In The Name Of God

Introduction to R

T:F. Yaghmayi

TA:A. Shokri

S:M.M. Esmaeil Zadeh

Semnan University

Winter 2021

1. R Introduction

1.1. R Operators

R has many operators to carry out different mathematical and logical operations. Operators in R can mainly be classified into the following categories.

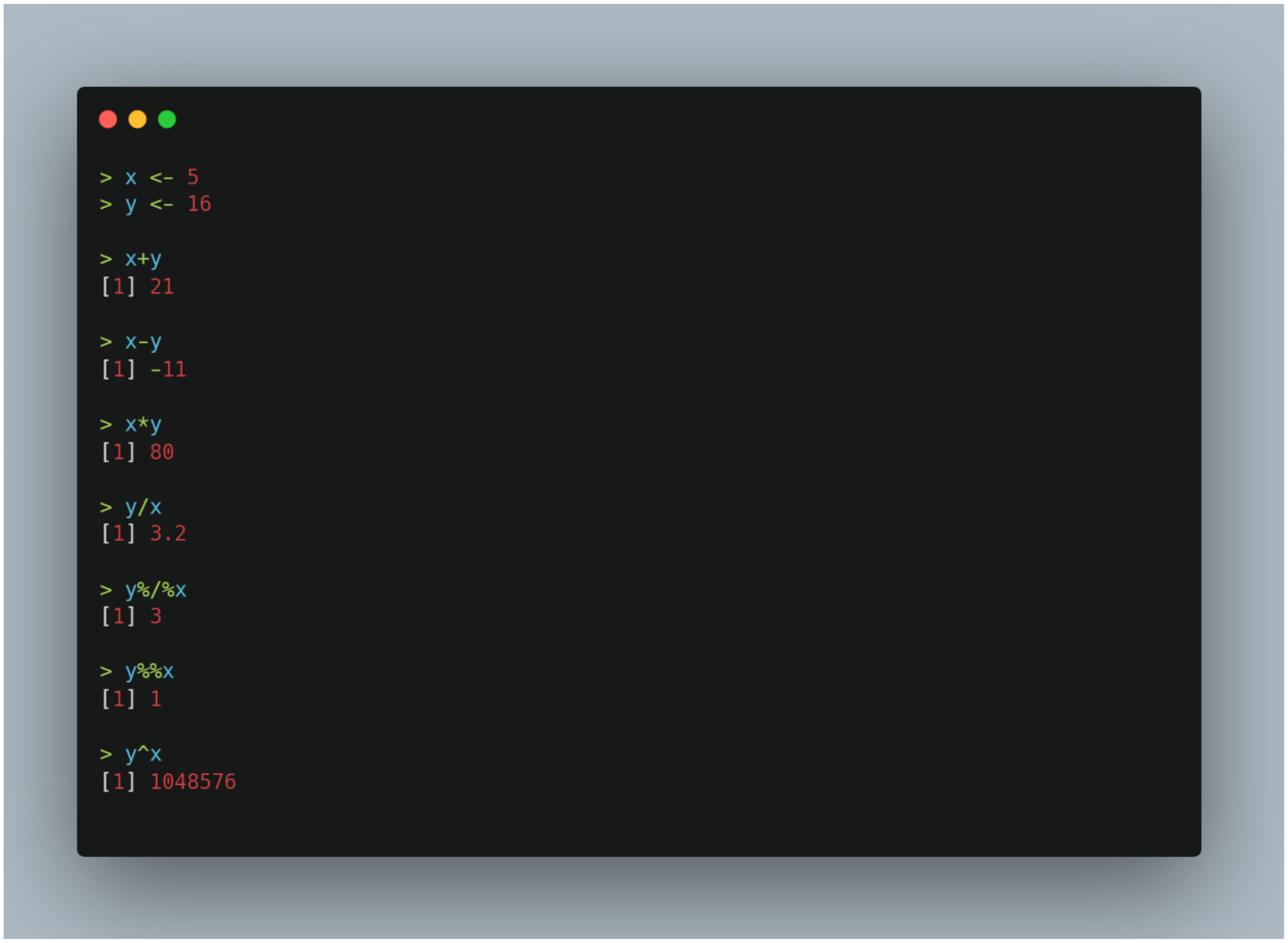
- Arithmetic operators

These operators are used to carry out mathematical operations like addition and multiplication. Here is a list of arithmetic operators available in R.

Arithmetic Operators in R

Operator	Description
+	Addition
–	Subtraction
*	Multiplication
/	Division
^	Exponent
%%	Modulus (Remainder from division)
%/%	Integer Division

Code example:



```
> x <- 5
> y <- 16

> x+y
[1] 21

> x-y
[1] -11

> x*y
[1] 80

> y/x
[1] 3.2

> y%/%x
[1] 3

> y%%x
[1] 1

> y^x
[1] 1048576
```

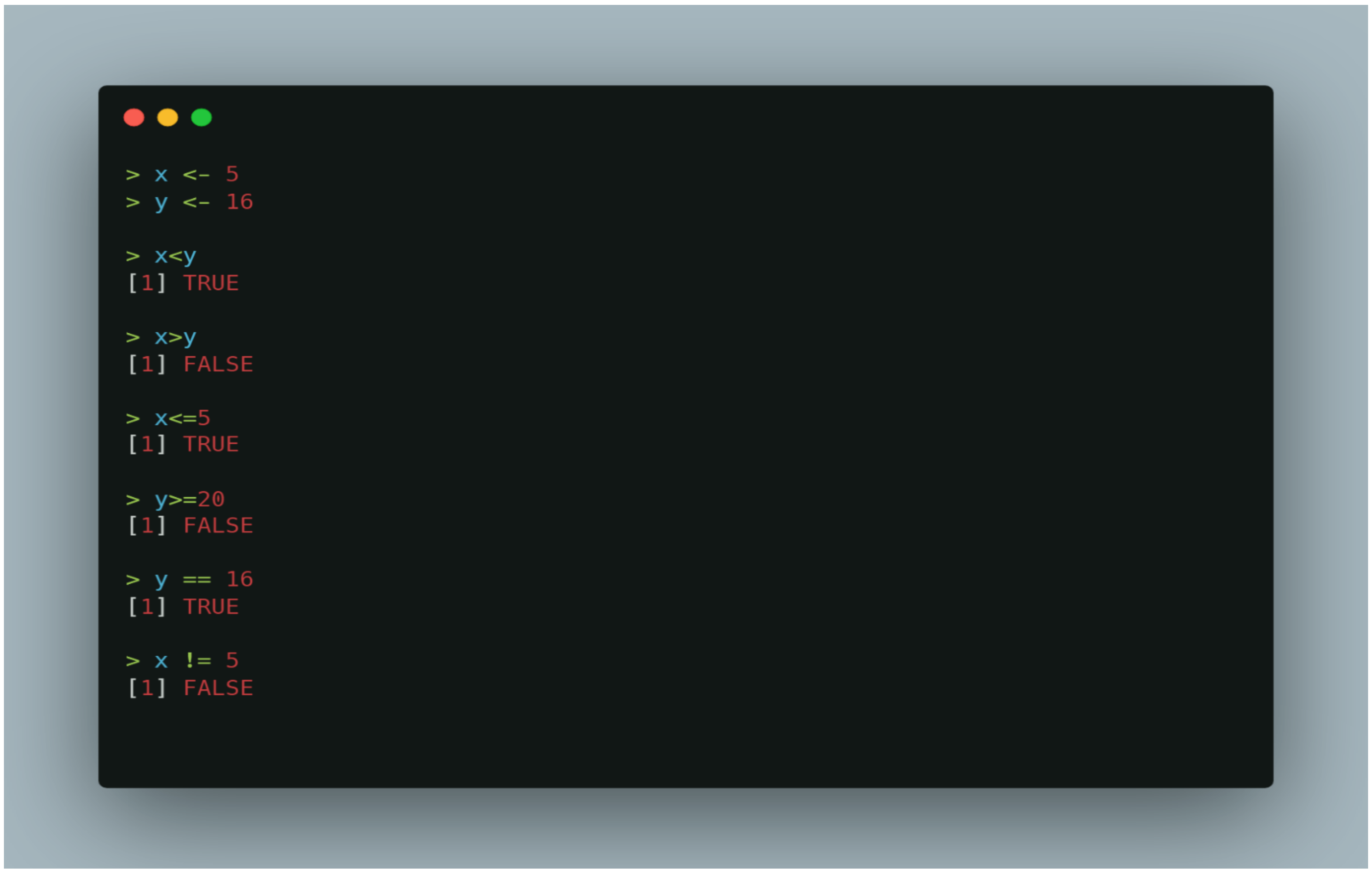
- Relational operators

  Relational operators are used to compare between values. Here is a list of relational operators available in R.

Relational Operators in R

Operator	Description
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Code example:



```
> x <- 5
> y <- 16

> x<y
[1] TRUE

> x>y
[1] FALSE

> x<=5
[1] TRUE

> y>=20
[1] FALSE

> y == 16
[1] TRUE

> x != 5
[1] FALSE
```

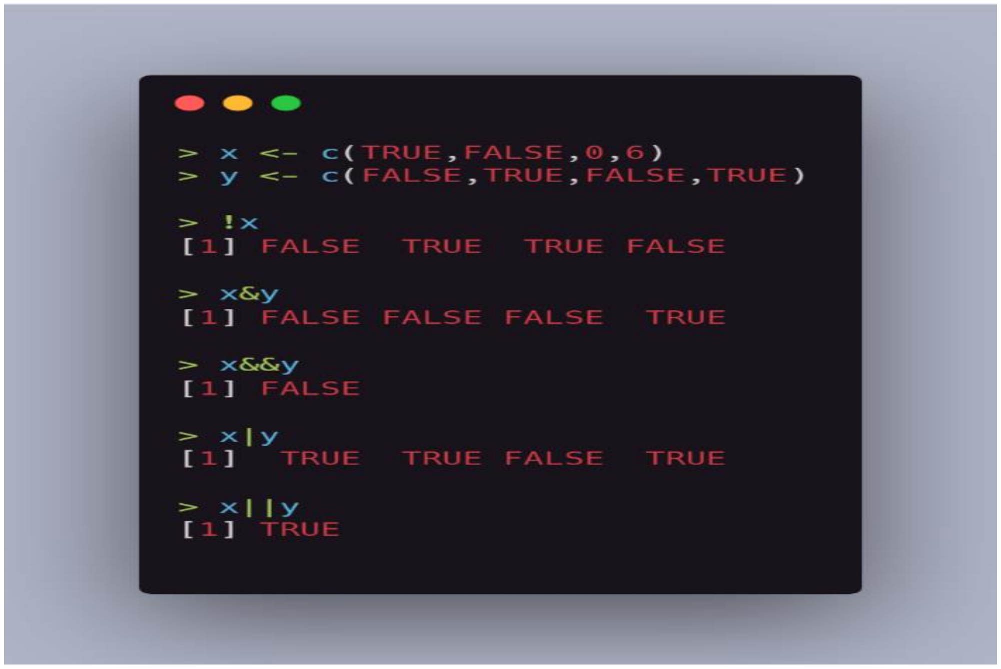
- Logical operators

Logical operators are used to carry out Boolean operations like AND, OR etc.

Logical Operators in R

Operator	Description
!	Logical NOT
&	Element-wise logical AND
&&	Logical AND
|	Element-wise logical OR
||	Logical OR

Code example:



```
> x <- c(TRUE,FALSE,0,6)
> y <- c(FALSE,TRUE,FALSE,TRUE)

> !x
[1] FALSE  TRUE  TRUE FALSE

> x&y
[1] FALSE FALSE FALSE  TRUE

> x&&y
[1] FALSE

> x|y
[1]  TRUE  TRUE FALSE  TRUE

> x||y
[1] TRUE
```

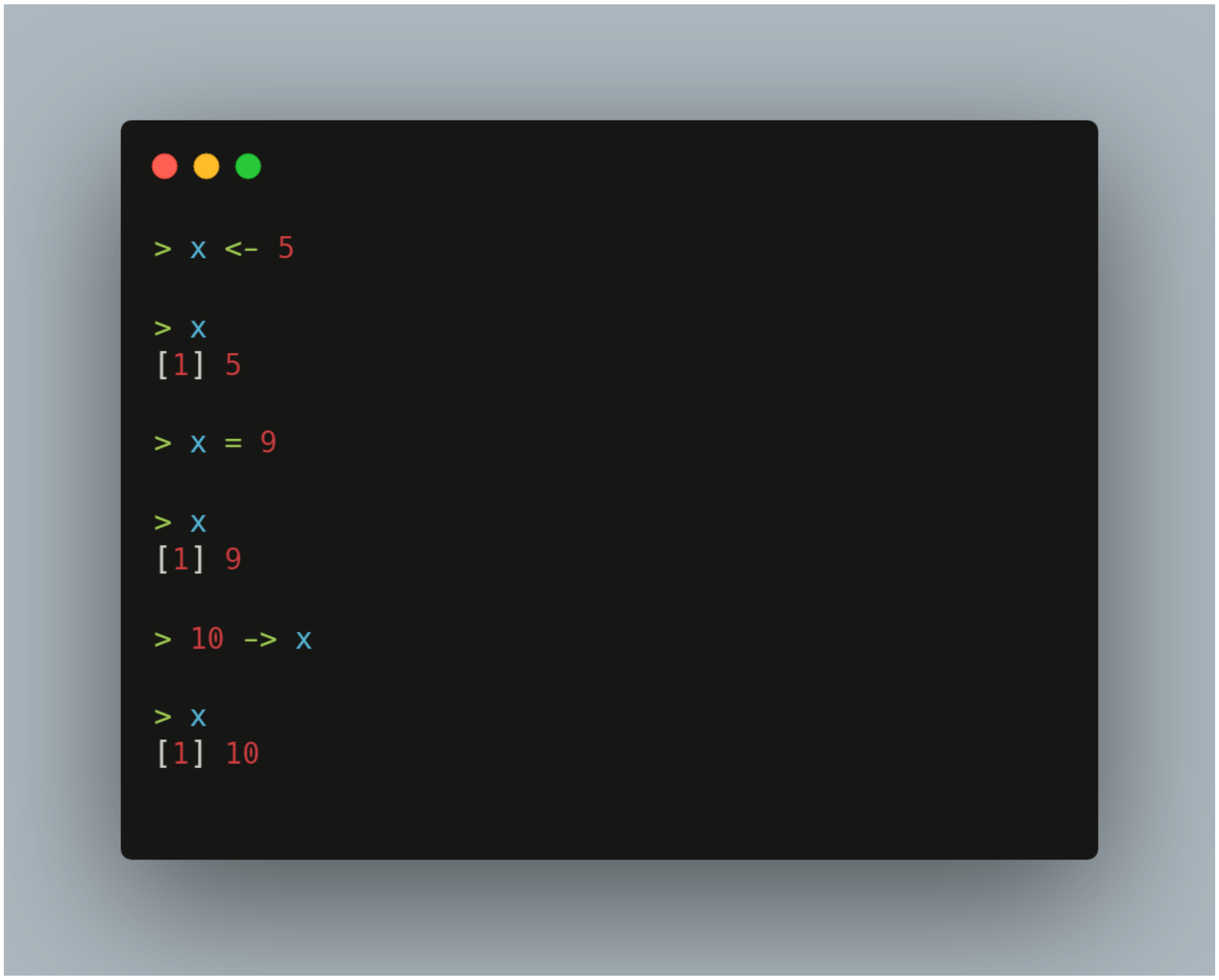
- Assignment operators

These operators are used to assign values to variables.

▪ Assignment Operators in R

Operator	Description
<-, <<-, =	Leftwards assignment
->, ->>	Rightwards assignment

Code example:



```
> x <- 5

> x
[1] 5

> x = 9

> x
[1] 9

> 10 -> x

> x
[1] 10
```

1.2. Variables & Constants

Variables are used to store data, whose value can be changed according to our need. Unique name given to variable (function and objects as well) is identifier.

- Valid identifiers in R

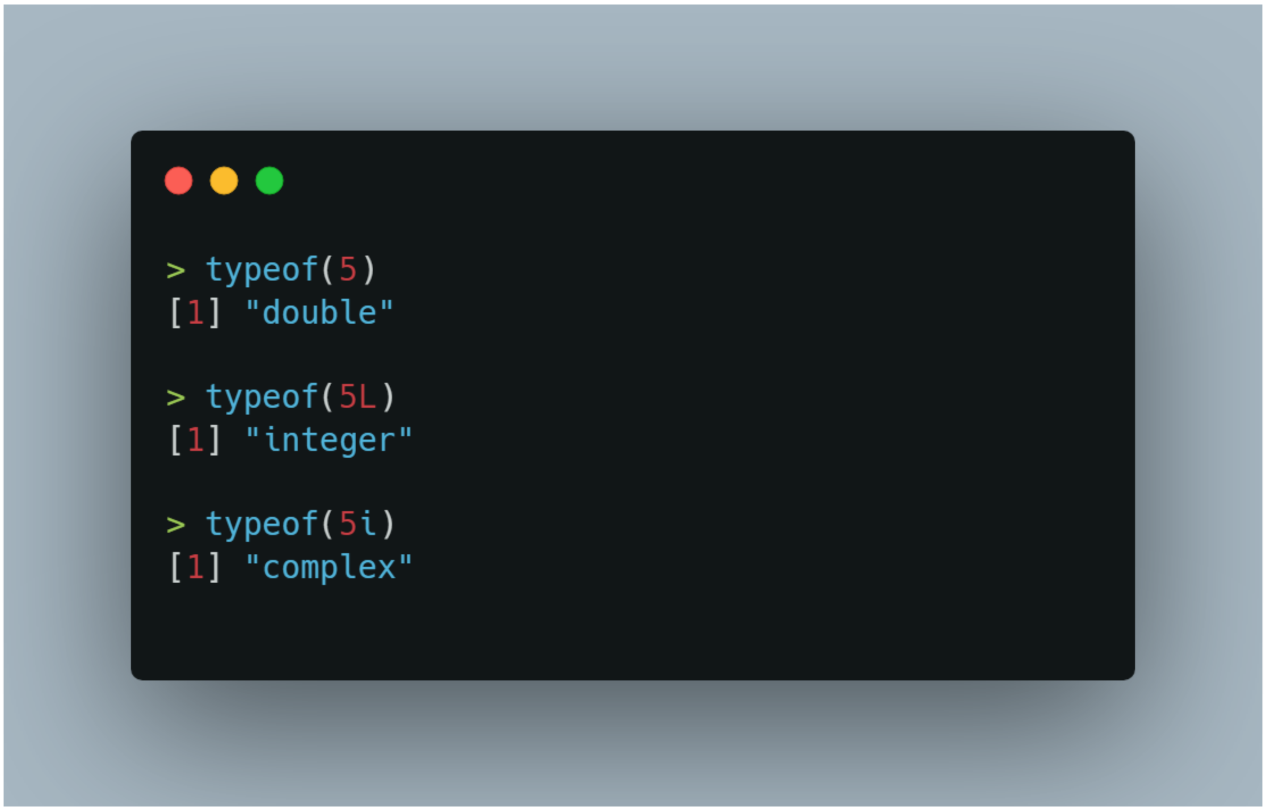
`total`, `Sum`, `.fine.with.dot`, `this_is_acceptable`, `Number5`

- Invalid identifiers in R

`tot@l`, `5um`, `_fine`, `TRUE`, `.0ne`

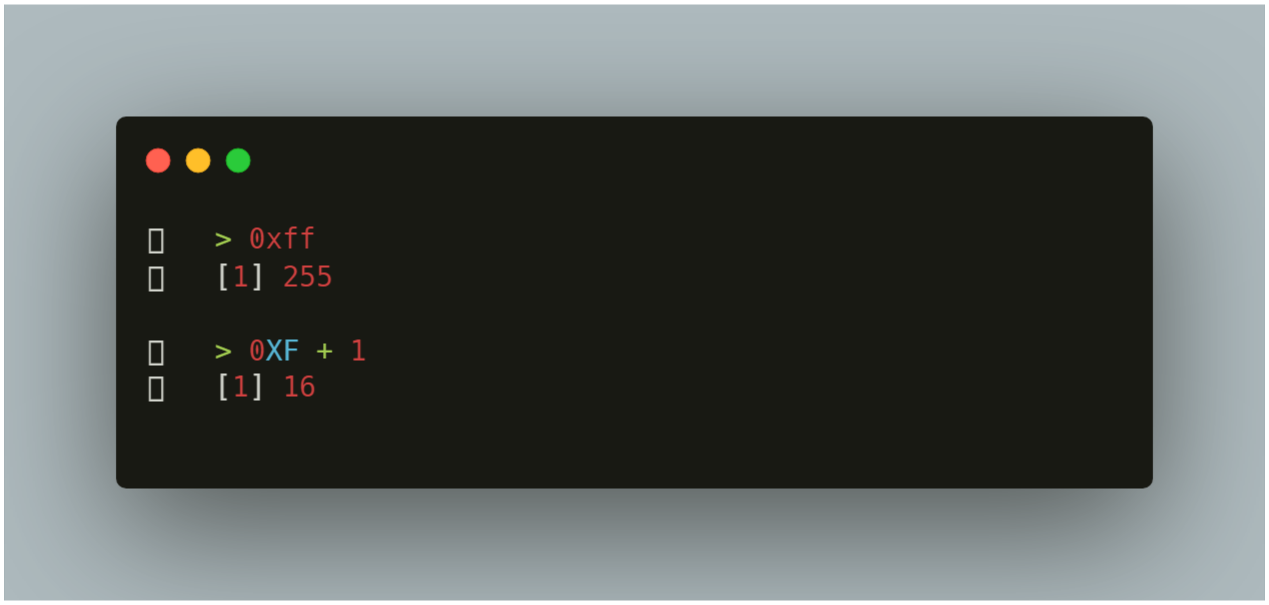
- Constants in R

1.2..1. Numeric Constants



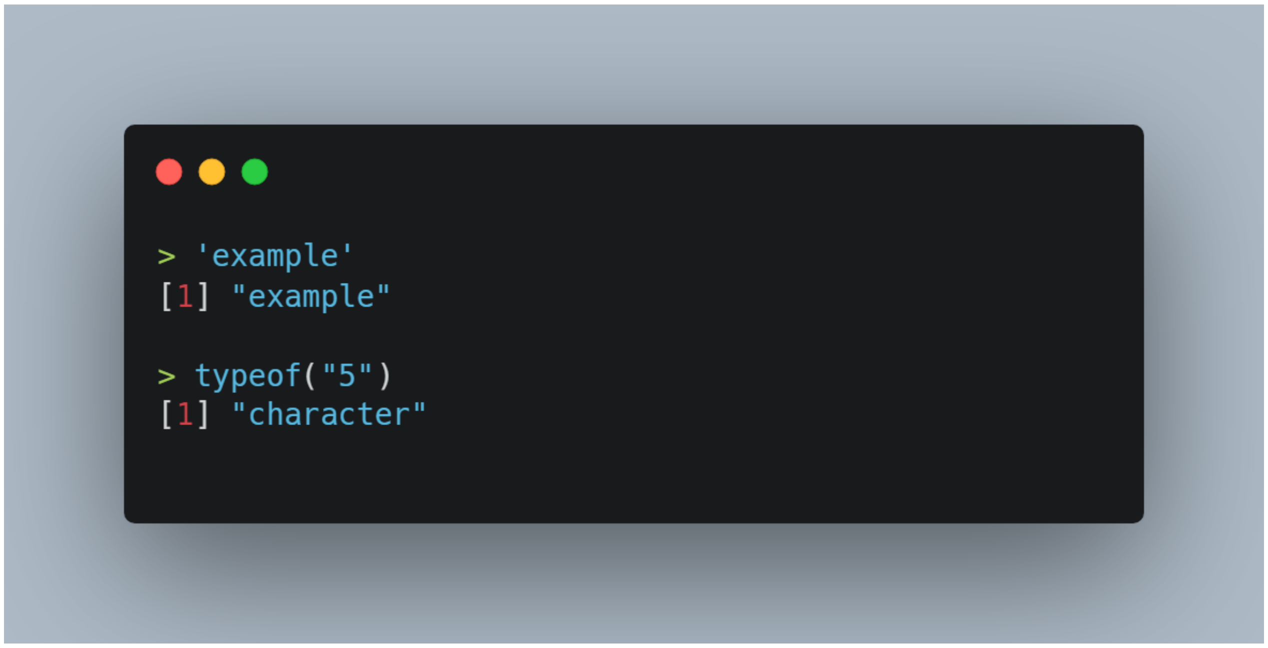
```
> typeof(5)
[1] "double"

> typeof(5L)
[1] "integer"

> typeof(5i)
[1] "complex"
```

Numeric constants preceded by 0x or 0X are interpreted as hexadecimal numbers.



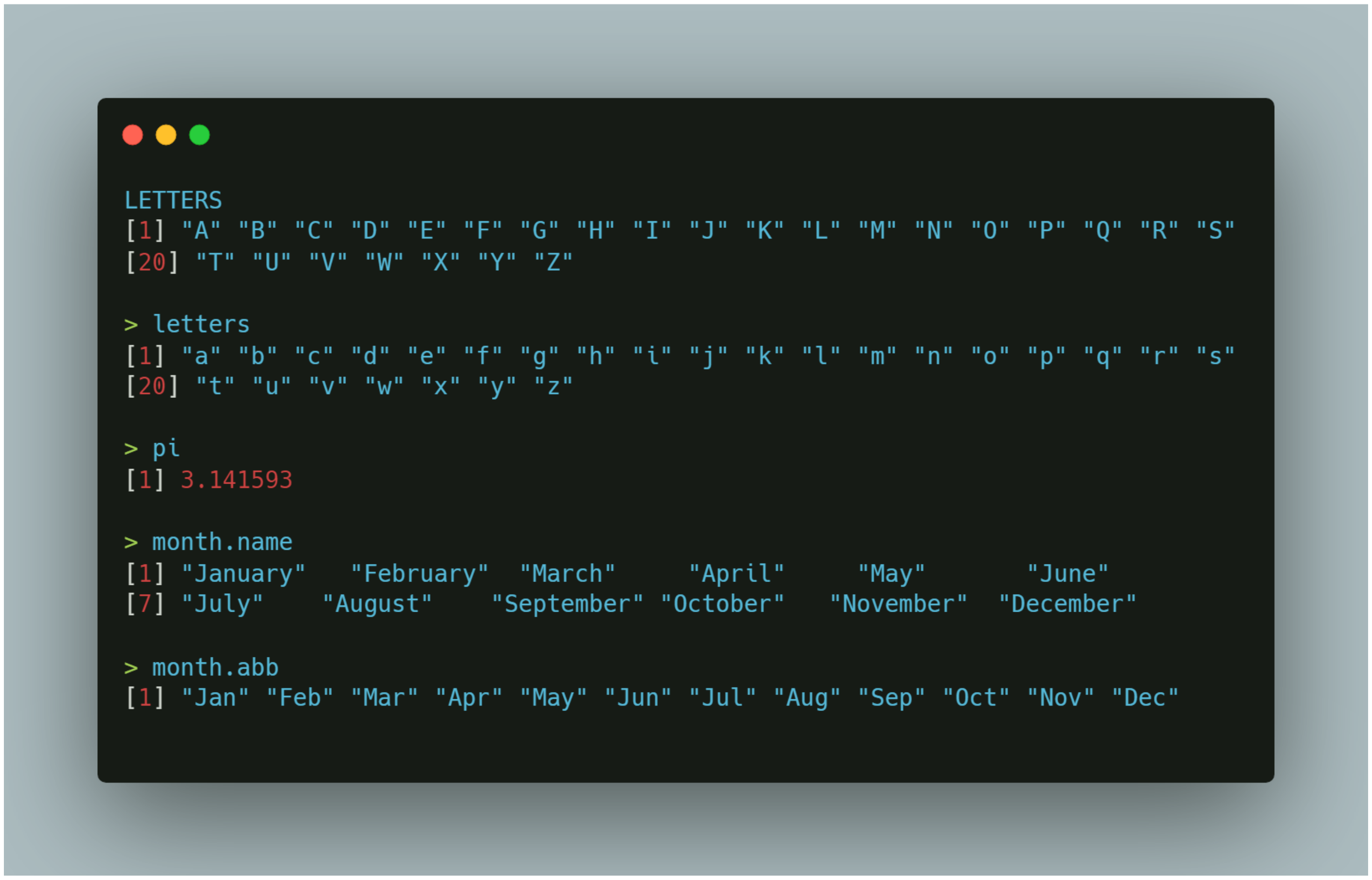
```
▯   > 0xff
▯   [1] 255

▯   > 0XF + 1
▯   [1] 16
```

1.2..2. Character Constants



```
> 'example'
[1] "example"

> typeof("5")
[1] "character"
```

1.2..3. Built-in Constants



```
LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R" "S"
[20] "T" "U" "V" "W" "X" "Y" "Z"

> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"

> pi
[1] 3.141593

> month.name
[1] "January"   "February"  "March"     "April"     "May"       "June"
[7] "July"      "August"    "September" "October"   "November"  "December"

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

1.3. Operator Precedence

Operator Precedence in R

Operator	Description	Associativity
^	Exponent	Right to Left
-x, +x	Unary minus, Unary plus	Left to Right
%%	Modulus	Left to Right
*, /	Multiplication, Division	Left to Right

+, –	Addition, Subtraction	Left to Right
<, >, <=, >=, ==, !=	Comparisions	Left to Right
!	Logical NOT	Left to Right
&, &&	Logical AND	Left to Right
|, ||	Logical OR	Left to Right
->, ->>	Rightward assignment	Left to Right
<-, <<-	Leftward assignment	Right to Left
=	Leftward assignment	Right to Left

1.4. R Reserved Words

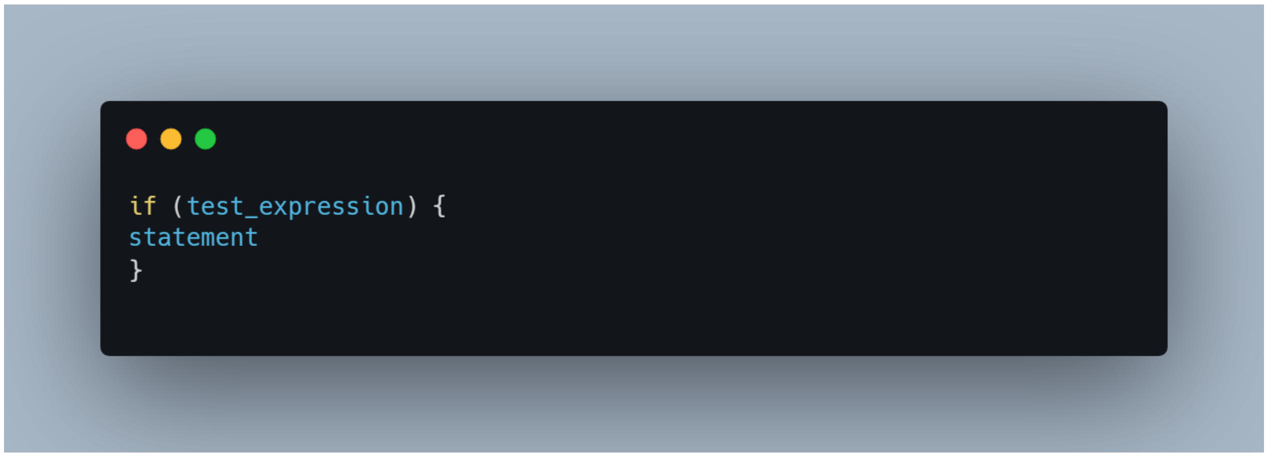
Reserved words in R

if	else	repeat	while	function
for	in	next	break	TRUE
FALSE	NULL	Inf	NaN	NA
NA_integer_	NA_real_	NA_complex_	NA_character_	…

2. DECISION AND LOOP

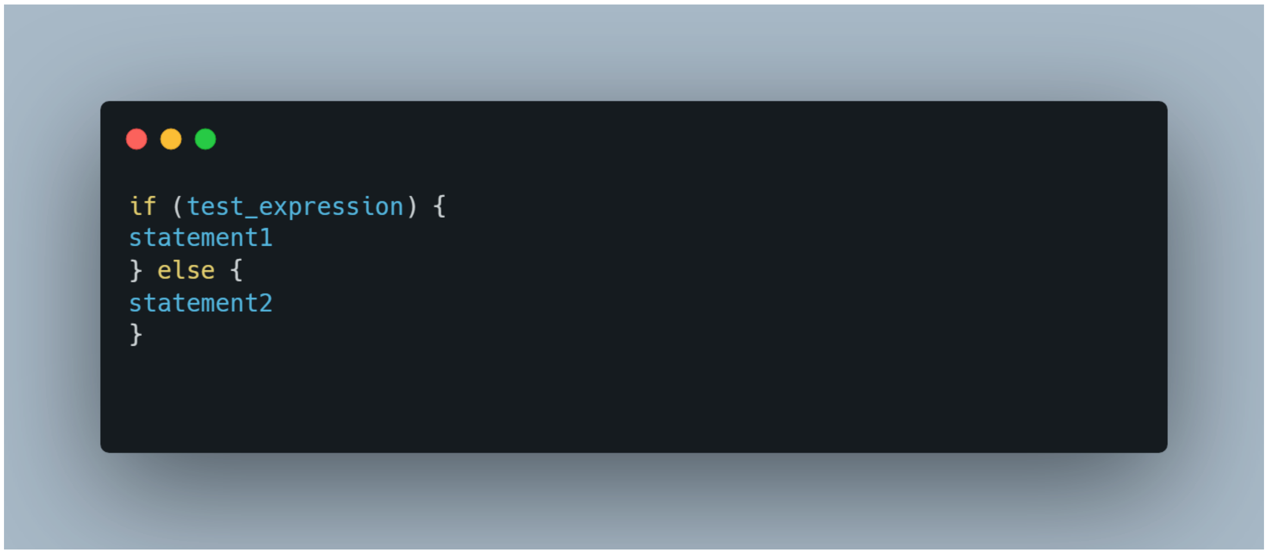
2.1. R Programming if…else

- The syntax of if statement is:



```
if (test_expression) {
statement
}
```

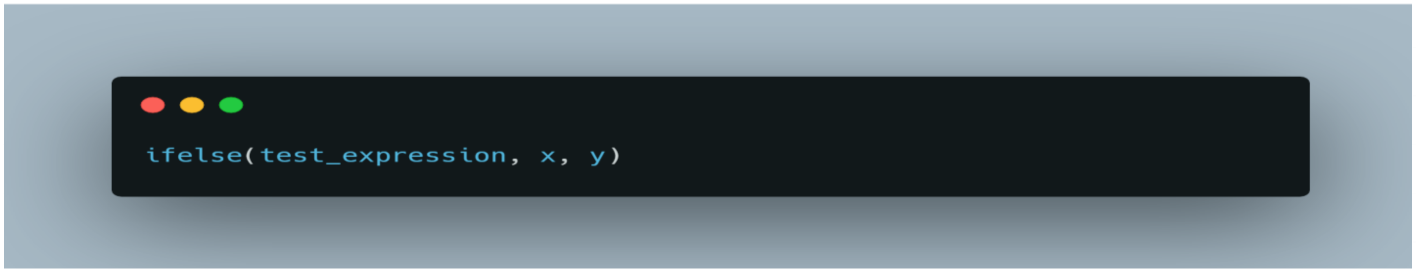
- The syntax of if…else statement is:



```
if (test_expression) {
statement1
} else {
statement2
}
```

2.2. R ifelse() Function

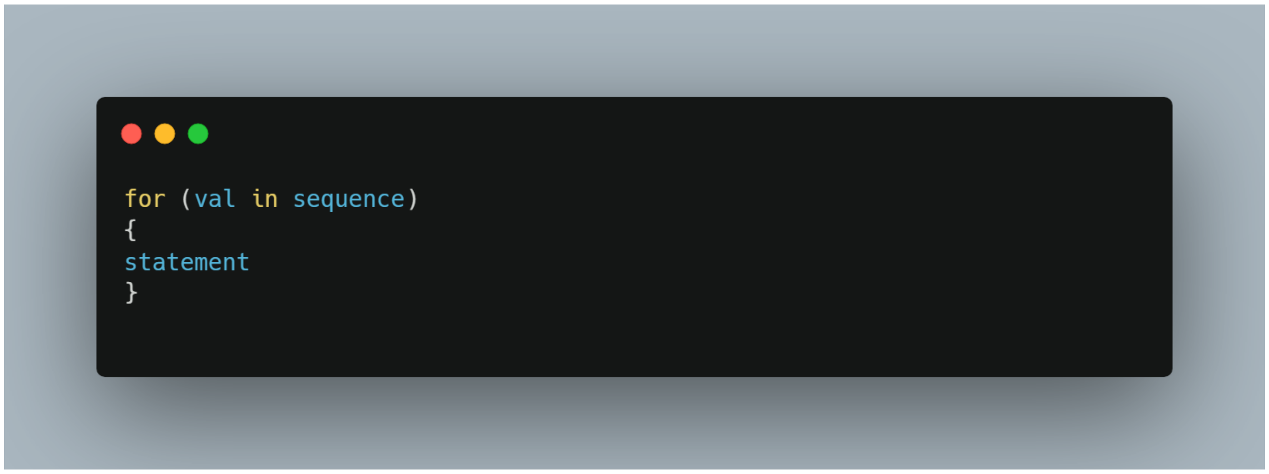
- The syntax of ifelse() Function is:



```
ifelse(test_expression, x, y)
```

2.3. R Programming for loop

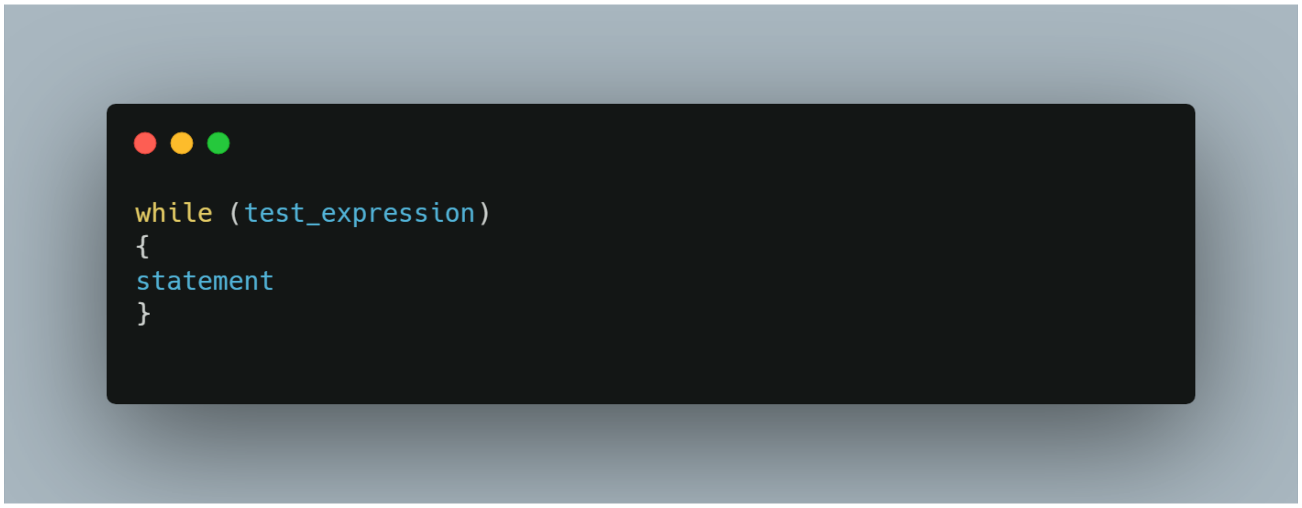
- The syntax of ifelse() Function is:



```
for (val in sequence)
{
statement
}
```

2.4. R while Loop

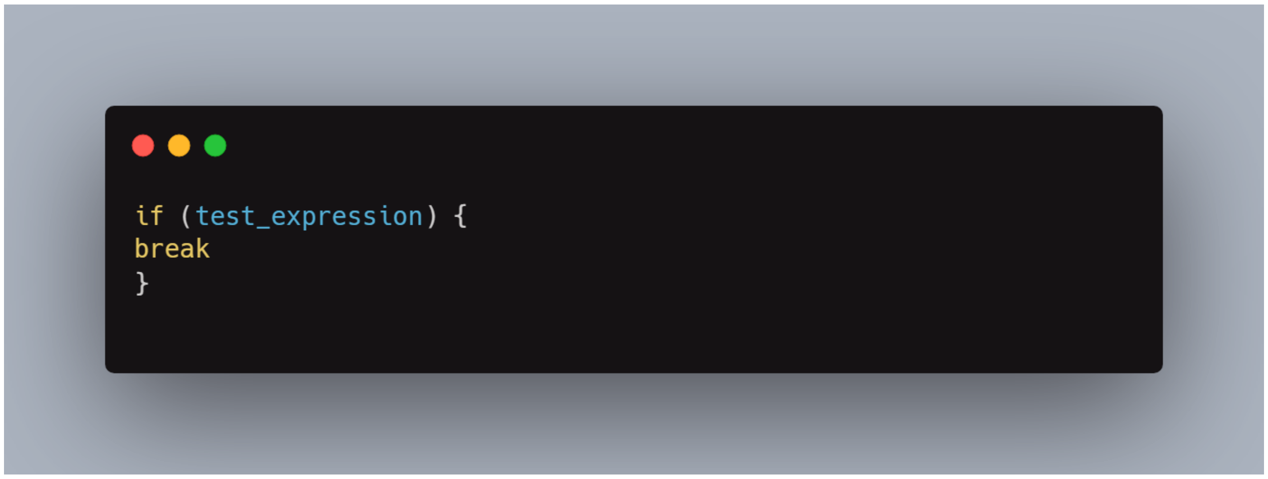
- Syntax of while loop is:



```
while (test_expression)
{
statement
}
```

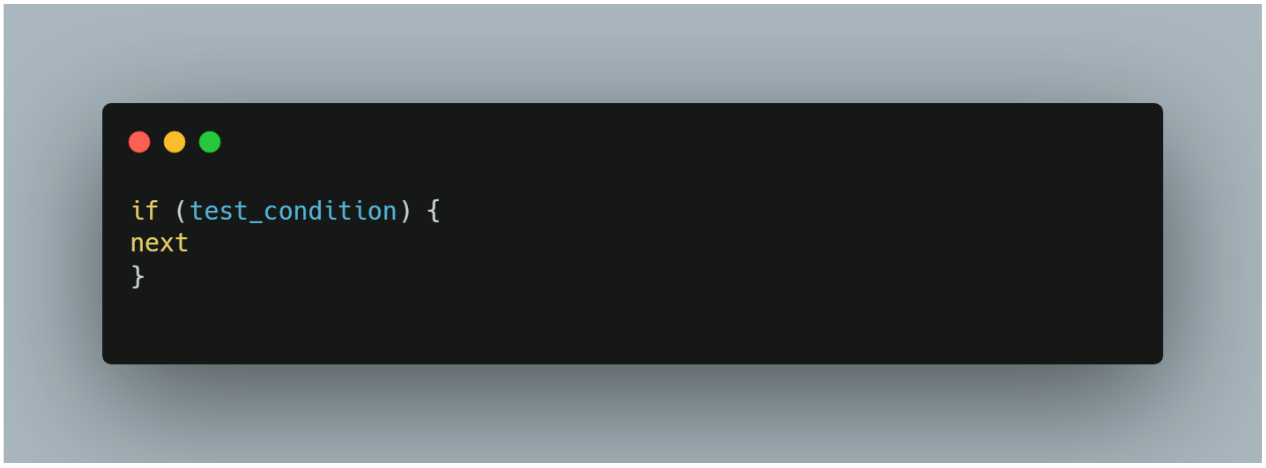
2.5. R break & next

- The syntax of break statement is:



```
if (test_expression) {
break
}
```

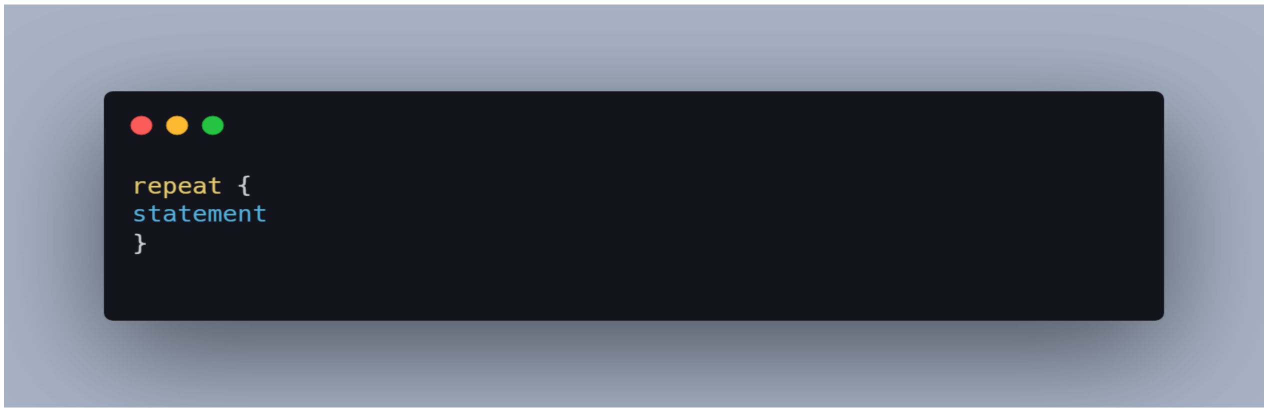
- The syntax of next statement is:



```
if (test_condition) {
next
}
```

2.6. R repeat Loop

- The syntax of repeat statement is:

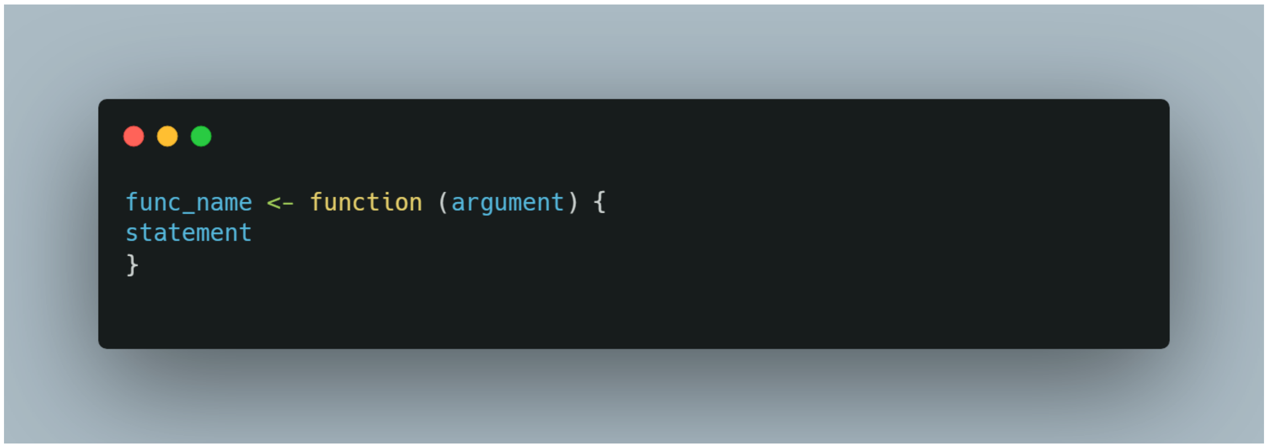


```
repeat {
statement
}
```

3. R FUNCTIONS

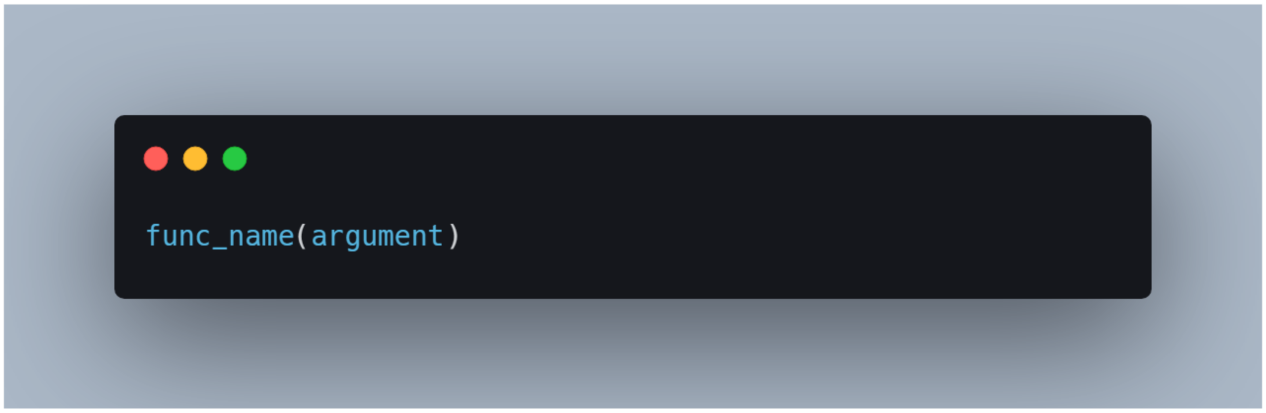
3.1. R Functions

- Syntax for Writing Functions in R:



```
func_name <- function (argument) {
statement
}
```

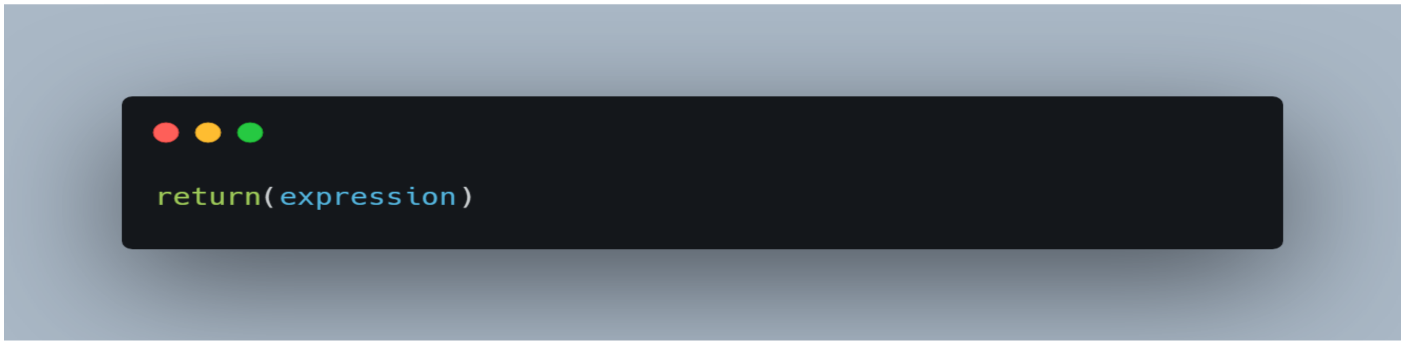
- How to call a function?



```
func_name(argument)
```

3.2. Function Return Value

- Syntax of return():

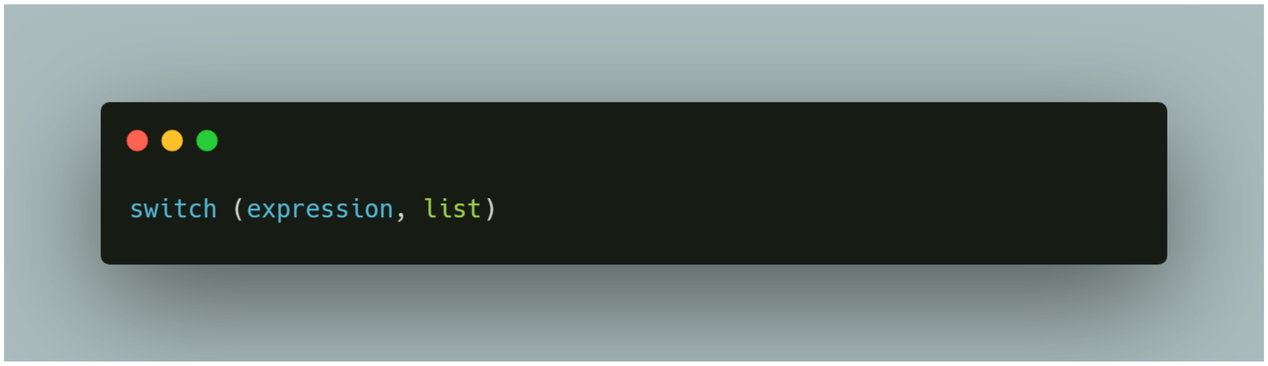


```
return(expression)
```

3.3. R Recursive Function

3.4. Switch Function

- Syntax of switch() function:



```
switch (expression, list)
```

4. R DATA STRUCTURES

4.1. R Vectors

Vector is a basic data structure in R. It contains element of the same type. The data types can be logical, integer, double, character, complex or raw.

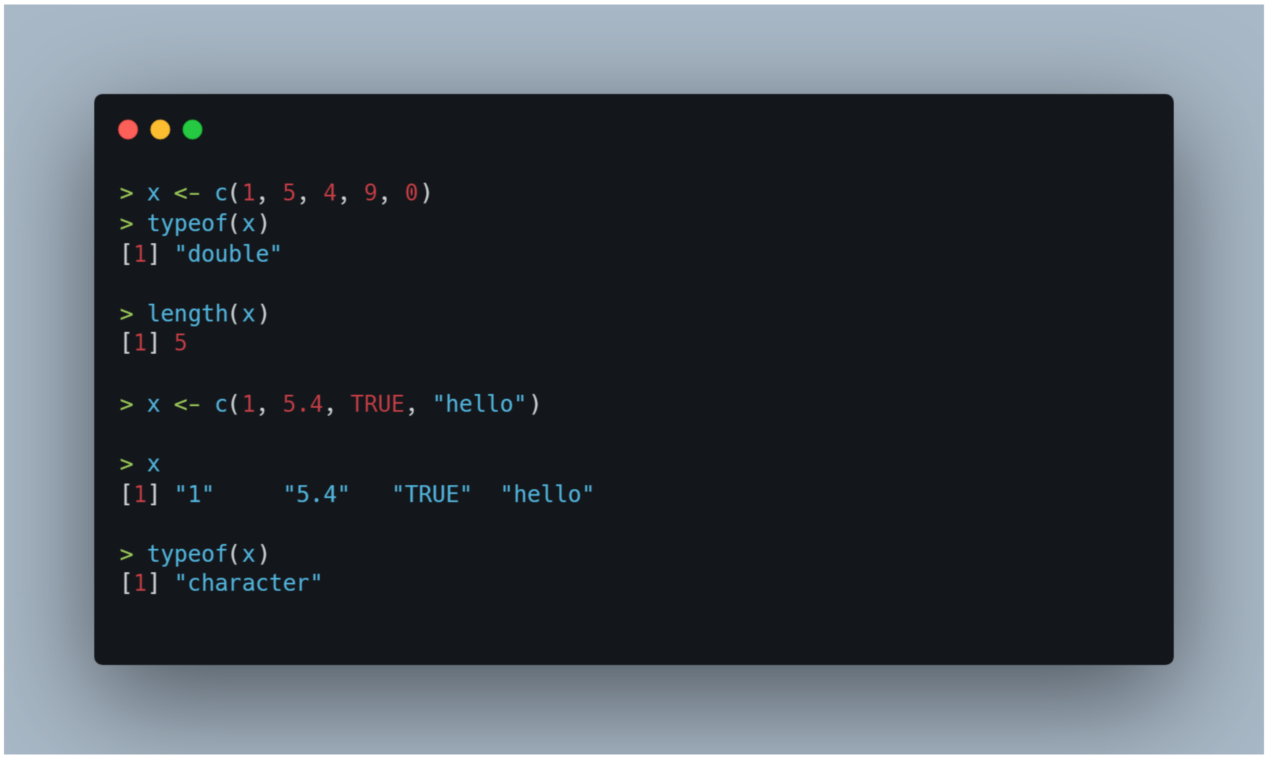
A vector's type can be checked with the typeof() function.

Another important property of a vector is its length. This is the number of elements in the vector and can be checked with the function length().

Since, a vector must have elements of the same type, this function will try and coerce elements to the same type, if they are different.

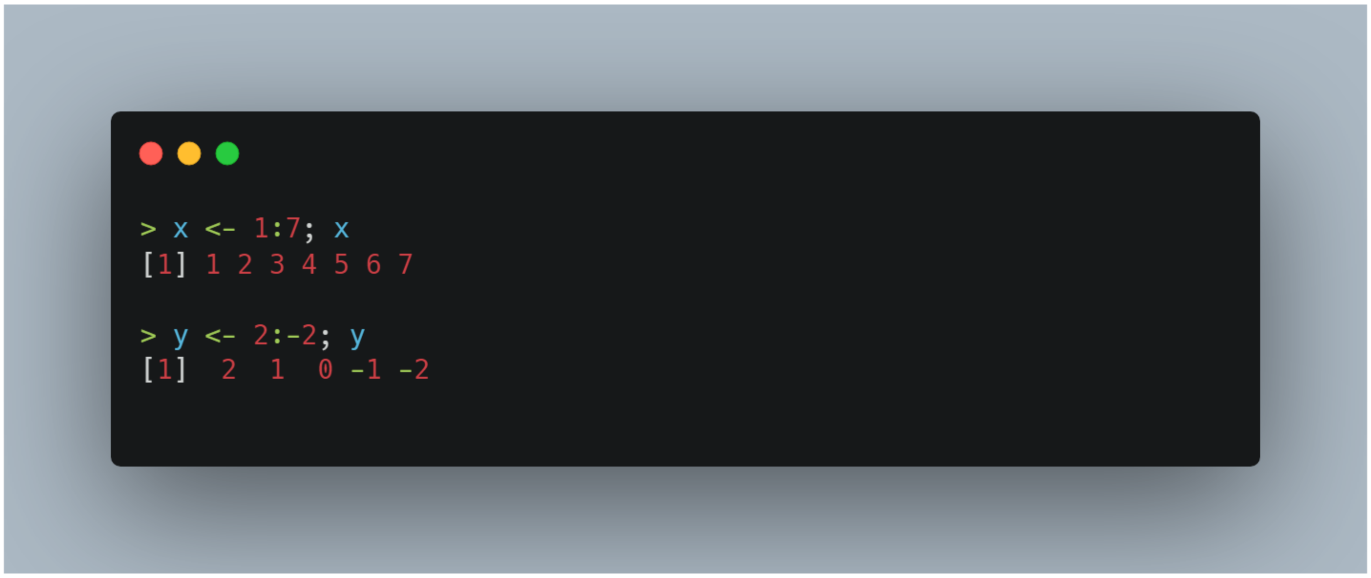
Coercion is from lower to higher types from logical to integer to double to character.

Code example:



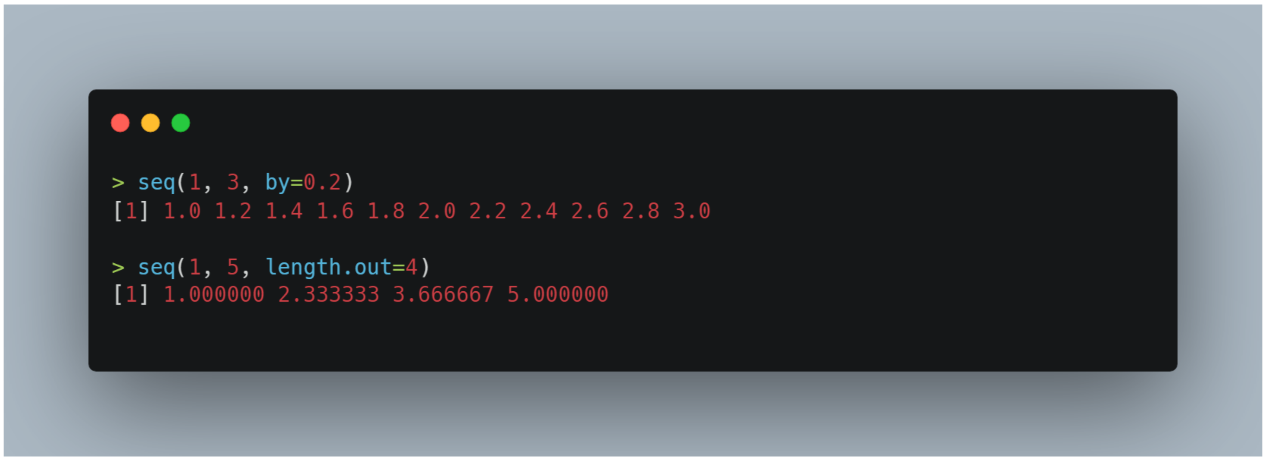
```
> x <- c(1, 5, 4, 9, 0)
> typeof(x)
[1] "double"

> length(x)
[1] 5

> x <- c(1, 5.4, TRUE, "hello")

> x
[1] "1"     "5.4"   "TRUE"  "hello"

> typeof(x)
[1] "character"
```

If we want to create a vector of consecutive numbers, the `:` operator is very helpful.



```
> x <- 1:7; x
[1] 1 2 3 4 5 6 7

> y <- 2:-2; y
[1]  2  1  0 -1 -2
```

More complex sequences can be created using the `seq()` function, like defining number of points in an interval, or the step size.



```
> seq(1, 3, by=0.2)
[1] 1.0 1.2 1.4 1.6 1.8 2.0 2.2 2.4 2.6 2.8 3.0

> seq(1, 5, length.out=4)
[1] 1.000000 2.333333 3.666667 5.000000
```

How to access Elements of a Vector?

Elements of a vector can be accessed using vector indexing. The vector used for indexing can be logical, integer or character vector.

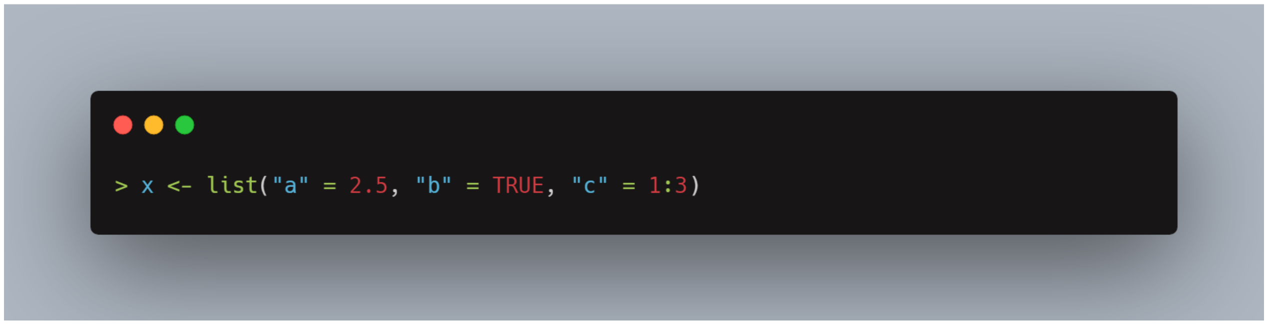
4.2. List in R programming

List is a data structure having components of mixed data types.

A vector having all elements of the same type is called atomic vector but a vector having elements of different type is called list.

We can check if it’s a list with typeof() function and find its length using length(). Here is an example of a list having three components each of different data type.

List can be created using the `list()` function.



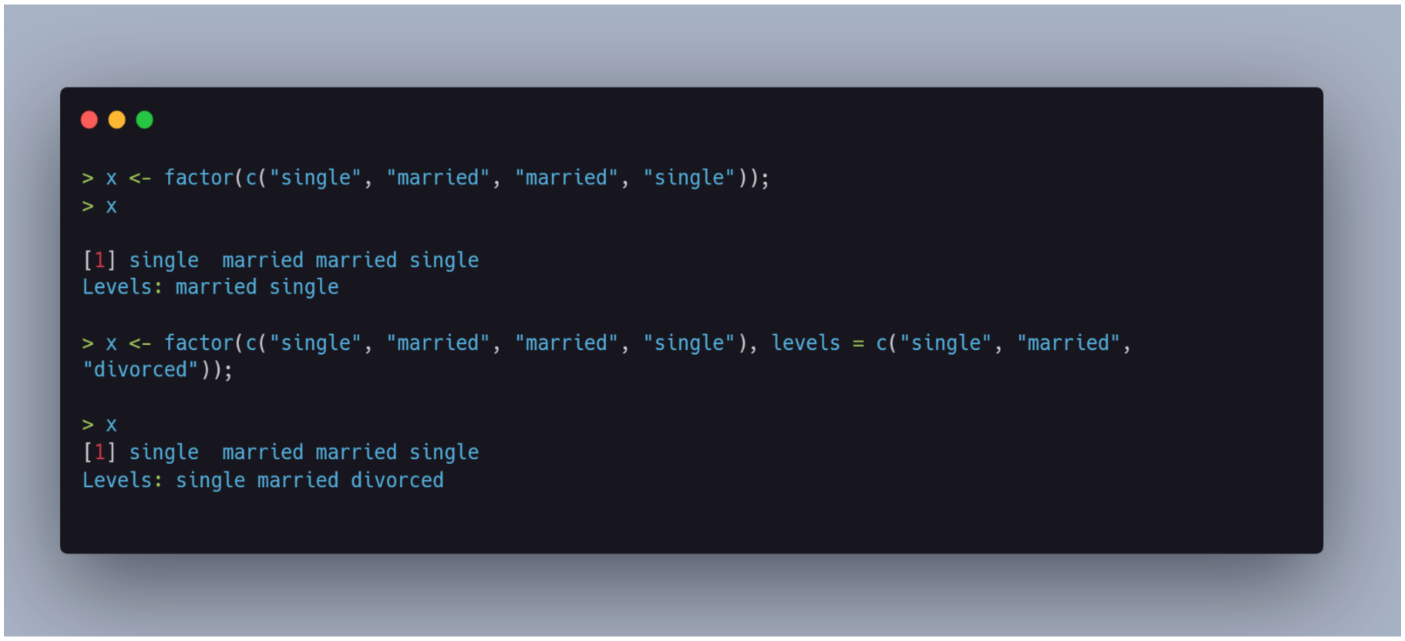
```
> x <- list("a" = 2.5, "b" = TRUE, "c" = 1:3)
```

4.3. R Factor

Factor is a data structure used for fields that takes only predefined, finite number of values (categorical data). For example: a data field such as marital status may contain only values from single, married, separated, divorced, or widowed.

In such case, we know the possible values beforehand and these predefined, distinct values are called levels. Following is an example of factor in R.

We can create a factor using the function `factor()`. Levels of a factor are inferred from the data if not provided.



```
> x <- factor(c("single", "married", "married", "single"));
> x

[1] single  married married single
Levels: married single

> x <- factor(c("single", "married", "married", "single"), levels = c("single", "married",
"divorced"));

> x
[1] single  married married single
Levels: single married divorced
```