```
# Assignment: ASSIGNMENT 0
# Name: Koby-Hercsky, Theodore
# Date: 2021-03-22

# Basics

## Add 8 and 5
8+5
#[1] 13
## Subtract 6 from 22
22-6
#[1] 16
## Multiply 6 by 7
6*7
#[1] 42
## Add 4 to 6 and divide the result by 2
(6+4)/2
#[1] 5
## Compute 5 modulo 2
5%%2
#[1] 1
## Assign the value 82 to the variable x
## Print x
x <-82
print(x)
#[1] 82
## Assign the value 41 to the variable y
## Print y
y <-41
print(y)
#[1] 41
## Assign the output of x + y to the variable z
## Print z
z <-(x+y)
>print(z)
#[1] 123
## Assign the string value "DSC520" to the variable class_name
## Print the value of class_name
class_name <- assign("DSC520",123)
print(DSC520)
## Assign the string value of TRUE to the variable is_good
## Print the value of is_good
is_good <-TRUE
print(is_good)
#[1] TRUE
## Check the class of the variable is_good using the `class()` function
class(is_good)
#[1] "logical"
## Check the class of the variable z using the `class()` function
class(z)
#[1] "numeric"
## Check the class of the variable class_name using the class() function
class(class_name)
#[1] "numeric"
```

```
# Assignment: ASSIGNMENT 1
# Name: Koby-Hercsky, Theodore
# Date: 2021-03-22

## Create a numeric vector with the values of 3, 2, 1 using the `c()` function
## Assign the value to a variable named `num_vector`
## Print the vector
num_vector<-c(3,2,1)
print(num_vector)
#[1] 3 2 1
## Create a character vector with the values of "three", "two", "one" "using the `c()` function
## Assign the value to a variable named `char_vector`
## Print the vector
char_vector <-c("three", "two", "one")
print(char_vector)
#[1] "three" "two"   "one"
## Create a vector called `week1_sleep` representing how many hours slept each night of the week
## Use the values 6.1, 8.8, 7.7, 6.4, 6.2, 6.9, 6.6
week1_sleep <-c(6.1, 8.8, 7.7, 6.4, 6.2, 6.9, 6.6)

## Display the amount of sleep on Tuesday of week 1 by selecting the variable index
week1_sleep[2]
#[1] 8.8
## Create a vector called `week1_sleep_weekdays`
## Assign the weekday values using indice slicing
week1_sleep_weekdays <- week1_sleep[1:5]
names(Week1_sleep) <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")
## Add the total hours slept in week one using the `sum` function
## Assign the value to variable `total_sleep_week1`
sum(week1_sleep)
#[1] 48.7
total_sleep_week1 <-assign("week1_sleep",48.7)
## Create a vector called `week2_sleep` representing how many hours slept each night of the week
## Use the values 7.1, 7.4, 7.9, 6.5, 8.1, 8.2, 8.9
week2_sleep <-c(7.1, 7.4, 7.9, 6.5, 8.1, 8.2, 8.9)
## Add the total hours slept in week two using the sum function
## Assign the value to variable `total_sleep_week2`
sum(week2_sleep)
#[1] 54.1
total_sleep_week2 <-assign("week2_sleep",54.1)
## Determine if the total sleep in week 1 is less than week 2 by using the < operator
week1_sleep<week2_sleep
#[1] TRUE
## Calculate the mean hours slept in week 1 using the `mean()` function
mean(week1_sleep, na.rm = TRUE)
#[1] 48.7
## Create a vector called `days` containing the days of the week.
## Start with Sunday and end with Saturday
days <-c("Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday")

## Assign the names of each day to `week1_sleep` and `week2_sleep` using the `names` function and `days` vector
names(week1_sleep) <-c(days)
names(week2_sleep) <-c(days)

## Display the amount of sleep on Tuesday of week 1 by selecting the variable name
week1_sleep[3]
Tuesday
7.7
## Create vector called weekdays from the days vector
weekdays <- days[2:6]

## Create vector called weekends containing Sunday and Saturday
weekends <- days[c(1,7)]
```

```
## Calculate the mean about sleep on weekdays for each week
## Assign the values to weekdays1_mean and weekdays2_mean
weekdays1_mean <- mean(week1_sleep[weekdays])
weekdays2_mean <- mean(week2_sleep[weekdays])
mean(weekdays1_mean, na.rm = TRUE)
#[1] 7.2
mean(weekdays2_mean, na.rm = TRUE)
#[1] 7.62
## Using the weekdays1_mean and weekdays2_mean variables,
## see if weekdays1_mean is greater than weekdays2_mean using the `>` operator
weekdays1_mean>weekdays2_mean
#[1] FALSE

## Determine how many days in week 1 had over 8 hours of sleep using the `>` operator
week1_sleep>8.00
#Sunday    Monday   Tuesday Wednesday  Thursday    Friday  Saturday
#FALSE      TRUE     FALSE     FALSE     FALSE     FALSE     FALSE

## Create a matrix from the following three vectors
student01 <- c(100.0, 87.1)
student02 <- c(77.2, 88.9)
student03 <- c(66.3, 87.9)
thedf<-data.frame(student01, student02, student03)
thedf
#student01 student02 student03
#1     100.0      77.2      66.3
#2      87.1      88.9      87.9
students_combined <-c(student01, student02, student03)
grades <- matrix(c(students_combined), ncol =3)
## Add a new student row with `rbind()'
student04 <- c(95.2, 94.1)
cbind(thedf, student04)
## Add a new assignment column with `cbind()`
assignment04 <- c(92.1, 84.3, 75.1, 97.8)
cbind(thedf, student04)
#student01 student02 student03 student04
#1     100.0      77.2      66.3      95.2
#2      87.1      88.9      87.9      94.1
## Add the following names to columns and rows using `rownames()` and `colnames()`
assignments <- c("Assignment 1", "Assignment 2", "Assignment 3")
students <- c("Florinda Baird", "Jinny Foss", "Lou Purvis", "Nola Maloney")

rownames(thedf)<-c("Assignment 1", "Assignment 2")
colnames(thedf)<-c("Florinda Baird", "Jinny Foss", "Lou Purvis", "Nola Maloney")
#Florinda.Baird Jinny.Foss Lou.Purvis Nola.Maloney
#1          100.0       77.2       66.3         95.2
#2           87.1       88.9       87.9         94.1
## Total points for each assignment using `colSums()`
colSums(students_combined)

## Total points for each student using `rowSums()`
rowSums(students_combined)

## Matrix with 10% and add it to grades
weighted_grades <- grades * 0.1 + grades
weighted_grades <- grades * 0.1 + students_combined
#[,1]  [,2]  [,3]  [,4]  [,5]  [,6]
#[1,]  110 95.81 84.92 97.79 72.93 96.69
#Florinda.Baird Jinny.Foss Lou.Purvis Nola.Maloney
#1          110.00      84.92      72.93       104.72
#2           95.81      97.79      96.69       103.51
## Create a factor of book genres using the genres_vector
## Assign the factor vector to factor_genre_vector
genres_vector <- c("Fantasy", "Sci-Fi", "Sci-Fi", "Mystery", "Sci-Fi", "Fantasy")
```

```
factor_genre_vector <- assign("genre_vector",TRUE)
## Use the `summary()` function to print a summary of `factor_genre_vector`
summary(factor_genre_vector)
#Mode    TRUE
#logical       1
## Create ordered factor of book recommendations using the recommendations_vector
## `no` is the lowest and `yes` is the highest
recommendations_vector <- c("neutral", "no", "no", "neutral", "yes")
factor_recommendations_vector <- factor(recommendations_vector, ordered = TRUE, levels = c("no", "neutral",
"yes"))

## Use the `summary()` function to print a summary of `factor_recommendations_vector`
summary(factor_recommendations_vector)

## Using the built-in `mtcars` dataset, view the first few rows using the `head()` function
head(factor_recommendations_vector, 2)

## Using the built-in mtcars dataset, view the last few rows using the `tail()` function
tail(factor_recommendations_vector)

## Create a dataframe called characters_df using the following information from LOTR
name <- c("Aragon", "Bilbo", "Frodo", "Galadriel", "Sam", "Gandalf", "Legolas", "Sauron", "Gollum")
race <- c("Men", "Hobbit", "Hobbit", "Elf", "Hobbit", "Maia", "Elf", "Maia", "Hobbit")
in_fellowship <- c(TRUE, FALSE, TRUE, FALSE, TRUE, TRUE, TRUE, FALSE, FALSE)
ring_bearer <- c(FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, TRUE)
age <- c(88, 129, 51, 7000, 36, 2019, 2931, 7052, 589)

characters_df <- data.frame(name, race, in_fellowship, ring_bearer, age)

## Sorting the characters_df by age using the order function and assign the result to the sorted_characters_df
sorted_characters_df <- characters_df[order(characters_df$age, decreasing = TRUE),]
## Use `head()` to output the first few rows of `sorted_characters_df`
head(sorted_characters_df)

## Select all of the ring bearers from the dataframe and assign it to ringbearers_df
ringbearers_df <- characters_df[characters_df$ring_bearer == TRUE,]
## Use `head()` to output the first few rows of `ringbearers_df`
head(ringbearers_df, 3)
```

```
# Assignment: ASSIGNMENT 2
# Name: Koby-Hercsky, Theodore
# Date: 2021-03-25

## Check your current working directory using `getwd()`
getwd()

## List the contents of the working directory with the `dir()` function
dir()
#[1] "dsc520" "GitHub"
## If the current directory does not contain the `data` directory, set the
## working directory to project root folder (the folder should contain the `data` directory
## Use `setwd()` if needed
etwd("~/Downloads/DSC520/dsc520/data")

## Load the file `data/tidynomicon/person.csv` to `person_df1` using `read.csv`
## Examine the structure of `person_df1` using `str()`
person_df1 <-read.csv('person.csv', header = TRUE, sep = ",")
str(person_df1)
## R interpreted names as factors, which is not the behavior we want
## Load the same file to person_df2 using `read.csv` and setting `stringsAsFactors` to `FALSE`
## Examine the structure of `person_df2` using `str()`
person_df2 <-read.csv('person.csv', stringsAsFactors = FALSE, sep = ",")
str(person_df2)
## Read the file `data/scores.csv` to `scores_df`
## Display summary statistics using the `summary()` function
scores_df <-read.csv('scores.csv', stringsAsFactors = FALSE, sep = ",")
summary(scores_df)
## Load the `readxl` library
library(`readxl`)

## Using the excel_sheets() function from the `readxl` package,
## list the worksheets from the file `data/G04ResultsDetail2004-11-02.xls`
excel_sheets('G04ResultsDetail2004-11-02.xls')

## Using the `read_excel` function, read the Voter Turnout sheet
## from the `data/G04ResultsDetail2004-11-02.xls`
## Assign the data to the `voter_turnout_df1`
## The header is in the second row, so make sure to skip the first row
## Examine the structure of `voter_turnout_df1` using `str()`
read_excel('G04ResultsDetail2004-11-02.xls', sheet = "Voter Turnout", skip = 1)
# created a data frame and excluded the first row from the voter turnout data sheet
voter_turnout_df1 <- data.frame(read_excel('G04ResultsDetail2004-11-02.xls', sheet = "Voter Turnout", skip = 1))

## Using the `read_excel()` function, read the Voter Turnout sheet
## from `data/G04ResultsDetail2004-11-02.xls`
## Skip the first two rows and manually assign the columns using `col_names`
## Use the names "ward_precint", "ballots_cast", "registered_voters", "voter_turnout"
## Assign the data to the `voter_turnout_df2`
## Examine the structure of `voter_turnout_df2` using `str()`
# Created a vector titled Voter_Vector
voter_vector <- c("ward_precint", "ballots_cast", "registered_voters", "voter_turnout")
# Read the voter turnout and removed the first two rows and inserted the voter_vector
read_excel('G04ResultsDetail2004-11-02.xls', sheet = "Voter Turnout", skip = 2, col_names = voter_vector)
# Created the datat frame from the voter turnout sheet and removed the first two rows and inserted the Voter_vector
voter_turnout_df2 <- data.frame(read_excel('G04ResultsDetail2004-11-02.xls', sheet = "Voter Turnout", skip = 2,
col_names = voter_vector))

## Load the `DBI` library
library(DBI)

## Create a database connection to `data/tidynomicon/example.db` using the dbConnect() function
## The first argument is the database driver which in this case is `RSQLite::SQLite()`
## The second argument is the path to the database file
## Assign the connection to `db` variable
# Use the library to connect to RSQLite
library(RSQLite)
# Use dbdrive to specify the driver SQLite
drv <- dbDriver('SQLite')
class(drv)
# Establish connection using dbconnect to connect to the example database
db <- dbConnect(drv, 'example.db')

## Query the Person table using the `dbGetQuery` function and the
## `SELECT * FROM PERSON;` SQL statement
## Assign the result to the `person_df` variable
## Use `head()` to look at the first few rows of the `person_df` dataframe
person_df <- dbGetQuery(db, "SELECT * Person", stringsAsFactors=FALSE)
head(person_df)

## List the tables using the `dbListTables()` function
## Assign the result to the `table_names` variable
dbListTables(db)
table_names <- dbListTables(db)

## Read all of the tables at once using the `lapply` function and assign the result to the `tables` variable
## Use `table_names`, `dbReadTable`, and `conn = db` as arguments
## Print out the tables
lapply(X = db, FUN = table_names)
print(tables)
## Use the `dbDisconnect` function to disconnect from the database
dbDisconnect(db)

## Import the `jsonlite` library
library(jsonlite)

## Convert the scores_df dataframe to JSON using the `toJSON()` function
scores_df <-toJSON(scores_df)

## Convert the scores dataframe to JSON using the `toJSON()` function with the `pretty=TRUE` option
scores_df <-toJSON(scores_df, pretty = TRUE)
```