R Cheat Sheet

(please reference this regularly)

Assignment vs Equality

- `x = 5` and `x<-5` are the same thing. They **assign** the value 5 to x. From then on, anywhere you type `x` it's equivalent to typing **5**; unless you later assign a different value to **x.**
- `x == 5` evaluates whether **x** is currently equal to the value **5**. If it is, this returns TRUE, otherwise FALSE. In no case does it set x equal to y.
- `x = y` sets **x** equal to whatever value **y** is currently. It does not change the value of **y**. If **y** has not yet been assigned a value, this will return an error. If **x** has already been assigned a value, this will ignore that and overwrite it with **y**'s value.

Vectors and Matrices

- Vectors are created using the **c()** function to combine elements. `c(1,4,7)` creates the vector **[1 4 7]**. If you don't assign it to a variable something it disappears after displaying.
- Access the 3rd element of a vector named vec with `vec[3]`. Suppose you have a Matrix named mat with 3 rows and 3 columns, then `mat[3,]` will return the complete 3rd row, `mat[2,1]` will return the element in the 2nd row and 1st column (columns are up/down, rows are across).
- All operations occur element by element, so multiplying two vectors will multiply each element by the other element and they must be the same length. Similarly with Matrices, they must have the same number of rows and columns to multiply two matrices together, since it's element by element.

If Statements

- The general structure of an if statement is

```
if(condition) {
    code to run if condition is TRUE
} else if(other condition) {
    code to run if condition is FALSE and other condition is TRUE
} else {
    code that runs if all previous conditions are FALSE
}
```

- If there is no else statement, then the if statement will do nothing if the statement is not true. For example, the follow if statment will check whether x is less than 3 and, if so, print "X is smaller than 3". If X is not bigger than 3 it does nothing.

```
if(x<3) {
     print("x is smaller than 3")
}
```

- Note: Conditions must return either TRUE or FALSE if you run them alone

Example For and While Loops

- Below shows how to display the numbers 1,2,3,4,5; one at a time, with both a for loop and a while loop. Note that the for loop doesn't require an initial value for i and increments i automatically each loop. In the while loop you need to do both manually. If you're not sure how many times you need to run the loop, then while loops are typically better.

For Loop

```
for(i in 1:5) {
   print(i)
}
```

While Loop

```
i=1
while(i<=5) {
   print(i)
   i = i+1

}
```

Functions

- The general structure for defining a function is

```
function_name <- function(inputs) {
      ##code that runs
      return(what the function gives back as output)
}
```

- For example the following function multiplies two inputs together

```
mult_xy <- function(x,y) {
      value = x*y
      return(value)
}
```

- You always *call* or use a function by writing the name with parenthesis and a value for each input. For example, running the above and then typing `mult_xy(3,6)` will return 18.
- Variable names assigned outside of functions can be used inside functions, but variable names inside functions cannot be used outside functions. So for example, if you run the above function and then try to call **value**, you will get an error, since it only exists inside the function and then is destroyed. Likewise with **x** and **y**. If you have an **x** outside of the function, and **x** as an input, the **x** outside of the function will be ignored while running the function, but will still exist and be unchanged afterwards.