

Bloque 3: Cadenas de Texto

El manejo de texto en R es esencial para limpiar datos, validar entradas y procesar información no estructurada. Las funciones clave son `substr()`, `nchar()`, `paste()`, `strsplit()` y `grep()`.

3.1 Funciones básicas de texto

```
cadena <- "Hola, mundo!"

nchar(cadena)          # longitud: 12
toupper(cadena)        # HOLA, MUNDO!
tolower(cadena)        # hola, mundo!
substr(cadena, 1, 4)    # "Hola"
paste("A", "B", "C")   # "A B C"
paste0("X", 1:3)       # "X1" "X2" "X3"
gsub("o", "0", cadena) # reemplazar 'o' por '0'
grep("mundo", c("hola mundo", "adios")) # indices
grepl("mundo", cadena)  # TRUE/FALSE
```

Ejercicio 7: Carácter inicial

Comprueba si una cadena empieza por la letra 'P'. Devuelve TRUE o FALSE.

```
cadena <- "Perico de los palotes"

# Extraemos el primer caracter
primer_caracter <- substr(cadena, 1, 1)

# Comparamos
primer_caracter == "P" # -> TRUE

# Version compacta con funcion
empieza_por <- function(cadena, letra) {
  substr(cadena, 1, 1) == letra
}
empieza_por("Redes", "R") # TRUE
empieza_por("Python", "R") # FALSE
```

Explicación: `substr(cadena, inicio, fin)` extrae una subcadena desde la posición inicio hasta fin. Los índices en cadenas también empiezan en 1.

Ejercicio 8: Detección de números en cadena

Dado un nombre de usuario, di *TRUE* si contiene al menos un número, y *TRUE* si solo contiene números.

```
usuario <- "NoobSlayer9"

# Dividimos la cadena en un vector de caracteres
letras <- strsplit(usuario, split = "")[[1]]

# Comprobamos cuales son digitos
es_digito <- letras %in% as.character(0:9)

any(es_digito)    # TRUE: tiene al menos un numero
all(es_digito)    # FALSE: no es solo numeros

# Alternativa moderna con grepl
grepl("[0-9]", usuario)    # al menos un numero
grepl("^[0-9]+$", usuario) # solo numeros
```

Explicación: `strsplit()` divide la cadena en un vector de caracteres. `%in%` comprueba pertenencia. `any()` y `all()` son los cuantificadores lógicos de R.

Ejercicio 9: Contar vocales

Cuenta cuantas vocales (incluyendo acentuadas) hay en una cadena de texto.

```
cuenta_vocales <- function(cad) {
  letras <- strsplit(tolower(cad), split = "")[[1]]
  vocales <- c("a", "e", "i", "o", "u",
              "a", "e", "i", "o", "u")
  sum(letras %in% vocales)
}

cuenta_vocales("Idiota")    # -> 4
cuenta_vocales("Murciélagos") # -> 5
cuenta_vocales("El veloz murciélago") # -> 8
```

Explicación: `tolower()` antes de comparar evita distinguir mayúsculas. Incluimos vocales acentuadas en el vector de referencia.

Bloque 4: Estructuras de Control

Las estructuras de control nos permiten tomar decisiones (if/else) y repetir acciones (for, while, repeat). En R se prefiere la programación vectorizada sobre los bucles, pero conocerlos es fundamental.

4.1 Condicionales if / else if / else

```
x <- 15

if (x > 0) {
  cat(x, "es positivo\n")
} else if (x < 0) {
  cat(x, "es negativo\n")
} else {
  cat("es cero\n")
}

# ifelse() vectorizado
notas <- c(3, 7, 5, 9, 2)
ifelse(notas >= 5, "Aprobado", "Suspenso")
```

4.2 Bucles

```
# Bucle for
for (i in 1:5) {
  cat("Iteracion", i, "\n")
}

# Bucle while
contador <- 0
while (contador < 3) {
  contador <- contador + 1
  cat("Contador:", contador, "\n")
}

# next (saltarse iteracion) y break (salir del bucle)
for (i in 1:10) {
  if (i %% 2 == 0) next      # saltar pares
  if (i > 7) break          # parar al llegar a 9
  cat(i, "")
}
```

Ejercicio 10: Semáforo con if/else

Dada la luz actual del semáforo, devuelve la siguiente luz.

```
actualiza_semaforo <- function(actual) {  
  if (actual == "verde") {  
    return("amarillo")  
  } else if (actual == "amarillo") {  
    return("rojo")  
  } else if (actual == "rojo") {  
    return("verde")  
  } else {  
    stop("Color no valido. Usa: verde, amarillo o rojo")  
  }  
}  
  
actualiza_semaforo("verde")      # "amarillo"  
actualiza_semaforo("amarillo")  # "rojo"  
actualiza_semaforo("rojo")      # "verde"
```

Explicación: stop() lanza un error con mensaje personalizado cuando la entrada no es válida. return() finaliza la función y devuelve el valor.

Ejercicio 11: Siglos

Dada una fecha (solo el año), determina a que siglo pertenece.

```
siglo <- function(anio) {  
  if (anio <= 0) stop("El año debe ser positivo")  
  s <- ceiling(anio / 100)  
  return(s)  
}  
  
siglo(2023)  # -> 21  
siglo(1900)  # -> 19  
siglo(40)    # -> 1  
siglo(1)     # -> 1  
  
# Con etiqueta  
siglo_texto <- function(a) paste0(siglo(a), " siglo")  
siglo_texto(2023)  # "21 siglo"
```

Explicación: ceiling() redondea hacia arriba. El año 100 pertenece al siglo 1, ceiling(100/100)=1. El año 101 al siglo 2, ceiling(101/100)=2.

Ejercicio 12: FizzBuzz

Para los números del 1 al 30: si es divisible por 3 imprime 'Fizz', si lo es por 5 imprime 'Buzz', si lo es por ambos 'FizzBuzz', y si no, el número.

```
for (i in 1:30) {  
  if (i %% 15 == 0) {  
    cat("FizzBuzz\n")  
  } else if (i %% 3 == 0) {  
    cat("Fizz\n")  
  } else if (i %% 5 == 0) {  
    cat("Buzz\n")  
  } else {  
    cat(i, "\n")  
  }  
}
```

Explicacion: FizzBuzz es un clasico de programacion. El orden importa: primero comprobar divisible por 15 (= 3x5), luego por 3, luego por 5.