

# Data Classes

# One dimensional vectors

## Character and numeric

We have already covered character and numeric types.

```
class(c("tree", "cloud", "stars_&_sky"))
## [1] "character"

class(c(1, 4, 7))
## [1] "numeric"
```

## Character and numeric

Character predominates if there are mixed classes.

```
class(c(1, 2, "tree"))
```

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

```
class(c("1", "4", "7"))
```

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

## Logical

`logical` is a type that only has two possible elements: `TRUE` and `FALSE`

```
x <- c(TRUE, FALSE, TRUE, TRUE, FALSE)  
class(x)
```

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

`logical` elements are NOT in quotes.

## Why is Class important?

The class of the data tells R how to process the data.

For example, it determines whether you can make summary statistics (numbers) or if you can sort alphabetically (characters).

## General Class Information

There is one useful functions associated with practically all R classes:

**as.CLASS\_NAME(x)** coerces between classes. It turns x into a certain class.

Examples:

- `as.numeric()`
- `as.character()`
- `as.logical()`

## Coercing: seamless transition

Sometimes coercing works great!

```
as.character(4)  
## [1] "4"  
  
as.numeric(c("1", "4", "7"))  
## [1] 1 4 7  
  
as.logical(c("TRUE", "FALSE", "FALSE"))  
## [1] TRUE FALSE FALSE  
  
as.logical(0)  
## [1] FALSE
```

## Coercing: not-so-seamless

When interpretation is ambiguous, R will return NA (an R constant representing “Not Available” i.e. missing value)

```
as.numeric(c("1", "4", "7a"))

## Warning: NAs introduced by coercion

## [1] 1 4 NA

as.logical(c("TRUE", "FALSE", "UNKNOWN"))

## [1] TRUE FALSE     NA
```

## GUT CHECK!

What is one reason we might want to convert data to numeric?

- A. So we can take the mean
- B. So the data looks better
- C. So our data is correct

## Number Subclasses

There are two major number subclasses or types

1. Double (1.003)
2. Integer (1)

## Number Subclasses

`Double` is equivalent to `numeric`. It is a number that contains **fractional values** .  
Can be any amount of places after the decimal.

`Double` stands for double-precision

For most purposes, the difference between integers and doubles doesn't matter.

## Significant figures and other formats

The `num` function of the `tibble` package can be used to change format. See here for more: <https://tibble.tidyverse.org/articles/numbers.html>

## Factors

A **factor** is a special character vector where the elements have pre-defined groups or 'levels'. You can think of these as qualitative or categorical variables. Order is often important.

Examples:

- red, orange, yellow, green, blue, purple
- breakfast, lunch, dinner
- baby, toddler, child, teen, adult
- Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree
- beginner, novice, intermediate, expert

\*\* We will learn more about factors in a later module. \*\*

# Classes Overview

| Example          | Class     | Type      | Notes   |
|------------------|-----------|-----------|---|
| 1.1              | Numeric   | double    | default for numbers   |
| 1                | integer   | integer   | Need to coerce to integer with as.integer() or use sample() or seq() with whole numbers |
| “FALSE”, “Ball”  | Character | Character | Need quotes   |
| FALSE, TRUE      | logical   | logical   | No quotes   |
| “Small”, “Large” | Factor    | Factor    | Need to coerce to factor with factor()  |

# Special data classes

## Dates

There are two most popular R classes used when working with dates and times:

- Date class representing a calendar date
- POSIXct class representing a calendar date with hours, minutes, seconds

We convert data from character to Date/POSIXct to use functions to manipulate date/date and time

`lubridate` is a powerful, widely used R package from “tidyverse” family to work with Date / POSIXct class objects

## Creating **Date** class object

```
class("2021-06-15")
## [1] "character"

library(lubridate)

x <- ymd("2021-06-15") # lubridate package Year Month Day
class(x)

## [1] "Date"
```

Note for function **ymd**: **year month day**

## Dates are useful!

```
a <- ymd("2021-06-15")
b <- ymd("2021-06-18")
a - b

## Time difference of -3 days
```

## The function matches the format

```
mdy("06/15/2021")
## [1] "2021-06-15"

dmy("15-June-2021")
## [1] "2021-06-15"

ymd("2021-06-15")
## [1] "2021-06-15"
```

## Class conversion in a dataset

Here's a dataset on the SARS-CoV-2 viral load measured in wastewater between 2022 and 2024, collected by the National Wastewater Surveillance System.

Let's look at the *date\_start* variable, the first date of the sampling window.

```
sars_ww <-  
  read_csv("https://daseh.org/data/SARS-CoV-2_Wastewater_Data.csv")  
  
# Selecting a few columns for easy viewing  
sars_ww <- sars_ww |> select(town_name, date_start)
```

## Class conversion in a dataset

Notice that date\_start is chr class, not date.

sars\_ww

```
## # A tibble: 2,813 × 2
##   town_name date_start
##   <chr>     <chr>
## 1 Barry     6/21/2020
## 2 Barry     6/22/2020
## 3 Barry     6/23/2020
## 4 Barry     6/24/2020
## 5 Barry     6/25/2020
## 6 Barry     6/26/2020
## 7 Barry     6/27/2020
## 8 Barry     6/28/2020
## 9 Barry     6/29/2020
## 10 Barry    6/30/2020
## # ... 2,803 more rows
```

## Class conversion in with a dataset

We would need to use `mutate()` to help us modify that column.

```
sars_ww |>
  mutate(date_start_fixed = mdy(date_start))

## # A tibble: 2,813 × 3
##   town_name date_start date_start_fixed
##   <chr>     <chr>      <date>
## 1 Barry     6/21/2020 2020-06-21
## 2 Barry     6/22/2020 2020-06-22
## 3 Barry     6/23/2020 2020-06-23
## 4 Barry     6/24/2020 2020-06-24
## 5 Barry     6/25/2020 2020-06-25
## 6 Barry     6/26/2020 2020-06-26
## 7 Barry     6/27/2020 2020-06-27
## 8 Barry     6/28/2020 2020-06-28
## 9 Barry     6/29/2020 2020-06-29
## 10 Barry    6/30/2020 2020-06-30
## # ... 2,803 more rows
```

# Other data classes

## Two-dimensional data classes

Two-dimensional classes are those we would often use to store data read from a file

- a data frame (`data.frame` or `tibble` class)
- a matrix (`matrix` class)
  - also composed of rows and columns
  - unlike `data.frame` or `tibble`, the entire matrix is composed of one R class
  - for example: all entries are `numeric`, or all entries are `character`

## Lists

- One other data type that is the most generic are **lists**.
- Can hold vectors, strings, matrices, models, list of other list!
- Lists are used when you need to do something repeatedly across lots of data - for example wrangling several similar files at once
- Lists are a bit more advanced but you may encounter them when you work with others or look up solutions

## Making Lists

- Can be created using `list()`

```
mylist <- list(c("A", "b", "c"), c(1, 2, 3))  
mylist
```

```
## [[1]]  
## [1] "A" "b" "c"  
##  
## [[2]]  
## [1] 1 2 3
```

```
class(mylist)
```

```
## [1] "list"
```

## Summary

- coerce between classes using `as.numeric()` or `as.character()`
- data frames, tibbles, matrices, and lists are all classes of objects
- lists can contain multiples of any other class of data including lists!
- calendar dates can be represented with the `Date` class using `ymd()`, `mdy()` functions from [lubridate package](#)
- can then easily subtract `Date` or `POSIXct` class variables or pull out aspects like year

# Lab

- [Class Website](#)
- [Lab](#)
- [Day 4 Cheatsheet](#)

For more advanced learning: see the extra slides in this file!



Image by [Gerd Altmann from Pixabay](#)

# Extra Slides

## Matrices

`as.matrix()` creates a matrix from a data frame or tibble (where all values are the same class).

`matrix()` creates a matrix from scratch.

```
matrix(1:6, ncol = 2)
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

## More about Lists

List elements can be named

```
mylist_named <- list(  
  letters = c("A", "b", "c"),  
  numbers = c(1, 2, 3),  
  one_matrix = matrix(1:4, ncol = 2)  
)  
mylist_named  
  
## $letters  
## [1] "A" "b" "c"  
##  
## $numbers  
## [1] 1 2 3  
##  
## $one_matrix  
##      [,1] [,2]  
## [1,]    1    3  
## [2,]    2    4
```

## Some useful functions from **lubridate** to manipulate **Date** objects

```
x <- ymd(c("2021-06-15", "2021-07-15"))
x

## [1] "2021-06-15" "2021-07-15"

day(x) # see also: month(x) , year(x)

## [1] 15 15

x + days(10)

## [1] "2021-06-25" "2021-07-25"

x + months(1) + days(10)

## [1] "2021-07-25" "2021-08-25"

wday(x, label = TRUE)

## [1] Tue Thu
## Levels: Sun < Mon < Tue < Wed < Thu < Fri < Sat
```

## Some useful functions from **lubridate** to manipulate **POSIXct** objects

```
x <- ymd_hms("2013-01-24 19:39:07")
x

## [1] "2013-01-24 19:39:07 UTC"

date(x)

## [1] "2013-01-24"

x + hours(3)

## [1] "2013-01-24 22:39:07 UTC"

floor_date(x, "1 hour") # see also: ceiling_date()

## [1] "2013-01-24 19:00:00 UTC"
```

## Differences in dates

```
x1 <- ymd(c("2021-06-15"))
x2 <- ymd(c("2021-07-15"))

difftime(x2, x1, units = "weeks")

## Time difference of 4.285714 weeks

as.numeric(difftime(x2, x1, units = "weeks"))

## [1] 4.285714
```

Similar can be done with time (e.g. difference in hours).

# Data Selection

# Matrices

```
n <- 1:9  
n  
  
## [1] 1 2 3 4 5 6 7 8 9  
  
mat <- matrix(n, nrow = 3)  
mat  
  
## [,1] [,2] [,3]  
## [1,] 1 4 7  
## [2,] 2 5 8  
## [3,] 3 6 9
```

## Vectors: data selection

To get element(s) of a vector (one-dimensional object):

- Type the name of the variable and open the rectangular brackets [ ]
- In the rectangular brackets, type index (/vector of indexes) of element (/elements) you want to pull. **In R, indexes start from 1** (not: 0)

```
x <- c("a", "b", "c", "d", "e", "f", "g", "h")  
x
```

```
## [1] "a" "b" "c" "d" "e" "f" "g" "h"
```

```
x[2]
```

```
## [1] "b"
```

```
x[c(1, 2, 100)]
```

```
## [1] "a" "b" NA
```

## Matrices: data selection

Note you cannot use `dplyr` functions (like `select`) on matrices. To subset matrix rows and/or columns, use `matrix[row_index, column_index]`.

```
mat
```

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

```
mat[1, 1] # individual entry: row 1, column 1
```

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

```
mat[1, 2] # individual entry: row 1, column 2
```

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

```
mat[1, ] # first row
```

```
## [1] 1 4 7
```

```
mat[, 1] # first column
```

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

```
mat[c(1, 2), c(2, 3)] # subset of original matrix: two rows and two columns
```

## Lists: data selection

You can reference data from list using \$ (if elements are named) or using [ [ ] ]

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

```
## [1] "A" "b" "c"
```

```
mylist_named[["letters"]] # works only for a list with elements' names
```

```
## [1] "A" "b" "c"
```

```
mylist_named$letters # works only for a list with elements' names
```

```
## [1] "A" "b" "c"
```