Introduction to R: Part I

Caterina Luz Sanchez Steinhagen, David Rieger and Elizabeth Waterfield

09/11/2026

Prerequisites

> **!** Important
>
> Before completing this session, please carefully read about the prerequisites.

Prerequisite	Description	Where to find it
R and RStudio installed	Latest versions of both R and RStudio	Download RStudio
Accessibility sheet for computers	Fundamentals of computer literacy	Download

Presenter Notes: These are the **presenter notes**. You will find a script for the presenter for every slide. In presentation mode, your audience will not be able to see these presenter notes, they are only visible to the presenter.

Before starting this session, there are two important technical prerequisites.
First, you should have R installed, and second, you should have RStudio installed. These are the tools we will use to run and explore simulations during the tutorial.
The table on the slide shows the recommended versions and where you can download them if needed. If you already worked with R before, you likely already have these installed.
If not, you can install them using the links provided. If you run into issues, this is a good moment to ask for help before we move on to the tutorial."

Instructor Notes: **Instructor Notes**: There are also **instructor notes**. For some slides, there will be pedagogical tips, suggestions for activities and troubleshooting tips for issues your audience might run into. You can find these notes underneath the speaker notes.

Quickly verify whether learners have the required software installed before moving to the practical tutorial. If several students do not have the software yet, briefly explain that they can still follow conceptually and install it later. Avoid spending too much class time on troubleshooting installations unless the session is designed for setup.

Accessibility Tips: Where applicable, this is a space to add any tips you may have to facilitate the accessibility of your slides and activities.

Consider pairing students if someone does not have the software installed. Refer to the accessibility sheet in the download section.

Before we start: Survey time!

Question 1

What is your level of familiarity with R and RStudio?

a. I have never heard of it before

b. I have heard of it but have never worked with it

c. I have basic understanding and experience with it

d. I am very familiar and have worked with it extensively

Question 2

On a scale of 1–5, how intimidated do you feel by programming or coding (e.g., R/RStudio)? *(1 = Not intimidated at all, 5 = Very intimidated)*

a. 1

b. 2

c. 3

d. 4

e. 5

Question 3

Have you used any programming language (R, Python, JavaScript, C) before?

a. No, never

b. Yes, a little (e.g., basic commands)

c. Yes, moderately (e.g., small projects)

d. Yes, extensively (e.g., on an almost daily basis)

Presenter Notes:
Before we move on, we'll take a short survey. The aim here is to understand your current familiarity with specific elements of this session. In the next few slides, you'll see some questions that invite you to reflect on where you are starting from. This is not a test, and it is not about getting the right answers. If you are unsure, that is completely fine, just respond based on what feels closest to your current understanding.

Instructor Notes:
Set a low-stakes, supportive tone before beginning the survey. Emphasize that uncertainty is expected and valuable for measuring learning progress.

Discussion of survey results

What do we see in the results?

Presenter Notes: Let us have a look at these results.

Instructor Notes: The aim is to understand the group's current standing in terms of knowledge and familiarity. Emphasize explicitly that the survey is not about correct or incorrect answers. Avoid evaluating answers.

Learning goals

At the end of this session, you will be able to …

- **Navigate** the RStudio interface (Console, Environment, Script, and Output panes).
- **Differentiate** between basic data types in R (numeric, character, logical, integer) and how they are used.
- **Execute** operations in the Console and R Script using appropriate operators.
- **Assign** and **re-assign** values, vectors, and lists to objects.
- **Specify** arguments within functions to modify how the function produces its output.

- **Index** dataframes to access and retrieve specific information.

Presenter Notes: This slide outlines the learning goals for the session. At this stage, you are not expected to fully understand or master all of these points. Think of them as a roadmap rather than a checklist; they show where we're headed, not what you already need to know. You can use these goals to keep track of your learning, notice what feels clear, and identify areas where you might want to ask questions later.

Key terms and definitions

You might see some words for the first time or some familiar words used in a different way. What do you associate with the terms below?

- **RStudio**
- **R command**
- **Run**
- **Comment**
- **Operator**
- **Operation**

Presenter Notes: These are key terms we will revisit throughout the lesson. What comes to mind when you see each of them? Which ones are familiar, and which feel new? Take a moment to think about what they might mean.

Instructor Notes: Use a think-pair-share approach. First, have learners reflect individually on their associations with each term for a few minutes, then discuss in pairs for a few minutes, and finally share with the group. Allow about 10 minutes for this activity.

Key terms and definitions

- **RStudio**: A program that provides an easy-to-use interface for writing and running R code.
- **R command**: A line of code that tells R to do something.
- **Run**: To execute an R command so that R carries out the instruction and shows the result.
- **Comment**: Text in your code (starting with `#`) that R ignores. You can use this function to explain what the code does or make notes throughout your work.

- **Operator**: A symbol (such as `+`, `-`, `*`, `<`, or `==`) that tells R to perform a specific action.
- **Operation**: The action performed using one or more operators on data or values.

Presenter notes: Let's go through what these key terms mean, and remenber that this will become less and less abstract as we go:

- RStudio is the environment we'll be working in. This is where you write and execute your code. Think of it as your workspace.
- An R command is simply an instruction you give to R. It could be something as simple as a calculation or storing a value.
- When we say "run" code, we just mean executing that command so R actually performs the action and shows you the result.
- Comments are notes you write for yourself or others in your code. They start with a hashtag, and R ignores them.
- An operator is simply the symbol that tells R what action to perform, such as + or ==.
- An operation is the complete expression where that operator is applied, such as 5 + 3 or x > 10. In other words, the operator is the instruction, while the operation is the action being carried out."

What is R?

R (https://www.r-project.org/) is an **free and open source** programming language

- Originally designed by statisticians for statisticians
- Wide range of applications: **data manipulation, visualization, interactive web applications**, **generation of full manuscripts and reports**

Why R?

- Knowledge of statistical software like R is useful when **conducting quantitative research**
- Has a **large online support** community
- Provides freely available **resources**

Presenter Notes: Let's start with the basics: What exactly is R? R is a programming language specifically designed for data analysis and statistics. Originally, R was designed by statisticians for statisticians. That means its core purpose wasn't general software development, but rather making statistical analysis easier, more accessible, and more reproducible. Over time, however, R has evolved far beyond its original purpose. Today it supports a wide

range of applications. It is commonly used for data manipulation, where users can clean, reshape, and transform large datasets efficiently. It is also widely used for data visualization, producing everything from quick exploratory plots to high-quality, publication-ready graphics.

It is also free and open source, which means anyone can download and use it, and developers from all over the world contribute to improving it. This has resulted in a large online community, so if you ever run into a problem, chances are someone else has had the same issue and posted a solution online. And because of that community, there are also countless free learning resources: tutorials, blog posts, videos, books, and packages that extend R's functionality.

Instructor Notes:
Keep this introduction brief and motivational. The goal is not to teach R, but to position it as an accessible and useful tool for research. You may optionally point students to the official R website or a beginner friendly resource, but avoid going into technical detail.

The RStudio environment

Open **RStudio** and check if you can see everything that's on the slide.

(You might only see three panes on your screen)

Presenter Notes: RStudio is the main program we'll be using to interact with R throughout this course. It's designed to make working with R much more organised and easier to follow.
On this slide, you can see the four main panes of the RStudio interface: In the top left, you'll find the script editor, this is where you usually write and edit your code.
In the top right, there's the Environment pane, which shows the objects you've created, like datasets or variables, so you can keep track of what's currently loaded.
In the bottom left is the Console. This is where R actually runs your commands and shows the output.
And in the bottom right, you'll see the Output panel for things like plots, help files, and packages.
Now, take a moment to open RStudio on your device and check if you can identify these panes. You might only see three panes at first, that is completely normal.

Instructor Notes: Pause briefly and allow students to open RStudio. Walk around if possible and help students locate panes. Reassure them that layouts may differ slightly (e.g., missing script pane until a file is opened). Keep this activity short and focused on orientation, not functionality.

Accessibility Tips: Verbally describe each pane clearly and allow extra time for students to locate them. Offer assistance to students who may struggle with navigation or opening the software.

Mathematical operations

Symbol	Operation
`+`	Addition
`-`	Subtraction
`*`	Multiplication
`/`	Division
`^`	Exponentiation
`sqrt()`	Square Root
`log()`	(Natural) Logarithm

Presenter Notes: In the Console, which is the bottom left pane in RStudio, we can directly run commands.
One of the simplest things we can do is perform basic mathematical calculations. R works very much like a calculator, but with more flexibility.
To do this, we use operators, i.e., symbols that represent mathematical operations. For example, plus for addition, minus for subtraction, and so on.
On this slide, you can see some of the most common ones. These include multiplication, division, powers, as well as functions like square root and logarithm.

If you have RStudio open, you can try typing a simple calculation into the Console, like 2 plus 2, and see what happens.

Instructor Notes: Keep this very brief and hands-on. Encourage students to try 1–2 simple commands in the Console to build confidence. Avoid going into syntax details beyond basic operators.

Accessibility Tips: Say each operator aloud (e.g., "asterisk for multiplication") to support learners unfamiliar with symbols.

Practical exercise 1

- ☐ Open RStudio on your device

Using the table on the previous slide, run the following math operations directly in the **Console** of RStudio.

- ☐ `12+17`
- ☐ `218/3`
- ☐ `357^2`

Let's use the Console!

You can run code from the Console by typing in the Console and then pressing Enter on your keyboard.

Presenter Notes: We have now reached our first short hands-on exercise.
Using the Console in RStudio — that's the bottom left pane — try running the following mathematical operations. Simply type them in and press Enter.
Note that for division in R, we use the forward slash. So instead of writing it as a fraction, you would type 218 divided by 3 using the slash.
Take a minute to try these out.

Instructor Notes: Give students 1–2 minutes to complete the exercise. Clarify the correct syntax for division if needed (`218/3`). Optionally demonstrate one example live in the Console. Walk around to assist students who may be unfamiliar with typing commands. Solution for this exercise is found at the end.

Accessibility Tips: Read out each command clearly and explain symbols (e.g., "slash for division"). Allow students to follow along at their own pace and provide support for those who may need help with keyboard input.

The R script

An R script is a **text file** containing your **R commands**.

- It is an easy way to save and share code
- You can run code from an R script by:
  - Pressing **Ctrl + Enter** (Windows) OR **Command + Enter** on a (Mac) on the keyboard
  - click on **Run**
- You can also add **comments** or **notes** to your code by using **#**

```
# Here is an example of using a comment
# sum up two numbers
4 + 5
```

```
[1] 9
```

Presenter Notes: In the previous exercise we only worked in the Console. But there's one important limitation: anything you type there is not saved once you close RStudio.
That's why we use R scripts. An R script is simply a file where you can write, save, and organize your code so you can come back to it later or share it with others.
To run code from a script, you place your cursor on a line,or highlight multiple lines, and then press Ctrl + Enter on Windows or Command + Enter on a Mac. You can also click the 'Run' button.
Another useful feature is comments. If you want to add notes or explanations that R should not execute as code, you use the hash symbol. Everything after that on the line is treated as a comment.
This helps make your code more readable and easier to understand — both for others and for your future self.

Creating a new R script

Presenter Notes: Now let's look at how to create a new R script from scratch.
To do this, go to the File menu, then choose New File, and then select R Script. Alternatively, you can use the shortcut Ctrl + Shift + N.
Once the script opens, you can immediately start writing your code or notes.
It's important to save your script early and in an organised folder, so you don't lose your work. To save it, go to File and click Save, or use Ctrl + S.

Instructor Notes: Guide students step-by-step while they follow along. Ensure everyone successfully creates and saves a script before moving on. If needed, pause and repeat navigation steps slowly.

Saving an R script

Step 1: On your device

- Decide where this R script should go
- Create a *new folder*



Step 2: In RStudio

- After making a new R script, click on `File` → then click `Save As...`
- → Name the file → then choose the designated folder
- → Click `Save`

Presenter Notes: It is very important to be organised with how you save and store your work. We want to make sure we know where things are so we can retrieve it later, so here's a quick run-through of some good practices when saving our R Script. First, go to where you want to store this R Script on your device- it can be on your desktop, a folder you already have, or even one you made specifically for this session. Create a new folder there and let's name it "R Lesson". Back in RStudio, after creating a new R Script, we want to save it before we even start adding any code. We do this by clicking on "File" at the top-left corner. In the drop down menu, we click "Save As..." since this is the first time saving the document, give it a name such as "R Script 1", navigate to the folder we just created for this R Script, and then click "Save". Great! Now you know exactly where your R Script is. You can now use "Save" (as opposed to Save As...) to save all the changes you will make to the file.

Instructor Notes: It may seem like a basic computer skill, but do NOT assume that everyone knows folder management and how to properly store + save Files to their device. This is an important section and can assist learners in how they manage their files beyond this lesson as well.

Accessibility Tips: Keep in mind that actions like "creating a folder" and "renaming" are all, in fact, learned skills that not everyone may have as yet. Always verbalize or demonstrate the instructions for even these seemingly basic tasks. Or, allow learners to work together with others who perform these actions regularly and can help out.

Practical exercise 2

- ☐ Create a new R script in RStudio
- ☐ Copy the comment below and add it to your empty R script

```
# This is my practice R Script
```

- ☐ Save it to the folder on your device

Presenter Notes: If you haven't already, complete this practical exercise: create a new R script, write a comment in the first line using the hash symbol, and save it in a new folder dedicated to this course. If you already created and saved your R Script, add the comment and hit "Save".

Instructor Notes: Clear up that the instructions in this exercise say to "Save" the RScript but since this is the first time we are saving it to our device, we do so by clicking "Save As..." not just "Save". "Save" comes afterward- to save changes to the existing file already saved and stored somewhere on your device.

Logical comparisons

With **R**, you can also:

- Check if two numbers are **equal or unequal**
- Check if a number is **larger or smaller** than another number

```
# Example 1:
49 == 50
```

```
[1] FALSE
```

```
# Example 2:
49 < 50
```

```
[1] TRUE
```

- R returns a **logical value**:
  - `TRUE` or `FALSE`

Presenter Notes: R can do more than just calculations; it can also help us make decisions by comparing values. These are called logical comparisons. For example, we can ask R whether two numbers are equal, or whether one number is larger or smaller than another.
On the slide, you can see two simple examples: checking whether 49 is equal to 50, and whether 49 is less than 50.
When we run these commands, R doesn't return a number — instead, it gives us a logical value: either TRUE or FALSE.

So, TRUE means the statement is correct, and FALSE means it is not.
These kinds of comparisons are very important, because they form the basis for more complex operations later on, like filtering data or writing conditions in code.

Instructor Notes: Keep the explanation intuitive and avoid introducing too many operators at once. Focus on understanding TRUE/FALSE outputs. Optionally demonstrate one example live. You may briefly mention combining conditions (AND, OR, NOT) but do not go into detail yet.

Accessibility Tips: Read the code examples aloud and explain symbols clearly (e.g., "double equals means 'is equal to' ").

Logical comparisons - operators

Symbol	Operation
`==`	Equals
`!=`	Not Equal
`>` and `<`	Greater and Lesser Than
`>=` and `<=`	Greater Than or Equal / Less Than or Equal
`&`	logical AND
`|`	logical OR
`!`	logical NOT

Presenter Notes: Here you can see an overview of the most common logical operators in R. These symbols allow us to compare values — for example, checking whether two numbers are equal, not equal, greater than, or less than each other.
In addition to simple comparisons, we can also combine multiple conditions.
For example, using AND means that both conditions need to be true, OR means that at least one condition needs to be true, and NOT reverses a condition.
You don't need to memorise all of these right now — this is just to familiarise you with what's possible. You'll get more comfortable with them as you start using them in practice."

Instructor Notes: Present this as a reference slide rather than content to be learned immediately. Focus on recognition, not recall. Avoid demonstrating all operators; one simple example (e.g., `49 < 50 & 50 < 60`) is sufficient if needed.

Practical exercise 3

TRUE or FALSE?

- ☐ Express the following statements in words and evaluate whether they are `TRUE` or `FALSE`.

```
(36 + 5) > 40

(4 == 5) | (25 > 17)

(5 == (2 + 3)) & (7 >= 8)
```

- ☐ Check your answers by copying and running the statements in **your R Script**.

Presenter Notes: Let's try another short exercise.
Here, you'll see a few logical statements written in R. Your task is to do two things:
First, translate each statement into plain words — what is it actually asking?
And second, decide whether the statement is TRUE or FALSE.
Take a moment to think through them, and then you can check your answers by running them in your R Script. The results will be shown in the Console.
For example, the first one is asking whether 36 plus 5 is greater than 40.
Take about one to two minutes to go through all three, and then we'll quickly review them together.

Instructor Notes: Give students 1–2 minutes to attempt the exercise individually or in pairs. Encourage them to verbalise the statements in plain language before running them in R. Afterward, quickly confirm answers and clarify any confusion, especially around logical OR (`|`) and AND (`&`).

Data types

Data types define what **kind** of values you are working with

Presenter Notes: Data types are essentially categories of data. They tell R what kind of value something is and it affects what processes and operations you carry out. We have already seen one type: logical values, which are simply TRUE or FALSE.
Numbers in R can be of different kinds. Most commonly, you'll work with numeric values, including decimals. There are also integers, which are whole numbers without decimals.
Then we have text data, which are called characters. These always need to be written in quotation marks " ".
Another important type is factors. These are used for categorical data with predefined levels. The difference between factor and character is a character is just text, but a factor is text

that represents a category or group. For example, categories like hair color, such as brown, blonde, or black.

Instructor Notes: Keep the explanation conceptual. The goal is recognition, not mastery. Avoid going into technical distinctions unless learners ask. You may briefly point to examples in the image to anchor understanding.

Data types

- These determine **how R stores** and **processes** data
- Data types can show different behavior across different operations

```
Examples:

5        # number (integer)
3.14     # number (numeric)

"hello"  # character
factor("brown")  # factor

TRUE     # logical
```

Presenter notes: It is important to note that R treats each data type differently, therefore, this affects your operations. In the following examples, we see that 5 without quotation marks is a number. This means you can use it for mathematical operations and R will give you the appropriate result. So, storing data as the correct data type is important, because it determines what operations you can perform on what. Let's look at this example closer on the next slide.

Data types: Operations

Can R perform the same operation on different data types?

- Two numeric variables can be summed up:

```
4 + 5
```

```
[1] 9
```

- A numeric and a character variable *cannot* be summed up:

```
4 + "5"
```

```
Error in `4 + "5"`:
! non-numeric argument to binary operator
```

Presenter Notes: Let's look at how these different data types behave when we try to use them in operations.
If we take two numeric values, like 4 and 5, we can easily add them together, and R will return the result as expected.
However, if we try to mix data types — for example, adding number and character values — R won't know how to handle that.
In the second example, we're trying to add the number 4 and "5" wrapped in quotation marks. Since R now reads `5` as a character and not a number, R returns an error.
This shows why it's important to be aware of the data types you're working with — because not all operations are possible with all types.

Instructor Notes: Use this slide to reinforce the practical importance of data types. Optionally demonstrate the error live to show how R responds. Emphasize that error messages are normal and informative, not something to be avoided.

ADDITIONAL EXAMPLE: Another example would be if you use the greater than sign , >. You could run a line of code saying $5 > 2$, which would give you TRUE as a result. But, you can also use greater than with character data or text, e.g., "pineapple" > "banana". What do you think it would give us? It would give us TRUE, but not because a pineapple could be larger in size than a banana, but because R performs a lexicographic rather than a numeric comparison: It checks whether alphabetically pineapple comes after banana in the alphabet, which is TRUE.

Objects and assignment

Objects and assignment

Everything in R is an **object**, and oftentimes, we want to work with an object repeatedly.

- **Assign a value** to an object using the assignment arrow **<-**

```r
x <- 1
y <- 2

# The example assigns the value 1 to the object "x"
# and the value 2 to the object "y"
```

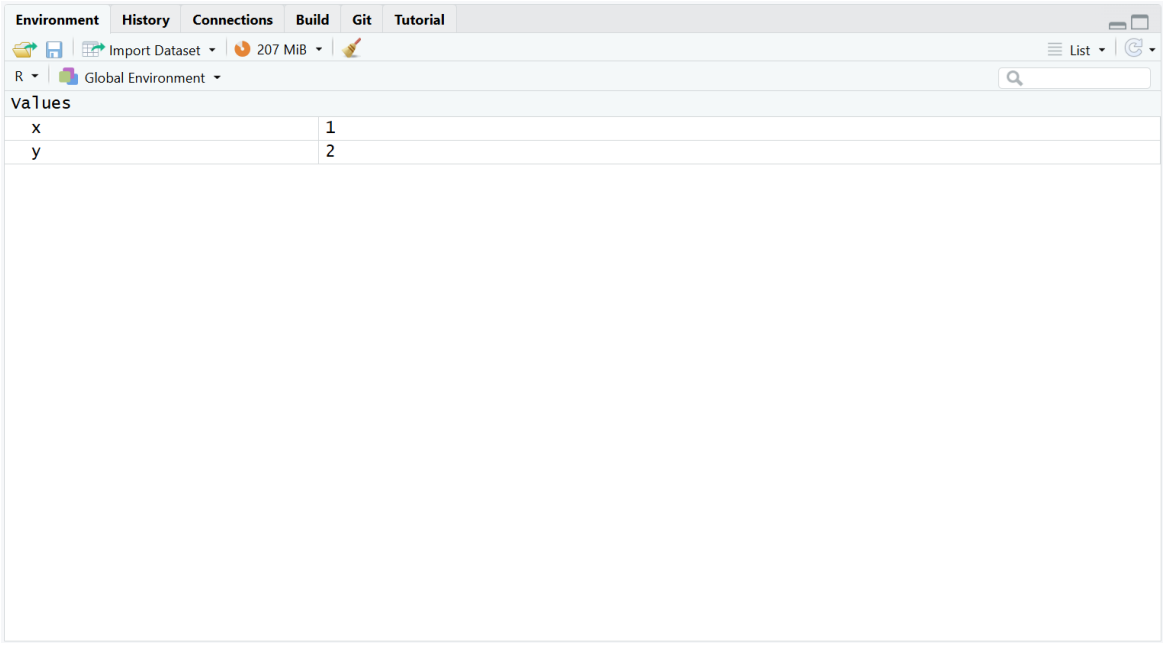
> ! The assignment arrow points to the object
>
> Take note of the **order**: we put the variable or object name first, then the assignment arrow `<-`, and lastly the number or value. The arrow should be pointing the object- it would not work if you put `1 <- x`

Presenter Notes: In R, almost everything we work with is treated as an object. That could be a number, a dataset, a result of a calculation, or even the outcome of a logical comparison. Often, we don't just want to calculate something once — we want to store it and use it again later. For that, we create objects and give them names.
We do this using the assignment arrow, written as `<-`.
For example, if we write `x <- 1`, we are telling R: store the value 1 inside an object called "x" and the value 2 inside "y".
Once we run this line, the object is saved in the Environment pane in RStudio, where we can see and reuse it anytime. Also, take note of the placement: the variable name goes in-front of the assignment arrow and the value or number goes behind. This is important for R to assign the value to the variable successfully.

Instructor Notes: Make sure students understand the idea of "storage" rather than just calculation. If possible, demonstrate live how the object appears in the Environment pane. Reinforce that naming objects clearly is important for readability and later use.

Objects and assignment

The object then appears in the **Environment** pane *(top right pane)*.



Presenter Notes: Here you can see an example of how an object appears in the Environment pane in RStudio.
Whenever you create an object using the assignment arrow, R stores it here. In this example, you can see the object name, along with its value and type.
This is very useful because it allows you to keep track of everything you have created during your session. Instead of recalculating something again, you can simply reuse the object from the Environment.
As you start working more with R, this pane will become your overview of what is currently stored in memory.

Instructor Notes: Point students' attention directly to the Environment pane on their own screens if available. Briefly explain how objects may change or accumulate as they run more code. Avoid going into memory management details at this stage.

Accessibility Tips: Verbally describe what is visible in the screenshot for students who cannot see it clearly. Allow extra time for learners to locate the Environment pane on their own interface.

Practical exercise 4

Try following the example and create your objects!

- ☐ Assign the value 4 to `x`.
- ☐ Assign the value 5 to `y`.
- ☐ Run the line(s) and check out your new objects in the Environment pane.

Presenter Notes: Let's assign some values to objects. Remember the placement matters: the variable goes in-front of the arrow and the value goes behind. Run this line to see how the object appears in your Environment- this also helps you keep track of all the objects you create.

Objects and assignments

Okay, I made objects... what now?

- Assigning objects helps organize your values
- Call on them and use specific objects in new operations

```
x <- 1
y <- 2

# In a previous slide, we assigned values to the objects

# Now, let's call on the objects and add them together
x + y
```

```
[1] 3
```

Presenter notes: Once you assign a value to an object, you don't have to keep remembering or retyping that value every time. You can just call the object name and use it in your calculations, which makes your code cleaner and much easier to work with. In the example, we see that we can call on and perform operations with x and y since we previously assigned values to those objects.

Practical exercise 5

Let's use the objects in a new operation:

- ☐ Re-assign the `x` object with the value `6`.
- ☐ Copy the following to your R Script:

```
x + y
```

- ☐ Run the line and see what happens.

Object reassignment

When you reuse an object name and assign it a new value, R overwrites the previous value.

Presenter Notes: Let's put this to practice by using our objects in an operation. But first, we will reassign object `x`. We will see that if you assign a new value to the same object name, it simply overwrites the old one. So if x was 4 and is now re-assigned as `x <- 6`, the value is now 6. R always uses the most recent assignment. When we complete this practical exercise what do we observe about object `x`?

Instructor Notes: It is an important point to make that you can re-assign objects with new values. Suggestion: Propose the question "*Why would we want to overwrite objects with new values?*".

Possible answer: This is a useful feature to avoid cluttering of objects- knowing that you can simply swap out the value of an object instead of creating a new object altogether.

Functions

What is a function?

- A **function** is a reusable block of code that performs a specific task.
- You provide input values (**arguments**), and R returns an **output** based on internal operations.
- Analogous to a mathematical function:
  - If `f(x) = 2 * x`, then inputting `x = 3` returns `6`.
- Every function has a unique name, followed by parentheses `()`, such as `sum()`.

```
# Example of a function taking multiple inputs
sum(1, 2, 3, 4)
```

Presenter Notes: Now that we know how to store data in our working directory, let's talk about how to manipulate it. A function is basically a recipe. You put ingredients in—these are called arguments—and the function gives you a finished dish as output. It saves us from writing complex mathematical formulas from scratch.

Instructor Notes: Emphasize the syntax: the function name is always immediately followed by parentheses. This is how R distinguishes a function from an object.

The `c()` function

- **Group items with the `c()` function** to manipulate multiple items at once.

```
# Example 1: Combine four numbers into one object
age <- c(19, 20, 21)

# Now we can apply a function to the entire group
sum(age)
```

```
[1] 60
```

```
# Example 2: Combine four character values into one object
food <- c("Pizza", "Pasta", "Burger", "Pasta")

# Now we can organize all the items in this object in a frequency table
table(food)
```

```
food
Burger  Pasta  Pizza
     1      2      1
```

Presenter Notes: Before we look at more functions, we need to know one very useful function: the letter 'c'. It stands for combine. Grouping items together allows us to work with groups of information at a time.

Let's look at the first example. First we will group a series of numbers and assign it to the object "age". Now, whenever we call on "age", we are calling on this group of numbers instead of just one number. So, if we want to calculate the sum of all the numbers, instead of writing out each number, we just run the `sum` operation on the "age" object.

In the second example, we first use the c() function to group several text values together into a vector. Because these are character values, mathematical operations such as `sum(food)` do not make sense and will return an error. However, we can use functions such as `table()` to count how often each item appears in the object "food".

Common built-in functions: Numeric

R comes with many built-in **numeric** functions:

Function	What it does
`sum()`	Adds all values together
`mean()`	Calculates the arithmetic mean (average)
`sd()`	Calculates the standard deviation
`exp()`	Calculates the exponential function (e^x)
`round()`	Rounds a number

Presenter Notes: Here are some of the most common mathematical and statistical functions you will use in R when working with numerical values. They do exactly what their names suggest. For example, `sd()` calculates the standard deviation, and `exp()` calculates the exponential.

Practical exercise 6

- ☐ Combine the numbers `4.4`, `14`, `5.26`, and `26` and assign it to an object named "`numbers`"
- ☐ Try out the functions `sum()`, `mean()`, and `sd()` using the object "`numbers`" as the argument and inspect the output

Presenter Notes: Let's test out the `c()` function to group some values into an object and then run some numerical operations. First we will group a series of numbers and assign it to the object "numbers". Then we can call on this object and run some numerical functions.

Instructor Notes: Give learners about 5-7 minutes. Walk around to ensure they are properly using the `c()` function to create the vector. A common beginner mistake is typing `mean(4.4, 14, 5.26, 26)` directly, which will not work as expected because `mean()` only calculates the mean of the first argument if they aren't combined into a vector first.

Common built-in functions: Character

R also comes with many built-in **character** functions:

Function	What it does
`sort()`	Sort values in alphabetical order
`unique()`	Shows the unique values in the set

Function	What it does
`table()`	Counts the values in a table
`toupper()`	Transforms all the letters to uppercase
`tolower()`	Transforms all the letters to lowercase

Presenter Notes: As we saw in the example, we can group and work with multiple character values as well. Here are some of the built-in functions that allow us to work with these.

Function arguments

Most functions take multiple arguments (**inputs**).

- Arguments are the pieces of information a function needs to **carry out an operation**.
- Different functions expect different types and numbers of arguments, and the **output** of a function often **depends on the arguments** that are supplied.
- It might look something like this: `function_name(argument1 = x, argument2 = y)`

> **i** Do all functions need multiple arguments?
>
> No, not all functions need multiple arguments, but for some, it is required.

Presenter Notes: Functions often need information to work with, and that information is provided through arguments, or inputs. Think of a function as having two parts: outside the parentheses is *what you want R to do*, while inside the parentheses is *what you want R to do it with*. For example, in `mean(x)`, mean tells R to calculate an average, and `x` tells it which data to calculate the average from. Some functions only need one argument, while others can take several depending on the task.

Instructor Notes: Some functions need more than one piece of information. Up to this point, learners have seen functions being used, but may not have thought about what goes inside the parentheses. Emphasize that the values inside the parentheses are called arguments and that they provide the information the function needs to work. Different functions need different information, so the arguments will vary depending on the task.

Function arguments

Example: The `round()` function

- This function uses two arguments: It approximates a number (**x**) to a specified amount of decimal places (**digits**).
- You can provide arguments by position or by name.

```
# By position (R assumes the first number is x, the second is digits)
round(3.14159, 2)

# By name (explicitly telling R which argument is which)
round(x = 3.14159, digits = 2)
```

> Naming your arguments makes your code easier to read!
>
> If you name the arguments (using the names "x" and "digits"), the order does not matter: `round(digits = 2, x = 3.14159)` works identically.

Presenter Notes: For instance, to round a number, R needs to know two things: what number to round and how many decimals to keep. You can just type the numbers in the right order, but explicitly naming the arguments, like writing `digits = 2`, is highly recommended because it makes your script much easier to understand when you read it a month later.

Practical exercise 7

- ☐ Assign a "`my_number`" object using `2.4637`
- ☐ Try out the function `round()` using "`my_number`" and approximating to 3 decimal places.

> Ignoring the **digits** argument
>
> If you are curious, try running the `round()` function without specifying the `digits =` argument. What happens? Why might that be?

Presenter Notes: Next, let's practice specifying arguments using the `round()` function. We can reuse the "my_number" object name, simply re-assign it with a new number. Then, follow the order from the previous slide on how to specify the arguments.

Instructor Notes: It may be confusing to remember the order and to know what goes where.

Pre-break quiz

Question 1

What does the # symbol do in R?

a. Runs the code

b. Creates a variable

c. Starts a comment

d. Multiplies numbers

Instructor notes: The answer for this question is c

Question 2

What happens when you "run" code in R?

a. The code is deleted

b. R executes the command

c. The code becomes a comment

d. RStudio closes automatically

Instructor notes: The answer for this question is b

Question 3

What will R return for this comparison?

```
49 < 50
```

a. TRUE
b. FALSE
c. 49
d. ERROR

Instructor notes: The answer for this question is a

Question 4

Which of the following is a character value in R?

a. TRUE
b. 5
c. “hello”
d. 4.2

Instructor notes: The answer for this question is c

Question 5

Why does the following produce an error?

```
4 + "five"
```

a. “five” is a logical value
b. The quotation marks are incorrect
c. 4 is not numeric
d. R cannot numerically add different data types together

Instructor notes: The answer for this question is d

Presenter Notes: Before we head into the break, let’s do a quick check-in to see how things are going so far. The questions are designed to help you reflect on your current understanding of R, RStudio, and the key ideas we’ve covered in this first part of the session.

Instructor Notes: Use this pre-break survey to gauge learners’ understanding of the session’s core concepts.

- If more than 80% of learners select the correct answer, briefly confirm the correct response and continue.
- If fewer than 80% answer correctly, give learners 1–2 minutes to discuss their reasoning with a partner before re-running the question. If results improve above 80%, continue; otherwise, explain the correct answer and address common misunderstandings.
- If a large portion of learners select the same distractor answer, pause to discuss why that option may seem plausible and clarify the misconception.

Accessibility Tip: This activity is designed for synchronous teaching settings and may be skipped or adapted if needed.

Break! 15 minutes

Post-break quiz discussion

What do we see in the results?

Presenter Notes: Let us have a look at these results.

Instructor Notes: Briefly examine the answers given to each question interactively with the group. Use visuals from the survey to highlight specific answers. Serves to clarify concepts and aspects that are not yet understood. Highlight specific answers given during the survey.

Accessibility Tip: Have people work in pairs if someone did not bring their phone/does not have a phone. This survey also cannot be completed in an asynchronous setting.

The Help function

Do I have to memorize all of the functions and arguments?

- **No!**
  - There is no need to memorize functions and their arguments.
  - You will **remember** the ones you use frequently **through practice**.
- When you need to understand how a function works, what inputs it needs, or what its defaults are, you can consult the **Help Function**.

Presenter Notes: A common worry when learning a programming language is feeling like you have to memorize a dictionary of functions. Please don't! Even advanced programmers look up functions every single day. The more you use certain functions, the more naturally they will come to you. For everything else, R has a built-in manual called the Help Function.

Instructor Notes: Normalize the act of looking things up. You can mention that googling errors or looking at documentation is 80% of programming.

How to access Help in R

There are two main ways to look up a function.

Method 1: Using the Console

- Type a question mark `?` into the console immediately followed by the function name and press Enter.

```
# Example: Getting help with using the round() function
?round
```

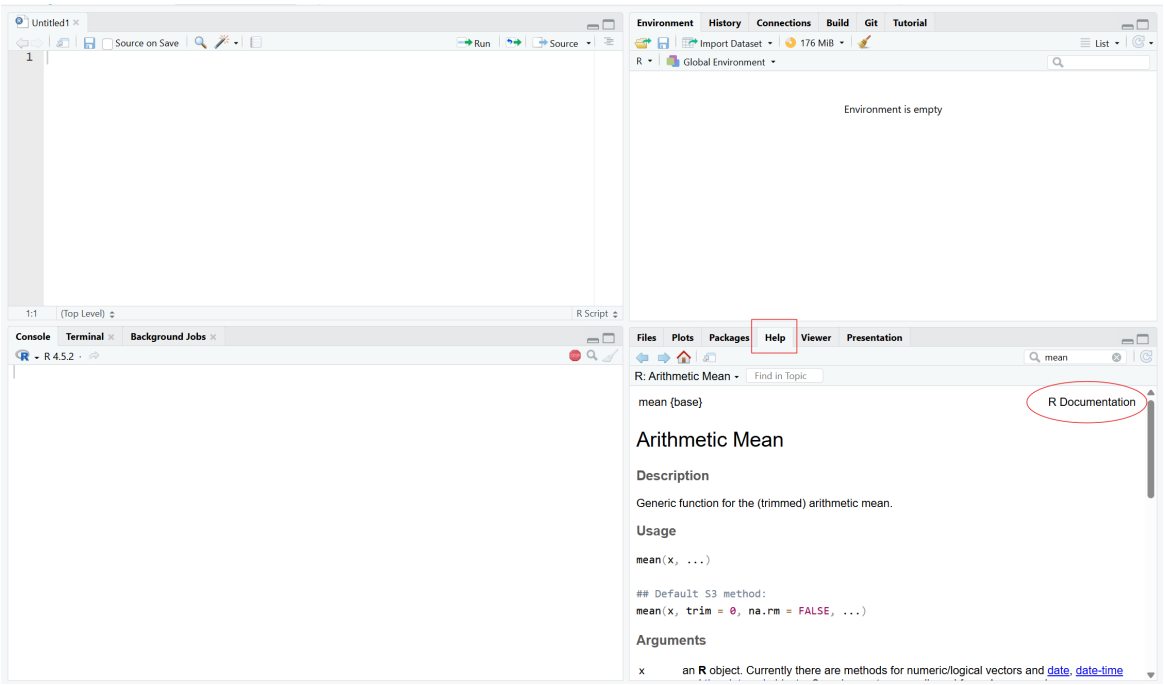
Presenter Notes: There are two simple ways to access help for a function in R. The first is by using a shortcut in the Console. Simply type a question mark (?) immediately before the function name without parentheses and press Enter. For example, typing `?round` opens the help page for the `round()` function, where you can find information about what it does, its arguments, and examples of how to use it.

Instructor Notes: Emphasize that the syntax matters. The question mark (?) must come before the function name, and parentheses should not be included. For example, `?round` is correct, whereas `?round()` is not. Encourage students to use this frequently whenever they are unsure how a function works.

How to access Help in R

Method 2: Using the Help Tab

- Navigate to the **Help** tab in the bottom right pane of RStudio and type the function name into the search bar.



Presenter Notes: Alternatively, you can use the search bar in the Help tab located in the bottom right pane of RStudio. Here, you can type directly in the *search bar* to look up the name of the function.

Reading the documentation

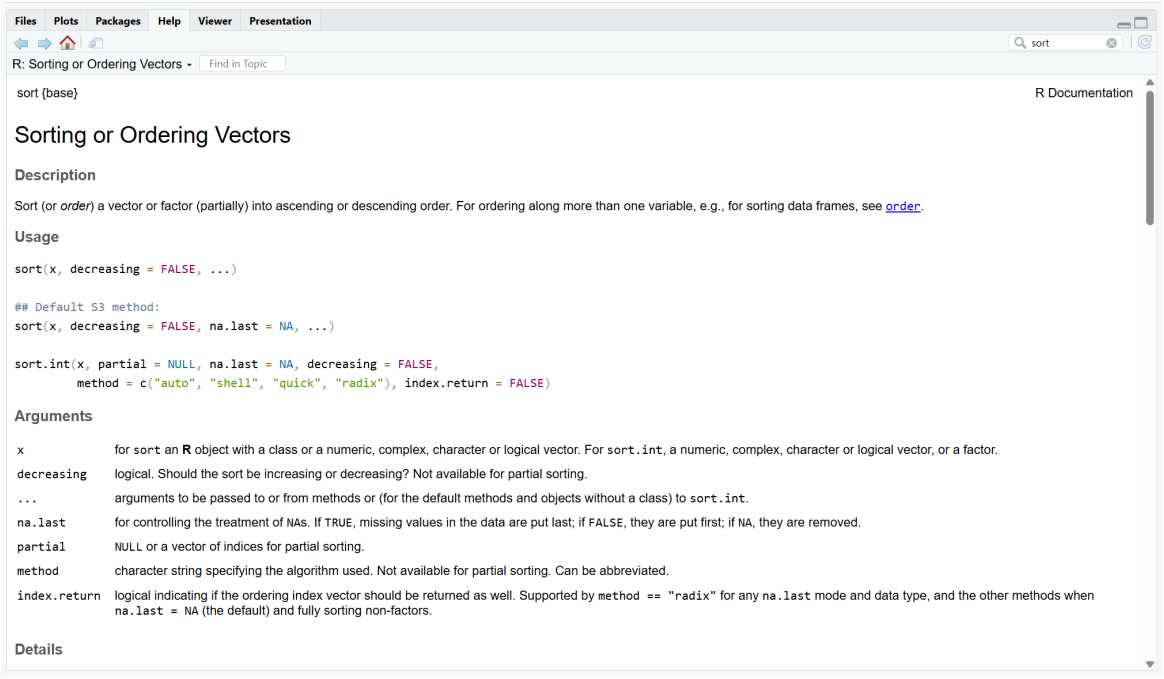
Help pages can look intimidating, but you only need to focus on a few key sections:

- **Description:** A brief summary of what the function does.
- **Usage:** Shows the function with all arguments. This is where you see *default values*.
- **Arguments:** Detailed explanations of what each input to the function should look like.

Presenter Notes: The documentation might look overwhelming at first because it uses technical terms. As a start, focus on the "Usage" and "Arguments" sections.

Reading the documentation

Example: For the `sort()` function we can see that it can also use character vectors or values as input and orders them.



Presenter Notes: In this example, we use the documentation for `sort()` to see that the function can work not only with numbers, but also with character values. By looking at the *Usage* and *Arguments* sections, we can identify that the first argument, `x`, is the object(s) we want to sort. We can also see that the argument `decreasing = FALSE` means that, by default, the values are sorted in ascending order (A to Z for text). This illustrates how the documentation can help us understand both what a function does and how we can customize its behavior.

Practical exercise 8

- ☐ Type `?sort` in the **Console** and press Enter.
- ☐ Look at the **Usage** and **Arguments** sections in the documentation that opens in the bottom right pane to see which inputs the function accepts.
- ☐ Put the following as the input for the function:

```
c("Tim", "Tom", "Anna", "Pizza", "Burger", "Pasta")
```

☐ Read the **Arguments** to figure out what must be included to put the words in *reverse alphabetical* order

> **i** Is reverse alphabetical order increasing or decreasing?
>
> Reverse alphabetical order means arranging items from **Z to A**. This means that the order is *decreasing*.

Presenter Notes: Let's try this exercise to become more comfortable navigating R's documentation. Don't worry if some of the terminology seems unfamiliar. Focus on identifying the important information: what the function does, what inputs it accepts, and how to change its behavior.

Instructor Notes: This exercise can be tricky since the Help documentation has a lot of information that can be confusing to learners. It's important to let them know that not every piece of information is relevant and just to focus on the pieces they need to complete the exercise.

Some issues you might encounter: - Missing parentheses () around ALL the arguments. - Missing comma , after the parentheses that hold the character values. - Must use the `decreasing = TRUE/FALSE` argument (in this case, it must be `decreasing = TRUE`). Using `increasing = FALSE` would NOT work. - They may feel the need to include the other arguments they see in the document but this is unnecessary.

Vectors, lists, and data frames

Data structures in R

- Collections of values are called **data**
- R provides ways to store data in different **data structures**:
  - **Vectors**
  - **Lists**
  - **Dataframes**

Presenter Notes: So far, we have mostly worked with single values. Most of the time, we are interested in a collection of values rather than a single value, allowing us to work with many observations at once. R provides different data structures to help us organize and work with collections of data.

Instructor Notes: This next section introduce several new technical concepts that beginners often confuse. Regularly remind students what vectors, lists, and data frames are as they appear, and briefly recap the differences before introducing the next concept.

Vectors

What does a vector do? Stores multiple values in an object.

- It is a one-dimensional collection of data
- Useful for storing scores, ages, measurements, names, and more

Example:

```
scores <- c(75, 82, 91, 68)
```

> Vectors and data type
>
> What do you notice about the values in the vector stored in `scores`? All values in a vector should be of the same data type.

Presenter Notes: A vector is one of the most common data structures in R. Think of it as a container that stores many related values under a single name. Instead of creating separate objects for each score, we can store them together in one vector. Important to note is that vectors only contain one data type, e.g., all values in a vector are numeric, or all values are character type.

Instructor Notes: For added clarity, you can emphasize that, in the example, "`c(75,82, 91, 68)`" is the vector and "scores" is just the object name in which the vector is stored.

Practical exercise 9

- ☐ Create a vector containing the numbers `19`, `20`, and `21` and assign it to an object called "`age`".
- ☐ Create a vector containing the names `Tim`, `Tom`, and `Anna` and assign it to an object called "`name`".
- ☐ Create a vector containing the foods `Pizza`, `Burger`, and `Pasta` and assign it to an object called "`food`".

> Character data type
>
> Reminder: character data types need to be wrapped in quotation marks `" "`.

Presenter Notes: Let's try making some vectors of our own using both numerical and word data types. But remember, for each vector, just use one datatype. At the end of this exercise, you should have two separate vectors each assigned to a different object.

Instructor Notes: Here are the solutions:

```
age <- c(19, 20, 21)
name <- c("Tim", "Tom", "Anna")
food <- c("Pizza", "Burger", "Pasta")
```

Though it is added as a callout-tip, still be mindful that a common mistake is forgetting to wrap the character and word data in quotation marks. This will lead to an error.

Lists

What does a list do? Stores different types of data together.

- It is also one-dimensional
- *But*, unlike vectors, it can contain **different data types** (e.g., numeric, character…)

Example for a list:

```
student <- list("Alex", 21, TRUE)
```

Presenter Notes: Unlike vectors, lists can contain different types of information. For example, text, numbers, and logical values can all be stored together. We won't use lists much in this introductory session, but it's useful to know they exist.

Instructor Notes: Since they are quite similar, it can be useful here to note that both lists and vectors group together pieces of information but the difference is that lists can contain different datatypes.

Also, for added clarity, you can mention that, in this example, "`list("Alex", 21, TRUE)`" is the list and "student" is simply the object name in which the list is stored or assigned to.

Dataframes

What does a dataframe do? Stores data in rows and columns.

- It is **two**-dimensional
- **Rows** represent observations, **columns** represent variables

Name	Age	Food
Tim	19	Pizza
Tom	20	Pizza
Anna	21	Pasta

> **i** Look familiar?
>
> Think of a dataframe like a spreadsheet. The top row is the heading and the information below (within each column) is the observation corresponding to the heading.

Presenter Notes: Dataframes are one of the most important data structures in R because most datasets are stored this way. If you've used Excel or any type of spreadsheets before, think of a dataframe as a table where rows contain observations and columns contain variables. See here in the example, this is how the information would be stored in a dataframe.

Intructor Notes: To avoid confusion, use the example to explain further the breakdown of a dataframe: `Name` is the observation- it's what we are interested in finding out. Then each name under it (`Tim, Tom, Anna`) are the variables within this specific observation.

Practical exercise 10

Create a dataframe with the `data.frame()` function

- ☐ Copy the following to combine the vectors you created before into the **dataframe** named "`people_data`"

```
people_data <- data.frame(name, age, food)
```

- ☐ Call on the dataframe and observe what happens

Presenter Notes: Here we're are using the three separate vectors we created in the previous exercise and combine them them into a dataframe with the `data.frame()` function. Notice how each vector becomes a column in the final table.

Instructor Notes: A concern that you can clear up here is confusing the observations in the dataframe with character data types- when using them in `data.frame()`, the observations do NOT need to be wrapped in quotations.

Calling on the dataframe is done by simply typing the object name of the dataframe ("`people_data`" in this case) in the script (or Console) and running it.

Indexing

What does indexing do? Allows us to access specific parts of our data, such as:

- **Select** individual or multiple values
- **Access** rows and columns in a dataframe

Presenter Notes: Indexing means selecting a specific part of an object. Instead of viewing all values at once, we can ask R to return only the piece we need.

Indexing vectors

Each element in a vector has a position.

```
age <- c(19, 20, 21)
```

Position	1	2	3
Value	19	20	21

Presenter Notes: Let's first look at this vector to understand that each element within that vector has a position. The values are stored sequentially, so the first value is in position 1, the second in position 2, and so on.

Indexing vectors

Selecting individual values

- Use `[ ]` to call on a value by its position

```
# Example 1: Calling on the value in position 1
age[1]
```

```
[1] 19
```

```
# Example 2: Calling on the value in position 3
age[3]
```

```
[1] 21
```

Presenter Notes: Now that we know each value has a position within the vector, we can use this to call on it. The first possible position that R assigns is 1, so calling on the item in position "`1`", is calling on the first item in the vector. We specify the position by using the square brackets [ ] where number inside the square brackets refers to the position of the value we want to retrieve. Let's see this in action with the examples: we first specify the object ("`age`") and then we use the square brackets to indicate which is position of the items we want ("`[1]`" or "`[3]`").

Indexing vectors

Selecting multiple values

You can retrieve **more than one** value at once.

```
age[c(1, 3)]
```

```
[1] 19 21
```

You can also select a **range**:

```
age[1:3]
```

```
[1] 19 20 21
```

Presenter Notes: We can also use indexing to select several positions at the same time. For this, we utilize a comma between the positions. Additionally, we can use the colon operator to call on a sequence of positions.

Indexing dataframes

Accessing rows and columns in a dataframe

- You can call on values directly from the row and column of data frames.
- To use indexing, use `[ ]` and follow this general format:
  - `dataframe[row,column]`

```
# Example 1: Calling the value in row 1, column 1
people_data[1, 1]
```

```
[1] "Tim"
```

```
# Example 2: Calling the value in row 3, column 2
people_data[3, 2]
```

```
[1] 21
```

Presenter Notes: Unlike vectors, dataframes are two-dimensional because they contain both observations (rows) and variables (columns). This means that when indexing a data frame, we need to specify two positions: the row position and the column position. Just like vectors, R assigns positions starting at 1. The columns are numbered from left to right, so the first column is in position 1, the second column is in position 2, and so on. Likewise, the rows are numbered from top to bottom, starting with the first observation **below the column headings**. To access a value, we first specify the row (observation) and then the column (variable).

Accessing columns by name

Calling values by using just numbers can get confusing.

Luckily, there is a way to call on the values by **column name** using the `$` operator: `dataframe$column_name`

- Use it to retrieve all values from that specific column
- It is easier to read than using column numbers

```
# Example: Calling on the "name" column within the "people_data" data frame

people_data$name
```

```
[1] "Tim"  "Tom"  "Anna"
```

Presenter Notes: The dollar sign (`$`) is a convenient way to access a column in a data frame. Instead of remembering which column number contains the information we want, we can simply use the column's name or header. This makes code easier to read and understand.

Combining `$` and `[]`

Look at the following example to see how `$` and `[ ]` can be used together:

```
people_data$food[3]
```

```
[1] "Pasta"
```

What does this do?

1. `$` selects the **food** column in the `people_data` data frame
2. `[]` gives you the **third** value

Presenter Notes: This is a very common pattern in R. We first select a column and then use indexing to retrieve a specific value from that column. Looking at the example, we first state `people_data` then we use `$` to let R know we will be selecting specific information from this dataframe. Then we specify the information we want: using the column name, we want to look into the "food" column and then, using the square brackets [ ], we specify we want the third variable in this column.

Practical exercise 11

Use `$` and `[]` accordingly to retrieve the following information from the the `people_data` data frame:

- ☐ The first name in the "name" column
- ☐ The second age in the "age" column
- ☐ The third food item in the "food" column

Presenter Notes: Now we'll apply what we've learned about indexing data frames using column names. For this exercise, the names of the columns have already been provided for you, so your task is to retrieve the correct value by combining the appropriate column name with the correct row position. Remember that you can use either `$` or `[]` notation where appropriate.

Instructor Notes: Students commonly make a few mistakes here. Remind them that column names should not be enclosed in quotation marks when using the `$` operator (e.g., `people_data$name`, not `people_data$"name"`). Also watch for students reversing the indexing order- the correct syntax is always column name first, then position within the column: dataframe$column_name[column_position]

Wrapping Up

Looking into the future

1. **Take a minute to think about the following questions, first for yourself, then in a pair:**

- Can you identify some **practical benefits** of using R/RStudio?
- What **obstacles** do you anticipate in using it consistently?
- How can R/RStudio support you in your **upcoming projects**?

2. **Share your thoughts with the group.**

Presenter Notes: To wrap up, let's take a moment to bring everything together and think about how this connects to your own work going forward.First, take about one minute to reflect individually on these questions. Then discuss them with a partner.Think about where R/RStudio could realistically support you, for example, in upcoming essays, group projects, or your thesis work. What benefits do you see in using it? What might make it difficult to

use it consistently? And how could it fit into your current workflow?
After that, we'll briefly share some of your thoughts with the group.

Instructor Notes: Facilitate an open discussion focusing on application rather than theory. Encourage concrete examples from learners' own academic context. Actively prompt for both benefits and barriers. Keep the discussion time-bound to avoid overextension at the end of the session.

Accessibility Tip: Allow silent reflection time before discussion. Summarise key contributions aloud for clarity.

Assignment: Consolidating what you learned

Create a short R script that builds on what was covered today

1. Assign at least two objects, for example numbers or words.
2. Create at least one vector using c() and assign it to an object.
3. Perform at least one mathematical operation using an object with numerical values. Use at least one built-in function, such as `sum()`, `mean()`, `length()`, `sort()`, or `table()`.
4. Use at least one logical comparison to check a condition.
5. Practice indexing by retrieving one specific value from a vector using `[]`.
6. Include clear comments using # to explain each step.

> Self-Assessment
>
> Use this assignment to assess your learning of, and progress with, R/RStudio; reflect on any challenges you faced and how you overcame them.

Presenter Notes: To finish, you have a short home assignment that brings together everything we've covered so far in R and RStudio.
You'll create a small R script where you first define at least two objects, for example, numbers. Then you'll use those objects in at least one mathematical operation, such as addition or multiplication.
Next, include at least one logical comparison, where you check a condition and get a TRUE or FALSE result.
Finally, make sure to add comments throughout your script using the hash symbol, so it is clear what each step is doing.
This is not about getting everything perfect, it's about practicing how to structure a script and combine the basic elements we've learned.

Instructor Notes:
Clarify that this is a practice exercise rather than an assessment of correctness. Encourage experimentation and reassure students that errors are expected and useful for learning.

Take-home messages

- R → **your go-to tool** for data analysis and quantitative research.
- RStudio interface → helps you **organize code, outputs, and data** efficiently.
- Writing and saving R scripts ensures your work is **reproducible and well-documented**.

Presenter Notes:
To conclude this part, there are three key ideas to remember.
First, R is your main tool for data analysis and quantitative research. It's what you will use to run calculations, manage data, and perform statistical analyses.
Second, RStudio is the environment that makes working with R much more structured. It helps you keep your code, outputs, and data organised in one place so you don't lose track of your work.
And third, using R scripts is essential because it ensures your work is reproducible. That means you can go back to your analysis later, or share it with others, and everything can be followed step by step.

Instructor Notes:
Keep this slide concise and use it as a clear closure of the R part 1 section.

Learning goals: Check-In

Where are we at?

- To **navigate** the RStudio interface (Console, Environment, Script, and Output panes)
- To **differentiate** between basic data types in R (numeric, character, logical, integer) and how they are used
- To **execute** operations in the R using appropriate operators
- To **assign** and **re-assign** values, vectors, and lists to objects
- To **specify arguments** within functions to modify how the function produces its output
- To **index** dataframes to access and retrieve specific information

Presenter Notes: Let us pause for a quick check-in. Take a moment to look back at the learning goals and reflect on how comfortable you feel with each one. If you notice any gaps, this is the right moment to bring them up. Feel free to ask questions or start a discussion so we can address them before moving on.

Instructor Notes: Use this slide to encourage self-reflection rather than evaluation. Emphasize that learners are not expected to feel equally confident about all goals at this stage. Keep

the discussion supportive and brief,nfocusing on clarifying concepts rather than introducing new material.

To conclude: Survey time!

Question 1

On a scale of 1–5, how intimidated do you feel by programming or coding (e.g., R/RStudio) after completing this session? (1 = Not intimidated at all, 5 = Very intimidated)

a. 1
b. 2
c. 3
d. 4
e. 5

Question 2

How confident do you feel using the RStudio environment after this session?

a. Not confident at all

b. Slightly confident

c. Moderately confident

d. Very confident

e. Extremely confident

Question 3

How comfortable do you feel about using R and RStudio for mathematical operations?

a. Not comfortable at all

b. Slightly comfortable

c. Moderately comfortable

d. Very comfortable

e. Extremely comfortable

Question 4

How comfortable do you feel using R objects (e.g., vectors and data frames) to organize and manipulate your research data?

a. Not comfortable at all

b. Slightly comfortable

c. Moderately comfortable

d. Very comfortable

e. Extremely comfortable

Question 5

What part of the lesson, if any, still feels confusing or unclear?

Question 6

4. Was there a practical exercise or example that helped your understanding the most? Why?

Discussion of survey results

What do we see in the results?

Thanks!

See you next time!

Practical exercise 1: Solutions

```
12 + 17
```

```
[1] 29
```

```
218 / 3
```

```
[1] 72.66667
```

```
357^2
```

```
[1] 127449
```

Instructor notes: Here is the solution to Practical exercise 1

Practical exercise 2: Solutions

```
(36 + 5) > 40
```

```
[1] TRUE
```

```
(4 == 5) | (25 > 17)
```

```
[1] TRUE
```

```
(5 == (2 + 3)) & (7 >= 8)
```

```
[1] FALSE
```

Presenter Notes: Here are the final results. Understanding how these operators combine conditions is a core skill for filtering data later on.

Instructor notes: Here is the solution to Practical exercise 2

Practical exercise 6 & 7: Solutions

```r
# Create a vector and assign it to "numbers"
numbers <- c(4.4, 14, 5.26, 26)

# Try out sum(), mean(), and sd()
sum(numbers)
mean(numbers)
sd(numbers)

# Assign the number to "my_number"
my_number <- 2.4637

# Try out exp() and round()
round(my_number)
round(x = my_number, digits = 3)
```

Practical exercise 8: Solutions

```r
sort(c("Tim", "Tom", "Anna", "Pizza", "Burger", "Pasta"), decreasing = TRUE)
```

```
[1] "Tom"    "Tim"    "Pizza"  "Pasta"  "Burger" "Anna"
```

Practical exercise 9: Solutions

```r
age <- c(19, 20, 21)

name <- c("Tim", "Tom", "Anna")

food <- c("Pizza", "Burger", "Pasta")
```

Practical exercise 10: Solutions

```r
people_data <- data.frame(name, age, food)

people_data
```

```
  name age   food
1  Tim  19  Pizza
2  Tom  20 Burger
3 Anna  21  Pasta
```

Practical exercise 11: Solutions

```r
people_data$name[1]
```

```
[1] "Tim"
```

```r
people_data$age[2]
```

```
[1] 20
```

```r
people_data$food[3]
```

```
[1] "Pasta"
```

Additional resources

- Datacamp offers a free, interactive course on R online: link to the course.
- Here is a free-to-use RSdtudio user guide: https://docs.posit.co/ide/user/

Presenter Notes:
If you'd like to continue practicing R after this session, there is an optional additional resource available.
Datacamp offers a free, interactive introduction to R, where you can practice directly in your browser. It's a useful way to reinforce what we covered today and get more comfortable with writing code at your own pace.

You are not required to complete it, but it can be helpful if you want extra practice or a more structured introduction to R outside of class. Included is also a link to an RStudio user guide to become more aquanted with the layout and navigating using RStudio and R.

Instructor Notes: Present this as an optional extension, not part of the core learning requirements. Avoid suggesting it as mandatory. Emphasise self-paced learning and reinforcement of basic R skills.

Accessibility Tips:
Ensure the link is accessible through the course platform or slide deck.

Contribution statement

Creator: Sanchez Steinhagen, Caterina Luz (0009-0006-4783-1110)

Creator: Sharma, Tejaswini (0009-0000-0305-9751)

Creator: Waterfield, Elizabeth (0009-0006-3725-6730)

Creator: Rieger, David (0009-0004-3701-3101)

Reviewer: Von Grebmer zu Wolfsthurn, Sarah (0000-0002-6413-3895)

Consultant: Middleton, Sara Lil (0000-0002-6413-3895)

License and citation statement

This work was originally created by Nicklas Hafiz. His work was subsequently adapted by Caterina Luz Sanchez Steinhagen, Tejaswini Sharma, David Rieger, Elizabeth Waterfield, Sara Lil Middleton and Sarah von Grebmer zu Wolfsthurn. The current work is licensed under a Creative Commons Attribution 4.0 SA International License (CC-BY-SA 4.0). Code snippets are dedicated to the public domain and are licensed under a Creative Commons Universal Licence (CC0 1.0).

Please cite these materials as: Sanchez-Steinhagen, C. L., Sharma, T., Rieger, D., Waterfield, E., Middleton, S., & Von Grebmer zu Wolfsthurn, S. (2026). *Introduction to R: Part I* [Quarto slides]. LMU Open Science Center. https://lmu-osc.github.io/teaching-material-for-novices/open-research-tools/r-part-I

Presenter Notes: The Creative Commons Attribution 4.0 International SA License, or CC BY-SA 4.0, allows others to copy, share, and adapt a work in any medium, including for

commercial purposes. These permissions are broad and cannot be withdrawn as long as the license terms are followed. The main requirement is attribution: users must give appropriate credit to the original creator, provide a link to the license, and clearly indicate whether any changes were made, without implying endorsement by the original author. In addition, the ShareAlike condition means that if someone modifies or builds upon the work, the resulting material must be distributed under the same CC BY-SA 4.0 license, or a compatible one. Finally, users are not allowed to apply legal or technical restrictions, such as DRM, that would prevent others from exercising these same rights.

Code snippets are dedicated to the public domain and licensed under a Creative Commons Universal License (CC0 1.0). You may use, modify, distribute, and sell the code snippets for any purpose, without permission or attribution. The code snippets are provided "as is", without warranty of any kind.