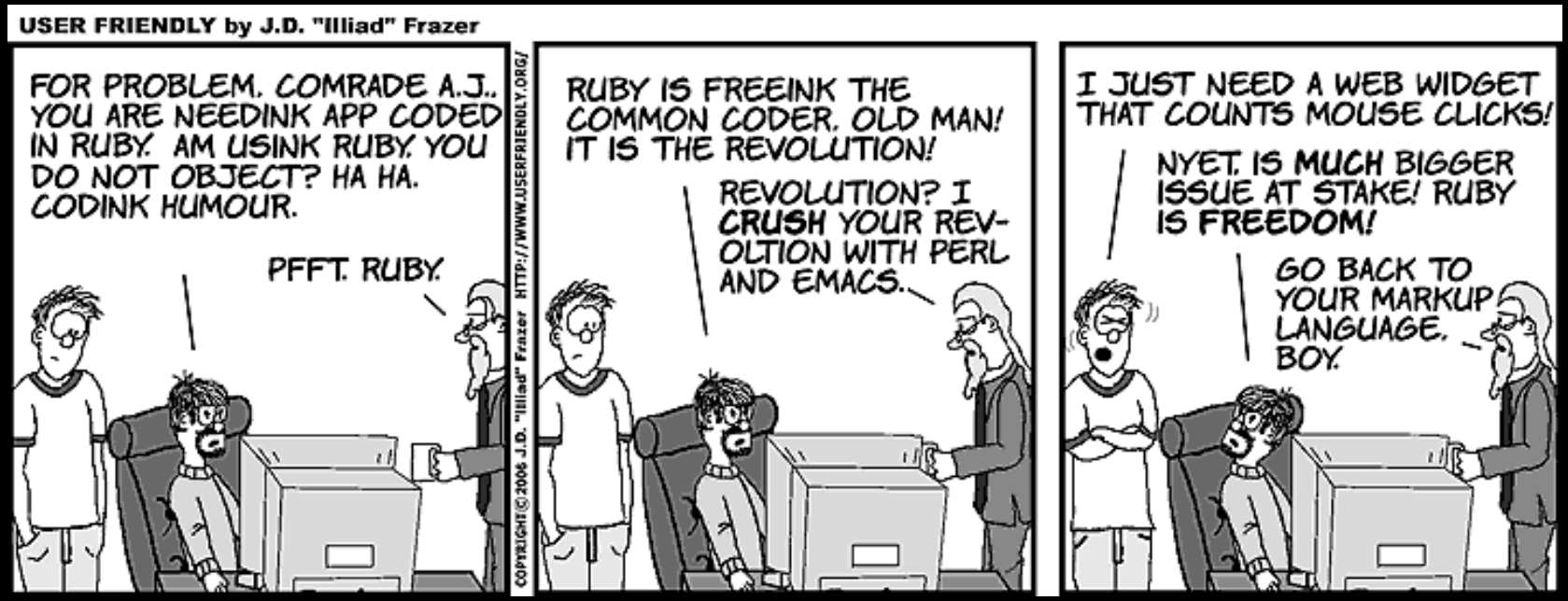


http://ars.userfriendly.org/cartoons/?id=20080627

CS 152: *Programming Language Paradigms*

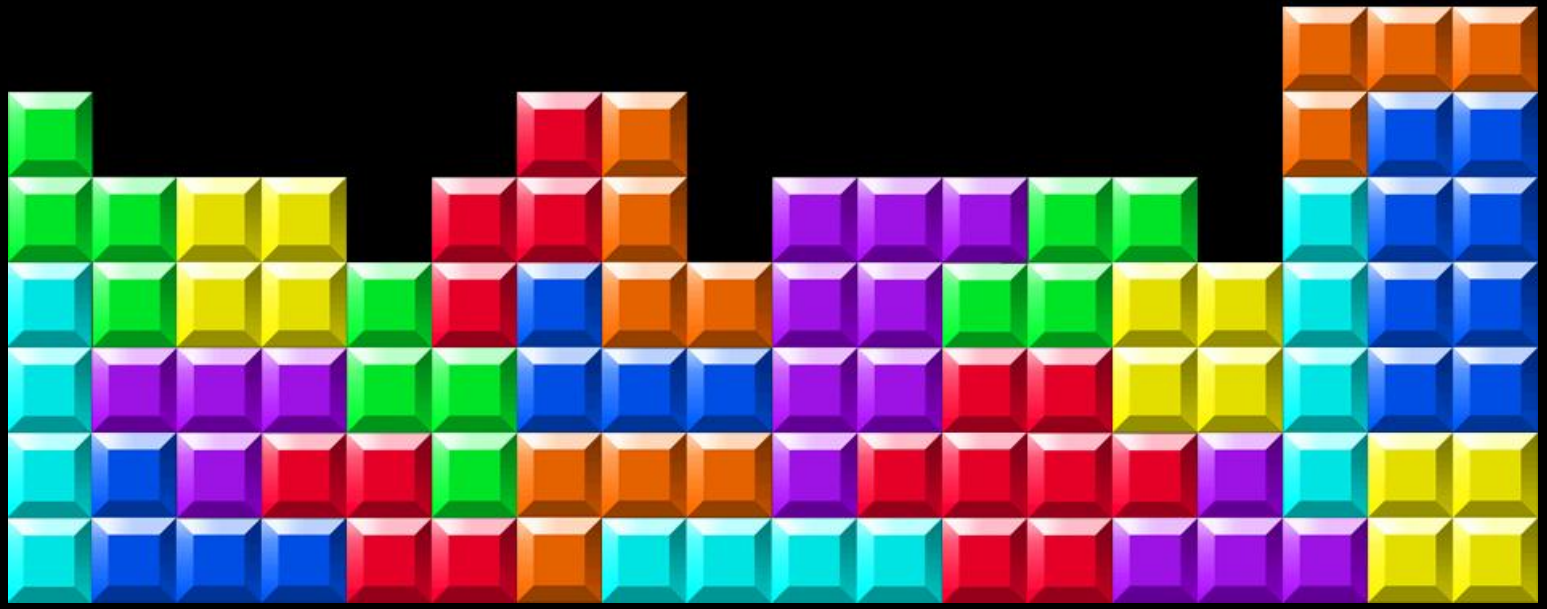
Blocks and Message Passing

Prof. Tom Austin
San José State University

Smalltalk influences on Ruby

- Everything is an object
- Blocks
- Message passing

Blocks



File I/O

```
file = File.open(
           'temp.txt', 'r')
file.each_line do |line|
  puts line
end
file.close
```

File I/O with blocks

```
File.open('file','r') do |f|
  f.each_line { |ln| puts ln }
end
```

Creating custom blocks

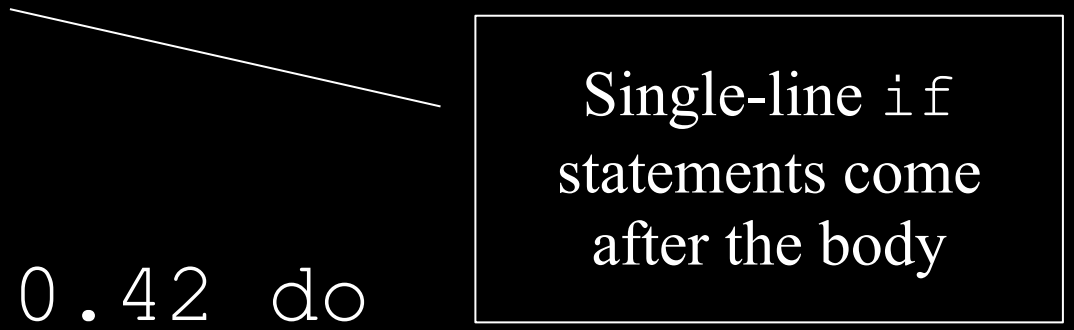
Ruby methods can accept blocks to

- create custom **control structures**
- eliminate **boilerplate** code

with_prob

- Accepts:
  - a probability (between 0 and 1)
  - a block
- Executes block probabilistically
  - `with_prob 0 { puts "hi" }`
  - `with_prob 1 { puts "hi" }`
  - `with_prob 0.5 { puts "hi" }`

```
def with_prob (prob)
  yield if (Random.rand < prob)
end

with_prob 0.42 do
  puts "There is a 42% chance "
    + "that this code will print"
end
```

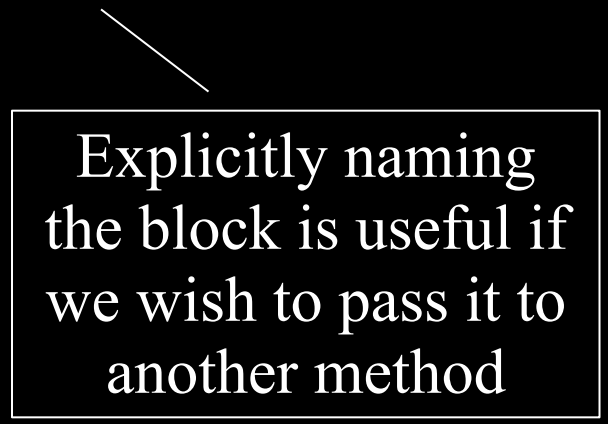


```
def with_prob (prob, &blk)
  blk.call if (Random.rand < prob)
end

with_prob 0.42 do
  puts "There is a 42% chance "
    + "that this code will print"
end
```

Note that there is no '&' here.

We can explicitly name the block of code

```
def with_prob (prob, &blk)
  blk.call if (Random.rand < prob)
end

def half_the_time (&block)
  with_prob(0.5, &block)
end
```



Conversion table example

```
def conversion_chart(from_units, to_units, values)
  puts "#{from_units}\t#{to_units}"
  left_line = right_line = ""
  from_units.length.times { left_line += '-' }
  to_units.length.times { right_line += '-' }
  puts "#{left_line}\t#{right_line}"
  for val in values
    converted = yield val
    puts "#{val}\t#{converted}"
  end
  puts
end
```

```
celsius_temps = [0,10,20,
  30,40,50,60,
  70,80,90,100]

conversion_chart("C", "F",
  celsius_temps) do |cel|
    cel * 9 / 5 + 32
end
```

C	F
0	32
10	50
20	68
30	86
40	104
50	122
60	140
70	158
80	176
90	194
100	212

```
conversion_chart("Km",
   "Miles", (1..10)) do |km|
     mile = 0.621371 * km
end
```

Km	Miles
1	0.621371
2	1.242742
3	1.8641130000000001
4	2.485484
5	3.106855
6	3.7282260000000003
7	4.349597
8	4.970968
9	5.592339
10	6.21371

Blocks are *closures*, though there are some differences between JavaScript functions and Ruby blocks.

Let's see how the two compare…

Writing `withProb` in JavaScript

```
function withProb(prob, f) {
  if (Math.random() < prob) {
    return f();
  }
}
```

JavaScript uses callbacks rather than blocks

What is the difference?

```
def coin_flip
  with_prob 0.5
  do
    return "H"
  end
  return "T"
end
```

```
function coinFlip() {
  withProb(0.5,
    () => {
      return "H";
    });
  return "T";
}
```

Demo: Running coin-flip.rb and coinFlip.js 100 times

Singleton classes

WARNING!!!

Singleton classes have NOTHING to do with singleton objects.

JavaScript & prototypes

```
function Employee(name, salary) {
  this.name = name;
  this.salary = salary;
}
var a = new Employee("Alice", 75000);
var b = new Employee("Bob", 50000);

b.signingBonus = 2000;
console.log(a.signingBonus);
console.log(b.signingBonus);
```

Can we do the same thing in Ruby?

In Ruby, every object has a special *singleton class*.

This class holds methods unique to that object.

Singleton Class Example

Employee class

```
class Employee
  attr_accessor :name, :ssid, :salary
  def initialize(name, ssid, salary)
    @name = name
    @ssid = ssid
    @salary = salary
  end
  def to_s
    @name
  end
end
```

```
alice = Employee.new("Alice Alley",
                     1234, 75000)
bob = Employee.new("Robert Tables",
                   5678, 50000)
# Adding to Bob's singleton class
class << bob
  def signing_bonus
    2000
  end
end

puts(bob.signing_bonus)
puts(alice.signing_bonus) # ERROR!
```

Why would anyone want this?

- Remember – classes are also objects
- Also remember – all objects have a unique singleton class
- So… every class has its own singleton class
- Used for "static" methods

Modified Employee constructor

```
def initialize(name, ssid, sal)
   @name = name
   @ssid = ssid
   @salary = sal
   Employee.add self
end
```

Adding "static" methods to Employee

```
class Employee
  class << self
    def add(emp)
      puts "Adding #{emp}"
      @employees = Hash.new unless @employees
      @employees[emp.name] = emp
    end
    def get_emp_by_name name
      @employees[name]
    end
  end
```

Using "static" method

```
bob = Employee.new("R. Tables", 5678, 50000)

# Looks up the employee by name.
b = Employee.get_emp_by_name "R. Tables"

# Prints out 50000
p b.salary
```


Message Passing

Message passing (object interaction)

- Sender sends
  - message: method name
  - data: method parameters
- Receiver
  - processes the message
  - (optionally) returns **data**

If receiver doesn't understand message?

```
irb> "hello".foo
NoMethodError: undefined
method `foo' for
"hello":String
    from (irb):2
    from /usr/bin/irb:12:in
`<main>'
irb>
```

`method_missing`

- You can override this behavior
  - Smalltalk: `doesNotUnderstand`
  - Ruby: `method_missing`
- This method is called when an unknown method is invoked

```
class Person
  attr_accessor :name
  def initialize(name)
    @name = name
  end
  def method_missing(m)
    puts "Didn't understand #{m}"
  end
end
```

Example invoking `method_missing`

```
alice = Person.new "Alice"


# Prints:
#   "Didn't understand foo"
alice.foo
```

```
bob = Person.new "Robert"
class << bob
  def method_missing m
   phrase = m.to_s.sub(
       /say_(.*)/, '\1')
    puts phrase
  end
end
```

Using Bob's `method_missing`

```
bob.say_goodnight_gracie

bob.say_hey
```

Rails ActiveRecord example

```
Person.find_by(first_name: 'David')

Person.find_by_first_name "John"

Person.find_by_first_name_and_last_name \
  "John", "Doe"
```

Record example

```
class Record
  def initialize(fields)
    @fields = fields
  end
  def method_missing(m, *args)
    mn = m.to_s
    @fields[mn.chop] = args[0] if (mn.end_with?("="))
    @fields[mn]
  end
end
r = Record.new ({ 'fname' => 'Rick',
                  'lname' => 'Grimes',
                  'profession' => 'Police Officer' })
puts r.profession
r.profession = 'Zombie hunter'
puts r.profession
```

Lab: Ruby Metaprogramming

Today's lab explores blocks and `method_missing`.

Download tree.rb from the course website. The lab description is available in both Canvas and the course website.