A. Introduction to R

- # sign for comments
- Arithmetic operators: +, -, *, /, ^, %% (modulo)
- Variables allow you to store a value (e.g. 4) or an object (e.g. a function description)

```
> # Assign the value 42 to x
> x <- 42
> # Print out the value of x
> x
>
> # Assign a value to the variables my_apples and my_oranges
> my_apples <- 5
> my_oranges <- 6
>
> # Add these two variables together
> my_apples + my_oranges
>
> # Create the variable my_fruit
> my_fruit <- my_apples + my_oranges
```

- **Numerics** are decimals (e.g. 4.5)
- **Integers** are natural numbers
- **Logical** values are Boolean
- **Characters** are string values, (e.g.”this is a string”)

```
> # Set my_numeric to be 42
> my_numeric <- 42
>
> # Set my_character to be "universe"
> my_character <- "universe"
>
> # Set my_logical to be FALSE
> my_logical <- FALSE
```

- Check the data type of a variable by using: **class()**

```
> # Check class of my_numeric
> class(my_numeric)
“numeric”
> # Check class of my_character
```

```
> class(my_character)
“character”
> # Check class of my_logical
> class(my_logical)
“logical”
```

Vectors

- **Vectors** are 1-d arrays that hold numeric, character, or logical data (think python list but of one data type)
    - Create vectors using combine function **c()**

```
> numeric_vector <- c(1, 10, 49)
> character_vector <- c("a", "b", "c")
>
> # Complete the code for boolean_vector
> boolean_vector <- c(TRUE, FALSE, TRUE)
```

- You can name the elements of a vector using **names()** function

```
> vector_a <- c(“John Doe”, “poker player”
> names( vector_a) <- c(“Name”, “Profession”)
```

Exercise - Casino winnings

```
> # Poker winnings & Roulette winnings from Monday to Friday
> poker_vector <- c(140, -50, 20, -120, 240)
> roulette_vector <- c(-24, -50, 100, -350, 10)
>
> # A. Assign days as names of poker_vector, roulette_vectors
> names(poker_vector) <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
> names(roulette_vector) <- c("Monday", "Tuesday", "Wednesday", "Thursday",
      ( "Friday")
>
> # B. Assign days as names of poker_vector, roulette_vectors using created variable
> days_vector <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")
> names(poker_vector) <- days_vector
> names(roulette_vector) <- days_vector
>
> roulette_vector
```

```
Monday   Tuesday Wednesday  Thursday    Friday
   -24         -50         100          -350          10
>
> total_daily <- roulette_vector+poker_vector
> total_daily
Monday   Tuesday Wednesday  Thursday    Friday
   116         -100         120          -470         250
```

- Adding vectors will take the element-wise sum:

```
> A_vector <- c(1, 2, 3)
> B_vector <- c(4, 5, 6)
> # Take the sum of A_vector and B_vector
> total_vector <- A_vector + B_vector
> total_vector
5 7 9
```

- To find the sum of the elements in a vector, use sum()

```
> total_poker <- sum(poker_vector)
> total_roulette <- sum(roulette_vector)
> # Check to see if poker winnings are greater than roulette winnings
> total_poker > total_roulette
TRUE
```

- Selecting element of a vector, matrix, data frame, etc: use **[ ]** (note: first element has index 1, not 0)

```
> # Define a new variable based on a selection
> poker_wednesday <- poker_vector[3]
```

- Select multiple elements of vector, matrix, data frame, etc: use **[c( )]**

```
> # Define a new variable based on a selection of Tuesday, Wednesday, Thursday
> poker_midweek <- poker_vector[c(2,3,4)]
```

- Select multiple elements using splicing "**:**"

```
> # Define a new variable based on a selection of Tuesday to Friday
> roulette_selection_vector <-roulette_vector[2:5]
```

- Select multiple elements using their names

```
> # Select poker results for Monday, Tuesday and Wednesday
> poker_start <- poker_vector[c("Monday", "Tuesday", "Wednesday")]
```

- Calculate the average of a vector using **mean()**

```
> mean(poker_start)
36.66667
```

- Comparison operators:
    - These command operators return TRUE or FALSE

<	>	<=	>=	==	!=

```
> # Which days did you make money on poker?
> selection_vector <- poker_vector>0
>
> # Print out selection_vector
> selection_vector
Monday   Tuesday Wednesday  Thursday    Friday
  TRUE     FALSE      TRUE      FALSE      TRUE
```

- R will select only elements that are TRUE when a logical vector is passed in square brackets

```
> # Which days did you make money on poker?
> selection_vector <- poker_vector > 0
>
> # Select from poker_vector these days
> poker_winning_days <- poker_vector[selection_vector]
>
> poker_winning_days
Monday Wednesday    Friday
  140          20       240
```

Matrices

- A matrix is a collection of elements of the same data type, arranged into a fixed number of rows and columns
- Construct a matrix with **matrix()** function
    - **First argument** is the collection of elements to be arranged into rows and columns
    - Argument **byrow** indicates that the matrix is filled by rows, TRUE or FALSE (FALSE for matrix filled by columns)
    - Argument **nrow** indicates how many rows in the matrix

```
> matrix(1:9, byrow = TRUE, nrow = 3)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9

> # Box office Star Wars (in millions!)
> new_hope <- c(460.998, 314.4)
> empire_strikes <- c(290.475, 247.900)
> return_jedi <- c(309.306, 165.8)
>
> # Create box_office
> box_office <- c(new_hope, empire_strikes, return_jedi)
>
> # Construct star_wars_matrix
> star_wars_matrix <- matrix(box_office, byrow=TRUE, nrow=3)
      [,1]     [,2]
[1,] 460.998 314.4
[2,] 290.475 247.9
[3,] 309.306 165.8
```

- Adding row and column names
    - **rownames(**matrix**)** <- row_names_vector
    - **colnames(**matrix**)** <- col_names_vector

```
> # Vectors region and titles, used for naming
> region <- c("US", "non-US")
> titles <- c("A New Hope", "The Empire Strikes Back", "Return of the Jedi")
>
> # Name the columns with region
> colnames(star_wars_matrix) <- region
>
> # Name the rows with titles
> rownames(star_wars_matrix) <- titles
>
```

```
> # Print out star_wars_matrix
> Star_wars_matrix
                           US     non-US
A New Hope                460.998   314.4
The Empire Strikes Back   290.475   247.9
Return of the Jedi        309.306   165.8
```

- **rowSums(**matrix**)** calculates the totals for each row of a matrix, creating a new vector as the result

```
> # Construct star_wars_matrix
> box_office <- c(460.998, 314.4, 290.475, 247.900, 309.306, 165.8)
> star_wars_matrix <- matrix(box_office, nrow = 3, byrow = TRUE,
          dimnames = list(c("A New Hope", "The Empire Strikes Back",
          "Return of the Jedi"),  c("US", "non-US")))
>
> # Calculate worldwide box office figures
> worldwide_vector <- rowSums(star_wars_matrix)
> worldwide_vector
>
> A New Hope   The Empire Strikes Back      Return of the Jedi
          775.398               538.375               475.106
```

- Adding columns to matrix using **cbind()**
    - Merges matrices and/or vectors together by column
    - Big_matrix = cbind(matrix1, matrix2, vector1...)

```
> # Bind the new variable worldwide_vector as a column to star_wars_matrix
> all_wars_matrix <- cbind(star_wars_matrix, worldwide_vector)

                           US       non-US     worldwide_vector
A New Hope                460.998  314.4         775.398
The Empire Strikes Back   290.475  247.9         538.375
Return of the Jedi        309.306  165.8         475.106
```

- Adding rows to matrix with **rbind**()

```
> star_wars_matrix2
                          US    non-US
The Phantom Menace   474.5  552.5
Attack of the Clones      310.7  338.7
Revenge of the Sith        380.3  468.5
> # Combine both Star Wars trilogies in one matrix
```

```
> all_wars_matrix <- rbind(star_wars_matrix, star_wars_matrix2)
> all_wars_matrix
                           US    non-US
A New Hope                 461.0  314.4
The Empire Strikes Back    290.5  247.9
Return of the Jedi         309.3  165.8
The Phantom Menace         474.5  552.5
Attack of the Clones       310.7  338.7
Revenge of the Sith        380.3  468.5
```

- **colSums()** to calculate totals for each column of a matrix

```
> # Total revenue for US and non-US
> total_revenue_vector <- colSums(all_wars_matrix)
>
> # Print out total_revenue_vector
> total_revenue_vector
  US      non-US
2226.3   2087.8
```

- Selection of matrix elements using brackets
    - My_matrix[1,2] selects element at the first row and second column
    - My_matrix[1:3, 2:4] results in a matrix with the data on the rows 1, 2, 3 and columns 2, 3, 4
    - My_matrix[,1] selects all elements of the first column
    - My_matrix[1,] selects all elements of the first row

```
> # Select the non-US revenue for all movies
> non_us_all <- all_wars_matrix[,2]
>
> # Average non-US revenue
> mean(non_us_all)
[1] 347.9667
> # Select the non-US revenue for first two movies
> non_us_some <- all_wars_matrix[1:2, 2]
>
> # Average non-US revenue for first two movies
> mean(non_us_some)
[1] 281.15
```

- Arithmetic on matrices
    - 2 * my_matrix will multiply each element of the matrix by 2
    - my_matrix/10 will divide each element of the matrix by 10

- My_matrix1 * my_matrix2 will create a matrix where each element is a product of the corresponding elements in the two matrices
  - Not the classic multiplication of matrices

```
> # Estimate the visitors, assuming each ticket is $5
> visitors <- all_wars_matrix/5
>
> # Print the estimate to the console
> visitors
                           US non-US
A New Hope                 92.20  62.88
The Empire Strikes Back 58.10  49.58
Return of the Jedi         61.86  33.16
The Phantom Menace     94.90 110.50
Attack of the Clones       62.14  67.74
Revenge of the Sith        76.06  93.70

> ticket_prices_matrix
                           US non-US
A New Hope                  5.0    5.0
The Empire Strikes Back   6.0    6.0
Return of the Jedi          7.0    7.0
The Phantom Menace       4.0    4.0
Attack of the Clones        4.5    4.5
Revenge of the Sith         4.9    4.9

> # Estimated number of visitors
> visitors <- all_wars_matrix/ticket_prices_matrix
>
> # US visitors
> us_visitors <- visitors[,1]
>
> # Average number of US visitors
> mean(us_visitors)
[1] 75.01401
```

Factors

- Factor is a data type used to store categorical variables (limited number of categories)
- To create a factor, use function **factor()**
- Factor levels are also known as the number of categories

```
> # Sex vector
> sex_vector <- c("Male", "Female", "Female", "Male", "Male")
>
> # Convert sex_vector to a factor
```

```
> factor_sex_vector <- factor(sex_vector)
>
> # Print out factor_sex_vector
> factor_sex_vector
[1] Male Female Female Male Male
Levels: Female Male
```

- Nominal categorical variables: categorical variable without an implied order
    - Ex: categories are “Elephant”, “Giraffe”, and “Cow”
- Ordinal categorical variables: categorical variables with natural ordering
    - Ex: categories are “Low”, “Medium”, and “High”
- Change the names of factor levels using **levels()**
    - levels(factor_vector) <- c(“name1”, “name2”,...)

```
> # Code to build factor_survey_vector
> survey_vector <- c("M", "F", "F", "M", "M")
> factor_survey_vector <- factor(survey_vector)
>
> # Specify the levels of factor_survey_vector
> levels(factor_survey_vector) <-c("Female", "Male")
>
> factor_survey_vector
[1] Male Female Female Male Male
Levels: Female Male
```

- Using **summary()** to see a quick overview of the contents of a variable

```
> summary(factor_survey_vector)
Female   Male
    2      3
```

- Ordered factors using **factor()** - need argument “ordered = TRUE” and levels = “...“

```
> # Create speed_vector
> speed_vector <- c("medium", "slow", "slow", "medium", "fast")
>
> # Convert speed_vector to ordered factor vector
> factor_speed_vector <- factor(speed_vector, ordered = TRUE, levels = c("slow", "medium",
"fast"))
>
> # Print factor_speed_vector
> factor_speed_vector
```

```
[1] medium slow   slow   medium fast
Levels: slow < medium < fast
>
> summary(factor_speed_vector)
  slow medium   fast
     2      2      1
>
> # Factor value for second data analyst
> da2 <-factor_speed_vector[2]
>
> # Factor value for fifth data analyst
> da5 <- factor_speed_vector[5]
>
> # Is data analyst 2 faster than data analyst 5?
> da2 > da5
[1] FALSE
```

Data Frames

- Has variables of a data set as columns and observations as rows, can hold different data types
- Example data frame:

```
> # Print out built-in R data frame
> mtcars
                   mpg  cyl  disp   hp    drat    wt   qsec  vs am gear carb
Mazda RX4          21.0   6  160.0  110   3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag      21.0   6  160.0  110   3.90 2.875 17.02  0  1    4    4
Datsun 710         22.8   4  108.0   93   3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive     21.4   6  258.0  110   3.08 3.215 19.44  1  0    3    1
Hornet Sportabout  18.7   8  360.0  175   3.15 3.440 17.02  0  0    3    2
Valiant            18.1   6  225.0  105   2.76 3.460 20.22  1  0    3    1
Duster 360         14.3   8  360.0  245   3.21 3.570 15.84  0  0    3    4
Merc 240D          24.4   4  146.7   62   3.69 3.190 20.00  1  0    4    2
```

- Looking at the top and end of a data frame - **head()** and t**ail()**
- Looking at the structure of a data frame - **str()**
- Creating a data frame
    - Function to create a dataframe: **data.frame()**, pass all the vectors to be included as arguments

```
> # Definition of vectors
> name <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn", "Uranus", "Neptune")
> type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",
```

```
         "Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")
> diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)
> rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)
> rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)
>
> # Create a data frame from the vectors
> planets_df <-data.frame(name, type, diameter, rotation, rings)
>
> planets_df
     name       type                diameter  rotation  rings
1 Mercury    Terrestrial planet    0.382    58.64     FALSE
2  Venus     Terrestrial planet    0.949    -243.02  FALSE
3  Earth     Terrestrial planet    1.000     1.00     FALSE
4  Mars      Terrestrial planet    0.532     1.03     FALSE
5 Jupiter     Gas giant            11.209    0.41     TRUE
6  Saturn     Gas giant             9.449    0.43     TRUE
7  Uranus     Gas giant             4.007    -0.72    TRUE
8 Neptune    Gas giant             3.883    0.67     TRUE
>
> # Check the structure of planets_df
> str(planets_df)
'data.frame':    8 obs. of  5 variables:
 $ name    : Factor w/ 8 levels "Earth","Jupiter",..: 4 8 1 3 2 6 7 5
 $ type    : Factor w/ 2 levels "Gas giant","Terrestrial planet": 2 2 2 2 1 1 1 1
 $ diameter: num  0.382 0.949 1 0.532 11.209 ...
 $ rotation: num  58.64 -243.02 1 1.03 0.41 ...
 $ rings   : logi  FALSE FALSE FALSE FALSE TRUE TRUE ...
>
> # Print out diameter of Mercury (row 1, column 3)
> planets_df[1,3]
[1] 0.382
>
> # Print out data for Mars (entire fourth row)
> planets_df[4,]
   name   type                diameter rotation rings
4 Mars   Terrestrial planet   0.532     1.03    FALSE
>
> # Select first 5 values of diameter column
> planets_df[1:5, "diameter"]
[1]  0.382  0.949  1.000  0.532 11.209
>
> # Select the rings variable from planets_df
> rings_vector <- planets_df$rings
>
> # Print out rings_vector
> rings_vector
[1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
```

- Call columns of a dataframe by name using **$**
    - Format: dataframe$column
- Using **subset()** to subset a dataframe
    - subset(my_df, subset = some_condition)

```
> # Select planets with diameter < 1
> subset(planets_df, subset=diameter <1)
   name       type              diameter rotation   rings
1 Mercury Terrestrial planet    0.382    58.64      FALSE
2 Venus   Terrestrial planet    0.949    -243.02    FALSE
4  Mars   Terrestrial planet    0.532     1.03      FALSE
```

- **order()** is a function that gives the ranked position of each element when it is applied on a variable
    - To reshuffle the order of a, use **a[order(a)]**

```
> # Use order() to create positions
>positions <-  order(planets_df$diameter)
>
> # Use positions to sort planets_df
> planets_df[positions, ]
    name            type          diameter  rotation   rings
1 Mercury   Terrestrial planet    0.382     58.64    FALSE
4 Mars      Terrestrial planet    0.532     1.03     FALSE
2 Venus     Terrestrial planet    0.949     -243.02 FALSE
3 Earth     Terrestrial planet    1.000     1.00     FALSE
8 Neptune   Gas giant             3.883     0.67     TRUE
7 Uranus    Gas giant             4.007     -0.72    TRUE
6 Saturn    Gas giant             9.449     0.43     TRUE
5 Jupiter   Gas giant            11.209     0.41      TRUE
```

Lists

- Lists allow you to gather a variety of objects under one name
- Use the function **list()**
    - My_list <- list(comp1, comp2…)
    - Name each component in the list using format: list(name1 = comp1, name2=comp2…)
    - Or by using names(my_list) <- c(“name1”, “name2”)

```
> # Vector with numerics from 1 up to 10
> my_vector <- 1:10
>
```

```
> # Matrix with numerics from 1 up to 9
> my_matrix <- matrix(1:9, ncol = 3)
>
> # First 10 elements of the built-in data frame mtcars
> my_df <- mtcars[1:10,]
>
> # Construct list with these different elements:
> my_list <- list(my_vector, my_matrix, my_df)
>
> # Adapt list() call to give the components names
> my_list <- list(vec=my_vector, mat=my_matrix, df=my_df)
>
># Print out my_list
>my_list
```

- Selecting elements from list - since lists can be built out of numerous elements and components, selecting a single element is not straightforward
    - Selecting by numbered position of the component:
        - Double brackets - My_list[[1]]
        - My_list[[2]][1] - will grab the second component and its first element
    - Selecting by $ sign and name
        - My_list$element1
- Adding elements to lists using **c()**
    - Ext_list <- c(my_list, my_name=my_val)

B. Intermediate R

Relational Operators

- Equality ==
- Inequality !=
- < and >, check for strings in alphabetical order
- TRUE = 1, FALSE = 0
- Comparing vectors to answer questions like:
    - On which days did the number of LinkedIn profile views exceed 15?
    - When was your LinkedIn profile viewed only 5 times or fewer?
    - When was your LinkedIn profile visited more often than your Facebook profile?

```
> # Linkedin and Facebook vectors
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
> facebook <- c(17, 7, 5, 16, 8, 13, 14)
>
> # Popular days
> linkedin>15
```

```
[1]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
>
> # Quiet days
> linkedin<=5
[1] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE
>
> # LinkedIn more popular than Facebook
> linkedin>facebook
[1] FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE
```

- Comparing matrices

```
> views <- matrix(c(linkedin, facebook), nrow = 2, byrow = TRUE)
>
> # When does views equal 13?
> views==13
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]
[1,] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
>
> # When is views less than or equal to 14?
> views<=14
      [,1] [,2] [,3]  [,4] [,5]  [,6] [,7]
[1,] FALSE TRUE TRUE  TRUE TRUE FALSE TRUE
[2,] FALSE TRUE TRUE FALSE TRUE  TRUE TRUE
```

- AND operator **&**
- **&&** only examines the first element of each vector
- OR operator **|**
- **||** only examines the first element of each vector
- NOT operator **!**

```
> # The linkedin and last variable are already defined for you
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
> last <- tail(linkedin, 1)
>
> # Is last under 5 or above 10?
> last<5 | last>10
[1] TRUE
>
> # Is last between 15 (exclusive) and 20 (inclusive)?
> last>15 & last<=20
[1] FALSE
>
> # linkedin exceeds 10 but facebook below 10
```

```
> linkedin > 10 & facebook < 10
[1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
>
> # When were one or both visited at least 12 times?
> linkedin>=12 | facebook>=12
[1]  TRUE FALSE  TRUE  TRUE FALSE  TRUE  TRUE
>
> # When is views between 11 (exclusive) and 14 (inclusive)?
> views>11 & views<=14
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6] [,7]
[1,] FALSE FALSE  TRUE FALSE FALSE FALSE TRUE
[2,] FALSE FALSE FALSE FALSE FALSE  TRUE TRUE
```

- Conditional statements -
    - **If, else, else if** statement structure
        - **if (**condition**) {**
          expression1
          } else if (condition**) {**
          expression2
          } else {
          expression2
          }
    - It's important that the **else** and **else if** keyword comes on the same line as the closing bracket of the **if** part!

```
> # Variables related to your last day of recordings
> medium <- "LinkedIn"
> num_views <- 14
>
> # Examine the if statement for medium
> if (medium == "LinkedIn") {
  print("Showing LinkedIn information")
}
> # Variables related to your last day of recordings
> li <- 15
> fb <- 9
>
> # Code the control-flow construct
> if (li >= 15 & fb >= 15) {
  sms <- 2 * (li + fb)
} else if (li < 10 & fb < 10) {
  sms <- 0.5 * (li + fb)
} else {
  sms <- (li + fb)
```

```
}
>
> # Print the resulting sms to the console
>sms
[1] 24
```

- While loop
    - **while (**condition**) {**
      expr
      }

```
> # Initialize the speed variable
> speed <- 64
>
> # Code the while loop
> while (speed>30) {
  print("Slow down!")
  speed <- speed-7
}
[1] "Slow down!"
[1] "Slow down!"
[1] "Slow down!"
[1] "Slow down!"
[1] "Slow down!"
>
> # Print out the speed variable
> speed
[1] 29
```

```
> # Initialize the speed variable
> speed <- 64
>
> # Extend/adapt the while loop
>while (speed > 30) {
  print(paste("Your speed is",speed))
  if (speed > 48 ) {
  print("Slow down big time!")
  speed <- speed-11
  } else {
  print("Slow down!")
  speed <- speed-6
  }
}
```

- Break statement: when R encounters this, the while loop stops

```
> # Initialize the speed variable
> speed <- 88
>
> while (speed > 30) {
    print(paste("Your speed is", speed))

    # Break the while loop when speed exceeds 80
    if (speed > 80) {
      break
    }

    if (speed > 48) {
      print("Slow down big time!")
      speed <- speed - 11
    } else {
      print("Slow down!")
      speed <- speed - 6
    }
  }
[1] "Your speed is 88"
```

- For loop
    - **Break** statement- stops the loop
    - **Next** statement - skips to the next iteration

```
> # The linkedin vector has already been defined for you
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
>
> # Loop version 1
> for (day in linkedin) {
  print(day)
}
>
> # Loop version 2
> for (i in 1:length(linkedin)) {
  print (linkedin[i])
}
```

- Looping over a matrix requires a nested for loop - to loop over the rows and columns

```
>ttt
     [,1] [,2] [,3]
[1,] "O"  NA   "X"
[2,] NA   "O"  "O"
[3,] "X"  NA   "X"
>
```

```
> # define the double for loop
> for (i in 1:nrow(ttt)) {
    for (j in 1:ncol(ttt)) {
      print(paste("On row i and column j the board contains x", ttt[i,j]))
    }
  }

> # The linkedin vector has already been defined for you
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
>
> # Extend the for loop
> for (li in linkedin) {
  if (li > 10) {
    print("You're popular!")
  } else {
    print("Be more visible!")
  }

  # Add if statement with break
  if (li > 16) {
    print ("This is ridiculous, I'm outta here!")
    break
  }

  # Add if statement with next
  if (li < 5) {
    print ("This is too embarrassing!")
    next
  }

  print(li)
}
```

Functions

- Function documentation: use **help (func)** or **?func**
- Standard deviation function: **sd( )**
- **abs( )** :absolute function
- **args(func)** is a shortcut way of finding what arguments are required in the function

```
> # The linkedin and facebook vectors have already been created for you
> linkedin <- c(16, 9, 13, 5, 2, 17, 14)
> facebook <- c(17, 7, 5, 16, 8, 13, 14)
>
> # Calculate the mean of the sum
> avg_sum <- mean(linkedin + facebook)
```

```
>
> # Calculate the trimmed mean of the sum
> avg_sum_trimmed <- mean(linkedin + facebook, trim=0.2)
>
> # Inspect both new variables
> avg_sum
[1] 22.28571
> avg_sum_trimmed
[1] 22.6
```

- **Mean(x, trim=0, na.rm=FALSE)**
    - **Na.rm** argument deals with missing values

```
> linkedin <- c(16, 9, 13, 5, NA, 17, 14)
> facebook <- c(17, NA, 5, 16, 8, 13, 14)
>
> # Basic average of linkedin
> mean(linkedin)
[1] NA
>
> # Advanced average of linkedin
> mean(linkedin, na.rm=TRUE)
[1] 12.33333
```

- Writing own functions
    - Template:
      my_func <- function(arg1, arg2) {
          body
      }

```
> # Define the interpret function
> interpret <- function(num_views) {
    if (num_views > 15) {
    print ("You're popular!")
    return (num_views)
    } else {
    print ("Try to be more visible!")
    return (0)
    }
  }
>
> # Call the interpret function twice
> interpret(linkedin[1])
> interpret(facebook[2])
> # Define the interpret_all() function
> # views: vector with data to interpret
```

```
> # return_sum: return total number of views on popular days?
> interpret_all <- function(views, return_sum=TRUE) {
  count <- 0

  for (v in views) {
  count <- count + interpret(v)
  }

  if (return_sum) {
  return (count)
  } else {
  return (NULL)
  }
 }
```

- R packages
    - Packages - base, ggvis (for visualization)
    - > search()
        - Shows current packages installed in R
    - > install.packages("ggvis")
        - This command goes to CRAN - Comprehensive R Archive Network
    - > library ("ggvis")
        - This loads the package and now can be used within R
    - > require (func)
        - This shows what package needs to be installed to use the function
- lapply()
    - **lapply(X, Func, …)**
    - Used to apply a function to a vector, list
    - Always returns as a list, unless you apply function **unlist()** on the lapply() function

```
> # The vector pioneers has already been created for you
> pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")
>
> # Split names from birth year
> split_math <- strsplit(pioneers, split = ":")
>
> # Convert to lowercase strings: split_low
> split_low <- lapply(split_math, tolower)
>
> # Take a look at the structure of split_low
> str(split_low)
List of 4
 $ : chr [1:2] "gauss" "1777"
 $ : chr [1:2] "bayes" "1702"
```

```
 $ : chr [1:2] "pascal" "1623"
 $ : chr [1:2] "pearson" "1857"
>
> # Generic select function
> select_el <- function(x, index) {
    x[index]
   }
>
> # Use lapply() twice on split_low: names and years
> names <- lapply(split_low, select_el, index=1)
> years <- lapply(split_low, select_el, index=2)
```

- Anonymous functions - useful for when using a function one time, no need to name function

```
> # Definition of split_low
> pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")
> split <- strsplit(pioneers, split = ":")
> split_low <- lapply(split, tolower)
>
> # Transform: use anonymous function inside lapply
> names <- lapply(split_low, function(x) {x[1]})
>
> # Transform: use anonymous function inside lapply
> years <- lapply(split_low, function(x) {x[2]})
```