Full Stack Notes

Scope and More Resources

Table of Contents

Scope defines where in a program a variable is accessible.

Scope

When a variable is accessible it is said to be “in scope”. Wherever a variable is not accessible by your program it is said to be “out of scope” or “outside the current scope.”

From the Poignant Guide:

> *"Microscopes narrow and magnify your vision. Telescopes extend the range of your vision. In Ruby, scope refers to a field of vision inside methods and blocks."*

Scope and Methods

Variables can be passed into a method as arguments. But, methods cannot 'see' variables defined outside its scope.

Data can also be returned from a method. But, variables defined within a method are only accessible until the method ends.

```
# How many legs are there in total if we have
# 'creature_count' number of 'creature's?
def leg_count creature, creature_count
  if creature == 'human'
    legs = 2 * creature_count
  elsif creature == 'spider'
    legs = 8 * creature_count
  else
    legs = 4 * creature_count
  end
  "#{legs} #{creature} legs in total."
```

```
end


puts leg_count 'spider', 4
puts legs # This variable should not be available in the current scope.
```

Output:

```
32 spider legs in total.
NameError: undefined local variable or method 'legs'
```

Scope and Blocks

Unlike methods, blocks can 'see' and modify variables that are defined in their vicinity. This can lead to confusing code:

```
fruit = 'dragon fruit'   # The fruit variable *will not* be overwritten by the loop.
double_fruit = fruit * 2 # The double_fruit variable *will* be overwritten by the loop.


['apple','pear','banana'].each do |fruit| # Creates a second fruit variable with a separate scope.
  double_fruit = fruit * 2  # double_fruit refers to the variable defined above the block.
  puts "I ate one #{fruit}. Doubled: #{double_fruit}"
```

```
end


puts "It's true I ate one #{fruit}." # Still 'dragon fruit'
puts "But the double fruit is #{double_fruit}!" # 'bananabanana'
```

Here the `double_fruit` variable within the block was the same variable as the one defined above.

But there were two versions of the `fruit` variable! It's confusing when block arguments have the same name as other variables.

Output:

```
I ate one apple. Doubled: appleapple
I ate one pear. Doubled: pearpear
I ate one banana. Doubled: bananabanana
It's true I ate one dragon fruit.
But the double fruit is bananabanana!
```

Resources

Free Ruby EBooks

- Why's Poignant Guide (Extra Silly)

- Humble Little Ruby Book (Medium Silly)

Ruby Tutorials

- Try Ruby in your Browser
- Ruby in Twenty Minutes
- Ruby @ Codecademy
- Things That Newcomers to Ruby Should Know

Ruby Reference

- Ruby API Docs
- Ruby Style Guide - Community Code Conventions for Rubyists
