University of Sheffield

COM1001 *Introduction to Software Engineering* SPRING SEMESTER

Debugging

Professor Phil McMinn

My Code Does Not Work!

Oops! Your code stopped working! Now what?

This is a common situation that interrupts our routine, and requires our immediate attention

Because the program mostly working up until now, we assume that something external has crept into our machine…

My Code Does Not Work!

Oops! Your code stopped working! Now what?

This is a common situation that interrupts our routine, and requires our immediate attention

Because the program mostly working up until now, we assume that something external has crept into our machine…

something natural and unavoidable …

My Code Does Not Work!

Oops! Your code stopped working! Now what?

This is a common situation that interrupts our routine, and requires our immediate attention

Because the program mostly working up until now, we assume that something external has crept into our machine…

something natural and unavoidable …

something we are not responsible for …

My Code Does Not Work!

Oops! Your code stopped working! Now what?

This is a common situation that interrupts our routine, and requires our immediate attention

Because the program mostly working up until now, we assume that something external has crept into our machine…

something natural and unavoidable …

something we are not responsible for …

… namely a "bug"

My Code Does Not Work!

Oops! Your code stopped working! Now what?

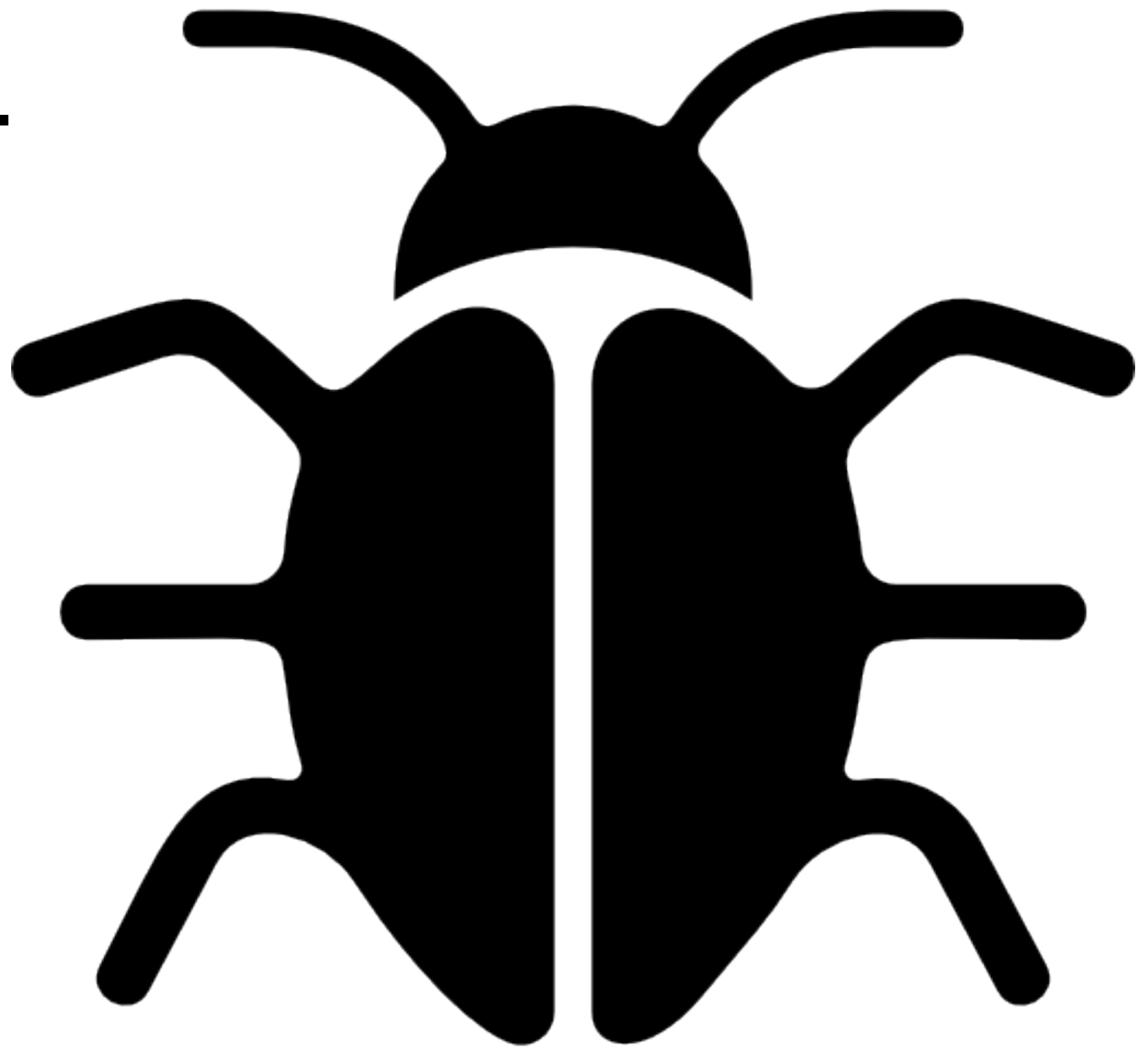
This is a common situation that interrupts our routine, and requires our immediate attention

Because the program mostly working up until now, we assume that something external has crept into our machine…

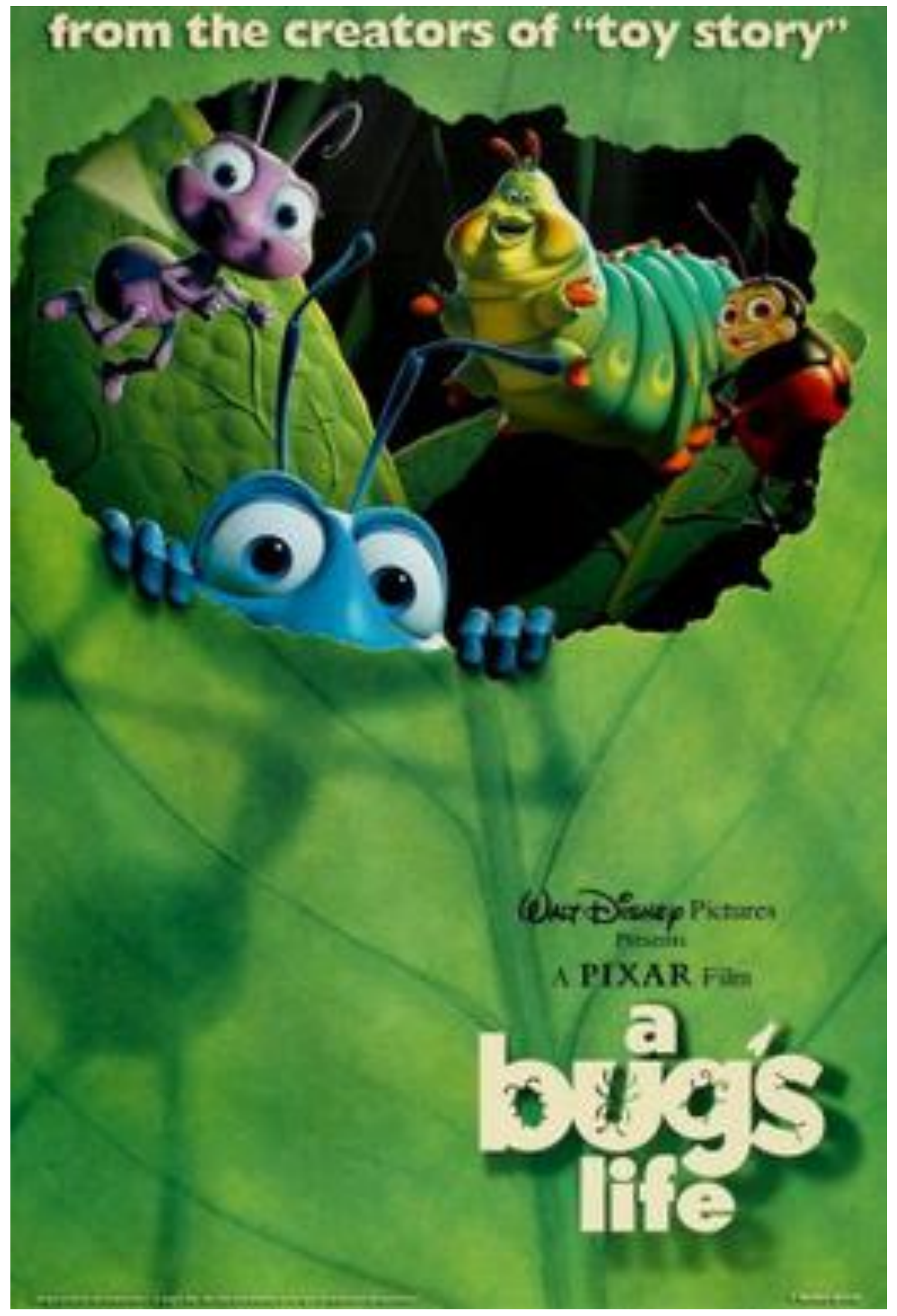
something natural and unavoidable …

something we are not responsible for …

… namely a “bug”



A Bug’s Life


from the creators of "toy story"
Walt Disney Pictures presents
A PIXAR Film
a bug's life

You’ve probably already learned to live with bugs.

You may even think that bugs are unavoidable
when it comes to software.

But as Computer Scientists and Software Engineers, we know that bugs do not creep out of “Mother Nature” and into programs.

At the beginning of any bug story stands a human who wrote the code in question…

I’m Writing a Method…

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  ...
end
```

But First, A Little Recap...

Strings in Ruby are a little bit like arrays of characters.

We can use array-like syntax to find out what character is at what position:

```
irb(main):001:0> str = "hello!"
=> "hello!"
irb(main):002:0> puts str[1]
e
```

0	1	2	3	4	5
h	e	l	l	o	!

```
irb(main):003:0> puts str[2]
l
```

0	1	2	3	4	5
h	e	l	l	o	!

If we try to index a character that does not exist, Ruby returns `nil`. "`nil`" is Ruby's way of saying "nothing". It's similar to `null` in Java.

```
irb(main):004:0> puts "is nil" if str[6].nil?
is nil
```

0	1	2	3	4	5	6	...
h	e	l	l	o	!	nil	nil

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

debugging/string_comparison_buggy.rb

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

`debugging/string_comparison_buggy.rb`

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

debugging/string_comparison_buggy.rb

Here I'm attempting to loop though each index of the two strings. I compare up to the index corresponding to the length of the smallest string.

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

debugging/string_comparison_buggy.rb

Here I'm attempting to loop though each index of the two strings. I compare up to the index corresponding to the length of the smallest string.

In the body of the loop, I'm comparing the character at each index `i` of the string. If they're not equal, I return `i`, since the characters are different

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

debugging/string_comparison_buggy.rb

Here I'm attempting to loop though each index of the two strings. I compare up to the index corresponding to the length of the smallest string.

In the body of the loop, I'm comparing the character at each index `i` of the string. If they're not equal, I return `i`, since the characters are different

If the loop terminates, my method hasn't found two different characters, otherwise it would have already have exited the method via the return statement in the loop body itself. So I return -1. Because it's the last line of the method doesn't require us to use the `return` keyword.

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

`debugging/string_comparison_buggy.rb`

Sometimes it works:

If I give it the arguments “`aardvark`” and “`aargh!`”, it gives me the answer **3**

Back to my method...

I want to find the index of the first character that differs between two strings, returning -1 if the strings are the same:

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

`debugging/string_comparison_buggy.rb`

Sometimes it works:

If I give it the arguments “`aardvark`” and “`aargh!`”, it gives me the answer **3**

Sometimes it doesn’t work:

If I give it the arguments “`sought`” and “`bought`”, it gives me the answer **-1**

Defects

My code contains a bug. Or more precisely, a **defect**.

```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

Defects

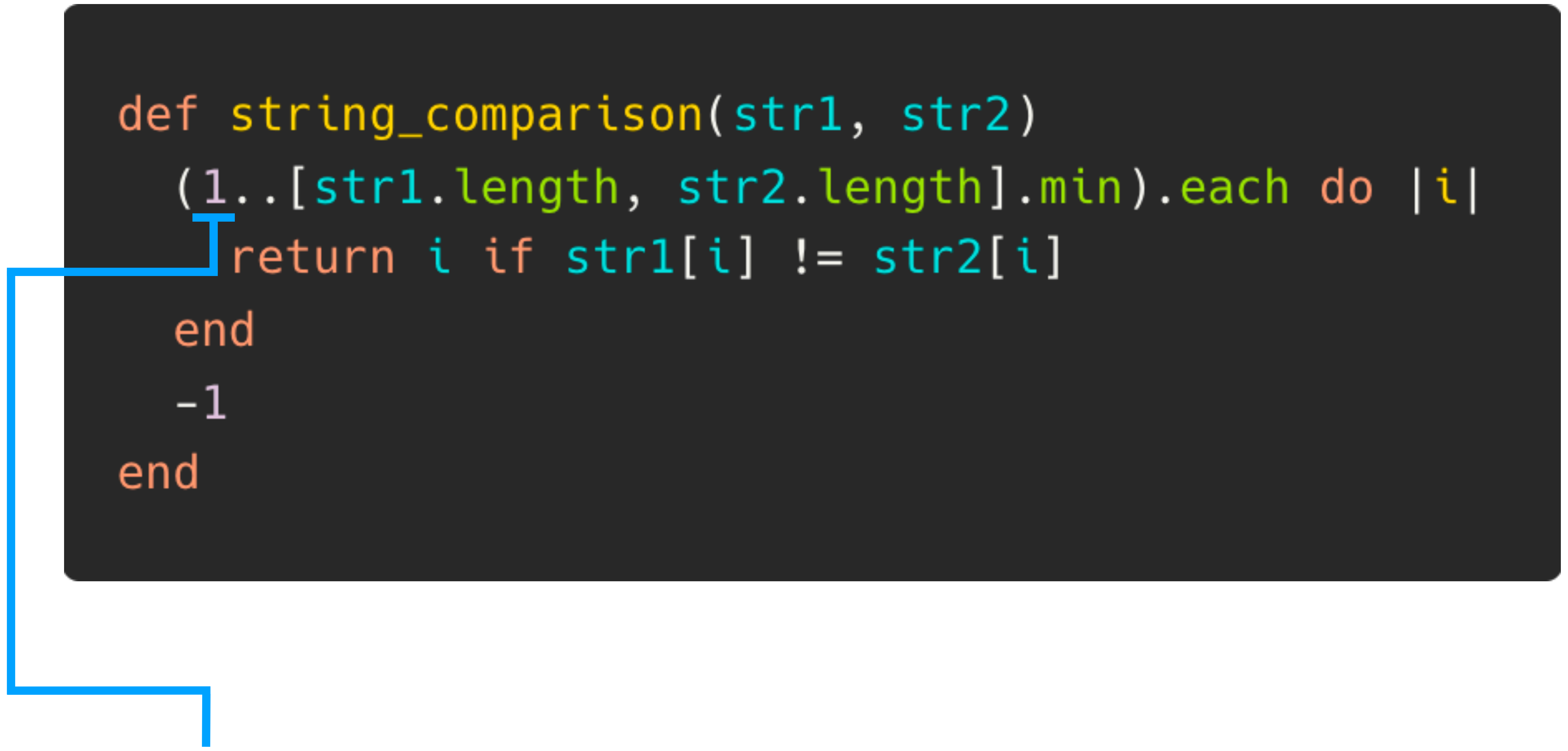
My code contains a bug. Or more precisely, a **defect**.

```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

Where is the **defect**?

Defects

My code contains a bug. Or more precisely, a **defect**.



```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

Where is the **defect**?

Defects

My code contains a bug. Or more precisely, a **defect**.

```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

Where is the **defect**?

How did it cause the method to fail?

From Defects to Failures

From Defects to Failures

1 The programmer creates a **defect**

… but is not always technically at fault – the defect may have originated from a poorly described set of requirements

From Defects to Failures

1. The programmer creates a **defect**

   … but is not always technically at fault – the defect may have originated from a poorly described set of requirements

2. The defect causes an **infection**

   The program, with the defect, is executed. The defect **infects** the intended state of the program.

From Defects to Failures

1. **The programmer creates a defect**

   … but is not always technically at fault – the defect may have originated from a poorly described set of requirements

2. **The defect causes an infection**

   The program, with the defect, is executed. The defect **infects** the intended state of the program.

3. **The infection propagates**

   The infected program state spreads into future states of the program – for example, variables taking on the wrong values, or the wrong parts of the program being executed.

From Defects to Failures

1. The programmer creates a **defect**

   … but is not always technically at fault – the defect may have originated from a poorly described set of requirements

2. The defect causes an **infection**

   The program, with the defect, is executed. The defect **infects** the intended state of the program.

3. The infection **propagates**

   The infected program state spreads into future states of the program – for example, variables taking on the wrong values, or the wrong parts of the program being executed.

4. The infection causes a **failure**

   A failure is an externally observable error in the program’s behaviour.

Debugging Video

Debugging

Debugging is the process of tracing a **failure** back to the **defect** that caused it, and then fixing the defect.

We try to follow a failure backwards through the program, from the faulty output to the infected states of the program, to the program line(s) containing the defect that caused it.

Debugging

Debugging is the process of tracing a **failure** back to the **defect** that caused it, and then fixing the defect.

We try to follow a failure backwards through the program, from the faulty output to the infected states of the program, to the program line(s) containing the defect that caused it.

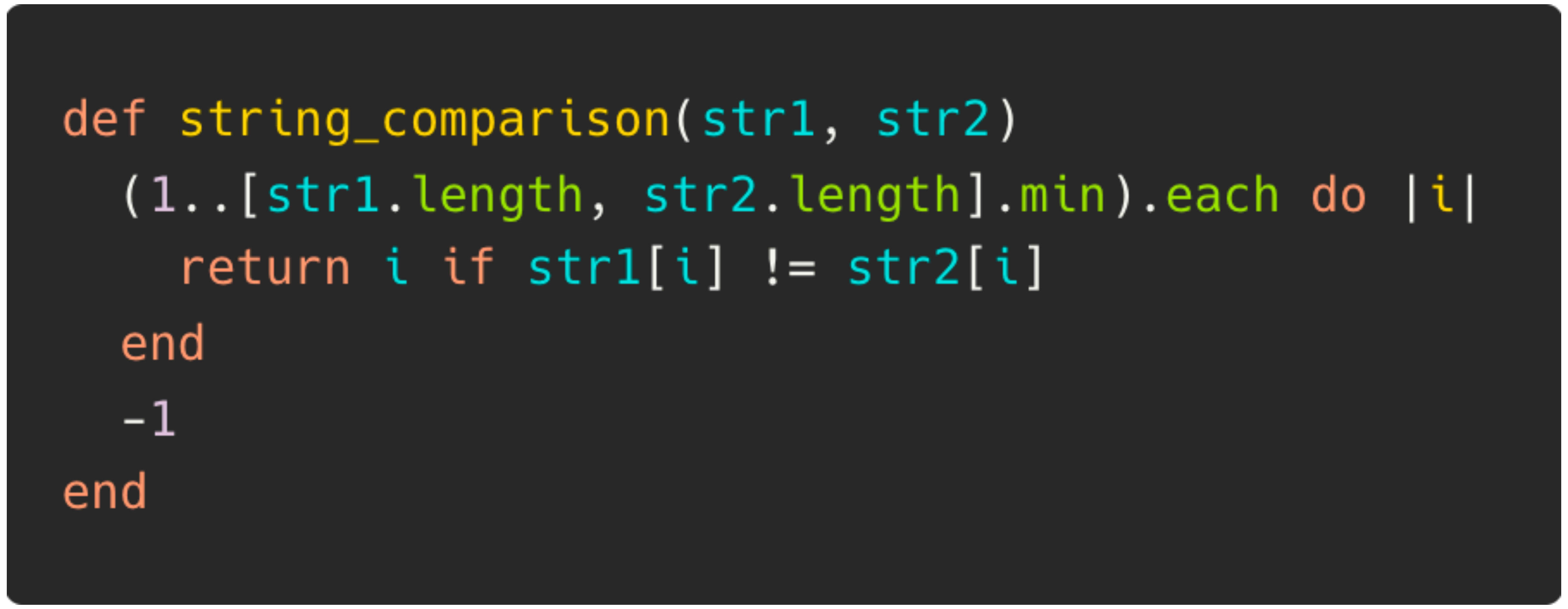
Debugging is mystery solving with a bit of time travel thrown in …

Step 1 – Reproduce the Failure

Easy in this case – we just give it the failing inputs again – “sought” and “bought”.

We call this failure-inducing inputs a **test case**.

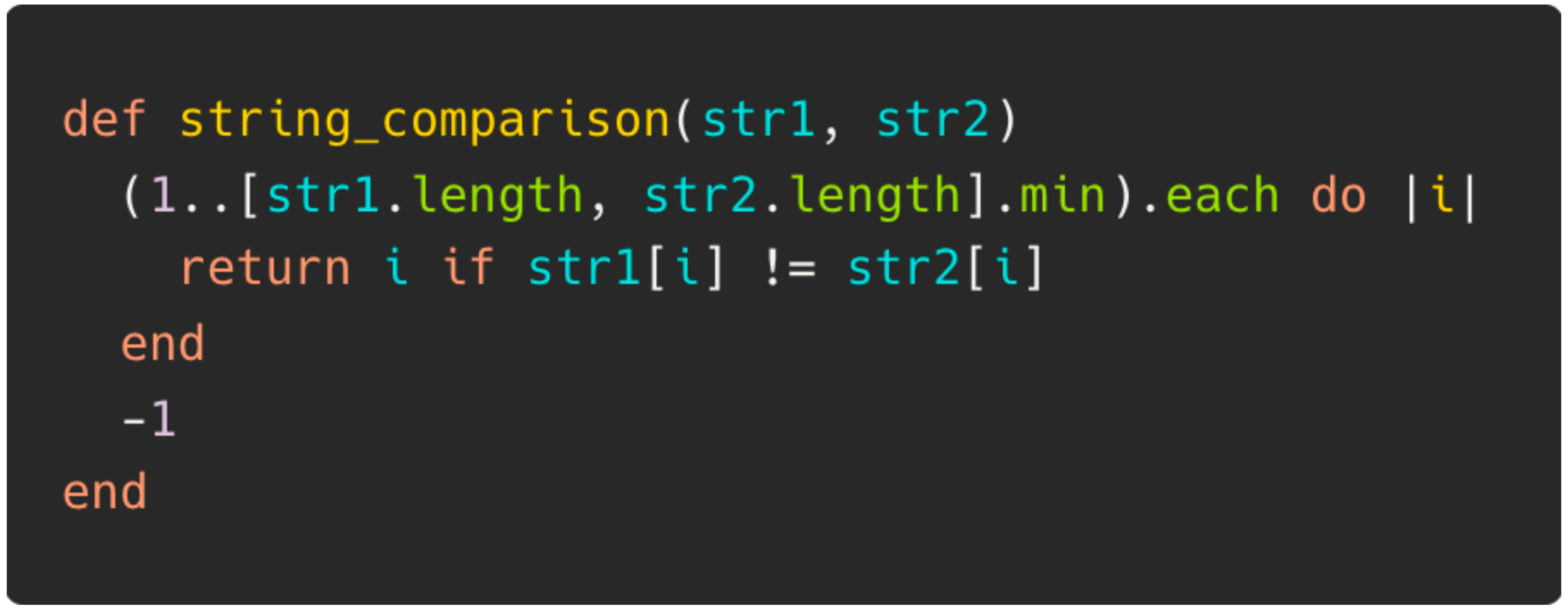
Reproducing the failure is not always this easy.



```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

Step 2 – Simplify the Test Case

We can simplify “sought” and “bought” to “s” and “b” and we still get the wrong answer.



```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

Step 3 – Focus on the Likely Origins

We can do this by temporarily **instrumenting** the program code.

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

debugging/string_comparison_buggy_instrumented.rb

Step 3 – Focus on the Likely Origins

We can do this by temporarily **instrumenting** the program code.

By **tracing** the flow of execution we can work out the path that the program executed and whether decisions in the program were taken as expected

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

debugging/string_comparison_buggy_instrumented.rb

Step 3 – Focus on the Likely Origins

We can do this by temporarily **instrumenting** the program code.

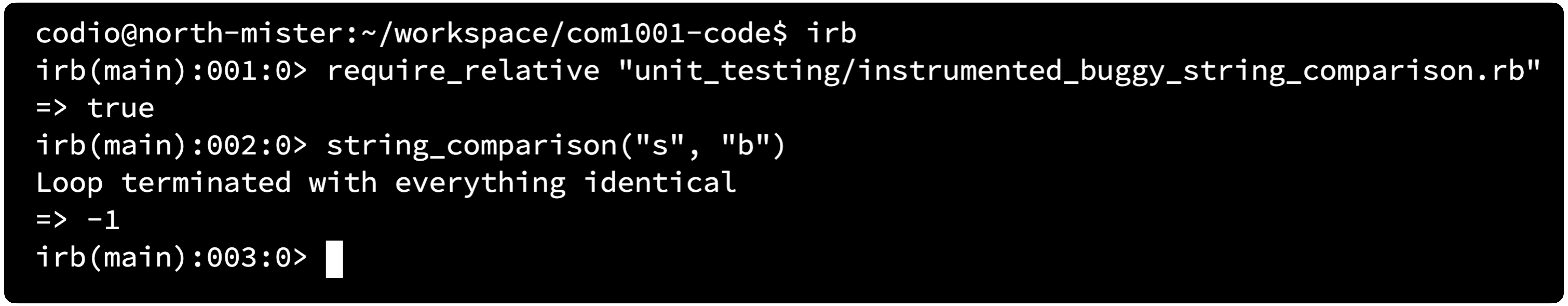
By **tracing** the flow of execution we can work out the path that the program executed and whether decisions in the program were taken as expected

```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

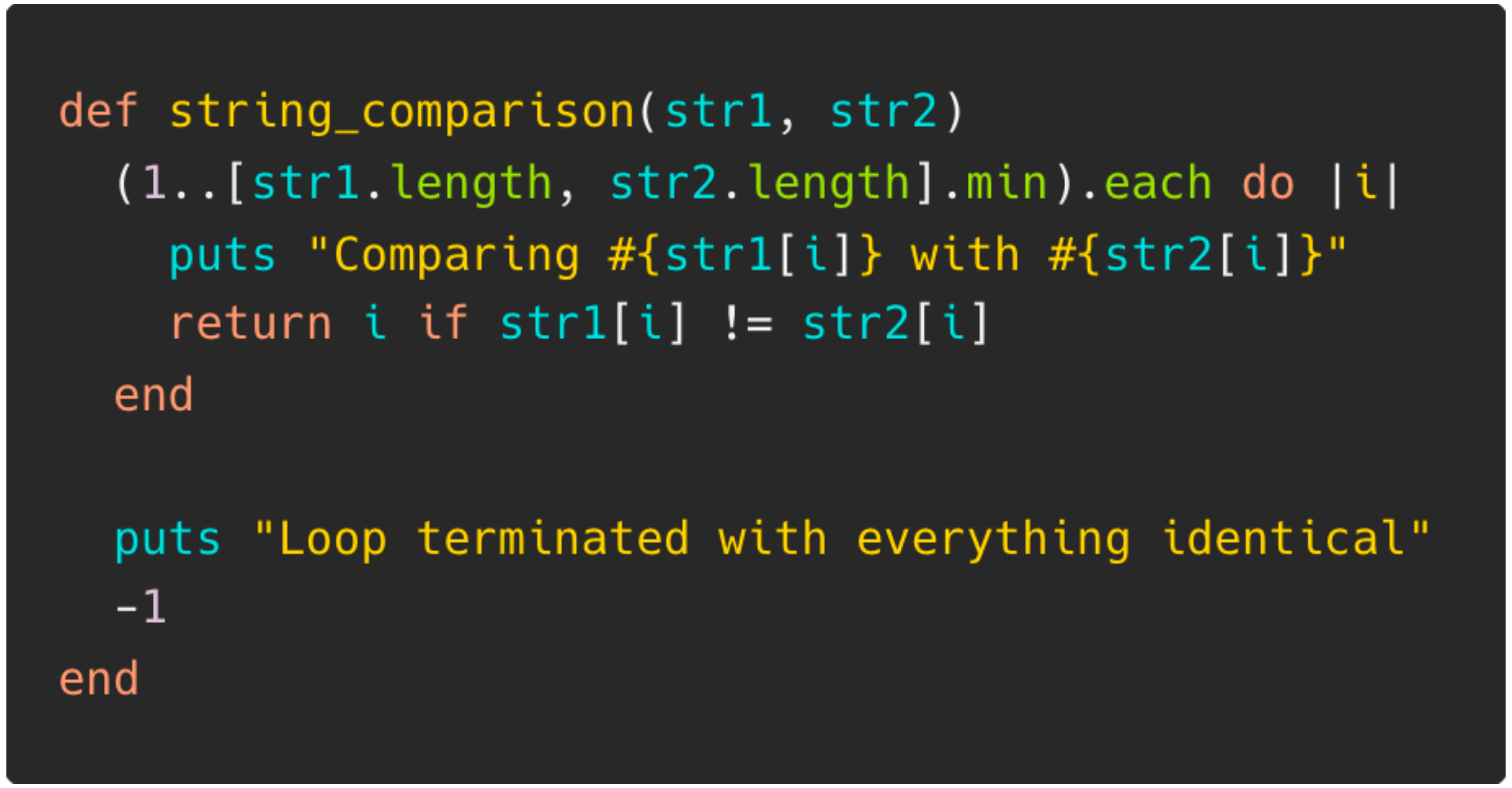
debugging/string_comparison_buggy_instrumented.rb

By printing out the values of variables we can **watch** them to see if they become **infected** with the wrong values

Step 4 – Understand How the Defect Came to Be

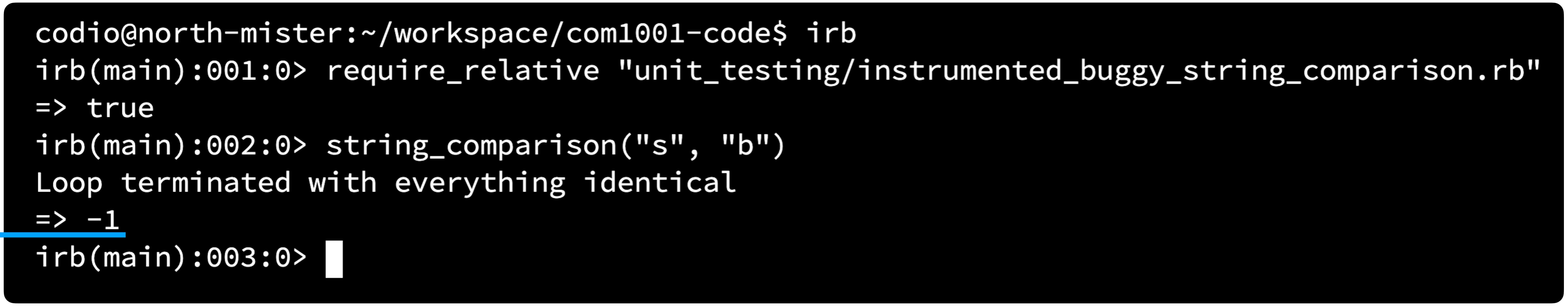


```
codio@north-mister:~/workspace/com1001-code$ irb
irb(main):001:0> require_relative "unit_testing/instrumented_buggy_string_comparison.rb"
=> true
irb(main):002:0> string_comparison("s", "b")
Loop terminated with everything identical
=> -1
irb(main):003:0>
```



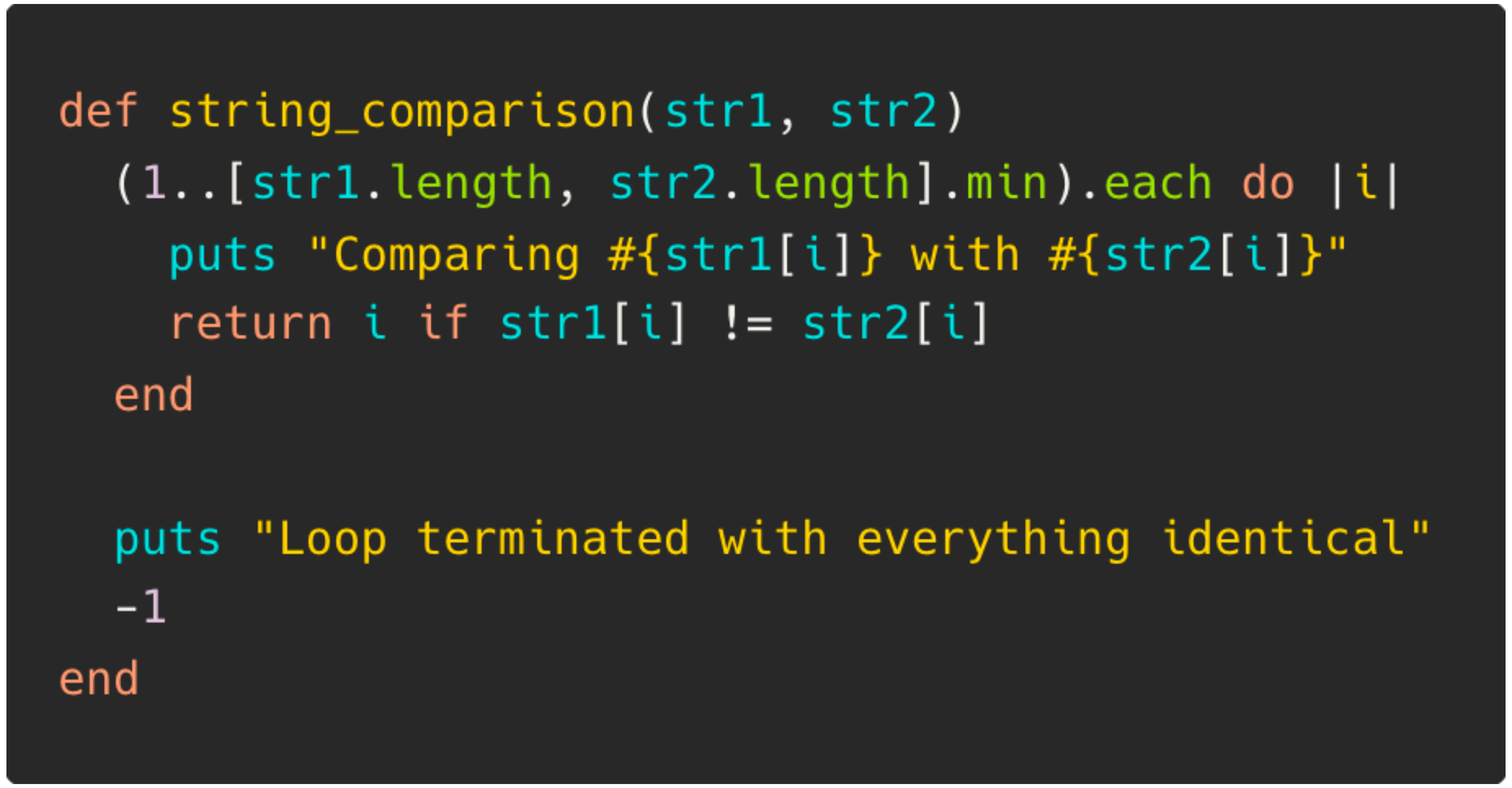
```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

Step 4 – Understand How the Defect Came to Be



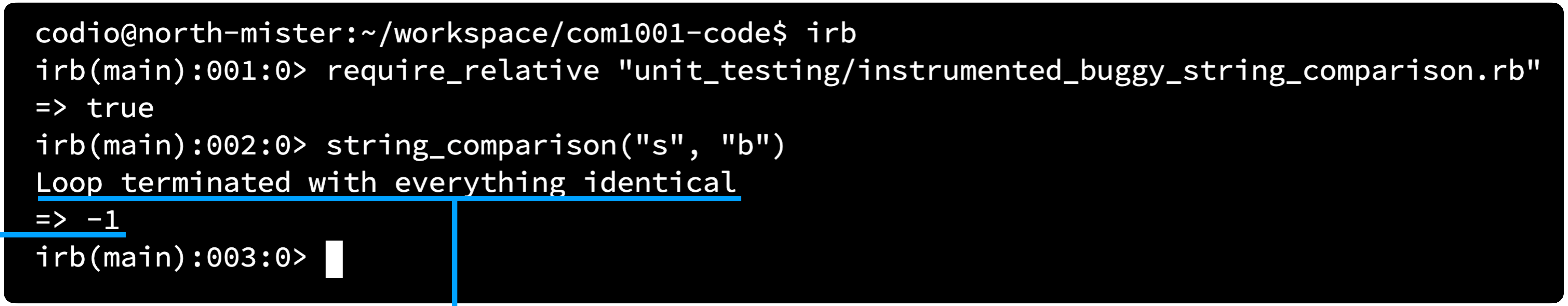
```
codio@north-mister:~/workspace/com1001-code$ irb
irb(main):001:0> require_relative "unit_testing/instrumented_buggy_string_comparison.rb"
=> true
irb(main):002:0> string_comparison("s", "b")
Loop terminated with everything identical
=> -1
irb(main):003:0>
```

The return value is **-1** rather than **0**



```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

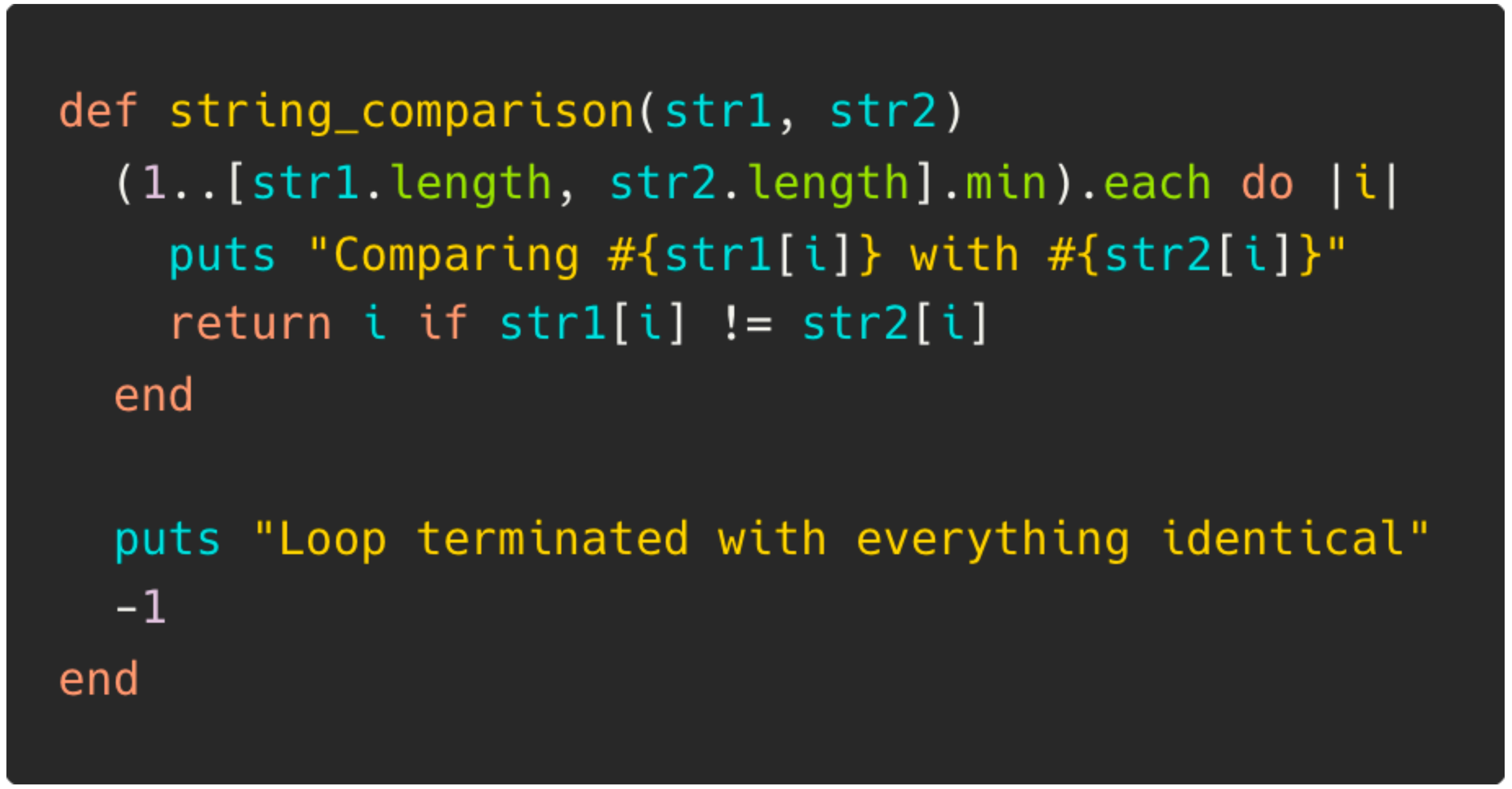
Step 4 – Understand How the Defect Came to Be



```
codio@north-mister:~/workspace/com1001-code$ irb
irb(main):001:0> require_relative "unit_testing/instrumented_buggy_string_comparison.rb"
=> true
irb(main):002:0> string_comparison("s", "b")
Loop terminated with everything identical
=> -1
irb(main):003:0>
```

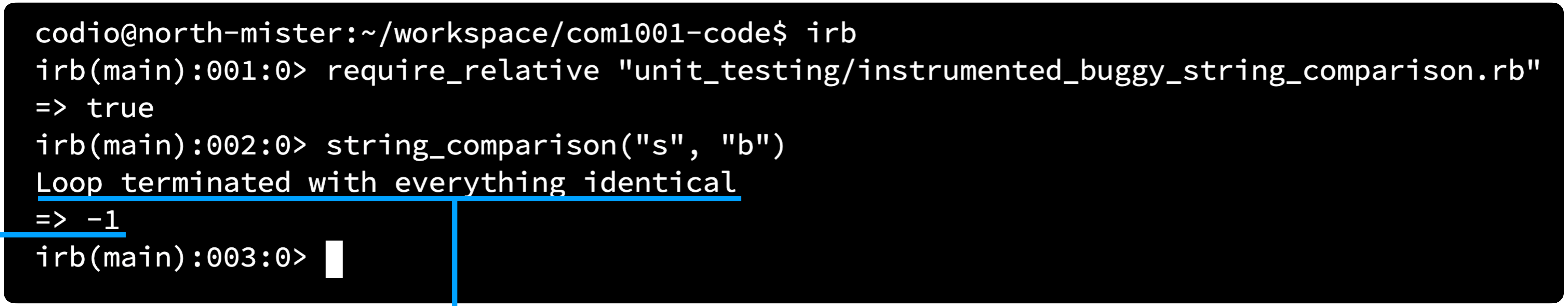
The return value is **-1** rather than **0**

We can see from the printouts to the terminal window this because the loop is not entered.



```
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

Step 4 – Understand How the Defect Came to Be

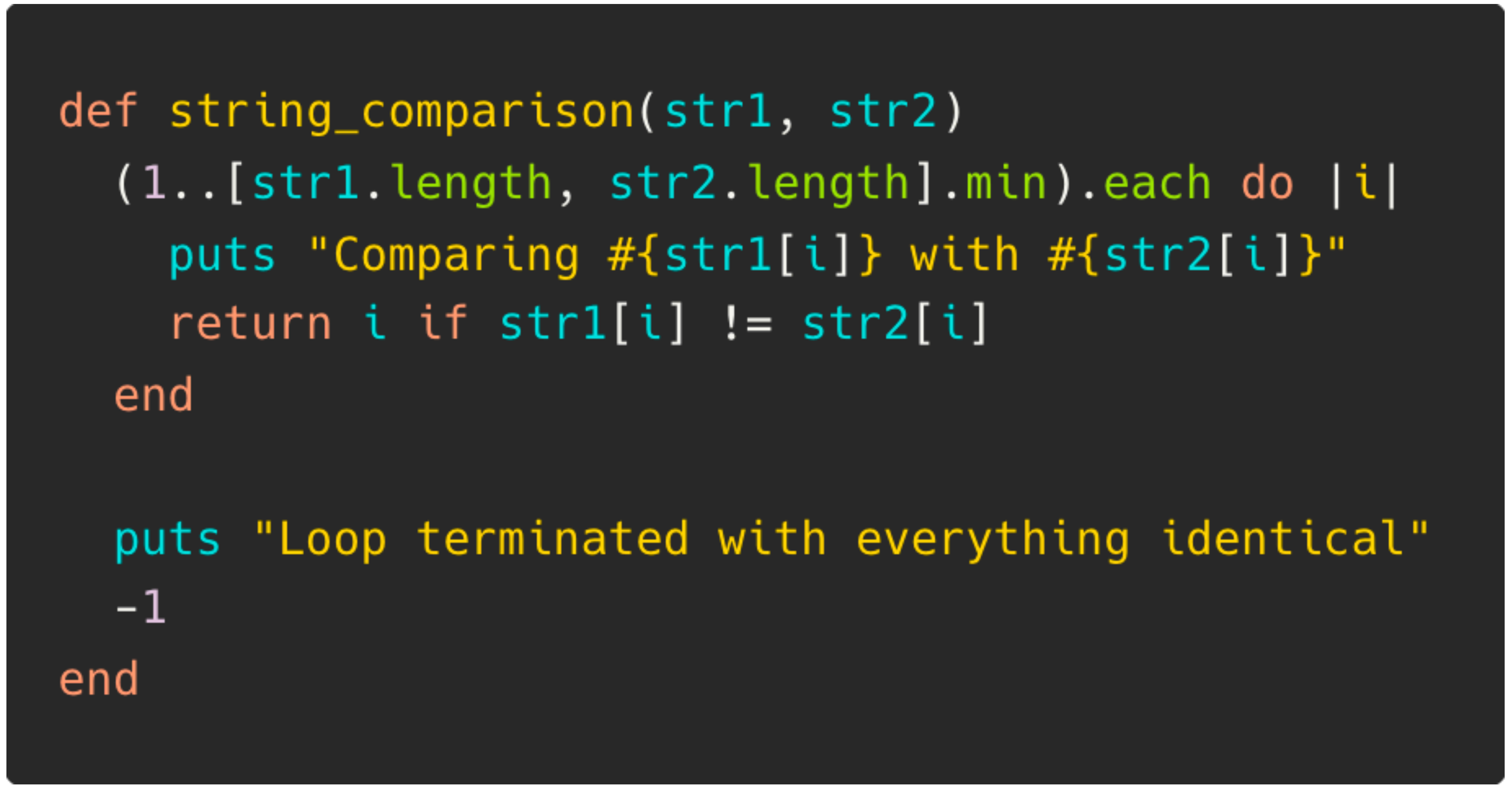


```
codio@north-mister:~/workspace/com1001-code$ irb
irb(main):001:0> require_relative "unit_testing/instrumented_buggy_string_comparison.rb"
=> true
irb(main):002:0> string_comparison("s", "b")
Loop terminated with everything identical
=> -1
irb(main):003:0>
```

The return value is **-1** rather than **0**

We can see from the printouts to the terminal window this because the loop is not entered.

This is because the loop begins iterating from index 1, rather than 0. That is, the first character is not considered.



```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    puts "Comparing #{str1[i]} with #{str2[i]}"
    return i if str1[i] != str2[i]
  end

  puts "Loop terminated with everything identical"
  -1
end
```

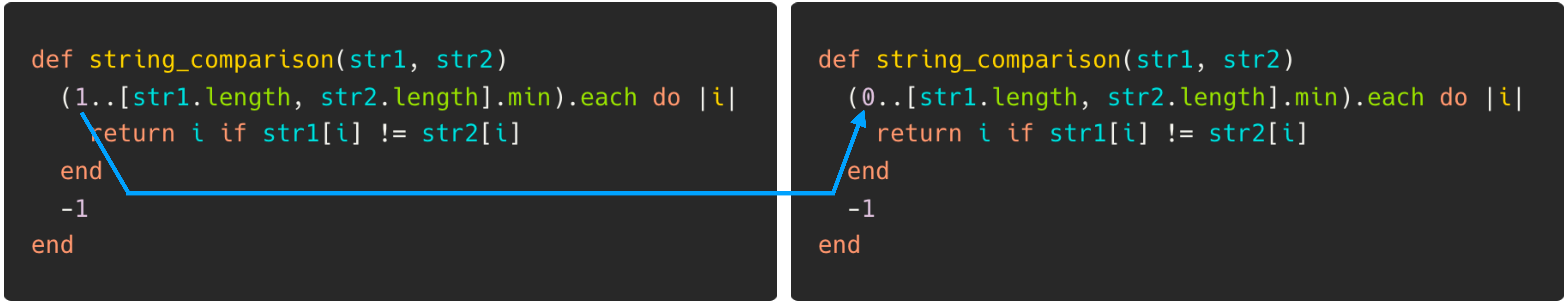
Step 5 – Fix the Defect

```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

```ruby
def string_comparison(str1, str2)
  (0..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

debugging/string_comparison.rb

Step 5 – Fix the Defect



```ruby
def string_comparison(str1, str2)
  (1..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

```ruby
def string_comparison(str1, str2)
  (0..[str1.length, str2.length].min).each do |i|
    return i if str1[i] != str2[i]
  end
  -1
end
```

debugging/string_comparison.rb

Review: Steps Involved in Debugging

Review: Steps Involved in Debugging

1. **Reproduce** the failure

   What circumstances caused the program to fail?

Review: Steps Involved in Debugging

1. **Reproduce** the failure

   What circumstances caused the program to fail?

2. **Simplify** the test case

   What is the simplest set of circumstances that cause the program to fail?

Review: Steps Involved in Debugging

1. **Reproduce** the failure

   What circumstances caused the program to fail?

2. **Simplify** the test case

   What is the simplest set of circumstances that cause the program to fail?

3. **Focus** on the likely origins

   Instrument the program to track the flow of execution – **tracing** – and the values of variables – **watching**

Review: Steps Involved in Debugging

1. **Reproduce** the failure

   What circumstances caused the program to fail?

2. **Simplify** the test case

   What is the simplest set of circumstances that cause the program to fail?

3. **Focus** on the likely origins

   Instrument the program to track the flow of execution – **tracing** – and the values of variables – **watching**

4. **Understand** how the defect came to be

   Work through the program traces and values of variables, tracking infected program states back to their origin – the original defect

Review: Steps Involved in Debugging

1. **Reproduce** the failure

   What circumstances caused the program to fail?

2. **Simplify** the test case

   What is the simplest set of circumstances that cause the program to fail?

3. **Focus** on the likely origins

   Instrument the program to track the flow of execution – **tracing** – and the values of variables – **watching**

4. **Understand** how the defect came to be

   Work through the program traces and values of variables, tracking infected program states back to their origin – the original defect

5. **Fix** the Defect