

Ruby Methods, Procs, etc.

CSCI400

21 September 2017

Color Key

- [Clickable URL link](#)
- **Write down an answer to this for class participation**
- Just a comment – don't confuse with yellow

Brief Refactor Exercise (1/2)

```
def fib(limit)
  yield 1
  yield 1
  a = 1
  b = 1
  while (i < limit)
    t = a
    a = a + b
    b = t
    yield a
    i = i + 1
  end
end
```

Brief Refactor Exercise (2/2)

```
def fib(limit)
  a, b = 0, 1
  (1..limit).each do
    a, b = b, a + b
    yield a
  end
end
```

11 lines → 4 lines (of actual logic/content)

Topics

- Method arguments
 - Some new, some review
- Proc

Method Arguments

Overview

- Default value
 - Recall: `Hangman.new` vs. `Hangman.new "myWords.txt"`
- Variable number of arguments
 - `def readwrite(*syms)`
- Pass arguments in `Hash`
- Block as function argument
 - When method has `yield`

Some of this section is review, and some is new

Default Value (1)

```
def title(name, len=3)
  name[0, len]
end
```

```
puts title("Mr. Brandon McCartney")
puts title("Mrs. Doubtfire", 4)
```


Default Value (2)

```
# can use expression in default value  
def shift(x, dx=x/100.0)  
  x + dx  
end
```

```
puts shift(5)  
puts shift(10, 1)
```

Variable Argument Count

- Similar to * (*splat*)
 - `a, *b = 1, 2, 3`
- * before param in *function definition*
 - Param → array of 0 or more args
- * before array param in *function call*
 - Param array → separate arguments

Variable Arg Examples

```
def limitedSum(max, *rest)
  total = rest.sum
  if total <= max
    total
  else
    max
  end
end
```

```
limitedSum(20, 1, 4, 5)
limitedSum(20, 10, 20, 30)

data = [1, 4, 5]
limitedSum(20, *data)
```

Hashes for Arguments

- Argument order
 - Could break client's code if changed
- Solution: send hash

Hashes for Arguments

```
def http(config)
  if config.key?(:tls)
    puts "Using HTTPS"
  end
  if config.key?(:basic_auth)
    puts "Using HTTP with BasicAuth"
  end
end
```

Hashes for Arguments

```
def http(config)
  if config.key?(:tls)
    puts "Using HTTPS"
  end
  if config.key?(:basic_auth)
    puts "Using HTTP with BasicAuth"
  end
end
```

```
httpConfig = {
  :tls => true,
  :basic_auth => { "user" => "pass" }
}
http(httpConfig)
```

Blocks

- Blocks are syntactic structures
 - *Not objects*
- Identified by...
 - Curly braces: { ... }
 - do/end keywords: do end
- Can create *objects that represent blocks*
 - Okay, but why?

yield

- `yield` statement
 - Gives control to *user-specified block*

yield Example

```
def fib(limit)
  a, b = 0, 1
  (1..limit).each do
    a, b = b, a + b
    yield a
  end
end
```

```
fib(10) { |x| puts "Fibonacci number: #{x}" }
```

Objects that Represent Blocks

- `Proc`
 - Has block-like behavior
 - But can pass *multiple procs* into function
- Lambdas^{*}
 - Generally: *anonymous functions*
 - Ruby: lambdas are `Proc` instances
- `Blocks vs. Procs vs. Lambdas`

*We'll do lambdas in Haskell

Implicit Block Argument

```
def fib(limit)
  a, b = 0, 1
  (1..limit).each do
    a, b = b, a + b
    yield a
  end
end
```

```
fib(10) { |x| puts x }
```

Explicit Block Argument

```
# block arg must be last argument  
# also, notice the `&` for the block arg  
def fib(limit, &block)  
  a, b = 0, 1  
  (1..limit).each do  
    a, b = b, a + b  
    block.call(a) # could stil use `yield`  
  end  
end
```

```
fib(10) { |x| puts x }
```

Proc Argument

```
# notice no `&` this time -- `Proc` is just an object
def fib(limit, proc)
  a, b = 0, 1
  (1..limit).each do
    a, b = b, a + b
    proc.call(a)
  end
end
```

```
p = Proc.new { |x| puts x }
fib(10, p)
```

Block vs. Proc

What's the difference?

- Block: *one-time use*
- Proc: *reusable*
 - Pass to multiple functions
 - Provide with library

Block and Proc Uses

- Goal: Encrypt file byte-by-byte
- while more data to read...
 - ① read a byte
 - ② encrypt byte (with block or Proc)
 - ③ write byte
- encrypt step is *up to user*

Real-World Example (1)

```
class RailsAppTest < ActiveSupport::TestCase
  test "start web server" do
    # sequence of instructions + assert
  end
end
```

`test` prints formatted results

Real-World Example (2)

```
class Order < ActiveRecord::Base
  before_save
    :normalize_card_number,
    if: Proc.new { |order| order.paid_with_card? }
    # if: proc is a hash, same as { :if => proc }
end
```

Proc Exercise (1/2)

- Write function `powers(max, proc1, proc2)`
 - Applies `proc1` and `proc2` to integers `[1, max]`
- See next slide for example output

Proc Exercise (2/2)

```
$ ruby procExample.rb
```

```
Calling powers -- square and cube
```

```
1 1 1
```

```
2 4 8
```

```
3 9 27
```

```
4 16 64
```

```
5 25 125
```

```
Calling powers -- square and fourth power
```

```
1 1 1
```

```
2 4 16
```

```
3 9 81
```

```
4 16 256
```

```
5 25 625
```