Functions

Functions

- Incorporate sets of instructions that you want to use repeatedly or that, because of their complexity, are better self-contained in a sub program and called when needed.
- A function is a piece of code written to carry out a specified task; it can or can not accept arguments or parameters and it can or can not return one or more values.

Functions

- Automate common tasks in a more powerful and general way than copy-and-pasting

- Advantages:
  - You can give a function an evocative name that makes your code easier to understand.
  - As requirements change, you only need to update code in one place, instead of many.
  - You eliminate the chance of making incidental mistakes when you copy and paste (i.e. updating a variable name in one place, but not in another).

When should you write a function?

- Whenever you’ve copied and pasted a block of code more than twice (i.e. you now have three copies of the same code)

Basic syntax

```r
function_name <- function(inputs) {
  output_value <- do_something(inputs)
  return(output_value)
}
```

Three key steps to creating a function

1. You need to pick a **name** for the function. Pick something that helps you understand what the function does

2. You list the inputs, or **arguments**, to the function inside function

3. You place the code you have developed in **body** of the function, a { } block that immediately follows function(...).

Three key steps to creating a function

1. You need to pick a **name** for the function. Pick something that helps you understand what the function does

2. You list the inputs, or **arguments**, to the function inside function

3. You place the code you have developed in **body** of the function, a { } block that immediately follows function(…).

Then it's a good idea to do some testing with different inputs to make sure the function works as expected

Naming your functions

- Short, but clearly evoke what the function does
  - Better to be clear than short
- Generally, function names should be verbs, and arguments should be nouns.
- Nouns are ok if the function computes a very well known noun (i.e. mean() is better than compute_mean())
- A noun might be a better choice is if you're using a very broad verb like "get", "compute", "calculate", or "determine"

Example names

```
# Too short
f()

# Not a verb, or descriptive
my_awesome_function()

# Long, but clear
impute_missing()
collapse_years()
```

Use consistent prefix for related functions

```
# Good
input_select()
input_checkbox()
input_text()

# Not so good
select_input()
checkbox_input()
text_input()
```

Use commenting to explain your code and separate sections

- # Explain “why”

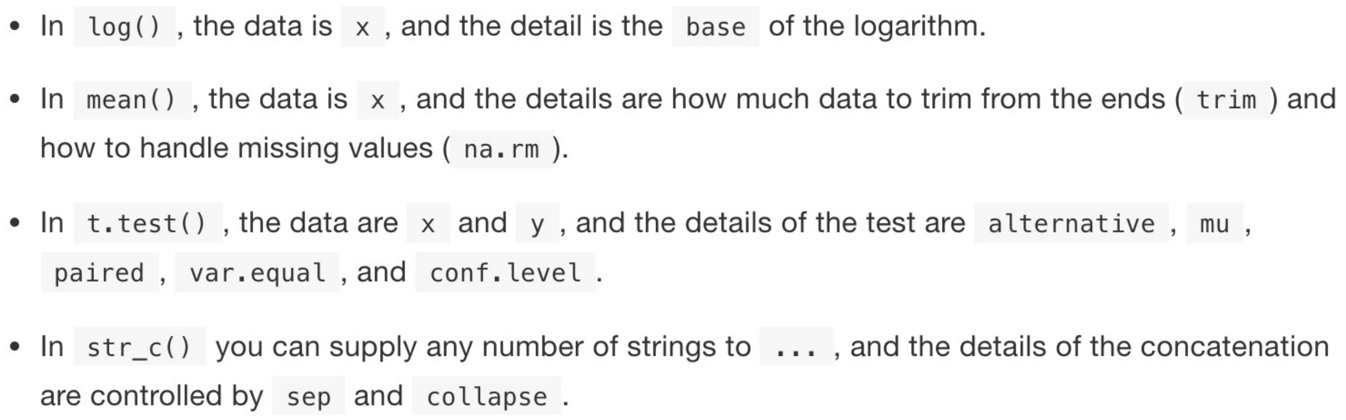
- Generally the ”what” and “how” should be clear from the code. If not, consider adding intermediate variables with useful names

Take-homes

- Functions are both for computers and humans
- R doesn’t care what your function is called, or what comments it contains, but these are important for human readers

Function arguments

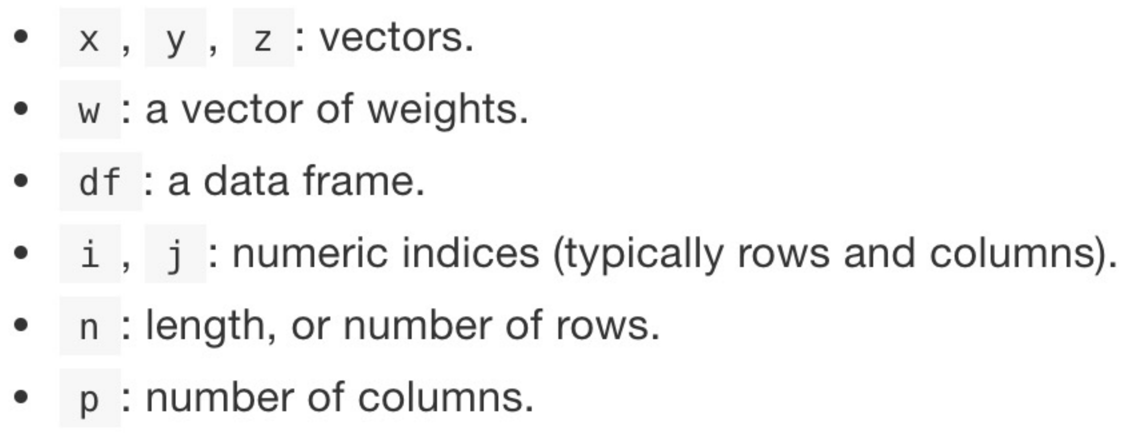
- Two broad sets:
  - One set supplies the **data** to compute on
  - The other supplies arguments that control the **details** of the computation



- In `log()`, the data is `x`, and the detail is the `base` of the logarithm.
- In `mean()`, the data is `x`, and the details are how much data to trim from the ends (`trim`) and how to handle missing values (`na.rm`).
- In `t.test()`, the data are `x` and `y`, and the details of the test are `alternative`, `mu`, `paired`, `var.equal`, and `conf.level`.
- In `str_c()` you can supply any number of strings to `...`, and the details of the concatenation are controlled by `sep` and `collapse`.

Choosing names

Common defaults:



- `x` , `y` , `z` : vectors.
- `w` : a vector of weights.
- `df` : a data frame.
- `i` , `j` : numeric indices (typically rows and columns).
- `n` : length, or number of rows.
- `p` : number of columns.

What does a function return?

- Usually the last statement it evaluates

- Can choose to return early by using return()

What does a function return?

- Usually the last statement it evaluates

- Can choose to return early by using return()

```r
complicated_function <- function(x, y, z) {
  if (length(x) == 0 || length(y) == 0) {
    return(0)
  }

  # Complicated code here
}
```