing the results of executing each statement as a comment attached to that statement. For example, in the following code, we show that the assignment generates the string "Billy Bob", and then result of extracting some substrings.

```
str = "Billy" + " Bob"           # -> "Billy Bob"
str[0,1] + str[2,1] + str[-2,2]  # -> "Blob"
```

Gotoken's `xmp` package, available from *http://www.ruby-lang.org/en/raa-list.rhtml?name=xmp* <http://www.ruby-lang.org/en/raa-list.rhtml?name=xmp> is a utility that annotates Ruby source code this way.

Emacs and vim users can integrate this with their editing environments, which is useful if you want to send people e-mail with annotated Ruby code. Having installed xmp, Emacs users can add the following to their .emacs file:

```
(defun ruby-xmp-region (reg-start reg-end)
  "Pipe the region through Ruby's xmp utility and replace the region with
   the result."
  (interactive "r")
  (shell-command-on-region reg-start reg-end
                           "ruby -r xmp -n -e 'xmp($_, \"%l\t\t# %r\n\")'"
                           t))

(global-set-key [(meta f10)] 'ruby-xmp-region)
```

Vim users can use the mapping (thanks to hipster):

```
map <M-F10> :!ruby -r xmp -n -e 'xmp($_, "\%l\t\t\# \%r\n")'<CR>
```

In both cases, highlight a region of code and hit Meta-F10 to annotate it.

1.13 I can't understand Ruby even after reading the manual!

The syntax of Ruby has been fairly stable since Ruby 1.0, but new features are added every now and then. So, the books and the online documentation can get behind.

If you have a problem, feel free to ask in the mailing list (see ruby-talk mailing list (see section 1.7 on page 7)). Generally you'll get timely answers from matz himself, the author of the language, from other gurus, and from those who've solved problems similar to your own.

Please include the output of `ruby -v` along with any problematic source code.

If you have a problem using `irb (see section 10.1 on page 41)`, be aware that it has some limitations. Try the script using `irb -single-irb`, or directly using the `ruby` command.

There might be similar questions in the mailing list, and it is good netiquette to read through recent mails (RFC1855:3.1.1, 3.1.2) before asking. But do ask on the list, and a correct answer will be forthcoming.

2 How Does Ruby Stack Up Against...?

2.1 How Does Ruby Compare With Python?

Python and Ruby are both object oriented languages that provide a smooth transition from procedural to OO programming styles. Smalltalk, by contrast, is object only - you can't do anything until you understand objects, inheritance and the sizable Smalltalk class hierarchy. By providing procedural training wheels, Python and Ruby "fix" one of the features that may have kept Smalltalk out of the mainstream. The two languages differ by approaching this solution from opposite directions.

Python is a hybrid language. It has functions for procedural programming and objects for OO programming. Python bridges the two worlds by allowing functions and methods to interconvert using the explicit "self" parameter of every method def. When a function is inserted into an object, the first argument automagically becomes a reference to the receiver.

Ruby is a pure OO language that can masquerade as a procedural one. It has no functions, only method calls. In a Ruby method the receiver, also called self, is a hidden argument like "this" in C++. A "def" statement outside of a class definition, which is a function in Python, is actually a method call in Ruby. These ersatz functions become private methods of class Object, the root of the Ruby class hierarchy. Procedural programming is neatly solved from the other direction - everything is an object. If the user doesn't grok objects yet, they can just pretend that "def" is a function definition and still get useful work done.

Ruby's OO purity provides a number features that Python lacks or is still working toward: a unified type/class hierarchy, metaclasses, the ability to subclass *everything*, and uniform method invocation (none of this `len()` is a function but `items()` is a method rubbish). Ruby, like Smalltalk, only supports single inheritance, but it does have a very powerful mixin concept: a class definition may include a module, which inserts that module's methods, constants, etc. into the class.

Ruby, again like Smalltalk, provides closures and code blocks and uses them to the same good effect. The Ruby collection classes and iterators are outstanding, much more powerful and elegant than the ad hoc solutions that Python is sprouting (lambdas and list comprehensions).

Ruby's syntax and design philosophy are heavily influenced by Perl. It has a lot of syntactic variability. Statement modifiers (if, unless, while, until, etc.) may appear at the end of any statement. Some key words are optional (the "then" in an "if" statement for example). Parentheses may sometimes be elided in method calls. The receiver of a method may usually be elided. Many, many things are lifted directly from Perl. Built in regular expressions, $_ and friends, here documents, the single-quoted / double-quoted string distinction, $ and @ prefixes to distinguish different kinds of names and so forth.

If you like Perl, you will like Ruby and be right at home with its syntax. If you like Smalltalk, you will like Ruby and be right at home with its semantics. If you like Python, you may or may not be put off by the huge difference in design philosophy between Python and Ruby/Perl.

Ruby is much more complex than Python but its features, for the most part, hang together well. Ruby is well designed and full of neat ideas that might be mined for P3K. I'm not sure how many Python programmers will be attracted to it though - it hasn't won me over (yet). But it is worthy of serious study and could be a real threat to Perl.

Posted by John Dell'Aquila `<mailto:jbd@alum.mit.edu>` *in* `comp.lang.python`*, 11/17/2000. Reproduced with permission.*

3 Installing Ruby

3.1 What operating systems support Ruby?

Ruby is developed under Linux, and is written in fairly straightforward C. It runs under UNIX, DOS, Windows 95/98/NT/2000, Mac OSX, BeOS, Amiga, Acorn Risc OS, and OS/2.

H Morita notes:

There's a MacOS (not X) port of Ruby, by Hisakuni FUJIMOTO at *http://www.imasy.or.jp/~hisa/ruby/macruby.html* <http://www.imasy.or.jp/~hisa/ruby/macruby.html>. However it's based on Ruby 1.1b7, and hasn't been updated since December 1999. It's highly experimental. It may crash and sometimes freeze the OS, even with the sample scripts included in the Ruby distribution. (Sounds like fun ;-).

>

He goes on to mention there's a precompiled binary for OS/2 by Kenji Nagasawa at *http://member.nifty.ne.jp/~kenn/soft.html* <http://member.nifty.ne.jp/~kenn/soft.html>.

3.2 Where can I get Ruby sources?

The latest version of Ruby can be downloaded from: *http://www.ruby-lang.org/en/download.html* <http://www.ruby-lang.org/en/download.html> Mirror sites are also listed on this page.

Also on this page is a link to a nightly snapshot of the development tree.

3.3 Can I get to the development source tree?

If you have a CVS client, you can check out the current source tree using:

```
% cvs -d :pserver:anonymous@cvs.netlab.co.jp:/home/cvs login
(Logging in to anonymous@cvs.netlab.co.jp)
CVS password: guest
% cvs -d :pserver:anonymous@cvs.netlab.co.jp:/home/cvs co ruby
```

If you do not have CVS you can get a nightly snapshot of the development source from *ftp://ftp.netlab.co.jp/pub/lang/ruby/snapshot.tar.gz* <ftp://ftp.netlab.co.jp/pub/lang/ruby/snapshot.tar.gz>.

3.4 How do I compile Ruby?

Under Unix, Ruby uses the autoconf system to configure the build environment. You don't need the autoconf command on your box to build Ruby from a distribution; just use the commands:

```
% ./configure  [configure options]
% make
% make test
% make install
```

You may need superuser privileges to install Ruby if you don't override the default installation location (`/usr/local`). You can get a full list of configure options using:

```
% ./configure -help
```

If you are working from the CVS archive, you may need to run `autoconf` before running `configure`.

3.5 **Changed** How do I tell Ruby where my libraries are?

On some systems, the build process may fail to find libraries used by extension modules (for example the `dbm` libraries).

You can tell Ruby where to find libraries using options to `configure`. From [ruby-talk:5041]:

```
./configure -with-xxx-yyy=DIR
```

where *xxx* is either

```
opt           extra software path in general
dbm           path for dbm library
gdbm          path for gdbm library
x11           ...for X11..
tk            ...for Tk...
tcl           ...for Tcl...
```

and *yyy* is either

```
dir           specifies -I DIR/include -L DIR/lib
include       specifies -I DIR
lib           specifies -L DIR
```

On HP-UX, there may be problems building with gcc. Try using the native compiler instead. WATANABE Tetsuya recommends:

```
CC="cc -Ae" CFLAGS=-O ./configure -prefix=/opt/gnu
```

There may also be problems with HP's native sed. He recommends installing the GNU equivalent.

3.6 Are precompiled binaries available?

A single download that contains everything you need to run Ruby under various Windows operating systems. is available from *RubyCentral's One-click Windows installer* <http://www.rubycentral.com/downloads/ruby-install.html>. This installation uses cygwin, and includes Tk support.

If you want other installation options, precompiled binaries for Windows are also available from *http://www.os.rim.or.jp/~eban/* <http://www.os.rim.or.jp/~eban/>. If you download the `ruby-1.x.y-yyyymmdd-i386-cygwin.tar.gz` package (which is a good choice), you'll also need to download the cygwin DLL, available from the same page.

> *Reuben Thomas* <mailto:Reuben.Thomas@cl.cam.ac.uk> writes: You could mention that there's a port to Acorn RISC OS, currently of v1.4.3. I made the port, and have no plans to maintain it, but I did send the patches to matz, so newer versions may well compile too.
>
> I do provide a binary distribution of 1.4.3 for the Acorn at *http://www.cl.cam.ac.uk/users/rrt1001/ruby.zip* <http://www.cl.cam.ac.uk/users/rrt1001/ruby.zip>.

3.7 What's all this 'cygwin', 'mingw', and 'djgpp' stuff?

Ruby is written to take advantage of the rich feature set of a Unix environment. Unfortunately, Windows is missing some of the functions, and implements others differently. As a result, some kind of mapping layer is needed to run Ruby (and other Unix-based programs) under windows.

You may come across different versions of the Ruby executable that use different wrapper mapping layers.

The rbdj version is a stand-alone version of the Windows binary of Ruby. It uses the DJ Delorie tools (*http://www.delorie.com* <www.delorie.com>).

The rbcw version is a Windows binary of Ruby that requires the cygwin library, available at *http://www.cygwin.com* <http://www.cygwin.com> or from the Ruby download pages. Cygwin is a both an emulation layer and a set of utilities initially produced by Cygnus Solutions (now part of Redhat). The Cygwin version of Ruby probably has the fullest set of features under Windows, so most programmers will want to use it.

To use the rbcw version, you will need to install the cygwin .dll separately. Once you have installed cygwin on your computer, copy cygwin1.dll (which is found in the `bin` subdirectory of the cygwin distribution) to your Windows\System32 folder (or somewhere else on your path).

Thanks to Anders Schneiderman for the basis of this description

3.8 Why doesn't Tk graphics work under Windows?

1. Is Tk installed correctly on your Windows box? Go to *http://dev.scriptics.com/software/tcltk/* <http://dev.scriptics.com/software/tcltk/> to find a precompiled binary Tcl/Tk distribution for your box.
2. Are the environment variables `TCL_LIBRARY` and `TK_LIBRARY` pointing to the directories containing tcl and tk?
3. Is the tk library in your path?

4 Variables, constants, and arguments

4.1 Does assignment generate a new copy of an object?

All variables and constants reference (point at) some object. (The exception is uninitialized local variables, which reference nothing. These raise a `NameError` exception if used). When you assign to a variable, or initialize a constant, you set the object that the variable or constant references.

Assignment on its own therefore never creates a new copy of an object.

There's a slightly deeper explanation in certain special cases. Instances of Fixnum, NilClass, TrueClass, and FalseClass are contained directly in variables or constants–there is no reference involved. A variable holding the number 42 or the constant `true`, actually holds the value, and not a reference to it. Assignment therefore physically produces a copy of objects of these types. We discuss this more in Immediate and Reference Objects (see section 9.3 on page 35).

4.2 What is the scope of a local variable?

A new scope for a local variable is introduced in the (1) toplevel (main), (2) a class (or module) definition, or (3) a method definition.

```
i  =  1       # (1)
class Demo
```

```
  i = 2       # (2)
  def meth
    i = 3     # (3)
    print "In method, i = ", i, "\n"
  end
  print "In class, i = ", i, "\n"
end
print "At top level, i = ", i, "\n"
Demo.new.meth

Produces:

In class, i = 2
At top level, i = 1
In method, i = 3
```

(Note that the class definition is executable code: the trace message it contains is written as the class is defined).

A block (“{” ... “}” or do ... end) almost introduces a new scope ;-) Locals created within a block are not accessible outside the block. However, if a local within the block has the same name as an existing local variable in the caller’s scope, then no new local is created, and you can subsequently access that variable outside the block.

```
a = 0
1.upto(3) do |i|
  a += i
  b = i*i
end
a                # -> 6
# b is not defined here
```

This becomes significant when you use threading–each thread receives its own copy of the variables local to the thread’s block:

```
threads = []

for name in [’one’, ’two’ ] do
  threads « Thread.new {
    localName = name
    a = 0
    3.times do |i|
      Thread.pass
      a += i
      print localName, ": ", a, "\n"
    end
  }
end

threads.each {|t| t.join }

Produces:

onetwo: : 00

onetwo: : 11
```

```
onetwo: : 33
```

`while`, `until`, and `for` are control structures, not blocks, so local variables within them will be accessible in the enclosing environment. `loop`, however, is a method and the associated block introduces a new scope.

4.3 When does a local variable become accessible?

Actually, the question may be better asked as: "at what point does Ruby work out that something is a variable?" The problem arises because the simple expression “a” could be either a variable *or* a call to a method with no parameters. To decide which is the case, Ruby looks for assignment statements. If at some point in the source prior to the use of “a” it sees it being assigned to, it decides to parse “a” as a variable, otherwise it treats it as a method. As a somewhat pathological case of this, consider this code fragment, submitted by Clemens Hintze:

```
def a
  print "Function ’a’ called\n"
  99
end

for i in 1..2
  if i == 2
    print "a=", a, "\n"
  else
    a = 1
    print "a=", a, "\n"
  end
end

Produces:

a=1
Function ’a’ called
a=99
```

During the parse, Ruby sees the use of “a” in the first print statement and, as it hasn’t yet seen any assignment to “a”, assumes that it is a method call. By the time it gets to the second print statement, though, it *has* seen an assignment, and so treats “a” as a variable.

Note that the assignment does not have to be executed—Ruby just has to have seen it. This program does not raise an error.

```
a = 1 if false; a  # -> nil
```

This issue with variables is not normally a problem. If you do bump into it, try putting an assignment such as `a = nil` before the first access to the variable. This has the additional benefit of speeding up the access time to local variables that subsequently appear in loops.

4.4 What is the scope of a constant?

A constant defined in a class/module definition can be accessed directly within that class or module’s definition.

You can directly access the constants in outer classes and modules from within nested classes and constants.

You can also directly access constants in superclasses and included modules.

Apart from these cases, you can access class and module constants using the `::` operator–`ModuleName::CONST1` or `ClassName::CONST2`.

4.5 How are arguments passed?

The actual argument is *assigned* to the formal argument when the method is invoked. (See assignment (see section 4.1 on page 13) for more on the semantics of assignment.)

```
def addOne(n)
  n += 1
end
a = 1
addOne(a)      # -> 2
a              # -> 1
```

As you are passing object references, it is possible that a method may modify the contents of a mutable object passed in to it.

```
def downer(string)
  string.downcase!
end
a = "HELLO"    # -> "HELLO"
downer(a)      # -> "hello"
a              # -> "hello"
```

There is no equivalent of other language's pass-by-reference semantics.

4.6 Does assignment to a formal argument influence the actual argument?

A formal argument is a local variable. Within a method, assigning to a formal argument simply changes the argument to reference another object.

4.7 What happens when I invoke a method via a formal argument?

All Ruby variables (including method arguments) act as references to objects. You can invoke methods in these objects to get or change the object's state and to make the object do something. You can do this with objects passed to methods. You need to be careful when doing this, as these kinds of side effects can make programs hard to follow.

4.8 What does "*" prepended to an argument mean?

When used as part of a formal parameter list, the asterisk allows arbitrary numbers of arguments to be passed to a method by collecting them into an array, and assigning that array to the starred parameter.

```
def foo(prefix, *all)
  for e in all
    print prefix, e, " "
  end
end

foo("val=", 1, 2, 3)
```

```
Produces:

val=1 val=2 val=3
```

When used in a method call, `*` expands an array, passing its individual elements as arguments.

```
a = [1, 2, 3]
foo(*a)
```

You can prepend `*` to the last argument of

1. Left hand side of a multiple assignment.
2. Right hand side of a multiple assignment.
3. Definition of method formal arguments.
4. Actual arguments in a method call.
5. In `when` clause of `case` structure.

For example:

```
x, *y = [7, 8, 9]
x               # -> 7
y               # -> [8, 9]
x,    = [7, 8, 9]
x               # -> 7
x     = [7, 8, 9]
x               # -> [7, 8, 9]
```

4.9 What does “&” prepended to an argument mean?

If the last formal argument of a method is preceeded with an ampersand, a block following the method call will be converted into a `Proc` object and assigned to the formal parameter.

If the last actual argument in a method invocation is a `Proc` object, you can preceed its name with an ampersand to convert in into a block. The method may then use `yield` to call it.

```
square = proc { |i| i*i }

def meth1(&b)
  print b.call(9), "\n"
end

meth1 { |i| i+i }

def meth2
  print yield(8), "\n"
end

meth2 { |i| i+i }
meth2 &square
```

```
Produces:

18
16
64
```

4.10 How can I specify a default value for a formal argument?

```
def greet(p1='hello', p2='world')
  print "#{p1} #{p2}\n"
end

greet
greet "hi"
greet "morning", "mom"

Produces:

hello world
hi world
morning mom
```

The default value (which can be an arbitrary expression) is evaluated when the method is invoked. It is evaluated using the scope of the method.

4.11 How do I pass arguments to a block?

The formal parameters of a block appear between vertical bars at the start of the block:

```
proc { |a, b| a <=> b }
```

These parameters are actually local variables. If an existing local of the same name exists when the block executes, that variable will be modified by the call to the block. This may or may not be a good thing.

Typically, arguments are passed to a block using `yield` (or an iterator that calls `yield`), or by using the `Proc.call` method.

4.12 Why did my object change unexpectedly?

```
A = a = b = "abc"
b.concat("d")    # -> "abcd"
a                # -> "abcd"
A                # -> "abcd"
```

Variables hold *references* to objects. The assignment `A = a = b = "abc"` put a reference to the string "abc" into `A`, `a`, and `b`.

When you called `b.concat("d")`, you invoked the `concat` method on that object, changing it from "abc" to "abcd". Because `a` and `A` also reference that same object, their apparent value changes too.

This is less of a problem in practice than it might appear.

In addition, as of Ruby 1.5.2, all objects may be *frozen*, protecting them from change.

4.13 Does the value of a constant ever change?

A constant is a variable whose name starts with an upper case letter. In older Ruby implementations, when a constant was assigned a new value, a warning was issued. In newer Rubies, constants may not be reassigned from within instance methods, but can otherwise be changed at will.

5 Iterators

5.1 What is an iterator?

An iterator is a method which accepts a block or a `Proc` object. In the source file, the block is placed immediately after the invocation of the method. Iterators are used to produce user-defined control structures–especially loops.

Let's look at an example to see how this works. Iterators are often used to repeat the same action on each element of a collection, like this:

```
data = [1, 2, 3]
data.each do |i|
  print i, "\n"
end

Produces:

1
2
3
```

The `each` method of the array `data` is passed the `do...end` block, and executes it repeatedly. On each call, the block is passed successive elements of the array.

You can define blocks with {...} in place of `do`...`end`.

```
data = [1, 2, 3]
data.each { |i|
  print i, "\n"
}

Produces:

1
2
3
```

This code has the same meaning as the last example. However, in some cases, precedence issues cause `do`...`end` and {...} to act differently.

```
foobar a, b do .. end # foobar is the iterator.
foobar a, b { .. }    # b is the iterator.
```

This is because {...} binds more tightly to the preceding expression than does a `do` block. The first example is equivalent to `foobar(a, b) do...`, while the second is `foobar(a, b {...})`.

5.2 How can I pass a block to an iterator?

You simply place the block after the iterator call. You can also pass a `Proc` object by prepending & to the variable or constant name that refers to the `Proc`.

5.3 How is a block used in an iterator?

There are three ways to execute a block from an iterator method: (1) the `yield` control structure; (2) calling a `Proc` argument (made from a block) with `call`; and (3) using `Proc.new` followed by a call.

The `yield` statement calls the block, optionally passing it one or more arguments.

```
def myIterator
  yield 1,2
end
myIterator { |a,b| puts a, b }

Produces:

1
2
```

If a method definition has a block argument (the last formal parameter has an ampersand (&) prepended), it will receive the attached block, converted to a `Proc` object. This may be called using `method.call(args...)`.

```
def myIterator(&b)
  b.call(2,3)
end
myIterator { |a,b| puts a, b }

Produces:

2
3
```

`Proc.new` (or the equivalent `proc` or `lambda` calls), when used in an iterator definition, takes the block which is given to the method as its argument, generates a procedure object from it. (`proc` and `lambda` are effectively synonyms.)

```
def myIterator
  Proc.new.call(3,4)
  proc.call(4,5)
  lambda.call(5,6)
end
myIterator { |a,b| puts a, b }

Produces:

3
4
4
5
5
6
```

Perhaps surprisingly, `Proc.new` and friends do not in any sense *consume* the block attached to the method–each call to `Proc.new` generates a new procedure object out of the same block.

You can tell if there is a block associated with a method by calling `block_given?`.

5.4 What does `Proc.new` without a block do?

`Proc.new` without a block cannot generate a procedure object and an error occurs. In a method definition, however, `Proc.new` without a block implies the existence of a block at the time the method is called, and so no error will occur.

5.5 How can I run iterators in parallel?

Matz's solution, in [ruby-talk:5252], uses threads:

```
require 'thread'

def combine(*args)
  queues = []
  threads = []
  for it in args
    queue = SizedQueue.new(1)
    th = Thread.start(it, queue) do |i,q|
      self.send(it) do |x|
        q.push x
      end
    end
    queues.push queue
    threads.push th
  end
  loop do
    ary = []
    for q in queues
      ary.push q.pop
    end
    yield ary
    for th in threads
      return unless th.status
    end
  end
end
public :combine

def it1 ()
  yield 1; yield 2; yield 3
end

def it2 ()
  yield 4; yield 5; yield 6
end

combine('it1','it2') do |x|
  # x is (1, 4), then (2, 5), then (3, 6)
end
```

6 Syntax

6.1 What does `:var` mean?

A colon followed by a name generates an integer(Fixnum) called a *symbol* which corresponds one to one with the identifier. `"var".intern` gives the same integer as `:var`, but the ":" form will create a local symbol if it doesn't already exist.

The routines "catch", "throw", "autoload", and so on, require a string or a symbol as an argument.

"method_missing", "method_added" and "singleton_method_added" (and others) require a symbol.

The fact that a symbol springs into existence the first time it is referenced is sometimes used to assign unique values to constants:

```
NORTH = :NORTH
SOUTH = :SOUTH
EAST  = :EAST
WEST  = :WEST
```

6.2 How can I access the value of a symbol?

To get the value of the variable corresponding to a symbol, you can use `id2name` to get the name of the variable, and then `eval` that to get that variable's contents. In the scope of "symbol", do `eval(:symbol.id2name)`.

```
a = 'This is the content of "a"'
b = eval(:a.id2name)
a.id == b.id  # b now references the same object as a
```

If your symbol corresponds to the name of a method, you can use the `Method.method` function to return a corresponding `Method` object, which you may then call.

```
class Demo
  def meth
    "Hello, world"
  end
end

d = Demo.new         # -> #<Demo:0x4018d554>
m = d.method(:meth)  # -> #<Method: Demo(Demo)#meth>
m.call               # -> "Hello, world"
```

6.3 Is `loop` a control structure?

Although `loop` looks like a control structure, it is actually a method defined in `Kernel`. The block which follows introduces a new scope for local variables.

6.4 Ruby doesn't have a post-test loop

Q: Ruby does not have a `do { ... } while` construct, so how can I implement loops that test the condition at the end.

Clemens Hintze says: You can use a combination of Ruby's `begin ... end` and the `while` or `until` statement modifiers to achieve the same effect:

```
i = 0
begin
  puts "i = #{i}"
  i += 1
end until i > 4

Produces:

i = 0
i = 1
i = 2
i = 3
i = 4
```

6.5 `a +b` gives an error!

Ruby works hard to distinguish method calls from operators, and variable names from method names. Unfortunately, there's no way it can get it right all the time. In this case, "`a +b`" is parsed as "`a(+b)`". Remove the space to the left of "+" or add a space to the right of "+," and it will be parsed as an addition.

6.6 `s = "x"; puts s *10` gives an error.

Again, Ruby sees the asymmetrical space and parses it as `puts(s(*10))` (which isn't too smart, really). Use "`s*10`" or "`s * 10`" to get the desired result.

6.7 Why can't I pass a hash literal to a method: `p {}`?

The {} is parsed as a block, not a Hash constructor. You can force the {} to be treated as an expression by making the fact that it's a parameter explicit: `p({})`.

6.8 I can't get `def pos=(val)` to work.

I have the following code, but I cannot use the method `pos = 1`.

```
def pos=(val)
   print @pos, "\n"
   @pos = val
end
```

Methods with = appended must be called with a receiver (without the receiver, you're just assigning to a local variable). Invoke it as `self.pos = 1`.

6.9 What is the difference between `'\1'` and `'\\1'`?

They have the same meaning. In a single quote string, only `\'` and `\\` are transformed and other combinations remain unchanged.

However, in a doubled quoted string, "\1" is the byte \001, while "\\1" is the two character string containing a backslash and the character "1".

6.10 What's the difference between "or" and "||"?

Q: "p(nil || "Hello")" prints "Hello", while "p(nil or "Hello")" gives a parse error.

A: || combines terms within an expression. Because the first term in this case is nil, the second term is evaluated.

or is used to combine expressions in conditionals. Ruby is not expecting a conditional statement in an argument list.

7 Methods

7.1 How does Ruby choose which method to invoke?

Ruby binds all messages to methods dynamically. It searches first for singleton methods in the receiver, then for methods defined in the receiver's own class, and finally for methods defined in the receiver's superclasses (including any modules which may have been mixed in). You can see the order of searching by displaying Classname.ancestors, which shows the ancestor classes and modules of ClassName.

If after searching the alternatives a matching method could not be found, Ruby tries to invoke a method called method_missing, repeating the same search procedure to find it. This allows you to handle messages to unknown methods, and is often used to provide dynamic interfaces to classes.

```
    module Indexed
      def [](n)
        to_a[n]
      end
    end
    class String
      include Indexed
    end
    String.ancestors                # -> [String, Indexed, Enumerable, Comparable, Ob-
ject, Kernel]
    "abcde".gsub!(/./, "\\&\n")[1]  # -> 10
```

This program does not return "b\n" as one expects, but returns 10. When the method [] is searched for, it is found in class String, before searching Indexed. You should directly redefine [] in class String.

7.2 Are +, -, * ... operators?

+, -, and the like are not operators but method calls. They can, therefore, be overloaded by new definitions.

```
    class MyString < String
      def -(other)            # New method
        self[0...other.size]  # self truncated to other's size
      end
    end
```

However, the following are built-in control structures, not methods, which cannot be overridden.

```
    =, .., ..., !, not, ||, &&, and, or, ::
```

To overload or to define unary operators, you can use `+@` and `-@` as the method names.

`=` is used to define a method to set an attribute of the object:

```
class Test
  def attribute=(val)
    @attribute = val
  end
end
t = Test.new
t.attribute = 1
```

If operators such as `+` and `-` are defined, Ruby automatically handles the self assignment forms (`+=`, `-=` and so on).

7.3 Where are ++ and - -?

Ruby does not have the autoincrement and autodecrement operators. You can use `+= 1` and `-= 1` instead.

7.4 All these objects are fine, but does Ruby have any simple functions?

Yes and no. Ruby has methods that *look* like functions in languages such as C or Perl:

```
def writeln(str)
  print(str, "\n")
end

writeln("Hello, World!")

Produces:

Hello, World!
```

However, they're actually method calls with the receiver omitted. In this case, Ruby assumes the receiver is `self`.

So, `writeln` resembles a function but it's actually a method belonging to class `Object` and sent as a message to the hidden receiver `self`. Ruby is a pure object-oriented language..

Of course you can use such methods as if they were functions.

7.5 So where do all these function-like methods come from?

All classes in Ruby are derived from class `Object`. The definition of class `Object` mixes-in the methods defined in the `Kernel` module. These methods are therefore available within every object in the system.

Even if you're writing a simple Ruby program without classes, you're actually working inside class `Object`.

7.6 Can I access an object's instance variables?

An object's instance variables (those variables starting with `@`) are not directly accessible outside the object. This promotes good encapsulation. However, Ruby makes it easy for you to define accessors to these instance variables in such a way that users of your class can treat instance variables just like attributes. Just use one or more of `Module.attr`, `attr_reader`, `attr_writer`, or `attr_accessor`.

```
class Person
  attr           :name           # read only
  attr_accessor  :wearing_a_hat  # read/write
  def initialize(name)
    @name = name
  end
end

p = Person.new("Dave")
p.name           # -> "Dave"
p.wearing_a_hat  # -> nil
p.wearing_a_hat = true
p.wearing_a_hat  # -> true
```

You can also define your own accessor functions (perhaps to perform validation, or to handle derived attributes). The read accessor is simply a method that takes no parameters, and the assignment accessor is a method name ending in `=` that takes a single parameter. Although there can be no space between the method name and the `=` in the method definition, you can insert spaces there when you *call* the method, making it look like any other assignment. You can also utilize self assignments such as `+=` and `-=`, as long as the corresponding `+` or `-` methods are defined.

7.7 What's the difference between `private` and `protected`?

The visibility keyword `private` makes a method callable only in a function form, and so it can only have `self` as a receiver. A private method is callable only within the class in which the method was defined or in its subclasses.

```
class Test
  def func
    return 99
  end
  def test(other)
    p func
    p other.func
  end
end
t1 = Test.new
t2 = Test.new

t1.test(t2)

# Now make 'func' private

class Test
  private :func
end

t1.test(t2)

Produces:

99
99
99
prog.rb:7:in 'test': private method 'func' called for #<Test:0x4018d400> (NameError)
        from prog.rb:21
```

Protected methods are also callable only from within their own class or its subclasses, but they can be called both as functions form and using a receiver. For example,

```
def <=>(other)
  age <=> other.age
end
```

Will compile if `age` is a protected method, but not if it is private.

These features help you control access to your class's internals.

7.8 How can I change the visibility of a method?

You change the visibility of methods using `private`, `protected` and `public`. When used without parameters during a class definition, they affect the visibility of subsequent methods. When used with parameters, they change the visibility of the named methods.

```
class Foo
  def test
    print "hello\n"
  end
  private :test
end

foo = Foo.new
foo.test

Produces:

prog.rb:9: private method `test' called for #<Foo:0x4018d7e8> (NameError)
```

You can make a class method private using `private_class_method`.

```
class Foo
  def Foo.test
    print "hello\n"
  end
  private_class_method :test
end

Foo.test

Produces:

prog.rb:8: private method `test' called for Foo:Class (NameError)
```

The default visibility for the methods defined in a class is public. The exception is the instance initializing method, `initialize`.

Methods defined at the toplevel are also public by default.

7.9 Can an identifier beginning with a capital letter be a method name?

Yes, you can, but we don't do it lightly! If Ruby sees a capitalized name followed by a space, it will probably (depending on the context) assume it's a constant, not a method name. So, if you use capitalized method names, always remember to put parameter lists in parentheses, and always put the parentheses next to the method name with no intervening spaces. (This last suggestion is a good idea anyway!)

7.10 Calling `super` gives an `ArgumentError`.

Invoking `super` with no parameters in a method passes all the arguments of that method to a method of the same name in a superclass. If the number of arguments to the original method disagrees with that of the higher-level method, an `ArgumentError` is raised. To get around this, simply call `super` and pass a suitable number of arguments.

7.11 How can I call the a method of the same name two levels up?

`super` invokes the same named method one level up. If you're overloading a method in a more distant ancestor, use `alias` to give it an new name before masking it with your method definition. You can then call it using that aliased name.

7.12 How can I invoke an original built-in method after redefining it?

Within the method definition, you can use `super`. You can also use `alias` to give it an alternative name. Finally, you can call the original method as a singleton method of `Kernel`.

7.13 What is a destructive method?

A destructive method is one which alters the state of an object. String, Array, and Hash, and others have such methods. Often there are two versions of a method, one with a plain name, the other with the same, but followed by `!`. The plain version takes a copy of the receiver, makes its change to it, and returns the copy. The version with the `!` modifies the receiver in place.

Beware, however, that there are a fair number of destructive methods that don't have an `!`, including assignment operators (`name=`), array assignment (`[]=`), and methods such as `Array.collect!`.

7.14 Why can destructive methods be dangerous?

Remember that assignment in most cases just copies object references, and that parameter passing is equivalent to assignment. This means you can end up with multiple variables referencing the same object. If one of those variables is used to invoke a destructive method, the object referenced by all of them will be changed.

```
def foo(str)
  str.sub!(/foo/, "baz")
end

obj = "foo"
foo(obj)         # -> "baz"
obj              # -> "baz"
```

In this case the actual argument is altered.

7.15 Can I return multiple values from a method?

Yes and no.

```
def m1
  return 1, 2, 3
end
def m2
  return [1, 2, 3]
end
m1              # -> [1, 2, 3]
m2              # -> [1, 2, 3]
```

So, only one *thing* is returned, but that thing can be an arbitrarily complex object. In the case of arrays, you can use multiple assignment to get the effect of multiple return values. For example:

```
def foo
  return 20, 4, 17
end

a, b, c = foo
a               # -> 20
b               # -> 4
c               # -> 17
```

8 Classes and modules

8.1 Can a class definition be repeated?

A class can be defined repeatedly. Each definition is added to the last definition. If a method is redefined, the former one is overridden and lost.

8.2 Are there class variables?

As of Ruby 1.5.3, there are. A variable prefixed with two at signs is a class variable, accessible within both instance and class methods of the class.

```
class CountEm
  @@children = 0
  def initialize
    @@children += 1
    @myNumber = @@children
  end
  def whoAmI
   "I'm child number #@myNumber (out of #@@children)"
  end
  def CountEm.totalChildren
    @@children
  end
end
```

```
  c1 = CountEm.new
  c2 = CountEm.new
  c3 = CountEm.new
  c1.whoAmI              # -> "I'm child number 1 (out of 3)"
  c3.whoAmI              # -> "I'm child number 3 (out of 3)"
  CountEm.totalChildren  # -> 3
```

Earlier versions of Ruby do not have class variables. However, container classes (`Array`, `Hash`, etc) assigned to a class constant can be used to give the same effect. This example uses an array. Some folks feel hashes are better.

```
class Foo
  F = [ 0 ]          # pseudo class variable - Array 'F'
  def foo
    F[0] += 1
    puts F[0]
  end
end
```

This reports on the number of times `foo` is called across *all* instances of class `Foo`.

8.3 What is a class instance variable?

```
class Foo
  @a = 123   # (1)
  def foo
    p @a     # (2) ... nil not 123
  end
end
```

(1) is a class instance variable, and (2) is an ordinary instance variable (which, not having been initialized, has a value of `nil`). (2) belongs to an instance of class `Foo`, and (1) belongs to the class object `Foo`, which is an instance of `Class` class. (phew!)

There is no way to access class instance variables from instance methods.

8.4 What is a singleton method?

A singleton method is an instance method associated with one specific object.

You create a singleton method by including the object in the definition:

```
class Foo
end

foo = Foo.new
bar = Foo.new
def foo.hello
  puts "Hello"
end
foo.hello
bar.hello

Produces:
```

```
Hello
prog.rb:10: undefined method ‘hello’ for #<Foo:0x4018d748> (NameError)
```

Singleton methods are useful when you want to add a method to an object and creating a new subclass is not appropriate.

8.5 Does Ruby have class methods?

A singleton method (see section 8.4 on the preceding page) of a class object is called a class method. (Actually, the class method is defined in the metaclass, but that is pretty much transparent). Another way of looking at it is to say that a class method is a method whose receiver is a class.

It all comes down to the fact that you can call class methods without having to have instances of that class (objects) as the receiver.

Let’s create a singleton method of class `Foo`:

```
class Foo
  def Foo.test
    "this is foo"
  end
end

#It is invoked this way.

Foo.test         # -> "this is foo"
```

In this example, `Foo.test` is a class method.

Methods which are defined in class `Class` can be used as class methods for every class(!)

8.6 What is a singleton class?

A Singleton class is an anonymous class that is created by subclassing the class associated with a particular object. They are another way of extending the functionality associated with just one object.

Take the lowly `Foo`:

```
class Foo        # -> hello«7»nil
  def hello
    print "hello"
  end
end

foo = Foo.new
foo.hello
```

Now let’s say we need to add class-level functionality to just this one instance:

```
class « foo
  attr :name, TRUE
  def hello
    "hello. I’m " +  @name + "\n"
```

```
  end
end

foo.name = "Tom"
foo.hello        # -> "hello. I'm Tom\n"
```

We've customized `foo` without changing the characteristics of `Foo`,

8.7 What is a module function?

A *module function* is a private, singleton method defined in a module. In effect, it is similar to a class method (see section 8.5 on the page before), in that it can be called using the `Module.method` notation:

```
Math.sqrt(2)     # -> 1.414213562
```

However, because modules can be mixed in to classes, module functions can also be used without the prefix (that's how all those `Kernel` functions are made available to objects):

```
include Math
sqrt(2)          # -> 1.414213562
```

Use `module_function` to make a method a module function.

```
module Test
  def thing
    # ...
  end
  module_function :thing
end
```

8.8 What is the difference between a class and a module?

Modules are collections of methods and constants. They cannot generate instances. Classes may generate instances (objects), and have per-instance state (instance variables).

Modules may be mixed in to classes and other modules. The mixed-in module's constants and methods blend into that class's own, augmenting the class's functionality. Classes, however, cannot be mixed in to anything.

A class may inherit from another class, but not from a module.

A module may not inherit from anything.

8.9 Can you subclass modules?

No. However, a module may be included in a class or another module to mimic multiple inheritance (the *mixin* facility).

This does not generate a subclass (which would require inheritance), but does generate an `is_a?` relationship between the class and the module.

8.10 Give me an example of a mix-in

The module `Comparable` provides a variety of comparison operators (<, <=, >, `between?` and so on). It defines these in terms of calls to the general comparison method, <=>. However, it does not itself define <=>.

Say you want to create a class where comparisons are based on the number of legs an animal has:

```
class MyClass
  include Comparable
  attr :legs
  def initialize(name, legs)
   @name, @legs = name, legs
  end
  def <=>(o)
    return @legs <=> o.legs
  end
end
c = MyClass.new('cat', 4)
s = MyClass.new('snake', 0)
p = MyClass.new('parrot', 2)

c < s          # -> false
s < c          # -> true
p >= s         # -> true
p.between?(s, c)  # -> true
[p, s, c].sort    # -> [snake, parrot, cat]
```

All `MyClass` must do is define its own semantics for the operator <=>, and mix-in the `Comparable` module. `Comparable`'s methods now become indistinguishable from `MyClass`'s and your class suddenly sprouts new functionality. And because the same `Comparable` module is used my many classes, your new class will share a consistent and well understood semantic.

8.11 Why are there two ways of defining class methods?

You can define a class method in the class definition, and you can define a class method at the top level?

```
class Demo
  def Demo.classMethod
  end
end

def Demo.anotherClassMethod
end
```

There is only one significant difference between the two. In the class definition you can refer to the class's constants directly, as the constants are within scope. At the top level, you have to use the `Class::CONST` notation.

8.12 What is the difference between `load` and `require`?

`load` will load and execute a Ruby program (`*.rb`).

`require` loads Ruby programs as well, but will also load binary Ruby extension modules (shared libraries or DLLs). In addition, `require` ensures that a feature is never loaded more than once.

8.13 What is the difference between `include` and `extend`?

`include` mixes a module into a class or another module. Methods from that the module are called function-style (without a receiver).

`extend` is used to include a module in an object(instance). Methods in the module become methods in the object.

8.14 What does `self` mean?

`self` is the currently executing receiver–the object to which a method is applied. A function-style method call implies `self` as the receiver.

8.15 Why can't I load variables from a separate file?

Say `file1` contains:

```
    var1 = 99
```

and some other file loads it in:

```
    require 'file1'
    puts var1

    Produces:

    prog.rb:2: undefined local variable or method 'var1' for #<Ob-
ject:0x4019ac90> (NameError)
```

You get an error because `load` and `require` arrange for local variables to be stored into a separate, anonymous namespace, effectively discarding them. This is designed to protect your code from being polluted.

9 Builtin libraries

9.1 What does `instance_methods(nil)` return?

The method `instance_methods` returns an array containing the names of methods that the receiver responds to. This will include the methods in superclasses and in mixed-in modules.

`instance_methods(nil)` returns the name of just those methods which are defined in the object's class.

9.2 How do random number seeds work?

It depends. In Ruby versions prior to 1.5.2, the random number generator had (by default) a constant seed, and so would produce the same series of numbers each time a program was run. If you needed less deterministic behaviors, you called `srand` to set up a less predictable seed.

Newer Rubys (Rubies?) have a different behavior. If `rand` is called without a prior call to `srand`, Ruby will generate its own random(ish) seed. Successive runs of a program that does not use `srand` will generate different sequences of random numbers. To get the old, predictable, behavior (perhaps for testing), call `srand` with a constant seed.

9.3 What is the difference between an immediate value and a reference?

`Fixnum`, `true`, `nil`, and `false` are implemented as immediate values. With immediate values, variables hold the objects themselves, rather than references to them.

Singleton methods cannot be defined for such objects. Two `Fixnum`s of the same value always represent the same object instance, so (for example) instance variables for the `Fixnum` with the value "one" are shared between all the "ones" is the system. This makes it impossible to define a singleton method for just one of these.

9.4 What is the difference between `nil` and `false`?

First the similarity. `nil` and `false` are the only two values that evaluate to false in a boolean context.

However, they are instances of different classes (`NilClass` and `FalseClass`), and have different behaviors elsewhere.

We recommend that predicate methods (those whose name ends with a question mark) return `true` or `false`. Other methods that need to indicate failure should return `nil`.

9.5 I read a file and changed it, but the file on disk has not changed.

```
open("example", "r+").readlines.each_with_index{|l, i|
  l[0,0] = (i+1).to_s + ": " }
```

This program does not add line numbers to the file "example". It does read the contents of the file, and for each line read prepend the line number, but the data is never written back. The code below *does* update the file (although somewhat dangerously, as it takes no backup before starting the update):

```
io = open("example", "r+")
ary = io.readlines
ary.each_with_index{|l, i| l[0,0] = (i+1).to_s + ": "}
io.rewind
io.print ary
io.close
```

9.6 How can I process a file and update its contents?

Using the command-line option `-i`, or built-in variable `$-i`, you can read a file and replace it.

The code in the preceding question, which added line numbers to file, is probably best written using this technique:

```
$ ruby -i -ne 'print "#$.: #$_"' example
```

If you want to preserve the original file, use `-i.bak` to create a backup.

9.7 I wrote a file, copied it, but the end of the copy seems to be lost.

This code will not work correctly:

```
open('file', 'w').print "This is a file.\n"
system 'cp file newfile'
```

Because I/O is buffered, `file` is being copied *before* its contents have been written to disk. `newfile` will probably be empty. However, when the program terminates, the buffers are flushed, and `file` has the expected content.

The problem doesn't arise if you close `file` before copying:

```
f = open('file', 'w')
f.print "This is a file.\n"
f.close
system "cp file newfile"
```

9.8 How can I get the line number in current input file?

As you read from a file, Ruby increments a line number counter in the global variable `$.`. This is also available using the `lineno` attribute of the `File` object.

The special constant `ARGF` is a file-like object that can be used to read all the input files specified on the command line (or standard input if there are no files). ARGF is used implicitly by code such as:

```
while gets
  print $_
end
```

In this case, `$.` will be the cumulative number of lines read across all input files. To get the line number in the current file, use

```
ARGF.file.lineno
```

You can also get the name of the current file using `ARGF.file.path`.

9.9 How can I use `less` to display my program's output?

I tried the following, but nothing came out:

```
f = open '|less', 'w'
f.print "abc\n"
```

That's because the program ends immediately, and `less` never gets a chance to see the stuff you've written to it, never mind to display it. Use `close` to wait until `less` ends.

```
f = open '|less', 'w'
f.print "abc\n"
f.close
```

9.10 What happens to a `File` object which is no longer referenced?

A `File` object which is no longer referenced becomes eligible for garbage collection. The file will be closed automatically when the `File` object is garbage collected.

9.11 I feel uneasy if I don't close a file.

There are at least four good ways of ensuring that you do close a file:

```
(1)  f = open "file"
     begin
       f.each {|l| print l}
     ensure
       f.close
     end

(2)  File.open("file") { |f|
       f.readlines.each { |l| print l }
     }

(3)  IO.foreach("file") {|l| print l}

(4)  IO.readlines("file").each {|l| print l}
```

9.12 How can I sort files by their modified time?

```
Dir.glob("*").sort{|a,b| File.mtime(b) <=> File.mtime(a)}
```

Although this works (returning a list in reverse chronological order) it isn't very efficient, as it fetches the files' modification times from the operating system on every comparison.

More efficiency can be bought with some extra complexity:

```
Dir.glob("*").collect! {|f| [File.mtime(f), f]}.
  sort{|a,b| b[0] <=> a[0]}.collect! {|e| e[1]}
```

9.13 How can I count the frequency of words in a file?

```
freq = Hash.new(0)
open("example").read.scan(/\w+/){|w| freq[w] += 1}
freq.keys.sort.each {|k| print k, "-", freq[k], "\n"}

Produces:

and-1
is-3
line-3
one-1
this-3
three-1
two-1
```

9.14 Why is an empty string not `false`?

Q: An empty string (`""`) returns `true` in a conditional expression! In Perl, it's `false`.

A: In Ruby, only `nil` and `false` are false in conditional contexts. This is a way of gaining speed–both `nil` and `false` have immediate values, so they can be tested without having to chase a reference to an object.

You can use `empty?`, compare the string to `""`, or compare `length` to `0` to find out if a string is empty.

9.15 How can I sort strings in alphabetical order?

If you want your strings to sort 'AAA', 'BBB', ..., 'ZZZ', 'aaa', 'bbb', then the built-in comparison will work just fine.

If you want to sort ignoring case distinctions, compare downcased versions of the strings in the sort block:

```
array = %w( z bB Bb BB bb Aa aA AA aa a )
puts array.sort { |a,b|  a.downcase <=> b.downcase }

Produces:

a
aa
AA
aA
Aa
bb
BB
bB
Bb
z
```

If you want to sort so that the 'A's and 'a's come together, but 'a' is considered greater than 'A' (so 'Aa' comes after 'AA' but before 'AB'), use:

```
puts array.sort { |a,b|
  (a.downcase <=> b.downcase).nonzero? || a <=> b
}

Produces:

a
AA
Aa
aA
aa
BB
Bb
bB
bb
z
```

9.16 What does `"abcd"[0]` return?

It returns the character code for "a", 97(Fixnum). You can express a character code as an integer constant by prefixing the character with a question mark, so `?a` is also 97(Fixnum).

9.17 How can I expand tabs to spaces?

If `a` holds the string to be expanded, you could use one of:

```
  1 while a.sub!(/(^[^\t]*)\t(\t*)/){$1+' '*(8-$1.size%8+8*$2.size)}
#or
  1 while a.sub!(/\t(\t*)/){' '*(8-$~.begin(0)%8+8*$1.size)}
#or
  a.gsub!(/([^\t]{8})|([^\t]*)\t/n){[$+].pack("A8")}
```

9.18 How can I escape a backslash in a regexp?

`Regexp.quote('\\')` escapes a backslash.

It gets trickier if you're using `sub` and `gsub`, Say you write `gsub(/\\/, '\\\\')`, hoping to replace each backslash with two. The second argument is converted to '\\' in syntax analysis. When the substitution occurs, the regular expression engine converts this to '\', so the net effect is to replace each single backslash with another single backslash. You need to write `gsub(/\\/, '\\\\\\')`!

However, using the fact that \& contains the matched string, you could also write `gsub(/\\/, '\&\&')`.

If you use the block form of `gsub`, i.e. `gsub(/\\/){'\\\\'}`, the string for substitution is analyzed only once (during the syntax pass) and the result is what you intended.

9.19 What is the difference between `sub` and `sub!`?

In `sub`, a copy of the receiver is generated, substituted and returned.

In `sub!`, the receiver is altered and returned if any match was found. Otherwise, `nil` is returned.

Methods like `sub!` are called destructive methods (see section 7.13 on page 28) which alter the attribute of the receiver. If there are two similar methods and one is destructive, the destructive one has a suffix !.

```
def foo(str)
    str = str.sub(/foo/, "baz")
end

obj = "foo"
foo(obj)         # -> "baz"
obj              # -> "foo"

def foo(str)
    str = str.sub!(/foo/, "baz")
end

foo(obj)         # -> "baz"
obj              # -> "baz"
```

9.20 Where does \Z match?

\Z matches just before the last \n if the string ends with a \n, otherwise it matches at the end of a string.

9.21 What is the difference between `"..."` and `"..."`?

`..` includes the right hand side in the range, `...` does not.

9.22 Does Ruby have function pointers?

A Proc object generated by `Proc.new`, `proc`, or `lambda` can be referenced from a variable, so that variable could be said to be a function pointer. You can also get references to methods within a particular object instance using `Object.method`.

9.23 What is the difference between `thread` and `fork`?

Ruby threads are implemented within the interpreter, while `fork` invokes the operating system to create a separately executing subprocess.

`Thread` and `fork` have following characteristics:

- `fork` is slow, `thread` is not.
- `fork` does not share the memory space.
- `thread` does not cause thrashing.
- `thread` works on DOS.
- when `thread` gets in a deadlock, the whole process stops.
- `fork` can take advantage of pauses waiting for I/O to complete, `thread` does not (at least not without some help).

You probably shouldn't mix `fork` and `thread`.

9.24 How can I use Marshal?

Marshal is used to store an object in a file or a string, and later reconstitute it. Objects may be stored using:

```
Marshal.dump obj [, io ] [, lev]
```

`io` is a writable IO object, `lev` designates the level to which objects are dereferred and stored. If `lev` levels of dereferring are done and object references still exist, then `dump` stores just the reference, not the object referenced. This is not good, as these referenced objects cannot be subsequently reconstructed.

If `io` is omitted, the marshaled objects are returned in a string.

You can load objects back using:

```
   obj = Marshal.load io
#or
   obj = Marshal.load str
```

where `io` is a readable IO object, `str` is the dumped string.

9.25 Does Ruby have exception handling?

Ruby supports a flexible exception handling scheme:

```
begin
  statements which may raise exceptions.
rescue [exception class names]
  statements when an exception occurred.
rescue [exception class names]
  statements when an exception occurred.
ensure
  statements that will always run
end
```

If an exception occurs in the `begin` clause, the `rescue` clause with the matching exception name is executed. The `ensure` clause is executed whether an exception occurred or not. `rescue` and `ensure` clauses may be omitted.

If no exception class is designated for `rescue` clause, `StandardError` exception is implied, and exceptions which are in a is_a? relation to `StandardError` are captured.

This expression returns the value of the begin clause.

The latest exception is accessed by the global variable `$!` (and so its type can be determined using `$!.type`).

9.26 How can I use `trap`?

`trap` associates code blocks with external events (signals).

```
trap("PIPE") {raise "SIGPIPE"}
```

10 Extension library

10.1 How can I use Ruby interactively?

You can try using `irb`. The following is paraphrased from Goto Kentaro (Gotoken), and originally appeared in ruby-talk:444.

1. Get the latest tarball of irb from *the contrib directory* `<ftp://ftp.netlab.co.jp/pub/lang/ruby/contrib/>` in the Ruby archive.
2. Extract the `irb` directory tree.
3. Add the location of the `irb/` directory to the $RUBYLIB environment variable
4. Make a symbolic link from $RUBYLIB/irb/irb.rb to a file called `irb` somewhere in your path.
5. `chmod +x $RUBYLIB/irb/irb.rb`
6. Possibly use `rehash` to tell your login shell about the new command.
7. Type `irb`

If the `readline` extension module works with your interpreter, it makes `irb` a lot more fun to use.

There is also a simple program, `eval`, in the samples/ directory of the Ruby distribution. It lets you enter expressions and view their values. You can copy `eval` into the `site_ruby` directory in the Ruby tree, and then invoke it using:

```
ruby -r eval -e0
```

10.2 Is there a debugger for Ruby?

There is a gdb-like debugger for Ruby.

```
ruby -r debug your_program
```

10.3 How can I use a library written in C from Ruby?

Of all the scripting languages, Ruby is probably the easiest to extend. There are no problems with reference counting and variable types, and very few interfaces to learn. In fact, C code used to extend Ruby often ends up looking surprisingly like Ruby code itself.

First, get the Ruby source distribution and read README.EXT. This is a good document, not only if you're writing an extension library, but also if you want to understand Ruby more deeply.

Next, have a look at the source of the interpreter itself, and at the various supplied extensions in the `ext/` directory. You'll also find good examples under `contrib/` on the Ruby ftp sites.

10.4 Can I use Tcl/Tk interface in Ruby?

There are two interfaces to Tcl/Tk included in the standard distribution. One is under `ext/tcltk/` and loaded with `require "tcltk"`. The syntax is very close to that `Tcl`, which is passed to `Tcl` interpreter. Unfortunately, the description for this library is written in Japanese.

The other is under `ext/tk/` and loaded with `require "tk"`. Its syntax closer to the style of the Tk interface provided by the Perl and Python interfaces.

10.5 Tk won't work.

Your Tk version may be old, try a newer version.

10.6 Can I use `gtk+` or `xforms` interfaces in Ruby?

You'll find `ruby-gtk-x.xx.tar.gz` and `ruby-forms-x.x.tar.gz` under `contrib/` in ftp sites.

10.7 How can I do date arithmetic?

A Time object can express only the dates between Jan 1, 1970 and Jan 19, 2038.

Two standard extension library modules are provided: `require "date"`, which is simple and uses the English calendar, and `require "date2"`, which is more general purpose.

Also see `sample/cal.rb`.

11 Other Features

11.1 What does `a ? b : c` mean?

It's the same as saying `if a then b else c end`.

11.2 How can I count the number of lines in a file?

Assuming that the file ends in a linefeed, the following code may give the fastest result.

```
open("example").read.count("\n")  # -> 3
```

11.3 What do `begin` and `end` of MatchingData return?

They act with $, and return the start index and the end index of the matched data ($0) in the original string. See an example in tab expansion (see section 9.17 on page 38).

11.4 How can I sum the elements in an array?

Rather than solve the specific problem, let's solve the general case. The first thing we'll do is produce a method that will iterate over an `Enumerable` object and collect a single result. Smalltalk calls that method `inject`, so we will too:

```
module Enumerable

  # inject(n) { |n, i| ...}
  def inject(n)
    each { |i|
      n = yield(n, i)
    }
    n
  end
end
```

Notice how we've added the method to `Enumerable`. This means that anything that includes `Enumerable` can now use inject. But how do we use it? It takes a single argument 'n' and a block. For each element in the thing being enumerated, it calls the block, passing in 'n' and the element itself. The result of the block is assigned back to 'n'. So, to define `sum`, we could write:

```
module Enumerable
  def sum
    inject(0) {|n, i|  n + i }
  end
end

[1,3,5,7,9].sum  # -> 25
(1..100).sum     # -> 5050
```

11.5 How can I use continuations?

Ruby's continuations allow you to create an object representing a place in a Ruby program, and then return to that place at any time (even if it has apparently gone out of scope). Continuations can be used to implement complex control structures, but are typically more useful as ways of confusing people.

In [ruby-talk:4482], Jim Weirich posted the following examples of continuations:

```
# ------------------------------------------
# Simple Producer/Consumer
# ------------------------------------------
# Connect a simple counting task and a printing task together using
# continuations.
#
# Usage:  count(limit)

def count_task(count, consumer)
  (1..count).each do
    |i|
    callcc {|cc| consumer.call cc, i }
  end
  nil
end

def print_task()
  producer, i = callcc { |cc| return cc }
  print "#{i} "
  callcc { |cc| producer.call }
end

def count(limit)
  count_task(limit, print_task())
  print "\n"
end

# ------------------------------------------
# Filtering Out Multiples of a Given Number
# ------------------------------------------
# Create a filter that is both a consumer and producer.  Insert it
# between the counting task and the printing task.
#
# Usage:  omit (2, limit)

def filter_task(factor, consumer)
  producer, i = callcc { |cc| return cc }
  if (i%factor) != 0 then
    callcc { |cc| consumer.call cc, i }
  end
  producer.call
end

def omit(factor, limit)
  printer = print_task()
  filter = filter_task(factor, printer)
  count_task(limit, filter)
  print "\n"
end

# ------------------------------------------
# Prime Number Generator
# ------------------------------------------
# Create a prime number generator.  When a new prime number is
# discovered, dynamically add a new multiple filter to the chain of
```

```
# producers and consumers.
#
# Usage:  primes (limit)

def prime_task(consumer)
  producer, i = callcc { |cc| return cc }
  if i >= 2 then
    callcc { |cc| consumer.call cc, i }
    consumer = filter_task(i, consumer)
  end
  producer.call
end

def primes(limit)
  printer = print_task()
  primes = prime_task(printer)
  count_task(limit, primes)
  print "\n"
end
```