Lambdas

Lambdas

- A way to define a Proc and its parameters with a special syntax

```
1.  say_something = -> { puts "This is a
    lambda" }
```

- Defining it with variables

```
irb(main):051:0> place_something = -> (x, y, z) { puts x, y, z }
=> #<Proc:0x00007fdc408e8928@(irb):51 (lambda)>
irb(main):052:0>
```

Procedure and lambda
Lamba is a procedure, a procedure with special syntax and context.

With lambdas you use the context of the own lambda.

Procs

- Different than lambdas when there are arguments.

```
irb(main):088:0> t = Proc.new { |x, y| puts x, y }
=> #<Proc:0x00007fdc43173398@(irb):88>
irb(main):089:0> t.call(1, 2)
1
2
=> nil
irb(main):090:0>
```

- If you use procs, you use the context of the method where you are calling it.
- If you use procs, you create your own context

Procs

- In the following example, we return in the call_proc method, not the proc itself.

```ruby
def call_proc
  puts "Before proc"
  my_proc = Proc.new { return 2 }
  my_proc.call
  puts "After proc"
end

p call_proc
# Prints "Before proc" but not "After proc"
```

If you are using a procedure inside a function and you call a return, this return is going to end the function.
You can still return something if you simply put the variable or value without return as the last line of the procedure.

procedures -> outside context
lambda -> inner context.

What can a program see within itself?

- What objects contain
- Current class hierarchy
- Contents and behaviors of objects
- Information on methods

Introspection

- With the information the program knows about themselves, they can call methods at runtime and modify themselves while running

The Object Space

- Where all objects inhabit in Ruby
- Objects alive and not yet collected by the Garbage Collector
  - This means an object is alive as long as it has any references pointing to it

In the object space is the list of all objects that are alive.

The Object Space

- What does it contain by default?
- Useful Tip: You can also use the method .each_object without an argument (class)

```
irb(main):001:0> ObjectSpace
=> ObjectSpace
irb(main):002:0> ObjectSpace.each_object(Numeric) { |x| p x }
(0+1i)
9223372036854775807
166676550149418238933563278187521080161
NaN
Infinity
1.7976931348623157e+308
2.2250738585072014e-308
=> 7
```

The object space is a module/class.
To access the object, we use the each_object method.

The Object Space

- Please note that the ObjectSpace doesn't know about Fixnum, true, false, or nil.

```
irb(main):099:0> ObjectSpace.count_objects[:TOTAL]
=> 35054
irb(main):100:0> b = nil
=> nil
irb(main):101:0> ObjectSpace.count_objects[:TOTAL]
=> 35054
irb(main):102:0>
```

Looking Inside Objects

- You can get the methods available for any variable with .methods

```
1    r = 1..10 #=> 1..10
2    list = r.methods #=> begin, end, bsearch.........
3    list.length #=> 125
4    list[0..3] #=> [begin end bsearch to_a]
5
```

To know if a class has a method:

```
?, :respond_to?, :object_id, :send, :__send__, :!, :!=
2.7.5 :008 > a.respond_to? :delete_prefix
 => true
2.7.5 :009 >
```

Calling methods dynamically

- Determining the objects direct class and class hierarchy

```
class Example

  MYCONSTANT = 'hello this is a test'

  def self.this_is_a_class_method
    puts 'ok'
  end

  def this_is_a_public_method
    puts 'ok'
  end

  protected

  def this_is_a_protected_method
    puts 'ok'
  end

  private

  def this_is_a_private_method
    puts 'ok'
  end
end
```

```
Example.private_instance_methods
#=> [:this_is_a_private_method, :irb_binding, ....]
Example.protected_instance_methods
#=> [:this_is_a_protected_method]
Example.public_instance_methods
#=> [ :this_is_a_public_method, :instance_variable_defined?, ...]
Example.singleton_methods #=> [:this_is_a_class_method]
Example.constants #=> [:MYCONSTANT]
```

(singleton methods are the class methods).

Calling the methods dynamically

- Send
- Eval
- Method.call

Send

- Sending a symbol (as best practice, althought it also accepts strings)
- Remember: Symbols represent methods (identifiers)

```ruby
"Ruby on Rails!".send(:length) #=> 14
"Ruby on Rails!".send("gsub", /Rails/, "wheels") #=> "Ruby on wheels!"
"Ruby on Rails!".send(:gsub, /Rails/, "wheels") #=> "Ruby on wheels!"
```

Send is calling the method itself, but it receives a symbol

```
 => true
2.7.5 :009 > a.send(:reverse)
 => "raksO"
2.7.5 :010 >
```

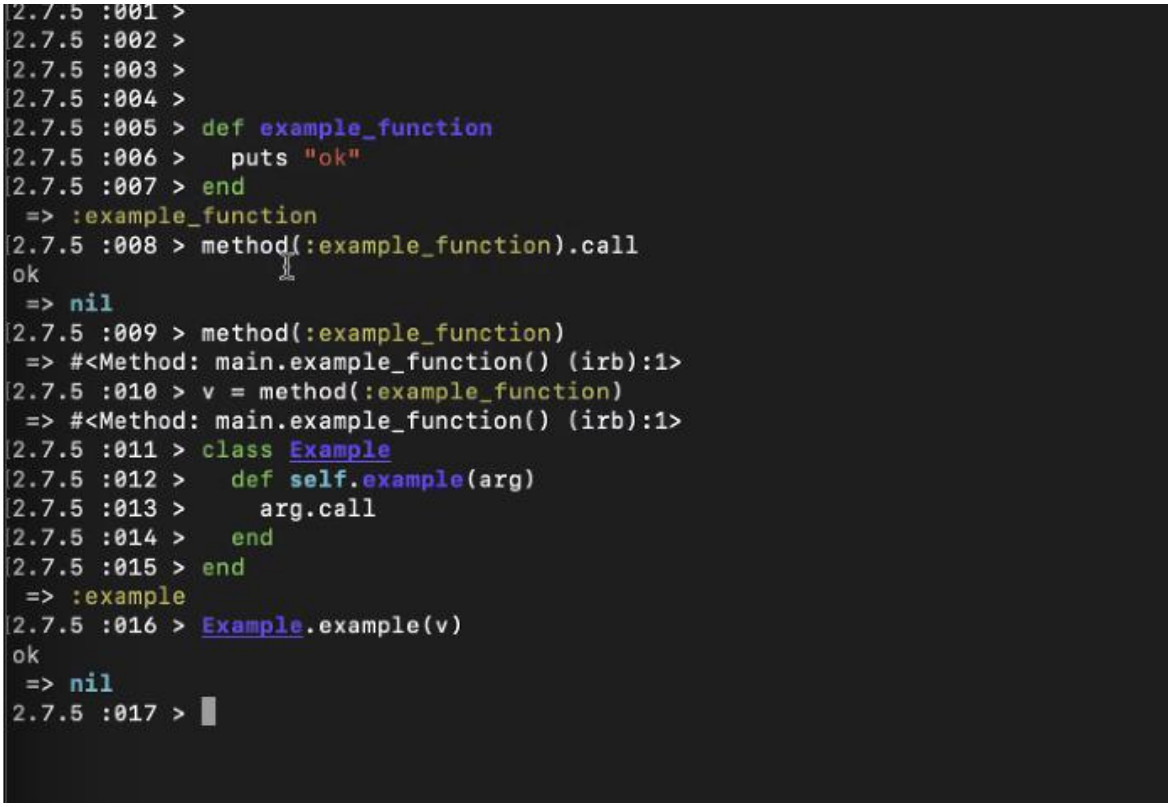
The Method Object

- Method class that creates a method that is not bound to a class
- This means it can be used anywhere

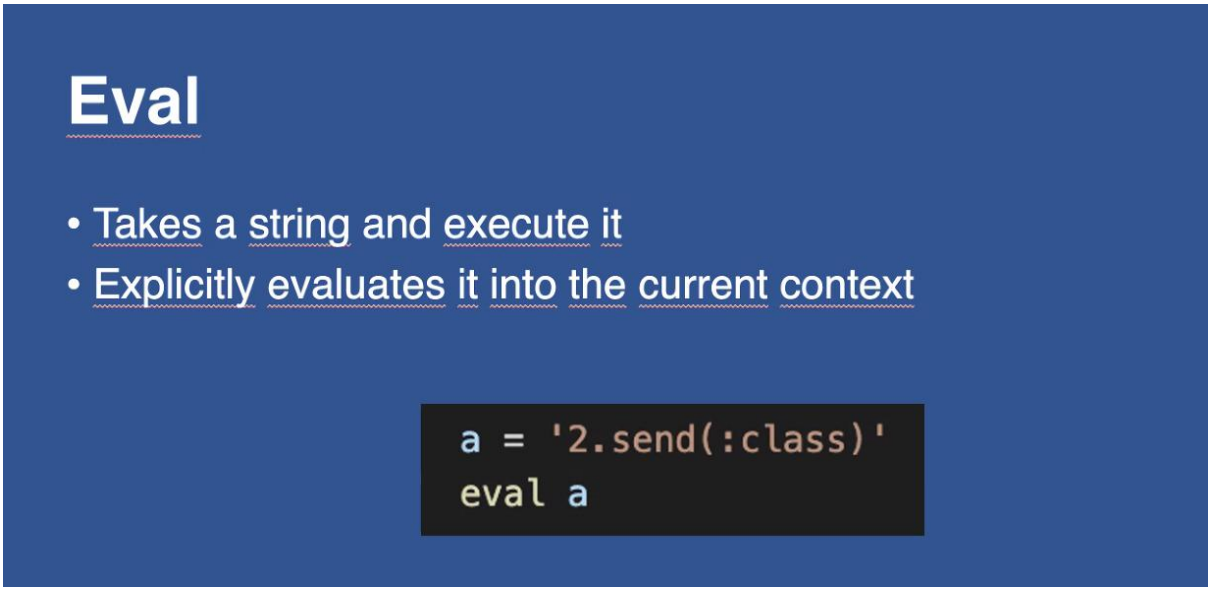
```ruby
def example_function(a)
  2 * a
end

method(:example_function).call(2) #=> 4
```

Llamar métodos que no están dentro de una clase.

Passing a method with the method name as a symbol, allows you to pass it as a variable.

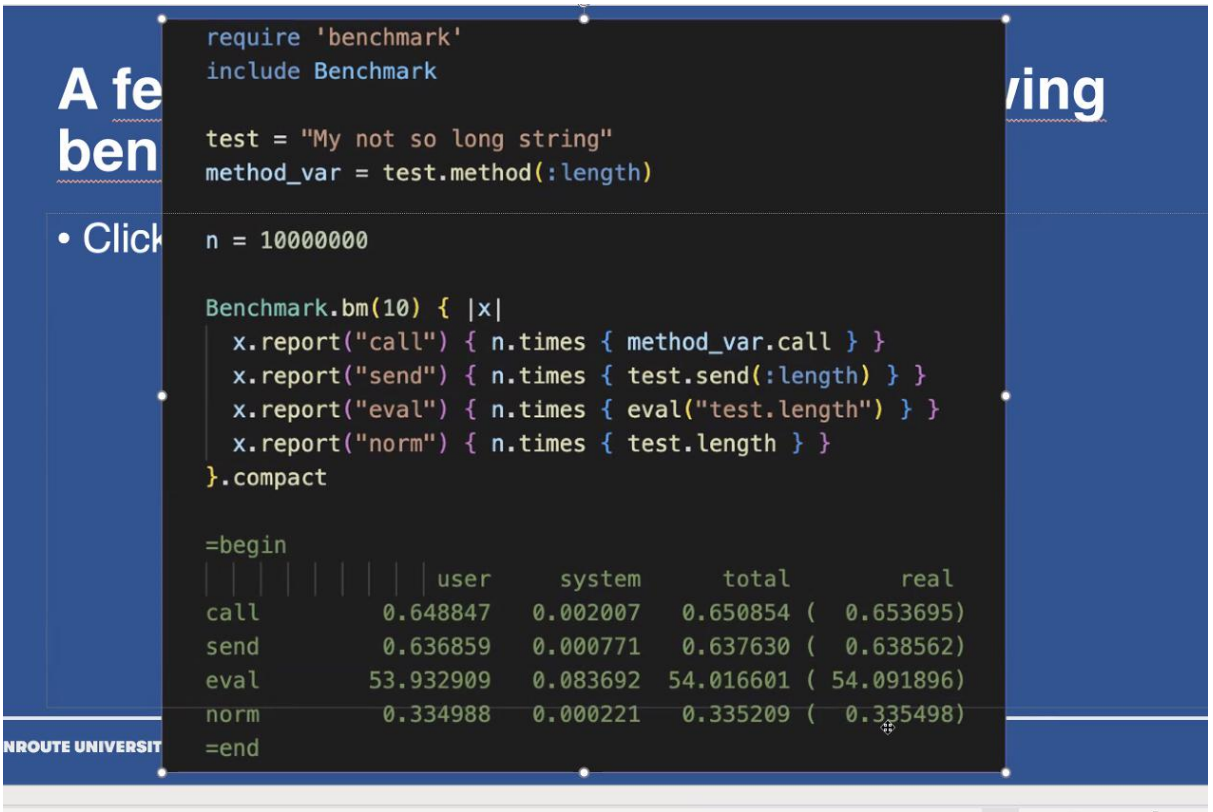


It is similar to passing it as a procedure.



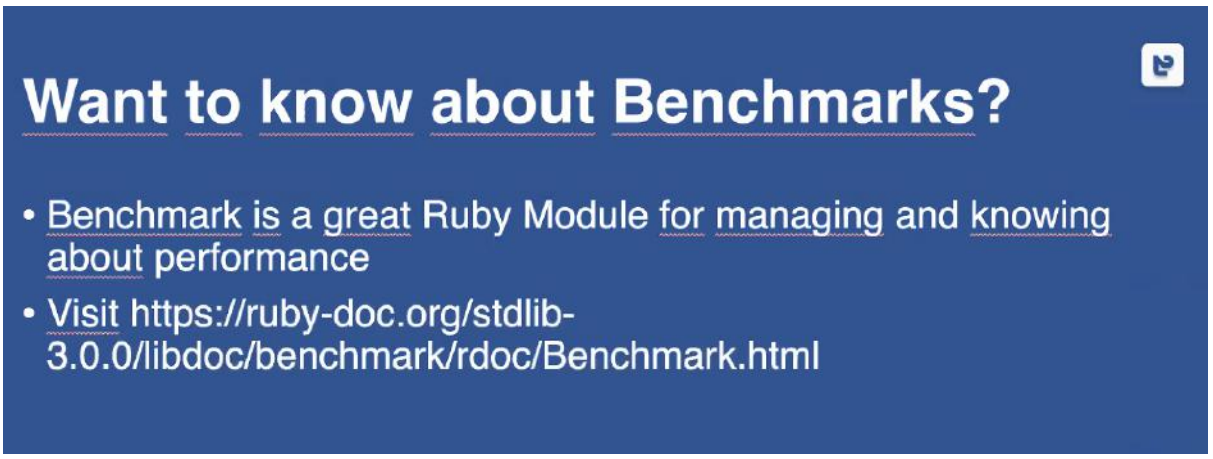
Receives a string and evaluates it with the current context.

THIS IS CALLED METAPROGRAMMING



The performance of the other methods is slower.

The benchmark is very used



System Hooks

- Technique that lets you trap events
- Most common event is time of creation
- Technique allowed by the *alias* method
  - Creates new aliases of the methods

System Hooks: An example

- Click to

```
class Object

  def timestamp
    return @timestamp
  end
  def timestamp=(aTime)
    @timestamp = aTime
  end
end

class Class
  alias_method :old_new,  :new
  def new(*args)
    result = old_new(*args)
    result.timestamp = Time.now
    return result
  end
end

class Test
end

Test.new.timestamp #=> Dependable of execution time but mine was 2022-08-12 14:25:13 -0500
```

ENROUTE UNIVERSITY

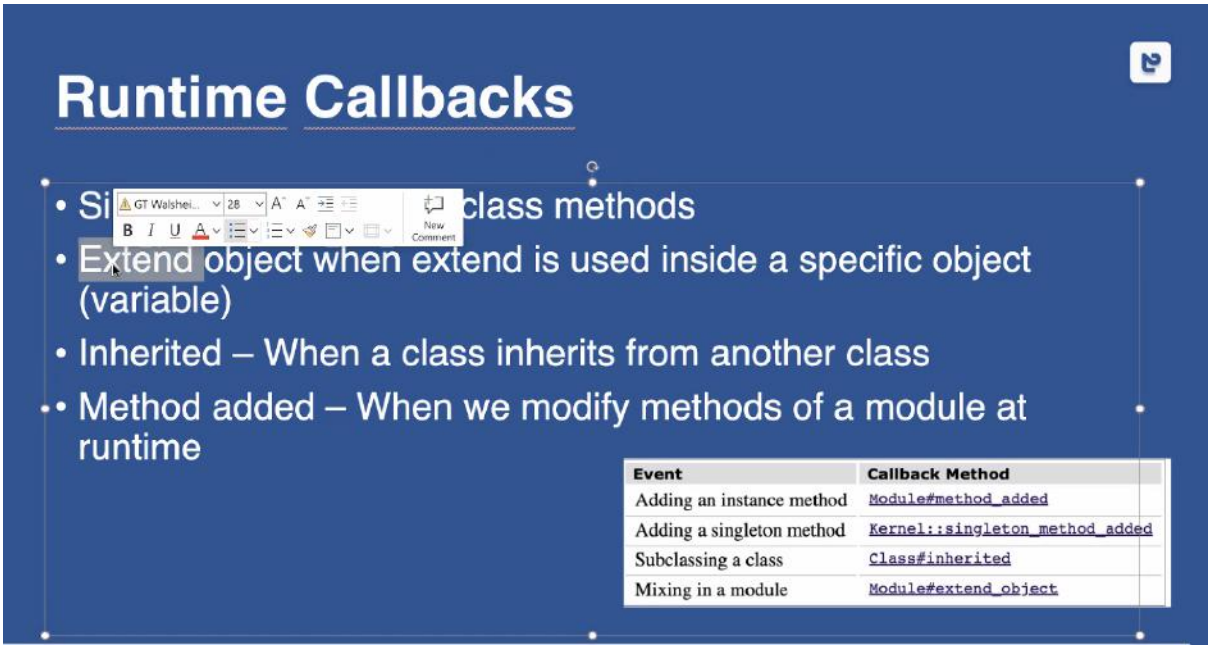
How to edit a class?
With alias_method it creates almost a sa shadow copy

System hook, a way to add additional properties to the variables we create.
How do we add a timestamp to a class?
…….

Runtime Callbacks

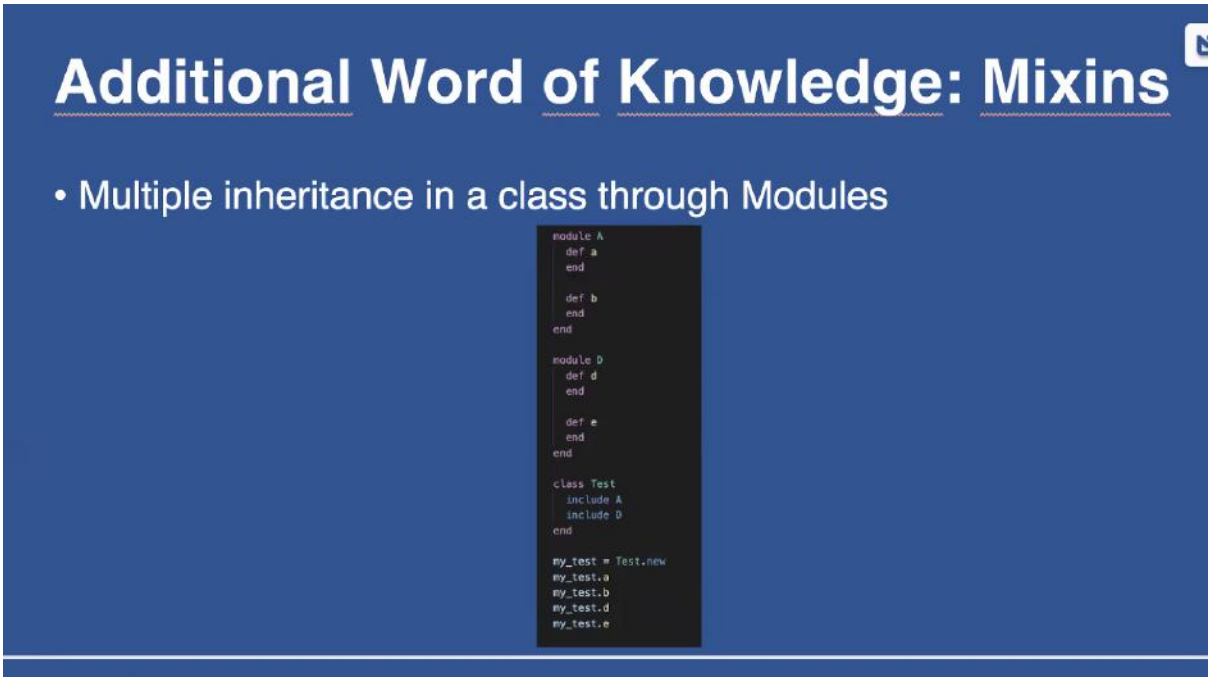
- Being notified when one of the following happens

Event	Callback Method
Adding an instance method	Module#method_added
Adding a singleton method	Kernel::singleton_method_added
Subclassing a class	Class#inherited
Mixing in a module	Module#extend_object



A way to adding properties to an object. USED THE MOST FOR THAT

```ruby
module Test
  def self.extend_object(o)
    puts "Added Test to #{ o.class } type"
  end
end

(3).extend(Test) #=> Added Test to Integer type
```

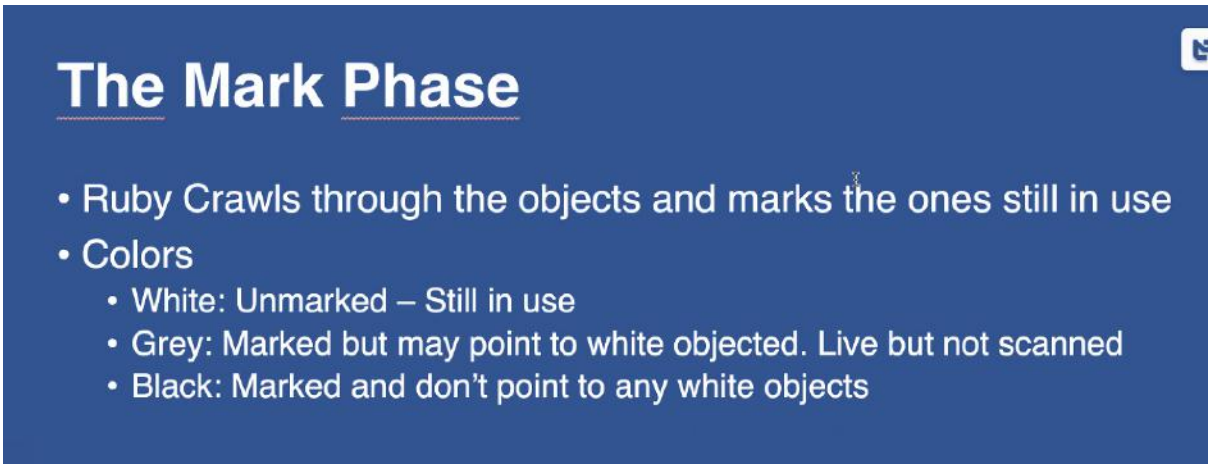


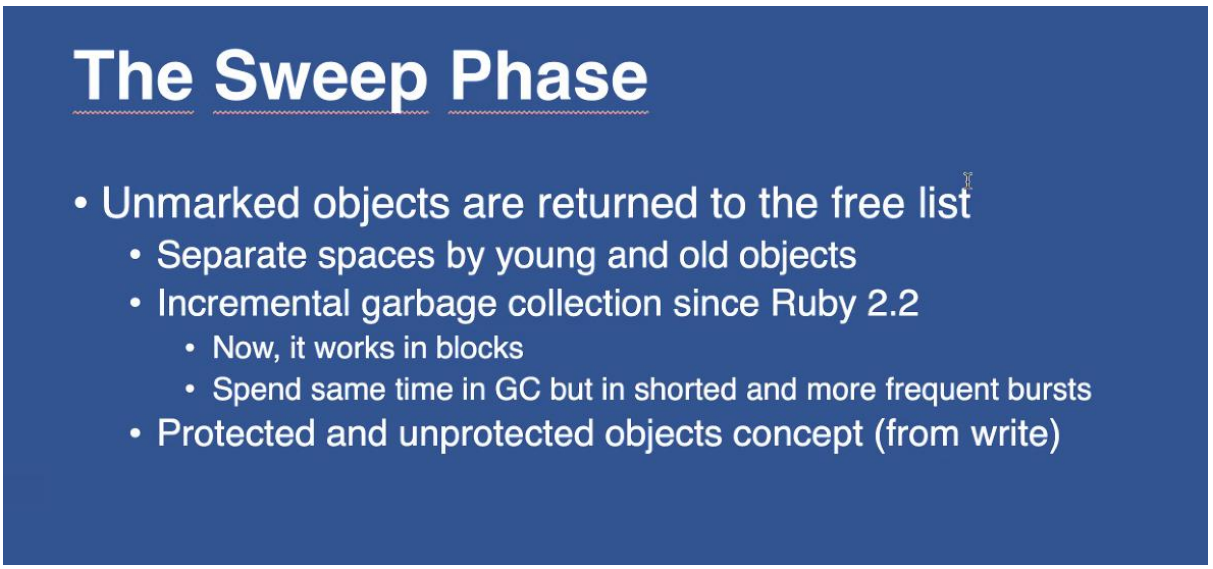
We cannot do multiple inheritance.

A module can have a class inside of it, but it can only be used inside of the module.

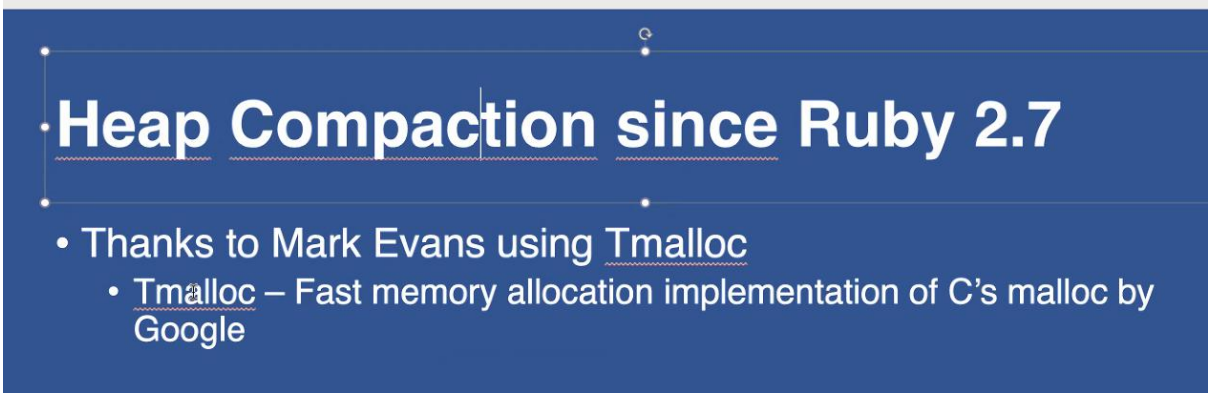
Removes things that are not useful from the memory automatically.

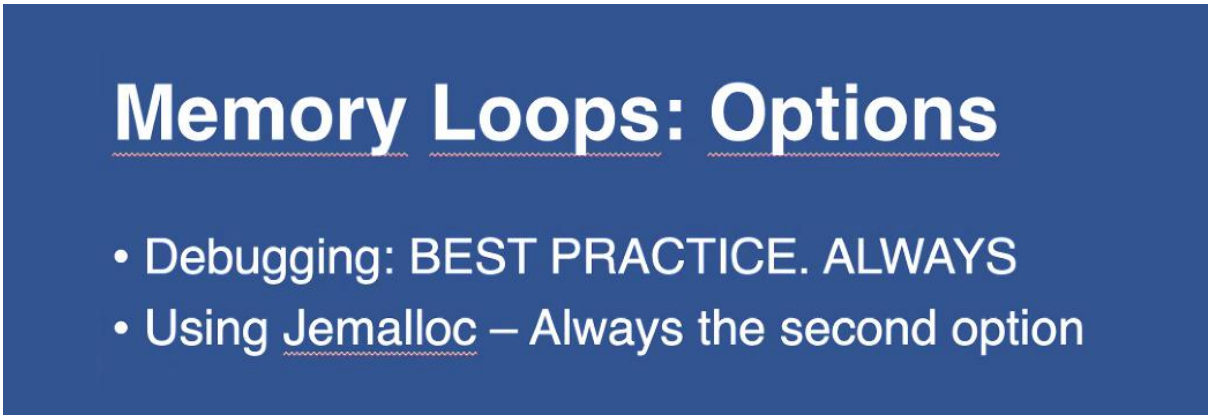
Phases





End of garbage collector phases…





Jemalloc
useful gem to debug memory loops, and memory usage?

MEMORY LOOP EQUALS MEMORY LEAK

```ruby
class MyClass

    def public_method
        my_method
    end

    protected

    def my_method
        puts "im protected"
    end
end


module MyModule
    class Hello
        def initialize
            puts "new class"
        end
    end

    def method
        Hello.new
    end
end

def ok
    a = 2
    b = Proc.new { a *= 2 }
    b.call
end

def oki
    c = -> { return 3 }
    c.call
end

MyClass.public_instance_methods[0]
```

Manera correcta
Para llamar un método de un módulo, solo se puede llamar dentro de un módulo. Estas están protegidas dentro del scope del módulo.