Lecture 5: Control Flow and Functions

Problem Set for Lecture 5

These questions will assist in bolstering your understanding of the material in Lecture 5. Emphasis should be placed on having the correct code/output as well as communication. The deliverable should be a knitted RMarkdown document to pdf without any code running off the page. You will be able to present these in office hours as an oral assessment interview along with the problem set for Lecture 6 and Lecture 7.

***** First, make sure you have read the Lecture Material's writeup *****

1. Interest accruing in a bank account can be modeled with compound interest. One common model is

$$A = P\left(1 + \frac{r}{n}\right)^{nt}$$

where P is the initial amount, r is the annual interest rate (as a decimal), n is the number of compounding periods per year, and t is the number of years.

   a) **Compound interest function.** Write a function called `compound_amount()`, which takes the parameters `P`, `r`, `n`, and `t` and returns the final amount A, rounded to the nearest cent. Make sure your function only checks that all inputted values are plausible. This includes the variables being positive with n being a whole number and that none of the inputted values are non-numeric or missing. Then test your function by computing the final amount when $P = 10000$, $r = 0.05$, $n = 12$, and $t = 20$. Report your answer (in dollars) rounded to the nearest cent.

   b) **Years needed to reach a target.** Write a function `years_to_target()` that returns the smallest whole number of years t such that the balance is at least a specified `target` amount (this will tell you how many years it will take you to save a certain amount of money). This should be done with a `while` loop (do not solve for t algebraically!!!). Include checks for unreasonable inputs and have the function stop running and output a message if the amount of time is greater than 1,000 and the function has not finished running yet. Then test your function by finding the smallest whole number of years needed to reach `target`=20000 when $P = 10000$, $r = 0.05$, and $n = 12$. Report the number of years.

   c) **Deposits over time.** In real life, people make repeated deposits. Write a function `balance_with_deposits()` that returns the final balance after t years if the account, starts at `P`, earns interest at annual rate `r` compounded `n` times per year, receives deposits of size `deposit` exactly `n` times per year. This should be done using a `for` loop to simulate the account one compounding period at a time. Round the final answer to the nearest cent. Then test your function by computing the balance after $t = 5$ years when $P = 0$, $r = 0.05$, $n = 12$, `deposit`=500. Report the final balance rounded to the nearest cent.

2. Often times teachers need to automate their grade-book so they can focus on teaching. Answer the following questions to make it more manageable for them:

   a) **Letter grades.** Write a function `letter_grade(scores)` that takes a numeric vector `scores` and returns a character vector of letter grades using

   $$\text{A: } [90, 100], \quad \text{B: } [80, 90), \quad \text{C: } [70, 80), \quad \text{D: } [60, 70), \quad \text{F: } [0, 60).$$

   This must be done using a nested `ifelse()` statement. Your function must use `stop()` and output a helpful message if `scores` is not numeric, contains missing values, is empty, or contains any values outside the range $[0, 100]$.

   b) **Risk level.** Write a function `risk_level(letter)` that takes a character vector of letter grades (like `"A"`, `"B"`, `"C"`, `"D"`, `"F"`) and returns a character vector of the same length with values `"Passing"`, `"At Risk"`, or `"Failing"` using the rules:

   `"A" or "B"` $\to$ `"Passing"`, `"C"` $\to$ `"At Risk"`, `"D" or "F"` $\to$ `"Failing"`.

   This function must be done using a `for` loop and a nested `if{} else{}` structure (do not use `ifelse()` for this part). Your function must use `stop()` if `letter` contains anything besides `"A"`, `"B"`, `"C"`, `"D"`, `"F"`, or if it contains missing values.

   c) **Grade summary.** Write a function `grade_summary(scores)` that returns a dataframe with columns `score`, `letter`, and `risk`, where:

   `letter` = `letter_grade(scores)`, `risk` = `risk_level(letter)`.

   Before calling the other two functions, `grade_summary(scores)` must sort the scores from largest to smallest. Your function should then pass the sorted scores into `letter_grade()`, and pass the resulting letters into `risk_level()`, and then combine everything into a dataframe. Finally, test `grade_summary(scores)` on the vector below and show the resulting dataframe.

   ```
   scores <- c(93, 72, 88, 59.5, 100, 60, 69.9, 81, 77, 0)
   ```

3. Answer the following questions relating to the `apply()` and `aggregate()` functions. Do <u>not</u> use `dplyr` for this problem. Use the built-in `iris` dataset.

   a) Use `aggregate()` to compute the mean of <u>all four</u> measurement columns at the same time (`Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`), grouped by `Species`. Store the result as `mean_by_species`. This should be done in one command (not four separate commands).

   b) Use an `apply()`-family function to compute the range (max $-$ min) of <u>all four</u> measurement columns at the same time, <u>within each species</u>. Store the result as `range_by_species`. Your final object should clearly show the range for each measurement for each species. This should be done by grouping by `Species` first and then applying the range calculation across the four measurement columns (do not do one measurement at a time and then combine).

   c) Use `aggregate()` on <u>only</u> `Petal.Length` to compute <u>both</u> the mean and the max at the same time, grouped by `Species`. Store the result as `petal_stats`. This should be done in one command.