

Resumen

Cosas a tener en cuenta

- Cuidado con las traducciones de libro, biblioteca y generic (específica, != tipos datos)
- Explicar magnitudes y variables codificadas como:
`Sexo = c("H", "M")`
- Para ver donde apunta una gráfica, verla como una navaja. -> ¿por donde picha más?
hist permite saber si es asimétrica y para donde apunta

Cosas básicas

- Asignación: <-, =
- Operadores lógicos: !, &&, ||, <, >, <=, >=, ==
- nombres largos de variables con puntos

vectores

- 1:10
- todos mismo tipo
- empiezan por el índice 1 (no 0)
- las operaciones entre vectores se hacen componente a componente
- selección de componentes de un vector:
`x[i]`
`x[seq(1,10,by=2)]` # subconjunto de índices explícito
`x[y==2]` # subconjunto de índices lógicos
- cuidado con `vector(, 50)`. mejor `vector(mode="numeric", 50)`

matrices

- `matrix(1:10, 2, 5)`
- todos mismo tipo
- empiezan por el índice 1 (no 0)
- las operaciones entre matrices se hacen componente a componente
- selección de componentes de una matriz:
`matriz[i,j]` # componente (i,j)
`matriz[1,]` # 1ª fila
`matriz[,1]` # 1ª columna
- producto matricial: `%*%`

listas

- `list(A=1:10, B=2)`
- pueden ser distinto tipo
- nombrar siempre a las variables
- seleccion de componentes

```
lista$A # elemento con nombre A
lista[[1]] # primer elemento
```
- usar para devolver datos desde una función

dataframe

- `data.frame(Peso=c(50, 60), Altura=c(1.60, 1.70), Sexo=c("H","M"))`
- pueden ser de distinto tipo
- nombrar siempre a las variables
- selección de componentes

```
hoja$Peso
hoja[[1]]
```
- `I()` para identificadores como nombres

funciones

- `function(*args)`
- usar siempre `return` al final
- es recomendable sanitizar la entrada:

```
if(is.numeric(x) && !anyNA(x) && !any(is.infinite(x))){
...
}
```

estructuras de control:

- `if (condición) acción1 [else acción2]`
Estructura escalar. No usar condiciones que devuelvan más de un valor
Útil para comparar valores individuales
- `ifelse(condición, acción en caso cierto, acción en caso falso)`
Estructura vectorial. Se ejecuta el cuerpo por cada elemento de la condición.

```
ifelse(x==0), NA, 1/x) # x es un vector
```
- `switch(expresión,[valor-1]=acción-1,...,[valor-n]=acción-n)`
Si la condición es de carácter, usar `as.character(x)` para evitar que valores numéricos accedan al `switch`. Se suele dejar

la ultima acción sin condición.

- for (variable in valores) acción

Cuidado con los valores del tipo 2:0 = 2 1 0, no vacío!

- while (condición) acción
- repeat acción
- next / break

simular un experimento

```
Distribucion <- function(experimento, repeticiones = 5) {  
  return(replicate(repeticiones, experimento))  
}  
d = Distribucion(foo())$salida_i  
summary(d)  
hist(d)
```

gráficos

- plot(x, y, type, col="steelblue" xlim=c(.), ylim=c(.), xlab="", ylab="", main="")

Las coordenadas pueden expresarse mediante dos vectores, una matriz de dos columnas, una lista con dos componentes llamados x e y

Las siguientes funciones pintan elementos sobre un gráfico existente:

- polygon(x, y, border="")
- lines(x, y, type="l")
- points(x, y, type="p")

Para expresar las coordenadas es cómodo usar:

```
matrix(c(_, _, _, _, ...), ncol=2, byrow=T)
```

Lista funciones

- help()
- ls(..., all.names=T/F, ...)
- mean()
- c()
- vector(mode, length)
- length()
- search()
- matriz(data, nrow, ncol, byrow)
- t()
- crossprod()
- outer()
- solve()
- eigen()
- determinant()
- list()
- data.frame()
- I()

- `is()`: `is.numeric()`, `is.integer()`, `is.infinite()`
- `any()`: `anyNA()`
- `as()`: `as.character()`
- `apropos()`
- `summary()`
- `typeof()`
- `sin()`
- `sum()`
- `prod()`
- `function()`
- `var()`

`var()` es un estimador de varianza poblacional (dividido por $n-1$).

Para calcular la varianza muestral (dividido por n) hay tres formas:

```
# 1 forma
mean((x-mean(x))^2)
# 2 forma
((n-1)/n)*var(x)
# 3 forma
library(moments)
moment(x, 2, TRUE)
```

- `sd()`

Para calcular la desviación típica hay tres formas también:

```
# 1 forma
sqrt( mean((x-mean(x))^2) )
# 2 forma
((n-1)/n)*sd(x)
# 3 forma
library(moments)
sqrt(moment(x, 2, TRUE))
```

Para calcular los momentos centrales de orden n hay dos formas:

```
# 1 forma
mean((x-mean(x))^n)
# 2 forma
library(moments)
moment(x, n, TRUE)
```

- `cat()`
- `seq()`
- `sample(x, size, replace)`
`sample(40, 1000, T) # 1000 numeros entre el 1 y el 40`
- `.Random.seed`
- `Sys.time()`
- `plot()`
- `search()`
- `searchpaths()`
- `read.csv()`

- library(help)
- polygon()
- lines()
- points()
- runif()

Fragmentos reutilizables

Si tenemos una serie de datos, es común crear la matriz utilizando solo el número de columnas (igual al número de características):

```
Peso <- c(74, 47, 83)
Altura <- c(1.73, 1.55, 1.79)
matrix(c(Peso, Altura), ncol=2)
```

Si queremos escribir rápidamente una matriz dada:

```
3 2
1 -1

matrix(c(3, 2, 1, -1), ncol=2, byrow=T)
```

Ejemplo de función

```
name <- function(x = NA) {
  x=x[!is.na(x)]
  # ...
  return(x)
}
```

Ejemplo de función recurrente vectorial

```
name <- function(x, ...) {
  # estructura a desarrollar
  x=vector(mode="numeric", n)

  # modificar x con un bucle
  for (i in 1:n){
    x[i] = ...
  }

  # devolvemos el dato que nos interese
  return(x[n])
}
```

Calcular la distancia entre una serie de puntos

```
distancia.ORIGEN <- function(x,y){
  return(outer(x, y, function(x,y) sqrt(x^2+y^2)))
}
```

Medición de tiempos

```
tiempoinicial=Sys.time()
# lo que se quiere medir
```

```
tiempofinal=Sys.time()  
cat(tiempofinal - tiempoinicial)
```