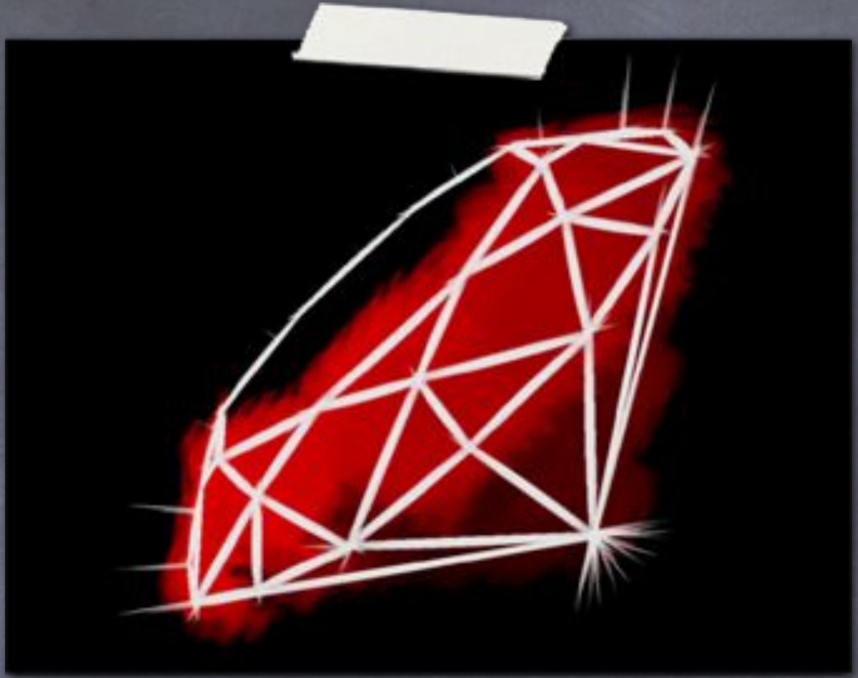


The Top 10 Reasons The Ruby Programming Language *Sucks!*

10. Too young

No Libraries

- A collection of good libraries, especially in something like Perl's CPAN, means less work to achieve better results, faster
- Ruby libraries:
  - 96 standard libraries
  - Ruby Application Archive (RAA) catalogs over 1,200 applications and libraries
  - RubyForge is hosting over 800 open source projects
  - RubyGems has served over 900,000 gems

No support

- Documentation
  - Core 100% documented
  - Standard library documentation in progress
  - Tutorials available for various skill levels
  - Facets of Ruby book series
- Community
  - Mailing lists in multiple languages
  - Usenet groups (with ML gateway)
  - Web forums

No one using it

- Companies using Ruby
  - HP, Intel, NASA, and NOAA
- Uses for Ruby
  - Simulation, data munging, code generation, image processing, prototyping, and more
- "Killer app"
  - Ruby on Rails
    - Already being used in profitable web applications like Basecamp and Blinksale

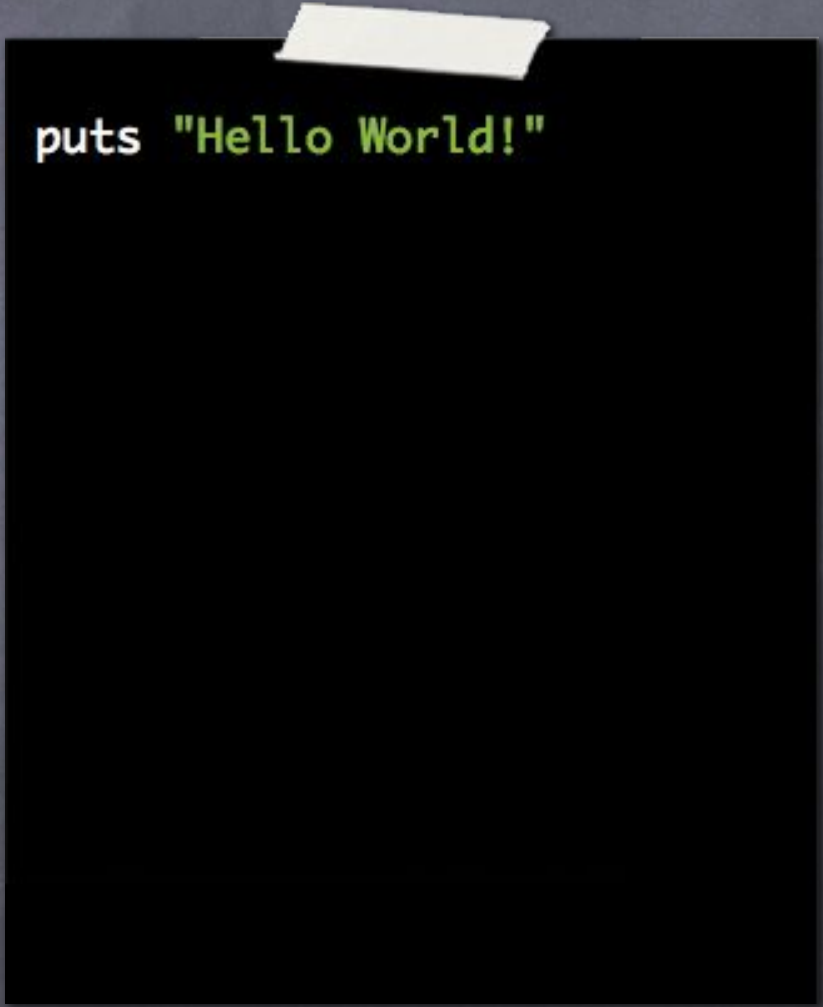
10. Too young

Ahead of its time!

9. Useless in obfuscation contests

Optional Syntax

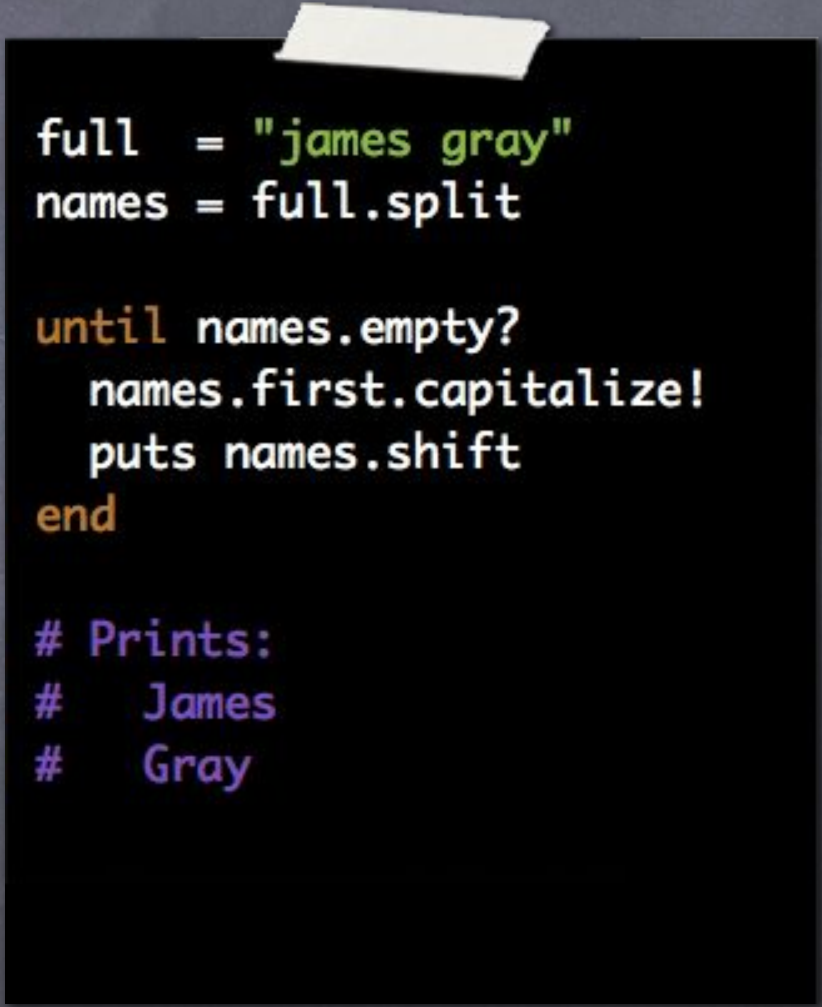
- No ;s needed
- Drop the “\n” characters
- Optional ()s



```
puts "Hello World!"
```

Objectified Syntax

- obj.attribute = methods
- dangerous! and query? methods



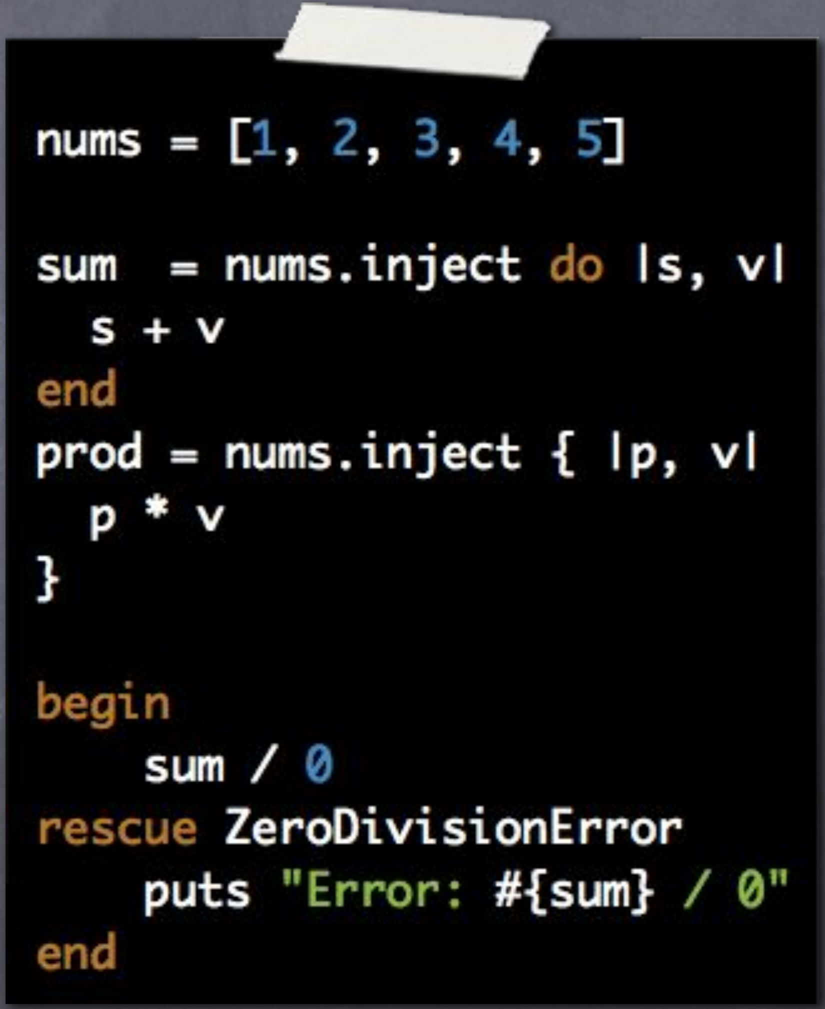
```
full  = "james gray"
names = full.split

until names.empty?
  names.first.capitalize!
  puts names.shift
end

# Prints:
#   James
#   Gray
```

Simple, flexible syntax

- Simple declarations:
  - local_var = ...
  - @instance_var = ...
  - $global_var = ...
- do...end or {...}
- Real exception handling, like Java
- String interpolation: any Ruby code inside #{...}



```
nums = [1, 2, 3, 4, 5]

sum  = nums.inject do |s, v|
  s + v
end
prod = nums.inject { |p, v|
  p * v
}

begin
    sum / 0
rescue ZeroDivisionError
    puts "Error: #{sum} / 0"
end
```

Compare with other languages

- Ruby: puts “Hello world!”
- Java: threeVeryLongLines.weHopeWork…
- Perl: #$<!&;
- Lisp: ((a(((b)))(c)))

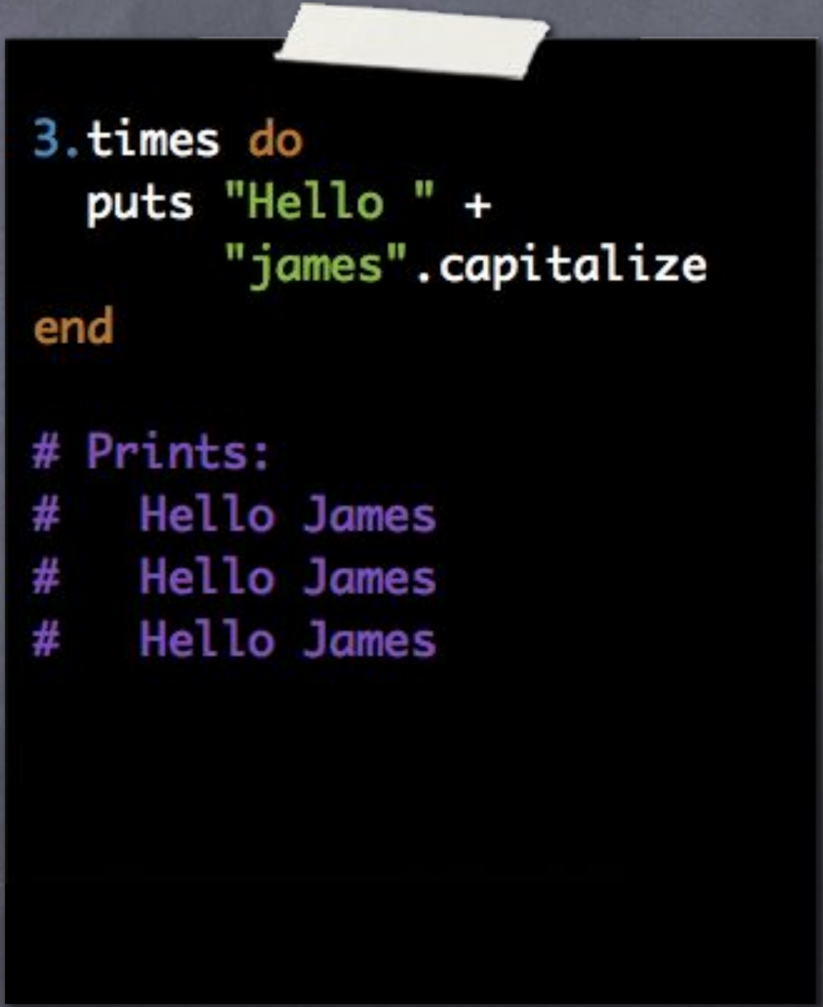
9. Useless in obfuscation contests

Clean syntax!

8. Object Oriented

Ruby is object oriented

- Everything is an object
    - Numbers, code blocks, everything
- Baked-in, not bolted-on
    - No need to use “self” everywhere, like Python

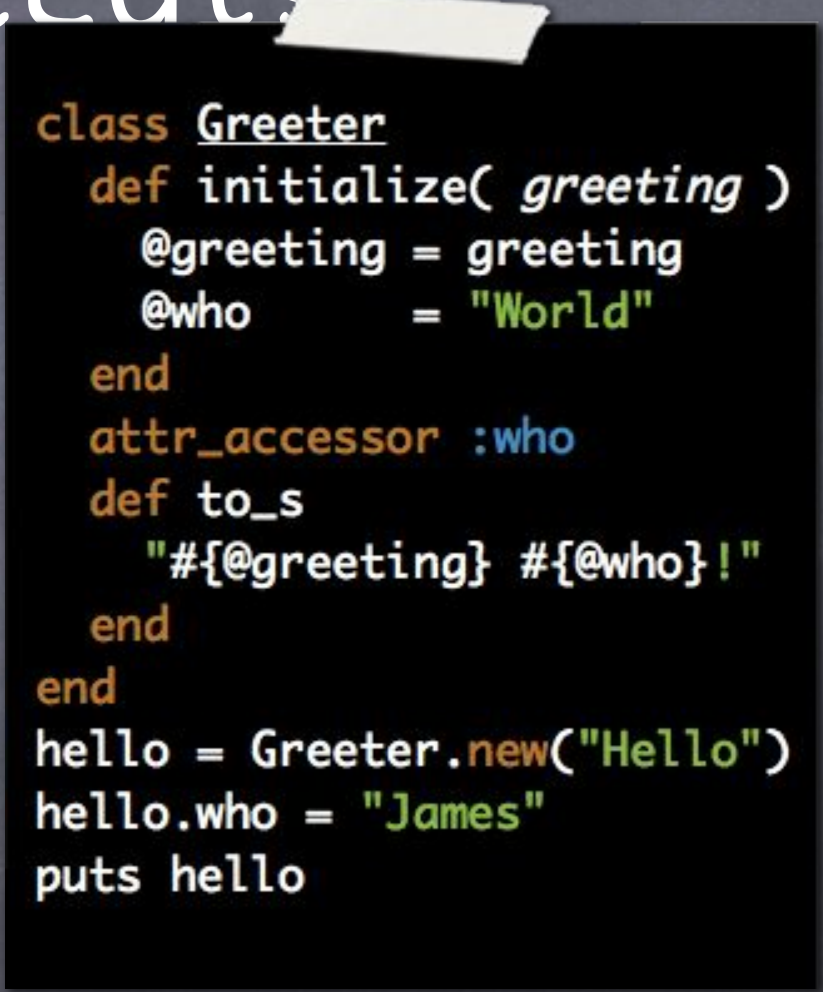


```
3.times do
  puts "Hello " +
       "james".capitalize
end

# Prints:
#   Hello James
#   Hello James
#   Hello James
```

Ruby has many object orientation shortcuts

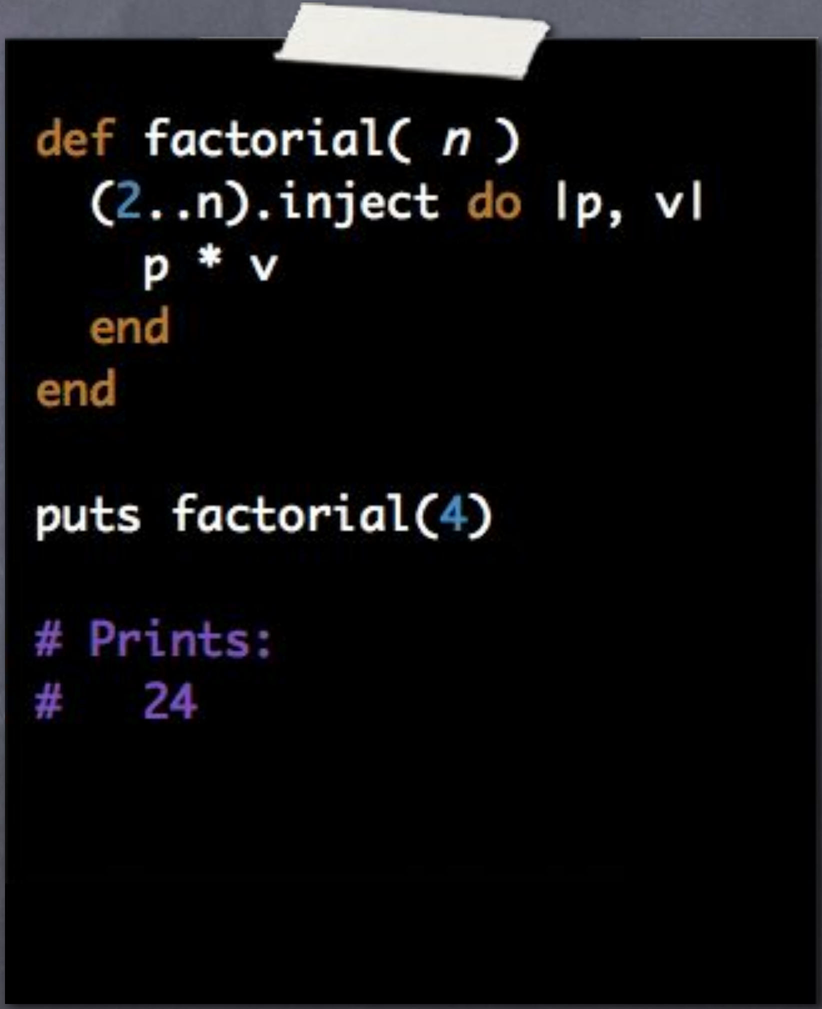
- Automatic constructor generation, unlike Perl
- Easy accessors
- Define methods to interact with Core Ruby



```ruby
class Greeter
  def initialize( greeting )
    @greeting = greeting
    @who      = "World"
  end
  attr_accessor :who
  def to_s
    "#{@greeting} #{@who}!"
  end
end
hello = Greeter.new("Hello")
hello.who = "James"
puts hello
```

Procedural code allowed

- You can ignore the class system as needed
- You can even mix and match objects with procedural code



```ruby
def factorial( n )
  (2..n).inject do |p, v|
    p * v
  end
end

puts factorial(4)

# Prints:
#   24
```

8. Object Oriented

Too flexible!

7. Uses “Mix-ins”

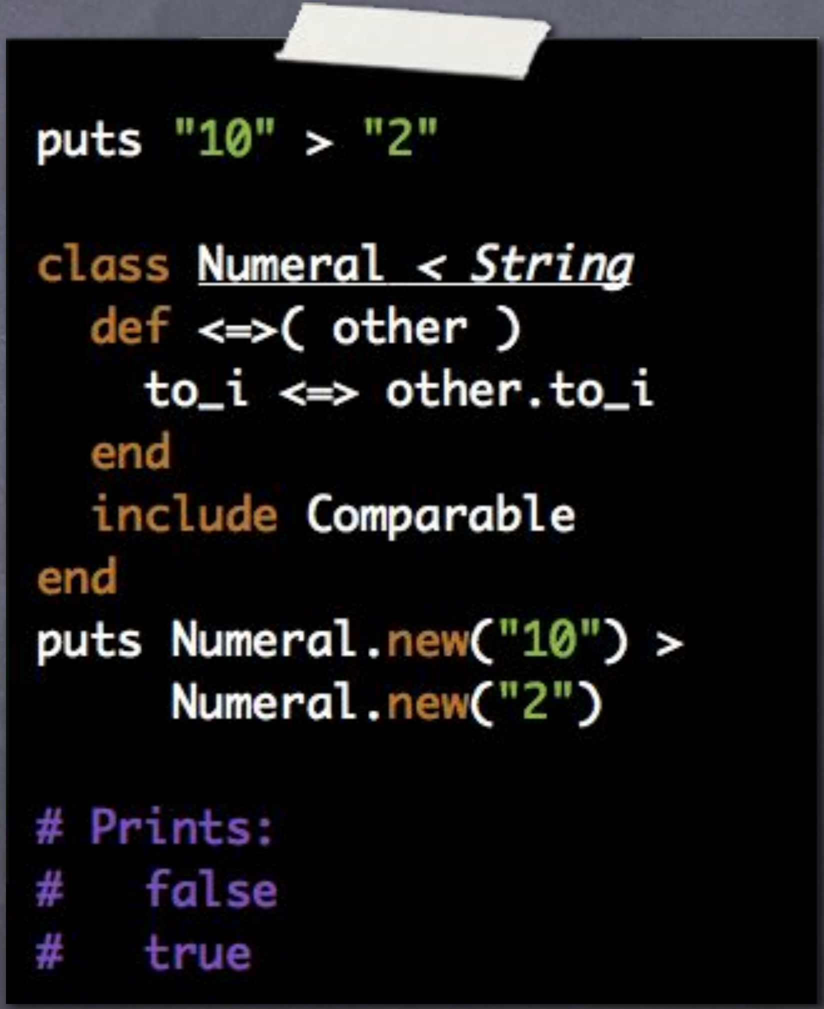
You can’t win with multiple inheritance

- Multiple inheritance allows a class to inherit from more than one parent
  - The good: Makes modeling complex object trees easier
  - The bad: The diamond inheritance problem
- You can’t please both sides

Ruby uses single inheritance...

…and "Mix-ins"

- Similar to Java's interfaces, plus implementation
- No limit to how many you use
- The benefits of multiple inheritance, without the minuses



```ruby
puts "10" > "2"

class Numeral < String
  def <=>( other )
    to_i <=> other.to_i
  end
  include Comparable
end
puts Numeral.new("10") >
     Numeral.new("2")

# Prints:
#   false
#   true
```

7. Uses “Mix-ins”

Makes too much sense!

6. No loops

The well-known loops

Most languages	Ruby

while { ... }
until { ... }
	while ... end
	until ... end

do { ... } while
do { ... } until
	~~begin ... end while~~
	~~begin ... end until~~

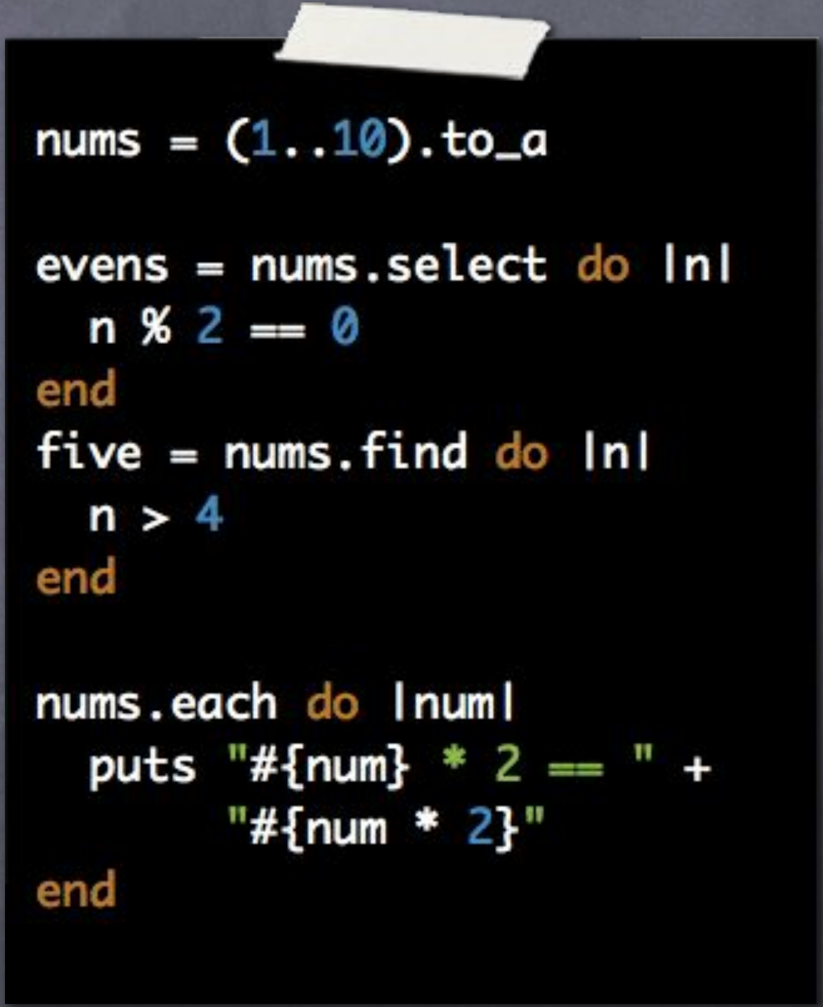
foreach { ... }	each do ... end
for(... ; ... ; ...) { ... }	

Aren’t loops proven to work by now?

- “N + 1” errors
- foreach { ... } is conceptually backwards
    - Objects should manage their own traversal

Iterators

- Objects manage their own traversal
- No more "N + 1" errors
- Code blocks still allow customizing behavior



```ruby
nums = (1..10).to_a

evens = nums.select do |n|
  n % 2 == 0
end
five = nums.find do |n|
  n > 4
end

nums.each do |num|
  puts "#{num} * 2 == " +
       "#{num * 2}"
end
```

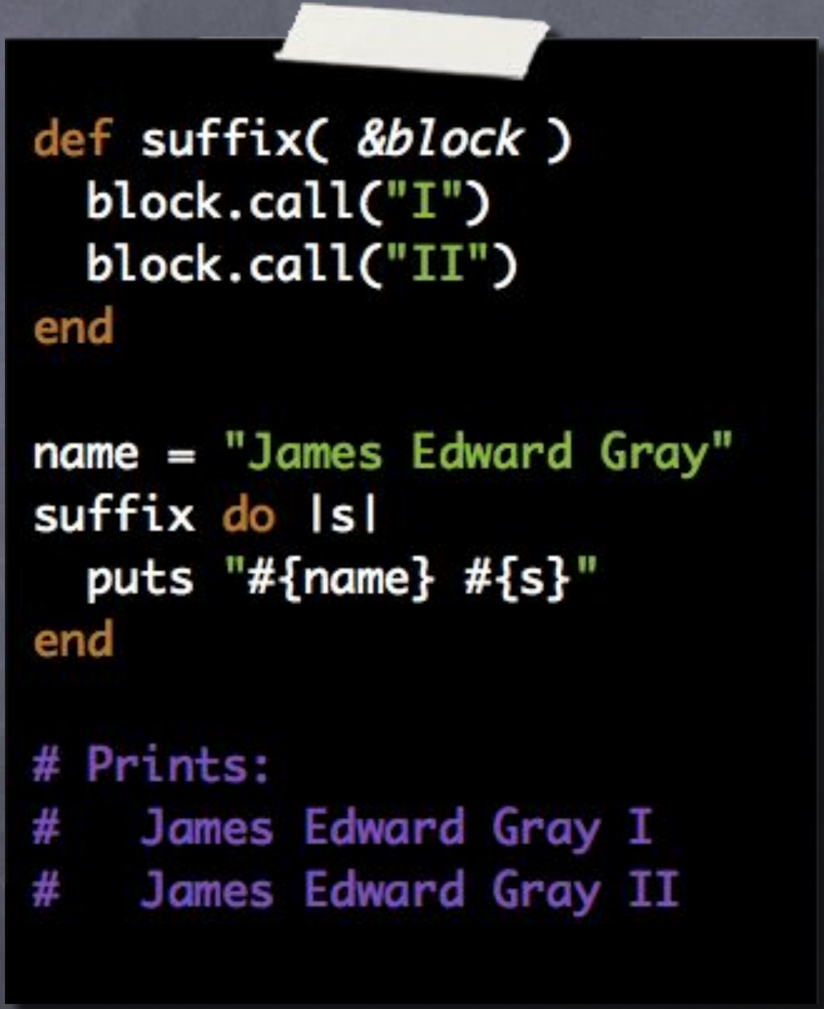
- No loops

Rebellio us!

5. Code blocks everywhere

What is a code block?

- Any method can accept a block
- Blocks can be called immediately or stored for later use
- Blocks are closures



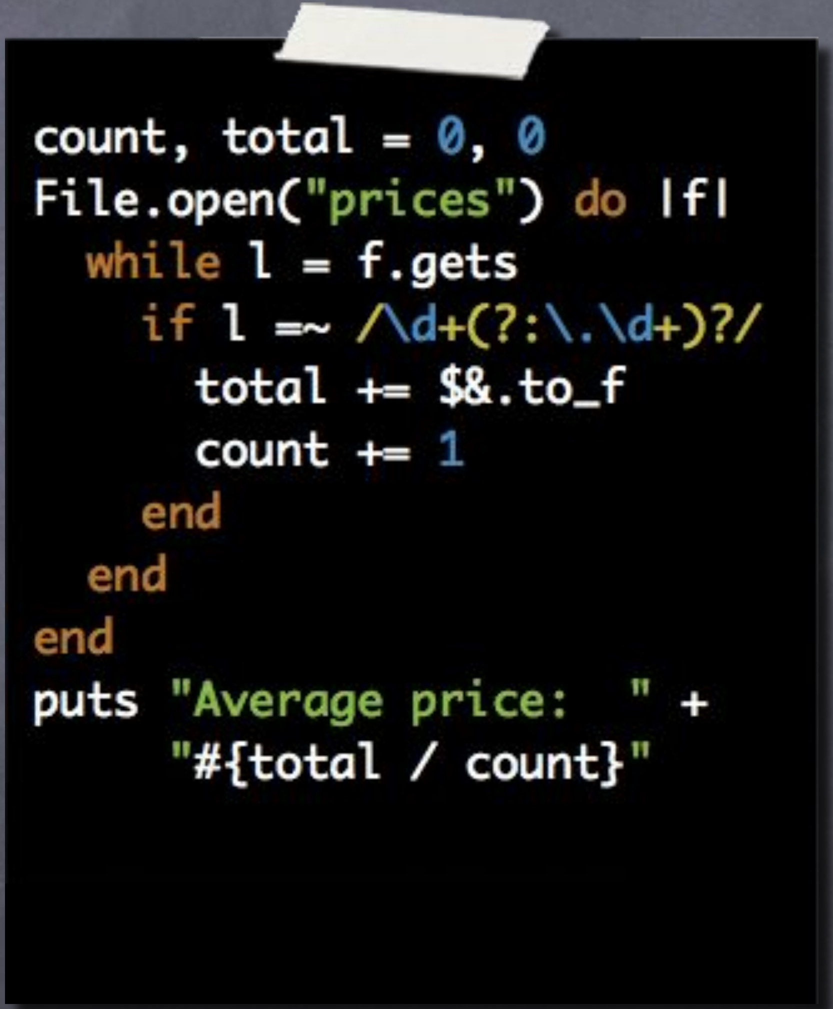
```ruby
def suffix( &block )
  block.call("I")
  block.call("II")
end

name = "James Edward Gray"
suffix do |s|
  puts "#{name} #{s}"
end

# Prints:
#   James Edward Gray I
#   James Edward Gray II
```

What are they for?

- Blocks can allow your code to react in according to user code
- Blocks are a great way to pass around behavior
- Blocks are ideal for transactions



```
count, total = 0, 0
File.open("prices") do |f|
  while l = f.gets
    if l =~ /^\d+(?:\.\d+)?/
      total += $&.to_f
      count += 1
    end
  end
end
puts "Average price:  " +
     "#{total / count}"
```

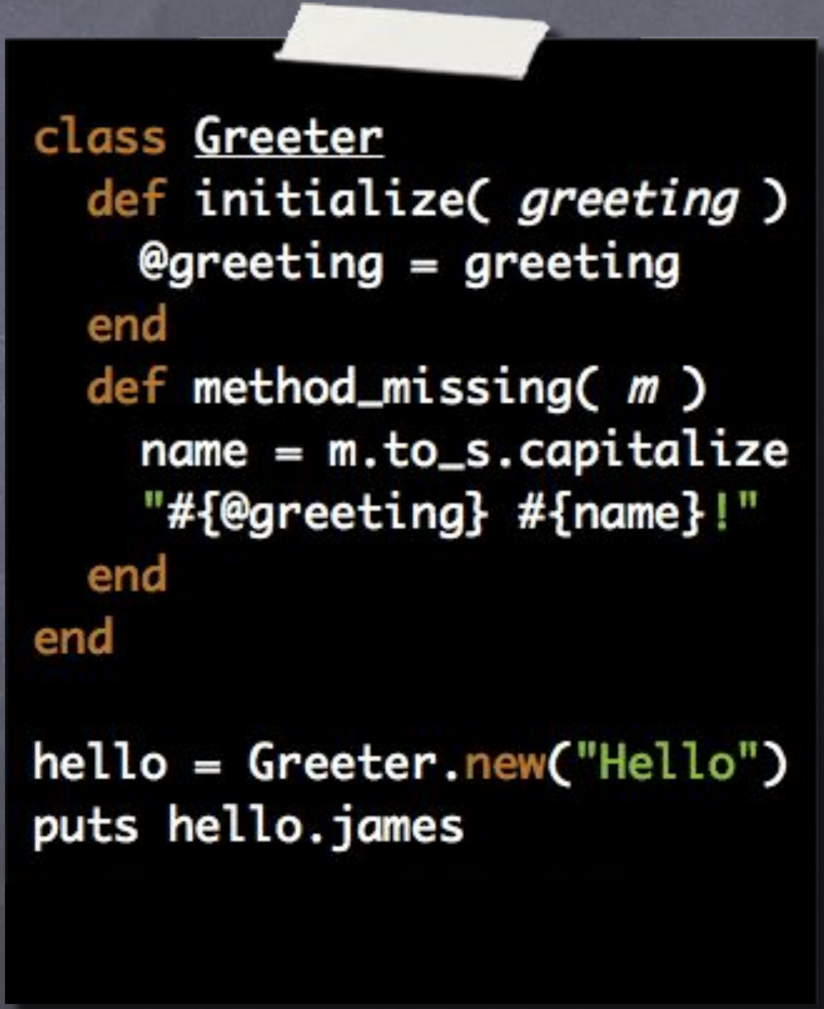
5. Code blocks everywhere

Too powerful!

4. Wide open, even at runtime

Dynamic tools

- Strong reflection
- eval()
    - instance_eval()
    - class_eval() and module_eval()
- Hooks for runtime events

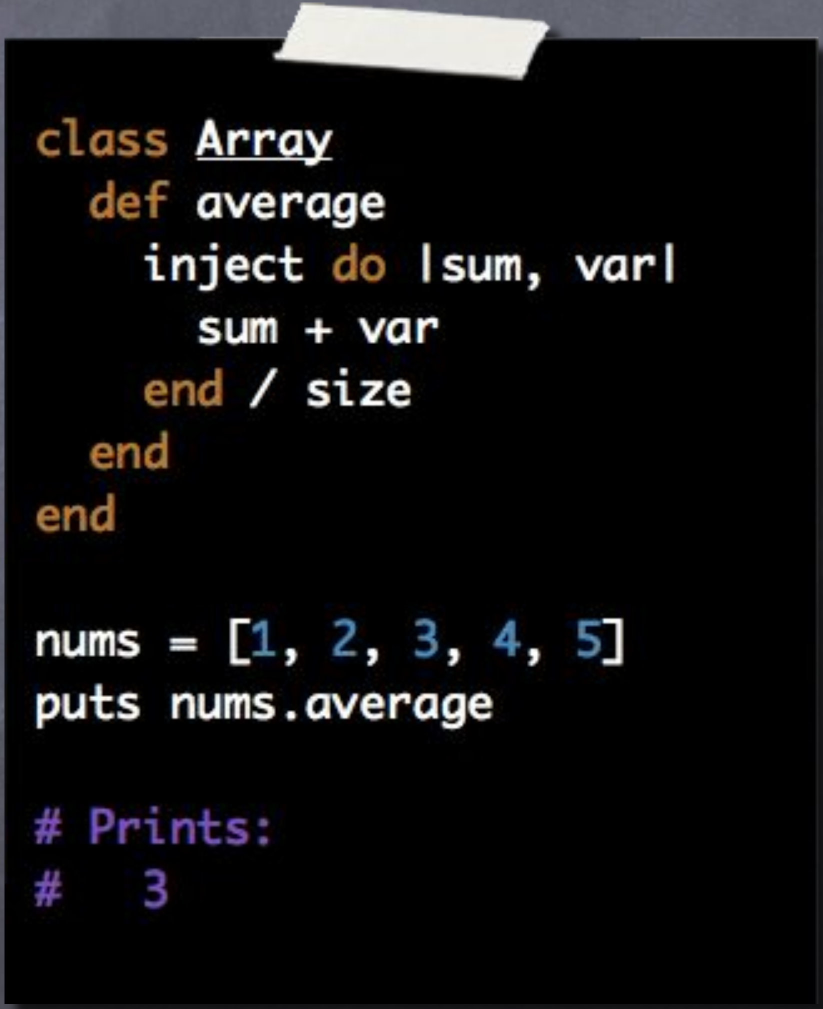


```ruby
class Greeter
  def initialize( greeting )
    @greeting = greeting
  end
  def method_missing( m )
    name = m.to_s.capitalize
    "#{@greeting} #{name}!"
  end
end

hello = Greeter.new("Hello")
puts hello.james
```

Classes are open

- Add methods to a class at any time
  - Even a core class
- Customize individual objects
- Overload operators
- Hook into Ruby's math and conversion operations



```
class Array
  def average
    inject do |sum, var|
      sum + var
    end / size
  end
end

nums = [1, 2, 3, 4, 5]
puts nums.average

# Prints:
#   3
```

4. Wide open, even at runtime

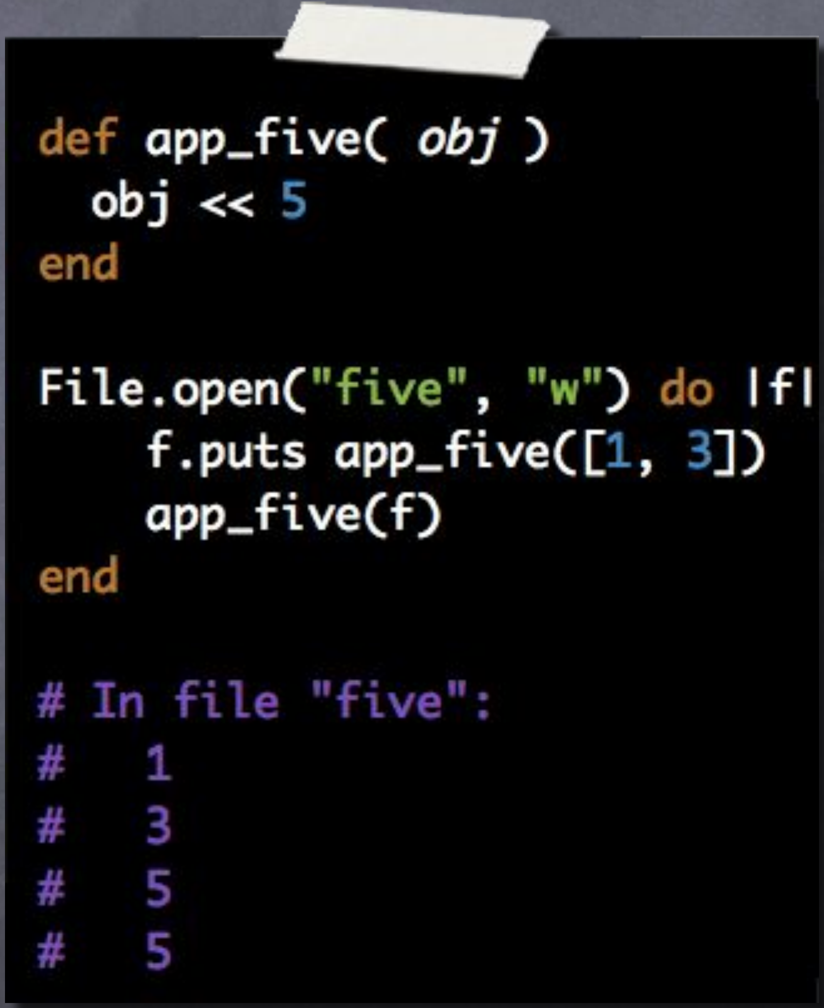
Lawles
s!

3. Ruby gurus are obsessed with ducks

"If it walks like a duck and talks like a duck, it's a duck!"

The "Duck Typing" philosophy

- We define an object by what it can do, not its type
- Most of the time, you shouldn't even check for methods



```
def app_five( obj )
  obj << 5
end

File.open("five", "w") do |f|
    f.puts app_five([1, 3])
    app_five(f)
end

# In file "five":
#   1
#   3
#   5
#   5
```

3. Ruby gurus are obsessed with ducks

Too strange!

2. Includes too many great toys

96 standard libraries

Read/Write	CSV	XML	YAML
Talk to	Email	FTP	Web
Serve	Code	Servlets	XML-RPC
Work with	Math	Templates	Threads
Tools for	Debugging	Docs	Testing

2. Includes too many great toys

Too distracting!

1. “It's entirely too fun and productive for most people.”

— Mike Clark