5.3

1. Test whether two vectors are exactly equal (element by element)

Vec1 = c(rownames(mtcars[1:15,]))

Vec2 = c(rownames(mtcars[11:25,]))

|  |
| --- |
| > vec1  [1] "Mazda RX4" "Mazda RX4 Wag" "Datsun 710" "Hornet 4 Drive" "Hornet Sportabout" "Valiant" "Duster 360"  [8] "Merc 240D" "Merc 230" "Merc 280" "Merc 280C" "Merc 450SE" "Merc 450SL" "Merc 450SLC"  [15] "Cadillac Fleetwood"  > vec2  [1] "Merc 280" "Merc 280C" "Merc 450SE" "Merc 450SL" "Merc 450SLC" "Cadillac Fleetwood"  [7] "Lincoln Continental" "Chrysler Imperial" "Fiat 128" "Honda Civic" "Toyota Corolla" "Toyota Corona"  [13] "Dodge Challenger" "AMC Javelin" "Camaro Z28" "Pontiac Firebird" "Fiat X1-9" "Porsche 914-2"  [19] "Lotus Europa" "Ford Pantera L" "Ferrari Dino" "Maserati Bora" "Volvo 142E"  > ifelse(unique(vec1) == unique(vec2), TRUE, FALSE)  [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  Warning message:  In unique(vec1) == unique(vec2) :  longer object length is not a multiple of shorter object length  > all(ifelse(unique(vec1) == unique(vec2), TRUE, FALSE))  [1] FALSE  Warning message:  In unique(vec1) == unique(vec2) :  longer object length is not a multiple of shorter object length |
|  |
| |  | | --- | | > | |

2. Sort the character vector in ascending order and descending order

Vec1 = c(rownames(mtcars[1:15,]))

Vec2 = c(rownames(mtcars[11:25,]))

|  |
| --- |
| > sort(vec1)  [1] "Cadillac Fleetwood" "Datsun 710" "Duster 360" "Hornet 4 Drive" "Hornet Sportabout" "Mazda RX4" "Mazda RX4 Wag"  [8] "Merc 230" "Merc 240D" "Merc 280" "Merc 280C" "Merc 450SE" "Merc 450SL" "Merc 450SLC"  [15] "Valiant"  > sort(vec1, decreasing = TRUE)  [1] "Valiant" "Merc 450SLC" "Merc 450SL" "Merc 450SE" "Merc 280C" "Merc 280" "Merc 240D"  [8] "Merc 230" "Mazda RX4 Wag" "Mazda RX4" "Hornet Sportabout" "Hornet 4 Drive" "Duster 360" "Datsun 710"  [15] "Cadillac Fleetwood"  > sort(vec2)  [1] "AMC Javelin" "Cadillac Fleetwood" "Camaro Z28" "Chrysler Imperial" "Dodge Challenger" "Ferrari Dino"  [7] "Fiat 128" "Fiat X1-9" "Ford Pantera L" "Honda Civic" "Lincoln Continental" "Lotus Europa"  [13] "Maserati Bora" "Merc 280" "Merc 280C" "Merc 450SE" "Merc 450SL" "Merc 450SLC"  [19] "Pontiac Firebird" "Porsche 914-2" "Toyota Corolla" "Toyota Corona" "Volvo 142E"  > sort(vec2, decreasing = TRUE)  [1] "Volvo 142E" "Toyota Corona" "Toyota Corolla" "Porsche 914-2" "Pontiac Firebird" "Merc 450SLC"  [7] "Merc 450SL" "Merc 450SE" "Merc 280C" "Merc 280" "Maserati Bora" "Lotus Europa"  [13] "Lincoln Continental" "Honda Civic" "Ford Pantera L" "Fiat X1-9" "Fiat 128" "Ferrari Dino"  [19] "Dodge Challenger" "Chrysler Imperial" "Camaro Z28" "Cadillac Fleetwood" "AMC Javelin" |
|  |
|  |

3. What is the major difference between str() and paste() show an example?

There are differences between `str\_c()` and paste0

**str\_c() treats missing values properly**

[paste0()](https://www.rdocumentation.org/packages/base/topics/paste) treats missing values as though they were the string "NA", whereas str\_c() respects their missingness.

x <- LETTERS

x[x %**in**% c("A", "E", "I", "O", "U")] <- NA

y <- letters

y[c(TRUE, FALSE, FALSE)] <- NA

stringr::str\_c(x, y)

paste0(x, y)

**str\_c() warns on inexact recycling**

If inputs are different lengths, and not exact multiples of each other, str\_c() follows the usual vector recycling convention of throwing a warning.

paste0(month.abb, letters)

stringr::str\_c(month.abb, letters)

**str\_c()** comes from the stringr package, whereas Paste() is from base package

**str\_c**() is faster than Paste()  but in micro seconds

4. Introduce a separator when concatenating the strings

> str1 = "happy"

> str2 = "learning"

> str3 = "R"

> str4 = "subject"

> result = paste(str1,str2,str3,str4, sep = "-")

> print(result)

[1] "happy-learning-R-subject"