Basic R

Pwani R Workshop

8th July 2024

In this session, we will cover:

- R syntax
- R functions
- Data types
- Data structures

Explaining R code input and outputs on slides

In slides, a command (also called a code) will look like this

```
sum(2,3)
```

```
## [1] 5
```

And then directly after it, will be the output of the code.
So `sum(2,3)` is the code and '[1] 5' is the output.

R as a calculator

```r
2 + 2
```

```
## [1] 4
```

```r
2 * 4
```

```
## [1] 8
```

```r
2^3
```

```
## [1] 8
```

Note: when you enter your command in the Console, R inherently thinks you want to print the result.

R as a calculator

- The R console is a full calculator
- Try to play around with it:
    - +, -, /, * are add, subtract, divide and multiply
    - ^ or ** is power
    - parentheses – (and) – work with order of operations
    - %% finds the remainder

R as a calculator

```r
2 + (2 * 3)^2
```

```
## [1] 38
```

```r
(1 + 3) / 2 + 45
```

```
## [1] 47
```

```r
6 / 2 * (1 + 2)
```

```
## [1] 9
```

R as a calculator

Try evaluating the following:

- `2 + 2 * 3 / 4 -3`
- `2 * 3 / 4 * 2`
- `2^4 - 1`

Assigning values to objects

You can create objects from within the R environment and from files on your computer.

R uses **<-** to assign values to an object name (you might also see **=** used, but this is not best practice).

<- assigns **values on the right** to **variables on the left**.

```
x <- 2
x
```

```
## [1] 2
```

```
x * 4
```

```
## [1] 8
```

R Data types

Data type: Defines the nature of a single value. Some data types are:

- `"numeric"`: for any numerical value
- `"character"`: for text values., denoted using quotes `("")`
- `"integer"`: for whole numbers
- `"logical"`: that we won’t discuss further
- `"complex"`: that we won’t discuss further

R Data types (examples)

Use the **`class()`** function to check the class of an object.

Numeric

```r
x <- 2
class(x)
```

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

Character

```r
y <- "hello world!"
class(y)
```

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

R Data structures

Data Structure

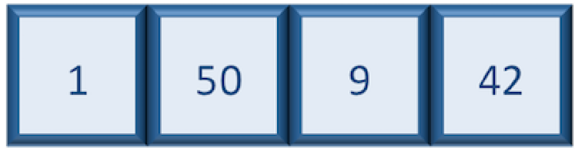
Defines how multiple values are organized and stored. Some are:

- Vector
- Matrix
- Data Frame
- List
- …there are more

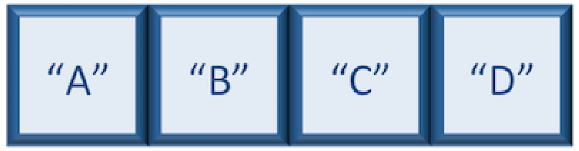
Vectors

- Most common and basic data structure in R
- They are one dimensional
- Can have multiple sets of observations, but must be of the same `class`.

vector of numbers



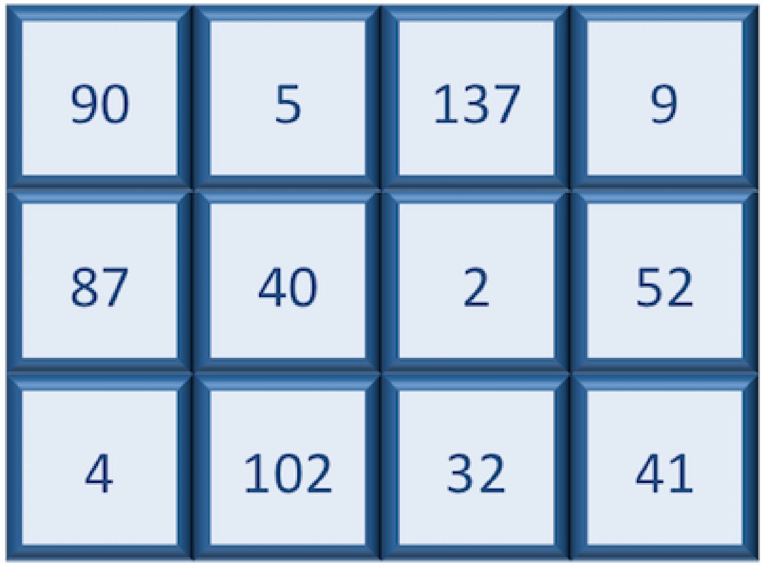
vector of characters



- Each value of a vector is referred to as an `element`.

Matrix

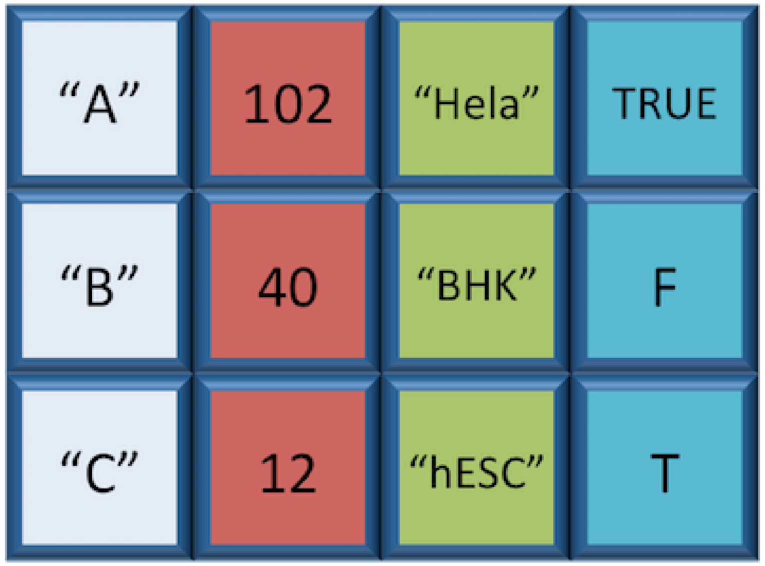
- A collection of vectors of the **same length and identical datatype.**
- Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.



90	5	137	9
87	40	2	52
4	102	32	41

Data Frame

- The *de facto* data structure for most tabular data is the `data.frame`.
- It's like an Excel file with rows (observations) and columns (variables).
- Unlike a `Matrix`, in a `data.frame` each vector (column) can be of a different data type (e.g., characters, integers, factors)



"A"	102	"Hela"	TRUE
"B"	40	"BHK"	F
"C"	12	"hESC"	T

Common Issues

TROUBLESHOOTING: R is case sensitive

Object names are case-sensitive, i.e., `X` and `x` are different

```
x
```

```
## [1] 2
```

```
X
```

```
## Error in eval(expr, envir, enclos): object 'X' not found
```

TROUBLESHOOTING: No commas in big numbers

Commas separate objects in R, so they shouldn’t be used when entering big numbers.

```
z <- 3,000
```

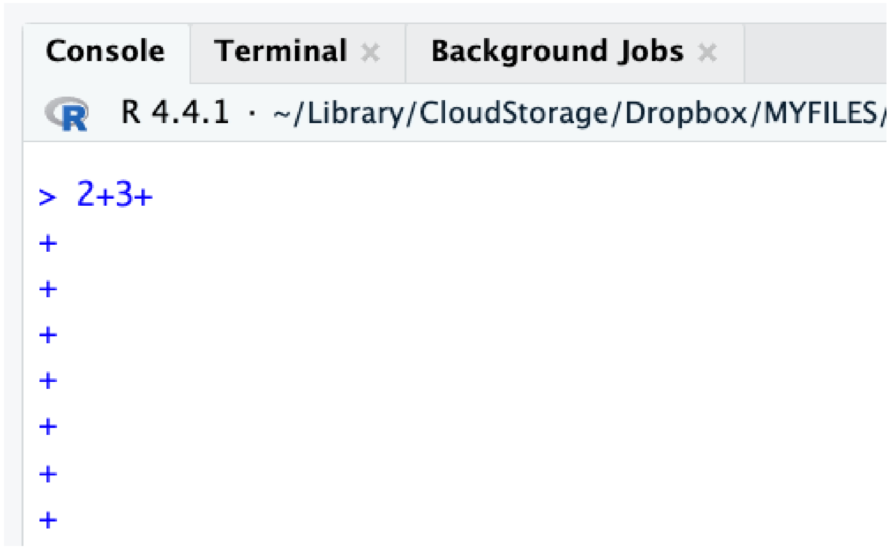
```
## Error: <text>:1:7: unexpected ','
## 1: z <- 3,
##           ^
```

TROUBLESHOOTING: Complete the commads

```r
2+2+
```

```
## Error: <text>:2:0: unexpected end of input
## 1: 2+2+
##    ^
```

+ indicates an incomplete statement. Hit “esc” to clear and bring back the >.



```
> 2+3+
+
+
+
+
+
+
+
```

Simple object practice

Try assigning your full name to an R object called `name`

Simple object practice

Try assigning your full name to an R object called `name`

```r
name <- "Rex Masai"
name
```

```
## [1] "Rex Masai"
```

The ‘combine’ function `c()`

The function `c()` collects/combines/joins single R objects into a vector of R objects. It is mostly used for creating vectors of numbers, character strings, and other data types.

```
x <- c(1, 4, 6, 8)
x
```

```
## [1] 1 4 6 8
```

```
class(x)
```

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

The ‘combine’ function `c()`

Try assigning your first and last name as 2 separate character strings into a single vector called `name2`

The ‘combine’ function `c()`

Try assigning your first and last name as 2 separate character strings into a length-2 vector called `name2`

```
name2 <- c("Rex", "Masai")
name2
```

```
## [1] "Rex"   "Masai"
```

Arguments inside R functions

- The contents you give to an R function are called “arguments”
- Here, R assumes all arguments should be objects contained in the vector
- We will talk more about arguments as we use more complicated functions!

```r
name2 <- c("Rex", "Masai")
# Arg 1    ^^^^^
```

```r
name2 <- c("Rex", "Masai")
# Arg 2           ^^^^^^^^^
```

`length` of R objects

`length()`: Get or set the length of vectors, and of any other R object for which a structure has been defined.

```
length(x)
```

```
## [1] 4
```

```
y
```

```
## [1] "hello world!"
```

```
length(y)
```

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

`length` of R objects

What do you expect for the length of the `name` object? What about the `name2` object?

What are the lengths of each?

`length` of R objects

What do you expect for the length of the `name` object? What about the `name2` object?

What are the lengths of each?

```
length(name)
```

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

```
length(name2)
```

```
## [1] 2
```

Math + vector objects

You can perform functions to entire vectors of numbers very easily.

```
x + 2
```

```
## [1]  3  6  8 10
```

```
x * 3
```

```
## [1]  3 12 18 24
```

```
x + c(1, 2, 3, 4)
```

```
## [1]  2  6  9 12
```

Math + vector objects

But things like algebra can only be performed on numbers.

```
name2 + 4
```

```
## Error in name2 + 4: non-numeric argument to binary operator
```

Reassigning to a new object

Save these modified vectors as a new vector called `y`.

```
y <- x + c(1, 2, 3, 4)
y
```

```
## [1]  2  6  9 12
```

Note that the R object `y` is no longer “hello world!” - It has been overwritten by assigning new data to the same name.

Reassigning to a new object

Reassigning allows you to make changes “in place”

```r
# results not stored:
x + c(1, 2, 3, 4)

# x remains unchanged, results stored in `y`:
y <- x + c(1, 2, 3, 4)

# replace `x` in place
x <- x + c(1, 2, 3, 4)
```

R objects

You can get more attributes than just class. The function `str()` gives you the structure of the object.

```
str(x)
```

```
##  num [1:4] 1 4 6 8
```

```
str(y)
```

```
##  num [1:4] 2 6 9 12
```

This tells you that `x` is a numeric vector and tells you the length.

Useful functions to create vectors **`seq()`**

For numeric: `seq()` can be very useful.
The `from` argument says what number to start on.
The `to` argument says what number to not go above.
The `by` argument says how much to increment by.

```
seq(from = 0, to = 1, by = 0.2)
```

```
## [1] 0.0 0.2 0.4 0.6 0.8 1.0
```

```
seq(from = 0, to = 10, by = 1)
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10
```

Useful functions to create vectors `rep()`

For character: `rep()` can create very long vectors. Works for creating character and numeric vectors.

The `each` argument specifies how many of each item you want repeated. The `times` argument specifies how many times you want the vector repeated.

`rep(WHAT_TO_REPEAT, arguments)`

```r
rep(c("black", "white"), each = 3)
```

```
## [1] "black" "black" "black" "white" "white" "white"
```

Useful functions to create vectors `rep()`

For character: `rep()` can create very long vectors. Works for creating character and numeric vectors.

The `each` argument specifies how many of each item you want repeated. The `times` argument specifies how many times you want the vector repeated.

`rep(WHAT_TO_REPEAT, arguments)`

```
rep(c("black", "white"), times = 3)
```

```
## [1] "black" "white" "black" "white" "black" "white"
```

Useful functions to create vectors **`rep()`**

For character: `rep()` can create very long vectors. Works for creating character and numeric vectors.

The `each` argument specifies how many of each item you want repeated. The `times` argument specifies how many times you want the vector repeated.

```
rep(WHAT_TO_REPEAT, arguments)
```

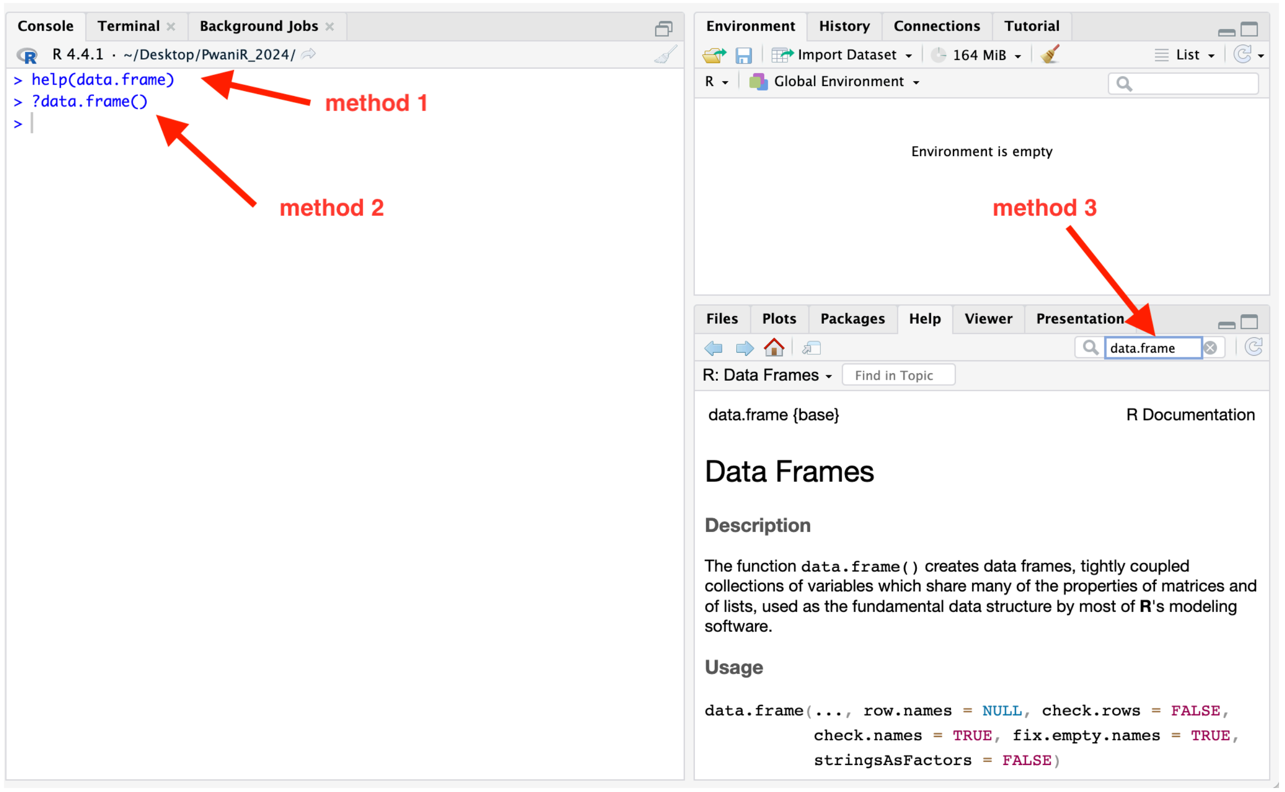
```r
rep(c("black", "white"), each = 2, times = 2)
```

```
## [1] "black" "black" "white" "white" "black" "black" "white" "white"
```

Getting help in R

`help()` and `?`

The `help()` function and `?` help operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.



```
> help(data.frame)
> ?data.frame()
>
```

data.frame {base} R Documentation

Data Frames

Description

The function `data.frame()` creates data frames, tightly coupled collections of variables which share many of the properties of matrices and of lists, used as the fundamental data structure by most of **R**'s modeling software.

Usage

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE, fix.empty.names = TRUE,
           stringsAsFactors = FALSE)
```

Summary

- R functions as a calculator
- Use `<-` to save (assign) values to objects
- Use `c()` to **combine** vectors
- `length()`, `class()`, and `str()` tell you information about an object
- The sequence `seq()` function helps you create numeric vectors (`from`,`to`, `by`, and `length.out` arguments)
- The repeat `rep()` function helps you create vectors with the `each` and `times` arguments