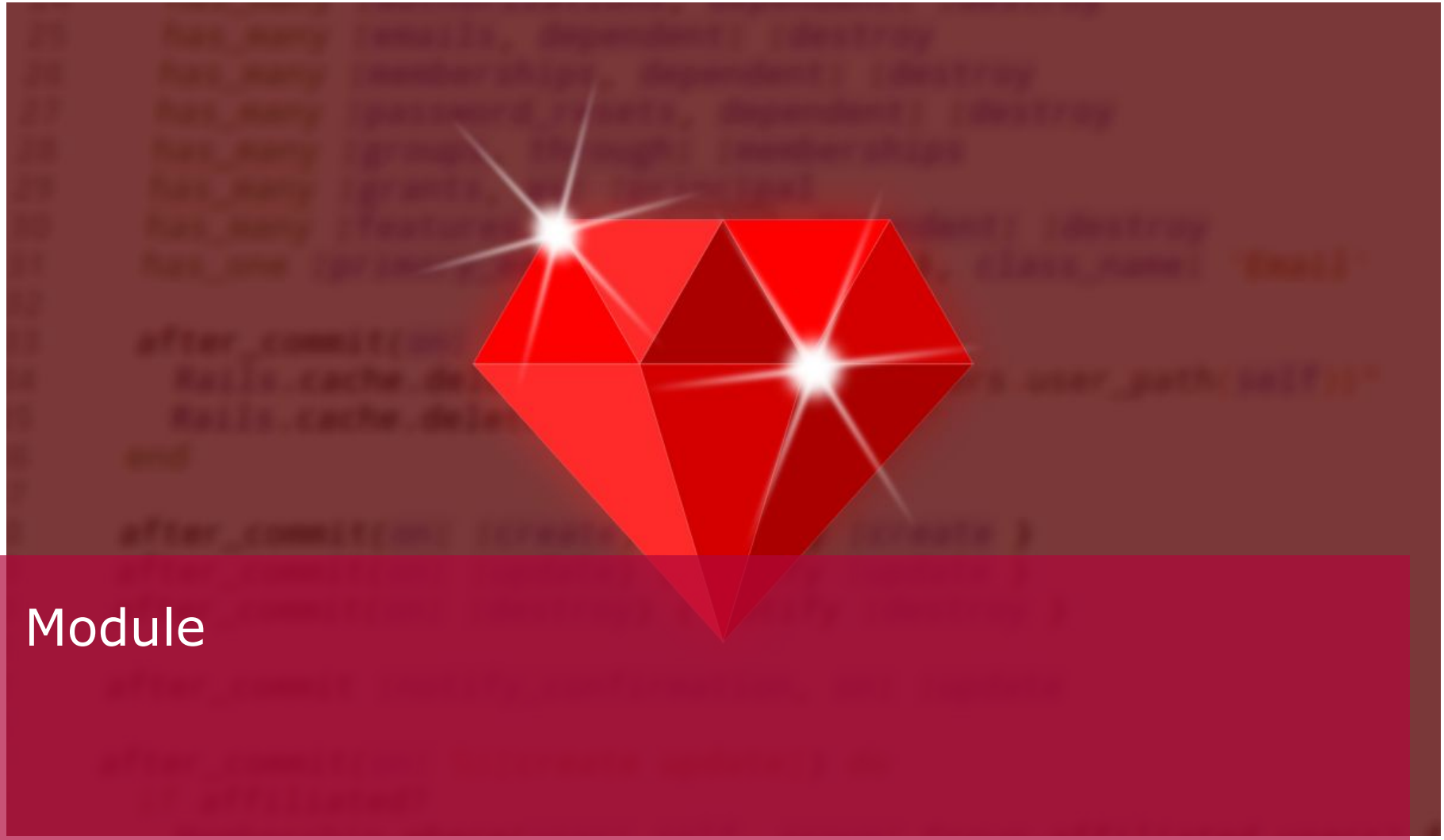


Module

Module

Module

- Lose Sammlung von zusammengehörigen Methoden
- Hilfreich zur Gruppierung zusammengehöriger Funktionalität

Funktionssammlungen

- Gruppierung zustandsloser Hilfsfunktionen

Erweitern von Objekten

- Klassen und Objekte können Module einbinden
- Übernehmen Modulfunktionen als Methoden

Module: Funktionssammlungen

Funktionssammlungen

- Sammlung zustandsloser Funktionen
- Aufrufbar direkt auf dem Modulobjekt

Beispiel: Mathematische Funktionen

- z.B. Wurzel

```ruby
Math.class  # => Module
Math.sqrt 9 # => 3.0
```

⇒ https://ruby-doc.org/core-2.5.1/Math.html

Module: Erweiterung von Klassen

`include`

- Inkludiert Modulfunktionen in Klasse (Mixin)
- Kann Methoden überschreiben und mit `super` erweitern

```ruby
module Yelling
  def say(message)
    super message.upcase
  end
end

class Announcer < Speaker
  include Yelling

  def say(message)
    super "Welcome #{message}"
  end
end

Announcer.new.say("James")
# WELCOME JAMES
```

Beispiel: `Comparable`

Comparable

- Sammlung von Vergleichsmethoden:
  `#>`, `#<`, `#<=`, `#>=`, `#==`, `#between?`, …

Voraussetzung

- Erfordert Definition der Methode `#<=>(other)`
  - Rückgabewert -1, wenn `other` größer ist
  - Rückgabewert 0, wenn Objekte “gleich” sind
  - Rückgabewert 1, wenn `other` kleiner ist

Beispiel: Comparable

```ruby
class Task
  attr_accessor :title

  include Comparable

  def initialize(title)
    @title = title
  end

  # Compare by lowercase name
  def <=>(other)
    title.downcase <=> other.title.downcase
  end
end

Task.new("Learn Ruby").respond_to? :between? # => true
Task.new("Learn Ruby") > Task.new("get certificate")
# => true
```

Beispiel: `Enumerable`

Enumerable

- Funktionen für abzählbare Objekte zum Suchen, Filtern und Auflisten: z.B. `#map`, `#select`, `#all?`, `#any`, ...

Voraussetzung

- Erfordert Definition der Methode `#each(&block)`
  - Gibt alle Elemente nacheinander an übergebenen Block

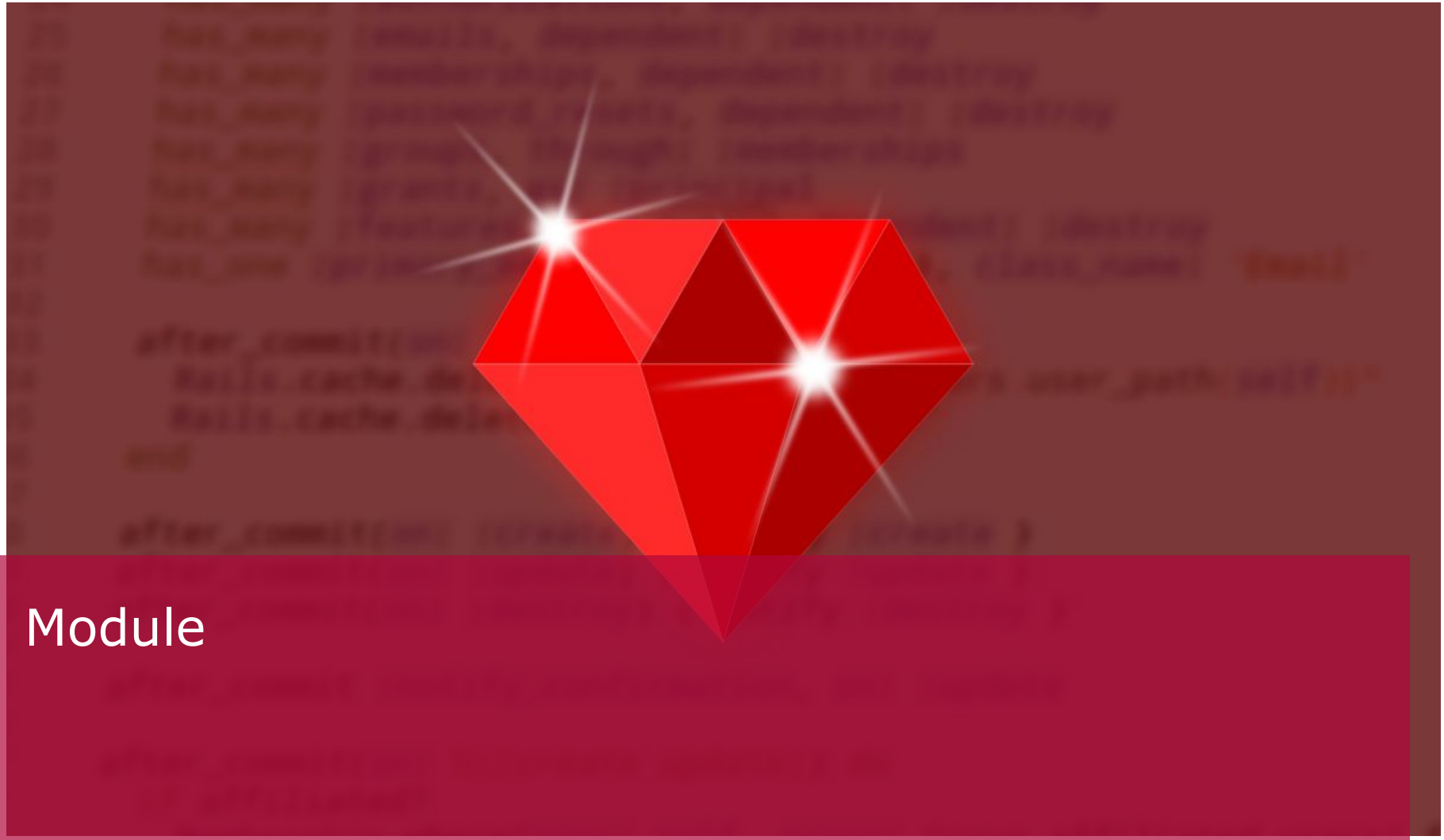
Beispiel: Enumerable

```ruby
class TaskList
  # ...

  include Enumerable

  def each
    @tasks.each { |task| yield task }
  end
end

TaskList.new(...).any? { |task| task.overdue? }
```



Module