Class 06

James Woolley (A16440072)

R functions.

Functions can be used to read data, calculate things, and do other computer magic.

R makes wrtiing functions accessible but wewe should always start by trying to get working code before we write.

Lab 6

The goal is to write a function capable of grading a class of student assignments. We will start with a simplified version.

```
# Example input vectors to start with
student1 <- c(100, 100, 100, 100, 100, 100, 100, 90)
student2 <- c(100, NA, 90, 90, 90, 90, 97, 80)
student3 <- c(90, NA, NA, NA, NA, NA, NA, NA)
```

You can calculate the average mean with the `mean()` function, which is modified with the `trim` and `na.rm` modifier thingies, in the format: `mean(student1, trim = 0, na.rm = FALSE)`

```
mean(student1, trim = 0, na.rm = TRUE)
```

```
[1] 98.75
```

That will give the average without trimming any values from either end of x before mean is computed, and will strip all NA values before computation.

If we were to drop the lowest score so the answer should be 100, you could use the `which.min()` function, which identifies the location of the first minimum numeric vector. Thus,

```
student1
```

```
[1] 100 100 100 100 100 100 100  90
```

```
which.min(student1)
```

```
[1] 8
```

This allows you to identify the lowest score from a vector, which can be used in conjunction with `[]` to call the score associated with the location.

```
student1[which.min(student1)]
```

```
[1] 90
```

But this only gives the lowest value, and we want to identify all scores except the lowest, so we can use `-` to remove the lowest value.

```
student1[-which.min(student1)]
```

```
[1] 100 100 100 100 100 100 100
```

Now that we have all scores except the lowest, we can finally calculate the average sans lowest score with the first working piece of code.

```
mean(student1[-which.min(student1)])
```

```
[1] 100
```

This function won't work on student 2, because it is attempting to calculate a mean from a set of data that includes something non-numeric (NA). In order to calculate the average, make the function drop NA values with `na.rm`. This will NOT include NA values in the calculation of the mean.

```
mean(student2, na.rm = TRUE)
```

```
[1] 91
```

The drawback of this function is that student3 ends up with a 90, even though they didn't do most of the homework assignments.

```
mean(student3, na.rm=TRUE)
```

```
[1] 90
```

Doesn't seem fair.

Also we're lazy and typing out `student1-3` over and over sucks so lets use an input called `x`

```
x <- student2
x
```

```
[1] 100  NA  90  90  90  90  97  80
```

Our goal is to overwrite NA values with zero, so that missing homework changes to 0. ChatGPT can give us answers <3

```
x[is.na(x)]
```

```
[1] NA
```

This shows all the NAs in X. You can define all the NAs in X as 0 with:

```
x[is.na(x)] <- 0
x
```

```
[1] 100   0  90  90  90  90  97  80
```

Now the mean is correctly shown as 11.25:

```
student3[is.na(student3)] <- 0
student3
```

```
[1] 90  0  0  0  0  0  0  0
```

```
mean(student3)
```

```
[1] 11.25
```

If we wanted to write a function that calculates the mean after dropping the lowest score and replacing the NAs with 0s so that you can calculate the mean, it would look something like this.

```
#define x so that the function works!
x <- student3
#this sets all the NA values to 0 so that mean won't break when it sees an NA
x[is.na(x)] <- 0
#this takes the mean of the vector sans the lowest score, which includes 0
mean(x[-which.min(x)])
```

```
[1] 12.85714
```

> Q1. Write a function grade() to determine an overall grade from a vector of student homework assignment scores dropping the lowest single score. If a student misses a homework (i.e. has an NA value) this can be used as a score to be potentially dropped. Your final function should be adquately explained with code comments and be able to work on an example class gradebook such as this one in CSV format: "https://tinyurl.com/gradeinput" [3pts]

```
grade <- function(x) {
  #this sets all the NA values to 0 so that mean won't break when it sees an NA
  x[is.na(x)] <- 0
  #this takes the mean of the vector sans the lowest score, which includes 0
  mean(x[-which.min(x)])
}
```

Use this function:

```
grade(student1)
```

```
[1] 100
```

```
grade(student2)
```

```
[1] 91
```

```
grade(student3)
```

```
[1] 12.85714
```

We want to be able to read a gradebook, so

```
gradebook <- read.csv("https://tinyurl.com/gradeinput")
gradebook
```

```
            X hw1 hw2 hw3 hw4 hw5
1   student-1 100  73 100  88  79
2   student-2  85  64  78  89  78
3   student-3  83  69  77 100  77
4   student-4  88  NA  73 100  76
5   student-5  88 100  75  86  79
6   student-6  89  78 100  89  77
7   student-7  89 100  74  87 100
8   student-8  89 100  76  86 100
9   student-9  86 100  77  88  77
10 student-10  89  72  79  NA  76
11 student-11  82  66  78  84 100
12 student-12 100  70  75  92 100
13 student-13  89 100  76 100  80
14 student-14  85 100  77  89  76
15 student-15  85  65  76  89  NA
16 student-16  92 100  74  89  77
17 student-17  88  63 100  86  78
18 student-18  91  NA 100  87 100
19 student-19  91  68  75  86  79
20 student-20  91  68  76  88  76
```

Notice that the first row is "x". We want the names column to be first, so:

```
gradebook <- read.csv("https://tinyurl.com/gradeinput", row.names=1)
gradebook
```

```
           hw1 hw2 hw3 hw4 hw5
student-1  100  73 100  88  79
student-2   85  64  78  89  78
student-3   83  69  77 100  77
student-4   88  NA  73 100  76
student-5   88 100  75  86  79
```

```
student-6   89  78 100  89  77
student-7   89 100  74  87 100
student-8   89 100  76  86 100
student-9   86 100  77  88  77
student-10  89  72  79  NA  76
student-11  82  66  78  84 100
student-12 100  70  75  92 100
student-13  89 100  76 100  80
student-14  85 100  77  89  76
student-15  85  65  76  89  NA
student-16  92 100  74  89  77
student-17  88  63 100  86  78
student-18  91  NA 100  87 100
student-19  91  68  75  86  79
student-20  91  68  76  88  76
```

You can use the `apply` function to apply the function to a matrix. `apply()` has the arguments ARRAY, MARGIN, and FUN. We first specify that we're working with the array gradebook, then we specify that we're working with rows with a 1 (2 would be columns), then we specify that we're working with the function `grade`

(If you wanted to know the average per homework assignment, you would change the MARGIN to 2.)

```
answer1 <- apply(gradebook, 1, grade)
answer1
```

```
 student-1  student-2  student-3  student-4  student-5  student-6  student-7
     91.75      82.50      84.25      84.25      88.25      89.00      94.00
 student-8  student-9 student-10 student-11 student-12 student-13 student-14
     93.75      87.75      79.00      86.00      91.75      92.25      87.75
student-15 student-16 student-17 student-18 student-19 student-20
     78.75      89.50      88.00      94.50      82.75      82.75
```

> Q2. Using your grade() function and the supplied gradebook, Who is the top scoring student overall in the gradebook? [3pts]

I could use my eyes, but because I'm lazy I use the `which.max` function to find which student scored the highest.

```
which.max(apply(gradebook, 1, grade, simplify=TRUE))
```

```
student-18
        18
```

Q3. From your analysis of the gradebook, which homework was toughest on students (i.e. obtained the lowest scores overall? [2pts]

We can use the apply function to calculate the average grade obtained for each individual homework assignment by changing the MARGIN from 1 to 2 (rows to columns). This will show the average score for each.

```
answer3 <- apply(gradebook, 2, grade)
answer3
```

```
     hw1      hw2      hw3      hw4      hw5
89.36842 76.63158 81.21053 89.63158 83.42105
```

Then you can use `which.min` to find which one was the lowest.

```
which.min(answer3)
```

```
hw2
  2
```

This gives us the answer, HW 2.