In this lecture, we will discuss…

- ✧ **Scope** of variables
- ✧ **Constants**
- ✧ How the **scope** works with **blocks**

Scope

- Methods and classes begin **new scope** for variables
- Outer scope variables **do not** get carried over to the inner scope
- Use `local_variables` method to see which variables are in (and which are not in) the current scope

Scope

```ruby
v1 = "outside"

class MyClass
  def my_method
    # p v1 EXCEPTION THROWN - no such variable exists
    v1 = "inside"
    p v1
    p local_variables
  end
end

p v1 # => outside
obj = MyClass.new
obj.my_method # => inside
              # => [:v1]
p local_variables # => [:v1, :obj]
p self # => main
```

Scope: Constants

- **Constant** is any reference that begins with **uppercase**, including classes and modules
- Constants' scope rules are **different** than variable scope rules
- Inner scope **can see** constants defined in outer scope and **can also override** outer constants
  - Value remains **unchanged** outside!

Scope - Constant

```ruby
module Test
  PI = 3.14
  class Test2
    def what_is_pi
      puts PI
    end
  end
end
Test::Test2.new.what_is_pi # => 3.14
```

```ruby
module MyModule
  MyConstant = 'Outer Constant'
  class MyClass
    puts MyConstant # => Outer Constant
    MyConstant = 'Inner Constant'
    puts MyConstant # => Inner Constant
  end
  puts MyConstant # => Outer Constant
end
```

Remains unchanged outside

Scope: Block

- Blocks **inherit** outer scope
- Block is a **closure**
  - **Remembers** the context in which it was defined and **uses** that context whenever it is called

Scope - Block

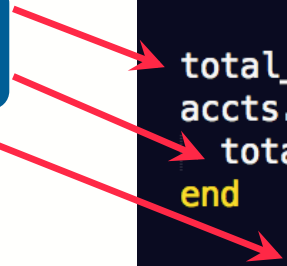
```ruby
class BankAccount
  attr_accessor :id, :amount
  def initialize(id, amount)
    @id = id
    @amount = amount
  end
end

acct1 = BankAccount.new(123, 200)
acct2 = BankAccount.new(321, 100)
acct3 = BankAccount.new(421, -100)
accts = [acct1, acct2, acct3]

total_sum = 0
accts.each do |eachAcct|
  total_sum += eachAcct.amount
end

puts total_sum # => 200
```

Same scope



Block – Local Scope

- Even though blocks **share** the outer scope – a variable created inside the block is **only available** to the block
- **Parameters** to the block are **always local** to the block – **even if** they have the **same name** as variables in the outer scope
- Can **explicitly declare** block-local variables after a **semicolon** in the block parameter list

Block: Local Scope

```ruby
arr = [5, 4, 1]
cur_number = 10
arr.each do |cur_number|
  some_var = 10 # NOT available outside the block
  print cur_number.to_s + " " # => 5 4 1
end
puts # print a blank line
puts cur_number # => 10
```

```ruby
adjustment = 5
arr.each do |cur_number;adjustment|
  adjustment = 10
  print "#{cur_number + adjustment} " # => 15 14 11
end
puts
puts adjustment # => 5 (Not affected by the block)
```

Summary

- ✧ Methods and classes **start** a new scope
- ✧ Constants **maintain** scope
- ✧ Blocks **inherit** outer scope
  - Could be **overridden**

What's next?

- ✧ Access Control