CMSC 330: Organization of Programming Languages

Ruby, Part 3

Standard Library: Hash

- A hash acts like an associative array, in which elements can be indexed by any types of values, not necessarily just integers
  - every Ruby object can be used as a hash key, because the Object class has a hash method
- Elements are referred to using [] like array elements, but Hash.new is the Hash constructor

```
italy= Hash.new()
italy["population"]= 58103033
italy["continent"]= "europe"
italy[1861]= "independence"
```


Standard Library: Hash (con't.)

- The Hash method values returns an array of a hash's values (in some order)
- And keys returns an array of a hash's keys (in some order)
- Iterating over a hash:

```
italy.keys().each() { |key|
  puts("key: #{key}, value: #{italy[key]}")
}
```


Standard Library: Hash (con't.)

Convenient syntax for creating literal hashes

- use { key => value, ... } to create hash table

```
credits = {
  "cmsc131" => 4,
  "cmsc330" => 3,
}

x = credits["cmsc330"]  # x now 3
credits["cmsc351"] = 3
```

Standard Library: File

- Lots of convenient methods for I/O

```
File.new("file.txt", "rw")      # open for rw access
File.readline                   # reads the next line from a file
File.readlines                  # returns an array of all file lines
File.eof                        # return true if at end of file
File.close                      # close file
f << object                     # convert object to string and write to f
$stdin, $stdout, $stderr        # global variables for standard UNIX IO
```

  By default stdin reads from keyboard, and stdout and stderr both write to terminal

- File inherits some of these methods from IO

Exceptions

- Use begin...rescue...ensure...end
  - like try...catch...finally in Java

```
begin
  f = File.open("test.txt", "r")
  while !f.eof
    line = f.readline
    puts line
  end
  f.close
rescue Exception => e
  puts("Exception:" + e + " (class "
      + e.class.to_s + ")")
end
```

Class of exception to catch

Local name for exception

Command Line Arguments

- Stored in predefined array variable $*
  - Can refer to as predefined global constant ARGV

- Example
  - If a program test.rb is invoked as ruby test.rb a b c
  - Then
    - ARGV[0] = "a"
    - ARGV[1] = "b"
    - ARGV[2] = "c"

Ruby Summary

- Interpreted
- Implicit declarations
- Dynamically typed
  - these three make it quick to write small programs
- Object-oriented
  - everything (!) is an object
- Code blocks
  - easy higher-order programming!
  - get ready for a lot more of this...
- Built-in regular expressions and easy string manipulation
  - this and the first three are the hallmark of scripting languages

Other Scripting Languages

- Perl and Python are also popular scripting languages
  - also are interpreted, use implicit declarations and dynamic typing, have easy string manipulation
  - both include optional “compilation” for speed of loading/execution
- Will look fairly familiar to you after Ruby
  - lots of the same core ideas
  - all three have their proponents and detractors
  - use whichever one you like best

Example Perl Program

```
#!/usr/bin/perl
foreach (split(//, $ARGV[0])) {
  if ($G{$_}) {
    $RE .= "\\" . $G{$_};
  } else {
    $RE .= $N ? "(?!\\" .
      join("|\\", values(%G)) . ')(\w)' : '(\w)';
    $G{$_} = ++$N;
  }
}
```


Example Python Program

```
#!/usr/bin/python
import re
list = ("cattle", "deep", "deer", "duck", "dog",
        "dogwood")
x = re.compile("^\S{3,5}.[aeiou]")
for i in list:
  if re.match(x, i):
    print I
  else:
    print
```