powered by *R programming*

Introducción general al uso de R
- Clase 05 -

minolicnp@gmail.com

Ignacio Minoli *pwp*

Observatorio de Biodiversidad del Bosque Atlántico (OBBA)
Instituto de Biología Subtropical (IBS) - CONICET - UNaM.

Condicionales operadores

- Una expresión condicional es una construcción de programación en la que se toma la decisión de ejecutar algún código en función de una condición booleana (FALSE o TRUE).
- Un término más comúnmente utilizado para la expresión condicional en la programación es una condición 'if-else'. R utiliza: **if** (condición es un elemento), **if - else** (condición es un elemento), **ifelse** (condición es un vector).

Operadores lógicos:

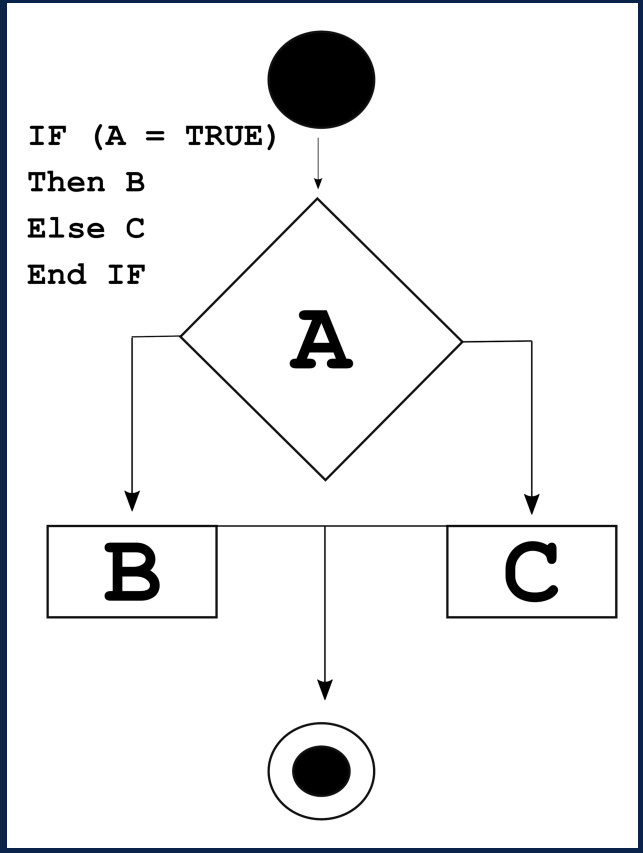
& : y, and
| : o, or
! : no, not

Operadores de comparación:

== : igual, equal
!= : no igual, not equal
< : menor a, less than
> : mayor a, greater than
>= : mayor o igual a, greater than or equal
<= : menor o igual a, less than or equal

Expresiones condicionales

- En lenguaje cotidiano, esto se expresa como ¿? ...
- Si esta condición es verdadera, entonces haga esta operación; de lo contrario, haga esta operación diferente.



Condicionales - Nuevas variables

```
options(width = 300)
library(palmerpenguins)
library(dplyr)
new_peng <- tibble(penguins)

# Veo las 1ras 2 lineas
new_peng %>% print(n = 2, width =Inf)
```

```
# A tibble: 344 × 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int> <fct>  <int>
1 Adelie  Torgersen           39.1          18.7               181        3750 male    2007
2 Adelie  Torgersen           39.5          17.4               186        3800 female  2007
# ℹ 342 more rows
```

- Condicional para crear una nueva variable categórica con 2 niveles de factor: “liviano” y “pesado”.

```
options(width = 300)
# Creo nueva variable
new_peng$masa_categ <- ifelse(new_peng$body_mass_g < 3760,
  "liviano", "pesado")

# Veo las 1ras 2 lineas
new_peng %>% print(n = 2, width =Inf)
```

```
# A tibble: 344 × 9
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year masa_categ
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int> <fct>  <int> <chr>
1 Adelie  Torgersen           39.1          18.7               181        3750 male    2007 liviano
2 Adelie  Torgersen           39.5          17.4               186        3800 female  2007 pesado
# ℹ 342 more rows
```

Condicionales - Identificar

- Otra función muy utilizada es **which()** que devuelve el número de filas que reunen las condiciones declaradas. Es decir, sirve para identificar observaciones o registros a través de la condición ... pero como obtengo un subset completo con **which()**?

```
# Que información me da este código??
which(penguins$body_mass_g >= 6000)
```

```
[1] 170 186 230 270
```

- Es posible complementar la función **which()** con indexación:

```
# Que información me da este código??
penguins[which(penguins$body_mass_g >= 6000), ]
```

```
# A tibble: 4 × 8
  species island bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex    year
  <fct>   <fct>           <dbl>         <dbl>             <int>       <int> <fct> <int>
1 Gentoo  Biscoe           49.2          15.2               221        6300 male   2007
2 Gentoo  Biscoe           59.6          17                 230        6050 male   2007
3 Gentoo  Biscoe           51.1          16.3               220        6000 male   2008
4 Gentoo  Biscoe           48.8          16.2               222        6000 male   2009
```

Condicionales - tidyverse

- Dentro de la familia **tidyverse** está el paquete **dplyr**. Este paquete es el más usado para manipulación de datos, ya sea en un data frame o tibble.
- La función **case_when()** es una alternativa al **ifelse()** del R base.

```
options(width = 300)
library(dplyr)

new_peng_2 <- penguins %>%
  mutate(
    masa_categ = case_when(
      body_mass_g < 3760 ~ "liviano",
      body_mass_g > 3760 ~ "pesado"
      )
    )

new_peng_2 %>% print(n = 2, width = Inf)
```

```
# A tibble: 344 × 9
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g sex     year masa_categ
  <fct>   <fct>              <dbl>         <dbl>             <int>       <int> <fct>  <int> <chr>
1 Adelie  Torgersen           39.1          18.7               181        3750 male    2007 liviano
2 Adelie  Torgersen           39.5          17.4               186        3800 female  2007 pesado
# ℹ 342 more rows
```

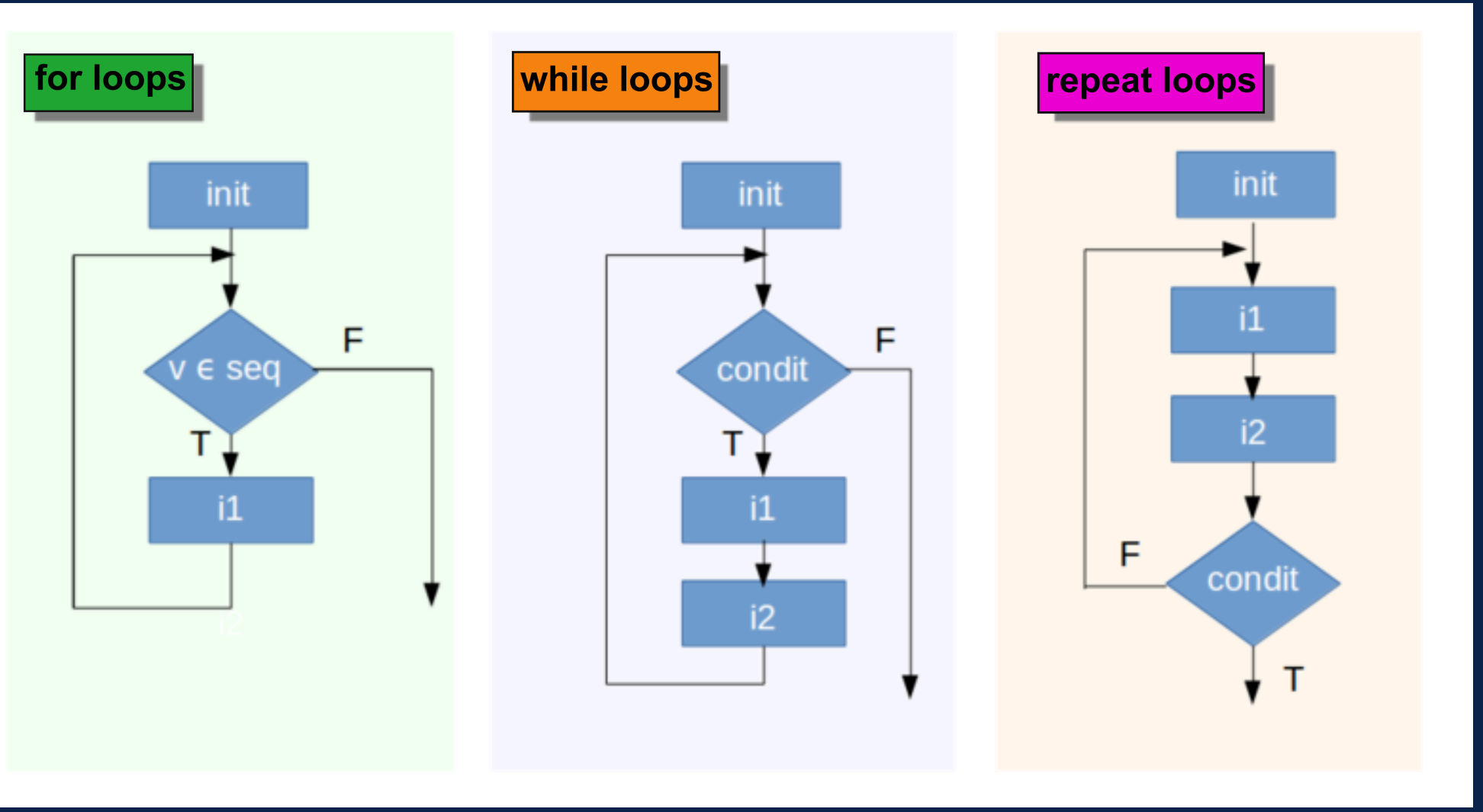
Loops en R

- “Loops”, “Ciclos”, “Iteraciones” o sólo replicar instrucciones existen desde que existen las computadoras.
- Es automatizar un proceso de múltiples pasos con la organización de una secuencia de acciones o procesos “por lotes” al agrupando las partes que deben repetirse.
- Todos los lenguajes de programación poseen construcciones especiales que permiten la repetición de intrucciones o bloques de instrucciones.

Hay tres tipos de loops:

- Loops que se ejecutan un determinado número de veces controlado por un índice o contador, que se incrementa en cada iteración: **for** loops.
- Loops que se basan en el inicio y la verificación de una condición lógica son los **while** loops.
- La condición se prueba al principio o al final de la construcción del loop son los **repeat** loops.

Loops en R


for loops
init
v ∈ seq
F
T
i1
i2
while loops
init
condit
F
T
i1
i2
repeat loops
init
i1
i2
F
condit
T

for loops en R

- Ejemplo de **for** loops. El primer valor de la secuencia es 1.

```
# El loop comienza reemplazando i por 1 y va iterando hasta llegar al 5.
for (i in 1:5) {
print(i)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

```
# El loop comienza reemplazando i por 1 y va iterando sumando 1 hasta llegar al 5.
for (i in 1:5) {
print(i + 1)
}
```

```
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
```

while loops en R

- Ejemplo de **while** loops. La estructura básica es while(logical_condition){ expression }.

```
        # El loop continua pasando valores a la instrucción hasta que supera
        # el valor definido (4) y se detiene.

        i <- 0
        while (i <= 4) {
        i <- i + 1
        print(i)
        }
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
```

repeat loops en R

- Ejemplo de **repeat** loops. Itera sobre un bloque de código varias veces una y otra vez hasta que se encuentra una declaración de ruptura.

```
        resultado <- c("No aprendí nada de R")
        i <- 1

        # Testea la expresión
        repeat {
        print(resultado)

        # Actualiza la expresión
        i <- i + 1

        # Rompe la condición
        if(i > 5) {
          break
          }
        }
```

```
[1] "No aprendí nada de R"
[1] "No aprendí nada de R"
[1] "No aprendí nada de R"
[1] "No aprendí nada de R"
[1] "No aprendí nada de R"
```

Loops en R

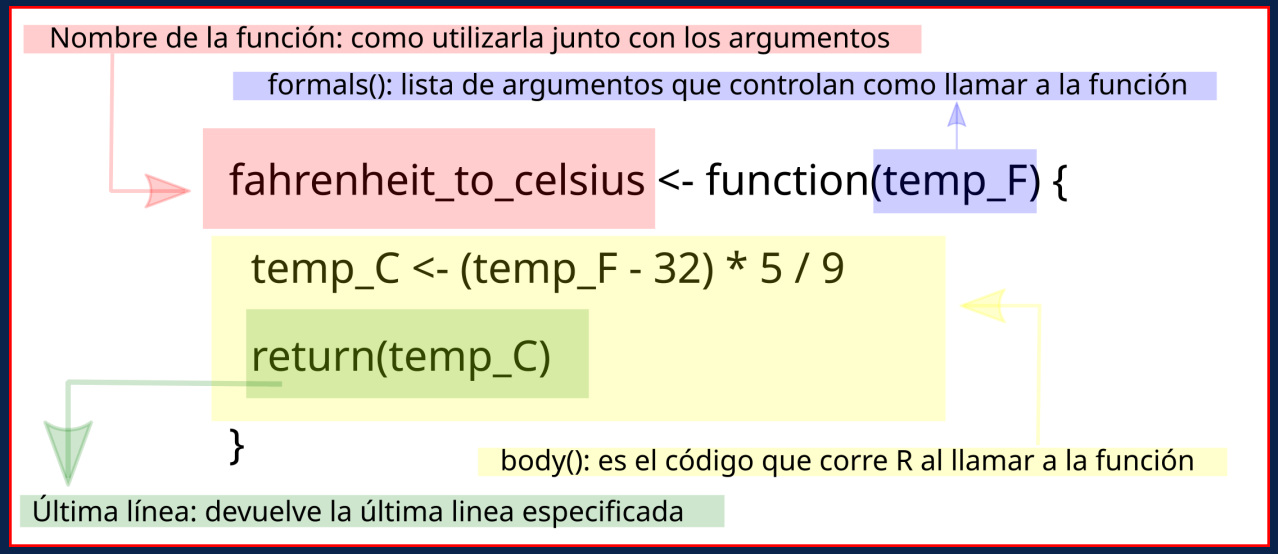
- Entonces ... ¿Cuando conviene usar loops?
- Ejemplos de cuando sería adecuado utilizarlos:
  * Trabajando en simulaciones.
  * Relaciones recursivas de los valores: valores que dependen de valores previos.
  * Problemas condicionales complejos.
  * **Instrucciones iterativas en lote sobre directorios con muchos archivos.**
- Entonces si no quiero o sé usar loops... ¿Qué puedo hacer?
- Si trabajo con data frames, se recomienda usar la familia de funciones apply: **apply()**, **lapply()**, **tapply()**, **sapply()**, **vapply()**, y **mapply()**.

```
        # sapply para cargar múltiples paquetes de una sola vez

        packages <- c("magrittr","dplyr","sf","tidyverse")
        sapply(packages, require, character.only=T)
```

```
magrittr     dplyr        sf tidyverse
    TRUE      TRUE      TRUE      TRUE
```

Funciones




```
fahrenheit_to_celsius_2 <- function(temp_F) {
  temp_C <- (temp_F - 32) * 5 / 9
  return(round(temp_C, 2))

}

library(tidyverse)
df_temps <- data.frame(
  grados_F = c(50, 202, 85, 76),
  Trat = c("Testigo", "Alcalino", "Acido", "Neutro")
  )

fahrenheit_to_celsius_2(df_temps$grados_F)
```

```
[1] 10.00 94.44 29.44 24.44
```

Funciones

- Se pueden crear funciones anidadas, es decir, una función aplicada al resultado de otra función:

```
        print(fahrenheit_to_celsius_2(
        df_temps$grados_F)
        )
```

```
[1] 10.00 94.44 29.44 24.44
```

```
        print(fahrenheit_to_celsius_2(
        df_temps$grados_F) %>% mean()
        )
```

```
[1] 39.58
```

- En una función propia puedo aprovechar y realizar:
  * Varios estadísticos en un mismo set de datos.
  * Imprimirlos en consola.
  * Crear variables nuevas.
  * Re estructurar un data frame con la función propia.
  * Guardar en un archivo nuevo (e.g., .csv).

Funciones

- Ejemplo de una función creando una variable nueva y re diseñando el data frame:

```
        df_temps
```

```
  grados_F     Trat
1       50  Testigo
2      202 Alcalino
3       85    Acido
4       76   Neutro
```

```
        df_temps_2 <- df_temps

        fahrenheit_to_celsius_3 <- function(temp_F) {
          new_df <- subset(temp_F)[1]
          names(new_df)[1] <- "grados_C"
          temp_C <- (new_df - 32) * 5 / 9
          cbind(temp_F, round(temp_C, 2))
        }

        df_temps_3 <- fahrenheit_to_celsius_3(df_temps_2)
        df_temps_3
```

```
  grados_F     Trat grados_C
1       50  Testigo    10.00
2      202 Alcalino    94.44
3       85    Acido    29.44
4       76   Neutro    24.44
```

Funciones

- Inspeccionamos cada objeto para ver como actuó la función:

```
kableExtra::kable(
  df_temps_2,
  align = "cc"
  )
```

grados_F	Trat
50	Testigo
202	Alcalino
85	Acido
76	Neutro

```
kableExtra::kable(
  df_temps_3,
  align = "ccc"
  )
```

grados_F	Trat	grados_C
50	Testigo	10.00
202	Alcalino	94.44
85	Acido	29.44
76	Neutro	24.44

Resumen

- Funciones Condicionales: **if()**, **ifelse()**, **case_when()**
- Ejemplos: Loops: **for(){}**, **while(){}**, **repeat{}**

QR - presentación

PDF Presentación: Clase 05 - Teoría

Fin. Clase 05 - Teoría

¡¡¡ Muchas gracias !!!

¿ Preguntas ? ... ¿ Consultas ?