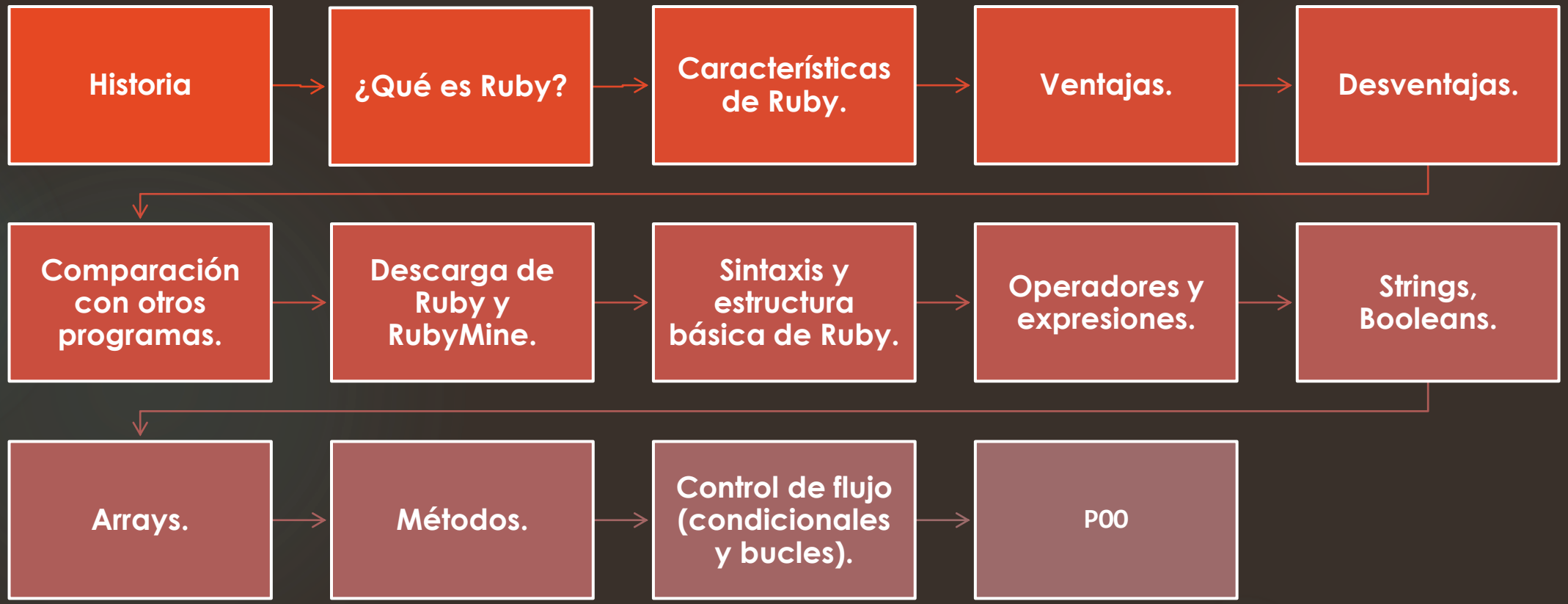
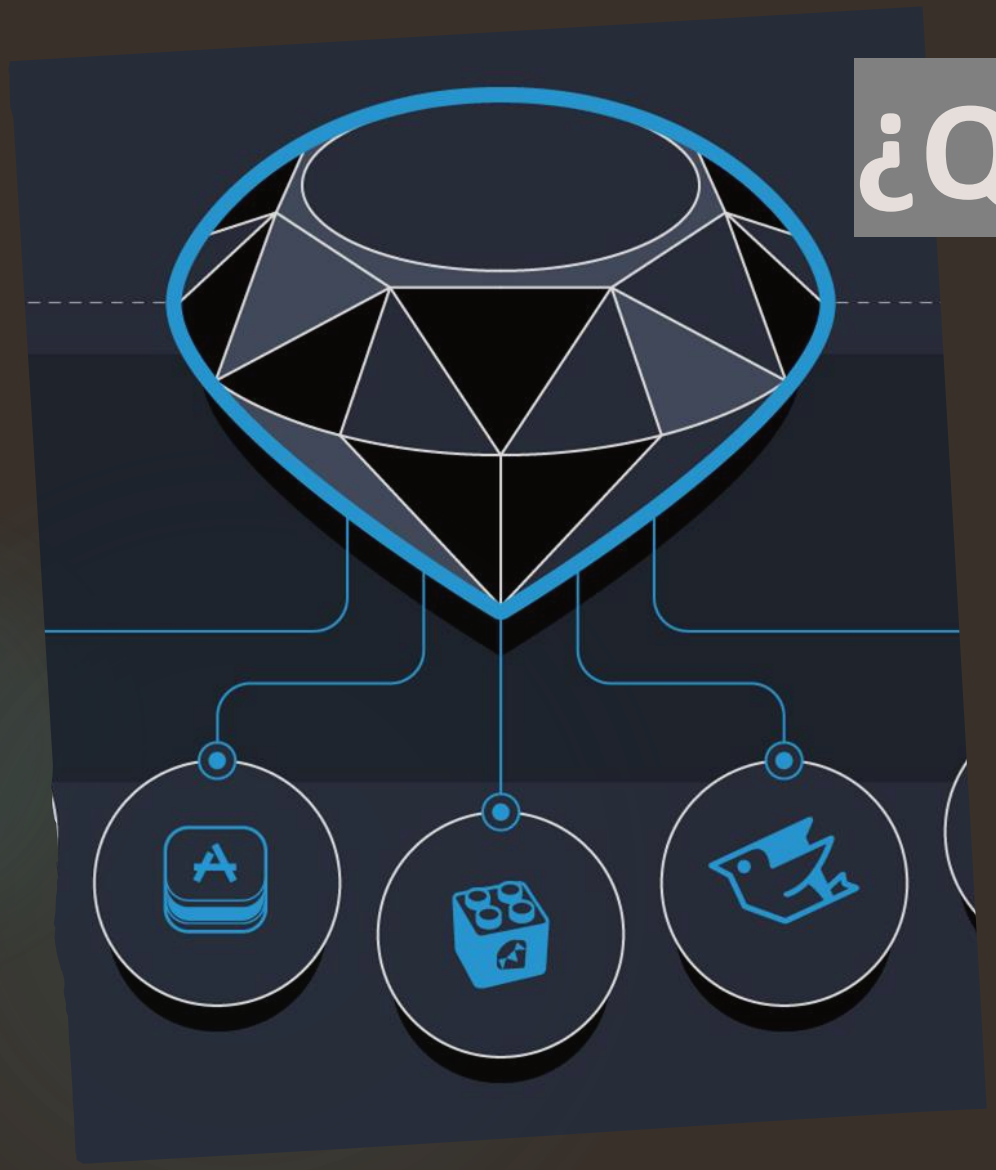
MARIEL ROJAS

Agenda


Historia
¿Qué es Ruby?
Características de Ruby.
Ventajas.
Desventajas.
Comparación con otros programas.
Descarga de Ruby y RubyMine.
Sintaxis y estructura básica de Ruby.
Operadores y expresiones.
Strings, Booleans.
Arrays.
Métodos.
Control de flujo (condicionales y bucles).
POO

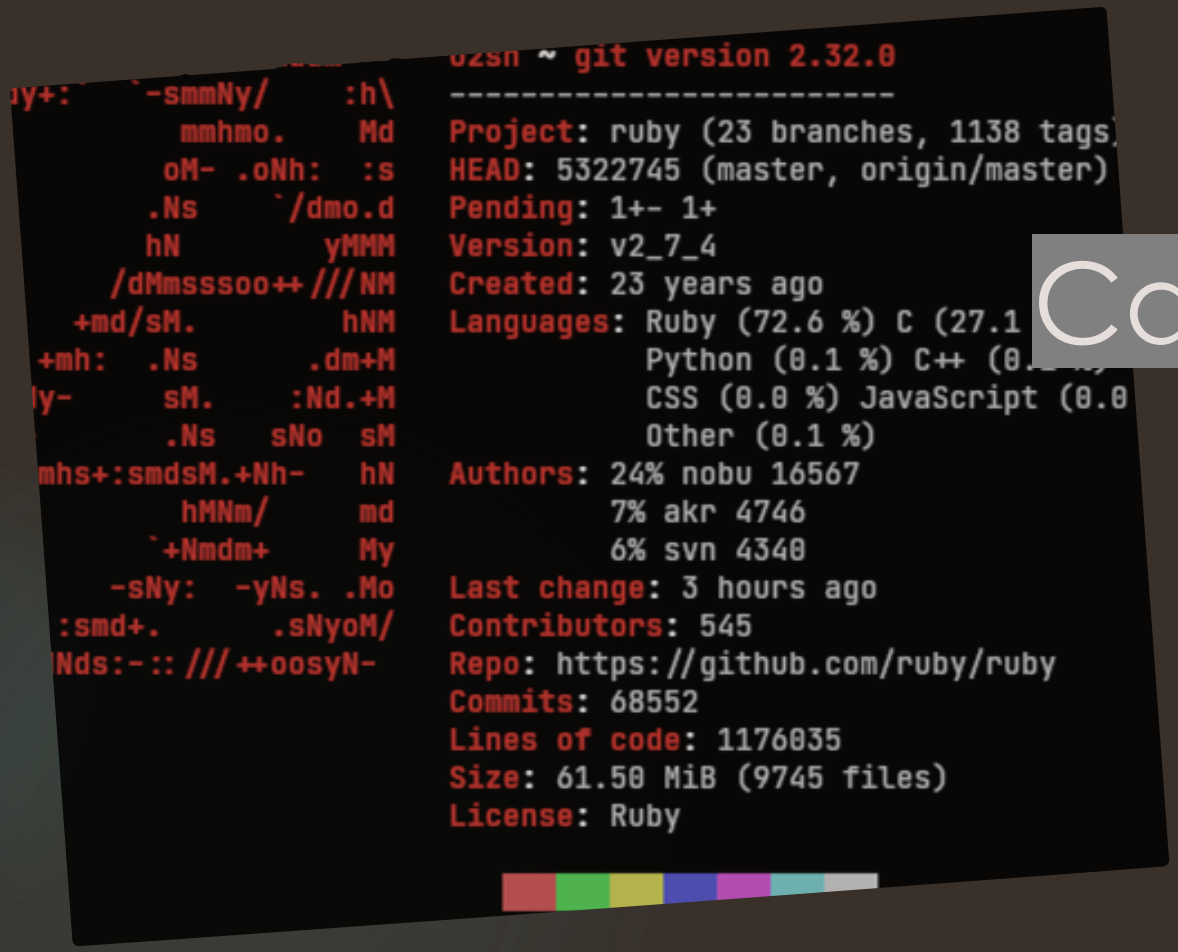
¿Historia?

"La programación debe ser divertida"



¿Qué es Ruby?

- Todo es un objeto.
- Desarrollo web.
- Prototipado rápido:
- Análisis de datos:
- Desarrollo de juegos.
- Desarrollo de juegos



Características

- Lenguaje de Alto Nivel
- Orientación a Objetos
- Interpretado
- Tipado Dinámico
- Colección de Basura (Garbage Collection)
- Idioma Multiplataforma
- Bibliotecas y Frameworks
- Comunidad Activa
- Filosofía de Diseño
- Código Abierto

Ventajas

- Sintaxis Limpia y Legible.
- Orientación a Objetos Pura
- Productividad del Desarrollador
- Comunidad Activa
- Ruby on Rails
- Flexibilidad y Dinamismo
- Multiplataforma
- Codigo Abierto
- Bibliotecas y Gemas

Desventajas

- Rendimiento Relativo
- Consumo de Recursos
- Curva de Aprendizaje
- No es tan popular
- Gestión de la Memoria Limitada
- Escaso Soporte Multihilo
- Limitado para Programación de Alto Rendimiento

Comparación

```
<% provide(:title, 'Sign up') %>
<h1>Sign up</h1>
<div class="row">
  <div class="col-md-6 col-md-offset-3">
    <%= form_with(model: @us, local: true) do |f| %>
      <%= render 'shar
      <%= f.label :nam
      <%= f.text_field
      <%= f.label :email %>
      <%= f.email_field :email, class: 'form-control' %>
```

```
@user          UsersController
@users         UsersController
@current_user  SessionsHelper
Press ^. to choose the selected (or first) sug. Next Tip
```

Ruby on Rails

Características de Ruby on Rails

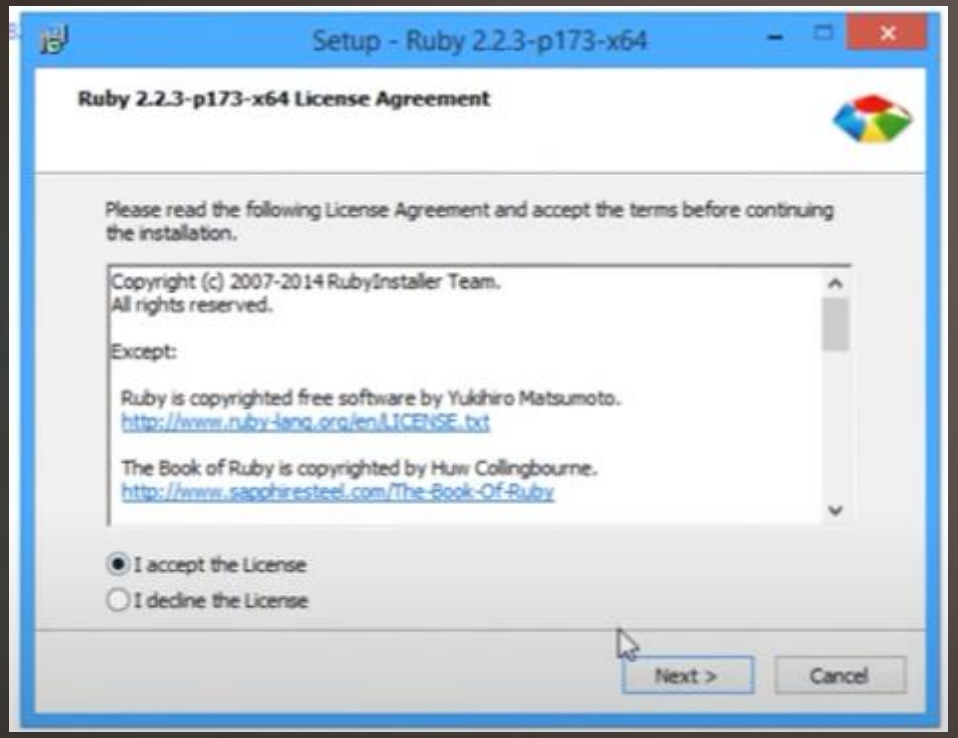
- Convención sobre Configuración (CoC)
- DRY (Don't Repite Yourself)
- Modelo-Vista-Controlador (MVC)
- Andamios
- Gestión de Base de Datos .
- Gemas (Gems).
- Enfoque RESTful .
- Seguridad.
- Comunidad Activa.
- Desarrollo Rápido.

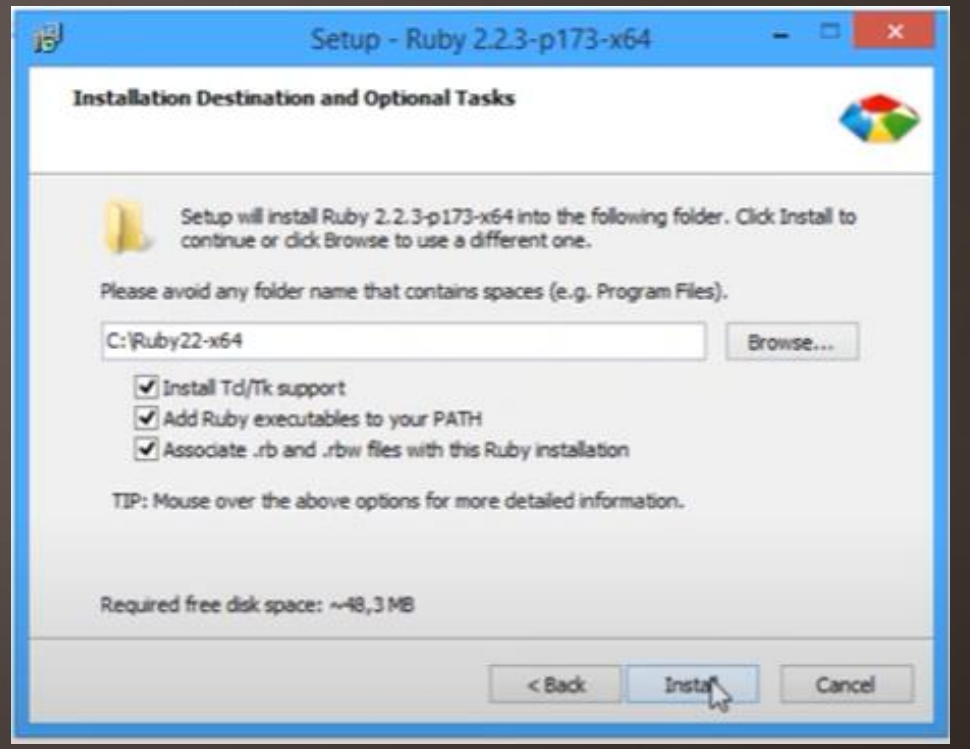
Comparación

Base	Ruby	Ruby on Rails
Principio	Ruby se construyó sobre el principio de composición de la interfaz de usuario.	Ruby on Rails se construyó sobre los principios de *"Convention over configuration"* (CoC) y *"Don't repeat yourself"* (DRY).
Programación	Ruby está programado en el lenguaje de programación C.	Ruby on Rails está programado en el lenguaje Ruby.
Marco de trabajo	Ruby no es un framework, es un lenguaje de programación de propósito general.	Ruby on Rails es un framework de desarrollo de aplicaciones web.
Inspiración	Ruby se inspiró en Smalltalk y Perl.	Ruby on Rails se inspiró en Django y Laravel de PHP y Python respectivamente.
Aplicaciones	Ruby se utiliza para construir aplicaciones de escritorio.	Ruby on Rails se utiliza para construir aplicaciones web
Lenguajes utilizados	Cuando se crean aplicaciones, se utilizan principalmente JAVA, C++ y Vb.net.	Mientras se construyen aplicaciones, se utilizan comúnmente XML, JavaScript, CSS y HTML.
Sintaxis	La sintaxis de Ruby está muy relacionada con la de Python y Perl.	La sintaxis de Ruby on Rails es bastante similar a la de Python, Phoenix en Elixir, etc.

Descarga de programas

- https://rubyinstaller.org/downloads/





RubyMine

- https://www.jetbrains.com/ruby/
- https://www.youtube.com/watch?v=87N3BfxgjIc

Comentarios ,salida y entrada

```ruby
# Así se hacen los comentarios en Ruby
# Mariel Rojas

=begin
Comentarios multilínea
Comentarios de más líneas
=end

puts "Hello con puts" # Crea un salto de línea

print "Hola con print" # No agrega una nueva línea al final

printf "Hola con printf \n" # Permite el formateo de salida

puts 'Hola con comillas simples'
```

```ruby
puts "Ingrese su Nombre: "
name = gets.chomp
puts "Bienvenido, #{name}"
```

Operadores aritméticos

Nombre	Suma	Resta	Multiplicación	División
Símbolo	+	-	*	/

```ruby
# Números
# reales y enteros

# enteros
puts "Números enteros"
puts 49 + 47  # suma
puts 89 - 74 # resta
puts 85 * 2 # multiplicación
puts 100 / 2 # división

puts "\n"
# reales
puts "Números reales"
puts 1.5 + 1.5  # suma
puts 2.5 - 1.5 # resta
puts 2.8 * 3.4 # multiplicación
puts 50.5 / 2 # división
```

```ruby
# Operadores Aritméticos
num1 = 10
num2 = 40

# suma
suma = num1 + num2
print  "El resultado de la suma es: "
puts suma

# resta
resta = num2 - num1
print  "El resultado de la resta es: "
puts resta

# multiplicación
multi = num2 * num1
print  "El resultado de la multiplicación es: "
puts multi

# división
division = num2 / num1
print "El resultado de la división es: "
puts division
```

```ruby
# operadores logicos

# &&
# devuelve true si ambas expresiones que conecta son verdaderas.
# Si al menos una de las expresiones es falsa, el resultado es false
a = true
b = false

resultado = a && b
puts resultado # => false

# ||
# devuelve true si al menos una de las expresiones que conecta es verdadera.
# Si ambas son falsas, el resultado es false
x = true
y = false

resultado = x || y
puts resultado # => true

# !
# Si una expresión es verdadera, ! la convierte en falsa y viceversa
z = true
resultado = !z
puts resultado
resultado = !z
puts

resultado = !
# => false
```

Nombre	Símbolo	Función
NOT	!,~, not	Niega un dato booleano (dato verdadero o falso)
AND	&&, and	Evalúa dos elementos a la vez. Los dos deben de tener el mismo valor (verdadero o falso), caso contrario devuelve falso.
OR	||, or	Escoge el valor entre dos datos, ya sea verdadero o falso.

Operadores lógicos

```ruby
# comparaciones
a = 5
b = 10

igual = (a == b)      # Igual a
menor_que = (a < b)   # Menor que
mayor_que = (a > b)   # Mayor que

puts igual       # => false
puts menor_que   # => true
puts mayor_que   # => false
```

Nombre	Símbolo	Función
Igual que	==	Determina si dos elementos son iguales
No igual o distinto de	=!	Determina si dos elementos son diferentes
Menor que	<	Si un elemento a es menor que un elemento b
Menor o igual que	<=	Determina si es un elemento a es menor o igual que un elemento b
Mayor que	>	Si un elemento a es mayor que un elemento b
Mayor o igual que	>=	Determina si es un elemento a es mayor o igual que un elemento b

Operadores de comparación

Tipos de datos

- Enteros
- Flotantes
- Booleanos
- Caracteres
- Strings

```ruby
# Tipado dinámico en Ruby

variable = 5.0
puts variable.class  # Imprime la clase de la variable (en este caso, Float)

variable = 8
puts variable.class  # Ahora la variable es un entero (Fixnum)

variable = "Mariel"
puts variable.class  # Finalmente, la variable es una cadena (String)
```

Variables de Escape

\\	**Representa un carácter de barra invertida ("\\") en la cadena de texto.**
\”	Representa una coma doble ("\"") en la cadena de texto.
\'	Representa una coma simple (') en la cadena de texto.
\n	: Representa un carácter de nueva línea. Provoca un salto de línea en el texto.
\t	Representa un carácter de tabulación.
\r	Representa un retorno de carro, que puede ser útil en sistemas donde se utiliza para regresar al principio de una línea.
\b	Representa un carácter de retroceso, que se utiliza para mover el cursor hacia atrás una posición.
\f	Representa un carácter de avance de página (avance de formulario).
\v	Representa un carácter de tabulación vertical.
\s	Representa un espacio en blanco

```ruby
# Espacios en blanco y secuencias de escape

puts "Hola" + "Mundo"
puts "Hola\s" + "Mundo"
puts "Hola " * 3
puts "Una cadena con un carácter de nueva línea:\nSegunda línea"
puts "Una cadena con una comilla doble: \"Texto entre comillas\""
puts 'Una cadena con una comilla simple: \'Texto entre comillas\''
puts "Una cadena con un tabulador:\tTexto después del tabulador"
puts "Una cadena con un carácter de retroceso: texto\bborrado"
```

Strings

```ruby
name = "Mariela"
name = "Mariel"
puts "Your name is: " + name

string1 = "Mariel"
string2 = " Rojas"
string3 = " Sánchez"
puts string1 + string2 + string3
```

```ruby
cadena = "Mariel"
puts "El largo de la cadena es de: #{cadena.length}"
puts "Cadena en reversa:  #{cadena.reverse}"
puts "Cadena en mayúscula:  #{cadena.upcase}"
puts "Cadena en minúscula:  #{cadena.downcase}"
puts "Cadena cambiada de mayúsculas a minúsculas y viceversa:  #{cadena.s
puts "Primer letra en mayúscula:  #{cadena.capitalize}"
puts "Mostrar parte de la cadena:  #{cadena.slice(0, 4)}"
```

```ruby
# Creación de un arreglo
nombres = ["Juan", "María", "Pedro", "Luisa"]

# Imprimir el arreglo
puts nombres

# Acceder a elementos del arreglo
primer_nombre = nombres[0]  # Obtiene el primer nombre (Juan)
segundo_nombre = nombres[1] # Obtiene el segundo nombre (María)
puts "Nombre del índice 0 = #{primer_nombre}"

# Modificar un elemento del arreglo
nombres[2] = "Carlos"  # Cambia el tercer nombre a "Carlos"
puts "Nombre del índice 2 cambiado = #{nombres[2]}"

# Obtener la cantidad de elementos en el arreglo
cantidad = nombres.length # Retorna 4
puts "Cantidad de elementos del arreglo = #{cantidad}"

# Funciones de arreglos
nombres = ["Juan", "María"]
nombres.push("Pedro")   # Agrega "Pedro" al final
nombres << "Luisa"    # Otra forma de agregar "Luisa" al final
nombres.pop           # Elimina el último elemento ("Luisa")
nombres.delete_at(0)   # Elimina el primer elemento ("Juan")
```

Arreglos:

COLECCIONES ORDENADAS DE OBJETOS, POR EJEMPLO, [1, 2, 3]

Booleanos y Nil

- Cuando una variable se establece en nil, significa que no contiene ningún valor válido. niles una forma de indicar que no hay datos disponibles o que una variable está vacía

```ruby
# Valores booleanos
es_verdadero = true
es_falso = false

# Imprimir los valores booleanos
puts "Es verdadero: #{es_verdadero}"
puts "Es falso: #{es_falso}"

# Valor nil (nulo)
valor_nulo = nil

# Imprimir el valor nil
puts "Valor nulo: #{valor_nulo}"
```

```ruby
valor = 0
puts valor

valor = false
puts valor

valor = true
puts valor

valor = nil
puts valor
```

Condicionales

```ruby
nota = 70

if nota >= 90
  puts "Aprobado con Honor"
elsif nota >= 70
  puts "Aprobado"
else
  puts "Reprobado"
end
```

```ruby
unless condición
  # Código a ejecutar si la condición es falsa
end
edad = 15

unless edad >= 18
  puts "Eres menor de edad"
else
  puts "Eres mayor de edad"
end
```

```ruby
condición ? valor_si_verdadero : valor_si_falso
edad = 20
mensaje = (edad >= 18) ? "Eres mayor de edad" : "Eres menor de edad"
```

```ruby
valor = 2

case valor
when 1
  puts "Hoy es Lunes"
when 2
  puts "Hoy es Martes"
when 3
  puts "Hoy es Miércoles"
when 4
  puts "Hoy es Jueves"
when 5
  puts "Hoy es Viernes"
else
  puts "Fin de semana"
end
```

Métodos

```ruby
# Variables globales
numero = 5
num = 30

# Método saludar
def saludar
  num = 10 # Variable local
  puts "Hola"
end

# Ejecutar el método saludar
print "Ejecutar el método saludar "
saludar

# Método con parámetros
def suma(a, b)
  result = a + b
  return result
end

# Ejecutar el método
num1 = 5
num2 = 3

result = suma(num1, num2)
puts "La suma de #{num1} y #{num2} es: #{result}"
```

Ciclos

```ruby
contador = 0

while contador < 5
  puts "El contador es #{contador}"
  contador = contador + 1 # También puede ser abreviado como contador += 1
end
```

```ruby
# Este código imprimirá los números del 1 al 5 utilizando un rango y un bucle
# Cada valor del rango se asigna a la variable numero en cada iteración.

(1..5).each do |numero|
  puts "Número: #{numero}"
end
```

```ruby
nombres = ["Mariel", "Claudio", "Jose"]

for nombre in nombres
  puts "Hola, #{nombre}!"
end
```

```ruby
# Solicita al usuario un número positivo y continúa solicitándolo hasta

loop do
  print "Ingresa un número positivo: "
  numero = gets.chomp.to_i

  if numero > 0
    puts "¡#{numero} es un número positivo!"
    break  # Sale del bucle
  else
    puts "Eso no es un número positivo. Intenta de nuevo."
  end
end
```

POO- Clase

Clase con atributos

```ruby
class Persona
  attr_reader :nombre, :edad

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

persona1 = Persona.new("Mariel", 23)
puts persona1.nombre  # Debería imprimir "Mariel"
puts persona1.edad    # Debería imprimir 23
```

Método

```ruby
class Persona
  attr_reader :nombre, :edad

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

  def presentarse
    puts "Hola, soy #{@nombre} y tengo #{@edad} años."
  end
end

persona1 = Persona.new("Mariel", 23)
persona1.presentarse  # Debería imprimir "Hola, soy Mariel y tengo 23 años.
```

Herencia

```ruby
class Estudiante < Persona
  def trabajar
    puts "Soy un estudiante y estoy aprendiendo."
  end
end

estudiante1 = Estudiante.new("Claudio", 30)
estudiante1.presentarse  # Llama al método 'presentarse' de la clase 'Person
estudiante1.trabajar     # Llama al método 'trabajar' de la clase 'Estudiante
```

Polimorfismo

```ruby
# Función que saluda a una persona
def saludar(persona)
  persona.presentarse
end

persona2 = Persona.new("Jose", 30)
estudiante2 = Estudiante.new("Gabi", 35)

saludar(persona2)    # Llama a 'saludar' con un objeto de la clase
saludar(estudiante2) # Llama a 'saludar' con un objeto de la clase
```

Hash

- Es una estructura de datos que almacena pares clave-valor. Cada valor se asocia con una clave única, lo que permite un acceso rápido y eficiente a los valores. Los hashes son similares a los diccionarios en otros lenguajes de programación

```ruby
# Crear un hash vacío
mi_hash = {}

# Agregar elementos al hash
mi_hash["nombre"] = "Juan"
mi_hash["edad"] = 30
mi_hash["ciudad"] = "Madrid"

# Acceder a los valores mediante las claves
puts "Nombre: " + mi_hash["nombre"]
puts "Edad: " + mi_hash["edad"].to_s  # Es importante convertir la edad
puts "Ciudad: " + mi_hash["ciudad"]

# Modificar un valor existente
mi_hash["edad"] = 31

# Eliminar un par clave-valor
mi_hash.delete("ciudad")

# Verificar si una clave existe en el hash
if mi_hash.key?("nombre")
  puts "La clave 'nombre' existe en el hash."
end

# Iterar a través de las claves y valores del hash
mi_hash.each do |clave, valor|
  puts "Clave: #{clave}, Valor: #{valor}"
end
```

Ejercicio

```ruby
# Crear un array para almacenar las tareas pendientes
tareas_pendientes = []

# Bucle para el menú de la aplicación
loop do
  puts "¡Gestión de Tareas Pendientes!"
  puts "1. Agregar tarea"
  puts "2. Ver tareas pendientes"
  puts "3. Salir"
  print "Selecciona una opción: "
  opcion = gets.chomp.to_i

  case opcion
  when 1
    print "Ingrese una nueva tarea: "
    tarea = gets.chomp
    tareas_pendientes << tarea
    puts "Tarea agregada con éxito."
  when 2
    puts "Tareas pendientes:"
    tareas_pendientes.each_with_index { |tarea, index| puts "#{index + 1}. #{tarea}" }
  when 3
    puts "Saliendo de la aplicación. ¡Hasta luego!"
    break
  else
    puts "Opción no válida. Por favor, elige una opción válida."
  end
end
```

Ejercicio Práctica

- Crear un programa en Ruby que permita a los usuarios llevar un registro de tareas pendientes. Pasos a seguir:

1. Definir las tareas y variables: Comenzaremos definiendo las variables que necesitaremos para almacenar las tareas pendientes.
2. Mostrar el menú: Crearemos un menú que permitirá a los usuarios seleccionar opciones como agregar tarea, ver lista de tareas, etc.
3. Agregar tareas: Los usuarios podrán agregar tareas a la lista. Utilizaremos un bucle para solicitar la entrada del usuario y agregar tareas a la lista.
4. Marcar tareas como completadas: Permitiremos a los usuarios marcar tareas como completadas. Esto implicará el uso de estructuras condicionales para actualizar el estado de la tarea.
5. Ver lista de tareas: Mostraremos la lista de tareas pendientes en el formato deseado.
6. Salir del programa: Los usuarios podrán salir del programa cuando lo deseen.

Definir las tareas y variables: Comenzaremos definiendo las variables que necesitaremos para almacenar las tareas pendientes.

```
# Inicializamos una lista vacía para almacenar las tareas.
tareas = []
```

Mostrar el menú: Crearemos un menú que permitirá a los usuarios seleccionar opciones como agregar tarea, marcar tarea como completada, ver lista de tareas, etc

```
# Inicializamos una lista vacía para almacenar las tareas.
tareas = []

# Definimos un método para mostrar el menú.
def mostrar_menu
  puts "\n--- Lista de Tareas ---"
  puts "1. Agregar tarea"
  puts "2. Marcar tarea como completada"
  puts "3. Ver lista de tareas"
  puts "4. Salir"
  print "Elige una opción: "
end
```

Agregar tareas: Los usuarios podrán agregar tareas a la lista. Utilizaremos un bucle para solicitar la entrada del usuario y agregar tareas a la lista.

```ruby
# Definimos un método para agregar una tarea a la lista.
def agregar_tarea(lista)
  print "Ingresa la nueva tarea: "
  tarea = gets.chomp
  lista << { tarea: tarea, completada: false }
  puts "Tarea agregada: #{tarea}"
end
```

Marcar tareas como completadas: Permitiremos a los usuarios marcar tareas como completadas. Esto implicará el uso de estructuras condicionales para actualizar el estado de la tarea.

```ruby
# Definimos un método para marcar una tarea como completada.
def marcar_completada(lista)
  puts "Lista de tareas pendientes:"
  lista.each_with_index do |tarea, index|
    puts "#{index + 1}. #{tarea[:tarea]} (Completada: #{tarea[:completada] ?
  end
  print "Ingrese el número de tarea a marcar como completada: "
  tarea_idx = gets.chomp.to_i - 1
  if tarea_idx >= 0 && tarea_idx < lista.length
    lista[tarea_idx][:completada] = true
    puts "Tarea marcada como completada: #{lista[tarea_idx][:tarea]}"
  else
    puts "Índice de tarea inválido."
  end
end
```

```ruby
# Ciclo principal del programa.
loop do
  mostrar_menu
  opcion = gets.chomp.to_i

  case opcion
  when 1
    agregar_tarea(tareas)
  when 2
    marcar_completada(tareas)
  when 3
    puts "\n--- Lista de Tareas Pendientes ---"
    tareas.each_with_index do |tarea, index|
      puts "#{index + 1}. #{tarea[:tarea]} (Completada: #{tarea[:completada] ? 'Sí' : 'No'})"
    end
  when 4
    puts "¡Hasta luego!"
    break
  else
    puts "Opción inválida. Por favor, elige una opción válida."
  end
end
```

Ejercicio

- **Creación de una Calculadora en Ruby**

1. Inicia un nuevo proyecto de Ruby o un archivo Ruby en blanco.
2. Defina las variables que necesitarán para almacenar los números de entrada y el operador (+, -, *, /).
3. Pide al usuario que ingrese dos números y un operador. Almacene estos valores en las variables.
4. Utilice estructuras de control para realizar la operación adecuada según el operador ingresado.
5. Imprima el resultado de la operación en la pantalla.
6. Incluir funciones o métodos que realicen las operaciones matemáticas (suma, resta, multiplicación, división).
7. Llama a los métodos correspondientes según el operador ingresado por el usuario.
8. Implemente un bucle para permitir que el usuario realice varias operaciones sin necesidad de reiniciar el programa cada vez.

Cuestionario

Gracias por su atención
y colaboración