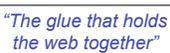


## Scripting Languages

Kathleen Fisher

## What are scripting languages?

- Unix shells: sh, ksh, bash 
  - job control
- Perl 
  - Slashdot, bioinformatics, financial data processing, cgi
- Python 
  - System administration at Google
  - BitTorrent file sharing system
- Ruby 
  - Various blogs, data processing applications
- PHP 
  - Yahoo web site
- JavaScript 
  - Google maps

"The glue that holds  
the web together"

## Characteristics

- Interpreted (no compilation step)
- Dynamically typed
- High-level model of underlying machine
- Garbage collected
- Don't have to declare variables

Designed to support "quick programming"

## Design philosophy

*Often people, especially computer engineers, focus on the machines. They think, "By doing this, the machine will run faster. By doing this, the machine will run more effectively. By doing this, the machine will do something something something." They are focusing on the machines. But in fact we need to focus on humans, on how humans care about doing programming or operating the application of the machines. We are the masters. They are the slaves.*

Yukihiro "Matz" Matsumoto  
Creator of Ruby



## Demo: Getting the homework

- What if I don't want to go to the web site to see if I have cs242 homework?
- Write a script to check for me!

```
> hwk http://www.stanford.edu/class/cs242/handouts/index.html
Hwk 1 was due on Wednesday, October 05.
Hwk 2 was due on Wednesday, October 12.
Hwk 3 is due on Wednesday, October 19.
```

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri= URI.parse(ARGV[0])
h=Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (ld*) \\\(due (ld*)\\(ld*)\\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, duedate|
  if duedate < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{duedate.strftime("%A, %B %d")}."
  else
    puts "Hwk #{assignment} is due on #{duedate.strftime("%A, %B %d")}."
  end
}
```

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri= URI.parse(ARGV[0])
h=Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (\d*) \\\(due (\d*)\\(\\d*)\\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, due_date|
  if due_date < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{due_date.strftime("%A, %B %d")}."
  else
    puts "Hwk #{assignment} is due on #{due_date.strftime("%A, %B %d")}."
  end
}
```

Shebang

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri = URI.parse(ARGV[0])
h = Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (\d*) \\\(due (\d*)\\(\\d*)\\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, due_date|
  if due_date < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{due_date.strftime("%A, %B %d")}."
  else
    puts "Hwk #{assignment} is due on #{due_date.strftime("%A, %B %d")}."
  end
}
```

Many useful libraries

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri= URI.parse(ARGV[0])
h=Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (\d*) \\\(due (\d*)\\(\\d*)\\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, due_date|
  if due_date < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{due_date.strftime("%A, %B %d")}."
  else
    puts "Hwk #{assignment} is due on #{due_date.strftime("%A, %B %d")}."
  end
}
```

Powerful regular expression support

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri= URI.parse(ARGV[0])
h=Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (\d*) \\\(due (\d*)\\(\\d*)\\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, due_date|
  if due_date < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{due_date.strftime("%A, %B %d")}."
  else
    puts "Hwk #{assignment} is due on #{due_date.strftime("%A, %B %d")}."
  end
}
```

Associative arrays

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri= URI.parse(ARGV[0])
h=Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (\d*) \\\(due (\d*)\\(\\d*)\\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, due_date|
  if due_date < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{due_date.strftime("%A, %B %d")}."
  else
    puts "Hwk #{assignment} is due on #{due_date.strftime("%A, %B %d")}."
  end
}
```

String processing

## Shebang

- In Unix systems, shebang tells the O/S how to evaluate an executable text file.

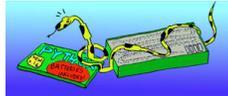
doit: `#!/ interp-path prog-text`  `> interp-path doit args`

`> ./doit args`

- Advantages:** Don't need file extensions, program looks built-in, and can change implementation transparently.

## Large standard library

- Date, ParseDate
- File, Tempfile
- GetoptLong: processing command line switches
- profile: automatic performance profiling
- Pstore: automatic persistence
- BasicSocket, IPSocket, TCPSocket, TCPServer, UDPSocket, Socket
- Net::FTP, Net::HTTP, Net::HTTPResponse, Net::POPMail, Net::SMTP, Net::Telnet
- CGI: cookies, session mngt.



## Contributing users

- Ruby Application Archive (RAA)
  - <http://raa.ruby-lang.org/>
  - 144 library categories, 833 libraries available
  - eg: URI library, database access
- Comprehensive Perl Archive Network (CPAN)
  - <http://www.cpan.org/>
  - 8853 Perl modules from 4655 authors
  - “With Perl, you usually don’t have to write much code: just find the code that somebody else has already written to solve your problem.”



## Example: URI and HTTP libs

```
require 'uri'; require 'net/http'

uri = URI.parse(ARGV[0])
h = Net::HTTP.new(uri.host,80)

resp_data = h.get(uri.path)
```

Require clauses cause Ruby to load named libraries.

## Example: URI and HTTP libs

```
require 'uri'; require 'net/http'

uri = URI.parse(ARGV[0])
h = Net::HTTP.new(uri.host,80)

resp_data = h.get(uri.path)
```

URI.parse converts argument string into a uri object, with **host** and **path** components (among other things).

## Example: URI and HTTP libs

```
require 'uri'; require 'net/http'

uri = URI.parse(ARGV[0])
h = Net::HTTP.new(uri.host,80)

resp_data = h.get(uri.path)
```

Net::HTTP.new creates an http connection object, ready to converse with the specified host on the indicated port.

## Example: URI and HTTP libs

```
require 'uri'; require 'net/http'

uri = URI.parse(ARGV[0])
h = Net::HTTP.new(uri.host,80)

resp_data = h.get(uri.path)
```

h.get asks to retrieve the headers and content of the given path from the site associated with h. It returns a pair of the response code and the payload data.

## Strings

- Strings are just objects:

```
"hermione".length yields 8
```

- Strings can include expressions with # operator:

```
"3 + 4 = #{3+4}" yields "3 + 4 = 7"
```

- Plus operator concatenates strings:

```
"Hermione" + " Granger" yields "Hermione Granger"
```

- Many more operations (more than 75!).

## Powerful regular expressions

- Regular expressions are patterns that match against strings, possibly creating bindings in the process. **Uses greedy matching.**
- In Ruby, regular expressions are objects created with special literal forms:

```
/reg-exp/ or %r{reg-exp}
```

- Examples:

```
/arr/ matches strings containing arr
/s*\s*/ matches a | with optional white space
```

## Simple matches

All characters except <code>.</code> <code>()[]^+\${}*?</code> match themselves	
<code>.( )[]^+\${}*?</code>	Precede by <code>\</code> to match directly
<code>.</code>	Matches any character
<code>[characters]</code>	Matches any single character in <code>[...]</code> May include ranges; Initial <code>^</code> negates
<code>\d</code>	Matches any digit
<code>\w</code>	Matches any "word" character
<code>\s</code>	Matches any whitespace
<code>^</code>	Matches the beginning of a line
<code>\$</code>	Matches the end of a line

## Compound matches

<code>re*</code>	Matches 0 or more occurrences of re.
<code>re+</code>	Matches 1 or more occurrences of re.
<code>re{m,n}</code>	Matches at least m and no more than n occurrences of re.
<code>re?</code>	Matches zero or one occurrence of re.
<code>re1   re2</code>	Matches either re1 or re2
<code>(...)</code>	Groups regular expressions and directs interpreter to introduce bindings for intermediate results.

## Introducing bindings

Matching a string against a regular expression causes interpreter to introduce bindings:

<code>\$'</code>	Portion of string that preceded match.
<code>\$&amp;</code>	Portion of string that matched.
<code>\$'</code>	Portion of string after match.
<code>\$1, \$2, ...</code>	Portion of match within <i>i</i> th set of parentheses.

## Using regular expressions

We can use these bindings to write functions to display the results of a match:

```
def showre(str,regexp)
  if str =~ regexp
    "#{$'}-->#{$&}<---#{$'}"
  else
    "match failed"
  end
end
```

```
def showone(str,regexp)
  if str =~ regexp
    "#{$1}"
  else
    "match failed"
  end
end
```

```
showre("hello", /l+/) yields "he--->ll<---o"
showone("hello", /(l+)/) yields "ll"
```

## Example: Finding homework

To match the homework assignment portion of the course website, we can use the regular expression:

```
/Homework (ld*) \(\due (ld*)\)(ld*)\)/
```

```
<TH>3</TH>
<TD>Homework 3 (due 10/19)</TD>
<!--<TD><a href="hw1.ps">PS</a></TD>-->
<TD><a href="hw3.pdf">PDF</a></TD>
</TR>
```

## Example: Finding homework

To match the homework assignment portion of the course website, we can use the regular expression:

```
/Homework (ld*) \(\due (ld*)\)(ld*)\)/
```

```
<TH>3</TH>
<TD>Homework 3 (due 10/19)</TD>
<!--<TD><a href="hw1.ps">PS</a></TD>-->
<TD><a href="hw3.pdf">PDF</a></TD>
</TR>
```

## Associative Arrays

- Like arrays, indexed collection of objects
- Unlike arrays, index can be any kind of object

```
aa = {'severus' => 'snape', 'albus' => 'dumbledore'}
aa['harry'] = 'potter'
aa['hermione'] = 'granger'
aa['ron'] = 'weasley'

def putaa(aa)
  aa.each{|first,last| puts first + " " + last}
end

puts aa['ginny']
```

```
#!/sw/bin/ruby
require 'uri'; require 'net/http'

uri= URI.parse(ARGV[0])
h=Net::HTTP.new(uri.host,80)

resp.data = h.get(uri.path)
hwk = {}
if resp.message == "OK"
  data.scan(/Homework (ld*) \(\due (ld*)\)(ld*)\)/)
    {[x,y,z] hwk[x] = Time.local(2005,y,z)}
end

hwk.each{| assignment, duedate|
  if duedate < (Time.now - 60 * 60 * 24)
    puts "Hwk #{assignment} was due on #{duedate.strftime("%A, %B %d")}"
  else
    puts "Hwk #{assignment} is due on #{duedate.strftime("%A, %B %d")}"
  end
}
```

## Other features

- Reflection allows querying an object for its capabilities at run-time
  - `obj.class` returns the class of an object
  - `obj.methods` returns its methods
- "Native" modules
  - Relatively easy to implement Ruby modules in C for better performance.
  - Provides APIs to access Ruby objects as C data structures
- Swig allows wrapping of existing C/C++ libraries to import into various scripting languages.

## Tainting

- **Problem:** How to ensure untrusted input data does not corrupt one's system?
- **Solution:**
  - Track the influence of input data, marking dependent data as *tainted*.
  - Disallow risky actions based on tainted data depending upon a programmer-specified safety level.

```
In Ruby, the default safety level (0) permits everything.
Levels 1 to 4 add various restrictions;
The demo program fails to run at level 1.
```

## Design Slogans

- Optimize for people, not machines
- Principle of Least Surprise (after you know the language well...)
- There's more than one way to do it (TMTOWTDI, pronounced Tim Toady)
- No built-in limits
- Make common things short
- Make easy tasks easy and hard tasks possible
- Executable pseudo-code

## Some downsides...

- "Write once, read never"
  - Perl in particular seems to facilitate writing difficult to read programs. A consequence of TMTOWTDI?
- Performance can be difficult to predict
  - Fast: regular expression processing
  - Slow: threads
  - Shell calls?
- Errors are detected dynamically

## The past...

- Unix shells: sh, ksh, bash (1971) 
- Perl (Larry Wall, 1987) 
- Python (Guido van Rossum, 1990) 
- Ruby (Yukihiro "Matz" Matsumoto, 1995) 
- PHP (Rasmus Lerdorf, 1995) 
- JavaScript (Brendan Eich, 1995) 

Two things to note:

- Each language was driven by one person
- 1995 was a big year...

## The present...

- Ruby, Perl, Python, etc., are all open source.
- Rely on volunteers to
  - Write documentation
  - Write test cases
  - Maintain the systems
  - Port to new platforms
  - Fix bugs
  - Implement libraries
  - Implement new features
  - and more...

## Sprit of fun

- "The joy of Ruby"
- "Golfing"
  - Competitions in which each entrant endeavors to solve some problem with the minimum of keystrokes.
- Poetry
  - About the language
  - In the language
  - Generated by the language
  - Original and transliterations
- Obfuscation competitions

```
print STDOUT q
Just another Perl hacker,
unless $spring
    Larry Wall
```

## Obfuscation contests

Any guesses as to what this Perl program does?

```
$P=split//,"URRUU\c8R";$d=split//,"nrekoah xiu / lreP rehtona tsuJ";sub p{
  $p("r$P","u$P")={P,P};pipe"r$P","u$P";++$p;($q+=2)+=$f=1fork;map($P=$P{$f"ord
  ($P{$_})%6};$p($_)=/ ^$P/ix?SP:close$ keys$P;p;p;p;p;map($P{$_}=~/[P.]%&
  close$_$P;wait until!$?;map{"/r/%&<$>"}$P;$=_$d($q);sleep rand(2)if/\$/;print
```

