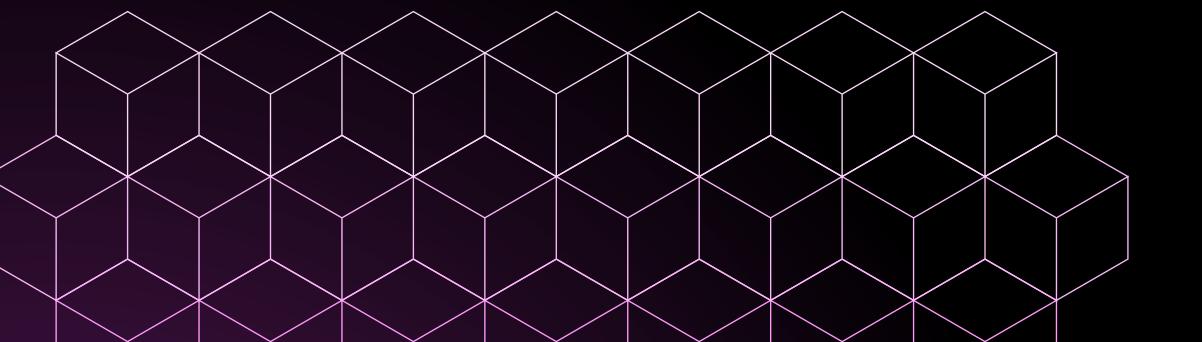
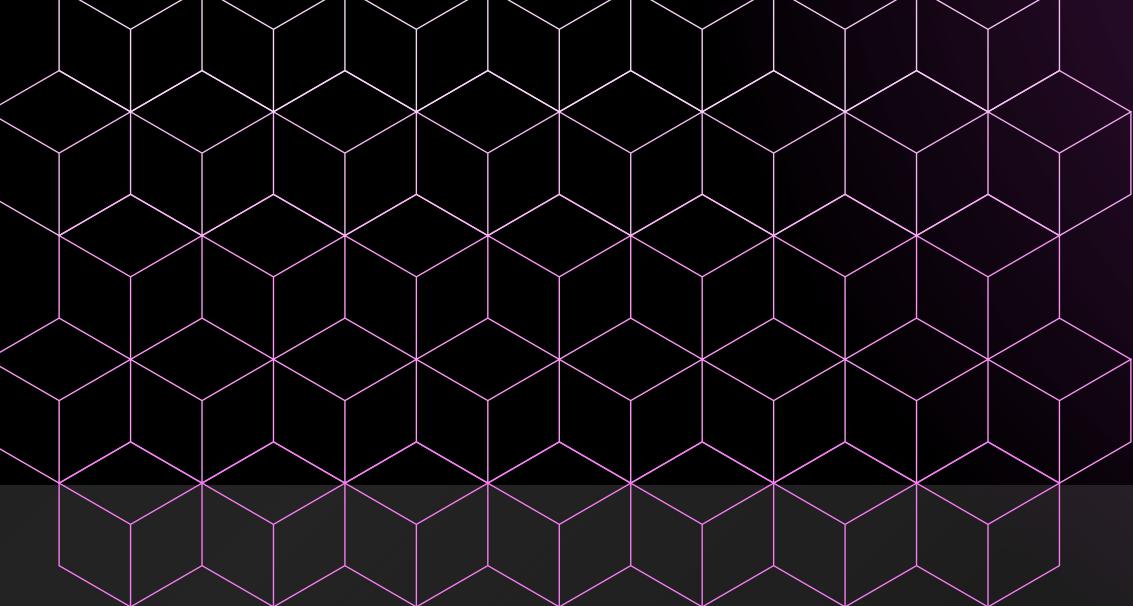
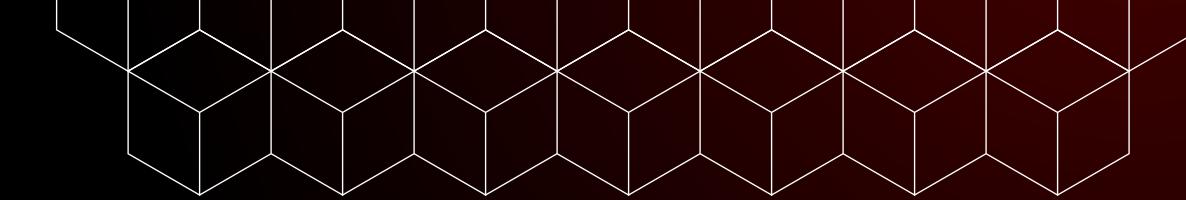


# RUBY

Yulian Andrés Ortega Machado - 1152485  
Julián Andrés Izquierdo Gallardo - 1152475  
Maria José Osorio Ovallos - 1152478  
Juan Felipe Corzo Arias - 1152473





## 3.3.6

La versión actual de Ruby es la 3.3.6, lanzada en Noviembre de 2024. El marco de versiones general es el 3.0.0, que benefició al desarrollo de videojuegos e Inteligencia Artificial.

Release Version	Release Date	Download URL	Release Notes
Ruby 3.3.6	2024-11-05	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.2.6	2024-10-30	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.4.0-preview2	2024-10-07	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.3.5	2024-09-03	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.2.5	2024-07-26	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.3.4	2024-07-09	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.3.3	2024-06-12	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.3.2	2024-05-30	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.1.6	2024-05-29	<a href="#">download</a>	<a href="#">more...</a>
Ruby 3.4.0-preview1	2024-05-16	<a href="#">download</a>	<a href="#">more...</a>

# VERSIÓN ACTUAL



## 1993-1995

Entre estos años, Yukihiro Matsumoto trabajó en la creación de un lenguaje pensado para la productividad del programador. El nombre del programa se debió a una burla provocada por las similitudes que tenía con Perl.



## 2005

En el año 2005, se presentó el framework Ruby on Rails. Rails hizo que el desarrollo web fuera mucho más rápido y eficiente, y esto catapultó a Ruby a un nivel de popularidad global. Rails convirtió a Ruby en el lenguaje preferido para startups como GitHub.

## 2013

En este año, se lanzó la versión 2.0 de Ruby, una versión que trajo características como los keyword arguments, la recolección de basura más eficiente (mejorando la memoria en aplicaciones grandes), y el nuevo motor de ejecución llamado "YARV" (Yet Another Ruby Virtual Machine).

# HISTORIA



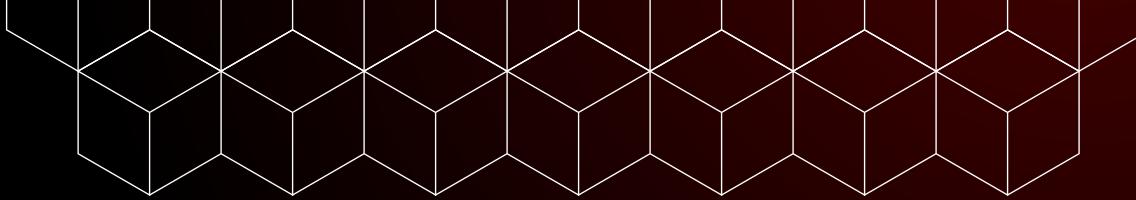
```
# --- 🔥 Instance variable
class Song
  def initialize(name: nil, genre: nil, artist: nil)
    @name = name
    @genre = genre
    @artist = artist
  end

  def details
    puts 'Song details'
    puts "Name -> #{@name}"
    puts "Genre -> #{@genre}"
    puts "Artist -> #{@artist}"
  end
end
# --
first_song = Song.new(artist: 'A.M.C', name: 'Mind the Gap', genre: 'Drum&Bass')
first_song.details
# puts @name # => nil
```

- Interpretado: Ruby es un lenguaje de programación que no compila, sino que usa un intérprete para traducir las líneas de código a lenguaje de la máquina.
- Orientado a Objetos: Todo en Ruby es un objeto, incluso los datos primitivos, facilitando la creación de clases.
- Tipado Dinámico: las variables pueden cambiar el tipo de dato que contienen frecuentemente.
- Sintaxis simple: hace uso de un inglés simple e intuitivo, evitando el uso de abreviaciones innecesarias.

# CARACTERÍSTICAS PRINCIPALES



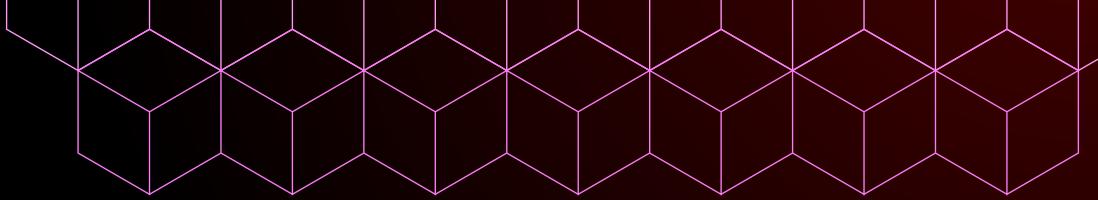


Ranking: Si bien, en general, Ruby no está dentro del top 10 lenguajes más usados según TIOBE, en el ámbito del desarrollo web, Ruby se ubica entre los 7 lenguajes de programación más usado para esta labor.

Utilidad: Ruby es un lenguaje de alto nivel, lo cual junto a su framework Rails ha contribuido a que se construyan aplicaciones web de forma rápida y eficiente gracias a su arquitectura basada en el modelo MVC.



# RANKING Y UTILIDAD



## MÉTODOS

Los métodos se inicializan con ‘def’ y cierran con end.

```
def saludar(nombre)
    puts "¡Hola, #{nombre}!"
end

saludar("Carlos") # Output
```

## PROPIEDADES

Las propiedades en Ruby son variables de instancia iniciadas por @

```
class Carro
    def initialize(marca, color)
        @marca = marca
        @color = color
    end

    def mostrar_info
        puts "Marca: #{@marca}, Color: #{@color}"
    end
end

mi_carro = Carro.new("Toyota", "Rojo")
mi_carro.mostrar_info # Output: Marca: Toyota
```

## ENCAPSULAMIENTO

En Ruby aparte de la existencia de private, public y protected, existen métodos de acceso.

```
class Persona
    attr_accessor :nombre

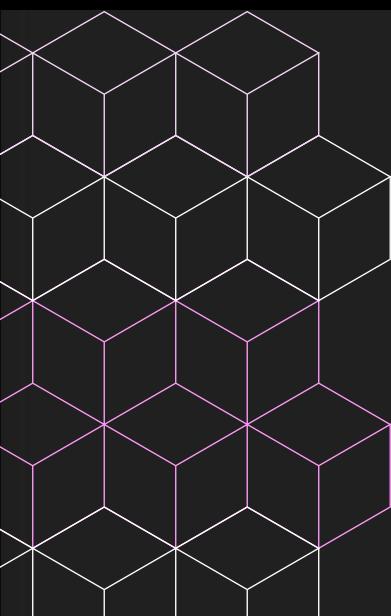
    def initialize(nombre, edad)
        @nombre = nombre
        @edad = edad
    end

    def mostrar_info
        puts "Nombre: #{@nombre}, Edad: #{@edad}"
    end

    private

    # Este método es privado y solo accesible desde dentro
    def mostrar_edad
        @edad
    end
end
```

# CLASES Y OBJETOS



## Polimorfismo

En Ruby, el polimorfismo se logra a través de herencia y duck typing (tipado pato). En tipado pato, un objeto se considera adecuado para un uso si tiene los métodos necesarios, sin importar su tipo explícito.

## Herencia

En Ruby, esto se logra utilizando el símbolo <. La clase que hereda se llama subclase (o clase hija) y la clase de la que hereda se llama superclase (o clase padre).

```
class Animal
  def respirar
    puts "Respirando..."
  end

  class Perro < Animal
    def ladrar
      puts "Guau guau!"
    end
  end

  class Gato < Animal
    def maullar
      puts "Miau miau!"
    end
  end
```

```
class Pato
  def hacer_sonido
    puts "Cuac cuac"
  end
end

class Perro
  def hacer_sonido
    puts "Guau guau"
  end
end

animales = [Pato.new, Perro.new]

animales.each do |animal|
  animal.hacer_sonido
end
```

# HERENCIA Y POLIMORFISMO

## Asociación

La asociación permite que un objeto se "conozca" o interactúe con otro. Se logra suele a través de la creación de referencias entre objetos.

```
class Persona
  attr_accessor :nombre

  def initialize(nombre)
    @nombre = nombre
  end
end

class Direccion
  attr_accessor :calle, :ciudad

  def initialize(calle, ciudad)
    @calle = calle
    @ciudad = ciudad
  end
end

class Empleado
  attr_accessor :persona, :direccion

  def initialize(persona, direccion)
    @persona = persona
    @direccion = direccion
  end
end
```

## Agregación

En Ruby, la agregación se puede representar creando un objeto "contenedor" que tiene una referencia a otro objeto, pero sin tener control sobre la creación o destrucción de ese objeto.

```
class Departamento
  attr_accessor :nombre, :empleados

  def initialize(nombre)
    @nombre = nombre
    @empleados = []
  end

  def agregar_empleado(empleado)
    @empleados << empleado
  end

  class Empleado
    attr_accessor :nombre

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

## Composición

En la composición, el ciclo de vida de los objetos contenidos depende del ciclo de vida del objeto que los contiene.

```
class Motor
  def arrancar
    puts "El motor está arrancando."
  end
end

class Coche
  attr_accessor :motor

  def initialize
    @motor = Motor.new
  end

  def arrancar_coche
    @motor.arrancar
    puts "El coche está arrancando."
  end
end
```

# CONTENEDORES