Web Application Architectures

Module 4: The Ruby Programming Language
Lecture 4: Strings, Regular Expressions and Symbols

THE UNIVERSITY of NEW MEXICO

Strings

- You can create string literals in Ruby using either single or double quotes.
- You can do a little bit more with double quoted strings. E.g., you can insert arbitrary Ruby expressions using string interpolation.
  Ex.

```
> "360 degrees=#{2*Math::PI} radians"
=> "360 degrees=6.283185307179586 radians"
```

- If you enclose a string in single backquotes (backticks), the string will be executed as a command in the underlying OS.
  Ex.

```
>  `date`
=> "Tue Oct 15 09:10:21 MDT 2013\n"
```

Strings

- Strings in Ruby are mutable, as in C/C++, but unlike Java. Thus, each time Ruby encounters a new string literal, it create a new String object. I.e, if you're creating a string literal within a loop, each iteration will create a new String object.
- The Ruby String class contains a number of methods which can be used to manipulate strings.

Ex.

```
> name = "Homer Blimpson" # => "Homer Blimpson"
> name.length     # => 14
> name[6]         # => "B"
> name[6..14]     # => "Blimpson"
> "Bart " + name[6..14]  # => "Bart Blimpson"
> name.encoding   # =>  #<Encoding:UTF-8>
```

Regular Expression Class

THE UNIVERSITY of NEW MEXICO

- Ruby has a regular expression class, called Regexp, that is closely related to strings.
- A regular expression provides a concise and flexible means for matching strings of text, such as particular characters, words, or patterns of characters.
- In Ruby, a regular expression is written in the form of:

  ```
  /pattern/modifiers
  ```

  where "pattern" is the regular expression itself, and "modifiers" are a series of characters indicating various options. The "modifiers" part is optional. This syntax is borrowed from Perl.
- To test if a particular Regex matches (part of) a string, use the =~ operator. This operator returns the character position in the string of the start of the match (which evaluates to true in a boolean test), or nil if no match was found (which evaluates to false).

  Ex. `"Homer" =~ /er/  # => 3`

Regular Expressions

THE UNIVERSITY of NEW MEXICO

The following have special meanings in patterns:

	meaning
[ ]	range specification, e.g., [a-z] means a letter between a and z
\w	word character, same as [0-9A-Za-z_]
\W	non-word character
\s	space character, same as [\t\n\r\f]
\S	non-space character
\d	digit character, same as [0-9]
\D	non-digit character
\b	backspace (if used in a range specification)
\b	word boundary (if not used in a range specification)
\B	non-word boundary
*	zero or more repetitions of the preceding
+	one or more repetitions of the preceding
{m,n}	at least m and at most n repetitions of the preceding
?	at most one repetition of the preceding, same as {0,1}
|	either preceding or next expression may match
( )	grouping

Regular Expressions

THE UNIVERSITY of NEW MEXICO

- The preceding table only contained a partial list of the special characters that can be used in a Ruby regular expression. Consult a Ruby reference for more details.
- Regular expression are often used to process strings.
  Ex. The following Ruby expression will replace all of the non-digit characters in `phone` with `""`. I.e., it will strip everything out of the phone number, except digits:

```
phone = phone.gsub!(/\D/, "")
```

- Regular expression are commonly used to validate emails, phone numbers, and other user-supplied input.
  Ex. The following regular expression can be used to validate email addresses:

```
/\A[\w\._%-]+@[\w\.-]+\.[a-zA-Z]{2,4}\z/
```

(Note: We didn't cover all of the characters used in this regular expression.)

Symbols

THE UNIVERSITY of NEW MEXICO

- Ruby symbols are also closely related to strings.
- A Ruby interpreter maintains a symbol table where it stores the names of all classes, methods and variables.
- You can add your own symbols to this table. Specifically, a symbol is created if you precede a name with a colon.
  Ex.

```
attr_reader :row, :col
```

- Ruby symbols are used to represent names and strings; however unlike String objects, symbols of the same name are initialized and exist in memory only once during a Ruby session.
- Ruby symbols are immutable, and cannot be modified during runtime.
  Ex.

```
:name = "Homer"     # => will yield an error
```

Symbols

- There's a big space advantage associated with symbols, as each unique is only stored once in memory. Multiple strings with the same name my exist in memory.
  Ex.

```
> puts :name.object_id    # => yields 20488
> puts :name.object_id    # => yields 20488
> puts "name".object_id   # => yields 2168472820
> puts "name".object_id   # => yields 2168484060
```

- When should you use a string and when should you use a symbol? General rules of thumb:
  - If the contents (i.e., the sequence of characters) of the object is important, e.g., if you need to manipulate these characters, use a string.
  - If the identity of the object is important (in which case you probably don't want to manipulate the characters), use a symbol.