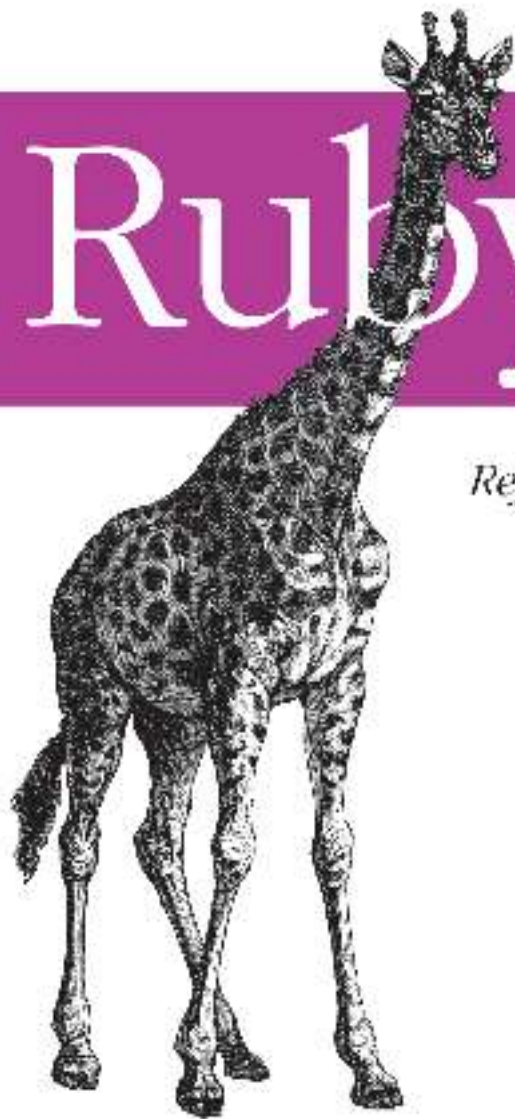


A Quick Guide to Ruby

Ruby

*Pocket
Reference*



O'REILLY®

Michael Fitzgerald

Ruby Pocket Reference



Ruby is an easy language to learn, but in the heat of action, you may forget the name of a method or the correct syntax for a conditional. This handy reference offers brief yet clear explanations of Ruby's core components—from operators to reserved words, from data structures to method syntax—highlighting those key features that you'll likely use every day when coding Ruby.

Organized to help you find what you need quickly, *Ruby Pocket Reference* will not only get you up to speed on how Ruby works, it will also provide you with a handy reference you can use anywhere, anytime.

Inside you'll find essential information on:

- Reserved words, operators, comments, numbers, variables, ranges, and symbols
- Predefined variables, global constants, instance variables, and more
- Conditional statements, methods, classes, and modules (mixins)
- Lists of methods from the `Object`, `String`, `Array`, and `Hash` classes, and the `Kernel` module
- `require` and line-formatting directives
- Interactive Ruby (irb) and RubyGems
- Ruby documentation (RDoc)

If you use Ruby daily and just want the facts—fast—this is the book for you.

www.oreilly.com

US \$9.99

CAN \$11.99

ISBN-10: 0-596-51481-6

ISBN-13: 978-0-596-51481-5



Includes
FREE 45-Day
Online Edition

Safari
BOOKS ONLINE
ENABLED

Ruby

Pocket Reference

Michael Fitzgerald

O'REILLY®

Beijing • Cambridge • Farnham • Köln • Paris • Sebastopol • Taipei • Tokyo

Ruby Pocket Reference

by Michael Fitzgerald

Copyright © 2007 Michael Fitzgerald. All rights reserved.
Printed in Canada.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North,
Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales
promotional use. Online editions are also available for most titles
(*safari.oreilly.com*). For more information, contact our corporate/
institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editor: Simon St. Laurent

Cover Designer: Karen Montgomery

Production Editor:

Interior Designer: David Futato

Rachel Monaghan

Illustrators: Robert Romano and

Proofreader: Rachel Monaghan

Jessamyn Read

Indexer: Ellen Troutman Zaig

Printing History:

July 2007:

First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are
registered trademarks of O'Reilly Media, Inc. The *Pocket Reference* series
designations, *Ruby Pocket Reference*, the image of a giraffe, and related trade
dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish
their products are claimed as trademarks. Where those designations appear
in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the
designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the
publisher and author assume no responsibility for errors or omissions, or for
damages resulting from the use of the information contained herein.

ISBN-10: 0-596-51481-6

ISBN-13: 978-0-596-51481-5

[TM]

Contents

Running Ruby	2
Running the Ruby Interpreter	3
Using a Shebang Line on Unix/Linux	5
Associating File Types on Windows	5
Reserved Words	7
Operators	9
Comments	10
Numbers	11
Variables	11
Local Variables	12
Instance Variables	12
Class Variables	13
Global Variables	13
Constants	13
Parallel Assignment of Variables	13
Symbols	14
Predefined Variables	14
Pseudovariables	17
Global Constants	18

Ranges	19
Methods	19
Parentheses	20
Return Values	20
Method Name Conventions	21
Default Arguments	22
Variable Arguments	22
Aliasing Methods	23
Blocks	23
Procs	25
Conditional Statements	27
The if Statement	27
The unless Statement	29
The while Statement	30
The until Statement	31
The case Statement	32
The for Loop	33
The Ternary Operator	34
Executing Code Before or After a Program	34
Classes	34
Instance Variables	36
Accessors	38
Class Variables	39
Class Methods	40
Singletons	40
Inheritance	42
Public, Private, or Protected	42
Modules and Mixins	44

Files	47
Creating a New File	47
Opening an Existing File	48
ARGV and ARGF	48
Renaming and Deleting Files	49
File Inquiries	50
File Modes and Ownership	51
The IO Class	52
Exception Handling	54
The rescue and ensure Clauses	55
The raise Method	55
The catch and throw Methods	56
Object Class	56
Object Instance Methods	57
Kernel Module	62
String Class	72
Expression Substitution	73
General Delimited Strings	73
Here Documents	73
Escape Characters	75
Character Encoding	75
Regular Expressions	76
String Methods	81
Array Class	94
Creating Arrays	94
Array Class Methods	96
Array Instance Methods	96

Hash Class	106
Creating Hashes	107
Hash Class Methods	108
Hash Instance Methods	108
Time Formatting Directives	113
Interactive Ruby (irb)	114
Ruby Debugger	117
Ruby Documentation	119
RDoc Options	121
RubyGems	125
Rake	131
Ruby Resources	133
Glossary	134
Index	151

Ruby Pocket Reference

Ruby is an open source, object-oriented programming language created by Yukihiro “Matz” Matsumoto. First released in Japan in 1995, Ruby has gained worldwide acceptance as an easy-to-learn, powerful, and expressive language, especially since the advent of Ruby on Rails, a web application framework written in Ruby (<http://www.rubyonrails.org>). Ruby’s core is written in the C programming language and runs on all major platforms. It is an interpreted rather than compiled language. For more information on Ruby, see <http://www.ruby-lang.org>.

Conventions Used in This Book

The following font conventions are used in this book:

Italic

Indicates pathnames and filenames (such as program names); Internet addresses, such as domain names and URLs; and emphasized or newly defined terms.

Constant width

Indicates commands and options that should be typed verbatim in a file or in *irb*; or names and keywords in Ruby programs, including method, variable, and class names.

Constant width italic

Indicates user-supplied values.

Constant width bold

Used to draw attention to parts of programs.

Comments and Questions

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (Fax)

There is a web page for this book, which lists errata, examples, or any additional information. You can access this page at:

<http://www.oreilly.com/catalog/9780596514815>

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For information about books, conferences, Resource Centers, and the O'Reilly Network, see the O'Reilly web site at:

<http://www.oreilly.com>

Acknowledgments

This book is dedicated to John H. Atkinson, Jr. (1934–2007).

I want to thank Simon St.Laurent, Ryan Waldron, and Rachel Monaghan for their help in creating, editing, and producing this book.

Running Ruby

Test to see whether Ruby is running on your computer by typing the following at a shell or command prompt:

```
ruby --version
```

An affirmative response will look similar to this (this example is for version 1.8.6 running on Mac OS X):

```
ruby 1.8.6 (2007-03-13 patchlevel 0) [powerpc-darwin8.9.0]
```

You can install Ruby on any of the major platforms. For Ruby file archives and installation instructions, see <http://www.ruby-lang.org/en/downloads>.

Running the Ruby Interpreter

Usage:

```
ruby [switches] [--] [program filename] [arguments]
```

Switches (or command-line options):

- o[*octal*]
Specify a record separator (\0 if no argument).
- a
Autosplit mode with -n or -p (splits \$_ into \$F).
- c
Check syntax only.
- C*directory*
cd to directory before executing your script or program.
- d
Set debugging flags (set predefined variable \$DEBUG to true).
- e '*command*'
Execute one line of script. Several -es allowed. Omit [*program filename*].
- F*pattern*
split() pattern for autosplit (-a).
- i[*extension*]
Edit ARGV files in place (make backup if extension supplied).
- I*directory*
Specify \$LOAD_PATH (predefined variable) directory; may be used more than once.

`-kkcode`
Specify the character set. See Table 16.

`-l`
Enable line-ending processing.

`-n`
Assume 'while gets(); ... end' loop around your script.

`-p`
Assume loop like `-n` but print line also like `sed`.

`-rlibrary`
Require the library before executing your script.

`-s`
Enable some switch parsing for switches after script name.

`-S`
Look for the script using `PATH` environment variable.

`-T[level]`
Turn on tainting checks.

`-v`
Print version number, then turn on verbose mode (compare `--version`).

`-w`
Turn warnings on for your script or program.

`-W[level]`
Set warning level: 0=silence, 1=medium, and 2=verbose (default).

`-x[directory]`
Strip off text before `#!` shebang line, and optionally `cd` to directory.

`--copyright`
Print the copyright.

`--version`
Print the version (compare `-v`).

Using a Shebang Line on Unix/Linux

A shebang line may appear on the first line of a Ruby program (or other program or script). Its job is to help a Unix/Linux system execute the commands in the program or script according to a specified interpreter—Ruby, in our case. (This does not work on Windows.) Here is a program named *hi.rb* with a shebang on the first line:

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

puts "Hello, Matz!"
```

Other alternative shebang lines are `#!/usr/bin/ruby` or `#!/usr/local/bin/ruby`. With a shebang in place, you can type the filename (followed by Return or Enter) at a shell prompt without invoking the Ruby interpreter directly:

```
$ hi.rb
```

Associating File Types on Windows

Windows doesn't know or care about shebang (`#!`), but you can achieve a similar effect by creating a file type association with the `assoc` and `ftype` commands on Windows (DOS). To find out whether an association exists for the file extension *.rb*, use the `assoc` command:

```
C:\Ruby Code>assoc .rb
File association not found for extension .rb
```

If it's not found, associate the *.rb* extension with a file type:

```
C:\Ruby Code>assoc .rb=rbFile
```

Then test to see whether the association exists:

```
C:\Ruby Code>assoc .rb
.rb=rbFile
```

Now test to see whether the file type for Ruby exists:

```
C:\Ruby Code>ftype rbfile
File type 'rbfile' not found or no open command associated with it.
```

If not found, you can create it with a command like this:

```
C:\Ruby Code>ftype rbfile="C:\Program Files\Ruby\bin\  
ruby.exe" "%1" %*
```

Be sure to put the correct path to the executable for the Ruby interpreter, followed by the substitution variables. %1 is a substitution variable for the file you want to run, and %* accepts all other parameters that may appear on the command line. Test it:

```
C:\Ruby Code>ftype rbfile  
rbfile="C:\Program Files\Ruby\bin\ruby.exe" "%1" %*
```

Finally, add .rb to the PATHEXT environment variable. See whether it is there already with set:

```
C:\Ruby Code>set PATHEXT  
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;  
.tcl
```

If it is not there, add it like this:

```
C:\Ruby Code>set PATHEXT=.rb;%PATHEXT%
```

Then test it again:

```
C:\Ruby Code>set PATHEXT  
PATHEXT=.rb;.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;  
.WSH;.tcl
```

All is now in order:

```
C:\Ruby Code>type hi.rb  
#!/usr/bin/env ruby
```

```
puts "Hello, Matz!"
```

Make sure you are able to execute the file:

```
C:\Ruby Code>cacls hi.rb /g username:f  
Are you sure (Y/N)?y  
processed file: C:\Ruby Code\hi.rb
```

Run the program by entering the program's file name at the command prompt, with or without the file extension:

```
C:\Ruby Code>hi  
Hello, Matz!
```

To preserve these settings, you may add these commands to your *autoexec.bat* file, or set the environment variables by selecting Star → Control Panel → System, clicking on the Advanced tab, and then clicking the Environment Variables button.

Reserved Words

Table 1 lists Ruby's reserved words or keywords.

Table 1. Ruby's reserved words

Reserved word	Description
BEGIN	Code, enclosed in { }, to run before the program runs.
END	Code, enclosed in { }, to run when the program ends.
alias	Creates an alias for an existing method, operator, or global variable.
and	Logical operator; same as && except and has lower precedence.
begin	Begins a code block or group of statements; closes with end.
break	Terminates a while or until loop, or a method inside a block.
case	Compares an expression with a matching when clause; closes with end. See when.
class	Defines a class; closes with end.
def	Defines a method; closes with end.
defined?	A special operator that determines whether a variable, method, super method, or block exists.
do	Begins a block, and executes code in that block; closes with end.
else	Executes following code if previous conditional is not true, set with if, elsif, unless, or case. See if, elsif.
elsif	Executes following code if previous conditional is not true, set with if or elsif.
end	Ends a code block (group of statements) started with begin, class, def, do, if, etc.

Table 1. Ruby's reserved words (continued)

Reserved word	Description
ensure	Always executes at block termination; use after last rescue.
false	Logical or Boolean false; instance of FalseClass; a pseudovvariable. See true.
for	Begins a for loop; used with in.
if	Executes code block if conditional statement is true. Closes with end. Compare unless, until.
in	Used with for loop. See for.
module	Defines a module; closes with end.
next	Jumps to the point immediately before the evaluation of the loop's conditional. Compare redo.
nil	Empty, uninitialized, or invalid; always false, but not the same as zero; object of NilClass; a pseudovvariable.
not	Logical operator; same as !.
or	Logical operator; same as except or has lower precedence.
redo	Jumps after a loop's conditional. Compare next.
rescue	Evaluates an expression after an exception is raised; used before ensure.
retry	When called outside of rescue, repeats a method call; inside rescue, jumps to top of block (begin).
return	Returns a value from a method or block. May be omitted, but method or block always return a value, whether it is explicit or not.
self	Current object (receiver invoked by a method); a pseudovvariable.
super	Calls method of the same name in the superclass. The superclass is the parent of this class.
then	Separator used with if, unless, when, case, and rescue. May be omitted, unless conditional is all on one line.
true	Logical or Boolean true; instance of TrueClass; a pseudovvariable. See false.
undef	Makes a method undefined in the current class.
unless	Executes code block if conditional statement is false. Compare if, until.

Table 1. Ruby's reserved words (continued)

Reserved word	Description
until	Executes code block while conditional statement is false. Compare if, unless.
when	Starts a clause (one or more) under case.
while	Executes code while the conditional statement is true.
yield	Executes the block passed to a method.
__FILE__	Name of current source file; a pseudovvariable.
__LINE__	Number of current line in the current source file; a pseudovvariable.

Operators

Table 2 lists all of Ruby's operators in descending order of precedence. Operators that are implemented as methods may be overridden and are indicated in the Method column.

Table 2. Ruby's operators

Operator	Description	Method
::	Scope resolution	
[] []=	Reference, set	✓
**	Raise to power (exponentiation)	✓
+ - ! ~	Positive (unary), negative (unary), logical negation, complement	✓ (not !)
* / %	Multiplication, division, modulo (remainder)	✓
+ -	Addition, subtraction	✓
<< >>	Shift left, shift right	✓
&	Bitwise <i>and</i>	✓
^	Bitwise <i>or</i> , bitwise exclusive <i>or</i>	✓
> >= < <=	Greater than, greater than or equal to, less than, less than or equal to	✓
<=> == === != =~ !~	Equality comparison (spaceship, equality, equality, not equal to, match, not match)	✓ (not != or !~)

Table 2. Ruby's operators (continued)

Operator	Description	Method
&&	Logical <i>and</i>	
	Logical <i>or</i>	
.. ...	Range inclusive, range exclusive	✓ (not ...)
? :	Ternary	
= += -= *= /= %=	Assignment, abbreviated	
**= <<= >>= &= =	assignment	
^= &&= =		
not	Logical negation	
and or	Logical composition	
defined?	Special operator (no precedence)	

Comments

A comment hides a line, part of a line, or several lines from the Ruby interpreter. You can use the hash character (#) at the beginning of a line:

```
# I am a comment. Just ignore me.
```

Or, a comment may be on the same line after a statement or expression:

```
name = "Floydene" # ain't that a name to beat all
```

You can make a comment run over several lines, like this:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Here is another form. This block comment conceals several lines from the interpreter with =begin/=end:

```
=begin
This is a comment.
This is a comment, too.
This is a comment, too.
I said that already.
=end
```

A block can comment out one line or as many lines as you want.

Numbers

Numbers are not primitives; each number is an object, an instance of one of Ruby's numeric classes. `Numeric` is Ruby's base class for numbers. The numeric class `Fixnum` is used for integers, fixed-length numbers with bit lengths of the native machine word, minus 1. The `Float` class is for floating-point numbers, which use the native architecture's double-precision floating-point representation internally. The `Bignum` class is used to hold integers larger than `Fixnum` can hold. `Bignums` are created automatically if any operation or assignment yields a result too large for `Fixnum`. The only limitation on the size integer `Bignum` can represent is the available memory in the operating system:

```
2411      # integer, of class Fixnum
2_411     # integer, of class Fixnum, underscore ignored
241.1     # float, of class Float
3.7e4     # scientific notation, of class Float
3E4       # scientific notation, of class Float
3E-4      # scientific notation, with sign before
          # exponent
0444      # octal, of class Fixnum
0xffff    # hexadecimal, of class Fixnum
0b1101    # binary, of class Fixnum
4567832704 # integer, of class Bignum
```

Figure 1 shows a hierarchy of Ruby's math classes.

Variables

A *variable* is an identifier that is assigned to an object, and that object may hold a value. The type of the value is assigned at runtime. Ruby variables are not declared nor statically typed. Ruby uses *duck typing*, a kind of dynamic typing. If a value behaves or acts like a certain type, such as an integer, Ruby gives it a context, and it is treated in that context.

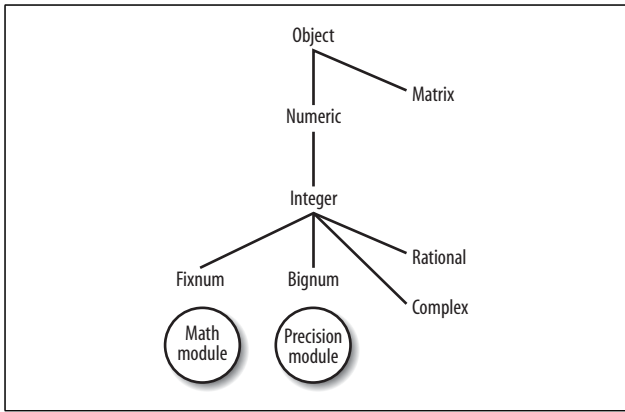


Figure 1. Hierarchy of Ruby math classes

Duck typing comes from the concept that if it walks like a duck, quacks like a duck, flies like a duck, and swims like a duck (or integer or float, etc.), then it is probably a duck. If a variable is able to act like an integer, for example, then it is legal to use it in that context.

Local Variables

A *local variable* has a local scope or context. For example, if a variable is defined inside of a method or a loop, its scope is within the method or loop where it was defined. Local variable names must start with a lowercase letter or with an underscore character (`_`), such as `alpha` or `_beta`, and cannot be prefixed with a special character (as in `@`, `@@`, or `$`).

Instance Variables

An *instance variable* belongs to a particular instance of a class (hence the name) and can only be accessed from outside that instance via an accessor (or helper) method. Instance variables are always prefixed with a single at sign (`@`), as in `@hello`. See the upcoming section “Classes.”

Class Variables

A *class variable* is shared among all instances of a class. Only one copy of a class variable exists for a given class. In Ruby, it is prefixed by two at signs (@@), such as @@times. You have to initialize (declare a value for) a class variable before you use it. See the upcoming section “Classes.”

Global Variables

Global variables are available globally to a program, inside any structure. Their scope is the whole program. They are prefixed by a dollar sign (\$), such as \$amount. Matz’s opinion on global variables is, and I quote, “They are ugly, so don’t use them.” I would take his advice. You can use a singleton instead. See the upcoming section “Singletons.”

Constants

Constant variable names must begin with a capital letter (Matz), and by convention are frequently all capitals (MATZ). This makes them easy to spot. As their name suggests, constants are not expected to change their value after their initial assignment. Because Ruby is a flexible language, there are a couple of notable exceptions to this. First, you can reassign a constant in Ruby, though Ruby will generate a warning if you do, and it’s not a good idea. Second, and more importantly, since constants refer to objects, the contents of the object *to which the constant refers* may change without Ruby generating a warning. Thus, Ruby constants are called *mutable*, because, although the constant is only expected to refer to a single object throughout the program, what’s contained in that object may vary.

Parallel Assignment of Variables

With parallel assignment, you can assign several values to several variables in a single expression. A list of variables, separated by commas, can be placed to the left of the equals

sign, with the list of values to assign them (in order) on the right. Here is an example:

```
x, y, z = 100, 200, 500
```

You can also assign values of different types:

```
a, b, c = "cash", 1.99, 100
```

Symbols

Ruby has a special object called a *symbol*. Symbols are like placeholders for identifiers and strings; they are always prefixed by a colon (:), such as `:en` and `:logos`. Most importantly, *only one copy* of the symbol is held in a single memory address, as long as the program is running. You don't directly create a symbol by assigning a value to one. You create a symbol by calling the `to_sym` or `intern` methods on a string, or by assigning a symbol to a symbol:

```
name = "Brianna"
name.to_sym # => :Brianna
:Brianna.id2name # => "Brianna"
name == :Brianna.id2name # => true
```

Predefined Variables

Table 3 lists all of Ruby's predefined variables.

Table 3. Predefined variables

Predefined variable	Description
<code>\$!</code>	The exception information message containing the last exception raised. <code>raise</code> sets this variable. Access with <code>=></code> in a <code>rescue</code> clause.
<code>\$@</code>	The stack backtrace of the last exception, retrievable via <code>Exception#backtrace</code> .
<code>\$&</code>	The string matched by the last successful pattern match in this scope, or <code>nil</code> if the last pattern match failed. Same as <code>m[0]</code> where <code>m</code> is a <code>MatchData</code> object. Read only. Local.

Table 3. Predefined variables (continued)

Predefined variable	Description
\$`	String preceding whatever was matched by the last successful pattern match in the current scope, or <code>nil</code> if the last pattern match failed. Same as <code>m.pre_match</code> where <code>m</code> is a <code>MatchData</code> object. Read only. Local.
\$'	String following whatever was matched by the last successful pattern match in the current scope, or <code>nil</code> if the last pattern match failed. Same as <code>m.post_match</code> where <code>m</code> is a <code>MatchData</code> object. Read only. Local.
\$+	Last bracket matched by the last successful search pattern, or <code>nil</code> if the last pattern match failed. Useful if you don't know which of a set of alternative patterns matched. Read only. Local.
\$1, \$2...	Subpattern from the corresponding set of parentheses in the last successful pattern matched, not counting patterns matched in nested blocks that have been exited already, or <code>nil</code> if the last pattern match failed. Same as <code>m[n]</code> where <code>m</code> is a <code>MatchData</code> object. Read only. Local.
\$~	Information about the last match in the current scope. <code>Regex#match</code> returns the last match information. Setting this variable affects match variables like <code>\$&</code> , <code>\$+</code> , <code>\$1</code> , <code>\$2</code> , etc. The <i>n</i> th subexpression can be retrieved by <code>\$~[nth]</code> . Local.
\$=	Case-insensitive flag; <code>nil</code> by default.
\$/	Input record separator, newline by default. Works like <code>awk</code> 's <code>RS</code> variable. If it is set to <code>nil</code> , a whole file will be read at once. <code>gets</code> , <code>readline</code> , etc. take the input record separator as an optional argument.
\$\	Output record separator for <code>print</code> and <code>IO#write</code> ; <code>nil</code> by default.
\$,	Output field separator between arguments; also the default separator for <code>Array#join</code> , which allows you to indicate a separator explicitly.
\$;	The default separator for <code>String#split</code> ; <code>nil</code> by default.
\$.	The current input line number of the last file that was read. Same as <code>ARGF.lineno</code> .

Table 3. Predefined variables (continued)

Predefined variable	Description
\$<	The virtual concatenation file of the files given by command-line arguments, or standard input (in case no argument file is supplied). <code>\$<.filename</code> returns the current filename. Synonym for ARGF.
\$>	Default output for <code>print</code> , <code>printf</code> , <code>\$stdout</code> by default. Synonym for <code>\$default</code> .
\$_	Last input line of string by <code>gets</code> or <code>readline</code> in the current scope; set to <code>nil</code> if <code>gets</code> or <code>readline</code> meets EOF. Local.
\$0	Name of the current Ruby program being executed.
\$*	Command-line arguments given for the script. The options for the Ruby interpreter are already removed.
\$\$	Process number (<code>process.pid</code>) of the Ruby program being executed.
\$?	Exit status of the last executed process.
\$:	Synonym for <code>\$LOAD_PATH</code> .
\$"	Array containing the module names loaded by <code>require</code> . Used for prevent <code>require</code> from loading modules twice.
\$DEBUG	True if <code>-d</code> or <code>--debug</code> switch is set.
\$default	Default output for <code>print</code> , <code>printf</code> ; <code>\$stdout</code> by default. Synonym for <code>\$></code> .
\$F	Receives output from <code>split</code> when <code>-a</code> specified. Set if <code>-a</code> is set along with <code>-p</code> and <code>-n</code> .
\$FILENAME	Name of the file currently being read from ARGF. Same as ARGF. <code>.filename</code> or <code>\$<.filename</code> .
\$LOAD_PATH	Synonym for <code>\$:</code> .
\$SAFE	Security level: <ul style="list-style-type: none"> 0 No checks on externally supplied (tainted) data. Default. 1 Potentially dangerous operations using tainted data are forbidden. 2 Potentially dangerous operations performed on processes and files are forbidden. 3 All newly created objects are considered tainted. 4 Modification of global data is forbidden.

- [Staying Well With Guided Imagery pdf, azw \(kindle\), epub, doc, mobi](#)
- [Spanish Vocabulary \(2nd Edition\) \(Practice Makes Perfect\) here](#)
- [Direct Action: Memoirs of an Urban Guerrilla online](#)
- [click **The 80/20 Principle: The Secret of Achieving More with Less**](#)

- <http://ramazotti.ru/library/Staying-Well-With-Guided-Imagery.pdf>
- <http://rodrigocaporal.com/library/Spanish-Vocabulary--2nd-Edition---Practice-Makes-Perfect-.pdf>
- <http://xn--d1aboelcb1f.xn--p1ai/lib/Direct-Action--Memoirs-of-an-Urban-Guerrilla.pdf>
- <http://test.markblaustein.com/library/Ancient-Greek-Athletics.pdf>