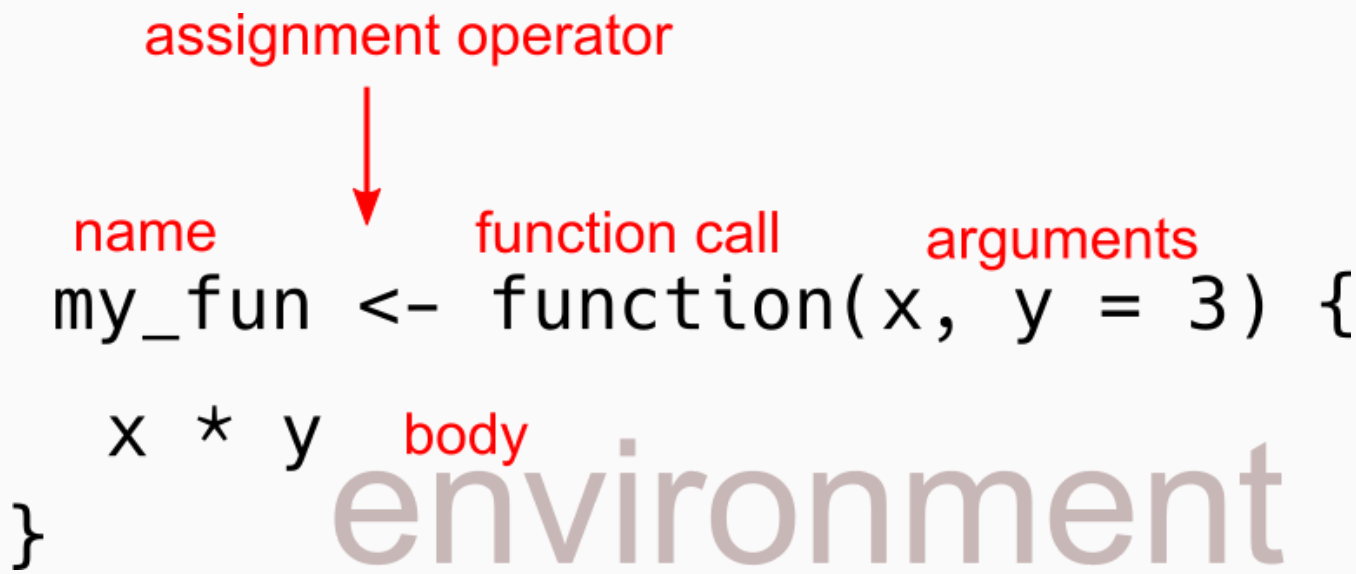
Programación funcional en R

for loops, purrr, tidyeval

Dr. Samuel D. Gamboa Tuz

Componentes de una función



- Puedes revisar los componentes de una función con:
  - `formals()`.
  - `body()`.
  - `environment()` (scoping).

Componentes de *if statement* y *for loop*

Components of if statement

```
if (TRUE/FALSE) "It's true." else "It's false."
```

if statement — condition — else/else if statement

output if TRUE — output if FALSE

Components of for loops

```
for (i in 1:5) print(i)
```

for/in statement — vector — body

item var

for loops

- Supongamos que queremos conocer el número de elementos únicos en cada columna de un dataframe. Usar `length(unique(x))` en cada columna podría ser poco práctico, especialmente si son muchas columnas.

- Una opción es escribir una función con un for loop:

```
library(tidyverse)
count_unique_values <- function(data) {
  output <- vector("double", length(data))
  for (i in seq_along(data)) {
    output[[i]] <- length(unique(data[[i]]))
  }
  rm(i)
  names(output) <- names(data)
  output
}

count_unique_values(diamonds)
```

```
##   carat     cut   color clarity   depth   table   price       x
##     273       5       7       8     184     127   11602     554
##       y       z
##     552     375
```

- A veces queremos aplicar una función solo a las columnas que cumplen cierta condición.
- Podemos incluir una declaración **if/else** en la función:

```
show_unique_values <- function(data) {
  output <- vector("list", length(data))
  for (i in seq_along(data)) {
    if(is.factor(data[[i]]) | is.character(data[[i]]) ) {
      output[[i]] <- as.character(unique(data[[i]]))
      names(output)[i] <- names(data[i])
    } else {
      output[[i]] <- NULL
    }
  }
  output[!is.na(names(output))]
}

show_unique_values(diamonds)
```

```
## $cut
## [1] "Ideal"     "Premium"   "Good"      "Very Good" "Fair"
##
## $color
## [1] "E" "I" "J" "H" "F" "G" "D"
##
## $clarity
## [1] "SI2"  "SI1"  "VS1"  "VS2"  "VVS2" "VVS1" "I1"   "IF"
```

purrr

- Escribir for loops puede ser "verboso" (*verbose*). Mejor usar purrr:
- Número de elementos únicos en todas las variables:

```
map_dbl(diamonds, n_distinct)
```

```
##   carat     cut   color clarity   depth   table   price       x
##     273       5       7       8     184     127   11602     554
##       y       z
##     552     375
```

- Elementos únicos en variables categorícas:

```
keep(diamonds, ~is.character(.x) | is.factor(.x)) %>%
  map(~unique(as.character(.x)))
```

```
## $cut
## [1] "Ideal"     "Premium"   "Good"      "Very Good" "Fair"
##
## $color
## [1] "E" "I" "J" "H" "F" "G" "D"
##
## $clarity
## [1] "SI2"  "SI1"  "VS1"  "VS2"  "VVS2" "VVS1" "I1"   "IF"
```

Tidyeval

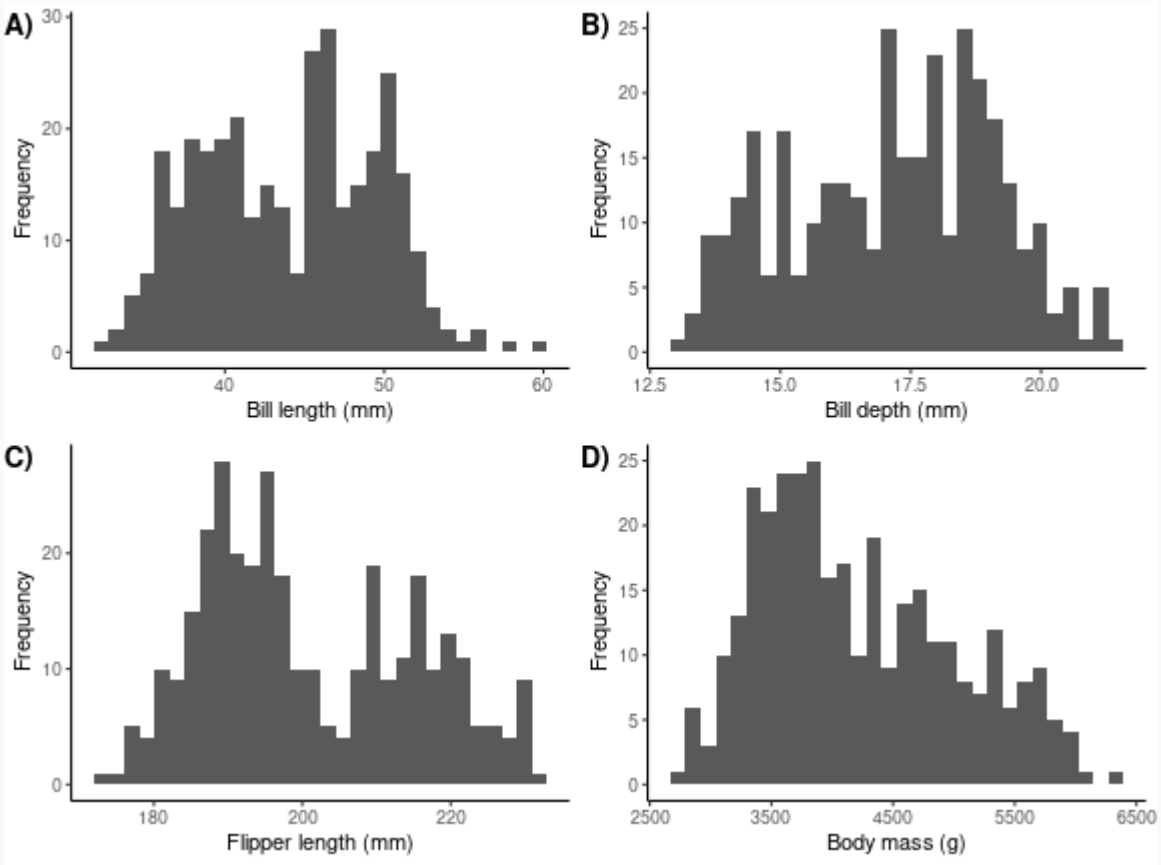
- Para usar funciones del tidyverse dentro de funciones personalizadas se necesita tidyeval.

- Por ejemplo, hagamos una función para hacer un histograma de cada una de las variables de palmer penguins

```
penguins <- na.omit(palmerpenguins::penguins)
get_label <- function(x) {
  x_var <- rlang::as_name(enquo(x))
  return(case_when(x_var == "bill_length_mm" ~ "Bill length (mm)",
                   x_var == "bill_depth_mm" ~ "Bill depth (mm)",
                   x_var == "flipper_length_mm" ~ "Flipper length (mm)",
                   x_var == "body_mass_g" ~ "Body mass (g)"))
}
plot_histogram <- function(data, x) {
  x_var <- enquo(x)
  ggplot(data, aes(!!x_var)) +
    geom_histogram() +
    labs(y = "Frequency", x = get_label(!!x_var)) +
    theme_classic()
}
p1 <- plot_histogram(penguins, bill_length_mm)
p2 <- plot_histogram(penguins, bill_depth_mm)
p3 <- plot_histogram(penguins, flipper_length_mm)
p4 <- plot_histogram(penguins, body_mass_g)
```

- Ahora podemos poner estas gráficas en una sola figura:

```
ggpubr::ggarrange(p1, p2, p3, p4,
                  labels = c("A)", "B)", "C)", "D)"),
                  hjust = 0)
```



Actividades recomendadas

- Investiga `switch()` e `ifelse()`; compara con `if`.
- Investiga `next()` y `break()` dentro de un *for loop*.
- Investiga *while loops* y `repeat()`; compara con *for loops*.
- Investiga acerca de las funciones de la familia "apply". Compáralas con purrr.

Bibilografía recomendada

- Capítulo 21 de R for Data Science (https://r4ds.had.co.nz/).
- Tidyeval (https://tidyeval.tidyverse.org/)