Module (2)

Lecture Notes: Vectors in R

Lecturer: Ayon

Key Topics:

- Objects in R
- Atomic Vectors
- Creating Vectors
- Mixing Objects in Vectors
- Vector Operations
- Vector Attributes
- Subsetting Vectors
- Vector Functions
- Plotting Vectors

Introduction

R is an object-oriented programming language built on C. The fundamental design principle of R is that **everything** in R is an object. This includes:

- **Variables:** Scalars, vectors, matrices, data frames
- **Functions**
- **Data:** Datasets, tables

Objects in R (Quick Detour)

In R, objects can be of various types and have different purposes.

- **Scalars:** Single values (e.g., 5, "hello", TRUE)
- **Vectors:** Ordered collections of elements of the same type
- **Matrices:** Two-dimensional arrays of elements
- **Data Frames:** Tables where columns can be of different types
- **Functions:** Blocks of code that perform a specific task

Code Snippet (Creating Objects in R)

Code snippet

```r
# Scalar (single value)
x <- 5

# Vector (ordered collection of numbers)
y <- c(1, 2, 3, 4, 5)

# Matrix (2D array)
m <- matrix(1:9, nrow = 3)
```

Atomic Vectors

Atomic vectors are the simplest type of data structure in R. They are one-dimensional and can hold elements of only one data type. The six basic atomic vector types in R are:

1. **Logical:** `TRUE` or `FALSE` values
2. **Integer:** Whole numbers (e.g., 1, 5, -3)
3. **Double (numeric):** Real numbers with decimals (e.g., 3.14, -0.5)
4. **Character (string):** Text data (e.g., "hello", "world")
5. **Complex:** Complex numbers (e.g., 1 + 2i)
6. **Raw:** Unprocessed bytes

Note: In the next section, we will cover how to create and manipulate these atomic vectors in R.

Lecture Notes: Atomic Vectors (continued)

Atomic Vectors in Detail

Atomic vectors are the most basic data structure in R, serving as the building blocks for more complex structures. They are one-dimensional and can only hold elements of the same class (data type).

Types of Atomic Vectors

1. **Logical:** Contains `TRUE` or `FALSE` values.
2. **Integer:** Contains whole numbers (e.g., 2L, 5L, -3L). The `L` suffix denotes an integer.
3. **Double (Numeric):** Contains real numbers with decimals (e.g., 3.14, -0.5).
4. **Character (String):** Contains text data (e.g., "hello", "world").
5. **Complex:** Contains complex numbers (e.g., 1 + 4i).
6. **Raw:** Contains raw bytes (e.g., 0x01, 0xFF).

Important Functions

- `c()`: Combines values into a vector or list.
- `length()`: Returns the number of elements in a vector.
- `class()`: Returns the class (data type) of a vector.
- `typeof()`: Returns the internal storage mode of a vector (more detailed than `class()`).
- `attributes()`: Returns the attributes of a vector (e.g., names, dimensions).

Examples

Code snippet

```
# Creating Atomic Vectors
x <- 3.5  # Numeric vector
y <- c(1, 2, 3, 4, 5)  # Numeric vector
z <- NA  # Logical vector (by default)
w <- 0/0  # Numeric vector (NaN - Not a Number)

# Testing Data Types
is.numeric(1.5)  # TRUE
is.character("a")  # TRUE

# Class and Typeof
class(z)  # "logical"
typeof(z)  # "logical"
```

Key Points

- **Missing Values:** Represented by `NA`. By default, it is logical but can be coerced to other types.
- **Not a Number (NaN):** Represents undefined or unrepresentable values in floating-point calculations. It is of the numeric type.
- **Coercion:** The process of converting an object from one class to another (e.g., converting a logical `NA` to a numeric `NA`).

Exercise

Try using the `is.character()` function on the numeric value `1.5`. What is the output?

Answer: The output will be `FALSE` because `1.5` is not a character.

Lecture Notes: Creating Vectors of Different Types

Creating Vectors using `c()`

- **Double (Numeric):**

Code snippet

```
double_vector <- c(0.5, 0.6, 0.7)
```

- **Logical:**

Code snippet

```
logical_vector <- c(TRUE, FALSE)
```

Note: Avoid using `T` and `F` as they are not protected like `TRUE` and `FALSE`.

- **Character (String):**

Code snippet

```
char_vector <- c("A", "B", "C")
```

- **Integer:**

Code snippet

```
int_vector <- c(1L, 2L, 3L)  # The 'L' denotes integer
```

- **Sequence:**

Code snippet

```
seq_vector <- 15:20  # Creates a sequence from 15 to 20
```

- **Complex:**

Code snippet

```
complex_vector <- c(1+0i, 2+4i, 5+2i)
```

Creating Vectors using `vector()`

- **Numeric Vector (Initialized with Zeros):**

Code snippet

```
num_vector <- vector("numeric", length = 5)
```

- **Other Types:** Can create logical, character, integer, and complex vectors similarly.

Creating Sequences

- **Using Colon Operator (`:`):**

Code snippet

```
x <- 1:10  # Creates a sequence from 1 to 10
```

- **Using `seq()` Function:**

Code snippet

```
seq_vector1 <- seq(from = 1, to = 10, by = 2)  # Sequence with step size 2
seq_vector2 <- seq(from = 1, to = 10, length.out = 5)  # Sequence with 5 values
```

- **Decreasing Order:**

Code snippet

```
dec_seq_vector <- seq(from = 10, to = 1, by = -1)
```

- **`seq_along()` Function:**

Code snippet

```
char_vector <- c("A", "B", "C")
index_vector <- seq_along(char_vector)  # Output: 1 2 3
```

Handy Functions

- `letters`: Generates lowercase letters a to z.

- `LETTERS`: Generates uppercase letters A to Z.

Code snippet

```
first_five_letters <- letters[1:5]
```

Replicating Elements

- `rep()` **Function:**

Code snippet

```
# Replicate each element multiple times
rep(2, times = 5)  # Output: 2 2 2 2 2

# Replicate each element with different repetitions
rep(c("Varanasi", "Guwahati", "Delhi"), times = c(1, 2, 3))

# Replicate the entire vector multiple times
rep(c("Varanasi", "Guwahati", "Delhi"), each = 2)
```

Lecture Notes: Mixing Objects in Vectors

Coercion

In R, when objects of different classes (data types) are combined into a vector, R automatically converts all elements to the most flexible class that can accommodate all the values. This process is called **coercion**.

- **Implicit Coercion:** R automatically decides the most flexible class and converts all elements to that class.
- **Explicit Coercion:** You can explicitly force conversion to a specific class using functions like `as.character()`, `as.numeric()`, etc.

Implicit Coercion Examples

Code snippet

```
x <- c(1, 2, "a", "b")
print(x)  # Output: "1" "2" "a" "b"  (All elements are characters)
class(x)  # Output: "character"

x <- c(1, 2, TRUE, FALSE)
```

```
print(x)  # Output: 1 2 1 0   (TRUE becomes 1, FALSE becomes 0)
class(x)  # Output: "numeric"
```

Explicit Coercion Examples

Code snippet

```
x <- c(1, 2, 3, 4)
y <- as.character(x)
print(y)  # Output: "1" "2" "3" "4"
class(y)  # Output: "character"
```

Limitations of Coercion

Coercion is not always possible. For example, you cannot convert a character vector like `c("A", "B", "C")` to a numeric vector. Attempting to do so will result in `NA` values and a warning.

Code snippet

```
x <- c("A", "B", "C")
y <- as.numeric(x)
print(y)  # Output: NA NA NA
# Warning message: NAs introduced by coercion
```

Lecture Notes: Vector Operations in R

Operations in R

R supports various operations on vectors, including:

- **Mathematical Operations:** `+`, `-`, `*`, `/`, `^` (or `**`), `%%` (modulo), `%/%` (integer division)
- **Relational Operations:** `==`, `!=`, `<`, `<=`, `>`, `>=`
- **Logical Operations:** `!` (negation), `&` (logical AND), `|` (logical OR), `xor` (exclusive OR)

Matching Operator (`%in%`)

The matching operator `%in%` checks if an element exists within a vector.

Code snippet

```
x <- c(1, 2, 3, 4, 5)
2 %in% x  # Output: TRUE
7 %in% x  # Output: FALSE
```

Arithmetic Operations on Vectors

Code snippet

```
x <- c(1, 2, 3, 4, 5)
2 * x  # Output: 2 4 6 8 10  (Each element multiplied by 2)
```

Element-wise Addition of Vectors

Code snippet

```
x <- c(1, 2, 3, 4, 5)
y <- c(2, 3, 4, 5, 6)
x + y  # Output: 3 5 7 9 11  (Element-wise addition)
```

Recycling Rule in Vector Operations

When adding vectors of different lengths, R recycles the shorter vector to match the length of the longer vector. If the lengths are not multiples, R will issue a warning but still perform the operation.

Code snippet

```
x <- c(1, 2, 3, 4, 5)
y <- c(2, 3)
x + y  # Output: 3 5 5 7 7   (Warning: longer object length is not a multiple of shorter
object length)
```

Logical Vector Operations

Logical vectors can be used in arithmetic operations where `TRUE` is treated as 1 and `FALSE` as 0.

Code snippet

```
x <- c(TRUE, FALSE, TRUE)
y <- c(TRUE, TRUE, FALSE)
```

```
x + y  # Output: 2 1 1
```

Operations with `NA`

Any operation involving `NA` (Not Available) will result in `NA`.

Code snippet

```
x <- c(1, 2, NA, 4, 5)
2 * x  # Output: 2 4 NA 8 10
```

Lecture Notes: Vector Attributes

Vector Attributes in R

In R, vector attributes are additional information associated with vectors, providing metadata about the data. This metadata can be useful for various operations and analyses.

Common Attributes

- **`names`**: A character vector containing the names of the elements within a vector.
- **`dim`**: The dimensions of an object (e.g., rows and columns for a matrix).
- **`class`**: The class (data type) of an object. This is a mandatory attribute.

Setting Attributes

- R sets some attributes automatically.
- You can manually set attributes using the `attributes()` function.

Example: Using Names

Code snippet

```
x <- c(age = 25, height = 5.8, weight = 70)
print(x)
# Output: age   height   weight
#         25     5.8     70

attributes(x)
# Output: $names
#          [1] "age"    "height" "weight"
```

```
names(x)
# Output: [1] "age"    "height" "weight"
```

Modifying Names

Code snippet

```
char_vector <- c("a", "b", "c")
names(x) <- char_vector
print(x)
# Output:  a    b    c
#          25   5.8  70
```

Usefulness of Named Vectors

Named vectors are particularly useful when you want to refer to elements by their name instead of their numerical index. This can make your code more readable and easier to understand.

Key Points

- Vector attributes provide additional information about the data stored in a vector.
- The `names` attribute allows you to label the elements of a vector for easier reference.
- You can set and modify vector attributes using the `attributes()` and `names()` functions.

Lecture Notes: Subsetting Vectors

Subsetting Vectors

Subsetting is the process of extracting specific elements from a vector. R provides multiple ways to subset vectors:

- **Indexing:** Using numerical positions.
- **Logical Vectors:** Selecting elements based on conditions.
- **Names:** Using the names of elements in named vectors.

Indexing

Code snippet

```
x <- c("Bob", "John", "Alice", "Mary")

x[1]     # Output: "Bob"
x[c(1, 3)]  # Output: "Bob" "Alice"
x[7]     # Output: NA (Index out of bounds)
```

Logical Vector Indexing

Code snippet

```
x <- 1:5
logical_vector <- c(TRUE, FALSE, TRUE, FALSE, TRUE)
x[logical_vector]  # Output: 1 3 5 (Elements at TRUE positions)
```

Combining Logical Operations

Code snippet

```
ages <- c(15, 20, 25, 30, 35)
ages[ages > 18]  # Output: 20 25 30 35
```

Head and Tail Functions

Code snippet

```
x <- c("Bob", "John", "Alice", "Mary")
head(x, n = 2)   # Output: "Bob" "John"
tail(x, n = 2)   # Output: "Alice" "Mary"
```

Subsetting with Names

Code snippet

```
x <- c(age = 25, height = 5.8, weight = 70)
x["age"]        # Output: age
                #         25
x[c("age", "weight")]
# Output: age weight
#         25     70
```

Negative Indexing

Code snippet

```
x <- c("Bob", "John", "Alice", "Mary")
x[-1]          # Output: "John" "Alice" "Mary"
x[-c(1, 3)]    # Output: "John" "Mary"
```

`which()` Function

The `which()` function returns the indices of elements that satisfy a given condition.

Code snippet

```
x <- c(1, 2, 3, 4, 5)
which(x > 3)   # Output: 4 5

# Finding indices of NA values
x <- c(1, 2, NA, 4, 5)
which(is.na(x))  # Output: 3
```

Double Square Brackets (`[[]]`)

Double square brackets extract a single element directly, without returning a vector or its name.

Code snippet

```
x <- c(age = 25, height = 5.8, weight = 70)
x[[1]]        # Output: 25
```

Lecture Notes: Vector Functions in R

Vector Functions

R offers many built-in functions to manipulate and analyze vectors, making it a powerful tool for data analysis.

Basic Functions

- `sum()`: Calculates the sum of all elements in a vector.
- `mean()`: Calculates the average (mean) of all elements.

- `median()`: Calculates the median (middle value) of the elements.
- `min()`: Finds the minimum value in a vector.
- `max()`: Finds the maximum value in a vector.
- `sd()`: Calculates the standard deviation of the elements.
- `var()`: Calculates the variance of the elements.
- `sqrt()`: Calculates the square root of each element.
- `sort()`: Sorts the elements in ascending order.
- `any()`: Checks if any element is TRUE.
- `all()`: Checks if all elements are TRUE.
- `unique()`: Returns a vector with duplicate elements removed.
- `duplicated()`: Returns a logical vector indicating which elements are duplicates.
- `rev()`: Reverses the order of elements in a vector.

Random Number Generation

- `runif()`: Generates random numbers from a uniform distribution.
- `rnorm()`: Generates random numbers from a normal distribution.
- `set.seed()`: Sets the random number seed for reproducibility.

Examples

Code snippet

```
# Random number generation
runif(n = 5, min = 0, max = 1)

rnorm(n = 5, mean = 0, sd = 1)

set.seed(123)  # Set seed for reproducibility
```

Code snippet

```
# Vector Information and Summary
x <- c(1, 2, 3, 4, 5)
str(x)
summary(x)
```

Handling NAs

The `na.rm` argument (available in many functions) allows you to exclude `NA` values from calculations.

Code snippet

```
x <- c(1, 2, NA, 4, 5)
sum(x, na.rm = TRUE)  # Output: 12
```

`any()` and `all()` Functions

Code snippet

```
x <- c(TRUE, FALSE, TRUE)
any(x)   # Output: TRUE
all(x)   # Output: FALSE
```

`unique()` and `duplicated()` Functions

Code snippet

```
x <- c(1, 2, 3, 4, 5, 1, 2, 3)
unique(x)       # Output: 1 2 3 4 5
duplicated(x)   # Output: FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
```

`rev()` Function

Code snippet

```
x <- c(1, 2, 3, 4, 5)
rev(x)  # Output: 5 4 3 2 1
```

Lecture Notes: Plotting Vectors

Plotting Vectors in R

R provides several functions for visualizing vector data, offering a basic introduction to plotting before diving deeper into data visualization later in the course.

Basic Plotting Functions

- `plot()`: Generic function for plotting vectors, often used to create scatterplots or line plots.
- `barplot()`: Creates bar charts to visualize categorical data or summaries.
- `pie()`: Creates pie charts to show proportions.
- `hist()`: Generates histograms for visualizing the distribution of numerical data.
- `boxplot()`: Creates box plots to display the distribution's summary statistics (median, quartiles, etc.).

Examples

Code snippet

```
# Plotting using plot()
x <- 1:5
plot(x)

# Enhanced plot
plot(x, xlab = "Index Position in the Vector", ylab = "Measurement", type = "b", col =
"blue")
```

Code snippet

```
# Bar plot
votes <- c(A = 100, B = 150, C = 200)
barplot(votes)

# Enhanced bar plot
barplot(votes, main = "Votes", xlab = "Candidate", ylab = "Number of Votes", col =
c("red", "green", "blue"))
```

Code snippet

```
# Pie chart
pie(votes)

# Enhanced pie chart
pie(votes, main = "Vote Distribution", col = c("red", "green", "blue"))
```

Code snippet

```
# Histogram
x1 <- rnorm(1000, mean = 0, sd = 1)
hist(x1)

# Enhanced histogram
hist(x1, main = "Histogram of Random Normal Values", xlab = "Value", ylab = "Frequency",
col = "lightblue")
```

Code snippet

```
# Box plot
boxplot(x1)
```

Additional Notes

- `summary()`: Provides summary statistics (mean, median, quartiles, etc.) for a vector.
- `na.rm` **Argument:** In functions like `sum()`, you can set `na.rm = TRUE` to exclude `NA` (Not Available) values from calculations.

Key Points

- R's plotting functions offer a basic yet effective way to visualize vector data.
- You can customize plots using various arguments to enhance their appearance and clarity.
- The `summary()` function provides a statistical summary of vector data, and its output can be related to the visual representation in box plots.