Ruby Programming

Wikibooks.org

December 1, 2012

Contents

1 Overview

Ruby is an object-oriented[1] scripting language[2] developed by Yukihiro Matsumoto[3] ("Matz"). The main web site for Ruby is ruby-lang.org[4]. Development began in February 1993 and the first alpha version of Ruby was released in December 1994. It was developed to be an alternative to scripting languages such as Perl[5] and Python[6].[7] Ruby borrows heavily from Perl and the class library is essentially an object-oriented reorganization of Perl's functionality. Ruby also borrows from Lisp[9] and Smalltalk[10]. While Ruby does not borrow many features from Python, reading the code for Python helped Matz develop Ruby.[11]

Mac OS X[13] comes with Ruby already installed. Most Linux[14] distributions either come with Ruby preinstalled or allow you to easily install Ruby from the distribution's repository of free software[15]. You can also download and install Ruby on Windows[16]. The more technically adept can download the Ruby source code[17] and compile it for most operating systems[19], including Unix[20], DOS[21], BeOS[22], OS/2[23], Windows, and Linux.[24]

1.1 Features

Ruby combines features from Perl, Smalltalk, Eiffel[26], Ada[27], Lisp, and Python.[28]

---

1 http://en.wikipedia.org/wiki/Object-oriented%20programming
2 http://en.wikipedia.org/wiki/Scripting%20programming%20language
3 http://en.wikipedia.org/wiki/Yukihiro%20Matsumoto
4 http://www.ruby-lang.org/
5 http://en.wikipedia.org/wiki/Perl
6 http://en.wikipedia.org/wiki/Python%20programming%20language
7 An Interview with the Creator of Ruby [8]. O'Reilly . Retrieved 2006-09-11
9 http://en.wikipedia.org/wiki/Lisp
10 http://en.wikipedia.org/wiki/Smalltalk
11 An Interview with the Creator of Ruby [12]. O'Reilly . Retrieved 2006-09-11
13 http://en.wikipedia.org/wiki/Mac%20OS%20X
14 http://en.wikipedia.org/wiki/Linux
15 http://en.wikipedia.org/wiki/free%20software
16 http://en.wikipedia.org/wiki/Microsoft%20Windows
17 Download Ruby [18]. . Retrieved 2006-09-11
19 http://en.wikipedia.org/wiki/operating%20system
20 http://en.wikipedia.org/wiki/Unix
21 http://en.wikipedia.org/wiki/DOS
22 http://en.wikipedia.org/wiki/BeOS
23 http://en.wikipedia.org/wiki/OS%2F2
24 About Ruby [25]. . Retrieved 2006-09-11
26 http://en.wikipedia.org/wiki/Eiffel%20%28programming%20language%29
27 http://en.wikipedia.org/wiki/Ada%20%28programming%20language%29
28 About Ruby [29]. . Retrieved 2006-09-11

1.1.1 Object Oriented

Ruby goes to great lengths to be a purely object oriented language. Every value in Ruby is an object, even the most primitive things: strings, numbers and even `true` and `false`. Every object has a *class* and every class has one *superclass*. At the root of the class hierarchy is the class `Object`, from which all other classes inherit.

Every class has a set of *methods* which can be called on objects of that class. Methods are always called on an object — there are no "class methods", as there are in many other languages (though Ruby does a great job of faking them).

Every object has a set of *instance variables* which hold the state of the object. Instance variables are created and accessed from within methods called on the object. Instance variables are completely private to an object. No other object can see them, not even other objects of the same class, or the class itself. All communication between Ruby objects happens through methods.

1.1.2 Mixins

In addition to classes, Ruby has *modules*. A module has methods, just like a class, but it has no instances. Instead, a module can be included, or "mixed in," to a class, which adds the methods of that module to the class. This is very much like inheritance but far more flexible because a class can include many different modules. By building individual features into separate modules, functionality can be combined in elaborate ways and code easily reused. Mix-ins help keep Ruby code free of complicated and restrictive class hierarchies.

1.1.3 Dynamic

Ruby is a very *dynamic* programming language. Ruby programs aren't compiled, in the way that C or Java programs are. All of the class, module and method definitions in a program are built by the code when it is run. A program can also modify its own definitions while it's running. Even the most primitive classes of the language like String and Integer can be opened up and extended. Rubyists call this *monkey patching* and it's the kind of thing you can't get away with in most other languages.

Variables in Ruby are dynamically typed, which means that any variable can hold any type of object. When you call a method on an object, Ruby looks up the method by name alone — it doesn't care about the type of the object. This is called *duck typing* and it lets you make classes that can pretend to be other classes, just by implementing the same methods.

1.1.4 Singleton Classes

When I said that every Ruby object has a class, I lied. The truth is, every object has *two* classes: a "regular" class and a *singleton class*. An object's singleton class is a nameless class whose only instance is that object. Every object has its very own singleton class, created automatically along with the object. Singleton classes inherit from their object's regular class and are initially empty, but you can open them up and add methods to them, which can then be called on the lone object belonging to them. This is Ruby's secret trick to avoid "class methods" and keep its type system simple and elegant.

1.1.5 Metaprogramming

Ruby is so object oriented that even classes, modules and methods are themselves objects! Every class is an instance of the class `Class` and every module is an instance of the class `Module`. You can call their methods to learn about them or even modify them, while your program is running. That means that you can use Ruby code to generate classes and modules, a technique known as *metaprogramming*. Used wisely, metaprogramming allows you to capture highly abstract design patterns in code and implement them as easily as calling a method.

1.1.6 Flexibility

In Ruby, everything is malleable. Methods can be added to existing classes without subclassing[30], operators can be overloaded[31], and even the behavior of the standard library can be redefined at runtime.

1.1.7 Variables and scope

You do not need to declare variables or variable scope in Ruby. The name of the variable automatically determines its scope.

- x is local variable (or something besides a variable).
- $x is a global variable.
- @x is an instance variable.
- @@x is a class variable.

1.1.8 Blocks

Blocks are one of Ruby's most unique and most loved features. A block is a piece of code that can appear after a call to a method, like this:

```
laundry_list.sort do |a,b|
  a.color <=> b.color
end
```

The block is everything between the `do` and the `end`. The code in the block is not evaluated right away, rather it is packaged into an object and passed to the `sort` method as an argument. That object can be called at any time, just like calling a method. The `sort` method calls the block whenever it needs to compare two values in the list. The block gives you a lot of control over how `sort` behaves. A block object, like any other object, can be stored in a variable, passed along to other methods, or even copied.

Many programming languages support code objects like this. They're called *closures* and they are a very powerful feature in any language, but they are typically underused because the code to create

30 http://en.wikipedia.org/wiki/Subclass%20%28computer%20science%29
31 http://en.wikipedia.org/wiki/Operator%20overloading

them tends to look ugly and unnatural. A Ruby block is simply a special, clean syntax for the common case of creating a closure and passing it to a method. This simple feature has inspired Rubyists to use closures extensively, in all sorts of creative new ways.

1.1.9 Advanced features

Ruby contains many advanced features.

- Exceptions[32] for error-handling.
- A mark-and-sweep garbage collector[33] instead of reference counting[34].
- OS-independent threading[35], which allows you to write multi-threaded applications even on operating systems such as DOS. (this feature will disappear in 1.9[36], which will use native threads)

You can also write extensions to Ruby in C[37] or embed Ruby in other software.

1.2 References

32 http://en.wikipedia.org/wiki/Exception%20handling
33 http://en.wikipedia.org/wiki/Garbage%20collection%20%28computer%20science%29
34 http://en.wikipedia.org/wiki/Reference%20counting
35 http://en.wikipedia.org/wiki/Thread%20%28computer%20science%29
36 http://en.wikipedia.org/wiki/YARV%20
37 http://en.wikipedia.org/wiki/C%20%28programming%20language%29

2 Installing Ruby

Ruby[1] comes preinstalled on Mac OS X[2] and many Linux[3] distributions. In addition, it is available for most other operating systems[4], including Microsoft Windows[5].

To find the easiest way to install Ruby for your system, follow the directions below. You can also install Ruby by compiling the source code, which can be downloaded from the Ruby web site[6].

2.1 Operating systems

2.1.1 Mac OS X

Ruby comes preinstalled on Mac OS X. To check what version is on your system:

1. Launch the Terminal[7] application, which is located in the "Utilities" folder, under "Applications".
2. At the command-line, enter: `ruby -v`

If you want to install a more recent version of Ruby, you can:

- Buy the latest version of Mac OS X, which may have a more recent version of Ruby.
- Install Ruby using RVM[8]. (This is the most popular way because you can manage ruby versions and install many other ruby packages)
- Install Ruby using Fink[9].
- Install Ruby using MacPorts[10].
- Install Ruby using Homebrew[11].

2.1.2 Linux

Ruby comes preinstalled on many Linux systems. To check if Ruby is installed on your system, from the shell run: `ruby -v`

---

1 http://en.wikipedia.org/wiki/Ruby%20%28programming%20language%29
2 http://en.wikipedia.org/wiki/Mac%20OS%20X
3 http://en.wikipedia.org/wiki/Linux
4 http://en.wikipedia.org/wiki/operating%20system
5 http://en.wikipedia.org/wiki/Microsoft%20Windows
6 http://www.ruby-lang.org/en/downloads/
7 http://en.wikipedia.org/wiki/Terminal%20%28application%29
8 http://beginrescueend.com/
9 http://en.wikipedia.org/wiki/Fink
10 http://en.wikipedia.org/wiki/MacPorts
11 http://en.wikipedia.org/wiki/Homebrew_%28package_management_software%29

If ruby is not installed, or if you want to upgrade to the latest version, you can usually install Ruby from your distribution's software repository. Directions for some distributions are described below.

Debian / Ubuntu

On Debian[12] and Ubuntu[13], install Ruby using either the graphical tool Synaptic[14] (on Debian, only if it is installed; it is included with Ubuntu) or the command-line tool apt[15].

Fedora Core

If you have Fedora Core[16] 5 or later, you can install Ruby using the graphical tool Pirut.[17] Otherwise, you can install Ruby using the command-line tool yum[19].

Arch Linux

If you have Arch Linux[20] you can install Ruby using the command-line tool pacman[21].

Mandriva Linux

On Mandriva Linux[22], install Ruby using the command-line tool urpmi[23].

PCLinuxOS

On PCLinuxOS[24], install Ruby using either the graphical tool Synaptic or the command-line tool apt.

Red Hat Linux

On Red Hat Linux[25], install Ruby using the command-line tool RPM[26].

---

12 http://en.wikipedia.org/wiki/Debian
13 http://en.wikipedia.org/wiki/Ubuntu%20%28Linux%20distribution%29
14 http://en.wikipedia.org/wiki/Synaptic%20Package%20Manager
15 http://en.wikipedia.org/wiki/Advanced%20Packaging%20Tool
16 http://en.wikipedia.org/wiki/Fedora%20Core
17 yum [18]. Fedora Wiki . Retrieved 2006-09-13
19 http://en.wikipedia.org/wiki/Yellow%20dog%20Updater%2C%20Modified
20 http://en.wikipedia.org/wiki/Arch%20Linux
21 http://en.wikipedia.org/wiki/Pacman%20%28package%20manager%29
22 http://en.wikipedia.org/wiki/Mandriva%20Linux
23 http://en.wikipedia.org/wiki/urpmi
24 http://en.wikipedia.org/wiki/PCLinuxOS%20
25 http://en.wikipedia.org/wiki/Red%20Hat%20Linux
26 http://en.wikipedia.org/wiki/RPM%20Package%20Manager

2.1.3 Windows

Ruby does not come preinstalled with any version of Microsoft Windows[27]. However, there are several ways to install Ruby on Windows.

- Download and install one of the compiled Ruby binaries from the Ruby web site[28].
- Download and run the one click RubyInstaller[29].
- Install Cygwin[30], a collection of free software[31] tools available for Windows. During the install, make sure that you select the "ruby" package, located in the "Devel, Interpreters" category.

Windows is slow

Currently Ruby on windows is a bit slow. Ruby isn't optimized for windows, because most core developers use Linux. Though 1.9.2 passes almost all core tests on windows.

Most of today's slowdown is because when ruby does a

```
require 'xxx'
```

it searches over its entire load path, looking for a file named xxx, or named xxx.rb, or xxx.so or what not. In windows, doing file stat's like that are expensive, so requires take a longer time in windows than linux.

1.9 further complicates the slowdown problem by introducing gem_prelude, which avoids loading full rubygems (a nice speedup actually), but makes the load path larger, so doing require's on windows now takes forever. To avoid this in 1.9.2, you can do a

```
require 'rubygems'
```

which reverts to typical load behavior.

If you want to speed it up (including rails) you can use

`http://github.com/rdp/faster_require`

Which have some work arounds to make loading faster by caching file locations.

Also the "rubyinstaller" (mingw) builds are faster than the old "one click" installers If yours comes from rubyinstaller.org, chances are you are good there.

NB that Jruby tends to run faster[32] but start slower, on windows, than its MRI cousins. Rubinius is currently not yet windows compatible.

27 http://en.wikipedia.org/wiki/Microsoft%20Windows
28 http://www.ruby-lang.org/en/downloads/
29 http://rubyinstaller.org/
30 http://en.wikipedia.org/wiki/Cygwin
31 http://en.wikipedia.org/wiki/free%20software
32 http://betterlogic.com/roger/?p=2841

2.2 Building from Source

If your distro doesn't come with a ruby package or you want to build a specific version of ruby from scratch, please install it by following the directions here[33]. Download from here[34].

2.3 Compile options

2.3.1 Building with debug symbols

If you want to install it with debug symbols built in (and are using gcc--so either Linux, cygwin, or mingw).

```
./configure --enable-shared optflags="-O0" debugflags="-g3 -ggdb"
```

2.3.2 Optimizations

Note that with 1.9 you can pass it `--disable-install-doc` to have it build faster.

To set the GC to not run as frequently (which tends to provide a faster experience for larger programs, like rdoc and rails), precede your build with

```
$ export CCFLAGS=-DGC_MALLOC_LIMIT=80000000
```

though you might be able to alternately put those in as opt or debug flags, as well.

2.4 Testing Installation

The installation can be tested easily.

```
$ ruby -v
```

This should return something like the following:

```
ruby 1.8.7 (2009-06-12 patchlevel 174) [i486-linux]
```

If this shows up, then you have successfully installed Ruby. However, if you get something like the following:

33 http://svn.ruby-lang.org/repos/ruby/trunk/README
34 http://www.ruby-lang.org/en/downloads/

```
-bash: ruby: command not found
```

Then you did not successfully install Ruby.

2.5 References

3 Ruby editors

Although you can write Ruby programs with any plain text editor[1], some text editors have additional features to aid the Ruby programmer. The most common is syntax highlighting[2].

Here[3] is a spreadsheet of the various options available.

Here[4] is a stackoverflow list.

1 http://en.wikipedia.org/wiki/text%20editor
2 http://en.wikipedia.org/wiki/syntax%20highlighting
3 https://spreadsheets.google.com/ccc?key=0Al_hzYODcgxwdG9tUFhqcVVoUDVaLTlqT2YtNjV1N0E&hl=en#gid=1
4 http://stackoverflow.com/questions/59968/best-editor-or-ide-for-ruby

4 Notation conventions

4.1 Command-line examples

In this tutorial, examples that involve running programs on the command-line will use the dollar sign to denote the shell prompt. The part of the example that you type will appear **bold**. Since the dollar sign denotes your shell prompt, you should *not* type it in.

For example, to check what version of Ruby is on your system, run:

```
$ ruby -v
```

Again, do not type the dollar sign – you should only enter "`ruby -v`" (without the quotes). Windows users are probably more familiar seeing "`C:\>`" to denote the shell prompt (called the command prompt on Windows).

An example might also show the output of the program.

```
$ ruby -v
ruby 1.8.5 (2006-08-25) [i386-freebsd4.10]
```

In the above example, "`ruby 1.8.5 (2006-08-25) [i386-freebsd4.10]`" is printed out after you run "`ruby -v`". Your actual output when you run "`ruby -v`" will vary depending on the version of Ruby installed and what operating system[1] you are using.

4.1.1 Running Ruby scripts

For simplicity, the following convention is used to show a Ruby script being run from the shell prompt.

```
$ hello-world.rb
Hello world
```

However, the actual syntax that you will use to run your Ruby scripts will vary depending on your operating system and how it is setup. Please read through the Executable Ruby scripts[2] section of the ../Hello world/[3] page to determine the best way to run Ruby scripts on your system.

1 `http://en.wikipedia.org/wiki/operating%20system`
2 Chapter 7.3 on page 22
3 Chapter 7 on page 21

4.1.2 Running irb

Ruby typically installs with "interactive ruby" (irb) installed along with it. This is a REPL that allows you to experiment with Ruby, for example:

```
$ irb
>> 3 + 4
=> 7
>> 'abc'
=> "abc"
```

5 Interactive Ruby

When learning Ruby, you will often want to experiment with new features by writing short snippets of code. Instead of writing a lot of small text files, you can use `irb`, which is Ruby's interactive mode.

5.1 Running irb

Run `irb` from your shell prompt.

```
$ irb --simple-prompt
>>
```

The >> prompt indicates that `irb` is waiting for input. If you do not specify `--simple-prompt`, the `irb` prompt will be longer and include the line number. For example:

```
$ irb
irb(main):001:0>
```

A simple `irb` session might look like this.

```
$ irb --simple-prompt
>> 2+2
=> 4
>> 5*5*5
=> 125
>> exit
```

These examples show the user's input in **bold**. `irb` uses => to show you the return value[1] of each line of code that you type in.

5.1.1 Cygwin users

If you use Cygwin's[2] Bash[3] shell[4] on Microsoft Windows[5], but are running the native Windows version of Ruby instead of Cygwin's version of Ruby, read this section.

1 http://en.wikipedia.org/wiki/return%20value
2 http://en.wikipedia.org/wiki/Cygwin
3 http://en.wikipedia.org/wiki/Bash
4 http://en.wikipedia.org/wiki/Shell%20%28computing%29
5 http://en.wikipedia.org/wiki/Microsoft%20Windows

To run the native version of `irb` inside of Cygwin's Bash shell, run `irb.bat`.

By default, Cygwin's Bash shell runs inside of the Windows console[6], and the native Windows version of `irb.bat` should work fine. However, if you run a Cygwin shell inside of Cygwin's rxvt[7] terminal emulator[8], then `irb.bat` will not run properly. You must either run your shell (and `irb.bat`) inside of the Windows console or install and run Cygwin's version of Ruby.

5.2 Understanding irb output

`irb` prints out the return value of each line that you enter. In contrast, an actual Ruby program only prints output when you call an output method such as `puts`.

For example:

```
$ irb --simple-prompt
>> x=3
=> 3
>> y=x*2
=> 6
>> z=y/6
=> 1
>> x
=> 3
>> exit
```

Helpfully, `x=3` not only does an assignment, but also returns the value assigned to `x`, which `irb` then prints out. However, this equivalent Ruby program prints nothing out. The variables get set, but the values are never printed out.

```
x=3
y=x*2
z=y/6
x
```

If you want to print out the value of a variable in a Ruby program, use the `puts` method.

```
x=3
puts x
```

6 http://en.wikipedia.org/wiki/Win32%20console
7 http://en.wikipedia.org/wiki/rxvt
8 http://en.wikipedia.org/wiki/terminal%20emulator

6 Mailing List FAQ

6.0.1 Etiquette

There is a list of Best Practices[1].

6.0.2 What is the Best editor?

Several editors exist for Ruby.

Commercial: The editor of preference on OS X is Textmate. RubyMine has also received good reviewed[2]

Free: NetBeans offers a version with Ruby support. RadRails is a port of eclipse to support Ruby syntax. Eclipse has a plugin DLTK that offers ruby support[3]. On windows for rails projects there is RoRed.

See some more[4] questions answered.

1 http://blog.rubybestpractices.com/posts/jamesbritt/and_your_Mom_too.html
2 http://www.rubyinside.com/rubymine-1-0-ruby-ide-1818.html
3 http://www.infoq.com/news/2007/08/eclipse-dltk-09
4 http://wiki.github.com/rdp/ruby_tutorials_core/ruby-talk-faq

7 Basic Ruby - Hello world

The classic "hello world[1]" program is a good way to get started with Ruby.

7.1 Hello world

Create a text file called `hello-world.rb` containing the following code:

```
puts ,Hello world,
```

Now run it at the shell prompt.

```
$ ruby hello-world.rb
Hello world
```

You can also run the short "hello world" program without creating a text file at all. This is called a one-liner[2].

```
$ ruby -e "puts 'Hello world'"
Hello world
```

You can run this code with irb[3], but the output will look slightly different. `puts` will print out "`Hello world`", but `irb` will also print out the return value of `puts` — which is `nil`.

```
$ irb
>> puts "Hello world"
Hello world
=> nil
```

1 http://en.wikipedia.org/wiki/Hello%20world%20program
2 http://en.wikipedia.org/wiki/One-liner%20program
3 Chapter 5 on page 17

7.2 Comments

Like Perl[4], Bash[5], and C Shell[6], Ruby uses the hash symbol (also called Pound Sign, number sign[7], Square, or octothorpe) for comments[8]. Everything from the hash to the end of the line is ignored when the program is run by Ruby. For example, here's our `hello-world.rb` program with comments.

```
# My first Ruby program
# On my way to Ruby fame & fortune!

puts ,Hello world,
```

You can append a comment to the end of a line of code, as well. Everything before the hash is treated as normal Ruby code.

```
puts ,Hello world,                # Print out "Hello world"
```

You can also comment several lines at a time:

```
=begin
This program will
print "Hello world".
=end

puts ,Hello world,
```

Although block comments can start on the same line as `=begin`, the `=end` must have its own line. You cannot insert block comments in the middle of a line of code as you can in C[9], C++[10], and Java[11], although you can have non-comment code on the same line as the `=end`.

```
=begin This program will print "Hello world"
=end puts ,Hello world,
```

7.3 Executable Ruby scripts

Typing the word `ruby` each time you run a Ruby script is tedious. To avoid doing this, follow the instructions below.

4 http://en.wikipedia.org/wiki/Perl
5 http://en.wikipedia.org/wiki/Bash
6 http://en.wikipedia.org/wiki/C%20Shell
7 http://en.wikipedia.org/wiki/number%20sign
8 http://en.wikipedia.org/wiki/comment
9 http://en.wikipedia.org/wiki/C%20%28programming%20language%29
10 http://en.wikipedia.org/wiki/C%2B%2B
11 http://en.wikipedia.org/wiki/Java%20%28programming%20language%29

7.3.1 Unix-like operating systems

In Unix-like[12] operating systems[13] − such as Linux[14], Mac OS X[15], and Solaris[16] − you will want to mark your Ruby scripts as executable using the `chmod`[17] command. This also works with the Cygwin[18] version of Ruby.

```
$ chmod +x hello-world.rb
```

You need to do this each time you create a new Ruby script. If you rename a Ruby script, or edit an existing script, you do *not* need to run "`chmod +x`" again.

Next, add a shebang line[19] as the *very first line* of your Ruby script. The shebang line is read by the shell to determine what program to use to run the script. This line cannot be preceded by any blank lines or any leading spaces. The new `hello-world.rb` program − with the shebang line − looks like this:

```
#!/usr/bin/ruby

puts 'Hello world'
```

If your `ruby` executable is not in the `/usr/bin` directory, change the shebang line to point to the correct path[20]. The other common place to find the `ruby` executable is `/usr/local/bin/ruby`.

The shebang line is ignored by Ruby − since the line begins with a hash, Ruby treats the line as a comment. Hence, you can still run the Ruby script on operating systems such as Windows whose shell does not support shebang lines.

Now, you can run your Ruby script without typing in the word `ruby`. However, for security reasons, Unix-like operating systems do not search the current directory for executables unless it happens to be listed in your PATH environment variable[21]. So you need to do one of the following:

1. Create your Ruby scripts in a directory that is already in your PATH.
2. Add the current directory to your PATH (*not recommended*).
3. Specify the directory of your script each time you run it.

Most people start with #3. Running an executable Ruby script that is located in the current directory looks like this:

12 http://en.wikipedia.org/wiki/Unix-like
13 http://en.wikipedia.org/wiki/operating%20systems
14 http://en.wikipedia.org/wiki/Linux
15 http://en.wikipedia.org/wiki/Mac%20OS%20X
16 http://en.wikipedia.org/wiki/Solaris%20Operating%20System
17 http://en.wikipedia.org/wiki/chmod
18 http://en.wikipedia.org/wiki/Cygwin
19 http://en.wikipedia.org/wiki/shebang%20line
20 http://en.wikipedia.org/wiki/path%20%28computing%29
21 http://en.wikipedia.org/wiki/environment%20variable

```
$ ./hello-world.rb
```

Once you have completed a script, it's common to create a `~/bin` directory, add this to your PATH, and move your completed script here for running on a day-to-day basis. Then, you can run your script like this:

```
$ hello-world.rb
```

Using env

If you do not want to hard-code the path to the `ruby` executable, you can use the `env`[22] command in the shebang line to search for the `ruby` executable in your PATH and execute it. This way, you will not need to change the shebang line on all of your Ruby scripts if you move them to a computer with Ruby installed in a different directory.

```
#!/usr/bin/env ruby

puts 'Hello world'
```

7.3.2 Windows

If you install the native Windows version of Ruby using the Ruby One-Click Installer[23], then the installer has setup Windows to automatically recognize your Ruby scripts as executables. Just type the name of the script to run it.

```
$ hello-world.rb
Hello world
```

If this does not work, or if you installed Ruby in some other way, follow these steps.

1. Log in as an administrator[24].
2. Run the standard Windows "Command Prompt", `cmd`.
3. At the command prompt (*i.e.* shell prompt), run the following Windows commands. When you run `ftype`, change the command-line arguments to correctly point to where you installed the `ruby.exe` executable on your computer.

```
$ assoc .rb=RubyScript
.rb=RubyScript

$ ftype RubyScript="c:\ruby\bin\ruby.exe" "%1" %*
RubyScript="c:\ruby\bin\ruby.exe" "%1" %*
```

22 http://en.wikipedia.org/wiki/env
23 http://www.ruby-lang.org/en/downloads/
24 http://en.wikipedia.org/wiki/System%20administrator

For more help with these commands, run "`help assoc`" and "`help ftype`".

8 Basic Ruby - Strings

Like Python[1], Java[2], and the .NET Framework[3], Ruby has a built-in String class.

8.1 String literals

One way to create a String is to use single or double quotes inside a Ruby program to create what is called a string literal. We've already done this with our "hello world[4]" program. A quick update to our code shows the use of both single and double quotes.

```
puts ,Hello world,
puts "Hello world"
```

Being able to use either single or double quotes is similar to Perl[5], but different from languages such as C[6] and Java[7], which use double quotes for string literals and single quotes for single characters.

So what difference is there between single quotes and double quotes in Ruby? In the above code, there's no difference. However, consider the following code:

```
puts "Betty,s pie shop"
puts ,Betty\,s pie shop,
```

Because "`Betty's`" contains an apostrophe, which is the same character as the single quote, in the second line we need to use a backslash to escape the apostrophe so that Ruby understands that the apostrophe is *in* the string literal instead of marking the end of the string literal. The backslash followed by the single quote is called an escape sequence[8].

8.2 Single quotes

Single quotes only support two escape sequences.

- `\'` – single quote
- `\\` – single backslash

Except for these two escape sequences, everything else between single quotes is treated literally.

---

1 http://en.wikipedia.org/wiki/Python%20%28programming%20language%29
2 http://en.wikipedia.org/wiki/Java%20%28programming%20language%29
3 http://en.wikipedia.org/wiki/.NET%20Framework
4 Chapter 7 on page 21
5 http://en.wikipedia.org/wiki/Perl
6 http://en.wikipedia.org/wiki/C%20%28programming%20language%29
7 http://en.wikipedia.org/wiki/Java%20%28programming%20language%29
8 http://en.wikipedia.org/wiki/escape%20sequence

8.3 Double quotes

Double quotes allow for many more escape sequences than single quotes. They also allow you to embed variables or Ruby code inside of a string literal — this is commonly referred to as interpolation[9].

```
puts "Enter name"
name = gets.chomp
puts "Your name is #{name}"
```

8.3.1 Escape sequences

Below are some of the more common escape sequences that can appear inside of double quotes.

- `\"` — double quote
- `\\` — single backslash
- `\a` — bell/alert[10]
- `\b` — backspace[11]
- `\r` — carriage return[12]
- `\n` — newline[13]
- `\s` — space[14]
- `\t` — tab[15]

Try out this example code to better understand escape sequences.

```
puts "Hello\t\tworld"

puts "Hello\b\b\b\b\bGoodbye world"

puts "Hello\rStart over world"

puts "1. Hello\n2. World"
```

The result:

```
$ double-quotes.rb
Hello world
Goodbye world
Start over world
1. Hello
2. World
```

Notice that the newline escape sequence (in the last line of code) simply starts a new line.

9 Chapter 8.4 on page 29
10 http://en.wikipedia.org/wiki/bell%20character
11 http://en.wikipedia.org/wiki/backspace
12 http://en.wikipedia.org/wiki/carriage%20return
13 http://en.wikipedia.org/wiki/newline
14 http://en.wikipedia.org/wiki/space%20character
15 http://en.wikipedia.org/wiki/tab%20key

The bell character[16], produced by escape code `\a`, is considered a control character[17]. It does not represent a letter of the alphabet, a punctuation mark, or any other written symbol. Instead, it instructs the terminal emulator[18] (called a console[19] on Microsoft Windows[20]) to "alert" the user. It is up to the terminal emulator to determine the specifics of how to respond, although a beep[21] is fairly standard. Some terminal emulators will flash briefly.

Run the following Ruby code to check out how your terminal emulator handles the bell character.

```
puts "\aHello world\a"
```

8.4 puts

We've been using the `puts` function quite a bit to print out text. Whenever `puts` prints out text, it automatically prints out a newline after the text. For example, try the following code.

```
puts "Say", "hello", "to", "the", "world"
```

The result:

```
$ hello-world.rb
Say
hello
to
the
world
```

8.5 print

In contrast, Ruby's `print` function only prints out a newline if you specify one. For example, try out the following code. We include a newline at the end of `print`'s argument list so that the shell prompt appears on a new line, after the text.

```
print "Say", "hello", "to", "the", "world", "\n"
```

The result:

```
$ hello-world.rb
Sayhellototheworld
```

The following code produces the same output, with all the words run together.

16 http://en.wikipedia.org/wiki/bell%20character
17 http://en.wikipedia.org/wiki/control%20character
18 http://en.wikipedia.org/wiki/terminal%20emulator
19 http://en.wikipedia.org/wiki/Win32%20console
20 http://en.wikipedia.org/wiki/Microsoft%20Windows
21 http://en.wikipedia.org/wiki/beep%20%28sound%29

```
print "Say"
print "hello"
print "to"
print "the"
print "world"
print "\n"
```

8.6 See also

String literals[22]

22 http://en.wikibooks.org/wiki/Ruby_Programming/Syntax/Literals#Strings

9 Basic Ruby - Alternate quotes

In Ruby, there's more than one way to quote a string literal. Much of this will look familiar to Perl[1] programmers.

9.1 Alternate single quotes

Let's say we are using single quotes to print out the following path.

```
puts ,c:\bus schedules\napolean\the portland bus schedule.txt,
```

The single quotes keep the `\b`, `\n`, and `\t` from being treated as escape sequences[2] (the same cannot be said for wikibooks' syntax highlighting). But consider the following string literal.

```
puts ,c:\napolean\,s bus schedules\tomorrow\,s bus schedule.txt,
```

Escaping the apostrophes makes the code less readable and makes it less obvious what will print out. Luckily, in Ruby, there's a better way. You can use the `%q` operator to apply single-quoting rules[3], but choose your own delimiter[4] to mark the beginning and end of the string literal.

```
puts %q!c:\napolean's documents\tomorrow's bus schedule.txt!
puts %q/c:\napolean's documents\tomorrow's bus schedule.txt/
puts %q^c:\napolean's documents\tomorrow's bus schedule.txt^
puts %q(c:\napolean's documents\tomorrow's bus schedule.txt)
puts %q{c:\napolean's documents\tomorrow's bus schedule.txt}
puts %q<c:\napolean's documents\tomorrow's bus schedule.txt>
```

Each line will print out the same text – "`c:\napolean's documents\tomorrow's bus schedule.txt`". You can use any punctuation you want as a delimiter, not just the ones listed in the example.

Of course, if your chosen delimiter appears inside of the string literal, then you need to escape it.

```
puts %q#c:\napolean's documents\tomorrow's \#9 bus schedule.txt#
```

If you use matching braces to delimit the text, however, you can nest braces, without escaping them.

1 http://en.wikipedia.org/wiki/Perl
2 http://en.wikipedia.org/wiki/escape%20sequences
3 Chapter 8.2 on page 27
4 http://en.wikipedia.org/wiki/delimiter

```
puts %q(c:\napolean's documents\the (bus) schedule.txt)
puts %q{c:\napolean's documents\the {bus} schedule.txt}
puts %q<c:\napolean's documents\the <bus> schedule.txt>
```

9.2 Alternate double quotes

The `%Q` operator (notice the case of Q in `%Q`) allows you to create a string literal using double-quoting rules[5], but without using the double quote as a delimiter. It works much the same as the `%q`[6] operator.

```
print %Q^Say:\tHello world\n\tHello world\n^
print %Q(Say:\tHello world\n\tHello world\n)
```

Just like double quotes, you can interpolate[7] Ruby code inside of these string literals.

```
name = 'Charlie Brown'

puts %Q!Say "Hello," #{name}.!
puts %Q/What is "4 plus 5"? Answer: #{4+5}/
```

5 Chapter 8.3 on page 28
6 Chapter 9.1 on page 31
7 Chapter 8.4 on page 29

10 Basic Ruby - Here documents

For creating multiple-line strings, Ruby supports here documents[1] (heredocs), a feature that originated in the Bourne shell[2] and is also available in Perl[3] and PHP[4].

10.1 Here documents

To construct a here document, the << operator is followed by an identifier that marks the end of the here document. The end mark is called the terminator. The lines of text prior to the terminator are joined together, including the newlines and any other whitespace.

```
puts <<GROCERY_LIST
Grocery list
------------
1. Salad mix.
2. Strawberries.*
3. Cereal.
4. Milk.*

* Organic
GROCERY_LIST
```

The result:

```
$ grocery-list.rb
Grocery list
------------
1. Salad mix.
2. Strawberries.*
3. Cereal.
4. Milk.*

* Organic
```

If we pass the `puts` function multiple arguments, the string literal created from the here document is inserted into the argument list wherever the << operator appears. In the code below, the here-document *(containing the four grocery items and a blank line)* is passed in as the third argument. We get the same output as above.

1 http://en.wikipedia.org/wiki/here%20document
2 http://en.wikipedia.org/wiki/Bourne%20shell
3 http://en.wikipedia.org/wiki/Perl
4 http://en.wikipedia.org/wiki/PHP

```
puts 'Grocery list', '------------', <<GROCERY_LIST, '* Organic'
1. Salad mix.
2. Strawberries.*
3. Cereal.
4. Milk.*

GROCERY_LIST
```

You can also have multiple here documents in an argument list. We added a blank line at the end of each here document to make the output more readable.

```
puts 'Produce', '-------', <<PRODUCE, 'Dairy', '-----', <<DAIRY, '*
Organic'
1. Strawberries*
2. Blueberries

PRODUCE
1. Yogurt
2. Milk*
3. Cottage Cheese

DAIRY
```

The result:

```
$ grocery-list.rb
Produce
-------
1. Strawberries*
2. Blueberries

Dairy
-----
1. Yogurt
2. Milk*
3. Cottage Cheese

* Organic
```

We have been using the `puts` function in our examples, but you can pass here documents to any function that accepts Strings.

10.2 Indenting

If you indent the lines inside the here document, the leading whitespace is preserved. However, there must not be any leading whitespace before the terminator.

```
puts 'Grocery list', '------------', <<GROCERY_LIST
    1. Salad mix.
    2. Strawberries.
    3. Cereal.
    4. Milk.
GROCERY_LIST
```

The result:

```
$ grocery-list.rb
Grocery list
------------
    1. Salad mix.
    2. Strawberries.
    3. Cereal.
    4. Milk.
```

If, for readability, you want to also indent the terminator, use the <<− operator.

```
puts 'Grocery list', '------------', <<-GROCERY_LIST
    1. Salad mix.
    2. Strawberries.
    3. Cereal.
    4. Milk.
    GROCERY_LIST
```

Note, however, that the whitespace before each line of text *within* the here document is still preserved.

```
$ grocery-list.rb
Grocery list
------------
    1. Salad mix.
    2. Strawberries.
    3. Cereal.
    4. Milk.
```

10.3 Quoting rules

You may wonder whether here documents follow single-quoting[5] or double-quoting[6] rules. If there are no quotes around the identifier, like in our examples so far, then the body of the here document follows double-quoting rules.

```
name = 'Charlie Brown'

puts <<QUIZ
Student: #{name}

1.\tQuestion: What is 4+5?
\tAnswer: The sum of 4 and 5 is #{4+5}
QUIZ
```

The result:

5 Chapter 8.2 on page 27
6 Chapter 8.3 on page 28

```
$ quiz.rb
Student: Charlie Brown

1.  Question: What is 4+5?
 Answer: The sum of 4 and 5 is 9
```

Double-quoting rules are also followed if you put double quotes around the identifier. However, do not put double quotes around the terminator.

```
puts <<"QUIZ"
Student: #{name}

1.\tQuestion: What is 4+5?
\tAnswer: The sum of 4 and 5 is #{4+5}
QUIZ
```

To create a here document that follows single-quoting rules, place single quotes around the identifier.

```
puts <<'BUS_SCHEDULES'
c:\napolean's documents\tomorrow's bus schedule.txt
c:\new documents\sam spade's bus schedule.txt
c:\bus schedules\the #9 bus schedule.txt
BUS_SCHEDULES
```

The result:

```
$ bus-schedules.rb
c:\napolean's documents\tomorrow's bus schedule.txt
c:\new documents\sam spade's bus schedule.txt
c:\bus schedules\the #9 bus schedule.txt
```

11 Basic Ruby - Introduction to objects

Like Smalltalk[1], Ruby is a pure object-oriented language — everything is an object. In contrast, languages such as C++[2] and Java[3] are hybrid languages that divide the world between objects and primitive types[4]. The hybrid approach results in better performance for some applications, but the pure object-oriented approach is more consistent and simpler to use.

11.1 What is an object?

Using Smalltalk[5] terminology, an object can do exactly three things.

1. Hold state, including references to other objects.
2. Receive a message, from both itself and other objects.
3. In the course of processing a message, send messages, both to itself and to other objects.

If you don't come from Smalltalk background, it might make more sense to rephrase these rules as follows:

1. An object can contain data, including references to other objects.
2. An object can contain methods, which are functions that have special access to the object's data.
3. An object's methods can call/run other methods/functions.

11.2 Variables and objects

Let's fire up `irb`[6] to get a better understanding of objects.

```
$ irb --simple-prompt
>> comedian = "Stephen Colbert"
=> "Stephen Colbert"
```

1 http://en.wikipedia.org/wiki/Smalltalk
2 http://en.wikipedia.org/wiki/C%2B%2B%20%28programming%20language%29
3 http://en.wikipedia.org/wiki/Java%20%28programming%20language%29
4 http://en.wikipedia.org/wiki/primitive%20types
5 http://en.wikipedia.org/wiki/Smalltalk
6 Chapter 5 on page 17

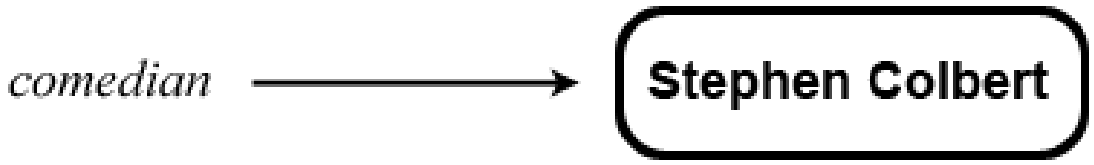

Figure 1 none

In the first line, we created a String object containing the text "`Stephen Colbert`". We also told Ruby to use the variable `comedian` to refer to this object.

Next, we tell Ruby to also use the variable `favorite_comedian` to refer to the same String object.

```
>> favorite_comedian = comedian
=> "Stephen Colbert"
```

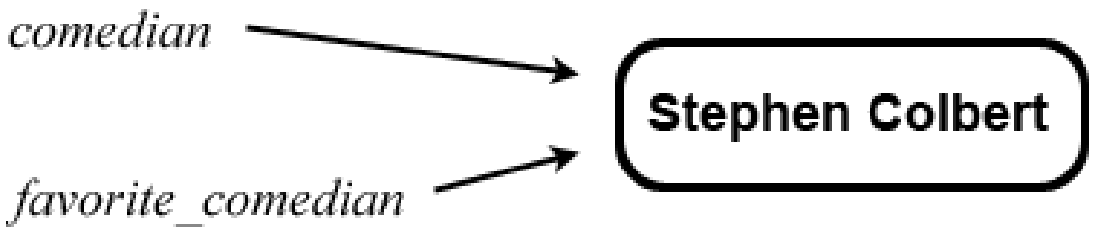

Figure 2 none

Now, we have two variables that we can use to refer to the same String object — `comedian` and `favorite_comedian`. Since they both refer to the same object, if the object changes (as we'll see below), the change will show up when using either variable.

11.3 Methods

In Ruby, methods that end with an exclamation mark[7] (also called a "bang") modify the object. For example, the method `upcase!` changes the letters of a String to uppercase.

```
>> comedian.upcase!
=> "STEPHEN COLBERT"
```

7 http://en.wikipedia.org/wiki/exclamation%20mark

Since both of the variables `comedian` and `favorite_comedian` point to the same String object, we can see the new, uppercase text using either variable.

```
>> comedian
=> "STEPHEN COLBERT"
>> favorite_comedian
=> "STEPHEN COLBERT"
```

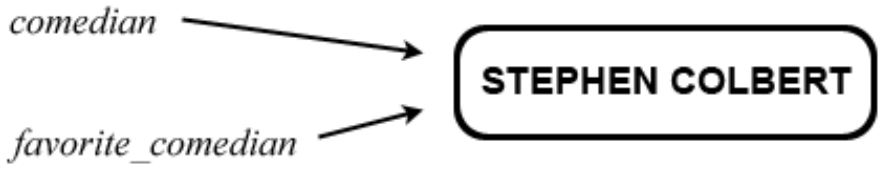

Figure 3 none

Methods that do not end in an exclamation point return data, but do not modify the object. For example, `downcase!` modifies a String object by making all of the letters lowercase. However, `downcase` returns a lowercase copy of the String, but the original string remains the same.

```
>> comedian.downcase
=> "stephen colbert"
>> comedian
=> "STEPHEN COLBERT"
```

Since the original object still contains the text "`STEPHEN COLBERT`", you might wonder where the new String object, with the lowercase text, went to. Well, after `irb` printed out its contents, it can no longer be accessed since we did not assign a variable to keep track of it.It's essentially gone, and Ruby will dispose of it.

11.4 Reassigning a variable

But what if your favorite comedian is not Stephen Colbert[8]? Let's point `favorite_comedian` to a new object.

```
>> favorite_comedian = "Jon Stewart"
=> "Jon Stewart"
```

Now, each variable points to a different object.

8 `http://en.wikipedia.org/wiki/Stephen%20Colbert`

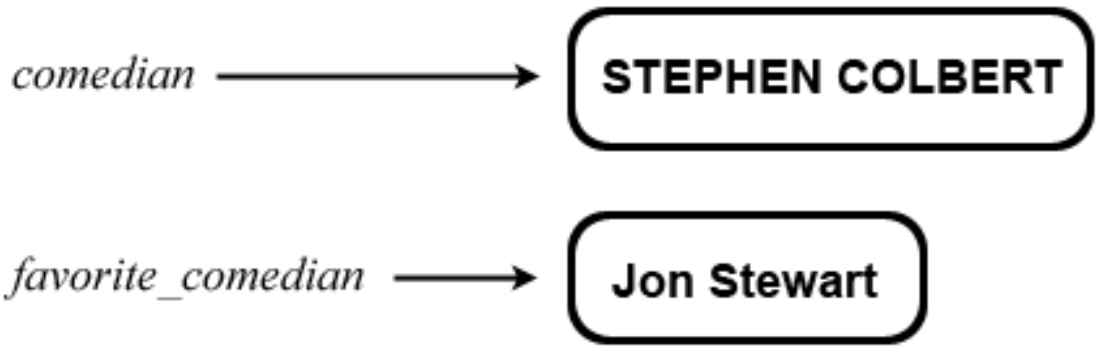

Figure 4 none

Let's say that we change our mind again. Now, our favorite comedian is Ellen DeGeneres[9].

```
>> favorite_comedian = "Ellen DeGeneres"
=> "Ellen DeGeneres"
```

Now, no variable points to the "`Jon Stewart`" String object any longer. Hence, Ruby will dispose of it.

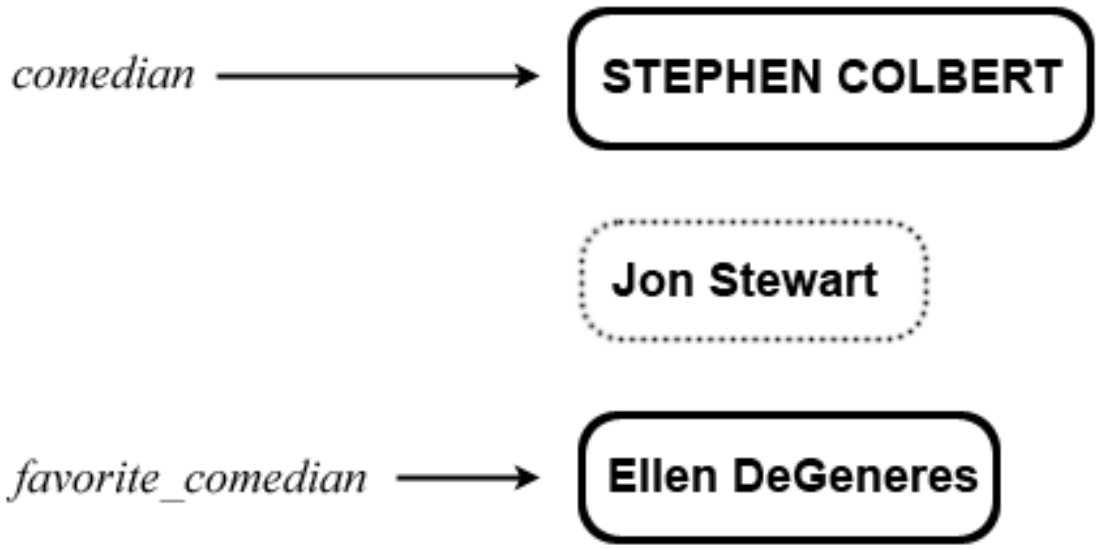

Figure 5 none

9 `http://en.wikipedia.org/wiki/Ellen%20DeGeneres`

12 Basic Ruby - Ruby basics

As with the rest of this tutorial, we assume some basic familiarity with programming language concepts (i.e. if statement, while loops) and also some basic understanding of object-oriented programming[1].

1 http://en.wikipedia.org/wiki/Object_oriented_programming

13 Dealing with variables

We'll deal with variables in much more depth when we talk about classes and objects. For now, let's just say your basic local variable names should start with either a lower case letter or an underscore, and should contain upper or lower case letters, numbers, and underscore characters. Global variables start with a `$`.

14 Program flow

Ruby includes a pretty standard set of looping and branching constructs: `if`, `while` and `case`

For example, here's `if` in action:

```
a = 10 * rand

if a < 5
  puts "#{a} less than 5"
elsif a > 7
  puts "#{a} greater than 7"
else
  puts "Cheese sandwich!"
end
```

[As in other languages, the `rand` function generates a random number between 0 and 1]

There will be plenty more time to discuss conditional statements in later chapters. The above example should be pretty clear.

Ruby also includes a negated form of `if` called `unless` which goes something like

```
unless a > 5
  puts "a is less than or equal to 5"
else
  puts "a is greater than 5"
end
```

Generally speaking, Ruby keeps an `if` statement straight as long as the conditional (`if ...`) and the associated code block are on separate lines. If you have to smash everything together on one line, you'll need to place the `then` keyword after the conditional

```
if a < 5 then puts "#{a} less than 5" end
if a < 5 then puts "#{a} less than 5" else puts "#{a} greater than
5" end
```

Note that the `if` statement is also an expression; its value is the last line of the block executed. Therefore, the line above could also have been written as

```
puts(if a < 5 then "#{a} less than 5" else "#{a} greater than 5"
end)
```

Ruby has also adopted the syntax from Perl where `if` and `unless` statements can be used as conditional modifiers *after* a statement. For example

```
puts "#{a} less than 5" if a < 5
puts "Cheese sandwich" unless a == 4
```

`while` behaves as it does in other languages -- the code block that follows is run zero or more times, as long as the conditional is true

```
while a > 5
  a = 10*rand
end
```

And like `if`, there is also a negated version of `while` called `until` which runs the code block *until* the condition is true.

Finally there is the `case` statement which we'll just include here with a brief example. `case` is actually a very powerful super version of the `if ... elsif...` system

```
a = (10*rand).round
#a = rand(11) would do the same

case a
when 0..5
  puts "#{a}: Low"
when 6
  puts "#{a}: Six"
else
  puts "#{a}: Cheese toast!"
end
```

There are some other interesting things going on in this example, but here the `case` statement is the center of attention.

15 Writing functions

In keeping with Ruby's all-object-oriented-all-the-time design, functions are typically referred to as methods. No difference. We'll cover methods in much more detail when we get to objects and classes. For now, basic method writing looks something like this:

```
# Demonstrate a method with func1.rb
```

```
def my_function( a )
  puts "Hello, #{a}"
  return a.length
end

len = my_function( "Giraffe" )
puts "My secret word is #{len} long"
```

```
$ func1.rb
Hello, Giraffe
My secret word is 7 long
```

Methods are defined with the `def` keyword, followed by the function name. As with variables, local and class methods should start with a lower case letter.

In this example, the function takes one argument (`a`) and returns a value. Note that the input arguments aren't typed (i.e. `a` need not be a string) ... this allows for great flexibility but can also cause a lot of trouble. The function also returns a single value with the `return` keyword. Technically this isn't necessary -- the value of the last line executed in the function is used as the return value -- but more often than not using `return` explicitly makes things clearer.

As with other languages, `Ruby` supports both default values for arguments and variable-length argument lists, both of which will be covered in due time. There's also support for code blocks, as discussed below.

16 Blocks

One very important concept in Ruby is the code block. It's actually not a particularly revolutionary concept -- any time you've written `if ... { ... }` in C or Perl you've defined a code block, but in Ruby a code block has some hidden secret powers...

Code blocks in Ruby are defined either with the keywords `do..end` or the curly brackets `{..}`

```
do
  print "I like "
  print "code blocks!"
end

{
  print "Me too!"
}
```

One very powerful usage of code blocks is that methods can take one as a parameter and *execute* it along the way.

[ed note: the Pragmatic Programmers actually want to point out that it's not very useful to describe it this way. Instead, the block of code behaves like a 'partner' to which the function occasionally hands over control]

The concept can be hard to get the first time it's explained to you. Here's an example:

```
$ irb --simple-prompt
>> 3.times { puts "Hi!" }
Hi!
Hi!
Hi!
=> 3
```

Surprise! You always thought 3 was just a number, but it's actually an object (of type `Fixnum`) As it's an object, it has a member function `times` which takes a block as a parameter. The function runs the block 3 times.

Blocks can actually receive parameters, using a special notation `|..|`. In this case, a quick check of the documentation for `times` shows it will pass a single parameter into the block, indicating which loop it's on:

```
$ irb --simple-prompt
>> 4.times { |x| puts "Loop number #{x}" }
Loop number 0
Loop number 1
Loop number 2
```

```
Loop number 3
=> 4
```

The `times` function passes a number into the block. The block gets that number in the variable `x` (as set by the `|x|`), then prints out the result.

Functions interact with blocks through the `yield`. Every time the function invokes `yield` control passes to the block. It only comes back to the function when the block finishes. Here's a simple example:

```
# Script block2.rb

def simpleFunction
  yield
  yield
end

simpleFunction { puts "Hello!" }
```

```
$ block2.rb
Hello!
Hello!
```

The `simpleFunction` simply yields to the block twice -- so the block is run twice and we get two times the output. Here's an example where the function passes a parameter to the block:

```
# Script block1.rb

def animals
  yield "Tiger"
  yield "Giraffe"
end

animals { |x| puts "Hello, #{x}" }
```

```
$ block1.rb
Hello, Tiger
Hello, Giraffe
```

It might take a couple of reads through to figure out what's going on here. We've defined the function "animals" -- it expects a code block. When executed, the function calls the code block twice, first with the parameter "Tiger" then again with the parameter "Giraffe". In this example, we've written a simple code block which just prints out a greeting to the animals. We could write a different block, for example:

```
animals { |x| puts "It's #{x.length} characters long!" }
```

which would give:

```
It's 5 characters long!
It's 7 characters long!
```

Two completely different results from running the same function with two different blocks.

There are many powerful uses of blocks. One of the first you'll come across is the `each` function for arrays -- it runs a code block once for each element in the array -- it's great for iterating over lists.

17 Ruby is really, really object-oriented

Ruby is very object oriented. Everything is an object -- even things you might consider constants. This also means that the vast majority of what you might consider "standard functions" aren't floating around in some library somewhere, but are instead methods of a given variable.

Here's one example we've already seen:

```
3.times { puts "Hi!" }
```

Even though 3 might seem like just a constant number, it's in fact an instance of the class `Fixnum` (which inherits from the class `Numeric` which inherits from the class `Object`). The method `times` comes from `Fixnum` and does just what it claims to do.

Here are some other examples

```
$ irb --simple-prompt
>> 3.abs
=> 3
>> -3.abs
=> 3
>> "giraffe".length
=> 7
>> a = "giraffe"
=> "giraffe"
>> a.reverse
=> "effarig"
```

There will be lots of time to consider how object-oriented design filters through Ruby in the coming chapters.

18 Basic Ruby - Data types

18.1 Ruby Data Types

As mentioned in the previous chapter, everything in Ruby is an object. Everything has a class. Don't believe me? Try running this bit of code:

```
h = {"hash?" => "yep, it\,s a hash!", "the answer to everything" =>
 42, :linux => "fun for coders."}
puts "Stringy string McString!".class
puts 1.class
puts nil.class
puts h.class
puts :symbol.class
```

displays

```
String
Fixnum
NilClass
Hash
Symbol
```

See? Everything is an object. Every object has a method called `class` that returns that object's class. You can call methods on pretty much anything. Earlier you saw an example of this in the form of

```
3.times
```

. (Technically when you call a method you're sending a message to the object, but I'll leave the significance of that for later.)

Something that makes this extreme object oriented-ness very fun for me is the fact that all classes are open, meaning you can add variables and methods to a class at any time during the execution of your code. This, however, is a discussion of datatypes.

18.2 Constants

We'll start off with constants because they're simple. Two things to remember about constants:

1. Constants start with capital letters. `Constant` is a constant. `constant` is not a constant.

2. You can change the values of constants, but Ruby will give you a warning. (Silly, I know... but what can you do?)

Congrats. Now you're an expert on Ruby constants.

18.3 Symbols

So did you notice something weird about that first code listing? "What the heck was that colon thingy about?" Well, it just so happens that Ruby's object oriented ways have a cost: lots of objects make for slow code. Every time you type a string, Ruby makes a new object. Regardless of whether two strings are identical, Ruby treats every instance as a new object. You could have "live long and prosper" in your code once and then again later on and Ruby wouldn't even realize that they're pretty much the same thing. Here is a sample irb session which demonstrates this fact:

```
irb>

"live long and prosper".object_id



=> -507772268

irb>

"live long and prosper".object_id



=> -507776538
```

Notice that the object ID returned by irb Ruby is different even for the same two strings.

To get around this memory hoggishness, Ruby has provided "symbols."

Symbol

s are lightweight objects best used for comparisons and internal logic. If the user doesn't ever see it, why not use a symbol rather than a string? Your code will thank you for it. Let us try running the above code using symbols instead of strings:

```
irb>

:my_symbol.object_id



=> 150808

irb>

:my_symbol.object_id
```

```
=> 150808
```

Symbols are denoted by the colon sitting out in front of them, like so:

```
:symbol_name
```

18.4 Hashes

Hashes are like dictionaries, in a sense. You have a key, a reference, and you look it up to find the associated object, the definition.

The best way to illustrate this, I think, is with a quick demonstration:

```
hash = { :leia => "Princess from Alderaan", :han => "Rebel without a
 cause", :luke => "Farmboy turned Jedi"}
puts hash[:leia]
puts hash[:han]
puts hash[:luke]
```

displays

```
Princess from Alderaan
Rebel without a cause
Farmboy turned Jedi
```

I could have also written this like so:

```
hash = { :leia => "Princess from Alderaan", :han => "Rebel without a
 cause", :luke => "Farmboy turned Jedi"}
hash.each do |key, value|
     puts value
end
```

This code cycles through each element in the hash, putting the key in the

```
key
```

variable and the value in the

```
value
```

variable, which is then displayed

```
Princess of Alderaan
Rebel without a cause
Farmboy turned Jedi
```

I could have been more verbose about defining my hash; I could have written it like this:

```
hash = Hash.[](:leia => "Princess from Alderaan", :han => "Rebel
 without a cause", :luke => "Farmboy turned Jedi")
hash.each do |key, value|
```

```
    puts value
end
```

If I felt like offing Luke, I could do something like this:

```
hash.delete(:luke)
```

Now Luke's no longer in the hash. Or lets say I just had a vendetta against farmboys in general. I could do this:

```
hash.delete_if {|key, value| value.downcase.match("farmboy")}
```

This iterates through each key-value pair and deletes it, but only if the block of code following it returns

```
true
```

. In the block I made the value lowercase (in case the farmboys decided to start doing stuff like "FaRmBoY!1!") and then checked to see if "farmboy" matched anything in its contents. I could have used a regular expression, but that's another story entirely.

I could add Lando into the mix by assigning a new value to the hash:

```
hash[:lando] = "Dashing and debonair city administrator."
```

I can measure the hash with

```
hash.length
```

. I can look at only keys with the

```
hash.inspect
```

method, which returns the hash's keys as an

```
Array
```

. Speaking of which...

18.5 Arrays

```
Array
```

s are a lot like

```
Hash
```

es, except that the keys are always consecutive numbers, and always starts at 0. In an

```
Array
```

with five items, the *last* element would be found at

```
array[4]
```

and the *first* element would be found at

`array[0]`

. In addition, all the methods that you just learned with

`Hash`

es can also be applied to

`Array`

s.

Here are two ways to create an

`Array`

:

```
array1 = ["hello", "this", "is", "an", "array!"]
array2 = []
array2 << "This"   # index 0
array2 << "is"     # index 1
array2 << "also"   # index 2
array2 << "an"     # index 3
array2 << "array!" # index 4
```

As you may have guessed, the

`<<`

operator pushes values onto the end of an

`Array`

. If I were to write

`puts array2[4]`

after declaring those two

`Array`

s the output would be

`array!`

. Of course, if I felt like simultaneously getting

`array!`

and deleting it from the array, I could just

`Array.pop`

it off. The

`Array.pop`

method returns the *last* element in an array and then immediately removes it from that array:

```
string = array2.pop
```

Then

```
string
```

would hold

```
array!
```

and

```
array2
```

would be an element shorter.

If I kept doing this, array2 wouldn't hold any elements. I can check for this condition by calling the

```
Array.empty?
```

method. For example, the following bit of code moves all the elements from one

```
Array
```

to another:

```
array1 << array2.pop until array2.empty?
```

Here's something that really excites me:

```
Array
```

s can be subtracted from, and added to, each other. I can't vouch for *every* language that's out there, but I know that Java, C++, C#, perl, and python would all look at me like I was a crazy person if I tried to execute the following bit of code:

```
array3 = array - array2
array4 = array + array2
```

After that code is evaluated, all of the following are true:

- ```
  array3
  ```
  contains all of the elements that
  ```
  array
  ```
  did, except the ones that were also in
  ```
  array2
  ```
  .
- All the elements of
  ```
  array
  ```
  , minus the elements of
  ```
  array2
  ```
  , are now contained within
  ```
  array3
  ```
  ```

- .

  `array4`
  now contains all the elements of both

  `array`
  and

  `array2`
  .

You may search for a particular value in variable

`array`

with the

`Array.include?`

method:

```
array.include?("Is this in here?")
```

If you just wanted to turn the whole

`Array`

into a

`String`

, you could:

```
string = array2.join(" ")
```

If array2 had the value that we declared in the last example, then

`string`

's value would be

```
This is also an array!
```

We could have called the

`Array.join`

method without any arguments:

```
string = array2.join
```

`string`

's value would now be

```
Thisisalsoanarray!
```

18.6 Strings

I would recommend reading the chapters on strings[1] and alternate quotes[2] now if you haven't already. This chapter is going to cover some pretty spiffy things with

`String`

s and just assume that you already know the information in these two chapters.

In Ruby, there are some pretty cool built-in functions where

`String`

s are concerned. For example, you can multiply them:

```
"Danger, Will Robinson!" * 5
```

yields

```
Danger, Will Robinson!Danger, Will Robinson!Danger, Will
 Robinson!Danger, Will Robinson!Danger, Will Robinson!
```

`String`

s may also be compared:

```
"a" < "b"
```

yields

```
true
```

The preceding evaluation is actually comparing the ASCII values of the characters. But what, I hear you ask, is the ASCII value of an given character? With ruby versions prior to 1.9 you can find the ASCII value of a character with:

```
puts ?A
```

However, With Ruby version 1.9 or later that no longer works[3]. Instead, you can try the

`String`.ord

method:

```
puts "A".ord
```

Either approach will display

```
65
```

1 Chapter 8 on page 27
2 Chapter 9 on page 31
3 http://stackoverflow.com/questions/1270209/getting-an-ascii-character-code-in-ruby-fails

which is the ASCII value of `A`. Simply replace `A` with whichever character you wish to inquire about.

To perform the opposite conversion (from `65` to `A`, for instance), use the

`Integer.chr`

method:

```
puts 65.chr
```

displays

```
A
```

Concatenation works the same as most other languages: putting a

`+`

character between two

`String`

s will yield a new

`String`

whose value is the same as the others, one after another:

```
"Hi, this is " + "a concatenated string!"
```

yields

```
Hi, this is a concatenated string!
```

For handling pesky

`String`

variables without using the concatenate operator, you can use interpolation. In the following chunk of code

`string1`

,

`string2`

, and

`string3`

are identical:

```
thing1 = "Red fish, "
thing2 = "blue fish."
string1 = thing1 + thing2 + " And so on and so forth."
string2 = "#{thing1 + thing2} And so on and so forth."
string3 = "#{thing1}#{thing2} And so on and so forth."
```

If you need to iterate through (that is, step through each of) the letters in a

`String`

object, you can use the

`String.scan`

method:

```
thing = "Red fish"
thing.scan(/./) {|letter| puts letter}
```

Displays each letter in

`thing`

(puts automatically adds a newline after each call):

```
R
e
d

f
i
s
h
```

But what's with that weird "

`/./`

" thing in the parameter? That, my friend, is called a *regular expression*[4]. They're helpful little buggers, quite powerful, but outside the scope of this discussion. All you need to know for now is that

`/./`

is "regex" speak for "any *one* character." If we had used

`/../`

then ruby would have iterated over each group of two characters, and missed the last one since there's an odd number of characters!

Another use for regular expressions can be found with the `=~` operator. You can check to see if a

`String`

matches a regular expression using the match *operator*,

`=~`

:

```
puts "Yeah, there,s a number in this one." if "C3-P0, human-cyborg
 relations" =~ /[0-9]/
```

4 Chapter 68 on page 219

displays

```
Yeah, there,s a number in this one.
```

The

`String.match`

method works much the same way, except it can accept a

`String`

as a parameter as well. This is helpful if you're getting regular expressions from a source outside the code. Here's what it looks like in action:

```
puts "Yep, they mentioned Jabba in this one." if "Jabba the
 Hutt".match("Jabba")
```

Alright, that's enough about regular expressions. Even though you can use regular expressions with the next two examples, we'll just use regular old

`String`

s. Lets pretend you work at the Ministry of Truth and you need to replace a word in a

`String`

with another word. You can try something like:

```
string1 = "2 + 2 = 4"
string2 = string1.sub("4", "5")
```

Now

`string2`

contains

`2 + 2 = 5`

. But what if the

`String`

contains lots of lies like the one you just corrected?

`String.sub`

only replaces the first occurrence of a word! I guess you could iterate through the

`String`

using

`String.match`

method and a

`while`

loop, but there's a much more efficient way to accomplish this:

```
winston = %q{   Down with Big Brother!
                Down with Big Brother!
                Down with Big Brother!
                Down with Big Brother!
                Down with Big Brother!}
winston.gsub("Down with", "Long live")
```

Big Brother would be *so* proud!

`String.gsub`

is the "*g*lobal *sub*stitute" function. Every occurrence of "

`Down with`

" has now been replaced with "`Long live`" so now `winston` is only proclaiming its love for Big Brother, not its disdain thereof.

On that happy note, lets move on to

`Integer`

s and

`Float`

s. If you want to learn more about methods in the

`String`

class, look at the end of this chapter for a quick reference table.

18.7 Numbers (Integers and Floats)

You can skip this paragraph if you know all the standard number operators. For those who don't, here's a crash course. `+` adds two numbers together. `-` subtracts them. `/` divides. `*` multiplies. `%` returns the remainder of two divided numbers.

Alright, integers are numbers with no decimal place. Floats are numbers with decimal places. `10 / 3` yields `3` because dividing two integers yields an integer. Since integers have no decimal places all you get is `3`. If you tried `10.0 / 3` you would get `3.33333...` If you have even one float in the mix you get a float back. Capice?

Alright, let's get down to the fun part. Everything in Ruby is an object, let me reiterate. That means that pretty much everything has at least one method. Integers and floats are no exception. First I'll show you some integer methods.

Here we have the venerable `times` method. Use it whenever you want to do something more than once. Examples:

```
puts "I will now count to 99..."
100.times {|number| puts number}
5.times {puts "Guess what?"}
puts "I'm done!"
```

This will print out the numbers 0 through 99, print out `Guess what?` five times, then say `I'm done!` It's basically a simplified `for` loop. It's a little slower than a `for` loop by a few hundredths of a second or so; keep that in mind if you're ever writing Ruby code for NASA. ;-)

Alright, we're nearly done, six more methods to go. Here are three of them:

```
# First a visit from The Count...
1.upto(10) {|number| puts "#{number} Ruby loops, ah-ah-ah!"}

# Then a quick stop at NASA...
puts "T-minus..."
10.downto(1) {|x| puts x}
puts "Blast-off!"

# Finally we'll settle down with an obscure Schoolhouse Rock video...
5.step(50, 5) {|x| puts x}
```

Alright, that should make sense. In case it didn't, `upto` counts up from the number it's called from to the number passed in its parameter. `downto` does the same, except it counts down instead of up. Finally, `step` counts from the number its called from to the first number in its parameters by the second number in its parameters. So `5.step(25, 5) {|x| puts x}` would output every multiple of five starting with five and ending at twenty-five.

Time for the last three:

```
string1 = 451.to_s
string2 = 98.6.to_s
int = 4.5.to_i
float = 5.to_f
```

`to_s` converts floats and integers to strings. `to_i` converts floats to integers. `to_f` converts integers to floats. There you have it. All the data types of Ruby in a nutshell. Now here's that quick reference table for string methods I promised you.

18.8 Additional String Methods

```
# Outputs 1585761545
"Mary J".hash

# Outputs "concatenate"
"concat" + "enate"

# Outputs "Washington"
```

```
"washington".capitalize

# Outputs "uppercase"
"UPPERCASE".downcase

# Outputs "LOWERCASE"
"lowercase".upcase

# Outputs "Henry VII"
"Henry VIII".chop

# Outputs "rorriM"
"Mirror".reverse

# Outputs 810
"All Fears".sum

# Outputs cRaZyWaTeRs
"CrAzYwAtErS".swapcase

# Outputs "Nexu" (next advances the word up one value, as if it were
 a number.)
"Next".next

# After this, nxt == "Neyn" (to help you understand the trippiness of
 next)
nxt = "Next"
20.times {nxt = nxt.next}
```

19 Basic Ruby - Writing methods

19.1 Defining Methods

Methods are defined using the def keyword and ended with the end keyword. Some programmers find the Methods defined in Ruby very similar to those in Python[1].

```
def myMethod
end
```

To define a method that takes in a value, you can put the local variable name in parentheses after the method definition. The variable used can only be accessed from inside the method scope.

```
def myMethod(msg)
    puts msg
end
```

If multiple variables need to be used in the method, they can be separated with a comma.

```
def myMethod(msg, person)
    puts "Hi, my name is " + person + ". Some information about
myself: " + msg
end
```

Any object can be passed through using methods.

```
def myMethod(myObject)
    if(myObject.is_a?(Integer))
          puts "Your Object is an Integer"
    end
    #Check to see if it defined as an Object that we created
    #You will learn how to define Objects in a later section
    if(myObject.is_a?(MyObject))
          puts "Your Object is a MyObject"
    end
end
```

The return keyword can be used to specify that you will be returning a value from the method defined.

```
def myMethod
    return "Hello"
end
```

It is also worth noting that ruby will return the last expression evaluated, so this is functionally equivalent to the previous method.

```
def myMethod
```

1 http://en.wikibooks.org/wiki/Python%20Programming

```
        "Hello"
    end
```

Some of the Basic Operators can be overridden using the def keyword and the operator that you wish to override.

```
    def ==(oVal)
        if oVal.is_a?(Integer)
            #@value is  a variable defined in the class where this
method is defined
            #This will be covered in a later section when dealing
with Classes
            if(oVal == @value)
                return true
            else
                return false
            end
        end
    end
```

20 Basic Ruby - Classes and objects

20.1 Ruby Classes

As stated before, everything in Ruby is an object. Every object has a class. To find the class of an object, simply call that object's `class` method. For example, try this:

```
puts "This is a string".class
puts 9.class
puts ["this","is","an","array"].class
puts {:this => "is", :a => "hash"}.class
puts :symbol.class
```

Anyhow, you should already know this. What you don't know however, is how to make your own classes and extend Ruby's classes.

20.2 Creating Instances of a Class

An instance of a class is an object that has that class. For example, `"chocolate"` is an instance of the String class. You already know that you can create strings, arrays, hashes, numbers, and other built-in types by simply using quotes, brackets, curly braces, etc., but you can also create them via the `new` method. For example, `my_string = ""` is the same as `my_string = String.new`. Every class has a `new` method: arrays, hashes, integers, whatever. When you create your own classes, you'll use the `new` method to create instances.

20.3 Creating Classes

Classes represent a type of an object, such as a book, a whale, a grape, or chocolate. Everybody likes chocolate, so let's make a chocolate class:

```
class Chocolate
  def eat
    puts "That tasted great!"
  end
end
```

Let's take a look at this. Classes are created via the `class` keyword. After that comes the name of the class. All class names must start with a Capital Letter. By convention, we use CamelCase for class name. So we would create classes like PieceOfChocolate, but not like Piece_of_Chocolate.

The next section defines a class method. A class method is a method that is defined for a particular class. For example, the String class has the `length` method:

```
# outputs "5"
puts "hello".length
```

To call the `eat` method of an instance of the Chocolate class, we would use this code:

```
my_chocolate = Chocolate.new
my_chocolate.eat # outputs "That tasted great!"
```

You can also call a method by using `send`

```
"hello".send(:length) # outputs "5"
my_chocolate.send(:eat) # outputs "That tasted great!"
```

However, using `send` is rare unless you need to create a dynamic behavior, as we do not need to specify the name of the method as a literal - it can be a variable.

20.4 Self

Inside a method of a class, the pseudo-variable `self` (a pseudo-variable is one that cannot be changed) refers to the current instance. For example:

```
class Integer
  def more
    return self + 1
  end
end
3.more # -> 4
7.more # -> 8
```

20.5 Class Methods

You can also create methods that are called on a class rather than an instance. For example:

```
class Strawberry
  def Strawberry.color
    return "red"
  end

  def self.size
    return "kinda small"
  end

  class << self
    def shape
      return "strawberry-ish"
    end
  end
end
Strawberry.color # -> "red"
Strawberry.size  # -> "kinda small"
Strawberry.shape # -> "strawberry-ish"
```

Note the three different constructions: `ClassName.method_name` and `self.method_name` are essentially the same - outside of a method definition in a class block, `self` refers to the class

itself. The latter is preferred, as it makes changing the name of the class much easier. The last construction, `class << self`, puts us in the context of the class's "meta-class" (sometimes called the "eigenclass"). The meta-class is a special class that the class itself belongs to. However, at this point, you don't need to worry about it. All this construct does is allow us to define methods without the `self.` prefix.

21 Basic Ruby - Exceptions

There is no exceptions chapter at present. Instead, here is a link to a chapter about exceptions from **Yukihiro Matsumoto**'s book, *Programming Ruby: The Pragmatic Programmer's Guide* `http://www.ruby-doc.org/docs/ProgrammingRuby/html/tut_exceptions.html` More detail and simpler examples about exceptions, by **Satish Talim**, maybe found in a tutorial at RubyLearning.com `http://rubylearning.com/satishtalim/ruby_exceptions.html`

22 Syntax - Lexicology

22.1 Identifiers

An **identifier** is a name used to identify a variable, method, or class.

As with most languages, valid identifiers consist of alphanumeric characters (`A-Za-z0-9`) and underscores (_), but may not begin with a digit (`0-9`). Additionally, identifiers that are *method* names may end with a question mark (`?`), exclamation point (`!`), or equals sign (=).

There are no arbitrary restrictions to the length of an identifier (i.e. it may be as long as you like, limited only by your computer's memory). Finally, there are reserved words[1] which may not be used as identifiers.

Examples:

```
foobar
ruby_is_simple
```

22.2 Comments

Line comments run from a bare '#' character to the end of the line. There are no multi-line comments.

Examples:

```
# this line does nothing
print "Hello" # this line prints "Hello"
```

22.3 Embedded Documentation

Example:

```
=begin
Everything between a line beginning with `=begin' down to
one beginning with `=end' will be skipped by the interpreter.
These reserved words must begin in column 1.
=end
```

1 Chapter 22.4 on page 78

22.4 Reserved Words

The following words are reserved in Ruby:

```
__FILE__  and    def       end     in      or      self   unless
__LINE__  begin  defined?  ensure  module  redo    super  until
BEGIN     break  do        false   next    rescue  then   when
END       case   else      for     nil     retry   true   while
alias     class  elsif     if      not     return  undef  yield
```

You can find some examples of using them here[2].

22.5 Expressions

Example:

```
true
(1 + 2) * 3
foo()
if test then okay else not_good end
```

All variables, literals, control structures, etcetera are expressions. Using these together is called a program. You can divide expressions with newlines or semicolons (;) — however, a newline with a preceding backslash (\) is continued to the following line.

Since in Ruby control structures are expressions as well, one can do the following:

```
foo = case 1
        when 1
          true
        else
          false
        end
```

The above equivalent in a language such as C would generate a syntax error since control structures are not expressions in the C language. [3]

2 http://ruby-doc.org/docs/keywords/1.9
3 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

23 Syntax - Variables and Constants

A variable in Ruby can be distinguished by the characters at the start of its name. There's no restriction to the length of a variable's name (with the exception of the heap size).

23.1 Local Variables

Example:

```
foobar
```

A variable whose name begins with a lowercase letter (a-z) or underscore (_) is a local variable or method invocation.

A local variable is only accessible from within the block of its initialization. For example:

```
i0 = 1
loop {
  i1 = 2
  puts defined?(i0)     # true; "i0" was initialized in the ascendant
block
  puts defined?(i1)     # true; "i1" was initialized in this block
  break
}
puts defined?(i0)     # true; "i0 was initialized in this block
puts defined?(i1)     # false; "i1" was initialized in the loop
```

23.2 Instance Variables

Example:

```
@foobar
```

A variable whose name begins with '@' is an instance variable of self. An instance variable belongs to the object[1] itself. Uninitialized instance variables have a value of nil[2].

1 http://en.wikibooks.org/wiki/Ruby%20Programming%2FObject
2 http://en.wikibooks.org/wiki/Ruby%20Programming%2FObject%2FNilClass

23.3 Class Variables

A class variable is shared by all instances of a class. Example:

```
@@foobar
```

An important note is that the class variable is shared by all the descendants of the class. Example:

```
class Parent
  @@foo = "Parent"
 end
 class Thing1 < Parent
   @@foo = "Thing1"
 end
 class Thing2 < Parent
   @@foo = "Thing2"
 end
 >>Parent.class_eval("@@foo")
 =>"Thing2"
 >>Thing1.class_eval("@@foo")
 =>"Thing2"
 >>Thing2.class_eval("@@foo")
 =>"Thing2"
 >>Thing2.class_variables
 =>[]
 Parent.class_variables
 =>[:@@foo]
```

This shows us that all our classes were changing the same variable. Class variables behave like global variables which are visible only in the inheritance tree. Because Ruby resolves variables by looking up the inheritance tree *first*, this can cause problems if two subclasses both add a class variable with the same name.

23.4 Global Variables

Example:

```
$foobar
```

A variable whose name begins with '$' has a global scope; meaning it can be accessed from anywhere within the program during runtime.

23.5 Constants

Usage:

```
FOOBAR
```

A variable whose name begins with an uppercase letter (A-Z) is a constant. A constant can be reassigned a value after its initialization, but doing so will generate a warning. Every class[3] is a constant.

Trying to access an uninitialized constant raises the NameError exception.

23.5.1 How constants are looked up

Constants are looked up based on your scope. For example

```
class A
  A2 = 'a2'
  class B
    def go
      A2
    end
  end
end
instance_of_b = A::B.new
a2 = A::A2
```

23.6 Pseudo Variables

`self`

Execution context of the current method.

`nil`

The sole-instance of the `NilClass`[4] class. Expresses nothing.

`true`

The sole-instance of the `TrueClass`[5] class. Expresses true.

`false`

The sole-instance of the `FalseClass`[6] class. Expresses false.

`$1, $2 ... $9`

These are contents of capturing groups for regular expression matches. They are local to the **current thread and stack frame**!

3 http://en.wikibooks.org/wiki/Programming%3ARuby%20Object%20Class
4 http://en.wikibooks.org/wiki/Ruby%20Programming%2FObject%2FNilClass
5 http://en.wikibooks.org/wiki/Ruby%20Programming%2FObject%2FTrueClass
6 http://en.wikibooks.org/wiki/Ruby%20Programming%2FObject%2FFalseClass

(**nil** also is considered to be **false**, and every other value is considered to be **true** in Ruby.) The value of a pseudo variable cannot be changed. Substitution to a pseudo variable causes an exception to be raised.

23.7 Pre-defined Variables

Name	Aliases	Description
`$!`	`$ERROR_INFO`[7]	The exception information message set by the last 'raise' (last exception thrown).
`$@`	`$ERROR_POSITION`[8]	Array of the backtrace of the last exception thrown.
`$&`	`$MATCH`[9]	The string matched by the last successful pattern match in this scope.
`` $` ``	`$PREMATCH`[10]	The string to the left of the last successful match.
`$'`	`$POSTMATCH`[11]	The string to the right of the last successful match.
`$+`	`$LAST_PAREN_MATCH`[12]	The last bracket matched by the last successful match.
`$1` to `$9`		The Nth group of the last successful regexp match.
`$~`	`$LAST_MATCH_INFO`[13]	The information about the last match in the current scope.
`$=`	`$IGNORECASE`[14]	The flag for case insensitive, nil by default (deprecated).

7 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
8 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
9 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
10 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
11 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
12 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
13 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
14 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}

Name	Aliases	Description
$/	$INPUT_RECORD_SEPARATOR[15], $RS[16] or $-0	The input record separator, newline by default.
$\	$OUTPUT_RECORD_SEPARATOR[17] or $ORS[18]	The output record separator for the print and IO#write. Default is nil.
$,	$OUTPUT_FIELD_SEPARATOR[19] or $OFS[20]	The output field separator for the print and Array#join.
$;	$FIELD_SEPARATOR[21], $FS[22] or $-F	The default separator for String#split.
$.	$INPUT_LINE_NUMBER[23] or $NR[24]	The current input line number of the last file that was read.
$<	$DEFAULT_INPUT[25]	An object that provides access to the concatenation of the contents of all the files given as command-line arguments, or $stdin (in the case where there are no arguments). Read only.
$FILENAME		Current input file from $<. Same as $<.filename.
$>	$DEFAULT_OUTPUT[26]	The destination of output for Kernel.print and Kernel.printf. The default value is $stdout.
$_	$LAST_READ_LINE[27]	The last input line of string by gets or readline.

15 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
16 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
17 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
18 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
19 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
20 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
21 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
22 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
23 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
24 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
25 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
26 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}
27 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}

Name	Aliases	Description
`$0`		Contains the name of the script being executed. May be assignable.
`$*`	`$ARGV`[28]	Command line arguments given for the script. Also known as `ARGV`
`$$`	`$PROCESS_ID`[29], `$PID`[30] or `Process.pid`	The process number of the Ruby running this script.
`$?`	`$CHILD_STATUS`[31]	The status of the last executed child process.
`$:`	`$LOAD_PATH`	Load path for scripts and binary modules by load or require.
`$"`	`$LOADED_FEATURES` or `$-I`	The array contains the module names loaded by require.
`$stderr`		The current standard error output.
`$stdin`		The current standard input.
`$stdout`		The current standard output.
`$-d`	`$DEBUG`	The status of the -d switch. Assignable.
`$-K`	`$KCODE`	Character encoding of the source code.
`$-v`	`$VERBOSE`	The verbose flag, which is set by the -v switch.
`$-a`		True if option -a ("autosplit" mode) is set. Read-only variable.
`$-i`		If in-place-edit mode is set, this variable holds the extension, otherwise nil.
`$-l`		True if option -l is set ("line-ending processing" is on). Read-only variable.
`$-p`		True if option -p is set ("loop" mode is on). Read-only variable.
`$-w`		True if option -w is set.

The use of cryptic two-character $? expressions is a thing that people will frequently complain about, dismissing Ruby as just another perl-ish line-noise language. Keep this chart handy. Note, a lot of these are useful when working with regexp code. Part of the standard library is "`English`" which defines longer names to replace the two-character variable names to make code more readable. The

28 English.rb ^{`http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html`} from the Ruby 1.9.2 Standard Library Documentation ^{`http://ruby-doc.org/stdlib/`}

29 English.rb ^{`http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html`} from the Ruby 1.9.2 Standard Library Documentation ^{`http://ruby-doc.org/stdlib/`}

30 English.rb ^{`http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html`} from the Ruby 1.9.2 Standard Library Documentation ^{`http://ruby-doc.org/stdlib/`}

31 English.rb ^{`http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html`} from the Ruby 1.9.2 Standard Library Documentation ^{`http://ruby-doc.org/stdlib/`}

defined names are also listed in the table. To include these names, just require the English library as follows.[32]

Without 'English':

```
$\ = ' -- '
"waterbuffalo" =~ /buff/
print $", $', $$, "\n"
```

With English:

```
require "English"

$OUTPUT_FIELD_SEPARATOR = ' -- '
"waterbuffalo" =~ /buff/
print $LOADED_FEATURES, $POSTMATCH, $PID, "\n"
```

23.8 Pre-defined Constants

Note that there are some pre-defined constants at parse time, as well, namely

```
__FILE__ (current file)
```

and

```
__LINE__ (current line)
```

23.9 Notes

32 English.rb ^{http://ruby-doc.org/stdlib/libdoc/English/rdoc/index.html} from the Ruby 1.9.2 Standard Library Documentation ^{http://ruby-doc.org/stdlib/}

24 Syntax - Literals

24.1 Numerics

```
123                     # Fixnum
-123                    # Fixnum (signed)
1_123                   # Fixnum (underscore is ignored)
-543                    # Negative Fixnum
123_456_789_123_456_789 # Bignum
123.45                  # Float
1.2e-3                  # Float
0xaabb                  # (Hexadecimal) Fixnum
0377                    # (Octal) Fixnum
-0b1010                 # (Binary [negated]) Fixnum
0b001_001               # (Binary) Fixnum
?a                      # ASCII character code for 'a' (97)
?\C-a                   # Control-a (1)
?\M-a                   # Meta-a (225)
?\M-\C-a                # Meta-Control-a (129)
```

Note: the meaning of "?x" notation has been changed. In ruby 1.9 this means not an ASCII numeric code but a string i.e. ?a == "a"

24.2 Strings

Examples:

```
"this is a string"
=> "this is a string"

"three plus three is #{3+3}"
=> "three plus three is 6"

foobar = "blah"
"the value of foobar is #{foobar}"
=> "the value of foobar is blah"

'the value of foobar is #{foobar}'
=> "the value of foobar is \#{foobar}"
```

A string expression begins and ends with a double or single-quote mark. Double-quoted string expressions are subject to backslash notation and interpolation. A single-quoted string expression isn't; except for \' and \\.

24.2.1 Backslash Notation

Also called *escape characters* or *escape sequences*, they are used to insert special characters in a string.

Example:

```
"this is a\ntwo line string"
"this string has \"quotes\" in it"
```

Escape Sequence	Meaning
\n	newline (0x0a)
\s	space (0x20)
\r	carriage return (0x0d)
\t	tab (0x09)
\v	vertical tab (0x0b)
\f	formfeed (0x0c)
\b	backspace (0x08)
\a	bell/alert (0x07)
\e	escape (0x1b)
\nnn	character with octal value nnn
\xnn	character with hexadecimal value nn
\unnnn	Unicode code point U+nnnn (Ruby 1.9 and later)
\cx	control-x
\C-x	control-x
\M-x	meta-x
\M-\C-x	meta-control-x
\x	character x itself (\" a single quote, for example)

For characters with decimal values, you can do this:

```
"" << 197 # add decimal value 304 to a string
```

or embed them thus:

```
"#{197.chr}"
```

24.2.2 Interpolation

Interpolation allows Ruby code to appear within a string. The result of evaluating that code is inserted into the string:

```
"1 + 2 = #{1 + 2}"    #=> "1 + 2 = 3"
```

```
#{expression}
```

The expression can be just about any Ruby code. Ruby is pretty smart about handling string delimiters that appear in the code and it generally does what you want it to do. The code will have the same side effects as it would outside the string, including any errors:

```
"the meaning of life is #{1/0}"
=> divided by 0 (ZeroDivisionError)
```

24.2.3 The % Notation

There is also a Perl-inspired way to quote strings: by using % (percent character) and specifying a delimiting character, for example:

```
%{78% of statistics are "made up" on the spot}
=> "78% of statistics are \"made up\" on the spot"
```

Any single non-alpha-numeric character can be used as the delimiter, `%[including these]`, `%?or these?`, `%~or even these things~`. By using this notation, the usual string delimiters " and ' can appear in the string unescaped, but of course the new delimiter you've chosen does need to be escaped. However, if you use `%(parentheses)`, `%[square brackets]`, `%{curly brackets}` or `%<pointy brackets>` as delimiters then those same delimiters can appear *unescaped* in the string as long as they are in *balanced* pairs:

```
%(string (syntax) is pretty flexible)
=> "string (syntax) is pretty flexible"
```

A modifier character can appear after the %, as in %q[], %Q[], %x[] - these determine how the string is interpolated and what type of object is produced:

Modifier	Meaning
%q[ ]	Non-interpolated String (except for \\ \[and \])
%Q[ ]	Interpolated String (default)
%r[ ]	Interpolated Regexp (flags can appear after the closing delimiter)
%s[ ]	Non-interpolated Symbol
%w[ ]	Non-interpolated Array of words, separated by whitespace
%W[ ]	Interpolated Array of words, separated by whitespace
%x[ ]	Interpolated shell command

Here are some more examples:

%Q{one\ntwo\n#{ 1 + 2 }} => "one\ntwo\n3"

%q{one\ntwo\n#{ 1 + 2 }}

```
=> "one\\ntwo\\n#{ 1 + 2 }"

%r{nemo}i
```

```
=> /nemo/i

%w{one two three}
=> ["one", "two", "three"]

%x{ruby --copyright}
=> "ruby - Copyright (C) 1993-2009 Yukihiro Matsumoto\n"
```

24.2.4 "Here document" notation

There is yet another way to make a string, known as a 'here document', where the delimiter itself can be any string:

```
string = <<END
on the one ton temple bell
a moon-moth, folded into sleep,
sits still.
END
```

The syntax begins with << and is followed immediately by the delimiter. To end the string, the delimiter appears alone on a line.

There is a slightly nicer way to write a here document which allows the ending delimiter to be indented by whitespace:

```
string = <<-FIN
           on the one-ton temple bell
           a moon-moth, folded into sleep
           sits still.

           --Taniguchi Buson, 18th century; translated by X. J.
Kennedy
         FIN
```

To use non-alpha-numeric characters in the delimiter, it can be quoted:

```
string = <<-"."
           Orchid - breathing
           incense into
           butterfly's wings.

           --Matsuo Basho; translated by Lucien Stryk
         .
```

Here documents are interpolated, *unless you use single quotes around the delimiter*.

The rest of the line after the opening delimiter is not interpreted as part of the string, which means you can do this:

```
strings = [<<END, "short", "strings"]
a long string
```

```
END

=> ["a long string\n", "short", "strings"]
```

You can even "stack" multiple here documents:

```
string = [<<ONE, <<TWO, <<THREE]
  the first thing
ONE
  the second thing
TWO
  and the third thing
THREE

=> ["the first thing\n", "the second thing\n", "and the third
thing\n"]
```

24.3 Arrays

An array is a collection of objects indexed by a non-negative integer. You can create an array object by writing `Array.new`, by writing an optional comma-separated list of values inside square brackets, or if the array will only contain string objects, a space-delimited string preceded by `%w`.

```
array_one   = Array.new
array_two   = []              # shorthand for Array.new
array_three = ["a", "b", "c"] # array_three contains "a", "b" and "c"
array_four  = %w[a b c]       # array_four also contains "a", "b" and "c"
```

```
array_three[0]                # => "a"
array_three[2]                # => "c"
array_four[0]                 # => "a"
#negative indices are counted back from the end
array_four[-2]                # => "b"
#[start, count] indexing returns an array of count objects beginning at index start
array_four[1,2]               # => ["b", "c"]
#using ranges.  The end position is included with two periods but not with three
array_four[0..1]              # => ["a", "b"]
array_four[0...1]             # => ["a"]
```

The last method, using `%w`, is in essence shorthand for the `String` method `split` when the substrings are separated by whitespace only. In the following example, the first two ways of creating an array of strings are functionally identical while the last two create very different (though both valid) arrays.

```
 array_one   = %w'apple orange pear'           # => ["apple", "orange", "pear"]
 array_two   = 'apple orange pear'.split       # => ["apple", "orange", "pear"]
 array_one   == array_two                      # => true

 array_three = %w'dog:cat:bird'                # => ["dog:cat:bird"]
 array_four  = 'dog:cat:bird'.split(':')       # => ["dog", "cat", "bird"]
 array_three == array_four                     # => false
```

24.4 Hashes

Hashes are basically the same as arrays, except that a hash not only contains values, but also keys pointing to those values. Each key can occur only once in a hash. A hash object is created by writing `Hash.new` or by writing an optional list of comma-separated `key => value` pairs inside curly braces.

```
hash_one   = Hash.new
hash_two   = {}                              # shorthand for Hash.new
hash_three = {"a" => 1, "b" => 2, "c" => 3} #=> {"a"=>1, "b"=>2, "c"=>3}
```

Usually Symbols[1] are used for Hash keys (allows for quicker access), so you will see hashes declared like this:

```
hash_sym   = { :a => 1, :b => 2, :c => 3}   #=> {:b=>2, :c=>3, :a=>1}
hash_sym   = { a: 1, b: 2, c: 3}            #=> {:b=>2, :c=>3, :a=>1}
```

The latter form was introduced in Ruby 1.9.

24.4.1 Hash ordering

Note that with 1.8, iterating over hashes will iterate over the key value pairs in a "random" order. Beginning with 1.9, it will iterate over them in the order they were inserted. Note however, that if you reinsert a key (or change an existing key's value), that does not change that keys' order in the iteration.

```
>> a = {:a => 1, :b => 2, :c => 3}
=> {:a=>1, :b=>2, :c=>3}
>> a.keys # iterate over, show me the keys
=> [:a, :b, :c]
>> a[:b] = 4
> a.keys
=> [:a, :b, :c] # same order
>> a.delete(:b)
>> a[:b] = 4 # re insert now
=> 4
>> a.keys
=> [:a, :c, :b] # different order
```

24.5 Ranges

A **range** represents a subset of all possible values of a type, to be more precise, all possible values between a start value and an end value.

This may be:

1 http://en.wikibooks.org/wiki/Ruby_Literals%23Symbols

- All integers between 0 and 5.
- All numbers (including non-integers) between 0 and 1, excluding 1.
- All characters between 't' and 'y'.

In Ruby, these ranges are expressed by:

```
0..5
0.0...1.0
't'..'y'
```

Therefore, ranges consist of a start value, an end value, and whether the end value is included or not (in this short syntax, using two . for including and three . for excluding).

A range represents a set of values, not a sequence. Therefore,

```
5..0
```

though syntactically correct, produces a range of length zero.

Ranges can only be formed from instances of the same class or subclasses of a common parent, which must be Comparable (implementing <=>).

Ranges are instances of the Range class, and have certain methods, for example, to determine whether a value is inside a range:

```
r = 0..5
puts r === 4  # => true
puts r === 7  # => false
```

For detailed information of all Range methods, consult the Range class reference[2].

Here[3] is a tutorial on their use.

[4]

2 Chapter 67 on page 217
3 http://www.engineyard.com/blog/2010/iteration-shouldnt-spin-your-wheels/
4 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

25 Syntax - Operators

25.1 Operators

25.1.1 1. Assignment

Assignment in Ruby is done using the equal operator "=". This is both for variables and objects, but since strings, floats, and integers are actually objects in Ruby, you're always assigning objects.

Examples:

```
  myvar = ,myvar is now this string,
  var = 321
  dbconn = Mysql::new(,localhost,,,root,,,password,)
```

Self assignment

```
  x = 1           #=>1
  x += x          #=>2
  x -= x          #=>0
  x += 4          #=>x was 0 so x= + 4 # x is positive 4
  x *= x          #=>16
  x **= x         #=>18446744073709551616 # Raise to the power
  x /= x          #=>1
```

A frequent question from C and C++ types is "How do you increment a variable? Where are ++ and -- operators?" In Ruby, one should use x+=1 and x-=1 to increment or decrement a variable.

```
  x = ,a,
  x.succ!         #=>"b" : succ! method is defined for String, but
 not for Integer types
</sourcE>

,,,Multiple assignments,,,

Examples:
<source lang="ruby" line>
  var1, var2, var3 = 10, 20, 30
  puts var1           #=>var1 is now 10
  puts var2           #=>var2 is now 20,var3...etc

  myArray=%w(John Michel Fran Doug) # %w() can be used as syntactic
 sugar to simplify array creation
  var1,var2,var3,var4=*myArray
  puts var1           #=>John
  puts var4           #=>Doug

  names,school=myArray,,St. Whatever,
  names               #=>["John", "Michel", "Fran", "Doug"]
  school              #=>"St. Whatever"
```

Conditional assignment

```
  x = find_something() #=>nil
  x ||= "default"      #=>"default" : value of x will be replaced
 with "default", but only if x is nil or false
  x ||= "other"        #=>"default" : value of x is not replaced if
 it already is other than nil or false
```

Operator ||= is a shorthand form of the expression:

```
x = x || "default"
```

Operator ||= can be shorthand for code like:

```
  x = "(some fallback value)" unless respond_to? :x or x
```

In same way &&= operator works:

```
  x = get_node() #=>nil
  x &&= x.next_node #=> nil : x will be set to x.next_node, but only
 if x is NOT nil or false
  x = get_node() #=>Some Node
  x &&= x.next_node #=>Next Node
```

Operator &&= is a shorthand form of the expression:

```
x = x && x.get_node()
```

25.2 Scope

In Ruby there's a local scope, a global scope, an instance scope, and a class scope.

25.2.1 Local Scope

Example:

```
var=2
4.times do |x|
  puts x=x*var
end
#=>0,2,4,6
puts x #=>undefined local variable or method `x' for main:Object (NameError)
```

This error appears because this x(toplevel) is not the x(local) inside the do..end block the x(local) is a local variable to the block, whereas when trying the puts x(toplevel) we're calling a x variable that is in the top level scope, and since there's not one, Ruby protests.

25.2.2 Global scope

```
4.times do |$global|
  $global=$global*var
end  #=>0,2,4,6 last assignment of $global is 6
puts $global
#=> 6
```

This works because prefixing a variable with a dollar sign makes the variable a global.

25.2.3 Instance scope

Within methods of a class, you can share variables by prefixing them with an @.

```
class A
  def setup
    @instvar = 1
  end
  def go
    @instvar = @instvar*2
    puts @instvar
  end
end
instance = A.new
instance.setup
instance.go
#=> 2
instance.go
#=> 4
```

25.2.4 Class scope

A class variable is one that is like a "static" variable in Java. It is shared by all instances of a class.

```
class A
  @@instvar = 1
  def go
    @@instvar = @@instvar*2
    puts @@instvar
  end
end
instance = A.new
instance.go
#=> 2
instance = A.new
instance.go
#=> 4 -- variable is shared across instances
```

Here's a demo showing the various types:

```
$variable
 class Test
   def initialize(arg1='kiwi')
     @instvar=arg1
     @@classvar=@instvar+' told you so!!'
```

```
      localvar=@instvar
    end
    def print_instvar
      puts @instvar
    end
    def print_localvar
      puts @@classvar
      puts localvar
    end
  end
  var=Test.new
  var.print_instvar              #=>"kiwi", it works because a @instance_var can be
accesed inside the class
  var.print_localvar             #=>undefined local variable or method `localvar' for
#<Test:0x2b36208 @instvar="kiwi"> (NameError).
```

This will print the two lines "kiwi" and "kiwi told you so!!", then FAIL! with a undefined local variable or method `localvar' for #<Test:0x2b36208 @instvar="kiwi"> (NameError). Why? well, in the scope of the method print_localvar there doesn't exists localvar, it exists in method initialize(until GC kicks it out). On the other hand ,class variables '@@classvar' and '@instvar' are in scope across the entire class and, in the case of @@class variables, across the children classes.

```
  class SubTest < Test
    def print_classvar
      puts @@classvar
    end
  end
  newvar=SubTest.new              #newvar is created and it has @@classvar with the
same value as the var instance of Test!!
  newvar.print_classvar           #=>kiwi told you so!!
```

Class variables have the scope of parent class AND children, these variables can live across classes, and can be affected by the children actions ;-)

```
  class SubSubTest < Test
    def print_classvar
      puts @@classvar
    end
    def modify_classvar
      @@classvar='kiwi kiwi waaai!!'
    end
  end
  subtest=SubSubTest.new
  subtest.modify_classvar          #lets add a method that modifies the contents of
@@classvar in SubSubTest
  subtest.print_classvar
```

This new child of Test also has @@classvar with the original value newvar.print_classvar. The value of @@classvar has been changed to 'kiwi kiwi waaai!!' This shows that @@classvar is "shared" across parent and child classes.

25.3 Default scope

When you don't enclose your code in any scope specifier, ex:

```
  @a = 33
```

it affects the default scope, which is an object called "main".

For example, if you had one script that says

```
 @a = 33
 require 'other_script.rb'
```

and other_script.rb says

```
 puts @a
  #=> 33
```

They could share variables.

Note however, that the two scripts don't share local variables.

25.4 Local scope gotchas

Typically when you are within a class, you can do as you'd like for definitions, like.

```
class A
  a = 3
  if a == 3

    def go
       3
    end
  else
    def go
       4
    end
  end

end
```

And also, procs "bind" to their surrounding scope, like

```
 a = 3
 b = proc { a }
 b.call # 3 -- it remembered what a was
```

However, the "class" and "def" keywords cause an *entirely new* scope.

```
class A
```

```
  a = 3
  def go
     return a # this won't work!
  end
end
```

You can get around this limitation by using define_method, which takes a block and thus keeps the outer scope (note that you can use any block you want, to, too, but here's an example).

```
class A
   a = 3
  define_method(:go) {
      a
  }
end
```

Here's using an arbitrary block

```
a = 3
PROC = proc { a } # gotta use something besides a local
# variable because that "class" makes us lose scope.

class A
  define_method(:go, &PROC)
end
```

or here

```
 class A
 end
a = 3
 A.class_eval do
   define_method(:go) do
       puts a
   end
end
```

25.5 Logical And

The binary "and" operator will return the logical conjunction of its two operands. It is the same as "&&" but with a lower precedence. Example:

```
  a = 1
  b = 2
  c = nil
  puts "yay all my arguments are true" if a and b
  puts "oh no, one of my argument is false" if a and c
```

25.6 Logical Or

The binary "or" operator will return the logical disjunction of its two operands. It is the same as "||" but with a lower precedence. Example:

```
a = nil
b = "foo"
c = a || b  # c is set to "foo" its the same as saying c = (a || b)
c = a or b  # c is set to nil   its the same as saying (c = a) || b
which is not what you want.
```

1

1 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

26 Syntax - Control Structures

26.1 Control Structures

26.1.1 Conditional Branches

Ruby can control the execution of code using Conditional branches. A conditional Branch takes the result of a test expression[1] and executes a block of code depending whether the test expression is true or false. If the test expression evaluates to the constant `false` or `nil`, the test is false; otherwise, it is true. Note that the number `zero` is considered true, whereas many other programming languages consider it false.

In many popular programming languages, conditional branches are statements. They can affect which code is executed, but they do not result in values themselves. In Ruby, however, conditional branches are expressions. They evaluate to values, just like other expressions do. An `if` expression, for example, not only determines whether a subordinate block of code will execute, but also results in a value itself. For instance, the following `if` expression evaluates to 3:

```
if true
  3
end
```

Ruby's conditional branches are explained below.

if expression

Examples:

```
a = 5
if a == 4
  a = 7
end
print a # prints 5 since the if-block isn't executed
```

You can also put the test expression and code block on the same line if you use `then`:

```
if a == 4 then a = 7 end
#or
if a == 4: a = 7 end
```

1 Chapter 22.5 on page 78

```
 #Note that the ":" syntax for if one line blocks do not work
anymore in ruby 1.9. Ternary statements still work
```

This is equal to:

```
a = 5
a = 7 if a == 4
print a # prints 5 since the if-block isn't executed
```

unless expression

The unless-expression is the opposite of the if-expression, the code-block it contains will only be executed if the test expression is false.

Examples:

```
a = 5
unless a == 4
  a = 7
end
print a # prints 7 since the unless-block is executed
```

The *unless* expression is almost exactly like a negated *if* expression:

```
if !expression # is equal to using unless expression
```

The difference is that the *unless* does not permit a following *elsif*. And there is no *elsunless*.

Like the if-expression you can also write:

```
a = 5
a = 7 unless a == 4
print a # prints 7 since the unless-block is executed
```

The "one-liners" are handy when the code executed in the block is one line only.

if-elsif-else expression

The elsif (note that it's `elsif` and not `elseif`) and else blocks give you further control of your scripts by providing the option to accommodate additional tests. The `elsif` and `else` blocks are considered only if the `if` test is false. You can have any number of `elsif` blocks but only one `if` and one `else` block.

Syntax:

```
if expression
  ...code block...
elsif another expression
  ...code block...
elsif another expression
  ...code block...
else
  ...code block...
end
```

short-if expression

The "short-if" statement provides you with a space-saving way of evaluating an expression and returning a value.

The format is:

```
(condition) ? (expr if true) : (expr if false)
```

It is also known as the ternary operator and it is suggested to only use this syntax for minor tasks, such as string formatting, because of poor code readability that may result.

```
irb(main):037:0> true ? 't' : 'f'
=> "t"
irb(main):038:0> false ? 't' : 'f'
=> "f"
```

This is very useful when doing string concatenation among other things.

Example:

```
a = 5
plus_or_minus = '+'
print "The number #{a}#{plus_or_minus}1 is: " + (plus_or_minus ==
'+' ? (a+1).to_s : (a-1).to_s) + "."
```

case expression

An alternative to the if-elsif-else expression (above) is the case expression. Case in Ruby supports a number of syntaxes. For example, suppose we want to determine the relationship of a number (given by the variable a) to 5. We could say:

```
a = 1
case
  when a < 5 then puts "#{a} less than 5"
  when a == 5 then puts "#{a} equals 5"
  when a > 5 then puts "#{a} greater than 5"
end
```

Note that, as with if, the comparison operator is ==. The assignment operator is =. Although Ruby will accept the assignment operator:

```
   when a = 5 then puts "#{a} equals 5"   # WARNING! This code
CHANGES the value of a!
```

This is not want we want! Here, we want the comparison operator.

Another equivalent syntax for case is to use ":" instead of "then":

```
  when a < 5 then puts "#{a} less than 5"
```

```
  when a < 5 : puts "#{a} less than 5"
```

A more concise syntax for case is to imply the comparison:

```
  case a
    when 0..4 then puts "#{a} less than 5"
    when 5 then puts "#{a} equals 5"
    when 5..10 then puts "#{a} greater than 5"
    else puts "unexpected value #{a} "         #just in case "a" is
bigger than 10 or negative
  end
```

Note: because the ranges are explicitly stated, it is good coding practise to handle unexpected values of a. This concise syntax is perhaps most useful when we know in advance what the values to expect. For example:

```
  a = "apple"
  case a
    when "vanilla" then "a spice"
    when  "spinach" then "a vegetable"
    when "apple"  then "a fruit"
    else "an unexpected value"
  end
```

If entered in the irb this gives:

```
  => "a fruit"
```

Other ways to use case and variations on its syntax maybe seen at Linuxtopia Ruby Programming `http://www.linuxtopia.org/online_books/programming_books/ruby_tutorial/Ruby_Expressions_Case_Expressions.html`

26.1.2 Loops

while

The `while` statement in Ruby is very similar to `if` and to other languages' `while` (syntactically):

```
while <expression>
  <...code block...>
end
```

The code block will be executed again and again, as long as the expression evaluates to `true`.

Also, like `if` and `unless`, the following is possible:

```
<...code...> while <expression>
```

Note the following strange case works...

```
 line = inf.readline while line != "what I'm looking for"
```

So if local variable `line` has no existence prior to this line, on seeing it for the first time it has the value `nil` when the loop expression is first evaluated.

until

The `until` statement is similar to the `while` statement in functionality. Unlike the `while` statement, the code block for the `until` loop will execute as long as the expression evaluates to `false`.

```
until <expression>
  <...code block...>
end
```

26.1.3 Keywords

return

`return value` causes the method in which it appears to exit at that point and return the value specified

Note that if no return of a value is specified in a method the value of the last value set is implicitly returned as the return value of the method.

2

2 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

fr:Programmation_Ruby/Contrôle[3]

3 http://fr.wikibooks.org/wiki/Programmation_Ruby%2FContr%F4le

27 Syntax - Method Calls

A **method** in Ruby is a set of expressions that returns a value. Other languages sometimes refer to this as a function[1]. A method may be defined as a part of a class[2] or separately.

27.1 Method Calls

Methods are called using the following syntax:

```
method_name(parameter1, parameter2,...)
```

If the method has no parameters the parentheses can usually be omitted as in the following:

```
method_name
```

If you don't have code that needs to use method result immediately, Ruby allows to specify parameters omitting parentheses:

```
results = method_name parameter1, parameter2           # calling
 method, not using parentheses

# You need to use parentheses if you want to work with the result
 immediately.
# e.g., if a method returns an array and we want to reverse element
 order:
results = method_name(parameter1, parameter2).reverse
```

27.2 Method Definitions

Methods are defined using the keyword `def` followed by the *method name*. Method parameters[3] are specified between parentheses following the method name. The *method body* is enclosed by this definition on the top and the word `end` on the bottom. By convention method names that consist of multiple words have each word separated by an underscore.

Example:

```
def output_something(value)
  puts value
end
```

1 http://en.wikipedia.org/wiki/Subroutine
2 http://en.wikibooks.org/wiki/Ruby%20Programming%2FClasses
3 http://en.wikipedia.org/wiki/parameter%20%28computer%20science%29

27.2.1 Return Values

Methods return the value of the last statement executed. The following code returns the value `x+y`.

```
def calculate_value(x,y)
  x + y
end
```

An explicit return statement can also be used to return from function with a value, prior to the end of the function declaration. This is useful when you want to terminate a loop or return from a function as the result of a conditional expression.

Note, if you use "return" within a block, you actually will jump out from the function, probably not what you want. To terminate block, use **break**. You can pass a value to break which will be returned as the result of the block:

```
six = (1..10).each {|i| break i if i > 5}
```

In this case, six will have the value 6.

27.2.2 Default Values

A default parameter value can be specified during method definition to replace the value of a parameter if it is not passed into the method.

```
def some_method(value=,default,, arr=[])
  puts value
  puts arr.length
end

some_method(,something,)
```

The method call above will output:

```
  something
  0
```

The following is a syntax error in Ruby 1.8

```
  def foo( i = 7, j ) # Syntax error in Ruby 1.8.7 Unexpected ,),,
 expecting ,=,
    return i + j
  end
```

The above code will work in 1.9.2 and will be logically equivalent to the snippet below

```
  def foo( j, i = 7)
    return i + j
  end
```

27.2.3 Variable Length Argument List, Asterisk Operator

The last parameter of a method may be preceded by an asterisk(*), which is sometimes called the 'splat' operator. This indicates that more parameters may be passed to the function. Those parameters are collected up and an array[4] is created.

```
def calculate_value(x,y,*otherValues)
  puts otherValues
end

calculate_value(1,2,,a,,,b,,,c,)
```

In the example above the output would be ['a', 'b', 'c'].

The asterisk operator may also precede an Array argument in a method call. In this case the Array will be expanded and the values passed in as if they were separated by commas.

```
arr = [,a,,,b,,,c,]
calculate_value(*arr)
```

has the same result as:

```
calculate_value(,a,,,b,,,c,)
```

Another technique that Ruby allows is to give a Hash[5] when invoking a function, and that gives you best of all worlds: named parameters, and variable argument length.

```
def accepts_hash( var )
  print "got: ", var.inspect # will print out what it received
end

accepts_hash :arg1 => ,giving arg1,, :argN => ,giving argN,
# => got: {:argN=>"giving argN", :arg1=>"giving arg1"}
```

You see, the arguments for accepts_hash got rolled up into one hash[6] variable. This technique is heavily used in the Ruby On Rails API.

Also note the missing parenthesis around the arguments for the accepts_hash function call, and notice that there is no { } Hash declaration syntax around the :arg1 => '...' code, either. The above code is equivalent to the more verbose:

```
 accepts_hash( :arg1 => ,giving arg1,, :argN => ,giving argN, )  #
argument list enclosed in parens
 accepts_hash( { :arg1 => ,giving arg1,, :argN => ,giving argN, } )
# hash is explicitly created
```

Now, if you are going to pass a code block to function, you need parentheses.

```
 accepts_hash( :arg1 => ,giving arg1,, :argN => ,giving argN, )  {
|s| puts s }
 accepts_hash( { :arg1 => ,giving arg1,, :argN => ,giving argN, } )
{ |s| puts s }
```

4 Chapter 24.3 on page 91
5 Chapter 24.4 on page 92
6 Chapter 24.4 on page 92

```
  # second line is more verbose, hash explicitly created, but
essentially the same as above
```

27.2.4 The Ampersand Operator

Much like the asterisk, the ampersand(&) may precede the last parameter of a function declaration. This indicates that the function expects a code block to be passed in. A Proc object will be created and assigned to the parameter containing the block passed in.

Also similar to the ampersand operator, a Proc object preceded by an ampersand during a method call will be replaced by the block that it contains. `Yield`[7] may then be used.

```
def method_call
  yield
end

method_call(&someBlock)
```

27.2.5 Understanding blocks, Procs and methods

Introduction

Ruby provides the programmer with a set of very powerful features borrowed from the domain of functional programming, namely closures, higher-order functions and first-class functions [1]. These features are implemented in Ruby by means of code blocks, Proc objects and methods (that are also objects) - concepts that are closely related and yet differ in subtle ways. In fact I found myself quite confused about this topic, having difficulty understanding the difference between blocks, procs and methods and unsure about the best practices of using them. Additionally, having some background in Lisp and years of Perl experience, I was unsure of how the Ruby concepts map to similar idioms from other programming languages, like Lisp's functions and Perl's subroutines. Sifting through hundreds of newsgroup posts, I saw that I'm not the only one with this problem, and in fact quite a lot of "Ruby Nubies" struggle with the same ideas.

In this article I lay out my understanding of this facet of Ruby, which comes as a result of extensive research of Ruby books, documentation and comp.lang.ruby, in sincere hope that other people will find it useful as well.

Procs

Shamelessly ripping from the Ruby documentation, Procs are defined as follows: Proc objects are blocks of code that have been bound to a set of local variables. Once bound, the code may be called in different contexts and still access those variables.

A useful example is also provided:

7 http://en.wikibooks.org/wiki/Ruby%20Programming%2FMethod%20Calling%23yield

```
def gen_times(factor)
    return Proc.new {|n| n*factor }
end

times3 = gen_times(3)      # ,factor, is replaced with 3
times5 = gen_times(5)

times3.call(12)               #=> 36
times5.call(5)                #=> 25
times3.call(times5.call(4))   #=> 60
```

Procs play the role of functions in Ruby. It is more accurate to call them function objects, since like everything in Ruby they are objects. Such objects have a name in the folklore - functors. A functor is defined as an object to be invoked or called as if it were an ordinary function, usually with the same syntax, which is exactly what a Proc is.

From the example and the definition above, it is obvious that Ruby Procs can also act as closures. On Wikipedia, a closure is defined as a function that refers to free variables in its lexical context. Note how closely it maps to the Ruby definition blocks of code that have been bound to a set of local variables.

More on Procs

Procs in Ruby are first-class objects, since they can be created during runtime, stored in data structures, passed as arguments to other functions and returned as the value of other functions. Actually, the gen_times example demonstrates all of these criteria, except for "passed as arguments to other functions". This one can be presented as follows:

```
def foo (a, b)
    a.call(b)
end

putser = Proc.new {|x| puts x}
foo(putser, 34)
```

There is also a shorthand notation for creating Procs - the Kernel method lambda [2] (we'll come to methods shortly, but for now assume that a Kernel method is something akin to a global function, which can be called from anywhere in the code). Using lambda the Proc object creation from the previous example can be rewritten as:

```
putser = lambda {|x| puts x}
```

Actually, there are two slight differences between lambda and Proc.new. First, argument checking. The Ruby documentation for lambda states: Equivalent to Proc.new, except the resulting Proc objects check the number of parameters passed when called. Here is an example to demonstrate this:

```
pnew = Proc.new {|x, y| puts x + y}
lamb = lambda {|x, y| puts x + y}

# works fine, printing 6
pnew.call(2, 4, 11)

# throws an ArgumentError
lamb.call(2, 4, 11)
```

Second, there is a difference in the way returns are handled from the Proc. A return from Proc.new returns from the enclosing method (acting just like a return from a block, more on this later):

```
def try_ret_procnew
    ret = Proc.new { return "Baaam" }
    ret.call
    "This is not reached"
end

# prints "Baaam"
puts try_ret_procnew
```

While return from lambda acts more conventionally, returning to its caller:

```
def try_ret_lambda
    ret = lambda { return "Baaam" }
    ret.call
    "This is printed"
end

# prints "This is printed"
puts try_ret_lambda
```

With this in light, I would recommend using lambda instead of Proc.new, unless the behavior of the latter is strictly required. In addition to being a whopping two characters shorter, its behavior is less surprising.

Methods

Simply put, a method is also a block of code. However, unlike Procs, methods are not bound to the local variables around them. Rather, they are bound to some object and have access to its instance variables [3]:

```
class Boogy
    def initialize
        @dix = 15
    end

    def arbo
        puts "#{@dix} ha\n"
    end
end

# initializes an instance of Boogy
b = Boogy.new

# prints "15 ha"
b.arbo
```

A useful idiom when thinking about methods is sending messages. Given a receiver - an object that has some method defined, we can send it a message - by calling the method, optionally providing some arguments. In the example above, calling arbo is akin to sending a message “arbo”, without arguments. Ruby supports the message sending idiom more directly, by including the send method in class Object (which is the parent of all objects in Ruby). So the following two lines are equivalent to the arbo method call:

```
# method/message name is given as a string
```

```
b.send("arbo")

# method/message name is given as a symbol
b.send(:arbo)
```

Note that methods can also be defined in the "top-level" scope, not inside any class. For example:

```
def say (something)
    puts something
end

say "Hello"
```

While it seems that say is "free-standing", it is not. When methods such as this are defined, Ruby silently tucks them into the Object class. But this doesn't really matter, and for all practical purposes say can be seen as an independent method. Which is, by the way, just what's called a "function" in some languages (like C and Perl). The following Proc is, in many ways similar:

```
say = lambda {|something| puts something}

say.call("Hello")

# same effect
say["Hello"]
```

The [] construct is a synonym to call in the context of Proc [4]. Methods, however, are more versatile than procs and support a very important feature of Ruby, which I will present right after explaining what blocks are.

Blocks

Blocks are so powerfully related to Procs that it gives many newbies a headache trying to decipher how they actually differ. I will try to ease on comprehension with a (hopefully not too corny) metaphor. Blocks, as I see them, are unborn Procs. Blocks are the larval, Procs are the insects. A block does not live on its own - it prepares the code for when it will actually become alive, and only when it is bound and converted to a Proc, it starts living:

```
# a naked block can,t live in Ruby
# this is a compilation error !
{puts "hello"}

# now it,s alive, having been converted
# to a Proc !
pr = lambda {puts "hello"}

pr.call
```

Is that it, is that what all the fuss is about, then ? No, not at all. The designer of Ruby, Matz saw that while passing Procs to methods (and other Procs) is nice and allows high-level functions and all kinds of fancy functional stuff, there is one common case that stands high above all other cases - passing a single block of code to a method that makes something useful out of it, for example iteration. And as a very talented designer, Matz decided that it is worthwhile to emphasize this special case, and make it both simpler and more efficient.

Passing a block to a method

No doubt that any programmer who has spent at least a couple of hours with Ruby has been shown the following examples of Ruby glory (or something very similar):

```
10.times do |i|
    print "#{i} "
end

numbers = [1, 2, 5, 6, 9, 21]

numbers.each do |x|
    puts "#{x} is " + (x >= 3 ? "many" : "few")
end

squares = numbers.map {|x| x * x}
```

(Note that `do |x| ... end` is equivalent to `{ |x| ... }`.)

Such code is IMHO part of what makes Ruby the clean, readable and wonderful language it is. What happens here behind the scenes is quite simple, or at least may be depicted in a very simple way. Perhaps Ruby doesn't implement it exactly the way I'm going to describe it, since there are optimization considerations surely playing their role - but it is definitely close enough to the truth to serve as a metaphor for understanding.

Whenever a block is appended to a method call, Ruby automatically converts it to a Proc object, but one without an explicit name. The method, however, has a way to access this Proc, by means of the yield statement. See the following example for clarification:

```
def do_twice
    yield
    yield
end

do_twice {puts "Hola"}
```

The method do_twice is defined and called with an attached block. Although the method didn't explicitly ask for the block in its arguments list, the yield can call the block. This can be implemented in a more explicit way, using a Proc argument:

```
def do_twice(what)
    what.call
    what.call
end

do_twice lambda {puts "Hola"}
```

This is equivalent to the previous example, but using blocks with yield is cleaner, and better optimized since only one block is passed to the method, for sure. Using the Proc approach, any amount of code blocks can be passed:

```
def do_twice(what1, what2, what3)
    2.times do
        what1.call
        what2.call
        what3.call
    end
end
```

```
do_twice(   lambda {print "Hola, "},
            lambda {print "querido "},
            lambda {print "amigo\n"})
```

It is important to note that many people frown at passing blocks, and prefer explicit Procs instead. Their rationale is that a block argument is implicit, and one has to look through the whole code of the method to see if there are any calls to yield there, while a Proc is explicit and can be immediately spotted in the argument list. While it's simply a matter of taste, understanding both approaches is vital.

The ampersand (&)

The ampersand operator can be used to explicitly convert between blocks and Procs in a couple of cases. It is worthy to understand how these work.

Remember how I said that although an attached block is converted to a Proc under the hood, it is not accessible as a Proc from inside the method ? Well, if an ampersand is prepended to the last argument in the argument list of a method, the block attached to this method is converted to a Proc object and gets assigned to that last argument:

```
def contrived(a, &f)
    # the block can be accessed through f
    f.call(a)

    # but yield also works !
    yield(a)
end

# this works
contrived(25) {|x| puts x}

# this raises ArgumentError, because &f
# isn,t really an argument - it,s only there
# to convert a block
contrived(25, lambda {|x| puts x})
```

Another (IMHO far more efficacious) use of the ampersand is the other-way conversion - converting a Proc into a block. This is very useful because many of Ruby's great built-ins, and especially the iterators, expect to receive a block as an argument, and sometimes it's much more convenient to pass them a Proc. The following example is taken right from the excellent "Programming Ruby" book by the pragmatic programmers:

```
print "(t)imes or (p)lus: "
times = gets
print "number: "
number = Integer(gets)
if times =~ /^t/
    calc = lambda {|n| n*number }
else
    calc = lambda {|n| n+number }
end
puts((1..10).collect(&calc).join(", "))
```

The collect method expects a block, but in this case it is very convenient to provide it with a Proc, since the Proc is constructed using knowledge gained from the user. The ampersand preceding calc makes sure that the Proc object calc is turned into a code block and is passed to collect as an attached block.

The ampersand also allows the implementation of a very common idiom among Ruby programmers: passing method names into iterators. Assume that I want to convert all words in an Array to upper case. I could do it like this:

```
words = %w(Jane, aara, multiko)
upcase_words = words.map {|x| x.upcase}

p upcase_words
```

This is nice, and it works, but I feel it's a little bit too verbose. The upcase method itself should be given to map, without the need for a separate block and the apparently superfluous x argument. Fortunately, as we saw before, Ruby supports the idiom of sending messages to objects, and methods can be referred to by their names, which are implemented as Ruby Symbols. For example:

```
p "Erik".send(:upcase)
```

This, quite literally, says send the message/method upcase to the object "Erik". This feature can be utilized to implement the map {|x| x.upcase} in an elegant manner, and we're going to use the ampersand for this ! As I said, when the ampersand is prepended to some Proc in a method call, it converts the Proc to a block. But what if we prepend it not to a Proc, but to another object ? Then, Ruby's implicit type conversion rules kick in, and the to_proc method is called on the object to try and make a Proc out of it. We can use this to implement to_proc for Symbol and achieve what we want:

```
class Symbol

    # A generalized conversion of a method name
    # to a proc that runs this method.
    #
    def to_proc
        lambda {|x, *args| x.send(self, *args)}
    end

end

# Voila !
words = %w(Jane, aara, multiko)
upcase_words = words.map(&:upcase)
```

27.3 Dynamic methods

You can define a method on "just one object" in Ruby.

```
a = ,b,
def a.some_method
  ,within a singleton method just for a,
end
>> a.some_method
=> ,within a singleton method just for a,
```

Or you can use define_method, which preserves the scope around the definition, as well.

```
a = ,b,
a.class.send(:define_method, :some_method) { # only available to
classes, unfortunately
```

```
  ,within a block method,
}
a.some_method
```

27.4 Special methods

Ruby has a number of special methods that are called by the interpreter. For example:

```
class Chameleon
  alias __inspect__ inspect
  def method_missing(method, *arg)
    if (method.to_s)[0..2] == "to_"
      @identity = __inspect__.sub("Chameleon",
method.to_s.sub(,to_,,,,).capitalize)
      def inspect
        @identity
      end
      self
    else
      super #method_missing overrides the default
Kernel.method_missing
            #pass on anything we weren,t looking for so the
Chameleon stays unnoticed and uneaten ;)
    end
  end
end
mrlizard = Chameleon.new
mrlizard.to_rock
```

This does something silly but `method_missing` is an important part of meta-programming in Ruby. In Ruby on Rails it is used extensively to create methods dynamically.

Another special methods is `initialize` that ruby calls whenever a class instance is created, but that belongs in the next chapter: Classes[8].

27.5 Conclusion

Ruby doesn't really have functions. Rather, it has two slightly different concepts - methods and Procs (which are, as we have seen, simply what other languages call function objects, or functors). Both are blocks of code - methods are bound to Objects, and Procs are bound to the local variables in scope. Their uses are quite different.

Methods are the cornerstone of object-oriented programming, and since Ruby is a pure-OO language (everything is an object), methods are inherent to the nature of Ruby. Methods are the actions Ruby objects do - the messages they receive, if you prefer the message sending idiom.

Procs make powerful functional programming paradigms possible, turning code into a first-class object of Ruby allowing to implement high-order functions. They are very close kin to Lisp's lambda forms (there's little doubt about the origin of Ruby's Proc constructor lambda)

The construct of a block may at first be confusing, but it turns out to be quite simple. A block is, as my metaphor goes, an unborn Proc - it is a Proc in an intermediate state, not bound to anything yet. I

8 Chapter 20 on page 71

think that the simplest way to think about blocks in Ruby, without losing any comprehension, would be to think that blocks are really a form of Procs, and not a separate concept. The only time when we have to think of blocks as slightly different from Procs is the special case when they are passed as the last argument to a method which may then access them using yield.

That's about it, I guess. I know for sure that the research I conducted for this article cleared many misunderstandings I had about the concepts presented here. I hope others will learn from it as well. If you see anything you don't agree with - from glaring errors to small inaccuracies, feel free to amend the book.

Notes

[1] It seems that in the pure, theoretical interpretation what Ruby has isn't first-class functions per se. However, as this article demonstrates, Ruby is perfectly capable of fulfilling most of the requirements for first-class functions, namely that functions can be created during the execution of a program, stored in data structures, passed as arguments to other functions, and returned as the values of other functions.

[2] lambda has a synonym - proc, which is considered 'mildly deprecated' (mainly because proc and Proc.new are slightly different, which is confusing). In other words, just use lambda.

[3] These are 'instance methods'. Ruby also supports 'class methods', and 'class variables', but that is not what this article is about.

[4] Or more accurately, call and [] both refer to the same method of class Proc. Yes, Proc objects themselves have methods !

9

9 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

28 Syntax - Classes

Classes[1] are the basic template from which object[2] instances are created. A class is made up of a collection of variables representing internal state and methods providing behaviours that operate on that state.

28.1 Class Definition

Classes are defined in Ruby using the `class` keyword followed by a name. The name must begin with a capital letter and by convention names that contain more than one word are run together with each word capitalized and no separating characters (CamelCase[3]). The class definition may contain method, class variable, and instance variable declarations as well as calls to methods that execute in the class context at read time, such as attr_accessor. The class declaration is terminated by the `end` keyword.

Example:

```
class MyClass
  def some_method
  end
end
```

28.1.1 Instance Variables

Instance variables are created for each class instance and are accessible only within that instance. They are accessed using the @ operator. Outside of the class definition, the value of an instance variable can only be read or modified via that instance's public methods.

Example:

```
class MyClass
  @one = 1
  def do_something
    @one = 2
  end
  def output
    puts @one
  end
end
instance = MyClass.new
instance.output
```

1 http://en.wikipedia.org/wiki/class%20%28object-oriented%20programming%29
2 http://en.wikipedia.org/wiki/Object%20%28computer%20science%29
3 http://en.wikipedia.org/wiki/CamelCase

```
instance.do_something
instance.output
```

Surprisingly, this outputs:

```
nil
2
```

This happens (nil in the first output line) because @*one* defined below *class MyClass* is an instance variable belonging to the class object (note this is not the same as a class variable and could not be referred to as @@*one*), whereas @*one* defined inside the *do_something* method is an instance variable belonging to instances of *MyClass*. They are two distinct variables and the first is accessible only in a class method.

28.1.2 Accessor Methods

As noted in the previous section, an instance variable can only be directly accessed or modified within an instance method definition. If you want to provide access to it from outside, you need to define public accessor methods, for example

```
class MyClass
  def initialize
    @foo = 28
  end

  def foo
    return @foo
  end

  def foo=(value)
    @foo = value
  end
end

instance = MyClass.new
puts instance.foo
instance.foo = 496
puts instance.foo
```

Note that ruby provides a bit of syntactic sugar to make it look like you are getting and setting a variable directly; under the hood

```
a = instance.foo
instance.foo = b
```

are calls to the foo and foo= methods

```
a = instance.foo()
instance.foo=(b)
```

Since this is such a common use case, there is also a convenience method to autogenerate these getters and setters:

```
class MyClass
  attr_accessor :foo
```

```
  def initialize
    @foo = 28
  end
end

instance = MyClass.new
puts instance.foo
instance.foo = 496
puts instance.foo
```

does the same thing as the above program. The attr_accessor method is run at read time, when ruby is constructing the class object, and it generates the foo and foo= methods.

However, there is no requirement for the accessor methods to simply transparently access the instance variable. For example, we could ensure that all values are rounded before being stored in foo:

```
class MyClass
  def initialize
    @foo = 28
  end

  def foo
    return @foo
  end

  def foo=(value)
    @foo = value.round
  end
end

instance = MyClass.new
puts instance.foo
instance.foo = 496.2
puts instance.foo #=> 496
```

28.1.3 Class Variables

Class variables are accessed using the @@ operator. These variables are associated with the class hierarchy rather than any object instance of the class and are the same across all object instances. (These are similar to class "static" variables in Java or C++).

Example:

```
class MyClass
  @@value = 1
  def add_one
    @@value= @@value + 1
  end

  def value
    @@value
  end
end
instanceOne = MyClass.new
instanceTwo = MyClass.new
puts instanceOne.value
instanceOne.add_one
puts instanceOne.value
puts instanceTwo.value
```

Outputs:

```
1
2
2
```

28.1.4 Class Instance Variables

Classes can have instance variables. This gives each class a variable that is not shared by other classes in the inheritance chain.

```
class Employee
  class << self; attr_accessor :instances; end
  def store
    self.class.instances ||= []
    self.class.instances << self
  end
  def initialize name
    @name = name
  end
end
class Overhead < Employee; end
class Programmer < Employee; end
Overhead.new(,Martin,).store
Overhead.new(,Roy,).store
Programmer.new(,Erik,).store
puts Overhead.instances.size    # => 2
puts Programmer.instances.size  # => 1
```

For more details, see MF Bliki: ClassInstanceVariables[4]

28.1.5 Class Methods

Class methods are declared the same way as normal methods, except that they are prefixed by `self`, or the class name, followed by a period. These methods are executed at the Class level and may be called without an object instance. They cannot access instance variables but do have access to class variables.

Example:

```
class MyClass
  def self.some_method
    puts ,something,
  end
end
MyClass.some_method
```

Outputs:

```
something
```

4 http://martinfowler.com/bliki/ClassInstanceVariable.html

28.1.6 Instantiation

An object instance is created from a class through the a process called *instantiation*. In Ruby this takes place through the Class method `new`.

Example:

```
anObject = MyClass.new(parameters)
```

This function sets up the object in memory and then delegates control to the initialize function of the class if it is present. Parameters passed to the new function are passed into the `initialize` function.

```
class MyClass
  def initialize(parameters)
  end
end
```

28.2 Declaring Visibility

By default, all methods in Ruby classes are public - accessible by anyone. There are, nonetheless, only two exceptions for this rule: the global methods defined under the Object class, and the initialize method for any class. Both of them are implicitly private.

If desired, the access for methods can be restricted by public, private, protected object methods.

It is interesting that these are not actually keywords, but actual methods that operate on the class, dynamically altering the visibility of the methods, and as a result, these 'keywords' influence the visibility of all following declarations until a new visibility is set or the end of the declaration-body is reached.

28.2.1 Private

Simple example:

```
class Example
  def methodA
  end

  private # all methods that follow will be made private: not
accessible for outside objects

  def methodP
  end
end
```

If private is invoked without arguments, it sets access to private for all subsequent methods. It can also be invoked with named arguments.

Named private method example:

```
class Example
  def methodA
```

```
  end

  def methodP
  end

  private :methodP
end
```

Here private was invoked with an argument, altering the visibility of methodP to private.

Note for class methods (those that are declared using def ClassName.method_name), you need to use another function: private_class_method

Common usage of private_class_method is to make the "new" method (constructor) inaccessible, to force access to an object through some getter function. A typical Singleton implementation is an obvious example.

```
class SingletonLike
  private_class_method :new

  def SingletonLike.create(*args, &block)
    @@inst = new(*args, &block) unless @@inst
    return @@inst
  end
end
```

Note : another popular way to code the same declaration

```
class SingletonLike
  private_class_method :new

  def SingletonLike.create(*args, &block)
    @@inst ||= new(*args, &block)
  end
end
```

More info about the difference between C++ and Ruby private/protected: `http://lylejohnson.name/blog/?p=5`

One person summed up the distinctions by saying that in C++, “private” means “private to this class”, while in Ruby it means “private to this instance”. What this means, in C++ from code in class A, you can access any private method for any other object of type A. In Ruby, you can not: you can only access private methods for your instance of object, and not for any other object instance (of class A).

Ruby folks keep saying "private means you cannot specify the receiver". What they are saying, if method is private, in your code you can say:

```
class AccessPrivate
  def a
  end
  private :a # a is private method

  def accessing_private
    a              # sure!
    self.a         # nope! private methods cannot be called with an
explicit receiver at all, even if that receiver is "self"
    other_object.a # nope, a is private, you can,t get it (but if
it was protected, you could!)
  end
end
```

Here, "other_object" is the "receiver" that method "a" is invoked on. For private methods, it does not work. However, that is what "protected" visibility will allow.

28.2.2 Public

Public is default accessibility level for class methods. I am not sure why this is specified - maybe for completeness, maybe so that you could dynamically make some method private at some point, and later - public.

In Ruby, visibility is completely dynamic. You can change method visibility at runtime!

28.2.3 Protected

Now, “protected” deserves more discussion. Those of you coming from Java or C++ have learned that in those languages, if a method is “private”, its visibility is restricted to the declaring class, and if the method is “protected”, it will be accessible to children of the class (classes that inherit from parent) or other classes in that package.

In Ruby, “private” visibility is similar to what “protected” is in Java. Private methods in Ruby are accessible from children. You can’t have truly private methods in Ruby; you can’t completely hide a method.

The difference between protected and private is subtle. If a method is protected, it may be called by any instance of the defining class or its subclasses. If a method is private, it may be called only within the context of the calling object---it is never possible to access another object instance's private methods directly, even if the object is of the same class as the caller. For protected methods, they are accessible from objects of the same class (or children).

So, from within an object "a1" (an instance of Class A), you can call private methods only for instance of "a1" (self). And you can not call private methods of object "a2" (that also is of class A) - they are private to a2. But you can call protected methods of object "a2" since objects a1 and a2 are both of class A.

Ruby FAQ[5] gives following example - implementing an operator that compares one internal variable with a variable from another class (for purposes of comparing the objects):

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

If age is private, this method will not work, because other.age is not accessible. If "age" is protected, this will work fine, because self and other are of same class, and can access each other's protected methods.

To think of this, protected actually reminds me of the "internal" accessibility modifier in C# or "default" accessibility in Java (when no accessibility keword is set on method or variable): method is accessible just as "public", but only for classes inside the same package.

5 http://www.rubycentral.com/faq/rubyfaq-7.html

28.2.4 Instance Variables

Note that object instance variables are not really private, you just can't see them. To access an instance variable, you need to create a getter and setter.

Like this (no, don't do this by hand! See below):

```
class GotAccessor
  def initialize(size)
    @size = size
  end

  def size
    @size
  end
  def size=(val)
    @size = val
  end
end

# you could the access @size variable as
# a = GotAccessor.new(5)
# x = a.size
# a.size = y
```

Luckily, we have special functions to do just that: attr_accessor, attr_reader, attr_writer. attr_accessor will give you get/set functionality, reader will give only getter and writer will give only setter.

Now reduced to:

```
class GotAccessor
  def initialize(size)
    @size = size
  end

  attr_accessor :size
end

# attr_accessor generates variable @size accessor methods
automatically:
# a = GotAccessor.new(5)
# x = a.size
# a.size = y
```

28.3 Inheritance

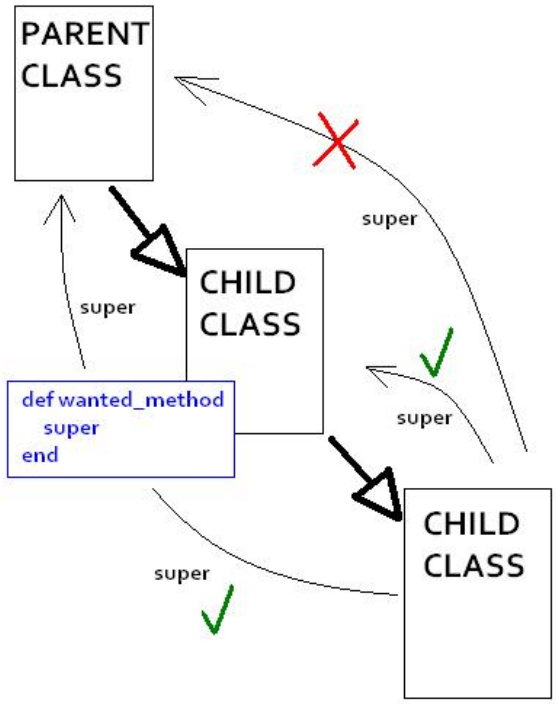

Figure 6 The *super* keyword only accessing the direct parents method. There is a workaround though.

A class can *inherit* functionality and variables from a *superclass*, sometimes referred to as a *parent class* or *base class*. Ruby does not support *multiple inheritance* and so a class in Ruby can have only one superclass. The syntax is as follows:

```
  class ParentClass
    def a_method
      puts ,b,
    end
  end

  class SomeClass < ParentClass  # < means inherit (or "extends" if
you are from Java background)
    def another_method
      puts ,a,
    end
  end

  instance = SomeClass.new
  instance.another_method
  instance.a_method
```

Outputs:

```
a
b
```

All non-private variables and functions are inherited by the child class from the superclass.

If your class overrides a method from parent class (superclass), you still can access the parent's method by using 'super' keyword.

```
class ParentClass
  def a_method
    puts ,b,
  end
end

class SomeClass < ParentClass
  def a_method
    super
    puts ,a,
  end
end

instance = SomeClass.new
instance.a_method
```

Outputs:

```
b
a
```

(because a_method also did invoke the method from parent class).

If you have a deep inheritance line, and still want to access some parent class (superclass) methods directly, you can't. *super* only gets you a direct parent's method. But there is a workaround! When inheriting from a class, you can alias parent class method to a different name. Then you can access methods by alias.

```
class X
  def foo
    "hello"
  end
end

class Y < X
  alias xFoo foo
  def foo
    xFoo + "y"
  end
end

class Z < Y
  def foo
    xFoo + "z"
  end
end

puts X.new.foo
puts Y.new.foo
puts Z.new.foo
```

Outputs

```
   hello
   helloy
   helloz
```

28.4 Mixing in Modules

First, you need to read up on modules. Modules are a way of grouping together some functions and variables and classes, somewhat like classes, but more like namespaces. So a module is not really a class. You can't instantiate a Module, and thus it does not have *self*.

This trait, however, allows us to include the module into a class. Mix it in, so to speak.

```
module A
  def a1
    puts ,a1 is called,
  end
end

module B
  def b1
    puts ,b1 is called,
  end
end

module C
  def c1
    puts ,c1 is called,
  end
end

class Test
  include A
  include B
  include C

  def display
    puts ,Modules are included,
  end
end

object=Test.new
object.display
object.a1
object.b1
object.c1
```

Outputs:

```
Modules are included
a1 is called
b1 is called
c1 is called
```

The code shows Multiple Inheritance using modules.

28.5 Ruby Class Meta-Model

In keeping with the Ruby principle that everything is an object, classes are themselves instances of the class Class. They are stored in constants under the scope of the module[6] in which they are declared. A call to a method on an object instance is delegated to a variable inside the object that contains a reference to the class of that object. The method implementation exists on the Class instance object itself. Class methods are implemented on meta-classes that are linked to the existing class instance objects in the same way that those classes instances are linked to them. These meta-classes are hidden from most Ruby functions.

[7]

6 http://en.wikibooks.org/wiki/Module
7 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

29 Syntax - Hooks

Ruby provides callbacks, to, for example, know when a new method is defined (later) in a class. Here[1] is a list of known callbacks.

29.1 const_missing

```
class Object
  def self.const_missing c
    p 'missing const was', c
  end
end
```

or more

```
class Object
  class << self
    alias :const_missing_old :const_missing
    def const_missing c
      p 'const missing is', c
      const_missing_old c
    end
  end
end
```

See also some rdoc[2] documentation on the various keywords.

1 http://www.nach-vorne.de/2007/3/18/list-of-callback-methods/
2 http://ruby-doc.org/docs/keywords/1.9/

30 References

31 Built-In Functions

By default many methods are available. You can see the available ones by running `methods` in an irb session, ex:

```
>> class A; end
>> A.singleton_methods

[:nil?, :===, :=~, :!~, :eql?, :class, :clone, :dup, :taint,
 :tainted?, :untaint, :untrust, :untrusted?, :trust, :freeze,
 :frozen?, :to_s, :inspect, :methods, :singleton_methods,
 :protected_methods, :private_methods, :public_methods,
 :instance_variables, :instance_variable_get, :instance_variable_set,
 :instance_variable_defined?, :instance_of?, :kind_of?, :is_a?, :tap,
 :send, :public_send, :respond_to?, :extend, :display, :method,
 :public_method, :define_singleton_method, :hash, :__id__, :object_id,
 :to_enum, :enum_for, :gem, :==, :equal?, :!, :!=, :instance_eval,
 :instance_exec, :__send__]
```

You can see where most of those methods are defined by inspecting the object hierarchy:

```
>> A.ancestors
=> [A, Object, Kernel, BasicObject]
```

1.9 introduced a few more has `__method__` (the current method name), as well as `require_relative`, which requires a file relative to the dir of the current file.

To see where each methods was defined, you can run something like:

```
>> A.instance_methods.map{|m| [m,  A.instance_method(m).owner] }
=> [[:nil?, Kernel], ...
```

32 Predefined Variables

Ruby's predefined (built-in) variables affect the behavior of the entire program, so their use in libraries isn't recommended.

The values in most predefined variables can be accessed by alternative means.

`$!`

- The last exception object raised. The exception object can also be accessed using => in rescue clause.

`$@`

- The stack backtrace for the last exception raised. The stack backtrace information can retrieved by Exception#backtrace method of the last exception.

`$/`

- The input record separator (newline by default). gets, readline, etc., take their input record separator as optional argument.

`$\`

- The output record separator (nil by default).

`$,`

- The output separator between the arguments to print and Array#join (nil by default). You can specify separator explicitly to Array#join.

`$;`

- The default separator for split (nil by default). You can specify separator explicitly for String#split.

`$.`

- The number of the last line read from the current input file. Equivalent to ARGF.lineno.

`$<`

- Synonym for ARGF.

`$>`

- Synonym for $defout.

`$0`

- The name of the current Ruby program being executed.

`$$`

- The process.pid of the current Ruby program being executed.

`$?`

- The exit status of the last process terminated.

`$:`

- Synonym for $LOAD_PATH.

`$DEBUG`

- True if the -d or --debug command-line option is specified.

`$defout`

- The destination output for print and printf ($stdout by default).

`$F`

- The variable that receives the output from split when -a is specified. This variable is set if the -a command-line option is specified along with the -p or -n option.

`$FILENAME`

- The name of the file currently being read from ARGF. Equivalent to ARGF.filename.

`$LOAD_PATH`

- An array holding the directories to be searched when loading files with the load and require methods.

`$SAFE`

- The security level.

```
   0   No checks are performed on externally supplied (tainted)
data. (default)

   1   Potentially dangerous operations using tainted data are
forbidden.

   2   Potentially dangerous operations on processes and files are
forbidden.

   3   All newly created objects are considered tainted.

   4   Modification of global data is forbidden.
```

`$stdin`

- Standard input (STDIN by default).

`$stdout`

- Standard output (STDOUT by default).

`$stderr`

- Standard error (STDERR by default).

`$VERBOSE`

- True if the -v, -w, or --verbose command-line option is specified.

`$- x`

- The value of interpreter option -x (x=0, a, d, F, i, K, l, p, v).

The following are local variables:

`$_`

- The last string read by gets or readline in the current scope.

`$~`

- MatchData relating to the last match. Regex#match method returns the last match information.

The following variables hold values that change in accordance with the current value of $~ and can't receive assignment:

`$ n ($1, $2, $3...)`

- The string matched in the nth group of the last pattern match. Equivalent to m[n], where m is a MatchData object.

`$&`

- The string matched in the last pattern match. Equivalent to m[0], where m is a MatchData object.

`$`` ` ``

- The string preceding the match in the last pattern match. Equivalent to m.pre_match, where m is a MatchData object.

`$'`

- The string following the match in the last pattern match. Equivalent to m.post_match, where m is a MatchData object.

`$+`

- The string corresponding to the last successfully matched group in the last pattern match.

33 Predefined Classes

In ruby even the **base types** (also **predefined classes**) can be hacked.[quirks] In the following example, `5` is an immediate,[immediate] a literal, an object, and an instance of `Fixnum`.

```
class Fixnum

        alias other_s to_s
        def to_s()
                a = self + 5
                return a.other_s
        end

end


a = 5
puts a.class  ## prints Fixnum
puts a        ## prints 10 (adds 5 once)
puts 0        ## prints 5  (adds 5 once)
puts 5        ## prints 10 (adds 5 once)
puts 10 + 0   ## prints 15 (adds 5 once)

b = 5+5
puts b        ## puts 15 (adds 5 once)
```

33.1 Footnotes

1. [immediate] Which means the 4 VALUE bytes are not a reference but the value itself. All `5` have the same object id (which could also be achieved in other ways).
2. [quirks] Might not always work as you would like, the base types don't have a constructor (`def initialize`), and can't have simpleton methods. There are some other minor exceptions.

34 Objects

Object is the base class of all other classes created in Ruby. It provides the basic set of functions available to all classes, and each function can be explicitly overridden by the user.

This class provides a number of useful methods for all of the classes in Ruby.

[1]

1 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

35 Array

Class Methods

```
Method: [ ]     Signature: Array[ [anObject]* ] -> anArray
```

Creates a new array whose elements are given between [and].

```
Array.[]( 1, 2, 3 ) -> [1,2,3]
Array[ 1, 2, 3 ] -> [1,2,3]
[1,2,3] -> [1,2,3]
```

1

1 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

36 Class

37 Comparable

38 Encoding

Encoding is basically a new concept introduced in Ruby 1.9

Strings now have "some bytes" and "some encoding" associated with them.

In 1.8, all strings were just "some bytes" (so basically treated as what BINARY/ASCII-8BIT encoding is in 1.9). You had to use helper libraries to use any m18n style stuff.

By default, when you open a file and read from it, it will read as strings with an encoding set to

`Encoding.default_external` (which you can change).

This also means that it double checks the strings "a bit" for correctness on their way in.

In windows, it also has to do conversion from "\r\n" to "\n" which means that when you read a file in non-binary mode in windows, it has to first analyze the incoming string for correctness, then (second pass) convert its line endings, so it is a bit slower.

Recommend 1.9.2 for windows users loading large files, as it isn't quite as slow. Or read them as binary (File.binread or a=File.open('name', 'rb').

38.0.1 External links

here[1] is one good tutorial. here[2] is another. `http://github.com/candlerb/string19/blob/master/string19.rb` is another.

1 `http://blog.grayproductions.net/articles/ruby_19s_three_default_encodings`
2 `http://yehudakatz.com/2010/05/17/encodings-unabridged/`

39 Enumerable

39.1 Enumerable

Enumerator appears in Ruby as Enumerable::Enumerator in 1.8.x and (just) Enumerator in 1.9.x.

40 Forms of Enumerator

There are several different ways in which an Enumerator can be used:

- As a proxy for “each”
- As a source of values from a block
- As an external iterator

40.1 1. As a proxy for “each”

This is the first way of using Enumerator, introduced in ruby 1.8. It solves the following problem: Enumerable methods like #map and #select call #each on your object, but what if you want to iterate using some other method such as #each_byte or #each_with_index?

An Enumerator is a simple proxy object which takes a call to #each and redirects it to a different method on the underlying object.

```
require 'enumerator'   # needed in ruby <= 1.8.6 only

src = "hello"
puts src.enum_for(:each_byte).map { |b| "%02x" % b }.join(" ")
```

The call to ‘enum_for’ (or equivalently ‘to_enum’) creates the Enumerator proxy. It is a shorthand for the following:

```
newsrc = Enumerable::Enumerator.new(src, :each_byte)
puts newsrc.map { |b| "%02x" % b }.join(" ")
```

In ruby 1.9, `Enumerable::Enumerator` has changed to just `Enumerator`

40.2 2. As a source of values from a block

In ruby 1.9, Enumerator.new can instead take a block which is executed when #each is called, and directly yields the values.

```
block =  Enumerator.new {|g| g.yield 1; g.yield 2; g.yield 3}

block.each do |item|
  puts item
end
```

“g << 1” is an alternative syntax for “g.yield 1”

No fancy language features such as Fiber or Continuation are used, and this form of Enumerator is easily retro-fitted to ruby 1.8[1]

It is quite similar to creating your own object which yields values:

```
block = Object.new
def block.each
  yield 1; yield 2; yield 3
end

block.each do |item|
  puts item
end
```

However it also lays the groundwork for "lazy" evaluation of enumerables, described later.

40.3 3. As an external iterator

ruby 1.9 also allows you turn an Enumerator around so that it becomes a "pull" source of values, sometimes known as "external iteration". Look carefully at the difference between this and the previous example:

```
block =  Enumerator.new {|g| g.yield 1; g.yield 2; g.yield 3}

while item = block.next
  puts item
end
```

The flow of control switches back and forth, and the first time you call #next a Fiber is created which holds the state between calls. Therefore it is less efficient that iterating directly using #each.

When you call #next and there are no more values, a `StopIteration` exception is thrown. This is silently caught by the while loop. `StopIteration` is a subclass of `IndexError` which is a subclass of `StandardError`.

The nearest equivalent feature in ruby 1.8 is Generator, which was implemented using Continuations.

```
require 'generator'
block = Generator.new {|g| g.yield 1; g.yield 2; g.yield 3}

while block.next?
  puts block.next
end
```

1 http://github.com/trans/facets/blob/master/lib/more/facets/enumerator.rb

41 Lazy evaluation

In an Enumerator with a block, the target being yielded to is passed as an explicit parameter. This makes it possible to set up a chain of method calls so that each value is passed left-to-right along the whole chain, rather than building up intermediate arrays of values at each step.

The basic pattern is an Enumerator with a block which processes input values and yields (zero or more) output values for each one.

```
  Enumerator.new do |y|
    source.each do |input|     # filter INPUT
      ...
      y.yield output           # filter OUTPUT
    end
  end
```

So let's wrap this in a convenience method:

```
class Enumerator
  def defer(&blk)
    self.class.new do |y|
      each do |*input|
        blk.call(y, *input)
      end
    end
  end
end
```

This new method 'defer' can be used as a 'lazy' form of both select and map. Rather than building an array of values and returning that array at the end, it immediately yields each value. This means you start getting the answers sooner, and it will work with huge or even infinite lists. Example:

```
res = (1..1_000_000_000).to_enum.
  defer { |out,inp| out.yield inp if inp % 2 == 0 }.   # like select
  defer { |out,inp| out.yield inp+100 }.               # like map
  take(10)
p res
```

Although we start with a list of a billion items, at the end we only use the first 10 values generated, so we stop iterating once this has been done.

You can get the same capability[1] in ruby 1.8 using the facets library.[2] For convenience it also provides a Denumberable[3] module with lazy versions of familiar Enumerable methods such as map, select and reject.

1 http://github.com/trans/facets/blob/master/lib/core/facets/enumerable/defer.rb
2 http://facets.rubyforge.org/
3 http://github.com/trans/facets/blob/master/lib/core/facets/denumerable.rb

42 Methods which return Enumerators

From 1.8.7 on, many Enumerable methods will return an Enumerator if not given a block.

```
>> a = ["foo","bar","baz"]
=> ["foo", "bar", "baz"]
>> b = a.each_with_index
=> #<Enumerable::Enumerator:0xb7d7cadc>
>> b.each { |args| p args }
["foo", 0]
["bar", 1]
["baz", 2]
=> ["foo", "bar", "baz"]
>>
```

This means that usually you don't need to call `enum_for` explicitly. The very first example on this page reduces to just:

```
src = "hello"
puts src.each_byte.map { |b| "%02x" % b }.join(" ")
```

This can lead to somewhat odd behaviour for non-map like methods - when you call #each on the object later, you have to provide it with the "right sort" of block.

```
=> ["foo", "bar", "baz"]
>> b = a.select
=> #<Enumerable::Enumerator:0xb7d6cfb0>
>> b.each { |arg| arg < "c" }
=> ["bar", "baz"]
>>
```

43 More Enumerator readings

- ticket 707[1]
- enum_for[2] in Strictly Untyped blog
- Generator[3] in Anthony Lewis' blog

43.0.1 each_with_index

each_with_index calls its block with the item and its index.

```
array = ['Superman','Batman','The Hulk']

array.each_with_index do |item,index|
  puts "#{index} -> #{item}"
 end

# will print
# 0 -> Superman
# 1 -> Batman
# 2 -> The Hulk
```

43.0.2 find_all

find_all returns only those items for which the called block is not false

```
range = 1 .. 10

# find the even numbers

array = range.find_all { |item| item % 2 == 0 }

# returns [2,4,6,8,10]
```

```
array = ['Superman','Batman','Catwoman','Wonder Woman']

array = array.find_all { |item| item =~ /woman/ }

# returns ['Catwoman','Wonder Woman']
```

1 http://redmine.ruby-lang.org/issues/show/707
2 http://www.strictlyuntyped.com/2008/09/ruby-187s-enumerator-class.html
3 http://anthonylewis.com/2007/11/09/ruby-generators/

44 Exception

Exception is the superclass for exceptions

Instance Methods

`backtrace`

- Returns the backtrace information (from where exception occurred) as an array of strings.

`exception`

- Returns a clone of the exception object. This method is used by raise method.

`message`

- Returns the exception message.

45 FalseClass

The only instance of FalseClass is false.

Methods:

```
false & other – Logical AND, without short circuit behavior
false | other – Logical OR, without short circuit behavior
false ^ other – Exclusive Or (XOR)
```

46 IO - Fiber

A Fiber is a unit of concurrency (basically a manually controlled thread). It is a new construct in 1.9

1.8 basically used green threads similar, to fibers, but would pre-empt them, which 1.9 does not do.

See its description[1].

Several useful things have been built using fibers, like neverblock[2], the revactor gem, et al.

1 http://ruby-doc.org/core-1.9/classes/Fiber.html
2 http://oldmoe.blogspot.com/2008/07/untwisting-event-loop.html

47 IO

The IO class is basically an abstract class that provides methods to use on streams (for example, open files, or open sockets).

Basically, you can call several different things on it.

47.1 Encoding

Note that with 1.9, each call will return you a String with an encoding set, based on either how the connection was established, ex:

```
a = File.new('some filename', 'rb:ASCII-8BIT') # strings from this
 will be read in as ASCII-8BIT
b = File.new('some filename', 'r') # strings from this will be read
 in as whatever the Encoding.default_external is
# you can change what the encoding will be
a.set_encoding "UTF-8" # from now on it will read in UTF-8
```

47.2 gets

`gets` reads exactly one line, or up to the end of the file/stream (and includes the trailing newline, if one was given). A blocking call.

47.3 recv

`recv(1024)` reads up to at most 1024 bytes and returns your the String. A blocking call. Reads "" if a socket has been closed gracefully from the other end. It also has non blocking companions.

47.4 read

`read` reads up to the end of the file or up to the when the socket is closed. Returns any number of bytes. Blocking. For information on how to avoid blocking, see Socket[1].

1 http://en.wikibooks.org/wiki/Socket

48 IO - File

49 File

The file class is typically used for opening and closing files, as well as a few methods like deleting them and stat'ing them.

If you want to manipulate a file, not open it, also check out the Pathname class, as well as the FileUtils class.

To create a directory you'll need to use the Dir.mkdir method.

49.1 File#chmod

Here's how to do the equivalent of "chmod u+x filename"

```
File.class_eval do
   def self.addmod(flags, filename)
     themode=stat(filename).mode | flags
     chmod(themode, filename)
   end
end
```

Although there's no error checking and the return value could probably be something better (like themode).

So after defining that, your 'u+w' would be: @File.addmod(0200, 'filename')@ from `http://www.ruby-forum.com/topic/196999#new`

49.2 File#grep

This is actually Enumerable#grep, and I believe it just works piece-wise, like `File.each_line{|l|yield l if l=~ /whatever/ }`

49.3 File.join

This is useful for combining objects that aren't (or might not be) strings into a path. See here[1].

1 `http://www.ruby-forum.com/topic/206959#new`

50 IO - File::Stat

51 File::Stat

A File::Stat object is one that contains a file's status information.

Example:

```
stat = File.stat('/etc') # a File::Stat object
if stat.directory?
  puts '/etc is a directory last modified at ' + stat.mtime.to_s
end
```

You can also get a stat object for open file descriptors.

```
a = File.open('some file')
a.stat # File::Stat object
```

1

1 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

52 IO - GC

53 GC

Ruby does automatic Garbage Collection.

53.1 Tuning the GC

MRI's GC is a "full mark and sweep" and is run whenever it runs out of memory slots (i.e. before adding more memory, it will sweep the existing to see if it can free up some first--if not it adds more memory). It also is triggered after GC_MALLOC_LIMIT of bytes has been allocated by extensions. Unfortunately this causes a traversal of all memory, which is typically slow. See a good description[1].

The GC is known to typically take 10% of cpu, but if you have a large RAM load, it could take much more.

GC can be tuned at "compile time" (MRI/KRI's < 1.9) `http://blog.evanweaver.com/articles/2009/04/09/ruby-gc-tuning` or can be tuned by using environment variables (REE, MRI/KRI's >= 1.9).

Some tips:

You can set the compiler variable GC_MALLOC_LIMIT to be a very high value, which causes your program to use more RAM but to traverse it much less frequently. Good for large apps, like rails.

You can use jruby/rubinius which use more sophisticated GC's.

You can use "native" libraries which store the values away so that Ruby doesn't have to keep track of them and collect them. Examples: "NArray" gem and "google_hash" gem.

To turn it off: @GC.disable@

To force it to run once: @GC.start@

53.2 Conservative

Ruby's (MRI's) GC is mark and sweep, which means it is conservative. To accomplish this, it traverses the stack, looking for any section of memory which "looks" like a reference to an existing ruby object, and marks them as live. This can lead to false positives, even when there are no references to an object remaining.

This problem is especially bad in the 1.8.x series, when they don't have the MBARI patches applied (most don't, REE does). This is because, when you use threads, it actually allocates a full copy of

1 `http://timetobleed.com/garbage-collection-and-the-ruby-heap-from-railsconf/`

the stack to each thread, and as the threads are run, their stack is copied to the "real" stack, and they can pick up ghost references that belong to other threads, and also because the 1.8 MRI interpreter contains huge switch statements, which leave a lot of memory on the stack untouched, so it can continue to contain references to "ghost" references in error.

This all means that if you call a GC.start, it's not *guaranteed* to collect anything.

Some hints around this:

- If you call your code from within a method, and come *out* of that method, it might collect it more readily.
- You can do something of your own GC by using an ensure block, like

```
 a = SomeClass.new
begin
 ...
ensure
 a.cleanup
end
```

- If you write a "whole lot more" memory it might clear the stack of its old references.

53.3 Tunning Jruby's GC.

"here":`http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-core/27550` is an example of how to tune Jruby's GC. The G1GC is theoretically a "never pause" GC, though in reality most of them are pretty good at being speedy. For long running apps you'll probably want to run in server mode (--server), for increased performance, though decreased startup time.

Rubinius is also said to have a better GC.

53.4 How to avoid performance penalty

Since MRI's GC is basically O(N) as it grows, you get a performance penalty when GC's occur and you are using a lot of RAM in your app (and MRI almost never gives its RAM back to the system). Workarounds:

1. use less RAM by allocating fewer objects
2. do work in a forked "child process" which returns back the values desired. The child will die, freeing up its memory.
3. use Jruby et al (jruby has an excellent GC which doesn't slow things down much, even with larger apps).
4. use a gem that allows for native types, like NArray or the RubyGoogle Hash.
5. use REE instead of 1.8.6 (since it incorporates the MBARI patches which make the GC much more efficient).
6. use 1.9.x instead of 1.8.6 (since it uses true threads there is much less reference ghosting on the stack, thus making the GC more efficient).
7. set your app to restart periodically (passenger can do this).

8. create multiple apps, one designed to be large and slow, the others nimble (run your GC intensive all in the large one).
9. call GC.start yourself, or mix it with GC.disable
10. use memprof gem to see where the leaks are occurring (or the dike gem or the like).

54 IO - GC - Profiler

This is a class in 1.9 MRI

```
GC::Profiler.enable
# ... stuff
GC::Profiler.report # outputs to stdout
# or
report = GC::Profiler.result # a is set to a verbose ascii string
GC::Profiler.disable # stop it
</code>

Note also the existence of GC.count (number of times it has run).

Note that you can get a more report by setting a compiler flag
 <pre>GC_PROFILE_MORE_DETAIL=1
```

Note that the outputs "invoke Time(sec)" is actually the sum of user cpu time until that invoke occurred--i.e. a sleep 1 will result it it increasing by 0, but a busy loop for one second will result in it increasing by 1.

55 Marshal

56 Marshal

The Marshal class is used for serializing and de-serializing objects to disk:

ex

```
serialized = Marhshal.dump(['an', 'array', 'of', 'strings'])
unserialized = Marshal.restore(serialized)
```

With 1.9, each dump also includes an encoding, so you *must* use Marshal's stream read feature if you wish to read it from an IO object, like a file.

```
a = File.open("serialized_data", "w")
a.write Marshal.dump(33)
b = File.open("serialized_data", "r")
unserialized = Marshal.restore(b)
b.close
```

57 Method

58 Math

59 Module

60 Module - Class

61 NilClass

62 Numeric

Numeric provides common behavior of numbers. Numeric is an abstract class, so it should not be instantiated.

Included Modules:

```
Comparable
```

Instance Methods:

+ n

```
    Returns n.
```

- n

```
    Returns n negated.
```

n + num

n - num

n * num

n / num

```
    Performs arithmetic operations: addition, subtraction,
 multiplication, and division.
```

n % num

```
    Returns the modulus of n.
```

n ** num

```
    Exponentiation.
```

n.abs

```
    Returns the absolute value of n.
```

n.ceil

```
    Returns the smallest integer greater than or equal to n.
```

n.coerce(num)

Returns an array containing num and n both possibly converted to a type that allows them to be operated on mutually

n.divmod(num)

```
    Returns an array containing the quotient and modulus from
 dividing n by num.
```

n.floor

Returns the largest integer less than or equal to n.

```
    1.2.floor            #=> 1
    2.1.floor            #=> 2
    (-1.2).floor         #=> -2
    (-2.1).floor         #=> -3
```

n.integer?

```
    Returns true if n is an integer.
```

n.modulo(num)

```
    Returns the modulus obtained by dividing n by num and rounding
 the quotient with floor. Equivalent to n.divmod(num)[1].
```

n.nonzero?

```
    Returns n if it isn't zero, otherwise nil.
```

n.remainder(num)

Returns the remainder obtained by dividing n by num and removing decimals from the quotient. The result and n alw

```
    (13.modulo(4))          #=>  1
    (13.modulo(-4))         #=> -3
    ((-13).modulo(4))       #=>  3
    ((-13).modulo(-4))      #=> -1
    (13.remainder(4))       #=>  1
    (13.remainder(-4))      #=>  1
    ((-13).remainder(4))    #=> -1
```

```
(-13).remainder(-4)    #=> -1
```

n.round

```
  Returns n rounded to the nearest integer.
```

```
1.2.round              #=> 1
2.5.round              #=> 3
(-1.2).round           #=> -1
(-2.5).round           #=> -3
```

n.truncate

```
  Returns n as an integer with decimals removed.
```

```
1.2.truncate           #=> 1
2.1.truncate           #=> 2
(-1.2).truncate        #=> -1
(-2.1).truncate        #=> -2
```

n.zero?

```
Returns zero if n is 0.
```

63 Numeric - Integer

Integer provides common behavior of integers (Fixnum and Bignum). Integer is an abstract class, so you should not instantiate this class.

Inherited Class: Numeric Included Module: Precision

Class Methods:

Integer::induced_from(numeric)

```
    Returns the result of converting numeric into an integer.
```

Instance Methods:

Bitwise operations: AND, OR, XOR, and inversion.

```
~i
```

```
i & int
```

```
i | int
```

```
i ^ int
```

```
i << int
```

```
i >> int
```

```
    Bitwise left shift and right shift.
```

```
i[n]
```

```
    Returns the value of the nth bit from the least significant bit,
  which is i[0].
```

```
    5[0]      # => 1
    5[1]      # => 0
    5[2]      # => 1
```

i.chr

```
    Returns a string containing the character for the character code
  i.
```

```
    65.chr    # => "A"
    ?a.chr    # => "a"
```

i.downto(min) {| i| ...}

```
    Invokes the block, decrementing each time from i to min.
```

```
    3.downto(1) {|i|
      puts i
    }
```

```
    # prints:
    #  3
    #  2
    #  1
```

i.next

i.succ

```
    Returns the next integer following i. Equivalent to i + 1.
```

i.size

```
    Returns the number of bytes in the machine representation of i.
```

i.step(upto, step) {| i| ...}

```
    Iterates the block from i to upto, incrementing by step each
  time.
```

```
    10.step(5, -2) {|i|
      puts i
    }
    # prints:
    #  10
    #  8
    #  6
```

i.succ

```
    See i.next
```

i.times {| i| ...}

```
    Iterates the block i times.
```

```
    3.times {|i|
      puts i
    }
    # prints:
    #  0
    #  1
    #  2
```

i.to_f

```
    Converts i into a floating point number. Float conversion may
  lose precision information.
```

```
    1234567891234567.to_f   # => 1.234567891e+15
```

i.to_int

```
    Returns i itself. Every object that has to_int method is treated
  as if it's an integer.
```

i.upto(max) {| i| ...}

```
    Invokes the block, incrementing each time from i to max.
```

```
    1.upto(3) {|i|
      puts i
    }
    # prints:
    #  1
    #  2
    #  3
```

64 Numeric - Integer - Bignum

65 Numeric - Integer - Fixnum

66 Numeric - Float

67 Range

68 Regexp

69 Regexp Regular Expressions

Class Regexp holds a regular expression, used to match a pattern of strings.

Regular expressions can be created using /your_regex_here/ or by using the constructor "new".

```
>> /a regex/
>> /a case insensitive regex/i
```

or use the new constructor with constants, like

```
>> Regexp.new('a regex')
>> Regexp.new('a regex', MULTILINE)
```

To see all available creation options, please see the regex rdoc[1].

69.1 oniguruma

Starting with 1.9, ruby has a new Regular Expression engine (oniguruma), which is slightly faster and more powerful, as well as encoding aware/friendly. To see a good explanation of how it works, please see its rdoc[2].

69.2 Simplifying regexes

Strategy: name them, then combine them.

```
float = /[\d]+\.[\d]+/
complex = /[+-]#{float}\.#{float}/
```

69.3 Helper websites

"rubular":`http://rubular.com` lets you test your regexps online

1 http://ruby-doc.org/core/classes/Regexp.html
2 http://github.com/ruby/ruby/blob/trunk/doc/re.rdoc

69.4 Alternative Regular Expression Libraries

Some other wrappers exist:

PCRE[3] Boost Regex[4]

[5]

3 http://github.com/rdp/pcre
4 http://github.com/michaeledgar/ruby-boost-regex
5 http://en.wikibooks.org/wiki/Category%3ARuby%20Programming

70 RubyVM

RubyVM is (as noted) very VM dependent. Currently it is defined only for 1.9.x MRI.

70.1 RubyVM::InstructionSequence.disassemble

Available on 1.9 only, this is basically a wrapper for yarv.

```
>> class A; def go; end; end
>> b = A.new.method(:go)
=> #
>> print RubyVM::InstructionSequence.disassemble b
== disasm: ======================
0000 trace 8 ( 1)
0002 putnil
0003 trace 16 ( 1)
0005 leave
```

I believe those trace methods represent calls to any Kernel#set_trace_func (or the C sibling to that ruby method).

Also note that with 1.9.2 you can pass in a proc.

71 String

String Class

71.0.1 Methods:

crypt(*salt*). Returns an encoded string using crypt(3). *salt* is a string with minimal length 2. If *salt* starts with "1", MD5 encryption is used, based on up to eight characters following "1". Otherwise, DES is used, based on the first two characters of *salt*.

72 Struct

73 Struct

Please see the rdoc[1] for Struct. Add any further examples/tutorials here.

1 http://ruby-doc.org/core/classes/Struct.html

74 Struct - Struct::Tms

75 Symbol

Symbols

A Ruby symbol is the internal representation of a name. You construct the symbol for a name by preceding the name with a colon. A particular name will always generate the same symbol, regardless of how that name is used within the program.

```
:Object
:myVariable
```

Other languages call this process “interning,*and call symbols “atoms.*

76 Time

```
class Tuesdays
        attr_accessor :time, :place

        def initialize(time, place)
                @time = time
                @place = place
        end
end

feb12 = Tuesdays.new("8:00", "Rice U.")
```

As for the object, it is clever let me give you some advice though. Firstly, you don't ever want to store a date or time as a string. This is always a mistake. -- though for learning purposes it works out, as in your example. In real life, simply not so. Lastly, you created a class called Tuesdays without specifying what would make it different from a class called Wednesday; that is to say the purpose is nebulous: there is nothing special about Tuesdays to a computer. If you have to use comments to differentiate Tuesdays from Wednesdays you typically fail.

```
class Event
        def initialize( place, time=Time.new )
                @place = place

                case time.class.to_s
                        when "Array"
                                @time = Time.gm( *time )
                        when "Time"
                                @time = time
                        else
                                throw "invalid time type"
                end
        end

        attr_accessor :time, :place

end

## Event at 5:00PM 2-2-2009  CST
funStart = Event.new( "evan-hodgson day",
 [0,0,17,2,2,2009,2,nil,false,"CST"] )

## Event now, (see time=Time.new -- the default in constructor)
rightNow = Event.new( "NOW!" );

## You can compaire Event#time to any Time object!!
if Time.new > funStart.time
        puts "We,re there"
else
        puts "Not yet"
end

## Because the constructor takes two forms of time, you can do
## Event.new( "Right now", Time.gm(stuff here) )
```

77 Thread

78 Thread

The Thread class in Ruby is a wrapper/helper for manipulating threads (starting them, checking for status, storing thread local variables).

Here[1] is a good tutorial.

Note that with MRI 1.8, Ruby used "green threads" (pseudo threads--really single threaded). With 1.9, MRI uses "native threads with a global interpeter lock" so "typically single threaded". Jruby uses true concurrent threads. IronRuby uses true concurrent threads. Rubinius has threading currently like MRI 1.9. See here[2] for a good background.

Because 1.8 uses green threads this means that, for windows, any "C" call (like gets, puts, etc.) will block all other threads until it returns. The scheduler can only switch from thread to thread when it is running ruby code. However, you can run sub processes in a thread and it will work, and you can run select and it will still be able to switch among threads. For Linux this means that you can select on $stdin and thus not block for input. For windows, though, you're really stuck. Any keyboard input will block all other threads. With 1.9 this isn't as much of a problem because of the use of native threads (the thread doing the IO call releases its hold on the GIL until the IO call returns). You can use Jruby[3] for a non blocking 1.8.

With 1.9 you can get "around" the global thread lock by wrapping a C call in rb_thread_blocking_-region (this will basically allow that thread to "go off and do its thing" while the other ruby threads still operate, one at a time. When the method returns, it will enter ruby land again and be one of the "globally locked" (one at a time) threads).

78.1 Thread local variables

If you want to start a thread with a specific parameter that is unique to that thread, you could use Thread.current

```
Thread.new {
  puts Thread.current # unique!
}
```

Or you can start it with a parameter, like

```
th = Thread.new(33) {|parameter|
  thread_unique_parameter = parameter
```

1 http://rubylearning.com/satishtalim/ruby_threads.html
2 http://www.igvita.com/2008/11/13/concurrency-is-a-myth-in-ruby/
3 http://betterlogic.com/roger/?p=2930

```
}
```

This avoids concurrency issues when starting threads, for example

```
# BROKEN
for x in [1,2,3] do
 Thread.new {
   puts x # probably always outputs 3 because x's value changes due
 to the outside scope
 }
end
```

78.2 Joining on multiple threads

For this one, use the ThreadsWait class[4].

```
require 'thwait'
ThreadsWait.join_all(th1, th2, th3) do |th_that_just_exited| # could
 also call it like join_all([th1, th2, ...])
  puts 'thread just exited', th_that_just_exited
end
# at this point they will all be done
```

78.3 Controlling Concurrency

See the Mutex[5] class for how to control interaction between threads.

4 http://ruby-doc.org/core/classes/ThreadsWait.html
5 http://en.wikibooks.org/wiki/%2FStandard%20Library%2FMutex

79 TrueClass

The only instance of TrueClass is true.

Methods:

```
true & other – Logical AND, without short circuit behavior
true | other – Logical OR, without short circuit behavior
true ^ other – Logical exclusive Or (XOR)
```

80 Contributors

Edits	User
30	Adrignola[1]
2	Ahy1[2]
1	Archaeometallurg[3]
1	Avicennasis[4]
1	Bapabooiee[5]
2	Ben.lagoutte[6]
2	Benjamin Meinl[7]
1	BiT[8]
2	Bluevine[9]
92	Briand[10]
2	Byassine52[11]
1	CAJDavidson[12]
1	Cspurrier[13]
7	Damien Karras[14]
4	Darklama[15]
4	Dasch[16]
9	DavidDraughn[17]
1	Derbeth[18]
1	Dirk Hünniger[19]
1	Dmw[20]
1	Eddy264[21]

1 http://en.wikibooks.org/w/index.php?title=User:Adrignola
2 http://en.wikibooks.org/w/index.php?title=User:Ahy1
3 http://en.wikibooks.org/w/index.php?title=User:Archaeometallurg
4 http://en.wikibooks.org/w/index.php?title=User:Avicennasis
5 http://en.wikibooks.org/w/index.php?title=User:Bapabooiee
6 http://en.wikibooks.org/w/index.php?title=User:Ben.lagoutte
7 http://en.wikibooks.org/w/index.php?title=User:Benjamin_Meinl
8 http://en.wikibooks.org/w/index.php?title=User:BiT
9 http://en.wikibooks.org/w/index.php?title=User:Bluevine
10 http://en.wikibooks.org/w/index.php?title=User:Briand
11 http://en.wikibooks.org/w/index.php?title=User:Byassine52
12 http://en.wikibooks.org/w/index.php?title=User:CAJDavidson
13 http://en.wikibooks.org/w/index.php?title=User:Cspurrier
14 http://en.wikibooks.org/w/index.php?title=User:Damien_Karras
15 http://en.wikibooks.org/w/index.php?title=User:Darklama
16 http://en.wikibooks.org/w/index.php?title=User:Dasch
17 http://en.wikibooks.org/w/index.php?title=User:DavidDraughn
18 http://en.wikibooks.org/w/index.php?title=User:Derbeth
19 http://en.wikibooks.org/w/index.php?title=User:Dirk_H%C3%BCnniger
20 http://en.wikibooks.org/w/index.php?title=User:Dmw
21 http://en.wikibooks.org/w/index.php?title=User:Eddy264

10 Eisel98[22]
1 Elexx[23]
1 Eshafto[24]
12 EvanCarroll[25]
1 Fhope[26]
6 Fishpi[27]
1 Florian bravo[28]
1 Formigarafa[29]
1 Fricky[30]
1 Gamache[31]
1 Geoffj[32]
1 Georgesawyer[33]
1 Gfranken[34]
2 Ghostzart[35]
1 Grokus[36]
1 HenryLi[37]
1 Huw[38]
4 IanVaughan[39]
2 Iron9light[40]
1 Iviney[41]
2 J36miles[42]
1 Jactheman[43]
1 Jastern949[44]
4 Jedediah Smith[45]
1 Jeffq[46]

22 http://en.wikibooks.org/w/index.php?title=User:Eisel98
23 http://en.wikibooks.org/w/index.php?title=User:Elexx
24 http://en.wikibooks.org/w/index.php?title=User:Eshafto
25 http://en.wikibooks.org/w/index.php?title=User:EvanCarroll
26 http://en.wikibooks.org/w/index.php?title=User:Fhope
27 http://en.wikibooks.org/w/index.php?title=User:Fishpi
28 http://en.wikibooks.org/w/index.php?title=User:Florian_bravo
29 http://en.wikibooks.org/w/index.php?title=User:Formigarafa
30 http://en.wikibooks.org/w/index.php?title=User:Fricky
31 http://en.wikibooks.org/w/index.php?title=User:Gamache
32 http://en.wikibooks.org/w/index.php?title=User:Geoffj
33 http://en.wikibooks.org/w/index.php?title=User:Georgesawyer
34 http://en.wikibooks.org/w/index.php?title=User:Gfranken
35 http://en.wikibooks.org/w/index.php?title=User:Ghostzart
36 http://en.wikibooks.org/w/index.php?title=User:Grokus
37 http://en.wikibooks.org/w/index.php?title=User:HenryLi
38 http://en.wikibooks.org/w/index.php?title=User:Huw
39 http://en.wikibooks.org/w/index.php?title=User:IanVaughan
40 http://en.wikibooks.org/w/index.php?title=User:Iron9light
41 http://en.wikibooks.org/w/index.php?title=User:Iviney
42 http://en.wikibooks.org/w/index.php?title=User:J36miles
43 http://en.wikibooks.org/w/index.php?title=User:Jactheman
44 http://en.wikibooks.org/w/index.php?title=User:Jastern949
45 http://en.wikibooks.org/w/index.php?title=User:Jedediah_Smith
46 http://en.wikibooks.org/w/index.php?title=User:Jeffq

17 Jguk[47]
2 Jim Mckeeth[48]
1 Jomegat[49]
1 Jordandanford[50]
2 Joti[51]
1 Karvendhan[52]
9 Keagan[53]
1 Kenfodder[54]
1 Knudvaneeden[55]
1 Krischik[56]
2 Lhbts[57]
10 Marburg[58]
1 Martindemello[59]
9 Mehryar[60]
1 Munificent[61]
1 N8chz[62]
1 Nbeyer[63]
3 Nmagedman[64]
1 NqpZ[65]
1 Nzhamstar[66]
1 Oleander[67]
1 Patek68[68]
1 Paul.derry[69]
2 Pbwest[70]
1 Quilz[71]

47 http://en.wikibooks.org/w/index.php?title=User:Jguk
48 http://en.wikibooks.org/w/index.php?title=User:Jim_Mckeeth
49 http://en.wikibooks.org/w/index.php?title=User:Jomegat
50 http://en.wikibooks.org/w/index.php?title=User:Jordandanford
51 http://en.wikibooks.org/w/index.php?title=User:Joti
52 http://en.wikibooks.org/w/index.php?title=User:Karvendhan
53 http://en.wikibooks.org/w/index.php?title=User:Keagan
54 http://en.wikibooks.org/w/index.php?title=User:Kenfodder
55 http://en.wikibooks.org/w/index.php?title=User:Knudvaneeden
56 http://en.wikibooks.org/w/index.php?title=User:Krischik
57 http://en.wikibooks.org/w/index.php?title=User:Lhbts
58 http://en.wikibooks.org/w/index.php?title=User:Marburg
59 http://en.wikibooks.org/w/index.php?title=User:Martindemello
60 http://en.wikibooks.org/w/index.php?title=User:Mehryar
61 http://en.wikibooks.org/w/index.php?title=User:Munificent
62 http://en.wikibooks.org/w/index.php?title=User:N8chz
63 http://en.wikibooks.org/w/index.php?title=User:Nbeyer
64 http://en.wikibooks.org/w/index.php?title=User:Nmagedman
65 http://en.wikibooks.org/w/index.php?title=User:NqpZ
66 http://en.wikibooks.org/w/index.php?title=User:Nzhamstar
67 http://en.wikibooks.org/w/index.php?title=User:Oleander
68 http://en.wikibooks.org/w/index.php?title=User:Patek68
69 http://en.wikibooks.org/w/index.php?title=User:Paul.derry
70 http://en.wikibooks.org/w/index.php?title=User:Pbwest
71 http://en.wikibooks.org/w/index.php?title=User:Quilz

3 QuiteUnusual[72]
5 Raevel[73]
1 Rdnk[74]
1 Recent Runes[75]
2 RichardOnRails[76]
1 Rklemme[77]
72 Rogerdpack[78]
1 Rory O'Kane[79]
7 Rsperberg[80]
1 Rule.rule[81]
23 Ryepdx[82]
1 Scientes[83]
5 Scientus[84]
5 Scopsowl[85]
6 Sebleclerc[86]
1 Shadeclan[87]
11 Sjc[88]
1 Slashme[89]
2 Snyce[90]
1 Srogers[91]
1 Stiang[92]
1 TeleComNasSprVen[93]
37 Valters[94]
16 Vanivk[95]
2 Vovk[96]

72 http://en.wikibooks.org/w/index.php?title=User:QuiteUnusual
73 http://en.wikibooks.org/w/index.php?title=User:Raevel
74 http://en.wikibooks.org/w/index.php?title=User:Rdnk
75 http://en.wikibooks.org/w/index.php?title=User:Recent_Runes
76 http://en.wikibooks.org/w/index.php?title=User:RichardOnRails
77 http://en.wikibooks.org/w/index.php?title=User:Rklemme
78 http://en.wikibooks.org/w/index.php?title=User:Rogerdpack
79 http://en.wikibooks.org/w/index.php?title=User:Rory_O%27Kane
80 http://en.wikibooks.org/w/index.php?title=User:Rsperberg
81 http://en.wikibooks.org/w/index.php?title=User:Rule.rule
82 http://en.wikibooks.org/w/index.php?title=User:Ryepdx
83 http://en.wikibooks.org/w/index.php?title=User:Scientes
84 http://en.wikibooks.org/w/index.php?title=User:Scientus
85 http://en.wikibooks.org/w/index.php?title=User:Scopsowl
86 http://en.wikibooks.org/w/index.php?title=User:Sebleclerc
87 http://en.wikibooks.org/w/index.php?title=User:Shadeclan
88 http://en.wikibooks.org/w/index.php?title=User:Sjc
89 http://en.wikibooks.org/w/index.php?title=User:Slashme
90 http://en.wikibooks.org/w/index.php?title=User:Snyce
91 http://en.wikibooks.org/w/index.php?title=User:Srogers
92 http://en.wikibooks.org/w/index.php?title=User:Stiang
93 http://en.wikibooks.org/w/index.php?title=User:TeleComNasSprVen
94 http://en.wikibooks.org/w/index.php?title=User:Valters
95 http://en.wikibooks.org/w/index.php?title=User:Vanivk
96 http://en.wikibooks.org/w/index.php?title=User:Vovk

48 Withinfocus[97]
2 Xania[98]
7 Yath[99]
1 Yb2[100]
6 Yuuki Mayuki[101]
6 Zemoxian[102]
5 Θεόφιλε[103]

97 http://en.wikibooks.org/w/index.php?title=User:Withinfocus
98 http://en.wikibooks.org/w/index.php?title=User:Xania
99 http://en.wikibooks.org/w/index.php?title=User:Yath
100 http://en.wikibooks.org/w/index.php?title=User:Yb2
101 http://en.wikibooks.org/w/index.php?title=User:Yuuki_Mayuki
102 http://en.wikibooks.org/w/index.php?title=User:Zemoxian
103 http://en.wikibooks.org/w/index.php?title=User:%CE%98%CE%B5%CF%8C%CF%86%CE%B9%CE%BB%CE%B5

List of Figures

- GFDL: Gnu Free Documentation License. http://www.gnu.org/licenses/fdl.html
- cc-by-sa-3.0: Creative Commons Attribution ShareAlike 3.0 License. http://creativecommons.org/licenses/by-sa/3.0/
- cc-by-sa-2.5: Creative Commons Attribution ShareAlike 2.5 License. http://creativecommons.org/licenses/by-sa/2.5/
- cc-by-sa-2.0: Creative Commons Attribution ShareAlike 2.0 License. http://creativecommons.org/licenses/by-sa/2.0/
- cc-by-sa-1.0: Creative Commons Attribution ShareAlike 1.0 License. http://creativecommons.org/licenses/by-sa/1.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/
- cc-by-2.0: Creative Commons Attribution 2.0 License. http://creativecommons.org/licenses/by/2.0/deed.en
- cc-by-2.5: Creative Commons Attribution 2.5 License. http://creativecommons.org/licenses/by/2.5/deed.en
- cc-by-3.0: Creative Commons Attribution 3.0 License. http://creativecommons.org/licenses/by/3.0/deed.en
- GPL: GNU General Public License. http://www.gnu.org/licenses/gpl-2.0.txt
- LGPL: GNU Lesser General Public License. http://www.gnu.org/licenses/lgpl.html
- PD: This image is in the public domain.
- ATTR: The copyright holder of this file allows anyone to use it for any purpose, provided that the copyright holder is properly attributed. Redistribution, derivative work, commercial use, and all other use is permitted.
- EURO: This is the common (reverse) face of a euro coin. The copyright on the design of the common face of the euro coins belongs to the European Commission. Authorised is reproduction in a format without relief (drawings, paintings, films) provided they are not detrimental to the image of the euro.
- LFK: Lizenz Freie Kunst. http://artlibre.org/licence/lal/de
- CFR: Copyright free use.

- EPL: Eclipse Public License. http://www.eclipse.org/org/documents/epl-v10.php

Copies of the GPL, the LGPL as well as a GFDL are included in chapter Licenses[104]. Please note that images in the public domain do not require attribution. You may click on the image numbers in the following table to open the webpage of the images in your webbrower.

104 Chapter 81 on page 253

1		cc-by-sa-2.5
2	en:Briand[105]	
3		cc-by-sa-2.5
4		cc-by-sa-2.5
5		cc-by-sa-2.5
6	Nzhamstar[106]	PD

105 http://en.wikibooks.org/wiki/Briand
106 http://en.wikibooks.org/wiki/User%3ANzhamstar

81 Licenses

81.1 GNU GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow. TERMS AND CONDITIONS 0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion. 1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work. 2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary. 3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures. 4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee. 5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

* a) The work must carry prominent notices stating that you modified it, and giving a relevant date. * b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices". * c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it. * d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate. 6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

* a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange. * b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge. * c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b. * d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements. * e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying. 7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

* a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or * b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or * c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or * d) Limiting the use for publicity purposes of names of licensors or authors of the material; or * e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or * f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way. 8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10. 9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so. 10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it. 11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law. 12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program. 13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such. 14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright

holder as a result of your choosing to follow a later version. 15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. 16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. 17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

<program> Copyright (C) <year> <name of author> This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’. This is free software, and you are welcome to redistribute it under certain conditions; type ‘show c’ for details.

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, your program’s commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/why-not-lgpl.html>.

81.2 GNU Free Documentation License

Version 1.3, 3 November 2008

Copyright © 2000, 2001, 2002, 2007, 2008 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. 0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference. 1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work’s title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License. 2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies. 3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document’s license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document. 4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

* A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission. * B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement. * C. State on the Title page the name of the publisher of the Modified Version, as the publisher. * D. Preserve all the copyright notices of the Document. * E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices. * F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below. * G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document’s license notice. * H. Include an unaltered copy of this License. * I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence. * J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission. * K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein. * L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles. * M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version. * N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section. * O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements". 6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document. 7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate. 8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title. 9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it. 10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document. 11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing. ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (C) YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with … Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

81.3 GNU Lesser General Public License

GNU LESSER GENERAL PUBLIC LICENSE

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below. 0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The “Minimal Corresponding Source” for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The “Corresponding Application Code” for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work. 1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL. 2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

* a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or * b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

* a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

* a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License. * b) Accompany the Combined Work with a copy of the GNU GPL and this license document. * c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document. * d) Do one of the following: o 0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source. o 1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user’s computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version. * e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

* a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License. * b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy’s public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.