

Rangkuman Hands-on Programming with R

Hendra Bunyamin

17 December 2020

Chapter 1 The Very Basics

Chapter 1 dari buku **Hands-on Programming with R**. Catatan ini merupakan bagian dari proyek 1: **Weighted Dice**.

The R User Interface

Tidak ada yang penting.

Objects

Tidak ada yang penting.

Functions

`round` function is used to round a number.

```
round(3.1415)
```

```
## [1] 3
```

`factorial` function is used to calculate the factorial of a number.

```
factorial(3)
```

```
## [1] 6
```

`mean` function is used to compute the mean of number(s).

```
mean(1:6)
```

```
## [1] 3.5
```

We can combine `round` and `mean` functions as follows:

```
round(mean(1:6))
```

```
## [1] 4
```

Sample with Replacement

The `sample` function is used to sample.

```
die <- 1:6  
sample(x = die, size=1)
```

```
## [1] 5
```

args can be used to view the arguments of a function.

```
args(round)
```

```
## function (x, digits = 0)
## NULL
```

```
args(sample)
```

```
## function (x, size, replace = FALSE, prob = NULL)
## NULL
```

round method can be accompanied with digits, for example:

```
round(3.1415, digits = 2)
```

```
## [1] 3.14
```

By **default**, the sample function is *without replacement*.

```
sample(die, size=2)
```

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

When we want to sample *with replacement*, we can set replace=TRUE as follows:

```
sample(die, size=2, replace = TRUE)
```

```
## [1] 1 1
```

If you want to add up the dice, you can feed your result straight into the sum function:

```
die <- sample(die, size=2, replace = TRUE)
dice
```

```
## [1] 2 4
```

```
sum(dice)
```

```
## [1] 6
```

```
die <- 1:6
```

Writing Your Own Functions

We are going to write a function which returns the sum of rolling two dice.

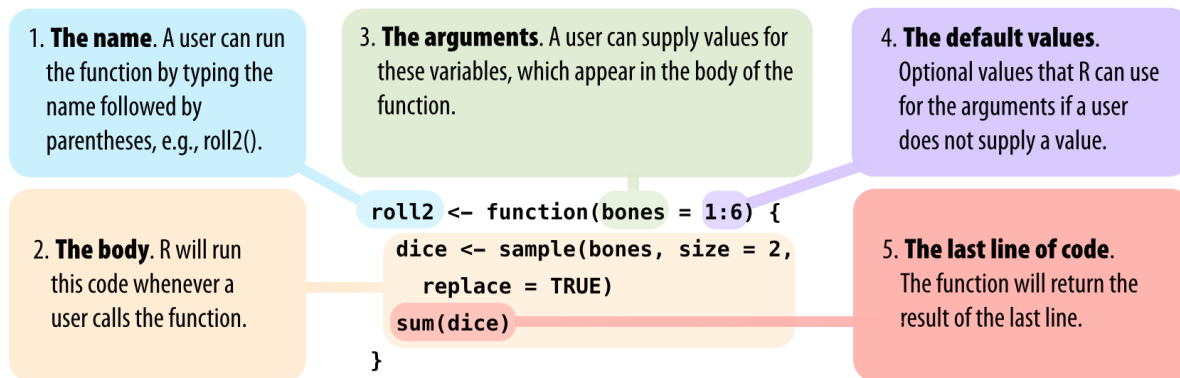
```
roll <- function(){
  die <- 1:6
  dice <- sample(die, size=2, replace = TRUE)
  sum(dice)
}
result <- roll()
result
```

```
## [1] 3
```

When you run a function in R, R will execute all of the code in the body and then return **the result of the last line of code**.

The Function Constructor

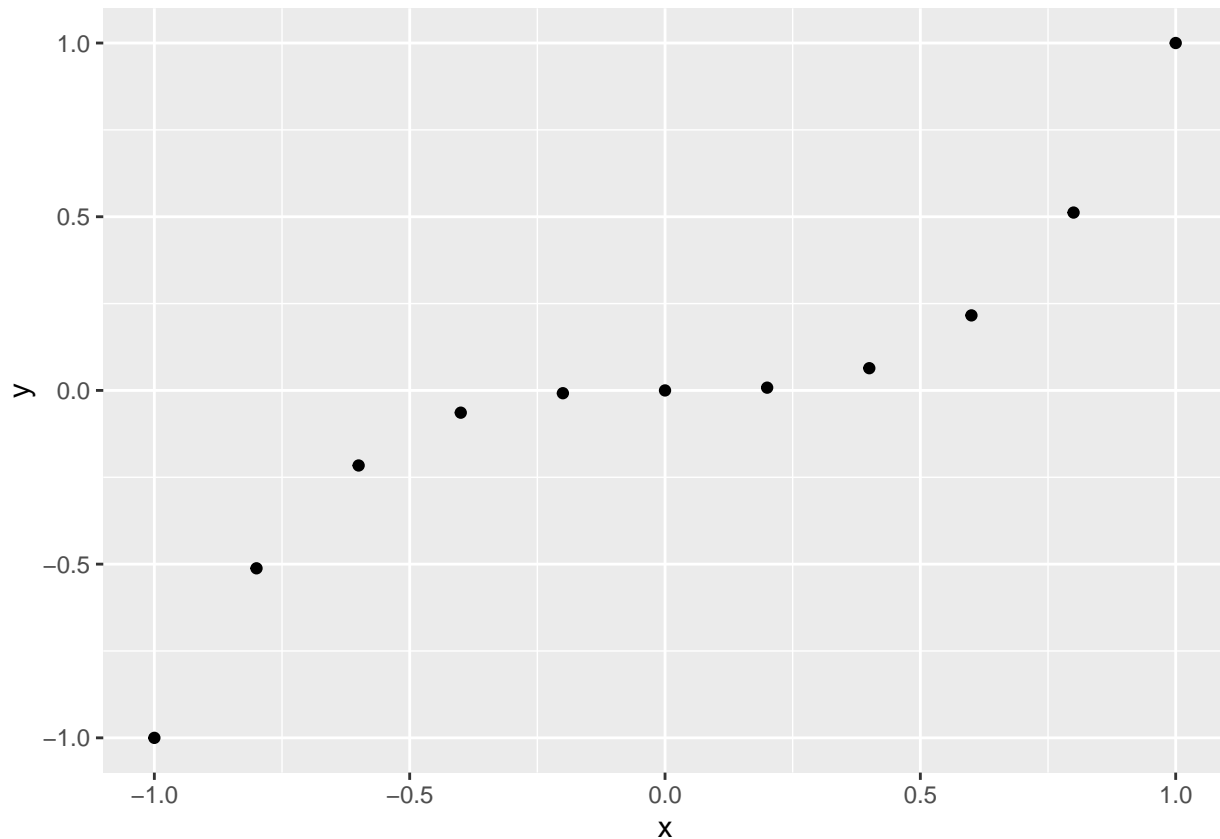
Arguments



Scripts

Menggunakan `qplot` dengan data pakai `c`.

```
library("ggplot2")  
  
x <- c(-1, -0.8, -0.6, -0.4, -0.2, 0, 0.2, 0.4, 0.6, 0.8, 1)  
x  
  
## [1] -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6 0.8 1.0  
## -1.0 -0.8 -0.6 -0.4 -0.2 0.0 0.2 0.4 0.6 0.8 1.0  
  
y <- x^3  
y  
  
## [1] -1.000 -0.512 -0.216 -0.064 -0.008 0.000 0.008 0.064 0.216 0.512  
## [11] 1.000  
## -1.000 -0.512 -0.216 -0.064 -0.008 0.000 0.008 0.064 0.216 0.512 1.000  
  
qplot(x,y)
```



Summary

Packages and Help Pages

Packages

- Package `qplot` berarti quick plot.
- Berikut adalah fungsinya dan `qplot` is a long function

```
library(ggplot2)
qplot
```

```
## function (x, y, ..., data, facets = NULL, margins = FALSE, geom = "auto",
##   xlim = c(NA, NA), ylim = c(NA, NA), log = "", main = NULL,
##   xlab = NULL, ylab = NULL, asp = NA, stat = deprecated(),
##   position = deprecated())
## {
##   deprecate_soft0("3.4.0", "qplot()")
##   caller_env <- parent.frame()
##   if (lifecycle::is_present(stat))
##     lifecycle::deprecate_stop("2.0.0", "qplot(stat)")
##   if (lifecycle::is_present(position))
##     lifecycle::deprecate_stop("2.0.0", "qplot(position)")
##   check_character(geom)
##   exprs <- enquos(x = x, y = y, ...)
##   is_missing <- vapply(exprs, quo_is_missing, logical(1))
##   is_constant <- (!names(exprs) %in% ggplot_global$all_aesthetics) |
##     vapply(exprs, quo_is_call, logical(1), name = "I")
```

```

## mapping <- new_aes(exprs[!is_missing & !is_constant], env = parent.frame())
## consts <- exprs[is_constant]
## aes_names <- names(mapping)
## mapping <- rename_aes(mapping)
## if (is.null(xlab)) {
##   if (quo_is_missing(exprs$x)) {
##     xlab <- ""
##   }
##   else {
##     xlab <- as_label(exprs$x)
##   }
## }
## if (is.null(ylab)) {
##   if (quo_is_missing(exprs$y)) {
##     ylab <- ""
##   }
##   else {
##     ylab <- as_label(exprs$y)
##   }
## }
## if (missing(data)) {
##   data <- data_frame0()
##   facetvars <- all.vars(facets)
##   facetvars <- facetvars[facetvars != "."]
##   names(facetvars) <- facetvars
##   facetsdf <- as.data.frame(mget(facetvars, envir = caller_env))
##   if (nrow(facetsdf))
##     data <- facetsdf
## }
## if ("auto" %in% geom) {
##   if ("sample" %in% aes_names) {
##     geom[geom == "auto"] <- "qq"
##   }
##   else if (missing(y)) {
##     x <- eval_tidy(mapping$x, data, caller_env)
##     if (is.discrete(x)) {
##       geom[geom == "auto"] <- "bar"
##     }
##     else {
##       geom[geom == "auto"] <- "histogram"
##     }
##     if (is.null(ylab))
##       ylab <- "count"
##   }
##   else {
##     if (missing(x)) {
##       mapping$x <- quo(seq_along(!mapping$y))
##     }
##     geom[geom == "auto"] <- "point"
##   }
## }
## p <- ggplot(data, mapping, environment = caller_env)
## if (is.null(facets)) {
##   p <- p + facet_null()

```

```

##   }
##   else if (is.formula(facets) && length(facets) == 2) {
##     p <- p + facet_wrap(facets)
##   }
##   else {
##     p <- p + facet_grid(rows = deparse(facets), margins = margins)
##   }
##   if (!is.null(main))
##     p <- p + ggtitle(main)
##   for (g in geom) {
##     params <- lapply(consts, eval_tidy)
##     p <- p + do.call(paste0("geom_", g), params)
##   }
##   logv <- function(var) var %in% strsplit(log, "")[[1]]
##   if (logv("x"))
##     p <- p + scale_x_log10()
##   if (logv("y"))
##     p <- p + scale_y_log10()
##   if (!is.na(asp))
##     p <- p + theme(aspect.ratio = asp)
##   if (!missing(xlab))
##     p <- p + xlab(xlab)
##   if (!missing(ylab))
##     p <- p + ylab(ylab)
##   if (!missing(xlim) && !all(is.na(xlim)))
##     p <- p + xlim(xlim)
##   if (!missing(ylim) && !all(is.na(ylim)))
##     p <- p + ylim(ylim)
##   p
## }
## <bytecode: 0x564fcd4b5f68>
## <environment: namespace:ggplot2>

```

- If you give `qplot` two vectors of equal lengths, `qplot` will draw a scatterplot for you.
- `qplot` will use the first vector as a set of x values and the second vector as a set of y values.
- Until now, we've been creating sequences of numbers with the `:` operator; but you can also create vectors of numbers with the `c` function.
- Give `c` all of the numbers that you want to appear in the vector, separated by a comma. `c` stands for *concatenate*, but you can think of it as “collect” or “combine”/