Data Analytics with R

Programming Fundamentals of R

Learning Objectives

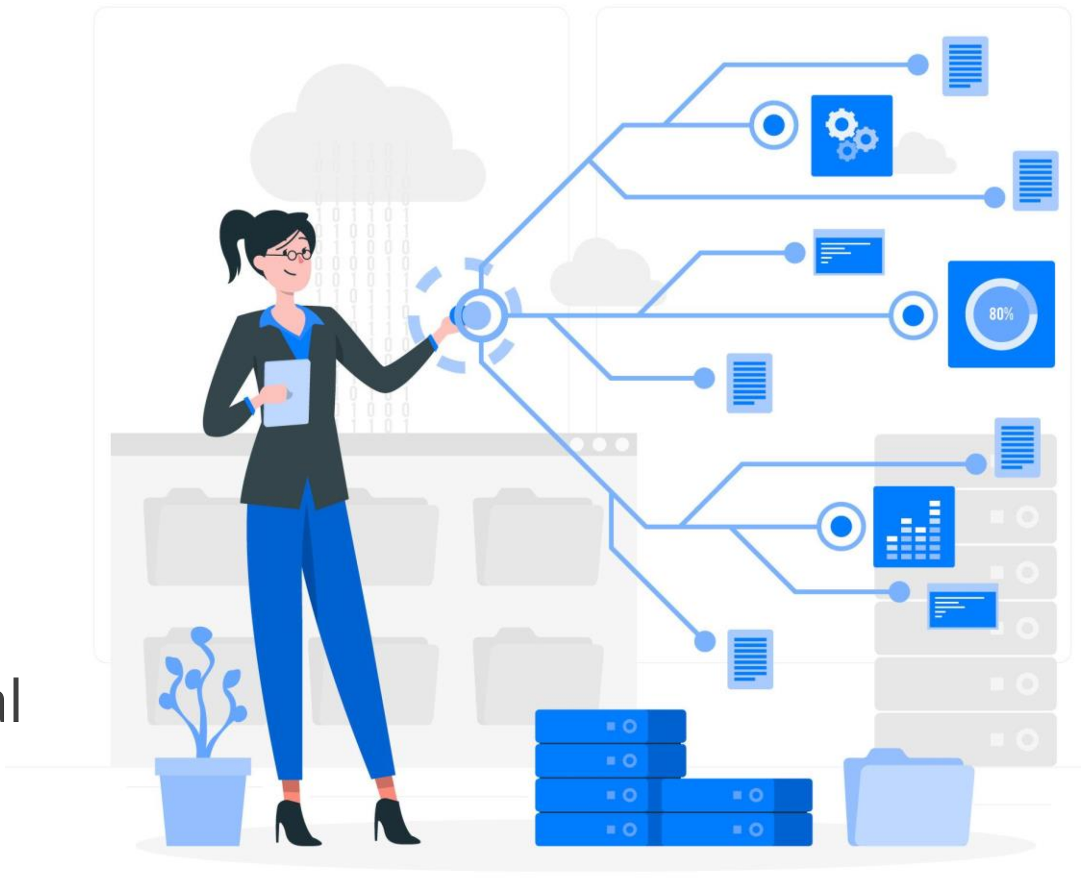
By the end of this lesson, you will be able to:

- Explain conditional statements
- Create and use different loops in R
- Define and use functions in R
- Work with date objects
- Use R Markdown

Business Scenario

- Anna joined an analytics center of excellence in a global organization. Her job is to share relevant data and information with different business units.

Approach: To create and manage data in R, Anna must understand the various data types, data structures, and operations in R and why R is the preferred tool for statistical computing.



Conditional Statements in R

Decision-Making

Decision-making can be achieved by evaluating conditional expressions.

- Conditional expressions return either True or False.
- R supports decision-making with if-else statement.

The if...else Statement

The if statement

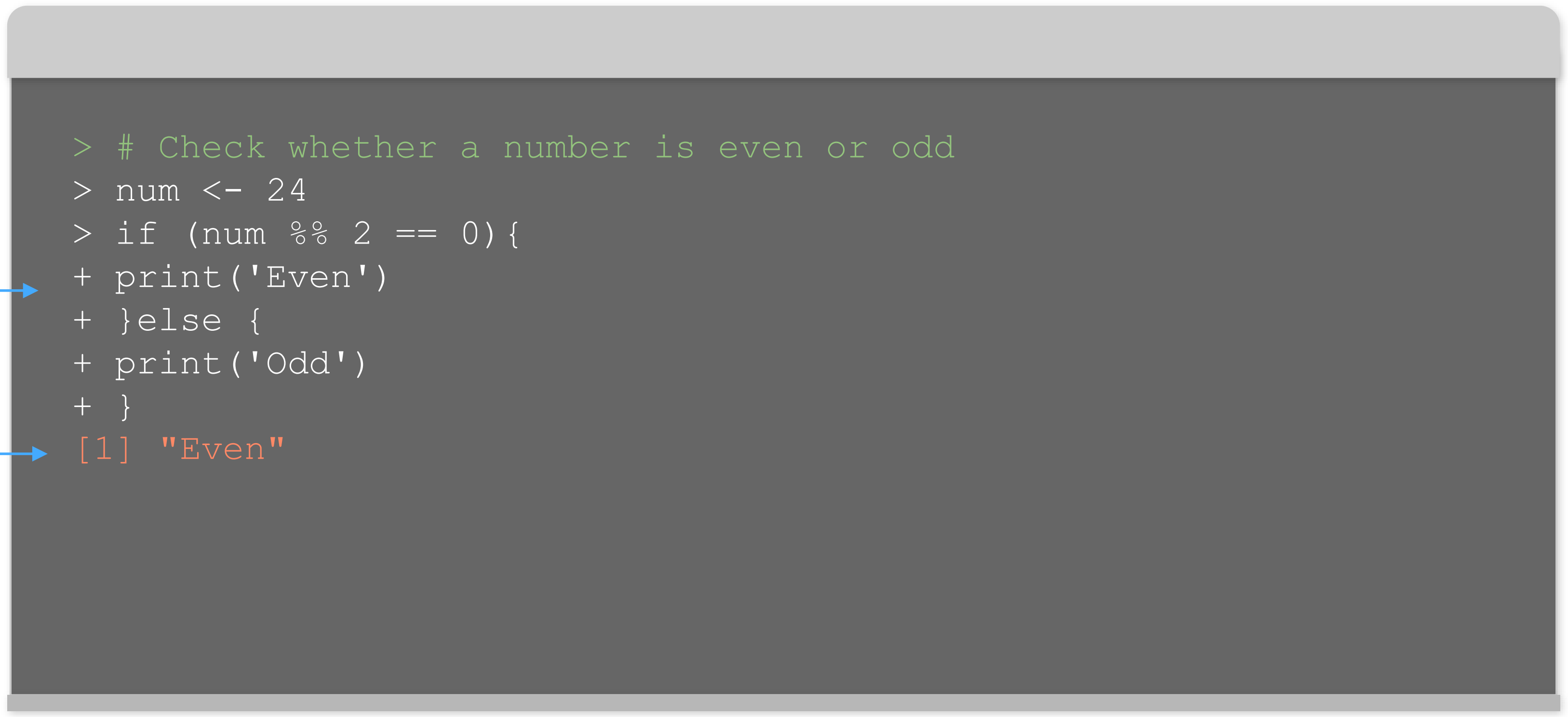
The if block gets executed when the conditional expression evaluates to True

The else statement

The else block gets executed when the conditional expression evaluates to False

The if…else Statement

Example: To check whether a number is even or odd



```
> # Check whether a number is even or odd
> num <- 24
> if (num %% 2 == 0){
+ print('Even')
+ }else {
+ print('Odd')
+ }
[1] "Even"
```

If…else statement

Output

The ifelse() Function

Below is a vector equivalent form of ifelse().

```
> # Check whether a number is even or odd using ifelse function
> num <- 37
> ifelse(num %% 2 == 0, 'Even', 'Odd')
[1] "Odd"
```

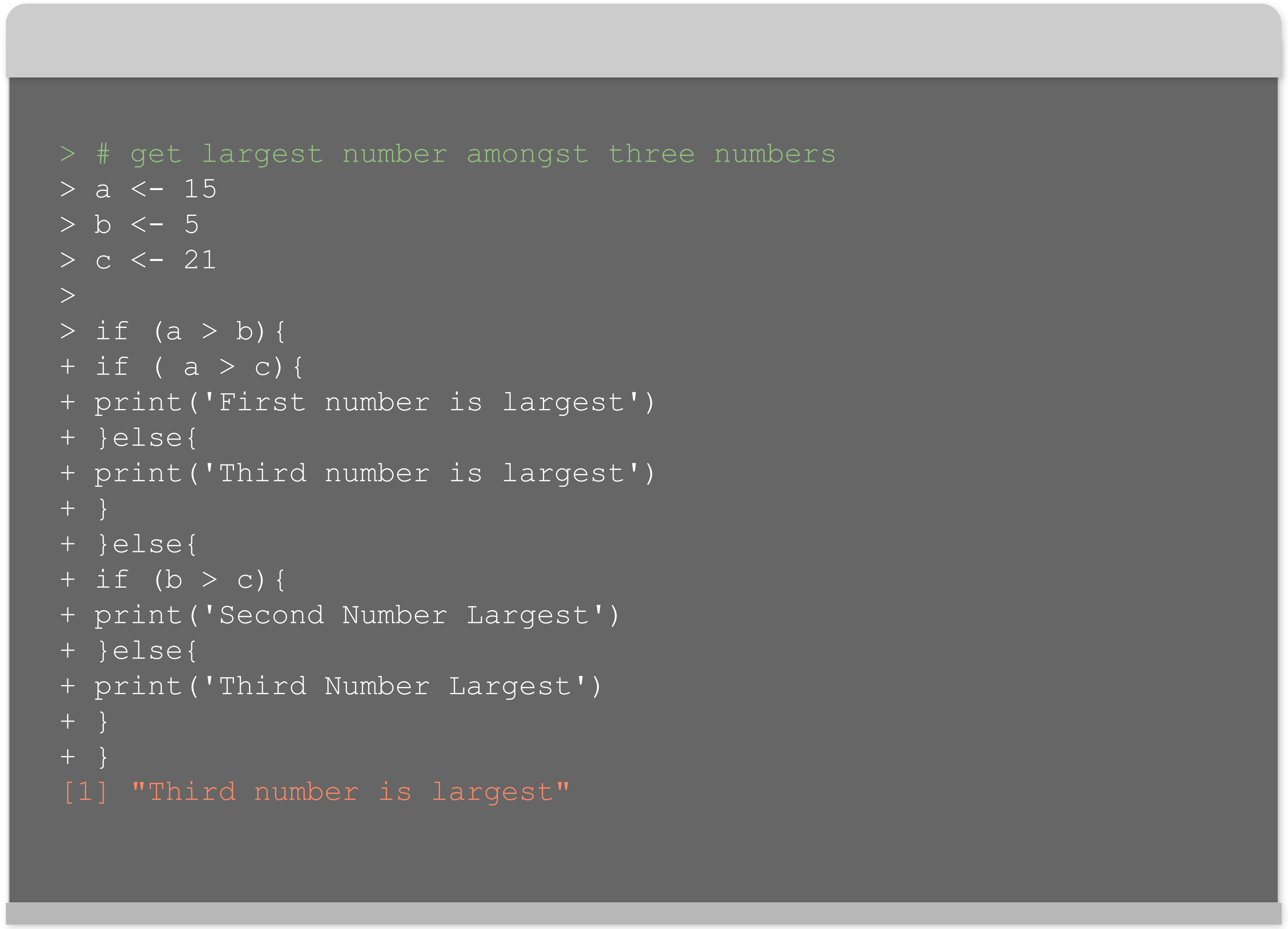
Nested Ifs

1 When one if-else statement is placed inside another, it is being nested.

2 The outer condition is checked first, and if the outer condition is True, then the inner condition is evaluated.

Nested Ifs

Example: Finding the largest number among the given three numbers using the nested if



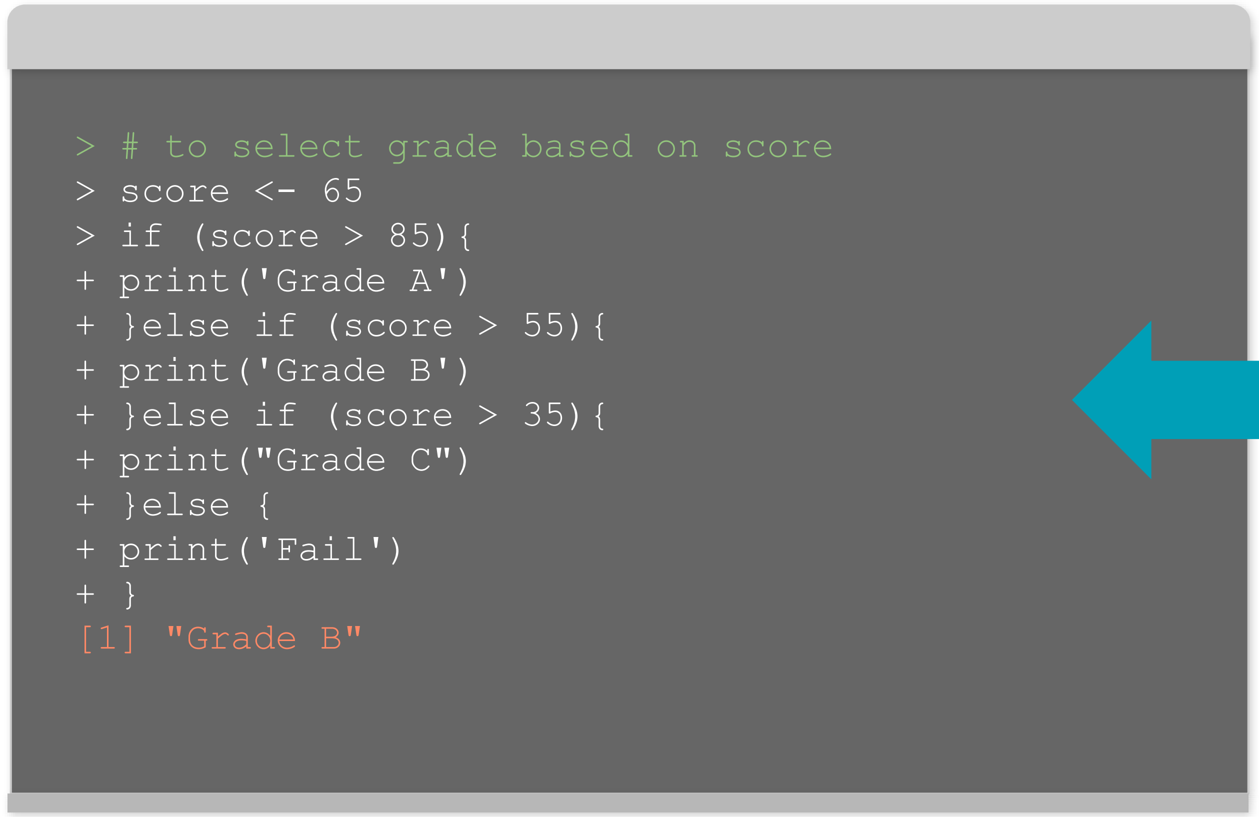
```
> # get largest number amongst three numbers
> a <- 15
> b <- 5
> c <- 21
>
> if (a > b){
+ if ( a > c){
+ print('First number is largest')
+ }else{
+ print('Third number is largest')
+ }
+ }else{
+ if (b > c){
+ print('Second Number Largest')
+ }else{
+ print('Third Number Largest')
+ }
+ }
[1] "Third number is largest"
```

If...else if Ladder

- It is another form of nested conditional statements.
- It is used to evaluate multiple cases sequentially.
- The conditional expressions are evaluated from the first condition to the last until a true expression is found.
- The associated statement is then executed and the rest of the ladder is bypassed.

If...else if Ladder

Example:



```
> # to select grade based on score
> score <- 65
> if (score > 85){
+ print('Grade A')
+ }else if (score > 55){
+ print('Grade B')
+ }else if (score > 35){
+ print("Grade C")
+ }else {
+ print('Fail')
+ }
[1] "Grade B"
```

The first condition is False as score is less than 85.

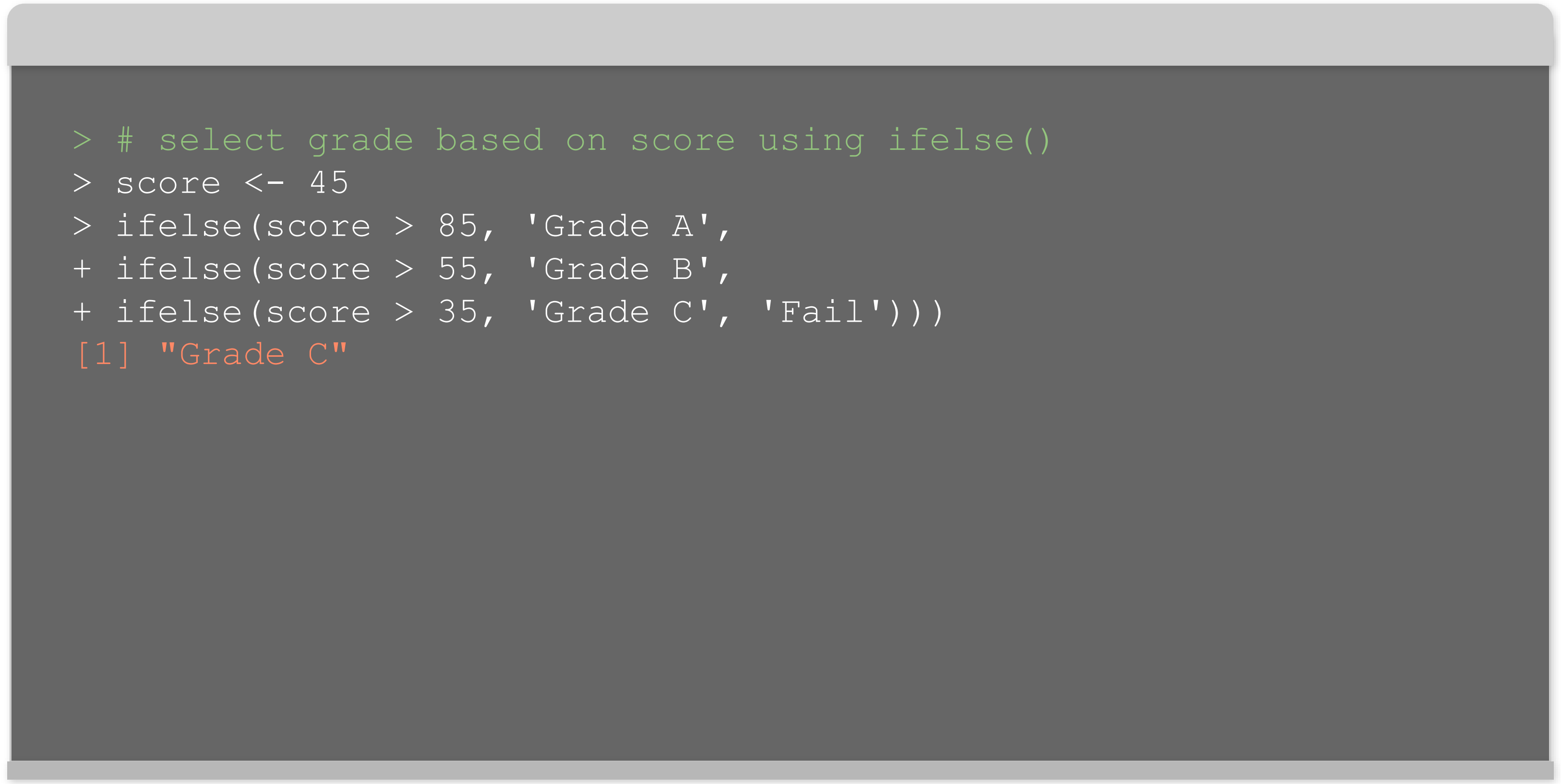
Hence, control moves to the next condition.

The second condition is found to be True.

The if block is executed, and the rest are skipped.

If…else if Ladder

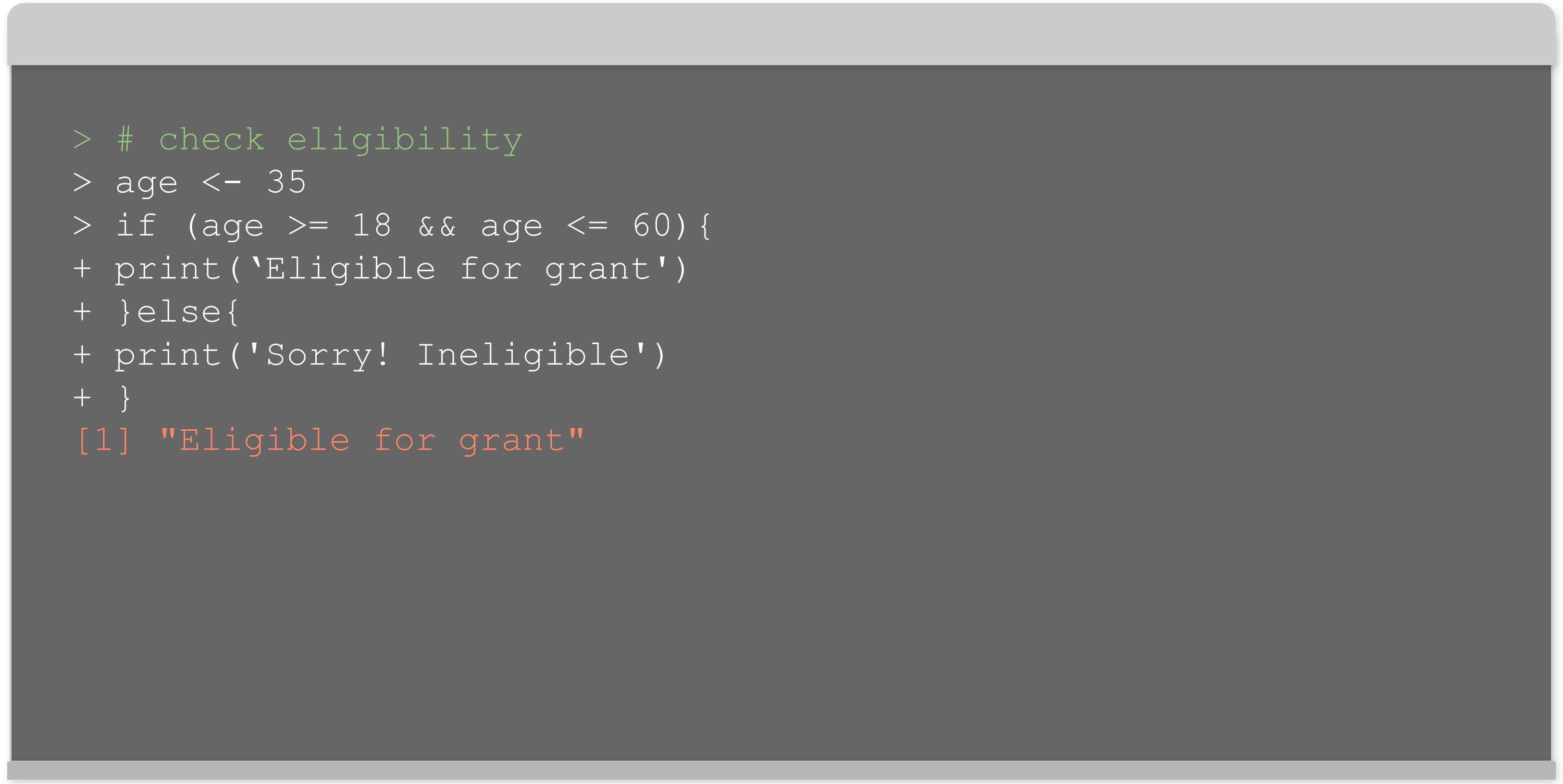
The if…else if ladder can also be created using the ifelse() function.



```
> # select grade based on score using ifelse()
> score <- 45
> ifelse(score > 85, 'Grade A',
+ ifelse(score > 55, 'Grade B',
+ ifelse(score > 35, 'Grade C', 'Fail')))
[1] "Grade C"
```

Multiple Conditions

Multiple conditional statements can be evaluated with a single if statement by combining them using operators such as *and (&&)* and *or (||).*



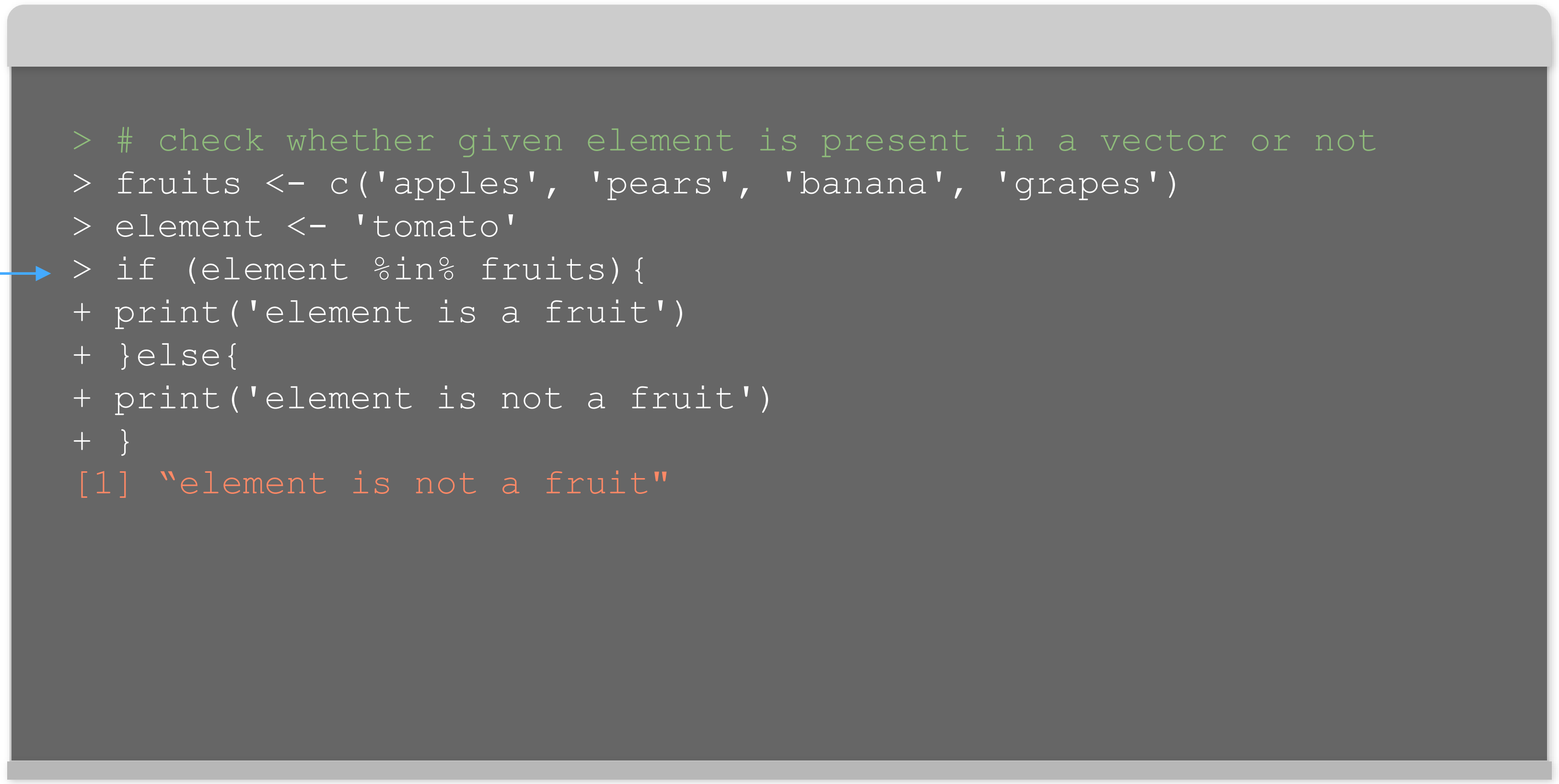
```
> # check eligibility
> age <- 35
> if (age >= 18 && age <= 60){
+ print(‘Eligible for grant')
+ }else{
+ print('Sorry! Ineligible')
+ }
[1] "Eligible for grant"
```

Example: Checking candidates' eligibility for a grant based on their age (18-60)

%in% Operator

The %in% operator allows one to search for a value in vector lists or dataframes.

Searching for an element in a vector list



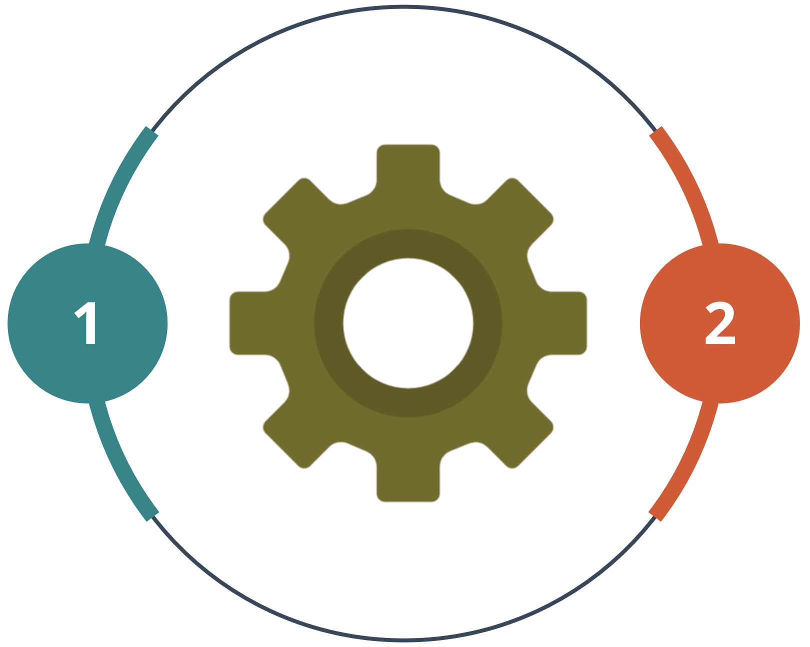
```
> # check whether given element is present in a vector or not
> fruits <- c('apples', 'pears', 'banana', 'grapes')
> element <- 'tomato'
> if (element %in% fruits){
+ print('element is a fruit')
+ }else{
+ print('element is not a fruit')
+ }
[1] "element is not a fruit"
```

Example: Checking if a given element is present in a vector

The switch() Statement

A switch() is a special type of conditional statement in R.

It compares an integer or a character value against a list of values and returns the corresponding value.

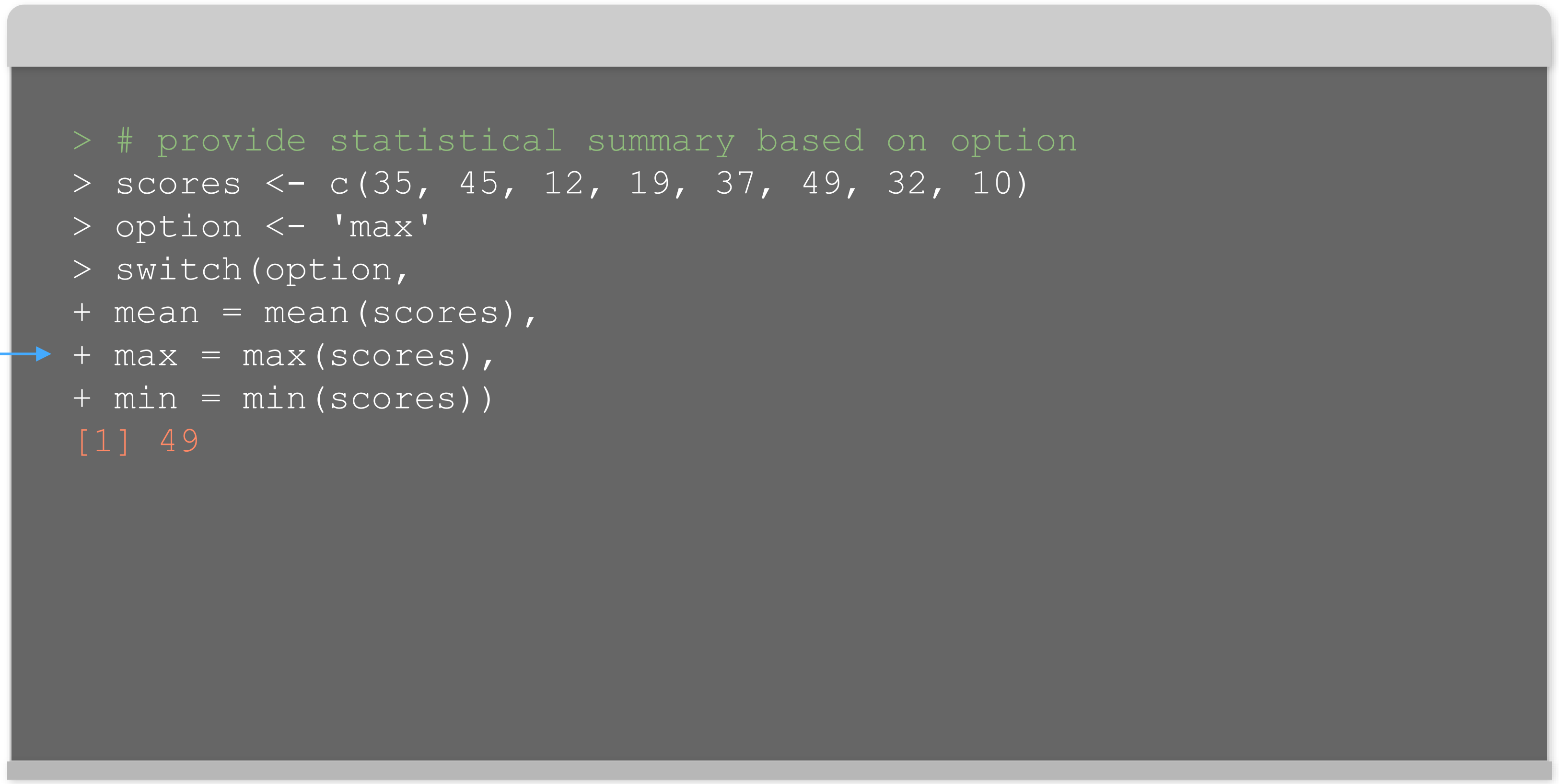


It is similar to a controlled branch of the if-else if statement.

The switch() Statement

Example: To provide statistical summary based on options listed

Using switch to list options



```
> # provide statistical summary based on option
> scores <- c(35, 45, 12, 19, 37, 49, 32, 10)
> option <- 'max'
> switch(option,
+ mean = mean(scores),
+ max = max(scores),
+ min = min(scores))
[1] 49
```

Loops in R

Loops in R

Loops are flow-control statements that iterate over a block of statements.

There are three types of loop statements in R:

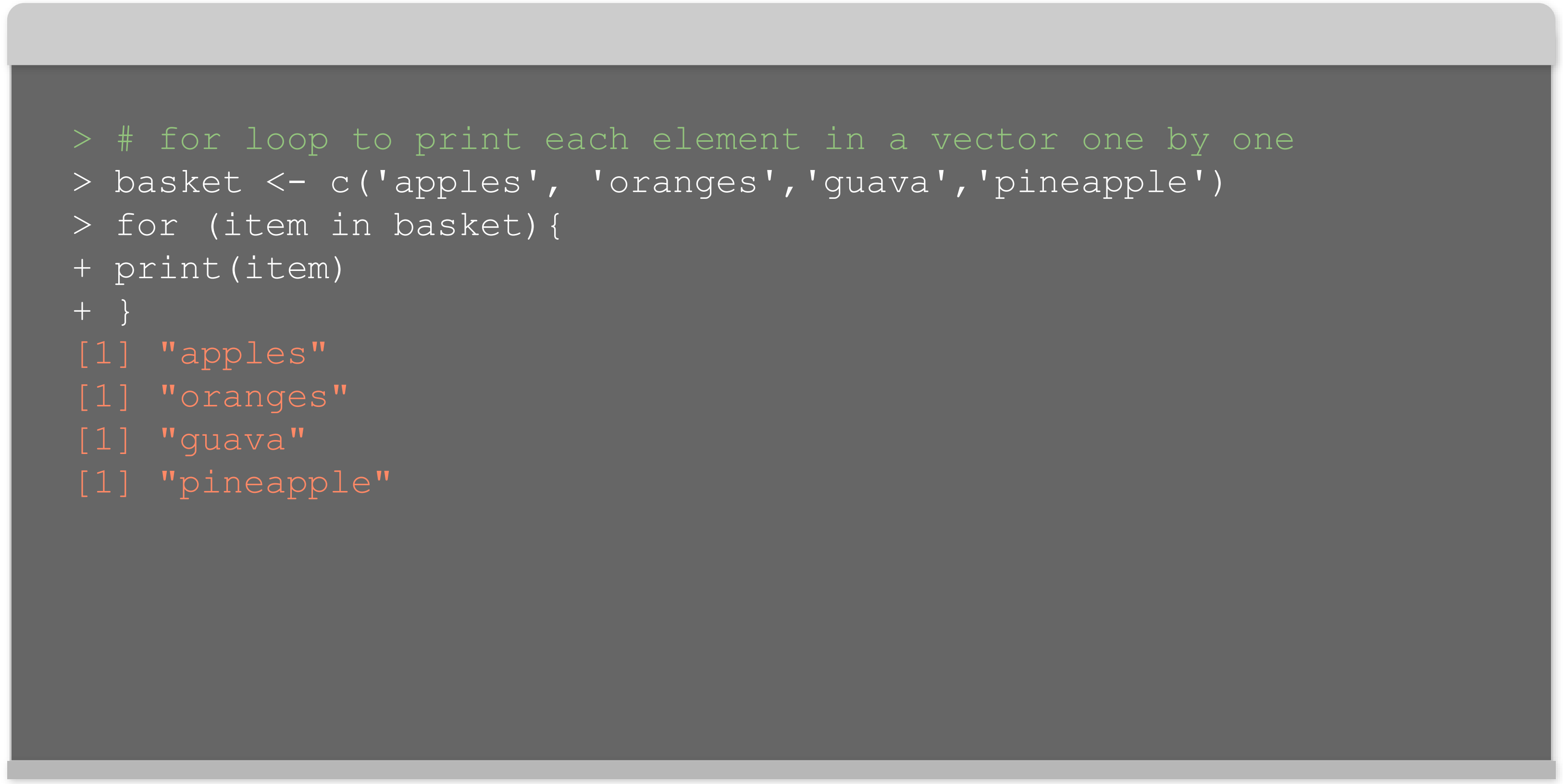
01 For loop

02 While loop

03 Repeat loop

For Loop

The for loop iterates over an R object, repeating a block of statements.

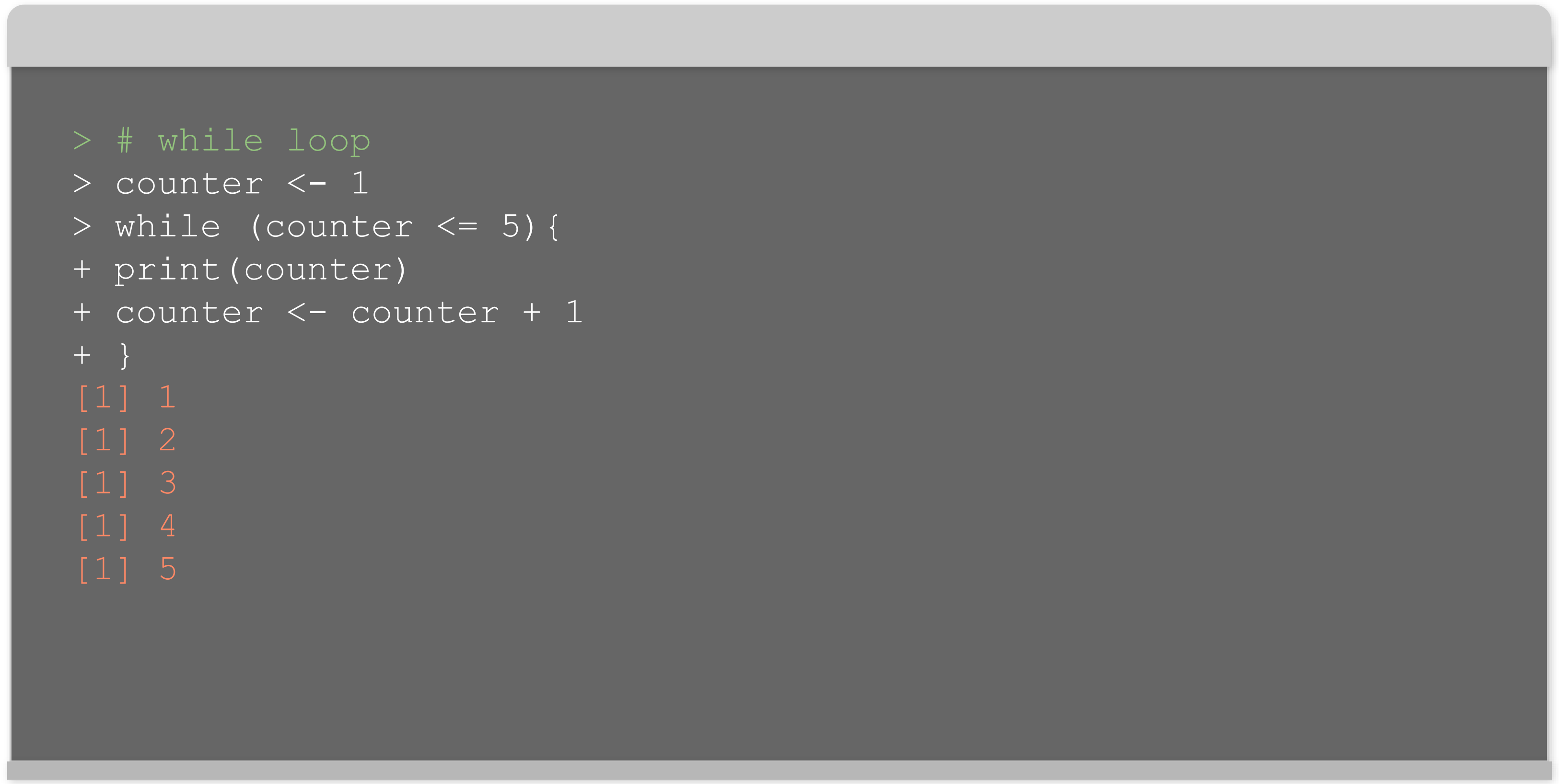


```
> # for loop to print each element in a vector one by one
> basket <- c('apples', 'oranges','guava','pineapple')
> for (item in basket){
+ print(item)
+ }
[1] "apples"
[1] "oranges"
[1] "guava"
[1] "pineapple"
```

Example: Using for loop to print each element in a vector one by one.

While Loop

The while loop iterates a block of statements till a test expression remains True.

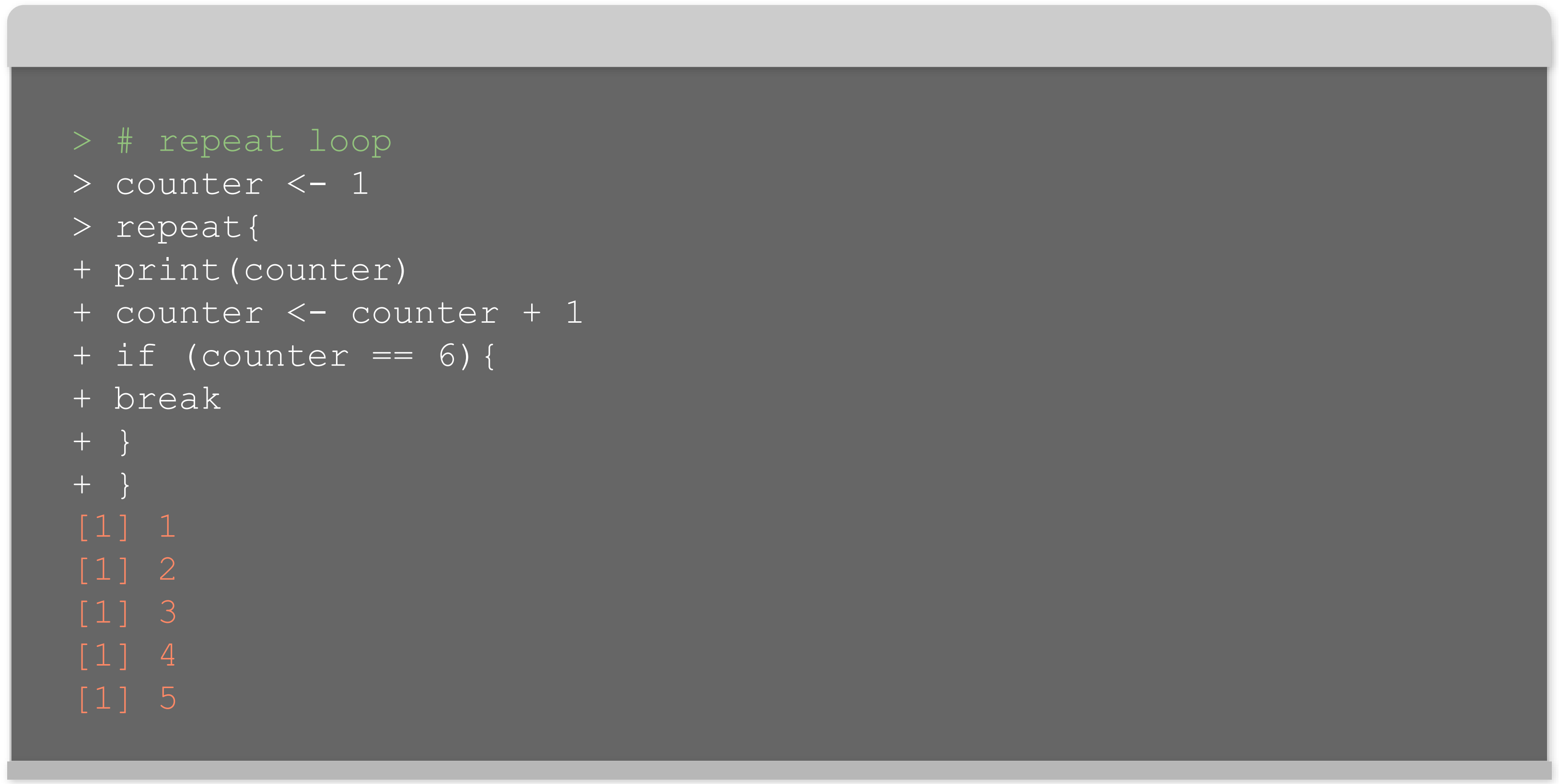


```
> # while loop
> counter <- 1
> while (counter <= 5){
+ print(counter)
+ counter <- counter + 1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Example: Using while loop for testing an expression

Repeat Loop

The repeat loop in R iterates a block of statement until explicitly stopped. To terminate the loop, a *break* statement is required.



```
> # repeat loop
> counter <- 1
> repeat{
+ print(counter)
+ counter <- counter + 1
+ if (counter == 6){
+ break
+ }
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Example of a repeat loop

Loop Control Statements

Break statement

When encountered, the loop is terminated and the control is shifted to the statement immediately after the loop.

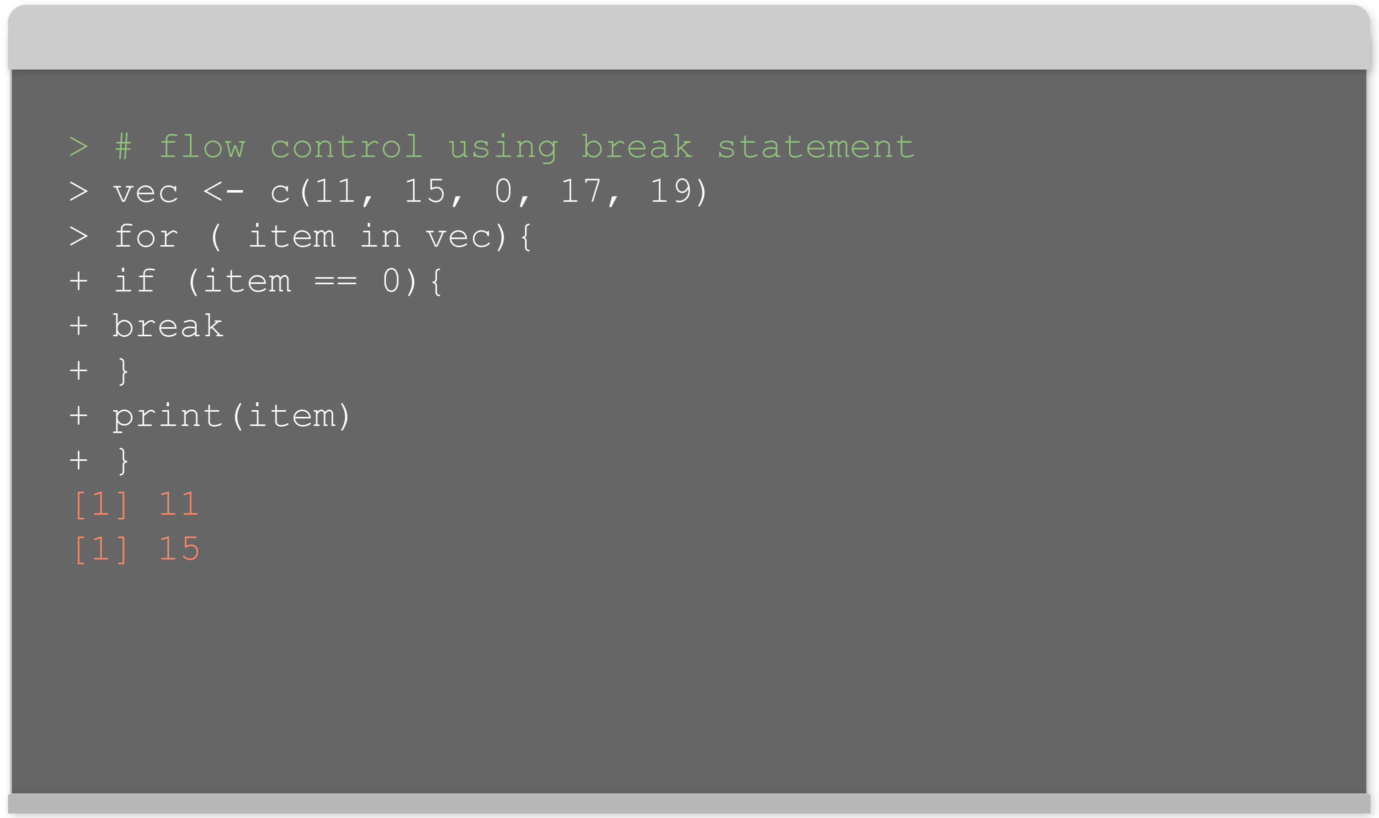
Next statement

When encountered, the remaining part of loop statement is skipped and the control is shifted to the next iteration of the loop.

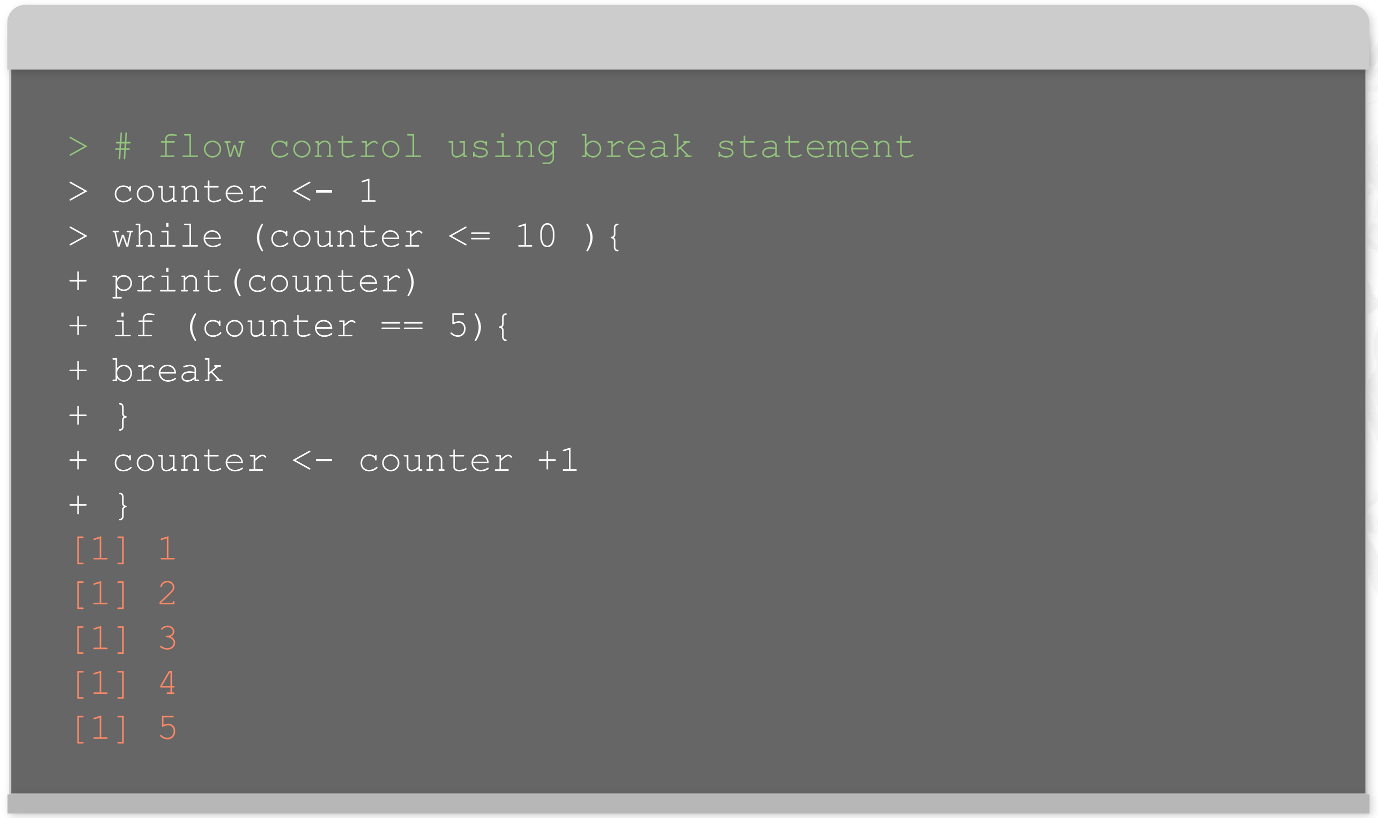
These control statements are generally used in combination with conditional statements.

Loop Control Statements

Break statement examples:



```
> # flow control using break statement
> vec <- c(11, 15, 0, 17, 19)
> for ( item in vec){
+ if (item == 0){
+ break
+ }
+ print(item)
+ }
[1] 11
[1] 15
```



```
> # flow control using break statement
> counter <- 1
> while (counter <= 10 ){
+ print(counter)
+ if (counter == 5){
+ break
+ }
+ counter <- counter +1
+ }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

Loop Control Statements

Next statement examples:

```
> # flow control using next statement
> vec <- c(11, 15, 0, 17, 19)
> for ( item in vec){
+ if (item == 0){
+ next
+ }
+ print(item)
+ }
[1] 11
[1] 15
[1] 17
[1] 19
```

```
> # flow control using next statement
> vec <- c('A', 'B', 'C', 'D', 'E', 'F')
> index <- 0
> while (index < length(vec) ){
+ index <- index +1
+ if (vec[index] == 'C'){
+ next
+ }
+ print(vec[index])
+
+ }
[1] "A"
[1] "B"
[1] "D"
[1] "E"
[1] "F"
```

Functions in R

Functions

Functions are reusable code snippets that may be built in or user defined and designed to perform specific tasks.

1. Arguments are passed as inputs to the function that may be necessary to accomplish the task.

2. A return statement is optional in R, but if present, it returns the control to the main program along with the results, if any.

User-Defined Function

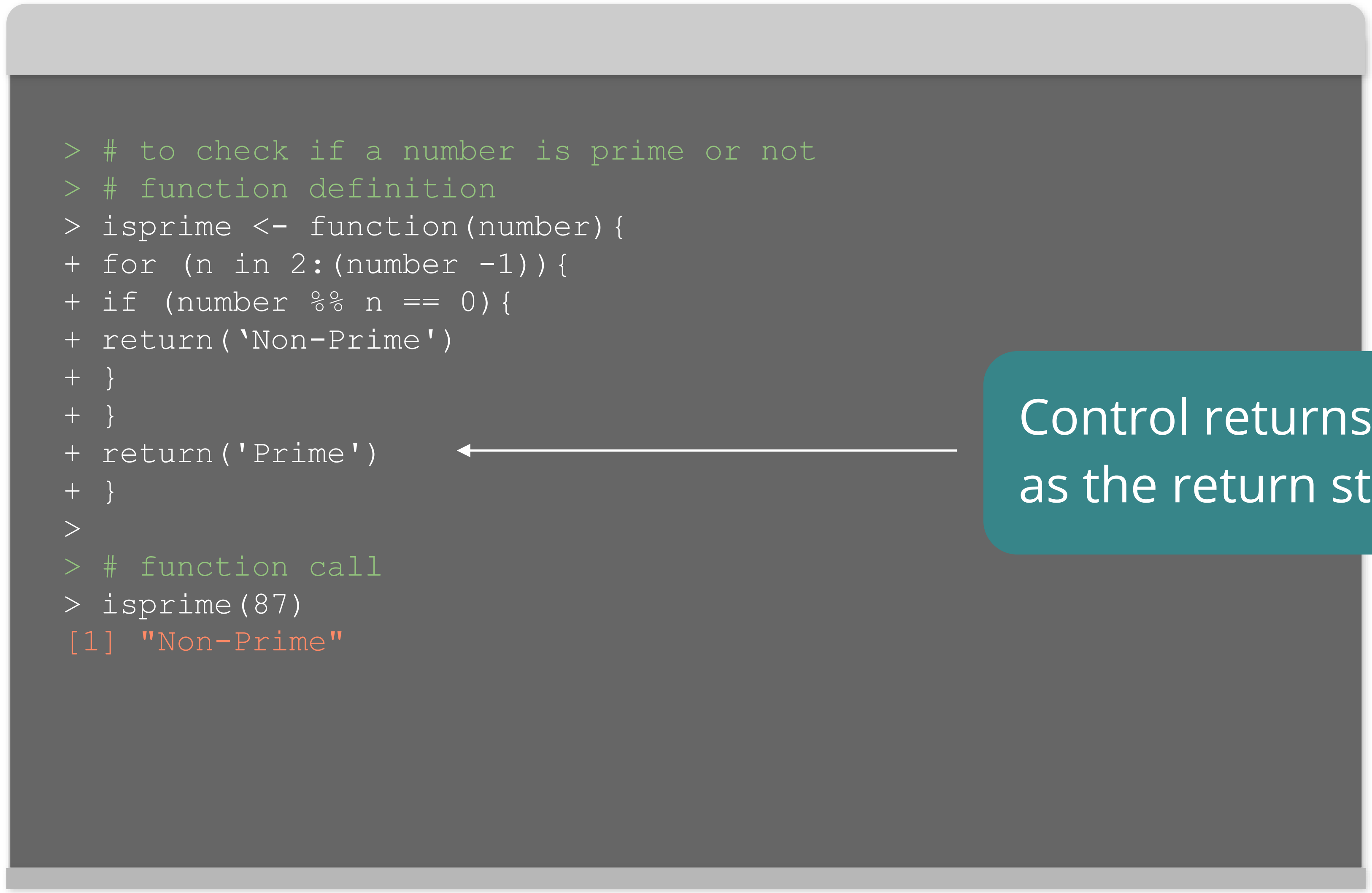
A function has two parts:

Function definition

Function call

User-Defined Function

Shown below are the custom functions that users can create to perform any task.



```
> # to check if a number is prime or not
> # function definition
> isprime <- function(number){
+ for (n in 2:(number -1)){
+ if (number %% n == 0){
+ return(‘Non-Prime')
+ }
+ }
+ return('Prime')
+ }
>
> # function call
> isprime(87)
[1] "Non-Prime"
```

Control returns to the main program as soon as the return statement is encountered.

Generating Factorial

Duration: 5 minutes

Problem Statement: Consider a vector number c(4,5,6,7,8,9). Now, create another vector containing the factorial of each of these numbers by using a user-defined function.

Note: Factorial of a number n is given as:

n! = n * (n-1) *(n-2) * (n-3) …. (n- (n-1))

Note: Please download the solution document from the **Course Resources** section and follow the steps given in the document

Argument Matching

- Arguments in R can be matched by position or by name.
- Positional arguments are required to be mentioned in the same order as in the function definition.
- Keyword arguments are referenced by name and may be in any order.
- Positional and keyword arguments can be mixed in a function call.
- Arguments in a function may be defined with a default value.

Scoping

Scope in programming defines the environment where the variables can be accessed and referenced.

It also defines the range of functionality of any variable or function.

If a value of a variable is to be retrieved by a function, it searches for it in:

- ✓ A global environment
- ✓ The namespaces of each package on the search list

Search list can be retrieved using the search function.

Global Scope vs. Local Scope

Global Scope	Local Scope
Variables and functions defined outside all classes and functions have global scope and are called global variables.	Variables or functions which are defined inside a function block, including argument names, have local scope and are called local variables.
Global scope allows the variables or functions to be accessed from anywhere in the file containing the program, including the functions defined in the same environment.	Local variables can be accessed only inside that function block.

In R, a global and a local variable may have the same names.
The preference would be given to the definition in the environment they are called in.

Scoping Rules in R

R supports lexical scoping (also known as static scoping).

Lexical scoping sets the scope of a variable such that it may be called from within the block of the code it is defined in.

Lexical scoping in R means that while retrieving the value of a free variable, it searches for it in the environment where the function is defined.

Note

Free variables are variables referenced in a function which are neither function arguments nor local variables, that is, these variables aren't created inside the function.

Packages in R

Packages in R

Packages in R are fundamental units of shareable code that are collections of functions and data along with their documentation.

CRAN repository contains 18000+ packages for different applications of data science, machine learning, and deep learning.

Installed Packages in R

The list of installed packages in R can be retrieved by using the function below:

```
installed.packages()
```

Additional packages can be installed by using:

```
install.packages('dplyr')
```

or

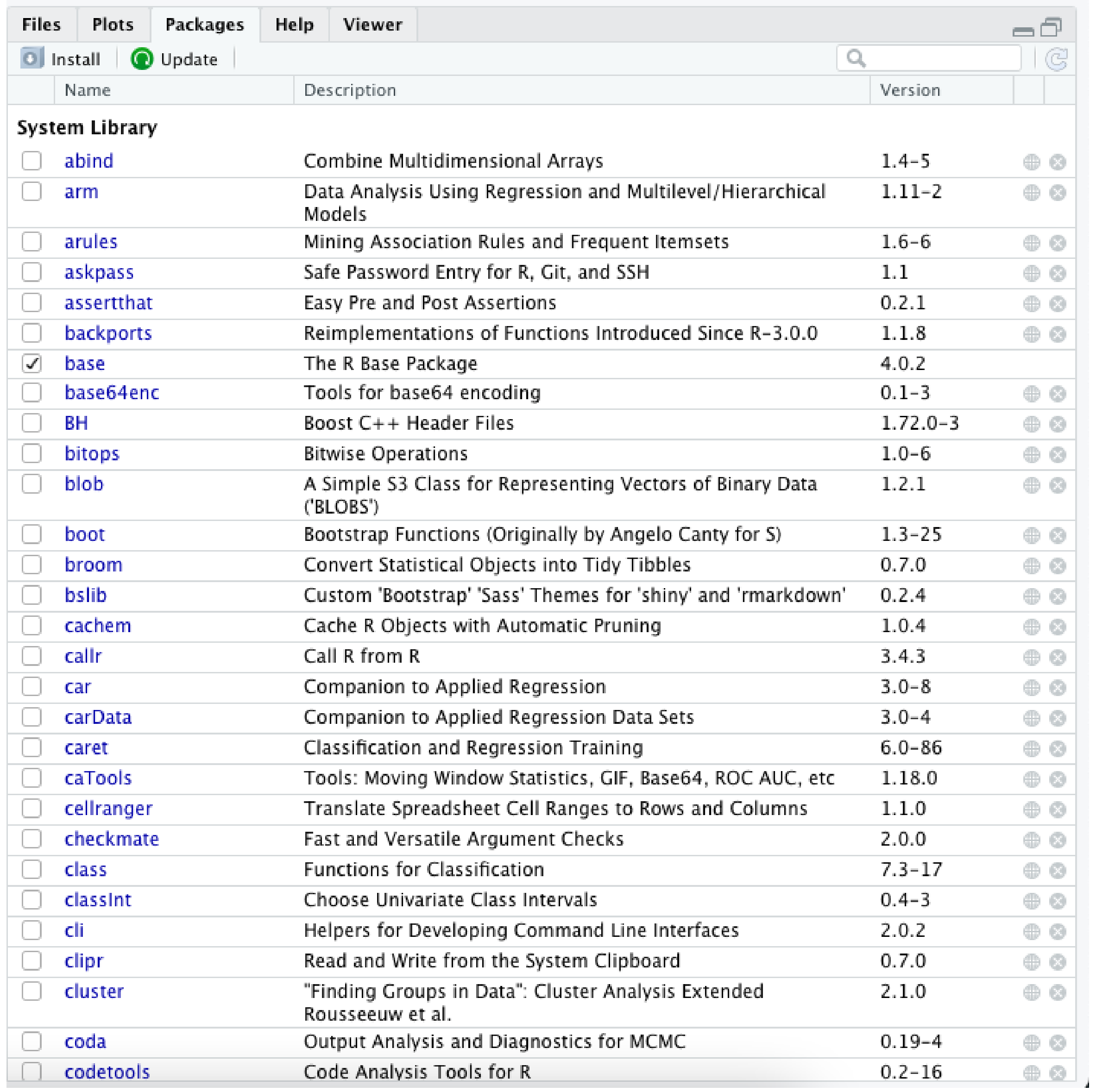
```
install.packages(c('ggplot2','dplyr'))
```

Note

If a given package is already installed, it will be checked for an update.

Installed Packages in R

The list can also be accessed in RStudio under the “Packages” tab of the “miscellaneous” pane.



	Name	Description	Version
System Library			
☐	abind	Combine Multidimensional Arrays	1.4-5
☐	arm	Data Analysis Using Regression and Multilevel/Hierarchical Models	1.11-2
☐	arules	Mining Association Rules and Frequent Itemsets	1.6-6
☐	askpass	Safe Password Entry for R, Git, and SSH	1.1
☐	assertthat	Easy Pre and Post Assertions	0.2.1
☐	backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.8
✓	base	The R Base Package	4.0.2
☐	base64enc	Tools for base64 encoding	0.1-3
☐	BH	Boost C++ Header Files	1.72.0-3
☐	bitops	Bitwise Operations	1.0-6
☐	blob	A Simple S3 Class for Representing Vectors of Binary Data ('BLOBS')	1.2.1
☐	boot	Bootstrap Functions (Originally by Angelo Canty for S)	1.3-25
☐	broom	Convert Statistical Objects into Tidy Tibbles	0.7.0
☐	bslib	Custom 'Bootstrap' 'Sass' Themes for 'shiny' and 'rmarkdown'	0.2.4
☐	cachem	Cache R Objects with Automatic Pruning	1.0.4
☐	callr	Call R from R	3.4.3
☐	car	Companion to Applied Regression	3.0-8
☐	carData	Companion to Applied Regression Data Sets	3.0-4
☐	caret	Classification and Regression Training	6.0-86
☐	caTools	Tools: Moving Window Statistics, GIF, Base64, ROC AUC, etc	1.18.0
☐	cellranger	Translate Spreadsheet Cell Ranges to Rows and Columns	1.1.0
☐	checkmate	Fast and Versatile Argument Checks	2.0.0
☐	class	Functions for Classification	7.3-17
☐	classInt	Choose Univariate Class Intervals	0.4-3
☐	cli	Helpers for Developing Command Line Interfaces	2.0.2
☐	clipr	Read and Write from the System Clipboard	0.7.0
☐	cluster	"Finding Groups in Data": Cluster Analysis Extended Rousseeuw et al.	2.1.0
☐	coda	Output Analysis and Diagnostics for MCMC	0.19-4
☐	codetools	Code Analysis Tools for R	0.2-16

Loading R Packages

- To access functions of any package in R, the package must be loaded and attached in the current R session.
- In any new R session, few base packages are already loaded.
- To load any add-on packages, use the library() function. This will allow one to use any function of the loaded package.

A sample code can be seen here.

```
library(ggplot2)
```

R Packages for Data Analysis

There are multiple other packages that help in performing several tasks for data analysis.

Package	Description
dplyr	Allows data manipulation
ggplot2	Allows one to create elegant data visualizations
caret	Contains classification and regression techniques
tidyr	Allows data cleaning
e1071	Includes miscellaneous functions for classification
forecast	Includes time series forecasting functions
arules	Contains association rules mining functions
readr	Allows one to read tabular data
stringr	Provides functions for string operations

Built-in Functions In R

Built–in Functions In R

Built-in functions are predefined functions in the programming framework.

There are several types of built-in functions in R.

- Mathematical Functions
- String Functions
- Statistical Functions
- Miscellaneous Functions

Mathematical Functions

Function	Description
abs(x)	Returns absolute value of any numeric value
sqrt(x)	Returns square root of a number
floor(x)	Returns the closest integer less than or equal to the value of x
ceiling(x)	Returns the closest integer greater than the value of x
round(value, digits)	Returns the value rounded to the specified number of decimal values
sin(x) cos(x) tan(x)..	Performs trigonometric functions
log(x)	Returns natural log of x
sum(...)	Returns the sum of all values passed
cumsum(...)	Returns the vector of cumulative sum of the values passed

String Functions

Function	Description
nchar(x)	Returns the count of characters
toupper(x) and tolower(x)	Changes the case of a string to uppercase or lowercase
substr (x, start, stop)	Returns a part of a string
paste(…)	Concatenates multiple string objects
strsplit(x, split)	Splits a string or each string in a vector at given character
sub(pattern, replacement, x)	Replaces a first occurrence of a substring with another
format()	Formats number and strings to a specific style
grep(pattern, x)	Searches for a pattern in a vector of strings and returns the matching indices.

Basic Statistical Functions

Function	Description
mean(x)	Returns the average of the object
median(x)	Returns the median value of the given object
sd(x)	Returns the standard deviation of the object
var(x)	Returns the variance of the object
quantile(x)	Returns the quartile values of x
range(x)	Returns the maximum and minimum value in the object x
scale(x)	Returns the standardized scores of the object x
summary(x)	Shows the statistical summary of the object x and given mean, median, minimum value, maximum value, and quantile values

Miscellaneous Functions

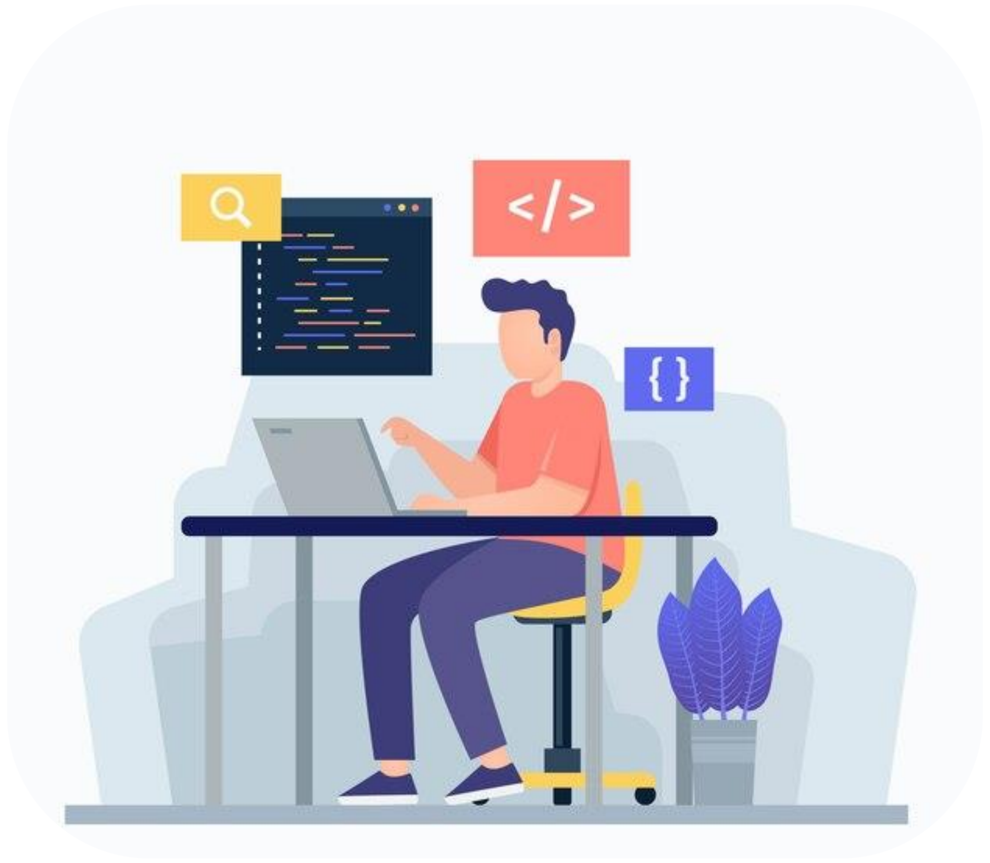
Function	Description
seq()	Generates a sequence with the given start and end
rep()	Repeats a vector for a given no. of times
cut()	Divides a continuous variable in factor with given levels

Apply Family Functions

Apply Functions

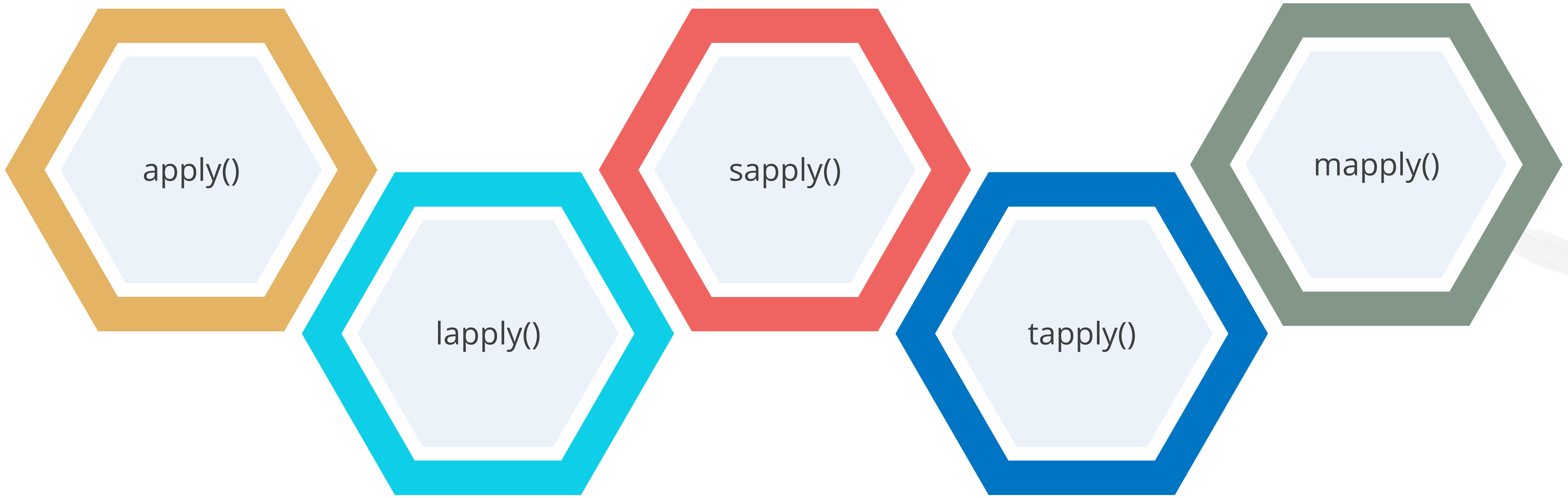
It helps to apply any function across a matrix array list or dataframe.

Apply family functions can be an alternative for loops.



Apply Functions

Major apply functions include:



The apply() Function

It takes a matrix or an array as input to apply the given function along the given margin and returns a vector containing the results.

Syntax:

```
apply(X , FUN , MARGIN)
```

The apply() Function

Example: Finding average of values for each row in a matrix.

```
> # Code to find mean of scores for each row of matrix using apply
function
> score <- matrix(seq(10, 65,5), nrow = 3, ncol = 4)
> score
 [,1] [,2] [,3] [,4]
[1,] 10 25 40 55
[2,] 15 30 45 60
[3,] 20 35 50 65
> apply(score, MARGIN = 1, FUN = mean)
[1] 32.5 37.5 42.5
```

The lapply() Function

lapply() applies any given function to each element of a vector or a list and returns a list.

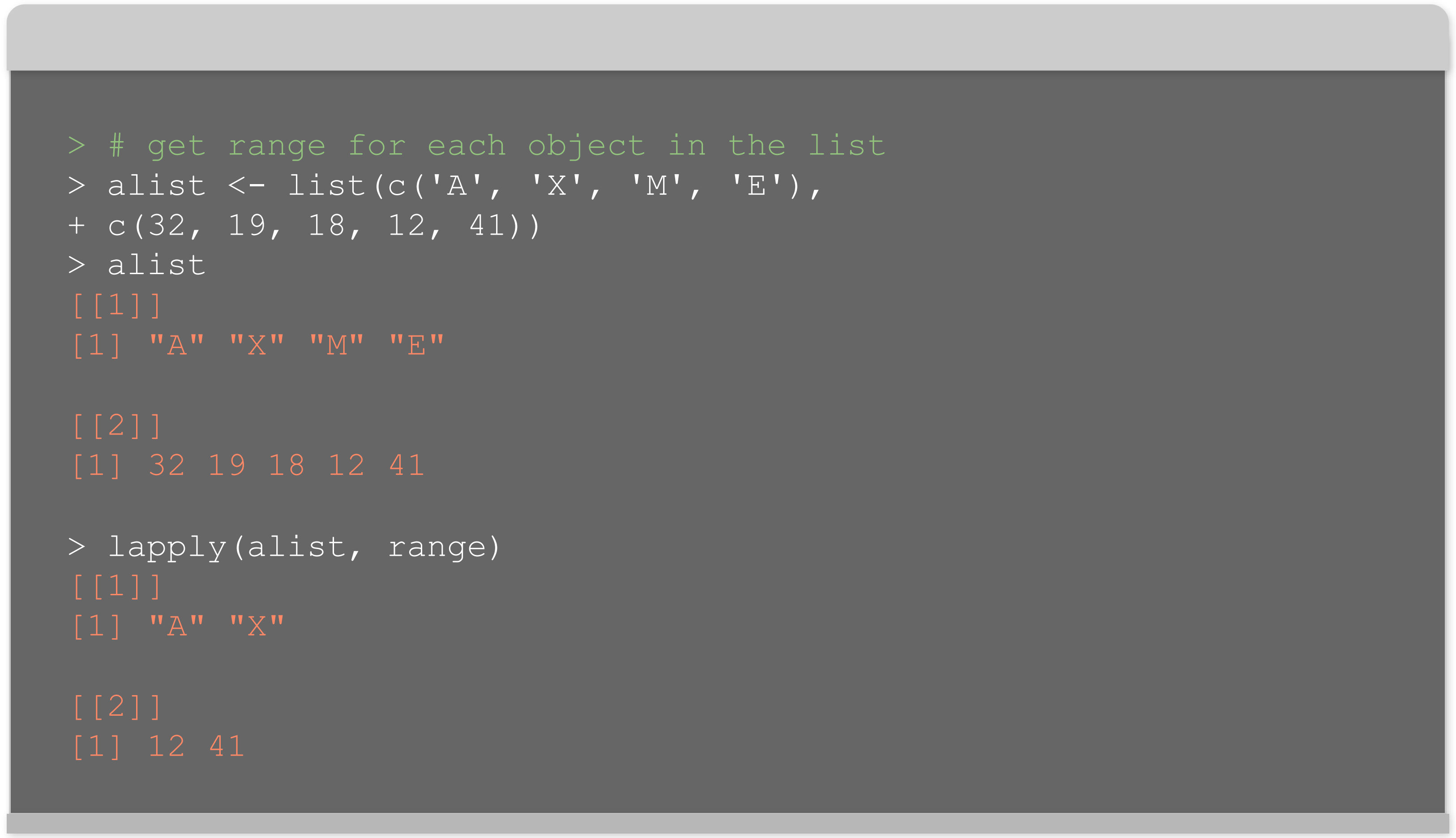
If any other object is passed, it is coerced to list.

Syntax:

```
lapply(x, FUN))
```

The lapply() Function

Example: Getting the range value of every object in a list



```
> # get range for each object in the list
> alist <- list(c('A', 'X', 'M', 'E'),
+ c(32, 19, 18, 12, 41))
> alist
[[1]]
[1] "A" "X" "M" "E"

[[2]]
[1] 32 19 18 12 41

> lapply(alist, range)
[[1]]
[1] "A" "X"

[[2]]
[1] 12 41
```

The sapply() Function

sapply() is a user-friendly version of lapply(). It simplifies the output of lapply() by coercing it into a simpler data structure.

If the output of lapply() is a list where each element has a length of 1, then it is converted to a vector in sapply().

If the output of lapply() is a list where each element has the same length, then it is converted to a matrix in sapply(). Else, the output is a list.

Syntax:

```
sapply(x, FUN))
```

sapply() vs. lapply()

Examples to compare outputs of sapply() and lapply():

```
> # sapply
> alist <- list(c(34.8, 90.1, 18.2, 15.5),
+ c(50, 45, 30, 25, 90))
> alist
[[1]]
[1] 34.8 90.1 18.2 15.5

[[2]]
[1] 50 45 30 25 90

> lapply(alist, sum)
[[1]]
[1] 158.6

[[2]]
[1] 240

> sapply(alist, sum)
[1] 158.6 240.0
```

```
> alist <- list(c(34.8, 90.1, 18.2, 15.5),
+ c('A', 'X', 'M', 'E'))
> alist
[[1]]
[1] 34.8 90.1 18.2 15.5

[[2]]
[1] "A" "X" "M" "E"

> lapply(alist, range)
[[1]]
[1] 15.5 90.1

[[2]]
[1] "A" "X"

> sapply(alist, range)
 [,1] [,2]
[1,] "15.5" "A"
[2,] "90.1" "X"
```

The tapply() Function

tapply() applies a function to a vector for each group in the factor vector.

Syntax:

```
tapply(X , INDEX , FUN)
```

The tapply() Function

Example: Computing average scores by gender.

```
> # get mean of scores by gender
> df <- data.frame(score = c(95, 78, 90, 13, 74, 83, 81, 34, 20, 15),
+ gender = c('F', 'M', 'F', 'F', 'F', 'M', 'M', 'F', 'M', 'M'))
> df
 score gender
1 95 F
2 78 M
3 90 F
4 13 F
5 74 F
6 83 M
7 81 M
8 34 F
9 20 M
10 15 M
> tapply(df$score,df$gender,mean )
 F M
61.2 55.4
```

The mapply() Function

mapply() is a multivariate version of sapply().

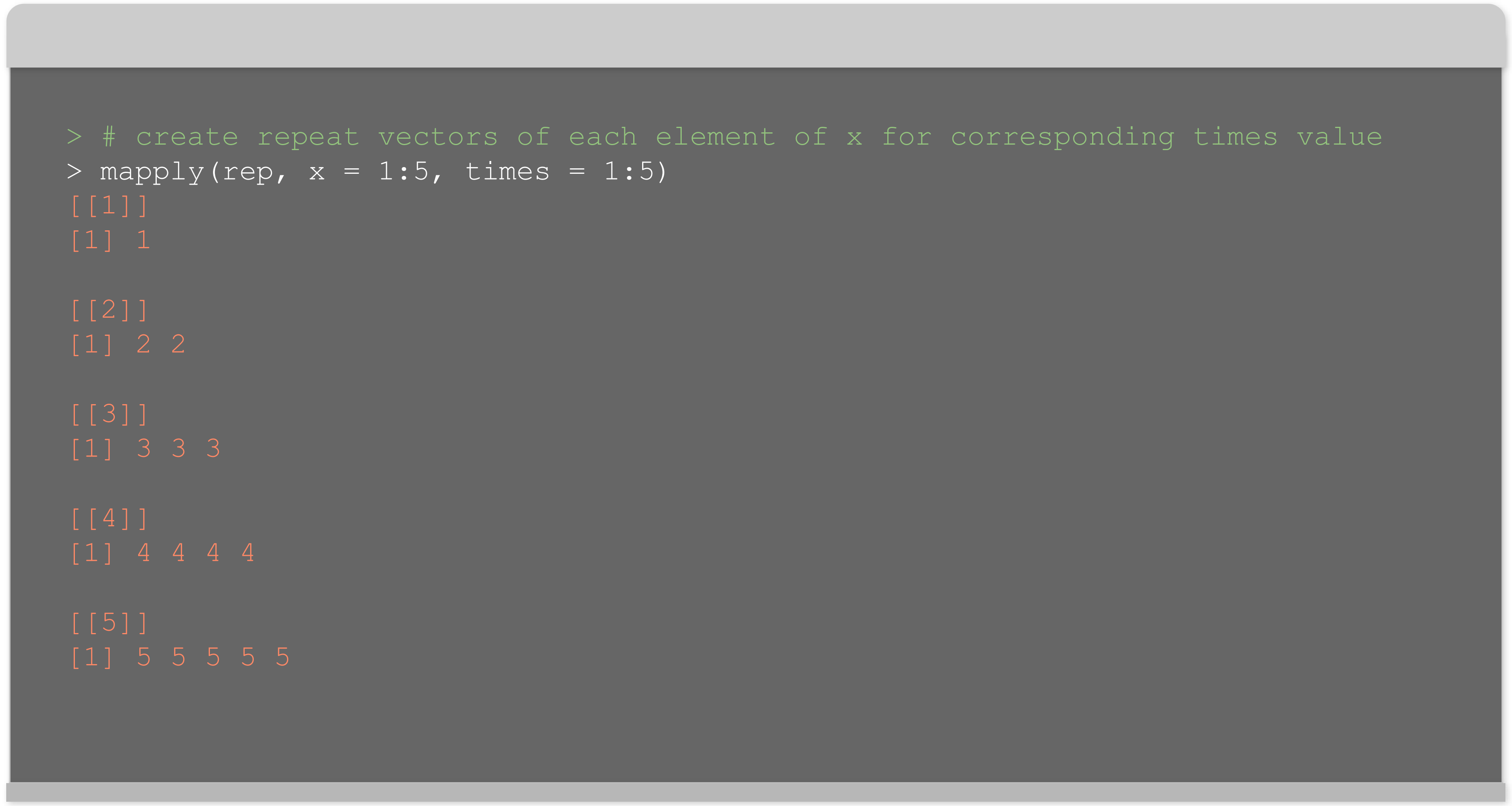
It enables one to pass values to arguments of a function in vector format, which do not, by syntax, take vector arguments.

Syntax:

```
mapply(FUN ,… )
```

The mapply() Function

Example: Creating repeat vectors of each element of x for the corresponding times value.



```
> # create repeat vectors of each element of x for corresponding times value
> mapply(rep, x = 1:5, times = 1:5)
[[1]]
[1] 1

[[2]]
[1] 2 2

[[3]]
[1] 3 3 3

[[4]]
[1] 4 4 4 4

[[5]]
[1] 5 5 5 5 5
```

Working With Dates In R

Dates in R

1. Handling date and time are important part of real-world data analysis.

2. R provides multiple functions to deal with dates.

3. Date objects are stored as integers in R.

R Functions For Date

Function	Description
as.Date()	Converts a character representation of date to a date object
Sys.Date()	Returns the present system date
difftime()	Returns difference between two date objects
weekdays()	Returns the day of the week for the given date
months()	Returns the name of month for the given date
quarters()	Returns the quarter number for the given date
julian()	Returns the difference in no. of days since an origin date; default is 1st January 1970

These functions are available in the base package. However, there are specialized packages that have functions to handle dates.

Lubridate Package For Dates

Lubridate package provides easy-to-use tools to handle and manipulate date objects.

It provides the following functions:

- Date and time parsing
- Extracting date components
- Computing time intervals
- Performing arithmetic operations with date and time

Functions in Lubridate Package

Function	Description
ymd(x), dmy(x), myd(x), etc.	Convert the character or numeric representation of date to datetime
date(x), year(x), month(x), hour(x), etc.	Extract and assign components of the datetime object
quarter(x)	Returns quarter of the year to which the date belongs
semester(x)	Returns semester of the year to which the date belongs
with_tz() and force_tz()	Are helper functions for handling time zones
years(x), months(x), weeks(x), etc.	Create a period with the same time unit
dyears(x), dmonths(x), and dweeks(x)	Create duration with the name of period
time_length(x)	Computes exact time span for a duration, period, or interval

R Markdown

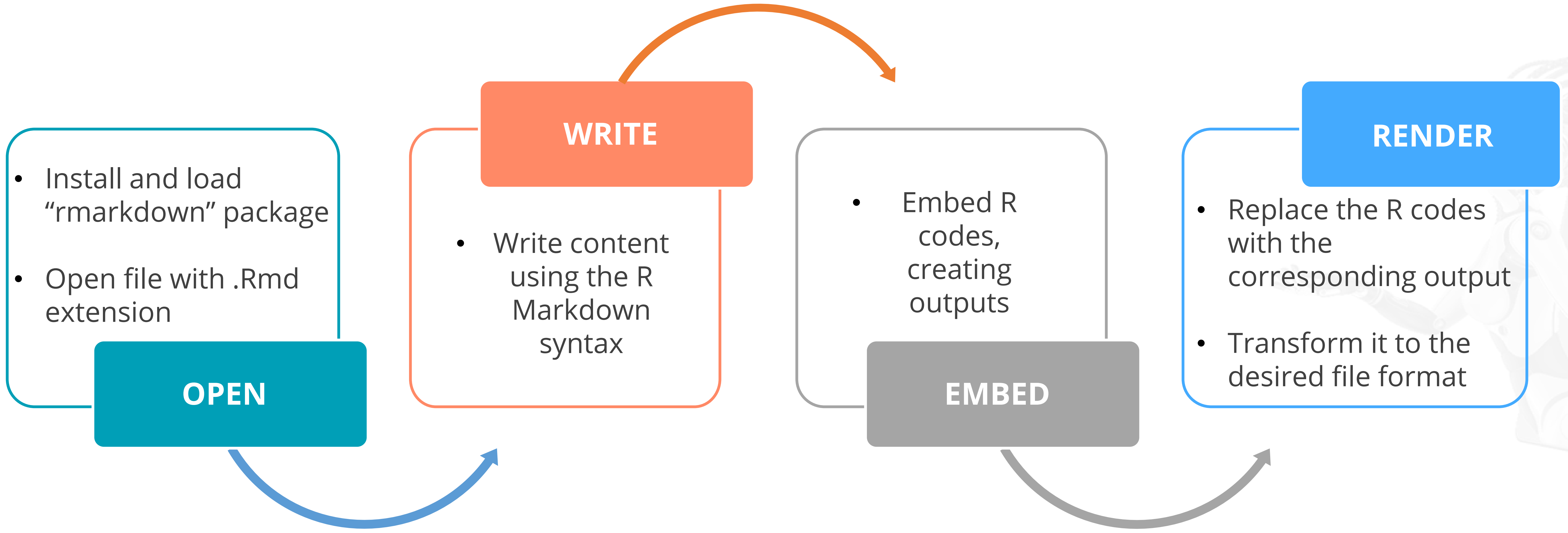
R Markdown

Markdown is a simplified markup language used to create formatted text from plain text files.

R Markdown allows the creation of a neat, organized, and documented record of analysis in the same file where the code script is written.

R Markdown documents allow the users to save and execute scripts as well as generate reports by including explanations and insights in the same file.

R Markdown: Workflow



OPEN

- Install and load "rmarkdown" package
- Open file with .Rmd extension

WRITE

- Write content using the R Markdown syntax

EMBED

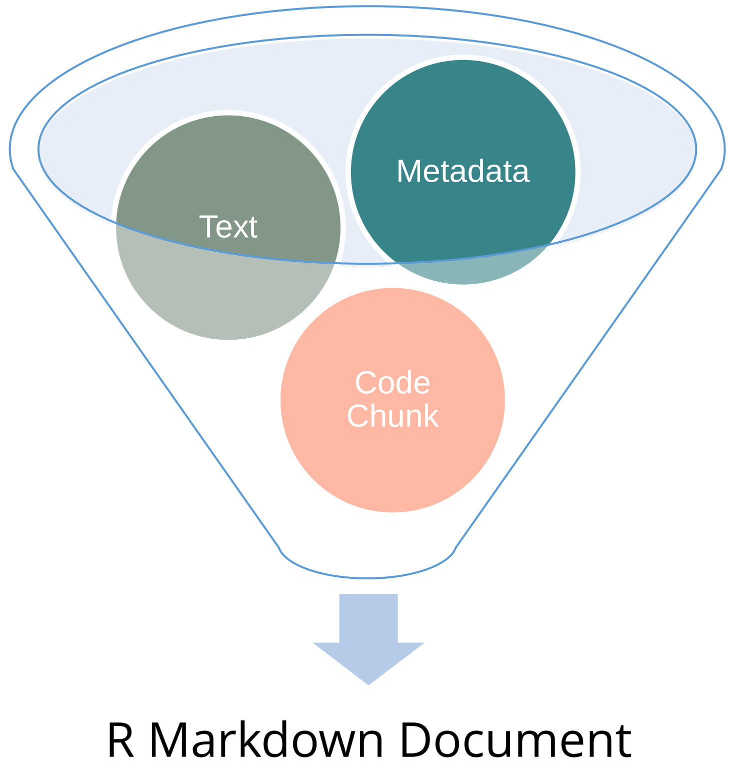
- Embed R codes, creating outputs

RENDER

- Replace the R codes with the corresponding output
- Transform it to the desired file format

Components of R Markdown Document

There are three basic components of an R Markdown document.



Metadata	Specifies information about the data
Text	Body of the document
Code Chunk	Code generating outputs in R

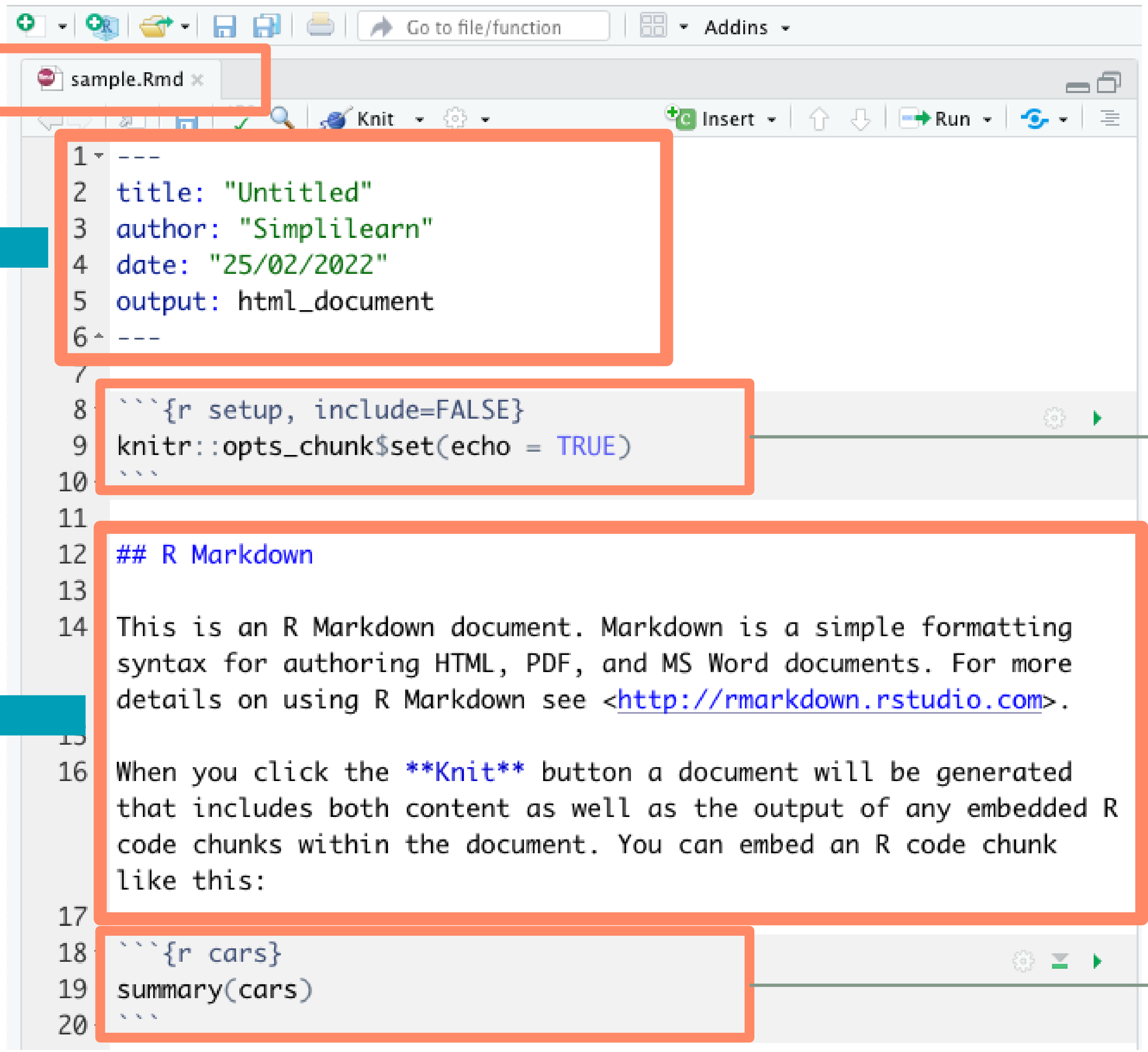
Components of R Markdown Document

File name

Metadata or YAML header

R Markdown text document

Code chunk



~~~
---
title: "Untitled"
author: "Simplilearn"
date: "25/02/2022"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

R Markdown

This is an R Markdown document. Markdown is a simple formatting
syntax for authoring HTML, PDF, and MS Word documents. For more
details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated
that includes both content as well as the output of any embedded R
code chunks within the document. You can embed an R code chunk
like this:

```{r cars}
summary(cars)
```
```
```

Code Chunk Options

Code chunk options are an additional set of rules for formatting the final Markdown document.

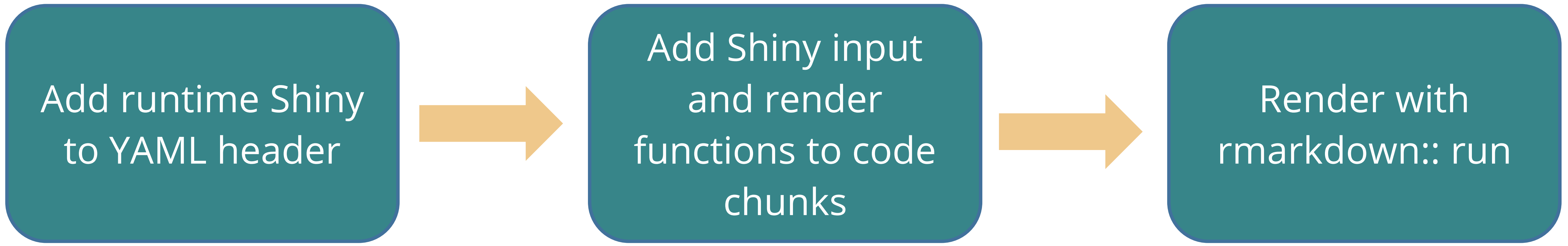
Option	Description
echo = TRUE	Displays the code alongside the output
eval = TRUE	Executes the code in code chunk
include = TRUE	Adds code chunk in the final output
warning = TRUE	Displays warning messages with the output
error = TRUE	Displays error messages with the output
results = “hide” or “asis” or “hold”	Specifies the treatment of results: - hide: displays no results - asis: displays result without formatting - hold: compiles results only at the end of the chunk

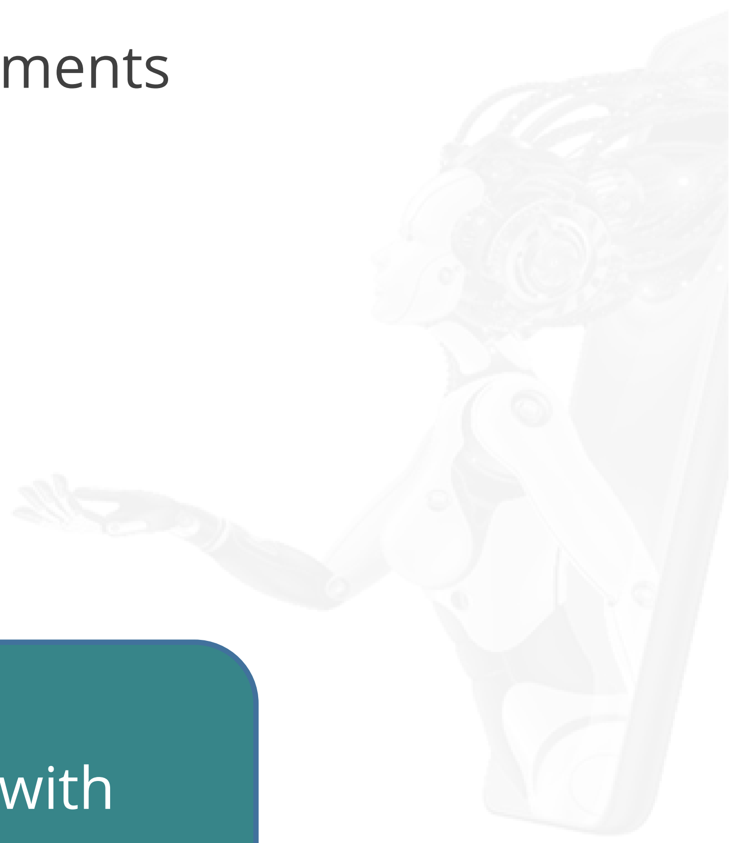
R Shiny

Shiny is a powerful package in R that allows one to build interactive web applications based on R.

R Markdown files containing Shiny widgets and outputs are the interactive documents that can be launched as a web application with a click.

To turn Markdown reports to interactive Shiny documents:


Add runtime Shiny to YAML header
Add Shiny input and render functions to code chunks
Render with rmarkdown:: run



Creating a Markdown file

Duration: 10 minutes

Problem Statement: Create a markdown file with a YAML header mentioning title as ‘test’, author as ‘your name’, time, and a summary of the mtcars dataset available in the R environment.

The markdown output should be in a PPTX format without code. Set options to not display the code error or warning messages.

Note: Please download the data set and the solution document from the Course Resources section and follow the steps given in the document.

Key Takeaways

- Decision-making is achieved by evaluation of conditional expression to decide the flow control.
- If-else, nested if-else, and if-else ladder are statements that help with decision-making in R.
- To execute a block of statements, multiple loops can be used. R provides for loop, while loop, and repeat loop.

Key Takeaways

- The flow control of these loops can be altered by using break and next statements.

- R provides a large set of predefined functions to perform tasks that are bundled together as packages and are stored in libraries. However, custom functions can also be created.

- R Markdown is a neat way of creating code and documenting a data analysis project.

Knowledge Check

Knowledge Check

1

Which of the following statements is true?

A. Positional and keyword arguments cannot be mixed in a function call.

B. Dates are stored as integers in R.

C. The output of sapply() is never a list.

D. Additional packages cannot be installed in R.

Knowledge Check
1

Which of the following statements is true?

A. Positional and keyword arguments cannot be mixed in a function call.

B. Dates are stored as integers in R.

C. The output of sapply() is never a list.

D. Additional packages cannot be installed in R.

The correct answer is **B**

Dates are stored as integers in R.

Knowledge Check

2

Consider the following code:

sapply(c(“A”, “K”, “M”, “P, “C”, “R”), c(45, 87, 90, 12, 34, 56), range)

What will be the output type?

A. Character vector

B. Matrix

C. List

D. Numeric vector

Knowledge Check

2

Consider the following code:

sapply(c(“A”, “K”, “M”, “P, “C”, “R”), c(45, 87, 90, 12, 34, 56), range)

What will be the output type?

A. Character vector

B. Matrix

C. List

D. Numeric vector

The correct answer is **B**

The range function returns min and max values for each vector. Hence, the result will be a type character matrix.

Knowledge Check

3

An R Markdown document can be rendered as:

A. A slideshow

B. A PDF

C. A web application

D. All of the above

Knowledge Check

3

An R Markdown document can be rendered as:

A. A slideshow

B. A PDF

C. A web application

D. All of the above

The correct answer is **D**

An R Markdown document can be rendered as a slideshow, PDF, or web application.