Tarea24

Yimmy Eman

2022-07-17

Pregunta 1

Describe la diferencia entre is.finite(x) y !is.infinite(x).

```
x <- c(0, NA, NaN, Inf, -Inf)
is.finite(x)
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE
```

```
!is.infinite(x)
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE
```

Pregunta 2

Lee el código fuente de la función dplyr::near(). Para ello elimina los paréntesis () del final. ¿Cómo funciona y qué hace?

```
dplyr::near
```

```
## function (x, y, tol = .Machine$double.eps^0.5)
## {
##     abs(x - y) < tol
## }
## <bytecode: 0x10abeb610>
## <environment: namespace:dplyr>
```

Pregunta 3

Un vector lógico puede tomar tres posibles valores. ¿Cuántos puede tomar un vector de enteros? ¿Y uno de doubles? Usa Google para investigar un poco la naturaleza de estos tipos de datos en R.

Para vectores enteros, `R` usa una representación de 32 bits. Esto significa que puede representar hasta 2^32 diferentes valores con números enteros. Uno de estos valores se reserva para NA_integer_. De la ayuda para enteros.

Pregunta 4

Investiga qué función del paquete rear permite convertir un string en un vector lógico, de enteros o de doubles.

```
parse_logical(c("TRUE", "FALSE", "1", "0", "true", "t", "NA"))
```

```
## [1]  TRUE FALSE  TRUE FALSE  TRUE  TRUE    NA
```

Pregunta 5

¿Qué nos dice la función mean(is.na(x)) acerca del vector x? ¿Y la función sum(!is.finite(x))?

El resultado de 0.286 es igual a 2/7 como se esperaba. Hay siete elementos en el vector x y dos elementos que son NA o NaN.

La expresión sum(!is.finite(x)) calcula el número de elementos en el vector que son iguales a los que faltan (NA), no son un número (NaN) o infinito (Inf).

```
x <- c(-Inf, -1, 0, 1, Inf, NA, NaN)
mean(is.na(x))
```

```
## [1] 0.2857143
```

```
sum(!is.finite(x))
```

```
## [1] 4
```

Pregunta 6

Lee detalladamente la documentación de is.vector(). ¿Para qué puede usarse? ¿Por qué la función is.atomic() no coincide con la declaración de vectores atómicos que hemos dado inicialmente?

La función is.vector() solo verifica si el objeto no tiene más atributos que nombres.

Pregunta 7

Compara y contrasta las funciones setNames() y purrr::set_names().

```
setNames(1:4, c("a", "b", "c", "d"))
```

```
## a b c d
## 1 2 3 4
```

```
purrr::set_names(1:4, c("a", "b", "c", "d"))
```

```
## a b c d
## 1 2 3 4
```

Pregunta 8

Crea una función que a partir de un vector de entrada devuelva: - Solo los números impares (y por supuesto no los NAs). - Todos los elementos excepto el último - Los elementos en posiciones impares - El último valor (¿debes usar [o [[?)

Pregunta 9

¿Por qué x[-which(x > 0)] no es lo mismo que x[x <= 0]?

Estas expresiones difieren en la forma en que tratan los valores perdidos. Probemos cómo funcionan creando un vector con números enteros positivos y negativos, y valores especiales (NA, NaN e Inf). Estos valores deben abarcar todos los tipos relevantes de valores que encontrarían estas expresiones.

```
x <- c(-1:1, Inf, -Inf, NaN, NA)
x[-which(x > 0)]
```

```
## [1]   -1    0 -Inf  NaN   NA
```

```
x[x <= 0]
```

```
## [1]   -1    0 -Inf   NA   NA
```

Pregunta 10

¿Qué ocurre cuando haces un subconjunto con un entero superior a la longitud del vector?

```
x <- c(a = 10, b = 20)
x[3]
```

```
## <NA>
##   NA
```

Pregunta 11

¿Qué ocurre cuando haces un subconjunto con un nombre de variable que no existe en el vector?

Error in x[[“c”]] : subscript out of bounds

```
#x[["c"]]
```