RUBY - SYNTAX

Let us write a simple program in ruby. All ruby files will have extension **.rb**. So, put the following source code in a test.rb file.

```
#!/usr/bin/ruby -w

puts "Hello, Ruby!";
```

Here, I assumed that you have Ruby interpreter available in /usr/bin directory. Now, try to run this program as follows:

```
$ ruby test.rb
```

This will produce the following result:

```
Hello, Ruby!
```

You have seen a simple Ruby program, now lets see few basic concepts related to Ruby Syntax:

Whitespace in Ruby Program:

Whitespace characters such as spaces and tabs are generally ignored in Ruby code, except when they appear in strings. Sometimes, however, they are used to interpret ambiguous statements. Interpretations of this sort produce warnings when the -w option is enabled.

Example:

```
a + b is interpreted as a+b ( Here a is a local variable)
a  +b is interpreted as a(+b) ( Here a is a method call)
```

Line Endings in Ruby Program:

Ruby interprets semicolons and newline characters as the ending of a statement. However, if Ruby encounters operators, such as +, -, or backslash at the end of a line, they indicate the continuation of a statement.

Ruby Identifiers:

Identifiers are names of variables, constants, and methods. Ruby identifiers are case sensitive. It mean Ram and RAM are two different idendifiers in Ruby.

Ruby identifier names may consist of alphanumeric characters and the underscore character _.

Reserved Words:

The following list shows the reserved words in Ruby. These reserved words may not be used as constant or variable names. They can, however, be used as method names.

BEGIN	do	next	then
END	else	nil	true
alias	elsif	not	undef
and	end	or	unless
begin	ensure	redo	until

break	false	rescue	when
case	for	retry	while
class	if	return	while
def	in	self	__FILE__
defined?	module	super	__LINE__

Here Document in Ruby:

"Here Document" refers to build strings from multiple lines. Following a << you can specify a string or an identifier to terminate the string literal, and all lines following the current line up to the terminator are the value of the string.

If the terminator is quoted, the type of quotes determines the type of the line-oriented string literal. Notice there must be no space between << and the terminator.

Here are different examples:

```
#!/usr/bin/ruby -w

print <<EOF
    This is the first way of creating
    here document ie. multiple line string.
EOF

print <<"EOF";                # same as above
    This is the second way of creating
    here document ie. multiple line string.
EOF

print <<`EOC`                 # execute commands
 echo hi there
 echo lo there
EOC

print <<"foo", <<"bar"  # you can stack them
 I said foo.
foo
 I said bar.
bar
```

This will produce the following result:

```
    This is the first way of creating
    her document ie. multiple line string.
    This is the second way of creating
    her document ie. multiple line string.
hi there
lo there
        I said foo.
        I said bar.
```

Ruby *BEGIN* Statement

Syntax:

```
BEGIN {
   code
}
```

Declares *code* to be called before the program is run.

Example:

```
#!/usr/bin/ruby

puts "This is main Ruby Program"

BEGIN {
   puts "Initializing Ruby Program"
}
```

This will produce the following result:

```
Initializing Ruby Program
This is main Ruby Program
```

Ruby *END* Statement

Syntax:

```
END {
   code
}
```

Declares *code* to be called at the end of the program.

Example:

```
#!/usr/bin/ruby

puts "This is main Ruby Program"

END {
   puts "Terminating Ruby Program"
}
BEGIN {
   puts "Initializing Ruby Program"
}
```

This will produce the following result:

```
Initializing Ruby Program
This is main Ruby Program
Terminating Ruby Program
```

Ruby 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 = "Madisetti" # This is again comment
```

You can comment multiple lines as follows:

```
# 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
```