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] #asking for the 3rd index
```

```
[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.

```
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`.

```
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.

```
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.

```
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.

```
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.

```
odd.rows = seq(from=1, to=10, by=2) #using sequential vector
df[odd.rows,] #using odd.rows in place of vector row number
```

```
  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[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`.

```
x1 = df[,1]
x2 = df[,2]
x3 = df[,3]
calculate.score = function(x1, x2, x3){(2*x1)+x2-x3}
df = data.frame(x1, x2, x3,
                total.score=calculate.score(x1,x2,x3))
df
```

```
   x1 x2 x3 total.score
1   8  1  8           9
2   3  7  2          11
3   5  3  3          10
4   3  6  2          10
5   2  2  3           3
6   3  8 10           4
7   1 10  1          11
8   2  6  1           9
9   6  2 10           4
10  2  4  1           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`.

```
evaluate.row = function(i){
                        if(i<4){
                         "low"
                        }
                        else if(i>=4 & i<=7){
                         "mid"
                        }
                        else{
                        "high"
                        }
}
sapply(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.

I'm choosing peace and joy rn.

2 Complete Tasks for One Recipe

2.1 Part a

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

```
#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.

```
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">\n\n<div id="quarto-search-results"> ...
```

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.

```
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>\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.

```
ingredients = html_elements(ingredients.section, "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`.

```
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>
```