Recap before the exam

From objects to dataframes

Objects

Creating and naming objects

`name <- value`: `<-` creates objects

```
age <- 25
```

```
age <- age + 1          # overwrite
```

```
age_months <- age * 12  # use it to create a new object
```

```
age_months
```

```
[1] 312
```

- names **start with a letter**, no spaces: `my_data`, `x1` (not `1x`, `my data`)
- R is **case-sensitive**: `Name` and `name` are two different objects
- `rm(x)` removes one object, `rm(list = ls())` removes all

Data types

```r
class(3.14)      # numeric
```

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

```r
class(5L)        # integer
```

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

```r
class("hello")   # character: always in quotes
```

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

```r
class(TRUE)      # logical
```

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

```r
class("5")       # in quotes: character
```

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

Operators

Mathematical operators and functions

operator	meaning	example	result
`+ - * /`	arithmetic	`6 * 2 / 4`	`3`
`^`	power	`2 ^ 3`	`8`
`%%`	remainder of the division	`9 %% 2`	`1`

```
(1 + 2) * 3                    # parentheses first
```

```
[1] 9
```

```
sqrt(16); abs(-7); round(3.14159, 2)
```

```
[1] 4
[1] 7
[1] 3.14
```

Relational and logical operators

```
x <- 30
```

```
x == 30; x != 30; x >= 18      # compare: TRUE or FALSE
```

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

```
x %in% c(10, 20, 30)           # is x one of these values?
```

```
[1] TRUE
```

```
x > 25 & x < 60                # AND: both TRUE
```

```
[1] TRUE
```

```
x < 18 | x > 60                # OR: at least one TRUE
```

```
[1] FALSE
```

```
!(x > 25)                      # NOT
```

```
[1] FALSE
```

TRUE and FALSE are numbers

`TRUE` counts as **1**, `FALSE` as **0**: `sum()` counts, `mean()` gives the proportion

```
1 scores <- c(18, 25, 30, 22, 27)
```

```
1 scores >= 25
```

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

```
1 sum(scores >= 25)       # how many?
```

```
[1] 3
```

```
1 mean(scores >= 25)      # which proportion?
```

```
[1] 0.6
```

Vectors

Creating vectors

One dimension, **all elements of the same type**

```r
num1 <- c(4, 8, 15); num1
```

```
[1]  4  8 15
```

```r
num2 <- 1:5; num2
```

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

```r
num3 <- seq(from = 0, to = 10, by = 2); num3
```

```
[1]  0  2  4  6  8 10
```

```r
ch1 <- rep(c("x", "y"), times = 2); ch1   # whole vector
```

```
[1] "x" "y" "x" "y"
```

```r
ch2 <- rep(c("x", "y"), each = 2); ch2    # each element
```

```
[1] "x" "x" "y" "y"
```

```r
ch3 <- rep(c("x", "y"), each = 2, times = 2); ch3
```

```
[1] "x" "x" "y" "y" "x" "x" "y" "y"
```

Types: `is.` and `as.`

Mixing types converts everything to **one type**:

```r
c(1, 2, "three")                    # all character
```

```
[1] "1"     "2"     "three"
```

```r
c(1, 2, NA, 4)                      # NA keeps the type
```

```
[1]  1  2 NA  4
```

`is.` tests the type, `as.` converts it:

```r
is.numeric(c(1, 0, 3))
```

```
[1] TRUE
```

```r
as.logical(c(1, 0, 3))              # 0 is FALSE, others TRUE
```

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

```r
as.numeric(c("1", "2", "three"))    # cannot convert: NA
```

```
Warning: NAs introduced by coercion
```

```
[1]  1  2 NA
```

Operations and missing values

Operations work **element by element**, summary functions return **one value**

```
temps <- c(12, 18, NA, 31, 7)
```

```
temps * 2
```

```
[1] 24 36 NA 62 14
```

```
mean(temps)                    # one NA makes the result NA
```

```
[1] NA
```

```
mean(temps, na.rm = TRUE)      # remove the NAs first
```

```
[1] 17
```

```
is.na(temps)
```

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

```
sum(is.na(temps))              # how many NAs?
```

```
[1] 1
```

Indexing by position

```r
temps <- c(12, 18, 25, 31, 7)
```

```r
temps[2]                    # the second element
```

```
[1] 18
```

```r
temps[c(1, 3)]              # the first and the third
```

```
[1] 12 25
```

```r
temps[2:4]                  # from the second to the fourth
```

```
[1] 18 25 31
```

```r
temps[length(temps)]        # the last one
```

```
[1] 7
```

```r
temps[-1]                   # all except the first
```

```
[1] 18 25 31  7
```

Indexing: logical conditions and `which()`

```
1 temps
```

```
[1] 12 18 25 31  7
```

```
1 temps > 20                          # the condition
```

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

```
1 temps[temps > 20]                   # keep where TRUE
```

```
[1] 25 31
```

```
1 temps[temps > 10 & temps < 30]
```

```
[1] 12 18 25
```

```
1 which(temps > 20)                   # the positions
```

```
[1] 3 4
```

Modifying with indexing

Indexing + `<-` **modifies** only the selected elements:

```r
scores <- c(18, NA, 30, 22, NA)
```

```r
scores[1] <- 20                    # by position
```

```r
scores[is.na(scores)] <- 0         # by condition
```

```r
scores[scores < 21] <- 21
```

```r
scores
```

```
[1] 21 21 30 22 21
```

Factors

Factors: codes + levels

```r
group <- factor(x = c("placebo", "caffeine", "nicotine",
                      "placebo", "caffeine"))
```

```r
group
```

```
[1] placebo  caffeine nicotine placebo  caffeine
Levels: caffeine nicotine placebo
```

```r
levels(group)
```

```
[1] "caffeine" "nicotine" "placebo"
```

```r
as.integer(group)  # caffeine=1, nicotine=2, placebo=3
```

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

```r
table(group)
```

```
group
caffeine nicotine  placebo
       2        1        2
```

Levels stay, even when unused

```
1 no_placebo <- group[group != "placebo"]
2 no_placebo # "placebo" is still a level
```

```
[1] caffeine nicotine caffeine
Levels: caffeine nicotine placebo
```

```
1 no_placebo <- droplevels(no_placebo)
```

```
1 levels(no_placebo)
```

```
[1] "caffeine" "nicotine"
```

Adding a new level

To use a **new** value, first add it to the levels

```
1 group
```

```
[1] placebo  caffeine nicotine placebo  caffeine
Levels: caffeine nicotine placebo
```

```
1 levels(group) <- c(levels(group), "water")
```

```
1 group[1] <- "water"
2 group
```

```
[1] water    caffeine nicotine placebo  caffeine
Levels: caffeine nicotine placebo water
```

Reorder vs rename

```r
group <- factor(x = c("placebo", "caffeine", "nicotine",
                      "placebo", "caffeine"))
```

```r
# REORDER: placebo first (reference category), same data
group_ord <- factor( x = c("placebo", "caffeine", "nicotine",
                      "placebo", "caffeine"),
  levels = c("placebo", "caffeine", "nicotine"))
```

```r
levels(group_ord)
```

```
[1] "placebo"  "caffeine" "nicotine"
```

```r
# RENAME: by position, check levels() first!
levels(group_ord) <- c("PLA", "CAF", "NIC")
```

```r
group_ord
```

```
[1] PLA CAF NIC PLA CAF
Levels: PLA CAF NIC
```

Reorder and rename in one step

`factor()` with both arguments:

- **`levels`**: the existing values, in the order we want
- **`labels`**: their new names, in the same order

```
group_new <- factor(x = c("placebo", "caffeine", "nicotine",
                          "placebo", "caffeine"),
  levels = c("placebo", "caffeine", "nicotine"),
  labels = c("PLA", "CAF", "NIC"))
```

```
group_new
```

```
[1] PLA CAF NIC PLA CAF
Levels: PLA CAF NIC
```

```
levels(group_new)
```

```
[1] "PLA" "CAF" "NIC"
```

Ordered factors and numbers

For **ordinal** categories (`<` and `>` make sense), set the order **and** `ordered = TRUE`

```
1 freq <- factor(x = c("often", "never", "sometimes"),
2   levels = c("never", "sometimes", "often"), ordered = TRUE)
```

```
1 freq > "never"
```

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

To get back numbers saved as a factor: first to text, then to numbers

```
1 points_f <- factor(c(10, 30, 10, 20))
```

```
1 as.numeric(as.character(points_f))
```

```
[1] 10 30 10 20
```

Lists

Lists: different types and sizes

```r
participant <- list(name = "Anna",
                    rts = c(450, 520, 380),
                    passed = TRUE,
                    other = list(mom_age = c(27, 34, 40, 18),
                                 degree = c("bachelor")
                    ))
```

```r
str(participant)
```

```
List of 4
 $ name  : chr "Anna"
 $ rts   : num [1:3] 450 520 380
 $ passed: logi TRUE
 $ other :List of 2
  ..$ mom_age: num [1:4] 27 34 40 18
  ..$ degree : chr "bachelor"
```

Looking at a list

```
1 length(participant)
```

```
[1] 4
```

```
1 names(participant)
```

```
[1] "name"   "rts"    "passed" "other"
```

[ ], [[ ]] and $

```r
participant[2]          # [ ]: a smaller LIST
```

```
$rts
[1] 450 520 380
```

```r
participant[[2]]        # [[ ]]: the element itself
```

```
[1] 450 520 380
```

```r
participant$rts         # $: the element, by name
```

```
[1] 450 520 380
```

```r
participant$rts[3]      # then index inside it
```

```
[1] 380
```

```r
class(participant[2]); class(participant[[2]])
```

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

Modifying lists

```r
participant$age <- 24                # add an element
```

```r
participant$passed <- FALSE          # change an element
```

```r
participant$rts[1] <- 455            # change inside an element
```

```r
str(participant)
```

```
List of 5
 $ name  : chr "Anna"
 $ rts   : num [1:3] 455 520 380
 $ passed: logi FALSE
 $ other :List of 2
  ..$ mom_age: num [1:4] 27 34 40 18
  ..$ degree : chr "bachelor"
 $ age   : num 24
```

Matrices and arrays

Matrices: two dimensions, one type

```r
m <- matrix(data = 1:6, nrow = 2, ncol = 3)   # by column
```

```r
m
```

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

```r
# by row
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
```

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

```r
dim(m); nrow(m); ncol(m)
```

```
[1] 2 3
```

```
[1] 2
```

```
[1] 3
```

Indexing matrices: `[row, column]`

```
m[2, 3]           # one element
```

```
[1] 6
```

```
m[1, ]            # the whole first row
```

```
[1] 1 3 5
```

```
m[, 2]            # the whole second column
```

```
[1] 3 4
```

```
m[m > 3]          # logical condition
```

```
[1] 4 5 6
```

```
m[m > 3] <- 0     # modify
```

```
m
```

```
     [,1] [,2] [,3]
[1,]    1    3    0
[2,]    2    0    0
```

Joining matrices and vectors

```r
a <- 1:3
```

```r
b <- c(10, 20, 30)
```

```r
cbind(a, b)   # as columns
```

```
     a  b
[1,] 1 10
[2,] 2 20
[3,] 3 30
```

```r
rbind(a, b)   # as rows
```

```
  [,1] [,2] [,3]
a    1    2    3
b   10   20   30
```

Still **one type only**: adding text turns everything into text

```r
cbind(a, c("x", "y", "z"))
```

```
     a
[1,] "1" "x"
[2,] "2" "y"
[3,] "3" "z"
```

Dataframes

Dataframes: columns of different types

Each column is a **vector** (all the same length): a list with a rectangular shape

```r
df <- data.frame(id = c("P1", "P2", "P3", "P4", "P5"),
                 group = c("ctrl", "trt", "ctrl", "trt", "trt"),
                 age = c(22, 27, 31, NA, 24),
                 score = c(18, 25, 30, 22, 27))
```

```r
df
```

```
  id group age score
1 P1  ctrl  22    18
2 P2   trt  27    25
3 P3  ctrl  31    30
4 P4   trt  NA    22
5 P5   trt  24    27
```

Looking at a dataframe

```r
str(df)
```

```
'data.frame':   5 obs. of  4 variables:
 $ id   : chr  "P1" "P2" "P3" "P4" ...
 $ group: chr  "ctrl" "trt" "ctrl" "trt" ...
 $ age  : num  22 27 31 NA 24
 $ score: num  18 25 30 22 27
```

```r
dim(df); names(df)
```

```
[1] 5 4
```

```
[1] "id"    "group" "age"   "score"
```

```r
head(df, n = 2)
```

```
  id group age score
1 P1  ctrl  22    18
2 P2   trt  27    25
```

`summary(df)` and `table(df$group)` are also useful checks

Extracting a column as a vector

```r
df$age                  # a column: a vector
```

```
[1] 22 27 31 NA 24
```

```r
df[["age"]]             # the same
```

```
[1] 22 27 31 NA 24
```

Extracting a column as a dataframe

```
1 df["age"]              # a dataframe with one column
```

```
  age
1  22
2  27
3  31
4  NA
5  24
```

```
1 df[3]                  # the same, by position
```

```
  age
1  22
2  27
3  31
4  NA
5  24
```

Extracting rows

`df[rows, columns]`: leave a position **empty** to keep everything

```
1 df
```

```
  id group age score
1 P1  ctrl  22    18
2 P2   trt  27    25
3 P3  ctrl  31    30
4 P4   trt  NA    22
5 P5   trt  24    27
```

```
1 df[2, ]              # the second row
```

```
  id group age score
2 P2   trt  27    25
```

Extracting cells

```r
df[2, "score"]        # one cell: row 2, column score
```

```
[1] 25
```

```r
df[1:2, c("id", "score")]
```

```
  id score
1 P1    18
2 P2    25
```

Selecting rows with conditions

`df[rows, columns]`: the condition goes in the **rows** position

```
1 df
```

```
  id group age score
1 P1  ctrl  22    18
2 P2   trt  27    25
3 P3  ctrl  31    30
4 P4   trt  NA    22
5 P5   trt  24    27
```

```
1 df[df$score > 20, ]
```

```
  id group age score
2 P2   trt  27    25
3 P3  ctrl  31    30
4 P4   trt  NA    22
5 P5   trt  24    27
```

Combining conditions

```r
df[df$group == "trt" & df$score > 22, c("id", "score")]
```

```
  id score
2 P2    25
5 P5    27
```

Selecting rows with `subset()`

Same result, without repeating `df$`:

```
1 subset(df, subset = group == "trt" & score > 22,
2         select = c(id, score))
```

```
  id score
2 P2    25
5 P5    27
```

Creating a new column

`df$new_column <- value`: one value for each row

```
1 df$passed <- df$score >= 25
2 df
```

```
  id group age score passed
1 P1  ctrl  22    18  FALSE
2 P2   trt  27    25   TRUE
3 P3  ctrl  31    30   TRUE
4 P4   trt  NA    22  FALSE
5 P5   trt  24    27   TRUE
```

A new column from another column

```r
df$score_100 <- df$score / 30 * 100
df
```

```
  id group age score passed score_100
1 P1  ctrl  22    18  FALSE  60.00000
2 P2   trt  27    25   TRUE  83.33333
3 P3  ctrl  31    30   TRUE 100.00000
4 P4   trt  NA    22  FALSE  73.33333
5 P5   trt  24    27   TRUE  90.00000
```

Creating a column with `ifelse()`

A new column that depends on a **condition**

```r
df$level <- ifelse(df$score >= 25, "high", "low")
df
```

```
  id group age score passed score_100 level
1 P1  ctrl  22    18  FALSE  60.00000   low
2 P2   trt  27    25   TRUE  83.33333  high
3 P3  ctrl  31    30   TRUE 100.00000  high
4 P4   trt  NA    22  FALSE  73.33333   low
5 P5   trt  24    27   TRUE  90.00000  high
```

Changing the type of a column

```r
df$group <- as.factor(df$group)
str(df)
```

```
'data.frame':   5 obs. of  7 variables:
 $ id       : chr  "P1" "P2" "P3" "P4" ...
 $ group    : Factor w/ 2 levels "ctrl","trt": 1 2 1 2 2
 $ age      : num  22 27 31 NA 24
 $ score    : num  18 25 30 22 27
 $ passed   : logi  FALSE TRUE TRUE FALSE TRUE
 $ score_100: num  60 83.3 100 73.3 90
 $ level    : chr  "low" "high" "high" "low" ...
```

Changing one cell

Select the cell with a condition, then assign the new value

```
df$age[df$id == "P4"] <- 29
df
```

```
  id group age score passed score_100 level
1 P1  ctrl  22    18  FALSE  60.00000   low
2 P2   trt  27    25   TRUE  83.33333  high
3 P3  ctrl  31    30   TRUE 100.00000  high
4 P4   trt  29    22  FALSE  73.33333   low
5 P5   trt  24    27   TRUE  90.00000  high
```

Removing a column

Assign `NULL` to the column

```
df$score_100 <- NULL
df
```

```
  id group age score passed level
1 P1  ctrl  22    18  FALSE   low
2 P2   trt  27    25   TRUE  high
3 P3  ctrl  31    30   TRUE  high
4 P4   trt  29    22  FALSE   low
5 P5   trt  24    27   TRUE  high
```

Counting

```r
table(df$group)                          # frequencies
```

```
ctrl  trt
   2    3
```

```r
table(df$group, df$level)                # contingency table
```

```
      high low
  ctrl    1   1
  trt     2   1
```

Sorting rows: `order()`

`order()` gives the **positions** of the rows, from the lowest to the highest value

```
1 df$score
```

```
[1] 18 25 30 22 27
```

```
1 order(df$score)
```

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

We use these positions to select the rows: the rows are now **sorted** by score

```
1 df_sorted <- df[order(df$score), ]
2 df_sorted
```

```
  id group age score passed level
1 P1  ctrl  22    18  FALSE   low
4 P4   trt  29    22  FALSE   low
2 P2   trt  27    25   TRUE  high
5 P5   trt  24    27   TRUE  high
3 P3  ctrl  31    30   TRUE  high
```

Sorting from the highest

`decreasing = TRUE`

```r
df_sorted <- df[order(df$score, decreasing = TRUE), ]
df_sorted
```

```
  id group age score passed level
3 P3  ctrl  31    30   TRUE  high
5 P5   trt  24    27   TRUE  high
2 P2   trt  27    25   TRUE  high
4 P4   trt  29    22  FALSE   low
1 P1  ctrl  22    18  FALSE   low
```

Missing values in a dataframe

```r
df$age[2] <- NA
```

```r
df$age
```

```
[1] 22 NA 31 29 24
```

```r
sum(is.na(df$age))                    # how many NAs?
```

```
[1] 1
```

```r
mean(df$age, na.rm = TRUE)
```

```
[1] 26.5
```

```r
df[complete.cases(df), "id"]          # rows without any NA
```

```
[1] "P1" "P3" "P4" "P5"
```

Adding rows: `rbind()`

The new rows need the same number of columns and the **same column names**

```
1 new <- data.frame(id = "P6", group = "ctrl", age = 25,
2                   score = 21, passed = FALSE, level = "low")
3 new
```

```
  id group age score passed level
1 P6  ctrl  25    21  FALSE   low
```

Adding rows: the result

```r
df2 <- rbind(df, new)
df2
```

```
  id group age score passed level
1 P1  ctrl  22    18  FALSE   low
2 P2   trt  NA    25   TRUE  high
3 P3  ctrl  31    30   TRUE  high
4 P4   trt  29    22  FALSE   low
5 P5   trt  24    27   TRUE  high
6 P6  ctrl  25    21  FALSE   low
```

```r
nrow(df2)
```

```
[1] 6
```

Adding columns: `merge()`

Add the columns of another dataframe, matching the rows by a **key** (here `id`)

```r
sleep <- data.frame(id = c("P3", "P1", "P6", "P2", "P5", "P4"),
                    hours = c(8, 7, 6, 6, 9, 7))
sleep
```

```
  id hours
1 P3     8
2 P1     7
3 P6     6
4 P2     6
5 P5     9
6 P4     7
```

`merge()`: the result

Each row of `df2` gets the `hours` of **its own** `id`, even if the order is different

```
1 df3 <- merge(df2, sleep, by = "id")
2 df3
```

```
  id group age score passed level hours
1 P1  ctrl  22    18  FALSE   low     7
2 P2   trt  NA    25   TRUE  high     6
3 P3  ctrl  31    30   TRUE  high     8
4 P4   trt  29    22  FALSE   low     7
5 P5   trt  24    27   TRUE  high     9
6 P6  ctrl  25    21  FALSE   low     6
```

`merge()` when some ids are missing

Here `sleep2` has only some of the ids of `df2`, and one more (`P7`)

```
1 sleep2 <- data.frame(id = c("P1", "P2", "P7"),
2                      hours = c(7, 6, 8))
3 sleep2
```

```
  id hours
1 P1     7
2 P2     6
3 P7     8
```

```
1 # only the ids in both dataframes
2 df4 <- merge(df2, sleep2, by = "id")
3 df4
```

```
  id group age score passed level hours
1 P1  ctrl  22    18  FALSE   low     7
2 P2   trt  NA    25   TRUE  high     6
```

Keeping all the rows: `all.x = TRUE`

The rows of `df2` without a match get `NA`

```
1 df5 <- merge(df2, sleep2, by = "id", all.x = TRUE)
2 df5
```

```
  id group age score passed level hours
1 P1  ctrl  22    18  FALSE   low     7
2 P2   trt  NA    25   TRUE  high     6
3 P3  ctrl  31    30   TRUE  high    NA
4 P4   trt  29    22  FALSE   low    NA
5 P5   trt  24    27   TRUE  high    NA
6 P6  ctrl  25    21  FALSE   low    NA
```

Before the exam

Remember

- `<-` creates objects, `==` compares
- text always goes **in quotes**: `"hello"`, `"5"`
- with missing values: `mean(x, na.rm = TRUE)`
- vectors and matrices hold **one type** of data
- before renaming levels, look at `levels(x)`; to reorder and rename use `factor(x, levels, labels)`
- to use a new value in a factor, **add its level** first
- lists: `[[ ]]` or `$` to get an element
- dataframes: `df[rows, columns]`, e.g. `df[df$score > 20, ]`
- `rbind()`: **same column names**; `merge()`: a **common key**

Function summary

structure	dimensions	types	create	index
vector	1	one	`c()`, `seq()`, `rep()`	`x[i]`
factor	1	categories	`factor()`	`x[i]`, `levels()`
list	1 (nested)	any	`list()`	`x[i]`, `x[[i]]`, `x$name`
matrix	2	one	`matrix()`, `cbind()`	`x[row, col]`
dataframe	2	one per column	`data.frame()`	`x$col`, `x[row, col]`

Good luck!

During the exam you may use the material provided during the lessons.

Other online material (or ChatGPT) is not allowed.