
RUBY
BITS
EXPRESSIONS
PRESS START

ICONS

Bad Code

Worse Code

Good Code

Awesome Code

RUBY BITS

UNLESS

```ruby
if ! tweets.empty?
  puts "Timeline:"
  puts tweets
end
```

Unless is more intuitive

```ruby
unless tweets.empty?
  puts "Timeline:"
  puts tweets
end
```

RUBY BITS

UNLESS WITH ELSE

```ruby
unless tweets.empty?
  puts "Timeline:"
  puts tweets
else
  puts "No tweets found - better follow some people!"
end
```

Unless with else is confusing

```ruby
if tweets.empty?
  puts "No tweets found - better follow some people!"
else
  puts "Timeline:"
  puts tweets
end
```

RUBY BITS

NIL IS FALSE-Y

```
if attachment.file_path != nil
  attachment.post
end
```

nil treated as false

```
if attachment.file_path
  attachment.post
end
```

RUBY BITS

ONLY NIL IS FALSE-Y

`""` treated as "true"!

`0` treated as "true"!

`[]` treated as "true"!

```
unless name.length
  warn "User name required"
end
```

will never be false!

RUBY BITS

INLINE CONDITIONALS

```ruby
if password.length < 8
  fail "Password too short"
end
unless username
  fail "No user name set"
end
```

try inline if/unless

```ruby
fail "Password too short" if password.length < 8
fail "No user name set" unless username
```

RUBY BITS

SHORT-CIRCUIT - "AND"

```
if user
  if user.signed_in?
    #...
  end
end
```

use && instead

```
if user && user.signed_in?
 #...
end
```

if user is nil
second half never runs

RUBY BITS

SHORT-CIRCUIT ASSIGNMENT

```
result = nil || 1
```
···▶ 1

```
result =   1 || nil
```
···▶ 1

```
result =   1 || 2
```
···▶ 1

RUBY BITS

DEFAULT VALUES - "OR"

```
tweets = timeline.tweets
tweets = [] unless tweets
```

if nil, default to empty array

```
tweets = timeline.tweets || []
```

RUBY BITS

SHORT-CIRCUIT EVALUATION

```
def sign_in
  current_session || sign_user_in
end
```

not evaluated unless
current session is nil

but consider whether
if/then would be more legible!

RUBY BITS

CONDITIONAL ASSIGNMENT

```
i_was_set = 1
i_was_set ||= 2

puts i_was_set
```

···▶ 1

assigns IF there's no existing value

```
i_was_not_set ||= 2

puts i_was_not_set
```

···▶ 2

CONDITIONAL ASSIGNMENT

```
options[:country] = 'us' if options[:country].nil?
options[:privacy] = true if options[:privacy].nil?
options[:geotag]  = true if options[:geotag].nil?
```

use conditional assignment

```
options[:country] ||= 'us'
options[:privacy] ||= true
options[:geotag]  ||= true
```

RUBY BITS

CONDITIONAL RETURN VALUES

```ruby
if list_name
  options[:path] = "/#{user_name}/#{list_name}"
else
  options[:path] = "/#{user_name}"
end
```

assign the value of the if statement

```ruby
options[:path] = if list_name
  "/#{user_name}/#{list_name}"
else
  "/#{user_name}"
end
```

RUBY BITS

CONDITIONAL RETURN VALUES

```ruby
def list_url(user_name, list_name)
  if list_name
    url = "https://twitter.com/#{user_name}/#{list_name}"
  else
    url = "https://twitter.com/#{user_name}"
  end
  url
end
```

CONDITIONAL RETURN VALUES

```ruby
def list_url(user_name, list_name)
  if list_name
    "https://twitter.com/#{user_name}/#{list_name}"
  else
    "https://twitter.com/#{user_name}"
  end
end
```

CASE STATEMENT VALUE

```
client_url = case client_name
  when "web"
    "http://twitter.com"
  when "Facebook"
    "http://www.facebook.com/twitter"
  else
    nil
end
```

CASE - RANGES

```
popularity = case tweet.retweet_count
  when 0..9
    nil
  when 10..99
    "trending"
  else
    "hot"
end
```

CASE - REGEXPS

```
tweet_type = case tweet.status
  when /\A@\w+/
    :mention
  when /\Ad\s+\w+/
    :direct_message
  else
    :public
end
```

CASE - WHEN/THEN

```
tweet_type = case tweet.status
  when /\A@\w+/       then :mention
  when /\Ad\s+\w+/    then :direct_message
  else                     :public
end
```


RUBY
BITS
EXPRESSIONS
PRESS START


RUBY
BITS
METHODS AND CLASSES
PRESS START

OPTIONAL ARGUMENTS

```ruby
def tweet(message, lat, long)
  #...
end

tweet("Practicing Ruby-Fu!", nil, nil)
```

location isn't always used, so let's add defaults

```ruby
def tweet(message, lat = nil, long = nil)
  #...
end

tweet("Practicing Ruby-Fu!")
```

location is now optional

RUBY BITS

NAMED ARGUMENTS - HASH

```ruby
def tweet(message, lat = nil, long = nil, reply_id = nil)
  #...
end
```

long parameter list

```ruby
tweet("Practicing Ruby-Fu!", 28.55, -81.33, 227946)
```

calls to it are hard to read

```ruby
tweet("Practicing Ruby-Fu!", nil, nil, 227946)
```

have to keep placeholders for arguments you're not using

RUBY BITS

HASH ARGUMENTS

```ruby
def tweet(message, options = {})
  status = Status.new
  status.lat = options[:lat]
  status.long = options[:long]
  status.body = message
  status.reply_id = options[:reply_id]
  status.post
end
```

hash argument

reference keys from hash

RUBY BITS

HASH ARGUMENTS

```ruby
def tweet(message, options = {})
```

```ruby
tweet("Practicing Ruby-Fu!",
  :lat => 28.55,
  :long => -81.33,
  :reply_id => 227946
)
```

keys show meaning

all combined into options argument

RUBY BITS

NAMED ARGUMENTS - HASH

Using Ruby 1.9 hash syntax

```
tweet("Practicing Ruby-Fu!",
    lat: 28.55,
    long: -81.33,
    reply_id: 227946
)
```

Repositioning the keys

```
tweet("Practicing Ruby-Fu!",
  reply_id: 227946,
  lat: 28.55,
  long: -81.33
)
```

RUBY
BITS

NAMED ARGUMENTS - HASH

Keys are optional

```
tweet("Practicing Ruby-Fu!",
  reply_id: 227946
)
```

Complete hash is optional

```
tweet("Practicing Ruby-Fu!")
```

RUBY BITS

EXCEPTIONS

```ruby
def get_tweets(list)
  if list.authorized?(@user)
    list.tweets
  else
    []
  end
end
```

"magic" return value

```ruby
tweets = get_tweets(my_list)
if tweets.empty?
  alert "No tweets were found!" +
    "Are you authorized to access this list?"
end
```

can't be sure it's an error

RUBY BITS

EXCEPTIONS

```ruby
def get_tweets(list)
  unless list.authorized?(@user)
    raise AuthorizationException.new
  end
  list.tweets
end
```

raise an Exception instead

```ruby
begin
  tweets = get_tweets(my_list)
rescue AuthorizationException
  warn "You are not authorized to access this list."
end
```

caller KNOWS there's a problem

RUBY BITS

"SPLAT" ARGUMENTS

```ruby
def mention(status, *names)
  tweet("#{names.join(' ')} #{status}")
end
```

RUBY BITS

YOU NEED A CLASS WHEN...

```
user_names = [
  ["Ashton", "Kutcher"],
  ["Wil", "Wheaton"],
  ["Madonna"]
]
user_names.each { |n| puts "#{n[1]}, #{n[0]}" }
```

OUTPUT:

```
Kutcher, Ashton
Wheaton, Wil
, Madonna
```

your users shouldn't have to deal with edge cases

RUBY BITS

YOU NEED A CLASS WHEN...

```ruby
class Name
  def initialize(first, last = nil)
    @first = first
    @last = last
  end
  def format
    [@last, @first].compact.join(', ')
  end
end
```

state!

behavior!

RUBY BITS

YOU NEED A CLASS WHEN...

```
user_names = []
user_names << Name.new('Ashton', 'Kutcher')
user_names << Name.new('Wil', 'Wheaton')
user_names << Name.new('Madonna')
user_names.each { |n| puts n.format }
```

OUTPUT:

```
Kutcher, Ashton
Wheaton, Wil
Madonna
```

◀ edge case handled!

RUBY BITS

OVERSHARING?

```ruby
class Tweet
  attr_accessor :status, :created_at
  def initialize(status)
    @status = status
    @created_at = Time.new
  end
end
```

```ruby
attr_accessor :baz
```

same as

```ruby
def baz=(value)
  @baz = value
end
def baz
  @baz
end
```

```ruby
tweet = Tweet.new("Eating breakfast.")
tweet.created_at = Time.new(2084, 1, 1, 0, 0, 0, "-07:00")
```

..... shouldn't be able to do this!

RUBY BITS

OVERSHARING?

```
class Tweet
  attr_accessor :status
  attr_reader :created_at
  def initialize(status)
    @status = status
    @created_at = Time.new
  end
end
```

doesn't define a setter!

```
tweet = Tweet.new("Eating breakfast.")
tweet.created_at =
  Time.new(2084, 1, 1, 0, 0, 0, "-07:00")
```

▶ undefined method 'created_at='

RUBY BITS

RE-OPENING CLASSES

```
tweet = Tweet.new("Eating lunch.")
puts tweet.to_s
```

```
...▶ #<Tweet:0x000001008c89e8>
```

not so readable...

```
class Tweet
  def to_s
    "#{@status}\n#{@created_at}"
  end
end
tweet = Tweet.new("Eating lunch.")
puts tweet.to_s
```

so just re-open the class and redefine it!

```
...▶ Eating lunch.
     2012-08-02 12:20:02 -0700
```

RUBY BITS

RE-OPENING CLASSES

- You can re-open and change any class.
- Beware! You don't know who relies on the old functionality.
- You should only re-open classes that you yourself own.

SELF

```ruby
class UserList
  attr_accessor :name
  def initialize(name)
    name = name
  end
end
```

this just re-sets the "name" local variable!

```ruby
list = UserList.new('celebrities')
list.name
```

nil

```ruby
class UserList
  attr_accessor :name
  def initialize(name)
    self.name = name
  end
end
```

but this calls "name=" on the current object

```ruby
list = UserList.new('celebrities')
list.name
```

"celebrities"

RUBY BITS


RUBY
BITS
METHODS AND CLASSES
PRESS START


RUBY
BITS
CLASSES
PRESS START

ENCAPSULATION

```
send_tweet("Practicing Ruby-Fu!", 14)
```

```
def send_tweet(status, owner_id)
  retrieve_user(owner_id)
  ...
end
```

should not be responsible for retrieving a user

RUBY BITS

ENCAPSULATION

- Passing around data as strings and numbers breaks encapsulation.
- Places using that data need to know how to handle it.
- Individual changes require updates at various places.

ENCAPSULATION

```ruby
tweet = Tweet.new
tweet.status = "Practicing Ruby-Fu!"
tweet.owner_id = current_user.id

send_tweet(tweet)
```

one parameter!

```ruby
def send_tweet(message)
  message.owner
  ...
end
```

```ruby
class Tweet
  attr_accessor ...

  def owner
    retrieve_user(owner_id)
  end
end
```

RUBY BITS

ENCAPSULATION

- May not be worth the overhead of a class if all you have is data.
- An option hash might suffice.
- When you have behavior to go with the data, it's time to introduce a class.

VISIBILITY

```ruby
class User

  def up_vote(friend)
    bump_karma
    friend.bump_karma
  end

  def bump_karma
    puts "karma up for #{name}"
  end


end
```

```ruby
joe = User.new 'joe'
leo = User.new 'leo'

joe.up_vote(leo)
```

"karma up for joe"
"karma up for leo"

should not be part of the public API

RUBY BITS

VISIBILITY

```ruby
class User

  def up_vote(friend)
    bump_karma
    friend.bump_karma
  end

  private

  def bump_karma
    puts "karma up for #{name}"
  end

end
```

```ruby
joe = User.new 'joe'
leo = User.new 'leo'

joe.up_vote(leo)
```

```
private method 'bump_karma' called
for #<User:0x10ad1f6b8>
```

private methods cannot be called with explicit receiver

RUBY BITS

VISIBILITY

```ruby
class User

  def up_vote(friend)
    bump_karma
    friend.bump_karma
  end

  protected

  def bump_karma
    puts "karma up for #{name}"
  end

end
```

```ruby
joe = User.new 'joe'
leo = User.new 'leo'

joe.up_vote(leo)
```

"karma up for joe"
"karma up for leo"

hidden from outside but accessible from other instances of same class

RUBY BITS

INHERITANCE

```ruby
class Image
  attr_accessor :title, :size, :url
  def to_s
    "#{@title}, {@size}"
  end
end

class Video
  attr_accessor :title, :size, :url
  def to_s
    "#{@title}, {@size}"
  end
end
```

duplicated functionality!

RUBY BITS

INHERITANCE

```ruby
class Attachment
  attr_accessor :title, :size, :url
  def to_s
    "#{@title}, #{@size}"
  end
end

class Image < Attachment
end

class Video < Attachment
end
```

much DRYer!

```ruby
class Video < Attachment
  attr_accessor :duration
end
```

if a method only makes sense for one subclass, put it there.

RUBY BITS

SUPER

```ruby
class User
  def initialize(name)
    @name = name
  end
end
```

doesn't call User#initialize

```ruby
class Follower < User
  def initialize(name, following)
    @following = following
  end
  def relationship
    "#{@name} follows #{@following}"
  end
end
```

```ruby
follower = Follower.new("Oprah", "aplusk")
follower.relationship
```

▶ " follows aplusk"

no @name!

RUBY BITS

SUPER

```ruby
class User
  def initialize(name)
    @name = name
  end
end
```

Calls User#initialize

```ruby
class Follower < User
  def initialize(name, following)
    @following = following
    super(name)
  end
  def relationship
    "#{@name} follows #{@following}"
  end
end
```

```ruby
follower = Follower.new("Oprah", "aplusk")
follower.relationship
```

▶ "Oprah follows aplusk"

RUBY BITS

SUPER

```ruby
class Grandparent
  def my_method
    "Grandparent: my_method called"
  end
end
```

```ruby
class Parent < Grandparent
end
```

not here

```ruby
class Child < Parent
  def my_method
    string = super
    "#{string}\nChild: my_method called"
  end
end
```

```ruby
child = Child.new
puts child.my_method
```

```
Grandparent: my_method called
Child: my_method called
```

RUBY BITS

SUPER

```ruby
class Grandparent
  def my_method(argument)
    "Grandparent: '#{argument}'"
  end
end
```

```ruby
class Child < Parent
  def my_method(argument)
    string = super
    "#{string}\nChild: '#{argument}'"
  end
end
```

same as super(argument)

```ruby
child = Child.new
puts child.my_method('w00t!')
```

```
Grandparent: 'w00t!'
Child: 'w00t!'
```


RUBY BITS

OVERRIDING METHODS

```
class Attachment
  def preview
    case @type
    when :jpg, :png, :gif
      thumbnail
    when :mp3
      player
    end
  end
end
```

typical case

the oddball

This is slow

RUBY BITS

OVERRIDING METHODS

```
class Attachment
  def preview
    thumbnail
  end
end
```

the default

```
class Audio < Attachment
  def preview
    player
  end
end
```

special handling

new subclass!

RUBY BITS

HIDE INSTANCE VARIABLES

```ruby
class User
  def tweet_header
    [@first_name, @last_name].join(' ')
  end
  def profile
    [@first_name, @last_name].join(' ') + @description
  end
end
```

a lot of repetition when working with these...

HIDE INSTANCE VARIABLES

```
class User
  def display_name
    [@first_name, @last_name].join(' ')
  end
  def tweet_header
    display_name
  end
  def profile
    display_name + @description
  end
end
```

You can use an accessor method, even within the same class

RUBY BITS

HIDE INSTANCE VARIABLES

```ruby
class User
  def display_name
    title = case @gender
      when :female
        married? ? "Mrs." : "Miss"
      when :male
        "Mr."
    end
    [title, @first_name, @last_name].join(' ')
  end
end
```

if you need to change the logic later, you can do it in just one place!

RUBY BITS


RUBY
BITS
PRESS START


RUBY
BITS
ACTIVESUPPORT
PRESS START

Install It

```
$ gem install activesupport
$ gem install i18n
```

not a dependency,
but it is helpful

Load It

```
require 'active_support/all'
```

CORE EXTENSIONS: ARRAY

```
array = [0, 1, 2, 3, 4, 5, 6]

array.from(4) ..............▶ [4, 5, 6]
array.to(2)   ..............▶ [0, 1, 2]
array.in_groups_of(3) ......▶ [[0, 1, 2], [3, 4, 5], [6, nil, nil]]
array.split(2)  ............▶ [[0, 1], [3, 4, 5, 6]]
```

removes the element it splits on

Pads with nil

CORE EXTENSIONS: DATE

```
apocalypse = DateTime.new(2012, 12, 21, 14, 27, 45)
```


```
apocalypse.at_beginning_of_day
```


```
apocalypse.at_end_of_month
```


```
apocalypse.at_beginning_of_year
```

ACTIVESUPPORT

RUBY BITS

CORE EXTENSIONS: DATE

```
apocalypse = DateTime.new(2012, 12, 21, 14, 27, 45)
```

```
Fri, 21 Dec 2012 14:27:45 +0000
```

```
apocalypse.advance(years: 4, months: 3, weeks: 2, days: 1)
```

```
Wed, 05 Apr 2017 14:27:45 +0000
```

```
apocalypse.tomorrow
```

```
Sat, 22 Dec 2012 14:27:45 +0000
```

```
apocalypse.yesterday
```

```
Thu, 20 Dec 2012 14:27:45 +0000
```

ACTIVESUPPORT

RUBY BITS

CORE EXTENSIONS: HASH

```
options = {user: 'codeschool', lang: 'fr'}
new_options = {user: 'codeschool', lang: 'fr', password: 'dunno'}
```

```
options.diff(new_options)
```

difference between two hashes

```
{:password=>"dunno"}
```

```
options.stringify_keys
```

turn keys into strings

```
{"user"=>"codeschool", "lang"=>"fr"}
```

CORE EXTENSIONS: HASH

```
options = {
  lang: 'fr',
  user: 'codeschool'
}
```

```
defaults = {
  lang: 'en',
  country: 'us'
}
```

```
options.reverse_merge(defaults)
```

values from this hash win

```
{
  lang: 'fr',
  user: 'codeschool',
  country: 'us'
}
```

RUBY BITS

ACTIVESUPPORT

CORE EXTENSIONS: HASH

```
new_options = {user: 'codeschool', lang: 'fr', password: 'dunno'}
```

```
new_options.except(:password)
```

... remove these keys

... ▶ {:user=>"codeschool", :lang=>"fr"}

```
new_options.assert_valid_keys(:user, :lang)
```

... ▶ Unknown key(s): password (ArgumentError)

throws an exception if the hash contains any keys besides those listed here

RUBY BITS

CORE EXTENSIONS: INTEGER

```ruby
def background_class(index)
  return 'white' if index.odd?
  return 'grey' if index.even?
end
tweets.each_with_index do |tweet, index|
  puts "<div class='#{background_class(index)}'>#{tweet}</div>"
end
```

determine odd and even numbers

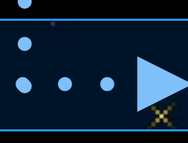
```
<div class='grey'>I had eggs for breakfast.</div>
<div class='white'>@codeschool pwns.</div>
<div class='grey'>Shopping!</div>
<div class='white'>Bedtime.</div>
```

INFLECTOR

```
"#{1.ordinalize} place!"
```

```
"1st place!"
```

```
"#{2.ordinalize} place."
```



```
"2nd place."
```

```
"#{23.ordinalize} place."
```

```
"23rd place."
```

ordinalize numbers to strings

RUBY BITS

ACTIVESUPPORT

INFLECTOR

```
"user".pluralize
```

▶ "users"

```
"women".singularize
```

◀ it knows common special cases

▶ "woman"

```
"octopus".pluralize
```

◀ and even some uncommon ones!

▶ "octopi"

pluralize and
singularize strings

ACTIVESUPPORT

INFLECTOR

```
"ruby bits".titleize
```

capitalize every word

```
"Ruby Bits"
```

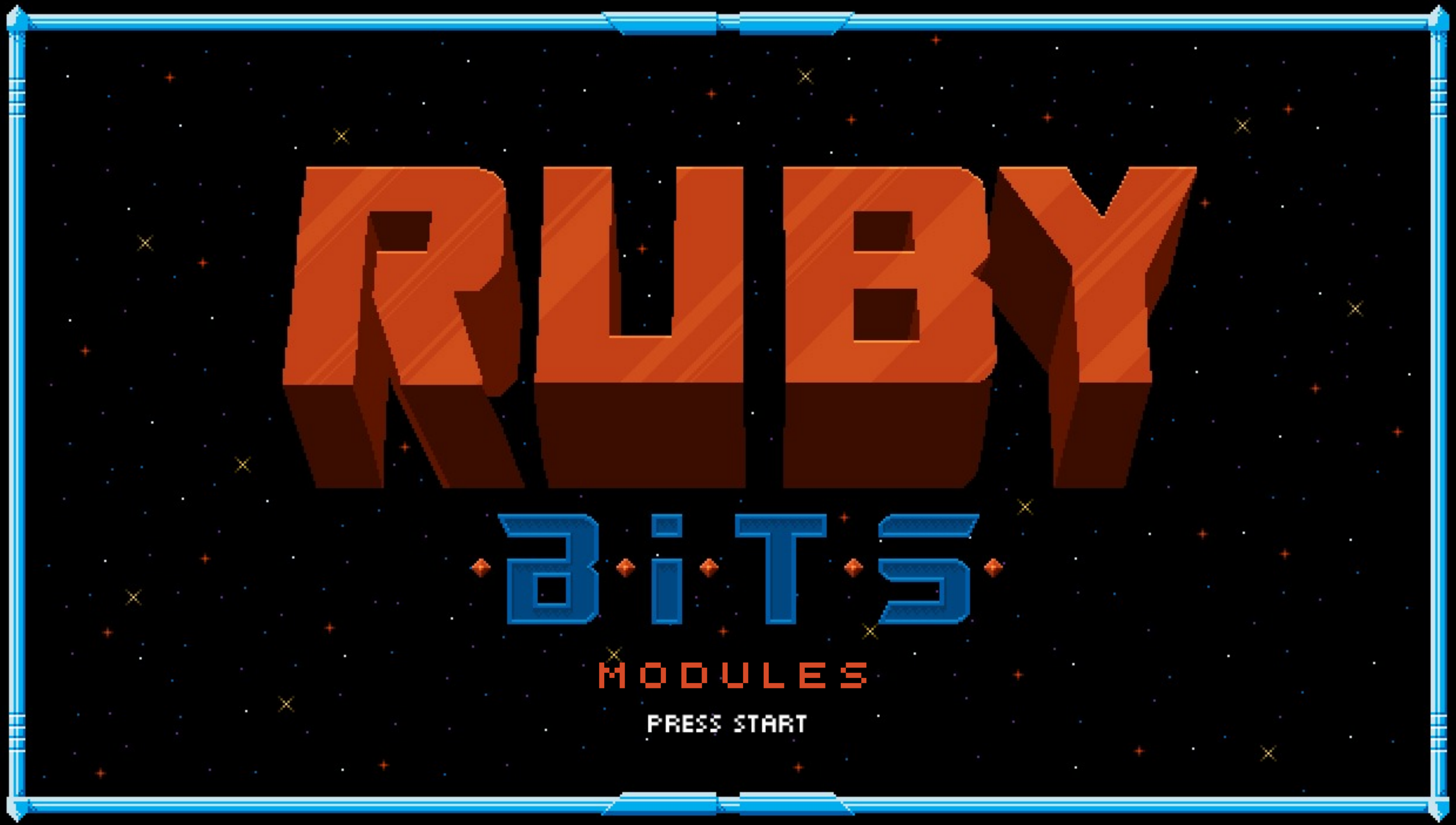
```
"account_options".humanize
```

add spaces and capitalize first word

```
"Account options"
```

ACTIVESUPPORT


RUBY
BITS
PRESS START


RUBY
BITS
MODULES
PRESS START

NAMESPACE

IMAGE_UTILS.RB

```ruby
def preview(image)
end

def transfer(image, destination)
end
```

pollutes global namespace

potential conflicts with methods with same name

RUN.RB

```ruby
require 'image_utils'

image = user.image
preview(image)
```

RUBY BITS

NAMESPACE

IMAGE_UTILS.RB

```ruby
module ImageUtils
  def self.preview(image)
  end

  def self.transfer(image, destination)
  end
end
```

RUN.RB

```ruby
require 'image_utils'

image = user.image
ImageUtils.preview(image)
```

RUBY BITS

MIXIN

IMAGE_UTILS.RB

```
module ImageUtils
  def preview
  end

  def transfer(destination)
  end
end
```

can access properties on objects from host class

Included as instance methods

AVATAR.RB

```
require 'image_utils'
class Image
  include ImageUtils
end
```

RUN.RB

```
image = user.image
image.preview
```

RUBY BITS

ANCESTORS

```
class Image
  include ImageUtils
end
```

Adding module to Image's ancestors chain

```
Image.ancestors
    [Image, ImageUtils, Object, Kernel, BasicObject]

Image.included_modules
    [ImageUtils, Kernel]
```

modules only

MIXINS VS CLASS INHERITANCE

```ruby
class Post
  def share_on_facebook
  end
end

class Image
  def share_on_facebook
  end
end

class Tweet
  def share_on_facebook
  end
end
```

```ruby
class Shareable
end
```

RUBY BITS

MIXINS VS CLASS INHERITANCE

```ruby
class Post < Shareable
end


class Image < Shareable
end


class Tweet < Shareable
end
```

```ruby
class Shareable
  def share_on_facebook
  end
end
```

A class can only have one superclass.

Inheritance suggests specialization.

Some behaviors are not fit for classes.

RUBY BITS

MIXINS VS CLASS INHERITANCE

```ruby
class Post
  include Shareable
end


class Image
  include Shareable
end


class Tweet
  include Shareable
end
```

```ruby
module Shareable
  def share_on_facebook
  end
end
```

RUBY BITS

MIXINS VS CLASS INHERITANCE

```ruby
class Post
  include Shareable
  include Favoritable
end

class Image
  include Shareable
  include Favoritable
end

class Tweet
  include Shareable
  include Favoritable
end
```

```ruby
module Shareable
  def share_on_facebook
  end
end
```

```ruby
module Favoritable
  def add_to_delicious
  end
end
```

RUBY BITS

MIXIN

```ruby
class Tweet
  extend Searchable
end
```

```ruby
module Searchable
  def find_all_from(user)
  end
end
```

Included as class methods

```ruby
Tweet.find_all_from('@GreggPollack')
```

RUBY BITS

MIXIN

```ruby
class Tweet
  extend Searchable
end
```

use extend to expose methods as class methods

```ruby
Tweet.find_all_from('@GreggPollack')
```

```ruby
class Image
  include ImageUtils
end
```

use include to expose methods as instance methods

```ruby
image = user.image
image.preview
```

RUBY BITS

MIXIN

```ruby
class Image
end

image = Image.new
image.extend(ImageUtils)
image.preview
```

```ruby
module ImageUtils
  def preview
  end
end
```

an object is extending the module

the module will not be available in other objects

```ruby
image = Image.new
image.preview
```

```
NoMethodError: undefined method `preview' for #<Image:0x10b448a98>
```

RUBY BITS

HOOKS - SELF.INCLUDED

```ruby
module ImageUtils

  def preview
  end

  def transfer(destination)
  end

  module ClassMethods
    def fetch_from_twitter(user)
    end
  end
end
```

```ruby
class Image
  include ImageUtils
  extend ImageUtils::ClassMethods
end
```

```ruby
image = user.image
image.preview

Image.fetch_from_twitter('gregg')
```

HOOKS - SELF.INCLUDED

```ruby
module ImageUtils
  def self.included(base)
    base.extend(ClassMethods)
  end

  def preview
  end

  def transfer(destination)
  end

  module ClassMethods
    def fetch_from_twitter(user)
    end
  end
end
```

```ruby
class Image
  include ImageUtils
  extend ImageUtils::ClassMethods
end
```

sellf.included is called by Ruby
when a module is included in a class

```ruby
image = user.image
image.preview

Image.fetch_from_twitter('gregg')
```

in this case, the module
name can be anything

ACTIVESUPPORT::CONCERN

```ruby
module ImageUtils

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

  module ClassMethods
    def fetch_from_twitter(user)
    end

    def clean_up
    end
  end
end
```

```ruby
class Image
  include ImageUtils
end
```

RUBY BITS

ACTIVESUPPORT::CONCERN

```
require 'active_support/concern'

module ImageUtils

  extend ActiveSupport::Concern

  included do
    clean_up
  end

  module ClassMethods
    def fetch_from_twitter(user)
    end

    def clean_up
    end
  end
end
```

gem install activesupport

```
class Image
  include ImageUtils
end
```

included block is executed in the context of the Image class

ActiveSupport::Concern looks for a module named ClassMethods

RUBY BITS

ACTIVESUPPORT::CONCERN

```ruby
module ImageUtils
  def self.included(base)
    base.extend(ClassMethods)
  end

  module ClassMethods
    def clean_up; end
  end
end
```

```ruby
class Image
  include ImageUtils
  include ImageProcessing
end
```

ImageProcessing depends on ImageUtils

base is Image class

```ruby
module ImageProcessing
  def self.included(base)
    base.clean_up
  end
end
```

calls method on Image class

RUBY BITS

ACTIVESUPPORT::CONCERN

```ruby
module ImageUtils
  def self.included(base)
    base.extend(ClassMethods)
  end

  module ClassMethods
    def clean_up; end
  end
end

module ImageProcessing

  include ImageUtils

  def self.included(base)
    base.clean_up
  end
end
```

```ruby
class Image
  include ImageProcessing
end
```

base is ImageProcessing module

undefined method error

RUBY BITS

ACTIVESUPPORT::CONCERN

```ruby
module ImageUtils
  def self.included(base)
    base.extend(ClassMethods)
  end

  module ClassMethods
    def clean_up; end
  end
end

module ImageProcessing
  extend ActiveSupport::Concern
  include ImageUtils

  def self.included(base)
    base.clean_up
  end
end
```

base is Image class

```ruby
class Image
  include ImageProcessing
end
```

Dependencies are properly resolved

RUBY BITS

ACTIVESUPPORT::CONCERN

```ruby
module ImageUtils
  extend ActiveSupport::Concern

  module ClassMethods
    def clean_up; end
  end
end
```

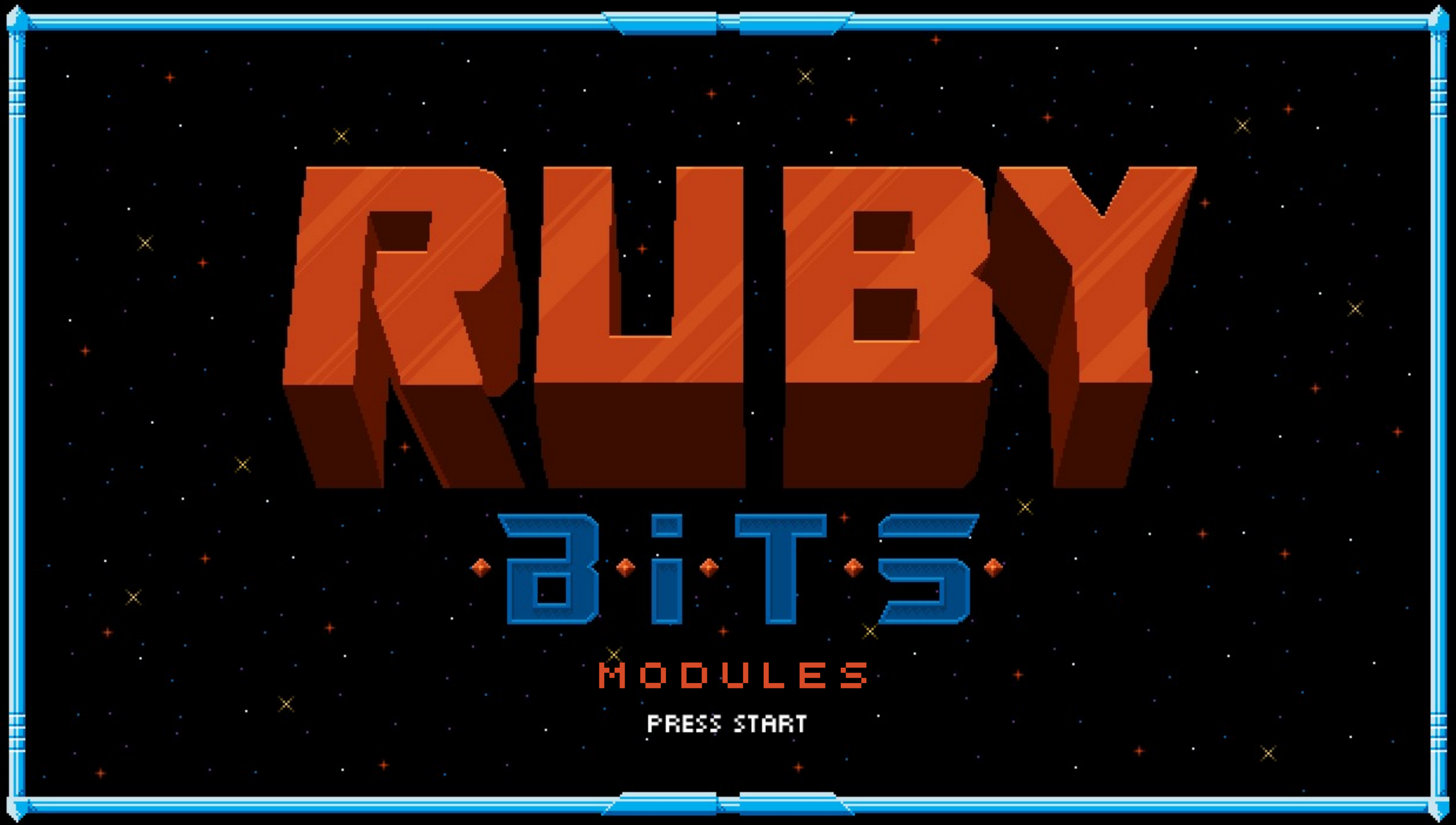
```ruby
class Image
  include ImageProcessing
end
```

Dependencies are properly resolved


```ruby
module ImageProcessing
  extend ActiveSupport::Concern
  include ImageUtils

  included do
    clean_up
  end
end
```

RUBY BITS


RUBY
BITS
MODULES
PRESS START


RUBY
BITS
BLOCKS
PRESS START

USING BLOCKS

```ruby
words = ['Had', 'eggs', 'for', 'breakfast.']
for index in 0..(words.length - 1)
  puts words[index]
end
```

```ruby
words = ['Had', 'eggs', 'for', 'breakfast.']
words.each { |word| puts word }
```

RUBY BITS

DECLARING BLOCKS

```
words.each { |word| puts word }
```

braces if the block is a single line

```
words.each do |word|
  backward_word = word.reverse
  puts backward_word
end
```

do/end if it's multiple lines

this is the FIRST of two conventions!

RUBY BITS

DECLARING BLOCKS

```
words.each do |word|
  puts word
end
```

do/end if the block DOES something (has a side effect)

```
backward_words = words.map { |word| word.reverse }
```

braces if you're just going to use its return value

this is the SECOND of two conventions!

RUBY BITS

YIELD

```ruby
def call_this_block_twice
  yield
  yield
end
```

```ruby
call_this_block_twice { puts "twitter" }
```

```
twitter
twitter
```

```ruby
call_this_block_twice { puts "tweet" }
```

```
tweet
tweet
```

YIELD - ARGUMENTS

```ruby
def call_this_block
  yield "tweet"
end
```

```ruby
call_this_block { |myarg| puts myarg }
```

▶ tweet

```ruby
call_this_block { |myarg| puts myarg.upcase }
```

▶ TWEET

RUBY BITS

YIELD - RETURN VALUE

```ruby
def puts_this_block
  puts yield
end
```

```ruby
puts_this_block { "tweet" }
```

▶ tweet

RUBY BITS

YIELD

```
def call_this_block

  block_result = yield "foo"
  puts block_result
end
```

▶ "oof"

```
call_this_block { |arg| arg.reverse }
```

USING BLOCKS

```
class Timeline
  def list_tweets
    @user.friends.each do |friend|
      friend.tweets.each { |tweet| puts tweet }
    end
  end
  def store_tweets
    @user.friends.each do |friend|
      friend.tweets.each { |tweet| tweet.cache }
    end
  end
end
```

same iteration, different logic

RUBY BITS

YOUR OWN "EACH"

```
class Timeline
  def each
    @user.friends.each do |friend|
      friend.tweets.each { |tweet| yield tweet }
    end
  end
end


timeline = Timeline.new(user)
timeline.each { |tweet| puts tweet }
timeline.each { |tweet| tweet.cache }
```

re-use iteration

vary logic

RUBY BITS

ENUMERABLE

```
class Timeline
  def each
    ...
  end
  include Enumerable
end
```

you implemented "each", now mix in Enumerable

```
timeline.sort_by  { |tweet| tweet.created_at }
timeline.map      { |tweet| tweet.status }
timeline.find_all { |tweet| tweet.status =~ /\@codeschool/ }
```

you instantly get all these methods, and more!

"EXECUTE AROUND"

```ruby
def update_status(user, tweet)
  begin
    sign_in(user)
    post(tweet)
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
  end
end
```

```ruby
def get_list(user, list_name)
  begin
    sign_in(user)
    retrieve_list(list_name)
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
  end
end
```

everything but the core logic is duplicated!

"EXECUTE AROUND"

```ruby
def while_signed_in_as(user)
  begin
    sign_in(user)
    yield
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
  end
end
```

```ruby
while_signed_in_as(user) do
  post(tweet)
end
```

```ruby
tweets = while_signed_in_as(user) do
  retrieve_list(list_name)
end
```

now you can just call the single method with a block!

"EXECUTE AROUND"

```ruby
def while_signed_in_as(user)
  sign_in(user)
  yield
  rescue ConnectionError => e
    logger.error(e)
  ensure
    sign_out(user)
end
```

no need for begin/end within a method!

RUBY BITS


RUBY
BITS
BLOCKS
PRESS START