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

x[3] # Solution
```

```
[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) # used head function to get first few rows
```

```
  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"] # got vector of x1 column
```

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

1.5 Part d

Ask for the fifth row of the data frame.

Solution:

```
df[5, ] # got vector of fifth row of df
```

```
  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"] # got the value of x1 in fifth row
```

```
[1] 2
```

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] # got value of fifth row, column 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.

```
odds <- seq(1, 10, by = 2) # created sequence
df[odds, ] # printed sequence
```

```
  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$x12 <- rep(NA, nrow(df))   # Create an empty column for x12
for(i in 1:nrow(df)){
  df$x12[i] <- df$x1[i] * df$x2[i] # created the values for column x12
}
```

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

```r
calculate.score <- function(x1, x2, x3) {
  score <- (2 * x1) + x2 - x3
  return(score)} # created the calc.score function

n <- nrow(df)                # How many rows to we have to fill?
total.score <- rep(NA, nrow(df))   # Create an empty column for x12
for(i in 1:nrow(df)){
  df$total.score[i] <- calculate.score(df$x1[i], df$x3[i], df$x3[i]) } # used a for loop to fill total.score column
```

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:

```r
evaluate.row <- function(x) {
  if (x < 4) {
   return("low")}
  else if (x >= 4 & x <= 7) {
   return("mid")}
  else {
   return("high")}
}

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: In part h, I didn't need to use a for loop. I could have instead gone line by line and used vectorization such as x1[1] * x2[1] for part i, the same thing could be done. the code would have been x1[1] * 2 + x2[1] - x3[1] in part j, I did not need to use a for loop. (seen above)

2 Complete Tasks for One Recipe

2.1 Part a

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

Solution:

```r
# install.packages("rvest") # installed rvest
library("rvest") # loaded rvest
```

2.2 Part b

Load the html of the `website/KimchiGrilledCheese.html` using the `read_html()` function. 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
html <- read_html("website/KimchiGrilledCheese.html") # loaded the html and saved it as an object called 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 <- html_element(html, "#ingredients") # created object called ingredients.section which pulled the html of the ingredients section
```

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 <- html_elements(ingredients.section, "li") # created object called ingredients which pulled the items of the ingredients section
```

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 <- html_element(html, "#instructions") # created object called instructions.section which pulled the html of the instructions sect
```

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 <- html_elements(instructions.section, "li") # created object called instructions which pulled the items of the instructions section
```

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 <- html_element(html, ".quarto-figure") # created object called image.element which pulled the html of the image
```

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 <- html_element(image.element, "img") # got the image tag using the image type and the object created in part g
sourcelink <- html_attr(image, "src") # pulled the source link for the image and saved it as an object
image.url <- paste("website/", sourcelink, sep = "") # created the full link as an object using the paste function
```

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

Ingredients

- 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

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!

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.

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: I did not use any conditionals in my code for the menu. I could have used a conditional statement when making my total ingredients list on the first page of the menu. I could have made it so if an ingredient was already on the list, I could code it to not add it again.

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?

I did use a for loop in my code to make so I wouldn't have to go through and create an ingredients, instructions, and image section for each recepie. Instead I created a vector of all the recepies and looped from the first one to the last one and created the three recepie parts for each inside the loop.

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?

I did use functions inside my code for the menu, specifically the same ones we used in part two of the lab such as html_element, html_elements, html_text, html_attr, paste, and sample which I learned for the randomization element.

References

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