# Intro to R - Data Types and Data Structures - 2

*One should look for what is and not what he thinks should be. (Albert Einstein)*

# Module completion checklist

| Objective | Complete |
|---|---|
| Introduce `vector` and `matrices` and their operations | |
| Introduce `list` and `dataframe` in R | |

## Review: basic data structures

| 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

```
# 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 using `c` and separate them with 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 them with 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 (cont'd)

- Now let's get some information about a vector

```
# 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: accessing vector values

- Remember that in R, all elements in a data structure correspond to an **index** of numerical values beginning with **1**

```
# 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"
```

- A special kind of vector in R is a sequence
- Let's create a sequence and then derive a subset

```
# Start by creating a sequence of numbers from 1
to 5.
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

- In R, all arithmetic operations will be carried out **element by element**

```
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
```

```
# 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: why atomic vectors?

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

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

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

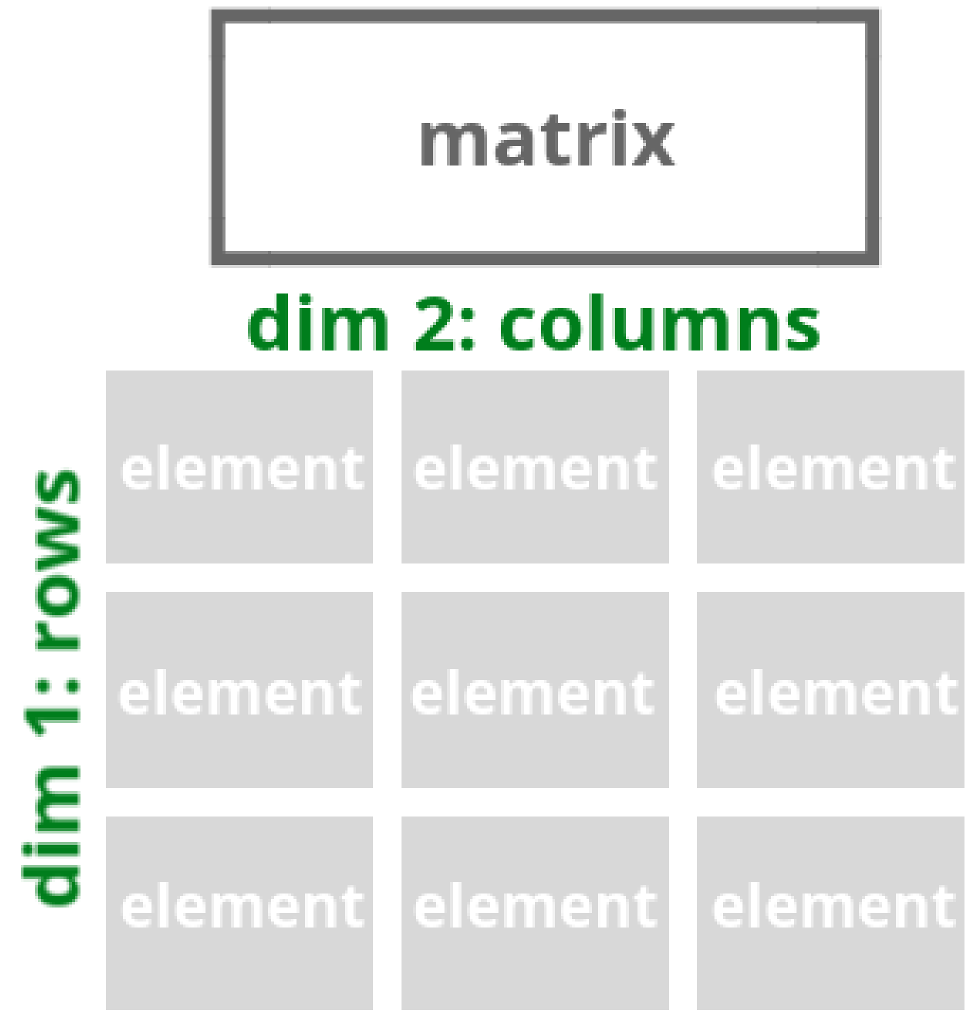
```r
# Check class of the resulting vector.
class(atomic_vec)
```

```
[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 computer science is equivalent to homogeneous or unsplittable

## Basic data structures: matrix

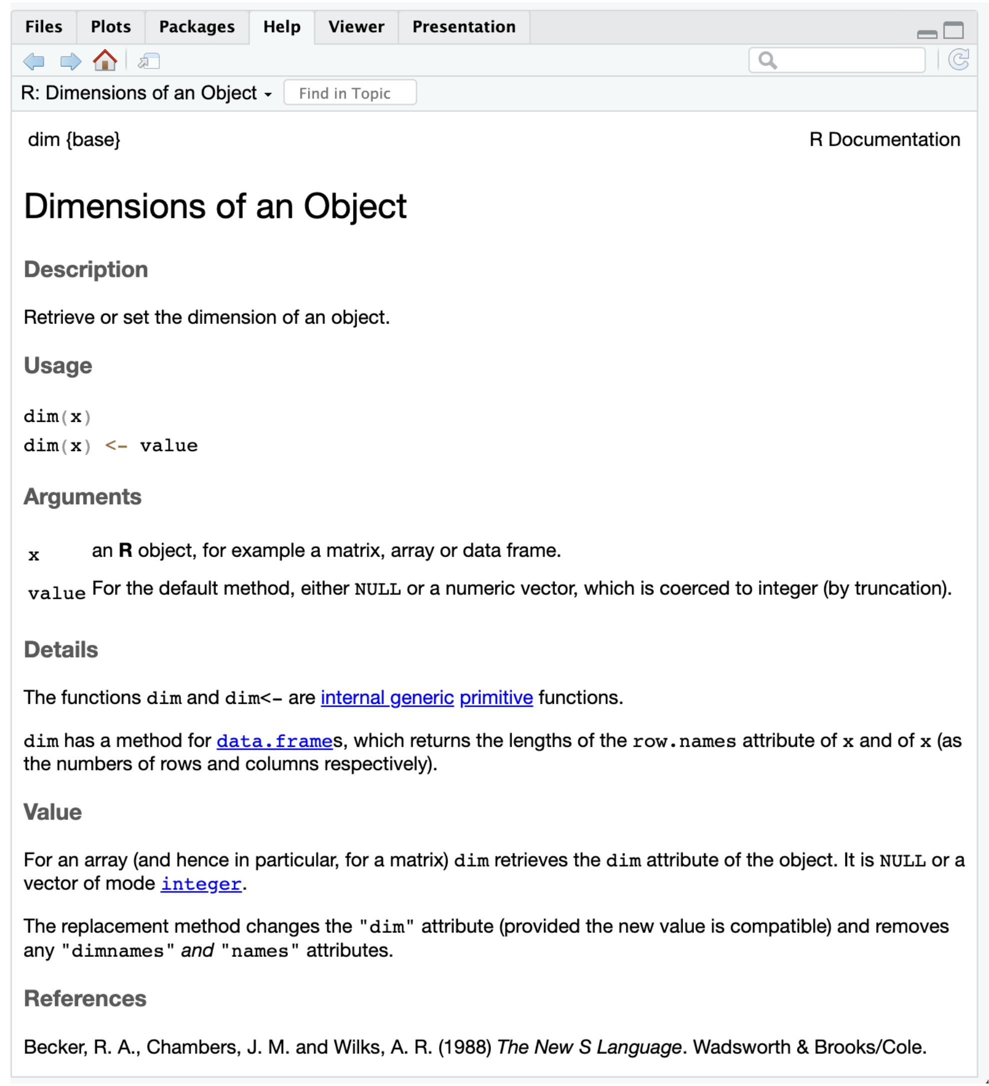
- A matrix is a two-dimensional vector
  - It is an array of elements with **2 dimensions** instead of 1
  - Matrix data structures only allow elements of the same type
  - Working with matrices is very similar to working with 1D vectors



## Basic data structures: working with multiple dimensions

- To quickly identify the dimensions of any object, use the base R function `dim()`
- It takes one argument
  - x: an R object like a vector, matrix, or dataframe
- It returns the number of columns and number of rows

```
?dim          #<- Check R documentation
dim(object)   #<- Any R object
```



dim {base}    R Documentation

Dimensions of an Object

**Description**

Retrieve or set the dimension of an object.

**Usage**

```
dim(x)
dim(x) <- value
```

**Arguments**

| | |
|---|---|
| `x` | an **R** object, for example a matrix, array or data frame. |
| `value` | For the default method, either `NULL` or a numeric vector, which is coerced to integer (by truncation). |

**Details**

The functions `dim` and `dim<-` are internal generic primitive functions.

`dim` has a method for `data.frame`s, which returns the lengths of the `row.names` attribute of `x` and of `x` (as the numbers of rows and columns respectively).

**Value**

For an array (and hence in particular, for a matrix) `dim` retrieves the `dim` attribute of the object. It is `NULL` or a vector of mode `integer`.

The replacement method changes the `"dim"` attribute (provided the new value is compatible) and removes any `"dimnames"` *and* `"names"` attributes.

**References**

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole.

## Basic data structures: making matrices

- Let’s start by creating 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
```

```
# An empty matrix will default to `NA`s.

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

```
[1] 3 3
```

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

```
[1] 9
```

- We can also make matrices out of a vector of numbers, like a sequence

```
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_matrix2)
```

```
[1] 3 3
```

## Basic data structures: working with matrices

- Again, let's gather some information about this `sample_matrix1`

```
# Check type of variable.
typeof(sample_matrix1)
```

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

```
# Check class of variable.
class(sample_matrix1)
```

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

```
# Check if the variable of type `integer`.
is.integer(sample_matrix1)
```

```
[1] FALSE
```

```
# Check metadata/attributes of variable.
attributes(sample_matrix1)
```

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

# 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)
```

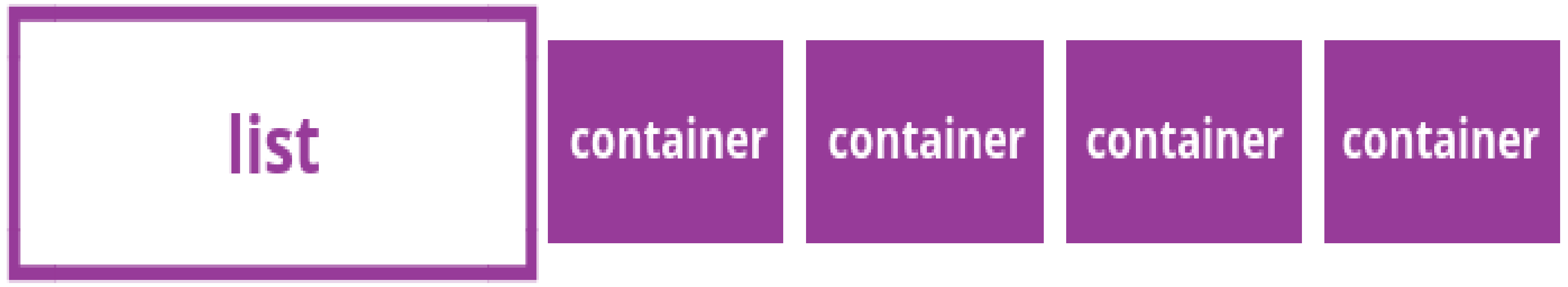
```
[1] 1
```

# Module completion checklist

| Objective | Complete |
|---|---|
| Introduce `vector` and `matrices` and their operations | ✓ |
| Introduce `list` and `dataframe` in R | |

# Basic data structures: lists

- A list is a collection of entries that act as a container
- Unlike an atomic vector, a list is a generic vector that can hold elements of different types
- A list has a single dimension at its top level, even if it contains elements like matrices
- Lists can be nested, which means that a list can contain elements that are also lists



## Basic data structures: lists

- Let's try creating a list

```
# To make an empty list in R,
# you have a few options:
# Option 1: use `list()` command.
list()
```

```
list()
```

- We can add elements of different types

```
# Make a list with different entries.
sample_list = list(1, "am", TRUE)
sample_list
```

```
[[1]]
[1] 1

[[2]]
[1] "am"

[[3]]
[1] TRUE
```

## Basic data structures: naming list elements

- Lists can have attributes such as names
- You can name list elements **when** you create a list

```
# Create a named list.
sample_list_named = list(One = 1,
                         Two = "am",
                         Three = TRUE)
sample_list_named
```

```
$One
[1] 1

$Two
[1] "am"

$Three
[1] TRUE
```

```
attributes(sample_list_named)
```

```
$names
[1] "One"   "Two"   "Three"
```

- You can also set element names **after** the list has been created

```
# Name existing list.
names(sample_list) = c("One", "Two", "Three")
sample_list
```

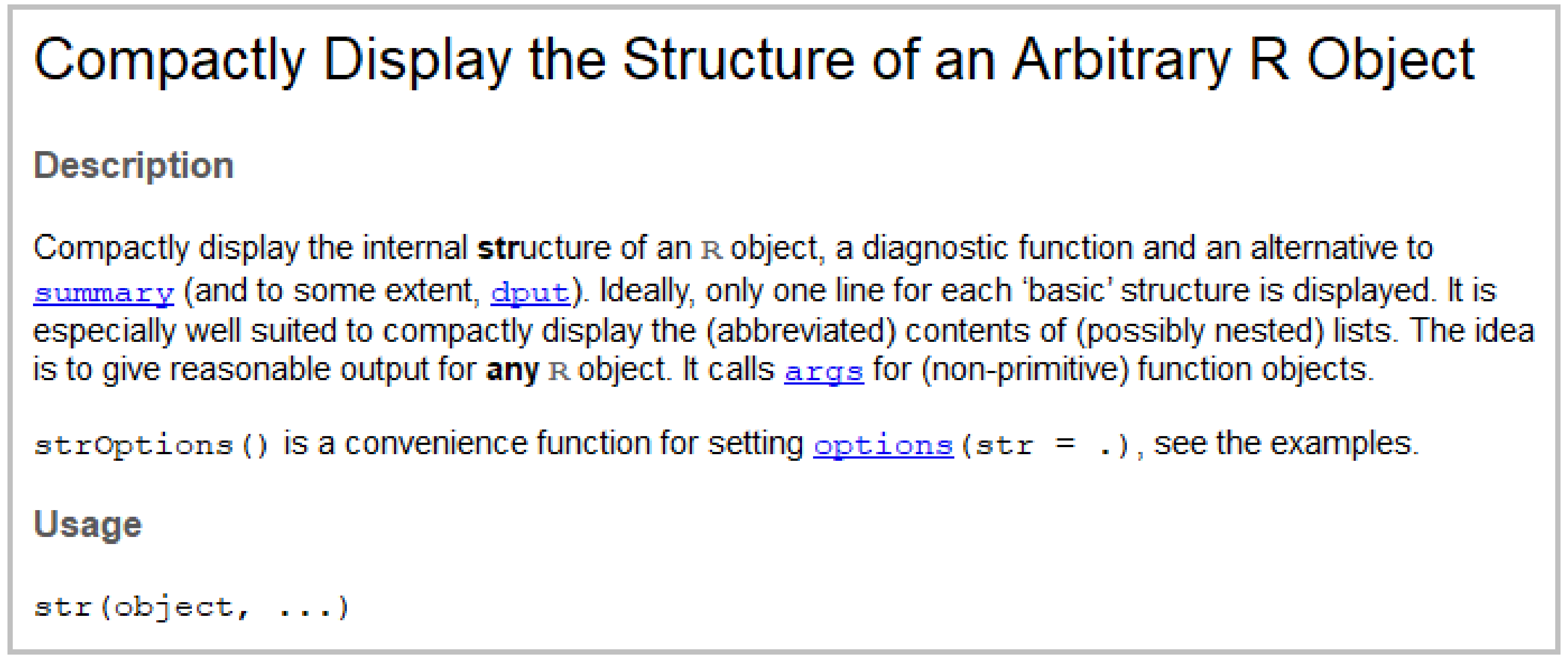
```
$One
[1] 1

$Two
[1] "am"

$Three
[1] TRUE
```

```
attributes(sample_list)
```

```
$names
[1] "One"   "Two"   "Three"
```

## Basic data structures: introducing structure

```
?str           #<- Check R documentation
str(object) #<- Any R object
```



### Compactly Display the Structure of an Arbitrary R Object

**Description**

Compactly display the internal **str**ucture of an R object, a diagnostic function and an alternative to summary (and to some extent, dput). Ideally, only one line for each 'basic' structure is displayed. It is especially well suited to compactly display the (abbreviated) contents of (possibly nested) lists. The idea is to give reasonable output for **any** R object. It calls args for (non-primitive) function objects.

`strOptions()` is a convenience function for setting `options(str = .)`, see the examples.

**Usage**

```
str(object, ...)
```

```
# Inspect the list's structure.
str(sample_list)
```

```
List of 3
 $ One   : num 1
 $ Two   : chr "am"
 $ Three: logi TRUE
```

- The command `str()` lets you inspect the structure of any R object such as a list or a dataframe
- It returns:
  - The `class` of the object (e.g. `List`)
  - The `length` of the object (e.g. `3`)
  - Snippet of each entry and its `type` (e.g. `One: num 1`, `Two: chr "am"`, `Three: logi TRUE`)

## Basic data structures: accessing data within lists

- To access an element in a list, you can use its **index**

```
# Access an element of a list.
sample_list[[2]]
```

```
[1] "am"
```

```
# Access a sub-list with its element(s).
sample_list[2]
```

```
$Two
[1] "am"
```

```
# Access a sub-list with its element(s).
sample_list[2:3]
```

```
$Two
[1] "am"

$Three
[1] TRUE
```

- You can also refer to an element by its name, using the `$` operator (as seen in the output of the `str` command)

```
# Access named list elements.
sample_list$One
```

```
[1] 1
```

```
sample_list$Two
```

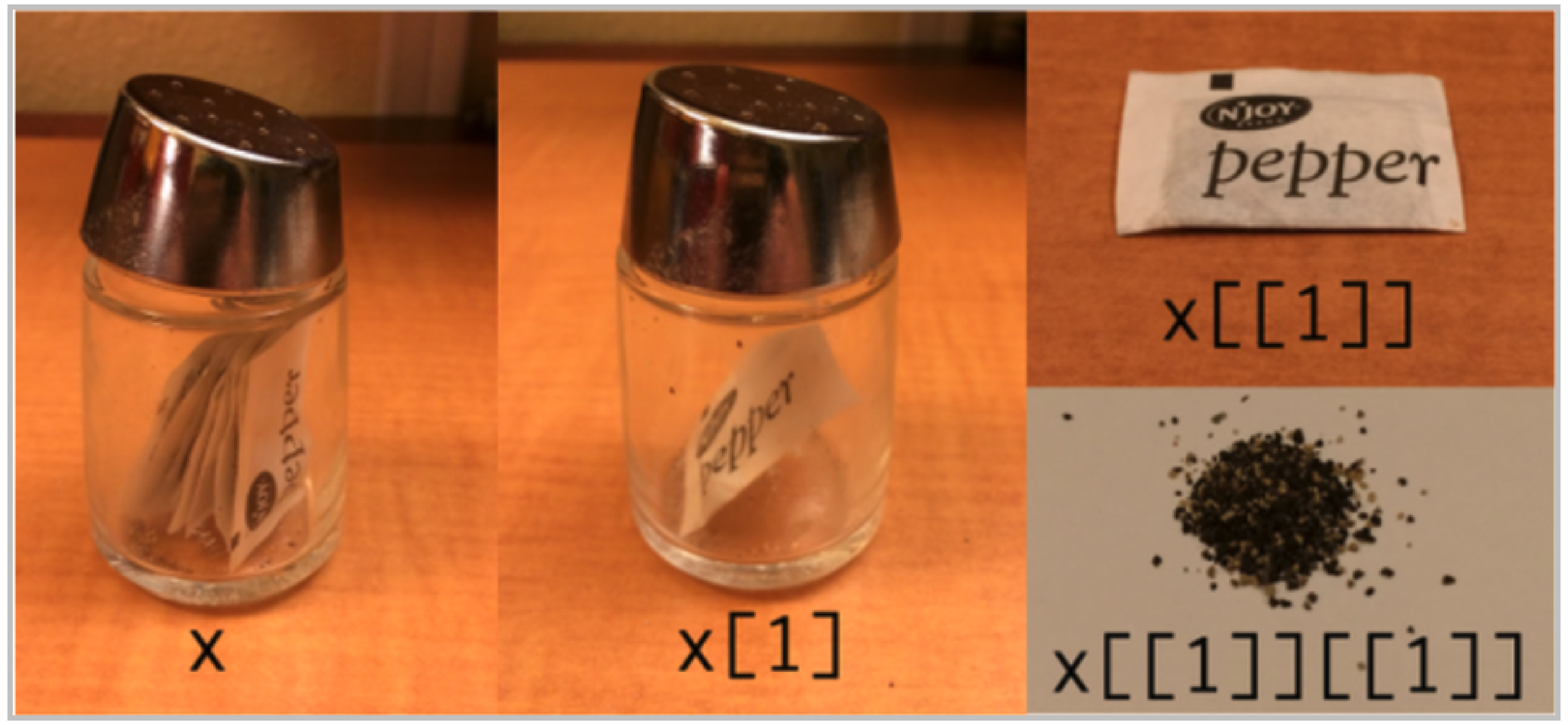
```
[1] "am"
```

```
sample_list$Three
```

```
[1] TRUE
```

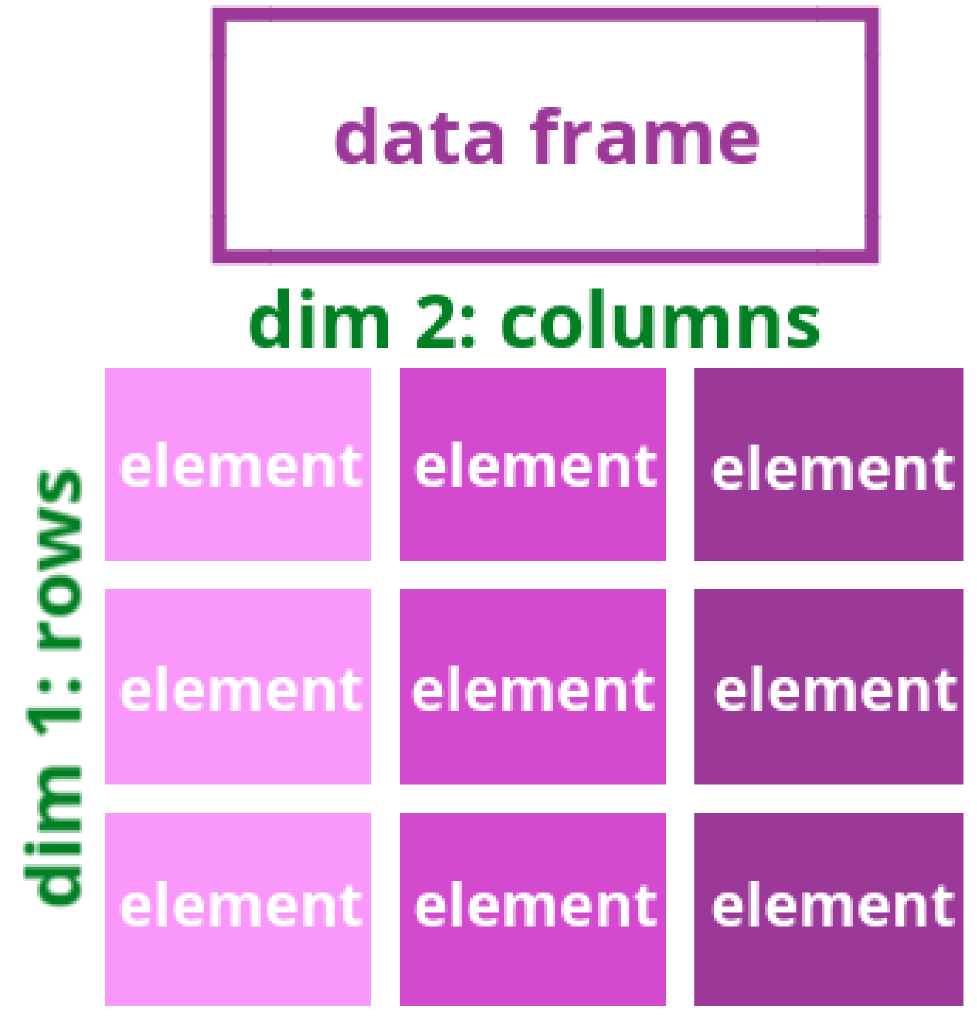
# Basic data structures: single vs. double brackets

- When using an index to access or replace values in a data structure, we use different configurations of **brackets [ ]** for different levels of granularity
- Single brackets [ ] extract a subset of a list
  - Since lists can nest, that subset might actually be a list containing multiple elements
- Double brackets [[ ]] extract a single element from a list



# Basic data structures: dataframes

- A dataframe is a special kind of list arranged in a 2D structure
- Each element in a list is a column with the same number of entries
- Different columns can be of different types (e.g. `character`, `numeric`, `logical`)
- Each entry within a given column should be of the same type
- Dataframes are what we generally think of as the “standard” data structure, like a table in which
    - Columns are vectors
    - Rows are lists



# Basic data structures: making dataframes

- As with vectors, matrices, and lists, a dataframe can be created empty

```
# To make an empty dataframe in R,
# use `data.frame()` command.
data.frame()
```

```
data frame with 0 columns and 0 rows
```

```
# To make a dataframe with several
# columns, pass column values
# to the `data.frame()` command just like
# you would do with lists.
data.frame(1:5, 6:10)
```

```
  X1.5 X6.10
1    1     6
2    2     7
3    3     8
4    4     9
5    5    10
```

- Note that without defined column names, `data.frame` auto-generates them
  - Column names in R cannot have **numbers** as their first character, which is why R adds an `X` to the head
- You can also combine pre-existing vectors into a dataframe

## Dataframes: converting a matrix

- We can cast a `matrix` into a `data.frame` with `as.data.frame()` command

```
# Make a dataframe from matrix.
sample_matrix1 = matrix(nrow = 3, ncol = 3)
sample_matrix1 = 1:9
dim(sample_matrix1) = c(3, 3)

sample_df1 = as.data.frame(sample_matrix1)
sample_df1
```

```
  V1 V2 V3
1  1  4  7
2  2  5  8
3  3  6  9
```

```
# Make a dataframe from matrix with named columns and rows.
sample_df2 = as.data.frame(sample_matrix1, row.names = c('Row1','Row2','Row3'))
cols = c('Col1','Col2','Col3') # defining the column names
colnames(sample_df2) = cols   # assigning the column names to df
sample_df2
```

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

# Dataframes: selecting columns

- Let's explore the different methods we have learned so far for selecting columns from a dataframe
  - Use `$column_name`
  - Use `[[column_index]]`
  - Use `[ , column_index]`

```
# To access a column of a dataframe
# Option 1: Use `$column_name`.
sample_df2$Col1
```

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

```
# To access a column of a dataframe
# Option 2: Use `[[column_index]]`.
sample_df2[[1]]
```

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

```
# To access a column of a dataframe
# Option 3: Use `[ , column_index]`.
sample_df2[, 1]
```

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

# Dataframes: subsetting rows

- And now, let's try selecting a row from a dataframe
  - Use `[row_index, ]`
  - Use `["row_name", ]`

```r
# To access a row of a dataframe
# Option 1: use `[row_index, ]`.
sample_df2[1, ]
```

```
     Col1 Col2 Col3
Row1    1    4    7
```

```r
# To access a row of a dataframe
# Option 2: use `["row_name", ]`.
sample_df2["Row1", ]
```

```
     Col1 Col2 Col3
Row1    1    4    7
```

# Dataframes: accessing individual values

- There are four common methods for accessing individual values within a dataframe, but they use different attributes
- Option 1: `$column_name[row_index]`
- Option 2: `[[column_index]][row_index]`
- Option 3: `[row_index, column_index]`
- Option 4: `["row_name", "column_name"]`

```
# Option 1:
# `data_frame$column_name[row_index]`
sample_df2$Col2[1]
```

```
[1] 4
```

```
# Option 2:
# `data_frame[[column_index]][row_index]`
sample_df2[[2]][1]
```

```
[1] 4
```

```
# Option 3:
# `data_frame[row_index, column_index]`
sample_df2[1, 2]
```

```
[1] 4
```

```
# Option 4:
# `data_frame["row_name", "column_name"]`
sample_df2["Row1", "Col2"]
```

```
[1] 4
```

## Knowledge check

# Exercise

You are now ready to try Tasks 4-12 in the Exercise for this topic

## Module completion checklist

| Objective | Complete |
|---|---|
| Introduce `vector` and `matrices` and their operations | ✓ |
| Introduce `list` and `dataframe` in R | ✓ |

## Data Types and Data Structures: Topic summary

In this part of the course, we have covered:

- Overview of R data types and data structures
- Lists, vectors, matrices, and dataframes

# Congratulations on completing this module!