R Programming: Last-Minute Exam Cheat Sheet

This document is designed for rapid review before your exam. It covers essential core concepts, crucial R functions (and their quirks), and all 20 reserved words.

Crucial Rules to Remember

- **Variable Naming:** Variables can start with letters or a period (`.`). However, if they start with a period, they **cannot** be followed by a digit (`.2var` is illegal).
- **1-Based Indexing:** Unlike C, Python, or Java (where arrays start at 0), **R indices start at 1**. The first element of `vector_x` is `vector_x[1]` .
- **Negative Indexing:** Using a minus sign *excludes* that element. `my_list[-2]` returns the list WITHOUT the 2nd element.
- **Type Coercion:** If you mix data types in a vector (e.g., `c(1, "A", TRUE)`), R will aggressively force them into a single type. The hierarchy is: `logical -> integer -> numeric -> character` .
- **Returning Multiple Values:** A function can only return ONE object. To return multiple answers, bundle them into a list: `return(list("Area"=area, "Perimeter"=peri))` .
- **Assignment:** While `=` works, `<-` is the strictly preferred assignment operator in R. `=` is mainly used for assigning arguments inside function calls.
- **Null vs NA vs NaN:** * `NA` : Missing data (Not Available).
  - `NaN` : Mathematical impossibility (Not a Number, like `0/0`).
  - `NULL` : An empty object / undefined.

Essential Operators

Operator	Name	Category	Description
`<-`	Left Assignment	Assignment	Assigns a value to a variable (Leftward).
`->`	Right Assignment	Assignment	Assigns a value to a variable (Rightward).
`%%`	Modulo	Arithmetic	Modulo (finds the remainder of division, e.g., `5 %% 2 == 1`).
`^`	Exponent	Arithmetic	Exponent/Power (e.g., `2^3 == 8`).
`< / >`	Less/Greater Than	Relational	Strictly less than / strictly greater than.

Operator	Name	Category	Description
`<= / >=`	Less/Greater or Equal	Relational	Less than or equal to / Greater than or equal to.
`== / !=`	Equality / Inequality	Relational	Equals to / Not equals to.
`&`	Element-wise AND	Logical	Element-wise AND (returns TRUE only if both are TRUE).
`&&`	Logical AND	Logical	Evaluates ONLY the first element of a vector. Used mainly in if-statements.
`` ` ``	`` ` ``	Element-wise OR	Logical
`` ` ``		`` ` ``	Logical OR
`!`	Logical NOT	Logical	NOT operator (reverses a boolean, `!TRUE == FALSE`).

Built-In Constants

Constant	Value	Description
`pi`	`3.141593...`	The mathematical constant pi.
`LETTERS`	`"A", "B", "C" ... "Z"`	A character vector of the 26 uppercase English letters.
`letters`	`"a", "b", "c" ... "z"`	A character vector of the 26 lowercase English letters.
`month.abb`	`"Jan", "Feb" ... "Dec"`	A character vector of the 3-letter abbreviations for the English months.
`month.name`	`"January", "February"...`	A character vector of the full names of the English months.

All 20 R Reserved Words (Keywords)

Reserved words have special meaning in R and **cannot** be used as variable or function names.

Reserved Word	Usage / Description	Quick Example
`if`	Starts a conditional control-flow statement.	`if (x > 0) print("Pos")`
`else`	Fallback for an `if` statement when the condition is FALSE.	`else print("Neg")`
`while`	Loop that executes as long as a condition is TRUE.	`while (x < 5) x <- x + 1`
`repeat`	Infinite loop that requires a `break` statement to exit.	`repeat { if(x>5) break }`
`for`	Loop that iterates over a list, vector, or sequence.	`for (i in 1:5) print(i)`
`function`	Keyword used to declare a user-defined function.	`f <- function(x) { x + 1 }`
`in`	Used within `for` loops to specify the sequence to iterate over.	`for (item in my_list)`
`next`	Skips the current loop iteration and moves to the next one.	`if (i %% 2 == 0) next`
`break`	Immediately terminates the execution of a loop.	`if (i == 5) break`
`TRUE`	Logical constant representing True (boolean 1).	`is_valid <- TRUE`
`FALSE`	Logical constant representing False (boolean 0).	`is_valid <- FALSE`
`NULL`	Represents the null object / undefined data.	`my_var <- NULL`
`Inf`	Constant representing positive infinity.	`1 / 0 -> Inf`
`NaN`	Not a Number (e.g., 0/0).	`0 / 0 -> NaN`
`NA`	Logical missing value (Not Available).	`c(1, NA, 3)`
`NA_integer_`	Missing value specifically typed as an integer.	`c(1L, NA_integer_)`
`NA_real_`	Missing value specifically typed as a double/numeric.	`c(1.5, NA_real_)`

Reserved Word	Usage / Description	Quick Example
`NA_complex_`	Missing value specifically typed as a complex number.	`c(1+2i, NA_complex_)`
`NA_character_`	Missing value specifically typed as a character/string.	`c("A", NA_character_)`
`...`	Ellipsis operator used to pass a variable number of arguments into a function.	`f <- function(...) print(list(...))`

Essential Functions Reference

Function	Usage	Example Result-Output	Edge Case / Quirks
`print()`	Prints values and variables to the console.	`print("R") -> [1] "R"`	Printing `NULL` prints `NULL` (without the `[1]` index).
`cat()`	Prints variables and strings concatenated, used with basic types.	`cat("Hi\n") -> Hi`	`cat(NULL)` outputs absolutely nothing, no errors.
`help()`	Views help documentation for functions or keywords.	`help(sum) ->` *(Opens help viewer)*	Searching for special operators requires backticks: `help("+")` .
`class()`	Returns the data type or class of a variable.	`class(TRUE) -> [1] "logical"`	If applied to a mixed vector `c(1, "a")` , returns `"character"` (coercion).
`paste()`	Prints a string and variable together with a default space separator.	`paste("Hi", "R") -> [1] "Hi R"`	`paste("A", NA)` outputs `"A NA"` (NA is converted to a string).
`paste0()`	Prints a string and variable together with NO default space separator.	`paste0("Hi", "R") -> [1] "HiR"`	Recycles shorter vectors: `paste0("A", 1:3) -> "A1" "A2" "A3"` .
`sprintf()`	Prints formatted strings using format specifiers	`sprintf("%d", 123) -> [1]`	Throws an error if the number of arguments

Function	Usage	Example Result-Output	Edge Case / Quirks
	like %s, %d, %f.	`"123"`	doesn't match specifiers.
`is.numeric()`	Evaluates to TRUE if the variable is of numeric type.	`is.numeric(5) -> [1] TRUE`	`is.numeric(NA)` is logical, but `is.numeric(NA_real_)` is TRUE.
`ifelse()`	A shorthand vectorized alternative to the standard if...else statement.	`ifelse(c(1) > 0, "Y", "N") -> [1] "Y"`	If the condition evaluates to `NA`, the output element will be `NA`.
`c()`	Combines or concatenates values into a vector.	`c(1, 2) -> [1] 1 2`	Mix types (e.g. number + string) and everything becomes strings.
`rep()`	Repeats elements of vectors using 'times' or 'each' arguments.	`rep(2, times=2) -> [1] 2 2`	`rep(1, times=0)` returns an empty vector: `numeric(0)`.
`length()`	Finds the number of elements present inside a vector or list.	`length(c(1, 2)) -> [1] 2`	`length(NULL)` evaluates to `0`.
`list()`	Creates a collection of similar or different types of data.	`list(1, "A") -> [[1]] 1 [[2]] "A"`	A list can contain lists inside itself (nested lists).
`append()`	Adds an item at the end of a list or vector.	`append(c(1), 2) -> [1] 1 2`	Does not modify the original variable in-place; must reassign.
`factorial()`	Calculates the factorial value of a number or vector.	`factorial(4) -> [1] 24`	`factorial(0)` returns `1`. `factorial(-1)` returns `NaN`.
`sum()`	Finds the sum of a sequence of numbers.	`sum(1, 2) -> [1] 3`	`sum(c(1, NA))` is `NA`. Use `na.rm=TRUE` to ignore NAs.
`max()` / `min()`	Finds the maximum/minimum value in a sequence of numbers.	`max(4:6) -> [1] 6`	`max(numeric(0))` returns `-Inf` and a warning message.

Function	Usage	Example Result-Output	Edge Case / Quirks
`seq()`	Generates regular sequences (more flexible than the `:` operator).	`seq(1, 5, by=2)` -> `[1] 1 3 5`	`seq(1, 1)` correctly handles start and end being the same.
`mean()`	Calculates the arithmetic mean (average).	`mean(c(2, 4))` -> `[1] 3`	`mean(c(1, NA))` returns `NA`. Requires `na.rm = TRUE`.
`is.na()`	Checks for missing values (`NA`) and returns a logical vector.	`is.na(c(1, NA))` -> `[1] FALSE TRUE`	`is.na(NaN)` also evaluates to `TRUE` in R.
`str()`	Compactly displays the internal structure of an R object.	`str(c(1,2))` -> `num [1:2] 1 2`	Calling `str(NULL)` safely returns `NULL`.
`typeof()`	Determines the internal R type or storage mode of any object.	`typeof(1L)` -> `[1] "integer"`	`typeof(c(1, 2))` returns `"double"`, while `class` is `"numeric"`.
`rm()`	Removes objects/variables from the current workspace environment.	`rm(x)` -> *(Removes x)*	`rm(list = ls())` completely clears the entire workspace.

55 Practice Questions (Basic to Advanced)

Test your knowledge! Click on any question to reveal the answer.

▼ **Q1: What is the primary difference between `<-` and `=` in R?**

> `<-` is the standard assignment operator for variables. `=` is typically used only for assigning values to parameters inside function calls.

▼ **Q2: Can you assign a variable from right to left in R?**

> Yes, R supports rightward assignment using `->` (e.g., `"Hello" -> my_var`).

▼ **Q3: How do you write a true multi-line comment in R?**

> R does not have a native multi-line comment syntax. You must either use `#` on every line or wrap text in a dummy `if (FALSE) { ... }` block.

▼ **Q4: What is the difference between `class(14)` and `class(14L)`?**

> `14` is a floating-point `numeric` by default. The `L` suffix explicitly declares `14L` as an `integer`.

▼ **Q5: Is R case-sensitive? Give an example.**

> Yes. `TRUE` is a valid boolean keyword, but `True` or `true` will throw an 'object not found' error.

▼ **Q6: What happens if you type `T <- FALSE`?**

> It will successfully overwrite `T` to mean `FALSE`. This is why using `TRUE` / `FALSE` is safer than the `T` / `F` shorthands, as the full words are reserved and cannot be overwritten.

▼ **Q7: What is the exact output of `paste("Hello", "R")`?**

> `"Hello R"`. The standard `paste()` function adds a single space between items by default.

▼ **Q8: What is the exact output of `paste0("Hello", "R")`?**

> `"HelloR"`. The `paste0()` function explicitly concatenates with no spaces.

▼ **Q9: If `x <- 12.34`, what does `sprintf("Value: %f", x)` do?**

> It formats the float as a string, outputting `"Value: 12.340000"`.

▼ **Q10: Why can't you use `cat()` to print a list?**

> `cat()` only works with basic atomic types (logical, integer, numeric, character). It cannot handle complex data structures like lists.

▼ **Q11: What happens when you run `c(1, "A", TRUE)`?**

> R coerces them all to the most flexible type (character). The output is `"1" "A" "TRUE"`.

▼ **Q12: What does `is.numeric(NA)` return?**

> `[1] logical`. Standard `NA` is a logical type by default, unless specified as `NA_real_`.

▼ **Q13: How is `ifelse()` different from a standard `if...else` block?**

> `ifelse()` is vectorized, meaning it can evaluate an entire vector of conditions at once and return a vector of results. `if...else` evaluates a single condition.

▼ **Q14: What happens if you pass a vector of length 5 into a standard `if (condition)` statement?**

> The `if` statement will only evaluate the very first element of the vector and will throw a warning.

▼ **Q15: What does the `%%` operator do?**

> It calculates the modulo (remainder of division). E.g., `5 %% 2` equals `1`.

▼ **Q16: When does a `while` loop terminate?**

> When its test condition evaluates to `FALSE`, or when a `break` statement is encountered inside the loop.

▼ **Q17: If a `break` statement is executed inside a nested loop, what happens?**

> It immediately terminates *only the innermost loop* where the `break` was called. The outer loop continues running.

▼ **Q18: What does the `next` statement do?**

> It immediately skips the rest of the current loop iteration and jumps to the evaluation of the next iteration.

▼ **Q19: What is the index of the first element in an R vector?**

> `1`. Unlike Python or C, R uses 1-based indexing.

▼ **Q20: What does negative indexing do, e.g., `my_vector[-2]`?**

> It returns all elements of the vector *except* the element at index 2.

▼ **Q21: What is the output of `length(NULL)`?**

> It returns `0`, because `NULL` represents an empty, undefined object.

▼ **Q22: What is the output of `length(NA)`?**

> It returns `1`. `NA` represents a single missing value, so it still takes up one slot in memory.

▼ **Q23: What is the difference between `1:3` and `c(1, 2, 3)`?**

> Practically nothing in the output, but `:` is a sequence generator operator that efficiently creates the integer sequence without explicitly typing each element.

▼ **Q24: What is the output of `rep(c(1, 2), times = 2)`?**

> `1 2 1 2`. It repeats the entire sequence twice.

▼ **Q25: What is the output of `rep(c(1, 2), each = 2)`?**

> `1 1 2 2`. It repeats each individual element twice before moving to the next.

▼ **Q26: How do you permanently change the 2nd element of `x` to 5?**

> `x[2] <- 5`.

▼ **Q27: What is the fundamental difference between a Vector and a List?**

> A vector can only hold elements of the *same* data type. A list can hold elements of *different* data types (strings, numbers, booleans, or even other lists).

▼ **Q28: Does `append(my_list, 5)` permanently change `my_list`?**

> No. Most functions in R do not modify in-place. You must overwrite the variable: `my_list <- append(my_list, 5)`.

▼ **Q29: How do you extract the actual value of the first element in a list, not just a sub-list?**

> Using double brackets: `my_list[[1]]`. Single brackets `my_list[1]` return a new list containing that element.

▼ **Q30: What keyword is used to create a custom function?**

> `function()`. Example: `my_func <- function(x) { ... }`.

▼ **Q31: What is a formal argument vs an actual argument?**

> Formal arguments are the parameters defined inside the `function()` parentheses. Actual arguments are the actual values passed when the function is called.

▼ **Q32: Can you change the order of arguments when calling a function?**

> Yes, by using named arguments (e.g., `power(b=3, a=2)`).

▼ **Q33: How do you give a parameter a default value?**

> By assigning it in the function definition: `my_func <- function(length = 5) { ... }`.

▼ **Q34: If an R function doesn't use the `return()` statement, what does it return?**

> It implicitly returns the result of the very last evaluated expression in the function body.

▼ **Q35: What is nested function calling?**

> Passing the result of one function directly as an argument into another function. E.g., `print(sum(1, 2))`.

▼ **Q36: What is 'Lazy Evaluation' in R?**

> R only evaluates function arguments when they are actually needed inside the function body.

▼ **Q37: If a function is defined as `f <- function(a, b) { print(a) }`, what happens if you call `f(5)` without providing `b`?**

> It successfully prints `5`. Because `b` is never used in the function, R's lazy evaluation ignores the missing argument.

▼ **Q38: Following the previous question, what happens if the function *does* use `b` later, e.g., `print(a + b)`?**

> R will throw an error only at the exact moment it tries to evaluate `b` and realizes it is missing.

▼ **Q39: What is an inline function?**

> A compact function written on a single line, often without braces. E.g., `f <- function(x) x^2`.

▼ **Q40: What is the output of `sum(4:6)`?**

> `15` (because 4 + 5 + 6 = 15).

▼ **Q41: What happens if you sum a vector containing an `NA`, like `sum(c(1, 2, NA))`?**

> The result is `NA`. You must use the argument `na.rm = TRUE` to ignore missing values.

▼ **Q42: What does `Inf` signify?**

> Positive Infinity, typically resulting from mathematical operations like dividing a positive number by 0.

▼ **Q43: What is the difference between `NaN` and `NA`?**

> `NaN` is 'Not a Number' (a mathematical impossibility like `0/0`). `NA` is a logical placeholder for missing data.

▼ **Q44: Is `NaN` considered an `NA`?**

> Yes, `is.na(NaN)` returns `TRUE`. However, `is.nan(NA)` returns `FALSE`.

▼ **Q45: If `x <- c(1, 2, 3)`, what does `str(x)` do?**

> It displays the compact internal structure of the object: `num [1:3] 1 2 3`.

▼ **Q46: What does `rm(x)` do?**

> It removes the variable `x` from the current workspace environment, freeing up memory.

▼ **Q47: How do you access the documentation/help file for the `sum` function?**

> Type `help(sum)` or `?sum` in the console.

▼ **Q48: How do you search for help regarding operators like `+` or reserved keywords like `if`?**

> You must wrap them in backticks or quotes: `help("+")` or `?"if"`.

▼ **Q49: What does `&` do vs `|` in R?**

> `&` is the logical AND operator (returns TRUE only if both sides are TRUE). `|` is the logical OR operator (returns TRUE if at least one side is TRUE).

▼ **Q50: What does the `!` operator do?**

> It is the logical NOT operator. It reverses booleans, turning `TRUE` to `FALSE` and vice versa.

▼ **Q51: What is the rule for starting a variable name with a period (.)?**

> A variable name can start with a period, but it CANNOT be followed immediately by a digit (e.g., `.myvar` is valid, `.2var` is invalid).

▼ **Q52: What is a Boolean shorthand in R?**

> You can type `T` or `F` instead of typing out `TRUE` or `FALSE`.

▼ **Q53: Are single characters (like "a") and strings (like "Word") considered different classes in R?**

> No, they both belong to the `character` class. R does not have a separate 'char' type like C or Java.

▼ **Q54: What does `%c`, `%e`, and `%u` do in the `sprintf()` function?**

> `%c` formats as a character, `%e` formats in scientific notation, and `%u` formats as an unsigned decimal integer.

▼ **Q55: What is a recursive function?**

> A function that calls itself from within its own code block (e.g., calculating a factorial mathematically).