Metaprogramming

Programs as Data

Metaprogramming

Programs that use other programs as data

Examples:

- Compilers
  - Templating and Generics
- Refactoring Tools

Reflective Programming

Programs that use themselves as data

Examples:

- Inspect variables, classes, and methods
- Create new variables, classes, and methods

Ruby

is a "scripting language"

Also:

- Interpreted
- Reflective
- Object-oriented

Ruby

Everything is an object, including classes and methods

Everything inherits from the class *Object*, including classes and methods

Ruby

Symbols are like global *enums*

Used to identify methods and variables

Examples:

- :foo
- :'1'
- :'@foo'

Ruby

Class Variable: `@@var`

Instance Variable: `@var`

An instance's class variables are a class's instance variables

Ruby

```ruby
array.each do |obj|
    ...
end

(1..10).inject(0) {|m,n| m + n}

def foo(arg, &block)
    ...
end

def greet
    @names.each {|n| yield n}
end
```

Ruby

No multiple inheritance; mixins instead

Inherited class variables aren't copied into the new class

Ruby

```ruby
class A
    @@words = []

    def <<(word)
        @@words << word
    end

    def print
        puts @@words.join(' ')
    end
end

class B < A; end
class C < A; end

(v0 = B.new) << 'hello'
(v1 = C.new) << 'world'
```

Object Methods

```
class
send                    instance_exec
extend                  instance_variables
method                  method_missing
methods
responds_to?
```

Object Methods

```
instance_variable_defined?(symbol)
instance_variable_get(symbol)
instance_variable_set(symbol, object)
```

`symbol` **looks like** `:'@name'`

"Sets the instance variable names by *symbol* to *object*, thereby frustrating the efforts of the class's author to attempt to provide proper encapsulation." - **Ruby Documentation**

Module Methods

```
module_eval                  instance_method
class_eval                   instance_methods
class_variable_defined?      method_defined?
included
```

Classes

`inherited` callback

How do you access class variables?

```
Klass.instance_variable_get
Klass.instance_variable_set
```

Anonymous Classes

```
klass = Class.new do
   method definitions
end
```

Use Cases: `method_missing`

debugging

dynamic function definition

error reporting

proxy objects

method families

```ruby
def method_missing(meth, *args, &block)
  if meth.to_s =~ /^find_by_(.+)$/
    run_find_by_method($1, *args, &block)
  else
    super
  end
end
```

Use Cases: `define_method`

reduce code duplication

form closures

code instrumentation

```
log  = Logger.new
meth = obj.method(name)

obj.define_method(name) do |*args|
    log.info("Called #{name}")
    meth.call(*args)
end
```

Case Study: RLTK::AST

```
def AST.value(name, type)
     if type.is_a?(Array) and type.length == 1
          t = type.first

     elsif type.is_a?(Class)
          t = type

     else
          raise Exception
     end

     if RLTK::subclass_of?(t, ASTNode)
          raise Exception
     end

     @value_names << name
     self.define_accessor(name, type)
end
```

Case Study: RLTK::AST

```
def self.define_accessor(name, type, set_parent = false)
     ivar_name = ('@' + name.to_s).to_sym

     define_method(name) do
          self.instance_variable_get(ivar_name)
     end

     define_method((name.to_s + '=').to_sym) do |value|
          if value.is_a?(type) or value == nil
               self.instance_variable_set(ivar_name, value)

               value.parent = self if value and set_parent
          else
               raise TypeMismatch.new(type, value.class)
          end
     end
end
```

Case Study: RLTK::AST

```
def values
     self.class.value_names.map { |name| self.send(name) }
end

def values=(values)
     if values.length != self.class.value_names.length
          raise Exception, 'Wrong number of values specified.'
     end

     self.class.value_names.each_with_index do |name, i|
          self.send((name.to_s + '=').to_sym, values[i])
     end
end
```

Case Study: `RLTK::Parser`

Motivation:

- Subclass `RLTK::Parser` to create new parsers
- Define any number of parsers
- Instantiate any number of parsers

Problem: Superclass class variables are shared between subclasses

Case Study: `RLTK::Parser`

```
def Parser.inherited(klass)
     klass.class_exec do
          @core = ParserCore.new

          def self.core
               @core
          end

          def self.method_missing(method, *args, &proc)
               @core.send(method, *args, &proc)
          end

          def self.parse(tokens, opts = {})
               opts[:env] ||= self::Environment.new

               @core.parse(tokens, opts)
          end

          def initialize
               @env = self.class::Environment.new
          end

          def env
               @env
          end

          def parse(tokens, opts = {})
               self.class.core.parse(tokens, {:env => @env}.update(opts))
          end
     end
end
```

Questions?