# More Ruby Tools: Blocks, Constants, Modules

Ruby Fundamentals

# Overview

- Blocks, procs and lambdas
- Constants
- Modules

# Blocks

```
[1, 2, 3].each do
  puts "This is Serenity, please respond"
end

ships = Spaceship.all
ships.each { |ship| puts ship.name }
```

# Invoking Blocks

```
class Spaceship
  def debug_only
    return nil unless @debug
    return nil unless block_given?
    puts "Running debug code..."
    yield
  end
end

ship.debug_only

ship.debug_only do
  puts "This is debug output"
end
```

# Invoking Blocks

```
class Spaceship
  def debug_only
    return nil unless @debug
    return nil unless block_given?
    puts "Running debug code..."
    yield @debug_attrs
  end
end

ship.debug_only do |attrs|
  puts "Debug attr values: #{attrs.inspect}"
end
```

# Block Arguments

- Default values
- Keyword arguments
- Array arguments (with the splat)

# Block Local Variables

- Block arguments shadow same name variables in outer scope
- Variables defined in block body don’t shadow outer scope
- Block local variables solve this problem

# Blocks Are Closures

# Other Uses for Blocks

```
def with_timing
  start = Time.now
  if block_given?
    yield
    puts "Time taken: #{Time.now - start} seconds"
  end
end

def run_operation_1
  sleep(1)
end
def run_operation_2; end

with_timing do
  run_operation_1
  run_operation_2
end

# Output: Time taken: 1.000057 seconds
```

# Other Uses for Blocks

```
class Database
  def transaction
    start_transaction
    begin
      yield
    rescue DBError => e
      rollback_transaction
      log_error e.message
      return
    end
    commit_transaction
  end
end

DB.transaction do
  # update multiple records
end
```

# Block Limitations

- Can only pass one block into a method
- Blocks can't be passed around between methods
- Passing the same block to several methods isn't DRY

# From Block to Proc

```
p = Proc.new {|bla| puts "I'm a proc that says #{bla}!" }
```

```
p = proc {|bla| puts "I'm a proc that says #{bla}!" }
```

```
p.call "yay!"
p.yield "wow!"
p.("nothing")
p["hello"]
```

# Lambdas

```
lmb = lambda {|bla| "I'm also a proc, and I say #{bla}" }

also_lmb = ->(bla) { "I'm also a proc, and I say #{bla}" }
```

# Differences between Procs and Lambdas

- Procs are like blocks, lambdas are like anonymous methods
- Lambdas are strict about their arguments
- *return* and *break* behave differently in procs and lambdas
- However, *next* behaves the same

# Differences in Argument Handling

- **Lambdas: too many or too few arguments cause an exception**
- **Procs: extra arguments discarded, missing arguments set to *nil***

# Differences in *return* and *break* Handling

- Procs: *return* is executed in the scope where the block was defined
- Procs: *break* isn't allowed outside a loop
- Lambdas: *break* and *return* both return control to the caller

# Some Things You Can Do

```
proc {|a, b| }.arity      #=> 2
proc {|a, *b, c| }.arity  #=> -3


weekend = proc {|time| time.saturday? || time.sunday?}
weekday = proc {|time| time.wday < 6 }

case Time.now
when weekend then puts "Wake up at 8:00"
when weekday then puts "Wake up at 7:00"
else puts "No wake up calls outside of time"
end
```

# One More Trick: Symbol to Proc Conversion

```
def debug_only(param = nil, &block)
  puts "Param class: #{param.class}"
  puts "Block class: #{block.class}" if block_given?
end

debug_only(p) # param == p

debug_only(&p) # param == nil, block == p
```

# Constants

```
MAX_SPEED = 1000

Spaceship
```

# Modules

```ruby
module SpaceStuff
end


module API
  def self.hatch_list
    # retrieve hatch list
  end
end

hatches = API.hatch_list


module SpaceStuff
  class Spaceship
  end
end

ship = SpaceStuff::Spaceship.new
```

# Modules

```ruby
module SpaceStuff
  module API
    def self.hatch_list
      # retrieve hatch list
    end
  end
end

hatches = SpaceStuff::API.hatch_list
```

# Mixins

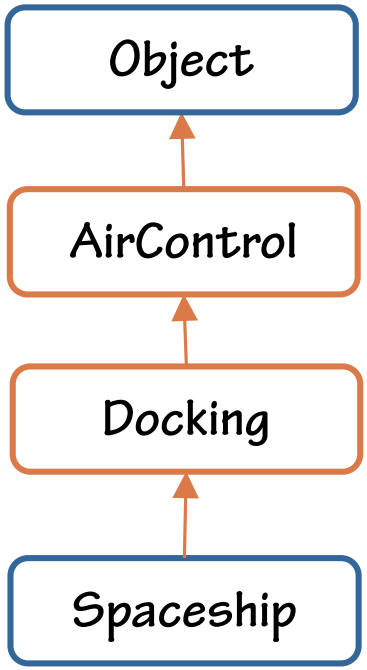
```ruby
module AirControl
  # air pumping, maintenance and regeneration
  def measure_oxygen
    # ...
  end
end

class Spaceship
  include AirControl
  # ...
end

ship = Spaceship.new
ship.measure_oxygen
```

# Mixins

```ruby
class Spaceship
  include AirControl
  include Docking
  # ...
end
```



# Mixing in Class Methods

```
module Docking
  def get_docking_params
    # returns params common to all spaceships
  end
end

class Spaceship
  extend Docking
end

Spaceship.get_docking_params
```

# Mixing in Class Methods

```
module Docking
  module ClassMethods
    def get_docking_params
      # returns params common to all spaceships
    end
  end

  def dock
    # ...
  end
end

class Spaceship
 include Docking
 extend Docking::ClassMethods
end
```

# Mixing in Class Methods

```
module Docking
  module ClassMethods
    def get_docking_params
      # returns params common to all spaceships
    end
  end

  def self.included(base)
    base.extend(ClassMethods)
  end

  def dock
    # ...
  end
end

class Spaceship
  include Docking
end
```

# Instance Variables in Modules

```
module AirControl
  attr_accessor :oxygen_level

  def measure_oxygen
    # ...
    self.oxygen_level = measured_value
  end

  def start_pump
    @pump_status = :started
  end
end
```

# Summary

- Defining and using blocks
- Procs and lambdas
- Constants
- Modules as namespaces and mixins