# Fundamentos de programación en R: Conceptos generales

|  |  |
| --- | --- |
| Lo primero que tienes que hacer es instalar R  <https://cran.r-project.org/bin/windows/base/> | Y después, R Studio, que es donde vamos a programar.  <https://www.rstudio.com/products/RStudio/#Desktop> |

|  |  |
| --- | --- |
|  | El cuadro verde es el entorno en el que vamos a programar. Por norma general, este recuadro estará cerrado. Para abrirlo ve a file/ new file/ R Script.  El cuadro amarillo es la Console. Aquí aparecerán los resultados de la programación. Puedes escribir código directamente en la Console, pero no se guardará lo que hagas.  En el cuadro rojo podrás ver las variables y funciones que hayas creado, así como su valor actual.  En el cuadro azul podrás indicar dónde estás trabajando (fichero), recorrer los gráficos que hayas creado, buscar e instalar funciones de terceros o usar la ayuda de R Studio. |

Para empezar, sitúate en el Script (recuadro verde) y escribe

Bienvenida <- “Hola mundo”

**Conjunto de teclas útiles:**

Ctrl + intro ejecuta las líneas que tengas seleccionadas. Una sola (en la que estás) si no has seleccionado nada.

Ctrl + L borra los resultados de la consola.

Ctrl +1 o 2 cambia entre console y el script.

Ctrl + alt + R ejecuta todo el fichero.

Si estás escribiendo una Instrucción, el tabulador te ayudará a completarla, mostrándote las posibilidades existentes.

**Parar un programa / pedir ayuda**

|  |  |
| --- | --- |
|  | **¿Cómo puedo detener la ejecución de un programa?**  En ocasiones, puedes querer detener la ejecución de un programa. Ya sea porque has entrado en un bucle infinito, o porque el programa está tardando demasiado en arrojar un resultado.  Para detener la ejecución basta con pulsar el botón rojo que aparece en la esquina superior derecha de la console.  Ojo, este botón aparece únicamente cuando se está ejecutando un programa. |
|  | **No sé hacer algo, ¿qué puedo hacer?**  Acude al botón de ayuda y escribe allí tu pregunta. Obtendrás documentación muy completa sobre la función, argumentos y ejemplos prácticos.  También puedes escribir en la Console help(función/ duda). Se te mostrará la ayuda en la misma ventana de abajo a la derecha.  Si no encuentras lo que buscas, en internet está todo. Para buscar algo de R escribe: [R] duda.  Probablemente, lo que estás intentando programar ya lo ha hecho alguien anteriormente. Busca en Cran algún paquete que haga lo que quieres. Cran es un repositorio de funciones.  Si has localizado una función que hace lo que quieres, pero quieres profundizar en su código, escribe en la Console el nombre de la función, pero sin los paréntesis. Se abrirá el código de la misma para que puedas inspeccionarlo y seleccionar lo que quieras. |

**Operadores básicos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| + suma | - resta | \* multiplicar | / dividir | ^ o \*\* potencia | %% resto | %/% división entera |
| < menor | > mayor | <= menor o igual | >= mayor o igual | != Distinto | == igual |  |

Asignar un valor a una variable

n <- 15

22 -> h

El punto y coma separa comandos en una misma línea. Es decir n<-15; 22->h

#Para realizar comentarios hay que usar la almohadilla.

Frase <- "Hola mundo" # para escribir una frase debemos ponerla entre comillas.

Frase<- "Para poder escribir comillas en mitad de una frase debemos usar \" así podemos no terminará la frase"

Cuidado cuando uses acentos en el código o la letra ñ. Lo más probable es que al volver a abrir el archivo, R no sabrá cómo interpretar esos caracteres. Para evitar tener problemas, guarda siempre todo lo que hagas con la codificación UTF-8. Y cuando abras un Script que no haya sabido interpretar los acentos o las ñ, ve a archivo, reopen with encoding, y selecciona UTF-8.

|  |  |
| --- | --- |
|  |  |

Limpiar entorno de trabajo. R guarda una imagen del proyecto con el que estás trabajando, así como de las variables y sus valores. Es muy probable que cuando vuelvas a abrir R te encuentres todo tal y como lo dejaste. Lo cual no es siempre deseable (especialmente cuando cambiamos de proyecto y solemos trabajar con el mismo nombre de variables).

rm(x,y) # borra los objetos x e y de la memoria.

rm(list=ls()) # borra todos los objetos que estén definidos. Es muy buena práctica escribir esta instrucción al inicio de los programas.

|  |  |
| --- | --- |
|  | Otra manera de hacerlo es usar la brocha que aparece en el recuadro donde se guardan los objetos declarados (y sus valores). |

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **Creación y asignación de una variable**  edades <- c(43, 42, 12, 8, 5)  **Para ver el contenido de una variable**  edades  **Algunas funciones**  sum(edades)  mean(edades)  range(edades)  **Operaciones básicas**  1+2+3  1+2\*3  (1+2)\*5  **Operadores relacionales**  TRUE == TRUE  TRUE == FALSE  "hello" == "goodbye"  3 == 2  TRUE != TRUE  TRUE != FALSE  "hello" != "goodbye"  3 != 2  5 >= 3  3 >= 3  **Operadores relacionales sobre vectores**  a <- c(16, 9, 13, 5, 2, 17, 14)  a  a > 10  b <- c(17, 7, 5, 16, 8, 13, 14)  a <= b  **Operadores lógicos**  x <- 12  x > 5 & x < 15  y <- 4  y < 5 | y > 15  !(x < 5) | **Operadores aritméticos**  # Suma 5 +5 y divide el resultado entre 2  (5 + 5) / 2  # Calcula 2 elevado a 5  2^5  # Calcula el resto de 28 entre 6  28%%6  **Asignación de variables**  # Asigna el valor 5 a la variable my\_apples  my\_apples <- 5  # Muestra el contenido de la variable my\_apples  my\_apples  # Asigna el valor 6 a la variable my\_pears  my\_pears <- 6  # Muestra el contenido de la variable my\_pears  my\_pears  # Suma peras con manzanas: crea una nueva variable my\_fruit con la suma  my\_fruit <- my\_apples + my\_pears  # Muestra el número total de piezas de fruta  my\_fruit  **Tipos básicos**  # Declara una variable numérica, my\_numeric, con el valor 42  my\_numeric <- 42  # Declara una variable de tipo texto, my\_character, con el valor "forty-two"  my\_character <- "forty-two"  # Declara una variable booleana, my\_logical, con el valor FALSE  my\_logical <- FALSE |

# Sentencias y bucles

**Sentencia IF**

Ejecuta un código u otro en función de si se cumple o no una condición. Esta estructura es muy útil a la hora de programar, ya que diriges el código en una u otra dirección.

x <- 5

if (x < 0) {

print("x es un número negativo")

} else if (x == 0) { #Fíjate que para poner igual tiene que poner dos (uno solo asigna valor, dos compara).

print("x es cero")

} else {

print("x es un número positivo o cero")

}

Se puede escribir la sentencia en una única línea, exactamente igual que en excel (condición, true, false).

ifelse(x > 0, "x es número positivo", "x es un número negativo")

También es posible poner más de una condición,

if (condicion1 && condicion2 && condicion3){

… }

&& es un And: todas las condiciones deben cumplirse.

|| es un Or: al menos una de las condiciones debe cumplirse. El símbolo se saca con Alt Gr + nº 1

**Bucle FOR**

Utilizamos el bucle for para repetir un código un número determinado de veces.

A diferencia de otros lenguajes, como VBA, el bucle FOR no tiene porqué iterar sobre una secuencia de números (ej de 1 a 50), sino que puede iterar sobre los elementos de una lista, o un vector.

for (año in 1:10){ # Este sería un FOR típico, igual a como lo haríamos en VBA.

print (año)

}

cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town") # Creamos un vector.

for (city in cities) { # Recorremos los elementos del vector.

if (nchar(city) == 6) # Si el número de letras del vector es igual a 6…

next # Ignoramos esa iteración.

print(city) # “Imprimimos el resultado”

}

for (elemento in 1:length(cities)) { # Recorremos el vector cities desde su posición 1 hasta el último elemento.

print(paste(cities[elemento], "está en la posición", elemento, "del vector de ciudades."))

} # paste se usa para unir diferentes elementos en una sola frase.

Next: Es una instrucción que anula la iteración en la que se encuentra (y solo esa).

Break: Es una instrucción que nos permite terminar el bucle antes de tiempo.

**Bucle While**

Utilizamos el bucle While para repetir un número de veces indeterminado un código en concreto. La repetición parará cuando se cumpla la condición que le indiquemos.

variable <- 1

while (variable <= 7) { # Repetir hasta que la variable sea mayor o igual a 7.

if (variable == 5){

break # Sal del bucle.

}

print(paste("variable vale", variable))

variable <- variable + 1

}

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **Sentencias condicionales**  x <- -3  if (x < 0) {  print("x es un número negativo")  }  x <- 5  if (x < 0) {  print("x es un número negativo")  } else {  print("x es un número positivo o cero")  }  x <- 5  if (x < 0) {  print("x es un número negativo")  } else if (x == 0) {  print("x es cero")  } else {  print("x es un número positivo o cero")  }  ifelse(x > 0, "x es número positivo", "x es un número negativo")  **Bucles**  ctr <- 1  while (ctr <= 7) {  print(paste("ctr vale", ctr))  ctr <- ctr + 1  }  variable <- 1  while (variable <= 7) {  if (variable == 5){  break  }  print(paste("variable vale", variable))  variable <- variable + 1  }  cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")  for (city in cities) {  print(city)  }  cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")  for (city in cities) {  if (nchar(city) == 6){  next  }  print(city)  }  cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")  for (i in 1:length(cities)) {  print(paste(cities[i], "está en la posición", i, "del vector de ciudades."))  } | **Recorre los elementos del vector y extrae los que empiecen por C**  numeros<-c("uno","dos","tres","cuatro","cinco","treinta")  empiezan\_por\_c<-c()  for (elemento in numeros){  if (substr(elemento,1,1)=="c"){  # Extrae una letra empezando por la 1ª y acabando por la 1ª  empiezan\_por\_c<-c(empiezan\_por\_c,elemento)  }  }  empiezan\_por\_c  **Escribe un programa que encuentre todos los enteros entre 2000 y 3200 (ambos incluidos) que sean divisibles por 7 pero no múltiplos de 5**  numeros\_que\_cumplen<-c()  for (numero in 2000:3200){  if (numero %% 7 ==0){  if (numero %% 5 !=0){  numeros\_que\_cumplen<-c(numeros\_que\_cumplen,numero)  }  }  }  numeros\_que\_cumplen  **Escribe un programa que calcule el número de vocales en la frase "En un lugar de la Mancha, de cuyo nombre no quiero acordarme"**  frase<-"En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  frase<-tolower(frase)  num\_vocales<-0  for (letra in 1:nchar(frase)){  if (substr(frase,letra,letra) == "a" || substr(frase,letra,letra) == "e" || substr(frase,letra,letra) == "i" || substr(frase,letra,letra) == "o" || substr(frase,letra,letra) == "u"){  num\_vocales<-num\_vocales+1  }  }  num\_vocales  Otra manera de hacerlo sería simplificando el IF.  frase<-"En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  vocales<-c("a","e","i","o","u")  frase<-tolower(frase)  num\_vocales<-0  for (letra in 1:nchar(frase)){  if (substr(frase,letra,letra) %in% vocales){  num\_vocales<-num\_vocales+1  }  }  num\_vocales |

# Crear e invocar funciones

Una función es un conjunto de líneas de código que se ejecutará cada vez que la invoquemos.

¿Cuándo debo hacer una función? Cuando el código vaya a ejecutarse más de una vez en el programa, deberías encapsularlo en una función.

do\_something <- function(a, b = 1) { # La función recibe dos parámetros, pero solo el 1º es obligatorio.

return (a \* b + a / b) # La función devuelve el parámetro return o la última instrucción ejecutada.

} # Ojo, solo devuelve un parámetro (aunque puede ser una lista).

¿Cómo Invocamos una función?

do\_something (4) # Llamamos a la función y le pasamos el parámetro obligatorio.

k <- do\_something (4, 3) # Llamamos a la función, le pasamos los parámetros y guardamos el rdo en una variable.

k <- do\_something (a=4, b=3) # Igual que antes, pero le pasamos los parámetros desordenados.

Funcion2 <- function(x) return(x^2) # Podemos construir una función en una sola línea.

Ojo, si ejecutamos una función, esta se declarará en la zona de las variables (estará cargada en memoria). Si cambiamos el código de la función deberemos volver a ejecutarla antes de llamarla. Si no lo hicieras, estarías invocando a la función anterior.

Entorno de función: Cuando creamos una función estamos creando un nuevo entorno. Esto significa que una variable definida dentro de una función no estará disponible fuera de la función (en entornos superiores, aunque sí en inferiores).

R pasa los argumentos por valor. Es decir, una función no puede alterar el valor de la variable que se pasa como argumento en la llamada (genera copias locales).

<<- Si asignamos así el valor dentro de una función estamos modificando el valor de la variable a nivel global (dentro y fuera de la función). Es posible hacerlo, pero se recomienda no usarlo.

**Sentencia Switch**

La sentencia Switch se usa para seleccionar elementos dentro de una lista de alternativas.

El primer argumento que le pasamos es el elemento que deberá buscar dentro de la lista, devolviéndonos el valor asociado a dicho elemento.

Un ejemplo práctico sería construir la siguiente función.

centre<-function(x,type){ # Creamos una función que recibe dos parámetros.

switch(type, mean=mean(x), media=median(x)) # Usamos un switch para indicar qué queremos hacer.

}

x <- c(10,5,25) # Creamos un vector con varios elementos.

centre(x, "mean") # Invocamos a la función, pasándole los parámetros.

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **Ejemplo de función sencilla**  do\_something <- function(a, b = 1) {  return (a \* b + a / b)  }  k <- do\_something (a=4, b=3  **Función que calcula la media o mediana, con los datos que le pases.**  centre<-function(x,type){  switch(type, mean=mean(x), media=median(x))  }  x <- c(10,5,25)  centre(x,"mean") | **Escribe una función a la que se le pase una vector de números y calcule su media.**  media\_numeros<-function(numeros){  if (length(numeros) == 0){  return ("Has pasado una lista vacía")  }  return(sum(numeros)/length(numeros))  }  numeros<-c(2,4,6,8,10)  numeros<-c()  media\_numeros(numeros)  **Escribe una función que reconozca palíndromos. Un palíndromo es una palabra que se lee igual al derecho que al revés.**  palindromo<-function(palabra){  palabra<-tolower(palabra)  capicua<-TRUE  for (letra in 1:nchar(palabra)){  if (substr(palabra,letra,letra)!=substr(palabra,nchar(palabra)-letra+1,nchar(palabra)-letra+1)){  capicua<-FALSE  break  }  }  if (capicua==TRUE){  return("Es un palíndromo")  } else {  return ("No es un palíndromo")  }  }  palabra<-"Anna"  palindromo(palabra)  # Una manera fácil de hacerlo.  library("miscset")  palindromo<-function(palabra){  palabra<-tolower(palabra)  if (palabra == str\_rev (palabra)){  return ("Es un palíndromo")  } else {  return ("No es un palíndromo")  }  }  palabra<-"Anna"  palindromo(palabra)  **Escribe una función que calcule el IVA (21%) de un producto dado su precio de venta sin IVA y devuelva el precio total.**  precio\_total<-function(precio\_sin\_iva){  return(precio\_sin\_iva\*1.21)  }  precio\_sin\_iva<-100  precio\_total(precio\_sin\_iva) |

# Vectores

Los vectores únicamente tienen una dimensión. Van desde 1 hasta la longitud del vector length(v).

Todos los elementos del vector deben ser del mismo tipo (logical, numeric, carácter…)

Tiene un tamaño fijo que es fijado en su creación (aunque puede modificarse sustituyendo el vector al completo).

**Creación de vectores**

X <- 1:30 genera un vector de nº de 1 a 30. El operador “:” tiene prioridad sobre otros operadores (sumar, dividir…)

1:10-1 genera un vector de 0 a 9

1:(10-1) genera un vector de 1 a 9

Seq(1, 5, 0.5) # genera un vector de 1.0 a 5.0 con incrementos de 0.5

C(1, 2, 3…) # permite asignar valores a un vector directamente.

Rep(1, 30) # crea un vector de 30 valores iguales de valor 1.

v5 <- seq(from = 0, to = 1, by = 0.1) # Asignamos el vector a una variable.

v11 <- c(a = 1, b = 2, c = 3) # Ponemos nombres a los valores del vector.

**Operaciones con vectores**

total\_vector <- v5 + v11

Ojo, si los vectores no tienen el mismo tamaño, R repite el menor de ellos tantas veces como sea necesario.

x<-1:4

y<-1:2

z<-x+y # da 2 4 4 6

|  |  |
| --- | --- |
| sum(x) # suma de elementos de x  prod(x)  max()  min()  mean() | median()  var(x,y) # Varianza  cov() # Covarianza  cor() # Correlación |

**Indexación de vectores (recuperar valores)**

v=1:100

v[1] # Seleccionamos / recuperamos el primer valor del vector.

v[c(1, 1, 4, 5)] # Seleccionamos el primero dos veces, el cuarto y el quinto

v[-1] # Seleccionamos todos menos el primer elemento.

v[-length(v)] # Seleccionamos todos menos el último elemento.

v[-c(1, 3, 4, 5)] # Seleccionamos todos menos el primero, el tercero, el cuarto y el quinto

v > 30 # Construyendo un índice booleano

v[v > 30] # Aplicando un índice booleano

which(v > 30) # Localizando posiciones a través de un índice booleano

v[which(v > 30)] # Seleccionamos los elementos por posiciones

v[v > 30 & v <= 50] # Seleccionamos todos los elementos cuyo valor sea > 30 y <= 50

v[v == 0] # Seleccionamos todos los elementos cuyo valor sea 0

v[v %in% c(10, 20, 30)] # Seleccionamos los elementos cuyo valor sea 10, 20 y 30

v[70:100] <- 0 # Asignamos el valor cero a los elementos entre los índices 70 y 100

v0 = v[1:5]

names(v0) <- c("SAN", "TEL", "ACC", "IAG", "MAP") # ponemos nombres a los valores del vector v0

v0[c("SAN", "IAG")] # Podemos seleccionar los elementos por su posición, por su valor o por su nombre…

v0[!(names(v0) %in% c("TEL", "MAP"))] #devuelve el nombre y valor de los elementos de v0 que no son alfa o beta

**Operadores relacionales**

& Compara todo el vector.

&& Compara solo el primer elemento del vector.

| “Or”, compara todo el vector.

|| Compara solo el primer elemento.

identical(x,y) # Da true (compara vectores).

**Ejemplos Ejercicios**

|  |  |  |  |
| --- | --- | --- | --- |
| **Vectores: Creación**  # Creación de vectores de longitud fija  v1 <- vector(mode = 'logical', length = 4)  v2 <- vector(mode = 'integer', length = 4)  # Usando el operador de secuencia  v3 <- 1:5  v4 <- 1.4:5.4  v5 <- seq(from = 0, to = 1, by = 0.1)  # Usando la función de combinación  v6 <- c(TRUE, FALSE)  v7 <- c(1.3, 7, 7/20)  v8 <- c('black', 'white')  v9 <- c(v1, v3)  # Creación de vector nombres  v10 <- c(a = 1, b = 2, c = 3)  **Vectores: Operaciones**  a\_vector <- c(1, 2, 3)  b\_vector <- c(4, 5, 6)  total\_vector <- a\_vector + b\_vector  total\_vector <- a\_vector + 1  sum(total\_vector)  max(total\_vector)  mean(total\_vector)  **Indexando con números positivos**  v <- 1:100  v[1] # Seleccionamos el primer elemento  v[c(1, 1, 4, 5)] # Seleccionamos el 1º dos veces, el 4º y el 5º  v[20:30] # Obtenemos los elementos entre el índice 20 y 30  v[70:100] <- 0 # Asignamos cero a los elementos entre 70 y 100  v[which(v > 30)] # Seleccionamos las posiciones de los elementos > 30  **Indexando con números negativos**  v[-1] # Seleccionamos todos menos el primer elemento  v[-c(1, 3, 4, 5)] # Seleccionamos todos menos el 1º, el 3º, el 4º y el 5º  v[-length(v)] # Todos menos el último  **Indexando con vectores lógicos o expresiones booleanas**  v0 <- v[1:5]  v0[c(TRUE, FALSE, TRUE, FALSE, FALSE)] # Seleccionamos el 1º y el 3º  v[v > 30] # Todos los > 30  v[v > 30 & v <= 50] # Todos los > 30 y <= 50  v[v == 0] # Todos los 0  v[v %in% c(10, 20, 30)] # Seleccionamos el 10, 20 y 30  **Indexando por nombre**  names(v0) <- c("alpha", "beta", "gamma", "delta", "omega")  v0["alpha"]  v0["beta"] <- 500  v0[c("delta", "omega")]  v0[!(names(v0) %in% c("alpha", "beta"))] | **Crea dos vectores con las ganancias y pérdidas de la semana.**   |  |  | | --- | --- | | poker\_vector  # On Monday you won 140$  # Tuesday you lost 50$  # Wednesday you won 20$  # Thursday you lost 120$  # Friday you won 240$ | roulette\_vector  # On Monday you lost 24$  # Tuesday you lost 50$  # Wednesday you won 100$  # Thursday you lost 350$  # Friday you won 10$ |   **Crea un vector con las ganancias en el póker de lunes a viernes**  poker\_vector <- c(140, -50, 20, -120, 240)  **Crea un vector con las ganancias en la ruleta de lunes a viernes**  roulette\_vector <- c(-24, -50, 100, -350, 10)  **Crea un vector con los días de la semana y asígnalo como nombre a los elementos de los vectores anteriores.**  days <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")  names(poker\_vector) <- days  names(roulette\_vector) <- days  **Calcula el total de ganancias diarias.**  total\_daily <- poker\_vector + roulette\_vector  **Calcula el total de ganancias en el póker.**  total\_poker <- sum(poker\_vector)  **Calcula el total de ganancias en la ruleta.**  total\_roulette <- sum(roulette\_vector)  **Calcula el total de la semana.**  total\_week <- total\_poker + total\_roulette  **Selecciona las ganancias del miércoles en el póker.**  poker\_wednesday <- poker\_vector[3]  poker\_wednesday <- poker\_vector["Wednesday"]  **Selecciona las ganancias del martes, miércoles y jueves en el póker.**  poker\_midweek <- poker\_vector[c(2, 3, 4)]  poker\_midweek <- poker\_vector[c("Tuesday", "Wednesday", "Thursday")]  **Selecciona las ganancias desde el martes al viernes en la ruleta.**  roulette\_selection\_vector <- roulette\_vector[2:5]  **Calcula la media de ganancias en el póker el lunes, martes y miércoles.**  P\_LM= poker\_vector[1:3]  P\_LM\_G = P\_LM[P\_LM>0]  Average = mean(P\_LM\_G)  # En una línea  average <- mean(poker\_vector[poker\_vector[c("Monday", "Tuesday", "Wednesday")]>0])  **¿Qué días de la semana hubo ganancias al póker?**  selection\_vector <- poker\_vector > 0  **Haz la selección anterior sobre el vector con los datos del póker.**  poker\_winning\_days <- poker\_vector[selection\_vector]  poker\_winning\_days <- poker\_vector[poker\_vector > 0]  **Haz lo mismo sobre el vector con los datos de la ruleta.**  roulette\_winning\_days <- roulette\_vector[roulette\_vector > 0] |

# Matrices

Las matrices son arrays de dos dimensiones, las cuales contienen elementos del mismo tipo.

La manera más simple de crear una matriz es usando matrix(vector datos, nº filas, nº columnas, rellenar por filas = TRUE)

**Creación de matrices**

m1 <- matrix(1:9, nrow = 3, byrow = TRUE) #rellena por filas

m2 <- matrix(c(0, -1, 4)) # Crea una matriz con una columna

m <- matrix(c(vector1, vector2, vector3), byrow=TRUE, nrow=3) # Ojo, los vectores deben tener la misma longitud.

m1<- matrix(1, nr = 2, nc = 2) # Creamos una matriz de unos.

m2<- matrix(2, nr = 2, nc = 2) # Creamos una matriz de doses.

**Indexar matrices**

Matriz[filas,columnas] Para seleccionar se usan corchetes y hay que indicar dos dimensiones.

m[1:2, ] # Seleccionamos las dos primeras filas y todas las columnas.

m[, c(1, 3)] # Seleccionamos todas las filas, la primera y tercer columna.

m[-1, ] # Seleccionamos todas las filas menos la primera

# Poniendo nombres a las filas y columnas, e indexando por ellos.

colnames(m) <- c("c1", "c2", "c3")

rownames(m) <- c("r1", "r2", "r3")

m[c("r2", "r3"), c("c1", "c2")] # Selección de filas y columnas por nombre

m[-nrow(m), -ncol(m)] # Quitamos la Última fila y la Última columna (selecciono todos los valores menos estos).

¿Hemos eliminado los valores de esta manera? No, únicamente hemos realizado una selección.

¿Cómo podemos eliminar los valores? m<- m[-nrow(m), -ncol(m)]

m[1, ] <- 0 # Asigna un vector de ceros a la primera fila. Cambiamos los valores.

m[m > 7] # Selecciono todos los valores > 7. Fíjate que estamos usando una máscara booleana.

Si escribiese m>7 me devolvería una matriz de True y False…

m[m == 0] # Selecciono todos los valores iguales a 0.

**Manipular matrices**

matrix2 <- cbind(a\_matrix, b\_matrix) # Unión de matrices por columnas.

matrix2 <- cbind(matrix2, c(1, 5, 6)) # Unión de una matriz con un vector por columnas.

matrix1 <- rbind(a\_matrix, b\_matrix) # Unión de matrices por filas.

matrix1 <- rbind(matrix1, c(1, 5, 6)) # Unión de una matriz y un vector por filas.

**Operaciones con matrices**

d <- det(m1) # Determinante de la matriz

Matriz\_resultante <- matriz + 2; o bien matriz – 5; matriz \*3; matriz / 7

rowSums(total\_matrix)

colMeans(total\_matrix)

max(matrix)

min(matrix)

Media <- mean(matriz[, 1]) # Calcula la media de la primera columna.

Suma y resta de matrices: Deben de tener la misma dimensión

Matriz\_resultante = matriz1 + matriz2

Producto de matrices:

Multiplicación elemento a elemento: Matriz\_resultante <- matriz1 \* matriz2

Ambas matrices deben de ser de igual dimensión.

Multiplicación matricial: Matriz\_resultante <- matriz1 %\*% matriz2

t(matriz) transpone una matriz.

diag(matriz) extrae la diagonal.

solve(matriz) invierte una matriz.

**Generar una matriz de números aleatorios**

Matriz\_aleatoria <- matrix(sample.int(50, replace = FALSE), 5, 5) # Genera una matriz de 25 elementos distribuidos en 5 filas y 5 columnas. Los números aleatorios son entre 1 y 50, sin reemplazamiento y con distribución uniforme.

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **Matrices: Creación**  m1 <- matrix(1:9, byrow = TRUE, nrow = 3)  m2 <- matrix(c(0, -1, 4)) # Crea una matriz con una columna (xq si no indicas nada, se rellena por filas).  d1 <- diag(3) # Crea una matriz diagonal 3x3  d2 <- diag(c(1, 2, 3)) # Crea matriz diagonal, asigna vector a la diagonal.  t\_m1 <- t(m1) # Traspuesta de m1  d <- det(m1) # Determinante de la matriz  **Matrices: Operaciones**  a\_matrix <- matrix(1:9, byrow = TRUE, nrow = 3)  b\_matrix <- matrix(11:19, byrow = TRUE, nrow = 3)  total\_matrix <- a\_matrix + b\_matrix  total\_matrix <- a\_matrix + 2  rowSums(total\_matrix)  colMeans(total\_matrix)  max(total\_matrix)  **Unión de matrices por columnas**  big\_matrix\_2 <- cbind(a\_matrix, b\_matrix)  **Unión de matriz y vector por columnas**  big\_matrix\_2 <- cbind(big\_matrix\_2, c(1, 5, 6))  **Unión de matrices por filas**  big\_matrix\_1 <- rbind(a\_matrix, b\_matrix)  **Unión de matriz y vector por filas**  big\_matrix\_1 <- rbind(big\_matrix\_1, c(1, 5, 6))  **Indexando con números positivos**  m <- matrix(1:9, byrow = TRUE, nrow = 3)  m[1, ] # Seleccionamos la primera fila  m[1:2, ] # Seleccionamos las dos primeras filas  m[, 3] # Seleccionamos la última columna  m[, c(1, 3)] # Seleccionamos la primera y la última columna  m[1, ] <- 0 # Asigna un vector de ceros a la primera fila  **Indexando con números negativos**  m[-1, ] # Seleccionamos todas las filas menos la primera  m[-nrow(m), -ncol(m)] # Quitamos la última fila y la última columna  **Indexando con vectores lógicos o expresiones booleanas**  m\_selection <- matrix(c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, FALSE, TRUE), byrow = TRUE, nrow = 3)  m[m\_selection] # Indexamos usando la selección anterior.  m[m > 7] # Todos los > 7  m[m == 0] # Todos los 0  **Indexando por nombre**  colnames(m) <- c("c1", "c2", "c3")  rownames(m) <- c("r1", "r2", "r3")  m[, c("c1", "c3")] # Selección de columnas por nombre.  m[c("r2", "r3"), c("c1", "c2")] # Selección de F y C por nombre. | Los siguientes vectores contienen la recaudación de las tres primeras películas de Star Wars en US y fuera de US (non-US)  nueva\_esperanza <- c(461, 314.4)  imperio\_contrataca<- c(290.5, 247.9)  retorno\_jedi <- c(309.3, 165.8)  **Crea una matriz que contenga toda la información (con tres filas)**  star\_wars <- matrix(c(nueva\_esperanza, imperio\_contrataca, retorno\_jedi), byrow=TRUE, nrow=3)  **Ponles nombres a las columnas: "US" y "non-US"**  colnames(star\_wars) <- c("US", "non-US")  **Ponles nombres a las filas: "Una nueva esperanza", "El imperio contraataca" y "El retorno del jedi"**  rownames(star\_wars) <- c("Una nueva esperanza ", " El imperio contraataca ", " El retorno del jedi ")  **Si el precio de la entrada es de 5$, estima el número de espectadores de cada película.**  visitors <- star\_wars / 5  **Como el precio de las entradas no es el mismo todos los años, creamos una matriz de precios**  ticket\_prices\_matrix <- matrix(c(5, 5, 6, 6, 7, 7), nrow = 3, byrow = TRUE, dimnames = list(rownames(star\_wars), colnames(star\_wars)))  **Repite el cálculo del número de espectadores con la matriz anterior.**  visitors <- star\_wars / ticket\_prices\_matrix  **Calcula el número de espectadores medio en US**  average\_us\_visitors <- mean(visitors[, 1])  **Calcula el número de espectadores medio fuera de US**  average\_non\_us\_visitors <- mean(visitors[, 2])  **Calcula los totales de recaudación por película**  worldwide\_vector <- rowSums(star\_wars)  **Añade el vector anterior como una nueva columna de la matriz star\_wars.**  all\_wars\_matrix <- cbind(star\_wars, worldwide\_vector)  **Crea una nueva matriz con las recaudaciones de las siguientes tres películas**.  amenaza\_fantasma <- c(474.5, 552.5)  ataque\_clon<- c(310.7, 338.7)  venganza\_sith <- c(380.3, 468.5)  star\_wars2 <- matrix(c(amenaza\_fantasma, ataque\_clon, venganza\_sith), byrow=TRUE, nrow=3)  **Ponles nombres a las columnas: "US" y "non-US"**  colnames(star\_wars2) <- c("US", "non-US")  **Ponles nombres a las filas: "Amenaza fantasma", "Ataque de los clones" y "Venganza Sith"**  rownames(star\_wars2) <- c("Amenaza fantasma ", " Ataque de los clones ", " Venganza Sith ")  **Une en una nueva matriz la recaudación de todas las películas, las tres primeras filas corresponderán a las tres primeras películas y las tres siguientes a las últimas películas.**  all\_wars\_matrix <- rbind(star\_wars, star\_wars2)  **Calcula los totales de recaudación de todas las películas en US y fuera de US**  total\_revenue\_vector <- colSums(all\_wars\_matrix)  **Calcula la media recaudada de las tres primeras películas fuera de US**  non\_us\_all <- mean(star\_wars[, 2])  **Calcula la media recaudada de las 2 primeras películas fuera de US**  non\_us\_some <- mean(star\_wars[1:2, 2]) |

# Array

Si quisiésemos crear un array de más de dos dimensiones usaríamos la función array(datos,dim). Datos es un vector con los valores que tendrá el array y dim es un vector con las dimensiones del array.

**Ejemplos**

|  |  |
| --- | --- |
| a <- array(1:8, dim=c(2, 2, 2))  m <- array(1:9, dim=c(3, 3)) |  |

# Factores

Un factor es una estructura que almacena variables categóricas.

Es decir, almacena variables que pueden tomar un nº limitado de valores (niveles). Por ejemplo, la nacionalidad.

A los distintos valores que puede tomar la variable se les denomina niveles.

La utilidad de un factor es que permite establecer un orden distinto al alfabético.

No es posible realizar operaciones aritméticas con factores, aunque sí booleanas.

**Ejemplos Ejercicios**

|  |  |
| --- | --- |
| **Creación de factor sin orden**  gender\_vector <- c('M', 'F', 'F', 'M', 'M', 'F')  gender\_factor <- factor(gender\_vector)  **Creación de factor con orden (sin especificar qué orden)**  size\_vector <- c('S', 'L', 'M', 'L', 'S', 'M')  size\_factor <- factor(size\_vector, ordered = TRUE) # L < M < S  **Creación de factor con orden (especificando el orden)**  size\_vector\_2 <- c('S', 'L', 'M', 'L', 'S', 'M')  size\_factor\_2 <- factor(size\_vector\_2, ordered = TRUE, levels = c("S", "M", "L")) # S < M < L  **Factores: Operaciones**  Comprobaciones en factores sin orden solo se puede usar ==  gender\_factor[1] == gender\_factor[2]  gender\_factor[1] == size\_factor[2] # Da ERROR: solo se pueden comparar factores si son del mismo tipo.  En factores con orden se puede usar >, < …  size\_factor[1] > size\_factor[2] # Da True.  **Obtener los niveles**  levels(size\_factor)  levels(size\_factor)[1]  **Comprobar la existencia de niveles**  any(levels(size\_factor) %in% c('L', 'S'))  **Añadir nuevos niveles**  levels(size\_factor)[length(levels(size\_factor)) + 1] <- 'XL' levels(size\_factor) <- c(levels(size\_factor), 'XS')  # En ambos casos, añadimos un nivel nuevo al final del todo.  **Reordenar niveles**  size\_factor <- factor(size\_factor, ordered = TRUE, levels(size\_factor) [c(5, 3:1, 4)])  **Cambiar/re-nombrar niveles**  levels(size\_factor)[5] <- 'Extra L'  **Eliminar niveles no utilizados**  droplevels(size\_factor) # No funciona  size\_factor<-as.character(size\_factor)  as.numeric(as.character(size\_factor)) # con datos numéricos.  **Unir factores**  a <- factor(1:10)  b <- factor(letters[a])  union <- factor(c(as.character(a), as.character(b))) # Une factores.  cross <- interaction(a, b) # Saca todas las combinaciones posibles 1A 2A 3A… 1B 2B 3B… 1C… | **Crea dos factores con los siguientes vectores**  1. Animales sin orden  2. Temperatura con orden  animals\_vector <- c("Elephant", "Giraffe", "Donkey", "Horse")  temperature\_vector <- c("High", "Low", "High","Low", "Medium")  factor\_animals\_vector <- factor(animals\_vector)  factor\_temperature\_vector <- factor(temperature\_vector, ordered = TRUE, levels = c("Low", "Medium", "High"))  **Modifica los niveles del factor para que sean: "Female" y "Male"**  survey\_vector <- c("M", "F", "F", "M", "M")  factor\_survey\_vector <- factor(survey\_vector)  levels(factor\_survey\_vector) <- c("Female", "Male")  **Crea un factor ordenado para el siguiente vector**  speed\_vector <- c("Fast", "Slow", "Slow", "Fast", "Ultra-fast")  factor\_speed\_vector <- factor(speed\_vector, ordered = TRUE, levels = c("Slow", "Fast", "Ultra-fast")) |

# Listas

Una lista es una colección de diferentes tipos de objetos.

Estos objetos pueden ser vectores, matrices, dataframes y otras listas.

Al igual que en los vectores, el índice de la lista va desde 1 hasta length().

No es posible realizar operaciones aritméticas sobre los elementos de una lista.

En ocasiones se puede simplificar una lista, convirtiéndola en un vector mediante la función unlist()

**Creación de listas**

lista <- list(my\_vector, my\_matrix, my\_df) # Creamos una lista.

lista2 <- list(vec = my\_vector, mat = my\_matrix, df = my\_df) # Otra manera de hacer una lista.

lista3 <- c(lista1, lista2) # Podemos crear una lista uniendo otras dos.

**Manipulación de listas**

lista <- c(lista, year = 1980) # Añade un nuevo elemento a lista (year = 1980).

lista[[length(lista)+1]]<-nuevo\_elemento # Otra manera de añadir un elemento.

lista<- append(lista, nuevo\_valor) # Añade un elemento. El parámetro after especifica la posición (por defecto length(x)).

l1<- append(l1, list(my\_df), 1) # Añade un elemento en la posición 1.

Lista[[vector1]]<-c(vector1,nuevo\_valor) # Modifica el contenido de un vector dentro de la lista.

Otra opción sería modificar el elemento y volver a asignárselo a la lista.

lista[[4]] <- NULL # Quita un elemento de la lista.

Otra opción sería seleccionar los elementos que queremos conservar y volver a asignarlo a la lista.

lista.recursiva <- list(list(my\_vector, my\_matrix), my\_df) # Genera una lista recursiva, donde el primer elemento es una lista.

lista.recursiva[[1]][[1]] <- my\_df # Modificar una lista dentro de una lista.

lista.recursiva[[1]][[1]] <- NULL #Elimina la lista dentro de la lista.

**Indexación de listas**

lista[[3]]# Si indexamos de este modo, el resultado es un dataframe. Le estamos pidiendo todo el contenido del 3er elemento.

lista[3]# Si indexamos de este otro modo, lo que nos devuelve es una lista

lista[[1]][1, ] #Selecciona la primera fila del primer elemento de la lista. Ojo, dicho elemento debe de tener 2 dimensiones.

lista[[2]][3] #Selecciona el tercer elemento del segundo elemento de la lista. Dicho elemento debe tener solo 1 dimensión.

**Ejemplos Ejercicios**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Creación de listas sin nombre**  my\_vector <- 1:10 # Creamos un vector.  my\_matrix <- matrix(1:9, ncol = 3) # Creamos una matriz.  my\_df <- data.frame(my\_vector) # Creamos un data frame.  l1 <- list(my\_vector, my\_matrix, my\_df)  **Creación de listas con nombre**  l2 <- list(vec = my\_vector, mat = my\_matrix, df = my\_df)  **Utilizando la función de composición**  l3 <- c(l1, l2)  **Operaciones con listas**  str(l2) # Resumen del contenido.  head(l2) # Muestra los primeros elementos de la lista.  tail(l2) # Muestra los últimos elementos de la lista.  **Indexación, selección por índice**  l2[2] #Devuelve una lista.  l2[[2]] #Devuelve una matriz.  **Indexación, selección por nombre**  l2[["mat"]] #Devuelve una matriz.  l2$"mat" #Devuelve una matriz.  l2["mat"] # Devuelve una lista.  l2[["mat"]][1, ] #Selecciona la primera fila de la matriz.  l2[["vec"]][3] #Selecciona el tercer elemento del vector. | actors <- c("Jack Nicholson", "Shelley Duvall", "Danny Lloyd", "Scatman Crothers", "Barry Nelson")  scores <- c(4.5, 4.0, 5.0)  comments <- c("Best Horror Film I Have Ever Seen", "A truly brilliant and scary film from Stanley Kubrick", "A masterpiece of psychological horror")  reviews <- data.frame(scores, comments)  **Crea una lista que contenga los siguientes componentes:**  1. moviename: "The Shining"  2. actors: el vector de actores  3. reviews: el data frame de reviews  shining\_list <- list(moviename = "The Shining", actors = actors, reviews = reviews)  **Selecciona el último actor del vector de actores de la lista:**  last\_actor <- shining\_list[["actors"]][5]  **Selecciona la segunda de las críticas del data frame de reviews de la lista:**  second\_review <- shining\_list[["reviews"]][2, ]  **Añade un nuevo elemento a lista:**  year: 1980  shining\_list\_full <- c(shining\_list, year = 1980)  **Comprueba el contenido de la lista empleando la función str:**  str(shining\_list\_full)   |  |  | | --- | --- | | vector <- 1:10 | matriz <- matrix(1:9, ncol = 3) |   frase <- "En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  lista <- list(vector, matriz, frase)   |  |  | | --- | --- | | **Añade el nº 11 al vector.**  lista[[1]]<-c(vector,11) | **Cambia el valor 9 de la matriz por un 22.**  lista[[2]][3,3]<-22 |   **Eliminar la frase de la lista.**  lista[[3]]<-NULL |

# Ejercicio completo: Búsqueda local del primer mejor

El ejercicio consiste en la búsqueda del valor óptimo por minimización.

Se debe recorrer una matriz de valores, empezando por la esquina inferior derecha, y encontrar el valor mínimo “mirando” alrededor del valor actual.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 8 | 7 | 6 | 7 | 5 | | 5 | 3 | 4 | 5 | 9 | | 6 | 2 | 7 | 6 | 7 | | 5 | 1 | 8 | 7 | 9 | | 3 | 0 | 8 | 8 | 9 |   Matriz con la que trabajar y resultado deseado. | Se ha de mirar el entorno del valor en el que estamos. Dado que empezamos en la esquina inferior derecha, el entorno que tenemos es el siguiente:    Miramos en el sentido de las agujas del reloj, quedándonos con el primer mejor valor. | En este caso, iteramos analizando el entorno, hasta que, de nuevo, encontramos un valor que es mejor que el actual.    En el momento que no encontremos un valor mejor, paramos, y devolvemos un vector con los valores recorridos. |

# Creamos el entorno

entorno<-matrix(c(8,7,6,7,5,5,3,4,5,9,6,2,7,6,7,5,1,8,7,9,3,0,8,8,9),nrow=5,ncol=5,byrow=TRUE)

#Inicializamos las variables de posición

fila\_actual<-5

columna\_actual <-5

# Creamos la función que recorrerá el entorno, buscando el primer mejor.

busqueda\_primer\_mejor<-function(entorno,fila\_actual,columna\_actual){

valor\_actual<-entorno[fila\_actual,columna\_actual]

camino<-c(valor\_actual)

# Creamos la matriz para mirar alrededor en el sentido de las agujas del reloj (la 1ª columna son las filas, la 2ª las columnas)

mirar<-matrix(c(-1,1,0,1,1,1,1,0,1,-1,0,-1,-1,-1,-1,0),nrow=8,ncol=2,byrow=TRUE)

colnames(mirar)<-c("fila","columna")

#Buscamos en el entorno

hay\_movimiento <-TRUE

while(hay\_movimiento==TRUE){

hay\_movimiento<-FALSE

for(alrededor in 1:8){ # Como máximo tendremos alrededor 8 posibles movimientos. Los recorremos.

posible\_fila <- fila\_actual + mirar[alrededor,1]

posible\_columna <- columna\_actual + mirar[alrededor,2]

#Comprobamos que no nos hemos salido del entorno

if (posible\_fila <=5 && posible\_fila >=1 && posible\_columna <=5 && posible\_columna >=1){

valor\_posible\_movimiento <- entorno[posible\_fila,posible\_columna]

# Si el valor de la casilla es mejor que el actual, nos movemos a esa casilla y actualizamos valores.

if (valor\_posible\_movimiento < valor\_actual){

fila\_actual <-posible\_fila

columna\_actual <- posible\_columna

valor\_actual <- valor\_posible\_movimiento

camino<-c(camino,valor\_actual)

hay\_movimiento<-TRUE

break() # Salimos del bucle for, dado que ya hemos encontrado el primer mejor

}

}

}

}

# Mostramos el camino seguido

return(camino)

}

# Invocamos a la función

busqueda\_primer\_mejor(entorno, fila\_actual, columna\_actual)