DATA SOCIETY®

Intro to R - Part 1

"One should look for what is and not what he thinks should be.
-Albert Einstein.


Welcome

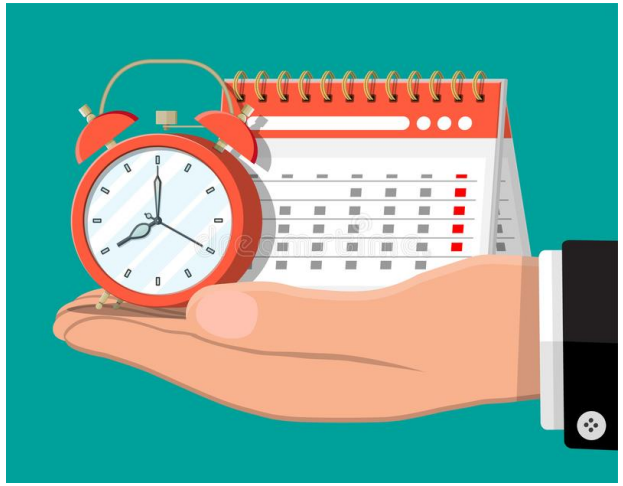
Who we are

- Data Society's mission is to **integrate Big Data and machine learning best practices across entire teams** and empower professionals to identify new insights
- We provide:
  - High-quality data science training programs
  - Customized executive workshops
  - Custom software solutions and consulting services
- Since 2014, we've worked with thousands of professionals to make their data work for them

Best practices for virtual classes

1. Find a quiet place, free of as many distractions as possible. Headphones are recommended.
2. Remove or silence alerts from cell phones, e-mail pop-ups, etc.
3. Participate in activities and ask questions.
4. Give your honest feedback so we can troubleshoot problems and improve the course.
5. Use the Q and A tab to ask questions.

The Essentials



- How we Teach:
  - Interactive polling questions to check your knowledge
  - Exercises (we give you 2 files and one has the answers)
  - Code files to help you work faster
  - Breaks (kept to 10 minutes)

Class overview

Today we will start exploring the R programming language. We will learn about:

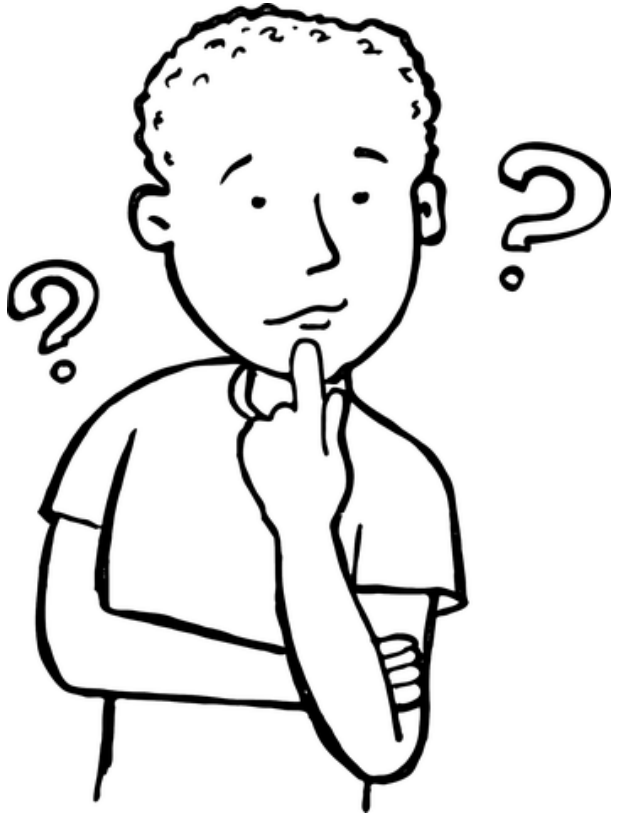
- Programming languages and their uses
- R, Rstudio and how to use them
- Performing basic calculations in R
- Working with variables in R

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	
Perform basic calculations in R	
Work with variables in R	
Identify correct naming conventions for variables in R	
Distinguish data types (`integer`, `character`, and `float`)	
Introduce vector and matrices and their operations	

What is a programming language?

- It is simply a set of instructions that you provide a computer to execute
- Computers think in 1's and 0's
- Through a programming language, we translate the 1's and 0's into something that is easily understandable by humans



Popular programming languages

- There are many programming languages that serve different purposes from statistical analysis to creating your own video games!
- Here are some of the popular programming languages:
  - **Python**: Simple, used for various applications like artificial intelligence, data science and web development
  - **JAVA**: Mostly related to developing client-server applications. It is a "write once, run anywhere" language used in mobile applications and software development
  - **R**: A popular statistical programming language amongst data scientists. It is used to analyze large amounts of data in many companies and is used for machine learning applications
  - **JavaScript**: Used to develop interactive websites
  - **C#**: Microsoft's programming language that is heavily used for game development and mobile apps
  - **Swift**: Apple's programming language used for iOS and macOS applications

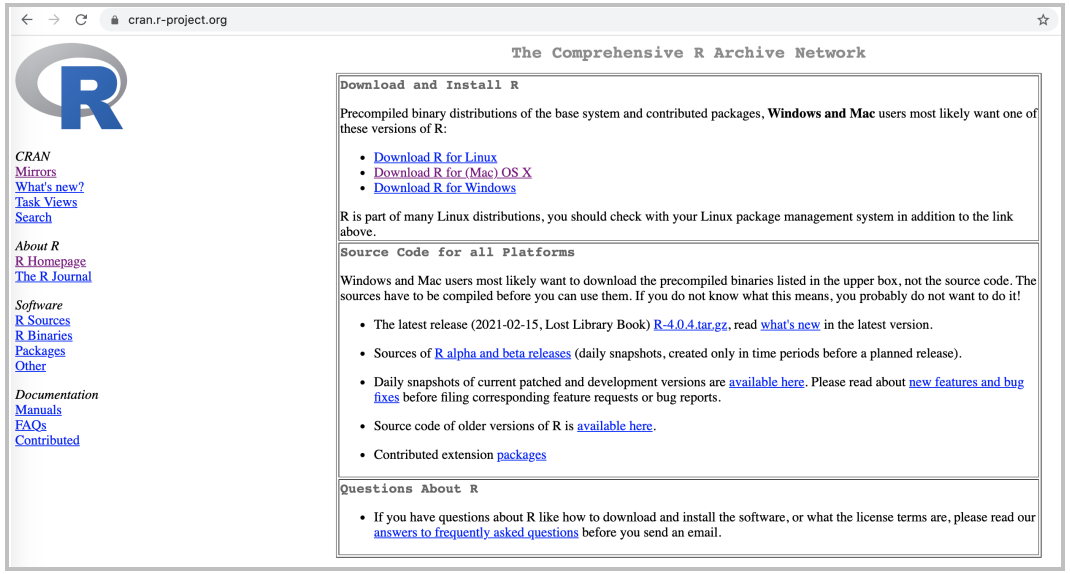
Why use R?

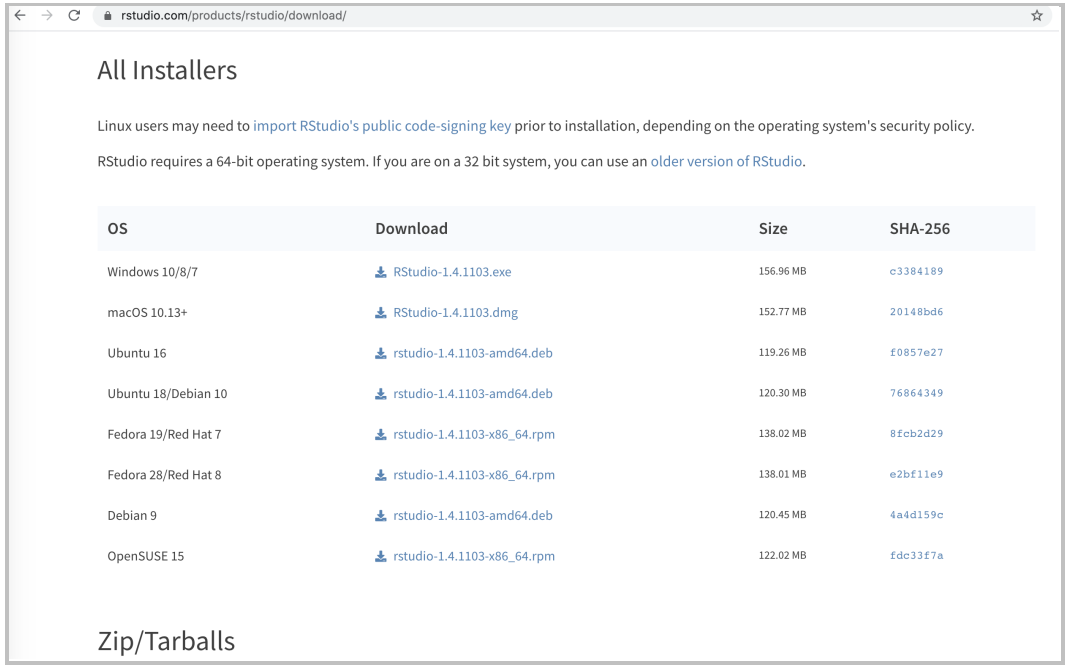
1. De facto standard among professional statisticians
2. Comparable and often superior in power to commercial products (SAS, SPSS, Stata)
3. Available for Windows, Mac and Linux operating systems
4. R is a general-purpose programming language, so you can use it to automate analyses
5. You can create dynamic graphics and visualization
6. Large community of users, many are prominent scientists: ***www.r-bloggers.com***
7. Pre-made packages to run data analyses contributed by user base (over 12,000 packages to date, and this number is constantly growing)

R compared to Excel

	R	Excel
Data capacity	R can read files as big as several gigabytes and trillions of data points; only limitation is your RAM	Excel can't read more than 1,048,576 rows and 16,384 columns (2013 version), files over ~300 megabytes can be very slow to work with
Customization	Can create custom visualizations through code, very flexible	Drop down menus limit ability to manipulate charts and graphs
Analyzing data	Powerful, pre-built packages that speed up work flow	Less flexible built-in analytic abilities that can be augmented by macros
Modeling	Data analysis and statistical models	Complex financial and accounting models
Seeing data	Built-in spreadsheet viewer	Easy to use spreadsheet interface
Usability	Direct commands similar to Excel if-statements	Keyboard shortcuts and slower point-and-click functionality

R & RStudio: installed



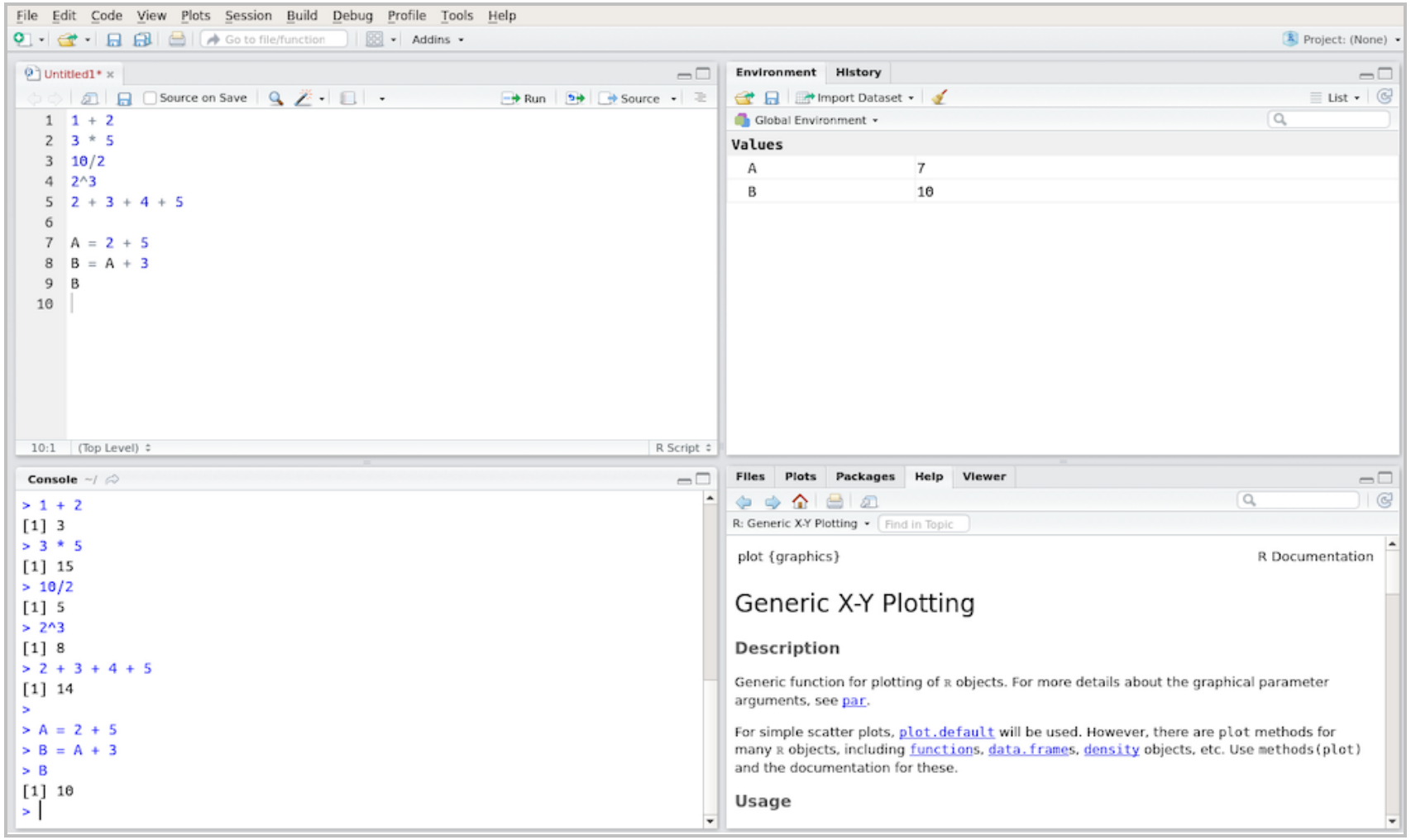


RStudio overview

A default RStudio layout includes 4 panes:

1. **Top left** pane is used as a `Script` pane where you can write and run your code as well as open other R scripts
2. **Bottom left** pane has a `Console`, which shows the output of running R commands
3. **Top right** is a helper pane that shows your `Environment` or `History`
4. **Bottom right** is another helper pane that shows `Files`, static `Plots` and interactive plots through `Viewer`, `Help`, and `Packages`

RStudio overview



Knowledge Check 1

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	✔
Perform basic calculations in R	
Work with variables in R	
Identify correct naming conventions for variables in R	
Distinguish data types (`integer`, `character`, and `float`)	
Introduce vector and matrices and their operations	

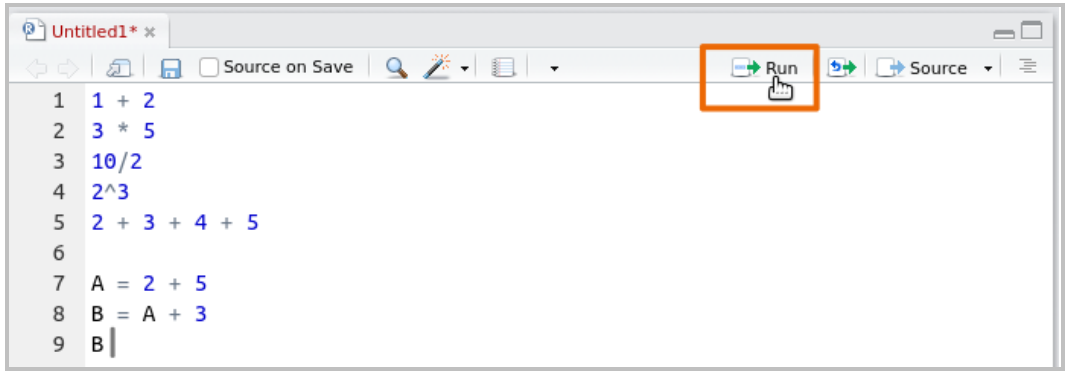
Working with R: comments

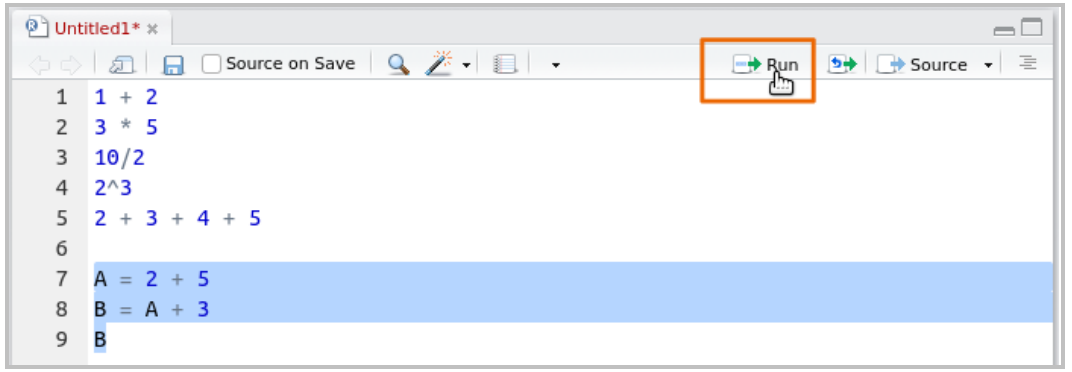
- Hashmark is used to add a comment and annotate your code
- It's good practice to leave detailed comments in your code if you intend to collaborate with others
- Comments can also help with orientation when revisiting previously written code in the future.

```
# This is a typical comment in R.
# You don't need a hashmark at
# the end of the line.
# You can add as many as you want,
# just be sure to read them afterwards :)

A = 2 + 5 #<- you can also add comments
B = A + 3 #<- to the end of the code lines
```

Executing commands in R

- Code is executed when you press `Run` in the top right corner of the script window
- R runs the line of code where your cursor is located
- You can highlight multiple lines to run at once
- An equivalent command from keyboard to `Run` button is a `Ctrl + Enter` (on PC) or `Command + Enter` (on Mac)





Basic calculations and operations

Adding (+)

```
# Add whole numbers.
1 + 2
```

```
[1] 3
```

```
# Add numbers with decimals.
3.23 + 4.65
```

```
[1] 7.88
```

Subtracting (-)

```
# Subtract whole numbers.
10 - 7
```

```
[1] 3
```

```
# Subtract numbers with decimals.
3.23 - 4.65
```

```
[1] -1.42
```

Basic calculations and operations

Multiplying (*)

```
# Multiply whole numbers.
1 * 2
```

```
[1] 2
```

```
# Multiply numbers with decimals.
3.23 * 4.65
```

```
[1] 15.0195
```

Dividing (/)

```
# Divide whole numbers.
9 / 3
```

```
[1] 3
```

```
# Divide numbers with decimals.
3.23 / 4.65
```

```
[1] 0.6946237
```

Basic calculations and operations

Square roots (sqrt())

```
# Take square root of a number with.
sqrt(100)
```

```
[1] 10
```

```
# Take square root of an expression.
sqrt(7 * 5)
```

```
[1] 5.91608
```

Exponents (^ or **)

```
# Raise number to a power with `^`.
9 ^ 3
```

```
[1] 729
```

```
# Raise number to a power with `**`.
9 ** 3
```

```
[1] 729
```

```
# Raise expression to a power.
(3.23 / 4.65)^2
```

```
[1] 0.482502
```

Basic calculations and operations

Get remainder from division (%%)

```
# Get remainder from division.
7 %% 3
```

```
[1] 1
```

```
# Get remainder from division.
4 %% 2
```

```
[1] 0
```

Perform integer division (%/%)

```
# Perform integer division.
7 %/% 3
```

```
[1] 2
```

```
# Perform integer division.
4 %/% 2
```

```
[1] 2
```

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	✔
Perform basic calculations in R	✔
Work with variables in R	
Identify correct naming conventions for variables in R	
Distinguish data types (`integer`, `character`, and `float`)	
Introduce vector and matrices and their operations	

Variables and assignment operators

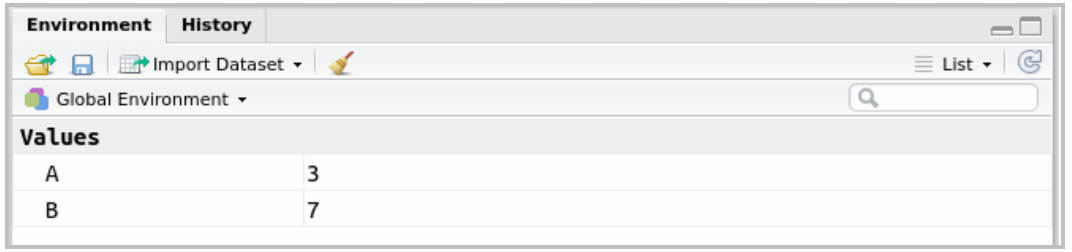
```
# Define a variable using `<-`
# as an assignment operator.
A <- 3
A
```

```
[1] 3
```

```
# Define a variable using `=`
# as an assignment operator.
B = 2 + 5
B
```

```
[1] 7
```

Notice that you not only can assign numbers to variables, you can assign any expression to a variable!



- You can set variables by setting numbers equal to letters or terms. R has **two** assignment operators: <– and =
- When a variable is named (instantiated), R stores it in its "environment"
- R session uses the values stored within its environment for all calculations within that session

Operations with variables

Adding

- You can add variables

```
# Add 2 variables.
C = A + B
C
```

```
[1] 10
```

```
# Add a variable and a number.
D = C + 5
D
```

```
[1] 15
```

The same stands for all other arithmetic operations!

Subtracting

- You can subtract variables

```
# Subtract 2 variables from each other.
D - C
```

```
[1] 5
```

```
# Subtract a variable from number.
33 - D
```

```
[1] 18
```

```
# Or a number from a variable.
D - 33
```

```
[1] -18
```

Other operators: comparison

```
# Check variables are equal.
A == B
```

```
[1] FALSE
```

```
# Check if variables are not equal.
A != B
```

```
[1] TRUE
```

```
# Check if one is greater than the other.
A > B
```

```
[1] FALSE
```

```
# Check if one is greater than or equal to 5.
A >= 5
```

```
[1] FALSE
```

```
# Check if one is less than or equal to 3.
A <= 3
```

```
[1] TRUE
```

```
# Check if one is smaller than the other.
A < B
```

```
[1] TRUE
```

Other operators: unary, logical, assignment

Operator	Description
`-x, +x`	Unary minus and unary plus are used to indicate whether a number/variable is negative or positive
`!`	Logical NOT is used to negate a statement
`&`	Element-wise logical AND
`&&`	Logical AND
`|`	Element-wise logical OR
`||`	Logical OR
`->, ->>`	Rightward assignment
`<-, <<-`	Leftward assignment
`=`	Leftward assignment

Grouping multiple operations with variables

```
# You can group several operations into
# a single one.
((D - C) * 2) ^ (1 / 3)
```

```
[1] 2.154435
```

```
# This is equivalent to a series of steps
# like ones below.
step1 = D - C
step1
```

```
[1] 5
```

```
step2 = step1 * 2
step2
```

```
[1] 10
```

```
step3 = 1 / 3
step3
```

```
[1] 0.3333333
```

```
step4 = step2 ^ step3
step4
```

```
[1] 2.154435
```

Grouping multiple operations with variables

```
# You can group several operations into a single one.
((D - C) * 2) ^ (1 / 3)
```

```
[1] 2.154435
```

```
# Be careful with your operators and your parentheses, the following expression
# will not return the same result as the one above!
((D - C) * 2) ^ 1 / 3
```

```
[1] 3.333333
```

We can see that when we removed parentheses around 1/3 we got a very different result, because the order of operations has changed!

Operator precedence

Operator	Description	Associativity
`()`	Parenthesis	Left to right
`^`	Exponent	Right to left
`-x, +x`	Unary minus, unary plus	Left to right
`%%, %/%`	Modulus, integer division	Left to right
`*, /`	Multiplication, division	Left to right
`+, -`	Addition, subtraction	Left to right
`<, >, <=, >=, ==, !=`	Comparisons	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

Variable value re-assignment

```
# 1. Create a variable and assign 67 to it.
this_variable = 67
this_variable
```

```
[1] 67
```

```
# 2. Create another variable and assign -54.
that_variable = -54
that_variable
```

```
[1] -54
```

```
# 3. Calculate their sum.
this_variable + that_variable
```

```
[1] 13
```

```
# 4. Re-assign a value to `this_variable`.
this_variable = 35
this_variable
```

```
[1] 35
```

```
# 5. Add two variables and store the result
#    in `that_variable`.
that_variable = this_variable + that_variable
that_variable
```

```
[1] -19
```

You can re-assign values, variables and expressions to variables you've already used, just be sure to keep track and not to overwrite something you didn't intend to!

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	✔
Perform basic calculations in R	✔
Work with variables in R	✔
Identify correct naming conventions for variables in R	
Distinguish data types (`integer`, `character`, and `float`)	
Introduce vector and matrices and their operations	

Naming variables and functions

Naming rules

1. Names of variables and functions can be a combination of letters, digits, period (`.`) and underscore (_)
2. They **must** start with a letter or a period; if it starts with a period, it cannot be followed by a digit
3. Reserved words in R **cannot** be used as variable or function names!

Although all of the examples have valid names, not all of them are easy to read and interpret. If you chose one style over the other, stick to it, it will make your coding style more consistent and easy to follow!

Examples

```
this_is_a_valid_name = -5
this_is_a_valid_name
```

```
[1] -5
```

```
This.Is.Also.A.Valid.Name = 3
This.Is.Also.A.Valid.Name
```

```
[1] 3
```

```
.another.valid.name3 = -Inf
.another.valid.name3
```

```
[1] -Inf
```

Naming variables and functions: CAUTION

- Variable and function names are **case sensitive**

```
# R is case sensitive!
X = 35.5 #<- this `X`
X
```

```
[1] 35.5
```

```
x = -9   #<- is not the same as this `x`
x
```

```
[1] -9
```

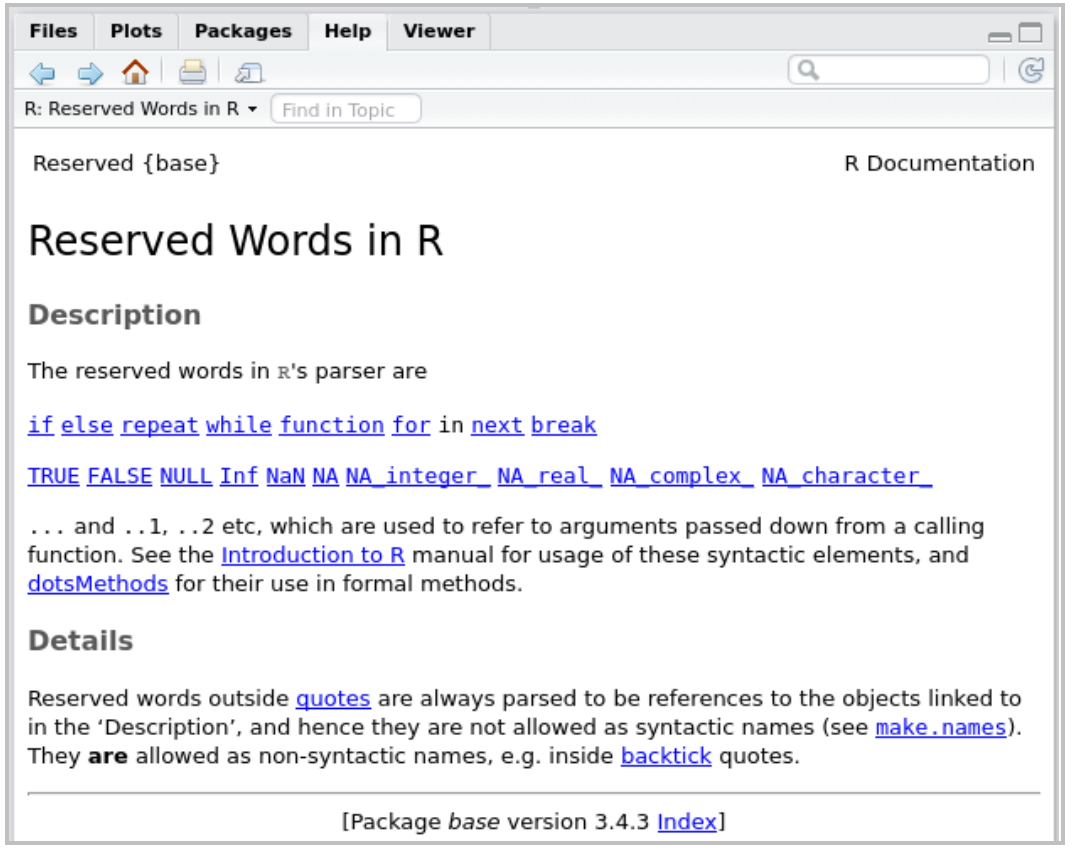
- Reserved names/letters you **cannot** use as variable names

```
# Don't use `T` or `F`, they are reserved
# as a shorthand for `TRUE` and `FALSE`.
T
F

# Don't use `TRUE` and `FALSE` either!
TRUE
FALSE

# Don't use `NULL`,`NA`, `NaN`, `Inf`!
NULL
NA
NaN
Inf
```

Naming variables and functions: CAUTION

```
?reserved
```

- To see a full list of reserved words in R you can run `?reserved` in R and you will find all of the documentation in the `Help` pane of RStudio



Knowledge Check 2

Exercise 1

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	✔
Perform basic calculations in R	✔
Work with variables in R	✔
Identify correct naming conventions for variables in R	✔
Distinguish data types (`integer`, `character`, and `float`)	
Introduce vector and matrices and their operations	

Type

- **Data type** is a set of values with common characteristics, from which expressions and functions may be formed
- It defines **the meaning of data and the way values of that type can be stored**
- For instance, a **web page** is a type and any web page has the following basic characteristics:
  - Address
  - Layout (or absence of thereof)
  - Data (or absence of thereof)
  - Integration with other web pages into a website
  - Community which allows people to update its content
  - Web server where web pages are stored

Basic data classes and types

- **Data type** describes how internal R language stores our data, while **data class** is more generic and determined by the object-oriented programming mechanism behind R
- In most business cases, we do not distinguish between data types and data classes
- The point is to adopt the data type or data class that fits best

Data class (high level)	Data type (low level)	Example
Integer	Integer	`-1, 5, or 1L, 5L`
Numeric	Double, float	`2.54`
Character	Character	`"Hello"`
Logical	Logical	`TRUE, FALSE`

- *Note: One of the common sources of errors for a person learning to use any programming language is the data type conversion.*

Basic data classes: what we will use

- To generate more insights within our data, here is a list of functions we can use

Item	Purpose
Value	Example of class
`typeof()`	Finds the type of the variable
`class()`	Returns the class of the variable
boolean function	Specific function that checks class and returns TRUE or FALSE
`attributes()`	Checks the metadata/attribute of the variable
`length()`	Checks the length of the object

Basic data classes: integer

Item	Integer
Value	24, 34L
`typeof()`	integer
`class()`	integer
boolean function	`is.integer()`
`attributes()`	`NULL`
`length()`	1

- *Note: we can use the **L** suffix to qualify any number with the intent of making it an explicit integer*

```
# Create an integer type variable.
integer_var = 34L

# Check type of variable.
typeof(integer_var)
```

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

```
# Check if the variable is integer.
is.integer(integer_var)
```

```
[1] TRUE
```

```
# Check length of variable
# (i.e. how many entries).
length(integer_var)
```

```
[1] 1
```

Basic data classes: numeric

Item	Numeric
Value	24.24
`typeof()`	double
`class()`	numeric
boolean function	`is.numeric()`
`attributes()`	`NULL`
`length()`	1

```
# Create a numeric class variable.
numeric_var = 24.24
typeof(numeric_var)
```

```
[1] "double"
```

```
# Check length of variable
# (i.e. how many entries).
length(numeric_var)
```

```
[1] 1
```

Basic data classes: character

Item	Character
Value	`"Hello"`
`typeof()`	character
`class()`	character
boolean function	`is.character()`
`attributes()`	NULL
`length()`	1

```
# Create a character class variable.
character_var = "Hello"
```

```
# Check if the variable is character.
is.character(character_var)
```

```
[1] TRUE
```

```
# Check metadata/attributes of variable.
attributes(character_var)
```

```
NULL
```

```
# Check length of variable
# (i.e. how many entries).
length(character_var)
```

```
[1] 1
```

Some useful character operations

```
# Create another character class variable.
case_study = "JUmbLEd CaSE"

# Convert a character string to lower case.
tolower(case_study)
```

```
[1] "jumbled case"
```

```
# Convert a character string to upper case.
toupper(case_study)
```

```
[1] "JUMBLED CASE"
```

```
# Count number of characters in a string.
nchar(case_study)
```

```
[1] 12
```

```
# Compare to the output of the `length` command.
length(case_study)
```

```
[1] 1
```

```
# Get just a part of character string.
substr(case_study, #<- original string
       1,          #<- start index of substring
       7)          #<- end index of substring
```

```
[1] "JUmbLEd"
```

Basic data classes: logical

Item	Logical
Value	`TRUE` or `FALSE`
`typeof()`	logical
`class()`	logical
boolean function	`is.logical()`
`attributes()`	`NULL`
`length()`	1

```
# Create a logical class variable.
logical_var = TRUE

# Check type of variable.
typeof(logical_var)
```

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

Basic data classes: summary & conversion

Item	Integer	Numeric	Character	Logical
Value	`24, 34L`	`24.34`	Hello	`TRUE` or `FALSE`
`typeof()`	integer	double	character	logical
`class()`	integer	numeric	character	logical
boolean function	`is.integer()`	`is.numeric()`	`is.character()`	`is.logical()`
`attributes()`	`NULL`	`NULL`	`NULL`	`NULL`
`length()`	1	1	1	1
To convert a variable to this type	`as.integer()`	`as.numeric()`	`as.character()`	`as.logical()`

Exercise 2

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	✔
Perform basic calculations in R	✔
Work with variables in R	✔
Identify correct naming conventions for variables in R	✔
Distinguish data types (`integer`, `character`, and `float`)	✔
Introduce vector and matrices and their operations	

Basic data structures

- In the past few slides, we have learned some of the most basic as well as common data types
- Next, we are going to focus on groupings of one or more data types organized in various ways - **data structure**
- A data structure is a method for **describing a certain way to organize pieces of data**, so operations and algorithms can be easily applied

Data structure	Number of dimensions	Single data type	Multiple data types
Vector (Atomic vector)	1 (entries)	✔	✘
Vector (List)	1 (entries)	✔	✔
Matrix	2 (rows and columns)	✔	✘
Data frame	2 (rows and columns)	✔	✔

Basic data structures: atomic vectors

- `Vector` is a collection of elements of the **same** type
  - `Mode` of a vector tells us which types of elements it contains
  - Most common `modes` of vectors are: `character`, `logical`, `numeric`
- Vector is the most universal, common, and simplest data structure present in nearly all programming languages including low-level programming languages
- It is called an **array** in all other programming languages except R
- An array with one dimension is almost the same as a vector so we will not differentiate between them here

Your computer's memory is one giant single-dimensional array!

Basic data structures: atomic vectors

```
# To make an empty vector in R,
# you have a few options:
# Option 1: use `vector()` command.
# The default in R is an empty vector of
# `logical` mode!
vector()
```

```
logical(0)
```

```
# Option 2: use `c()` command
# (`c` stands for concatenate).
# The default empty vector produced by `c()`
# has a single entry `NULL`!
c()
```

```
NULL
```

- *The length of an empty vector will always be 0 since it has no entries in it!*

- To make a vector out of a given set of character strings, you can wrap them into `c` and separate by commas

```
# Make a vector from a set of char. strings
c("My", "name", "is", "Vector")
```

```
[1] "My"     "name"   "is"     "Vector"
```

- To make a vector out of a given set of numbers you can wrap them into a vector `c` and separate elements by commas

```
# Make a vector out of given set of numbers
c(1, 2, 3, 765, -986, 0.5)
```

```
[1]    1.0    2.0    3.0  765.0 -986.0    0.5
```

Basic data structures: atomic vectors

```
# Create a vector of mode `character` from
# pre-defined set of character strings.
character_vec = c("My", "name", "is", "Vector")
character_vec
```

```
[1] "My"     "name"   "is"     "Vector"
```

```
# Check if the variable is character.
is.character(character_vec)
```

```
[1] TRUE
```

```
# Check metadata/attributes of variable.
attributes(character_vec)
```

```
NULL
```

Item	Vector
Value	`character_vec`
`typeof()`	character
`class()`	character
boolean function	`is.character()`
`attributes()`	`NULL`
`length()`	4

```
# Check length of variable
# (i.e. how many entries).
length(character_vec)
```

```
[1] 4
```

Basic data structures: access vectors values

```
# To access an element inside of the
# vector use `[]` and the index of the element.
character_vec[1]
```

```
[1] "My"
```

```
# To access multiple elements inside of
# a vector use the start and end indices
# with `:` in-between.
character_vec[1:3]
```

```
[1] "My"   "name" "is"
```

- *All data structures, including vectors, start at index 1!*

```
# A special form of a vector in R is a sequence.
number_seq = seq(from = 1, to = 5, by = 1)
number_seq
```

```
[1] 1 2 3 4 5
```

```
# Check class.
class(number_seq)
```

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

```
# Subset the first 3 elements.
number_seq[1:3]
```

```
[1] 1 2 3
```

Basic data structures: operations on vectors

```
number_seq      #<- Let's take our vector.
```

```
[1] 1 2 3 4 5
```

```
number_seq + 5  #<- Add a number to every entry.
```

```
[1]  6  7  8  9 10
```

```
number_seq - 5  #<- Subtract a number from every entry.
```

```
[1] -4 -3 -2 -1  0
```

```
number_seq * 2  #<- Multiply every entry by a number.
```

```
[1]  2  4  6  8 10
```

- *Note: All arithmetic operations in R are element-wise, which means data is operated element by element*

```
# To sum all elements use `sum`.
sum(number_seq)
```

```
[1] 15
```

```
# To multiply all elements use `prod`.
prod(number_seq)
```

```
[1] 120
```

```
# To get the mean of all vector
# values use `mean`.
mean(number_seq)
```

```
[1] 3
```

```
# To get the smallest value
# in a vector use `min`.
min(number_seq)
```

```
[1] 1
```

Basic data structures: appending & naming

```
# To name each entry in a vector use `names`.
names(number_seq) = c("First", "Second",
                      "Third", "Fourth",
                      "Fifth")

# Check the attributes of vector.
attributes(number_seq)
```

```
$names
[1] "First"  "Second" "Third"  "Fourth" "Fifth"
```

```
# Check the length of vector.
length(number_seq)
```

```
[1] 5
```

Item	Vector
Value	number_seq
typeof()	double
class()	numeric
boolean function	is.numeric()
attributes()	names
length()	5

```
# To append elements to a vector, just
# wrap the vector and additional element(s)
# into `c` again!
character_vec = c(character_vec, "!")
character_vec
```

```
[1] "My"     "name"   "is"     "Vector" "!"
```

Basic data structures: why ATOMIC vectors?

- What happens if you mix different types of data inside of an atomic vector?

```
# Create a vector with entries of different type.
atomic_vec = c(333, "some text", TRUE, NULL)
atomic_vec
```

```
[1] "333"       "some text" "TRUE"
```

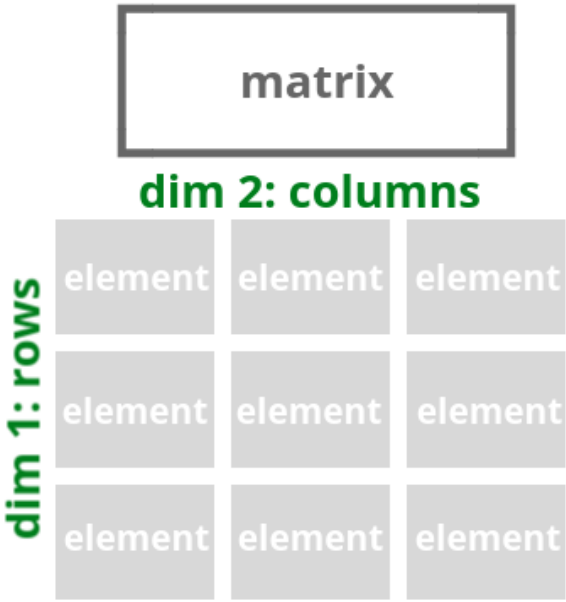
```
# Check class of the resulting vector.
class(atomic_vec)
```

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

- R will **cast** (i.e. coerce) all elements of that vector to a type/class that **can most easily accommodate all elements it contains**!
- This is why this type of data structure is called `atomic`, which, in the computer science world, is equivalent to homogeneous or unsplittable (although we all know we can split the `atom`)

Basic data structures: matrix

- A `matrix` is a **two-dimensional `vector`**
  - It is an array of elements with 2 dimensions instead of 1
  - `Matrix` data structure only allows elements of the same **type**
  - Working with matrices is very similar to working with 1D vectors



Basic data structures: making matrices

```
# Create a matrix with 3 rows and 3 columns.
sample_matrix1 = matrix(nrow = 3, #<- n rows
                        ncol = 3) #<- m cols
sample_matrix1
```

```
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
[3,]   NA   NA   NA
```

```
# Notice that by default an empty matrix
# will be filled with `NA`s.

# Check matrix dimensions.
dim(sample_matrix1)
```

```
[1] 3 3
```

```
# Notice that the `length` command will produce
# the total number elements in the matrix
# (length = n rows x m cols).
length(sample_matrix1)
```

```
[1] 9
```

```
# Another way to create a matrix is to make
# it out of a vector of numbers.
sample_matrix2 = 1:9 #<- another way to make
                     #   a sequence of numbers!

# Assign dimensions to matrix:
# 1st number is for rows, 2nd is for columns.
dim(sample_matrix2) = c(3, #<- n rows
                        3) #<- m cols

sample_matrix2
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
# Check matrix dimensions.
dim(sample_matrix1)
```

```
[1] 3 3
```

Basic data structures: making matrices

- The shorthand version of the previous two commands looks like this

```
# Create a matrix from a sequence of numbers
# with 3 rows & 3 columns.
sample_matrix3 = matrix(1:9,       #<- entries
                        nrow = 3, #<- n rows
                        ncol = 3) #<- m cols
sample_matrix3
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

- *Notice that the `matrix` command arranges the values by `column` by default!*

- Create the same matrix but with values arranged by **rows**

```
# Create a matrix from a sequence of numbers
# with 3 rows & 3 columns arranged by row.
sample_matrix4 = matrix(1:9,
                        nrow = 3,
                        ncol = 3,
                        byrow = TRUE)
sample_matrix4
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Basic data structures: working with matrices

```
# Check type of variable.
typeof(sample_matrix4)
```

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

```
# Check class of variable.
class(sample_matrix4)
```

```
[1] "matrix" "array"
```

```
# Check if the variable of type `integer`.
is.integer(sample_matrix4)
```

```
[1] TRUE
```

```
# Check metadata/attributes of variable.
attributes(sample_matrix4)
```

```
$dim
[1] 3 3
```

Basic data structures: working with matrices

```
# To append rows to a matrix, use `rbind`.
new_matrix1 = rbind(sample_matrix4,
                    10:12)
new_matrix1
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12
```

```
# To append columns to a matrix, use `cbind`.
new_matrix2 = cbind(sample_matrix3,
                    10:12)
new_matrix2
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

```
# To access an element of a matrix use
# the row and column indices separated
# by a comma inside of `[]`.
new_matrix1[1, 2] #<- element in row 1, col 2
```

```
[1] 2
```

```
# To access a row leave the space in
# column index empty.
new_matrix1[1 , ]
```

```
[1] 1 2 3
```

```
# To access a column leave the space in
# row index empty.
new_matrix1[ , 2]
```

```
[1]  2  5  8 11
```

Basic data structures: operations on matrices

```
# Let's take a sample matrix.
sample_matrix2
```

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

```
# Add a number to every entry.
sample_matrix2 + 5
```

```
     [,1] [,2] [,3]
[1,]    6    9   12
[2,]    7   10   13
[3,]    8   11   14
```

```
# Multiply every entry by a number.
sample_matrix2 * 2
```

```
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
```

```
# To sum all elements use `sum`.
sum(sample_matrix2)
```

```
[1] 45
```

```
# To multiply all elements use `prod`.
prod(sample_matrix2)
```

```
[1] 362880
```

```
# To get the mean of all matrix
# values use `mean`.
mean(sample_matrix2)
```

```
[1] 5
```

```
# To get the smallest value
# in a matrix use `min`.
min(sample_matrix2)
```

```
[1] 1
```

Basic data structures: names & attributes

```
# To name columns of a matrix use `colnames`.
colnames(sample_matrix2) = c("Col1", "Col2",
"Col3")

# To name rows of a matrix use `rownames`.
rownames(sample_matrix2) = c("Row1", "Row2",
"Row3")
sample_matrix2
```

```
     Col1 Col2 Col3
Row1    1    4    7
Row2    2    5    8
Row3    3    6    9
```

```
# Check the attributes of a matrix.
attributes(sample_matrix2)
```

```
$dim
[1] 3 3

$dimnames
$dimnames[[1]]
[1] "Row1" "Row2" "Row3"

$dimnames[[2]]
[1] "Col1" "Col2" "Col3"
```

Item	Matrix
To create	`matrix()`
Value	`sample_matrix2`
`typeof()`	integer
`class()`	matrix
boolean function	`is.matrix()`
`attributes()`	`dim`, `dimnames[[1]]`, `dimnames[[2]]`
`length()`	9

Knowledge Check 3

Exercise 3

Module completion checklist

Objective	Complete
Introductions and overview	✔
Overview of programming languages and introduce R	✔
Perform basic calculations in R	✔
Work with variables in R	✔
Identify correct naming conventions for variables in R	✔
Distinguish data types (`integer`, `character`, and `float`)	✔
Introduce vector and matrices and their operations	✔

Summary

- We have successfully started our journey with R and gotten familiar with several tools we will use throughout this course
- You have **named your own variables, performed basic operations and explored data types, vectors and matrices** in R to get familiar with the coding environment
- **In our next module**, we are going to learn more about other data structures like lists and dataframes and how to perform various operations on them

This completes our module

Congratulations!