Miscellaneous Syntax

Ending an Expression

Ruby uses a newline as the end of an expression. When ending a line with an operator, open parentheses, comma, etc. the expression will continue.

You can end an expression with a ; (semicolon). Semicolons are most frequently used with ruby -e.

Indentation

Ruby does not require any indentation. Typically ruby programs are indented two spaces.

If you run ruby with warnings enabled and have an indentation mis-match you will receive a warning.

alias

The alias keyword is most frequently used to alias methods. When aliasing a method you can use either its name or a symbol:

```
alias new_name old_name
alias :new_name :old_name
```

For methods, Module#alias_method can often be used instead of alias.

You can also use alias to alias global variables:

```
$old = 0

alias $new $old

p $new # prints 0
```

You may use alias in any scope.

undef

The undef keyword prevents the current class from responding to calls to the named methods.

```
undef my_method
```

You may use symbols instead of method names:

```
undef :my_method
```

You may undef multiple methods:

```
undef method1, method2
```

You may use undef in any scope. See also Module#undef_method

defined?

defined? is a keyword that returns a string describing its argument:

```
p defined?(UNDEFINED_CONSTANT) # prints nil
p defined?(RUBY_VERSION)       # prints "constant"
p defined?(1 + 1)              # prints "method"
```

You don't need to use parenthesis with defined? but they are recommended due to the low precedence of defined?.

For example, if you wish to check if an instance variable exists and that the instance variable is zero:

```
defined? @instance_variable && @instance_variable.zero?
```

This returns "expression" which is not what you want if the instance variable is not defined.

```
@instance_variable = 1
defined?(@instance_variable) && @instance_variable.zero?
```

Adding parentheses when checking if the instance variable is defined is a better check. This correctly returns nil when the instance variable is not defined and false when the instance variable is not zero.

Using the specific reflection methods such as instance_variable_defined? for instance variables or const_defined? for constants is less error prone than using defined?.

BEGIN and END

BEGIN defines a block that is run before any other code in the current file. It is typically used in one-liners with ruby -e. Similarly END defines a block that is run after any other code.

BEGIN must appear at top-level and END will issue a warning when you use it inside a method.

Here is an example:

```
BEGIN {
  count = 0
}
```

You must use { and } you may not use do and end.

Here is an example one-liner that adds numbers from standard input or any files in the argument list:

```
ruby -ne 'BEGIN { count = 0 }; END { puts count }; count += gets.to_i'
```