Tutorial 01: Introduction to R

```
#| edit: false
#| output: false
webr::install("gradethis", quiet = TRUE)
library(gradethis)
options(webr.exercise.checker = function(
  label, user_code, solution_code, check_code, envir_result, evaluate_result,
  envir_prep, last_value, engine, stage, ...
) {
  if (is.null(check_code)) {
    # No grading code, so just skip grading
    invisible(NULL)
  } else if (is.null(label)) {
    list(
      correct = FALSE,
      type = "warning",
      message = "All exercises must have a label."
    )
  } else if (is.null(solution_code)) {
    list(
      correct = FALSE,
      type = "warning",
      message = htmltools::tags$div(
        htmltools::tags$p("A problem occurred grading this exercise."),
        htmltools::tags$p(
          "No solution code was found. Note that grading exercises using the ",
          htmltools::tags$code("gradethis"),
          "package requires a model solution to be included in the document."
        )
      )
    )
  } else {
    gradethis::gradethis_exercise_checker(
```

```
      label = label, solution_code = solution_code, user_code = user_code,
      check_code = check_code, envir_result = envir_result,
      evaluate_result = evaluate_result, envir_prep = envir_prep,
      last_value = last_value, stage = stage, engine = engine)
  }
})
```

Q1 — Make the sum equal 10

Replace with a number so the expression evaluates to **10**.

```
#| setup: true
#| exercise: ex_sum
 <- NA_real_
```

```
#| exercise: ex_sum
1 + 2 +  +
```

Replace the two with 2 numbers that add up to 7.

Solution.

```
#| exercise: ex_sum
#| solution: true
1 + 2 + 3 + 4
```

```
#| exercise: ex_sum
#| check: true
gradethis::grade_this({
  if (grepl(" ", .user_code, fixed = TRUE)) {
    fail("Replace  with a number.")
  }
  # Pass if the evaluated result is (numerically) 10
  if (is.numeric(.result) && length(.result) == 1 &&
      isTRUE(all.equal(.result, 10))) {
    pass(" Correct!")
  } else {
    fail("Not quite - try to make the expression evaluate to 10.")
  }
})
```

Q2 — Create a vector

Make a vector called `my_vec` that contains the numbers **5, 10, 15, 20**.

```
#| exercise: ex_vec
#| exercise.lines: 2
#| echo: false
my_vec <- NULL
```

Use the c() function to combine numbers: c(1, 2, 3)

Solution.

```
#| exercise: ex_vec
#| solution: true
my_vec <- c(5, 10, 15, 20)
```

```
#| exercise: ex_vec
#| check: true
gradethis::grade_this({
  if (!exists("my_vec", envir = .envir_result)) {
    fail("You need to define `my_vec`.")
  }
  v <- get("my_vec", envir = .envir_result)
  if (!is.numeric(v)) {
    fail("`my_vec` should be numeric.")
  }
  if (!identical(v, c(5, 10, 15, 20))) {
    fail("`my_vec` should contain exactly 5, 10, 15, 20 in that order.")
  }
  pass("  Nice! Your vector is correct.")
})
```

Q3 — Find the average

Compute the mean of the vector `c(2, 4, 6, 8, 10)`.
Your answer should be stored in a variable called `avg_val`.

```
#| exercise: ex_mean
#| exercise.lines: 2
#| echo: false
avg_val <- NULL
```

Use the mean() function: mean(c(…))

Solution.

```
#| exercise: ex_mean
#| solution: true
avg_val <- mean(c(2, 4, 6, 8, 10))
```

```
#| exercise: ex_mean
#| check: true
gradethis::grade_this({
  if (!exists("avg_val", envir = .envir_result)) {
    fail("You need to define `avg_val`.")
  }
  v <- get("avg_val", envir = .envir_result)
  if (!is.numeric(v) || length(v) != 1) {
    fail("`avg_val` should be a single numeric value.")
  }
  if (!isTRUE(all.equal(v, 6, tol = 1e-8))) {
    fail("Not quite - the mean of c(2,4,6,8,10) is 6.")
  }
  pass("  Correct! You computed the mean successfully.")
})
```

Q4 — Draw a histogram from ToothGrowth (type the full call)

Use exactly one of these numeric columns and nothing else: len or dose

Photo by The Humble Co. on Unsplash

> **Preview**
>
> Find a preview of the ToothGrowth dataset here:
>
> ```
> #| echo: true
> head(ToothGrowth, 8)
> ```

```
#| exercise: ex_hist_exact
# Type your answer on the next line (one command only):
```

Use the hist command.

Solution.

```
#| exercise: ex_hist_exact
#| solution: true
hist(ToothGrowth$len)
```

```
#| exercise: ex_hist_exact
#| check: true
gradethis::grade_this({
code <- .user_code
lines <- strsplit(paste(code, collapse = "\n"), "\n", fixed = TRUE)[[1]]
lines <- sub("#.*$", "", lines)
trim <- function(s) sub("^[[:space:]]+|[[:space:]]+$", "", s)
lines <- trim(lines); lines <- lines[nzchar(lines)]

if (length(lines) == 0) fail("Type your answer; it can't be empty.")
if (length(lines) > 1)  fail("Enter exactly one command.")

one <- lines[1]
if (!inherits(.result, "histogram")) fail("Your code should draw a histogram.")

expr <- try(parse(text = one)[[1]], silent = TRUE)
if (inherits(expr, "try-error")) fail("Your code must be a single valid R expression.")
if (!(is.call(expr) && identical(as.character(expr[[1]]), "hist"))) fail("Use hist(...).")

args <- as.list(expr)[-1]
if (length(args) != 1L) fail("Pass exactly one argument to hist().")

arg_txt <- paste0(deparse(args[[1]]), collapse = "")
allowed <- c("ToothGrowth$len","ToothGrowth$dose")

if (arg_txt %in% allowed) pass("OK")
else if (arg_txt == "ToothGrowth") fail("hist(ToothGrowth) won't work-choose len or dose.")
else fail("Type exactly hist(ToothGrowth$len) or hist(ToothGrowth$dose).")
})
```

Q5 — Draw a boxplot from ToothGrowth (type the full call)

Use exactly one of these numeric columns and nothing else: len or dose

Preview

Find a preview of the ToothGrowth dataset here:

```
#| echo: true
head(ToothGrowth, 8)
```

```
#| exercise: ex_boxplot_exact
# Type your answer on the next line (one command only):
```

Use the boxplot command.

Solution.

```
#| exercise: ex_boxplot_exact
#| solution: true
boxplot(ToothGrowth$len)
```

```
#| exercise: ex_boxplot_exact
#| check: true
gradethis::grade_this({
code <- .user_code
lines <- strsplit(paste(code, collapse = "\n"), "\n", fixed = TRUE)[[1]]
lines <- sub("#.*$", "", lines)
trim <- function(s) sub("^[[:space:]]+|[[:space:]]+$", "", s)
lines <- trim(lines); lines <- lines[nzchar(lines)]

if (length(lines) == 0) fail("Type your answer; it can't be empty.")
if (length(lines) > 1)  fail("Enter exactly one command.")

one <- lines[1]

expr <- try(parse(text = one)[[1]], silent = TRUE)
if (inherits(expr, "try-error")) fail("Your code must be a single valid R expression.")
if (!(is.call(expr) && identical(as.character(expr[[1]]), "boxplot"))) fail("Use boxplot(...)

args <- as.list(expr)[-1]
if (length(args) != 1L) fail("Pass exactly one argument to boxplot().")

arg_txt <- paste0(deparse(args[[1]]), collapse = "")
allowed <- c("ToothGrowth$len","ToothGrowth$dose")
```

```
if (arg_txt %in% allowed) {
if (!(is.list(.result) && !is.null(.result$stats))) pass("OK (note: base boxplot())") else pa
} else if (arg_txt == "ToothGrowth") {
fail("boxplot(ToothGrowth) won't work-choose len or dose.")
} else {
fail("Type exactly boxplot(ToothGrowth$len) or boxplot(ToothGrowth$dose).")
}
})
```

Q6 — Add a color to the histogram

Re-draw a histogram of ToothGrowth$len and set any bar color using the col= argument.

```
#| exercise: ex_hist_col
# (one command)
```

add col as an additional argument here and give it a color of your choice.

Solution.

```
#| exercise: ex_hist_col
#| solution: true
hist(ToothGrowth$len, col = "steelblue")
```

```
#| exercise: ex_hist_col
#| check: true
gradethis::grade_this({
line <- paste(.user_code, collapse = "\n")
if (!inherits(.result, "histogram")) fail("Use hist(...) on ToothGrowth$len.")

expr <- try(parse(text = line)[[1]], silent = TRUE)
if (inherits(expr, "try-error")) fail("Your code must be a single valid R expression.")
if (!(is.call(expr) && identical(as.character(expr[[1]]), "hist"))) fail("Call hist(...).")

args <- as.list(expr)[-1]
arg1_txt <- paste0(deparse(args[[1]]), collapse = "")
if (arg1_txt != "ToothGrowth$len") fail("Use ToothGrowth$len as the data argument.")

has_col <- any(names(args) == "col")
if (!has_col) fail("Add a color with col = ...")
```

```
pass("OK")
})
```

Q7 — Boxplot (make it horizontal)

Draw a boxplot of ToothGrowth$dose and make it horizontal using horizontal = TRUE as an argument.

```
#| exercise: ex_box_h
# (one command)
```

add horizontal as an argument just like col from the previous question

Solution.

```
#| exercise: ex_box_h
#| solution: true
boxplot(ToothGrowth$dose, horizontal = TRUE)
```

```
#| exercise: ex_box_h
#| check: true
gradethis::grade_this({
line <- paste(.user_code, collapse = "\n")

expr <- try(parse(text = line)[[1]], silent = TRUE)
if (inherits(expr, "try-error")) fail("Your code must be a single valid R expression.")
if (!(is.call(expr) && identical(as.character(expr[[1]]), "boxplot"))) fail("Use boxplot(...)

args <- as.list(expr)[-1]
if (length(args) < 1L) fail("Pass ToothGrowth$dose to boxplot().")

arg1_txt <- paste0(deparse(args[[1]]), collapse = "")
if (arg1_txt != "ToothGrowth$dose") fail("Use ToothGrowth$dose as the data argument.")

arg_names <- names(args)
if (is.null(arg_names)) arg_names <- rep("", length(args))
idx <- which(arg_names == "horizontal")
horiz_ok <- FALSE
if (length(idx) > 0) {
hval <- args[[idx[1]]]
```

```
horiz_ok <- isTRUE(eval(hval))
}
if (!horiz_ok) fail("Add horizontal = TRUE.")

if (!(is.list(.result) && !is.null(.result$stats))) {
pass("OK (note: ensure you're using base boxplot())")
} else {
pass("OK")
}
})
```

Q8 — Load a CSV as a DataFrame

Load iris.csv into a dataframe named df.

Photo by Andrew Small on Unsplash

```
# Q9 SETUP - create iris.csv inside the webR filesystem
#| setup: true
#| exercise: ex_csv_load
write.csv(iris, "iris.csv", row.names = FALSE)
```

```
## Q9 - Load a CSV as a data frame
# Read data/iris.csv into a data frame named df
# Read iris.csv into a data frame named df
#| exercise: ex_csv_load
#| exercise.lines: 3
#| echo: false
df <-
```

Use read.csv here.

Solution.

```
#| exercise: ex_csv_load
#| solution: true
#| files: ["data/iris.csv"]   # solution runs in its own sandbox - mount again
df <- read.csv("iris.csv")
```

```
#| exercise: ex_csv_load
#| check: true
gradethis::grade_this({
  if (!exists("df", envir = .envir_result)) fail("Create `df` with `read.csv(\"iris.csv\")`.
  x <- get("df", envir = .envir_result)
  if (!is.data.frame(x)) fail("`df` should be a data frame.")
  need <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species")
  if (!all(need %in% names(x))) fail("`df` doesn't look like the expected columns.")
  pass(" Loaded correctly!")
})
```

Q9 — Count rows in iris.csv

Count the number of rows in iris.csv and store them in the variable rows

```
# Q10 setup
#| setup: true
#| exercise: ex_csv_nrows
write.csv(iris, "iris.csv", row.names = FALSE)
```

```
#| exercise: ex_csv_nrows
#| exercise.lines: 3
#| echo: false
df <- read.csv("iris.csv")
rows <-
```

Use nrow.

Solution.

```
#| exercise: ex_csv_nrows
#| solution: true
#| files: ["data/iris.csv"]   # solution runs in its own sandbox - mount again
df <- read.csv("iris.csv")
rows <- nrow(df)
```

```
#| exercise: ex_csv_nrows
#| check: true
gradethis::grade_this({
  stopifnot(exists("rows", envir=.envir_result))
  x <- get("rows", envir=.envir_result)
```

```
  if (!is.numeric(x) || length(x)!=1) fail("`rows` must be a single number.")
  if (x==150) pass("  150 rows.") else pass("  Count recorded.")
})
```

Q10 — Show column names

Print the column names from iris.csv

```
#| setup: true
#| exercise: ex_csv_names
write.csv(iris, "iris.csv", row.names = FALSE)
```

```
#| exercise: ex_csv_names
#| exercise.lines: 4
#| echo: false
# Read the CSV and print the column names
df   <- read.csv("iris.csv")
cols <-
```

Use names.

Solution.

```
#| exercise: ex_csv_names
#| solution: true
#| files: ["data/iris.csv"]   # solution runs in its own sandbox - mount again
df   <- read.csv("iris.csv")
cols <- names(df)
cols
```

```
#| exercise: ex_csv_names
#| check: true
gradethis::grade_this({
  if (!exists("cols", envir=.envir_result)) fail("Create `cols <- names(df)` and print it.")
  x <- get("cols", envir=.envir_result)
  if (!is.character(x)) fail("`cols` should be a character vector from `names(df)`.")
  need <- c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species")
  if (!all(need %in% x)) fail("Those don't look like the iris columns-did you read the CSV ar
  pass("  Columns detected: Sepal/ Petal and Species.")
})
```

Q11 — Print some of the data

Print the first 5 rows from iris.csv.

```
#| setup: true
#| exercise: ex_csv_head
write.csv(iris, "iris.csv", row.names = FALSE)
```

```
#| exercise: ex_csv_head
#| exercise.lines: 4
#| echo: false
# Read the CSV and print the column names
df   <- read.csv("iris.csv")
#your code here
```

Use head.

Solution.

```
#| exercise: ex_csv_head
#| solution: true
#| files: ["data/iris.csv"]   # solution runs in its own sandbox - mount again
df <- read.csv("iris.csv")
head(df)
```

```
#| exercise: ex_csv_head
#| check: true
gradethis::grade_this({
  # If the *last* expression is head(df, 5), .result should be a 5-row data.frame
  if (!(is.data.frame(.result) && nrow(.result)==5)) {
    fail("Show the first 5 rows (e.g., `head(df, 5)`) as the **last** line.")
  }
  if (!all(c("Sepal.Length","Sepal.Width","Petal.Length","Petal.Width","Species") %in% names
    fail("Your preview should be from `df` read from the CSV (it must have the iris columns)
  }
  pass("  Nice-now you can clearly see the columns and a few rows.")
})
```

Q12 — Make a histogram from iris.csv

Make a histogram of the column Sepal.Length from iris.csv.

```
#| setup: true
#| exercise: ex_csv_hist
write.csv(iris, "iris.csv", row.names = FALSE)
```

```
#| exercise: ex_csv_hist
#| exercise.lines: 3
#| echo: false
df <- read.csv("iris.csv")
```

Use hist (see Q5).

Solution.

```
#| exercise: ex_csv_hist
#| solution: true
#| files: ["data/iris.csv"]   # solution runs in its own sandbox - mount again
df <- read.csv("iris.csv")
hist(df$Sepal.Length, col = "lightgray", main = "Sepal.Length", xlab = "cm")
```

```
#| exercise: ex_csv_hist
#| check: true
gradethis::grade_this({
  if (!inherits(.result,"histogram")) fail("Use base `hist(df$Sepal.Length, ...)`.")
  if (!grepl("Sepal\\.Length", paste(.user_code, collapse="\n"))) fail("Plot `df$Sepal.Length
  pass("  Histogram detected.")
})
```

Q13 — Write a function using division

Define a function called `divide_nums` that takes two arguments (`a` and `b`) and returns the results of:

1. `a / b`
2. `b / a`

Both results should be stored in separate variables before being returned.

> **Info**
>
> Remember: in R, division by zero (1/0) will return Inf rather than an error.

Info

In R, lists are written with list(a, b, c). Each element can be anything: a number, a string, or even another list.

```
#| exercise: ex_fun_div
#| exercise.lines: 8
#| echo: false
divide_nums <- function(a, b) {
  # your code here
}
```

Start with: res1 <- a / b res2 <- b / a and then use the list command.

Solution.

```
#| exercise: ex_fun_div
#| solution: true
divide_nums <- function(a, b) {
  res1 <- a / b
  res2 <- b / a
  list(res1, res2)
}
```

```
#| exercise: ex_fun_div
#| check: true
gradethis::grade_this({
  code <- .user_code

  # 0) Empty/untouched?
  if (!nzchar(gsub("\\s|#.*", "", code))) {
    fail("Type your answer in the editor.")
  }

  # 1) Find the student's env (webR/gradethis names vary)
  env <- get0(".envir_result", ifnotfound = NULL)
  if (is.null(env)) env <- get0(".user_env", ifnotfound = NULL)
  if (is.null(env)) env <- parent.frame()

  # 2) Get their function (prefer the named binding; fall back to .result)
  f <- NULL
  if (exists("divide_nums", envir = env, inherits = FALSE)) {
```

```
    f <- get("divide_nums", envir = env)
  } else if (is.function(.result)) {
    f <- .result
  }

  if (!is.function(f)) {
    fail("Define a function named `divide_nums(a, b)` (use `divide_nums <- function(a, b) {
  }

  # 3) Run it and grade
  val <- tryCatch(f(8, 2), error = function(e) e)
  if (inherits(val, "error")) {
    msg <- conditionMessage(val)
    if (grepl("object .* not found", msg)) {
      fail("Looks like you referenced a variable that wasn't created (e.g., `res1` or `res2`)
    } else {
      fail(paste0("Your function raised an error: ", msg))
    }
  }

  if (!is.list(val) || length(val) != 2) {
    fail("Return a **list** of two elements: `list(res1, res2)`.")
  }
  if (!(is.numeric(val[[1]]) && is.numeric(val[[2]]))) {
    fail("Both returned values must be numeric (`a/b` and `b/a`).")
  }

  ok <- isTRUE(all.equal(val[[1]], 4,    tol = 1e-8)) &&
        isTRUE(all.equal(val[[2]], 0.25, tol = 1e-8))
  if (!ok) {
    fail("Close! Compute `res1 <- a / b` and `res2 <- b / a`, then return `list(res1, res2)`
  }

  pass("  Well done! Function name, outputs, and return format are correct.")
})
```