learning byebug

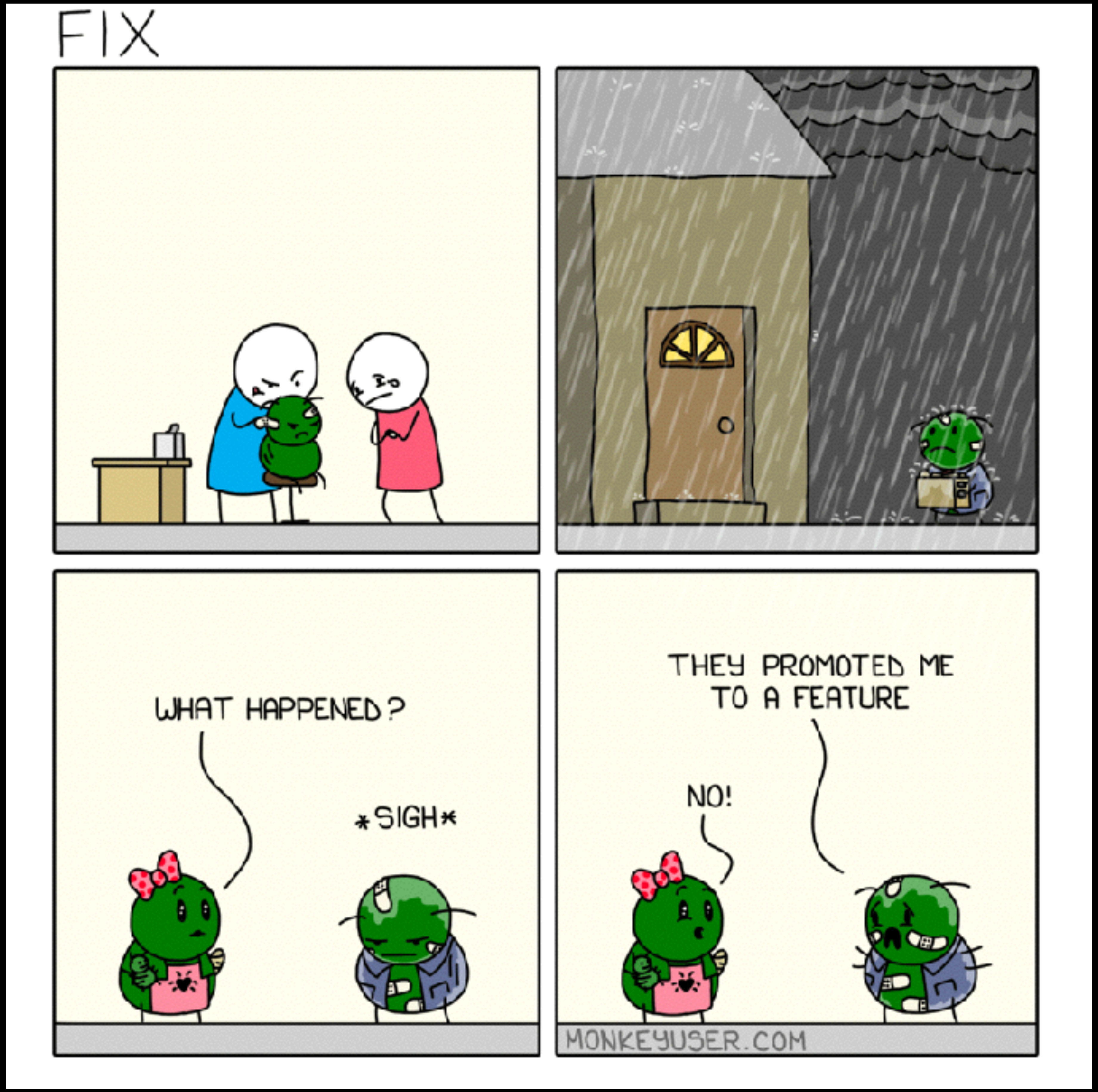
how to up your puts debugging game

about

```
def about

• After 6pm Rubyist
• Thinks about engineering systems
  and processes
• @ghosteathuman

end
```

Service ROCKET

we ship ~~bugs~~ features



https://www.monkeyuser.com/2020/fix/

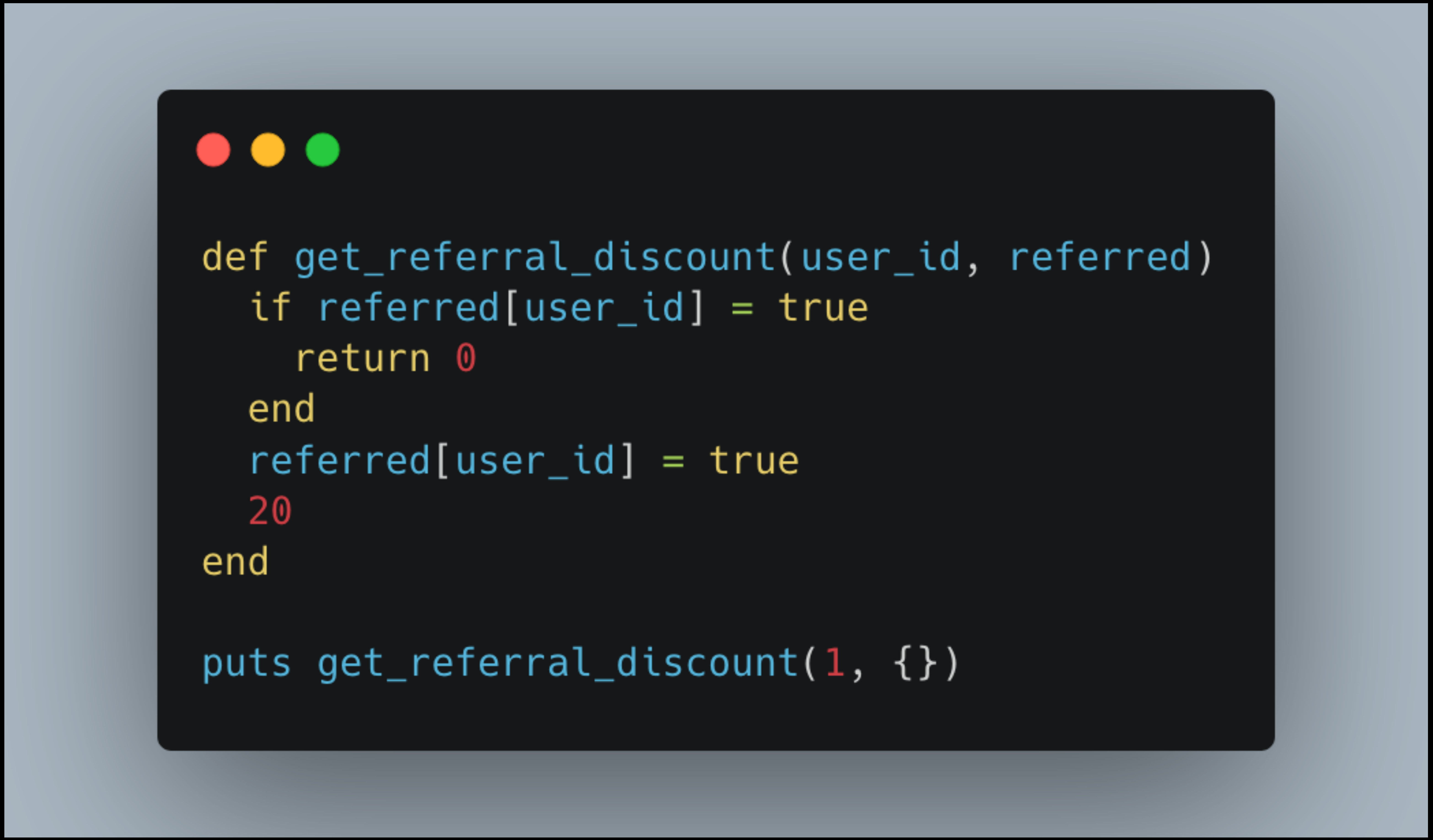
a study in ruby - I

Read The Code

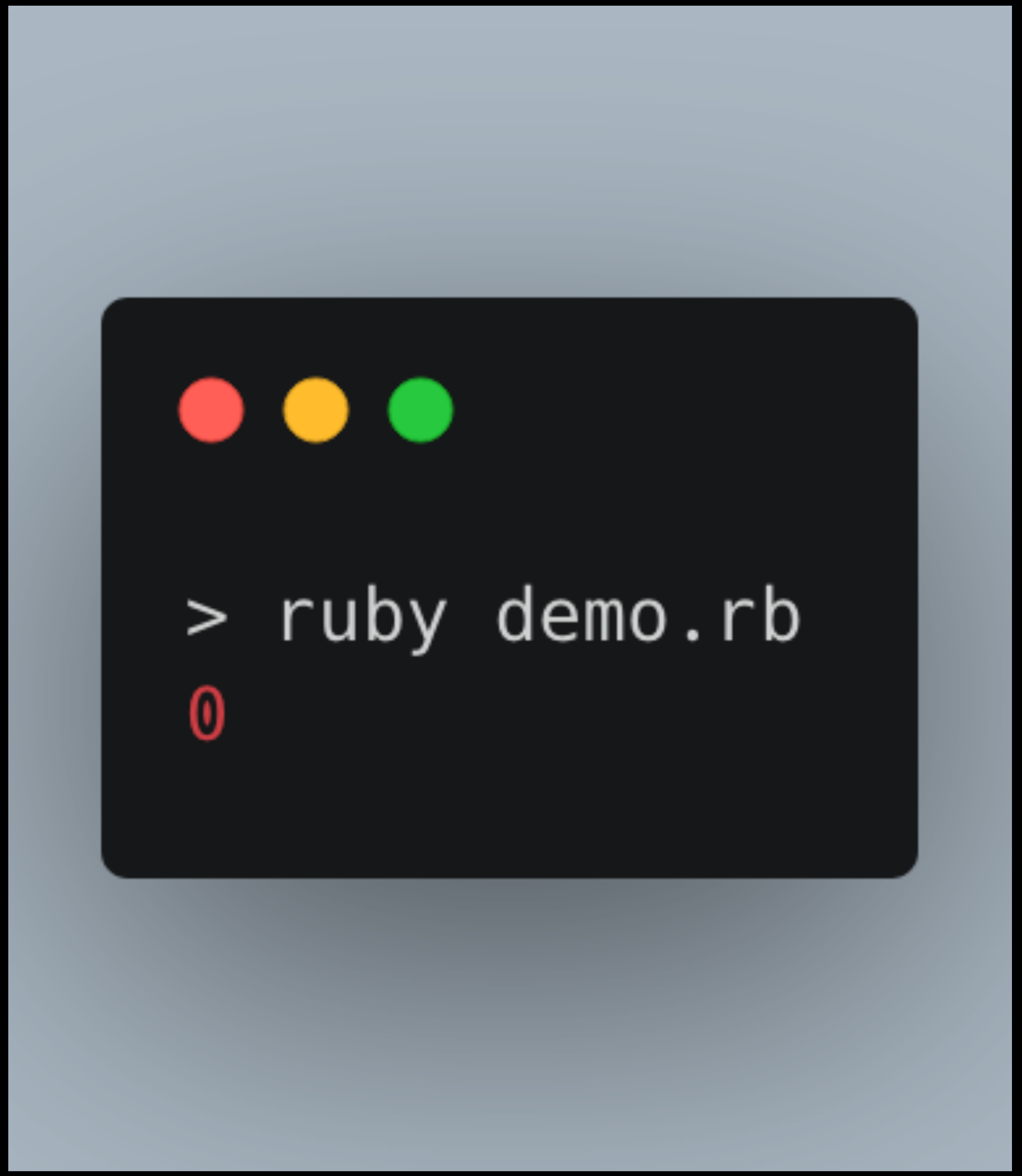
a study in ruby - II

a study in ruby - III

sample ~~bug~~ feature

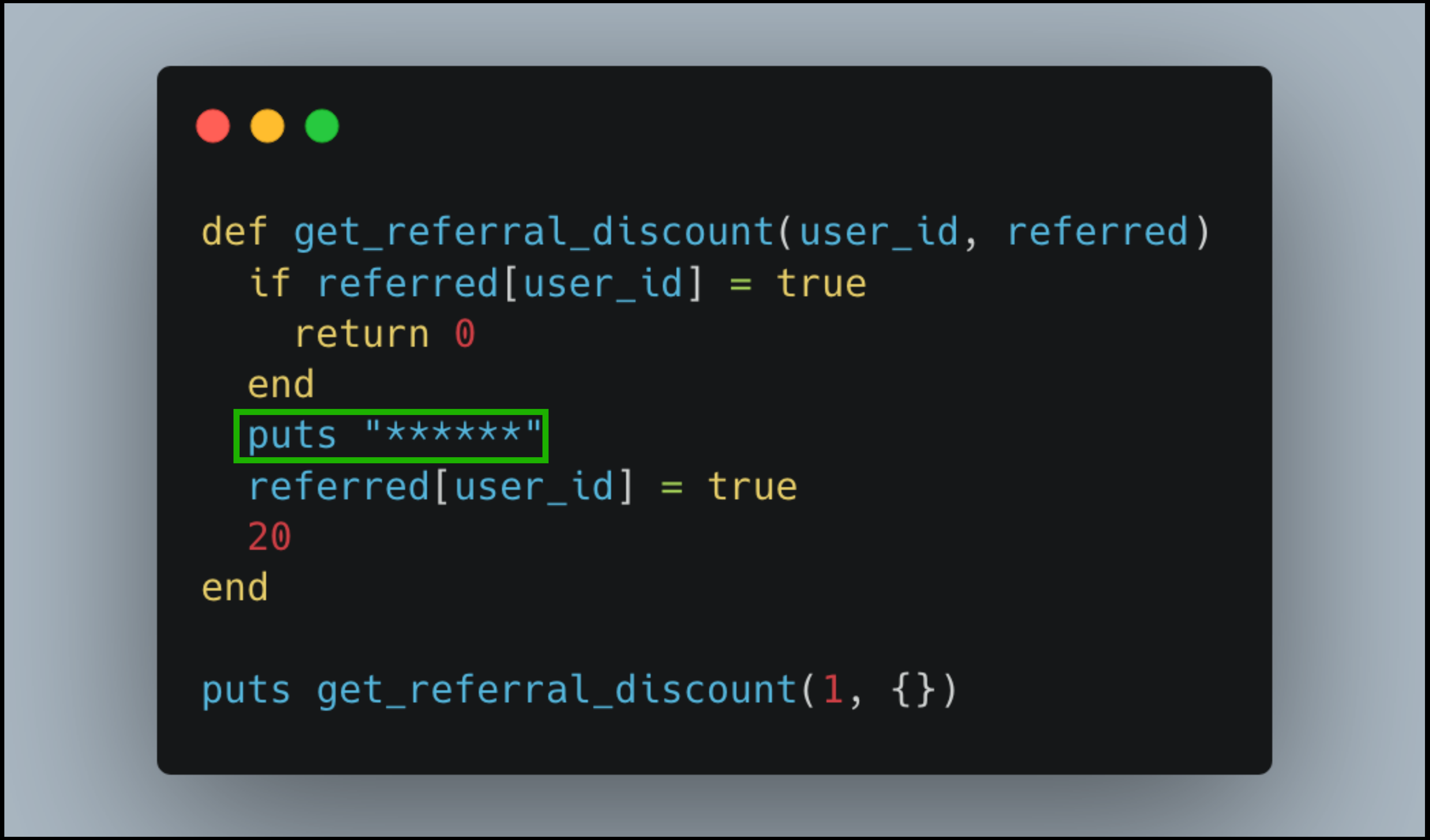


```ruby
def get_referral_discount(user_id, referred)
  if referred[user_id] = true
    return 0
  end
  referred[user_id] = true
  20
end

puts get_referral_discount(1, {})
```

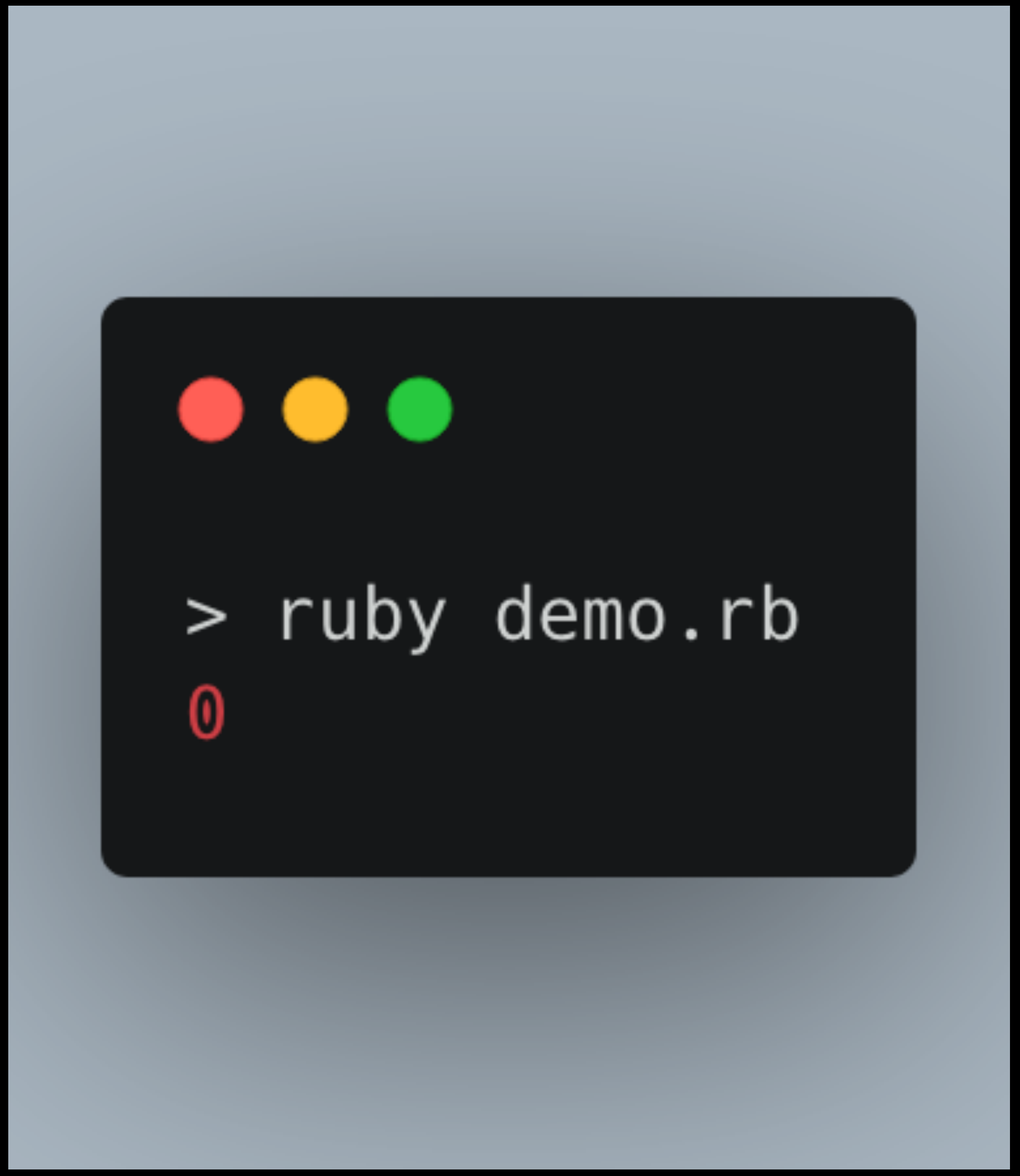


```
> ruby demo.rb
0
```

using puts



```ruby
def get_referral_discount(user_id, referred)
  if referred[user_id] = true
    return 0
  end
  puts "******"
  referred[user_id] = true
  20
end

puts get_referral_discount(1, {})
```

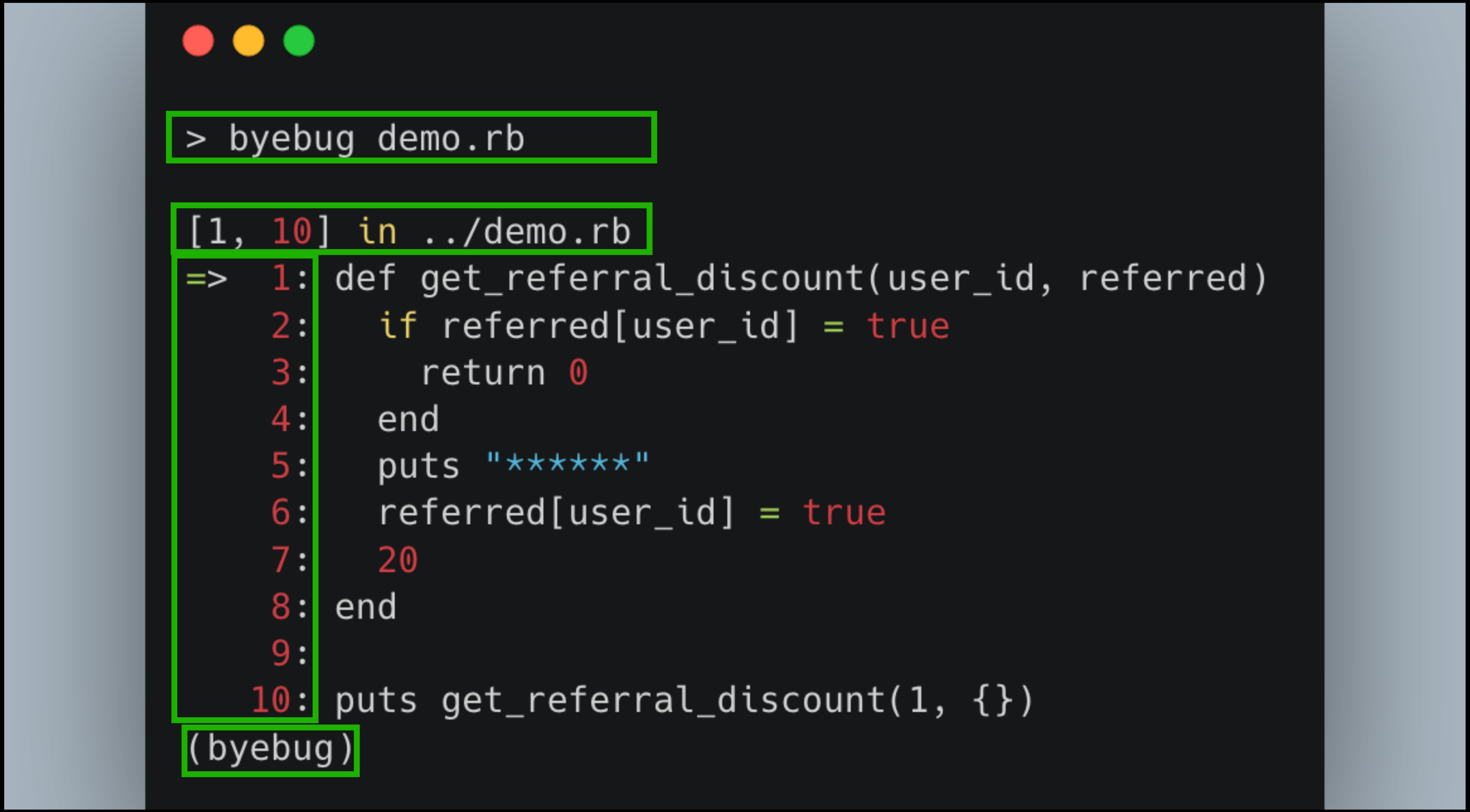


```
> ruby demo.rb
0
```

byebug

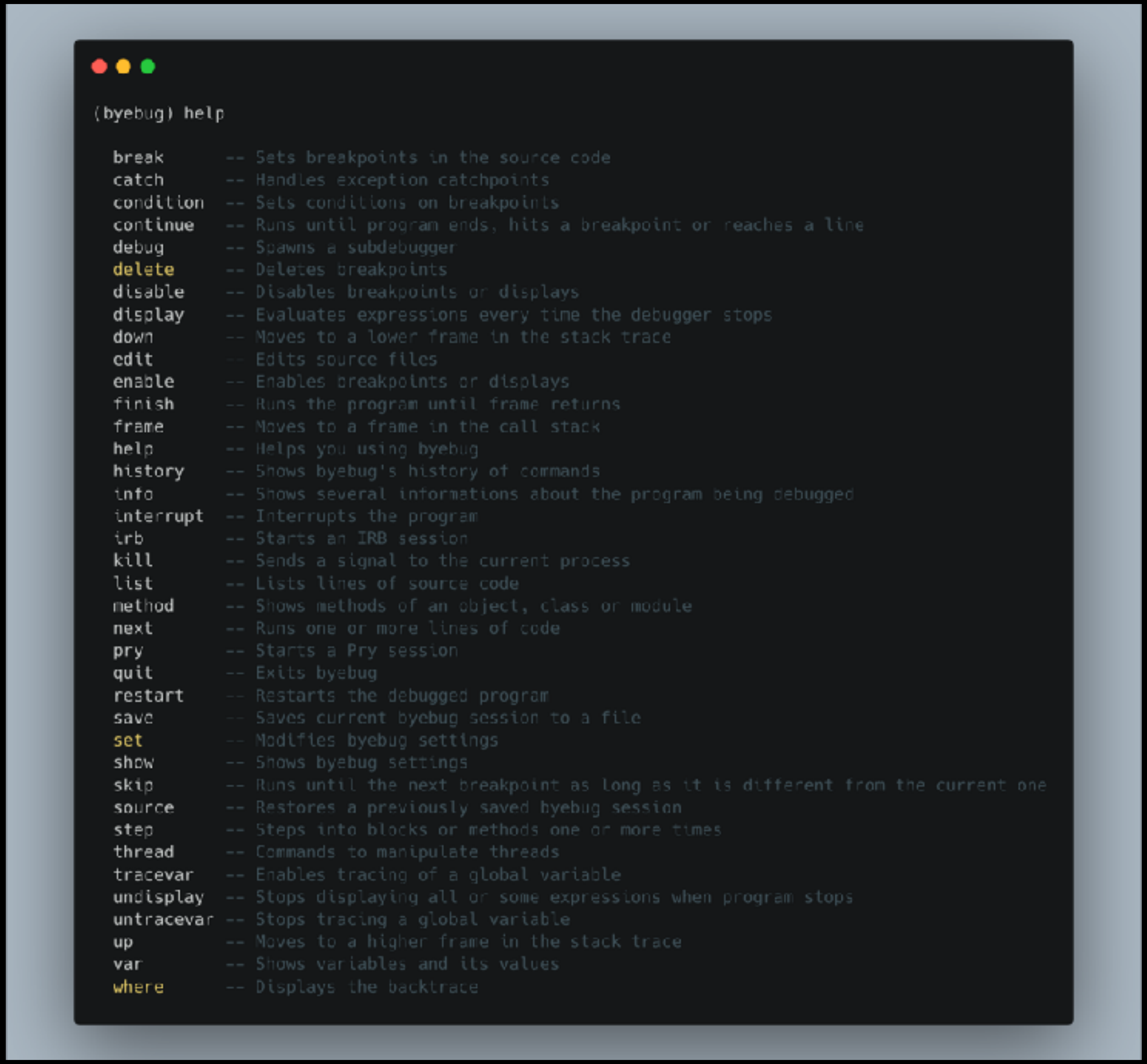
- Ruby debugger
- Uses Tracepoint API
- Supports MRI 2.4.0 or higher*, doesn’t not support other implementations like JRuby & TruffleRuby.

using byebug



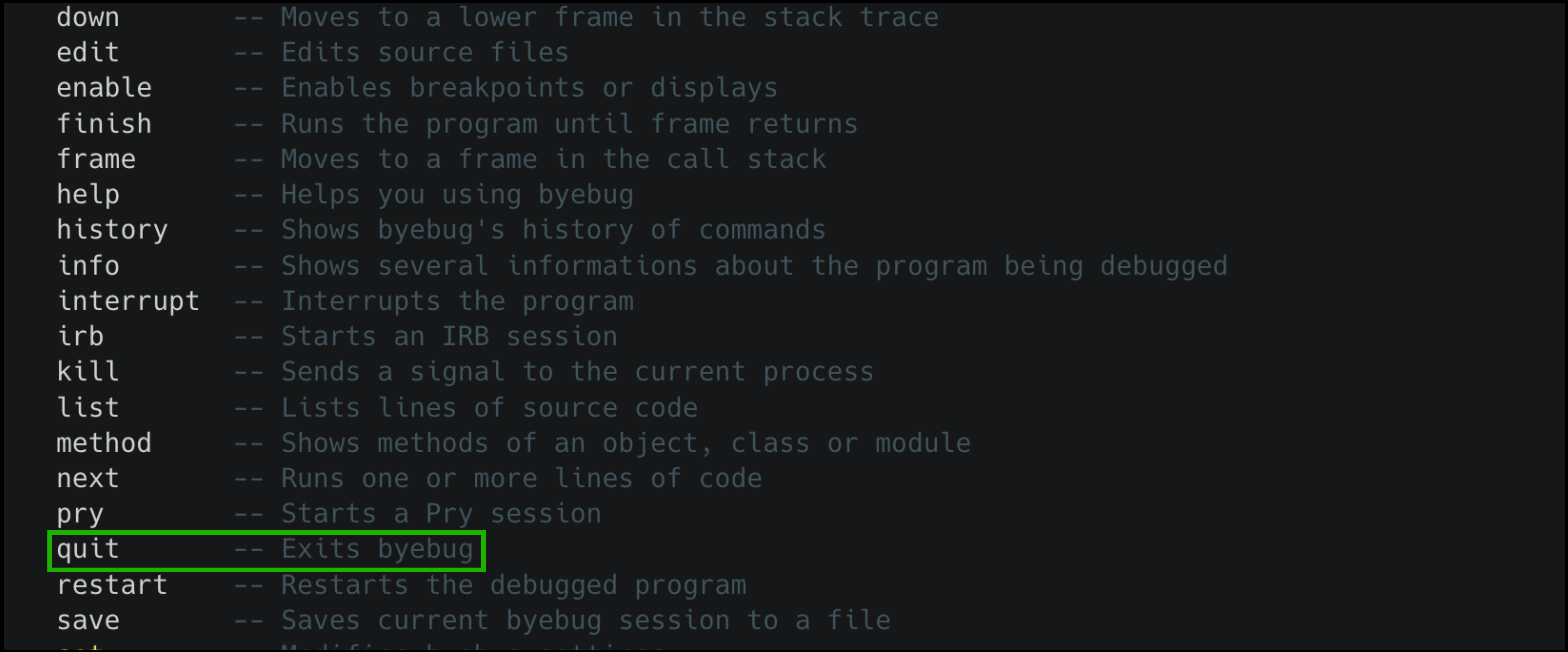
```
> byebug demo.rb

[1, 10] in ../demo.rb
=>  1: def get_referral_discount(user_id, referred)
    2:   if referred[user_id] = true
    3:     return 0
    4:   end
    5:   puts "******"
    6:   referred[user_id] = true
    7:   20
    8: end
    9:
   10: puts get_referral_discount(1, {})
(byebug)
```

help



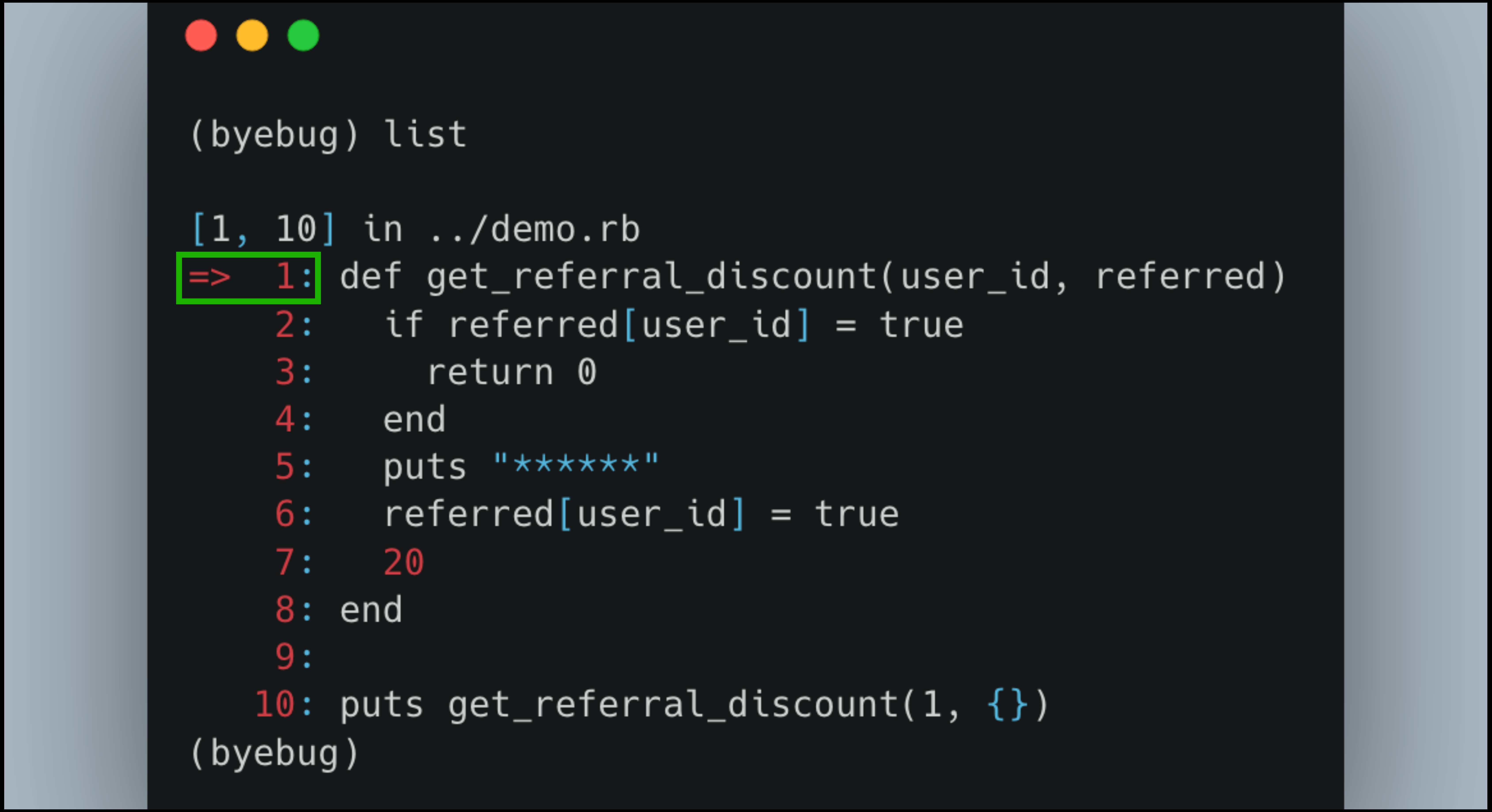
```
(byebug) help

  break      -- Sets breakpoints in the source code
  catch      -- Handles exception catchpoints
  condition  -- Sets conditions on breakpoints
  continue   -- Runs until program ends, hits a breakpoint or reaches a line
  debug      -- Spawns a subdebugger
  delete     -- Deletes breakpoints
  disable    -- Disables breakpoints or displays
  display    -- Evaluates expressions every time the debugger stops
  down       -- Moves to a lower frame in the stack trace
  edit       -- Edits source files
  enable     -- Enables breakpoints or displays
  finish     -- Runs the program until frame returns
  frame      -- Moves to a frame in the call stack
  help       -- Helps you using byebug
  history    -- Shows byebug's history of commands
  info       -- Shows several informations about the program being debugged
  interrupt  -- Interrupts the program
  irb        -- Starts an IRB session
  kill       -- Sends a signal to the current process
  list       -- Lists lines of source code
  method     -- Shows methods of an object, class or module
  next       -- Runs one or more lines of code
  pry        -- Starts a Pry session
  quit       -- Exits byebug
  restart    -- Restarts the debugged program
  save       -- Saves current byebug session to a file
  set        -- Modifies byebug settings
  show       -- Shows byebug settings
  skip       -- Runs until the next breakpoint as long as it is different from the current one
  source     -- Restores a previously saved byebug session
  step       -- Steps into blocks or methods one or more times
  thread     -- Commands to manipulate threads
  tracevar   -- Enables tracing of a global variable
  undisplay  -- Stops displaying all or some expressions when program stops
  untracevar -- Stops tracing a global variable
  up         -- Moves to a higher frame in the stack trace
  var        -- Shows variables and its values
  where      -- Displays the backtrace
```

help - I



```
down        -- Moves to a lower frame in the stack trace
edit        -- Edits source files
enable      -- Enables breakpoints or displays
finish      -- Runs the program until frame returns
frame       -- Moves to a frame in the call stack
help        -- Helps you using byebug
history     -- Shows byebug's history of commands
info        -- Shows several informations about the program being debugged
interrupt   -- Interrupts the program
irb         -- Starts an IRB session
kill        -- Sends a signal to the current process
list        -- Lists lines of source code
method      -- Shows methods of an object, class or module
next        -- Runs one or more lines of code
pry         -- Starts a Pry session
quit        -- Exits byebug
restart     -- Restarts the debugged program
save        -- Saves current byebug session to a file
```

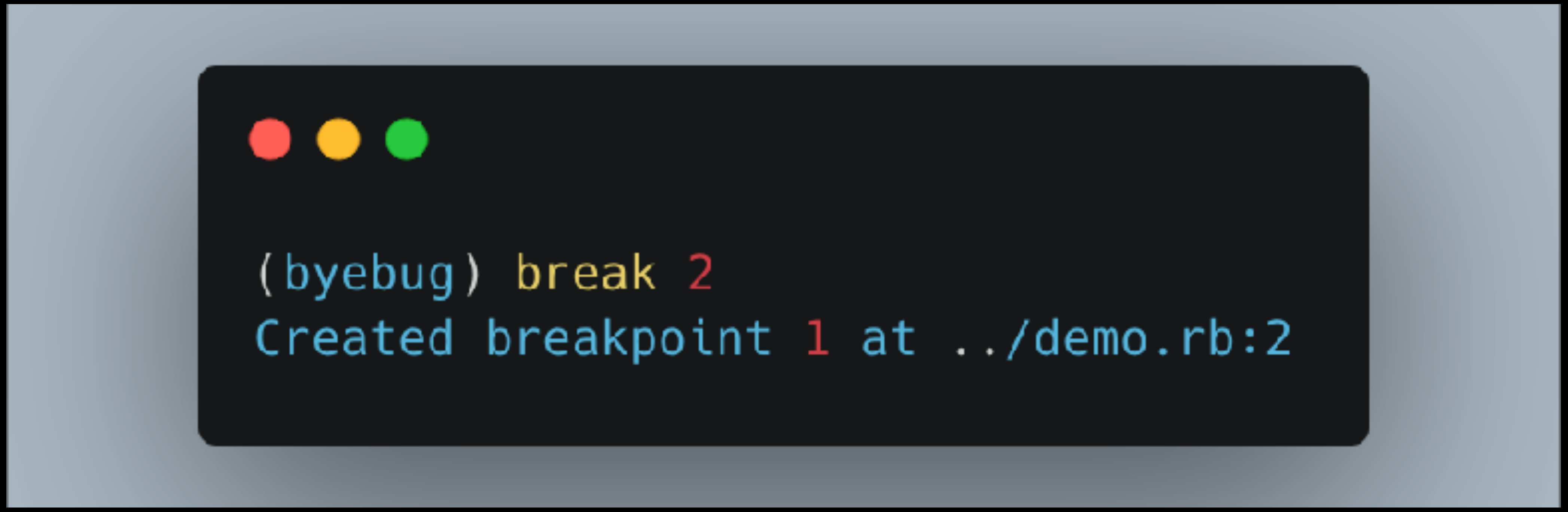
list



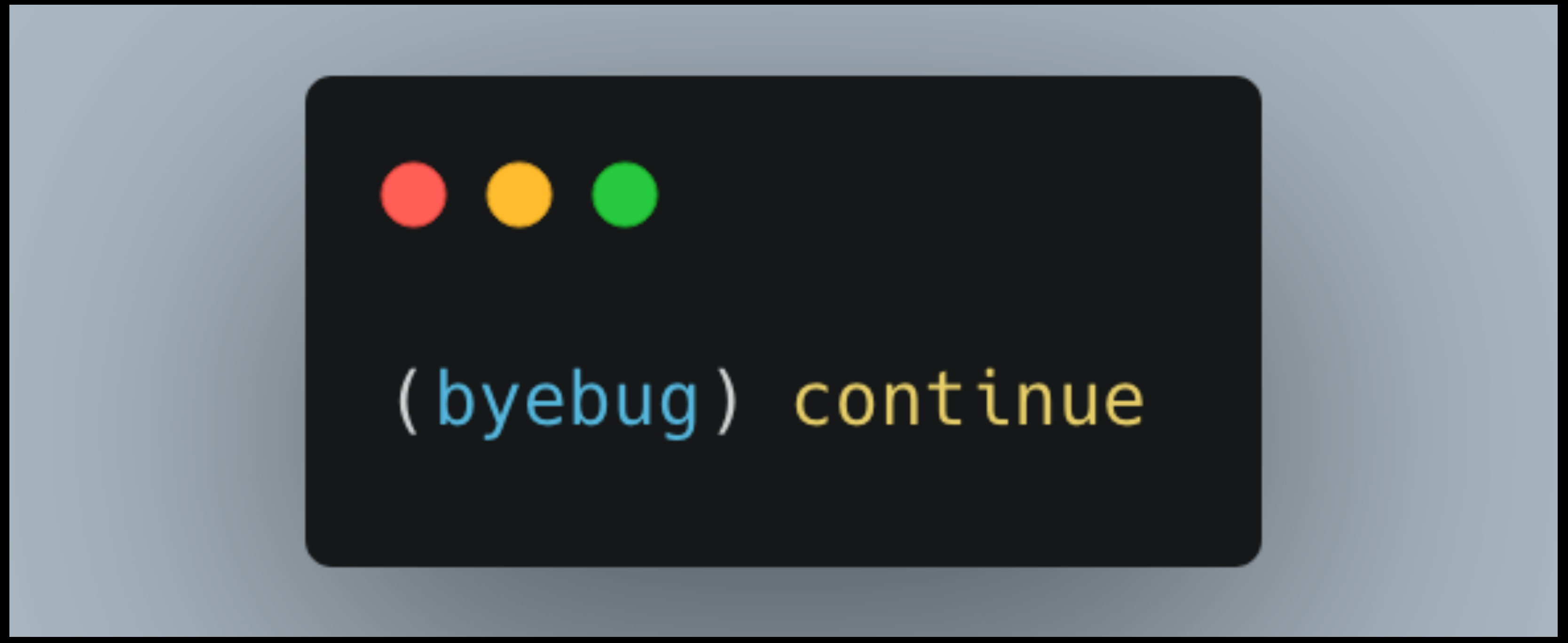
```
(byebug) list

[1, 10] in ../demo.rb
=>  1: def get_referral_discount(user_id, referred)
    2:   if referred[user_id] = true
    3:     return 0
    4:   end
    5:   puts "******"
    6:   referred[user_id] = true
    7:   20
    8: end
    9:
   10: puts get_referral_discount(1, {})
(byebug)
```

break & continue

- Setup breakpoints in byebug.
- continues to next breakpoint / end

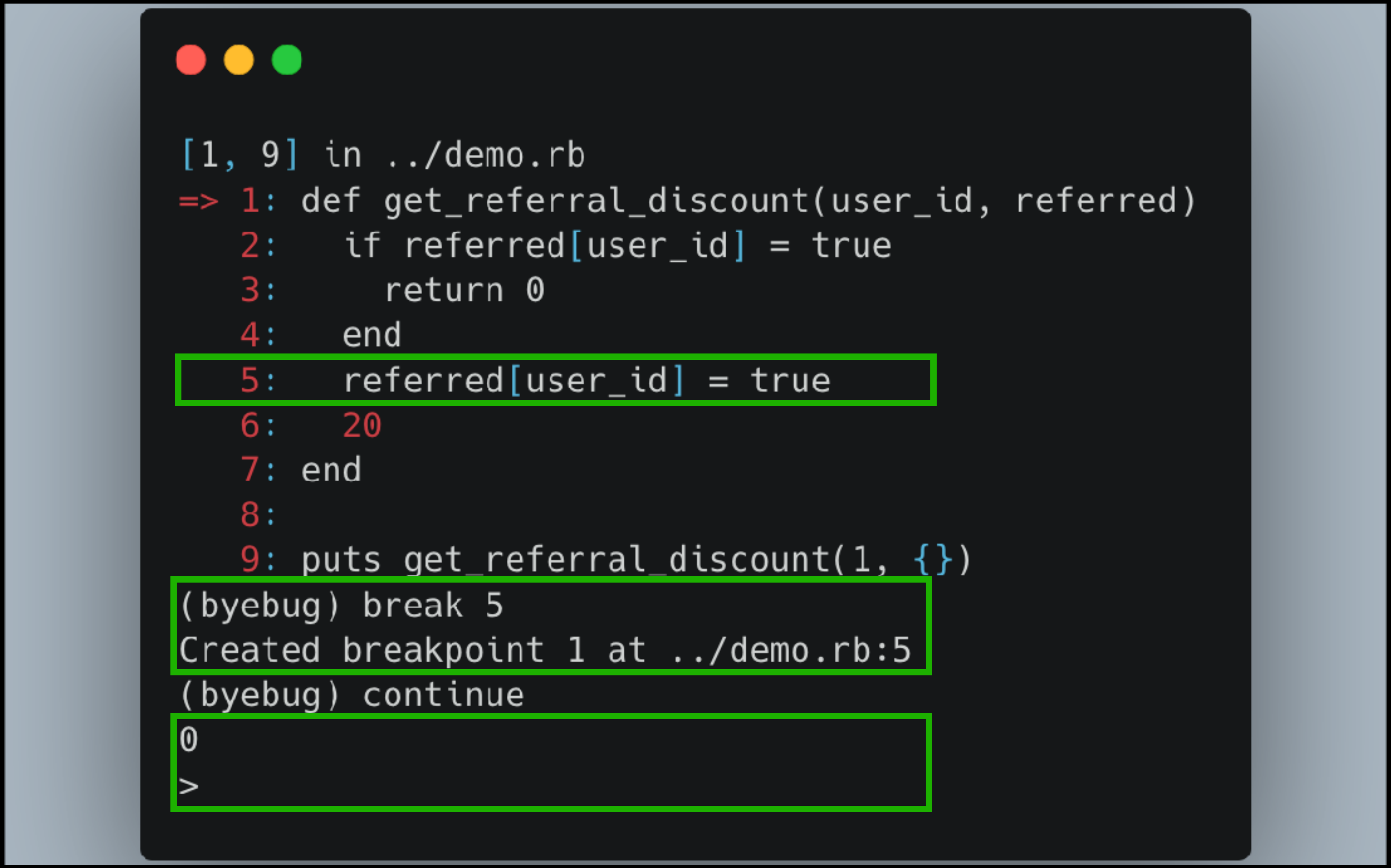


```
(byebug) break 2
Created breakpoint 1 at ../demo.rb:2
```



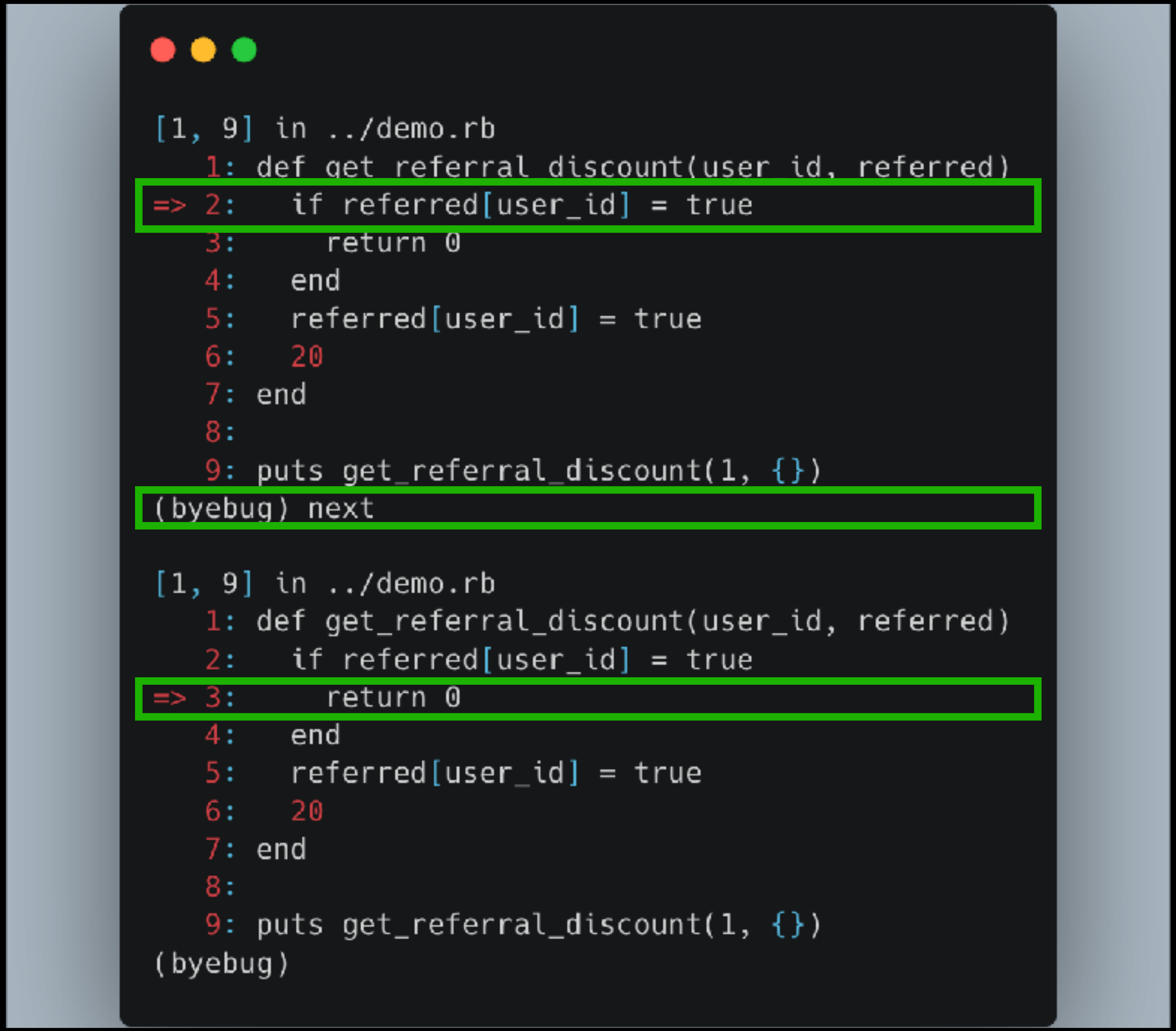
```
(byebug) continue
```

break & continue



```
[1, 9] in ../demo.rb
=> 1: def get_referral_discount(user_id, referred)
   2:   if referred[user_id] = true
   3:     return 0
   4:   end
   5:   referred[user_id] = true
   6:   20
   7: end
   8:
   9: puts get_referral_discount(1, {})
(byebug) break 5
Created breakpoint 1 at ../demo.rb:5
(byebug) continue
0
>
```

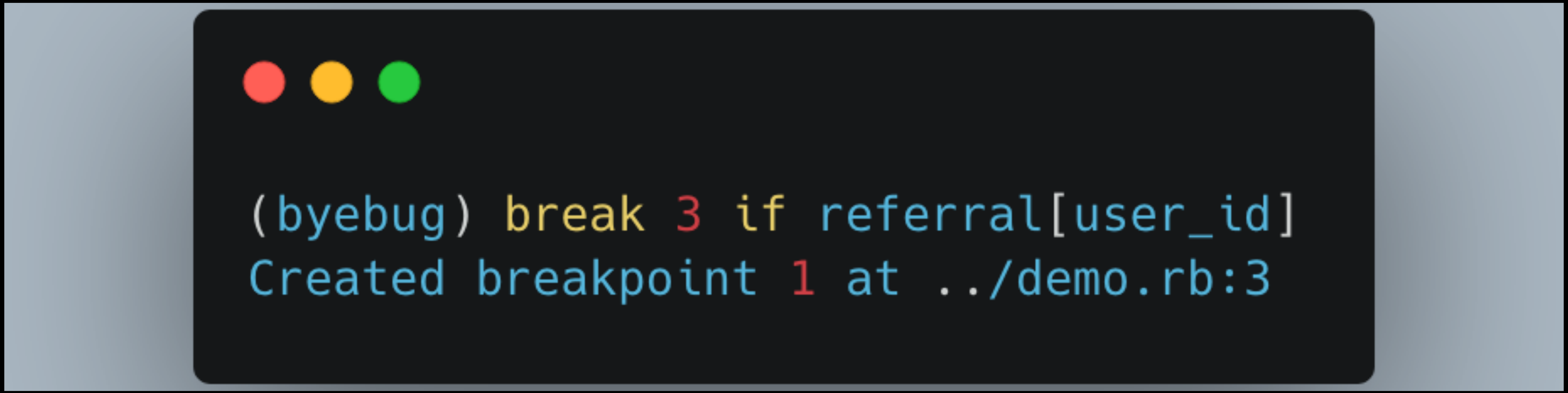
next



```
[1, 9] in ../demo.rb
   1: def get_referral_discount(user_id, referred)
=> 2:   if referred[user_id] = true
   3:     return 0
   4:   end
   5:   referred[user_id] = true
   6:   20
   7: end
   8:
   9: puts get_referral_discount(1, {})
(byebug) next

[1, 9] in ../demo.rb
   1: def get_referral_discount(user_id, referred)
   2:   if referred[user_id] = true
=> 3:     return 0
   4:   end
   5:   referred[user_id] = true
   6:   20
   7: end
   8:
   9: puts get_referral_discount(1, {})
(byebug)
```

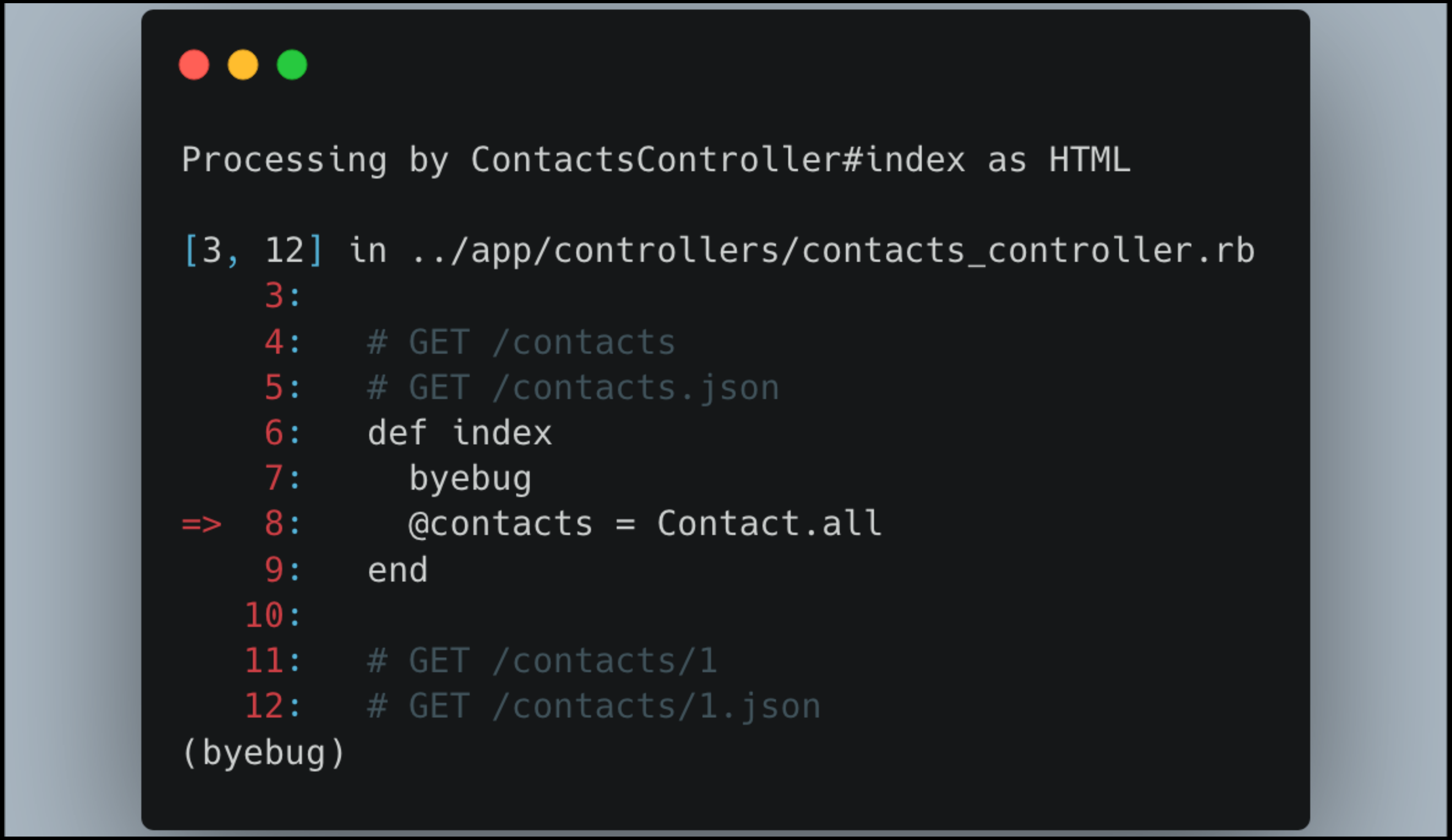
conditional breakpoints

- byebug supports conditional breakpoints
- *break [file:]line [if <expr>*



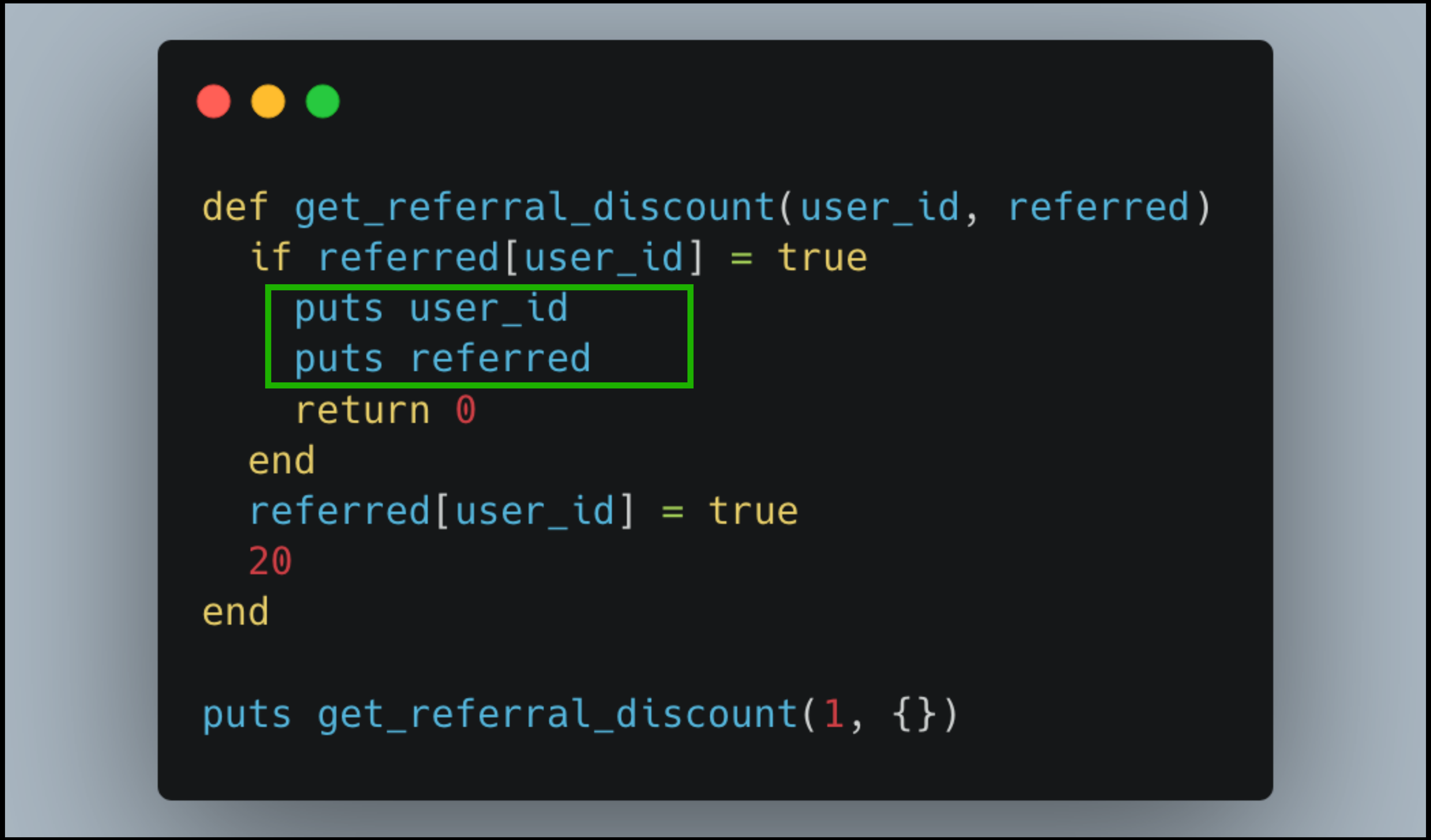
```
(byebug) break 3 if referral[user_id]
Created breakpoint 1 at ../demo.rb:3
```

byebug in rails

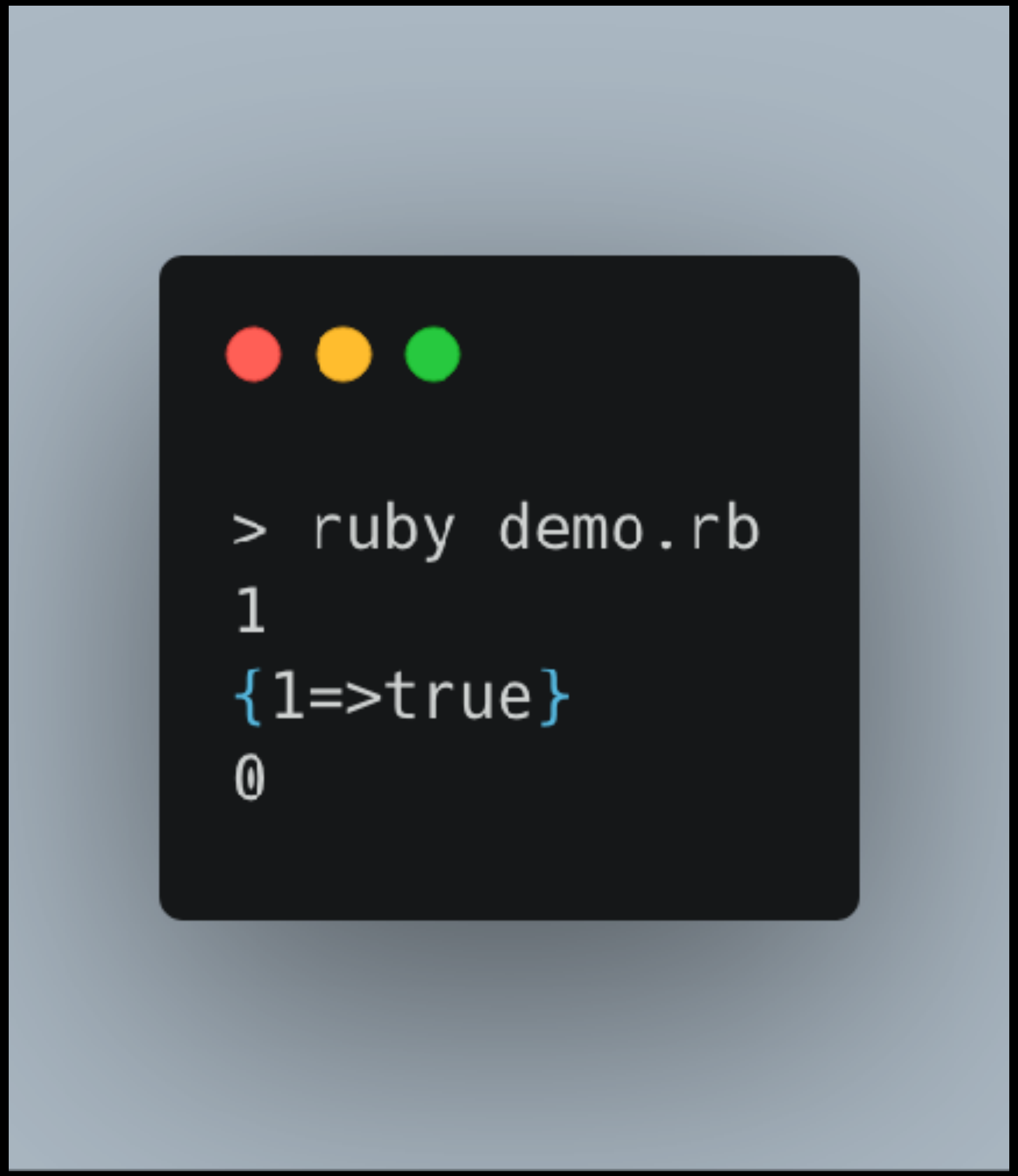


```
Processing by ContactsController#index as HTML

[3, 12] in ../app/controllers/contacts_controller.rb
    3:
    4:   # GET /contacts
    5:   # GET /contacts.json
    6:   def index
    7:     byebug
=>  8:     @contacts = Contact.all
    9:   end
   10:
   11:   # GET /contacts/1
   12:   # GET /contacts/1.json
(byebug)
```

puts variable

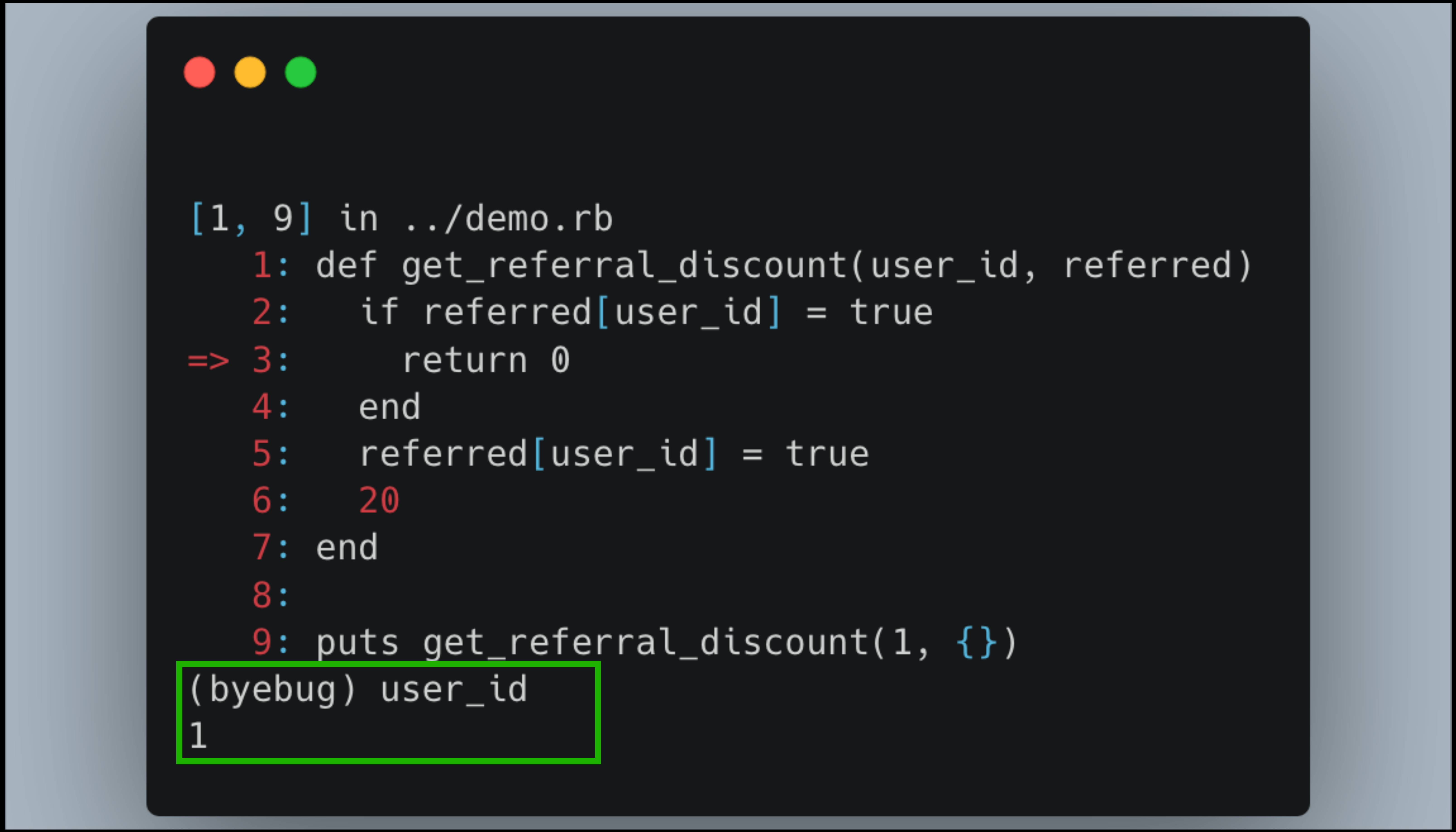


```
def get_referral_discount(user_id, referred)
  if referred[user_id] = true
    puts user_id
    puts referred
    return 0
  end
  referred[user_id] = true
  20
end

puts get_referral_discount(1, {})
```



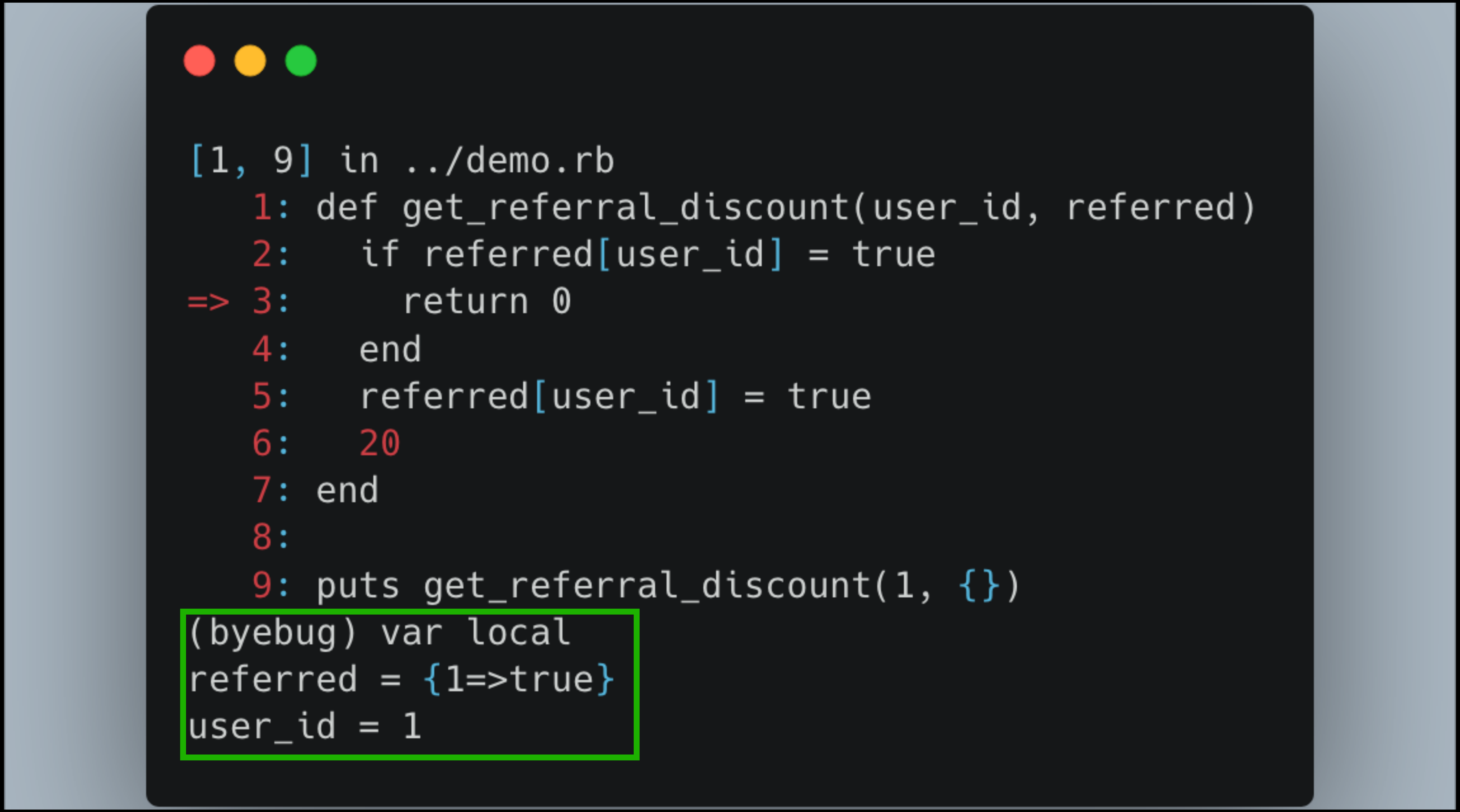
```
> ruby demo.rb
1
{1=>true}
0
```

byebug variable



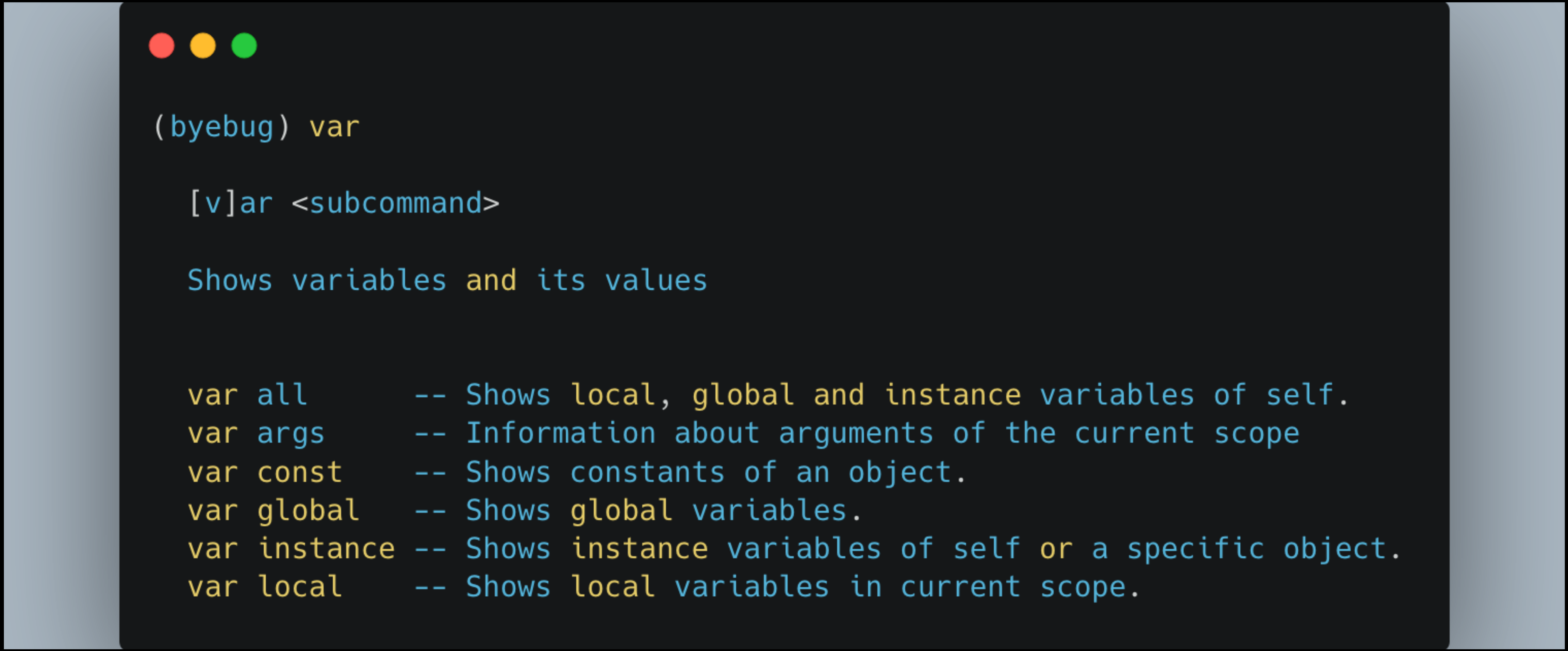
```
[1, 9] in ../demo.rb
   1: def get_referral_discount(user_id, referred)
   2:   if referred[user_id] = true
=> 3:     return 0
   4:   end
   5:   referred[user_id] = true
   6:   20
   7: end
   8:
   9: puts get_referral_discount(1, {})
(byebug) user_id
1
```

byebug local variable



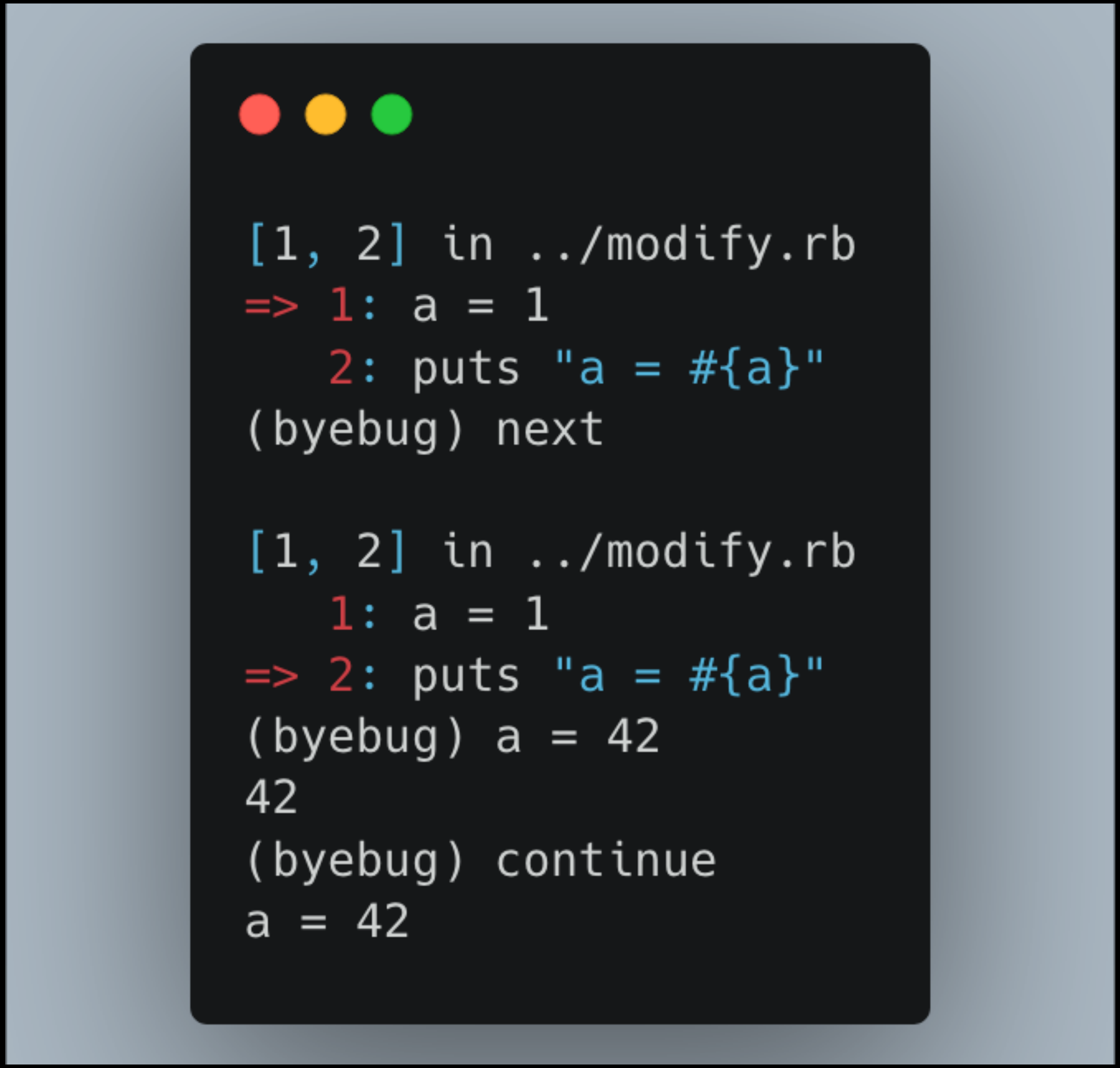
```
[1, 9] in ../demo.rb
   1: def get_referral_discount(user_id, referred)
   2:   if referred[user_id] = true
=> 3:     return 0
   4:   end
   5:   referred[user_id] = true
   6:   20
   7: end
   8:
   9: puts get_referral_discount(1, {})
(byebug) var local
referred = {1=>true}
user_id = 1
```

byebug variable scope



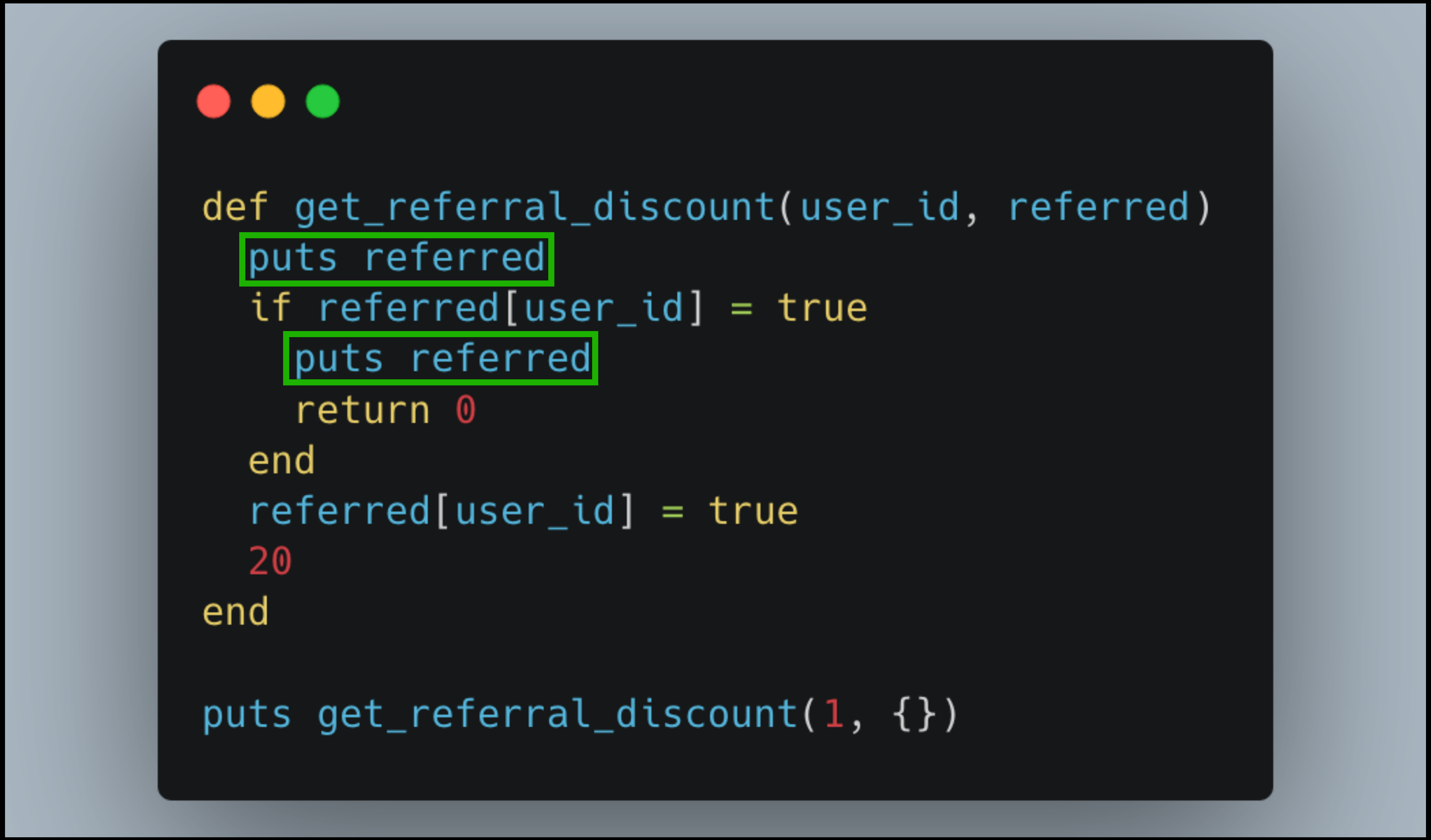
```
(byebug) var

  [v]ar <subcommand>

  Shows variables and its values


  var all      -- Shows local, global and instance variables of self.
  var args     -- Information about arguments of the current scope
  var const    -- Shows constants of an object.
  var global   -- Shows global variables.
  var instance -- Shows instance variables of self or a specific object.
  var local    -- Shows local variables in current scope.
```

byebug mutate variable

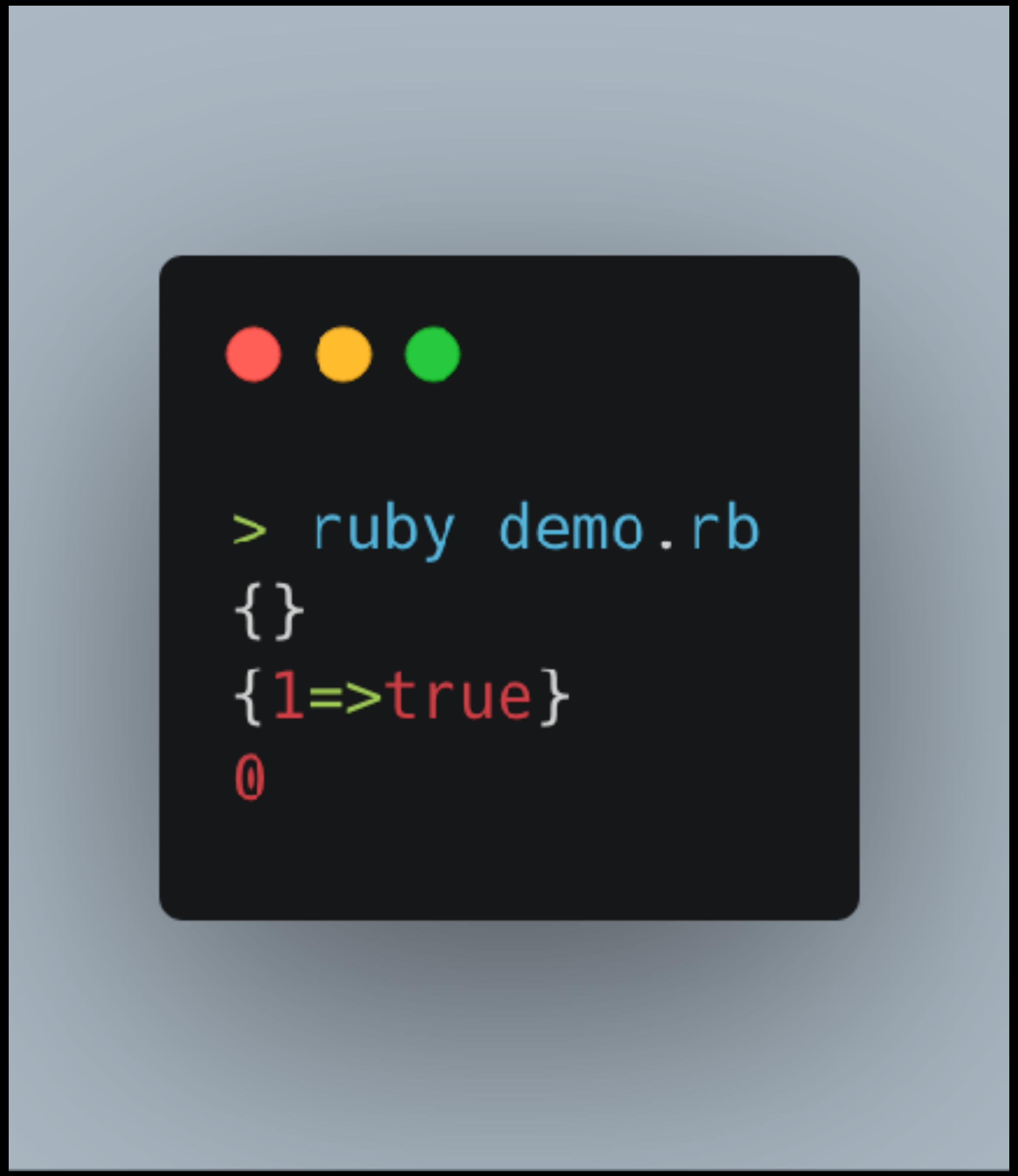


```
[1, 2] in ../modify.rb
=> 1: a = 1
   2: puts "a = #{a}"
(byebug) next

[1, 2] in ../modify.rb
   1: a = 1
=> 2: puts "a = #{a}"
(byebug) a = 42
42
(byebug) continue
a = 42
```

puts watch variable

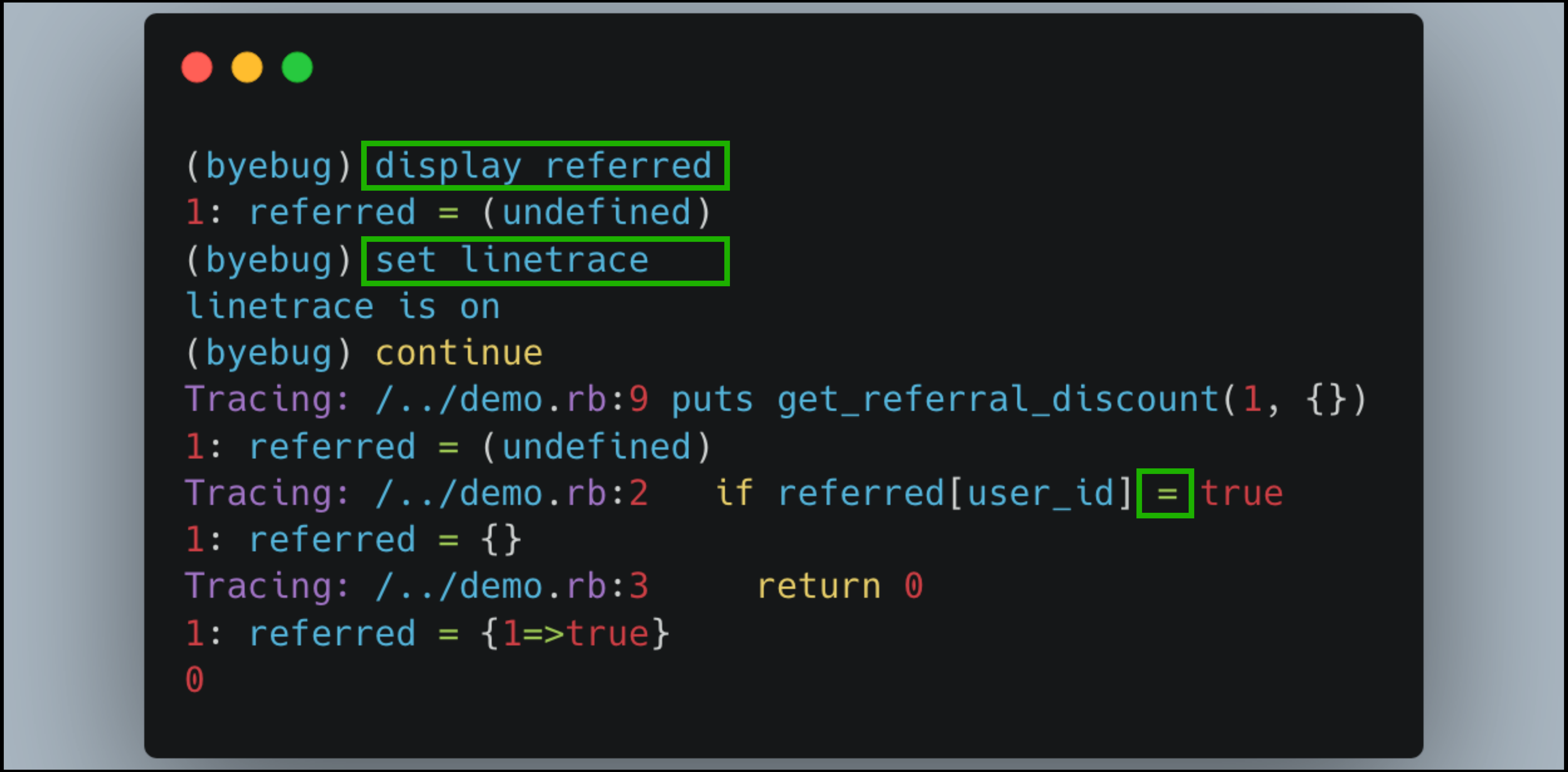


```ruby
def get_referral_discount(user_id, referred)
  puts referred
  if referred[user_id] = true
    puts referred
    return 0
  end
  referred[user_id] = true
  20
end

puts get_referral_discount(1, {})
```



```
> ruby demo.rb
{}
{1=>true}
0
```

byebug watch variable



```
(byebug) display referred
1: referred = (undefined)
(byebug) set linetrace
linetrace is on
(byebug) continue
Tracing: /../demo.rb:9 puts get_referral_discount(1, {})
1: referred = (undefined)
Tracing: /../demo.rb:2   if referred[user_id] = true
1: referred = {}
Tracing: /../demo.rb:3     return 0
1: referred = {1=>true}
0
```

others

- Can be used to debug libraries (`step`)
- Can be used to trace function caller (`where`)

references

- https://github.com/deivid-rodriguez/byebug
- https://guides.rubyonrails.org/debugging_rails_applications.html#debugging-with-the-byebug-gem
- https://www.youtube.com/watch?v=OMjp1T0zmVs

~~question & answer~~

tips & tricks

thanks!