

Soluciones — Retos de R

Explicación paso a paso de cada ejercicio

Lee cada paso antes de mirar el código. Intenta entender el razonamiento, no solo memorizar la solución.

1

Analizador de contraseñas — Solución completa

Bloques 1 (variables y tipos) • 2 (vectores y lógica) • 3 (cadenas de texto)

La estrategia es dividir el problema en partes independientes: primero comprobar la longitud, luego cada tipo de carácter por separado, y finalmente combinar todo en una puntuación y un nivel.

1

Comprobar la longitud con `nchar()`

La función `nchar()` devuelve el número de caracteres de una cadena. Guardamos ese valor en una variable para reutilizarlo. La longitud sola no determina el nivel final, pero es el primer criterio a evaluar.

```
contrasena <- "miPerro99!"

# nchar() cuenta los caracteres, incluyendo símbolos y números
longitud <- nchar(contrasena)
longitud # -> 10
```

2

Detectar mayúsculas y minúsculas con `strsplit()` y `%in%`

Dividimos la cadena en un vector de caracteres individuales con `strsplit()`. Luego usamos `%in%` para preguntar si alguno de esos caracteres pertenece al conjunto de letras mayúsculas (o minúsculas). `any()` devuelve `TRUE` si al menos uno cumple la condición.

```
# Dividir en vector de caracteres: c('m','i','P','e','r','r','o','9','9','!')
letras <- strsplit(contrasena, split = '')[[1]]

# Letras mayúsculas de A a Z
mayusculas <- LETTERS # vector de 'A' a 'Z' en R
minusculas <- letters # vector de 'a' a 'z' en R

tiene_mayus <- any(letras %in% mayusculas) # TRUE
tiene_minus <- any(letras %in% minusculas) # TRUE
```

! LETTERS y letters son constantes predefinidas en R: vectores con las 26 letras del alfabeto ingles en mayusculas y minusculas. Son equivalentes a escribir c('A','B',..., 'Z') y c('a','b',..., 'z').

3 Detectar digitos con as.character(0:9)

Los digitos del 0 al 9 los generamos como un vector de caracteres con as.character(0:9). La misma logica que antes: si algun caracter del vector pertenece a ese conjunto, hay numeros.

```
digitos <- as.character(0:9) # '0' '1' '2' ... '9'

tiene_nums <- any(letras %in% digitos) # TRUE (hay '9' y '9')
```

4 Detectar simbolos especiales con grepl()

Un simbolo es cualquier caracter que NO sea letra ni numero. La expresion regular [^a-zA-Z0-9] significa 'cualquier caracter excepto letras y numeros'. grepl() devuelve TRUE si la contrasena contiene al menos uno.

```
# [^...] en regex significa 'cualquier cosa que NO este en este conjunto'
tiene_simbolos <- grepl('[^a-zA-Z0-9]', contrasena) # TRUE (hay '!')
```

5 Calcular puntuacion y nivel de seguridad

Sumamos los cuatro criterios logicos (TRUE vale 1, FALSE vale 0). La puntuacion maxima es 4. Con if/else if asignamos el nivel segun la puntuacion obtenida.

```
puntuacion <- sum(tiene_mayus, tiene_minus, tiene_nums, tiene_simbolos)

nivel <- if (puntuacion <= 1) {
  "DEBIL"
} else if (puntuacion <= 3) {
  "MEDIA"
} else {
  "FUERTE"
}
```

6 Funcion completa con cat()

Reunimos todo en una unica funcion. Usamos cat () con el separador \n para formatear la salida linea a linea. La funcion no necesita devolver nada con return() porque solo imprime.

```

analizar_contrasena <- function(contrasena) {
  longitud <- nchar(contrasena)
  letras <- strsplit(contrasena, split = '')[[1]]

  tiene_mayus <- any(letras %in% LETTERS)
  tiene_minus <- any(letras %in% letters)
  tiene_nums <- any(letras %in% as.character(0:9))
  tiene_simbolos <- grepl('[^a-zA-Z0-9]', contrasena)

  puntuacion <- sum(tiene_mayus, tiene_minus, tiene_nums, tiene_simbolos)

  nivel <- if (puntuacion <= 1) 'DEBIL' else if (puntuacion <= 3) 'MEDIA' else
  'FUERTE'

  si_no <- function(x) if (x) 'SI' else 'NO'

  cat('Longitud:', longitud, 'caracteres\n')
  cat('Tiene mayusculas :', si_no(tiene_mayus), '\n')
  cat('Tiene minusculas :', si_no(tiene_minus), '\n')
  cat('Tiene numeros :', si_no(tiene_nums), '\n')
  cat('Tiene simbolos :', si_no(tiene_simbolos), '\n')
  cat('Puntuacion:', puntuacion, '/ 4\n')
  cat('Seguridad:', nivel, '\n')
}

```

SALIDA PARA LAS TRES CONTRASENAS DE PRUEBA

```

> analizar_contrasena("miPerro99!")
Longitud: 10 caracteres
Tiene mayusculas : SI
Tiene minusculas : SI
Tiene numeros : SI
Tiene simbolos : SI
Puntuacion: 4 / 4 Seguridad: FUERTE

> analizar_contrasena("abc")
Longitud: 3 caracteres
Tiene mayusculas : NO
Tiene minusculas : SI
Tiene numeros : NO
Tiene simbolos : NO
Puntuacion: 1 / 4 Seguridad: DEBIL

> analizar_contrasena("Gato2024")
Longitud: 8 caracteres
Tiene mayusculas : SI
Tiene minusculas : SI
Tiene numeros : SI
Tiene simbolos : NO
Puntuacion: 3 / 4 Seguridad: MEDIA

```

2

Simulador de notas de clase — Solucion completa

Bloques 1 (variables) • 2 (vectores) • 4 (bucles y control) • 5 (funciones)

La clave de este reto es entender que el vector de notas es nuestro unico dato de entrada. Todo lo demas se calcula a partir de el: estadisticas basicas, conteos por franja y porcentajes. La funcion es el envoltorio que lo organiza y presenta.

1 Generar las notas y entender pmax() / pmin()

`rnorm(30, 5.5, 2)` puede generar valores negativos o mayores de 10, lo cual no tiene sentido como nota. `pmax(0, notas)` sustituye cualquier valor menor que 0 por 0. `pmin(10, notas)` sustituye cualquier valor mayor que 10 por 10. Aplicados juntos, recortan el vector al rango valido.

```
set.seed(2024)
notas <- round(rnorm(n = 30, mean = 5.5, sd = 2), 1)

# Sin recortar podriamos tener notas como -0.3 o 11.2
# pmax actua elemento a elemento: max(0, cada_nota)
# pmin actua elemento a elemento: min(10, cada_nota)
notas <- pmax(0, pmin(10, notas))

# Resultado: 30 notas entre 0 y 10
range(notas) # min y max del vector -> 1.1 9.2
```

! `pmax()` y `pmin()` son las versiones vectorizadas de `max()` y `min()`. `max(0, notas)` devolveria UN solo numero; `pmax(0, notas)` compara elemento a elemento y devuelve un vector del mismo tamano.

2 Calcular estadisticas basicas del vector

R tiene funciones incorporadas para todas las estadisticas descriptivas habituales. Usamos `length()` en vez de `nrow()` porque trabajamos con un vector, no con un data frame.

```
n <- length(notas) # 30
media <- round(mean(notas), 1) # media aritmetica
desv <- round(sd(notas), 1) # desviacion tipica
nota_max <- max(notas) # nota mas alta
nota_min <- min(notas) # nota mas baja
```

3 Contar alumnos por franja con filtrado logico

Cada franja se define con una condicion doble: mayor o igual al limite inferior Y menor al superior. `sum()` sobre un vector logico cuenta los TRUE. El porcentaje es simplemente el conteo dividido entre el total, multiplicado por 100.

```
# Definimos las franjas del sistema de calificacion espanol
insuf <- sum(notas >= 0 & notas < 5) # Insuficiente
sufic <- sum(notas >= 5 & notas < 6) # Suficiente
bien <- sum(notas >= 6 & notas < 7) # Bien
notab <- sum(notas >= 7 & notas < 9) # Notable
sobre <- sum(notas >= 9 & notas <= 10) # Sobresaliente

# Funcion auxiliar para calcular el porcentaje con 1 decimal
pct <- function(x) round(x / n * 100, 1)
```

4 Contar aprobados y suspensos

El corte de aprobado esta separado como argumento de la funcion. Esto nos permite reutilizar la misma funcion con distintos criterios sin modificar el codigo interno.

```
# 'aprobado' es el argumento con valor por defecto 5
aprobados <- sum(notas >= aprobado)
suspensos <- n - aprobados
```

5 Formatear el informe con cat()

`cat()` concatena e imprime sin anadir comillas. Usamos `\n` para saltar de linea y `\t` para tabulacion. `formatC()` alinea numeros a la derecha dentro de un campo de ancho fijo, lo que da el aspecto de tabla.

```
fila <- function(nombre, cuenta, total) {
  cat(formatC(nombre, width=20, flag='-'),
      formatC(cuenta, width=3),
      'alumnos (',
      formatC(pct(cuenta), format='f', digits=1, width=5),
      '%)\n')
}
```

6 Funcion completa con argumento por defecto

Reunimos todos los pasos dentro de la funcion. El argumento `aprobado = 5` tiene un valor por defecto, por lo que llamar a `informe_clase(notas)` funciona igual que `informe_clase(notas, aprobado = 5)`.

```

informe_clase <- function(notas, aprobado = 5) {
  n <- length(notas)
  media <- round(mean(notas), 1)
  desv <- round(sd(notas), 1)
  nota_max <- max(notas)
  nota_min <- min(notas)

  insuf <- sum(notas >= 0 & notas < 5)
  sufic <- sum(notas >= 5 & notas < 6)
  bien <- sum(notas >= 6 & notas < 7)
  notab <- sum(notas >= 7 & notas < 9)
  sobre <- sum(notas >= 9 & notas <= 10)

  pct <- function(x) round(x / n * 100, 1)

  aprobados <- sum(notas >= aprobado)
  suspensos <- n - aprobados

  cat('--- Informe de clase ---\n')
  cat('Alumnos:', n, '| Media:', media, '| Desv. tipica:', desv, '\n')
  cat('Nota max:', nota_max, '| Nota min:', nota_min, '\n\n')
  cat('Insuficiente (0 - 4.9) :', insuf, 'alumnos (', pct(insuf), '%)\n')
  cat('Suficiente (5 - 5.9) :', sufic, 'alumnos (', pct(sufic), '%)\n')
  cat('Bien (6 - 6.9) :', bien, 'alumnos (', pct(bien), '%)\n')
  cat('Notable (7 - 8.9) :', notab, 'alumnos (', pct(notab), '%)\n')
  cat('Sobresaliente(9 - 10 ) :', sobre, 'alumnos (', pct(sobre), '%)\n\n')
  cat('Aprobados:', aprobados, '(', pct(aprobados), '%)',
    '| Suspensos:', suspensos, '(', pct(suspensos), '%)\n')
}

```

SALIDA CON APROBADO = 5 Y CON APROBADO = 4

```

> informe_clase(notas)
--- Informe de clase ---
Alumnos: 30 | Media: 5.4 | Desv. tipica: 1.9
Nota max: 9.2 | Nota min: 1.1

Insuficiente (0 - 4.9) : 7 alumnos ( 23.3 %)
Suficiente (5 - 5.9) : 8 alumnos ( 26.7 %)
Bien (6 - 6.9) : 6 alumnos ( 20.0 %)
Notable (7 - 8.9) : 6 alumnos ( 20.0 %)
Sobresaliente(9 - 10 ) : 3 alumnos ( 10.0 %)

Aprobados: 23 ( 76.7 %) | Suspensos: 7 ( 23.3 %)

> informe_clase(notas, aprobado = 4) # reto extra
Aprobados: 26 ( 86.7 %) | Suspensos: 4 ( 13.3 %)

```

! Reto extra: al cambiar el argumento aprobado = 4, la funcion no necesita ninguna modificacion interna. Este es el poder de los argumentos con valor por defecto: permiten personalizar el comportamiento sin duplicar codigo.