

# AI and Biotechnology/Bioinformatics

## R Crash Course (2025)

### Session 4: Basics of R Programming Language Part III

#### *(Operators in R)*


Welcome Back to Your R Programming Course!

---

#### Quick Recap:

In the previous sessions, we covered the foundational steps to start coding in R:

- Installed R and RStudio, and set up your environment
- Learned basic R syntax
- Explored conditional logic using `if` and `if-else`
- Built simple user-defined functions

You can find all the practice scripts, datasets, and resources in the  **GitHub repository** associated with this course.

Today, we'll understand another essential component of R – operators.

---

## Operators in R

---

In programming, operators are special symbols or keywords that instruct the computer to perform specific operations on variables or values.

They serve as the fundamental building blocks for:

- Assigning values
- Expressing logic
- Performing calculations
- Comparing data

Operators are essential for writing functional and executable code,enable to manipulate data and control program flow efficiently.

---

## Understanding Operators in R

Operators in R are like a set of instructions that tell the R what action to perform on values or variables. While **syntax** refers to the basic grammar rules of the programming language, **operators** provide the structure needed to write meaningful code that R can understand and execute

Imagine a teacher says to a student:

“I want you to add 2 to 3, and then divide the result by 5.”

This instruction can be communicated verbally or in writing. But in both cases, the teacher and student rely on a common medium of instruction (like English language) so that the message is understood clearly.

In the same way, when writing code in R, operators act as the common language between the user and R. They help R understand what operation needs to be performed and how to carry it out.

In R,one would write this as:

```
(2 + 3) / 5
```

- `+` is the arithmetic operator for addition
- `/` is the operator for division
- The parentheses ensure the addition happens before division (order of operations)

---

## Types of Operators in R

---

There are four main types of operators in R:

1. Assignment Operators
2. Arithmetic Operators
3. Relational Operators
4. Logical Operators

---

### Assignment Operators

Used to assign values to variables.

```
# <-   (Most common; rightward assignment)
# ->   (Same as above, but leftward assignment)
# <<-  (Global assignment)
# ->>  (Global assignment – reversed version)
# =     (Also assigns, used in function arguments)
```

## Examples

### 1. <- (Most common assignment operator)

- Direction: From right to left.
- Recommended and most used in R.

```
x <- 10
```

### 2. -> (Same as <- but reversed)

- Direction: From left to right.
- Rare; used for style or special cases.

```
10 -> x
```

### 3. <<- (Global assignment)

- Assigns a value outside the current function (to the global environment or parent environment).
- **Use case:** When you want to change a variable outside the function (not common in clean coding).

```
myfunc <- function() {
  x <<- 5 # Assigns x global environment
}
myfunc()
print(x) # x is available outside the function
```

### 4. ->> (Also a global assignment, but reversed))

- Same as <<- but assignment is from left to right.
- Uses very rarely.

```
5 ->> x # Same as x <<- 5
```

### 5. = (Also assigns, used in functions)

```
x = 10

mean(x = c(1, 2, 3))
```

# Difference Between `<-` and `<<-` (Global vs Local Assignment)

---

In R, assignment operators define where a variable will be stored:

`<-` Local (default) → Creates or updates a variable in the current environment. If used inside a function, it stays local.

`<<-` Global/Parent → Assigns a variable in the parent environment (e.g., global environment if used in a function).

This makes the variable available outside the function.

---

## Example: Local Assignment with `<-`

```
my_func <- function() {  
  a <- 10 # a exists only inside this function  
}  
my_func()  
print(a) # Error! a doesn't exist in the global environment  
# a only lives inside the local environment created for my_func
```

OR

```
b <- 5  
my_func <- function() {  
  print(x) # R doesn't find b inside the function, so it checks the  
            parent (global)  
}  
my_func() # Output: 5  
print(b) # Output: 5, already in global environment
```

## Example: Global Assignment with `<<-`

Normally, variables created inside a function don't affect the outside.

But if you use `<<-`, you force R to look outside the function to assign the value either in the parent or global environment.

```
my_func <- function() {  
  b <<- 20 # This puts b into the global environment  
}  
my_func()  
print(b) # Works! b is now global  
# b is available outside the function!
```

But this is not recommended in most cases because it can make your code confusing or buggy.

### Summary:

- Use `<-` when you want variables to stay local and predictable.
- Use `<<-` only when you intentionally want to affect a variable outside the current environment.

**Note:** Overusing `<<-` can make debugging hard and is not recommended in most programming situations. Prefer returning values from functions instead.

**Environment:** An environment in R is like a workspace or a storage area where variables live.

**Global Environment(.GlobalEnv):** This is your main workspace where you type and run your code in the R console or script.

```
# When you do this:
x <- 10
# You're storing the variable x in the global environment

# You can view what's in it using:
ls()
```

**Local Environment (inside functions):** When you run a function, R creates a new local environment to hold the function's temporary variables.

Example:

```
my_func <- function() {
  y <- 20 # y exists only inside this function
}
my_func()
print(y) # Error! y is not in the global environment
y only lives inside the local environment created for my_func
```

### Parent Environment:

Every environment (except the global one) has a parent.

If R can't find a variable in the current environment, it looks in the parent, and so on, until it reaches the global environment.

```
environment() # Where am I now?
parent.env(environment()) # What's my parent?
```

## Arithmetic Operators

Used for mathematical calculations. You can use R as giant calculator.

Operator	Function	Example
+	Addition	3 + 2
-	Subtraction	5 - 1
*	Multiplication	4 * 3
/	Division	10 / 2
^	Exponentiation	2 ^ 3

```
(3 + 3 - 2) / 5
```

```
# OR
```

```
result <- 3 + 4 * 6 / 7  
print(result)
```

## Relational Operators (Comparison)

Used to compare values.

Operator	Decription	Example
==	Equal to	x == 5
!=	Not equal to	x != 5
>	Greater than	x > 3
<	Less than	x < 10
>=	Greater or equal	x >= 7
<=	Less or equal	x <= 9

```
# Example data
```

```
data <- data.frame(age = c(70, 80, 85, 75, 85, 86, 80))
```

```
# Exactly 80 years old
```

```
data$age == 80
```

```
# 80 or younger
```

```
data$age <= 80
```

```
# 80 or older
data$age >= 80
```

## Logical Operators

Used to combine multiple conditions.

Operator	Description	Example
&	AND (both TRUE)	<code>x &gt; 5 &amp; y &lt; 10</code>
	OR (either TRUE)	<code>x &gt; 5   y &lt; 10</code>
!	NOT (negation)	<code>!TRUE</code> → <code>FALSE</code>

```
# Example dataset
disease_status <- data.frame(
  Age = c(55, 45, 60, 75, 56, 49),
  Gender = c("Female", "Male", "Female", "Female", "Male", "Female"),
  Country = c("Pakistan", "UK", "USA", "USA", "Pakistan", "UK"),
  Disease = c("Cancer", "Hypertension", "Cancer", "Hypertension", "Cancer", "Cancer")
)

# Age > 50 and Cancer
subset(disease, Age > 50 & Disease == "Cancer")

# Female OR from Pakistan
subset(disease, Gender == "Female" | Country == "Pakistan")

# Female from Pakistan with Hypertension
subset(disease, Gender == "Female" & Country == "Pakistan" & Disease == "Hypertension")

# Female NOT from Pakistan
subset(disease, Gender == "Female" & Country != "Pakistan")

# People not from UK
disease[disease$Country != "UK", ]
```

### Takeaway Points

- Always use `<-` for assignment (preferred style)
- Use `==` for comparison
- Logical operators help you **filter and query data effectively**

That's All for Today!

In this session, we learned:

- What operators are in R
- Different types of operators
- How to use each operator with examples

Want to try it yourself? Get the R script from GitHub

 **[Access GitHub Repository](#)**

---