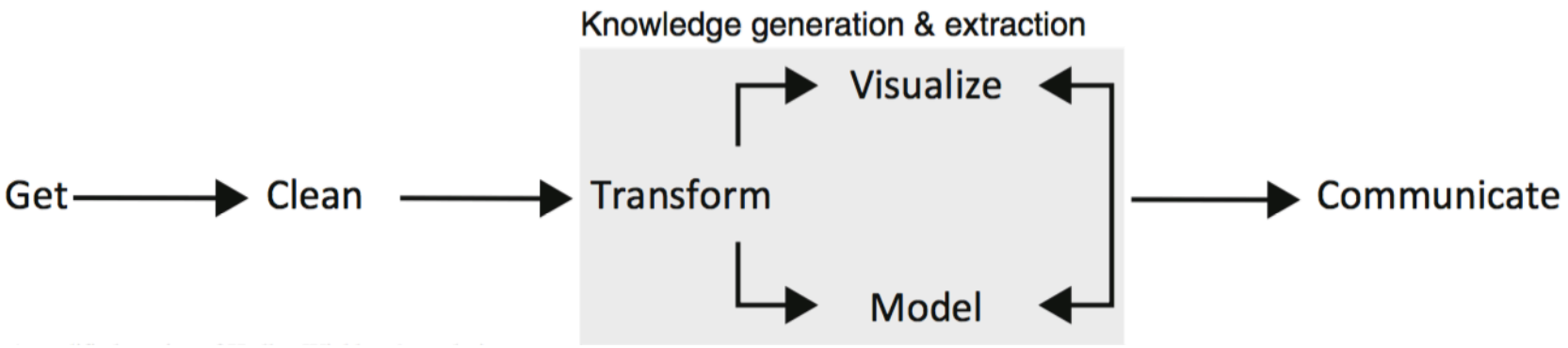
Manejo de datos en R (I)

Introducción a la Línea de Comandos para Análisis Bioinformáticos

10 de Agosto, 2021

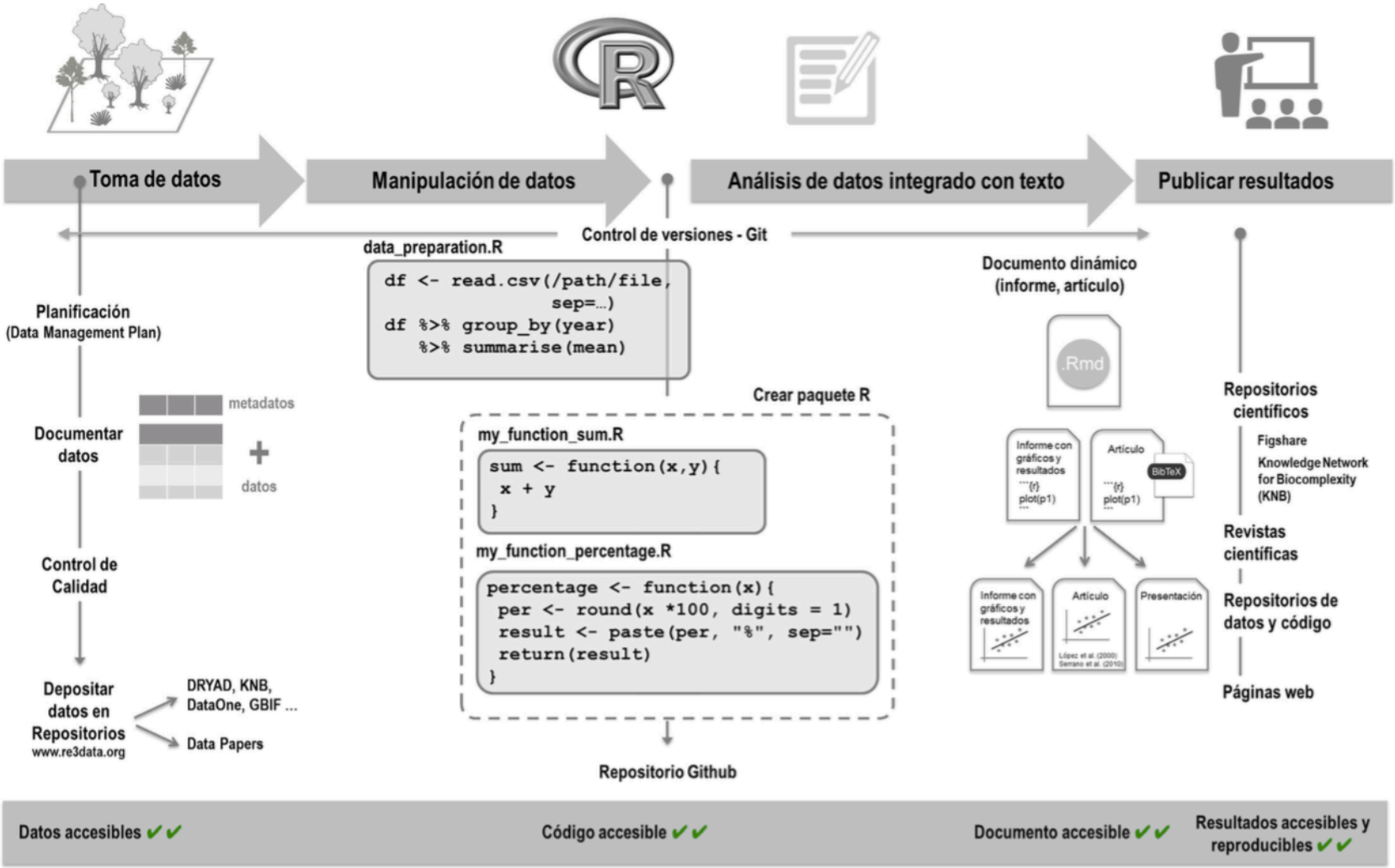
Que vamos a hacer?

Manejo de datos y análisis reproducible



Data Wrangling with R (Boehmke, 2016)

Análisis reproducible en R



Ciencia reproducible: qué, por qué, cómo (Rodríguez-Sánchez et al., 2016)

Manejo de datos

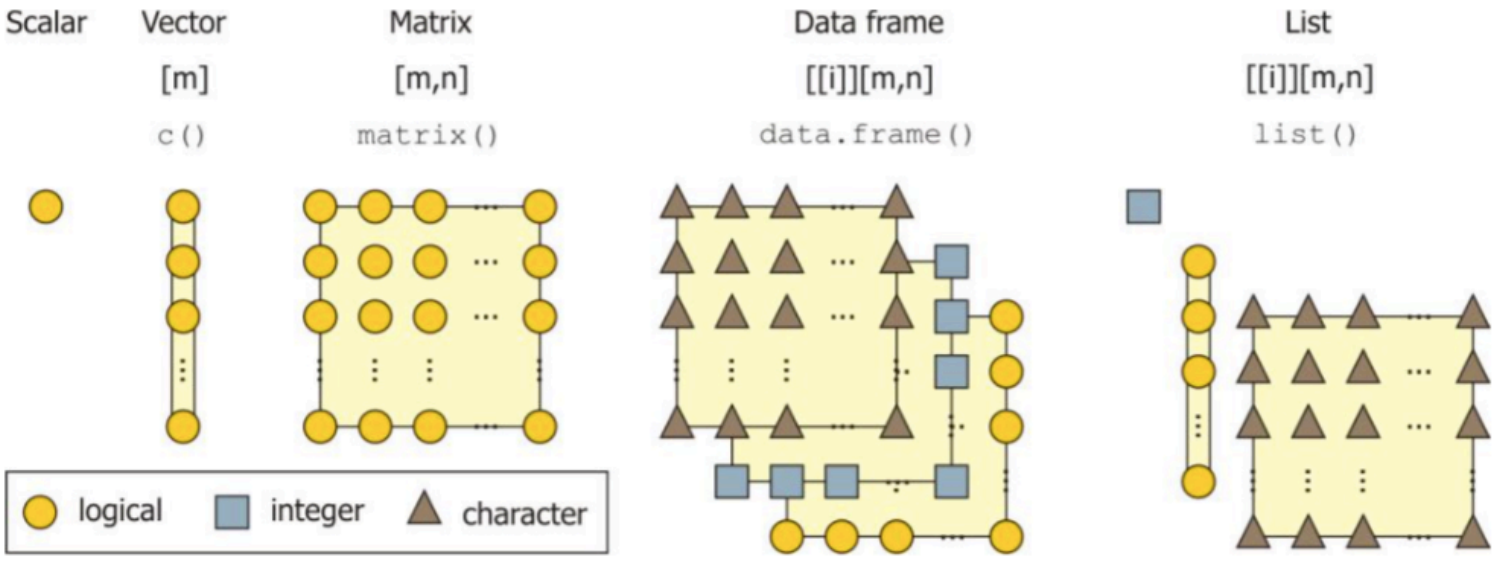
- ***Data wrangling***: es el proceso mediante el cual modificamos datos iniciales con el fin de analizarlos.
- Incluye la edición, el filtrado, la obtención de nuevos valores y más.
- *"In our experience, the tasks of exploratory data mining and data cleaning constitute 80% of the effort that determines 80% of the value of the ultimate data mining results. (...)"*. \ **Dasu & Johnson.** *Exploratory Data Mining and Data Cleaning* (2003).

Estructura de las clases

- Teórico/práctico.
- Práctico 11: repaso de loops y armado de funciones en R
  - accionar sobre los datos para transformarlos: funciones
  - realizar acciones repetitivas: loops
- Práctico 12: manejo de datos con paquetes de la librería **tidyverse**.
  - filtrado y edición de datos
  - visualización de los datos

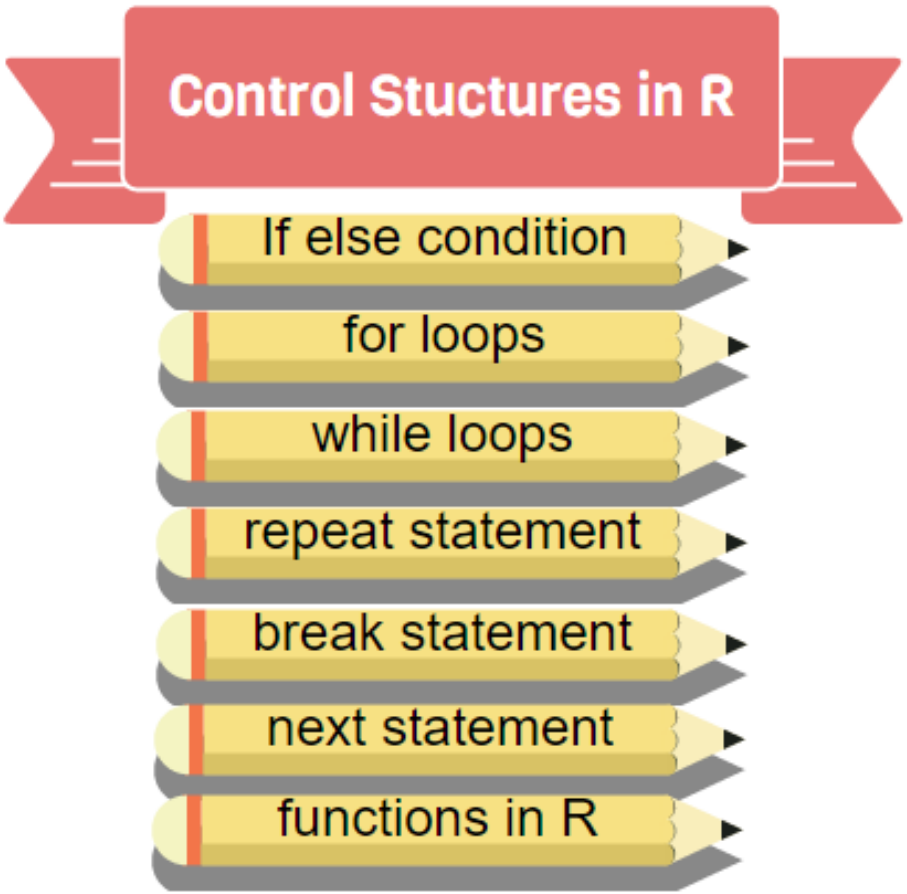
Breve repaso de R

Breve repaso



A practical guide to the R package Luminescence (Dietze et al., 2013)

Breve repaso

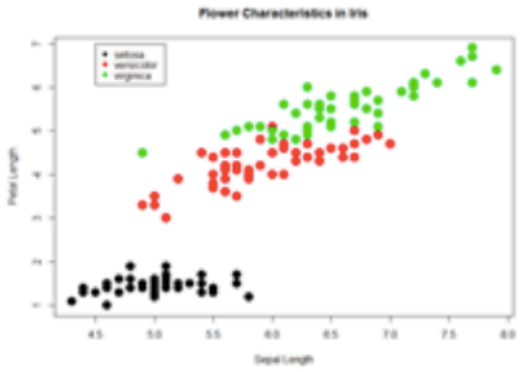


Control Structures in R (R-Bloggers)

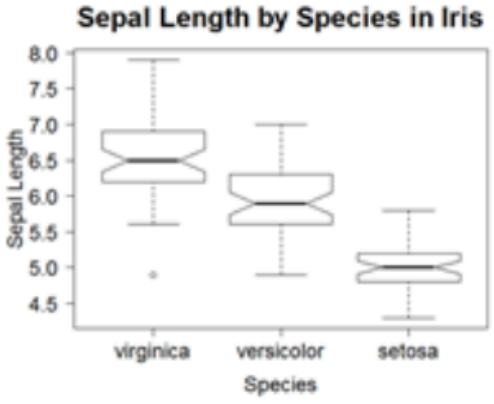
Breve repaso

1. Basic Histogram

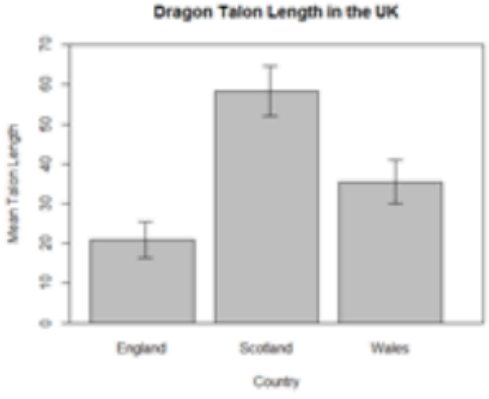
2. Line Graph with Regression



3. Scatterplot with Legend



4. Boxplot with reordered/ formatted axes



5. Boxplot with Error Bars

Funciones en R

Funciones: una parte central de R

- Qué es una función? Un conjunto de **operaciones** definidas que toman **argumentos** para dar un resultado.
- R Es un lenguaje de programación en base a funciones: casi todo lo que hacemos las utiliza.
  - Otros lenguajes operan de forma diferente.
- Ventajas
  - Le podemos dar nombre descriptivo
  - Nos ahorramos copiar y pegar codigo varias veces (y errar en el proceso)
  - Si hay que cambiar algo, es solo cambiar la funcion
- Las funciones en R salen de muchos lados:
  - R tiene funciones que vienen incorporadas por defecto
  - Utilizando librerías obtenemos nuevas funciones (como las de **seqinr**, por ejemplo)
  - Nosotros podemos hacer nuestras propias funciones

Ejemplo de una función

En la clase pasada operaron sobre variables que alojaban secuencias, utilizando funciones como la funcion **GC** de la libreria seqinr

```
library(seqinr)

seqinr::GC(secuencia)
```

```
## [1] 0.5454545
```

Como funciona esto?

Componentes de una función

- **cuerpo**: el código que define a la función
- **formales**: la lista de argumentos que controlan cómo se llama a la función
- **ambiente**: el "mapa" de la locación de las variables de la función
  - Es la unica que se define implicitamente, dependiendo de donde uno define la funcion

Componentes de una función

```
library(seqinr)

body(seqinr::GC)
```

```
## {
##     if (length(seq) == 1 && is.na(seq))
##         return(NA)
##     if (nchar(seq[1]) > 1)
##         stop("sequence is not a vector of chars")
...
```

Toda funcion tiene que especificar como hace lo que hace en algun lugar...

Componentes de una función

```
library(seqinr)

formals(seqinr::GC)
```

```
## $seq
##
##
## $forceToLower
## [1] TRUE
##
## $exact
## [1] FALSE
##
## $NA.GC
...
```

Los argumentos tambien se alojan en algun lugar.

Componentes de una función

```
library(seqinr)

environment(seqinr::GC)
```

```
## <environment: namespace:seqinr>
```

La funcion va a buscar, en primer lugar, variables definidas en el ambiente "seqinr".

Definición de funciones

Nosotros podemos hacer nuestras propias funciones, a medida de lo que necesitamos.

```
mi_funcion = function(argumento_1, argumento_2, ...){
#<-> el indentado no es obligatorio, pero ayuda a leer
  # en este bloque suceden operaciones con argumento_1
  ...
  # en este bloque suceden operaciones con argumento_2
  ...
  # se devuelve algo como resultado de aplicar
  # la funcion a los argumentos
  return(una_variable_nueva)
} # una linea sola para este parentesis ayuda a leer
```

Definición de funciones

Hagamos una funcion que opere con dos numeros cualesquiera, x e y.

```
eleva_y_resta = function(x,y){
  resultado = x^2 - y^2
  return(resultado)
}
```

Definición de funciones

Hagamos una funcion que opere con dos numeros cualesquiera, x e y.

```
eleva_y_resta = function(x,y){
  resultado = x^2 - y^2
  return(resultado)
}

eleva_y_resta(2,3)
```

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

Puedo pasar argumentos sin especificarlos explicitamente. Se interpreta el valor de cada argumento segun el orden de entrada.

Definición de funciones

Hagamos una funcion que opere con dos numeros cualesquiera, x e y.

```
eleva_y_resta = function(x,y){
  resultado = x^2 - y^2
  return(resultado)
}

eleva_y_resta(y = 2, x =3)
```

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

Definición de funciones

Hagamos una funcion que opere con dos numeros cualesquiera, x e y.

```
eleva_y_resta = function(x,y){
  resultado = x^2 - y^2
  return(resultado)
}

eleva_y_resta(y = 2)
```

```
## Error in eleva_y_resta(y = 2): argument "x" is missing, with
```

Si un argumento utilizado en la funcion no tiene un valor asignado, sucede un error!

Las funciones estan en todos lados...

"Para entender el computo en *R*, dos sloganes son utiles:

- Todo lo que existe es un objeto.
- Todo sucede llamando a una funcion."

— Joe Chambers

En R usamos funciones hasta sin saberlo

```
un_vector = c(1, 4, 6) # una funcion me permitio hacer esto
un_vector[1] # y tambien una funcion me permitio hacer esto
```

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

Tanto **c**, como **=**, como **[** son funciones, aunque no lo parezcan.

Las funciones estan en todos lados...

```
`[`
```

```
## .Primitive("[")
```

```
`for`
```

```
## .Primitive("for")
```

```
`c`
```

```
## function (...)  .Primitive("c")
```

Son funciones llamadas *primitivas*. Estan escritas en el lenguaje C de programacion, y no podemos acceder a su codigo.

Loops en R

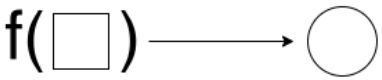
For loop (abstracto)

En las clases anteriores vieron estructuras del estilo:

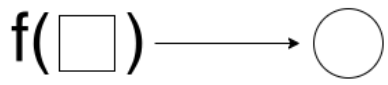
```
# for loop

un_vector = ...
for (i in ____) {
  ...
  ... un_vector[i] ....
  ...
}
```

En R los loops utilizan indices para acceder a elementos de un vector/lista

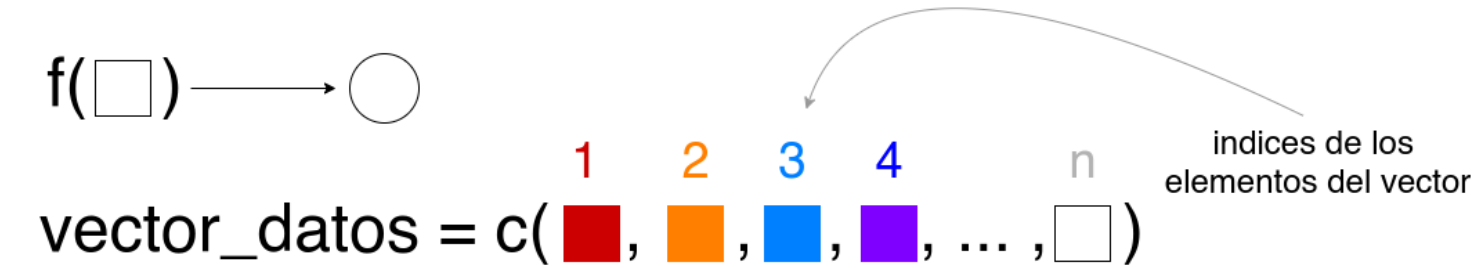
For loop (la logica)



For loop (la logica)



For loop (la logica)

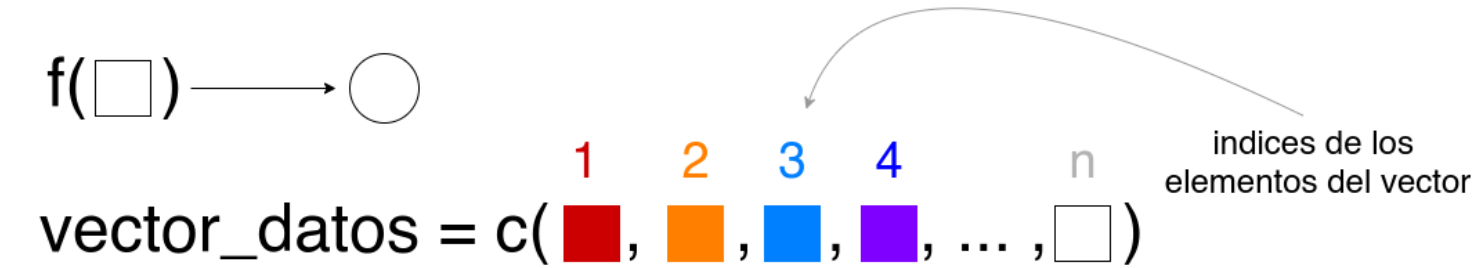


Sabemos que en R

vector_datos[1] = 

Accedemos a los datos con su indice en el vector. ⟶ La variable que vamos a hacer ir cambiando en el loop es esa (indice).

For loop (la logica)



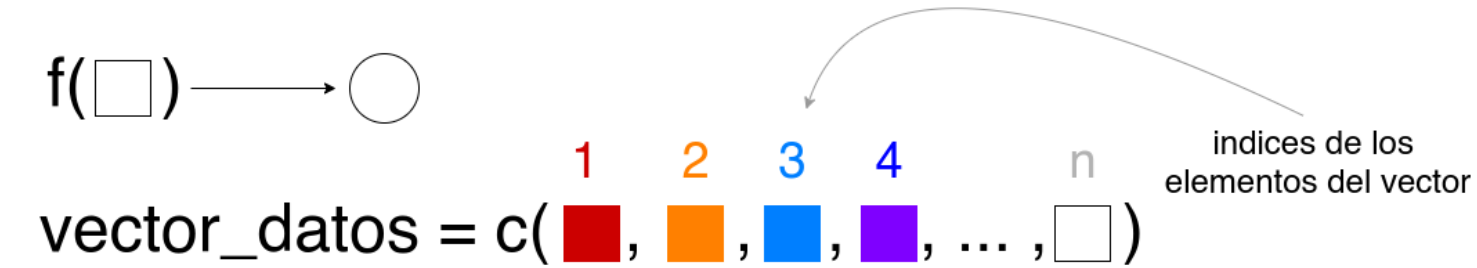
Sabemos que en R
vector_datos[1] =

Accedemos a los datos con su indice en el vector. → La variable que vamos a hacer ir cambiando en el loop es esa (indice).

vector con los numeros del 1 a n

```
for i in 1:length(vector_datos){



}
```

For loop (la logica)



Sabemos que en R
vector_datos[1] =

Accedemos a los datos con su indice en el vector. ⟶ La variable que vamos a hacer ir cambiando en el loop es esa (indice).

vector con los numeros del 1 a n

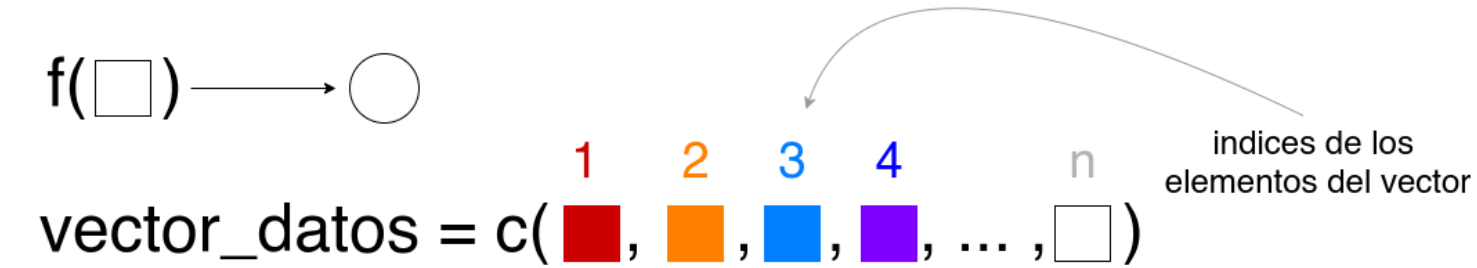
```
for i in 1:length(vector_datos){
```

al empezar el loop i = 1

vector_datos[1]

}

For loop (la logica)



Sabemos que en R
vector_datos[1] =

Accedemos a los datos con su indice en el vector. → La variable que vamos a hacer ir cambiando en el loop es esa (indice).

vector con los numeros del 1 a n

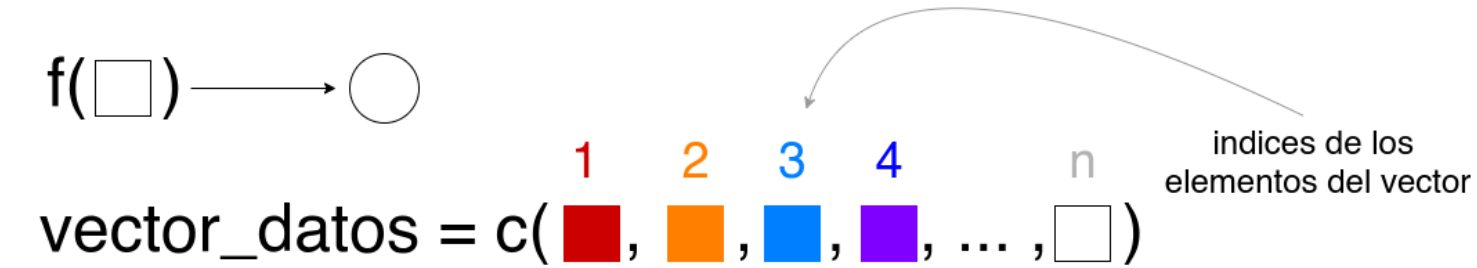
```
for i in 1:length(vector_datos){
```

al avanzar el loop i = 2

vector_datos[2]

}

For loop (la logica)



Sabemos que en R
vector_datos[1] =

Accedemos a los datos con su indice en el vector. → La variable que vamos a hacer ir cambiando en el loop es esa (indice).

vector con los numeros del 1 a n

```
for i in 1:length(vector_datos){
```

al avanzar el loop i = 3

vector_datos[3]

f( )

}

For loop (ejemplo con codigo)

```
numeros = c(3,40,15,6)
numeros_cuadrado = c()

for (i in 1:length(numeros)) {
  numeros_cuadrado[i] = numeros[i]^2
}
```

Funciones *apply(): otra forma de *loopear*

Son *funciones de alto rango*: uno de sus argumentos es otra funcion, la cual aplican sobre un objeto.

```
# definimos un vector
numeros = c(1,2,3,4)

# aplicamos una funcion anonima sobre este vector
numeros_cuadrado = sapply(X = numeros, FUN = function(x){x^2})

numeros_cuadrado
```

```
## [1]  1  4  9 16
```

Tenemos varias opciones, que dan diferentes clases de salidas: sapply (vectores), lapply (listas), mapply (matrices).

Comentarios, dudas existenciales?

A programar se ha dicho :)

- 10 minutos de pausa y volvemos.
- El practico de hoy esta en este link (clickear).