Terminal Application Presentation

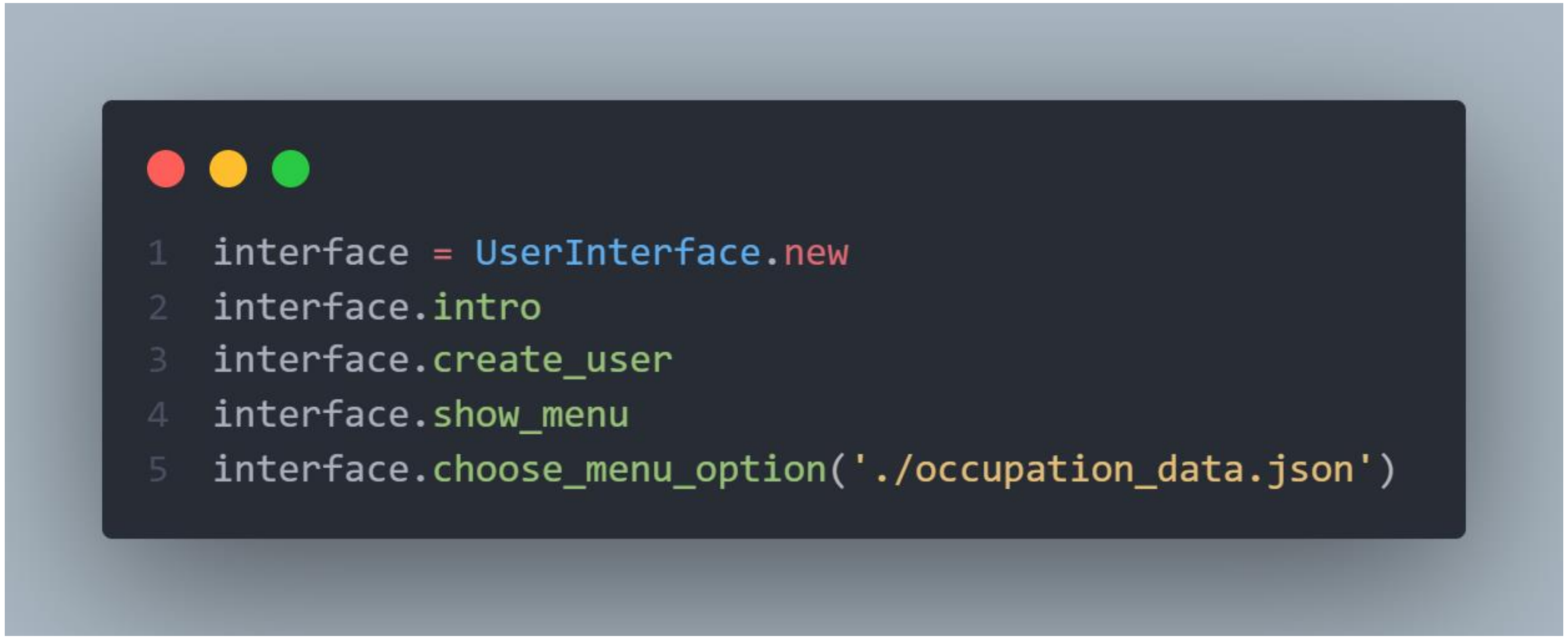
Quentin Haly-Summerfield

Purpose

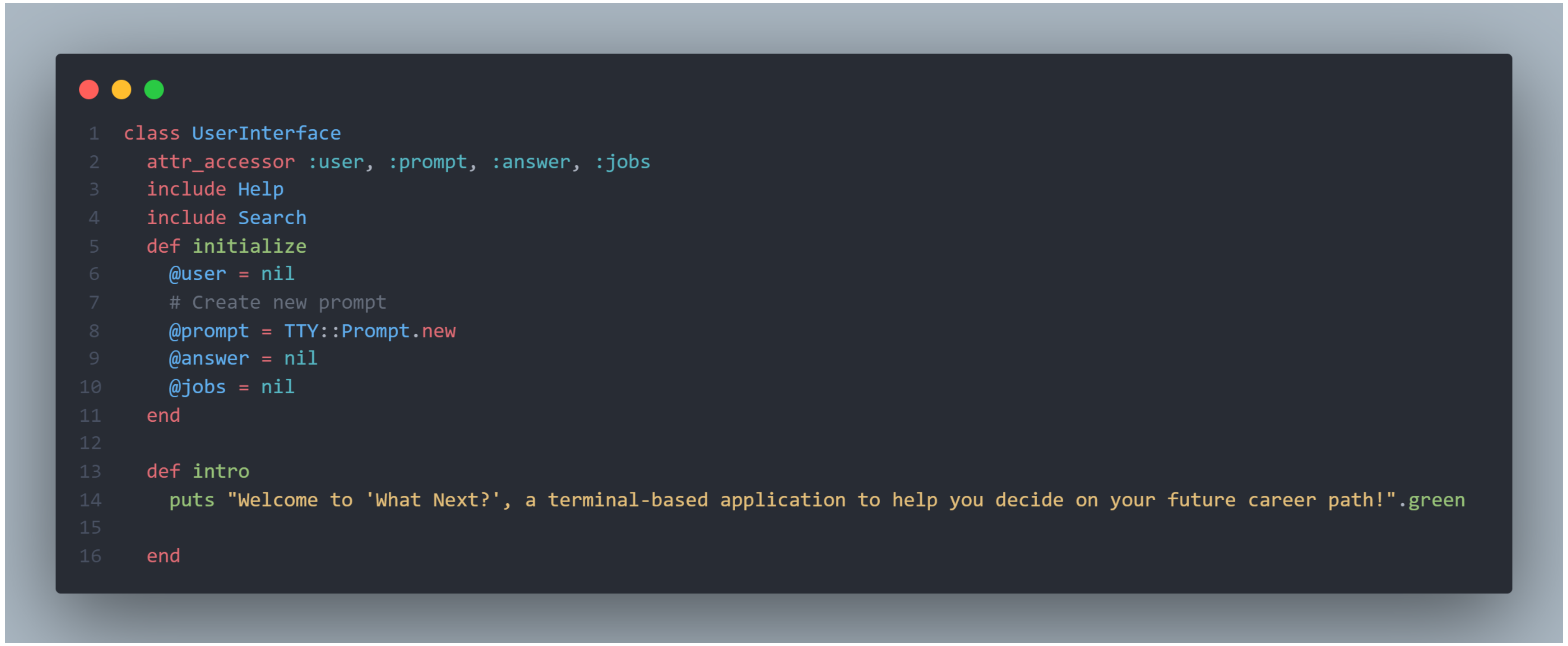
- Career Suggestion and Comparison App
- Aimed primarily at graduating high school students, secondarily at others looking for career suggestions or information
- Intended to provide useful career information across a range of metrics
- Utilises both user terminal input and json file input

How is the app used?

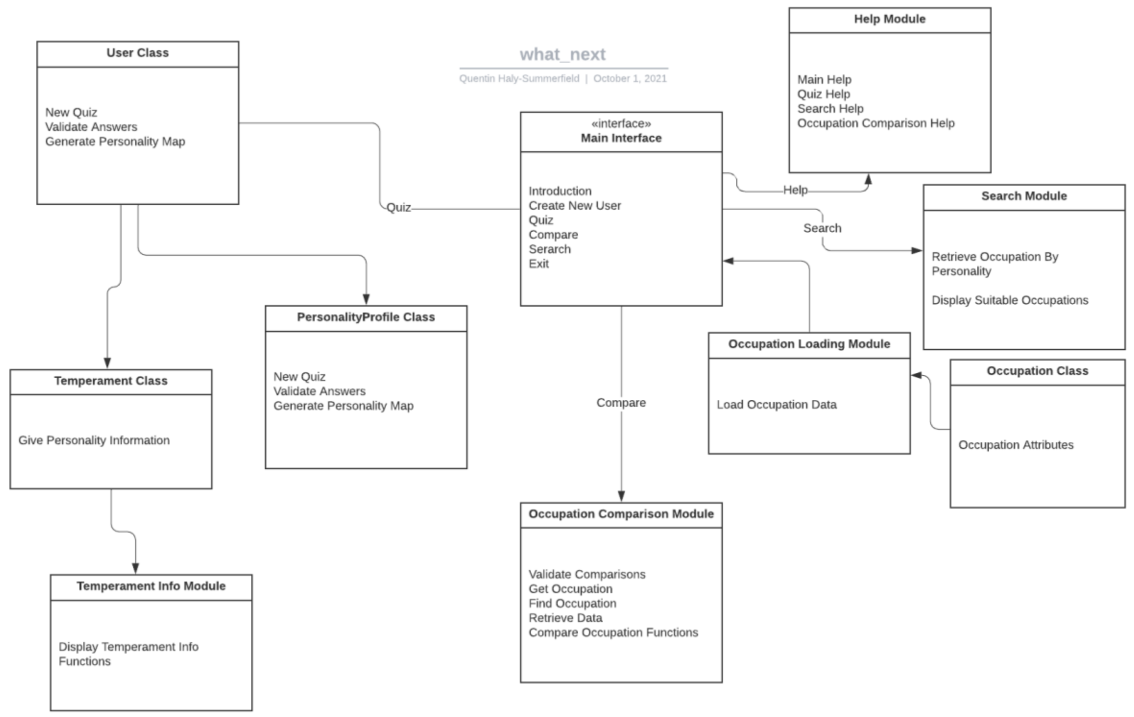
- User is given prompts – take a quiz, compare two occupations, search for occupations, seek help, or exit
- Terminal input uses both tty-prompt and keyboard input – minimal input needed
- The option to quit and/or go back to the main menu is always available
- Upon using a given feature, user is returned to the main screen



```ruby
interface = UserInterface.new
interface.intro
interface.create_user
interface.show_menu
interface.choose_menu_option('./occupation_data.json')
```

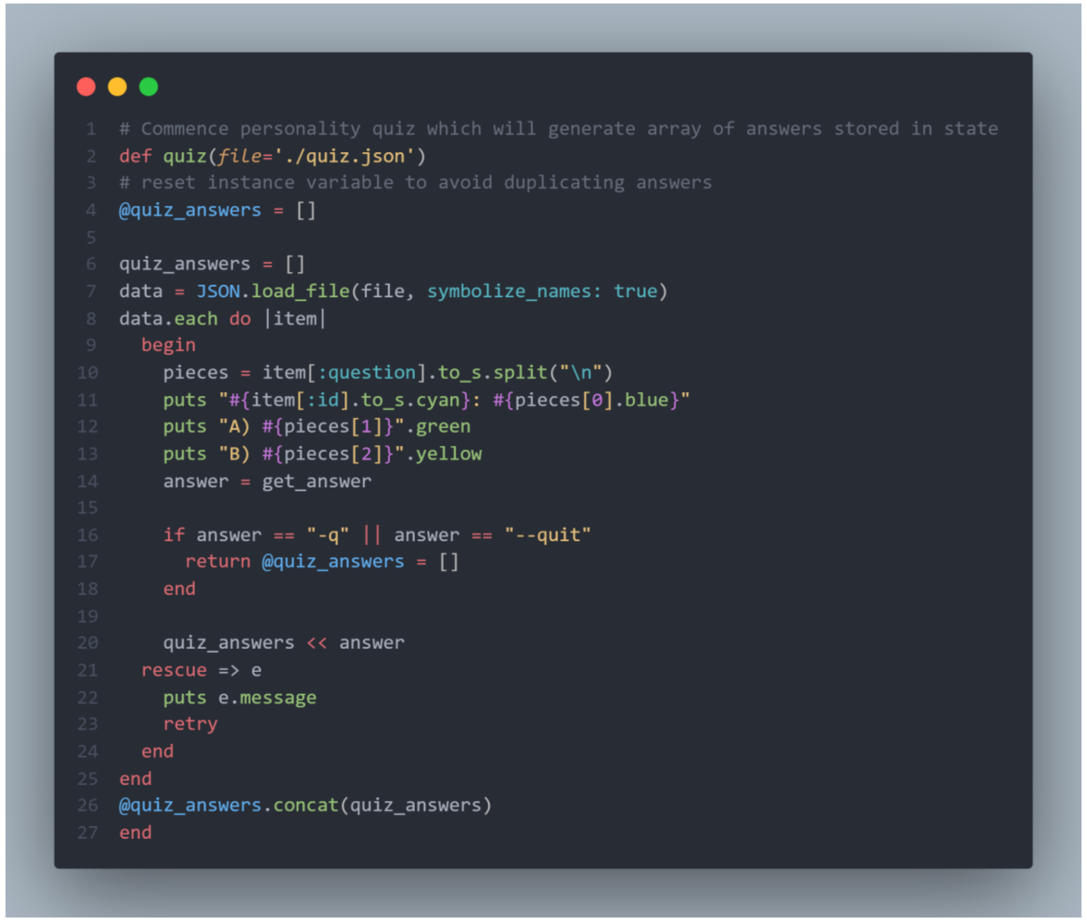


```ruby
class UserInterface
  attr_accessor :user, :prompt, :answer, :jobs
  include Help
  include Search
  def initialize
    @user = nil
    # Create new prompt
    @prompt = TTY::Prompt.new
    @answer = nil
    @jobs = nil
  end

  def intro
    puts "Welcome to 'What Next?', a terminal-based application to help you decide on your future career path!".green

  end
```

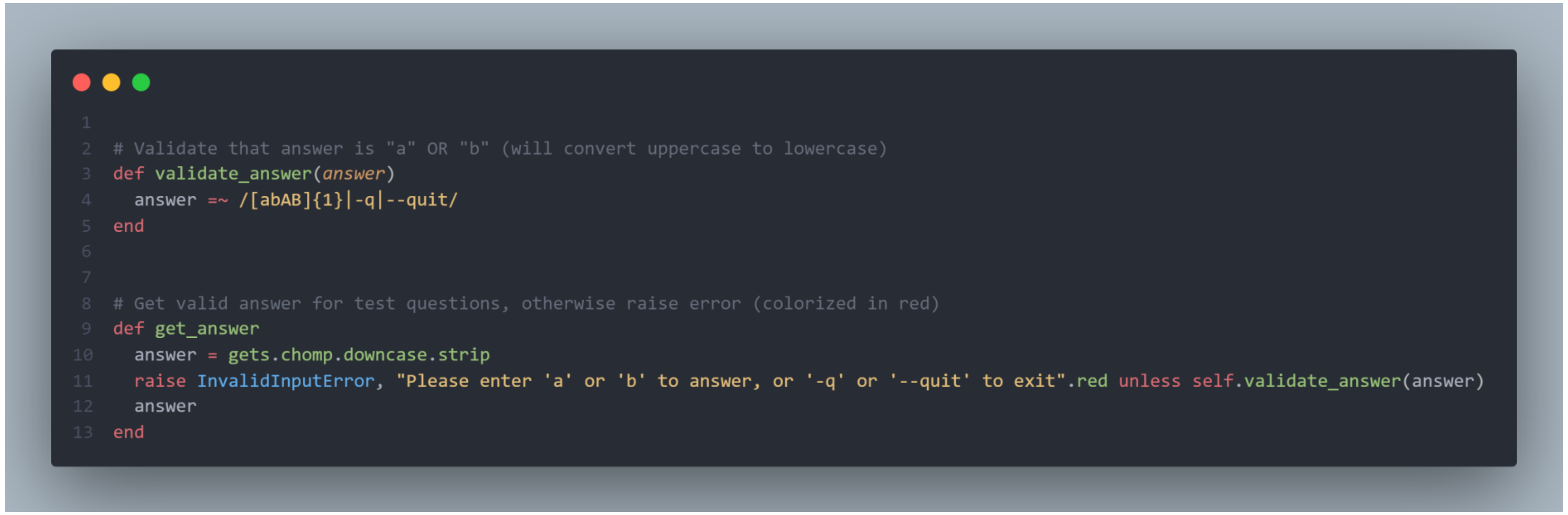

what_next
Quentin Haly-Summerfield | October 1, 2021
User Class
New Quiz
Validate Answers
Generate Personality Map
Quiz
«interface»
Main Interface
Introduction
Create New User
Quiz
Compare
Serarch
Exit
Help Module
Main Help
Quiz Help
Search Help
Occupation Comparison Help
Help
Search
Search Module
Retrieve Occupation By Personality
Display Suitable Occupations
PersonalityProfile Class
New Quiz
Validate Answers
Generate Personality Map
Temperament Class
Give Personality Information
Compare
Occupation Loading Module
Load Occupation Data
Occupation Class
Occupation Attributes
Temperament Info Module
Display Temperament Info Functions
Occupation Comparison Module
Validate Comparisons
Get Occupation
Find Occupation
Retrieve Data
Compare Occupation Functions

Feature One – Personality Quiz

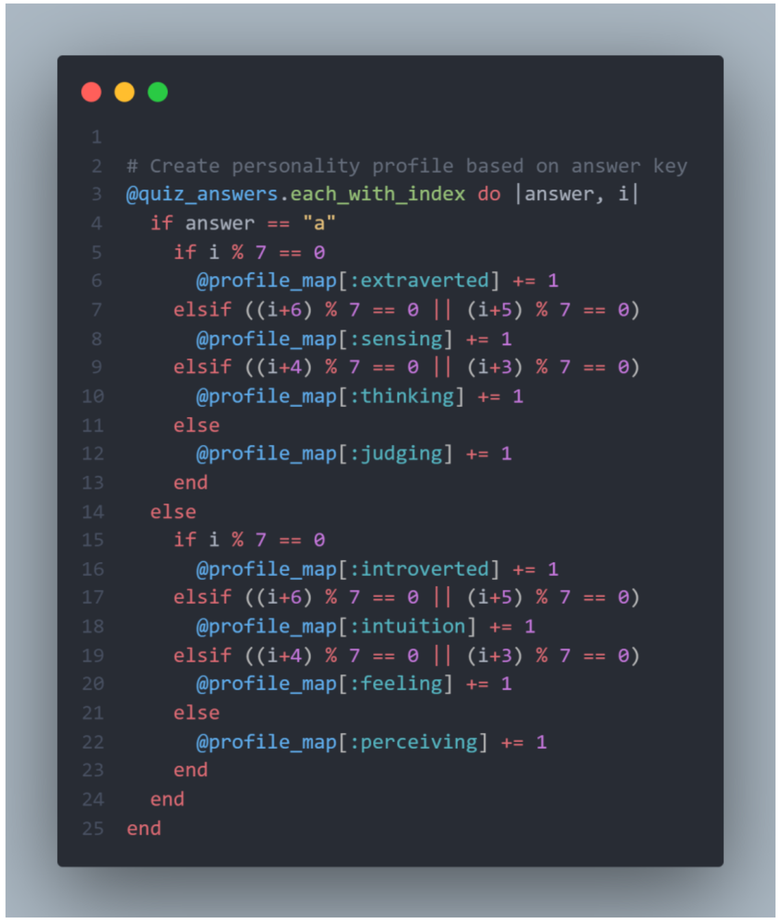
- Utilises Myer-Briggs/Keirsey Temperament Sorter-style personality quiz
- Gathers answers from 70 A) | B) questions, assigns personality attributes based on the result
- From these, one of sixteen personality types is assigned to the user
- Then a search is performed cross-referencing that personality type to occupations suitability for that personality type, along with additional filters



```ruby
# Commence personality quiz which will generate array of answers stored in state
def quiz(file='./quiz.json')
# reset instance variable to avoid duplicating answers
@quiz_answers = []

quiz_answers = []
data = JSON.load_file(file, symbolize_names: true)
data.each do |item|
  begin
    pieces = item[:question].to_s.split("\n")
    puts "#{item[:id].to_s.cyan}: #{pieces[0].blue}"
    puts "A) #{pieces[1]}".green
    puts "B) #{pieces[2]}".yellow
    answer = get_answer

    if answer == "-q" || answer == "--quit"
      return @quiz_answers = []
    end

    quiz_answers << answer
  rescue => e
    puts e.message
    retry
  end
end
@quiz_answers.concat(quiz_answers)
end
```



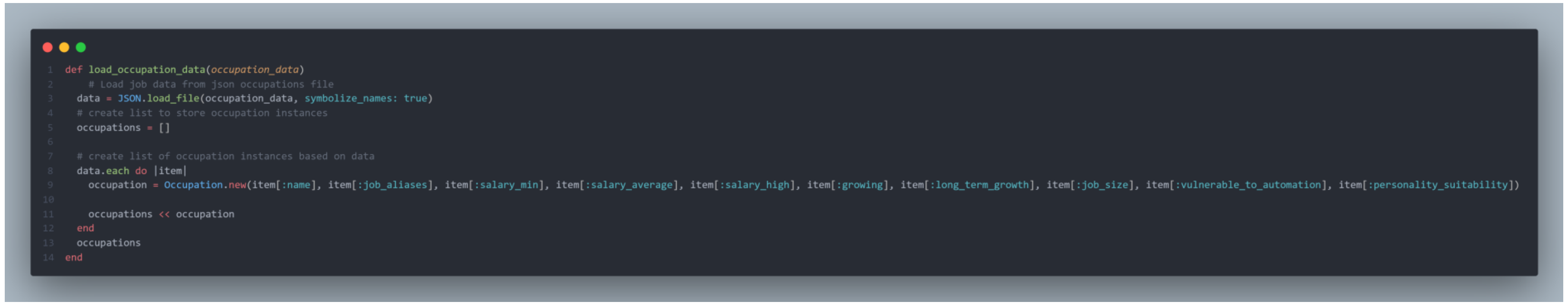
```ruby

# Validate that answer is "a" OR "b" (will convert uppercase to lowercase)
def validate_answer(answer)
  answer =~ /[abAB]{1}|-q|--quit/
end


# Get valid answer for test questions, otherwise raise error (colorized in red)
def get_answer
  answer = gets.chomp.downcase.strip
  raise InvalidInputError, "Please enter 'a' or 'b' to answer, or '-q' or '--quit' to exit".red unless self.validate_answer(answer)
  answer
end
```



```ruby

# Create personality profile based on answer key
@quiz_answers.each_with_index do |answer, i|
  if answer == "a"
    if i % 7 == 0
      @profile_map[:extraverted] += 1
    elsif ((i+6) % 7 == 0 || (i+5) % 7 == 0)
      @profile_map[:sensing] += 1
    elsif ((i+4) % 7 == 0 || (i+3) % 7 == 0)
      @profile_map[:thinking] += 1
    else
      @profile_map[:judging] += 1
    end
  else
    if i % 7 == 0
      @profile_map[:introverted] += 1
    elsif ((i+6) % 7 == 0 || (i+5) % 7 == 0)
      @profile_map[:intuition] += 1
    elsif ((i+4) % 7 == 0 || (i+3) % 7 == 0)
      @profile_map[:feeling] += 1
    else
      @profile_map[:perceiving] += 1
    end
  end
end
```

Feature Two – Career Comparison

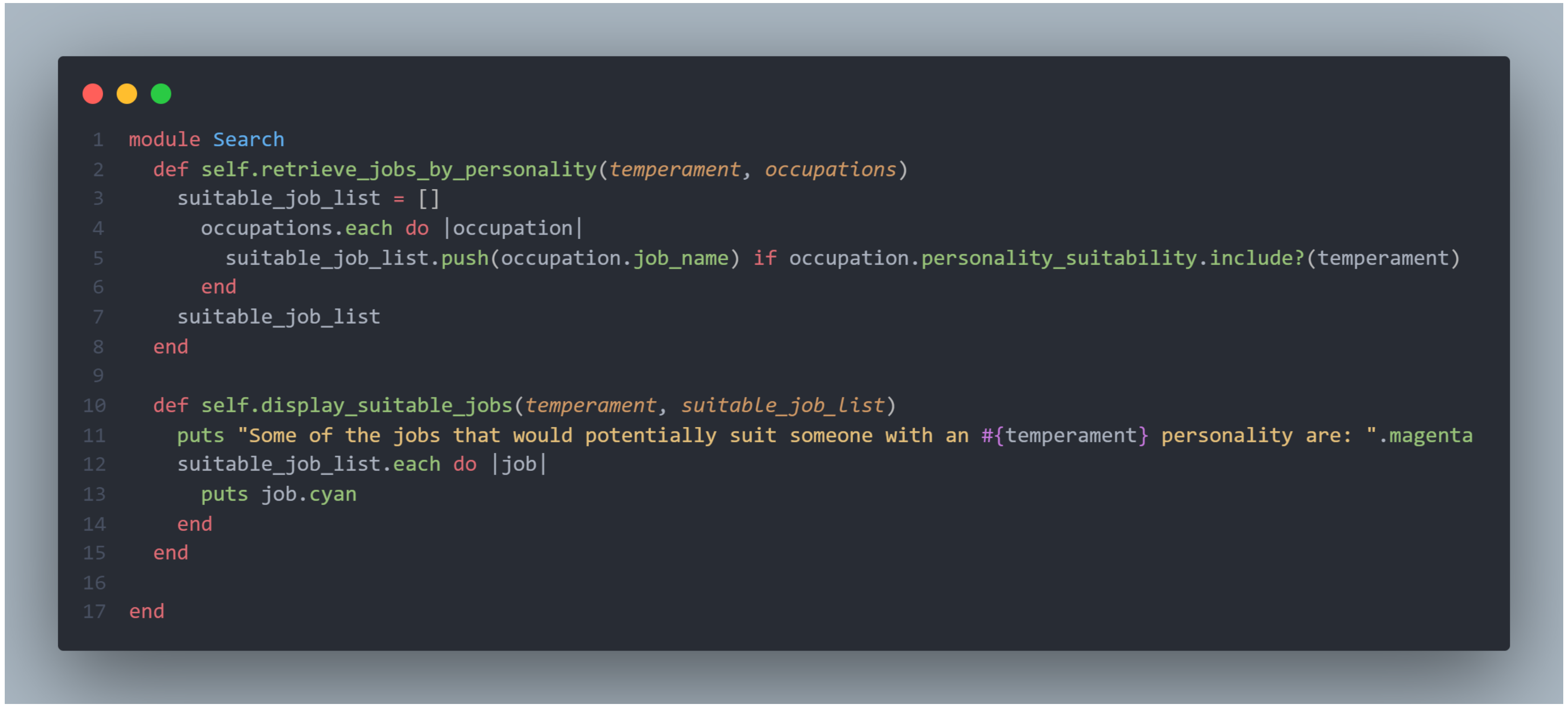
- Career comparison feature
- Receives and validates two user inputs
- Compares the two on metric of choice – salary, job size, growth etc.
- Jobs are aliased to increase search flexibility
- Use of conditional rendering in outputting display



```ruby
def load_occupation_data(occupation_data)
    # Load job data from json occupations file
  data = JSON.load_file(occupation_data, symbolize_names: true)
  # create list to store occupation instances
  occupations = []

  # create list of occupation instances based on data
  data.each do |item|
    occupation = Occupation.new(item[:name], item[:job_aliases], item[:salary_min], item[:salary_average], item[:salary_high], item[:growing], item[:long_term_growth], item[:job_size], item[:vulnerable_to_automation], item[:personality_suitability])

    occupations << occupation
  end
  occupations
end
```

Feature Three – Occupation Search

- Search feature based on personality type
- Allows users who already know personality type or prefer to take another one online to input results directly
- Returns list of all occupations suitable to a given personality type without the filtering used in the quiz feature



```ruby
module Search
  def self.retrieve_jobs_by_personality(temperament, occupations)
    suitable_job_list = []
      occupations.each do |occupation|
        suitable_job_list.push(occupation.job_name) if occupation.personality_suitability.include?(temperament)
      end
    suitable_job_list
  end

  def self.display_suitable_jobs(temperament, suitable_job_list)
    puts "Some of the jobs that would potentially suit someone with an #{temperament} personality are: ".magenta
    suitable_job_list.each do |job|
      puts job.cyan
    end
  end

end
```

Challenges

- Initial algorithm design – had to devise a formula to associate given responses to personality attributes
- Data schema – constantly changing as more data was needed
- Refactoring – initially started with few classes and modules, gradually looked to encapsulate elements
- Debugging – entire program broke at times
- Commenting/documenting - keeping track of various elements in a text heavy app

Ethical Issues

- Potential copyright issues – Myer-Briggs and Keirsey names are trademarked
- Have to attribute where necessary, make significant modifications to test question/answers and disavow association with any particular test format or type
- Have to make clear test is only a loose guide or suggestion, not definitive advice or in any way scientific

Favourite Elements

- The fact that the algorithms (mostly) worked
- Using JSON data to create new class instances, trying to mimic a real API
- Being able to reset the app relatively seamlessly, which took some debugging