4. Basic Functions and Commands in R

Code snippet

```
# Basic arithmetic operations
x <- 5 # Assign value 5 to x
y <- 10 # Assign value 10 to y
sum <- x + y # Add x and y
difference <- x - y # Subtract x from y
product <- x * y # Multiply x and y
quotient <- x / y # Divide x by y

# Comparison operators
is_greater <- x > y # Check if x is greater than y
is_less <- x < y # Check if x is less than y
is_greater_equal <- x >= y # Check if x is greater than or equal to y
is_less_equal <- x <= y # Check if x is less than or equal to y
is_equal <- x == y # Check if x is equal to y
is_not_equal <- x != y # Check if x is not equal to y

# Logical operators
is_both_true <- x & y # Check if both x and y are true
is_either_true <- x | y # Check if either x or y is true
is_x_not_true <- !x # Check if x is not true

# Assignment operators
x <- 10 # Assign value 10 to x
y <- x # Assign value of x to y
z <- x + 5 # Assign the sum of x and 5 to z

# Vector operations
v <- c(1, 2, 3, 4, 5) # Create a vector with values 1 to 5
sum_v <- sum(v) # Calculate the sum of elements in v
mean_v <- mean(v) # Calculate the mean of elements in v
median_v <- median(v) # Calculate the median of elements in v
variance_v <- var(v) # Calculate the variance of elements in v
std_dev_v <- sd(v) # Calculate the standard deviation of elements in v
```

Use code with caution. Learn more

content_copy

5. Word Cloud

Code snippet

```
# Install and load wordcloud package
if (!require("wordcloud")) install.packages("wordcloud")
library(wordcloud) # Load wordcloud library

# Load sample text
text <- paste(readLines("your_text_file.txt"), collapse = "\n") # Load
text from a file

# Create word cloud
set.seed(123) # Set random seed for reproducibility
wordcloud(
  words = str_split(text, "\\s+"), # Split text into words
  scale = c(4, 1) # Set scaling factors for word size
)
```

Use code with caution. Learn more

content_copy

6. Fibonacci Series

Code snippet

```
fibonacci <- function(n) {
  # Recursive function to calculate the nth Fibonacci number
  if (n == 0) { # Base case
    return(0)
  } else if (n == 1) { # Base case
    return(1)
  } else { # Recursive case
    return(fibonacci(n - 1) + fibonacci(n - 2))
  }
}

# Generate Fibonacci numbers up to 10
fib_series <- sapply(1:10, fibonacci) # Apply fibonacci function to
numbers 1 to 10
```

```
print(fib_series)
```

Use code with caution. Learn more

content_copy

7. Armstrong Number

Code snippet

```
is_armstrong <- function(n) {
  # Check if a number is an Armstrong number
  digits <- str_split(as.character(n), "") # Split the number into digits
  sum(digits^nlen(digits)) == n # Check if the sum of digits raised to the power of their number is equal to the original number
}

# Check if 153 is an Armstrong number
is_armstrong(153)
```

Use code with caution. Learn more

content_copy

8. Prime Numbers

Code snippet

```
is_prime <- function(n) {
  # Check if a number is prime
  if (n == 0 || n == 1) { # Special cases
    return(FALSE)
  }
  for (i in 2:sqrt(n)) { # Check for divisors from 2 to the square root of the number
    if (n %% i == 0) { # If a divisor is found, it is not prime
      return(FALSE)
    }
  }
  return(TRUE) # If no divisors are found, it is prime
}

# Find prime numbers from 1 to 100
prime_numbers <- sapply(1:100, is_prime) # Apply is_prime function to
numbers 1 to 100
```

12. Implementation of Matrices

[redacted]

Code snippet

```
# Create a matrix
matrix1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, ncol = 3)

# Access elements by row and column indices
element1 <- matrix1[1, 2]

# Modify elements
matrix1[2, 1] <- 10

# Get dimensions of the matrix
dim(matrix1)

# Matrix operations
matrix2 <- matrix(c(7, 8, 9, 10, 11, 12), nrow = 2, ncol = 3)
sum_matrix <- matrix1 + matrix2
product_matrix <- matrix1 * matrix2

# Transpose the matrix
t(matrix1)

# Apply functions to matrix elements
sqrt_matrix <- sapply(matrix1, sqrt)

# Print the matrix
print(matrix1)
```

Use code with caution. Learn more

content_copy

13. Implementation of Lists

[redacted]

Code snippet

```
# Create a list
mylist <- list(1, "John", TRUE, c(4, 5, 6))

# Access elements by index
element1 <- mylist[[1]]

# Add elements to the list
mylist[[5]] <- "New York"

# Get the length of the list
length(mylist)

# Combine lists
combined_list <- c(mylist1, mylist2)

# Check if an element exists in the list
"John" %in% mylist

# Apply functions to list elements
uppercase_list <- sapply(mylist, toupper)

# Print the list
print(mylist)
```

Use code with caution. Learn more

content_copy

14. Installation of Hadoop Software

Windows:

1. Download Hadoop tarball from the Apache website (https://hadoop.apache.org/releases.html).
2. Extract the tarball to your desired directory.
3. Set environment variables for HADOOP_HOME and PATH in your system settings.
4. Configure Hadoop by editing the `core-site.xml` and `hdfs-site.xml` files in the `conf` directory.
5. Verify the installation by running `hdfs dfs -ls /`.

Ubuntu:

6. Open a terminal window.
7. Update package list and upgrade existing packages: `sudo apt update && sudo apt upgrade -y`
8. Install Java: `sudo apt install openjdk-11-jdk -y`
9. Install Hadoop: `sudo apt install hadoop -y`
10. Configure Hadoop by editing the `core-site.xml` and `hdfs-site.xml` files in the `/etc/hadoop/conf` directory.
11. Verify the installation by running `hadoop fs -ls /`.

15. Find Even or Odd Number using R Functions

[redacted]

Code snippet

```
# Define a function to check if a number is even
is_even <- function(n) {
  if (n %% 2 == 0) {
    return("Even")
  } else {
    return("Odd")
  }
}

# Check if 10 is even or odd
is_even(10)

# Check if 11 is even or odd
is_even(11)
```

Use code with caution. Learn more

content_copy

16. String Operations in R

[redacted]

1. Concatenation:

Code snippet

```
str1 <- "Hello"
str2 <- "World"
str_combined <- paste(str1, str2) # Hello World
```

Use code with caution. Learn more

content_copy

2. Substring extraction:

Code snippet

```
str <- "This is a string"
substring <- substr(str, 5, 10) # is a string
```

Use code with caution. Learn more

content_copy

3. String replacement:

Code snippet

```
str <- "The quick brown fox jumps over the lazy dog"
str_replaced <- gsub("brown", "red", str) # The quick red fox jumps
over the lazy dog
```

Use code with caution. Learn more

content_copy

4. String conversion:

Code snippet

```
str <- "12345"
numeric_value <- as.numeric(str) # 12345
```

Use code with caution. Learn more

content_copy

5. String searching:

Code snippet

```
str <- "This is a string"
position <- gregex("string", str) # 8
```

Basic Functions and Commands in R Programming:

[redacted]

1. Arithmetic Operations:

- Addition: x + y
- Subtraction: x - y
- Multiplication: x * y
- Division: x / y
- Exponentiation: x ^ y
- Square Root: sqrt(x)
- Natural Logarithm: log(x)

Example:

Code snippet

```
x <- 10
y <- 5
```

```
sum <- x + y
difference <- x - y
product <- x * y
quotient <- x / y
power <- x ^ y
square_root <- sqrt(x)
natural_log <- log(y)

print(sum) # 15
print(difference) # 5
print(product) # 50
print(quotient) # 2
print(power) # 100
print(square_root) # 3.162278
print(natural_log) # 1.609438
```

Use code with caution. Learn more

content_copy

2. Comparison Operators:

- Equal to: x == y
- Not equal to: x != y
- Greater than: x > y
- Less than: x < y
- Greater than or equal to: x >= y
- Less than or equal to: x <= y

Example:

Code snippet

```
is_equal <- x == y
is_not_equal <- x != y
is_greater <- x > y
is_less <- x < y
is_greater_equal <- x >= y
is_less_equal <- x <= y

print(is_equal) # FALSE
print(is_not_equal) # TRUE
print(is_greater) # TRUE
print(is_less) # FALSE
print(is_greater_equal) # TRUE
print(is_less_equal) # FALSE
```

Use code with caution. Learn more

content_copy

3. Logical Operators:

- AND: x & y (True only if both x and y are true)
- OR: x | y (True if either x or y is true)
- NOT: !x (Reverses the logical value of x)

Example:

Code snippet

```
is_both_true <- is_greater & is_equal
is_either_true <- is_greater | is_less_equal
is_not_true <- !is_equal

print(is_both_true) # FALSE
print(is_either_true) # TRUE
print(is_not_true) # TRUE
```

Use code with caution. Learn more

content_copy

4. Assignment Operators:

- <-: Assign a value to a variable
- +=, -=, *=, /=: Increment, decrement, multiply, and divide by a value

Example:

Code snippet

```
# Assign value 10 to x
x <- 10

# Increment x by 5
x += 5

# Multiply x by 2
x *= 2

# Divide x by 3
x /= 3

print(x) # 8.333333
```

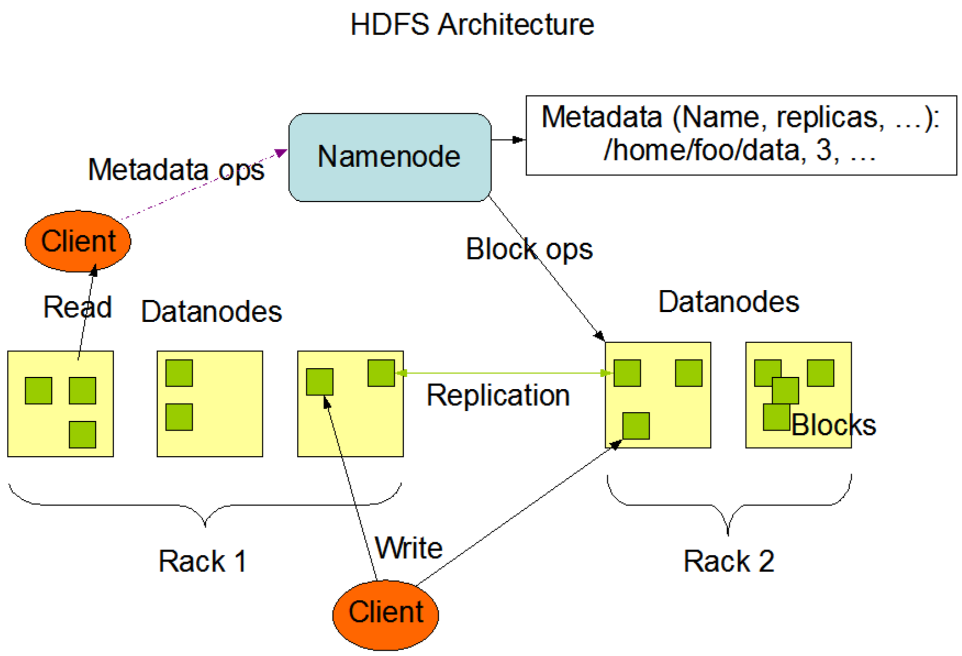
Use code with caution. Learn more

content_copy

5. Other Basic Commands:

- print(x): Print the value of x.
- cat(x): Print the value of x without a new line.
- length(x): Get the length of a vector.
- sum(x): Get the sum of elements in a vector.
- mean(x): Get the mean of elements in a vector.
- median(x): Get the median of elements in a vector.
- sd(x): Get the standard deviation of elements in a vector.

[redacted]



1. Hadoop Architecture and Ecosystem with Word Count Example:

Hadoop Architecture:

Hadoop is a distributed computing framework designed to handle large datasets. It consists of several core components:

- HDFS (Hadoop Distributed File System): Stores data across multiple nodes in a cluster.
- YARN (Yet Another Resource Negotiator): Manages resources like CPU, memory, and network bandwidth across the cluster.
- MapReduce: A programming model for parallel processing of large datasets.

Hadoop Ecosystem:

Hadoop integrates with various tools and frameworks to provide a comprehensive big data ecosystem. Some key components include:

- Pig and Hive: Data warehousing tools for querying and analyzing large datasets.
- HBase: A NoSQL database for real-time access to big data.
- Spark: A general-purpose distributed processing engine for fast data processing.

Word Count Example:

The Word Count application counts the occurrence of each word in a text file. It demonstrates the MapReduce programming model:

- Map: Reads each line of the text file, splits it into words, and emits (word, 1) pairs.
- Reduce: Receives (word, 1) pairs from multiple mappers, aggregates the counts for each word, and outputs the final word count.

Case Study:

- A company receives large log files from its web servers.
- They want to analyze the logs to understand user behavior and identify trends.
- They can use Hadoop to process the log files and count the occurrences of different keywords.

Benefits of using Hadoop:

- Scalability: Handles large datasets efficiently.
- Fault tolerance: Can recover from hardware failures.
- Cost-effective: Uses commodity hardware.
- Flexible: Integrates with various tools and frameworks.

Drawing the architecture:

The architecture can be represented visually using a diagram showing the interaction between HDFS, YARN, MapReduce, and other ecosystem components.

Explaining the components:

Each component can be explained in detail, including its functionalities and role in the overall architecture.

2. File Management Tasks in HDFS:

Adding files and directories:

- Use `hdfs dfs -put <source_file> <destination_path>` to upload a file.
- Use `hdfs dfs -mkdir <directory_path>` to create a directory.

Retrieving files:

- Use `hdfs dfs -get <source_path> <destination_file>` to download a file.
- Use `hdfs dfs -ls <path>` to list files and directories in a path.

Deleting files:

- Use `hdfs dfs -rm <path>` to delete a file or directory.

Additional commands:

- `hdfs dfs -mv <source_path> <destination_path>`: Move files or directories.
- `hdfs dfs -cp <source_path> <destination_path>`: Copy files or directories.
- `hdfs dfs -chown <owner> <group> <path>`: Change file ownership.

It's important to note that these commands are executed from the command line within the Hadoop environment.

3. Running Word Count in MapReduce:

Word Count implementation:

- Write the Map and Reduce functions using Java or another programming language supported by Hadoop.
- Package the program into a JAR file.
- Submit the JAR file to the Hadoop cluster using the `hadoop jar` command with the appropriate arguments.
- Monitor the job execution using the `yarn application -list` command.
- View the output file generated by the Reduce phase.

Calculating execution time:

- Use the job completion time and start time to calculate the execution time.
- Analyze logs for detailed information on performance metrics.

Resources:

- https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Here's how to build a Word Cloud using R:

1. Install and Load Required Packages:

[redacted]

Code snippet

```
if (!require("wordcloud")) install.packages("wordcloud")
library(wordcloud)
```

Use code with caution. Learn more

content_copy

2. Load Text Data:

[redacted]

Code snippet

```
# Example using a text file
text <- paste(readLines("your_text_file.txt"), collapse = "\n")

# Example using a string variable
text <- "This is a sample text string for word cloud analysis."
```

Use code with caution. Learn more

content_copy

3. Preprocess Text Data (Optional):

[redacted]

Code snippet

```
# Remove punctuation and special characters
text <- gsub("[[:punct:]]", "", text, fixed = TRUE)

# Convert text to lowercase
text <- tolower(text)
```

```
# Remove stop words (optional)
# library(stopwords)
# text <- gsub(stopwords("english"), "", text)
```

Use code with caution. Learn more

content_copy

4. Generate Word Cloud:

[redacted]

Code snippet

```
set.seed(123) # Set random seed for reproducibility
wordcloud(words = str_split(text, "\\s+"), scale = c(4, 1))
```

Use code with caution. Learn more

content_copy

5. Customize Word Cloud:

[redacted]

- `colors`: A vector of colors for the words.
- `rot.per`: The percentage of words to rotate.
- `min.freq`: Minimum frequency of a word to be included.
- `max.words`: Maximum number of words to display.
- `font.family`: Font family to use for the words.

Example:

Code snippet

```
wordcloud(
  words = str_split(text, "\\s+"),
  scale = c(4, 1),
  colors = brewer.all.palettes[["Set2"]][1:6],
  rot.per = 0.25,
  min.freq = 5,
  max.words = 100,
  font.family = "Times New Roman"
)
```

Use code with caution. Learn more

content_copy

6. Save Word Cloud:

You can save the Word Cloud as an image file using the `ggsave()` function.

Code snippet

```
ggsave("word_cloud.png", width = 800, height = 600)
```

Use code with caution. Learn more