**Other Built-in Functions in R**

| **Functions** | **Syntax** |
| --- | --- |
| **Mathematical Functions** |  |
| a. [abs()](https://www.geeksforgeeks.org/calculate-the-absolute-value-in-r-programming-abs-method/) | calculates a number’s absolute value. |
| b. [sqrt()](https://www.geeksforgeeks.org/calculate-square-root-of-a-number-in-r-language-sqrt-function/) | calculates a number’s square root. |
| c. [round()](https://www.geeksforgeeks.org/rounding-off-values-in-r-language-round-function/) | rounds a number to the nearest integer. |
| d. [exp()](https://www.geeksforgeeks.org/calculate-exponential-of-a-number-in-r-programming-exp-function/) | calculates a number’s exponential value |
| e. [log()](https://www.geeksforgeeks.org/performing-logarithmic-computations-in-r-programming-log-log10-log1p-and-log2-functions/) | which calculates a number’s natural logarithm. |
| f. [cos()](https://www.geeksforgeeks.org/calculate-cosine-of-a-value-in-r-programming-cos-function/), [sin()](https://www.geeksforgeeks.org/calculate-sine-of-a-value-in-r-programming-sin-function/), and [tan()](https://www.geeksforgeeks.org/calculate-tangent-of-a-value-in-r-programming-tan-function/) | calculates a number’s cosine, sine, and tang. |
| **Statistical Functions** |  |
| a. [mean()](https://www.geeksforgeeks.org/calculate-arithmetic-mean-in-r-programming-mean-function/) | A vector’s arithmetic mean is determined by the mean() function. |
| b. [median()](https://www.geeksforgeeks.org/mean-median-and-mode-in-r-programming/) | A vector’s median value is determined by the median() function. |
| c. [cor()](https://www.geeksforgeeks.org/covariance-and-correlation-in-r-programming/) | calculates the correlation between two vectors. |
| d. [var()](https://www.geeksforgeeks.org/compute-variance-and-standard-deviation-of-a-value-in-r-programming-var-and-sd-function/) | calculates the variance of a vector and calculates the standard deviation of a vector. |
| **Data Manipulation Functions** |  |
| a. [unique()](https://www.geeksforgeeks.org/unique-function-in-r/) | returns the unique values in a vector. |
| b. [subset()](https://www.geeksforgeeks.org/subsetting-in-r-programming/) | subsets a data frame based on conditions. |
| c. [aggregate()](https://www.geeksforgeeks.org/how-to-use-aggregate-function-in-r/) | groups data according to a grouping variable. |
| d. [order()](https://www.geeksforgeeks.org/sorting-of-arrays-in-r-programming/) | uses ascending or descending order to sort a vector. |
| **File Input/Output Functions** |  |
| a.  [read.csv()](https://www.geeksforgeeks.org/read-contents-of-a-csv-file-in-r-programming-read-csv-function/) | reads information from a CSV file. |
| b. [Write.csv()](https://www.geeksforgeeks.org/writing-to-csv-files-in-r/) | publishes information to write a CSV file. |
| c. [Read. table()](https://www.geeksforgeeks.org/reading-files-in-r-programming/) | reads information from a tabular. |
| d. [Write.table()](https://www.geeksforgeeks.org/how-to-use-write-table-in-r/) | creates a tabular file with data. |

**User-defined Functions in R Programming Language**

R provides built-in functions like **print()**, **cat()**, etc. but we can also create our own functions. These functions are called user-defined functions.

**Example**

* R

|  |
| --- |
| # A simple R function to check  # whether x is even or odd    evenOdd = **function**(x){    if(x %% 2 == 0)      return("even")  **else**      return("odd")  }    print(evenOdd(4))  print(evenOdd(3)) |

**Output**

[1] "even"

[1] "odd"

**R Function Example**

**Single Input Single Output**

Now create a function in R that will take a single input and gives us a single output.

Following is an example to create a function that calculates the area of a circle which takes in the arguments the radius. So, to create a function, name the function as “areaOfCircle” and the arguments that are needed to be passed are the “radius” of the circle.

* R

|  |
| --- |
| # A simple R function to calculate  # area of a circle    areaOfCircle = **function**(radius){    area = pi\*radius^2    return(area)  }    print(areaOfCircle(2)) |

**Output**

12.56637

**Multiple Input Multiple Output**

Now create a function in R Language that will take multiple inputs and gives us multiple outputs using a list.

The functions in R Language take multiple input objects but returned only one object as output, this is, however, not a limitation because you can create lists of all the outputs which you want to create and once the list is created you can access them into the elements of the list and get the answers which you want.

Let us consider this example to create a function “Rectangle” which takes “length” and “width” of the rectangle and returns area and perimeter of that rectangle. Since R Language can return only one object. Hence, create one object which is a list that contains “area” and “perimeter” and return the list.

* R

|  |
| --- |
| # A simple R function to calculate  # area and perimeter of a rectangle    Rectangle = **function**(length, width){    area = length \* width    perimeter = 2 \* (length + width)      # create an object called result which is    # a list of area and perimeter    result = list("Area" = area, "Perimeter" = perimeter)    return(result)  }    resultList = Rectangle(2, 3)  print(resultList["Area"])  print(resultList["Perimeter"]) |

**Output**

$Area

[1] 6

$Perimeter

[1] 10

**Inline Functions in R Programming Language**

Sometimes creating an R script file, loading it, executing it is a lot of work when you want to just create a very small function. So, what we can do in this kind of situation is an inline function.

To create an inline function you have to use the function command with the argument x and then the expression of the function.

**Example**

* R

|  |
| --- |
| # A simple R program to  # demonstrate the inline function    f = **function**(x) x^2\*4+x/3    print(f(4))  print(f(-2))  print(0) |

**Output**

65.33333

15.33333

0

**Passing Arguments to Functions in R Programming Language**

There are several ways you can pass the arguments to the function:

* **Case 1**: Generally in R, the arguments are passed to the function in the same order as in the function definition.
* **Case 2**: If you do not want to follow any order what you can do is you can pass the arguments using the names of the arguments in any order.
* **Case 3**: If the arguments are not passed the default values are used to execute the function.

Now, let us see the examples for each of these cases in the following R code:

* R

|  |
| --- |
| # A simple R program to demonstrate  # passing arguments to a function    Rectangle = **function**(length=5, width=4){    area = length \* width    return(area)  }    # Case 1:  print(Rectangle(2, 3))    # Case 2:  print(Rectangle(width = 8, length = 4))    # Case 3:  print(Rectangle()) |

**Output**

6

32

20

**Lazy Evaluations of Functions in R Programming Language**

In R the functions are executed in a lazy fashion. When we say lazy what it means is if some arguments are missing the function is still executed as long as the execution does not involve those arguments.

**Example**

In the function “Cylinder” given below. There are defined three-argument “diameter”, “length” and “radius” in the function and the volume calculation does not involve this argument “radius” in this calculation. Now, when you pass this argument “diameter” and “length” even though you are not passing this “radius” the function will still execute because this radius is not used in the calculations inside the function.   
Let’s illustrate this in an R code given below:

* R

|  |
| --- |
| # A simple R program to demonstrate  # Lazy evaluations of functions    Cylinder = **function**(diameter, length, radius ){    volume = pi\*diameter^2\*length/4    return(volume)  }    # This'll execute because this  # radius is not used in the  # calculations inside the function.  print(Cylinder(5, 10)) |

**Output**

196.3495

If you do not pass the argument and then use it in the definition of the function it will throw an error that this “radius” is not passed and it is being used in the function definition.

**Example**

* R

|  |
| --- |
| # A simple R program to demonstrate  # Lazy evaluations of functions    Cylinder = **function**(diameter, length, radius ){    volume = pi\*diameter^2\*length/4    print(radius)    return(volume)  }    # This'll throw an error  print(Cylinder(5, 10)) |

**Output**

Error in print(radius) : argument "radius" is missing, with no default