Lesson 5:
Using modules & exception handling

What is a module?

A module is like a class, except:
– you cannot create a new instance of a module
– you cannot extend a module to create a child module

Modules are defined using a module … end block

```
module Morals
    def has_morals?
        true
    end
end

module1 = Morals.new
=> Undefined method ’new’ for Morals
```

What are modules used for then?

As a collection of useful methods and constants

module methods are called by module name

```
module Morals
    def has_morals?
        true
    end
end

puts Morals.has_morals?
=> true
```

What are modules used for then?

As a way to include common functionality in a class.

You can include a module in a class.

```
module Morals
    def has_morals?
        true
    end
end

class User
    include Morals
end

rik = User.new
puts rik.has_morals?
=> true
```

Why would you write code in modules?

– good code should be reused
– not all code logically groups together as a class, sometimes you just have a set of related methods and constants
– modules provide a “namespace” to organize and keep otherwise similarly named functionality distinct

```
# a method for calculating logarithm of x
Math.log(3.56)

# a method for logging messages in a game
Game.log(“Player died”)
```

Constants

Modules are commonly used to hold frequently used constants

A constant is a value which does not change at runtime

By convention, constants are named in all capital letters, a warning will be given if you change a variable which starts with a capital letter

```
# All caps means constant
APP_HOME = 127.0.0.1

# Overwrite constant
APP_HOME = 229.42.8.16

puts APP_HOME
=> warning: already initialized constant APP_HOME
```

A module is defined in a module … end block

Like a class, you'll commonly define a module in a distinct file

In this block, you define constants and methods

```
module Morals
    # Constants
    VERY_BAD = 0
    VERY_GOOD = 5

    # Method
    def sinned
       “Naughty naughty”
    end
end
```

To use a module defined in a separate file, require it just like a class then include the module by module name before using it

```
# Make sure we can require the right files
$LOAD_PATH.unshift(File.dirname(__FILE__))

# Require the modules file
require 'lib/morals'

# Including the module in the file
include Morals

puts Morals::VERY_GOOD
=> 5

puts Morals.sinned()
=> Naughty naughty
```

Exercise:
Defining & building a custom module

What modules come built-in with Ruby?

Ruby exposes much core functionality through modules and the built-in modules do not need an include statement to use.

A full list can be seen here:
http://www.ruby-doc.org/docs/ProgrammingRuby/html/builtins.html

A commonly used built in module is Math
The :: operator is used to refer to a constant set in a module

```
puts Math.sqrt(9)
=> 3.0

puts Math::PI
=> 3.1415926
```

If you include a module in class, it gains its constants and methods.

This is an easy way to reuse functionality among classes and can be used to ensure your class includes behavior other classes expect.

```
module MyAppUtils
    def sentence_shuffler(sentence)
        return sentence.split.shuffle!.to_s
    end
end

class Paragraph
    include MyAppUtils
end

p1 = Paragraph.new
puts p1.sentence_shuffler("It was dark and stormy")
=> ["stormy", "dark", "and", "It", "was"]
```

Can built in modules be used as mixins?

Enumerable module provides dozens of collection handling methods including: find, include? and group_by

http://www.ruby-doc.org/core-1.9.3/Enumerable.html

Comparable module provides support for comparison operators (methods): <, <=, ==, >=, >, between?

http://www.ruby-doc.org/core-1.9.3/Comparable.html

```
class Team
    include Enumerable

    def initialize(*members)
        @team = members
    end

    def each
        for member in @team
            # yield is like return but doesn’t stop method
            yield(member)
        end
    end
end

flintstones = Team.new("Fred", "Wilma", "Barney", "Betty")

# include? is from Enumerable
puts flintstones.include?("Wilma")
=> true
```

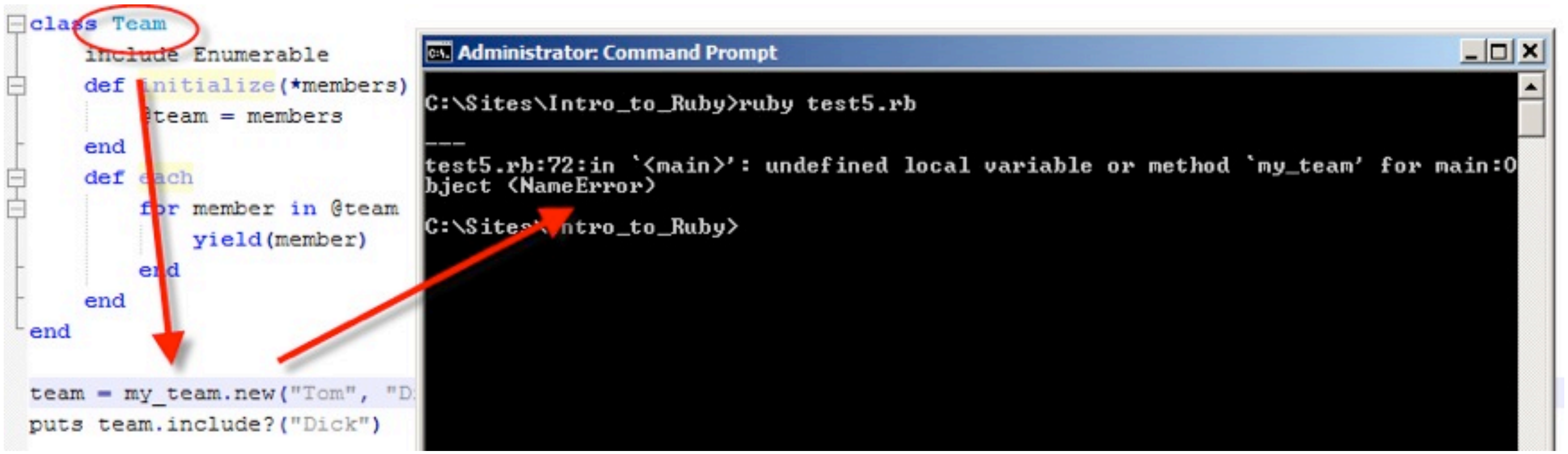
```
class Song
    # more code up here, initialize, etc

    include Comparable
    def <=>(other)
        self.duration <=> other.duration
    end
end

song1 = Song.new("My Way", "Frank Sinatra", 225)
song2 = Song.new("Bohmenian Rhapsody", "Queen", 355)

song1 <=> song2
=> -1

song1 < song2
=> true

song1 > song2
=> false
```

Exercise:
Using a module as a class mixin

Exceptions

By default, program errors stop execution and raise an exception

The exception identifies the nature of the error

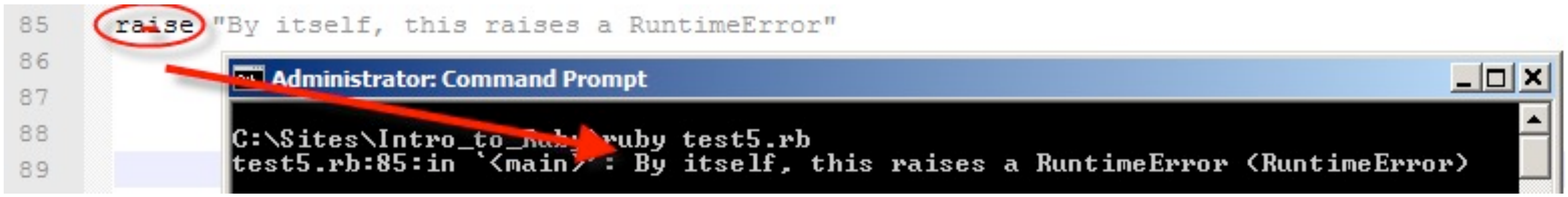


```
class Team
    include Enumerable
    def initialize(*members)
        @team = members
    end
    def each
        for member in @team
            yield(member)
        end
    end
end

team = my_team.new("Tom", "D
puts team.include?("Dick")
```

```
C:\Sites\Intro_to_Ruby>ruby test5.rb
---
test5.rb:72:in `<main>': undefined local variable or method `my_team' for main:O
bject (NameError)

C:\Sites\Intro_to_Ruby>
```

Can you choose when and what type of exceptions occur?

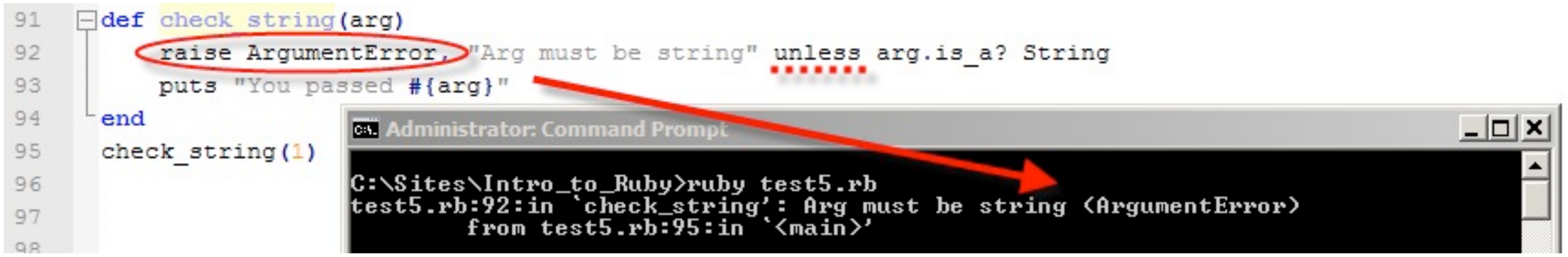
Yes, by default the raise statement cause a RuntimeError object to be thrown



```ruby
raise "By itself, this raises a RuntimeError"
```

```
C:\Sites\Intro_to_Ruby>ruby test5.rb
test5.rb:85:in `<main>': By itself, this raises a RuntimeError (RuntimeError)
```

You may also conditionally raise specific exception types



```ruby
def check_string(arg)
    raise ArgumentError, "Arg must be string" unless arg.is_a? String
    puts "You passed #{arg}"
end
check_string(1)
```

```
C:\Sites\Intro_to_Ruby>ruby test5.rb
test5.rb:92:in `check_string': Arg must be string (ArgumentError)
        from test5.rb:95:in `<main>'
```

You or the system can raise an instance of the Exception class or any derived subclass from the Exception class.

Generally :
– your code would raise exceptions from the StandardError branch
– system code raises exceptions from other branches of Exception

The "family tree" of the Exception class can be viewed here
http://rubylearning.com/images/exception.jpg

You can also define your own sub-classes from Exception and StandardError

Can Ruby respond more gracefully than shutting down?

Yes, you implement exception handlers to route exceptions to desired responses

A rescue block catches a raised exception and runs its own code

You can access exception object properties by mapping it to a variable with =>

Program execution continues after the end of the block which raised the exception

```
def check_me(x)
    unless x.is_a?(String)
        raise ArgumentError, “must be a string!”
    end

    puts “You passed me #{x}”

    rescue ArgumentError => error
        puts “Oops, #{error.message}”
end

check_me(2)
=> “Oops, must be a string!”

check_me(“Yep”)
=> “You passed me Yep”
```

```ruby
begin
    # some code which may
    # raise an exception
rescue SomeError => error
    # run if SomeError occurs
rescue SomeOtherError => error
    # run if SomeOtherError occurs
rescue
    # run if any other child
    # of StandardError occurs
ensure
    # always run whether or not
    # an exception is raised
else
    # run only if no exceptions
    # are raised
end

# execution continues here
```

How do I implement a custom exception type?

You may extend StandardError to define your own program specific exceptions.

Commonly you may override the built class message variable

```
class ColorChoiceException < StandardError
    def initialize(message=”My default error message.”)
        super(message)
    end
end
```

```
$LOAD_PATH.unshift(File.dirname(__FILE__))
require ’lib/color_choice_exception‘

def check_color(color)
    unless color != ”red”
        raise ColorChoiceException, ”No red!”
    end

    puts ”Color is: #{color}”

    rescue ColorChoiceException => error
        puts error.message
end

check_color(”red”)
=> No red!
```

Exercise:

Raise and handle

built-in & custom exceptions

Lab:

Using modules &

exception handling