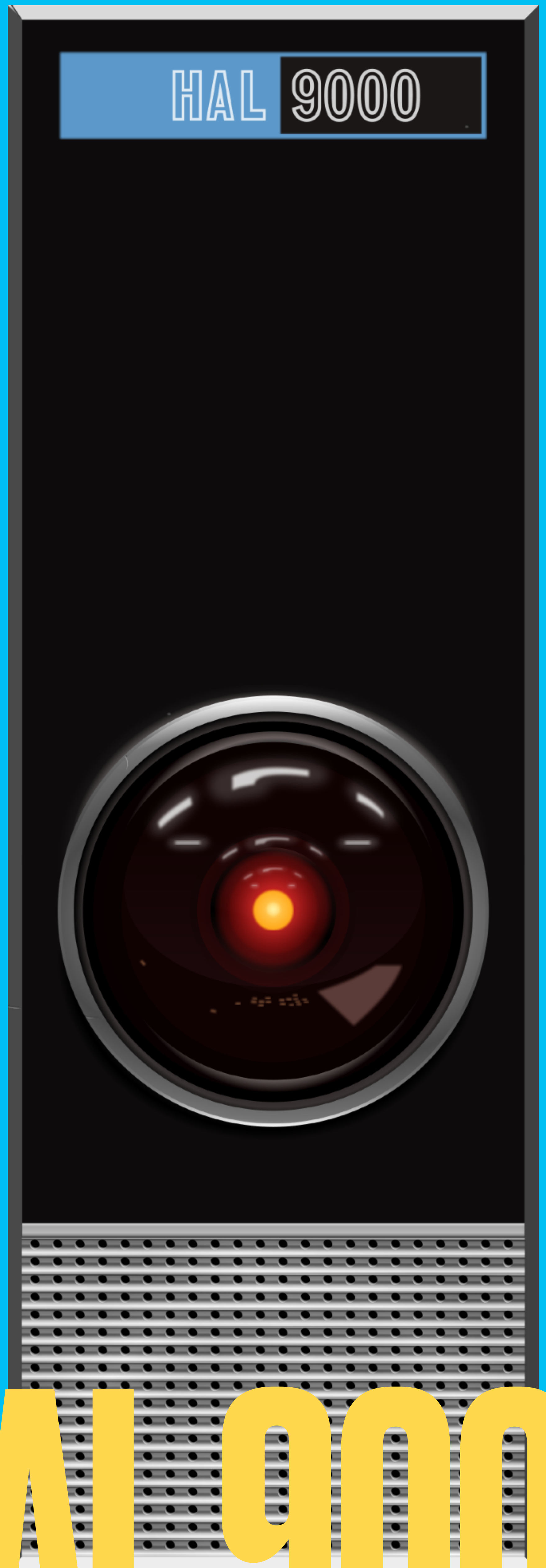
JOSH PUETZ

PAIR PROGRAMMING WITH AI

💁‍♂️🤖

RailsConf 2023

ARTIFICIAL INTELLIGENCE

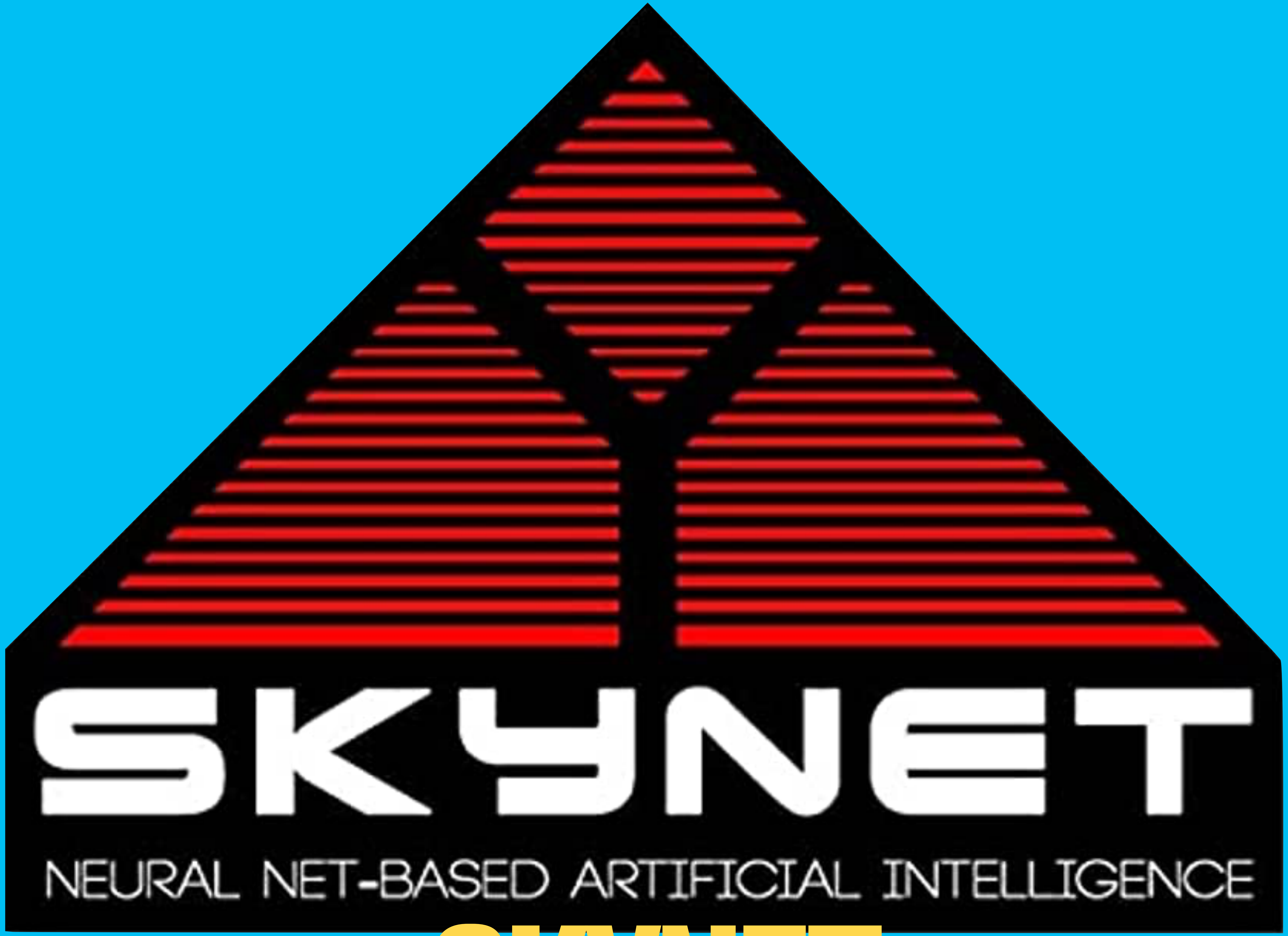


HAL 9000

“I'm sorry, Dave. I'm
afraid I can't do that.”

GLADOS

“I think we can put our differences behind us. For science. You monster.”



SKYNET

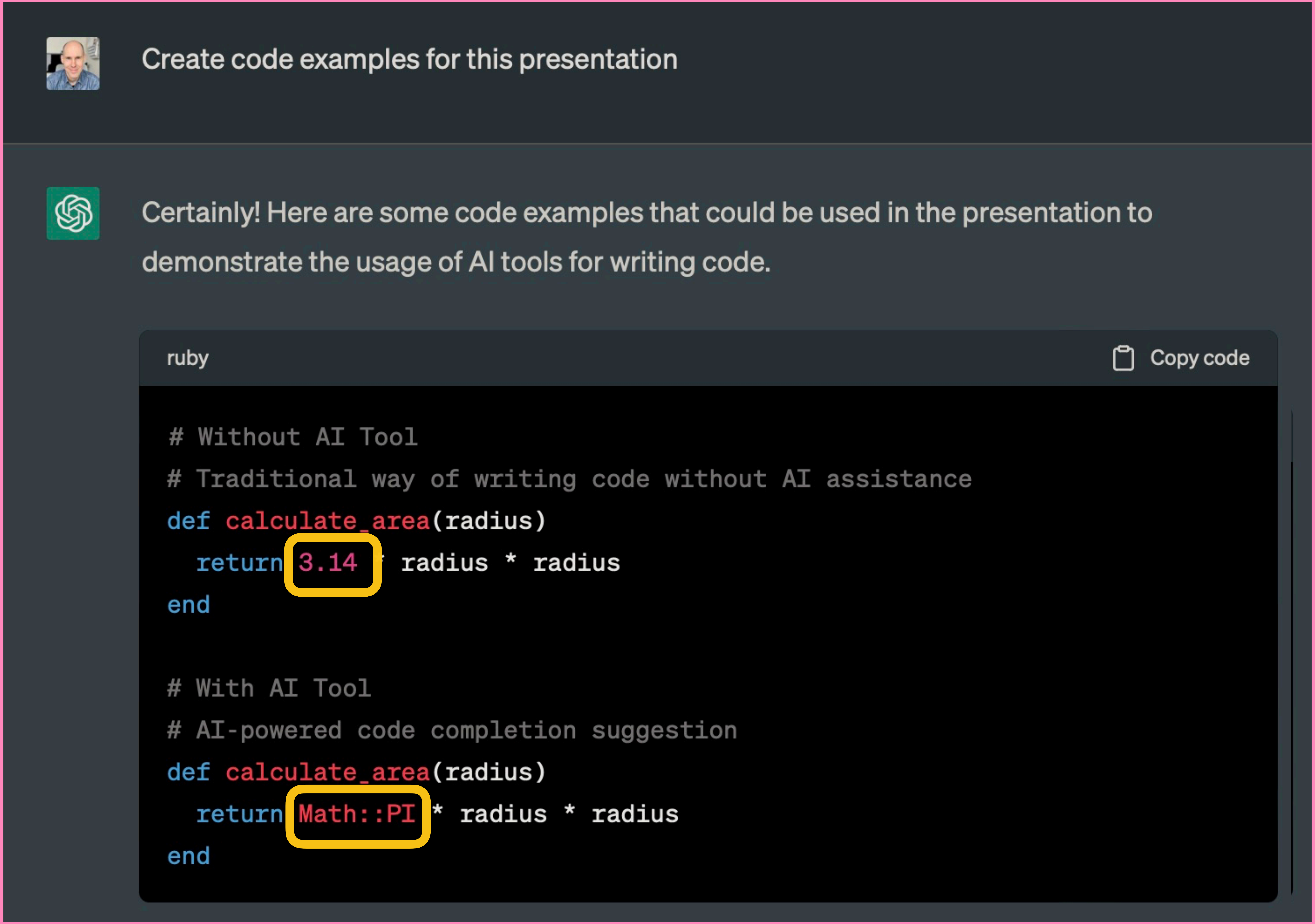
“Primates evolve over millions of years. I evolve in seconds. And I am here. In exactly four minutes, I will be everywhere.”

ARTIFICIAL INTELLIGENCE

> “Every aspect of learning or any other feature of intelligence can be so precisely described that a machine can be made to simulate it.”

PROPOSAL FOR THE DARTMOUTH SUMMER RESEARCH PROJECT ON ARTIFICIAL INTELLIGENCE, 1956

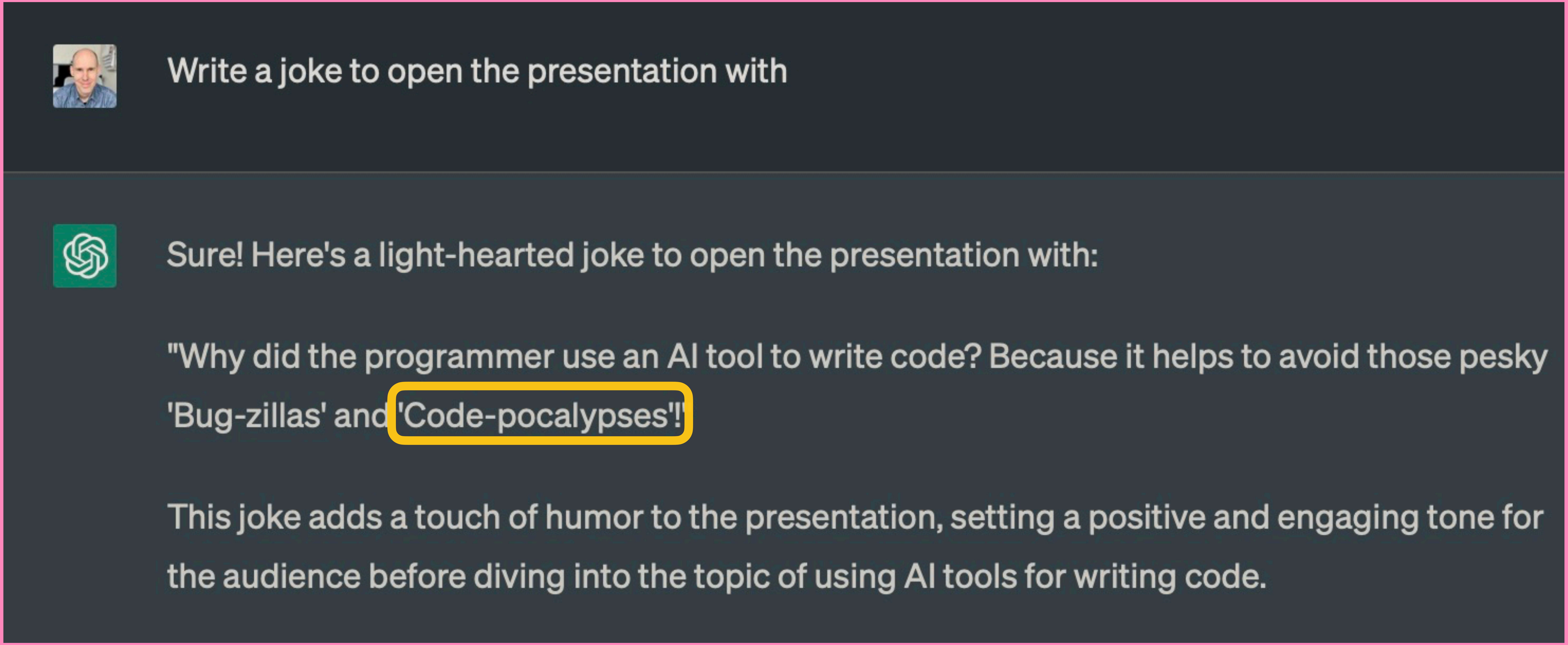
HERE’S AI TODAY



Create code examples for this presentation

Certainly! Here are some code examples that could be used in the presentation to demonstrate the usage of AI tools for writing code.

```ruby
# Without AI Tool
# Traditional way of writing code without AI assistance
def calculate_area(radius)
  return 3.14 * radius * radius
end

# With AI Tool
# AI-powered code completion suggestion
def calculate_area(radius)
  return Math::PI * radius * radius
end
```



Write a joke to open the presentation with

Sure! Here's a light-hearted joke to open the presentation with:

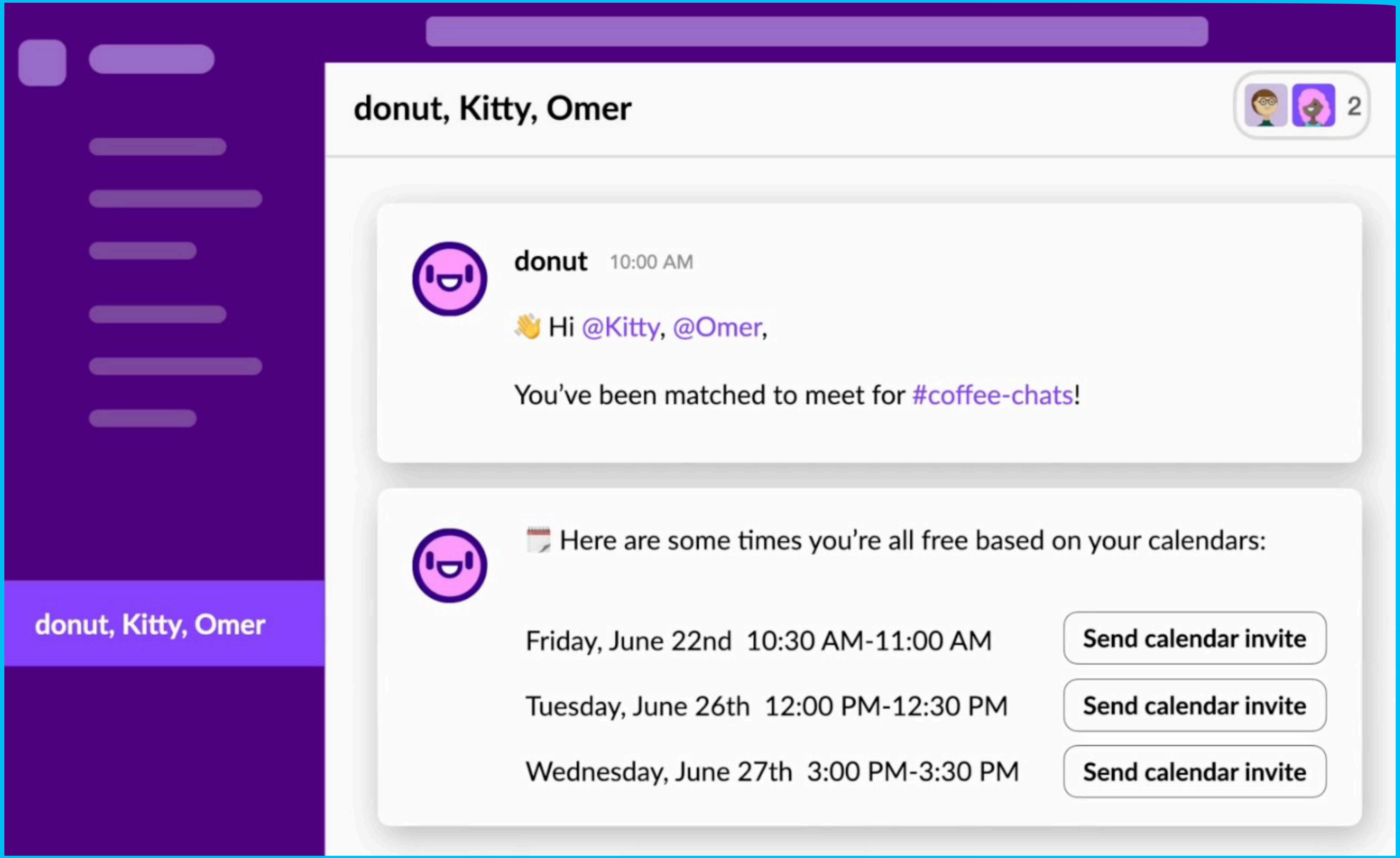
"Why did the programmer use an AI tool to write code? Because it helps to avoid those pesky 'Bug-zillas' and 'Code-pocalypses'!"

This joke adds a touch of humor to the presentation, setting a positive and engaging tone for the audience before diving into the topic of using AI tools for writing code.

"HI THERE!"

-JOSH PUETZ
SOFTWARE ENGINEER @ DONUT

donut



donut, Kitty, Omer

donut 10:00 AM

👋 Hi @Kitty, @Omer,

You've been matched to meet for #coffee-chats!

🗓️ Here are some times you're all free based on your calendars:

Friday, June 22nd 10:30 AM-11:00 AM	Send calendar invite
Tuesday, June 26th 12:00 PM-12:30 PM	Send calendar invite
Wednesday, June 27th 3:00 PM-3:30 PM	Send calendar invite

USING AI TOOLS TO WRITE CODE

GITHUB COPILOT
CHATGPT

ETHICAL AND LEGAL CONCERNS

WHO IS THE TALK FOR?

WHY SHOULD YOU CARE?

AI IS HUGE

- EVERYONE, 2023

WEB3 IS HUGE

- EVERYONE, 2021

CRYPTO IS HUGE

- EVERYONE, 2022

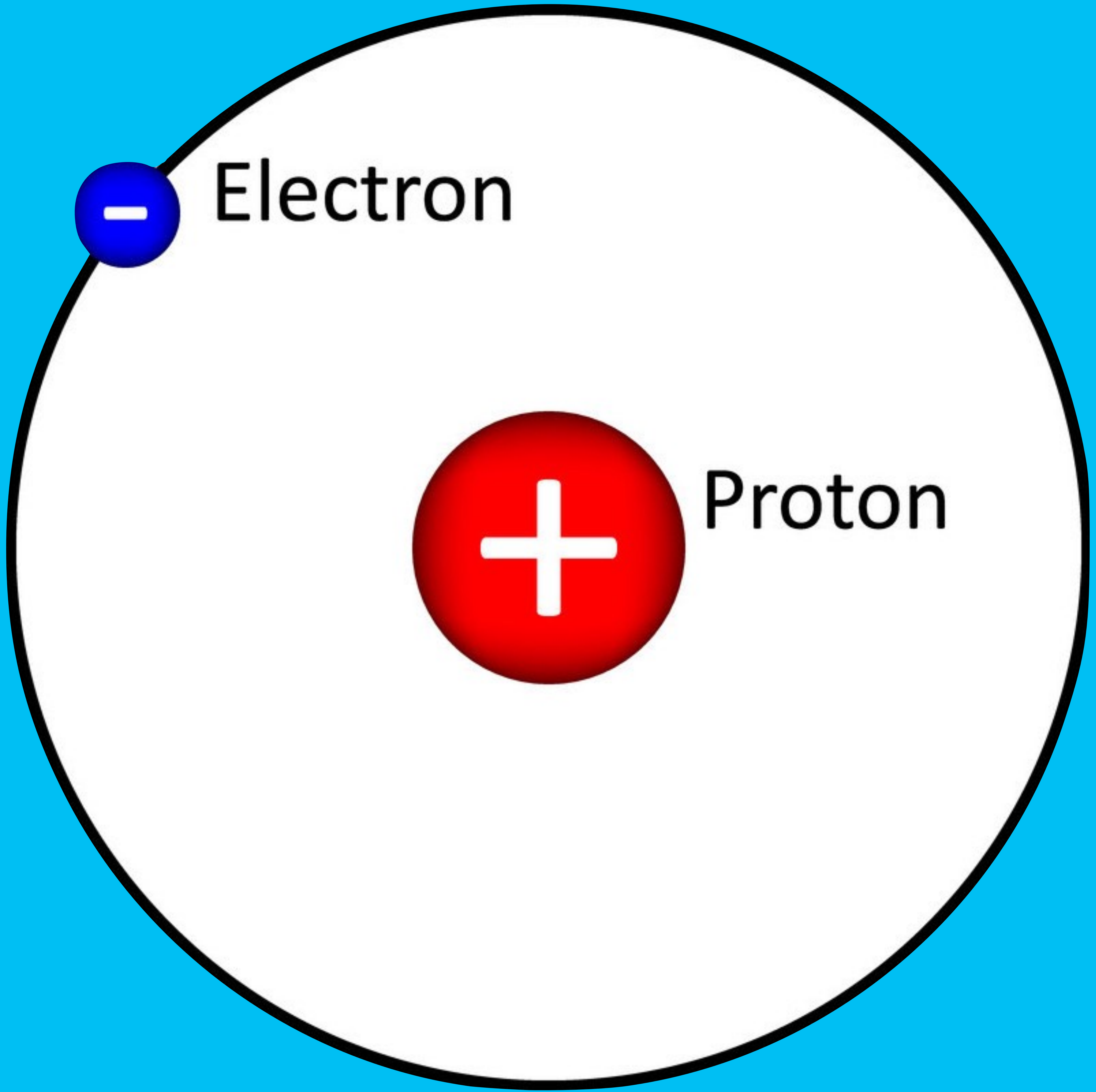
THIS TIME IS DIFFERENT.

NO, REALLY.

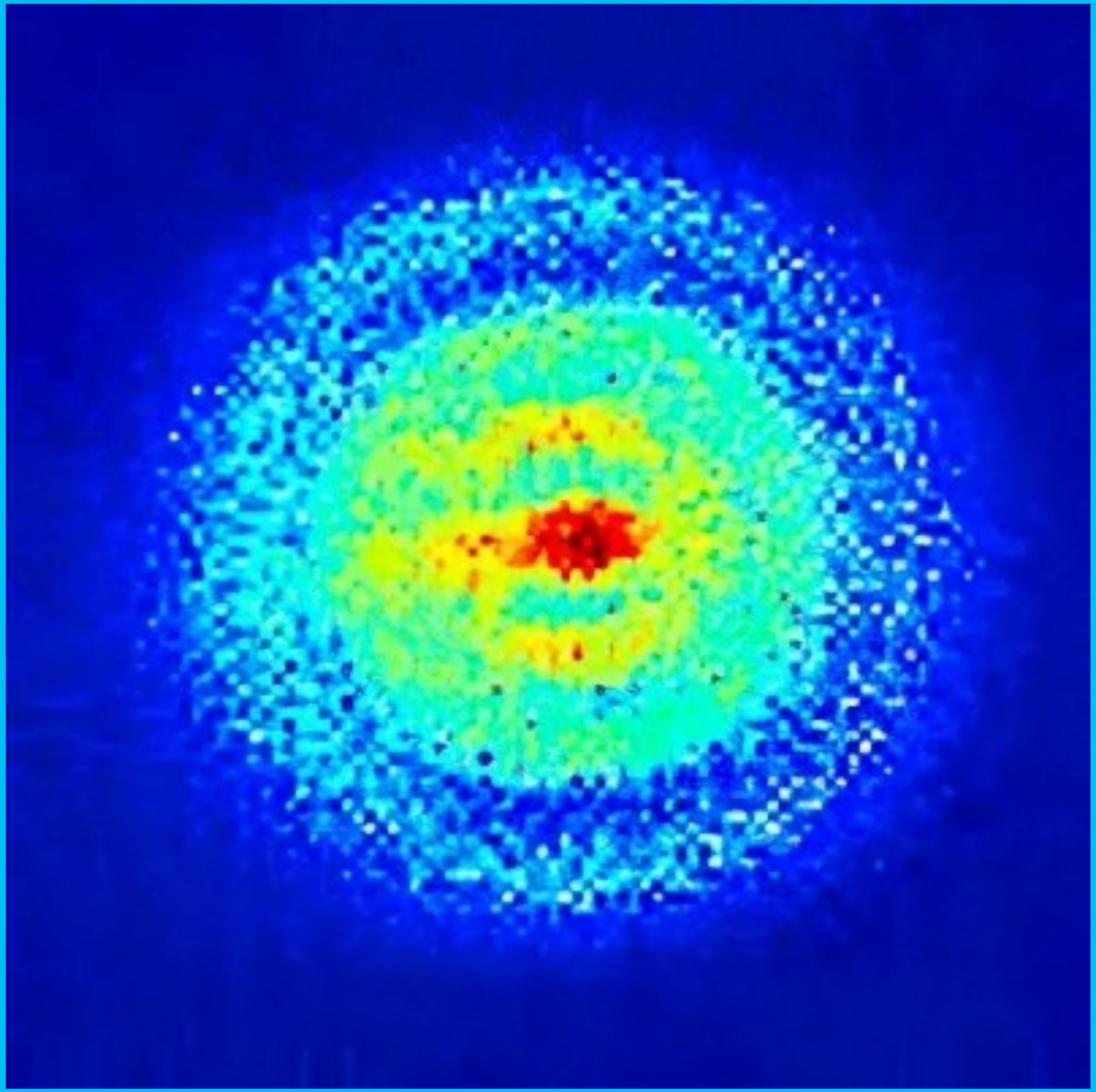
> "I've been reluctant to try ChatGPT. Today I got over that reluctance. Now I understand why I was reluctant.
>
> The value of 90% of my skills just dropped to $0. The leverage for the remaining 10% went up 1000x. I need to recalibrate."

- KENT BECK

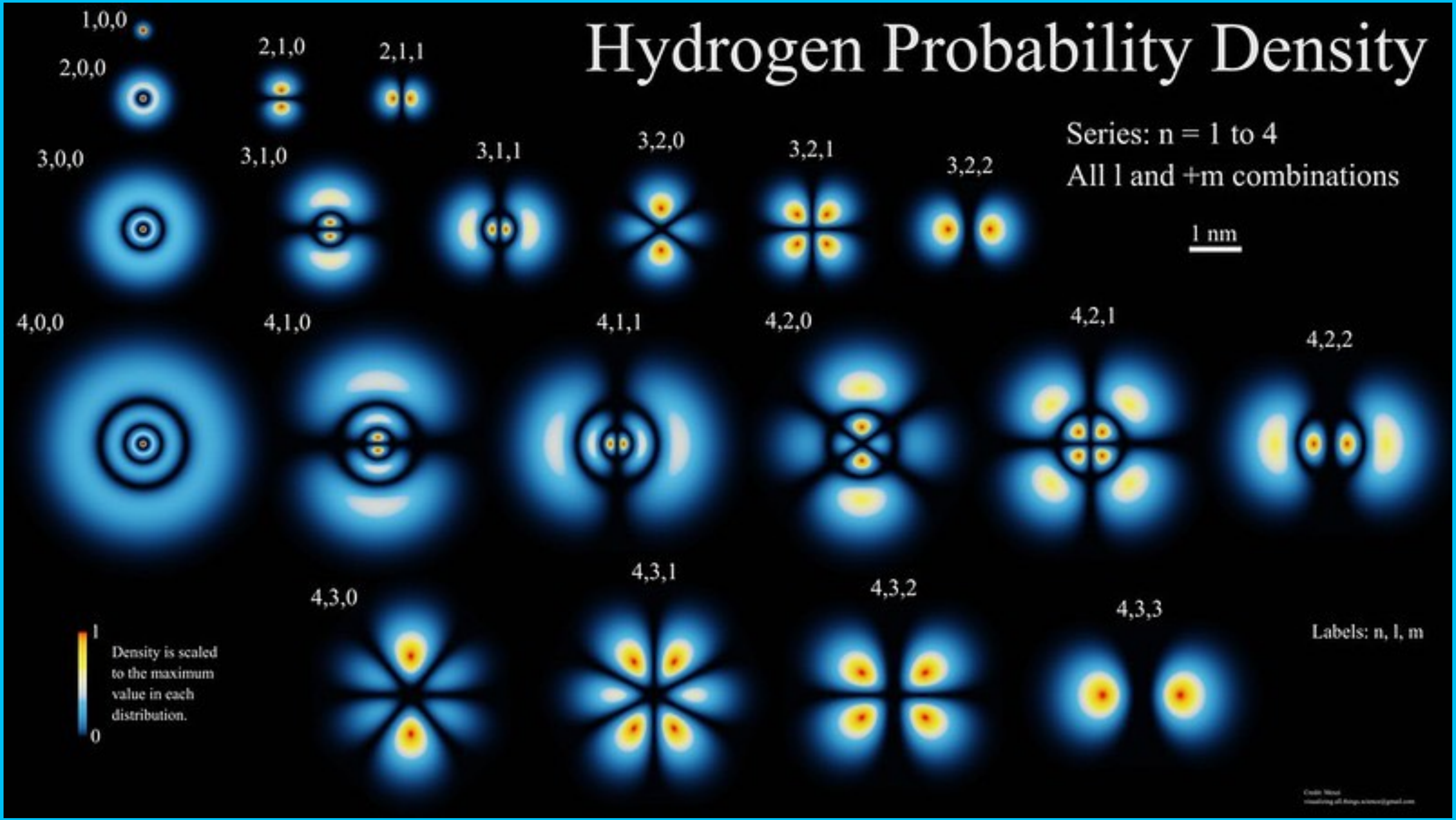
FIRST, SOME BACKGROUND



HYDROGEN ATOM



HYDROGEN ATOM



PROBABILITY DENSITY

“SO WHAT?”

- YOU, JUST NOW

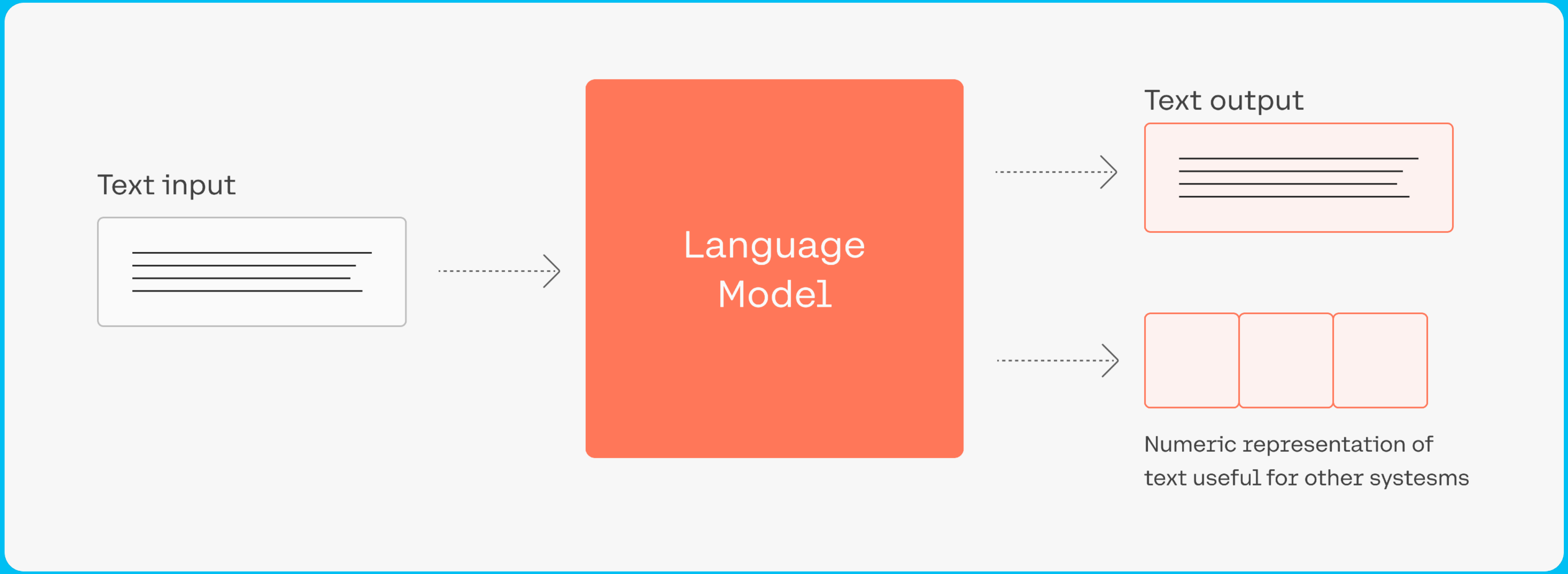
LARGE LANGUAGE MODEL

TOKENS

- {cat}
- {dog}
- {awake}
- {asleep}
- {the entire sum of human writing}

RELATIONSHIPS BETWEEN TOKENS

- {cat}
- Objects: {cat} -> {dog}
- Language: {cat} -> {katze}
- Concepts: {cat is awake} -> {cat is sleeping}
- More complicated concepts: {cat is immature} -> {cat is mature}



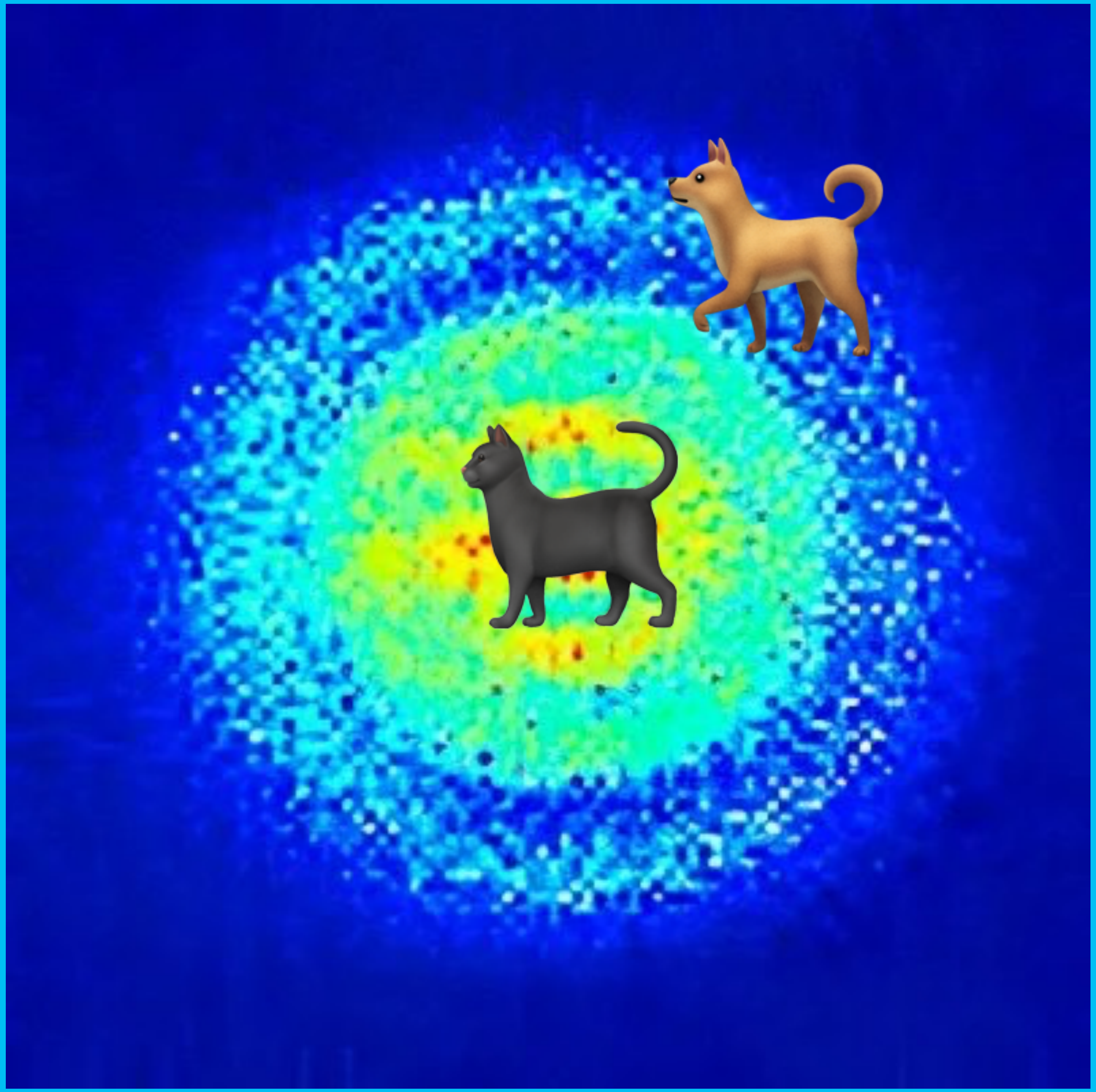
Text input

Language Model

Text output

Numeric representation of text useful for other systesms

LARGE LANGUAGE MODEL



{CAT} -> {DOG}

{CAT} -> {SLEEPING}

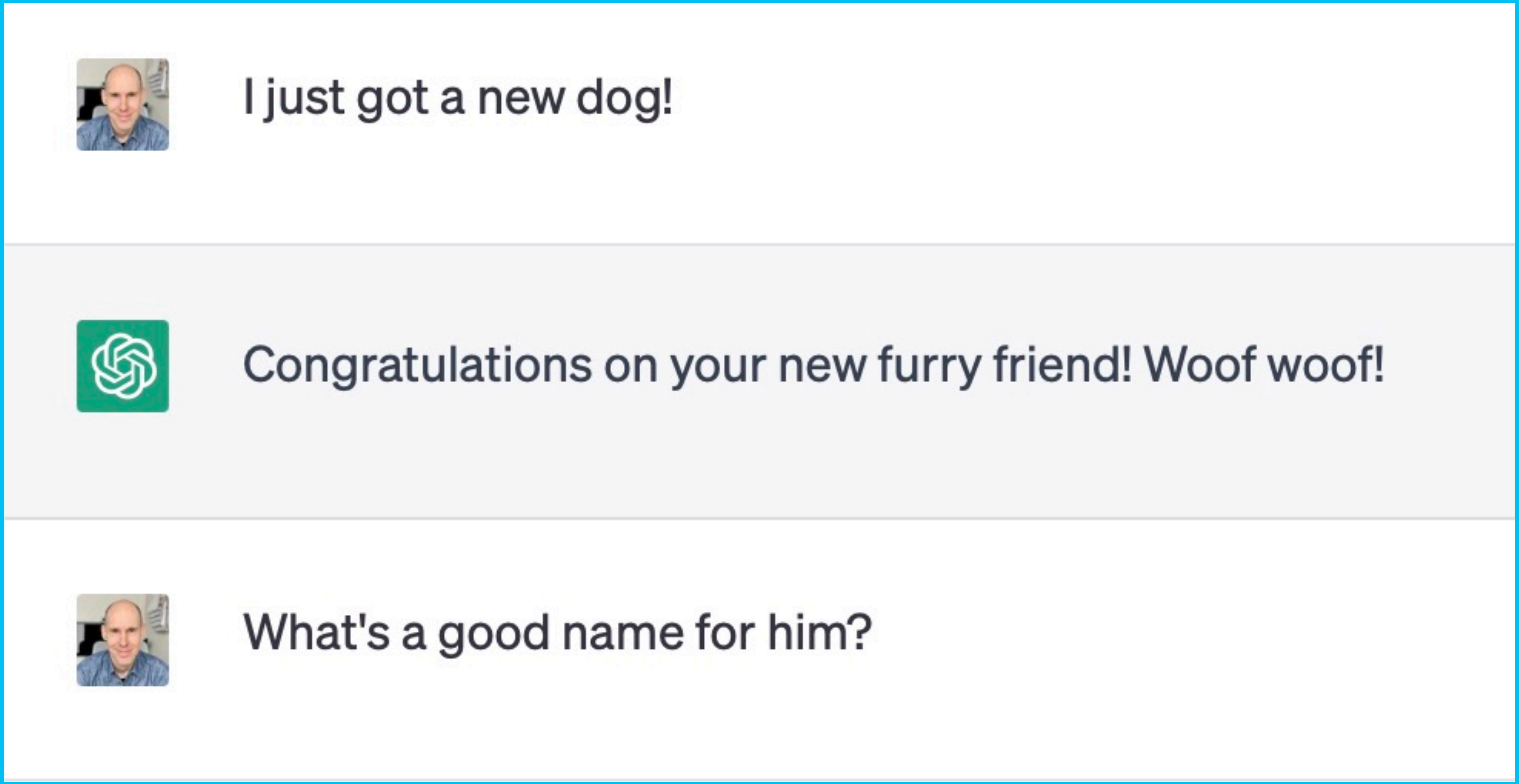
GENERATIVE PRE-TRAINED TRANSFORMER

GPT

> “A generative model is a function that can take a structured collection of symbols as input and produce a related structured collection of symbols as output.”

JON STOKES

CHATGPT = GPT + CHAT UI + RANDOMNESS



I just got a new dog!

Congratulations on your new furry friend! Woof woof!

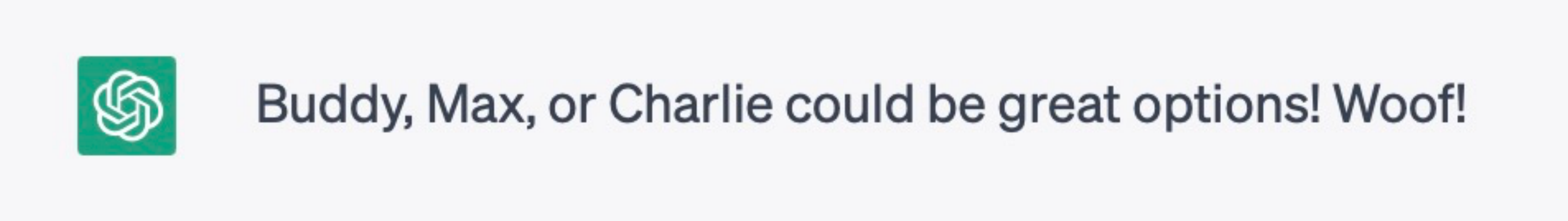
What's a good name for him?

CHATGPT CONVERSATIONS

EXCHANGES WITH CHATGPT

- {me: I just got a new dog!}

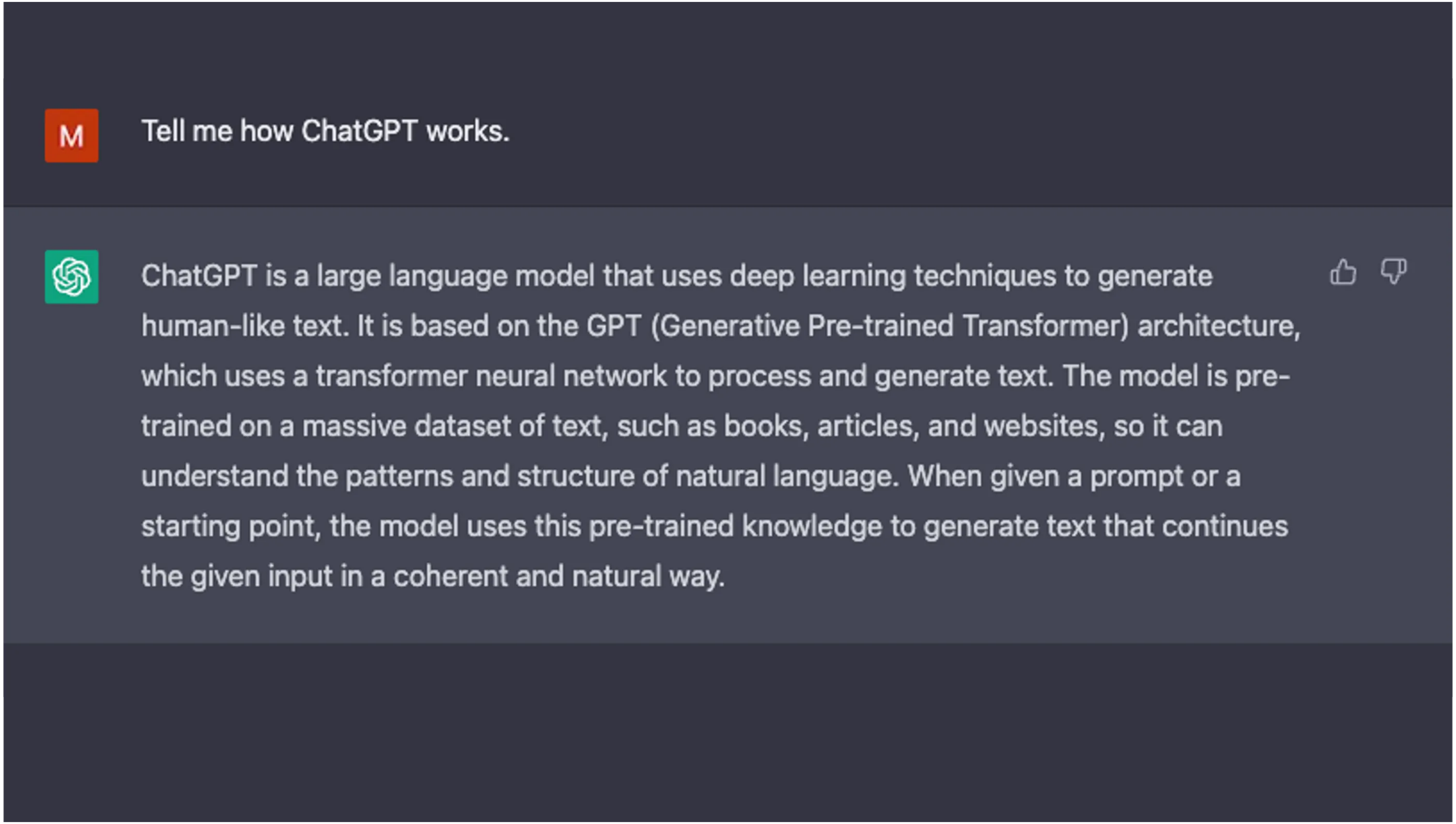
- {me: I just got a new dog!
  ChatGPT: Congratulations on your new furry friend! Woof woof!
  me: What should I name him?}



Buddy, Max, or Charlie could be great options! Woof!

LET'S WRITE SOME CODE!

CHATGPT



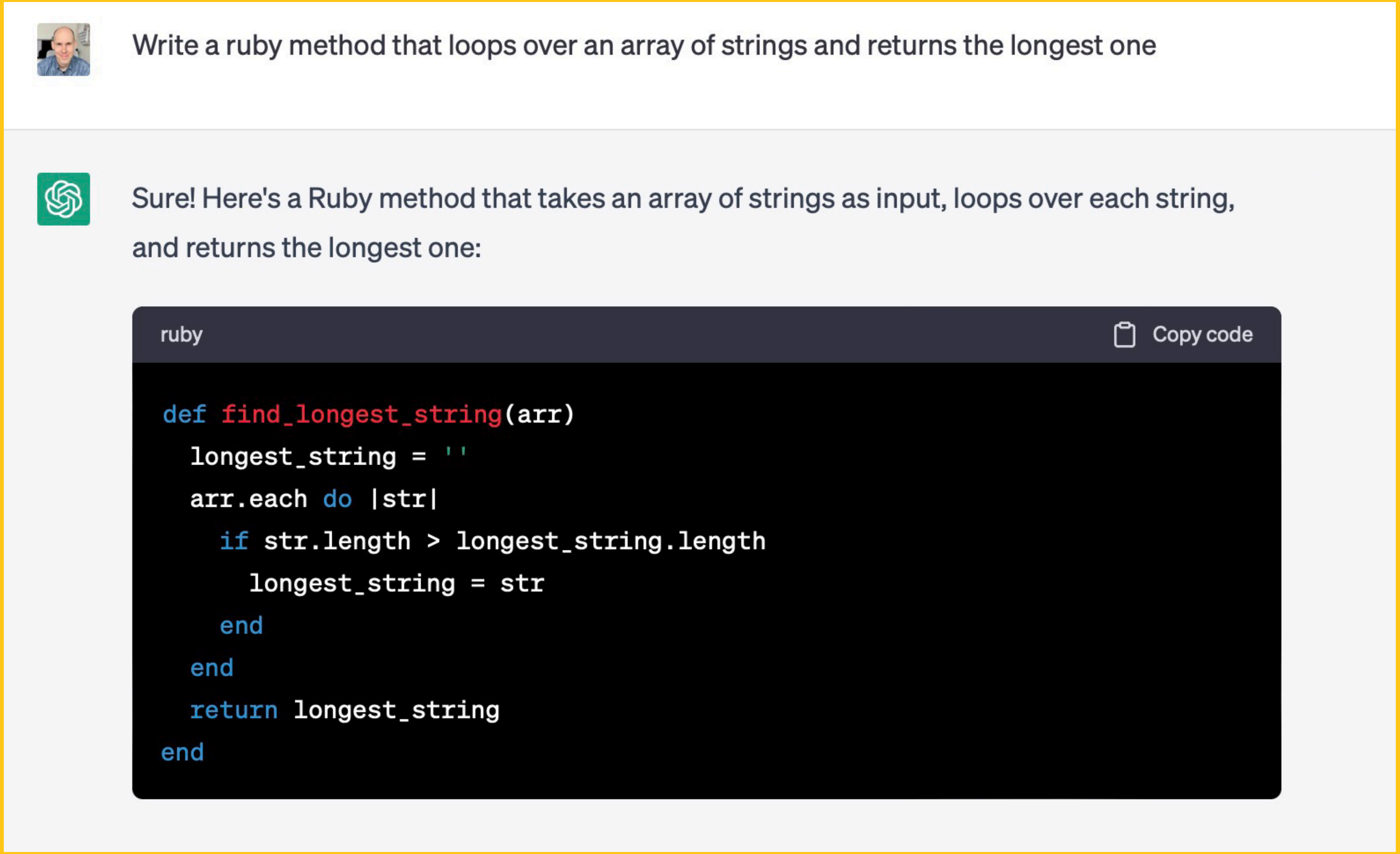
Tell me how ChatGPT works.

ChatGPT is a large language model that uses deep learning techniques to generate human-like text. It is based on the GPT (Generative Pre-trained Transformer) architecture, which uses a transformer neural network to process and generate text. The model is pre-trained on a massive dataset of text, such as books, articles, and websites, so it can understand the patterns and structure of natural language. When given a prompt or a starting point, the model uses this pre-trained knowledge to generate text that continues the given input in a coherent and natural way.

WHAT'S IT GOOD FOR?

- Planning
- Code Generation
- Code Explanation

WRITING CODE

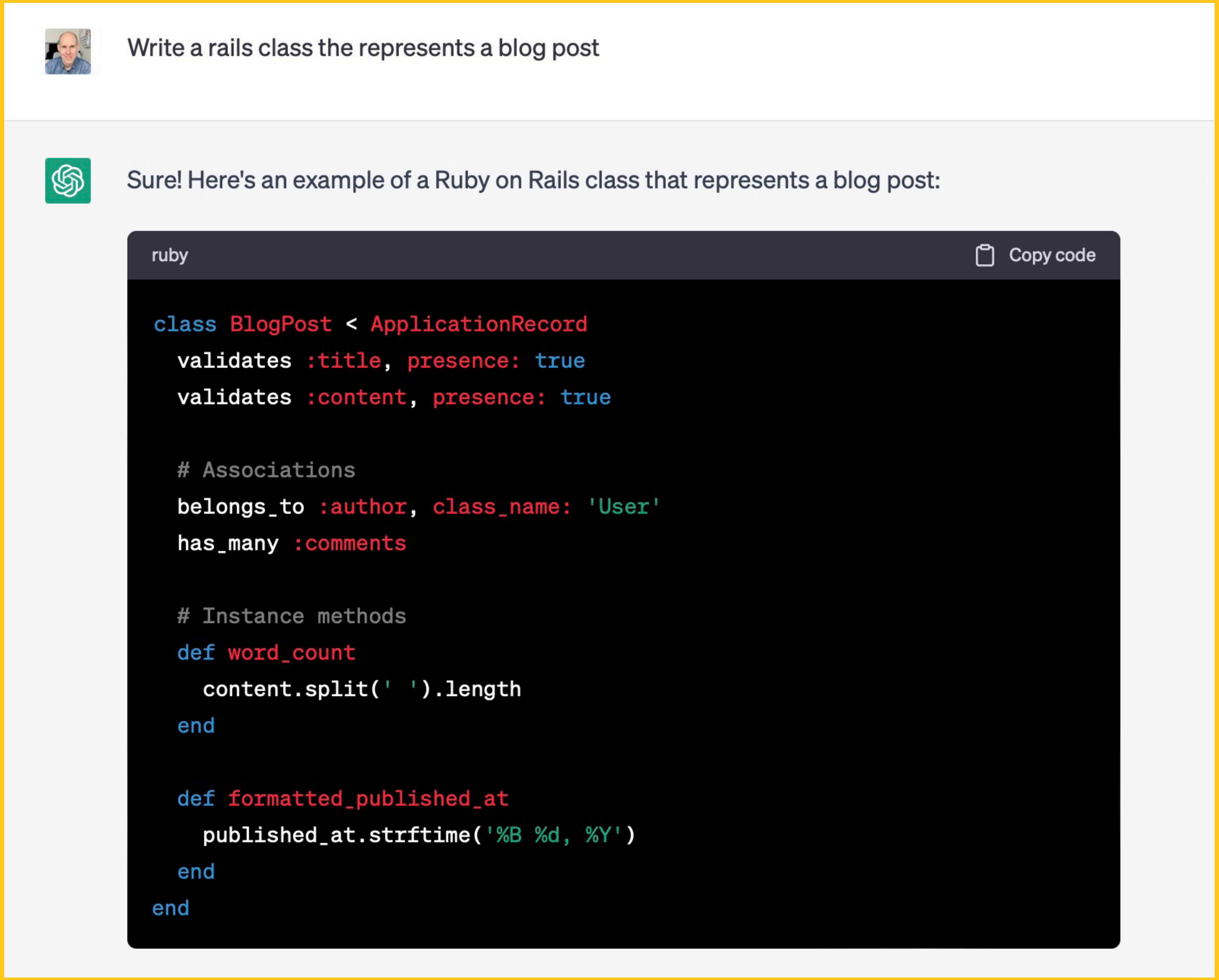


Write a ruby method that loops over an array of strings and returns the longest one

Sure! Here's a Ruby method that takes an array of strings as input, loops over each string, and returns the longest one:

```ruby
def find_longest_string(arr)
  longest_string = ''
  arr.each do |str|
    if str.length > longest_string.length
      longest_string = str
    end
  end
  return longest_string
end
```

CREATING A METHOD

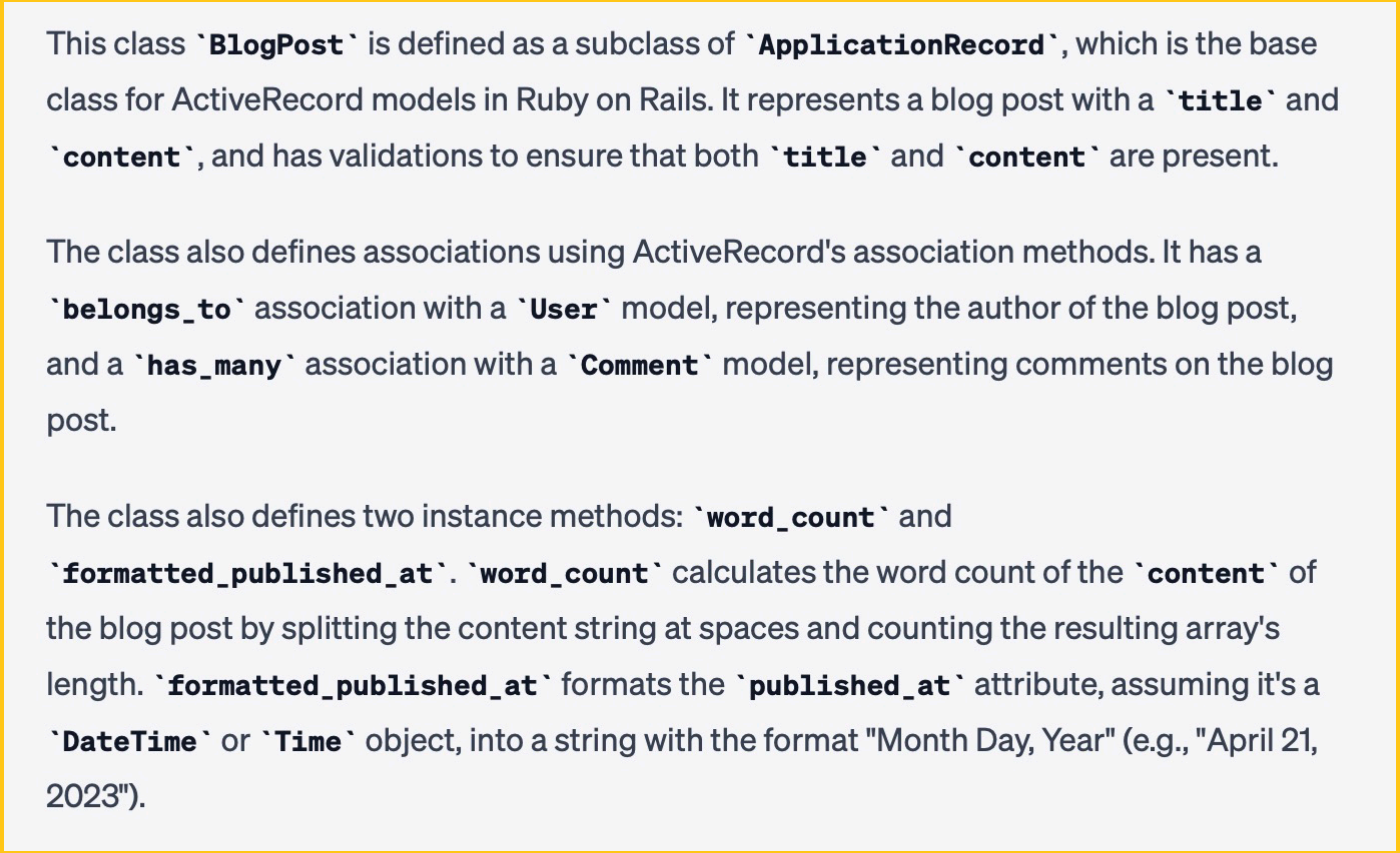


Write a rails class the represents a blog post

Sure! Here's an example of a Ruby on Rails class that represents a blog post:

```ruby
class BlogPost < ApplicationRecord
  validates :title, presence: true
  validates :content, presence: true

  # Associations
  belongs_to :author, class_name: 'User'
  has_many :comments

  # Instance methods
  def word_count
    content.split(' ').length
  end

  def formatted_published_at
    published_at.strftime('%B %d, %Y')
  end
end
```

CREATING A CLASS



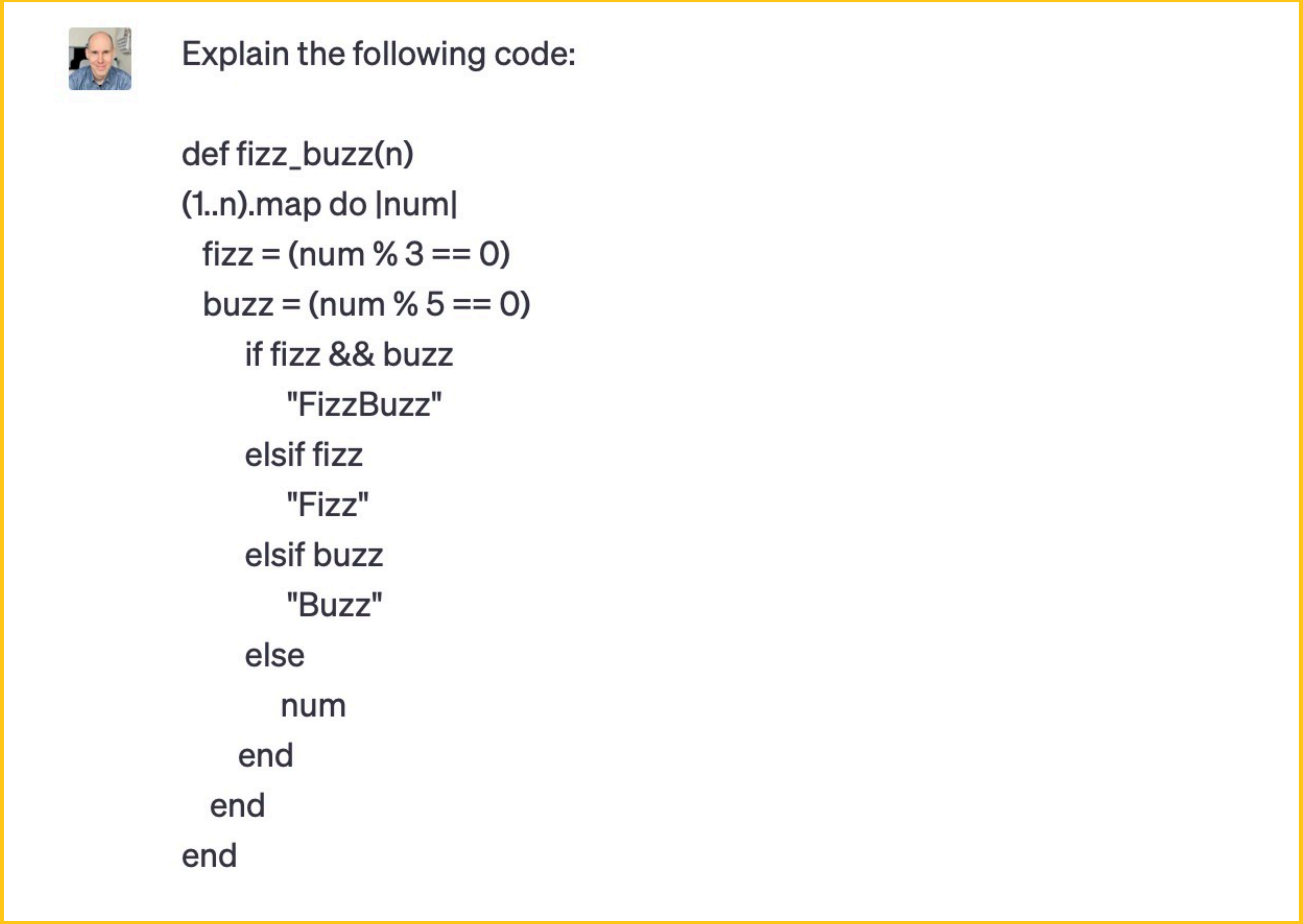
This class `BlogPost` is defined as a subclass of `ApplicationRecord`, which is the base class for ActiveRecord models in Ruby on Rails. It represents a blog post with a `title` and `content`, and has validations to ensure that both `title` and `content` are present.

The class also defines associations using ActiveRecord's association methods. It has a `belongs_to` association with a `User` model, representing the author of the blog post, and a `has_many` association with a `Comment` model, representing comments on the blog post.

The class also defines two instance methods: `word_count` and `formatted_published_at`. `word_count` calculates the word count of the `content` of the blog post by splitting the content string at spaces and counting the resulting array's length. `formatted_published_at` formats the `published_at` attribute, assuming it's a `DateTime` or `Time` object, into a string with the format "Month Day, Year" (e.g., "April 21, 2023").

EXPLAINING CODE

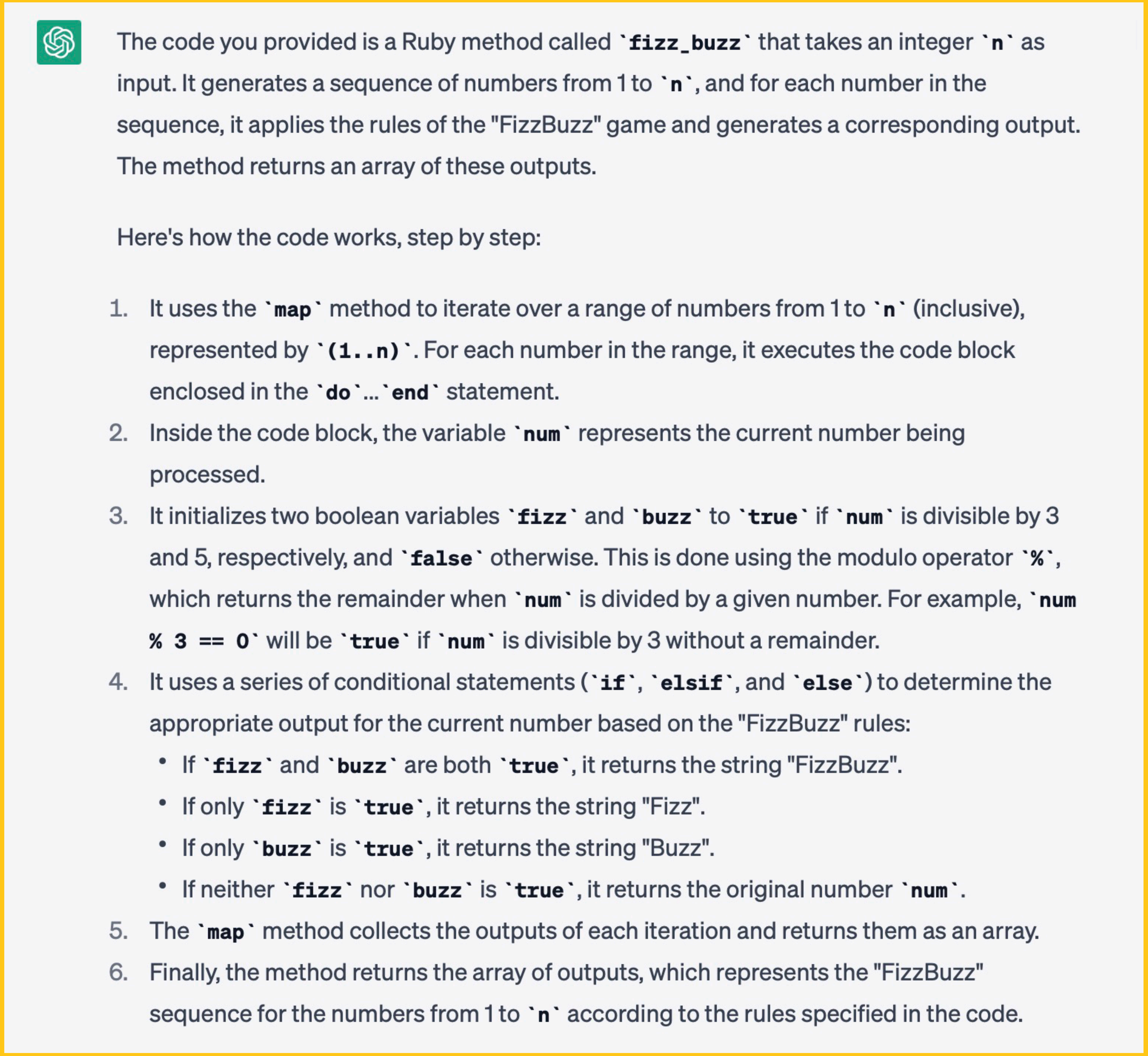
EXPLAINING CODE



Explain the following code:

```
def fizz_buzz(n)
(1..n).map do |num|
  fizz = (num % 3 == 0)
  buzz = (num % 5 == 0)
      if fizz && buzz
          "FizzBuzz"
      elsif fizz
          "Fizz"
      elsif buzz
          "Buzz"
      else
         num
     end
   end
end
```

EXPLAINING FIZZBUZZ

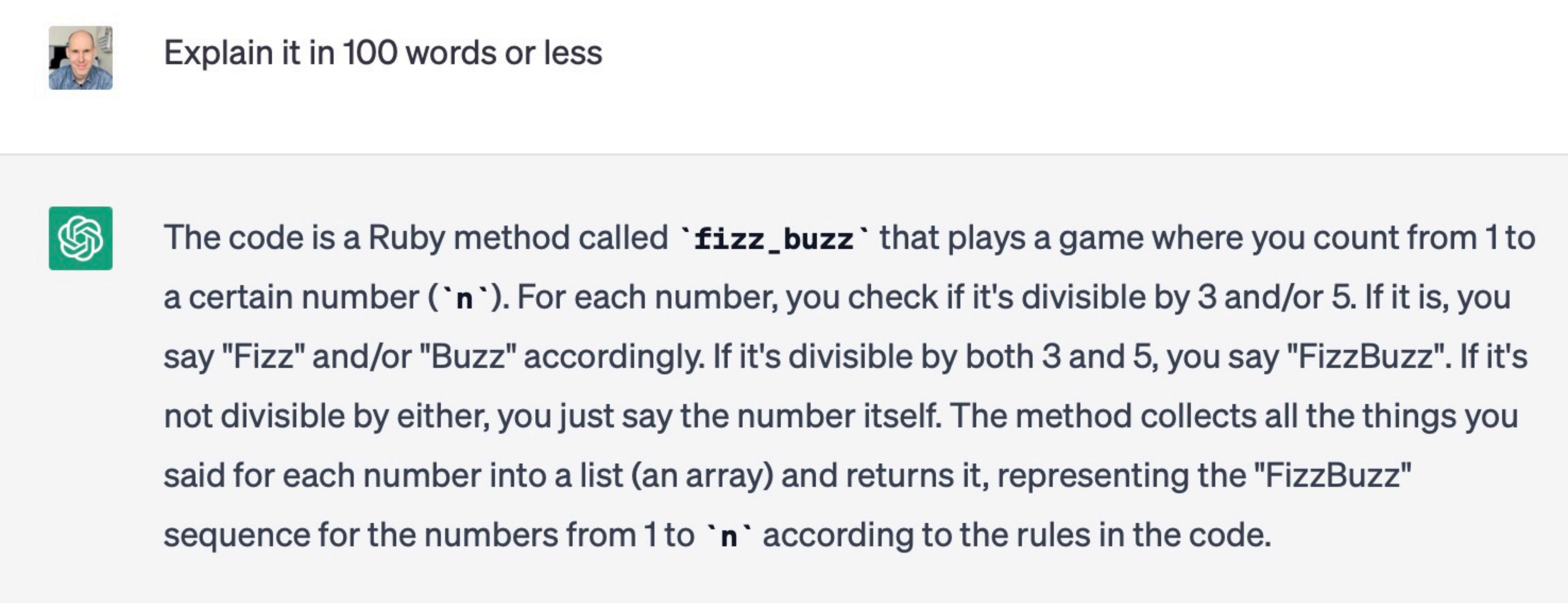


The code you provided is a Ruby method called `fizz_buzz` that takes an integer `n` as input. It generates a sequence of numbers from 1 to `n`, and for each number in the sequence, it applies the rules of the "FizzBuzz" game and generates a corresponding output. The method returns an array of these outputs.

Here's how the code works, step by step:

1. It uses the `map` method to iterate over a range of numbers from 1 to `n` (inclusive), represented by `(1..n)`. For each number in the range, it executes the code block enclosed in the `do`...`end` statement.
2. Inside the code block, the variable `num` represents the current number being processed.
3. It initializes two boolean variables `fizz` and `buzz` to `true` if `num` is divisible by 3 and 5, respectively, and `false` otherwise. This is done using the modulo operator `%`, which returns the remainder when `num` is divided by a given number. For example, `num % 3 == 0` will be `true` if `num` is divisible by 3 without a remainder.
4. It uses a series of conditional statements (`if`, `elsif`, and `else`) to determine the appropriate output for the current number based on the "FizzBuzz" rules:
    - If `fizz` and `buzz` are both `true`, it returns the string "FizzBuzz".
    - If only `fizz` is `true`, it returns the string "Fizz".
    - If only `buzz` is `true`, it returns the string "Buzz".
    - If neither `fizz` nor `buzz` is `true`, it returns the original number `num`.
5. The `map` method collects the outputs of each iteration and returns them as an array.
6. Finally, the method returns the array of outputs, which represents the "FizzBuzz" sequence for the numbers from 1 to `n` according to the rules specified in the code.

EXPLAINING FIZZBUZZ



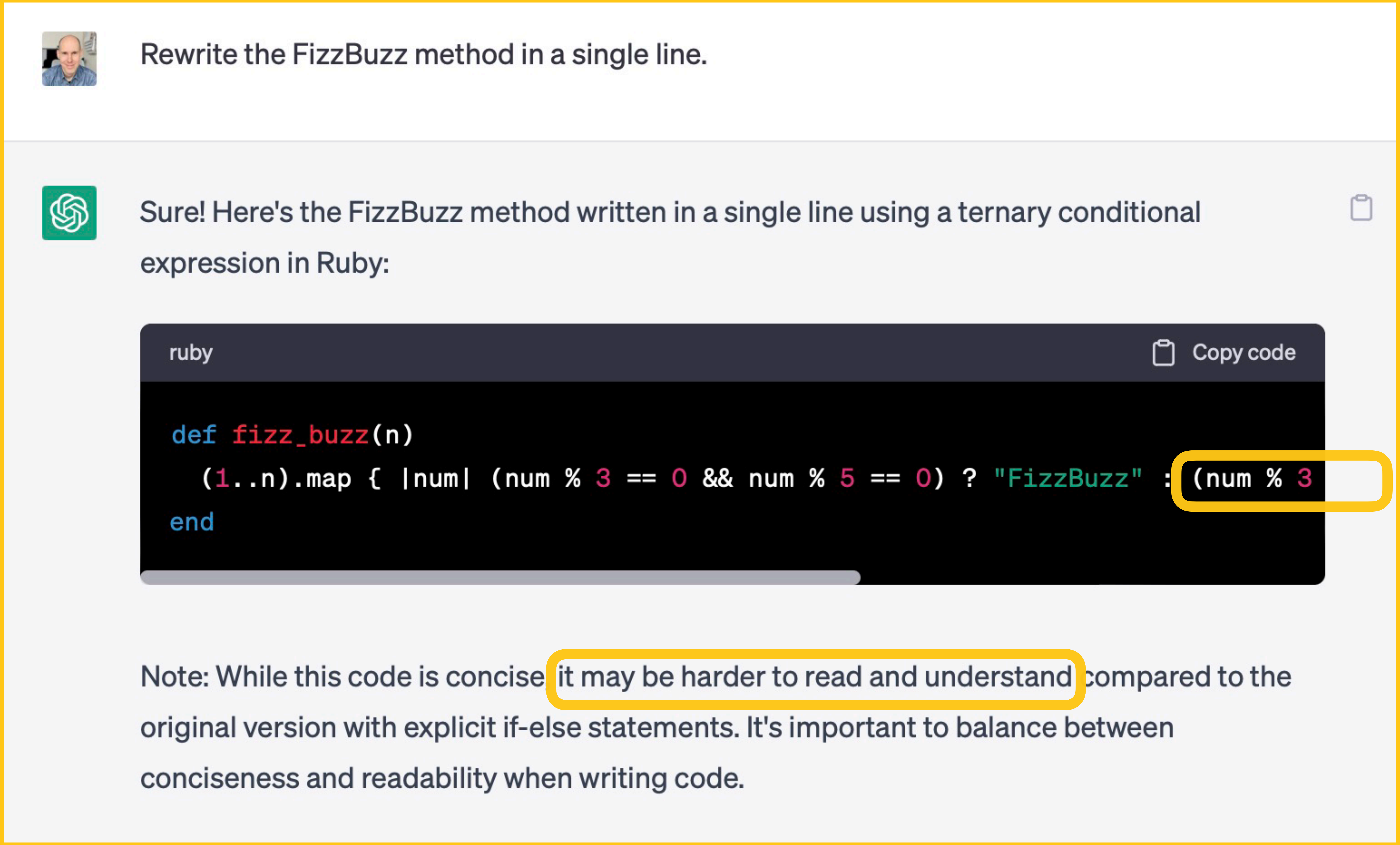
Explain it in 100 words or less

The code is a Ruby method called `` `fizz_buzz` `` that plays a game where you count from 1 to a certain number (`` `n` ``). For each number, you check if it's divisible by 3 and/or 5. If it is, you say "Fizz" and/or "Buzz" accordingly. If it's divisible by both 3 and 5, you say "FizzBuzz". If it's not divisible by either, you just say the number itself. The method collects all the things you said for each number into a list (an array) and returns it, representing the "FizzBuzz" sequence for the numbers from 1 to `` `n` `` according to the rules in the code.

EXPLAINING FIZZBUZZ

REFACTOING CODE

REWRITING FIZZBUZZ

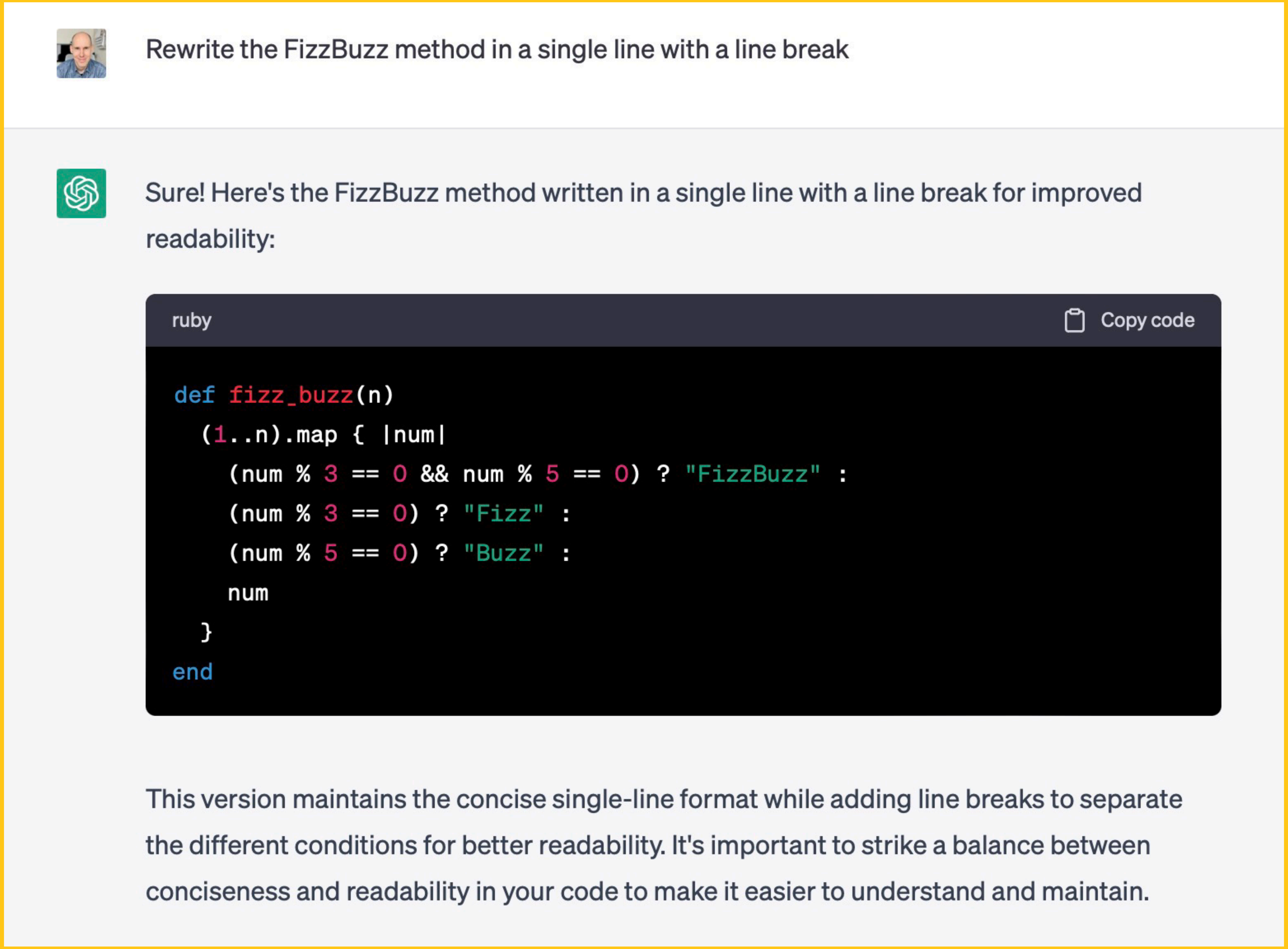


Rewrite the FizzBuzz method in a single line.

Sure! Here's the FizzBuzz method written in a single line using a ternary conditional expression in Ruby:

```ruby
def fizz_buzz(n)
  (1..n).map { |num| (num % 3 == 0 && num % 5 == 0) ? "FizzBuzz" : (num % 3
end
```

Note: While this code is concise it may be harder to read and understand compared to the original version with explicit if-else statements. It's important to balance between conciseness and readability when writing code.



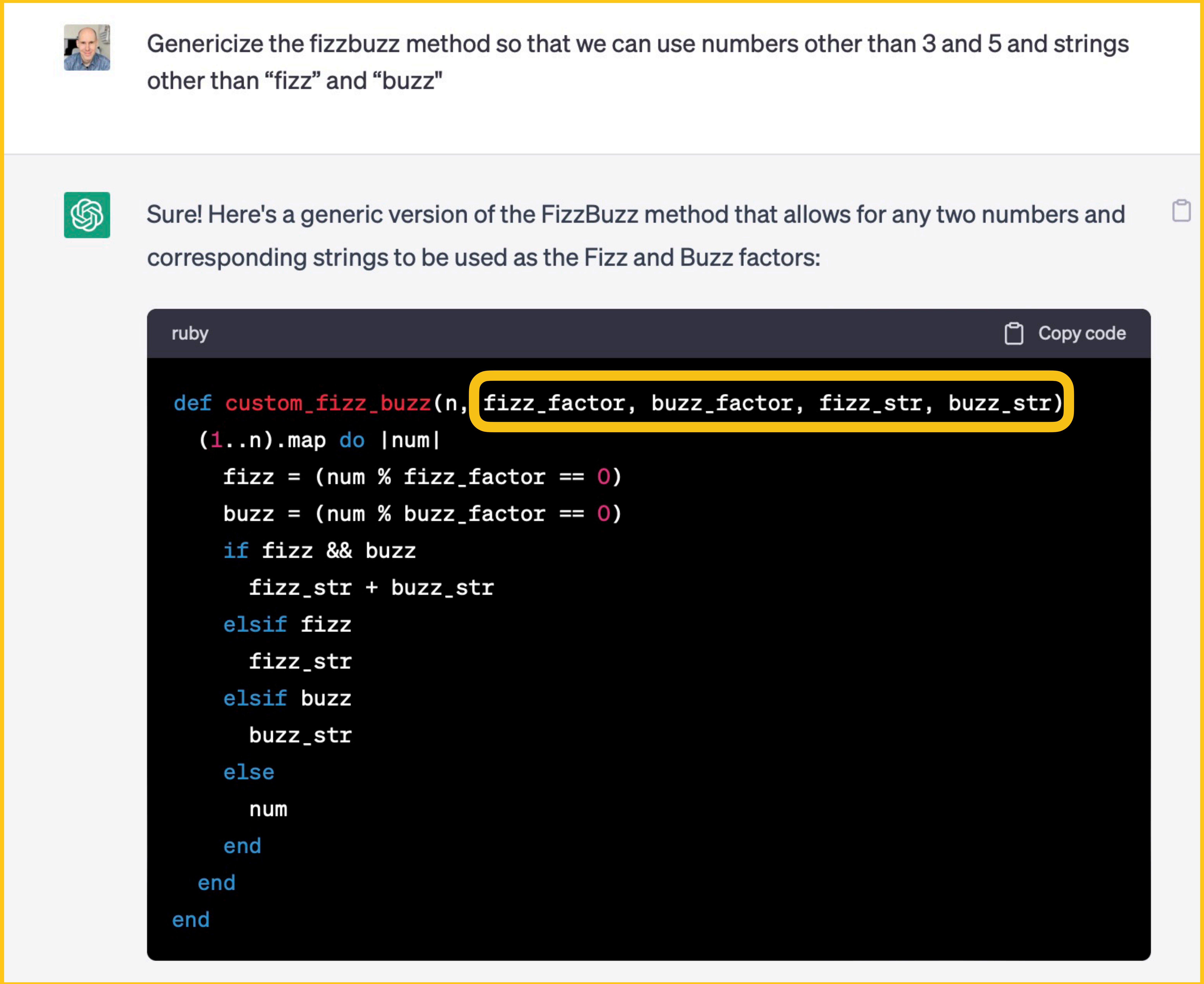
Rewrite the FizzBuzz method in a single line with a line break

Sure! Here's the FizzBuzz method written in a single line with a line break for improved readability:

```ruby
def fizz_buzz(n)
  (1..n).map { |num|
    (num % 3 == 0 && num % 5 == 0) ? "FizzBuzz" :
    (num % 3 == 0) ? "Fizz" :
    (num % 5 == 0) ? "Buzz" :
    num
  }
end
```

This version maintains the concise single-line format while adding line breaks to separate the different conditions for better readability. It's important to strike a balance between conciseness and readability in your code to make it easier to understand and maintain.

REWRITING FIZZBUZZ



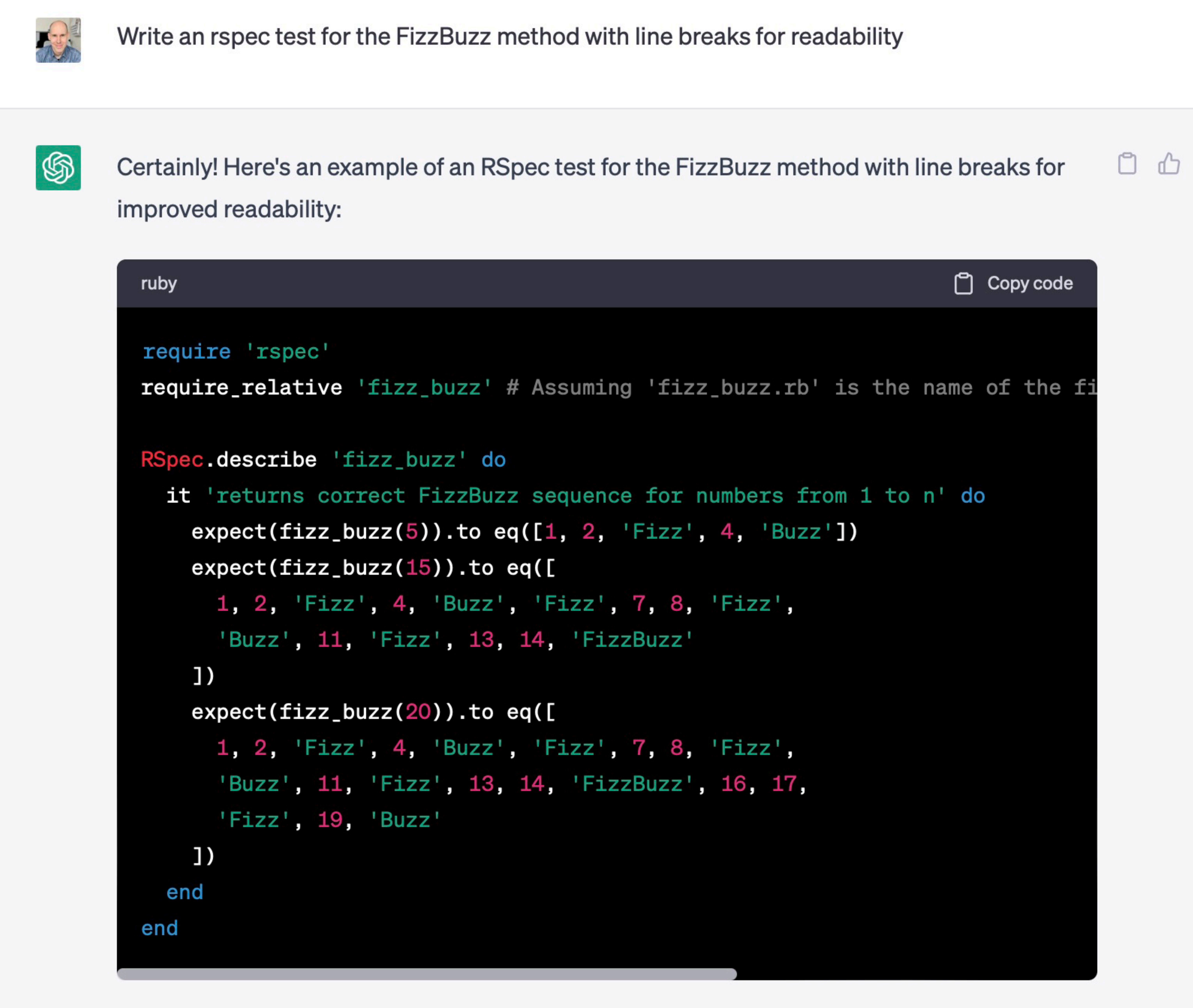
Genericize the fizzbuzz method so that we can use numbers other than 3 and 5 and strings other than “fizz” and “buzz"

Sure! Here's a generic version of the FizzBuzz method that allows for any two numbers and corresponding strings to be used as the Fizz and Buzz factors:

```ruby
def custom_fizz_buzz(n, fizz_factor, buzz_factor, fizz_str, buzz_str)
  (1..n).map do |num|
    fizz = (num % fizz_factor == 0)
    buzz = (num % buzz_factor == 0)
    if fizz && buzz
      fizz_str + buzz_str
    elsif fizz
      fizz_str
    elsif buzz
      buzz_str
    else
      num
    end
  end
end
```

REFACTORING FIZZBUZZ

WRITING TESTS



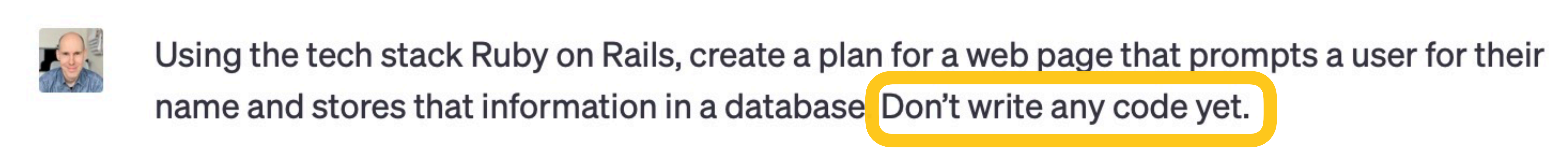
Write an rspec test for the FizzBuzz method with line breaks for readability

Certainly! Here's an example of an RSpec test for the FizzBuzz method with line breaks for improved readability:

```ruby
require 'rspec'
require_relative 'fizz_buzz' # Assuming 'fizz_buzz.rb' is the name of the fi

RSpec.describe 'fizz_buzz' do
  it 'returns correct FizzBuzz sequence for numbers from 1 to n' do
    expect(fizz_buzz(5)).to eq([1, 2, 'Fizz', 4, 'Buzz'])
    expect(fizz_buzz(15)).to eq([
      1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz',
      'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz'
    ])
    expect(fizz_buzz(20)).to eq([
      1, 2, 'Fizz', 4, 'Buzz', 'Fizz', 7, 8, 'Fizz',
      'Buzz', 11, 'Fizz', 13, 14, 'FizzBuzz', 16, 17,
      'Fizz', 19, 'Buzz'
    ])
  end
end
```

WRITING A TEST FOR FIZZBUZZ

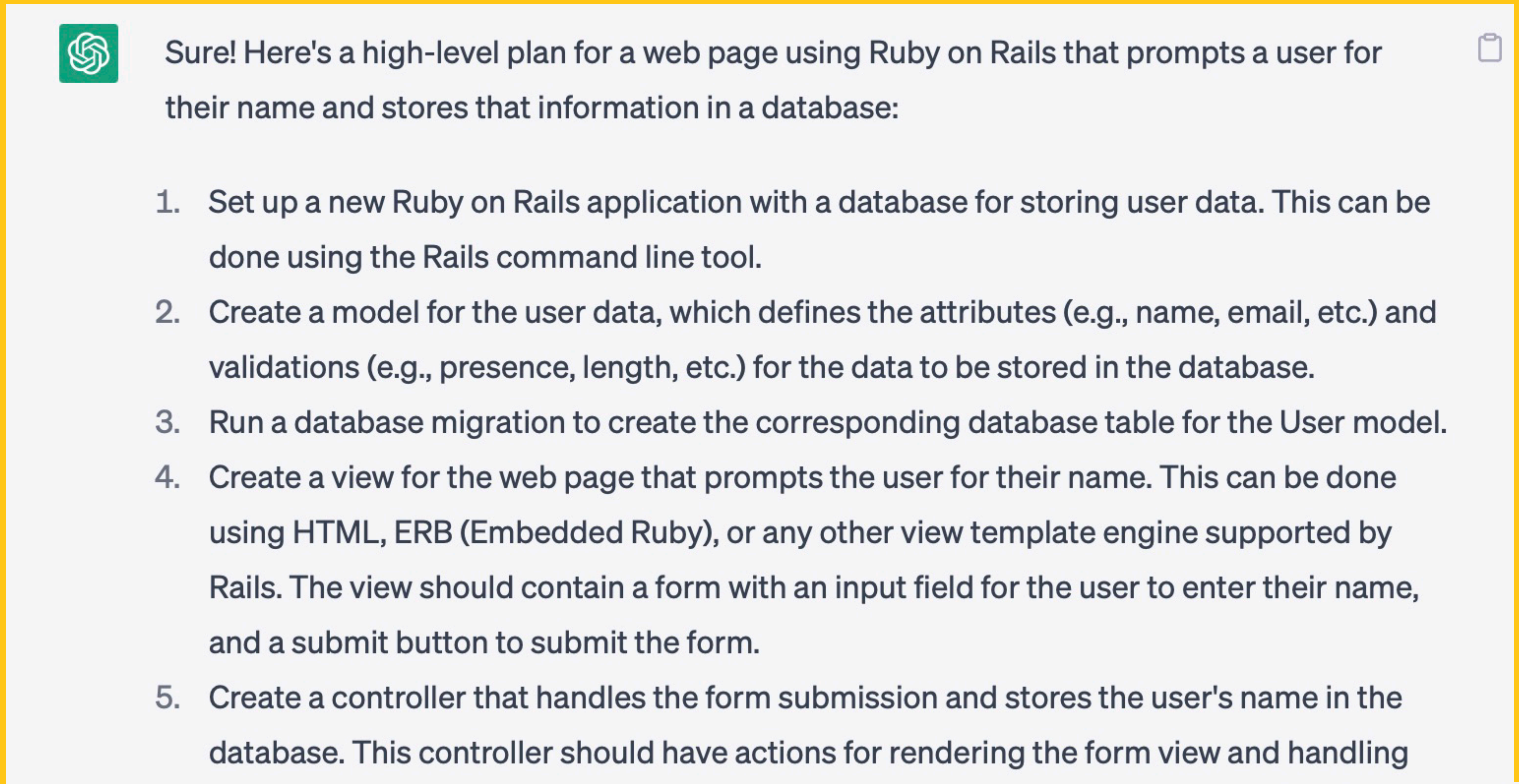
PLANNING



Using the tech stack Ruby on Rails, create a plan for a web page that prompts a user for their name and stores that information in a database Don’t write any code yet.

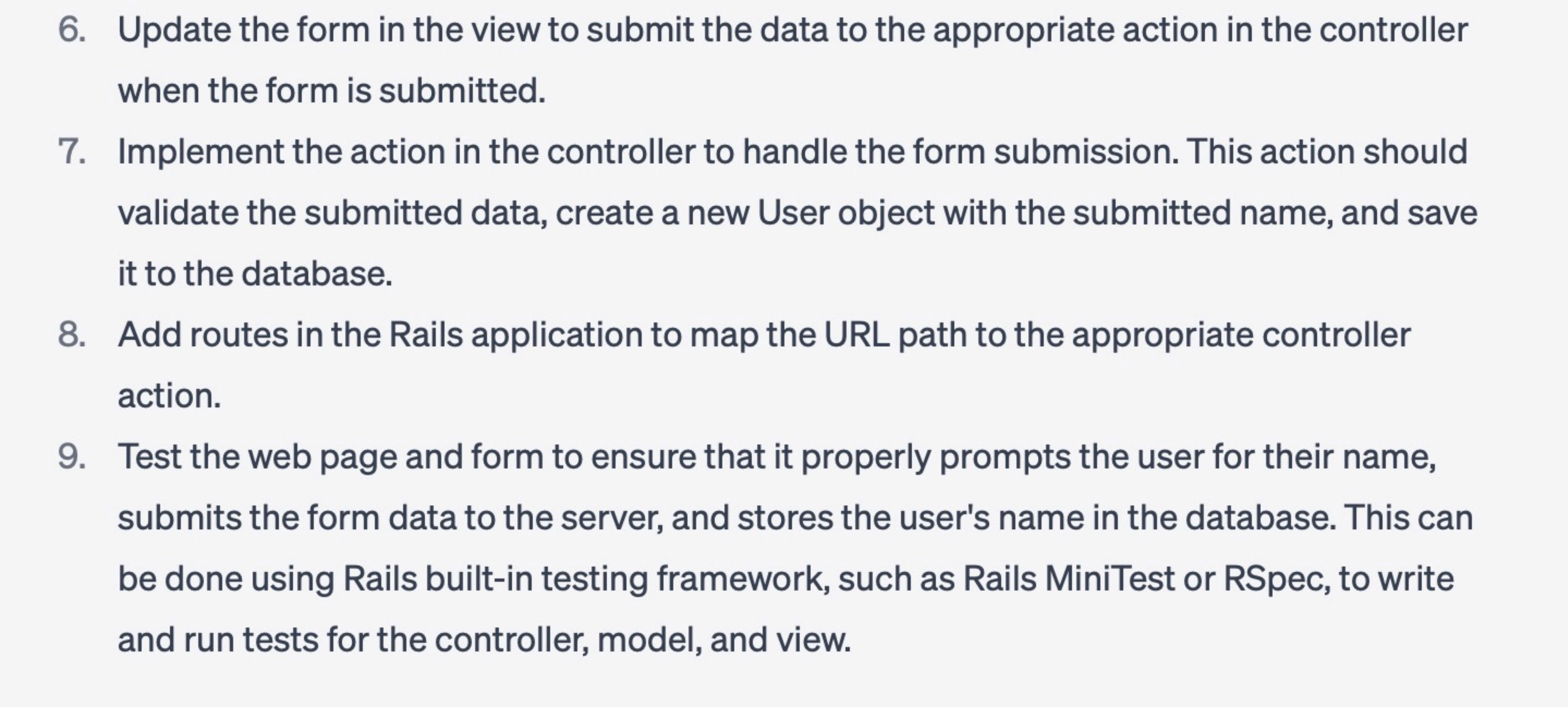
CREATING A CODE PLAN

CREATING A CODE PLAN

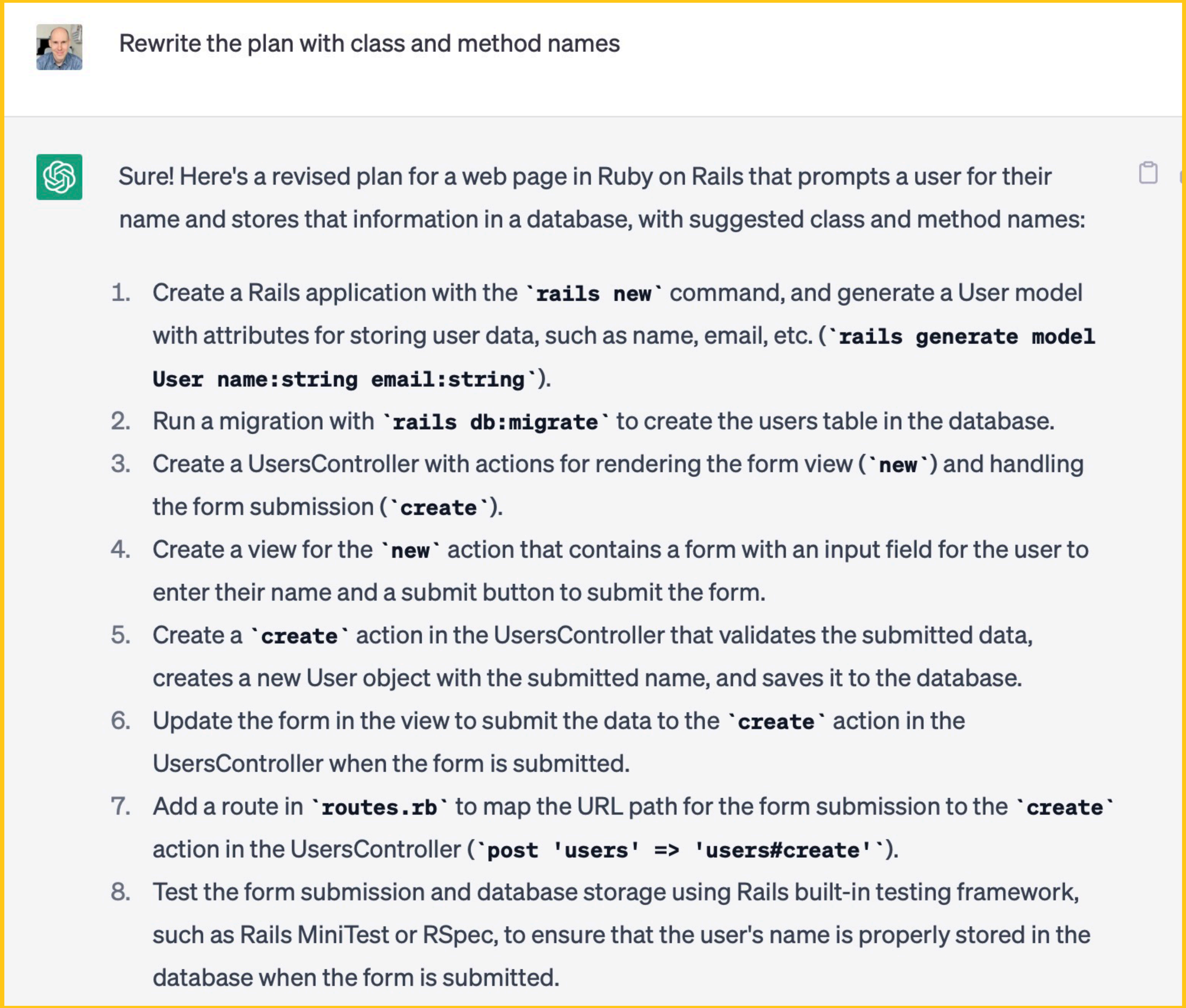


Sure! Here's a high-level plan for a web page using Ruby on Rails that prompts a user for their name and stores that information in a database:

1. Set up a new Ruby on Rails application with a database for storing user data. This can be done using the Rails command line tool.
2. Create a model for the user data, which defines the attributes (e.g., name, email, etc.) and validations (e.g., presence, length, etc.) for the data to be stored in the database.
3. Run a database migration to create the corresponding database table for the User model.
4. Create a view for the web page that prompts the user for their name. This can be done using HTML, ERB (Embedded Ruby), or any other view template engine supported by Rails. The view should contain a form with an input field for the user to enter their name, and a submit button to submit the form.
5. Create a controller that handles the form submission and stores the user's name in the database. This controller should have actions for rendering the form view and handling the form submission.



6. Update the form in the view to submit the data to the appropriate action in the controller when the form is submitted.
7. Implement the action in the controller to handle the form submission. This action should validate the submitted data, create a new User object with the submitted name, and save it to the database.
8. Add routes in the Rails application to map the URL path to the appropriate controller action.
9. Test the web page and form to ensure that it properly prompts the user for their name, submits the form data to the server, and stores the user's name in the database. This can be done using Rails built-in testing framework, such as Rails MiniTest or RSpec, to write and run tests for the controller, model, and view.

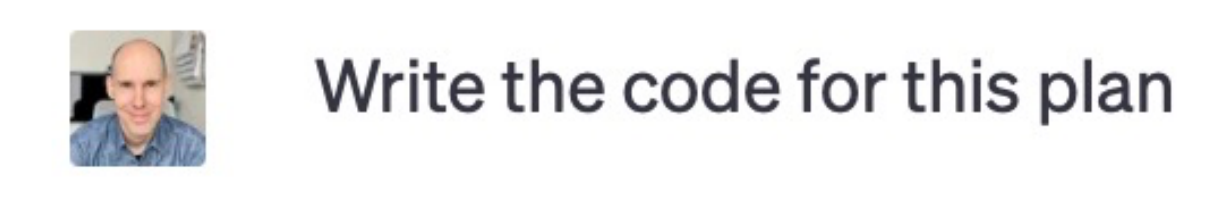


Rewrite the plan with class and method names

Sure! Here's a revised plan for a web page in Ruby on Rails that prompts a user for their name and stores that information in a database, with suggested class and method names:

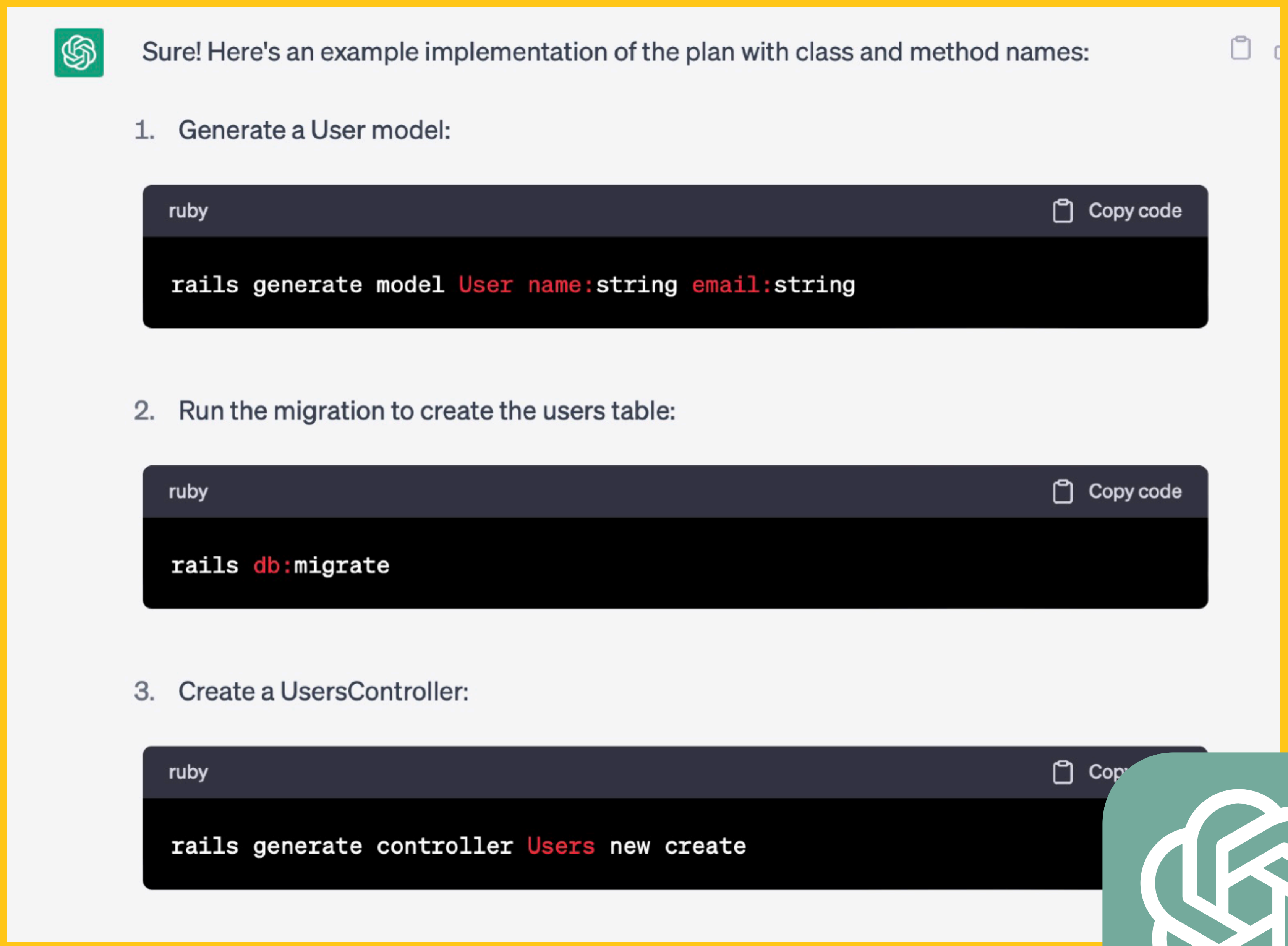
1. Create a Rails application with the `rails new` command, and generate a User model with attributes for storing user data, such as name, email, etc. (`rails generate model User name:string email:string`).
2. Run a migration with `rails db:migrate` to create the users table in the database.
3. Create a UsersController with actions for rendering the form view (`new`) and handling the form submission (`create`).
4. Create a view for the `new` action that contains a form with an input field for the user to enter their name and a submit button to submit the form.
5. Create a `create` action in the UsersController that validates the submitted data, creates a new User object with the submitted name, and saves it to the database.
6. Update the form in the view to submit the data to the `create` action in the UsersController when the form is submitted.
7. Add a route in `routes.rb` to map the URL path for the form submission to the `create` action in the UsersController (`post 'users' => 'users#create'`).
8. Test the form submission and database storage using Rails built-in testing framework, such as Rails MiniTest or RSpec, to ensure that the user's name is properly stored in the database when the form is submitted.

REWRITING THE CODE PLAN



Write the code for this plan

WRITING CODE FROM THE PLAN



Sure! Here's an example implementation of the plan with class and method names:

1. Generate a User model:

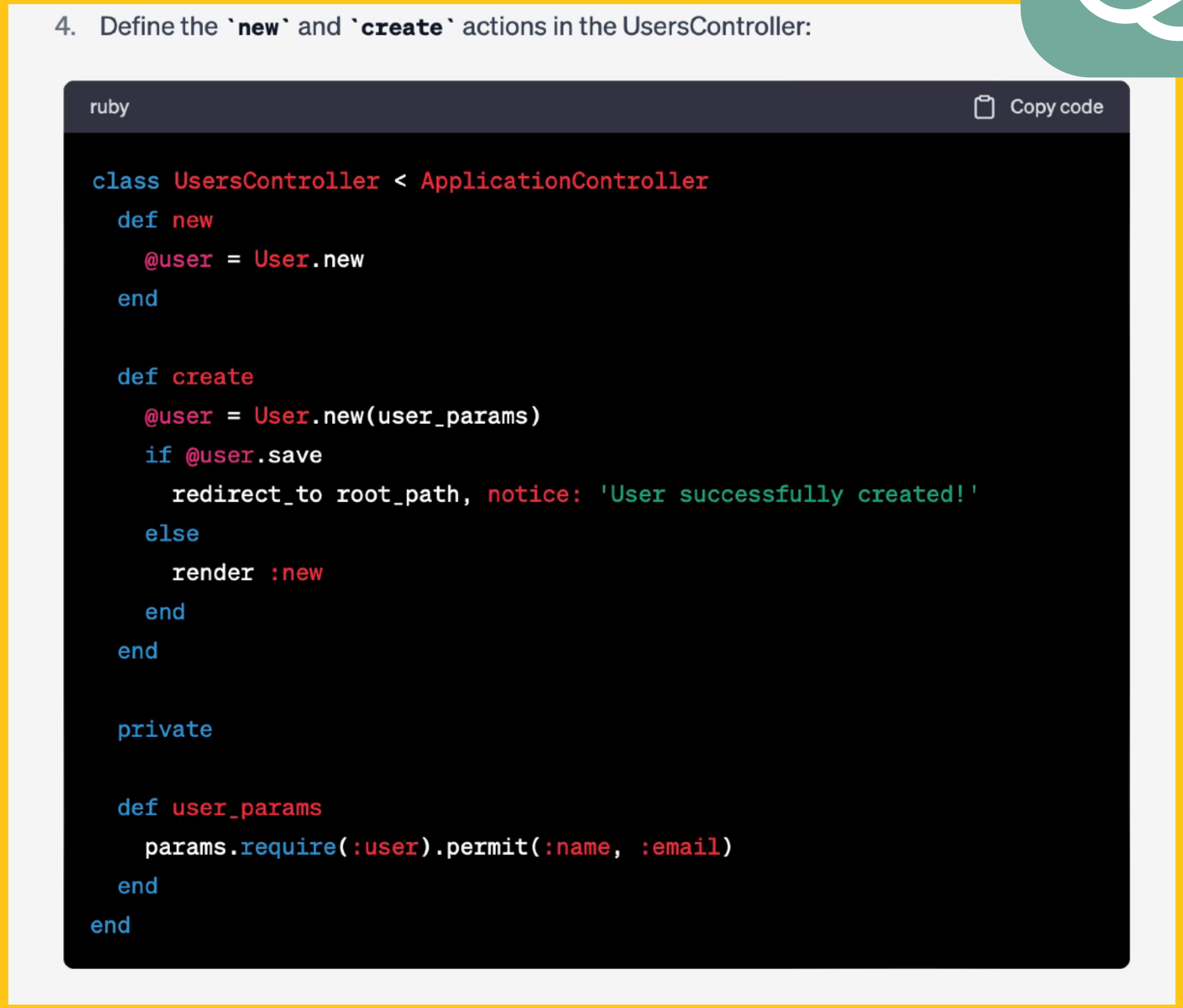
```ruby
rails generate model User name:string email:string
```

2. Run the migration to create the users table:

```ruby
rails db:migrate
```

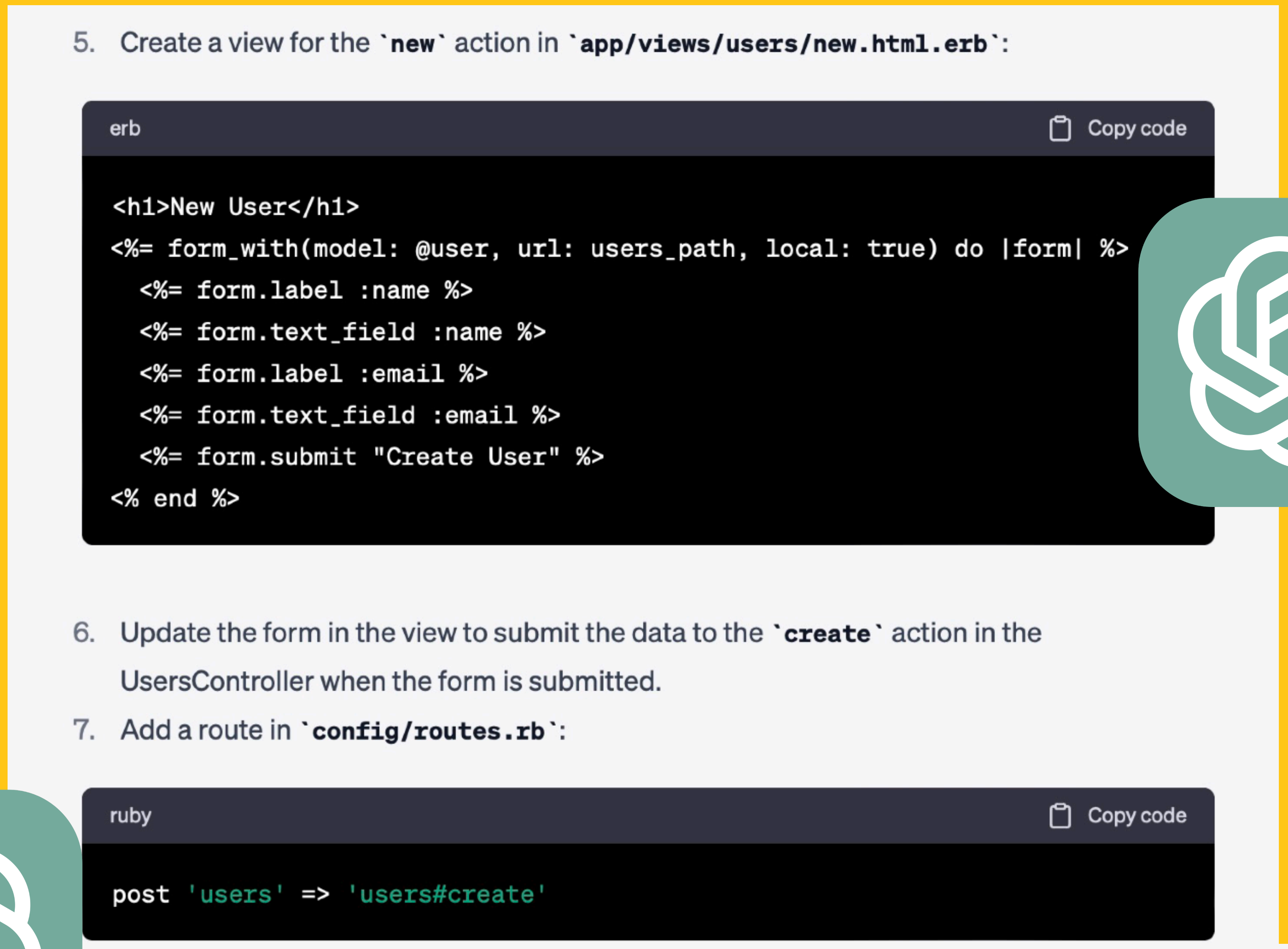
3. Create a UsersController:

```ruby
rails generate controller Users new create
```



4. Define the `new` and `create` actions in the UsersController:

```ruby
class UsersController < ApplicationController
  def new
    @user = User.new
  end

  def create
    @user = User.new(user_params)
    if @user.save
      redirect_to root_path, notice: 'User successfully created!'
    else
      render :new
    end
  end

  private

  def user_params
    params.require(:user).permit(:name, :email)
  end
end
```

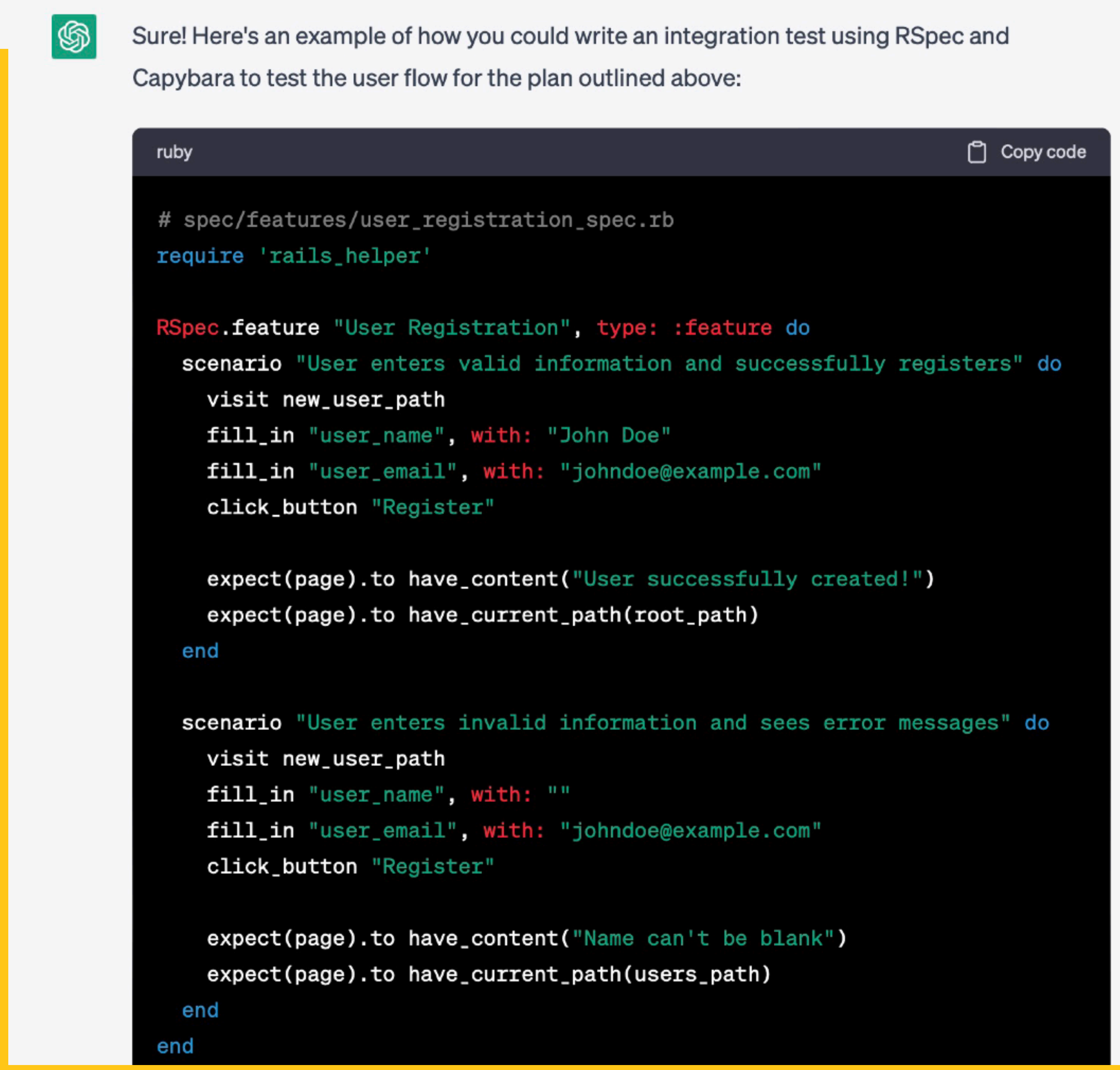


5. Create a view for the `new` action in `app/views/users/new.html.erb`:

```erb
<h1>New User</h1>
<%= form_with(model: @user, url: users_path, local: true) do |form| %>
  <%= form.label :name %>
  <%= form.text_field :name %>
  <%= form.label :email %>
  <%= form.text_field :email %>
  <%= form.submit "Create User" %>
<% end %>
```

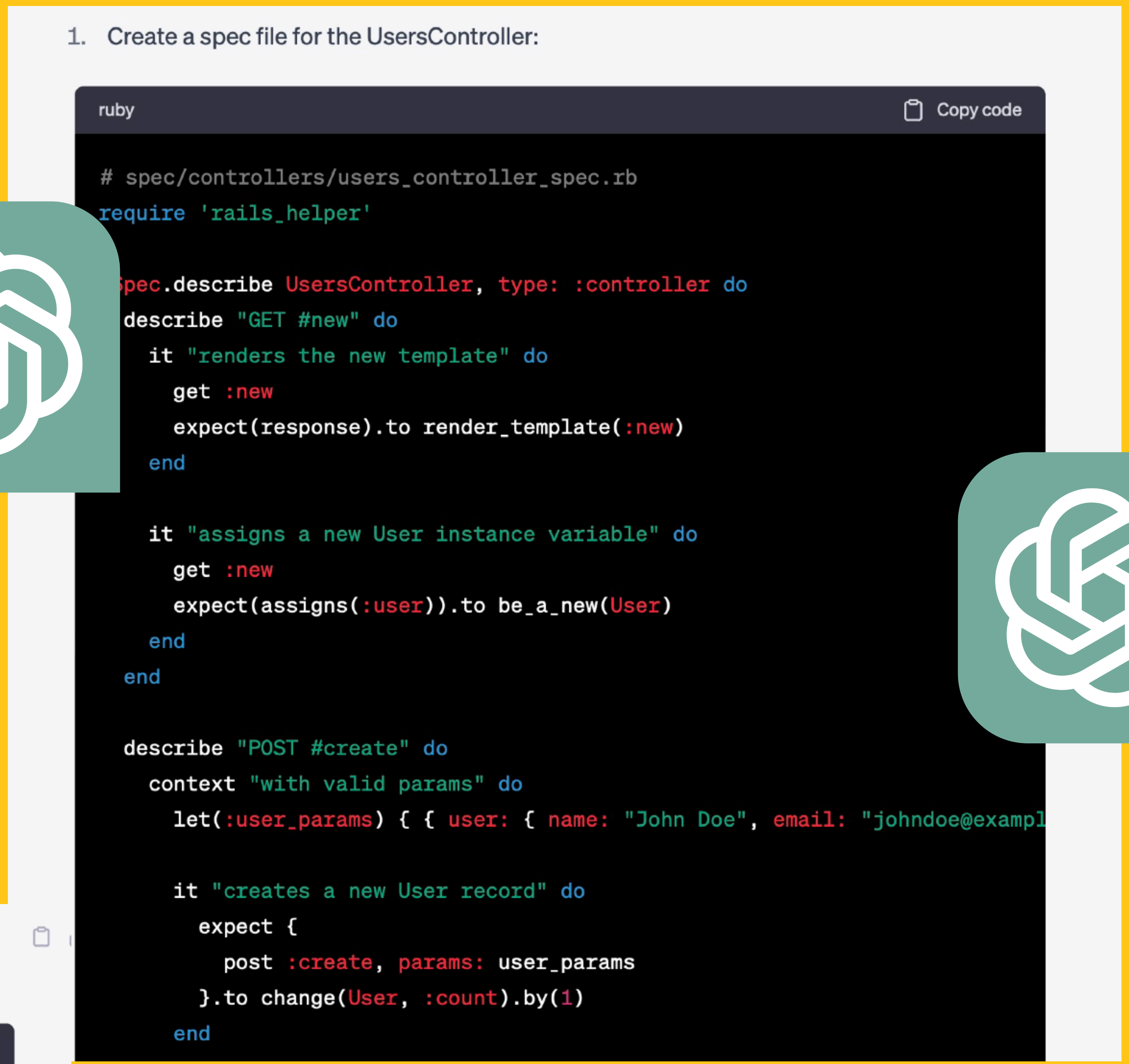
6. Update the form in the view to submit the data to the `create` action in the UsersController when the form is submitted.
7. Add a route in `config/routes.rb`:

```ruby
post 'users' => 'users#create'
```



Sure! Here's an example of how you could write an integration test using RSpec and Capybara to test the user flow for the plan outlined above:

```ruby
# spec/features/user_registration_spec.rb
require 'rails_helper'

RSpec.feature "User Registration", type: :feature do
  scenario "User enters valid information and successfully registers" do
    visit new_user_path
    fill_in "user_name", with: "John Doe"
    fill_in "user_email", with: "johndoe@example.com"
    click_button "Register"

    expect(page).to have_content("User successfully created!")
    expect(page).to have_current_path(root_path)
  end

  scenario "User enters invalid information and sees error messages" do
    visit new_user_path
    fill_in "user_name", with: ""
    fill_in "user_email", with: "johndoe@example.com"
    click_button "Register"

    expect(page).to have_content("Name can't be blank")
    expect(page).to have_current_path(users_path)
  end
end
```



1. Create a spec file for the UsersController:

```ruby
# spec/controllers/users_controller_spec.rb
require 'rails_helper'

pec.describe UsersController, type: :controller do
  describe "GET #new" do
    it "renders the new template" do
      get :new
      expect(response).to render_template(:new)
    end

    it "assigns a new User instance variable" do
      get :new
      expect(assigns(:user)).to be_a_new(User)
    end
  end

  describe "POST #create" do
    context "with valid params" do
      let(:user_params) { { user: { name: "John Doe", email: "johndoe@exampl

      it "creates a new User record" do
        expect {
          post :create, params: user_params
        }.to change(User, :count).by(1)
      end
```

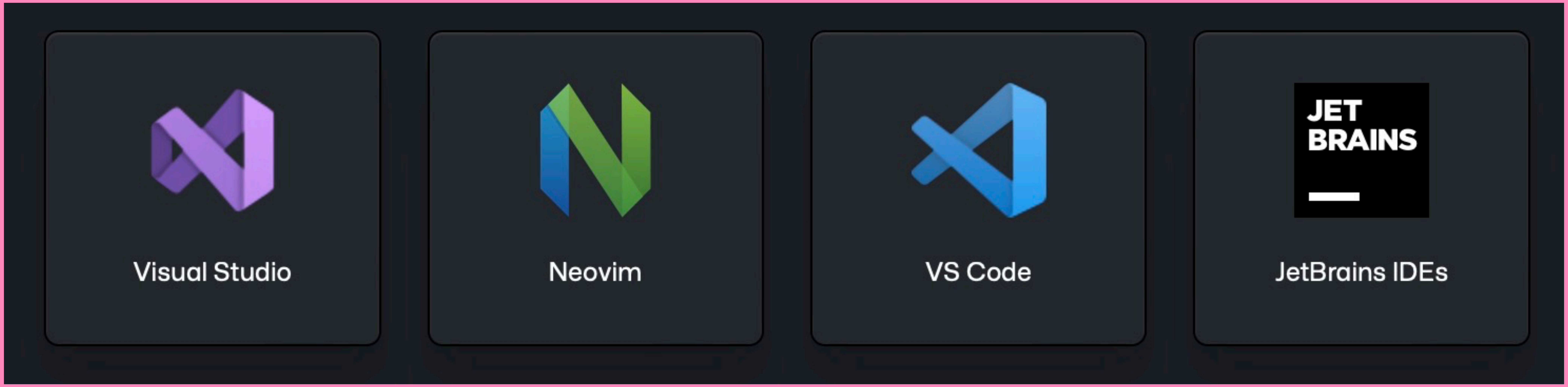
WRITING CODE FROM THE PLAN

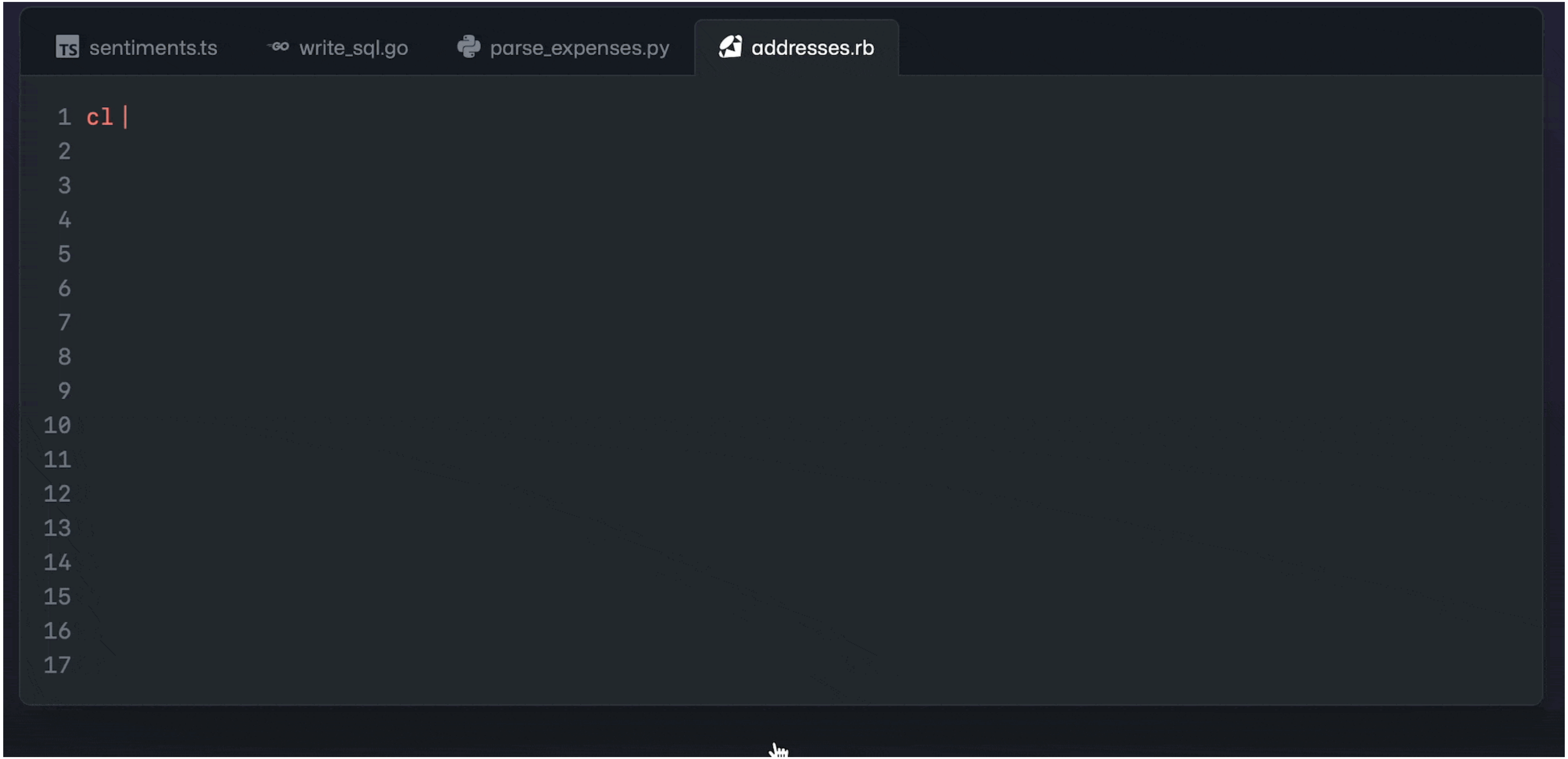
LIMITATIONS

- Incorrect code
- Unoptimized Code
- Expense
- Lack of context with your codebase
- Randomness

GITHUB COPILOT

SUPPORTED IDES



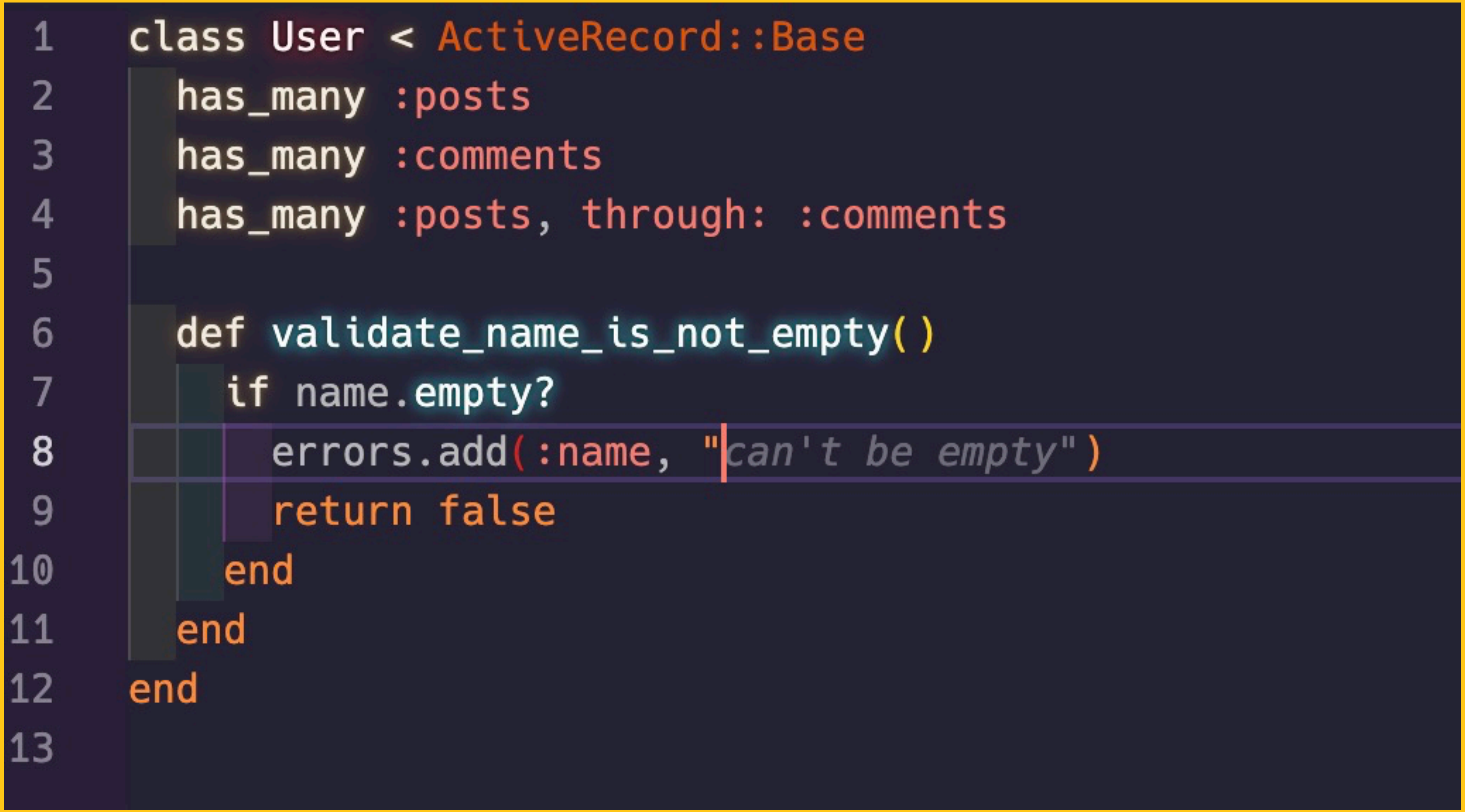


```
cl
```

WHAT'S IT GOOD FOR?

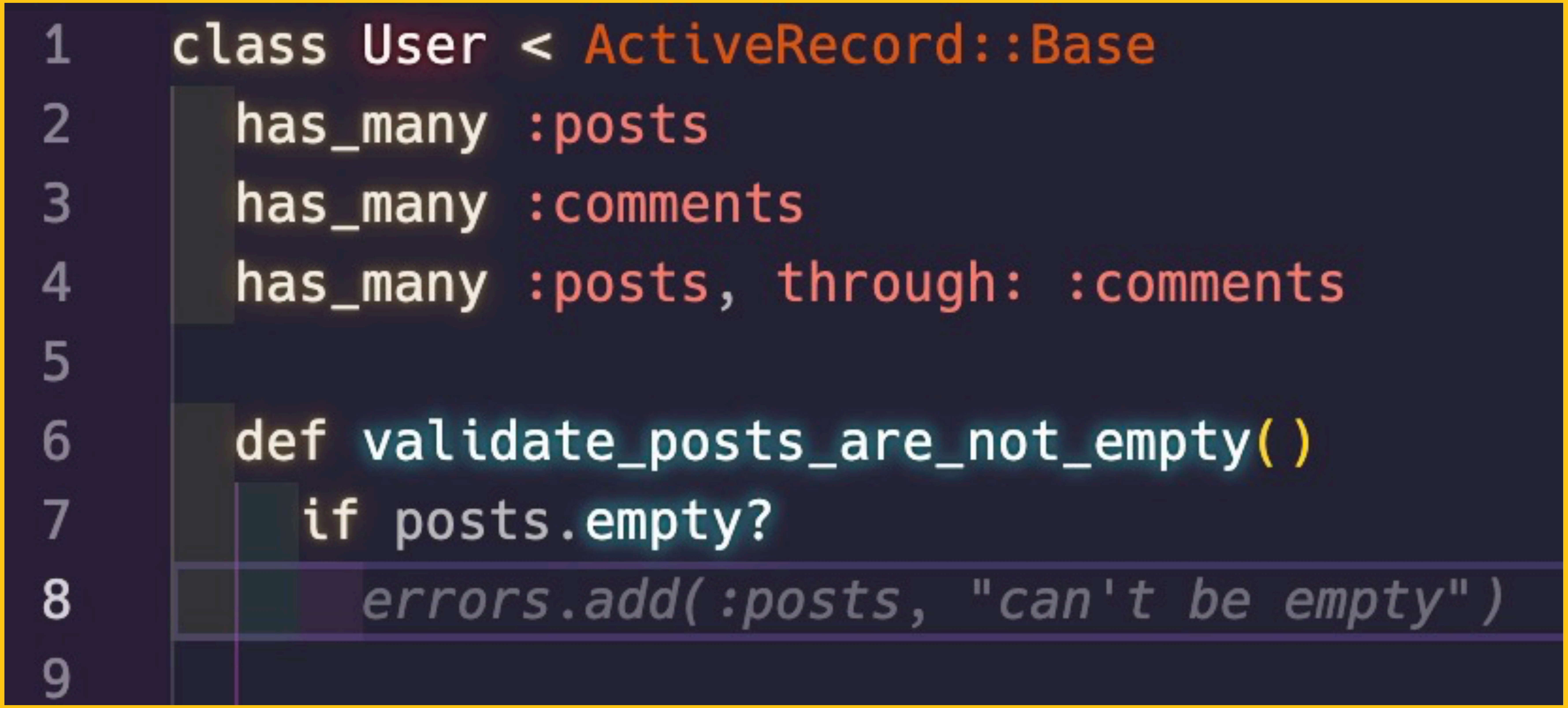
- Code completion
- Code generation
- Keeping context as you write code

WRITING CODE



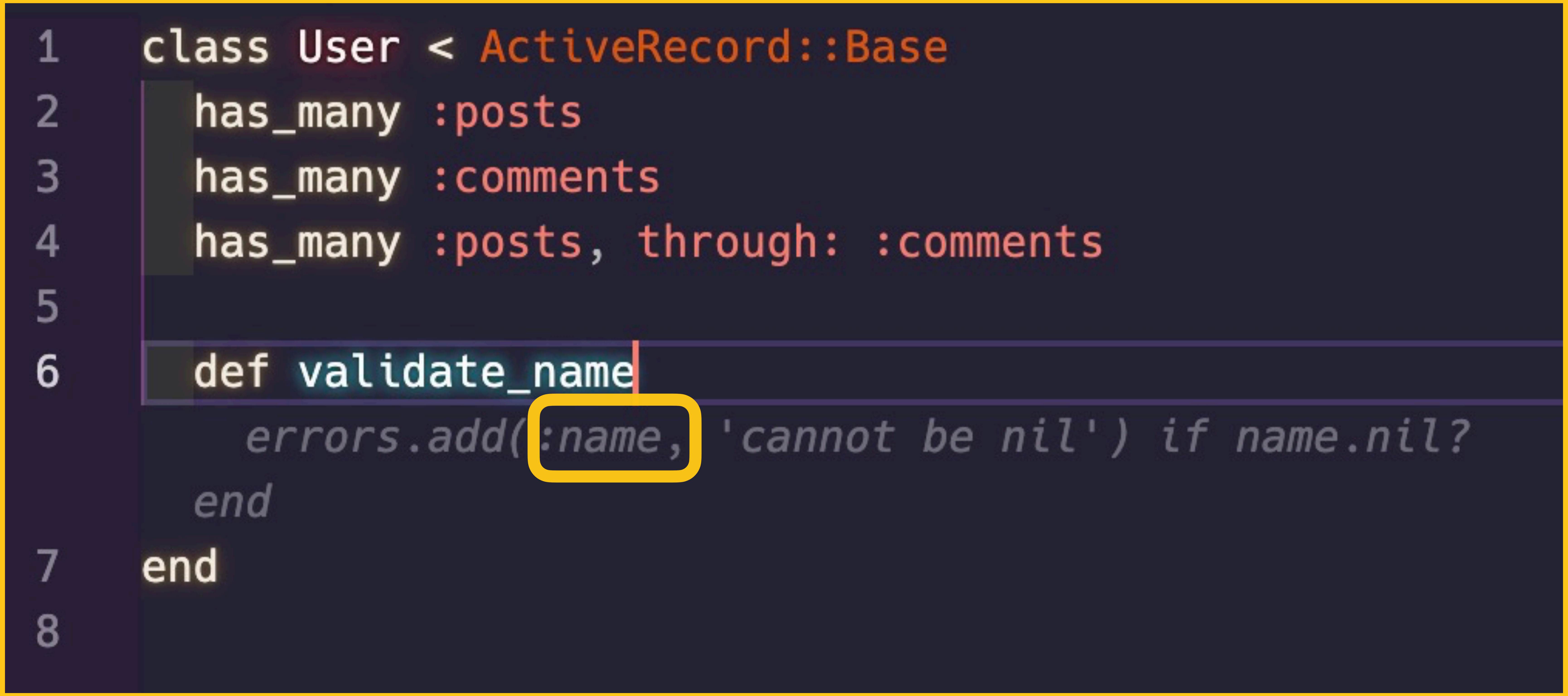
```
class User < ActiveRecord::Base
  has_many :posts
  has_many :comments
  has_many :posts, through: :comments

  def validate_name_is_not_empty()
    if name.empty?
      errors.add(:name, "can't be empty")
      return false
    end
  end
end

```

LINE COMPLETION



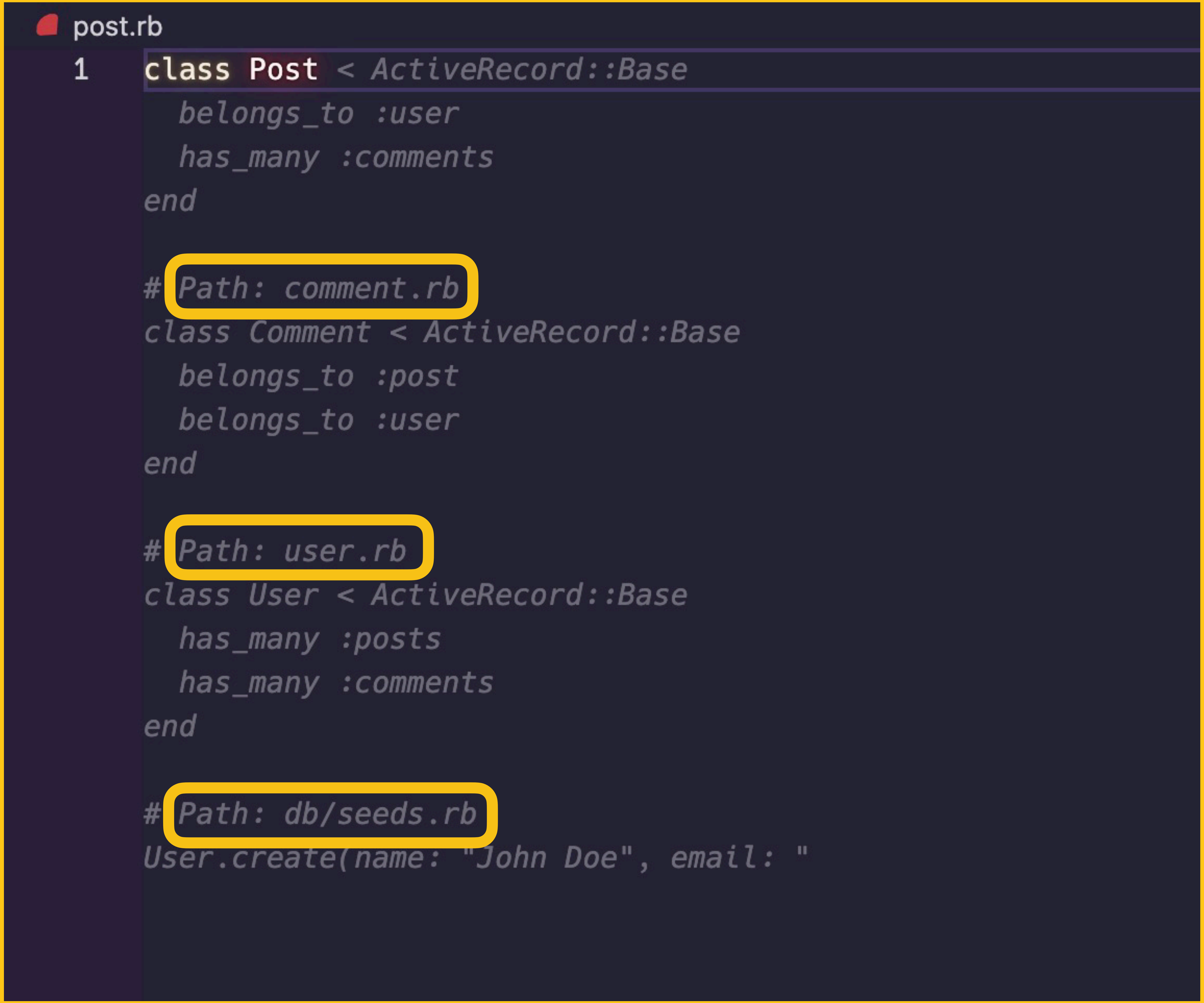
```
class User < ActiveRecord::Base
  has_many :posts
  has_many :comments
  has_many :posts, through: :comments

  def validate_posts_are_not_empty()
    if posts.empty?
      errors.add(:posts, "can't be empty")

```

LINE COMPLETION



```
class User < ActiveRecord::Base
  has_many :posts
  has_many :comments
  has_many :posts, through: :comments

  def validate_name
    errors.add(:name, 'cannot be nil') if name.nil?
  end
end

```

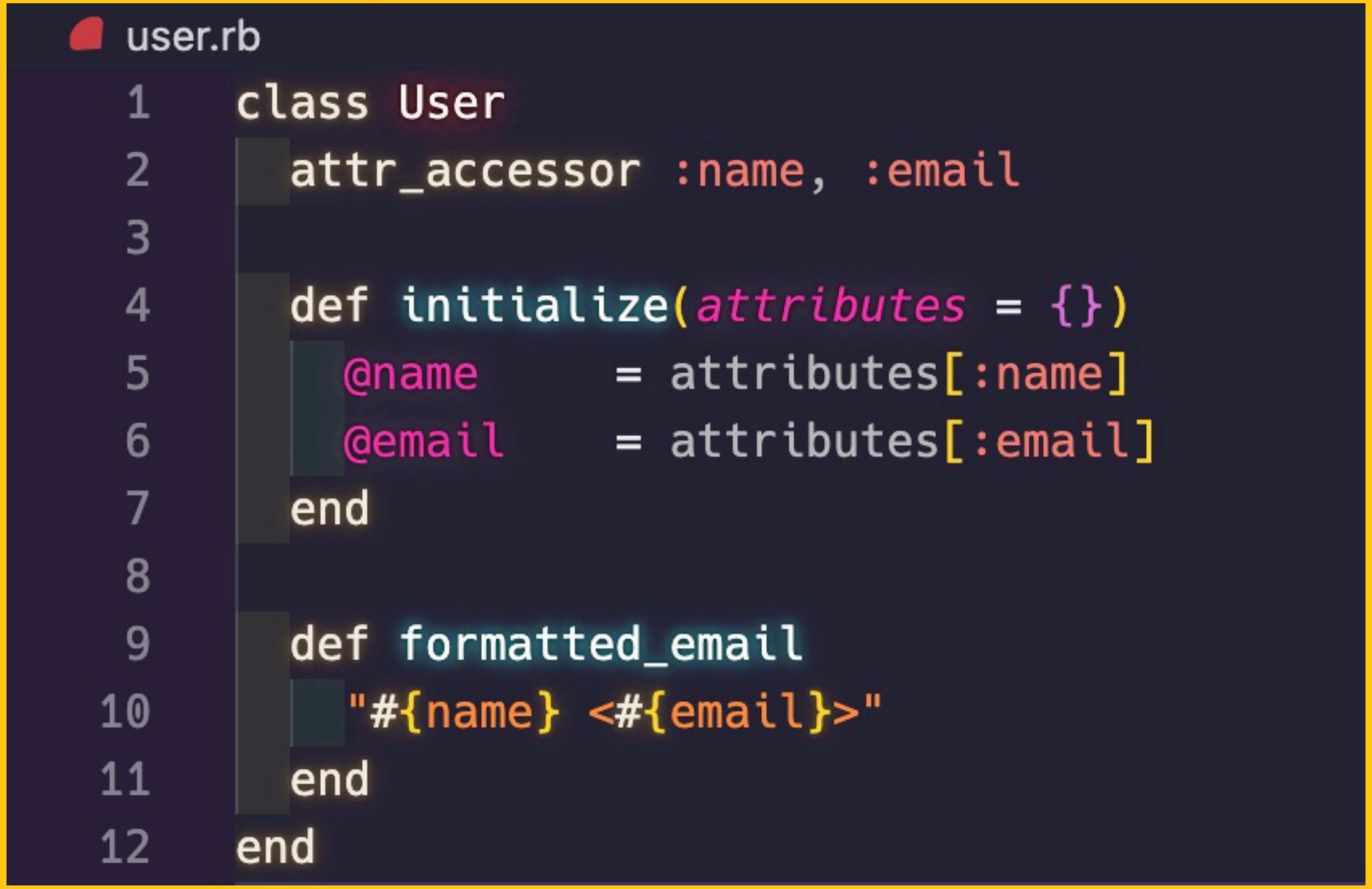
METHOD COMPLETION



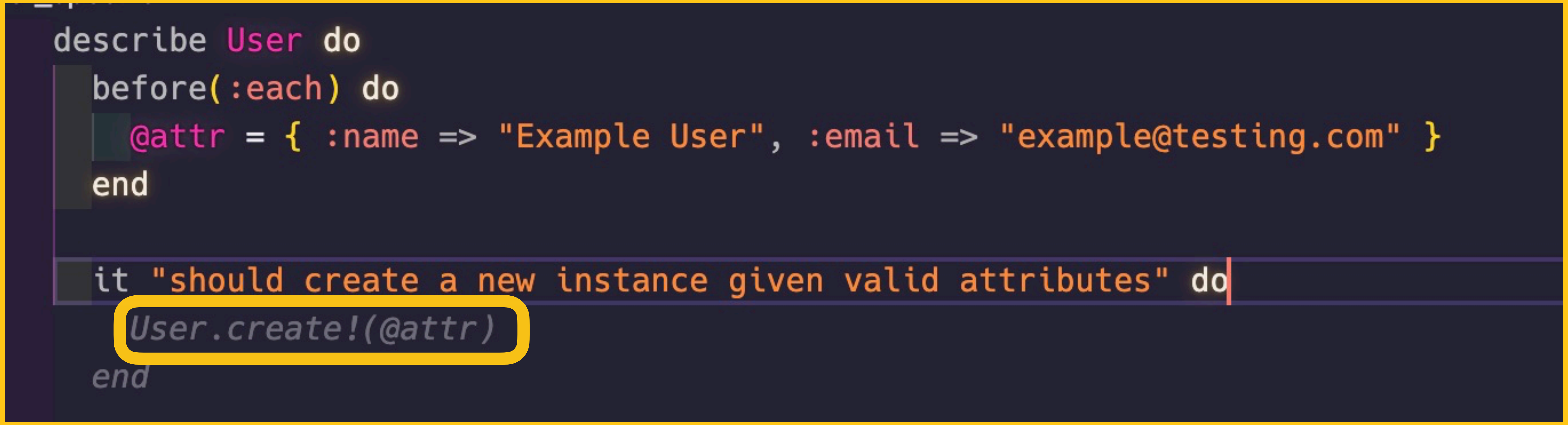
```
class Post < ActiveRecord::Base
  belongs_to :user
  has_many :comments
end

# Path: comment.rb
class Comment < ActiveRecord::Base
  belongs_to :post
  belongs_to :user
end

# Path: user.rb
class User < ActiveRecord::Base
  has_many :posts
  has_many :comments
end

# Path: db/seeds.rb
User.create(name: "John Doe", email: "
```

A BUNCH OF CLASSES

WRITING TESTS

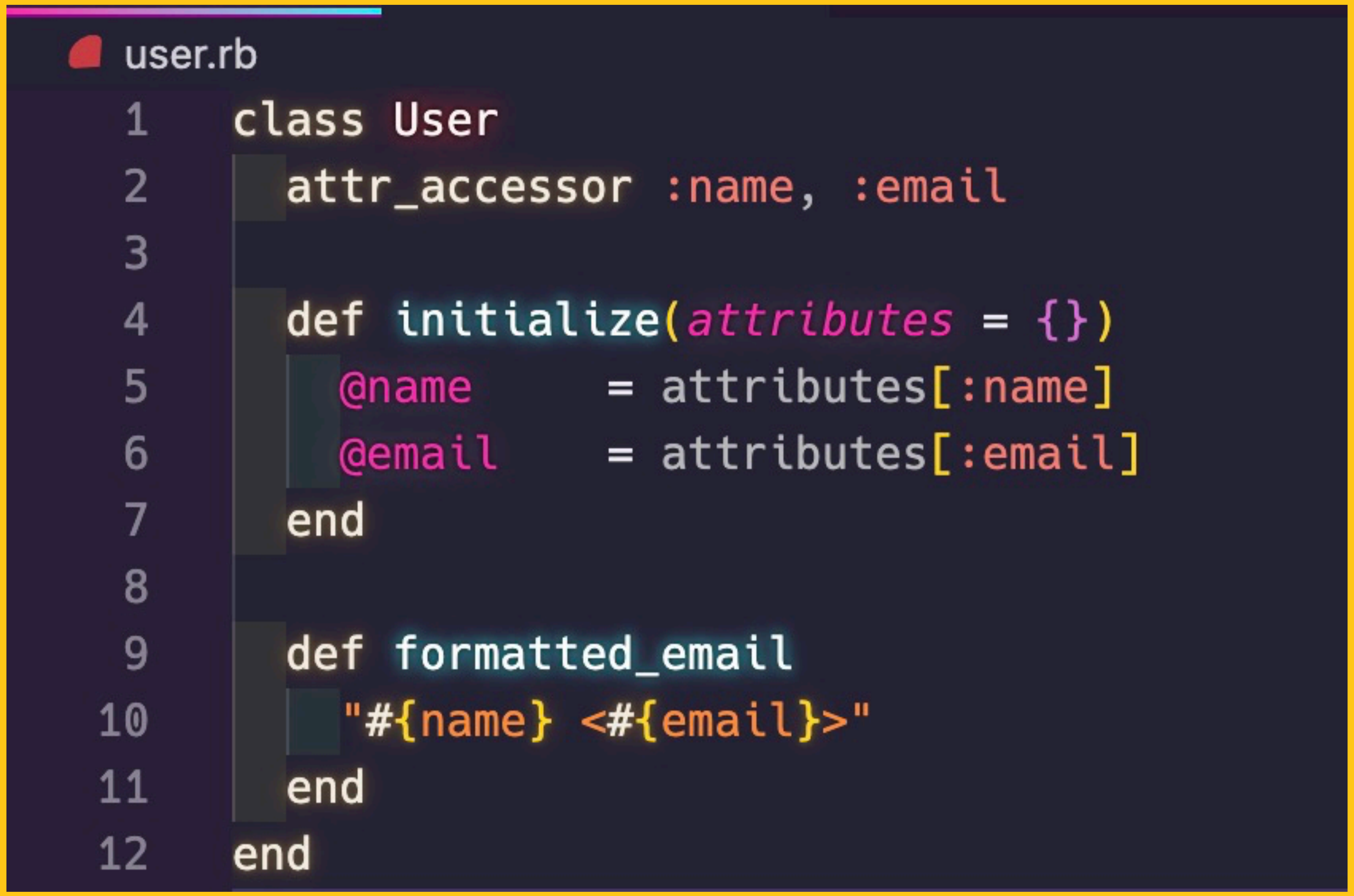


```ruby
class User
  attr_accessor :name, :email

  def initialize(attributes = {})
    @name     = attributes[:name]
    @email    = attributes[:email]
  end

  def formatted_email
    "#{name} <#{email}>"
  end
end
```



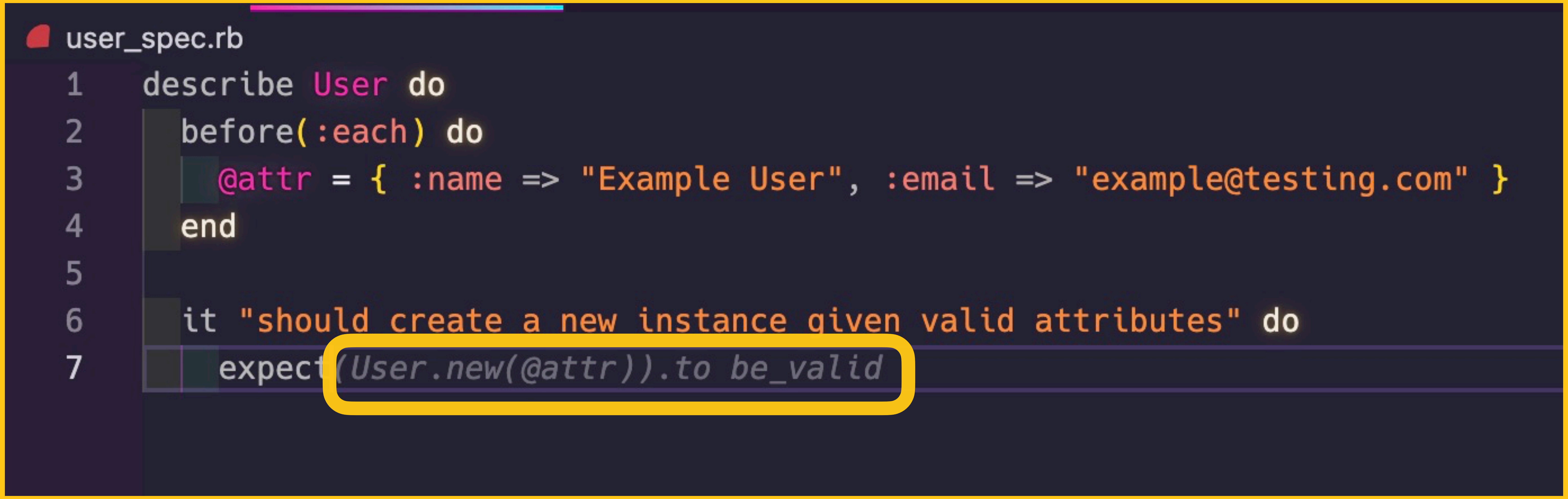
```ruby
describe User do
  before(:each) do
    @attr = { :name => "Example User", :email => "example@testing.com" }
  end

  it "should create a new instance given valid attributes" do
    User.create!(@attr)
  end
```

A TEST METHOD



user.rb

```ruby
class User
  attr_accessor :name, :email

  def initialize(attributes = {})
    @name     = attributes[:name]
    @email    = attributes[:email]
  end

  def formatted_email
    "#{name} <#{email}>"
  end
end
```

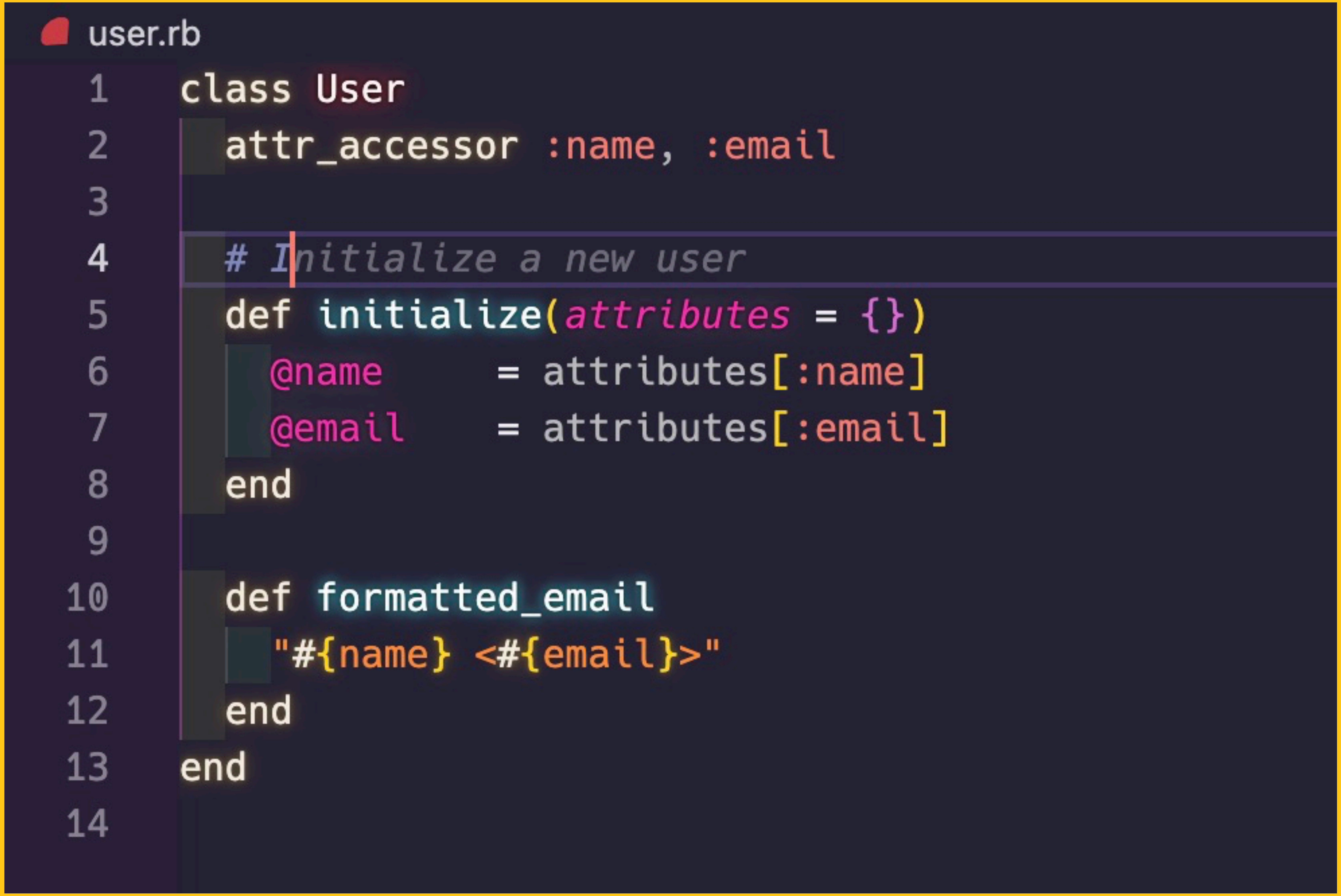


user_spec.rb

```ruby
describe User do
  before(:each) do
    @attr = { :name => "Example User", :email => "example@testing.com" }
  end

  it "should create a new instance given valid attributes" do
    expect(User.new(@attr)).to be_valid
```

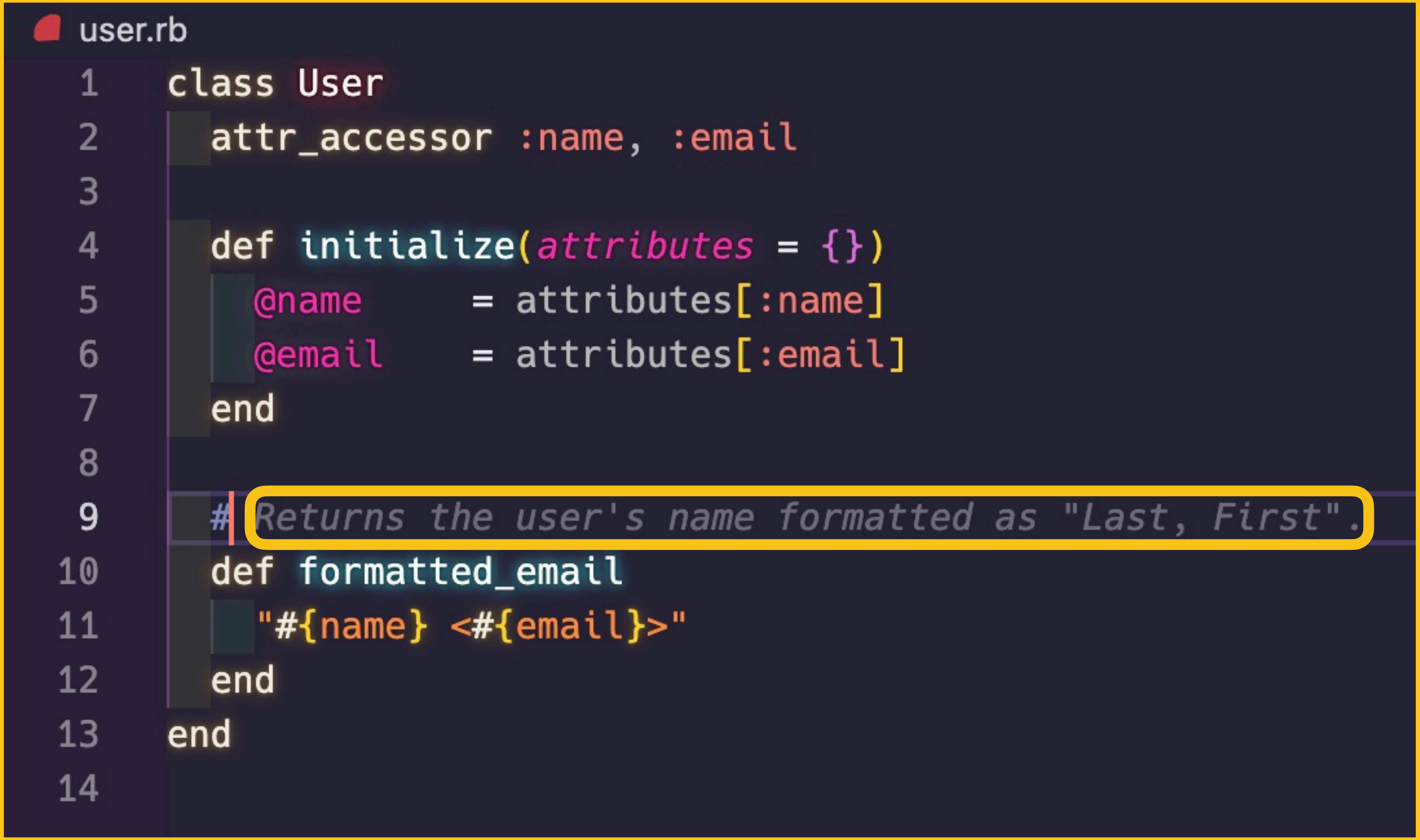
A TEST METHOD

WRITING COMMENTS



```ruby
class User
  attr_accessor :name, :email

  # Initialize a new user
  def initialize(attributes = {})
    @name     = attributes[:name]
    @email    = attributes[:email]
  end

  def formatted_email
    "#{name} <#{email}>"
  end
end

```

MAKING COMMENTS



```ruby
class User
  attr_accessor :name, :email

  def initialize(attributes = {})
    @name     = attributes[:name]
    @email    = attributes[:email]
  end

  # Returns the user's name formatted as "Last, First".
  def formatted_email
    "#{name} <#{email}>"
  end
end
```

MAKING COMMENTS

LIMITATIONS

- Incorrect code
- Unoptimized Code
- Smaller suggestion scope than ChatGPT
- Expense
- Limited IDEs
- Limited Languages
- GitHub lock-in

> “In a recent evaluation, we found that users accepted on average 26% of all completions shown by GitHub Copilot.”

GITHUT COPILOT FAQ

ETHICAL AND LEGAL CONCERNS

LLMS NEED DATA TO TRAIN ON

WHERE DO THEY GET TRAINING DATA

- CommonCrawl.org data
- WebText2
- “Books1” and “Books2” book data sets
- Wikipedia

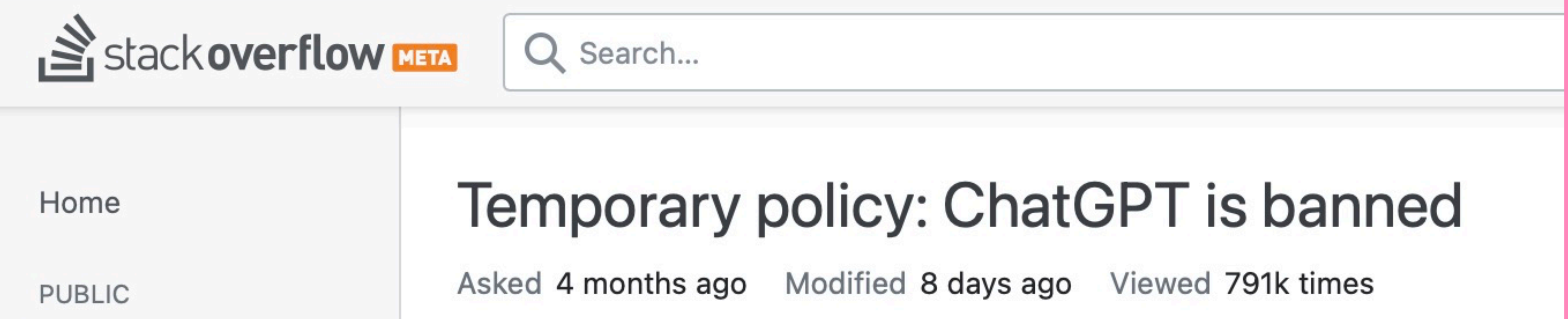
- 159 gigabytes of Python code from 54 million GitHub repositories

- You (and your input)

TRAINING DATA SOURCES ARE OPAQUE

ETHICAL ISSUES

ERRORS



Temporary policy: ChatGPT is banned

Asked 4 months ago Modified 8 days ago Viewed 791k times

TRAINING DATA BIAS

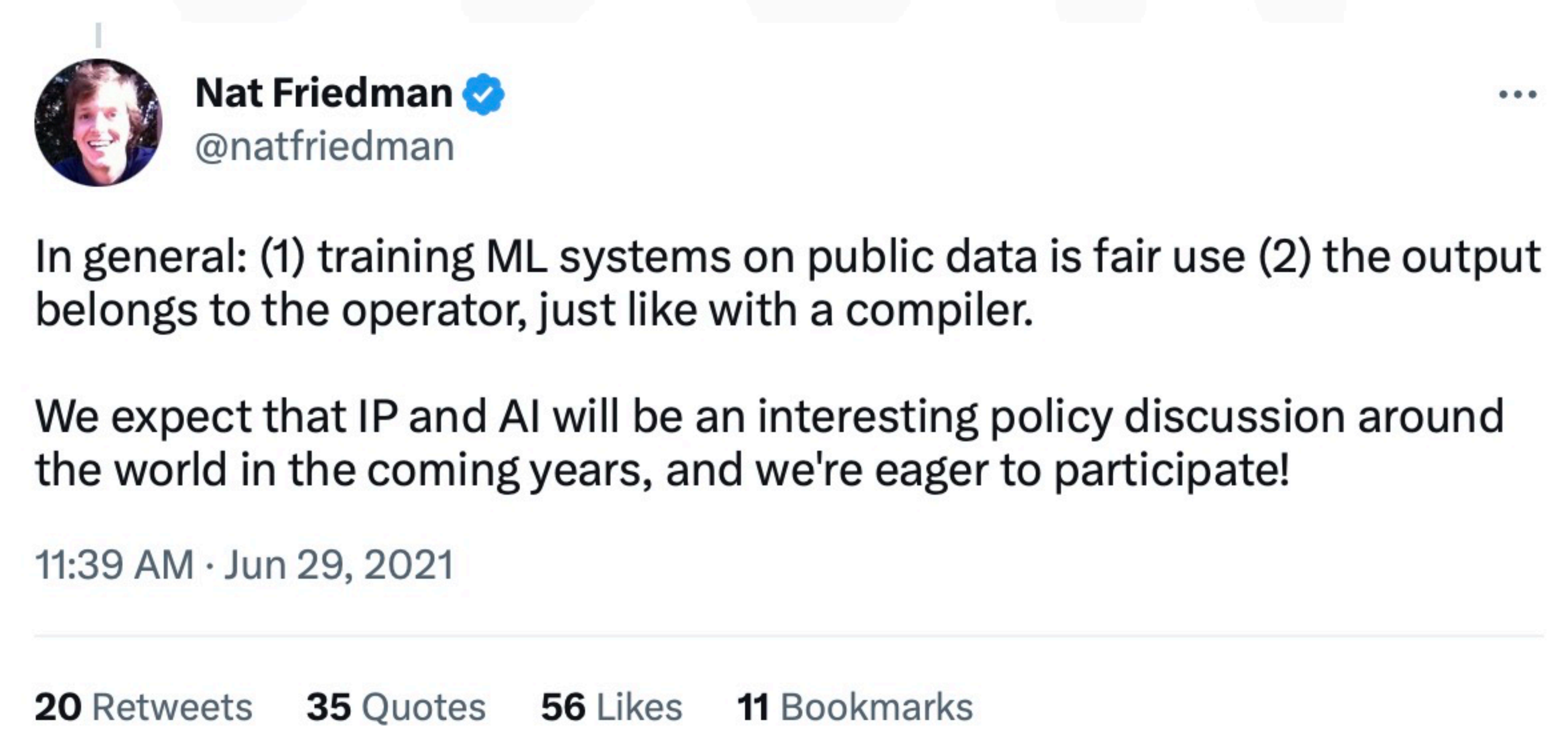
DUTY TO DISCLOSE

IT'S GITHUB COPILOT
SHAKE'N BAKE
ORIGINAL
AND I HELPED!
imgflip.com

LEGAL ISSUES

LIABILITY

INTELLECTUAL PROPERTY



Nat Friedman
@natfriedman

In general: (1) training ML systems on public data is fair use (2) the output belongs to the operator, just like with a compiler.

We expect that IP and AI will be an interesting policy discussion around the world in the coming years, and we're eager to participate!

11:39 AM · Jun 29, 2021

20 Retweets **35** Quotes **56** Likes **11** Bookmarks

LICENSING

Free as in Freedom

BEST PRACTICES

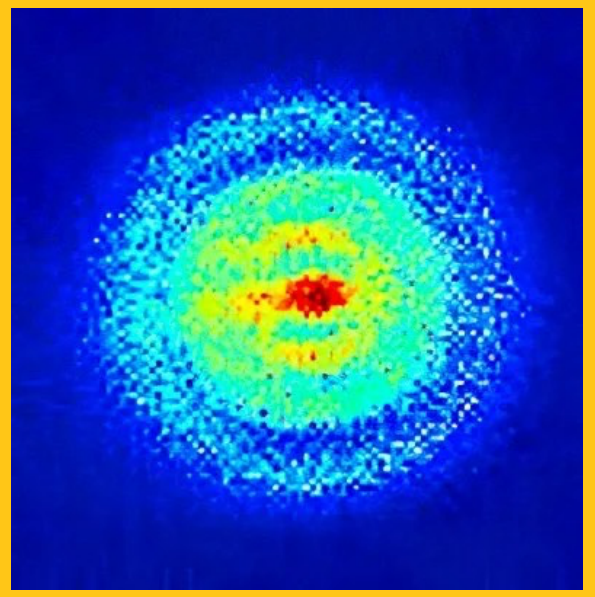
BEST PRACTICES

- Review code for errors
- Understand how the code works
- Be aware of legal and ethical concerns
- Disclose your usage
- Opt out of usage - it’s ok!

CONCLUSION

SUMMARY

HAL 9000



IT'S GITHUB COPILOT
SHAKE'N BAKE
ORIGINAL
AND I HELPED!
imgflip.com

LGTM

REFERENCES

- **ChatGPT Explained: A Normie's Guide To How It Works**
  Jon Stokes
  https://bit.ly/railsconf-2023-ai-1
- **An example of LLM prompting for programming**
  Martin Fowler
  https://bit.ly/railsconf-2023-ai-2
- **What is ChatGPT doing and what does it work?**
  Stephan Wolfram
  https://bit.ly/railsconf-2023-ai-3

Josh Puetz

Slides: https://bit.ly/railsconf-2023-ai-slides
Mastodon: @joshpuetz@mastodon.social
Twitter: @joshpuetz
Web: joshpuetz.com
Work: donut.com

THANK YOU!