R Tutorial

R is a programming language used for statistical computing and data visualization.

Example:

```
"Hello World!"
5 + 5
plot(1:10)
```

Result:

- `"Hello World!"`
- `10` (calculation)
- A simple graph (plot)

Practice with **exercises, quizzes, and examples** at W3Schools! □

R Introduction

R is a popular language for **data analysis, visualization, and machine learning**. It offers **statistical techniques, graphing tools, cross-platform support, and a vast library of packages**—all for free! □

No prior programming

R Get Started

Download R from cloud.r-project.org and install it on **Windows, Mac, or Linux**.

Try **5 + 5** in R to see **10** as output.

Learn R with interactive examples at W3Schools! □

R Syntax

Use quotes for text: `"Hello World!"`
Use numbers directly: `5, 10, 25`
Do calculations: `5 + 5` → **10**

Congrats on writing your first R code! □

R Print Output

R can output text directly: `"Hello World!"`
Or use `print()`: `print("Hello World!")`

Use `print()` inside loops:

```
for (x in 1:10) {
  print(x)
}
```

Tip: Use `print()` in expressions like loops! □

R Comments

Use # to add comments in R:

```
# This is a comment
"Hello World!"
```

At the end of a line:

```
"Hello World!"  # This is a comment
```

Disable code execution:

```
# "Good morning!"
"Good night!"
```

For multiline comments, use # on each line. □

R Variables

Assign values using <- or =:

```
name <- "John"
age <- 40
name  # Outputs "John"
age   # Outputs 40
```

Use `print()` when needed, especially in loops:

```
for (x in 1:10) {
  print(x)
}
```

□ **Tip:** <- is preferred over = in most cases!

R Variables

- Assign values using <- or =:
- `name <- "John"`
- `age <- 40`
- `name  # Outputs "John"`
- `age   # Outputs 40`
- <- is preferred over =.
- Use `print()` when required (e.g., inside loops):

```
• for (x in 1:10) {
•   print(x)
• }
```

□ **Tip:** Directly type variable names to print their values!

R Multiple Variables

Assign the same value to multiple variables in one line:

```
var1 <- var2 <- var3 <- "Orange"

var1  # Outputs "Orange"
var2
var3
```

□ **Tip:** Use this to simplify assignments!

R Variable Names

Rules:
✅Must start with a letter, can include letters, digits, _, .

✅Case-sensitive (`age`, `Age`, `AGE` are different)
❌Cannot start with a number or _
❌Reserved words (e.g., `TRUE`, `NULL`) are not allowed

Examples:
✔️ `my_var <- "John"`
✔️ `.myvar <- "John"`
❌`2myvar <- "John"`
❌`my var <- "John"`

R Data Types

R variables can change types dynamically.

Basic Data Types:

- **Numeric**: `10.5`, `55`
- **Integer**: `1L`, `100L`
- **Complex**: `9 + 3i`
- **Character (String)**: `"R is exciting"`
- **Logical (Boolean)**: `TRUE`, `FALSE`

Use `class(x)` to check a variable's type.

R Numbers

R has three number types: **numeric, integer, and complex**.

Examples:

- **Numeric**: `x <- 10.5`
- **Integer**: `y <- 10L` (use `L` for integers)
- **Complex**: `z <- 3+5i`

Type Conversion:

- `as.numeric()`,`as.integer()`, `as.complex()`

R Math

R supports basic math operations:

- Addition: `10 + 5`
- Subtraction: `10 - 5`

Built-in Math Functions:

- **Min/Max:** `min(5,10,15)`, `max(5,10,15)`

- **Square root:** `sqrt(16)`
- **Absolute value:** `abs(-4.7)`
- **Rounding:** `ceiling(1.4)`, `floor(1.4)`

R Strings

- Strings can be in **single** or **double** quotes: `"hello"`, `'hello'`.
- Assign a string: `str <- "Hello"`
- Multiline strings: Use `cat(str)` to maintain line breaks.
- **String functions:**
  - Length: `nchar("Hello World!")`
  - Search: `grepl("Hello", str)`
  - Concatenate: `paste("Hello", "World")`

R Escape Characters

- Use \ to insert special characters in strings.

- Example: `str <- "We are the so-called \"Vikings\", from the north."`
- Use `cat(str)` to print without backslashes.

Common Escape Characters:

- `\\` → Backslash
- `\n` → New Line
- `\r` → Carriage Return
- `\t` → Tab
- `\b` → Backspace

R Booleans (Logical Values)

- Expressions return `TRUE` or `FALSE`.
- Example:
  - `10 > 9` → `TRUE`
  - `10 == 9` → `FALSE`
- Compare variables:
- `a <- 10`
- `b <- 9`
- `a > b  # TRUE`

- Used in conditions:

```
if (b > a) print("b is greater")
else print("b is not greater")
```

R Operators

- **Arithmetic**: + (add), - (subtract), * (multiply), / (divide), ^ (exponent), %% (modulus), %/% (integer division).
- **Assignment**: <-, <<-, ->, ->> assign values to variables.
- **Comparison**: == (equal), != (not equal), >, <, >=, <=.
- **Logical**: &, && (AND), |, || (OR), ! (NOT).
- **Miscellaneous**: **:** (sequence), %in% (check element in vector), %*% (matrix multiplication).

R If...Else

- **Operators**: ==, !=, >, <, >=, <= for conditions.

- **if Statement**: Executes code if condition is `TRUE`.
- **else if**: Checks another condition if the first is `FALSE`.
- **else**: Executes if no conditions are met.

Example:

```
a <- 200
b <- 33

if (b > a) {
  print("b is greater")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater")
}
```

R Nested If

- **Nested if**: An `if` statement inside another `if`.

Example:

```
x <- 41

if (x > 10) {
  print("Above ten")
  if (x > 20) {
    print("Also above 20!")
  } else {
    print("But not above 20.")
  }
} else {
  print("Below 10.")
}
```

R AND & OR Operators

- **& (AND)**: Both conditions must be `TRUE`.
- **| (OR)**: At least one condition must be `TRUE`.

Example:

```
if (a > b & c > a) {
  print("Both conditions are
true")
}
```

```
if (a > b | a > c) {
  print("At least one condition
is true")
}
```

R While Loop

- **`while` loop**: Runs as long as the condition is `TRUE`.
- `i <- 1`
- `while (i < 6) {`
- `  print(i)`
- `  i <- i + 1`
- `}`
- **`break`**: Stops the loop when condition is met.
- `if (i == 4) break`
- **`next`**: Skips an iteration without stopping the loop.
- `if (i == 3) next`
- **Example - Yahtzee Game:**
- `while (dice <= 6) {`

- `  if (dice < 6) print("No Yahtzee")`
- `  else print("Yahtzee!")`
- `  dice <- dice + 1`

`}`

R For Loop

- **Basic Loop**: Iterates over a sequence.
- `for (x in 1:10) print(x)`
- **Looping Through a List:**
- `fruits <- list("apple", "banana", "cherry")`
- `for (x in fruits) print(x)`
- **`break`**: Stops the loop at a condition.
- `if (x == "cherry") break`
- **`next`**: Skips an iteration.
- `if (x == "banana") next`
- **Yahtzee Example:**
- `for (x in 1:6) {`
- `  if (x == 6) print("Yahtzee!")`
- `  else print("Not Yahtzee")`
- `}`
- **R Nested Loops**

- Loops inside loops iterate over multiple sequences.
- **Example:** Print adjectives with fruits.

```
adj <- list("red", "big", "tasty")
fruits <- list("apple", "banana", "cherry")

for (x in adj) {
  for (y in fruits) {
    print(paste(x, y))
  }
}

```

- **R Function Recursion**
- Recursion allows a function to call itself, useful for iterative calculations.
- **Example:** Sum numbers recursively.

```
tri_recursion <- function(k) {
  if (k > 0) {
    result <- k + tri_recursion(k - 1)
    print(result)
  } else {
    return(0)
  }
}
tri_recursion(6)
```

- Recursion stops when `k` reaches 0.
-
- **R Global Variables**
- Global variables are accessible inside and outside functions.
- **Example:** Using a global variable inside a function.

```
txt <- "awesome"
my_function <- function() {
  paste("R is", txt)
}
my_function()
```

- Local variables inside functions do not affect global ones.
- **Global Assignment Operator (`<<-`)** Allows modifying global variables inside functions.

- `txt <- "awesome"`
- `my_function <- function() {`
- `  txt <<- "fantastic"`
- `}`
- `my_function()`
- `print(txt)  # Outputs "fantastic"`

R Vectors

A **vector** is a list of items of the same type, created using `c()`.

Examples:

```
fruits <- c("banana", "apple", "orange")  # String vector
numbers <- c(1, 2, 3)  # Numeric vector
seq_vec <- 1:10  # Sequence vector
log_values <- c(TRUE, FALSE, TRUE)  # Logical vector
```

Vector Operations:

- **Length:** `length(fruits)`
- **Sort:** `sort(fruits)`
- **Access:** `fruits[1]` (First item), `fruits[c(1,3)]` (Multiple items), `fruits[-1]` (Exclude first item)
- **Modify:** `fruits[1] <- "pear"`
- **Repeat:**
- `rep(c(1,2,3), each = 3)  # Repeat each`
- `rep(c(1,2,3), times = 3)  # Repeat sequence`
- **Sequence Generation:**
- `seq(from = 0, to = 100, by = 20)`

R Vectors

A **vector** is a list of items of the same type, created using `c()`.

Examples:

```
fruits <- c("banana", "apple",
"orange")  # String vector
```

```
numbers <- c(1, 2, 3)  # Numeric vector
seq_vec <- 1:10  # Sequence vector
log_values <- c(TRUE, FALSE, TRUE)  # Logical vector
```

Vector Operations:

- **Length:** `length(fruits)`
- **Sort:** `sort(fruits)`
- **Access:** `fruits[1]` (First item), `fruits[c(1,3)]` (Multiple items), `fruits[-1]` (Exclude first item)
- **Modify:** `fruits[1] <- "pear"`
- **Repeat:**
- `rep(c(1,2,3), each = 3)  # Repeat each`
- `rep(c(1,2,3), times = 3)  # Repeat sequence`
- **Sequence Generation:**
- `seq(from = 0, to = 100, by = 20)`

R Lists

- **Remove Item:** `newlist <- thislist[-1]` (Removes first item)
- **Range of Indexes:** `thislist[2:5]` (Returns items 2 to 5)
- **Loop Through List:**
- `for (x in thislist) { print(x) }`
- **Join Lists:**
- `list3 <- c(list1, list2)`

R Matrices

- **Create Matrix:** `matrix(c(1,2,3,4,5,6), nrow=3, ncol=2)`
- **Access Items:** `thismatrix[1,2]` (Row 1, Col 2)
- **Access Row/Column:** `thismatrix[2,]` (Row) | `thismatrix[,2]` (Column)
- **Multiple Rows/Cols:** `thismatrix[c(1,2),]` | `thismatrix[,c(1,2)]`

- **Add Row/Col:** `rbind()|cbind()`
- **Remove Row/Col:** `thismatrix[-c(1), -c(1)]`
- **Check Item:** `"apple" %in% thismatrix`
- **Matrix Size:** `dim(thismatrix)|length(thismatrix)`
- **Loop:**
- `for (r in 1:nrow(thismatrix)) { for (c in 1:ncol(thismatrix)) print(thismatrix[r, c]) }`
- **Combine Matrices:** `rbind(Matrix1, Matrix2)|cbind(Matrix1, Matrix2)`

R Factors Summary

Factors are used to categorize data. Examples:

- **Demography:** Male/Female
- **Music:** Rock, Pop, Jazz, Classic
- **Training:** Strength, Stamina

Creating a Factor

```
music_genre <- factor(c("Jazz",
"Rock", "Classic", "Classic",
"Pop", "Jazz", "Rock", "Jazz"))
print(music_genre)
```

Levels: Classic, Jazz, Pop, Rock

Getting Factor Levels

```
levels(music_genre)
```

Setting Custom Levels

```
music_genre <-
factor(music_genre, levels =
c("Classic", "Jazz", "Pop",
"Rock", "Other"))
```

Factor Length

```
length(music_genre)  # Output: 8
```

Access & Modify Factors

```
music_genre[3]  # Access
music_genre[3] <- "Pop"  #
Modify
```

□ **Note:** You can only assign predefined levels.

R Plotting Summary

Basic Plotting

- `plot(x, y)` → Plots points at given coordinates.
- Example:
- `plot(c(1, 8), c(3, 10))  # Two points at (1,3) and (8,10)`

Multiple Points

- Ensure equal number of x & y values:
- `x <- c(1, 2, 3, 4, 5)`
- `y <- c(3, 7, 8, 9, 12)`
- `plot(x, y)`

Sequences

- `plot(1:10)` → Plots numbers 1 to 10.

Drawing a Line

- `plot(1:10, type="l")` → Connects points with a line.

Adding Labels

```
plot(1:10, main="My Graph",
xlab="X-Axis", ylab="Y-Axis")
```

Customizing Appearance

- **Color:** `plot(1:10, col="red")`
- **Size:** `plot(1:10, cex=2)` (Larger points)
- **Shape:** `plot(1:10, pch=25, cex=2)` (Change point shape)

R Line Graph Summary

Basic Line Plot

- `plot(1:10, type="l")` → Draws a line graph.

Customizing Lines

- **Color:** `plot(1:10, type="l", col="blue")`

- **Width:** `plot(1:10, type="l", lwd=2)`
- **Style:** `plot(1:10, type="l", lty=3)` (Dotted line)

Line Styles (`lty` values)

- `1` Solid (default)
- `2` Dashed
- `3` Dotted
- `4` Dot-dash
- `5` Long dash
- `6` Two dashes

Multiple Lines

```
line1 <- c(1,2,3,4,5,10)
line2 <- c(2,5,7,8,9,10)

plot(line1, type="l",
col="blue")
lines(line2, type="l",
col="red")
```

R Scatter Plot Summary

Basic Scatter Plot

- `plot(x, y)` → Plots dots for two numerical variables.

Adding Labels & Title

```
plot(x, y, main="Observation of
Cars", xlab="Car age", ylab="Car
speed")
```

Comparing Two Datasets

```
plot(x1, y1, main="Observation
of Cars", xlab="Car age",
ylab="Car speed", col="red",
cex=2)
points(x2, y2, col="blue",
cex=2)
```

- **Red**: Day 1
- **Blue**: Day 2

Conclusion

- **Newer cars tend to drive faster.**

R Pie Charts Summary

Basic Pie Chart

```
x <- c(10,20,30,40)
pie(x)
```

Start Angle

```
pie(x, init.angle = 90)  #
Starts at 90°
```

Adding Labels & Title

```
mylabel <- c("Apples",
"Bananas", "Cherries", "Dates")
pie(x, label = mylabel, main =
"Fruits")
```

Adding Colors

```
colors <- c("blue", "yellow",
"green", "black")
pie(x, label = mylabel, main =
"Fruits", col = colors)
```

Adding a Legend

```
legend("bottomright", mylabel,
fill = colors)
```

Legend Positions: bottomright, bottom, bottomleft, left, top, topleft, topright, right, center.

R Pie Charts (Short Summary)

```
x <- c(10,20,30,40)
pie(x)  # Basic Pie Chart
pie(x, init.angle = 90)  # Start
at 90°

mylabel <- c("Apples",
"Bananas", "Cherries", "Dates")
pie(x, label = mylabel, main =
"Fruits")  # Add Labels & Title

colors <- c("blue", "yellow",
"green", "black")
pie(x, label = mylabel, main =
"Fruits", col = colors)  # Add
Colors
```

```
legend("bottomright", mylabel,
fill = colors)  # Add Legend
```

Legend Positions: bottomright, bottom, left, top, topright, center.

R Data Set: mtcars

The **mtcars** dataset, from the 1974 *Motor Trend* US magazine, contains fuel consumption and performance metrics for **32 cars** (1973-74 models).

Key Features:

- **Columns (11):** mpg, cyl, disp, hp, drat, wt, qsec, vs, am, gear, carb
- **Usage:** `mtcars`
- **Info:** `?mtcars`
- **Functions:**
    - `dim(mtcars)`: Dimensions
    - `names(mtcars)`: Column names
    - `summary(mtcars)`: Data summary
    - `pairs(mtcars)`: Scatterplot matrix

Example Visualization:

```
pairs(mtcars, main = "mtcars
data", gap = 1/4)
```

R Max and Min (Short Summary)

- **Finding Max & Min:**
  - `max(mtcars$hp)` → 335 (Highest HP: Maserati Bora)
  - `min(mtcars$hp)` → 52 (Lowest HP: Honda Civic)
- **Finding Car Names:**
  - `rownames(mtcars)[which.max(mtcars$hp)]` → "Maserati Bora"
  - `rownames(mtcars)[which.min(mtcars$hp)]` → "Honda Civic"
- **Outliers:**
  - Extreme values like very high gears, weight, or zero HP can indicate outliers.

Mean, Median, and Mode in R

- **Mean**: The average value, calculated as the sum of all values divided by the count.
- **Example**:

- `mean(mtcars$wt)   # Output: 3.21725`
- **Sorted `wt` values**: 1.513, 1.615, ..., 5.424.

Use `mean()` in R to easily find the average! □

Median in R

- **Median**: The middle value in a sorted dataset.
- If two middle values exist, their average is taken.
- **Example**:
- `median(mtcars$wt)   # Output: 3.325`

Use `median()` in R to find it instantly! □

Median in R: Middle value in a sorted dataset. If two, take the average.

```
median(mtcars$wt)   # Output:
3.325
```

Use `median()` in R instantly! □

Percentiles in R: Show values below which a given % falls.

```
quantile(mtcars$wt, 0.75)  # Output: 3.61
```

Use `quantile()` for percentiles & quartiles! □