Introduction:

- Scripting refers to the process of writing scripts, which are sequences of instructions of a computer program.
- Scripting languages are programming languages that are typically designed to be easy to write and execute.
- Python, Ruby, JavaScript, Perl, PHP, etc are some of the scripting languages.
- Scripting Languages are often used for building automation applications due to their ease of use, rapid development capabilities, high-level abstractions, platform independence, integration with system tools, flexibility, and strong community support.
- Not all but most of the scripting languages are interpreted programming languages.

Ruby:

- Ruby is a dynamic, multithreaded, interpreted and object-oriented programming language.
- It is known for its simplicity and productivity.

Features of Ruby:

1. Dynamic Typing: Memory allocations to the variables of a Ruby program are done on runtime, which can also change on the runtime itself.
2. Interpreted: Ruby programs are executed line by line, without any pre-compilation step. (No Compiler)
3. Loosely Typed: Variables in ruby don't have a fixed type, and their types can change at runtime.
4. Multithreaded: Ruby supports multithreading, which is the process of separating program instructions and executing them parallelly.
5. Object-oriented: Every variable in Ruby is an object. Every object is an instance of a class. Ruby supports all OOP features such as inheritance, Polymorphism and Encapsulation.
6. Garbage Collection: Ruby has automatic memory management through garbage collection, which helps developers avoid memory leaks and manage memory efficiently.
7. Extensibility: Ruby has a large number and variety of libraries(known as Gems) for extended functionalities. A package management tool known as RubyGems is used to install/remove/manage the packages and libraries.

Rails / Ruby on Rails:

- Rails, also known as Ruby on Rails, is a popular open-source server-side web application framework written in the Ruby programming language.
- Rails follows the model-view-controller (MVC) architectural pattern, which separates the application's data, user interface, and control logic into distinct components.
- Ruby is widely used for server-side web development (backend), where it runs on the server and handles tasks such as processing HTTP requests, interacting with databases, and generating dynamic content.
- Ruby can also be used on client-side (frontend) upto some extent, with some additional tools and modifications.
- Features of Rails:

    1. Don't Repeat Yourself (DRY) principle: Rails encourages developers to write reusable code and avoid duplicating logic. This leads to cleaner, more maintainable codebases.
    2. Active Record: Rails includes an Object-Relational Mapping (ORM) library called Active Record, which simplifies database interactions by abstracting database tables into Ruby objects. This allows developers to perform database operations using familiar Ruby syntax.
    3. Scaffolding: Rails provides scaffolding generators that automatically generate boilerplate code for common tasks such as creating models, controllers, and views. This helps developers get started quickly and reduces the amount of repetitive coding required.
    4. Rich ecosystem: Rails has a vast ecosystem of plugins (gems) and libraries that extend its functionality, allowing developers to add features such as authentication, authorization, and caching to their applications with ease.

The Structure and Execution of Ruby Programs:

1. **Source Code**: Ruby programs are written in plain text files with a .rb extension. These files contain the source code written in the Ruby programming language.
2. **Comments**: Comments in Ruby start with the # character and are used to provide explanations or annotations within the code. Comments are ignored by the Ruby interpreter during execution.
3. **Statements and Expressions**: Ruby programs consist of statements and expressions. Statements are instructions that perform actions, such as variable assignments or method calls. Expressions produce a value and can be composed of literals, variables, operators, and method calls.
4. **Variables and Constants**: Ruby uses variables to store data values. Variables are declared using an assignment operator (=) and can hold different types of data. Constants are similar to variables but their values cannot be changed once defined.
5. **Data Types**: Ruby has several built-in data types, including integers, floats, strings, symbols, arrays, hashes, booleans, and nil. Each data type has its own set of operations and methods for manipulation.
6. **Control Structures**: Ruby provides control structures such as conditionals (if, elsif, else), loops (while, until, for), and iterators (each, map, select) for controlling the flow of program execution.
7. **Methods**: Methods are reusable blocks of code that perform specific tasks. They are defined using the def keyword and can accept parameters (arguments) and return values using the return keyword.
8. **Classes and Objects**: Ruby is an object-oriented programming language, so programs are typically organized around classes and objects. Classes define the blueprint for creating objects, which encapsulate data and behavior.
9. **Modules and Mixins**: Modules are collections of methods and constants that can be included in classes to provide additional functionality. Mixins allow classes to inherit behavior from multiple modules.
10. **Exception Handling**: Ruby supports exception handling using begin, rescue, and ensure blocks to handle errors and exceptions gracefully.

```ruby
# Define a method to print a welcome message
def welcome(name)
  puts "Welcome, #{name}!"
end

# Define a method to calculate the sum of two numbers
def calculate_sum(a, b)
  return a + b
end

# Main program starts here
puts "=== Welcome to My Ruby Program ==="

# Prompt the user for their name
print "Enter your name: "
user_name = gets.chomp

# Call the welcome method to greet the user
welcome(user_name)

# Get two numbers from the user
print "Enter the first number: "
num1 = gets.chomp.to_i

print "Enter the second number: "
num2 = gets.chomp.to_i

# Call the calculate_sum method to calculate the sum
sum = calculate_sum(num1, num2)

# Print the result
puts "The sum of #{num1} and #{num2} is: #{sum}"

puts "=== End of Program ==="
```

- You can run this Ruby program using a Ruby interpreter, such as by saving it to a file with a .rb extension and executing it from the command line:

  ruby my_program.rb

Package Management with RubyGems:

- A Gem is nothing but a packaged ruby program/script.
- Gems (ruby packages) are created and used for code reusability.
- RubyGems is the package manager for the Ruby programming language.
- It allows developers to easily install, manage, and share Ruby libraries/packages (gems) with others.
- So, if you have created a gem, you can use it in other programs as well by importing. But if you want others to use it as well, then you would publish that gem. By this, others can install and use your gem remotely, in their ruby scripts.
- Gems are often referred to as dependencies (or) packages (or) libraries.
- Here are some RubyGems related CLI commands:

1. Installation of RubyGems: RubyGems is typically included with Ruby installations by default. However, if you need to install it separately or update it, you can use the following command:

```
gem update --system
```

2. Installation of Gems: To install a gem, you can use the gem install command followed by the gem name.

```
gem install rails
```

3. Listing Installed Gems: You can list the gems installed on your system using the gem list command.

```
gem list
```

4. Updating Gems: You can update installed gems to the latest version using the gem update command followed by the gem name.

```
gem update rails
```

5. Uninstalling Gems: To uninstall a gem, you can use the gem uninstall command followed by the gem name.

```
gem uninstall rails
```

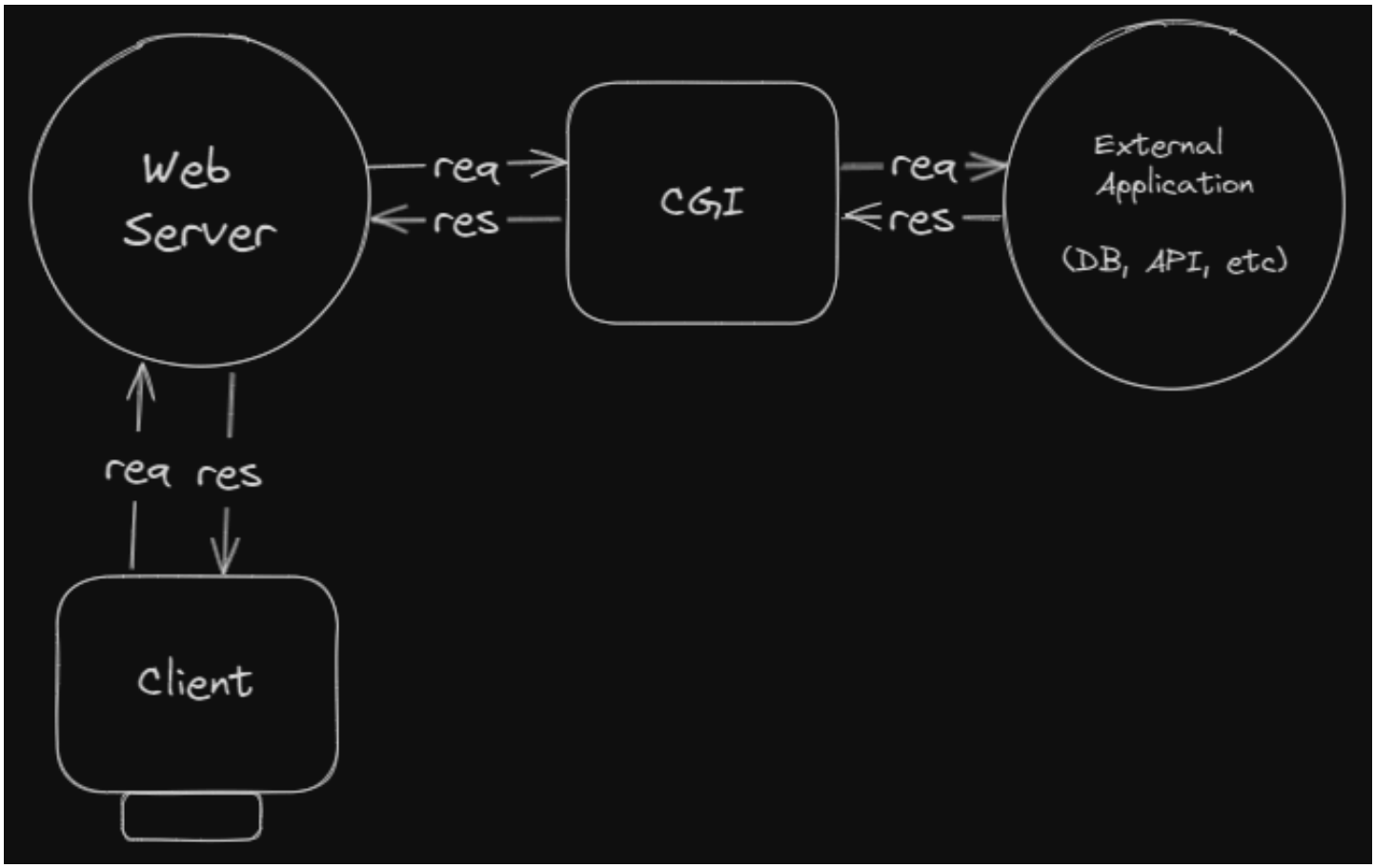
- **Creating and Publishing Your Own Gems:**

=>Ruby and Web:

- Ruby is a versatile programming language. That is, it can be used for writing logical scripts and as well as for web development.
- There are several web frameworks that are built on top of ruby such as Sinatra, Rails, Cuba, Volt, etc.
- Rails is the popular framework used for web development, which follows MVC Architecture.

Writing CGI Scripts:

- CGI (Common Gateway Interface) is a standard protocol (low-level logic) for interfacing external applications with web servers. [external applications => database, APIs, etc]
- It defines a set of rules and conventions for communication between a web server and external programs, allowing web servers to execute programs or scripts in response to HTTP requests.
- That is, CGI defines a way for communicating with external applications from a web server.



- Here's how CGI works:
1. Client Request: A client (such as a web browser) sends an HTTP request to the web server, typically for a resource.
2. Web Server Handling: When the web server receives the request, it checks whether the requested resource is associated with a CGI script or program.
3. CGI Invocation: If the requested resource is a CGI script or program, the web server invokes it as a separate process.
4. Script Execution: The CGI script or program executes in the server's environment and generates an HTTP response, typically as HTML content.
5. Response Transmission: The web server collects the response from the CGI script or program and sends it back to the client as the HTTP response.

- CGI Scripts are nothing but the scripts written in a high-level scripting language.
- A CGI Script inputs the requests from a web server, which will be requested back to an external application by the CGI. The output from the CGI to a Web Server is data received from an external application.
- The output data from a CGI script can be in any form such as JSON, XML, Plain text, etc
- The CGI logic is usually provided by any web framework as an inbuilt feature. Hence we don't need to write the CGI scripts explicitly. If there is a requirement of writing them on our own, then CGI Scripts can be written in two ways:

1. **Without CGI Class (Raw method):**
   - We can write a CGI Script directly in ruby, as follows:

```ruby
#!/usr/bin/env ruby

puts "Content-type: text/html"
puts ""

puts "<!DOCTYPE html>"
puts "<html>"
puts "<head>"
puts "<title>Simple CGI Script in Ruby</title>"
puts "</head>"
puts "<body>"
puts "<h1>Hello, CGI World!</h1>"
puts "<p>This is a simple CGI script written in Ruby.</p>"
puts "</body>"
puts "</html>"
```

- Save this script with a .rb extension (e.g., simple_cgi.rb) and make sure it has executable permissions. Place it in a directory accessible by your web server with CGI enabled.
- The above ruby code (CGI script) generates an XML file as the response to a request received from a web server.
- This looks simple, although not a preferred way for writing CGI scripts, because it gets complicated for large applications. (such as, when we have to include headers, query parameters, etc)

2. **With CGI Class:**
   - Ruby has a standard library (gem) which provides classes and methods for writing CGI Scripts, known as “cgi”.
   - CGI class (an instance of CGI library) is the one which provides all the methods and classes for accessing request parameters, managing cookies, generating HTML, managing Environment Variables, managing headers, etc.

```ruby
#!/usr/bin/env ruby

require 'cgi'

cgi = CGI.new

puts cgi.out('text/html') {
  cgi.html {
    cgi.head {
      cgi.title { "Simple CGI Script in Ruby" }
    } +
    cgi.body {
      cgi.h1 { "Hello, CGI World!" } +
      cgi.p { "This is a simple CGI script written in Ruby." }
    }
  }
}
```

- The above ruby code (CGI Script) generates the below html content:

```html
<!DOCTYPE html>
<html>
<head>
<title>Simple CGI Script in Ruby</title>
</head>
<body>
<h1>Hello, CGI World!</h1>
<p>This is a simple CGI script written in Ruby.</p>
</body>
</html>
```

Cookies:

- Cookies are small pieces of data (in key-value pairs) that are sent by a website to a user's web browser and stored on the user's device, in the persistent storage space.
- They are used for session management, storing user preferences, personalizing user experience, tracking user behavior, collecting analytics data, etc.
- They are created and manipulated on a web server, and stored on the persistent storage of a user's browser.
- In Ruby, the CGI class (from CGI gem) provides a method called "cookies" for cookies management.
- so, cgi.cookies returns a CookieJar, which is an object containing all cookies sent by the client's browser.

```ruby
require 'cgi'

cgi = CGI.new

# Retrieve a specific cookie by name
cookie_value = cgi.cookies['cookie_name']&.value

# Set a new cookie
cgi.cookies['new_cookie'] = 'new_cookie_value'

# Delete a cookie
cgi.cookies.delete('cookie_to_delete')

# Output the cookies
cgi.cookies.each { |cookie|
  puts "Cookie name: #{cookie.name}, value: #{cookie.value}"
}
```

Choice of WebServers:

- The choice of web server for Ruby applications depends on various factors such as performance, scalability, ease of use, and specific requirements of the application. Here are some popular options for Ruby web servers:

1. Apache: Apache HTTP Server is a widely used open-source web server known for its stability, flexibility, and extensive feature set. It's capable of serving static and dynamic content, supporting various programming languages and modules through its robust plugin architecture. Apache is commonly used in combination with application servers like Phusion Passenger or mod_proxy for deploying Ruby applications.
2. WEBrick: WEBrick is a lightweight and easy-to-use web server that comes bundled with Ruby. It's suitable for development and testing purposes due to its simplicity and convenience. However, WEBrick may not be as performant or scalable as other web servers for production use.
3. Thin: Thin is a fast and lightweight web server for Ruby applications. It's based on EventMachine and supports concurrency through non-blocking I/O, making it efficient at handling concurrent requests. Thin is well-suited for deploying Ruby web applications with moderate traffic loads.
4. Puma: Puma is a high-performance web server for Ruby applications. It's multithreaded and can handle concurrent requests efficiently, making it suitable for deploying large-scale applications with high traffic volumes. Puma is the default web server used by Ruby on Rails in production.
5. Nginx: Nginx is a powerful and lightweight web server and reverse proxy known for its high performance, scalability, and efficient resource utilization. While Nginx itself is not specifically designed for serving Ruby applications, it's commonly used as a reverse proxy in front of application servers like Puma or Unicorn to improve performance and handle static file serving, caching, and load balancing.
6. Unicorn: Unicorn is a Unix-based HTTP server for Ruby applications. It's designed for stability and performance, using a pre-forking worker model to handle concurrent requests. Unicorn is commonly used for deploying Ruby on Rails applications in production, often in combination with Nginx as a reverse proxy.

Ex:

```ruby
require 'webrick'

content = <<~HTML
  <!DOCTYPE html>
  <html>
  <head>
    <title>WEBrick Example</title>
  </head>
  <body>
    <h1>Hello, WEBrick!</h1>
    <p>This is a simple example of using WEBrick to create a web server in Ruby.
  </body>
  </html>
HTML

server = WEBrick::HTTPServer.new(Port: 8000)

server.mount_proc '/' { |req, res|
  res.content_type = 'text/html'
  res.body = content
}

trap('INT') { server.shutdown }

server.start
```

SOAP and WebServices:

- Any two remotely communicating programs (known as web services) have a specific format of transmitting data/messages between them.
- The data/message format can be XML, JSON, etc.
- SOAP, REST, GraphQL, Microservices, Serverless, etc are the architectures used for building web services.
- Hence, SOAP (Simple Object Access Protocol) is a protocol (or) web-service building architecture, for exchanging structured information in the form of XML.
- It relies on XML as its message format and operates over standard internet protocols such as HTTP and SMTP.
- SOAP allows programs running on different operating systems, platforms, and programming languages to communicate with each other over the internet.
- Hence, Features of SOAP architecture are:
    - Human readable format (XML)
    - Platform, OS, Programming Language independent
    - Transport Protocol Independent (can be used over HTTP, SMTP and TCP)
- Web-services built with SOAP architecture are secure and reliable.

RubyTk:

- RubyTk is a Ruby Interface to the Tk(Toolkit) graphical user interface library used to create GUI for desktop applications.
- Tk is a cross-platform GUI toolkit that provides a set of widgets and tools for building desktop specific graphical user interface applications, in a platform independent manner.
- Features of RubyTk:

1. **Cross-Platform**: Desktop applications created with RubyTk are platform independent. That is they can run on Windows, MacOs, and Linux.
2. **Simple and Lightweight**: RubyTk provides a simple and lightweight way to create graphical user interfaces in Ruby, making it suitable for rapid prototyping and simple applications.
3. **Variety of Widgets**: RubyTk provides a variety of built-in widgets for building GUI applications, such as TkButton, TkLabel, TkText, TkCanvas, TkMenu, etc.
4. **Event-Driven Programming**: RubyTk follows an event-driven programming model, where actions (events) such as button clicks, mouse movements, and keyboard inputs trigger corresponding event handlers (callbacks) that execute predefined actions in response.
5. **Layout Management**: RubyTk provides various layout managers for arranging widgets within the window, including pack, grid, and place managers. These managers allow you to control the positioning and resizing of widgets in a flexible and dynamic manner.
6. **Customization**: RubyTk allows you to customize the appearance and behavior of widgets by setting various properties such as size, color, font, and event bindings.
7. **Integration with Ruby**: Since RubyTk is implemented in Ruby, it seamlessly integrates with the Ruby language and ecosystem. You can use Ruby's object-oriented features, libraries, and modules to enhance and extend your GUI applications built with RubyTk.

Simple Tk Application:

```ruby
require 'tk'

# Create a new main window
root = TkRoot.new { title "Hello, World!" }

# Create a text widget
TkLabel.new(root){
   text 'Hello, World!'
   pack { padx 15 ; pady 15; side 'left' }
}

# Start the Tk event loop
Tk.mainloop
```

Widgets:

- Widgets are graphical elements or controls that you can use to build the user interface of your desktop application.

- These widgets represent various interactive components such as buttons, labels, text entry fields, menus, and more.
- Widgets are the building blocks of GUI applications, and you can arrange them within the application window to create a user-friendly interface.
- RubyTk provides rich-set and variety of widgets in the form of classes, such as:
    1. **TkFrame**: Creates and manipulates frame widgets.
    2. **TkButton**: A button widget that users can click to trigger an action.
    3. **TkLabel**: A widget for displaying text or images.
    4. **TkEntry**: A single-line text entry field where users can input text.
    5. **TkText**: A multiline text widget for displaying and editing text.
    6. **TkCanvas**: A widget for drawing graphics, shapes, and images.
    7. **TkListbox**: A widget for displaying a list of items that users can select from.
    8. **TkMenu**: A widget for creating menus and dropdown lists.
    9. **TkScale**: A widget for selecting a value from a range using a slider.
    10. **TkCheckButton**: Creates and manipulates checkbutton widgets.
    11. **TkRadioButton**: Creates and manipulates radio-button widgets.
    12. **TkListbox**: Creates and manipulates list-box widgets.
- Each widget has its own set of properties(options), methods, and events that you can use to customize its appearance and behavior.
- Widget options in RubyTk are attributes that you can use to customize the appearance and behavior of widgets.
- Widget options allow you to control various aspects of the widget's appearance, such as its size, color, font, position, and behavior.
- Some common widget options that are used with most of the Tk widgets are:
    1. **Text Options**:
        - text: Specifies the text displayed on the widget.
        - font: Specifies the font used for displaying text.
        - foreground (or fg): Specifies the color of the text.
        - background (or bg): Specifies the background color of the widget.
    2. **Size and Position Options**:
        - width: Specifies the width of the widget.
        - height: Specifies the height of the widget.
        - padx: Specifies the padding (in pixels) to be added horizontally around the widget's contents.
        - pady: Specifies the padding (in pixels) to be added vertically around the widget's contents.
    3. **Layout Options**:
        - anchor: Specifies the alignment of the widget within its container.
        - side: Specifies the side of the container where the widget should be placed (e.g., 'left', 'right', 'top', 'bottom').
    4. **Behavior Options**:
        - command: Specifies the action to be performed when the widget is interacted with (e.g., clicked, selected).
        - state: Specifies the state of the widget ('normal', 'disabled').
    5. **Event Binding Options**:
        - bind: Specifies event bindings for handling user interactions (e.g., mouse clicks, key presses).
        - unbind: Removes event bindings.
    6. **Appearance Options**:
        - relief: Specifies the appearance of the widget's border ('flat', 'raised', 'sunken', 'groove', 'ridge').
        - borderwidth (or bd): Specifies the width of the widget's border.

Ex:

```ruby
require 'tk'

# Create a new main window
root = TkRoot.new { title "Button Example" }

# Create a button widget with options
button = TkButton.new(root){
  text "Click me!"
  padx 10
  pady 5
  background "blue"
  foreground "white"
  pack { padx 15; pady 10 }
}

# Define the action to be performed when the button is clicked
button.command{
  button.background("red")
  button.foreground("black")
  button.text("Clicked!")
}

# Start the Tk event loop
Tk.mainloop
```

- RubyTk also provides layout managers to arrange widgets within the window or container in a specific layout configuration.
- Layout managers help control the positioning and sizing of widgets, allowing you to create organized and visually appealing user interfaces.
- There are three geometry managers (place, grid and pack) that are responsible for controlling the size and location of each of the widgets in the interface.

1. **Pack Layout Manager**: It arranges widgets in the order they are packed into the container, stacking them either horizontally or vertically.
   Example: widget.pack { side 'left'; padx 10; pady 5 }
2. **Grid Layout Manager**: It arranges widgets in a grid-like structure of rows and columns.
   Example: widget.grid(row: 0, column: 0, padx: 10, pady: 5)
3. **Place Layout Manager**: It allows you to specify the exact coordinates (x, y) for each widget within the container.
   Example: widget.place(x: 100, y: 50)

Building Events:

- An action performed by a user on the GUI components, is known as an Event.
  (ex: mouse click, keypress, mouse hover, mouse pointer movement, etc)

- Event binding, also known as event handling or event listening, is the process of associating an event on a GUI component, with an event handler.
- Event Binding allows the creation of rich, intuitive, and interactive user experiences in desktop applications.
- When the specified event occurs, the respective event handler is triggered (executed).
- As we know that RubyTk is an Event-driven Toolkit. That is, it supports Event handling functionality on the Tk widgets.
- Event binding typically involves the following steps:
    1. **Selecting the Event:** Choose the specific event or events that you want to respond to, such as button clicks, key presses, mouse movements, etc.
    2. **Defining the Event Handler:** Write a function or method that specifies the actions to be performed when the event occurs. This function or method is known as the event handler or callback.
    3. **Binding the Event:** Associate the event with the event handler by using a binding mechanism provided by the GUI framework or library. This informs the application that when the specified event occurs, it should execute the associated event handler.
    4. **Executing the Event Handler:** When the event occurs, the application invokes the event handler, which then executes the predefined actions or logic.

```ruby
require 'tk'

# Create a new main window
root = TkRoot.new { title "Event Binding Example" }

# Create a button widget
button = TkButton.new(root) {
  text "Click me!"
  padx 10
  pady 5
  pack { padx 20; pady 20 }
}

# Define the action to be performed when the button is clicked
button.bind("ButtonPress-1"){
  Tk.messageBox(message: "Button clicked!")
}

# Start the Tk event loop
Tk.mainloop
```

Canvas:

- A canvas is a drawing surface or container that allows you to display and manipulate graphical objects such as shapes, lines, text, and images dynamically.
- The canvas provides a blank area where you can draw or paint custom graphics, diagrams, charts, and interactive visualizations.
- RubyTk provides the canvas in the form of a widget known as TkCanvas.

- Characteristics of a canvas widget are:
    1. **Drawing Surface:** A canvas provides a blank drawing surface where you can draw or render graphical objects. You can specify the size and dimensions of the canvas, and it typically occupies a rectangular area within the application window.
    2. **Graphics Primitives:** Canvas widgets support various graphics primitives such as points, lines, rectangles, circles, polygons, text, and images. You can use these primitives to create custom drawings and visualizations on the canvas.
    3. **Coordinate System:** Canvas widgets typically use a Cartesian coordinate system to specify the position and size of graphical objects on the canvas. You can specify coordinates (x, y) to indicate the location of objects relative to the canvas origin.
    4. **Interactivity:** Canvas widgets can be made interactive by binding events to graphical objects drawn on the canvas. For example, you can respond to mouse clicks, mouse movements, or keyboard events within the canvas area.
    5. **Manipulation:** Canvas widgets often provide methods for manipulating and transforming graphical objects dynamically. You can move, resize, rotate, or delete objects on the canvas programmatically.

```
require 'tk'

# Create a new main window
root = TkRoot.new { title "Canvas Example" }

# Create a canvas widget
canvas = TkCanvas.new(root) {
  width 300
  height 200
  background "white"
  pack { padx 20; pady 20 }
}

# Draw a rectangle on the canvas
rectangle = canvas.create_rectangle(50, 50, 200, 150, fill: 'blue', outline: 'black')

# Start the Tk event loop
Tk.mainloop
```

Scrolling: