SPEC REU R Resources: Intro to R - Calculations & Logical Operations

Alix Ziff, Gaea Morales, Zachary Johnson

Summer 2024

In this document, we'll continue to dive deeper into the essential basics of R, focusing on two fundamental aspects of R: arithmetic and logical operations. These are key skills that will lay the groundwork for everything you'll do in R, from data analysis to advanced statistical modeling.

Arithmetic in R

R is not only a statistical programming language but also a highly capable calculator. You can perform a variety of arithmetic operations directly in R.

	Operator	Example
Addition	`+`	`2 + 4`
Subtraction	`-`	`2 - 4`
Multiplication	`*`	`2 * 4`
Division	`/`	`4 / 2`
Exponentiation	`^ or **`	`2 ^ 4`
Square Root	`sqrt()`	`sqrt(144)`
Absolute Value	`abs()`	`abs(-4)`
Finding the Remainder	`%%`	`2 %% 4`
Division w/ round down	`%/%`	`2 %/% 4`

Let's put these operators into practice with a few examples:

```
# Multiplication
4*9
```

```
## [1] 36
```

```
# Square root
sqrt(144)
```

```
## [1] 12
```

```
# Exponentiation
## Note: Spaces between operators do not affect the outcome
4^5
```

```
## [1] 1024
```

```
4 ^ 5
```

```
## [1] 1024
```

Note that spaces between operators do not affect the calculation, so `4 + 5` is the same as `4+5`. However, adding spaces makes the code more readable, which is considered good practice. Yet, spaces **do** matter when defining character values, which we will discuss in a later walkthrough.

Just like with a regular calculator, it's important to remember the order of operations! Let's look at an example:

```
# Using order of operations
6 * 8 - sqrt(7) + abs(-10) * (4 / 5)
```

```
## [1] 53.35425
```

```
# Changing the order with parentheses
6 * (8 - sqrt(7)) + abs(-10) * (4 / 5)
```

```
## [1] 40.12549
```

Basic R Operations

Beyond basic arithmetic, R is also equipped to handle more complex numerical descriptions. Here are some of the basic functions you can use to analyze numerical data:

	Operator
Sum	`sum()`
Mean	`mean()`
Median	`median()`
Quantile	`quantile()`
Minimum	`min()`
Maximum	`max()`
Variance	`var()`
Standard Deviation	`sd()`
Summary	`summary()`
Sort	`sort()` for ascending order OR `sort(, decreasing = TRUE)` for descending order

The `summary()` function provides a comprehensive output, displaying the minimum, maximum, mean, median, and quantiles of your data all at once. The `sort()` function, on the other hand, arranges the values in your dataset in ascending order by default. However, you can sort the values in descending order by adding a comma after the vector name and including the argument `decreasing = TRUE` within the parentheses.

Let's practice! Let's say we have a vector, `r_fun`, that contains 20 randomly generated numbers. We'll use the `rnorm()` function generates a vector of 20 values drawn from a normal distribution.

Note: While we'll explore the `rnorm()` function and the concept of normal distribution in more detail in future modules, it's important to know that R offers various ways to generate or define numbers. For instance, we could also manually create a vector using the `c()` function to concatenate specific values into a single object.

```
# Create vector 'r_fun'
r_fun <- rnorm(20)

## The r_fun vector now appears in your Global Environment under 'Values'
```

Now, calculate the sum and the mean of vector `r_fun`, sort the vector in descending order, and get its statistical summary.

```r
# Calculate the sum of the vector
sum(r_fun)
```

```
## [1] 1.496529
```

```r
# Find the mean of the vector
mean(r_fun)
```

```
## [1] 0.07482643
```

```r
# Sort the vector in descending order
sort(r_fun, decreasing = TRUE)
```

```
##  [1]  2.015669022  1.558470254  1.108959178  1.043953758  0.990352708
##  [6]  0.539682535  0.221251043  0.108851223  0.007810539 -0.151979401
## [11] -0.178937750 -0.234622390 -0.239181941 -0.297712674 -0.322234498
## [16] -0.448042911 -0.550255644 -0.789815171 -1.024612691 -1.861076538
```

```r
# Get a statistical summary of the vector
summary(r_fun)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -1.86108 -0.35369 -0.16546  0.07483  0.65235  2.01567
```

Logical operators

Logical operators in R are essential for tasks such as data subsetting, exploratory analysis, and data cleaning. Unlike arithmetic operations that yield numerical outputs, logical operations return either `TRUE` or `FALSE` based on the evaluation of conditions.

Here are the basic logical operators and some examples:

	Operator	Example
Less than	`<`	`4 > 2`
Less than or equal to	`<=`	`4 <= 2`
Greater than	`>`	`4 > 2`
Greater than or equal to	`>=`	`4 >= 2`
Exactly equal to	`==`	`5 == 2 ^ 2`
Not equal to	`!=`	`5 != 2`
Not `x`	`!x`	`!TRUE`
`x` or `y`	`x | y`	`4 > 2 | 4 < 2`
`x` and `y`	`x & y`	`4 > 2 & 4 < 2`
`%in%`	Checks if a value is in a set	`2 %in% c(2, 3, 4)`

Logical operators are particularly powerful when applied across large datasets. For instance, you can use them to filter or subset data based on specific conditions.

Avoiding Common Pitfalls with Logical Operators

One common mistake when working with logical operations is confusing the single equal sign as an assignment operator (`=`) with the equality operator (`==`). For example:

```r
# Incorrect usage leading to an error
# 2 = 4
## Error: R reads this as an assignment
```

```
# Correct usage for a logical comparison
# 2 == 4
## FALSE: R checks if 2 is equal to 4
```

We've now covered the basics of arithmetic and logical operations in R—essential skills you'll use frequently in your data science journey. Make sure to keep practicing these concepts to strengthen your understanding and boost your confidence in R!