STAT 33A Workbook 11

CJ HINES (3034590053)

Nov 12, 2020

This workbook is due **Oct Nov 12, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

While-loops

Watch the "While-loops" lecture video.

Exercise 1

The *decimal* or *base 10* number system uses 10 different digits (0 through 9) to represent numbers. Each *place* in a base 10 number represents a successive power of 10, which makes it possible to represent any number.

For example, the number 270 has a 2 in the hundreds place, a 7 in the tens place, and a 0 in the ones place. We can describe this mathematically as:

$$2 \times 10^2 + 7 \times 10^1 + 0 \times 10^0$$

Base 10 is the most popular number system, but we can also use other bases to represent numbers. The *binary* or *base 2* number system uses 2 different digits (0 and 1) to represent numbers. Each place in a base 2 number represents a successive power of 2.

You can compute the base 10 value of a number in any base by adding up each digit multiplied by the corresponding power of the base. For example, the 10 value of the base 2 number 1001 is:

$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 9$$

As another example, the base 10 value of the base 3 number 211 is:

$$2 \times 3^2 + 1 \times 3^1 + 1 \times 3^0 = 22$$

Write a function `as_base10` that computes the base 10 value of numbers in other bases. Your function should have two parameters: `x` and `base`. Your function should return the base 10 value as a number.

You can assume that the argument for `x` will be a vector of numbers where each element corresponds to one place. For example, if someone wants use your function to convert the base 2 number 1001, they will write:

```
base2 = c(1, 0, 0, 1)
as_base10(base2, 2)
```

Similarly, if they want to convert the base 3 number 211, they will write:

```
base3 = c(2, 1, 1)
as_base10(base3, 3)
```

Show that your function works for the examples above and a few other cases.

*Hint 1: You **DO NOT** need to write a loop for this exercise.*

Hint 2: There are many base converter websites online, if you want to check your work.

Note: Your computer hardware uses base 2 for everything. Software in your computer converts numbers to base 10 for display to humans.

YOUR ANSWER GOES HERE:

```
as_base10 = function(x, base) {
  sum(x * base ** c((length(x)-1):0))
}

#Tests
as_base10(c(1, 0, 0, 1), 2) #base2 example
```

```
## [1] 9
```

```
as_base10(c(2, 1, 1), 3) #base3 example
```

```
## [1] 22
```

```
as_base10(c(5, 9, 6), 2)
```

```
## [1] 44
```

```
as_base10(c(9, 9, 9), 10)
```

```
## [1] 999
```

Exercise 2

You can convert a base 10 number into some other base by doing repeated division. The exact algorithm is:

1. Divide the number by the base to get a dividend and remainder.
2. Record the remainder as a digit, to the left of any digits already recorded.
3. If the dividend is 0, you're finished.
4. If the divident is not 0, repeat these steps, replacing the number with the dividend.

For example, suppose we want to convert 365 into base 3. If we divide 365 by 3, we get dividend 121 and remainder 2. So we record the digit `2`.

Since the dividend was not 0, we repeat the steps. If we divide 121 by 3, we get dividend 40 and remainder 1. So we record the remainder to the left of the digit we already have to get `12`.

Again since the dividend was not 0, we repeat. If we divide 40 by 3, we get dividend 13 and remainder 1. So we record the remainder and have `112`.

Again we repeat. If we divide 13 by 3, we get dividend 4 and remainder 1. So we record the remainder and have `1112`.

Yet again we repeat. If we divide 4 by 3, we get dividend 1 and remainder 1. So we record the remainder and have `11112`.

Again we repeat. If we divide 1 by 3, we get dividend 0 and remainder 1. So one last time, we record the remainder and have `111112`. Since the dividend is 0, we are done. The base 10 number 365 is 111112 in base 3 (you can check this with your function from Exercise 1).

Write a function `as_base` that converts a number in base 10 into some other base. Your function should have two parameters: `x` and `base`. Your function should return the result as a **vector** of numbers, where each element of the vector corresponds to one place.

For example, if someone calls `as_base(365, 3)`, the function should return the vector `(1, 1, 1, 1, 1, 2)`. Similarly, if someone calls `as_base(5, 2)`, the function should return the vector `(1, 0, 1)`.

Show that your function works for the examples above and a few other cases.

Hint: You do need to write a loop for this exercise.

YOUR ANSWER GOES HERE:

```
as_base = function(x, base) {
  final = c()
  while (x / base != 0) {
    remainder = x %% base
    x = x %/% base
    final = c(remainder, final)
  }
  final
}

#Tests
as_base(365, 3)
```

```
## [1] 1 1 1 1 1 2
```

```
as_base(5, 2)
```

```
## [1] 1 0 1
```

```
as_base(596, 2)
```

```
##  [1] 1 0 0 1 0 1 0 1 0 0
```

```
as_base(3275, 10)
```

```
## [1] 3 2 7 5
```

Developing Iterative Code

Watch the "Developing Interactive Code" lecture video.

Note: The slides and video mention repeat-loops and recursion. These are two more kinds of iteration that are rarely used in practice (at least in R). If you want to learn more, see Matloff's "The Art of R Programming" Section 7.1 (for repeat-loops) and Section 7.9 (for recursion).

No exercises for this section.

Messages, Warnings, and Errors

Watch the "Messages, Warnings, and Errors" lecture video.

Exercise 3

A vectorized implementation of the `to_kelvin` function from Workbook 9 is:

```
to_kelvin = function(temperature, unit) {
  unit = match.arg(unit, c("celsius", "fahrenheit"), several.ok = TRUE)

  # First convert Fahrenheit to Celsius.
  is_f = unit == "fahrenheit"
  temperature[is_f] = (temperature[is_f] - 32) * 5 / 9

  temperature + 273.15
}
```

Write a modified version of `to_kelvin` that checks for potential problems. In particular, your version should check the assumptions that:

- `temperature` is numeric.
- `temperature` and `unit` are the same length, or `unit` has length 1.

Your version should raise an error (with a descriptive message) if either of these assumptions don't hold.

Test your function to show that it checks for potential problems. You can use `error = TRUE` on an RMarkdown code chunk to allow errors when knitting.

YOUR ANSWER GOES HERE:

```
to_kelvin = function(temperature, unit) {
  if (class(temperature) != "numeric") {
    stop("This is an Error! Temperature is not numeric!")
  }
  if (length(temperature) != length(unit) || length(unit) == 1) {
    stop("This is an Error! Temperature and Unit are not the same length!")
  }

  unit = match.arg(unit, c("celsius", "fahrenheit"), several.ok = TRUE)
  is_f = unit == "fahrenheit"
  temperature[is_f] = (temperature[is_f] - 32) * 5 / 9
  temperature + 273.15
}

#Tests
to_kelvin(c(99, 35, 11), c("fahrenheit", "celsius", "celsius")) #no error
```

```
## [1] 310.3722 308.1500 284.1500
```

```
to_kelvin(c(0, 32, 15, 69, 80), c("fahrenheit", "fahrenheit", "fahrenheit"))
```

```
## Error in to_kelvin(c(0, 32, 15, 69, 80), c("fahrenheit", "fahrenheit", : This is an Error! Temperatu
```

```
to_kelvin(c(500, 30, 2, 7), "celsius")
```

```
## Error in to_kelvin(c(500, 30, 2, 7), "celsius"): This is an Error! Temperature and Unit are not the s
```

```
to_kelvin(c("7", 0L, 80.20, TRUE), c("fahrenheit", "celsius","fahrenheit", "fahrenheit"))
```

```
## Error in to_kelvin(c("7", 0L, 80.2, TRUE), c("fahrenheit", "celsius", : This is an Error! Temperature
```

Preventing Bugs

Watch the "Preventing Bugs" lecture video.

No exercises for this section.

The R Debugger

Watch the "The R Debugger" lecture video.

No exercises for this section.