Data Types, Arithmetic Functions

Set Working Directory and Clear the Environment

```
rm(list=ls())
setwd("C:/Users/19107/Desktop/R Stuff/2023 or Earlier/Recreation")
```

Libraries

```
library(ggplot2)
library(haven)
library(patchwork)
library(gridExtra)
library(dplyr)
library(stargazer)
library(countrycode)
library(stringi)
library(devtools)
```

Data Generation and Types

Generate a Variable

You can use the = symbol to "assign" a value to a variable or you can use the <- symbol for the same purpose, I use =

NOTE if you want to use the logical operator "equal to" you must use == otherwise R will think you are assigning not logically evaluating.

Value Variable of type "numeric"

```
x = 5
x == 5
```

```
[1] TRUE
```

Some simiple math R uses most of the conventional forms of operators e.g. /, *, +, -

```
124/12
```

```
[1] 10.33333
```

```
100-90
```

```
[1] 10
```

```
45*197
```

```
[1] 8865
```

```
345.32+1098.00087
```

```
[1] 1443.321
```

```
x*7+3-10/2
```

```
[1] 33
```

Value Variable of type "character" or "string"

```
x = "five"
x
```

```
[1] "five"
```

```
x = "Tyler is the greatest TA there has ever been"
x
```

```
[1] "Tyler is the greatest TA there has ever been"
```

More complex data forms

Vector Variable

```
x = c(1, 3, 5, 7)
x = 1:7
x = rep(5,5) # rep(repeat this, this many times)
x = seq(1,7,.2) # seq(start, stop, interval)
```

Since we have a vector we can now do a few more complex math operations

```
median(x)
```

```
[1] 4
```

```
mean(x)
```

```
[1] 4
```

```
sd(x) # Standard Deviation
```

```
[1] 1.818424
```

Matrix Variable

```
x = matrix(data = c(1, 2, 3, 4, 5, 6),
           ncol= 2,
           nrow= 3,
           byrow= TRUE
```

```
)
x
```

```
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

"byrow" on vs off

```
x = matrix(data = c(1, 2, 3, 4, 5, 6),
           ncol= 2,
           nrow= 3,
           byrow= FALSE

)
x
```

```
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Dataframe

A type you all may be closely familiar with, a collection of some number of variables, most of our data comes as a dataframe.

```
id= 1:10
grade= rep("A", 10)
name = c("Alex", "Rael", "Furkan", "Se Yoon", "Mohsin", "Shannon",
 ↪  "Burran", "Kathleen", "Kaan", "Alice")

x = data.frame(id,
               grade,
               name,
               stringsAsFactors = FALSE)
x
```

```
   id grade     name
1   1     A     Alex
2   2     A     Rael
3   3     A   Furkan
4   4     A  Se Yoon
5   5     A   Mohsin
6   6     A  Shannon
7   7     A   Burran
8   8     A Kathleen
9   9     A     Kaan
10 10     A    Alice
```

You can add new variables to your dataframe or call on existing variables using the $ operator

```
# Call an Existing Variable
x$grade
```

```
 [1] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
```

```
# New Variable
x$absences = c(0,15,6,0,0,0,0,8,0,0)
```

Indexing

you can call on specific elements of a variable by indexing from a list/vector using the [] and numbers for the relevant position

```
y= 1:7
y[2]
```

```
[1] 2
```

You can also use indexing to replace elements of a variable

```
x$grade[2]="C"
x
```

```
   id grade     name absences
1   1     A     Alex        0
2   2     C     Rael       15
3   3     A   Furkan        6
4   4     A  Se Yoon        0
5   5     A   Mohsin        0
6   6     A  Shannon        0
7   7     A   Burran        0
8   8     A Kathleen        8
9   9     A     Kaan        0
10 10     A    Alice        0
```

Indexing a Matrix/Dataframe

```
x = matrix(data = c(1, 2, 3, 4, 5, 6),
           ncol= 3,
           nrow= 2,
           byrow= TRUE

)
x
```

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

```
# in a matrix row comes first then column, if you leave either blank
 ↪  after the comma the whole row/column is returned

x[1,2]
```

```
[1] 2
```

```
x[1,]
```

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

```r
x[,2]
```

```
[1] 2 5
```

```r
# Indexing will also allow you to change elements in a matrix just like
 ↪  with a vector or list

x[1,2]=1000
x
```

```
     [,1] [,2] [,3]
[1,]    1 1000    3
[2,]    4    5    6
```

```r
# indexing works on dataframes as well but often using
# the $ operator is easier,
# also you can apply indexing in conjunction with the $ operator

id= 1:10
grade= rep("A", 10)
name = c("Alex", "Rael", "Furkan", "Se Yoon", "Mohsin", "Shannon",
 ↪  "Burran", "Kathleen", "Kaan", "Alice")

x = data.frame(id,
               grade,
               name,
               stringsAsFactors = FALSE)
# These are Equivalent

x[1,3]
```

```
[1] "Alex"
```

```r
x$name[1]
```

```
[1] "Alex"
```