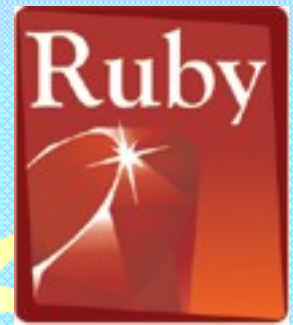


Lesson #11



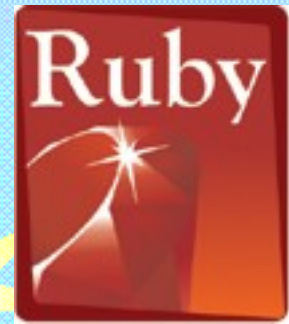
Web Programming with Ruby

Ruby



- Ruby is a reflective, dynamic, object-oriented programming language. It combines syntax inspired by Perl with Smalltalk-like object-oriented features.
- Ruby is a single-pass interpreted language. Its official implementation is free software written in C.

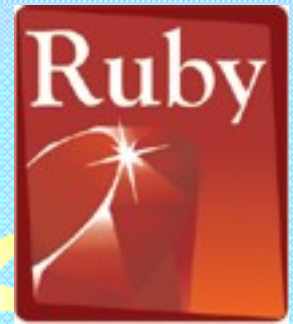
Ruby



- Ruby originated in Japan during the mid-1990s and was initially developed and designed by Yukihiro "Matz" Matsumoto. It is based on Perl, Smalltalk, Eiffel, Ada, and Lisp.

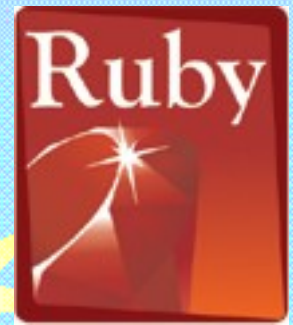


Ruby Syntax



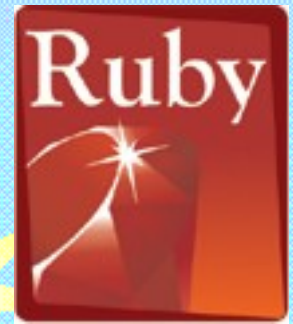
- The syntax of Ruby is broadly similar to Perl. Class and method definitions are signaled by keywords. In contrast to Perl, variables are not obligatorily prefixed with a sigil (like \$). When used, the sigil changes the semantics of scope of the variable.
- In Ruby, ordinary variables lack sigils, but "\$" is prefixed to global variables, "@" is prefixed to instance variables, and "@@" is prefixed to class variables.

Ruby Syntax



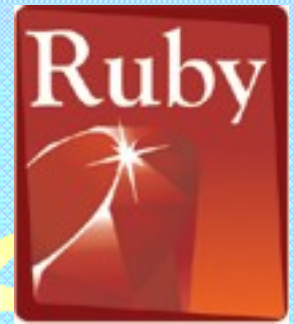
- The most striking difference from C and Perl is that keywords are typically used to define logical code blocks, without braces. Line breaks are significant and taken as the end of a statement; a semicolon may be equivalently used.

Ruby Syntax



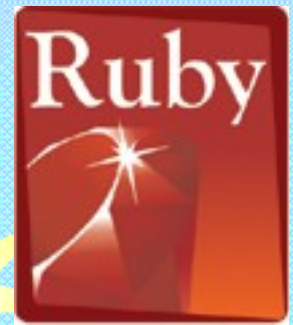
- Comments start with a pound/sharp (#) character and go to the end of line.
- Ruby programs are sequence of expressions.
- Each expression is delimited by semicolons(;) or newlines unless obviously incomplete (e.g. trailing '+').
- Backslashes at the end of line does not terminate expression.

Ruby Types



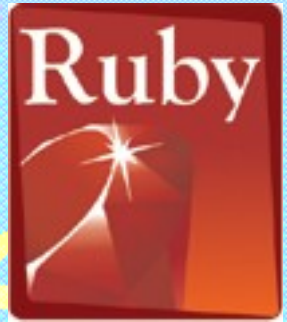
- Basic types are numbers, strings, ranges, symbols, arrays, and hashes. Also included are files.
- They are a combination of types known in Perl and C and they behave pretty much the same.
- For numbers, underscore can be used instead of , (1_000). The exponent operator is `**`.

Ruby Strings



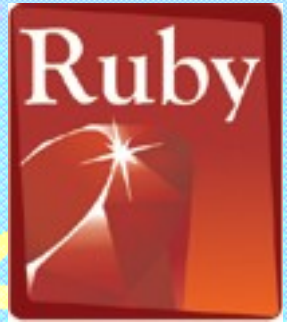
- "this is a string".
- "hello" + "world" => "helloworld"
- "ho " * 3 => "ho ho ho"
- "hello".capitalize => "Hello"
- "hello".reverse => "olleh"
- "hello".next => "hellp"
- "hello".upcase => "HELLO"
- "hello".length => 5
- Other methods include downcase, swapcase and length.

Classes and Objects



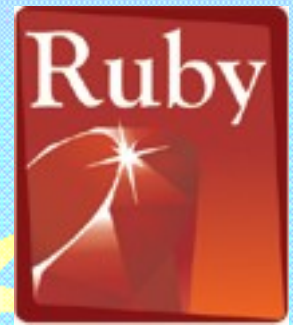
- An object is a unit of data. A class is what kind of data it is. The main classes are Integer, Float and String.
- Division (/) doesn't work the same with integers and floats.
- Addition (+) doesn't work the same with strings as it does with integers.
- Strings have several methods that integers and floats don't have (e.g., capitalize, length, and upcase).

Converting between classes



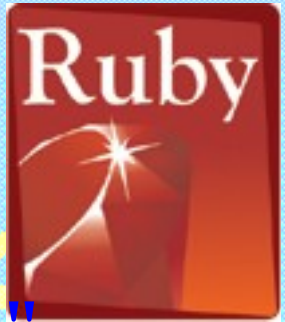
- Some examples of conversions:
- `"hello".to_i => 0`
- `"hello".to_f => 0.0`
- `3.5.to_i => 3`
- `3.5.to_s => "3.5"`
- `4.to_f => 4.0`
- `5.to_s => "5"`
- You can verify the class of an object:
- `12.is_a?(Integer) => true`

Constants and Variables



- Variable names must begin with a lower-case letter.
- Constants are like variables, except that you are telling Ruby that their value is supposed to remain fixed. If you try to change the value of a constant, Ruby will give you a warning (but just a warning!).
- You define constants just like variables, except that the first letter is uppercase.

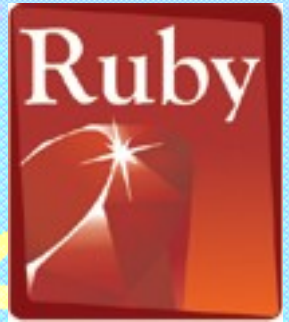
Writing Programs



```
#!/usr/local/bin/ruby -w
print "Content-type: text/html\n\n"
puts "Hello World!\n"
n = 2 + 4 + 6 + 8
n /= 5
m = 2 * 3 * 4
m -= n
puts "The answer is: " + m.to_s
```

- Note that ruby with cgi works like Perl in autonomous programs. Not embedded like PHP or ASP.

Loops



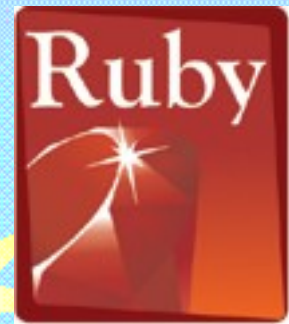
```
#!/usr/local/bin/ruby -w
print "Content-type: text/html\n\n"

4.times do
  puts "Hello World!\n"
end

puts "<br /><br />"

count = 0

5.times do
  count = count + 1
  puts "Count= " + count.to_s
end
```



Conditionals

`==` equal

`!=` not equal to

`>` greater than

`<` less than

`>=` greater than or equal to

`<=` less than or equal to

- If and elsif statements are very similar to Perl's

While loops

The Ruby logo, featuring the word "Ruby" in a white serif font on a dark red background with a stylized white starburst.

```
count = 0
while count < 10
  puts "count = " + count.to_s
  count += 1
end
```

While loops

The Ruby logo, featuring the word "Ruby" in a white serif font on a dark red background with a stylized white gem or starburst.

```
# Calculate the highest power of 2 less  
# than 10,000
```

```
number = 1  
while number < 10_000  
  number *= 2  
end
```

```
# Now 'number' is the highest power of 2  
# greater than 10,000. Divide by 2 to get  
# the value we want.
```

```
number /= 2
```

```
puts number.to_s + " is the highest " + \  
  "power of 2 less than 10,000"
```

To spread
statements
on
multiple
lines

Sorting Arrays

The Ruby logo, featuring the word "Ruby" in a white serif font on a dark red background with a stylized white starburst.

```
primes = [ 11, 5, 7, 2, 13, 3 ]
```

```
[11, 5, 7, 2, 13, 3]
```

```
primes.sort
```

```
[2, 3, 5, 7, 11, 13]
```

```
names = [ "Melissa", "Daniel", "Samantha", "Jeffrey"]
```

```
["Melissa", "Daniel", "Samantha", "Jeffrey"]
```

```
names.sort
```

```
["Daniel", "Jeffrey", "Melissa", "Samantha"]
```

Operations on Arrays

The Ruby logo, featuring the word "Ruby" in a white serif font on a dark red background with a stylized white gemstone.

```
names
["Melissa", "Daniel", "Samantha", "Jeffrey"]
names.reverse
["Jeffrey", "Samantha", "Daniel", "Melissa"]
```

```
names.length
4
```

```
names = [ "Melissa", "Daniel", "Jeff" ]
["Melissa", "Daniel", "Jeff"]
names + [ "Joel" ]
["Melissa", "Daniel", "Jeff", "Joel"]
names - [ "Daniel" ]
["Melissa", "Jeff"]
names * 2
["Melissa", "Daniel", "Jeff", "Melissa", "Daniel", "Jeff"]
```

Operations on Arrays

The Ruby logo, featuring the word "Ruby" in a white serif font on a dark red background with a stylized white gemstone graphic.

```
names
```

```
["Melissa", "Daniel", "Jeff"]
```

```
names.to_s
```

```
"MelissaDanielJeff"
```

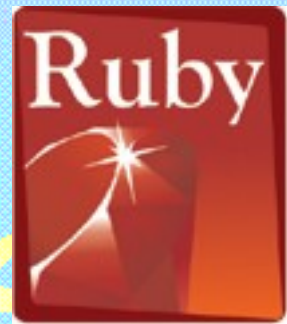
```
primes
```

```
[11, 5, 7, 2, 13, 3]
```

```
primes.to_s
```

```
"11572133"
```

Arrays and Iterators



```
friends = ["Melissa", "Jeff", "Ashley", "Rob"]
friends.each do |friend|
  puts "I have a friend called " + friend
end
```

```
I have a friend called Melissa
I have a friend called Jeff
I have a friend called Ashley
I have a friend called Rob
```

```
friends.sort.each do |friend|
  puts "I have a friend called " + friend
end
```

```
I have a friend called Ashley
I have a friend called Jeff
I have a friend called Melissa
I have a friend called Rob
```

Hashes

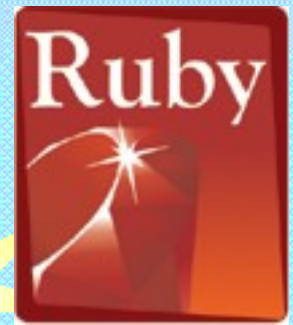
- Hashes are the same as in Perl.
They are pairs of keys and values.

```
student = {  
  "first name" => "Micheal",  
  "last name"  => "Li",  
  "address"    => "2000 Yonge St.",  
  "city"       => "Toronto",  
  "province"   => "Ontario"  
}
```

- You can add to a hash:

```
student["country"] = "Canada"
```

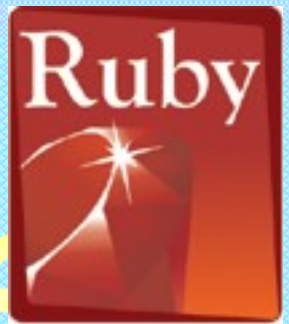
Iterations and Hashes



```
student.each do |key, value|  
  puts key + " => " + value  
end
```

```
city => Toronto  
last name => Li  
country => Canada  
province => Ontario  
address => 2000 Yonge St.  
first name => Michael
```

Functions



```
def say_hi  
  puts "Hello, How are you?"  
end
```

```
say_hi  
say_hi
```

```
"Hello, How are you?"  
"Hello, How are you?"
```

Functions with arguments

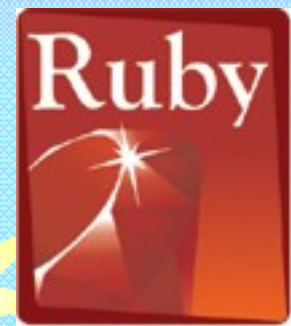
Ruby

```
def say_hi(name)
  puts "Hello " + name + ", How are you?"
end
```

```
say_hi("Daniel")
say_hi "Sandy"
```

```
Hello Daniel, How are you?
Hello Sandy, How are you?
```

The yield method



```
def twice
  yield
  yield
end
```

```
twice { puts "Hello World" }
```

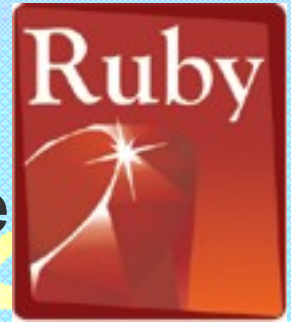
```
Hello World
Hello World
```

```
def names
  yield("Joe")
  yield("Sandy")
  yield("Melissa")
end
```

```
names do |name|
  puts "Hello " + name + ", how are you?"
end
```

```
Hello Joe, how are you?
Hello Sandy, how are you?
Hello Melissa, how are you?
```

Using forms with Ruby/CGI

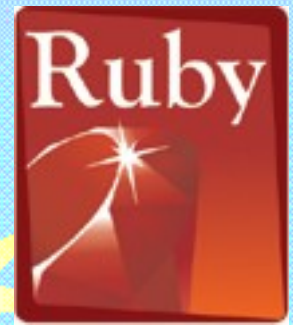


- Using the CGI class gives you access to the HTML query string parameters.

```
#!/usr/local/bin/ruby -w
print "Content-type: text/html\n\n"

require 'cgi'
cgi = CGI.new
puts cgi['FirstName']
puts cgi['LastName']
```

Cookies and Sessions

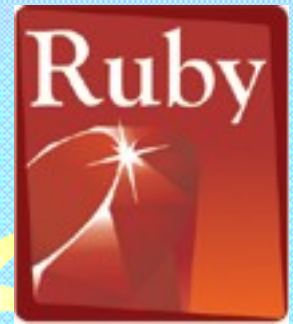


- Visit those URLs for more information on cookies and sessions in Ruby.

http://www.tutorialspoint.com/ruby/ruby_cgi_cookies.htm

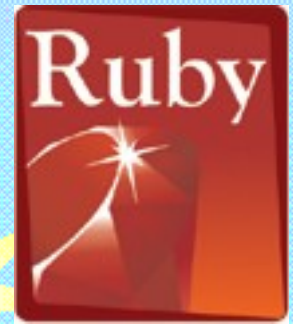
http://www.tutorialspoint.com/ruby/ruby_cgi_sessions.htm

Ruby on Rails



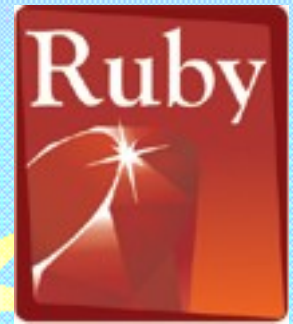
- So far we have been using Ruby under the cgi framework.
- Ruby on Rails is a free web application framework. It aims to increase the speed and ease with which database-driven web sites can be created, and offers skeleton code frameworks (scaffolding) from the outset.
- The fundamental Ruby on Rails principles include Convention over Configuration (CoC) and Don't repeat yourself (DRY).

Ruby on Rails



- "Convention over Configuration" means a developer only needs to specify unconventional aspects of the application.
- For example, if there's a class `Sale` in the model, the corresponding table in the database is called `sales` by default. It is only if one deviates from this convention, such as calling the table `products_sold`, that one needs to write code regarding these names.

Ruby on Rails



- "Don't repeat yourself" means that information is located in a single, unambiguous place.
- For example, using ActiveRecord, the developer does not need to specify database column names in class definitions. Instead, Ruby can retrieve this information from the database.

Using Ruby on Rails

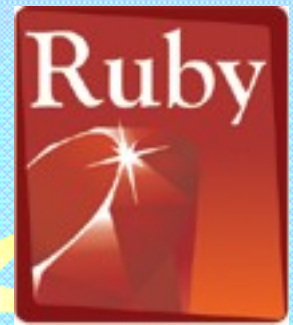


- Ruby on Rails is easy to install on client workstations. There are versions for just about any operating system.
- It installs in to steps:
 - 1) *Ruby*
 - 2) *Rails*

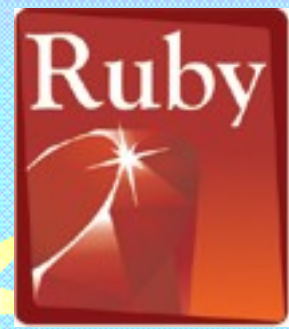
www.ruby-lang.org

www.rubyonrails.org

Using Ruby on Rails



- The easiest way is, of course, to find a Web host that supports it.
- At this time, however, Ruby on Rails packages tend to be more expensive than popular Apache/PHP or IIS/ASP solutions.
- There is also the less known but powerful Zope framework based on the Python language. (www.zope.org)



End of lesson