Getting Started With R

R is an **interpreted programming language**. It also allows you to carry out modular programming with the help of functions. It is widely used to analyze statistical information as well as graphical representation.
R allows you to integrate with programming procedures written in C, C++, Python, .Net, etc. Today, R is widely used in the field of data science by data analysts, researchers, statisticians, etc. It is used to retrieve data from datasets, clean it, analyze and visualize it, and present it in the most suitable way.

Run R Programs

You can run R programs in two different ways:

- Installing R in your local machine
- Using an online environment

R Comments

Comments are portions of a computer program that are used to describe a piece of code. For example,

```
# declare variable
age = 24

# print variable
print(age)
```

Here, `# declare variable` and `# print variable` are two comments used in the code.
Comments have nothing to do with code logic. They do not get interpreted or compiled and are completely ignored during the execution of the program.

Types of Comments in R

In general, all programming languages have the following types of comments:

- single-line comments
- multi-line comments

However, in R programming, there is no functionality for multi-line comments. Thus, you can only write single-line comments in R.

---

1. R Single-Line Comments

You use the `#` symbol to create single-line comments in R. For example,

```
# this code prints Hello World
print("Hello World")
```

Output

```
[1] "Hello World"
```

In the above example, we have printed the text `Hello World` to the screen. Here, just before the print statement, we have included a single-line comment using the `#` symbol.

Note: You can also include a single-line comment in the same line after the code. For example,

```
print("Hello World") # this code prints Hello World
```

---

2. R Multi-Line Comments

As already mentioned, R does not have any syntax to create multi-line comments.

However, you can use consecutive single-line comments to create a multi-line comment in R. For example,

```
# this is a print statement
# it prints Hello World

print("Hello World")
```

Output

```
[1] "Hello World"
```

In the above code, we have used multiple consecutive single-line comments to create a multi-line comment just before the print statement.

Purpose of Comments

As discussed above, R comments are used to just document pieces of code. This can help others to understand the working of our code.

Here are a few purposes of commenting on an R code:

- It increases readability of the program for users other than the developers.
- Comments in R provide metadata of the code or the overall project.
- Comments are generally used by programmers to ignore some pieces of code during testing.
- They are used to write a simple pseudo-code of the program.

How to Create Better Comments?

As an R developer, your task is not only to write effective code. At times, you may also need to read codes written by other developers and modify them. In such a case, a well-written comment might be a lifesaver.

You should always keep in mind the following points while writing comments.

- Use comments only to describe what a particular block of code does, not how it does.
- Don't overuse comments. Try to make your code self-explanatory.
- Try to create comments that are as precise as possible.
- Don't use redundant comments.

R Variables and Constants

In computer programming, a variable is a named memory location where data is stored. For example,

```
x = 13.8
```

Here, `x` is the variable where the data **13.8** is stored. Now, whenever we use `x` in our program, we will get **13.8**.

```
x = 13.8

# print variable
print(x)
```

Output

```
[1] 13.8
```

As you can see, when we print `x` we get `13.8` as output.

Rules to Declare R Variables

As per our requirements, we can use any name for our variables. However, there are certain rules that need to be followed while creating a variable:

- A variable name in R can be created using letters, digits, periods, and underscores.
- You can start a variable name with a letter or a period, but not with digits.
- If a variable name starts with a dot, you can't follow it with digits.

- R is case sensitive. This means that `age` and `Age` are treated as different variables.
- We have some reserved words that cannot be used as variable names.

> **Note**: In earlier versions of R programming, the period `.` was used to join words in a multi-word variable such as `first.name`, `my.age`, etc. However, nowadays we mostly use `_` for multi-word variables For example, `first_name`, `my_age`, etc.

Types of R Variables

Depending on the type of data that you want to store, variables can be divided into the following types.

1. Boolean Variables

It stores single bit data which is either `TRUE` or `FALSE`. Here, `TRUE` means yes and `FALSE` means no. For example,

```
a = TRUE

print(a)
print(class(a))
```

Output

```
[1] TRUE
[1] "logical"
```

Here, we have declared the boolean variable `a` with the value `TRUE`. Boolean variables belong to the logical class so `class(a)` returns `"logical"`.

---

2. Integer Variables

It stores numeric data without any decimal values. For example,

```
A = 14L

print(A)
print(class(A))
```

Output

```
[1] 14
[1] "integer"
```

Here, `L` represents integer value. In R, integer variables belong to the integer class so, `class(a)` returns `"integer"`.

---

3. Floating Point Variables

It stores numeric data with decimal values. For example,

```
x = 13.4

print(x)
print(class(x))
```

Output

```
[1] 13.4
[1] "numeric"
```

Here, we have created a floating point variable named `x`. You can see that the floating point variable belongs to the `numeric` class.

---

4. Character Variables

It stores a single character data. For example,

```
alphabet = "a"

print(alphabet)
print(class(alphabet))
```

Output

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

Here, we have created a character variable named `alphabet`. Since character variables belong to the character class, `class(alphabet)` returns `"character"`.

---

5. String Variables

It stores data that is composed of more than one character. We use double quotes to represent string data. For example,

```
message = "Welcome to Programiz!"

print(message)
print(class(message))
```

Output

```
[1] "Welcome to Programiz!"
[1] "character"
```

Here, we have created a string variable named `message`. You can see that the string variable also belongs to the `character` class.

Changing Value of Variables

Depending on the conditions or information passed into the program, you can change the value of a variable. For example,

```
message = "Hello World!"
print(message)

# changing value of a variable
message <- "Welcome to Programiz!"


print(message)
```

Output

```
[1] "Hello World!"
[1] "Welcome to Programiz!"
```

In this program,

- `"Hello World!"` - initial value of `message`
- `"Welcome to Programiz!"` - changed value of `message`

You can see that the value of a variable can be changed anytime.

R Constants

Constants are those entities whose values aren't meant to be changed anywhere throughout the code. In R, we can declare constants using the `<-` symbol. For example,

```
x <- "Welcome to Programiz!"
```

```
print(x)
```

Output

```
[1] "Welcome to Programiz!"
```

Here, `"Welcome to Programiz!"` is a string constant.

> **Note**: Constants are also known as scalars.

Types of R Constants

In R, we have the following types of constants.

- The five types of R constants - `numeric`, `integer`, `complex`, `logical`, `string`.
- In addition to these, there are 4 specific types of R constants - `Null`, `NA`, `Inf`, `NaN`.

Let's discuss each of these types one by one.

---

1. Integer Constants

Integer constants are the integer values we use in our code. These constants end with the letter `L`. For example,

```
x <- 15L
print(typeof(x))
print(class(x))
```

Output

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

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

Here, `15L` is a constant which has been assigned to `x`. You can see that the type and class of the constant is `integer`.

We can use different types of integer constants in our code. For example,

```
# hexadecimal value
x <- 0x15L

print(x)

# exponential value
x <- 1e5L

print(x)
```

Output

```
[1] 21
[1] 100000
```

---

2. Numeric Constants

In R programming, numeric constants can be integers (`4`), floating-point numbers (`0.55`), or exponential numbers (`3e-3`). For example,

```
z <- 3e-3

print(z)  # 0.003
print(class(z))  # "numeric"

y <- 3.4
```

```
print(y)  # 3.4
print(class(z))  # "numeric"
```

Output

```
[1] 0.003
[1] "numeric"
[1] 3.4
[1] "numeric"
```

---

3. Logical Constants

Logical constants in R are either `TRUE` or `FALSE`. For example,

```
x <- TRUE
y <- FALSE
print(x)
print(y)
```

Output

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

Note: We can also use a single character to create logical constants. For example,

```
x <- T
print(x) # TRUE
```

---

4. String Constants

String constants are the string data we use in our code. For example,

```
message <- "Welcome to Programiz!!"
print(message)
```

Output

```
[1] "Welcome to Programiz!!"
```

---

5. Complex Constants

A complex constant is data that contains a real and an imaginary part (denoted by the suffix `i`). For example,

```
y <- 3.2e-1i
print(y)
print(typeof(y))
```

Output

```
[1] 0+0.32i
[1] "complex"
```

Note: Complex constants can only be purely imaginary. For example,

```
y <- 3i
print(y)    # 0+3i
print(typeof(y))  # "complex"
```

Special R Constants

R programming also provides 4 special types of constants.

- `NULL` - to declare an empty R object. For example,

```
x <- NULL
print(x)  # NULL
```

```
print(typeof(x))  # "NULL"
```

- `Inf/-Inf` - represents positive and negative infinity. For example,

```
# result is too big so it represents positive infinity
```

```
a <- 2^2020

print(a)   # Inf

# result is too big
# represents negative infinity
b <- -2^2020

print(b)    # -Inf
```

- `NaN` (Not a Number) - represents undefined numerical value. For example,

```
print(0/0)      # NaN
```

```
print(Inf/Inf)  # NaN
```

- `NA` - represents value which is not available. For example,

```
print(NA + 20) # NA
```

Built-In R Constants

R programming provides some predefined constants that can be directly used in our program. For example,

```
# print list of uppercase letters
print(LETTERS)

# print list of lowercase letters
print(letters)

# print 3 letters abbreviation of English months
print(month.abb)

# print numerical value of constant pi
print(pi)
```

Output

```
 [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"
 [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"
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
[1] 3.141593
```

In the above example, we have used the following built-in constants:

- `LETTERS` - to display a list of all uppercase letters
- `letters` - to display a list of all small letters
- `month.abb` - to print 3 letter abbreviations of all English months
- `pi` - to print the numerical value of the constant pi

R Data Types

A variable can store different types of values such as numbers, characters etc. These different types of data that we can use in our code are called **data types**. For example,

```
x <- 123L
```

Here, `123L` is an integer data. So the data type of the variable `x` is `integer`. We can verify this by printing the class of `x`.

```
x <- 123L


# print value of x
print(x)

# print type of x
print(class(x))
```

Output

```
[1] 123
[1] "integer"
```

Here, `x` is a variable of data type `integer`.

Different Types of Data Types

In R, there are 6 basic data types:

- `logical`
- `numeric`
- `integer`
- `complex`

- `character`
- `raw`

Let's discuss each of these R data types one by one.

---

1. Logical Data Type

The `logical` data type in R is also known as **boolean** data type. It can only have two values: `TRUE` and `FALSE`. For example,

```
bool1 <- TRUE

print(bool1)
print(class(bool1))

bool2 <- FALSE

print(bool2)
print(class(bool2))
```

Output

```
[1] TRUE
[1] "logical"
[1] FALSE
[1] "logical"
```

In the above example,

- `bool1` has the value `TRUE`,
- `bool2` has the value `FALSE`.

Here, we get `"logical"` when we check the type of both variables.

Note: You can also define logical variables with a single letter - `T` for `TRUE` or `F` for `FALSE`. For example,

```
is_weekend <- F
print(class(is_weekend))  # "logical"
```

2. Numeric Data Type

In R, the `numeric` data type represents all real numbers with or without decimal values. For example,

```
# floating point values
weight <- 63.5

print(weight)
print(class(weight))

# real numbers
height <- 182

print(height)
print(class(height))
```

Output

```
[1] 63.5
[1] "numeric"
[1] 182
[1] "numeric"
```

Here, both `weight` and `height` are variables of `numeric` type.

---

3. Integer Data Type

The `integer` data type specifies real values without decimal points. We use the suffix `L` to specify integer data. For example,

```
integer_variable <- 186L
print(class(integer_variable))
```

Output

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

Here, `186L` is an integer data. So we get `"integer"` when we print the class of `integer_variable`.

---

4. Complex Data Type

The `complex` data type is used to specify purely imaginary values in R. We use the suffix `i` to specify the imaginary part. For example,

```
# 2i represents imaginary part
complex_value <- 3 + 2i


# print class of complex_value
print(class(complex_value))
```

Output

```
[1] "complex"
```

Here, `3 + 2i` is of `complex` data type because it has an imaginary part `2i`.

---

5. Character Data Type

The `character` data type is used to specify character or string values in a variable.

In programming, a string is a set of characters. For example, `'A'` is a single character and `"Apple"` is a string.

You can use single quotes `''` or double quotes `""` to represent strings. In general, we use:

- `''` for character variables
- `""` for string variables

For example,

```
# create a string variable
fruit <- "Apple"

print(class(fruit))

# create a character variable
my_char <- 'A'

print(class(my_char))
```

Output

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

Here, both the variables - `fruit` and `my_char` - are of `character` data type.

---

6. Raw Data Type

A `raw` data type specifies values as raw bytes. You can use the following methods to convert character data types to a raw data type and vice-versa:

- `charToRaw()` - converts character data to raw data
- `rawToChar()` - converts raw data to character data

For example,

```
# convert character to raw
raw_variable <- charToRaw("Welcome to Programiz")


print(raw_variable)
print(class(raw_variable))
```

```
# convert raw to character
char_variable <- rawToChar(raw_variable)


print(char_variable)
print(class(char_variable))
```

Output

```
[1] 57 65 6c 63 6f 6d 65 20 74 6f 20 50 72 6f 67 72 61 6d 69 7a
[1] "raw"
[1] "Welcome to Programiz"
[1] "character"
```

In this program,

- We have first used the `charToRaw()` function to convert the string `"Welcome to Programiz"` to raw bytes.

  This is why we get `"raw"` as output when we print the class of `raw_variable`.
- Then, we have used the `rawToChar()` function to convert the data in `raw_variable` back to character form.

  This is why we get `"character"` as output when we print the class of `char_variable`.

R Print Output

R print() Function

In R, we use the `print()` function to print values and variables. For example,

```
# print values
print("R is fun")


# print variables
```

```
x <- "Welcome to Programiz"
print(x)
```

Output

```
[1] "R is fun"
[1] "Welcome to Programiz"
```

In the above example, we have used the `print()` function to print a string and a variable. When we use a variable inside `print()`, it prints the value stored inside the variable.

---

paste() Function in R

You can also print a string and variable together using the `print()` function. For this, you have to use the `paste()` function inside `print()`. For example,

```
company <- "Programiz"

# print string and variable together
print(paste("Welcome to", company))
```

Output

```
Welcome to Programiz
```

Notice the use of the `paste()` function inside `print()`. The `paste()` function takes two arguments:

- **string** - "Welcome to"
- **variable** - company

By default, you can see there is a space between string `Welcome to` and the value `Programiz`.

If you don't want any default separator between the string and variable, you can use another variant of `paste()` called `paste0()`. For example,

```
company <- "Programiz"

# using paste0() instead of paste()
print(paste0("Welcome to", company))
```

Output

```
[1] "Welcome toProgramiz"
```

Now, you can see there is no space between the string and the variable.

R sprintf() Function

The `sprintf()` function of C Programming can also be used in R. It is used to print formatted strings. For example,

```
myString <- "Welcome to Programiz"

# print formatted string
sprintf("String: %s", myString)
```

Output

```
[1] "String: Welcome to Programiz"
```

Here,

- `String: %s` - a formatted string
- `%s` - format specifier that represents string values
- `myString` - variable that replaces the format specifier `%s`

Besides `%s`, there are many other format specifiers that can be used for different types values.

Specifier	Value Type

%c	Character
%d or %i	Signed Decimal Integer
%e or %E	Scientific notation
%f	Decimal Floating Point
%u	Unsigned Decimal Integer
%p	Pointer address

Let's use some of them in examples.

```
# sprintf() with integer variable
myInteger <- 123
sprintf("Integer Value: %d", myInteger)


# sprintf() with float variable
myFloat <- 12.34
sprintf("Float Value: %f", myFloat)
```

Output

```
[1] "Integer Value: 123"
[1] "Float Value: 12.340000"
```

R cat() Function

R programming also provides the `cat()` function to print variables. However, unlike `print()`, the `cat()` function is only used with basic types like logical, integer, character, etc.

```
# print using Cat
cat("R Tutorials\n")


# print a variable using Cat
message <- "Programiz"
cat("Welcome to ", message)
```

Output

```
R Tutorials
Welcome to  Programiz
```

In the example above, we have used the `cat()` function to display a string along with a variable. The `\n` is used as a newline character.

> **Note**: As mentioned earlier, you cannot use `cat()` with list or any other object.

Print Variables in R Terminal

You can also print variables inside the R terminal by simply typing the variable name. For example,

```
# inside R terminal
x = "Welcome to Programiz!"

# print value of x in console
x

// Output: [1] "Welcome to Programiz"
```

R Numbers

Numbers in R can be divided into 3 different categories:

- **Numeric**: It represents both whole and floating-point numbers. For example, 123, 32.43, etc.
- **Integer:** It represents only whole numbers and is denoted by `L`. For example, 23L, 39L, etc.
- **Complex**: It represents complex numbers with imaginary parts. The imaginary parts are denoted by `i`. For example, 2 + 3i, 5i, etc.

Numeric Data Type

Numeric data type is the most frequently used data type in R. It is the default data type whenever you declare a variable with numbers.

You can store any type of number (with or without decimal) in a variable with `numeric` data type. For example,

```
# decimal variable
my_decimal <- 123.45

print(class(my_decimal))

# variable without decimal
my_number <- 34

print(class(my_number))
```

Output

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

Here, both the `my_decimal` and `my_number` variables are of `numeric` type.

Integer Data Type

Integers are a type of numeric data that can take values without decimal. It's mostly used when you are sure that the variable can not have any decimal values in the future.

In order to create an `integer` variable, you must use the suffix `L` at the end of the value. For example,

```
my_integer <- 123L


# print the value of my_integer
print(my_integer)

# print the data type of my_integer
print(class(my_integer))
```

Output

```
[1] 123
[1] "integer"
```

Here, the variable `my_integer` contains the value `123L`. The suffix `L` at the end of the value indicates that `my_integer` is of `integer` type.

Complex Data Type

In R, variables with complex data types contain values with an imaginary part. This can be indicated by using the `i` as a suffix. For example,

```
# variable with only imaginary part
z1 <- 5i
```

```
print(z1)
print(class(z1))

# variable with both real and imaginary parts
z2 <- 3 + 3i

print(z2)
print(class(z2))
```

Output

```
[1] 0+5i
[1] "complex"
[1] 3+3i
[1] "complex"
```

Here, the variables `z1` and `z2` have been declared as `complex` data types with an imaginary part denoted by the suffix `i`.