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
```

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:

Running `head(df)` to look at first few elements in the df

```
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:

Asking for the `x1` column

```
df[,1]
```

```
 [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:

Getting 5th row

```
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:

5th row in the `x1` column

```
df[5,1]
```

```
[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:

Getting the 5th row, 3rd column

```
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:

Printing the odd rows in data frame by using `seq` function.

```
df[seq(1, 10, 2),]
```

```
  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`.

Solution:

Created a new column `df$x12` that is a product of values in columns `x1` and `x2`.

```r
n <- nrow(df)               # How many rows to we have to fill? 10 rows
df$x12 <- rep(NA, n)   # Create an empty column for x12
for(i in 1:n){
  # REPLACE WITH THE NECESSARY CODE
  df$x12[i]=df[i,1]*df[i,2]
}
(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:

Created a `calculate.score` function and used a for-loop to loop throuh elements in the column. I then used the formula given to compute and store the results in `total.score`.

```r
calculate.score<-function(x_1,x_2,x_3){
  total.score<-rep(NA, nrow(df))
  for (i in 1:nrow(df)){
    total.score[i]=(x_1[i]*2) + x_2[i] - x_3[i]
  }
  return(total.score)
}

calculate.score(df[,1], df[,2], df[,3])
```

```
 [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:

Used if-statements in the `evaluate.row` function to determine is a number is "high", "low", or "mid"

```r
evaluate.row<-function(dat){
  if(dat<4){
    return("low")
  }else if(dat>=4 && dat<=7){
    return("mid")
  }else if(dat>=8){
    return("high")
  }
}
df$category <- sapply(df$x1, evaluate.row)
(df$category)
```

```
 [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 (h), instead of using the loops, I can directly multiply `x1` column and `x2` column and have it in `df$x12.2` through vectorization. `R` will autmoatically compute row by row.

```r
(df$x12.2 <- df$x1 * df$x2)
```

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

Like in (h), a function and for loop aren't needed in (i), and I can also directly apply the formula to get the results. I can call the columns `df$x1`, `df$x2`, and `df$x3` and pass them into the formula and `R` will automatically compute row by row

```
(df$total.score <- (df$x1 * 2) + df$x2 - df$x3)
```

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

In (j), the function is necessary to pass the argument in the conditionals. Since the outcome is dependant on the if-else conditions, I would need the function to get the result of a single value then use `sapply` to apply it to the whole column `df$x1` in the data frame. I could also use loops like this:

```
df$category <- rep(NA, nrow(df))

for(i in 1:nrow(df)){
  if(df$x1[i] < 4){
    df$category[i] <- "low"
  }else if(df$x1[i] <= 7){
    df$category[i] <- "mid"
  }else{
    df$category[i] <- "high"
  }
}
(df$category)
```

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

2 Complete Tasks for One Recipe

2.1 Part a

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

Solution:

Installed and loaded `rvest` package.

```
# 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:

Read the html website using the `read_html()` function

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

```
{html_document}
<html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
[1] <head>\n<meta http-equiv="Content-Type" content="text/html; charset=UTF-8 ...
[2] <body class="nav-fixed quarto-light">\r\n\r\n<div id="quarto-search-resul ...
```

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:

Pulled the ingredients section by ID then stored it in the `ingredients.section` variable

```
ingredients.section<-html_element(recipe.item, "#ingredients")
(ingredients.section)
```

```
{html_node}
<section id="ingredients" class="level2">
[1] <h2 class="anchored" data-anchor-id="ingredients">Ingredients</h2>
[2] <ul>\n<li>1/8 tsp chili flakes</li>\r\n<li>4 garlic cloves</li>\r\n<li>5  ...
```

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:

Found all itemized items using `html_elements()` then stores them in `ingredients`

```
ingredients<-html_elements(ingredients.section,"li")
(ingredients)
```

```
{xml_nodeset (9)}
[1] <li>1/8 tsp chili flakes</li>
[2] <li>4 garlic cloves</li>
[3] <li>5 oz spinach</li>
[4] <li>1 tsp gochujang</li>
[5] <li>3 tsp mayonnaise</li>
[6] <li>2/3 cup kimchi</li>
[7] <li>4 slices of bread</li>
[8] <li>2/3 cup cheddar cheese</li>
[9] <li>4 tablespoons everything seasoning.</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:

Pulled the instructions section by ID using `html_element()` and stored it in the `instruction.section` variable

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

```
{html_node}
<section id="instructions" class="level2">
[1] <h2 class="anchored" data-anchor-id="instructions">Instructions</h2>
[2] <ol type="1">\n<li>In a frying pan, heat 2 tablespoons olive oil with chi ...
```

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:

Pulled all the individual instructions then stored them in `instructions`

```
instructions<-html_elements(instructions.section,"li")
(instructions)
```

```
{xml_nodeset (5)}
[1] <li>In a frying pan, heat 2 tablespoons olive oil with chili flakes.</li>
[2] <li>Add spinach and cook until wilted.</li>
[3] <li>Mix gochujang and mayonnaise in a small bowl and spread on one side o ...
[4] <li>Build sandwiches by layering spinach, kimchi, and cheese between two  ...
[5] <li>Fry in butter until golden brown on both sides.</li>
```

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.

Solutions:

Pulled the image html by its class name using `html_element()` and stored it in `image.element` object

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

```
{html_node}
<img src="images/KimchiGrilledCheese.jpg" class="img-fluid 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:

Pulled image source link by attribute then pasted "website/" to the link using `paste()`. Stored the final link in `image.url`

```
image.url <-html_attr(image.element,"src")
image.url <- paste("website/", image.url, sep = "")
(image.url)
```

```
[1] "website/images/KimchiGrilledCheese.jpg"
```

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

Solution:

Removed `#| eval: false` from the YAML header and added `#| echo: false` and `#| results: 'asis'` to test code.

Figure 1: Kimchi Grilled Cheese

Ingredients

Solution:

Removed `#| eval: false` from the YAML header and added `#| echo: false` and `#| results: 'asis'` to test code.

- 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

Solution:

Removed `#| eval: false` from the YAML header and added `#| echo: false` and `#| results: 'asis'` to test code.

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.

Solution:

See `menu.qmd` for this assignment.

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.

Solution:

Many commercial sites restrict data collection in their Terms of Service even if the content is publicly visible. Since the website we used was made by the professor, I was able to extract the contents and do the lab assignment while also ensuring that I don't violating a real company's terms of service.

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 conditional statements in my `menu.qmd` assignment. The structure of the recipe pages was consistent (every page had an image, an ingredients list, and an instructions list with the same HTML IDs), so there was no need to check whether elements existed before extracting them.

Although I manually checked if the elements and IDs were there, I could have added conditional statements to handle potential errors. Like if there was a file without the element, the conditionals would help print an error message instead of breaking the document.

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:

Yes. I used a for loop when creating the grocery list on the first page of `menu.qmd`. After randomly selecting one breakfast recipe and three dinner recipes using the `sample()` function, I stored them in a vector `recipe_files` and iterated through each file. For each selected recipe, I extracted the ingredients and printed them together to create a combined grocery list.

I chose not to use loops for printing the full recipes. Instead, I referenced each randomly selected file directly. A loop could have also been used to print each recipe page automatically, but I kept the sections separate to control the formatting of each page.

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:

I didn't use functions in `menu.qmd`. I chose to keep the code explicit in each section to make the process easier to follow for me. However, I could have used functions to avoid repeating code when printing each recipe (since the same steps are performed for each recipe).

References

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