Ruby

Intro

- Ruby was first released in 1995 by Yukihiro "Matz" Matsumoto
  - It took several years to gain popularity outside of Japan
- In 2004, Ruby's popularity increased with the release of the Rails package
- Current version is 2.5.0

Technical Details

- Is a scripting language
- Is very object oriented
  - Everything is an object!
- The original interpreter was replaced with YARV (Yet another Ruby VM) in 2009
  - Lead to a large performance boost
- Great support for both text processing as well as interacting with the OS

Popular Projects in Ruby

- Ruby on Rails
  - A popular web framework, an alternative to PHP
- GitHub's Linguist
  - The code responsible for determing the language a file is written in
- Homebrew
  - A Linux style package manager available for MacOS
- Vagrant
  - Software to manage VMs

Running Ruby

- Ruby is run differently depending on if it is interactive or batch mode
- The interactive ruby interpreter is launched using the command `irb`
- To run an existing file, use the `ruby` command
  - Files usually end in `.rb`

Integers

- Everything is an object!! Including integers

```
5.+(4) == 5 + 4
```

- There are two integer types, Ruby will switch between them as needed
  - Fixnum
  - Bignum

In [ ]:
```
5.+(4) == 5 + 4
```

In [ ]:
```
5.class
```

In [ ]:

```
alot = 200 ** 200
puts alot
puts alot.class
```

Integer Methods

- There is no decrement or increment method, so code needs to use something like

```
counter += 1
counter = counter.next
```

- Conversion methods
  - `5.to_s`
  - `5.to_f`
- Common mathemetical methods
  - `.abs`
  - `.even?`
  - `5[0]`

In [ ]:
```
10.next
```

In [ ]:
```
-1.abs()
```

In [ ]:
```
puts 1.even?
puts 2.even?
```

In [ ]:
```
print 5[0], 5[1], 5[2]
```

In [ ]:
```
print 4[0], 4[1], 4[2]
```

Floats

- Floats in Ruby are double precision, so they can overflow
- Most standard mathematicl methods are available
  - `10.50.ceil`
  - `10.50.round`
  - `10.50.to_i`
  - `10.50.to_r`

In [ ]:
```
10.50.to_i
```

In [ ]:
```
10.5.to_r
```

In [ ]:
```
10.49.round
```

In [ ]:
```
10.49.ceil
```

Strings

- Strings can be single or double quotes
  - Single quotes strings don't evaluate escapes like \n, etc.
- Variable interpolation is available in double quoted strings only
  - The syntax is `#{varname}`
- Many string methods exist in two forms
  - Those that return a string are named normally
  - Those that modify the string in place have a `!` at the end of the method name

In [ ]:
```
str = "A string. A nother"
int = 10
```

In [ ]:
```
str.upcase
```

In [ ]:
```
str.capitalize
```

In [ ]:
```
str.downcase!
puts str
```

In [ ]:
```
str.length
```

In [ ]:
```
str.ascii_only?
```

In [ ]:
```
puts "      soooo many spacess and        such".squeeze
puts "      soooo many spacess and        such".squeeze(" s")
```

In [ ]:
```
puts str.include?("ri")
puts str.include?("ro")
```

In [ ]:

```
puts "Hello, you are #{str}"
puts "Hello, you are #{str.upcase}"
puts "Hello you are a float now: #{int.to_f}"
```

Arrays

- Arrays are Ruby's list type
  - They are heterogenous
- They can be created using the `Array` constructor

```
Array.new(capacitiy,initial)
```

- The shortcut syntax to initialize an array is

```
var = [1, 2, 3, 4]
```

Array Indexing

- Ruby supports negative indexing from the end of the array
- Ruby also has two ways to get a subsection of the list
  - `array[index,howmany]`
  - `array[start..end]`
    - Must have a value for `end`, could be `-1`

In [ ]:
```
arr = [10,20,4,6,7,1,20,0,nil]
puts arr[0]
puts arr[-2]
```

In [ ]:
```
puts arr[1,2]
puts arr[4,1]
```

In [ ]:
```
puts arr[0..2]
puts arr[5..-1]
```

Array Methods

- There are many many array methods
- Common/Interesting ones are:
  - `.push`/`.pop`
  - `.compact` - removes `nil` elements
  - `.uniq`
  - `.inspect`

In [ ]:
```
puts arr.compact
```

In [ ]:
```
puts arr.uniq
```

In [ ]:
```
puts arr.inspect
```

Set Operations

- Arrays can also be used like sets, by applying set operations between two arrays
  - & - Set Intersection
  - | - Set Union
- There is also the difference operator, but this isn't set difference

In [ ]:
```
[1, 2, 4, 4] | [5, 6 ,7]
```

In [ ]:
```
[1, 2, 4, 4] & [5 , 6, 7]
```

In [ ]:
```
[1, 2, 4, 4] & [4, 10, 20]
```

In [ ]:
```
[1 ,2 , 3 ,4 ,4 ] - [1]
```

Hashes

- The associative array structure in Ruby is a hash
- Can be declared using `Hash.new`, but is often initialized expclicitly

```
hash = {'key1' => 'val1' , 'key2' => 'val2', ....}
```

- The key can be any type
- To set a default value for keys not in the hash, pass it as a paramter to the constructor

In [ ]:

```
hash1 = {'google' => 'www.google.com', 'umbc' => 'umbc.edu',
  'state of maryland' => 'md.gov'}
hash2 = Hash.new(0)
puts hash1
puts hash2
```

Hashes

- Indexing into hashes is done using square brackets (`[]`) like arrays
- New keys can be added by using square brackets as well
- Common hash methods
  - `.keys`/`.values`
  - `.has_key?`
  - `.delete`
  - `.invert`

In [ ]:
```
puts hash1['google']
```

In [ ]:
```
puts hash1['something']
```

In [ ]:
```
puts hash2['not_here']
```

In [ ]:
```
puts hash1.keys
```

In [ ]:
```
puts hash2.has_key?("not_here")
```

In [ ]:
```
hash2["not_here"] = 10
puts hash2.invert.inspect
hash3 = hash2.invert
puts hash3['missing']
```

Control Structures

- If
  - `then` is optional, but exists
  - `elsif`
- Case
  - Uses keyword `case`
  - Cases are denoted using `when`
    - Can be any logical statement

In [ ]:
```
if 10 > 11 then puts "HMMM" else puts "Seems good" end
```

In [ ]:

```ruby
a = 1
b = 2
if a > b
  puts "#{a} is bigger than #{b}"
elsif a == b
  puts "#{a} equals #{b}"
else
  puts "#{a} is less than #{b}"
end
```

In [ ]:

```
result = if a > b
  "#{a} is bigger than #{b}"
elsif a == b
  "#{a} equals #{b}"
else
  "#{a} is less than #{b}"
end

puts "I have compared #{a} and #{b} and have determined that #{result}"
```

In [ ]:

```ruby
case "Hello"
  when /^A/
    puts "You start with an A"
  when /^H/
    puts "You start with an H"
  else
    puts "You start with something else"
end
```

In [ ]:

```ruby
number = 10
case
  when number % 2 == 0
    puts "#{number} is even"
  when number % 2 == 1
    puts "#{number} is odd"
end
```

For Loops

- There is no count-based for loop in Ruby
- The for-in loop takes its place, and it useful, but there is a much better solution in Ruby

```ruby
for var in array
  #Do something
end
```

In [ ]:

```ruby
arr = [1,2,4,"Thing",nil]
for el in arr
  puts el
end
```

In [ ]:

```
hash = {'today' => "Thursday", "tomorrow" => "Friday"}
for el in hash
  puts el
end
```

In [ ]:

```ruby
hash = {'today' => "Thursday", "tomorrow" => "Friday"}
for k,v in hash
  puts "#{k.capitalize} is #{v}"
end
```

Iterators

- Almost every object has at least one method that is an interator
- This is a special method that you can provide a block of code to
  - The block can be one line between curly braces
  - Can be multiple lines, denoted by `do |vars| ... end`

Common Iterators

- Intengers
  - `.times`
  - `.upto`/`.downto`
- Arrays and Hashes
  - `.each`
- Strings
  - `.each_line`
  - `.each_char`

In [ ]:
```
5.times {puts "Hi"}
```

In [ ]:

```ruby
5.upto(10) do |i|
  puts i * i
end
```

In [ ]:

```ruby
5.downto(10) do |i|
  puts i * i
end
```

In [ ]:

```
hash = {'today' => "Thursday", "tomorrow" => "Friday"}
hash.each {|k,v| puts "#{k.capitalize} is #{v}"}
```

In [ ]:

```
str_long = "This is a really long string\n I have put some new line characters\n t
o see what happens"
str_long.each_line do |l|
  puts l.strip
end
```

In [ ]:

```
str_long.each_char do |l|
  puts l.strip
end
```

Methods

- Methods in Ruby are defined using the `def` keyword
- They don't need to be in a class
- Global variables in Ruby must start with a `$`
- To process a block, use the `yield` keyword

In [ ]:

```
def square(i)
  i * i
end
square(10)
```

In [ ]:
```
def wrapper(i)
  yield
end
```

In [ ]:
```
wrapper(10) {puts "Hello"}
```

In [ ]:
```ruby
def wrapper_with_var
  x = yield("Hello")
  puts "Yield returned #{x}"
end
```

In [ ]:
```ruby
wrapper_with_var do |i|
  puts "i was passed as #{i}"
  i.downcase
end
```

Process Control

- Ruby is often used as a nicer system scripting language
- The method `system` will run the enclosed commands, returning true or false
- To get data back from a system call, it needs to be opened using the IO class
  - `IO.popen {block}`
  - The block takes one parameter, a stream, that we can use to `gets` data from

In [ ]:

```
IO.popen("ls -lh *.html") do |stream|
  while line = stream.gets do
    parts = line.split(" ")
    puts "#{parts[-1]} is #{parts[4]} big"
  end
end
```

In [ ]:

```
IO.popen("ls -lh *.html") do |stream|
 stream.each do |line|
    parts = line.split(" ")
    puts "#{parts[-1]} is #{parts[4]} big"
  end
end
```

Objects

- You can make your own objects....because everything is an object
- Prefix with the `class` keyword
  - Name must start with a capital letter
- A member variable should start with `@`
  - This makes it private
- The constructor is written as `intialize`
- Getters have the same name as the variable you are trying to get, setters do too, but end with =

In [ ]:

```ruby
class TIME

  def initialize(hour,min)
    @hour = hour
    @min = min
  end

  def hour
    @hour
  end

  def hour=(nHour)
    @hour = nHour
  end

  def to_s
    "It's #{@hour}:#{@min}"
  end
end
```

In [ ]:

```
now = TIME.new(11,15)
puts now
```

Objects Continued

- To overload an operator, use the literal operator name
  - To overload +, define + in your class
- Classes in Ruby are open, meaning they can be added to at any time
  - The syntax is the same, and methods are either added or overwritten
  - This applies to classes that are defined as part of the Ruby langauge too!

In [ ]:
```
class TIME

  def min
    @min
  end

  def +(anotherTime)
    TIME.new(self.hour + anotherTime.hour,
      self.min + anotherTime.min)
  end
end
```

In [ ]:
```
puts now + TIME.new(11,12)
```

In [ ]:
```ruby
class Array
  def beMean
    self[0] , self[-1] = self[-1], self[0]
  end
end
```

In [ ]:
```ruby
arr = [1,43,9,68,19,6890,185,3]
arr.beMean
puts arr
```

Gems

- Packages in Ruby are known as Gems
- The `gem` command line program is usually installed when installing Ruby

```
gem install packageName
```

- The main repository is RubyGems.org