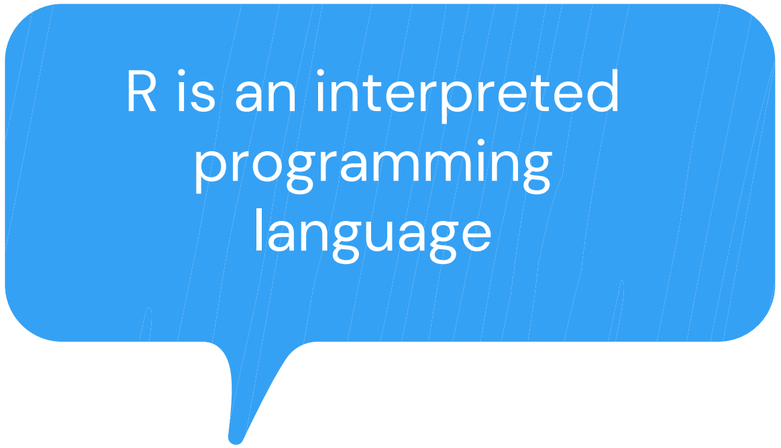


R Programming

Created by Pankaj Chouhan

@codeswithpankaj.com

What is R Programming?

R is a programming language and also a software environment for statistical computing and data analysis. R was developed by Ross Ihaka and Robert Gentleman at the university of Auckland, New Zealand. R is an open-source programming language and it is available on widely used platforms e.g. Windows, Linux, and Mac. It generally comes with a command-line interface and provides a vast list of packages for performing tasks. R is an interpreted language that supports both procedural programming and object-oriented programming.

Getting Started With R

Write and execute R code in a Jupyter Notebook.

1 Install R and Jupyter:

```
# Install Jupyter Notebook using pip
pip install notebook
```

2 Install R Kernel for Jupyter:

```
install.packages("IRkernel")
IRkernel::installspec()
```

3 Launch Jupyter Notebook:

```
jupyter notebook
```

4 Create a New Notebook:

5 Write and Execute R Code:

```
# This is an R code cell.
x <- c(1, 2, 3, 4, 5)
mean_x <- mean(x)
mean_x
```

Ready to start?

R Comments

R, comments are used to add explanatory or descriptive notes within your code. Comments are ignored by the R interpreter and are not executed as part of the program.

1 Single-line Comments:

Single-line Comments Example :

```
# This is a single-line comment
x <- 10  # This comment explains the
purpose of the following code
```

2 Multi-line Comments:

Multi-line Comments Example 1 :

```
'''
Codes With Pankaj
a multi-line comment in R.
'''
```

Multi-line Comments Example 2 :

```
"""
This is yet another way to create p4n
a multi-line comment in R.
"""
```

1 Commenting Out Code

Commenting Out Code

```
# This code is temporarily disabled
# x <- 5
# y <- 10
```

2 Inline Comments:

Inline Comments:

```
result <- x + y  # Calculate the sum
of x and y
```

Ready to start ?

R Variables and Constants

What Are Variables?

Variables are used to store and manipulate data. Variables can hold various types of data, such as numbers, text, logical values, and more. Here are some key points about variables in R:

Variable Names:

- Variable names in R are case-sensitive, meaning that "myVariable" and "myvariable" would be treated as distinct variables.
- Variable names must start with a letter (a-z or A-Z) or a period (.), followed by letters, numbers, or periods.

Variables

1 Declaration

Declaration

```
x <- 5
#Assigns the integer 5 to the variable 'x'
name <- "John"
#Assigns the string "John" to the variable 'name'
```

2 Data Types

Data Types

```
age <- 30              # Integer
temperature <- 98.6    # Double
name <- "Alice"        # String
is_student <- TRUE     # Logical (TRUE or FALSE)
```

Constants

There isn't a specific keyword or syntax for declaring constants like some other programming languages. However, you can achieve the concept of constants by convention. By naming a variable in ALL_CAPS and not modifying its value throughout your code, you can indicate that it's intended to be a constant.

1 Constants

Constants

```
PI <- 3.14159
```

Ready to start ?

Data Types

Unlock the world of coding

Data Types

data types are essential for categorizing and processing different types of data. R has a variety of built-in data types to handle various data structures and values. Here are some of the primary data types in R:

A variable can store different types of values such as numbers, characters etc. These different types of data that we can use in our code are called data types.

Types of DataType

1. Numeric

Numeric

```
x <- 42       # Integer
y <- 3.1415   # Double (floating-point)
```

2. Character

Character

```
name <- "p4n"
message <- 'p4n, Hello!'
```

3. Logical

Logical

```
is_student <- TRUE
is_adult <- FALSE
```

Types of DataType

4 Integer

Integer

```
count <- as.integer(10)
```

5 Complex

Complex

```
complex_num <- 3 + 2i
```

6 Factor

Factor

```
gender <- factor(c("Male", "Female", "Male", "Female"))
```

Ready to start ?

R Output and Input

Types of DataType

1. print() Function:

print() Function:

```
x <- 42
print(x)
```

2. cat() Function:

cat() Function:

```
a <- 10
b <- 20
cat("The sum of a and b is:", a + b, "\n")
```

3. paste() and paste0() Functions:

paste() and paste0() Functions:

```
# code to illustrate the paste0() function
paste0("codes","with","pankaj")
paste(1,'two',3,'four',5,'six')
```

Take user input in R

1 readline() function

readline() function

```
# Prompt the user for input
user_input <- readline("Enter something: ")

# Display the user's input
cat("You entered:", user_input, "\n")
```

In this code:

- readline("Enter something: ") is used to prompt the user for input with the message "Enter something: ". The user enters text, and it's stored in the user_input variable.

- cat("You entered:", user_input, "\n") is used to display the user's input along with a message.

Ready to start ?

R Operator

Operators

1 Arithmetic Operators:

1. + (Addition)
2. - (Subtraction)
3. * (Multiplication)
4. ^ (Exponentiation)
5. %% (Modulus)
6. %/% (Integer Division)

Arithmetic Operators

```
x <- 10 # Numeric Variables
y <- 3  # Numeric Variables

sum_result <- x + y # Addition

difference_result <- x - y # Subtraction

product_result <- x * y # Multiplication

quotient_result <- x / y # Division

exponent_result <- x^y # Exponentiation

remainder_result <- x %% y # Modulus (Remainder)

# Integer Division (Quotient)
integer_division_result <- x %/% y
```

Operators

2 Comparison Operators:

1. == (Equal to)
2. != or <> (Not equal to)
3. < (Less than)
4. > (Greater than)
5. <= (Less than or equal to)
6. >= (Greater than or equal to)

Comparison Operators:

```
a <- 5
b <- 10
is_equal <- a == b          # FALSE
is_not_equal <- a != b      # TRUE
is_less_than <- a < b       # TRUE
is_greater_than <- a > b    # FALSE
```

Operators

3 Logical Operators

1. ! (Logical NOT)
2. & (Logical AND)
3. | (Logical OR)
4. && (Short-circuit AND)
5. || (Short-circuit OR)

Logical Operators

```
is_true <- TRUE
is_false <- FALSE
logical_not <- !is_true      # FALSE
logical_and <- is_true & is_false  # FALSE
logical_or <- is_true | is_false   # TRUE
```

Operators

4 Assignment Operators

1. <- or = Assigns

Assignment Operators

```
x <- 10
y = 3
```

5 Miscellaneous Operators

1. %in% :

Checks if an element belongs to a vector or list

2. :

Creates a sequence of numbers.

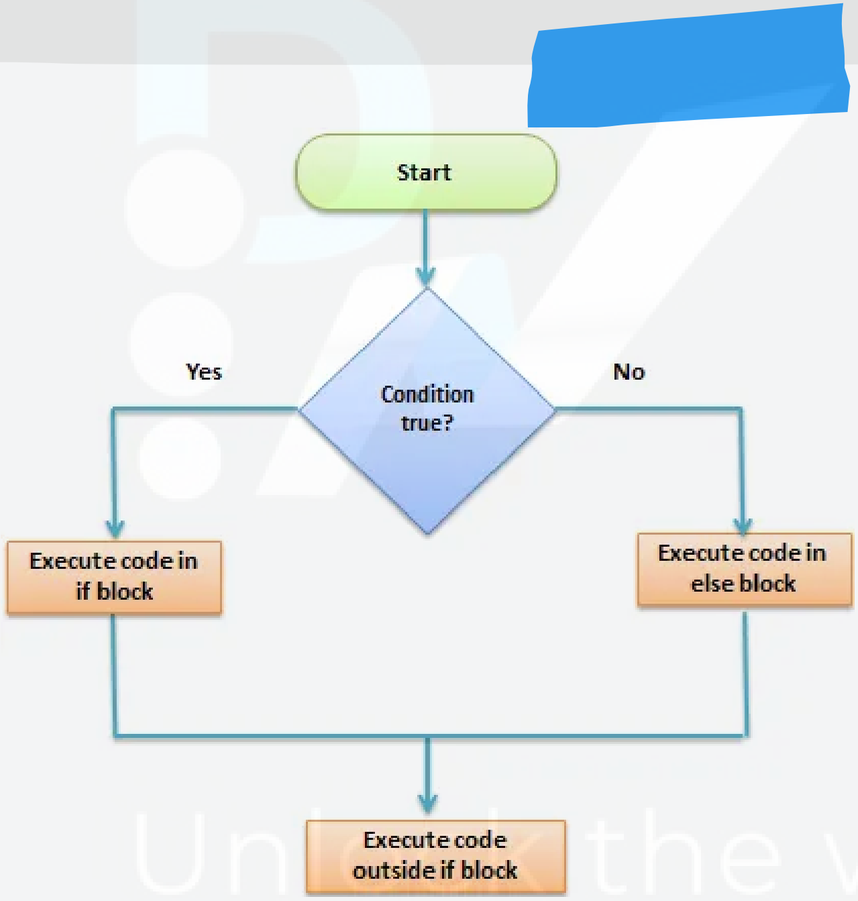
Miscellaneous Operators

```
is_present <- 5 %in% c(1, 2, 3, 4, 5)  # TRUE
seq_numbers <- 1:5  # Creates a sequence 1, 2, 3, 4, 5
```

Ready to start ?

R if..else

R if..else



R if..else

```
if (condition) {
  # Code to execute if the condition is TRUE
} else {
  # Code to execute if the condition is FALSE
}
```

The if...else statement is used for conditional execution of code. It allows you to specify a condition, and based on whether that condition is TRUE or FALSE, different blocks of code will be executed. The basic syntax of the if...else statement in R is as follows

if...else

1 Example 1:

Example 1: Simple if...else

```
x <- 10if (x > 5) {
  cat("x is greater than 5.\n")
} else {
  cat("x is not greater than 5.\n")
}
```

2 Example 2

Example 2: Nested if...else

```
y <- 3if (y > 5) {
  cat("y is greater than 5.\n")
} else if (y == 5) {
  cat("y is equal to 5.\n")
} else {
  cat("y is less than 5.\n")
}
```

R ifelse() Function

ifelse() function is a vectorized way to perform conditional operations. It allows you to apply a specified condition to each element of a vector or data frame and return a new vector or data frame based on the condition. The basic syntax of the ifelse() function is as follows:

- **test_expression:** The condition to be tested. It can be a logical vector or expression.
- **yes_expression:** The value to be returned when the condition is TRUE.
- **no_expression:** The value to be returned when the condition is FALSE.

R ifelse() Function

```
ifelse(test_expression, yes_expression, no_expression)
```

ifelse() function

1 Example

ifelse() function

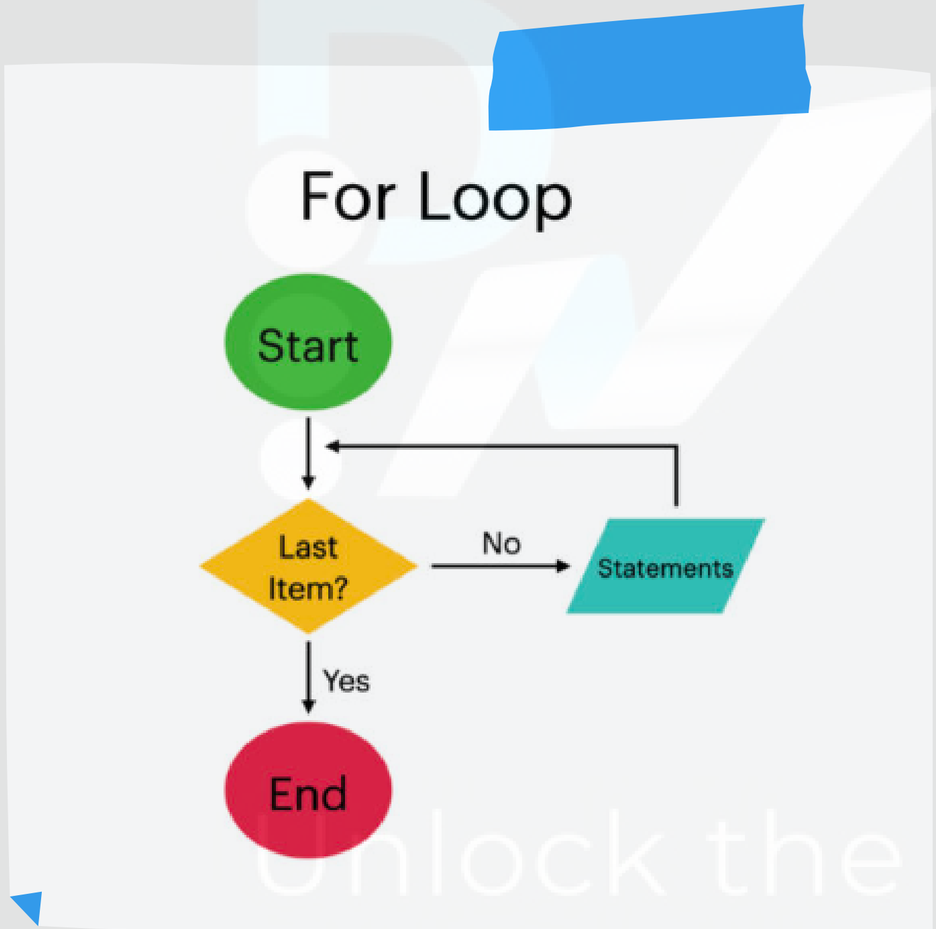
```
# Example: Using ifelse() to categorize exam scores
scores <- c(78, 92, 64, 88, 75)
grades <- ifelse(scores >= 90, "A", ifelse(scores >= 80, "B",
ifelse(scores >= 70, "C", "D")))

# Print the result
cat("Grades:", grades, "\n")
```

Ready to start ?

R for Loop

R for Loop



R for Loop

```
for (variable in sequence) {
  # Code to be executed for each element in the sequence
}
```

For loop is used to repeatedly execute a block of code a specified number of times or iterate through elements in a sequence (such as a vector or a list). It provides a way to create iterative processes in your program.

for loop

1 Example

Using a for loop to iterate through numbers from 1 to 5

```
# Example: Using a for loop to iterate through numbers from 1 to 5
for (i in 1:5) {
  cat("Count:", i, "\n")
}
```

2 Example

Using a for loop to iterate through elements in a vector

```
# Example: Using a for loop to iterate through elements in a vector
fruits <- c("apple", "banana", "cherry", "date")

for (fruit in fruits) {
  cat("Fruit:", fruit, "\n")
}
```

for loop

3 Example

Using seq_along() in a for loop

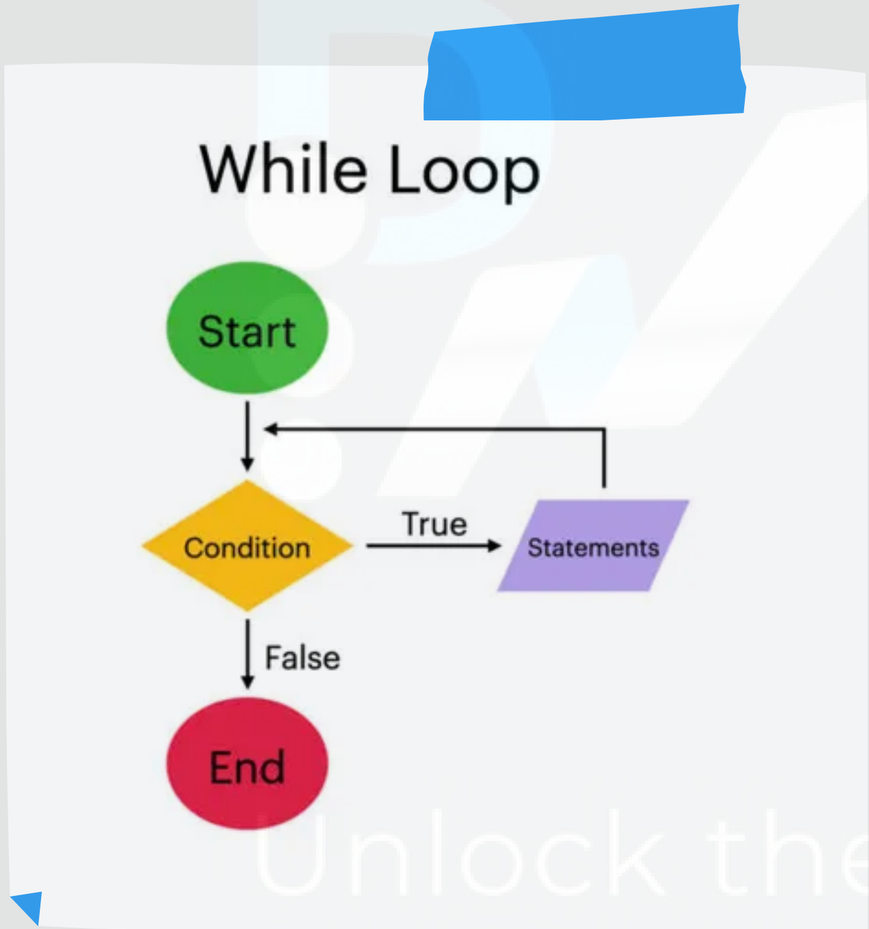
```
# Example: Using seq_along() in a for loop
fruits <- c("apple", "banana", "cherry", "date")

for (i in seq_along(fruits)) {
  cat("Index:", i, "Fruit:", fruits[i], "\n")
}
```

Ready to start ?

R While Loop

R While Loop



R while Loop

```
while (condition) {
   # Code to be executed while the condition
is TRUE
   # The condition should eventually become
FALSE to exit the loop
}
```

While loop is used to repeatedly execute a block of code as long as a specified condition remains TRUE. It provides a way to create iterative processes in your program

While loop

1 Example

Using a while loop to count from 1 to 5

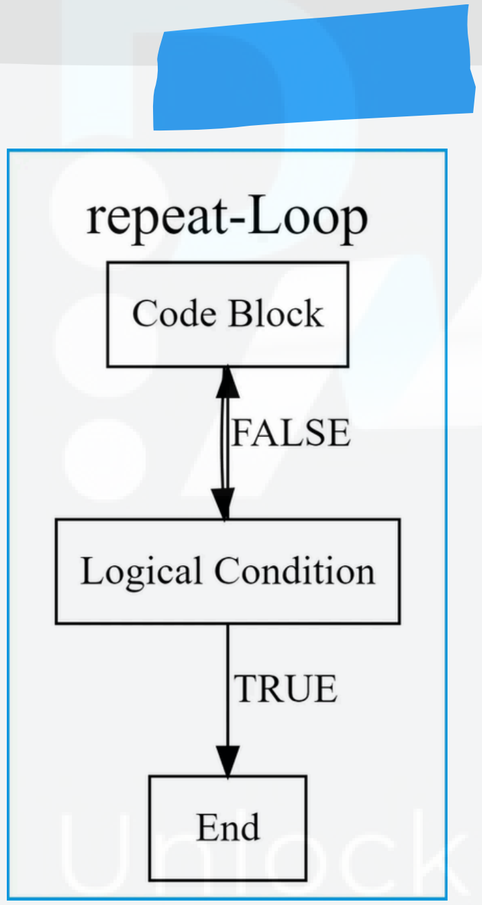
```
# Example: Using a while loop to count from 1 to 5
count <- 1

while (count <= 5) {
  cat("Count:", count, "\n")
  count <- count + 1
}
```

Ready to start ?

R repeat Loop

R repeat Loop



basic structure of a repeat loop in R is as follows:

```
repeat {
  # Code to be executed repeatedly
  if (condition) {
       break   # Terminate the loop if the
condition is met
  }
}
```

Repeat loop is used to create an infinite loop that continues to execute a block of code until a specified condition is met or until an explicit break statement is encountered within the loop. repeat loops are typically used when you need to repeatedly perform a task until a specific condition is satisfied.

repeat loop

1 Example

Using a repeat loop to count from 1 to 5

```
# Example: Using a repeat loop to count from 1 to 5
count <- 1

repeat {
  cat("Count:", count, "\n")
  count <- count + 1
  if (count > 5) {
    break  # Terminate the loop when count reaches 5
  }
}
```

In this example, the repeat loop continues to execute as long as count is less than or equal to 5. Once count becomes greater than 5, the break statement is encountered, and the loop terminates.

repeat loops are useful when you need to create loops that don't have a predetermined number of iterations and rely on a condition or user input to exit. However, it's important to be cautious when using repeat loops to avoid creating infinite loops that run indefinitely. Always include a condition and a break statement to ensure that the loop can be terminated.

Ready to start ?

R break and next

Break Statement:

1 Example

Using break in a for loop

```
for (i in 1:10) {
  if (i == 5) {
    break  # Exit the loop when i equals 5
  }
  cat("Value:", i, "\n")
}
```

- The break statement is used to exit (terminate) a loop prematurely when a certain condition is met.
- When break is encountered inside a loop, the loop is immediately terminated, and the program continues with the code following the loop.
- break is often used to exit loops when a specific condition is satisfied.

Next Statement :

1 Example

Using next in a for loop

```
for (i in 1:5) {
  if (i %% 2 == 0) {
    next  # Skip even numbers
  }
  cat("Value:", i, "\n")
}
```

- The next statement is used to skip the current iteration of a loop and move on to the next iteration.
- When next is encountered inside a loop, the current iteration is immediately terminated, and the loop proceeds to the next iteration.
- next is often used to skip specific iterations based on a condition.

Ready to start ?

R Functions

R Functions

Functions are blocks of code that can be defined and reused to perform specific tasks or operations. Functions encapsulate a series of statements, accept input (arguments), and often return output values. Functions are a fundamental concept in R programming and are essential for modularizing code and making it more organized and reusable.

Key components of a function:

- function_name: The name of the function you define.
- arg1, arg2, ...: Arguments or parameters that the function accepts. You can have zero or more arguments.
- result: The value the function returns (optional).
- arg1_value, arg2_value, ...: Actual values or expressions provided when calling the function.

basic structure of defining and using a function in R

```
# Function definition
function_name <- function(arg1, arg2, ...) {
  # Function body: code to perform a specific task
  # You can use the arguments (arg1, arg2, ...) within the function

  # Return a value (optional)
  return(result)
}

# Function call
output <- function_name(arg1_value, arg2_value, ...)
```

R Functions :

1 Example

Define a function to add two numbers

```
# Define a function to add two numbers
add_numbers <- function(x, y) {
  result <- x + y
  return(result)
}

# Call the function
sum_result <- add_numbers(5, 3)
cat("Sum:", sum_result, "\n")
```

In this example:

- add_numbers is the function name.
- x and y are the function's arguments.
- Inside the function, result is calculated as the sum of x and y.
- The return(result) statement returns the result.
- We call the function with values 5 and 3 and store the result in sum_result.

Examples of different types of functions in R

1 Built-in Functions

Calculate the sum of a vector

```
# Example of built-in functions
# Calculate the sum of a vector
numbers <- c(5, 10, 15, 20)
sum_result <- sum(numbers)
cat("Sum of numbers:", sum_result, "\n")

# Calculate the mean of a vector
mean_result <- mean(numbers)
cat("Mean of numbers:", mean_result, "\n")

# Find the length of a vector
length_result <- length(numbers)
cat("Length of vector:", length_result, "\n")
```

Examples of different types of functions in R

2 User-Defined Function

Example of a user-defined function

```
# Example of a user-defined function
# Define a function to calculate the area of a rectangle
calculate_rectangle_area <- function(length, width) {
  area <- length * width
  return(area)
}

# Call the user-defined function
rectangle_area <- calculate_rectangle_area(4, 6)
cat("Area of rectangle:", rectangle_area, "\n")
```

Examples of different types of functions in R

3 Anonymous Function (Lambda Function)

Example of an anonymous function

```
# Example of an anonymous function
# Use lapply to square each element in a vector
numbers <- c(1, 2, 3, 4, 5)
squared_numbers <- lapply(numbers, function(x) x^2)
cat("Squared numbers:", unlist(squared_numbers), "\n")
```

4 Higher-Order Function

Example of a higher-order function

```
# Example of a higher-order function
# Use sapply to apply a function to each element in a list
fruits_list <- list("apple", "banana", "cherry")
lengths <- sapply(fruits_list, length)
cat("Lengths of fruits:", lengths, "\n")
```

Ready to start ?

R String

R String

Strings are used to represent and manipulate text data. You can create and manipulate strings in various ways in R. Here are some fundamental operations and examples related to strings in R

1 Creating Strings

Creating strings

```
# Creating strings
string1 <- "Hello, p4n!"  # Using double quotes
string2 <- 'codes with pankaj'  # Using single quotes
```

R String

2 Concatenating Strings

Concatenating strings

```
# Concatenating strings
str1 <- "p4n"
str2 <- "World"
concatenated <- paste(str1, str2)
cat("Concatenated:", concatenated, "\n")
```

3 String Length

Finding the length of a string

```
# Finding the length of a string
text <- "Welcome to p4n."
length_text <- nchar(text)
cat("Length of the string:", length_text, "\n")
```

R String

4 Substring Extraction

Extracting substrings

```
# Extracting substrings
text <- "R Programming"
substring <- substr(text, start = 3, stop = 7)
cat("Substring:", substring, "\n")
```

5 String Manipulation

Converting to lowercase

```
# Converting to lowercase
text <- "Hello,p4n!"
lower_text <- tolower(text)
cat("Lowercase:", lower_text, "\n")
```

R String

6 String Comparison

String comparison

```
# String comparison
text1 <- "apple"
text2 <- "banana"
result <- text1 < text2
cat("Comparison result:", result, "\n")
```

7 String Interpolation

String Interpolation

```
# String interpolation
name <- "Nishant"
age <- 12
greeting <- sprintf("Hello, my name is %s and I am %d years old.",
name, age)
cat("Greeting:", greeting, "\n")
```

R String

8 Splitting and Joining Strings:

String comparison

```
# Splitting and joining strings
text <- "apple,banana,cherry"
split_text <- strsplit(text, ",")[[1]]
joined_text <- paste(split_text, collapse = ";")
cat("Joined text:", joined_text, "\n")
```

Ready to start ?

R Vectors

R Vector

Vector is a fundamental data structure that allows you to store and manipulate a collection of values of the same data type. Vectors can be used to store numbers, characters, logical values, and other data types. Here are some key points and examples related to vectors in R:

You can create vectors in R using the c() function or by specifying a sequence of values within c(). Here are examples:

1 Creating Vectors

Creating Vectors

```
# Creating numeric vectors
numeric_vector <- c(1, 2, 3, 4, 5)

# Creating character vectors
character_vector <- c("apple", "banana", "cherry")

# Creating logical vectors
logical_vector <- c(TRUE, FALSE, TRUE, FALSE)
```

R Vectors

2 Vector Operations

Vector Operations

```
# Arithmetic operations on numeric vectors
vector1 <- c(1, 2, 3)
vector2 <- c(4, 5, 6)
result_addition <- vector1 + vector2
result_multiplication <- vector1 * vector2

# Element-wise comparisons on numeric vectors
comparison_result <- vector1 > vector2

# Logical operations on logical vectors
logical_result <- vector1 < 3 & vector2 >= 5
```

R Vectors

3 Accessing Elements:

Accessing Elements:

```
# Accessing elements of a vector
numeric_vector <- c(10, 20, 30, 40, 50)
first_element <- numeric_vector[1]
third_element <- numeric_vector[3]
```

4 Vector Functions

Vector functions

```
# Vector functions
numeric_vector <- c(10, 20, 30, 40, 50)
vector_length <- length(numeric_vector)
vector_sum <- sum(numeric_vector)
vector_mean <- mean(numeric_vector)
```

R Vectors

5 Vectorized Operations

Vectorized Operations

```
# Vectorized operations
numeric_vector <- c(1, 2, 3, 4, 5)
square_vector <- numeric_vector^2
```

6 Vector Attributes

Vector Attributes

```
# Assigning names to vector elements
vector <- c(a = 10, b = 20, c = 30)
```

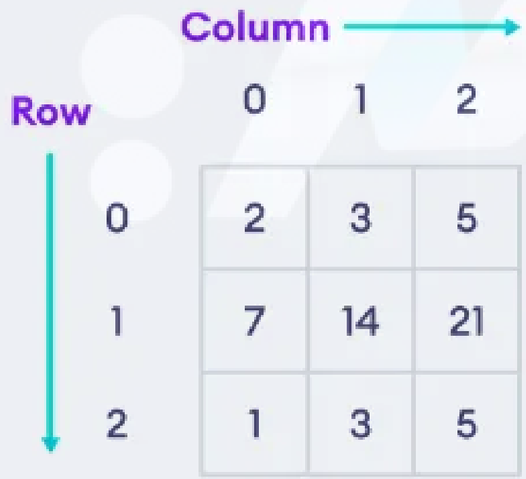
Ready to start ?

R Matrix

R Matrix

Matrix is a two-dimensional data structure that consists of rows and columns, similar to a table or a spreadsheet. Matrices are used to store and manipulate data in a structured format. Here are some key points and examples related to matrices in R:

R Matrix



Matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. In R programming, matrices are two-dimensional, homogeneous data structures

R Matrix

Example

Matrix

```
# R program to create a matrix
A = matrix(
# Taking sequence of elements
c(1, 2, 3, 4, 5, 6, 7, 8, 9),
# No of rows
nrow = 3,
# No of columns
ncol = 3,
# By default matrices are in column-wise order
# So this parameter decides how to arrange the matrix
byrow = TRUE
)
# Naming rows
rownames(A) = c("a", "b", "c")

# Naming columns
colnames(A) = c("c", "d", "e")

cat("The 3x3 matrix:\n")
print(A)
```

R Vectors

1 Creating Matrices

Vectorized Operations

```
# Creating a numeric matrix
numeric_matrix <- matrix(data = 1:12, nrow = 3, ncol = 4)
# Creating a character matrix with row and column nameschar_matrix <-
matrix(data = c("A", "B", "C", "D"), nrow = 2, ncol = 2,dimnames =
list(c("Row1", "Row2"), c("Col1", "Col2")))
```

Ready to start ?

R List

R List

list is a versatile data structure that can hold various types of data elements, including numbers, characters, vectors, other lists, and even functions. Lists are often used to store and manage heterogeneous data and can be nested within each other to create complex data structures. Here are some key points and examples related to lists in R

1 Creating Matrices

Creating Matrices

```
# Creating a list with different data types
my_list <- list(name = "Alice", age = 30, scores = c(85, 92, 78), has_pet = TRUE)
```

R List

2 Accessing List Elements

Accessing List Elements

```
# Accessing list elements by name
name <- my_list$name
age <- my_list$age

# Accessing list elements by position
first_score <- my_list[[3]][1]
```

3 Nested Lists

Nested Lists

```
# Creating a nested list
nested_list <- list(person1 = list(name = "Bob", age = 25), person2 =
list(name = "Alice", age = 30))
```

R List

4 Modifying Lists

Modifying Lists

```
# Adding a new element to the list
my_list$city <- "New York"

# Modifying an existing element
my_list$age <- 31

# Removing an element from the list
my_list$city <- NULL
```

5 List Functions

List Functions

```
# List functions
list_length <- length(my_list)
list_names <- names(my_list)
str(my_list)
```

R List

6 List of Functions

List of Functions

```
# Creating a list of functions
function_list <- list(square = function(x) x^2, double = function(x)
2*x)

# Calling functions from the list
result1 <- function_list$square(5)
result2 <- function_list$double(7)
```

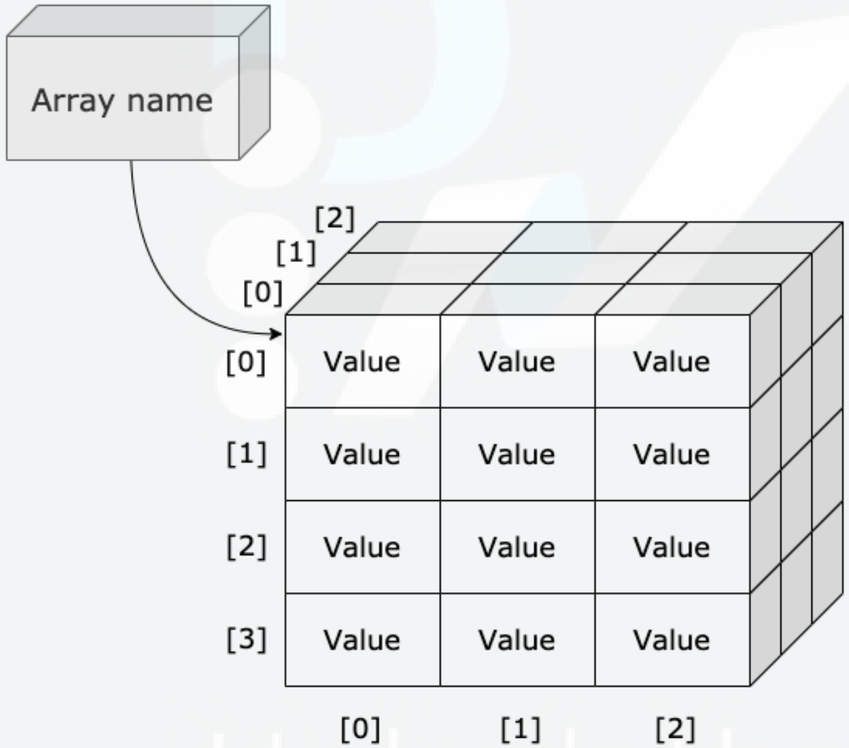
Ready to start ?

R Array

R Array

In R, an array is a multi-dimensional data structure that can store data of the same data type. Unlike matrices, which are two-dimensional, arrays can have more than two dimensions, making them suitable for representing and working with higher-dimensional data. Here are some key points and examples related to arrays in R:

R While Loop



An Array is a data structure which can store data of the same type in more than two dimensions.

R Array

1 Creating Arrays

Creating Arrays

```
# Creating a 2-dimensional array
data_matrix <- matrix(1:12, nrow = 3, ncol = 4)
my_array <- array(data_matrix, dim = c(3, 4, 2))

# Creating a 3-dimensional array with dimension names
data_array <- array(1:24, dim = c(3, 4, 2),
                          dimnames = list(c("A", "B", "C"),
c("X", "Y", "Z"), c("M", "N")))
```

2 Accessing Array Elements

Accessing Array Elements

```
# Accessing elements of an array
element1 <- my_array[1, 2, 1]
element2 <- data_array["A", "Y", "M"]
```

R Array

3 Array Operations

Array Operations

```
# Transposing a 3-dimensional array
transposed_array <- aperm(data_array, c(3, 2, 1))
```

4 Array Functions

Array Functions

```
# Array functions
array_dim <- dim(my_array)
array_dimnames <- dimnames(data_array)
str(data_array)
```

R Array

5 Array Type

Array Type

```
# Creating a logical array
logical_array <- array(c(TRUE, FALSE, TRUE, TRUE), dim =
c(2, 2))
```

Ready to start ?

R Data Frame

R Array

In R, a data frame is a two-dimensional data structure that resembles a table or a spreadsheet. It is one of the most commonly used data structures for handling and analyzing data because it can store data of different types (numeric, character, logical) in columns and rows. Data frames are particularly useful for representing structured data, such as datasets from spreadsheets or databases. Here are some key points and examples related to data frames in R:

R Data Frames

1 Creating Data Frames

Creating Data Frames

```
# Creating a data frame
student_data <- data.frame(
  Name = c("Alice", "Bob", "Charlie", "David"),
  Age = c(25, 22, 24, 23),
  Grade = c("A", "B", "A", "C"),
  Passed = c(TRUE, TRUE, TRUE, FALSE)
)
```

You can create data frames in R using the data.frame() function. You provide vectors of data for each column, and each vector becomes a column in the data frame

2 Accessing Data Frame Elements

Accessing Data Frame Elements

```
# Accessing elements of a data frame
name1 <- student_data$Name[1]
# Accessing the first student's name
age3 <- student_data$Age[3]
# Accessing the age of the third student
```

R Data Frames

3 Modifying Data Frames

Modifying Data Frames

```
# Adding a new column
student_data$City <- c("New York", "Los Angeles", "Chicago",
"Houston")

# Modifying an existing column
student_data$Grade[4] <- "B"

# Removing a column
student_data$City <- NULL
```

4 Data Frame Functions

Data Frame Functions

```
# Data frame functions
num_rows <- nrow(student_data)
num_cols <- ncol(student_data)
column_names <- names(student_data)
data_summary <- summary(student_data)
```

R Data Frames

5 Data Frame Type

Data Frame Type

```
# Creating a data frame with mixed data types
mixed_data <- data.frame(
  Name = c("Alice", "Bob"),
  Age = c(25, 30),
  Passed = c(TRUE, FALSE)
)
```