R programming basics: objects, classes, and operators

Ecological Systems Modeling

Jan 13-17, 2025

Active participation (optional)

- Open RStudio in Jupyter Hub
- In the Files/Plots/etc. pane, navigate to: `$HOME/Labs/Intro_R_part2/`
- Click on `Intro_to_R_part2b_objectsOperators.rmd`
- File should open in the Source pane
- Run the code chunks, add to chunks, or type code in Console

Learning objectives

- Create and modify objects
- Get information about an object’s class and data type
- Use logical, arithmetic, and other types of operators

Create an object

- Everything in R is an **object**
- Objects are stored in the environment (memory)
- An object is created using the `->` operator

```
# Object is not created because there's no assignment
1 + 3
```

```
[1] 4
```

```
# Create an object using "->"
x <- 1 + 3
x
```

```
[1] 4
```

```
# Create a second object
y <- sqrt(x)
y
```

```
[1] 2
```

Create an object

- An object can also be created using the `=` operator
- However, it's considered better etiquette to use `->`
- `=` is used for assigning arguments in functions (more on this later)

```
1 # Create an object using "="
2 y = 1 + 3
3 y
```

```
[1] 4
```

Modify an object

- Objects can be modified
- The previous version of an object is forgotten
- i.e., R only stores the most recent version of the object

```
# Create an object
x <- 1 + 3
x
```

```
[1] 4
```

```
# Modify x
x <- x + 4
x
```

```
[1] 8
```

Types of data: numeric

- Numeric values can be whole numbers or decimal numbers
- A fraction will be converted to a decimal value

```r
# A numeric vector
die_num <- c(1, 2, 3, 4, 5, 6)
die_num
```

```
[1] 1 2 3 4 5 6
```

```r
# Fractions are converted to decimal values
frac_nums <- c(1/4, 2/3, 5/4, 3/9)
frac_nums
```

```
[1] 0.2500000 0.6666667 1.2500000 0.3333333
```

Check the data class

- `class()` returns the class of data

```r
# A numeric vector
die_num <- c(1, 2, 3, 4, 5, 6)

# Check the type of data class
class(die_num)
```

```
[1] "numeric"
```

Check the specific type of data

- `typeof()` returns the specific type of data

```
# Check the specific type of data that you have
# Double means double precision
typeof(die_num)
```

```
[1] "double"
```

Types of data: integer

- R automatically assumes that numbers are numeric
- Specify that you want integers using the `as.integer()` function

```
# An integer vector
die_num <- c(1, 2, 3, 4, 5, 6)
die_int <- as.integer(die_num)
die_int
```

```
[1] 1 2 3 4 5 6
```

Types of data: integer

- Class and specific type of data are both `integer`

```r
# An integer vector
die_num <- c(1, 2, 3, 4, 5, 6)
die_int <- as.integer(die_num)

# Check the type of data class
class(die_int)
```

```
[1] "integer"
```

```r
# Check the specific type of data that you have
typeof(die_int)
```

```
[1] "integer"
```

Types of data: character

- Character data are text (also called “strings”)
- You must surround text with quotation marks

```r
# A character vector
months <- c("Dec", "Apr", "Jan", "Mar")
months
```

```
[1] "Dec" "Apr" "Jan" "Mar"
```

Types of data: character

- Character data are text (also called “strings”)
- You must surround text with quotation marks
- Class and specific type of data are both `character`

```
# A character vector
months <- c("Dec", "Apr", "Jan", "Mar")
months
```

```
[1] "Dec" "Apr" "Jan" "Mar"
```

```
# Check the type of data class
class(months)
```

```
[1] "character"
```

```
# Check the specific type of data that you have
typeof(months)
```

```
[1] "character"
```

Types of data: factor

- Useful when you to display values in a specific order
  - Can start with any vector (e.g., a character vector)

```
# A character vector
# The months are in the wrong order
months <- c("Dec", "Apr", "Jan", "Mar")
months
```

```
[1] "Dec" "Apr" "Jan" "Mar"
```

Types of data: factor

- Useful when you to display values in a specific order
  - Can start with any vector (e.g., a character vector)
  - Define levels

```r
# A character vector
# The months are in the wrong order
months <- c("Dec", "Apr", "Jan", "Mar")

# Create vector of categories (levels)
# Months are in correct order
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
month_levels
```

```
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Types of data: factor

- Useful when you to display values in a specific order
  - Can start with any vector (e.g., a character vector)
  - Define levels
  - Convert data to factor

```r
# A character vector
# The months are in the wrong order
months <- c("Dec", "Apr", "Jan", "Mar")

# Create vector of categories (levels)
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
                  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")

# Convert vector to categorical data
months_fac <- factor(months, levels = month_levels)
months_fac
```

```
[1] Dec Apr Jan Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

Types of data: factor

- Class is `factor`
- Factor variables in R are stored as integers
- So, specific type of data is `integer`

```
# Check the type of data class
class(months_fac)
```

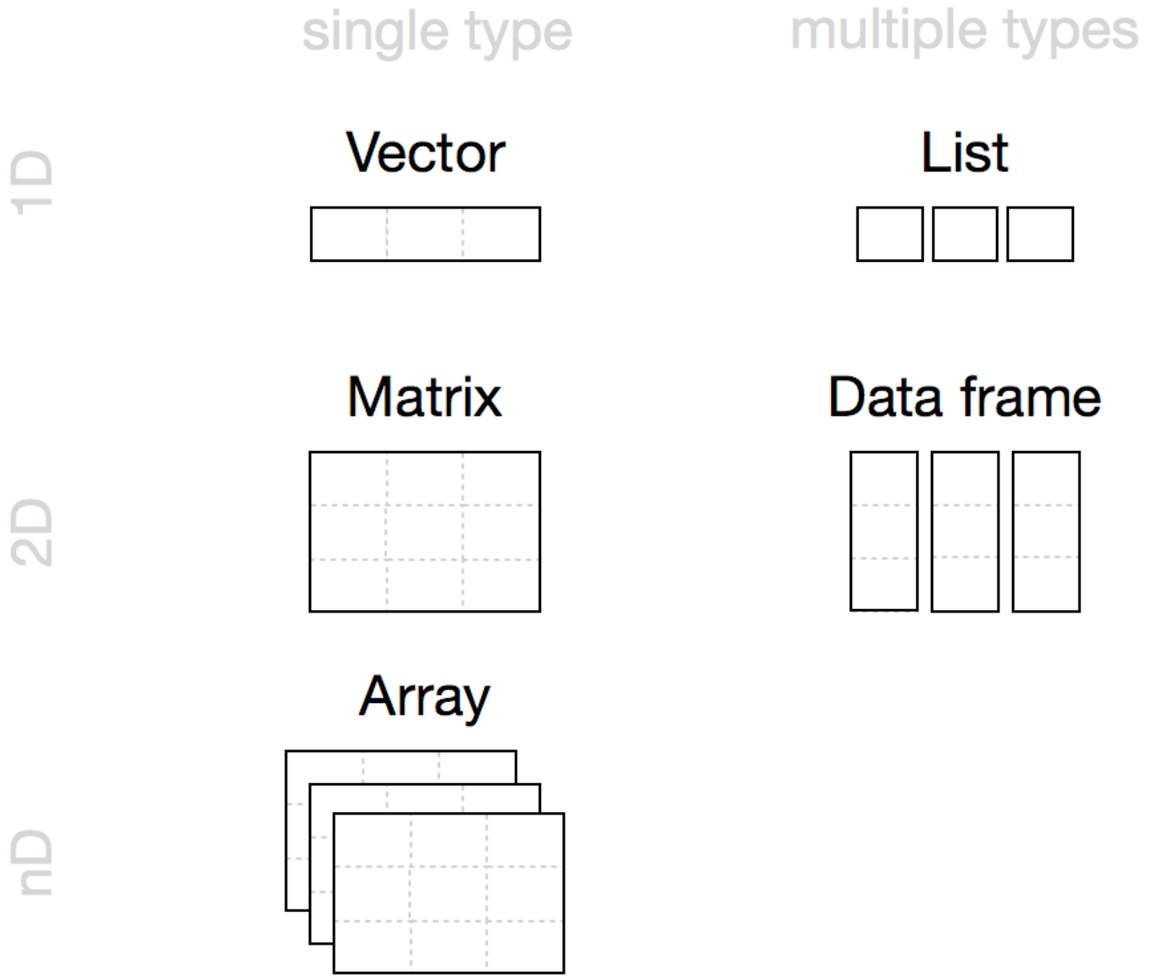
```
[1] "factor"
```

```
# Check the specific type of data that you have
typeof(months_fac)
```

```
[1] "integer"
```

Commonly used objects

- Commonly used objects include
  - **vectors**
  - **matrices**
  - **arrays**
  - **data frames**
  - **lists**



Source: https://rstudio-education.github.io/hopr/r-objects.html

Vectors

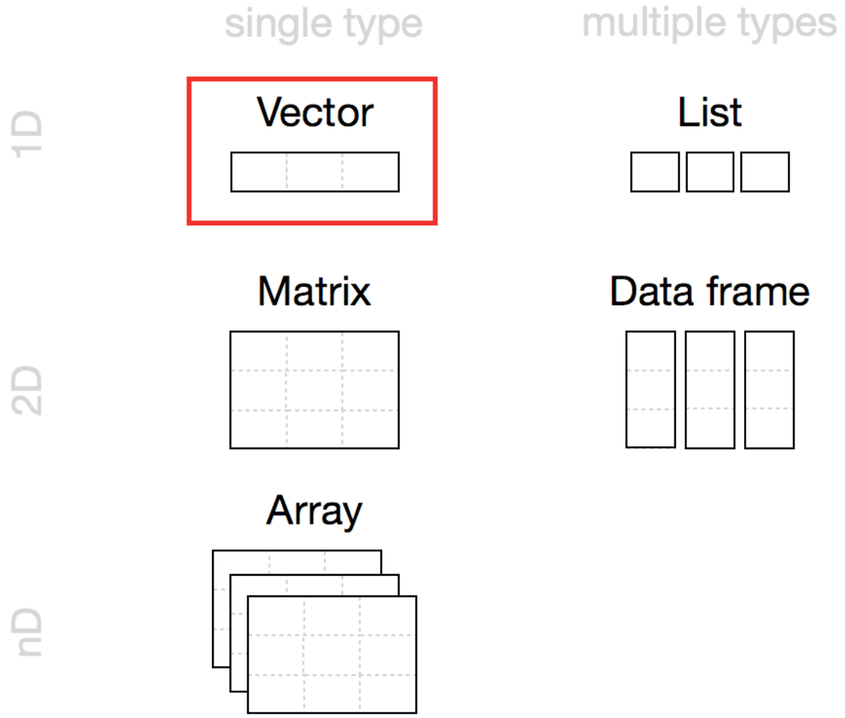
- A vector is 1D (line of values)
- Can only contain a single data type
- Previous slides showed vectors

```
1  # A numeric vector
2  die_num <- c(1, 2, 3, 4, 5, 6)
3  class(die_num)
```

```
[1] "numeric"
```

```
1  # A character vector
2  die_chr <- c("one", "two", "three",
3               "four", "five", "six")
4  class(die_chr)
```

```
[1] "character"
```



Matrix

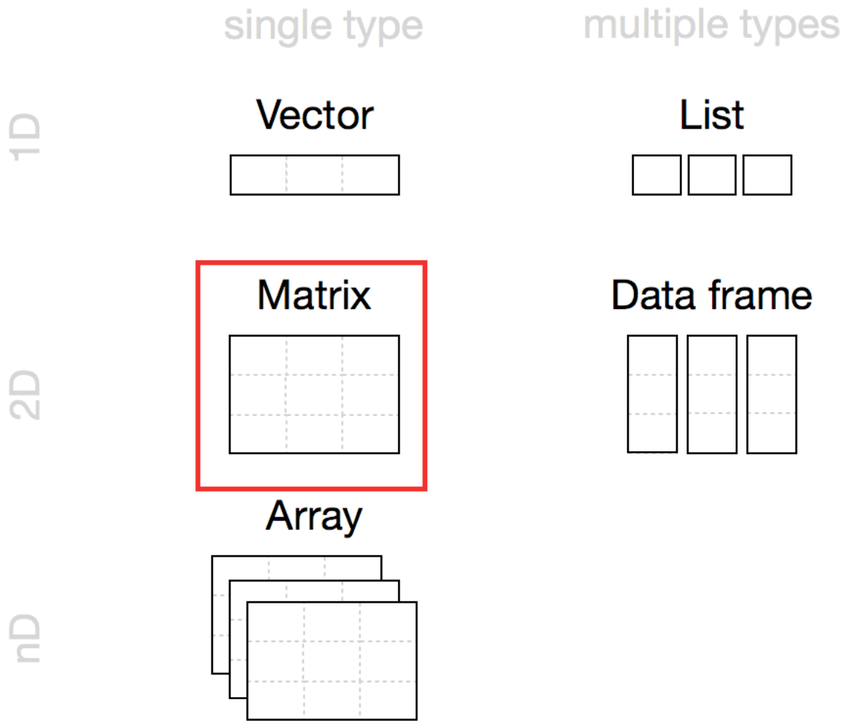
- A matrix is 2D (has columns and rows)
- Can only contain a single data type
- Dimensions set using `nrow` and/or `ncol`

```
# A matrix
die_mat <- matrix(die_num, nrow = 2)
die_mat
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```
# A matrix is a 2D array
class(die_mat)
```

```
[1] "matrix" "array"
```



Array

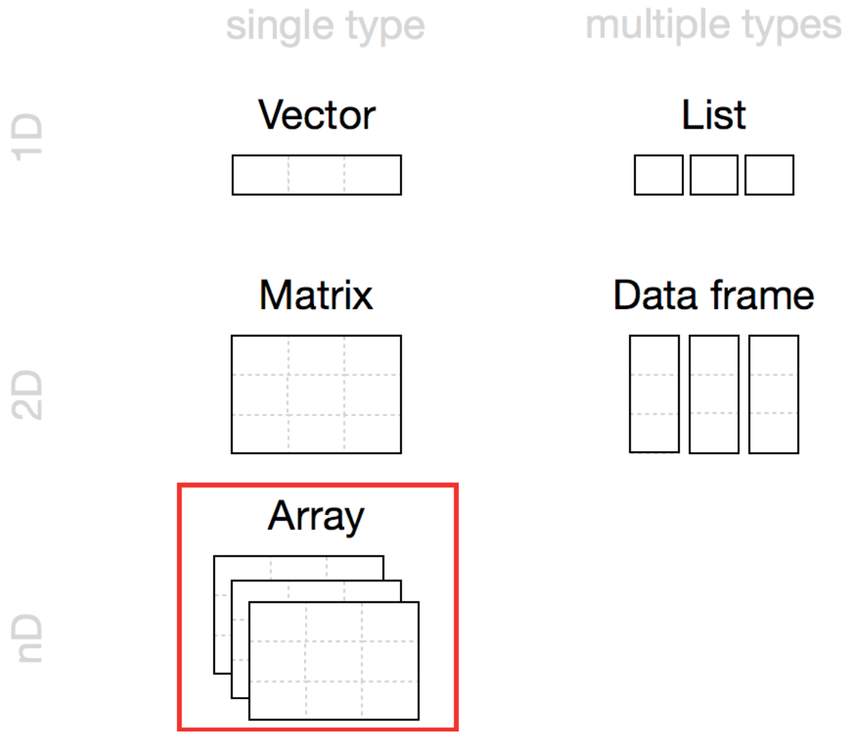
- An array can be nD
- Can combine multiple vectors or matrices
- Dimensions set using `dim`

```
# An array
# Recycles `die_num` 3 times
# Specify dimension (3 rows, 6 columns)
die_array <- array(die_num, dim = c(3,6))
die_array
```

```
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    4    1    4    1    4
[2,]    2    5    2    5    2    5
[3,]    3    6    3    6    3    6
```

```
class(die_array)
```

```
[1] "matrix" "array"
```



Data frames

- A data frame is 2D (has columns and rows)
- Can include vectors with different data types

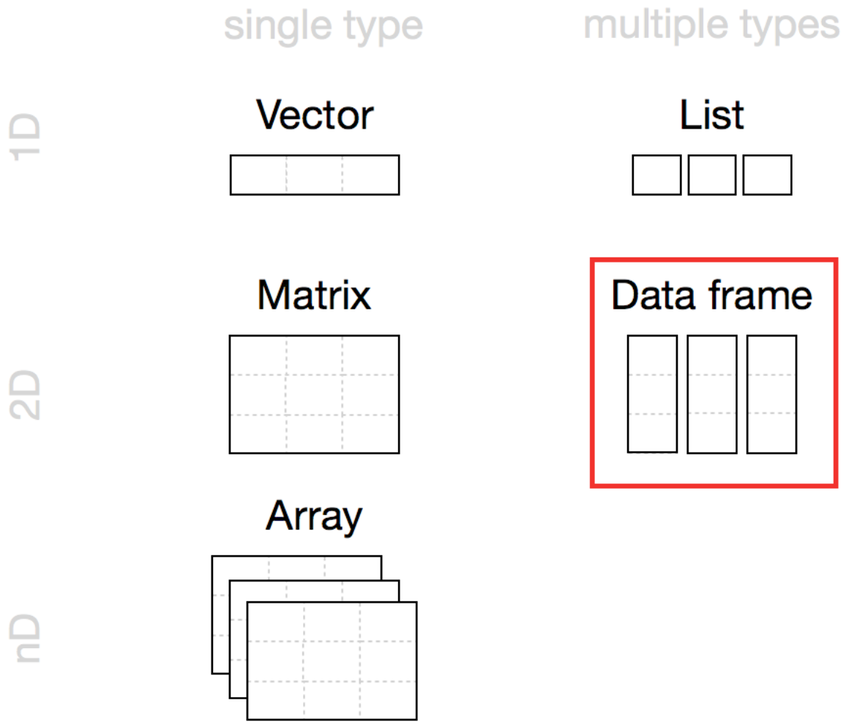
```
# Plant species and perc. cover
species_id <- c("BRTE", "PUTR2", "FESIDA",
                "AGDE", "ARTNOV")
perc_cover <- c(35, 10, 20, 30, 5)

# Make a data frame
spp_cover_df <- data.frame(species_id, perc_cove
spp_cover_df
```

```
  species_id perc_cover
1       BRTE          35
2      PUTR2          10
3     FESIDA          20
4       AGDE          30
5     ARTNOV           5
```

```
class(spp_cover_df)
```

```
[1] "data.frame"
```



Lists

- A list groups data into a 1D set
- However, objects in list can be 1D, 2D, or nD

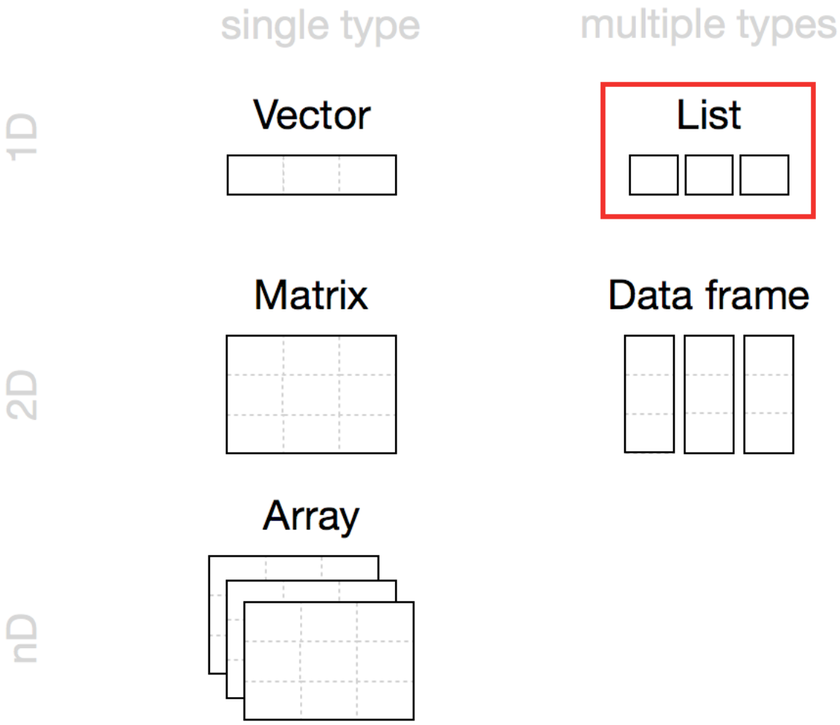
```
# A list containing a matrix and two data frames
a_list <- list(spp_cover_df, months_fac)
a_list
```

```
[[1]]
  species_id perc_cover
1       BRTE          35
2      PUTR2          10
3     FESIDA          20
4       AGDE          30
5     ARTNOV           5

[[2]]
[1] Dec Apr Jan Mar
Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov
```

```
class(a_list)
```

```
[1] "list"
```



Data dimensions

- `dim()` returns the data dimensions
  - Number of rows and columns

```
# Dimensions of "die_mat" (matrix)
dim(die_mat) # 2 rows, 3 cols
```

```
[1] 2 3
```

```
# Dimensions of "die_array" (array)
dim(die_array) # 3 rows, 6 cols
```

```
[1] 3 6
```

```
# Dimensions of "spp_cover_df" data frame
dim(spp_cover_df) # 5 rows, 2 cols
```

```
[1] 5 2
```

Data structure

- `str()` returns the data structure
  - Dimensions, class of data for each column, and data preview

```
# Structure of "die_mat" and "die_array"
str(die_mat) # All numeric data
```

```
num [1:2, 1:3] 1 2 3 4 5 6
```

```
str(die_array)
```

```
num [1:3, 1:6] 1 2 3 4 5 6 1 2 3 4 ...
```

```
# Structure of "spp_cover_df"
# 1 character col, 1 numeric col
str(spp_cover_df)
```

```
'data.frame':   5 obs. of  2 variables:
 $ species_id: chr  "BRTE" "PUTR2" "FESIDA" "AGDE" ...
 $ perc_cover: num  35 10 20 30 5
```

Peeking at objects

- Get a glimpse of objects using the `head()` and `tail()` functions
- Useful when you have a large data frame
- Specify how much to view with `n`

```r
# Create a very long data frame
long_df <- data.frame(x = 1:100, y = 1001:1100)

# Peek at top (first 4 rows)
head(long_df, n = 4)
```

```
  x    y
1 1 1001
2 2 1002
3 3 1003
4 4 1004
```

```r
# Peek at bottom (last 2 rows)
tail(long_df, n = 2)
```

```
      x    y
99   99 1099
100 100 1100
```

Useful data frame functions

- `names()` or `colnames()` returns the column names
- `rownames()` returns the row names
- `nrow()` returns the number of rows
- `ncol()` returns the number of columns

```
# Column names
names(spp_cover_df)
```

```
[1] "species_id" "perc_cover"
```

```
# Row names
rownames(spp_cover_df)
```

```
[1] "1" "2" "3" "4" "5"
```

```
# Number of columns and rows
ncol(spp_cover_df)
```

```
[1] 2
```

```
nrow(spp_cover_df)
```

```
[1] 5
```

Useful data frame functions

- `min()` and `max()` value in a column
- Column is indicated by `$` symbol
- `summary()` provides summary statistics for entire data frame

```
1 spp_cover_df
```

```
  species_id perc_cover
1       BRTE          35
2      PUTR2          10
3     FESIDA          20
4       AGDE          30
5     ARTNOV           5
```

```
1 # Min perc. cover
2 min(spp_cover_df$perc_cover)
```

```
[1] 5
```

```
1 # Max perc. cover
2 max(spp_cover_df$perc_cover)
```

```
[1] 35
```

```
1 # Summary
2 summary(spp_cover_df)
```

```
 species_id          perc_cover   
Length:5           Min.   : 5.0  
Class :character   1st Qu.:10.0  
Mode  :character   Median :20.0  
                   Mean   :20.0  
                   3rd Qu.:30.0  
                   Max.   :35.0  
```

Arithmetic operators

- Perform arithmetic on vectors

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

```
1  # Create an object
2  x <- 15
3  # Addition
4  x + 5
```

```
[1] 20
```

Arithmetic operators

- Perform arithmetic on vectors

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

```
1 # Create an object
2 x <- 15
3 # Addition
4 x + 5
```

```
[1] 20
```

```
1 # Subtraction
2 x - 5
```

```
[1] 10
```

Arithmetic operators

- Perform arithmetic on vectors

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

```
# Create an object
x <- 15
# Addition
x + 5
```

```
[1] 20
```

```
# Subtraction
x - 5
```

```
[1] 10
```

```
# Multiplication
x * 5
```

```
[1] 75
```

Arithmetic operators

- Perform arithmetic on vectors

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

```
# Create an object
x <- 15
# Addition
x + 5
```

```
[1] 20
```

```
# Subtraction
x - 5
```

```
[1] 10
```

```
# Multiplication
x * 5
```

```
[1] 75
```

```
# Exponentiation
x ^ 5
```

```
[1] 759375
```

Arithmetic operators

- Perform arithmetic on vectors

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
x %% y	modulus (x mod y) 5%%2 is 1
x %/% y	integer division 5%/%2 is 2

```
1 # Create an object
2 x <- 15
3 # Addition
4 x + 5
```

```
[1] 20
```

```
1 # Subtraction
2 x - 5
```

```
[1] 10
```

```
1 # Multiplication
2 x * 5
```

```
[1] 75
```

```
1 # Exponentiation
2 x ^ 5
```

```
[1] 759375
```

```
1 # Modulus
2 x %% 5
```

```
[1] 0
```

Relational operators

- Returns either `FALSE` (`0`) or `TRUE` (`1`)

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x

```
1  # A vector of numeric values
2  x <- c(5, 6, 7, 8, 9, 10)
3
4  # 5-9 are < 10
5  x < 10
```

```
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

Relational operators

- Returns either `FALSE` (`0`) or `TRUE` (`1`)

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x

```
# A vector of numeric values
x <- c(5, 6, 7, 8, 9, 10)

# All values are <= 10
x <= 10
```

```
[1] TRUE TRUE TRUE TRUE TRUE TRUE
```

Relational operators

- Returns either `FALSE` (`0`) or `TRUE` (`1`)

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x

```
# A vector of numeric values
x <- c(5, 6, 7, 8, 9, 10)

# No values are >10
x > 10
```

```
[1] FALSE FALSE FALSE FALSE FALSE FALSE
```

Relational operators

- Returns either `FALSE` (`0`) or `TRUE` (`1`)

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x

```
1  # A vector of numeric values
2  x <- c(5, 6, 7, 8, 9, 10)
3
4  # 10 is >=10
5  x >= 10
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

Relational operators

- Returns either `FALSE` (`0`) or `TRUE` (`1`)

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x

```
# A vector of numeric values
x <- c(5, 6, 7, 8, 9, 10)

# Only 10 == 10
x == 10
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

Relational operators

- Returns either `FALSE` (`0`) or `TRUE` (`1`)

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x

```
1  # A vector of numeric values
2  x <- c(5, 6, 7, 8, 9, 10)
3
4  # Only 10 == 10
5  x == 10
```

```
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
```

Logical operators

- **AND** operator: &
  - TRUE returned if all conditions are met

```
# A vector of numeric values
x <- c(5, 6, 7, 8, 9, 10)

# Is 6 and 9 in x?
6 & 9 %in% x
```

```
[1] TRUE
```

```
# Is 6 and 11 in x?
6 & 11 %in% x
```

```
[1] FALSE
```

Logical operators

- **AND** operator: &
  - TRUE returned if all conditions are met

```
# A vector of numeric values
x <- c(5, 6, 7, 8, 9, 10)

# Is 6 and 9 in x?
6 & 9 %in% x
```

```
[1] TRUE
```

```
# Is 6 and 11 in x?
6 & 11 %in% x
```

```
[1] FALSE
```

- **OR** operator: |
  - TRUE returned if any conditions are met

```
# A vector of numeric values
x <- c(5, 6, 7, 8, 9, 10)

# Is 6 or 9 in x?
6 | 9 %in% x
```

```
[1] TRUE
```

```
# Is 6 or 11 in x?
6 | 11 %in% x
```

```
[1] TRUE
```

Logical operators

- **NOT** operator: !

```
# Values in x != 10
x != 10
```

```
[1]  TRUE  TRUE  TRUE  TRUE  TRUE FALSE
```

```
# Values in x != 6
x != 6
```

```
[1]  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

- **IN** operator: %:%
  - Identify if an element belongs to a vector

```
# Is 6 in x?
6 %in% x
```

```
[1] TRUE
```

```
# Is 20 in x?
20 %in% x
```

```
[1] FALSE
```

Other common operators

- Assignment operators: `<-` and `=`

```
x <- 1 + 3
x = 1 + 3
```

- Colon operator: `:`
  - Create a sequence of numbers

```
# Shortcut for typing c(5, 6, 7, 8, 9, 10)
x <- c(5:10)
x
```

```
[1]  5  6  7  8  9 10
```