

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.

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

Solution:

```
1 (third.element = x[3])
```

```
[1] 5
```

1.2 Part b

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

```
1 set.seed(7272) # sets the randomization seed for replication
2 df <- data.frame(x1 = sample(x=1:10,          # sample from 1, 2, 3, ..., 10
3                           size=10,          # sample of 10
4                           replace = TRUE),  # allowed to sample the same item multiple times
5                 x2 = sample(x=1:10,          # sample from 1, 2, 3, ..., 10
6                           size=10,          # sample of 10
7                           replace = TRUE),  # allowed to sample the same item multiple times
8                 x3 = sample(x=1:10,          # sample from 1, 2, 3, ..., 10
9                           size=10,          # sample of 10
10                          replace = TRUE)  # allowed to sample the same item multiple times,
11 )
```

1.3 Part b

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

Solution:

```
1 (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:

```
1 (column1 = df[,c("x1")])

[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:

```
1 (row5 = 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:

```
1 (row5.x1 = row5[c('x1')])

   x1
5    2
```

1.7 Part f

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

Solution:

```
1 (value = 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:

```
1 indices = seq(from=1,to=10,by=2)
2 (rows.odds = df[indices,])

   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`.

```
1 n <- nrow(df)           # How many rows to we have to fill?
2 df$x12 <- rep(NA, nrow(df)) # Create an empty column for x12
3 for(i in 1:nrow(df)){
4   df[i,"x12"] = df[i,"x1"] * df[i,"x2"]
5 }
6 (head(df))
```

	x1	x2	x3	x12
1	8	1	8	8
2	3	7	2	21
3	5	3	3	15
4	3	6	2	18
5	2	2	3	4
6	3	8	10	24

Vectorized Solution

```
1 df$x12 = df$x1 * df$x2
2 (head(df))
```

	x1	x2	x3	x12
1	8	1	8	8
2	3	7	2	21
3	5	3	3	15
4	3	6	2	18
5	2	2	3	4
6	3	8	10	24

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:

```
1 calculate.score = function(x1,x2,x3) {
2   (x1 * 2) + x2 - x3
3 }
4 df$total.score = rep(NA,nrow(df))
5 for (i in 1:nrow(df)) {
6   df[i,"total.score"] = calculate.score(df[i,'x1'],df[i,'x2'],df[i,'x3'])
7 }
8 (head(df))
```

	x1	x2	x3	x12	total.score
1	8	1	8	8	9
2	3	7	2	21	11
3	5	3	3	15	10
4	3	6	2	18	10
5	2	2	3	4	3
6	3	8	10	24	4

Vectorized Solution:

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

	x1	x2	x3	x12	total.score
1	8	1	8	8	9
2	3	7	2	21	11
3	5	3	3	15	10
4	3	6	2	18	10
5	2	2	3	4	3

```
6 3 8 10 24      4
```

Vectorized Solution (No Function)

```
1 df$total.score = (df$x1 * 2) + df$x2 - df$x3
2 head(df)
```

	x1	x2	x3	x12	total.score
1	8	1	8	8	9
2	3	7	2	21	11
3	5	3	3	15	10
4	3	6	2	18	10
5	2	2	3	4	3
6	3	8	10	24	4

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 argument is 8 or larger. Then, use `sapply(...)` to apply it to column `x1`.

Solution:

```
1 evaluate.row = function(val) {
2   if (val < 4) {
3     return("low")
4   }
5   else if (val <= 7) {
6     return("mid")
7   }
8   else {
9     return("high")
10  }
11 }
12
13 (result = sapply(df$x1, evaluate.row))
```

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

In this instance, vectorization wouldn’t immediately simplify the logic of the condition, and abstracting this step within a function seems to be the most reasonable approach.

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.

2 Complete Tasks for One Recipe

2.1 Part a

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

Solution

```
1 packages.install("rvest")
1 library("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

```
1 kimchi.grilled.cheese = 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

```
1 (ingredients.section = html_element(kimchi.grilled.cheese, "#ingredients"))
```

```
{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>\n<li>4 garlic cloves</li>\n<li>5 oz s ...
```

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

```
1 (ingredients = html_elements(ingredients.section, "li"))
```

```
{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

```
1 (instructions.section = html_element(kimchi.grilled.cheese, "#instructions"))
```

```
{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

```
1 (instructions = html_elements(instructions.section, "li"))

{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.

Solution

```
1 (image = html_element(kimchi.grilled.cheese, ".img-fluid"))

{html_node}

```

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

```
1 (link = html_attr(image, "src"))

[1] "images/KimchiGrilledCheese.jpg"
1 (image.url = paste("website/", link, sep=""))

[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

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

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 recipe 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 used one conditional statement. When I was working on generalizing the recipe numbering, I realized I was going to run into trouble if I had to expand on the english numbers by hand. So I decided to use the library `english` (no citation listed on CRAN, but credit to John Fox, Bill Venables, Anthony Damico, and Pier Salverdi). I also realized that I shouldn't be numbering the items if there's only a single one in the category (one breakfast item), so my conditional statement simply checked whether to omit the numbering.

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 used a for loop for my master `display.recipes` function and within my logic for printing the ingredient list. I think to avoid using any looping I would have had to store my recipes and ingredients a single, different data structure.

My solutions were not reliant on any vectorization, which suggests that there were significant opportunities to make it more “R native” that I didn’t see how to cleanly implement.

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

My final solution had 3 functions, `display.recipes`, `display.recipe`, and `load.recipe`. The printing logic for the ingredient list could have been placed within a function, but since there’s no repetition of the logic, it did not seem necessary. In earlier drafts I additionally separated out some of the html handling to tiny helper functions, but I decided that the benefits in readability from maximal functional decomposition were less than the cons of more code (largely due to whitespace).

References

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