

An introduction to the R platform

Day 3 - Functions

Tineka Blake

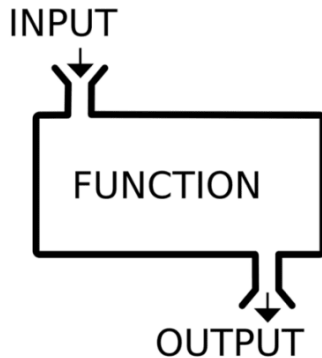
University of Nottingham



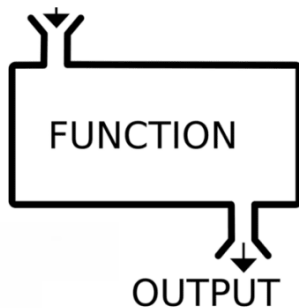
Outline

- What are functions?
- Why they are important
- When to use functions in R
- The power of writing your own
- What to do when you get stuck...

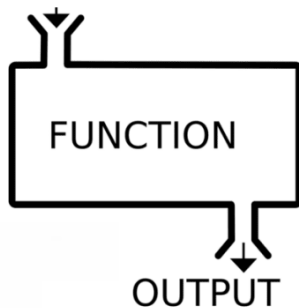
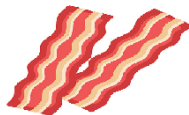
What is a function?



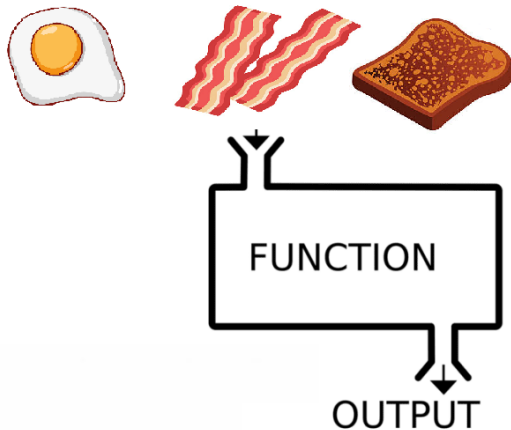
What is a function?



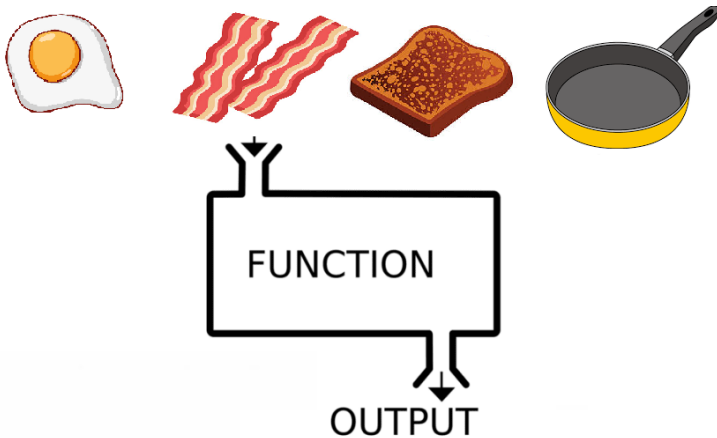
What is a function?



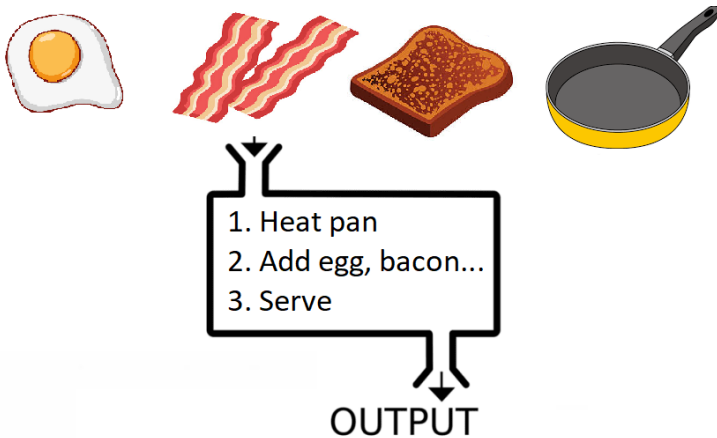
What is a function?



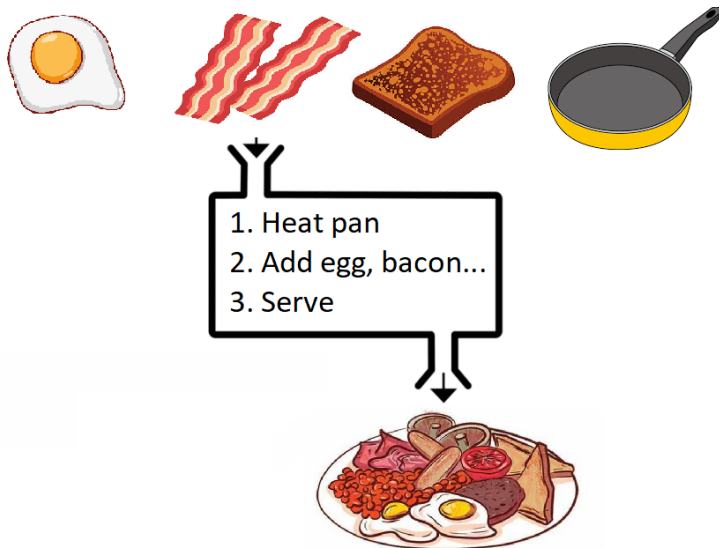
What is a function?



What is a function?



What is a function?



What is a function?

A set of instructions that performs a specific task

Examples:

- to print characters
- to sum numbers together
- to calculate the mean
- to run a linear regression model

When to use them?

- You are already using functions **all the time!**
- R is a *functional* programming language
- In R, you can use the functions:
 - 1 'pre-baked' in R language (base R, utils)
 - 2 within packages (i.e. dplyr, tidyr)
 - 3 write your own

Perhaps you already recognise these base R functions?

- `print("hello")`
- `sum(2,3)`
- `mean(1:5)`
- `lm()` *stats

Let's delve a little deeper into *lm*

```
> lm
function (formula, data, subset, weights, na.action, method = "qr",
  model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
  contrasts = NULL, offset, ...)
{
  ret.x <- x
  ret.y <- y
  cl <- match.call()
  mf <- match.call(expand.dots = FALSE)
  m <- match(c("formula", "data", "subset",
    "weights", "na.action", "offset"),
    names(mf), 0L)
  mf <- mf[c(1L, m)]
  mf$drop.unused.levels <- TRUE
  mf[[1L]] <- quote(stats::model.frame)
  mf <- eval(mf, parent.frame())
  if (method == "model.frame")
    return(mf)
  else if (method == "qr")
    warning(gettextf("method '%s' is not supported. Using 'qr'",
      method), domain = NA)
  mt <- attr(,"terms")
  y <- model.response(mf, "numeric")
  w <- as.vector(model.weights(mf))
  if (!is.null(w) && !is.numeric(w))
    stop("'weights' must be a numeric vector")
  offset <- model.offset(mf)
  nlm <- is.matrix(y)
  ny <- if (nlm)
    nrow(y)
  else length(y)
  if (!is.null(offset)) {
    if (nlm)
      offset <- as.vector(offset)
    if (NROW(offset) != ny)
      stop(gettextf("number of offsets is %d, should equal %d (number of observations)",
        NROW(offset), ny), domain = NA)
  }
  if (is.empty.model(mt)) {
    x <- NULL
    z <- list(coefficients = if (nlm) matrix(NA_real_, 0,
      ncol(y)) else numeric(), residuals = y, fitted.values = 0 =
      y, weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w !=
      0) else ny)
    if (!is.null(offset)) {
      z$fitted.values <- offset
      z$residuals <- y - offset
    }
  }
  else {
    x <- model.matrix(mt, mf, contrasts)
    z <- if (!is.null(w))
      lm.fit(x, y, offset = offset, singular.ok = singular.ok,
        ...
      )
    else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
      ...
    )
  }
  class(z) <- c(if (nlm) "nlm", "lm")
  z$na.action <- attr(mf, "na.action")
  z$offset <- offset
  z$contrasts <- attr(x, "contrasts")
  z$xlevels <- getLevels(mt, mf)
  z$call <- cl
  z$terms <- mt
  if (model)
    z$model <- mf
  if (ret.x)
    z$x <- x
  if (ret.y)
    z$y <- y
  if (qr)
    z$qr <- NULL
  z
}
<bytecode: 0x00000293a0e8ff5a>
<environment: namespace:stats>
```

`lm(pHw ~ pHCaCl, data = data.df)`

Why use functions?

The main purpose of functions is to break up **complex code into simpler digestable chunks**

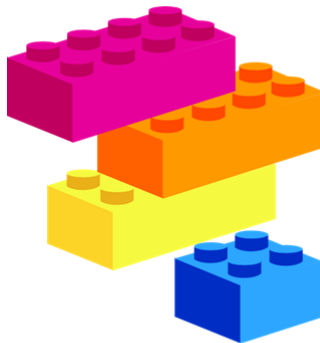
Advantages:

- Reduces code redundancy
- Improves modularity (changes are isolated)
- Resuable
- Debugging

Disadvantages:

- Thinking of a name! (*Functions names are usually verbs - as functions "do" things*)

Function modularity

 $f_1(x)$ $f_2(x)$ $f_3(x)$ $f_4(x)$ 

The power of writing your own

- There are things that you are doing over and over and over again...
- Can construct your own algorithm. What is it that *you* need to do?
- You get to become a creator
- Combine your functions into your own package! For yourself and others too

You can go as far as you would like to with this...

Where to go if I get stuck?

- Book: The Art of R Programming (Norman Matloff)
- Book: The Pragmatic Programmer (Andrew Hunt & David Thomas)
- Stackoverflow
- Google / Duck Duck Go

Chances are somebody else has struggled too...

- Resist the temptation to just copy and paste a solution.
- Aim to *understand* the solution.
- There's fun to be had in programming. It's a *skill* that can be developed over time - not everything is 'intuitive'.