Lab Four – Programming in R

- Complete the tasks below. Make sure to start your solutions in on a new line that starts with "**Solution**:".
- Make sure to use the Quarto Cheatsheet. This will make completing and writing up the lab *much* easier.

In this lab, we will build a mealkit recipe generator. I created a small website of about 40 recipes. We will scrape recipes from that website, randomly select three meals, and print a grocery list with recipe cards.

1 Preliminaries

1.1 Part a

Below, I create a numeric vector filled with random observations. Ask for the 3rd item. **Note**: The `set.seed(7272)` portion ensures we all get the same answer.

```
set.seed(7272) # sets the randomization seed for replication
x <- sample(x=1:10,          # sample from 1, 2, 3, ..., 10
            size=10,         # sample of 10
            replace = TRUE)  # allowed to sample the same item multiple times
```

Solution

```
x[3]
```

```
[1] 5
```

1.2 Part b

Below, I create a data frame filled with random observations.

```
set.seed(7272) # sets the randomization seed for replication
df <- data.frame(x1 = sample(x=1:10,          # sample from 1, 2, 3, ..., 10
                             size=10,         # sample of 10
                             replace = TRUE), # allowed to sample the same item multiple times
                 x2 = sample(x=1:10,          # sample from 1, 2, 3, ..., 10
                             size=10,         # sample of 10
                             replace = TRUE), # allowed to sample the same item multiple times
                 x3 = sample(x=1:10,          # sample from 1, 2, 3, ..., 10
                             size=10,         # sample of 10
                             replace = TRUE)  # allowed to sample the same item multiple times,
)
```

1.3 Part b

Use the `head(...)` function to peek at the data frame. **Solution**

```
head(df)
```

```
  x1 x2 x3
1  8  1  8
2  3  7  2
3  5  3  3
4  3  6  2
5  2  2  3
```

```
6  3  8 10
```

1.4 Part c

Ask for the column `x1`.

Solution

```
df$x1
```

```
 [1] 8 3 5 3 2 3 1 2 6 2
```

```
# or equivalently:
# df[ , "x1"]
```

1.5 Part d

Ask for the fifth row of the data frame.

Solution

```
df[5, ]
```

```
  x1 x2 x3
5  2  2  3
```

1.6 Part e

Ask for the the value of `x1` in the fifth row of the data frame.

Solution

```
df[5, "x1"]
```

```
[1] 2
```

```
# or:
# df$x1[5]
```

1.7 Part f

Ask for the the value of in the third column and the fifth row of the data frame.

Solution

```
df[5, 3]
```

```
[1] 3
```

1.8 Part g

Create a sequence from 1 to 10 by 2 and use it to print the odd rows of the data frame.

Solution

```
odd_rows = seq(1, 10, by = 2)
df[odd_rows, ]
```

```
  x1 x2 x3
1  8  1  8
3  5  3  3
5  2  2  3
7  1 10  1
9  6  2 10
```

1.9 Part h

Below, I create an empty column called `x12`. Fill in the details of the `for(...)` loop to ensure `x12` is the product of `x1` and `x2`.

```
n <- nrow(df)               # How many rows to we have to fill?
df$x12 <- rep(NA, nrow(df))   # Create an empty column for x12
for(i in 1:nrow(df)){
  # REPLACE WITH THE NECESSARY CODE
}
```

Solution

```
n = nrow(df)
df$x12 = rep(NA, nrow(df))
for(i in 1:nrow(df)){
  df$x12[i] = df$x1[i] * df$x2[i]
}
df$x12
```

```
 [1]  8 21 15 18  4 24 10 12 12  8
```

1.10 Part i

Write a function called `calculate.score(...)` that takes three arguments representing `x1`, `x2`, and `x3` and returns a single value based on the formula:

$$Score = (x_1 \times 2) + x_2 - x_3$$

Use your function to create a new column `total.score` in our data frame `df`.

Solution

```
calculate.score = function(x1, x2, x3){
  (x1 * 2) + x2 - x3
}

(df$total.score = calculate.score(df$x1, df$x2, df$x3))
```

```
 [1]  9 11 10 10  3  4 11  9  4  7
```

1.11 Part j

Create a function called `evaluate.row(...)` that takes one argument and returns "low" when the argument is less than 4, "mid" when the argument is between 4 and 7 (inclusive), and "high" when the arguement is 8 or larger. Then, use `sapply(...)` to apply it to column `x1`.

Solution

```
evaluate.row = function(value){
  if(value < 4){
    "low"
  } else if(value <= 7){
    "mid"
  } else {
    "high"
  }
}

sapply(df$x1, evaluate.row)
```

```
 [1] "high" "low"  "mid"  "low"  "low"  "low"  "low"  "low"  "mid"  "low"
```

```
#just prints it
#below will store it
```

```r
#may or may not have gone down sapply rabbit hole
(df$x1.category = sapply(df$x1, evaluate.row))
```

```
 [1] "high" "low"  "mid"  "low"  "low"  "low"  "low"  "low"  "mid"  "low"
```

1.12 Part k

Did we need to use loops or functions in (h.)-(j.)? That is, can we use vectorization to attain the same results in 1 line each? Where it is possible, write the line of code. Where it is not, explain why.

Solution

```r
# Vectorized version of Part h (no loop needed):
(df$x12 = df$x1 * df$x2)
```

```
 [1]  8 21 15 18  4 24 10 12 12  8
```

```r
# Vectorized version of Part i (no function needed):
(df$total.score = (df$x1 * 2) + df$x2 - df$x3)
```

```
 [1]  9 11 10 10  3  4 11  9  4  7
```

```r
# For Part j, we *can* partially vectorize using ifelse:
(df$x1.category = ifelse(df$x1 < 4, "low",
                         ifelse(df$x1 <= 7, "mid", "high")))
```

```
 [1] "high" "low"  "mid"  "low"  "low"  "low"  "low"  "low"  "mid"  "low"
```

Explanation Parts h and i dont need loops or functions because math on vectors in R is vectorized. Algthough, for j, I wrote a function that used if/else, you could still do it in one vectorized line with a nested ifelse, so loops are not needed

2 Complete Tasks for One Recipe

2.1 Part a

Install and load the `rvest` package (Wickham 2025).

Solution

```r
# run this in console OLY
# install.packages("rvest")
library(rvest)
```

2.2 Part b

Load the html of the `website/KimchiGrilledCheese.html` using the `read_html()` function and save the result to an object called `recipe.item`. We will use the Kimchi Grilled Cheese recipe as our prototype and extend this workflow to all recipes, so try to be as general as possible. If you do look at it, you'll notice it contains the html we saw in the developer tools in class. **Hint**: You can use `read_html(...)` like `read.csv(...)`. Don't forget you can use the documentation to help use it.

Solution

```r
library(rvest)

recipe.item = read_html("website/KimchiGrilledCheese.html")
```

2.3 Part c

Open the html file in a web browser and open developer tools. In chrome-based browsers, you can do this by pressing the three verticle dots in the upper-right corner, clicking "more tools", then "developer tools".

Find the name of the Ingredients section of the website and pull the HTML of the function using `html_element(...)` and save the results to an object called `ingredients.section`. **Hint**: You can ask for elements by id using a preceeding "#". See `?html_element(...)` for a helpful example.

Solution

```
ingredients.section = recipe.item |>
  html_element("#ingredients")
```

2.4 Part d

Now, we want to obtain all of the itemized items. Find the element type the individual ingredients and pull all of them from `ingredients.section` using `html_elements(...)` (note the added s) and save the results to an object called `ingredients`. **Hint**: You can ask for elements by type by simply specifying the tag (e.g., "p" for paragraph). See '?html_element for a helpful example.

Solution

```
ingredients <- ingredients.section |>
  html_elements("li")
```

2.5 Part e

Similar to Part c. Find the name of the Instructions section of the website and pull the HTML of the function using `html_element(...)` and save the results to an object called `instructions.section`.

Solution

```
instructions.section <- recipe.item |>
  html_element("#instructions")
```

2.6 Part f

Now, we want to obtain all of the enumerated items. Find the element type the individual instructions and pull all of them from `instructions.section` using `html_elements(...)` (note the added s) and save the results to an object called `instructions`.

Solution

```
instructions <- instructions.section |>
  html_elements("li")

html_text(instructions)
```

```
[1] "In a frying pan, heat 2 tablespoons olive oil with chili flakes."
[2] "Add spinach and cook until wilted."
[3] "Mix gochujang and mayonnaise in a small bowl and spread on one side of each slice of bread. Sprinkle w
[4] "Build sandwiches by layering spinach, kimchi, and cheese between two slices of bread."
[5] "Fry in butter until golden brown on both sides."
```

2.7 Part g

Find the class of the recipe image and pull the HTML using `html_element(...)` and save the results to an object called `image.element`. **Hint**: You can ask for elements by class using a preceeding ".". See `?html_element(...)` for a helpful example. Further, note that HTML elements may have more than one class (separated by a space). When that is the case, you need to choose one.

Solution

```
image.element <- recipe.item |>
  html_element(".figure-img")
```

2.8 Part h

Use the `html_attr(...)` function to pull the source link ("src"). Then, use `paste(...)` to prepend the source link with "website/" so we have the full link. **Note**: This would be like adding "https://www.website.com" to get the absolute link.

Solution

```
image.src <- html_attr(image.element, "src")
image.url <- paste0("website/", image.src)
```

2.9 Part i

Now that we have all the things we need, let's try to print the recipe. Below, I have written code to print from the objects. One by one, remove `#| eval: false` from the YAML header and add `#| echo: false` and `#| results: 'asis'`, and test. Let me know if you're stuck!

Image

```
# width="50%" shrinks the image for printing
recipe.title <- recipe.item |> html_element("h1") |> html_text()
cat(paste('![', recipe.title, '](', image.url, "){width=50%}", sep=""))
```

Solution

Figure 1: Kimchi Grilled Cheese

Ingredients

```
formatted_list <- paste("-", html_text(ingredients))
cat(formatted_list, sep = "\n")
```

Solution

- 1/8 tsp chili flakes
- 4 garlic cloves
- 5 oz spinach
- 1 tsp gochujang

- 3 tsp mayonnaise
- 2/3 cup kimchi
- 4 slices of bread
- 2/3 cup cheddar cheese
- 4 tablespoons everything seasoning.

Instructions

```
formatted_list <- paste("1.", html_text(instructions))
cat(formatted_list, sep = "\n")
```

Solution

1. In a frying pan, heat 2 tablespoons olive oil with chili flakes.
2. Add spinach and cook until wilted.
3. Mix gochujang and mayonnaise in a small bowl and spread on one side of each slice of bread. Sprinkle with everything seasoning and press it into the bread.
4. Build sandwiches by layering spinach, kimchi, and cheese between two slices of bread.
5. Fry in butter until golden brown on both sides.

3 Complete a Full Menu!

Open `Menu.qmd` and add code to complete the following.

1. Randomly select three dinner recipes and one breakfast recipe at random. **Hint:** Use the `sample(...)` function.
2. Pull the image, ingredients, and instructions for each selected recipe.
3. Combine all of the ingredients into a grocery list on the first page.
4. Print the image, ingredients, and instructions for each receipe on the subsequent pages.

When you render this document, no code should be visible. Instead, you should see a five-page document as described above.

4 Describe your work!

Solution This took an incredibly long period of time since it took me a while to stop ending up with nulls for the images so it would automaticly try and render as pdfs so I had to change some of the stuff I am pretty sure that I was not supposed to so there is most likely an easier way of doing it, but it works so victory. After that then I had to fix my brackets which allowed it to get a bit closer then for some reason I had messeed up the printing for the grocery list so it still would not render so ended up doing ingredients in two sections

4.1 Why a fake website?

Read https://www.scrapingbee.com/blog/is-web-scraping-legal/. Originally, I conceived doing this with recipes from Purple Carrot (my favorite subscription service). Being a large company, their terms of service are *very* long and precluded us from copying their recipes. We also checked a few smaller recipe websites we like and even they had terms of service that restricted automated collection of data.

Solution I have gone over the article and after hearing what you were saying about not actually running them yourself incase someone gets stuck in a loop

4.2 Conditionals

Did you use conditional statements in your code? If yes, how. If not, are there places you could have used them but did something else?

Solution Yes, I used conditional statements in the scrape_recipe() function to deal with the missing images issues. my code looks like below, if(length(image_element) > 0 && !is.na(html_attr(image_element, "src"))) checks two things:

Does the .figure-img exist? (length() > 0)

Does it have a src attribute? (!is.na())

If both then extract and fix image path

If either are false then it sets image_url = NA (no image)

I needed it since some recipes might not have images, or like the class selector might not work. Without it, html_attr() would return NA and crash gsub()…. which may or may not have happened like 3 times before I figured out how to get it right, but who is counting.

there are technically three times I could have used one and didnt but one of them was not really an option. i should explain that better.

i could have skipped printing images if NA (in recipe chunks), done like different formatting for breakfast vs dinner, and lastly i could have handled empty ingredient lists with one.

i will code the last one since i dont think you need me to show you all three. what is below would probably work (didnt test it)

```
if(length(recipe1$ingredients) == 0) {
  cat("No ingredients listed.\n\n")
}
```

4.3 Loops

Did you use loops in your code? If yes, how. If not, are there places you could have used them but did something else?

Solution I did not use like an explicit for or while loops, and there were times where I could have but if I am being honest all the talk about like messing them up and creating infinite loops stressed me out so i did my best to avoid them luckily i was able to complete it without any.

I could have done it in the like grocery list creation or for printing the recipes since I elected to put the recipes in seperate chunks

4.4 Functions

Did you use functions in your code? If yes, how. If not, are there places you could have used them but did something else?

Solution Yes, I used a function called scrape_recipe() so i wouldnt have to repeat the same scraping code for every recipe.

I suppose i could have also used a function to do the printing of the recipes and like the grocery list formating but i needed a lot of help to do everything already so i didnt want to rtry and make a whole other function work

References

Wickham, Hadley. 2025. *Rvest: Easily Harvest (Scrape) Web Pages*. https://doi.org/10.32614/CRAN.package.rvest.