

# Programming for **Everybody**

## 9. Object Oriented Programming



# Classes and Instances

ruby has some built in classes you already know: string, integer, array, hash, etc.

a Ruby *class* is like a “muffin pan” from which several *instances* can be originated, all of them sharing similar methods and their respective attributes

each instance of a class is a Ruby *object*



```
"John".length  
=> 4
```

the “John” object is a string with a .length method and a length attribute of 4;  
what makes "John" a string is the fact that it's an *instance* of the string *class*



# Building your own classes

we can also create new classes from scratch

class syntax: **class** keyword + **class name** + **end** keyword

within this, we include the **.initialize** method, which “boots up” each object created by the class and which includes the **instance variables** (these set the new objects’ specifics)

```
class NewCar
  def initialize(make, model)
    @make = make
    @model = model
  end
end
```

```
NewCar.new("Honda", "Civic");
```

← this class will allow us to create as many car instances as we want

each car object will have its own make and model

we can create an *instance* of a class just by calling **.new** on the class name and **defining values for the instance variables**

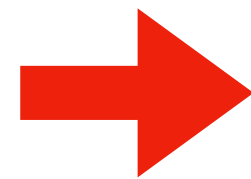
# Instance methods

we often define other methods for our classes so that their instances can do interesting stuff

while instance variables define an object's attributes, methods define its *behaviour*

```
class Person
  def initialize(name)
    @name = name
  end

  def greeting
    puts "Hi!"
  end
end
```



```
mariana = Person.new("Mariana")
mariana.greeting
prints out "Hi!"
```



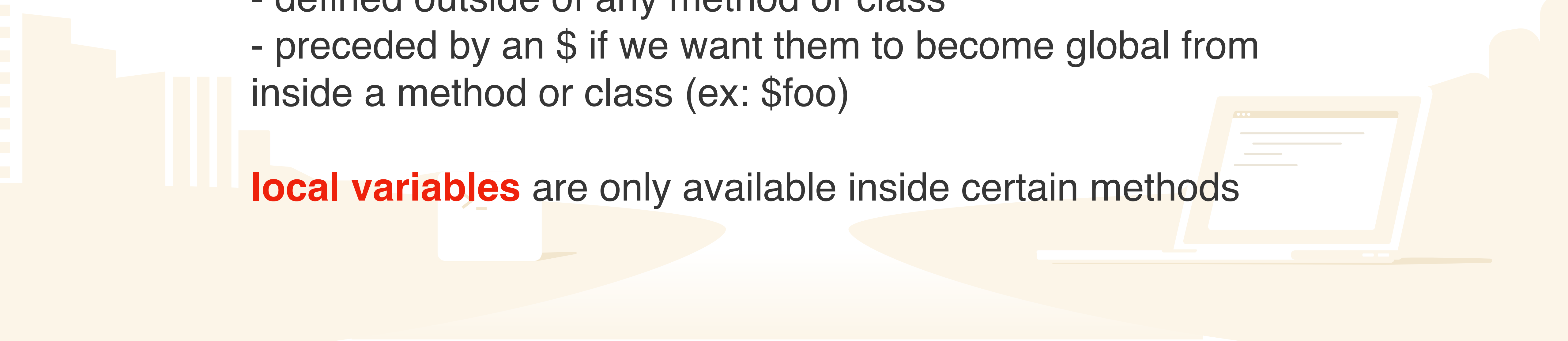
# Scope

an important aspect of Ruby classes is their *scope* -> the context in which they're available

**global variables** are available everywhere and can be declared in two ways:

- defined outside of any method or class
- preceded by an \$ if we want them to become global from inside a method or class (ex: \$foo)

**local variables** are only available inside certain methods



# Scope

**class variables** belong to a certain class, are preceded by two @s (ex: @@files) and there's only one copy of a class variable which is then shared by all instances of that class

**instance variables** are only available to particular instances of a class and are preceded by an @

global variables can be changed from anywhere in the program and it's better to create variables with limited scope that can only be changed from a few places (ex: instance variables that belong to a particular object)

# Scope

The same goes for **methods**

**global methods** are available everywhere

**class methods** are only available to members of a certain class

**instance methods** are only available to particular instances



# Inheritance syntax

**inheritance** is the process by which one class takes on the attributes and methods of another

the derived class (or *subclass*)  
is the new class we're creating

the base class (or *parent* or *superclass*) is the  
class from which the derived class inherits

inheritance syntax: `class DerivedClass < BaseClass`  
`# some stuff`  
`end`

we read "<" as "inherits from"





# Inheritance + super

we can directly access the attributes or methods of a parent class with Ruby's built-in **super** keyword

```
class DerivedClass < ParentClass
  def some_method
    super(optional args)
    # Some stuff
  end
end
end
```

when we call `super` from inside a method we're telling Ruby to look in the parent class of the current class and find a method with the same name as the one from which `super` is called

if it finds it, Ruby will use the parent class' version of the method



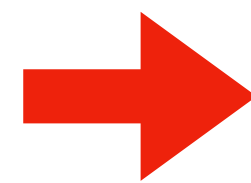
# Overriding inheritance

**inheritance** is the process by which one class takes on the attributes and methods of another

```
class Creature
```

```
  def initialize(name)  
    @name = name  
  end
```

```
  def skin_color  
    puts "Green"  
  end  
end
```

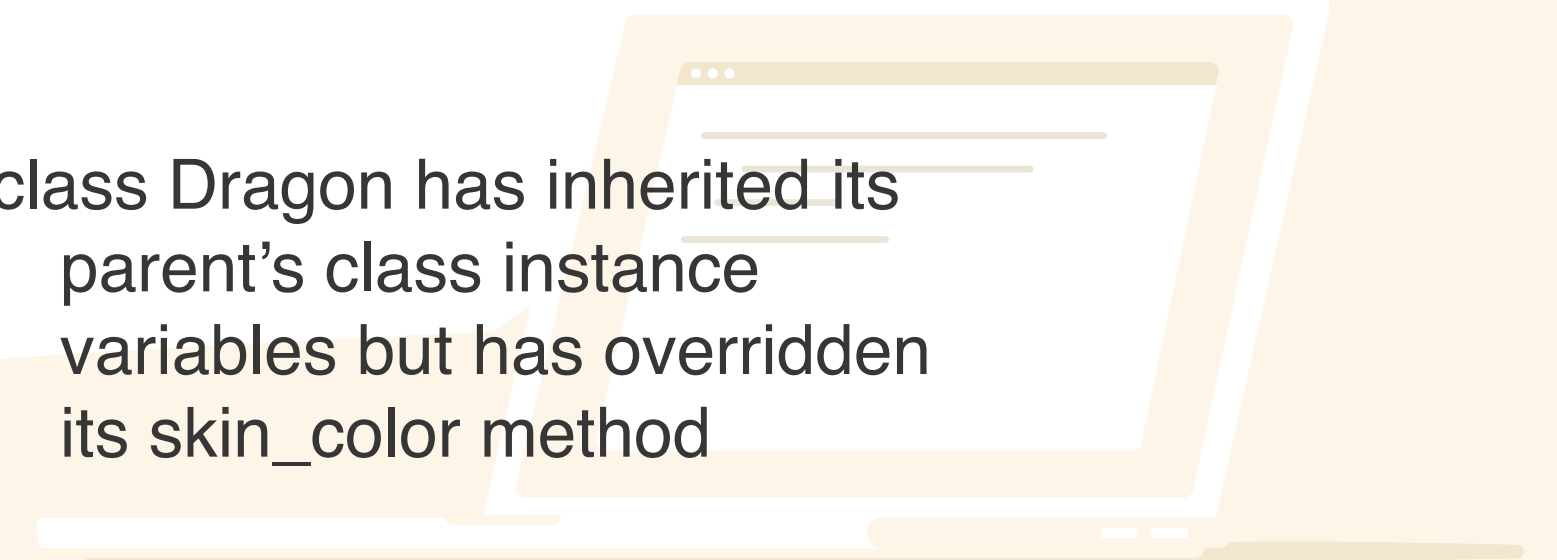


```
class Dragon < Creature  
  def skin_color  
    puts "Purple"  
  end  
end
```

```
bob = Dragon.new("bob")  
bob.skin_color
```

prints out "Purple"

→ class Dragon has inherited its parent's class instance variables but has overridden its skin\_color method



**Thank you!**

