Module 1

# Overview of R

## Course Introduction

This course has been designed by **Yan Lou, PhD**, a data scientist and developer at IBM, Canada. He has built innovative AI and cognitive applications in areas such as:

* Mining software repositories
* Personalized health management
* Wireless networks
* Digital banking

Yan Lou earned his PhD in machine learning from the **University of Western Ontario**.

## Why Choose R?

Many people wonder which programming language to learn for data science. **R is an excellent choice** for the following reasons:

### Open-Source Language

* R is **free and open-source**, allowing users to review and modify the source code.
* Enables integration of cutting-edge AI research into analytics.

### Rich Data Science Libraries

R offers powerful libraries for:

* **Data Processing**
* **Data Visualization**
* **Modeling**
* **Statistical Computing**

### Simple and Easy to Learn

R has a **simple syntax**, making it easy to understand and learn quickly.

### Strong Community Support

R benefits from a **vibrant and supportive community**, ensuring help is readily available.

## Popularity of R

R has been a widely used language since **2014**. According to the **TIOBE index**:

* R ranked **8th** in **2020**.
* As of **July 2023**, R remains among the top choices for data professionals worldwide.

# Introduction to the R Programming Language

## Purpose of the R Language

R is an interpreted programming language designed for statistical computing and data analysis. It is widely used in various fields, including data science, statistical modeling, and machine learning.

## Features of R

### Interpreted Language

* R is an interpreted language, meaning it does not require compilation before execution.
* Other interpreted languages include Python and JavaScript.
* Accessed through a command-line interpreter.

### History of R

* Developed at Bell Laboratories.
* Initially released in 1994.

## Statistical Computation with R

### Descriptive Statistics

* R is used for statistical computation, including:
  + Mean
  + Variance
  + Median
  + Quantiles
* Example: If you have a variable numtested representing the total number of patients tested, R can quickly return its median and maximum values.

### Data Sampling

* Used to obtain a representative data subset.

### Correlation Analysis

* R can compute and visualize a correlation matrix to analyze relationships between different variables.

### Hypothesis Testing

* Used to test if the mean values of two groups are statistically different.

### Inferential Statistics

* R helps infer properties of unknown distributions, such as estimating a population’s mean from sample data.

## Data Visualization in R

* R offers various visualization packages for:
  + Creating interactive and visual analytics.
  + Building dashboards for end-users.
  + Visualizing data on interactive maps, such as analyzing zip-related data.

## Predictive Analysis with R

### Machine Learning and AI

* R is a powerful tool for predictive analytics.
* Used for tasks such as:
  + Predicting pandemic trends.
  + Understanding human languages in various forms, such as documents and speeches.
  + Sentiment analysis of user reviews to determine positive or negative sentiments.
  + Visual object recognition, such as identifying dog breeds.

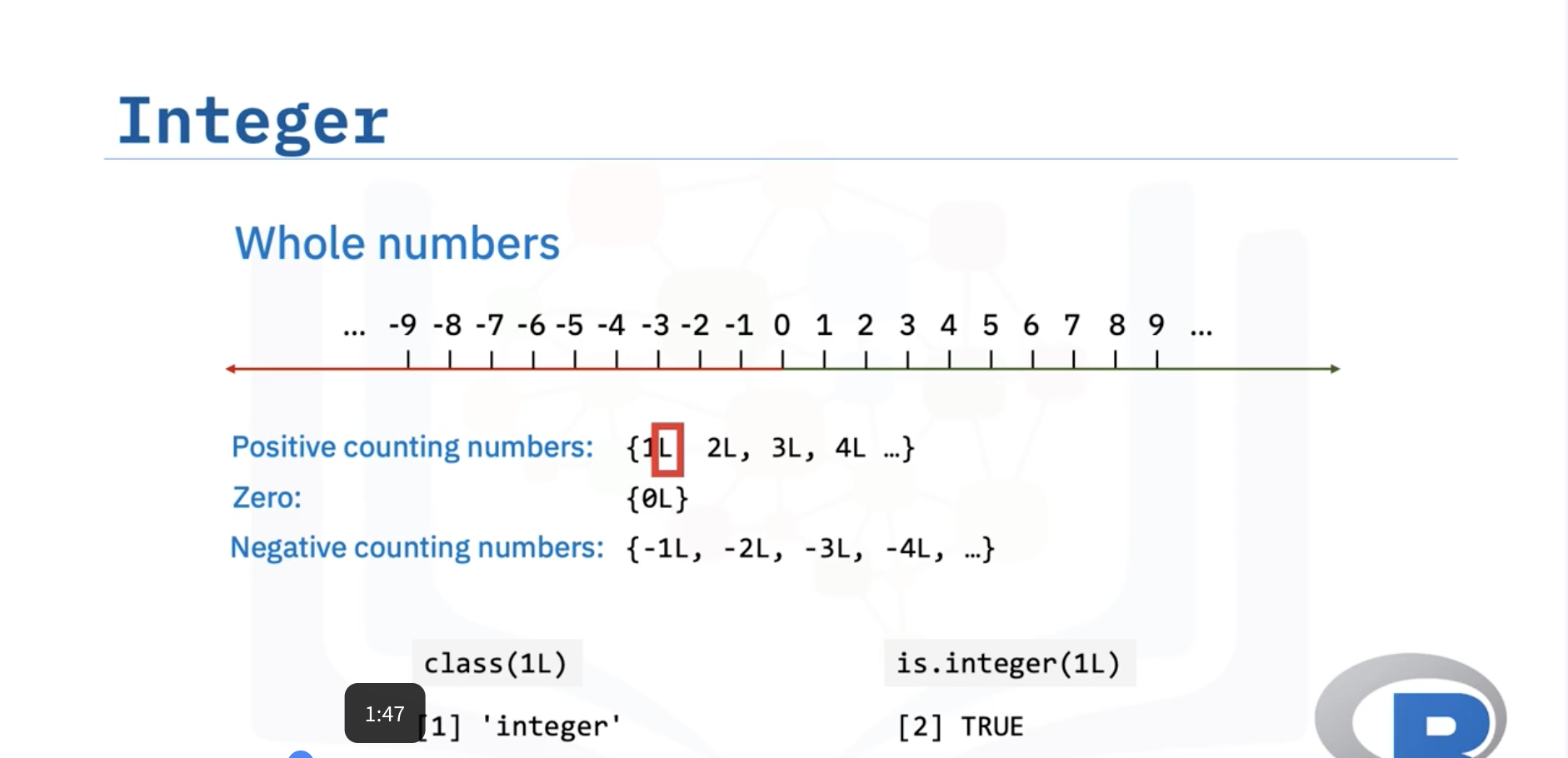
## Summary

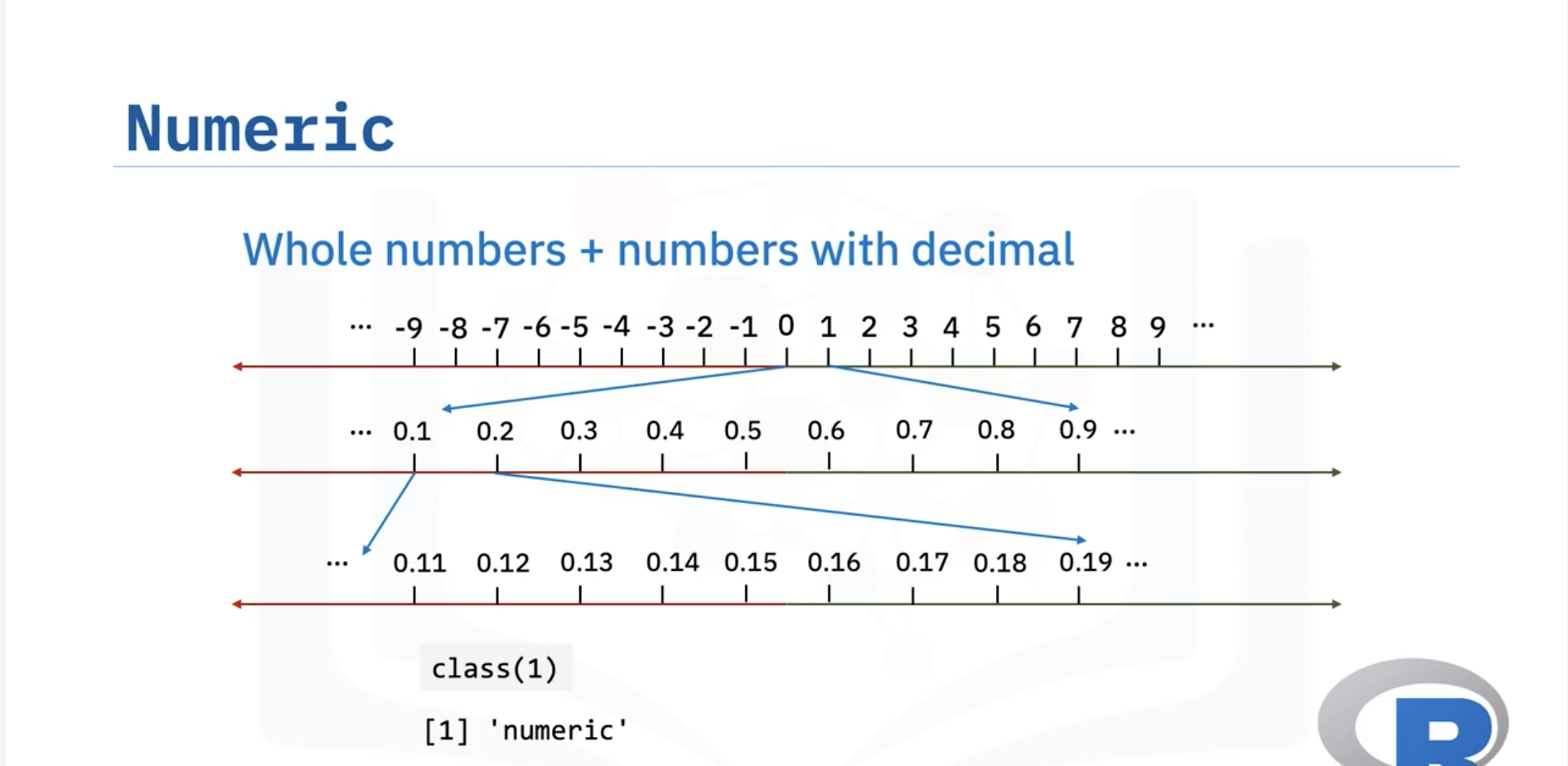
* R is an interpreted language designed for statistical computing.
* It is used for descriptive statistics, correlation analysis, hypothesis testing, and inferential statistics.
* R is widely used for data visualization and predictive analytics.
* It is a valuable tool in machine learning and AI applications.

# R Basic Data Types

* **Data Types**: R has several basic data types, including:
  + **Integer**: Whole numbers, denoted with an uppercase L (e.g., 1L).
  + **Numeric**: Includes integers and decimal numbers (e.g., 1.5).
  + **Character**: Textual data, defined with single or double quotes (e.g., "Hello").
  + **Logical**: Represents TRUE or FALSE values, used in logical operations.
* **Data Type Checking**: You can use the class() function to determine the data type and is.integer(), is.numeric(), is.character(), and is.logical() functions to check specific types.
* **Data Type Conversion**: R allows conversion between data types using functions like:
  + as.numeric()
  + as.integer()
  + as.character()
  + as.logical()
* **Advanced Data Types**: There are also complex and raw data types, though they are less commonly used.

Understanding these data types and their conversions is fundamental for processing data effectively in R.





# Math, Variables, and Strings

* **Addition, Subtraction, Multiplication, Division, and Exponentiation**: You saw how to perform these operations using numbers, such as calculating the total time of two movies and converting minutes to hours.
* **Variable Assignment**: You learned how to store results in variables (e.g., using x for total minutes and y for hours) and how to reassign values to variables.
* **Using Meaningful Variable Names**: It’s important to use descriptive names for variables to enhance code readability.
* **Order of Operations**: You were introduced to the concept of operator precedence and the use of parentheses to ensure correct calculations.
* **Creating Strings**: You learned how to create strings in R by enclosing characters in quotes.

This foundational knowledge is essential for manipulating data in R as you progress in your data science journey.

# The R Environment

## Understanding the R Environment

The R environment consists of key components such as the R Console, R script files, and workspaces. Additionally, it supports development tools like RStudio and Jupyter Notebook.

## R Console

When starting with R programming, the first interaction typically happens in the **R Console**:

* It is a shell interface where users can write and execute R code interactively.
* Results of executed commands appear immediately.

## R Script Files

As R programming tasks become more complex, organizing code in script files becomes necessary:

* R scripts have a **.R extension**.
* These files contain multiple lines of code that can be executed in **batch mode**.
* Regardless of where the code is written (Console or Script files), the **R interpreter** processes it into objects in memory.

## Objects in R

R handles different types of objects:

* **Variables** – Store data.
* **Functions** – Code snippets designed to perform specific tasks.
* **Data** – Collections of values used in analysis.
* These objects are translated into instructions executable by the CPU.

## Workspaces in R

The **workspace** in R is an environment where all created objects are stored:

* Functions, variables, and data exist in memory as part of the workspace.
* Workspaces can be saved as **RData files** and reloaded later.
* The ls function lists all objects in the current workspace.

## Working Directory in R

R uses a **current working directory** to manage files:

* The getwd function retrieves the current working directory.
* If necessary, the setwd function changes the working directory.
* To save a workspace, use the save.image("myworkspace.RData") function.
* To reload a saved workspace, use the load("myworkspace.RData") function.

## R Development Tools

To enhance R programming, two popular tools are commonly used:

### RStudio

* The most popular **Integrated Development Environment (IDE)** for R.
* Offers features such as:
  + Syntax highlighting
  + Command history
  + Workspace management
  + Direct code execution

### Jupyter Notebook

* A **web-based interactive code editor**.
* Useful for:
  + Writing and executing code
  + Managing data and experimental results
  + Creating educational content and prototypes

## Summary

* The R Console allows interactive execution of R code.
* R script files help organize and execute code in batch mode.
* Workspaces store all objects and can be saved or reloaded using RData files.
* The working directory manages file locations.
* RStudio and Jupyter Notebook are essential tools for R development and experimentation.
* Future lessons will provide more hands-on experience with these tools.

# Introduction to RStudio

## Advantages of Using RStudio

RStudio provides several benefits for R development, making it an efficient and user-friendly Integrated Development Environment (IDE).

### Facilitating Code Writing

RStudio enhances the coding experience through two key features:

#### Syntax Highlighting

* The RStudio File Editor color codes different R keywords.
* Increases readability and helps identify target variables easily.

#### Code Auto Completion

* RStudio suggests options while typing, including function names and documentation.
* Saves keystrokes and improves code syntax accuracy.

## R Working Environment

The workspace is the core concept of the R environment, allowing easy management of objects like data, variables, and functions.

### Visualizing Objects

* Use the ls function to list all objects in the workspace.
* RStudio provides a visual representation of these objects for easy access and management.
* Users can clean unnecessary objects to free up memory.

### File Management

* R projects often contain different file types, such as scripts, data, or plots.
* RStudio includes an intuitive File Explorer for managing files.
* Users can check project structures and open files directly.
  + Example: Double-clicking addresses.csv opens the file in the File Editor.

## Main User Interface Components

RStudio integrates multiple panels to streamline R development:

### File Editor Panel

* Used for writing R code and text files.
* Features syntax highlighting and auto completion for efficient coding.

### Console Panel

* Allows users to execute R commands and see results instantly.

### Workspace Panel

* Displays created objects, including data, variables, and functions.

### File Explorer Panel

* Helps manage files and other assets such as plots and packages.

## Summary

* RStudio enhances R coding through syntax highlighting and auto code completion.
* The workspace provides a visual interface for managing objects and memory.
* The user interface consists of essential components like the File Editor, Console, Workspace, and File Explorer for efficient workflow management.

# Writing and Running Code in Jupyter Notebook

## Introduction to Jupyter Notebook

Jupyter Notebook is a powerful tool for writing, running, and interacting with code. It consists of a series of **cells** where users can enter different types of text, including:

* Programming code
* Markdown files
* Raw text (which can later be converted into other formats)

## Executing Code in Jupyter Notebook

### Cell Execution

* Lines of code, when executed, produce **cell output**.
* Example: Executing print("Hello Jupyter") displays Hello Jupyter in the output.

### Execution Order and Variable Access

* Cells can be executed in a specific sequence.
* Objects and outputs from previous cells can be accessed in later cells.
* Example: Defining x = 1 in **Cell 1** allows access to x in **Cell 3**.

## Jupyter Notebook Kernel

* Each Jupyter Notebook requires a **kernel**, which maintains an interactive session.
* The kernel executes code and returns results.
* Jupyter supports multiple kernels, including:
  + **R** (used in this course)
  + **Python**
  + **Julia**

## Benefits of Jupyter Notebook

### All-in-One Coding Environment

Jupyter Notebook integrates essential elements needed for coding tasks and experiments, including:

* **Narration** – Provides context or instructions.
* **Code** – Implements logical tasks.
* **Data** – Used for processing tasks.
* **Visual Insights** – Includes plots, images, or videos.

### Interactive Code and Data Exploration

* Users can explore data dynamically using different cells.
* Cells allow updates and modifications while referencing outputs from previous cells.
* Example: Loading COVID-19 data from a CSV file, displaying column names, and calculating max confirmed cases (numconf).

### Exporting Jupyter Notebooks

* Jupyter Notebooks can be easily converted into formats like:
  + **PDF**
  + **HTML**
  + **LaTeX**
* This makes it convenient for presentations and sharing experiment results.

## Summary

* Jupyter Notebook is made up of cells containing code, markdown, or raw text.
* It integrates narration, code, data, and visualizations in one environment.
* The kernel maintains an interactive session to execute code.
* Jupyter Notebook enhances interactivity, making it useful for teaching, learning, and data exploration.

# Summary & Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

* You can use the R programming language to perform statistical computation, data visualization, and predictive analysis.
* The four most common data types in R are integer, numeric, character, and logical.
* You can use the class() function and the is.integer(), is.numeric(), is.character(), and is.logical() functions to determine the data type.
* You can convert some data types to other data types using the as.integer(), as.numeric(), as.character(), and as.logical() functions.
* R provides math operators that you can use to perform calculations on your data.
* Using variables in your calculations and providing them with descriptive names can help shorten your code and make it easier to read.
* You can control the order of operations using parenthesis.
* The R development environment includes the R console, R script files, and workspaces.
* Two important tools for working with R code are RStudio and Jupyter.
* RStudio features, like syntax highlighting and auto code completion, make writing code easier.
* The main components of RStudio include the File Editor, Console, Workspace, and File, Plots, and Packages Explorer.
* A Jupyter Notebook is made up of cells that can contain code, markdown files, or raw text.
* An all-in-one Jupyter Notebook contains narration, code, data, and plots, images, or videos.

Module 2

# Vectors and Factors

* **Vectors**:
  + A vector is a one-dimensional array that can store various types of data.
  + You can create a vector using the c() command, for example, c(1, 2) or c("Toy Story", "Akira").
  + Vectors can hold numerical data, character strings, and logical values (True/False).
  + You can create sequences using the colon operator, e.g., 1:10 for numbers from 1 to 10.
* **Factors**:
  + Factors are categorical variables that can take on a limited number of values.
  + They can be nominal (no order) or ordinal (with order).
  + You can convert a vector into a factor using the factor() function, specifying the order if needed.
* **Summary Function**:
  + The summary() function provides information about the structure of vectors and counts occurrences in factors.

# Working with Vectors in R

## Introduction

In this video, we will demonstrate several useful operations that can be applied to vectors in the R programming language.

## Naming Vector Elements

Suppose we have a vector containing the production years of four movies. To make it easier to remember which movies these years correspond to, we can use the names function to map a title to each year. Once mapped, we can call our year vector while passing in a title in the brackets, and the output will display the corresponding year.

## Finding the Number of Elements

To determine the number of elements in a vector, use the length function and pass the vector as input. This will return the number of elements in the vector.

## Sorting a Vector

To sort a vector in ascending order, use the sort function while passing the vector as input. This arranges the elements from the smallest to the largest. If the names function is used, the titles will appear alongside the years in the output.

## Finding Minimum and Maximum Values

* Use the min function to find the smallest number in the vector.
* Use the max function to find the largest number in the vector.

For example, if the years in a vector range from 1985 to 2010, min(years) will return 1985, and max(years) will return 2010.

## Computing the Average of a Vector

To calculate the average value of a vector:

* Use the sum function to sum all elements and divide by the number of elements.
* Alternatively, use the mean function for a simpler approach.

Both methods will yield the same result.

## Descriptive Statistics with summary

The summary function provides descriptive statistics about a vector, including:

* Minimum and maximum values
* Median
* Mean
* 1st and 3rd quartiles

## Accessing Vector Elements

To retrieve an element from a vector, write the vector's name followed by the desired index in square brackets:

* vector[2] retrieves the second element.
* vector[c(2,3)] retrieves the second and third elements.
* vector[1:3] retrieves a range of elements from index 1 to 3.
* Using a negative index (e.g., vector[-1]) removes that particular element.
* Accessing an index beyond the length of the vector results in an NA (missing value).

## Logical Operations on Vectors

Logical operations can be applied to each element in a vector. For example, applying a condition on a cost vector will return TRUE or FALSE for each element depending on whether the condition holds.

* To retrieve only the TRUE elements, write the condition inside square brackets: vector[vector > 10].

## Handling Missing Values (NA)

Missing values (NA) frequently appear in real-world datasets for various reasons. For example, if a vector represents movie age restrictions, NA values may indicate unknown age restrictions.

## Element-wise Arithmetic Operations

All arithmetic operations in R are performed element-wise:

* Multiplying two vectors results in element-wise multiplication.
* Multiplying a vector by a number scales each element accordingly.

## Conversion of Vector into factor

In **R Language**, converting a **vector** into a **factor** is important because **factors** help handle categorical data efficiently. Here’s why it's needed:

### ****1. Efficient Storage & Representation****

* Factors store **unique categorical values** as **levels** rather than repeating values, reducing memory usage.
* Example: Instead of storing "Male", "Female", "Male", "Female", it stores them as {1: "Male", 2: "Female"} with integer mapping.

### ****2. Better Handling of Categorical Data****

* Factors are useful in statistical modeling and data analysis where categorical variables must be treated differently from numeric data.
* Example: In regression models, R automatically treats **factors** as categorical variables instead of numerical ones.

### ****3. Useful for Data Visualization & Analysis****

* Many **R functions** (like ggplot2 and table()) expect categorical variables as **factors** for correct grouping and ordering.

### ****4. Enables Ordered Categories (Ordinal Data)****

* Factors allow for **ordered categories** (e.g., "Low" < "Medium" < "High").

## Conclusion

By now, you should have a good understanding of how to access elements of a vector and apply useful operations in R.

# Lists

* **Lists in R**: A list is a collection of objects that can contain different data types (e.g., strings, numbers, vectors).
* **Creating a List**: Use the list() function to create a list by specifying the elements.
* **Accessing Elements**: Access elements using their index in square brackets (e.g., list[2] for the second element) or by name using the dollar sign (e.g., list$name).
* **Modifying Lists**: You can add new elements, modify existing ones, or remove elements by assigning NULL to them.

# Arrays & Matrices

## Arrays:

* Structures that hold data of the same type (e.g., strings, integers).
* Can be multi-dimensional (multiple rows and columns).
* Created using the c command to form a vector, followed by the array function to define dimensions.
* Access elements using their index (row and column).

## Matrices:

* Similar to arrays but strictly two-dimensional.
* Created using the matrix function, specifying the vector and the number of rows and columns.
* By default, matrices are filled by columns, but this can be changed with the byrow = TRUE parameter.
* Subsets of matrices can be accessed by specifying row and column ranges.

# Data Frame

* A **data frame** is a structure that holds correlated information, such as movie titles and their corresponding years.
* Use the data.frame function to create a data frame, with each argument representing a column as a vector.
* Access data using the **dollar sign** symbol or by specifying the column number in square brackets.
* Retrieve individual elements by specifying the row and column numbers in square brackets.
* Use the str function to get information about the data frame's structure.
* The head and tail functions display the first and last six elements of the data frame, respectively.
* To add a new column, specify the column name in square brackets and assign a vector of values.
* Use the rbind function to add a new row.
* Delete rows using negative indexing and assign the result back to the data frame.
* Remove a column by assigning a NULL value to it.

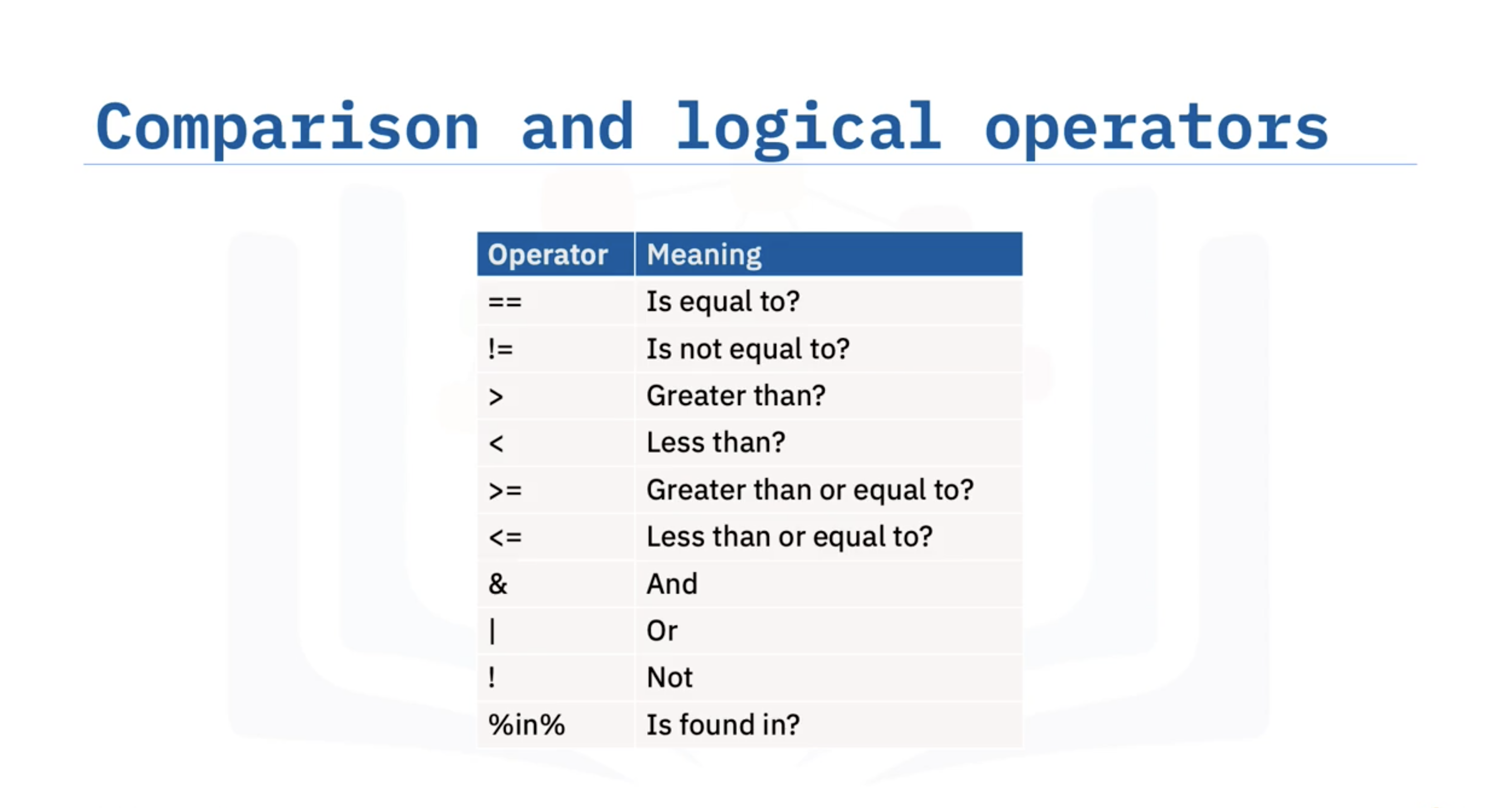
# Summary & Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

* A vector is a string of numbers, characters, or logical data.
* Factors (also known as categorical variables) are variables that take on a limited number of different values that can be nominal or ordinal.
* You can use R to perform operations on a vector, such as sorting the items, finding the smallest or largest number, or performing arithmetic on its values.
* Lists can store different types of data, unlike vectors, which can only store data of a single type.
* An array is a single or multidimensional structure containing data of the same type (strings, characters, or integers)
* A matrix is like an array but must be two-dimensional and can be arranged by columns or rows.
* The main difference between a data frame and other data structures, like a list, is that each variable has a vector of elements of the same type.

Module 3

# Conditional statements & Loops



* **Conditional Statements**:
  + Use if statements to evaluate conditions (e.g., checking if a movie's release year is greater than 2000).
  + An else block can be added for alternative actions when the if condition is false.
  + Logical operators (greater than, less than, equals) are used for comparisons.
* **For Loops**:
  + Iterate through elements in a vector.
  + Can be combined with if else statements to control output based on conditions.
* **While Loops**:
  + Execute as long as a specified condition remains true.
  + Useful for situations where the number of iterations is unknown.

# Functions in R

* **Definition of Functions**: Functions are blocks of code that can be reused in different parts of a program.
* **Types of Functions**:
  + **Pre-defined Functions**: Built-in functions in R or provided by packages (e.g., mean, sort).
  + **User-defined Functions**: Functions created by the user (e.g., printHelloWorld, add).
* **Function Arguments**: Functions can take arguments, and the return statement is used to output values.
* **Conditional Logic**: Functions can include conditional statements (e.g., if-else).
* **Default Values**: You can set default values for function arguments.
* **Nesting Functions**: Functions can be used within other functions.
* **Variable Scope**: Variables can be defined as global or local, affecting their accessibility outside the function.

# Strings

* **Reading Text**: A text file is read into a variable called "summary," which holds three lines of text.
* **Character Count**: The nchar function counts the number of characters in a string.
* **Case Conversion**: The toupper and tolower functions convert strings to upper and lower case, respectively.
* **Character Replacement**: The chartr function replaces specific characters in a string.
* **String Splitting**: The strsplit function breaks a string into a list based on a specified delimiter, and unlist converts it into a character vector.
* **Sorting**: The sort function can be used to order elements of a character vector alphabetically.
* **Concatenation**: The paste function combines elements of a character vector into a single string.
* **Substring Extraction**: The substr function retrieves specific portions of a string, while trimws removes leading and trailing whitespace.
* **Negative Indexing**: The str\_sub function from the stringr library allows for substring extraction by counting back from the end of the string.

# Regular Expression

* **Regular Expressions**: Used to match patterns in strings and text, particularly useful for data analysis.
* **Email Structure**: An email can be represented as a set of characters followed by an "at sign" and another set of characters.
* **Special Characters**:
  + **Period (.)**: Acts as a wildcard, matching any character.
  + **Plus Sign (+)**: Matches one or more occurrences of the preceding element.
  + **Asterisk (\*)**: Matches zero or more occurrences of the preceding element.
  + **Backslash (\)**: Used to escape special characters like the period.
* **Functions in R**:
  + **grep**: Finds matches based on a regular expression.
  + **gsub**: Replaces matched strings with a specified replacement.
  + **regexpr**: Finds matching substrings in detail.
  + **regmatches**: Extracts the matched strings.

# Date Format

* **Date Representation**: R represents dates as the number of days since January 1, 1970, based on the **Gregorian calendar**.
* **Data Frame Structure**: You explored a data frame of Oscar-winning actors, where the "date of birth" was in a non-standard format (UNIX time).
* **Date Conversion**:
  + Use the as.POSIXct function to convert UNIX time to a full timestamp.
  + Use the as.Date function to convert timestamps or character strings into Date Class objects.
* **Date Formatting**: You can specify the date format (e.g., "Year Month Day") when converting.
* **Date Operations**: You can perform operations like subtracting dates, comparing them, and using functions like weekdays, months, and quarters to extract information.
* **Creating Date Sequences**: The seq function allows you to create sequences of dates.

# Dealing with Bugs in R Programming

## Understanding Errors in R

Errors in R occur when an operation is invalid. For example, attempting to add a character and a number will result in an error message indicating that one of the arguments is non-numeric.

### Impact of Errors

* Errors halt code execution immediately.
* In a loop, once an error is encountered, execution stops, and subsequent iterations do not run.

## Debugging in R

Debugging is the process of identifying and fixing programming bugs. In large codebases, locating errors can be challenging, making debugging an essential skill for R programmers.

## Handling Errors with tryCatch

If you anticipate an error, you can use a tryCatch statement to handle it gracefully and prevent script termination.

### How tryCatch Works

* Runs the code normally if no errors occur.
* If an error occurs, tryCatch executes custom error-handling code instead of displaying the default error message.
* Custom messages or return values can be specified within tryCatch.

### Example

* Consider an invalid addition operation inside tryCatch.
* Instead of stopping execution, tryCatch handles the error and prints a custom message.
* If no error occurs (e.g., 10 + 10), the code executes normally.

### tryCatch with Loops

* If an error occurs inside a loop, tryCatch handles it, but the loop does not resume execution after the error.
* This results in a single error message rather than repeated messages for each iteration.

## Handling Warnings in R

Warnings indicate potential issues but do not stop code execution.

### Example

* Passing a non-numeric value (e.g., "A") to as.integer() generates a warning.
* Warnings can be caught using tryCatch, similar to error handling.
* Custom print statements can be used to handle warnings effectively.

# Summary & Highlights

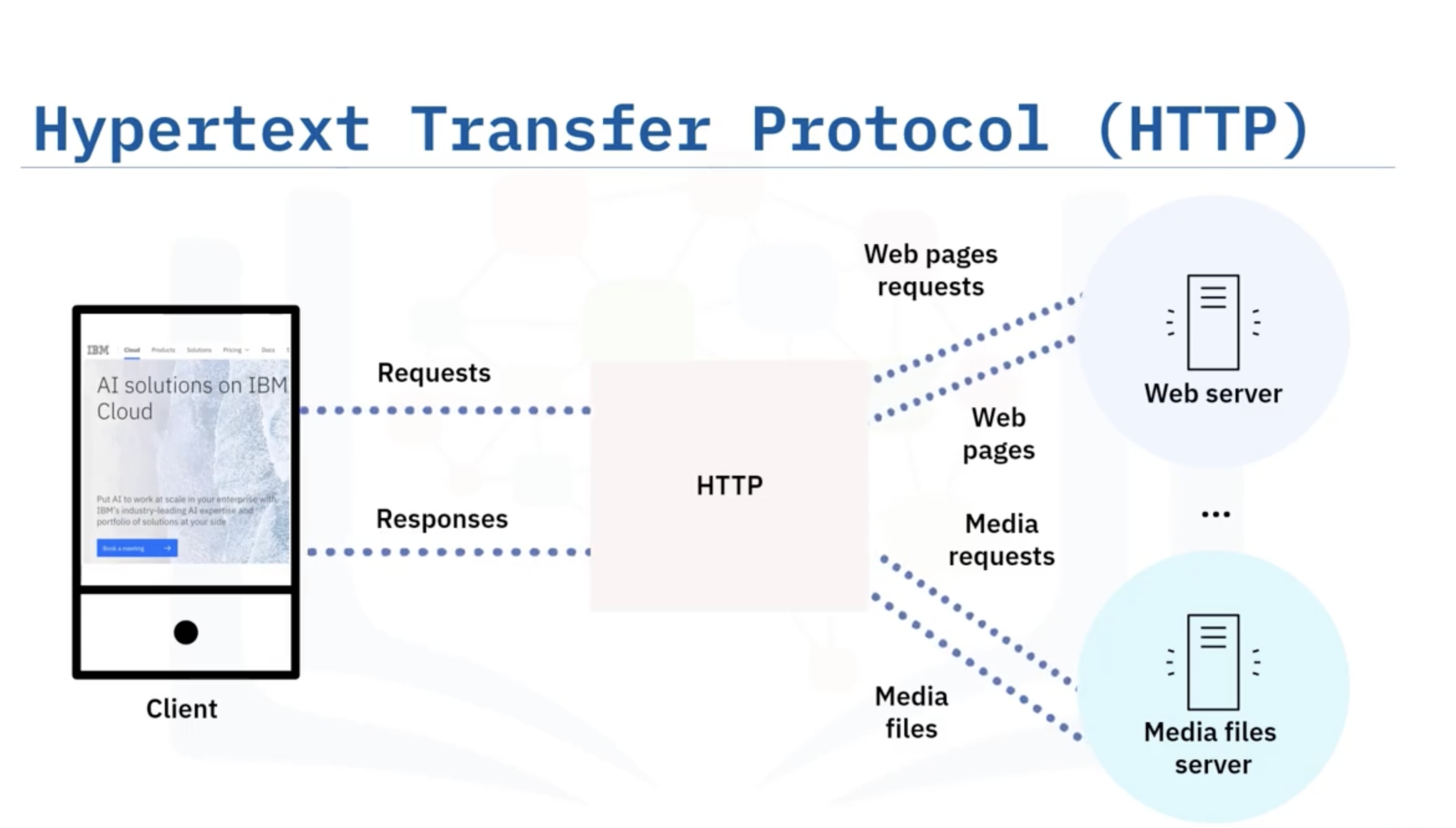
Congratulations! You have completed this lesson. At this point in the course, you know:

* If statements use comparison and logical operators to test conditions in code.
* For loops perform an operation for each item in a list, vector, or data frame column.
* While loops perform an operation until a condition is no longer true.
* Functions can be pre-defined or user-defined.
* In user-defined functions, you can control the return value of a function, add logic using if statements, and call other functions.
* You can define global variables using the <<- variable assignment operator.
* You can use R functions to manipulate the characters in a string, split a string into a vector, and retrieve specific substrings from within a string.
* Regular expressions are used to match patterns in strings and text.
* You can convert dates from one format to another using the as.POSIXct() and as.Date() functions.
* You can perform operations on Data objects using functions, like Sys.Date(), Sys.Time(), date(), and as.Date().
* You can intercept errors in R code and provide custom error and warning handling using tryCatch() statements.

Module 4

# HTTP Request & REST APIs

* **HTTP Protocol**: The primary communication method for fetching web resources from servers. It involves a workflow where a browser sends requests to servers, which respond with the necessary resources.
* **HTTP Requests and Responses**:
  + **HTTP Request**: Comprises methods (like GET, POST, PUT, DELETE), a URL, headers, and optionally a body.
  + **HTTP Response**: Includes a status code (e.g., 200 for success, 404 for not found), headers, and a body containing the resource content.
* **REST API**: A web service that follows REST architecture, which is resource-centric, stateless, and typically uses JSON or XML for data exchange. It simplifies the design of web applications.
* **Using httr Package in R**: This package allows you to perform HTTP requests easily. You can use functions like GET() for fetching data and POST() for sending data to a server.



# Summary & Highlights

Congratulations! You have completed this lesson. At this point in the course, you know:

* You can read a CSV file or an Excel file into a dataset using the read\_csv() and read\_excel() functions and then access its rows, columns, and individual data points.
* The readLines() function reads a text file into a character vector.
* The length() and nchar() functions return information about a character vector.
* The scan() function reads a text file into a character vector with each individual word as an element.
* The write()function exports a dataset as a text file.
* The write.csv() and write.table() functions export a dataset as a .csv file.
* The write.xlsx() function exports a dataset as an Excel file.
* The save() function saves R objects in .RData files.
* HTTP is a communication protocol for fetching web resources for clients from servers on the Internet and that each instance is comprised of a request and response.
* The REST API is a web service that uses the REST architecture to handle a request on a frontend web service.
* The httr package in R has functions that perform common HTTP and REST operations.
* The rvest package in R has functions that you can use to perform common web scraping tasks, such as reading HTML from a character variable, reading HTML from a URL, downloading a web page and reading it offline, extracting node data from a web page, and converting a table from a web page to a data frame.