Exercises: Practising R Programming

YSC2210 - DAVis with R

Michael T. Gastner

1 Days of the week in the Shire calendar

In J. R. R. Tolkien's novels, the hobbits of the Shire use a calendar that differs from the Gregorian calendar. Table 1 shows the relationship between the days of the week in these two calendars.[1]

(a) Write a function `shire_day_from_gregorian()` that converts a Gregorian day of the week into the name of the corresponding Shire day using
  (I) `if_else()`.
  (II) `case_when()`.
  (III) `recode()`. You may need to look up the documentation at `?recode`.
  If the function argument is not the name of a Gregorian weekday, the function should return `NA`. Which of these three functions would you stylistically prefer?

(b) The **bsts** package contains a vector called `weekday.names`. Install the package and test whether `shire_day_from_gregorian(weekday.names)` returns the correct result.

2 Measuring run-times with the microbenchmark package

We learned in section 2.4.2 that R has a built-in vectorised function `abs()` that calculates the absolute values of elements in a numeric input vector. Remember that the absolute value of a number x is defined as

$$|x| = \begin{cases} x & \text{if } x \geq 0, \\ -x & \text{otherwise.} \end{cases}$$

Below are four alternatives to `abs()` that produce the same numeric output.

-
  ```
  abs_with_if_else <- function(x) {
    dplyr::if_else(x >= 0, x, -x)
  }
  ```

[1]Data from http://tolkiengateway.net/wiki/Shire__Calendar, accessed on 27 January 2022.

Table 1: Days of the week

Shire day	**Day in Gregorian calendar**
Sterday	Monday
Sunday	Tuesday
Monday	Wednesday
Trewsday	Thursday
Hevensday	Friday
Mersday	Saturday
Highday	Sunday

-
  ```
  abs_with_subsetting <- function(x) {
    neg <- (x < 0)
    x[neg] <- -x[neg]
    x
  }
  ```

-
  ```
  abs_with_data_type_conversion <- function(x) {
    ((x > 0) - (x < 0)) * x
  }
  ```

-
  ```
  abs_with_for_loop <- function(x) {
    for (i in seq_along(x)) {
      if (x[i] < 0) {
        x[i] <- -x[i]
      }
    }
    x
  }
  ```

Go through the code of `abs_with_if_else()`, `abs_with_subsetting()` and `abs_with_data_type_conversion()`. Make sure you understand why these functions return the absolute values of elements in `x`.[2] We want to compare the performance of these functions in terms of their run-times.

(a) Load the **microbenchmark** library. Look through the documentation of the `microbenchmark()` function. You may want to test a few examples in the documentation and, if necessary, search the World Wide web for more information.

(b) Compare the run-times of
  - `abs()`.
  - `abs_with_if_else()`.
  - `abs_with_subsetting()`.
  - `abs_with_data_type_conversion()`.
  - `abs_with_for_loop()`.

  For a fair comparison, run all five functions with identical input (e.g. one million standard normal random numbers, generated with `rnorm(1e6)`).

(c) Comment on your results in (b). Is it worth writing our own function to replace `abs()`? Suppose we need to write our own function for something less common than the absolute value. Judging from your results in (b), which programming strategy should we choose (e.g. conditional element selection, subsetting, data type conversion or `for`-loops)?

3 Comparing two functions for calulating Pythagorean sums

Below is the code for two functions, called `pythag_1()` and `pythag_2()`, that both try to calculate the 'Pythagorean sum' $\sqrt{a^2 + b^2}$ for numeric input vectors a and b. In this problem, we assess their advantages and disadvantages.

```
pythag_1 <- function(a, b) {
  sqrt(a^2 + b^2)
}
pythag_2 <- function(a, b) {
  absa <- abs(a)
  absb <- abs(b)
  p <- pmax(absa, absb)
  q <- pmin(absa, absb)
```

[2]There is no need to memorise the syntax of `abs_with_for_loop()`. In this book, we are not going to work with `for`-loops. One of the reasons for this decision will be evident by the end of this exercise.

```
  ratio <- q / p
  ratio[is.nan(ratio)] <- 1
  p * sqrt(1.0 + ratio^2)
}
```

(a) What is the purpose of `is.nan(ratio)` in the second-to-last line of `pythag_2()`'s function body?
(b) Compare the run-times of `pythag_1()` and `pythag_2()` with the **microbenchmark** package. Use identical input consisting of long numeric vectors. Summarise your observation in a few sentences.
(c) By performing numerical tests, find out under which conditions the functions numerically overflow. When do the functions underflow? Comment on the observed differences between `pythag_1()` and `pythag_2()`.
(d) Is `pythag_1()` or `pythag_2()` better as a general-purpose method? There is no simple right-or-wrong answer. I am interested in your reasoning.

4 Floating-point accuracy

(a) Explain the following output. Keep your explanation shorter than ten sentences. Feel free to search the World Wide Web for help.

```
0.1 + 0.2
```

```
## [1] 0.3
```

```
0.1 + 0.2 == 0.3
```

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

(b) Which R function should we use instead of `==` to test for equality of two floating-point numbers?