RUBY EMG

> Ruby, A Programmer's Best Friend

Table Of Contents

Overview

Ruby: Interpreted, dynamic, object-oriented scripting language.

Created By: Matz 1993

Inspiration: Combines Python’s simplicity with Smalltalk’s true OOP.

Rise in Popularity

Boom: After David Heinemeier Hansson (DHH) released Ruby on Rails (Rails) in 2005.

Why Popular?:

- Rails = full-stack MVC web framework.
- Helped startups rapidly build web apps.
- Used By: Twitter, Shopify, GitHub, Airbnb.

What is an Object in OOP (Object-Oriented Programming)?

An object is a real-world entity or instance of a class that contains:

- Data (called attributes or properties)
- Behavior (defined through methods or functions)

MVC: Model–View–Controller

MVC is a software design pattern commonly used for developing user interfaces that divide an application into three interconnected components:

Components of MVC

Component	Responsibility
Model	Handles data, business logic, and rules of the application.
View	Manages the UI – what the user sees.
Controller	Acts as the intermediary between Model and View – handles user input, processes it, and updates Model/View accordingly.

Package Management

RubyGems: The official package manager.

Gem: Self-contained library/module/package.

Setup

```
sudo apt install ruby
ruby -v    # Should print something like: ruby 3.x.x
```

```
nano file.rb
ruby file.rb
```

Takeaways

- Simple, English-like syntax
- Everything is an object
- Full OOP + functional flexibility
- Duck typing allows polymorphism without inheritance
- Rails made it legendary for web apps
- Huge ecosystem with RubyGems

ruby.rb

- install ruby `sudo apt-get install ruby-full`
- save file in extension `file.rb`
- run command `ruby file.rb`

```ruby
# extension: file.rb
# run:ruby file.rb
# comment

# ruby is true OPPS lang
# dynamically typed
# everything is an object and every object can be modified


# var, now datatype declare
name = "Bali"
$age = 25 # $global_var
# CONST
LANG = "Ruby"
# $GlobalVar
$GlobalVar = "3.0"

# each stuff is an object
# thus object methods play is possible
puts "hello".upcase() # () parentheses are optional
LANG = LANG.upcase #this works but with warning as re-assinging constants
is not supposed norm,warning: already initialized constant LANG

puts LANG

# method declare
def greet(name)
    "Hello, #{name}"
end

puts greet("Bhati")


name = "robin"
puts "Hi, my name is #{name.upcase}"
puts "2+2=#{2+2}"
```

```
print " ,okay :)\n"

# print, print without \n
# puts , print + \n
# return, return obj

dt1=-14 #int
dt2=-14.2 #float
dt3="String"#string
dt4=true#boolean
dt5=nil#null
dt6=[1,2,1.2]#array
dt7={id:"2123"}#hash
dt8=:lightstring #lightstring

puts dt1.class
puts dt2.class
puts dt3.class
puts dt4.class
puts dt5.class
puts dt6.class
puts dt7.class
puts dt8.class
# Integer
# Float
# String
# TrueClass
# NilClass
# Array
# Hash
# Symbol

#string methods
name = "Ruby"
puts name.downcase
puts name.upcase #upcase, not uppercase
puts name.class
puts name.length
puts name.reverse
puts name.include?("B")
# ruby
# RUBY
# String
# 4
# ybuR
# false

# numbers
a= 3
b = 16
puts a + b # 19
# + - * / % .to_f
a=a+b/(a*b)
puts a.to_f #3.0
```

```
# boolean
puts true && false # and
puts true || false # or
puts !true # not
# false
# true
# false

# Array
fruits = ["apple", "banana", "mango"]
puts fruits
# apple
# banana
# mango
puts fruits[0] # apple
fruits.each {|n| puts n.upcase() }
# APPLE
# BANANA
# MANGO

# Hashes
user = { name: "Alice", age: 25 }
puts user # {:name=>"Alice", :age=>25}
puts user[:name] #Alice
puts user.keys
# name
# age


# conditions
# ruby uses if,else,elsif(not else if or elif) like python but req. 'end'
to close
# rube is indented language
age = 18
if age >=18
    puts "YouAreAdult"
elsif age == 17
    puts "WaitOneYear"
else
    puts "Minor"
end

# .between?()
# if shortcut
puts "TEen" if age.between?(13,19)

# comparison operators
# == != > < <= >=

# Loops in ruby
# 1 while loop
# do till its true
i = 1
```

```
while i<=5
    puts "Counter #{i}"
    i+=1# not i++
end

# 2 Until loop
# do untill this is true
# i.e. do till only false
until i>10
    puts "Counter #{i}"
    i+=1
end

# 3 times loop
10.times do |i| # i is var inside times loop (default 0)
    puts "TImes #{i}"
end

puts "TImes #{i}"

# 4 for loop
for i in 1..15
    puts "#{i}"
end

# 5 each loop
arr1=[1,2,4]
arr1.each do |e|
    puts "arr1 #{e}"
end

# Methods in Ruby
# functions
# def -- end
# return only one object
# they return last evaluated (explicit return is optional)

def greet(name="Bali") # arguement and its default setup
  # This method takes one argument and returns a greeting string
  puts "Hello, #{name}!"
end

greet("Bhati")
greet()

def sum(n,n2)
    return n+n2  # Optional; could omit `return`
end
puts sum(1,2)

# methods with conditions
def even_or_odd(num)
  if num % 2 == 0
    "Even"
  else
    "Odd"
  end
end

puts even_or_odd(10)   # Even
puts even_or_odd(7)    # Odd

# Blocks in ruby
# anonymous chunks of code you can pass to methods
# written do ... end or { ... }
# anonymous: not stored in a variable or named method.

# each with do ... end block
[1,2,4].each do |n| # |pipe cahracter are arguements starting from 0|
    puts n
end

# same with { ... } block
[5,6,8].each{ |n| puts n }


# yield = putting statement as arguement , can be repetitive in method
# yield is a keyworld used inside a method to call a block that was passed
implicitly


def greet
  puts "Before yield"
  yield
  puts "After yield"
end

greet { puts "STATEMENT AS ARGUEMENT "}


# multiple calls
def twice
  yield
  yield
end

twice { puts "Run Block" }


# block with arguements
def food_time
  yield("Pizza", 2)
end

food_time do |food,qty|
    puts "#{qty} #{food}s" # 2 Pizzas
end
```

```
# Symbols
# immutable identifier used as name or label
# begins with colon :eg

:admin
:email
:username
:token

# symbols are memory effient than strings when reused
puts :admin == :admin #true
puts "admin" == "admin" #true
# but symbols always point to same internal object
puts :admin.object_id
puts :admin.object_id
puts :admin.object_id
puts "admin".object_id
puts "admin".object_id
puts "admin".object_id
# 1355868
# 1355868
# 1355868
# 60
# 80
# 100

# keyword args in methods
# Instead of passing args by position, Ruby lets you name them (like
Python)

def creatuser(name:,age:) # here its name: (not :name)
    puts "Created #{name}, age #{age}" # here name (not name:)
end

creatuser(name:"Bob",age:22) #here name:
# Created Bob, age 22

# OOPS basics
# Classes and Objects
# class
# blueprint for creating objs with shared behaviour
class User
  # attr_accessor creates both getter and setter methods automatically
  # this gives access to @name and @email from outside the object
  attr_accessor :name, :email

  # initialize method
  # run when you do User.new(...)
  # generates instance variables like @name, @email

  def initialize(name,email)
    @name = name #instance variable for this object
    @email =email
  end

  # instance is a single object created form class
  # class = blueprint = recipe
  # instance = actual object made from it = cake
  # instance method

  def greet
    "Hi, i am #{@name}"
  end
end

# create an obj
user1 = User.new("SIddhant Bali","kintsugidevstudio@gmail.com")

# Call instance methods and accessors
puts user1.greet
puts user1.name
puts user1.email
# Hi, i am SIddhant Bali
# SIddhant Bali
# kintsugidevstudio@gmail.com

# Changing values using setter
user1.name="Siddhant Bali"
puts user1 # #<User:0x0000735b55c97880>
puts user1.name
puts user1.greet
# Siddhant Bali
# Hi, i am Siddhant Bali

# Modules
# container of reuseable methods
# can't initailized like class
# You "include" a module in a class to add its behavior (mixin)
module Printer
  def putts
    print "Name #{@name}\n"
  end
end

# Inheritance
# classes can inherit form others classes using '<'
# inheriting allows to reuse code logic defined in prent class

# parent class
class Employee
  attr_accessor :name, :id

  def initialize(name,id)
    @name = name
    @id = id
  end

  def greet
    "Welcome #{@name}, Your Id is #{@id}"
  end

end


class Admin < Employee
  include Printer  # Mixin: Adds methods from module as instance methods
end

# create object
admin1 = Admin.new("Bali",2022496)
puts admin1.greet
admin1.putts
# Welcome Bali, Your Id is 2022496
# Name Bali
```

Ruby Concepts + Detailed Code Explanation

---

1. Introduction to Ruby Programming

- Ruby is an Object-Oriented Programming Language (OOP).
- It is dynamically typed, meaning variable types are inferred at runtime, not at compile-time.
- Everything in Ruby is an object, and every object can be modified.

```
# Variable Declaration
name = "Bali"  # String variable
$age = 25      # Global variable
```

- Global Variables: `$age` is a global variable, meaning it is accessible throughout the program.

---

2. Constants

- Constants in Ruby are declared using uppercase names and should not be reassigned.

```
# Constants in Ruby
LANG = "Ruby"             # Constant
$GlobalVar = "3.0"        # Another global variable (not a constant)
```

- Warning: If a constant is reassigned, Ruby will give a warning.

---

3. Methods in Ruby

- Methods are declared using `def` and closed with `end`.

- They can return a single object (implicitly or explicitly).
- Parameters can have default values.

```
# Method with a parameter
def greet(name)
    "Hello, #{name}"
end

puts greet("Bhati")  # Output: Hello, Bhati
```

- Default Values in Methods: If no argument is passed, a default value can be used.

```
def greet(name="Bali")
  puts "Hello, #{name}!"
end

greet("Bhati")  # Output: Hello, Bhati
greet()         # Output: Hello, Bali
```

---

4. String Manipulation Methods

- Ruby has various built-in string methods for manipulation.

```
# String Methods
name = "Ruby"
puts name.downcase  # ruby
puts name.upcase    # RUBY
puts name.class     # String
puts name.length    # 4
puts name.reverse   # ybuR
puts name.include?("B") # false
```

- Explanation:
  - `downcase`, `upcase`: Convert the string to lowercase/uppercase.
  - `reverse`: Reverses the string.
  - `include?`: Checks if a substring exists in the string.

---

5. Numbers in Ruby

- Ruby supports basic arithmetic operations and conversion between data types.

```
# Numbers in Ruby
a = 3
```

```
b = 16
puts a + b             # 19
puts (a + b) / (a * b)  # Fraction result
puts a.to_f            # 3.0 (convert to float)
```

6. Boolean Operations

- Ruby provides boolean logic (`and`, `or`, `not`).

```
# Boolean Operations
puts true && false     # false (AND)
puts true || false     # true (OR)
puts !true             # false (NOT)
```

7. Arrays in Ruby

- Arrays are ordered collections that can hold multiple data types.

```
# Arrays in Ruby
fruits = ["apple", "banana", "mango"]
puts fruits
puts fruits[0]           # apple
fruits.each {|n| puts n.upcase}  # APPLE, BANANA, MANGO
```

8. Hashes in Ruby

- Hashes are key-value pairs.

```
# Hash in Ruby
user = { name: "Alice", age: 25 }
puts user              # {:name=>"Alice", :age=>25}
puts user[:name]       # Alice
puts user.keys         # [:name, :age]
```

9. Conditional Statements

- Ruby uses `if`, `elsif`, `else`, and `end`.

```
# Conditional Statements
age = 18
if age >= 18
```

```
    puts "You are an adult"
elsif age == 17
    puts "Wait one year"
else
    puts "Minor"
end
```

- Shortcut for conditions: You can use `between?` to check ranges.

```
# If shortcut
puts "Teen" if age.between?(13, 19)  # Output: Teen
```

---

10. Loops in Ruby

- Ruby supports multiple loop types:

1. While Loop

```
i = 1
while i <= 5
    puts "Counter #{i}"
    i += 1
end
```

2. Until Loop

```
i = 1
until i > 10
    puts "Counter #{i}"
    i += 1
end
```

3. Times Loop

```
10.times do |i|
    puts "Times #{i}"
end
```

4. For Loop

```
for i in 1..15
    puts "#{i}"
end
```

5. Each Loop

```
arr1 = [1, 2, 4]
arr1.each do |e|
    puts "arr1 #{e}"
end
```

---

11. Defining Blocks

- Blocks are anonymous functions passed to methods.

```
# Each loop with a block
[5, 6, 8].each { |n| puts n }

# Yielding inside a method
def greet
    puts "Before yield"
    yield
    puts "After yield"
end

greet { puts "Hello from the block!" }
```

---

12. Symbols

- Symbols are lightweight strings used for identifiers.

```
# Symbols in Ruby
puts :admin == :admin  # true
puts "admin" == "admin" # true
```

- Symbols are memory efficient and point to the same internal object.

---

13. Keyword Arguments in Methods

- You can define named arguments in Ruby methods.

```
# Keyword arguments
def create_user(name:, age:)
    puts "Created #{name}, age #{age}"
end

create_user(name: "Bob", age: 22)
```

---

14. Classes and Objects

- Classes define the blueprint for objects. Objects are instances of a class.

```
# Class Definition
class User
  attr_accessor :name, :email

  def initialize(name, email)
    @name = name
    @email = email
  end

  def greet
    "Hi, I am #{@name}"
  end
end

# Creating an object
user1 = User.new("Siddhant Bali", "kintsugidevstudio@gmail.com")

# Calling methods
puts user1.greet     # Hi, I am Siddhant Bali
puts user1.name      # Siddhant Bali
```

- `attr_accessor`: Automatically creates getter and setter methods for instance variables.

---

15. Modules in Ruby

- Modules provide reusable methods that can be mixed into classes.

```
# Defining a module
module Printer
  def print_name
    puts "Name #{@name}"
  end
end

# Including a module
```

```ruby
class Admin
  include Printer
end

admin = Admin.new("Bali", 2022496)
admin.print_name  # Name Bali
```

---

16. Inheritance in Ruby

- Ruby allows classes to inherit from other classes, enabling reuse of code.

```ruby
# Parent Class
class Employee
  attr_accessor :name, :id

  def initialize(name, id)
    @name = name
    @id = id
  end

  def greet
    "Welcome #{@name}, Your Id is #{@id}"
  end
end

# Child Class Inheritance
class Admin < Employee
  include Printer
end

admin1 = Admin.new("Bali", 2022496)
puts admin1.greet    # Welcome Bali, Your Id is 2022496
admin1.print_name    # Name Bali
```

- Inheritance: `Admin` class inherits from `Employee`, reusing `greet` and adding `print_name` functionality through `Printer` module.