1. Why write functions

- Avoid copy-pasting repeated code.
- Makes your code clearer, easier to maintain, and reusable.
- Consider writing a function if you've repeated the same code three times or more.

2. Types of functions

    1. Vector functions
        - Take vectors as input, return vectors or single summaries.
        - Examples:
            - rescale01() – rescales numeric vectors between 0 and 1.
            - z_score() – standardizes to mean 0, SD 1.
            - clamp() – ensures values stay within min/max.
            - first_upper() – capitalizes the first letter of strings.
            - clean_number() – removes special characters and converts to numeric.
            - cv() – coefficient of variation.
            - mape() – mean absolute percentage error.
    2. Data frame functions
        - Take a data frame as input, return a data frame or vector.
        - Tidyverse challenge: indirection arises because dplyr uses tidy evaluation.
        - Use {{ }} (embracing) for data-masking variables.
        - Use pick() for tidy-selection of multiple variables.
        - Examples:
            - grouped_mean(df, group_var, mean_var) – mean of a variable grouped by another.

        - summary6() – min, mean, median, max, n, n_miss.
        - count_prop() – counts + proportion.
        - unique_where() – finds unique values based on a condition.
        - subset_flights() – subsets flights dataset.
    3. Plot functions
        - Return ggplot2 plots.
        - Use data-masking with {{ }} for variable names in aes().
        - Examples:
            - histogram(df, var, binwidth) – basic histogram.
            - linearity_check(df, x, y) – scatterplot with loess + linear fit.
            - hex_plot(df, x, y, z) – hexbin summary plots.
            - sorted_bars(df, var) – frequency-sorted bar chart.
            - conditional_bars(df, condition, var) – filtered bar chart.
        - Labeling plots: use rlang::englue() to dynamically insert variable names or parameters into titles.

3. Best practices / style

- Function names: Short but descriptive; generally verbs.
- Argument names: Nouns that clearly describe what they accept.
- Indent the function body for readability.
- Use := in mutate() when assigning a new variable name dynamically.
- Extra spacing inside {{ }} improves readability.

4. Exercises

- Turn repeated code snippets into functions.

- Write functions for:
    - Handling NA values.
    - Rescaling vectors with -Inf → 0, Inf → 1.
    - Age calculation from birthdates.
    - Variance and skewness.
    - Counting positions with NA in two vectors (both_na()).
- Data frame functions: filters, summaries, converting clock time, generalizing count_prop() for multiple variables.
- Plotting: build scatterplots with optional linear fit and titles.

5. Takeaways

- Functions make your code more readable, less error-prone, reusable.
- Tidy evaluation is essential for writing functions that use dplyr or ggplot2
- Vector functions are simple; data frame and plot functions are slightly more advanced due to tidy evaluation.
- Naming and formatting style matters a lot for human readers.

6. Structures of functions

- if and else: testing a condition and acting on it
- for: execute a loop a fixed number of times
- while: execute a loop *while* a condition is true
- repeat: execute an infinite loop (must break out of it to stop)
- break: break the execution of a loop
- next: skip an iteration of a loop