

# Intro to Ruby Programming

## GDI Rochester Chapter

[https://github.com/chorn/intro\\_to\\_ruby](https://github.com/chorn/intro_to_ruby)

# Welcome

Girl Develop It is here to provide affordable and accessible programs to learn software through mentorship and hands-on instruction.

# Thank You

- >> GDI Rochester
- >> GDI Cincinnati – Ryan Dlugosz
- >> GDI Seattle

# The Golden Rules

- » We are here for you!
- » Every question is important.
- » Help each other.
- » Have fun!

# @chorn

>> Chris Horn

>> chorn@chorn.com

# What to expect

- >> See Ruby
- >> Write Ruby
- >> Meet people who are learning with you
- >> Get an overview of Ruby

# Ruby

- » Ruby is a powerful language that reads like English
- » "I hope to see Ruby help every programmer in the world to be productive, and to enjoy programming, and to be happy. That is the primary purpose of Ruby language." - Yukihiro "Matz" Matsumoto

[https://en.wikipedia.org/wiki/Ruby\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Ruby_(programming_language))  
[https://en.wikipedia.org/wiki/Yukihiro\\_Matsumoto](https://en.wikipedia.org/wiki/Yukihiro_Matsumoto)

# You will get stuck

1. You are not the first to get stuck.
2. You are not the first to see that error message.
3. Google it.

# You!

- >> Who are you?
- >> What is your programming experience?
- >> What is your favorite hobby that has nothing to do with technology?

# Take notes

- >> Make a copy of these slides
- >> [https://github.com/chorn/intro\\_to\\_ruby](https://github.com/chorn/intro_to_ruby)
- >> Take notes on paper
- >> Take notes in an editor

# Repl.it

<https://repl.it/languages/ruby>

**ruby 2.3.1p112 (2016-04-26 revision 54768) [x86\_64-linux]**

>

*A REPL is a Read, Evaluate, and Print Loop.*

# What does this code do?

Type this into the right side and hit return:

**1 + 1**

# What does this mean?

> 1 + 1

=> 2

>

What does > mean?

What does => mean?

# You wrote a Program!

» Programs are the steps that computers follow to meet a goal.

# What does this code do?

**1 +\*** 1

# Errors – SyntaxError

```
1 ** 1
```

```
(repl):1: syntax error, unexpected *
```

```
1 ** 1
```

^

# All languages have Syntax

» Different for each language

So how do we fix:

**1    +\***    **1**

# Math

What do these do?

**1 + 1**

**2 \* 2**

**3 - 3**

**4 / 2**

**2 / 4**

**1 / 0**

**2 \*\* 3 # Exponents**

**3 % 2 # Modulo or Remainder**

# What does this code do?

**2   +++   2**

# All language have Semantics

» Similar across languages

# Ada Lovelace

Ada Lovelace is generally regarded as the first computer programmer. She was a mathematician and wrote an algorithm that a machine could use to generate a specific series of numbers. The background of this slide are her notes--the first computer program.

[https://en.wikipedia.org/wiki/Ada\\_Lovelace](https://en.wikipedia.org/wiki/Ada_Lovelace)

# What does this code do?

```
puts "Welcome to Ruby!"
```

» What is puts?

» What is ""

# Strings

```
"Ruby".upcase  
"Ruby".downcase  
"ruby".capitalize  
"Ruby".reverse  
"Ruby".length  
"Ruby".swapcase  
"Ruby".empty?  
"".empty?
```

## What are Strings?

# Ideas

*We translate ideas into code*

# Objects

*Objects are our ideas, as code, put into practice.*

# Messages

*Sending and receiving messages is the fundamental concept of Object Orient Programming.*

**1 + 1**

- >> 1 is an Object
- >> + is a message (or in Ruby, a method)
- >> 1 is an argument to + (and also an Object)

# What does this do?

**1 + 1**

**1 + (1)**

**1 .+ (1)**

**1.+ (1)**

# Kernel

>> puts isn't really just puts

>> Kernel.puts

>> Kernel.rand

# What does this do?

**1 + 2 + 3**

# What does this do?

`"Ruby".upcase.reverse`

# Explore

# Object Oriented Programming

- >> Match the semantics of our code to the idea
- >> I keep saying Object
- >> We code in a Class

# Our first class

- » Let's make a brick!
- » Type this into the *left* pane of and hit *Run*

```
class Brick  
end
```

**new**

» Update the *left* pane to look like this and hit *Run* this:

```
class Brick  
end
```

```
Brick.new
```

What does => #<Brick:0x007f5275a13708> mean?

# Brick

- » What questions do I want to ask a Brick?
- » What properties do all Bricks have?
- » What properties do some Bricks have?

**def**

*Methods are the names of our messages.*

**class Brick**

**def color**

**"gold"**

**end**

**end**

**Brick.new.color**

# Add 2 more methods to Brick

- » One with a String
- » Another with an Integer

# What did you make?

- >> What worked?
- >> What didn't work?

```
class Brick
  def color
    "gold"
  end
  def length
    8
  end
  def width
    4
  end
  def height
    2
  end
  def location
    "On the table."
  end
end
```

```
Brick.new.color
Brick.new.length
Brick.new.width
Brick.new.height
Brick.new.location
```

# Booleans

>> true

>> false

>> !

**true**

**false**

**!true**

**!false**

# Add 2 more methods to Brick

- » Method names that end in a ?
- » Return true or false

# What did you make?

- >> What worked?
- >> What didn't work?

```
class Brick
  def color
    "gold"
  end
  def length
    8
  end
  def width
    4
  end
  def height
    2
  end
  def location
    "On the table."
  end
  def heavy?
    true
  end
  def edible?
    false
  end
end
```

```
Brick.new.color
Brick.new.length
Brick.new.width
Brick.new.height
Brick.new.location
Brick.new.heavy?
Brick.new.edible?
```

# End of Day 1

- » What have we learned?
- » What's missing?

## *Homework*

- » What is an algorithm?

# Day 2

>> Welcome back!

# Review

- >> Syntax & Semantics
- >> Object Oriented Programming
  - >> Match the semantics of our code to the idea
- >> Integers
- >> Strings
- >> Booleans
- >> Classes

# Back to Bricks

» Update the *left* pane to look like this and hit *Run* this:

```
class Brick
  def color
    "gold"
  end
end

my_brick = Brick.new
puts my_brick.color
```

# Variables

```
my_favorite_number = 5
```

- » Can change
- » Has a name
- » Has a value
- » Has a class

# Literals

```
my_favorite_number = 5
```

```
the_worst_day = "Tuesday"
```

# Re-assignment

What does this do?

`my_favorite_number = 5`

`my_favorite_number = 7`

`my_favorite_number += 1`

`my_favorite_number *= 2`

# Variables and Literals

```
my_favorite_number = 5
```

```
your_favorite_number = my_favorite_number
```

```
your_favorite_number = my_favorite_number + 1
```

```
your_favorite_number = my_favorite_number * 1
```

# Expressions

- » Code is a mashup of natural language (English), math, and logic
- » Programming is powerful because it allows you to combine lots and lots and lots of expressions

# Exploring Strings

```
greeting = "Hello"
```

```
language = "Ruby"
```

What does this code do?

```
greeting + language
```

```
greeting * language
```

```
greeting * 2
```

```
language + 2
```

```
greeting.upcase.reverse
```

```
language.downcase * greeting.length
```

# What about Brick?

>> So far we've been using `Brick.new.some_method_name`

# A new kind of Brick

» Type this into the *left pane*

```
class Brick  
end
```

```
my_brick = Brick.new
```

# Initialize

```
class Brick
  def initialize
  end
end

my_brick = Brick.new
```

# @color

```
class Brick
  def initialize
    @color = "gold"
  end
end

my_brick = Brick.new

>> @ creates an instance variable
>> @color exists and is accessible throughout my class
```

# Instance Variables

What does this code do?

```
class Brick
  def initialize
    @color = "gold"
  end
end

my_brick = Brick.new
puts my_brick.@color
```

# Instance Variables

Instance variables are not messages

```
class Brick
  def initialize
    @color = "gold"
  end
  def color
    @color
  end
end
```

```
my_brick = Brick.new
puts my_brick.color
```

# Not Instance Variables

## What if I forget the @?

```
class Brick
  def initialize
    my_color = "gold"
  end

  def color
    my_color
  end
end

my_brick = Brick.new
puts my_brick.color
```

# Do names matter?

```
class Brick
  def initialize
    @qwerty = "gold"
  end
  def color
    @qwerty
  end
end
```

```
my_brick = Brick.new
puts my_brick.color
```

- » What happens?
- » Is this a good idea?

# Arguments

```
my_string = String.new("Hello Ruby")
```

```
my_empty_string = String.new()
```

```
Brick.new("gold")
```

```
class Brick
  def initialize(color)
    @color = color
  end
  def color
    @color
  end
end
```

```
my_brick = Brick.new("gold")
puts my_brick.color
```

# Messages

*Let's rewrite our use of Brick as plain English.*

- >> So far we have
  - >> I would like to make a gold colored brick.
  - >> What is the color of my brick?
- >> Let's add
  - >> Who owns my brick?
  - >> What is my brick made of?

*Change initialize, add owner*

```
class Brick
  def initialize(color, owner)
    @color = color
    @owner = owner
  end
  def color
    @color
  end
  def owner
    @owner
  end
end

my_brick = Brick.new("gold", "Me!")
puts my_brick.color
puts my_brick.owner
```

# `initialize(color, owner)`

>> Is it:

```
my_brick = Brick.new("gold", "Me!")
```

>> Or:

```
my_brick = Brick.new("Me!", "gold")
```

# Named Parameters

```
class Brick
  def initialize(color:, owner:)
    @color = color
    @owner = owner
  end
  def color
    @color
  end
  def owner
    @owner
  end
end

my_brick = Brick.new(color: "gold", owner: "Me!")
puts my_brick.color
puts my_brick.owner
```

# attr\_reader

```
class Brick

  attr_reader :color
  attr_reader :owner

  def initialize(color:, owner:)
    @color = color
    @owner = owner
  end
end

my_brick = Brick.new(color: "gold", owner: "Me!")
puts my_brick.color
puts my_brick.owner
```

# Changing attributes

*What does this do?*

```
my_brick = Brick.new(color: "gold", owner: "Me!")
```

```
puts my_brick.color
```

```
puts my_brick.owner
```

```
my_brick.owner = "You?"
```

```
puts my_brick.owner
```

# attr\_writer

```
class Brick

  attr_reader :color
  attr_reader :owner
  attr_writer :owner

  def initialize(color:, owner:)
    @color = color
    @owner = owner
  end
end

my_brick = Brick.new(color: "gold", owner: "Me!")
puts my_brick.color
puts my_brick.owner

my_brick.owner = "You?"
puts my_brick.owner
```

# Default Attributes 1

```
class Brick

  attr_reader :color
  attr_writer :color
  attr_reader :owner
  attr_writer :owner

  def initialize(color: "gold", owner: "Me!")
    @color = color
    @owner = owner
  end
end

my_brick = Brick.new(color: "red")
puts my_brick.color
puts my_brick.owner

my_brick.owner = "You?"
puts my_brick.owner
```

# Exploration

- » Make more bricks
- » Change the parameter order around
- » Write expressions to compare parts of your bricks

# Volume

- >> How do we determine the volume of a Brick?
- >> Length \* Width \* Height

# Volume

```
class Brick
  def initialize(l: 5, w: 2, h: 1)
    @l = l
    @w = w
    @h = h
  end

  def volume
    @l * @w * @h
  end
end

my_brick = Brick.new
puts my_brick.volume
```

# Naming

Pick the best variable name

**my\_favorite\_number = 5**

**fav\_num = 5**

**fav = 5**

**number = 5**

**n = 5**

**mfn = 5**

**temp = 5**

# Naming

*Names should be*

- » Meaningful
- » Easy to read
- » Concise
- » Specific
- » Speling

# Good Names

```
occupation = "Software Developer"
```

```
occupation_name = "Software Developer"
```

```
job_title = "Software Developer"
```

# Writing code

- >> Code is read more often than it is written
- >> The next programmer to read your code should
  - >> Understand what you are trying to do
  - >> Even if it doesn't work
  - >> Especially if you are the next programmer

# Expressing your intent

- >> Huge names are awesome
- >> If it takes a whole sentence to clearly explain what a variable is for, use it
- >> `adjustment_to_be_made_at_time_of_purchase` is perfect

# Leveling up

*What is the problem we're solving?*

- >> We work for the Brick Store
- >> They sell Bricks
- >> We are building a new website for them so that:
  - >> Customers can buy Bricks
  - >> We can manage the inventory of Bricks

# Default Attributes 2

```
class Brick

  attr_reader :color, :owner
  attr_writer :color, :owner

  def initialize(color: "gold", owner: "Me!")
    @color = color
    @owner = owner
  end

end

my_brick = Brick.new(color: "red")
puts my_brick.color
puts my_brick.owner

my_brick.owner = "You?"
puts my_brick.owner
```

# attr\_accessor

```
class Brick

  attr_accessor :color, :owner

  def initialize(color: "gold", owner: "Me!")

    @color = color
    @owner = owner
  end
end

my_brick = Brick.new(color: "red")
puts my_brick.color
puts my_brick.owner

my_brick.owner = "You?"
puts my_brick.owner
```

# `attr_accessor :color`

*What does it really do?*

```
def color
```

```
  @color
```

```
end
```

```
def color=(new_color)
```

```
  @color = new_color
```

```
end
```

```
def color
```

>> When our brick receives the message:

*What is your color?*

>> We answer with:

*This is my color.*

```
def color
```

*This should be the only place we return @color*

```
def color=
```

» When our brick receives the message:

*I want to change your color to be purple.*

» We answer with:

*I have changed my to purple.*

```
def color=
```

*This should be the only place we change @color*

```
class Brick
  def initialize(color: "gold")
    @color = color
  end
  def color
    @color
  end
  def color=(new_color)
    @color = new_color
  end
end
```

```
my_brick = Brick.new
puts my_brick.color
my_brick.color = "blue"
puts my_brick.color
```

*The use of color, color, color in initialize is tricky.*

# Clearly named parameters

```
class Brick
  def initialize(initializing_brick_color)
    @color = initializing_brick_color
  end
  def color
    @color
  end
  def color=(new_color)
    @color = new_color
  end
end

my_brick = Brick.new("gold")
puts my_brick.color
my_brick.color = "blue"
puts my_brick.color
```

# self with clearly named parameters

```
class Brick
  def initialize(initializing_brick_color)
    self.color = initializing_brick_color
  end
  def color
    @color
  end
  def color=(new_color)
    @color = new_color
  end
end

my_brick = Brick.new("gold")
puts my_brick.color
my_brick.color = "blue"
puts my_brick.color
```

# self with named parameters

```
class Brick
  def initialize(color: "gold")
    self.color = color
  end
  def color
    @color
  end
  def color=(new_color)
    @color = new_color
  end
end

my_brick = Brick.new(color: "gold")
puts my_brick.color
my_brick.color = "blue"
puts my_brick.color
```

# Debugging a Brick

```
class Brick
  def initialize(initializing_brick_color)
    puts "Initialize BEGIN with initializing_brick_color '#{initializing_brick_color}'"
    self.color = initializing_brick_color
    puts "Initialize END   with initializing_brick_color '#{initializing_brick_color}'"
  end
  def color
    puts "Reader BEGIN/END 'color' with @color #{@color}"
    @color
  end
  def color=(new_color)
    puts "Writer BEGIN 'color=' with @color #{@color} and new_color '#{new_color}'"
    @color = new_color
    puts "Writer END   'color=' with @color #{@color} and new_color '#{new_color}'"
  end
  puts "----Program Start----"

  puts "1. Make a gold brick."
  my_brick = Brick.new("gold")

  puts
  puts "2. Print the color of my brick."
  puts my_brick.color

  puts
  puts "3. Change the color of my brick to be blue"
  my_brick.color = "blue"

  puts
  puts "4. Print the color of my brick."
  puts my_brick.color

  puts
  puts "----Program Stop----"
```

# Our new Boss

- » As employees of the Brick Store, we have a new boss.
- » His name is Jack
- » Jack considers himself to be a master of bricks.
- » Jack prefers the blue bricks and will not share.
- » Jack wants our website enforce these rules.

# Constants

*The Brick Master*

```
class Brick
  attr_accessor :color, :owner

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.color = color
    self.owner = owner
  end
end

my_brick = Brick.new(owner: "Our Local Store")
puts my_brick.owner
my_brick.owner = "Customer"
puts my_brick.owner
```

*What does this do?*

```
class Brick

attr_accessor :color, :owner

BRICK_MASTER = "Jack"

def initialize(color: "gold", owner:)
  self.color = color
  self.owner = owner

  BRICK_MASTER = owner
end

my_brick = Brick.new(owner: "Someone")
puts my_brick.owner
my_brick.owner = "You?"
puts my_brick.owner
```

*What does this do?*

```
class Brick

attr_accessor :color, :owner

BRICK_MASTER = "Jack"

def initialize(color: "gold", owner:)
    self.color = color
    self.owner = owner
end

my_brick = Brick.new(owner: "Someone")
puts my_brick
```

# to\_s

```
class Brick
  attr_accessor :color, :owner

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.color = color
    self.owner = owner
  end

  def to_s
    "I am a Brick"
  end
end

my_brick = Brick.new(owner: "Someone")
puts my_brick
```

# to\_s

```
def to_s
```

```
end
```

```
>> @color
```

```
>> "#{@color}"
```

```
>> "Color: #{@color}"
```

```
>> "Owner: #{@owner} Color: #{@color}"
```

```
>> "Owner: #{@owner}\nColor: #{@color}"
```

```
>> "Owner: #{@owner}\tColor: #{@color}"
```

# to\_s

```
class Brick
  attr_accessor :color, :owner

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.color = color
    self.owner = owner
  end

  def to_s
    "Owner: #{self.owner}\tColor: #{self.color}"
  end
end

my_brick = Brick.new(owner: "Someone")
puts my_brick
```

# The Brick Master's Rules

```
def owned_by_the_brick_master?  
# ???  
end
```

# Logic

**true**

**false**

**!true**

**!false**

**true && true**

**true && false**

**true || false**

**false || true**

**false && false**

**false || false**

# Logic

*Expressions!*

**1 == 1**

**1 == 2**

**1 != "Hello"**

**"Ruby" == "Ruby?"**

**1 < 2**

**1 > 2**

**1 <= 1**

**1 <= 2**

**2 >= 1**

**2 >= 2**

# Logic

## *Order of Operations*

**(1) == (2)**

**(1 + 1) == 2**

**(1 + 1) == (2 + 0)**

**(1 + 1) == (2 + 100)**

**(1 + 1) <= (2 + 100)**

# nil

- » Defined vs. Undefined
- » Does this thing exist?
- » Does it have a value?

**puts nil**

**nil && true**

**nil || true**

**false || nil**

**nil && nil**

# Truthy

>> Truthy

*true is truthy*

*Any expression with a defined value is truthy*

>> Falsey

*false is falsey*

*nil is falsey*

# Truthy!

» These are all Truthy

"hello"

String

1

Time.new

# Truthy!!

0 is truthy

*Which ones are truthy?*

**true**

**1**

**3.14 == 3.14**

**3.14 \* 100 == 3.14 \* 10\*\*2**

**0**

**0 || false**

**0 && 0**

**false && 0**

**!nil**

**( !false || nil ) && 0**

# Part of a Brick Master Rule

```
def owned_by_the_brick_master?  
  @owner == BRICK_MASTER  
end
```

What do we put in color=?

```
class Brick
  attr_accessor :owner
  attr_reader :color

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.owner = owner
    self.color = color
  end

  def color=(new_color)
    # What goes here?
  end

  def owned_by_the_brick_master?
    self.owner == BRICK_MASTER
  end

  def to_s
    "Owner: #{self.owner}\tColor: #{self.color}"
  end
end

my_brick = Brick.new(owner: "Someone")
puts my_brick
puts my_brick.owned_by_the_brick_master?
```

# End of Day 2

## *Homework*

>> Ruby Core Library

>> <http://ruby-doc.org/core-2.4.0/>

>> What can Strings do?

>> <http://ruby-doc.org/core-2.4.0/String.html>

# Day 3

>> Welcome back!

# Review

- >> Variables – Literals, Constants, Instance Variables
- >> Classes
  - >> attr\_reader, attr\_writer, attr\_accessor, to\_s
  - >> Initializers, Default Values
- >> Names
- >> Named Parameters
- >> Expressions – Logic, nil, Truthy, Falsey

# Just Bricks

```
class Brick
  attr_accessor :owner
  attr_reader :color

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.owner = owner
    self.color = color
  end
  def color=(new_color)
    @color = new_color
  end
  def owned_by_the_brick_master?
    self.owner == BRICK_MASTER
  end
  def to_s
    "Owner: #{self.owner}\nColor: #{self.color}"
  end
end

someones_brick = Brick.new(owner: "Someone", color: "red")
privileged_brick = Brick.new(owner: "Jack", color: "blue")
forbidden_brick = Brick.new(owner: "Someone", color: "blue")
```

# Hoardings Blue Bricks

*What do we put in color=?*

**## The rest of Brick is the same**

```
def color=(new_color)
```

```
# ???
```

```
end
```

**## The rest of Brick is the same**

# Our Goal

*Let's state it in english:*

- » Jack is the Brick Master and he wants to keep all the blue bricks for himself.
- » Anyone can buy bricks as long as they are not blue.
- » Jack can buy blue bricks.
- » If someone other than Jack tries to buy a blue brick, they must be stopped.

# Hoardings Blue Bricks

```
def color=(new_color)
```

- » If this new color is not blue then I should change the color.
- » If the owner is Jack then I should change the color.
- » If this new color is blue and the owner is not Jack then we must not change the color.

```
end
```

# Happy and Unhappy

```
def color=(new_color)
```

- » These are the ways we can stay happy
  - » If this new color is not blue then I should change the color.
  - » If the owner is Jack then I should change the color.
- » This is the way we are not happy
  - » If this new color is blue and the owner is not Jack then we must not change the color.

```
end
```

# The Happy Path

```
def color=(new_color)
```

1. If this **new color** *is not* **blue** then I should **change the color**.
2. If the **owner** *is* **Jack** then I should **change the color**.

```
end
```

# Happy Path #1

*Let's break this into smaller pieces*

- >> If this
  - >> **new color**
  - >> *is not*
  - >> **blue**
  - >> then I should
  - >> **change the color**

# Happy Path #1

*Now we'll shed some English*

>> if

>> **new color**

>> *is not*

>> **blue**

>> then

>> **change the color**

# Happy Path #1

*Now with some more Ruby*

```
def color=(new_color)
```

```
>> if
```

```
>> new_color
```

```
>> is not
```

```
>> blue
```

```
>> then
```

```
>> @color = new_color
```

```
end
```

*How do we write not blue?*

# Happy Path #1

*Almost all Ruby?*

```
def color=(new_color)
>> if
>> new_color
>> != "blue"
>> then
>> @color = new_color
end
```

# Happy Path #1

*Almost all Ruby?*

```
def color=(new_color)
  if new_color != "blue"
    then @color = new_color
  end
```

# Happy Path #1

*Almost all Ruby?*

```
def color=(new_color)
  if new_color != "blue"
    then
      @color = new_color
  end
```

# Happy Path #1

```
def color=(new_color)
  if new_color != "blue"
    @color = new_color
  end
end
```

# The Happy Path

```
def color=(color)
```

1. If this **new color** *is not* **blue** then I should **change the color**.
2. If the **owner** *is* **Jack** then I should **change the color**.

```
end
```

# Happy Path #2

*Let's break this into smaller pieces*

- >> If the
  - >> **owner**
  - >> *is*
  - >> **Jack**
  - >> then I should
  - >> **change the color**

# Happy Path #2

*Shed some English*

```
>> if  
    >> owner  
    >> is  
    >> Jack  
>> then  
    >> change the color  
>> end
```

# Happy Path #2

*Add some Ruby*

```
>> if  
    >> owner  
    >> is  
    >> Brick Master  
>> then  
    >> @color = new_color  
>> end
```

# Happy Path #2

*Add more Ruby*

```
>> if  
>>   owner  
>>     is  
>>       BRICK_MASTER  
>>     then  
>>       @color = new_color  
>>   end
```

# Happy Path #2

*Add more Ruby*

```
>> if  
  
    >> self.owner == BRICK_MASTER  
  
>> then  
  
    >> @color = new_color  
  
>> end
```

# Ruby Fun Facts

true and true

true or false

# Happy Path #2

*Add more Ruby*

```
if self.owner == BRICK_MASTER  
  @color = new_color  
end
```

# Both Happy Paths

```
def color=(new_color)

  if new_color != "blue"
    @color = new_color
  end

  if self.owner == BRICK_MASTER
    @color = new_color
  end
end
```

# The Unhappy Path

>> If this new color is blue and the owner is not Jack then we must not change the color.

# The Unhappy Path

```
if new_color == "blue" && self.owner != BRICK_MASTER
```

» We must not change the color.

```
end
```

# The Unhappy Path

```
if new_color == "blue" && self.owner != BRICK_MASTER  
    fail "We must not change the color."  
end
```

# It Works!

We're not quite done

```
def color=(new_color)

  if new_color != "blue"
    @color = new_color
  end

  if self.owner == BRICK_MASTER
    @color = new_color
  end

  if new_color == "blue" && self.owner != BRICK_MASTER
    fail "We must not change the color."
  end
end
```

# What about owned\_by\_the\_brick\_master?`

```
def color=(new_color)

  if new_color != "blue"
    @color = new_color
  end

  if owned_by_the_brick_master?
    @color = new_color
  end

  if new_color == "blue" && !owned_by_the_brick_master?
    fail "We must not change the color."
  end

end
```

# Simpler?

```
def color=(new_color)

  if (new_color != "blue") || owned_by_the_brick_master?
    @color = new_color
  end

  if new_color == "blue" && !owned_by_the_brick_master?
    fail "We must not change the color."
  end

end
```

# Pertinent Negatives

```
def color=(new_color)

if (new_color != "blue") || owned_by_the_brick_master?
  @color = new_color
end

if new_color == "blue" && !owned_by_the_brick_master?
  fail "We must not change the color."
end

end
```

# Clearer message

```
def color=(new_color)

  if new_color == "blue" && !owned_by_the_brick_master?
    fail "Blue bricks are reserved for #{BRICK_MASTER}."

  end

  if (new_color != "blue") || owned_by_the_brick_master?
    @color = new_color
  end

end
```

```
else
```

```
def color=(new_color)
```

```
if new_color == "blue" && !owned_by_the_brick_master?
```

```
  fail "Blue bricks are reserved for #{BRICK_MASTER}."
```

```
else
```

```
  @color = new_color
```

```
end
```

```
end
```

# Hoarded Bricks

```
class Brick
  attr_accessor :owner
  attr_reader :color

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.owner = owner
    self.color = color
  end

  def color=(new_color)
    if new_color == "blue" && !owned_by_the_brick_master?
      fail "Blue bricks are reserved for #{BRICK_MASTER}."
    else
      @color = new_color
    end
  end

  def owned_by_the_brick_master?
    self.owner == BRICK_MASTER
  end

  def to_s
    "Owner: #{self.owner}\nColor: #{self.color}"
  end
end

someones_brick = Brick.new(owner: "Someone", color: "red")
privileged_brick = Brick.new(owner: "Jack", color: "blue")
forbidden_brick = Brick.new(owner: "Someone", color: "blue")
```

# End of Day 3

# Day 4

>> Welcome back!

# Review

- >> Conditionals
  - >> Happy/Unhappy Paths, Pertinent Negatives
  - >> if (then)
  - >> else
- >> Rejecting bad things
  - >> fail
- >> Translating English into Ruby
- >> Debugging

# Our Goal

*What is the problem we're solving?*

- >> We work for the Brick Store
- >> They sell Bricks
- >> We are building a new website for them where
  - >> We manage the inventory of Bricks
  - >> Customers can buy Bricks
- >> Jack is our boss and he's keeping the blue bricks for himself

# repl.it

```
class Brick
  attr_accessor :owner
  attr_reader :color

  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.owner = owner
    self.color = color
  end

  def color=(new_color)
    if new_color == "blue" && !owned_by_the_brick_master?
      fail "Blue bricks are reserved for #{BRICK_MASTER}."
    else
      @color = new_color
    end
  end

  def owned_by_the_brick_master?
    self.owner == BRICK_MASTER
  end

  def to_s
    "Owner: #{self.owner}\nColor: #{self.color}"
  end
end
```

# **BrickStore?**

*Leave our class Brick .... end code unchanged from now on.*

```
class BrickStore  
end
```

```
brick_store = BrickStore.new(name: "Ruby's Bricks")
```

# BrickStore

```
class BrickStore

  attr_accessor :name

  def initialize(name:)
    self.name = name
  end

end

brick_store = BrickStore.new(name: "Ruby's Bricks")
```

# What does a Brick Store need?

- >> Bricks?
- >> Strict controls on what brick colors are acceptable to society as a whole?
- >> Authoritarian enforcement of the brick supply chain as it manifests in each and every franchised store?

# approved\_brick\_colors

*In the REPL on the right side:*

```
colors = [ "red", "green", "blue", "yellow", "gray" ]  
colors.class
```

# Arrays

`colors.any?`

`colors.empty?`

`colors.size`

`colors.first`

`colors.last`

`colors.sort`

`colors.reverse`

`colors.shuffle`

```
colors.sample  
colors.include?("blue")  
colors.index("yellow")  
colors.at(0)  
colors.at(2)  
colors[3]  
colors[99]  
colors[1..4]  
colors.sort.reverse[-2]
```

```
colors.push("brown")
colors << "magenta"
colors << 12345
colors << Time.now
colors << nil
colors.compact
colors.compact!
colors.delete("brown")
colors.delete_at(2)
colors.delete_at(-1)
colors.join("\n")
colors.join(" --- ")
```

# BrickStore

```
class BrickStore
  attr_accessor :name
  def initialize(name:)
    self.name = name
  end

  def approved_brick_colors
    [ "red", "green", "blue", "yellow", "gray" ]
  end
end

brick_store = BrickStore.new(name: "Ruby's Bricks")
brick_store.approved_brick_colors
```

# Symbols

```
class BrickStore
  attr_accessor :name
  def initialize(name:)
    self.name = name
  end

  def approved_brick_colors
    [ :red, :green, :blue, :yellow, :gray ]
  end
end

brick_store = BrickStore.new(name: "Ruby's Bricks")
brick_store.approved_brick_colors
```

# Inventory

```
class BrickStore
  attr_accessor :name
  attr_reader :inventory
  def initialize(name:)
    self.name = name
    @inventory = []
  end

  def approved_brick_colors
    [ "red", "green", "blue", "yellow", "gray" ]
  end
end

brick_store = BrickStore.new(name: "Ruby's Bricks")
brick_store.approved_brick_colors
brick_store.inventory
```

# Our Goal

*What is the next part of the problem we can solve?*

- >> We work for the Brick Store
- >> They sell Bricks
- >> We are building a new website for them where
  - >> Customers can buy Bricks
  - >> We can manage the inventory of Bricks
- >> Jack is our boss and he's keeping the blue bricks for himself

# What does Inventory mean?

*How do normal stores handle inventory?*

# A Shipment of Bricks

- >> Jack, as our boss, is in charge of receiving all shipments of bricks. When the shipment arrives, he effectively takes ownership of all the bricks until they are sold.
- >> Each shipment we receive has a total of 10 Bricks that are from a random assortment of acceptable colors.

# How do we make a brick as part of a shipment?

*Jack, as our boss, is in charge of receiving all shipments of bricks. When the shipment arrives, he effectively takes ownership of all the bricks until they are sold.*

```
def one_new_brick  
# ???  
end
```

# How do we add to our inventory?

*Each shipment we receive has a total of 10 Bricks that are from a random assortment of acceptable colors.*

```
def receive_new_shipment!
```

```
# ???
```

```
end
```

# Another Goal

- >> Let's make a report that
  - >> scans our inventory
  - >> reports on the # of bricks for each color

*Can we write this using only arrays?*

# Is there an easier way?

*What if we want to "store" the report as we go?*

# Group Programming

*Let's add some more features to our BrickStore*

# A Final BrickStore

```
class Brick
  attr_accessor :owner
  attr_reader :color
  BRICK_MASTER = "Jack"

  def initialize(color: "gold", owner:)
    self.owner = owner
    self.color = color
  end

  def color=(new_color)
    if new_color == "blue" && owned_by_the_brick_master?
      fail "Blue bricks are reserved for #{BRICK_MASTER}."
    else
      @color = new_color
    end
  end

  def owned_by_the_brick_master?
    self.owner == BRICK_MASTER
  end

  def to_s
    "Owner: #{self.owner}\nColor: #{self.color}"
  end
end

class BrickStore
  attr_accessor :name, :default_delivery_size
  attr_reader :inventory
  def initialize(name, default_delivery_size)
    self.name = name
    self.default_delivery_size = default_delivery_size
    @inventory = []
  end

  def approved_brick_colors
    [ :red, :green, :blue, :yellow, :gray ]
  end

  def one_new_brick
    Brick.new(
      owner: Brick::BRICK_MASTER,
      color: approved_brick_colors.sample
    )
  end

  def accept_delivery(delivery_size: default_delivery_size)

    delivery_size.times do
      @inventory << one_new_brick
    end
  end

  def to_s
    report = {}
    approved_brick_colors.sort.each do |color|
      report[color] = 0
    end
    inventory.each do |brick|
      if brick.color == color
        report[brick.color] += 1
      end
    end
    report.inspect
  end
end

brick_store = BrickStore.new(name: "Ruby's Bricks", default_delivery_size: 5)
brick_store.accept_delivery
brick_store.accept_delivery(delivery_size: 25)
puts brick_store.inventory.size
puts brick_store
```

# The End!

- >> There is a lot more to Ruby, programming, and OO.
- >> You are ready to tackle RubyMonk:
  - >> <https://rubymonk.com/learning/books/1-ruby-primer>

*Stuck? Try @chorn.*