Lab Two – Basic Tasks in R

- Use `R` (R Core Team 2025) to 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.

1 Libraries and Packages

1.1 Part a

Install the `esquisse` package for `R`. Report the code below, ensuring to set the `eval` option to `false`, using "#| eval: false", so that we don't install the package every time you compile your Quarto document.

Solution

```
install.packages("esquisse")
```

1.2 Part b

Load the `esquisse` package for `R`. Report the code below. It does not matter whether you set "#| eval: false" because loading the package does not add a lot of time. Note you will not evaluate any code that uses the `esquisse` package, so there is no need to load it as you compile your Quarto document.

Solution

```
library(esquisse)
```

1.3 Part c

How can you ask for help about the `esquisse` package for `R`? Report the code below, ensuring to set the `eval` option to `false`, using "#| eval: false", so that the help document isn't loaded every time you compile your Quarto document.

```
help("esquisse")
```

1.4 Part d

Is a demo or vignette available for the `esquisse` package for `R`? Report the code below, ensuring to set the `eval` option to `false`, using "#| eval: false", so that the you don't get errors when you compile your Quarto document.

Solution

Yes, there is a vignette available for `esquisse`. One is called get-started, and one is called shiny-usage.

```
vignette(package = "esquisse")
```

There are no demos for esquisse.

```
demo(package = "esquisse")
```

1.5 Part e

Add the BibTeX citation for `esquisse` package for `R` to your `.bib` file and cite it in a sentence describing what the package does below. Note you can reference a citation named `esquisse` using "[@esquisse]".

Solution

The `esquisse` (Meyer and Perrier 2025) package creates a GUI in `r` for graphs, allowing users to interact with graphs and data using drag-and-drop operations rather than strictly `r` code.

1.6 Part f

Run `esquisser()` in your console – not in a chunk of `R` code in your Quarto document.

i. Select `palmerpenguins` from the "Select an environment in which to search:" dropdown. This should automatically select `penguins` in the "Select a data.frame:" dropdown. Click "Import Data".
ii. Drag `body_mass_g` to the X box and `species` to the fill box.
iii. Describe what you see in words. What can you conclude about Adelie, Chinstrap, and Gentoo penguins based on the resulting plot?

Solution

The overall distribution of body mass has a peak around 3500g, mainly driven by the large count of the Adelie species with a large concentration of penguins around the 3500g mark. The Chinstrap penguins are similar, though there were less of them represented in the data, and they appear to skew slightly further right than the Adelie penguins. The Gentoo penguins are the heaviest, with a relatively even distribution of body masses between 4500g and 6000g. Overall, the order of the body mass of the average penguin of each species goes Adelies, Chinstraps, and then Gentoos from least to greatest.

2 Objects and Vectors

Create the following vectors in `R`. In some cases, you may be able to use `seq()` or `rep()` while in others you cannot. Use these functions when possible, otherwise manually create the vector and explain why that was necessary.

Some Snowday Notes

There are three ways we will create a vector. Most simply, we can create one from scratch. For example, I create a vector of odds, and evens less than 10 below.

```
(odds <- c(1, 3, 5, 7, 9))
```

```
[1] 1 3 5 7 9
```

```
(evens <- c(2, 4, 6, 8))
```

```
[1] 2 4 6 8
```

There are also built-in functions like `seq(...)` for doing this:

```
(odds <- seq(from=1, to=9, by=2))
```

```
[1] 1 3 5 7 9
```

```
(evens <- seq(from=2, to=8, by=2))
```

```
[1] 2 4 6 8
```

Either approach is easy enough with a small number of elements, but what if I wanted odds less than 100? 1000? 1 million? The `rep(...)` and `seq()` approaches would be far more efficient.

We can also create repeating sequences by hand

```
(repeating.seq1 <- c(1, 2, 3, 1, 2, 3, 1, 2, 3))
```

```
[1] 1 2 3 1 2 3 1 2 3
```

```
(repeating.seq2 <- c(1, 1, 1, 2, 2, 2, 3, 3, 3))
```

```
[1] 1 1 1 2 2 2 3 3 3
```

or using a built-in function `rep(...)`

```
(repeating.seq1 <- rep(c(1,2,3), times=3))
```

```
[1] 1 2 3 1 2 3 1 2 3
```

```
(repeating.seq2 <- rep(c(1,2,3), each=3))
```

```
[1] 1 1 1 2 2 2 3 3 3
```

There are some vectors for which we can't use a `seq()` or `rep()`. For example, consider the prime numbers less than 10. The primes are not sequential (e.g., jump by a fixed amount), nor are they repeating.

```
primes <- c(2, 3, 5, 7)
```

Note: There is a `primes` package for `R` (Keyes and Egeler 2025) that contains a `generate_primes(min, max)` function for generating a vector of primes from `min` to `max`.

2.1 Part a

The Fibonacci Sequence is a recursive formula:

$$F_n = F_{n-1} + F_{n-2}$$

where $F_0 = 0$ and $F_1 = 1$.

Create a vector of the first 10 Fibonacci numbers.

Solution

With what we've learned in class, the best way to create this vector would probably be to list it explicitly using the `c()` function:

```
(fib.c = c(0,1,1,2,3,5,8,13,21,34))
```

```
 [1]  0  1  1  2  3  5  8 13 21 34
```

I was curious about how we could create functions with their outputs stored to vectors, and came up with this:

```
fib = numeric(10)
fib[1] = 0
fib[2] = 1

for (i in 3:10) {
  fib[i] = fib[i-2] + fib[i-1]
}

fib
```

```
 [1]  0  1  1  2  3  5  8 13 21 34
```

2.2 Part b

Triangular Numbers are the cumulative sums of natural numbers:

$$F_n = \frac{n(n+1)}{2}.$$

Create a vector of the first 10 Triangular Numbers.

Again, I do not believe that this is easy to do with the `seq()` and `rep()` functions. We use the following function to store the sequence output to the `tri` vector:

```
tri = numeric(10)

for (i in 1:10) {
  tri[i] = (i)*(i+1)/2
}

tri
```

```
 [1]  1  3  6 10 15 21 28 36 45 55
```

2.3 Part c

Suppose I were designing a repeated meausres experiment with three treatment conditions. Each of $n = 10$ participants (with `ID` 1 to 10) will receive *all* experimental conditions, call them "Control", "Treatment A", and "Treatment B".

Consider setting up data entry for this experiment.

i. Create a vector containing each `ID` repeated three times, once for each treatment.
ii. Create a vector containing each `Treatment` repeated for each participant ID.

Solution

i.

```
(ID = rep(x = c(1:10), each = 3))
```

```
 [1]  1  1  1  2  2  2  3  3  3  4  4  4  5  5  5  6  6  6  7  7  7  8  8  8  9
[26]  9  9 10 10 10
```

ii.

```
(Treatment = rep(c("Control", "Treatment A", "Treatment B"), times = 10))
```

```
 [1] "Control"     "Treatment A" "Treatment B" "Control"     "Treatment A"
 [6] "Treatment B" "Control"     "Treatment A" "Treatment B" "Control"
[11] "Treatment A" "Treatment B" "Control"     "Treatment A" "Treatment B"
[16] "Control"     "Treatment A" "Treatment B" "Control"     "Treatment A"
[21] "Treatment B" "Control"     "Treatment A" "Treatment B" "Control"
[26] "Treatment A" "Treatment B" "Control"     "Treatment A" "Treatment B"
```

2.4 Part d

Create a vector containing the `character` "MATH" and the `numeric` 240. What is the resulting class? Explain why in a sentence.

```
(class = c("MATH", 240))
```

```
[1] "MATH" "240"
```

```
class(class)
```

```
[1] "character"
```

The class is `character`. This is because the `character` class is "lower" than the `numeric` class, so the vector will resolve all inputs to the `character` class.

References

Keyes, Os, and Paul Egeler. 2025. *Primes: Fast Functions for Prime Numbers*. https://doi.org/10.32614/CRAN.package.primes.

Meyer, Fanny, and Victor Perrier. 2025. *Esquisse: Explore and Visualize Your Data Interactively*. https://doi.org/10.32614/CRAN.package.esquisse.

R Core Team. 2025. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.