## Morning checklist

*Greetings* Setup Etherpad: <http://pad.software-carpentry.org/2016-01-30-UTA> \*Workshop website: <https://annawilliford.github.io/2016-01-30-UTA>

## 1. Intro to RStudio

* Why R
* RStudio as interface for R(IDE)
* The 4 Windows of R studio
  + open new file: go to File->New File->R script. We will use it to record our commands
  + focus on console: interactive; read-execute-print loop
  + Navigation commands
    - From console: getwd() Press enter to execute command. Did it work?
      * Paranthesese identify command as a function
      * Built-in vs custom-made functions
      * Function call without arguments
    - From console: setwd() Press enter to execute. Error! Can you tell why?
    - **Challenge 1.1** purpose: setwd() to SCW\_Jan2016; function arguments
* Command flow: from console to editor or the other way around. Try both ways.
  + Look at the history tab in the top right window. Find getwd()andsetwd()` commands and move them to source.
  + Make Sunday\_AM folder in SCW\_Jan2016: Run the command from the editor window (Ctrl+Enter) dir.create("Sunday\_AM")
* Which way do you like better? Beginers -> console to editor; later on -> editor to console
* **Q** why use commands if we can use GUI?
* More about console window:
  + Incomplete commands:
    - Type 5+12; push enter -> 17 is the output on the console
    - Type '5+'; push enter-> + is the output; incomplete command! Push Esc to return to >: ready to accept input
  + Cannot reuse the output unless... **Q** How to make the output reusable?
* Variables and Assignment
  + To save the output of any command for future use, assign it a variable. We are making objects now that are stored in R's memory.
    - myKids<-3; yourKids<-2; ourKids <-myKids+yourKids
    - mySchool<-"UTA"; yourSchool<-"UTDallas"; ourSchool<-paste(mySchool,"and",yourSchool)
  + assigning a new value to the variable with the same name will change the old value
    - myKids<-1
    - myKids<-myKids+2 The right hand side is *fully evaluated* before the assignment occurs.
  + newly created objects are stored in memory, you can see what they are in Environment tab, top right window
  + a word about variable names:no spaces, do not start with numbers. Whatever else you do, be consistent
  + **Challenge 1.2** Purpose: assign values to variables
* Manage the environment
  + ls() list stored objects in the global environment
  + rm(x) remove an object
  + rm(list=ls()) remove all objects from environment
    - notice that functions can be nested!!! Very common in R
      * **Together** print the value of sqrt(4) with print() function
* Help functions for understnding functions:
  + ?(functionName); ?(rm), ?(ls): you will be taken to documentation (Help tab/bottom right window )
  + args(functionName)
* Where do these functions come from? The functionality of R. Package management. There are over 7,000 packages available on CRAN.
  + You can see what packages are installed by typing installed.packages()
  + You can install packages by typing install.packages("packagename"), where packagename is the package name, in quotes.
  + You can update installed packages by typing update.packages()
  + You can remove a package with remove.packages("packagename")
  + You can make a package available for use with library(packagename)

## 2. Data types and Data structures

We have already seen variables that store numbers and variable that store text. These objects are examples of atomic data types. These the smallest building blocks of R. We need to understand data types because functions are written to work on specific data types.

There are 5 frequently used atomic types in R. They cannot be broken down into anything smaller:

### **Data Types**

* numeric
  + integer (e.g, 2L, as.integer(3))
  + double (i.e. decimal) (e.g, -24.57, 2.0, pi)
* character (e.g, "a", "swc", 'This is a cat')
* logical (e.g., TRUE, FALSE)
* complex (i.e. complex numbers) (e.g, 1 + 0i, 1 + 4i)

There are also help functions to identify the data type of objects:

* typeof() # what is its atomic type?
* is.logical() # is it TRUE/FALSE data?
* is.numeric() # is it numeric?
* is.integer() # is it an integer?
* is.complex() # is it complex number data?
* is.character() # is it character data?
* str() # what object is it?

**Together** Suppose I have an object called score that has value of 79:

score<-79  
  
Let's see what data type it is. Looks like an integer to me.  
  
is.integer(score)  
typeof(score)  
typeof(is.integer(score))  
typeof(typeof(score))  
str(typeof(is.integer(score)))

The nested functions are commom, but can be very difficult to read at first. You can always break expressions like this into smaller chuncks to understand the command

**Challenge 2.1** purpose: Understand nested functions while learning how to use help functions to identify various data types

There are functions to convert one data type into another. Not always possible: \* y<-as.integer(3) coersion to integer from double \* y<-as.integer(3.6) coersion to integer from decimal [NOT ROUNDING] **Together** What do you think the output would be? \* y<-as.integer("5.27") can coerce a decimal string \* y<-as.integer("UTA") output: NA: special object = missing value \* y<-as.integer("TRUE") you can guess/check y<-as.integer("FALSE")

### **Data Structures**

Now we are getting to something interesting. So far we made single-element objects. We can now combine them to make more complex objects, or *data structures*

# Download gapminder data set: take a look at the table.

This table is represented by a single object in R, known as a data frame. But it is made up of smaller structures that in turn are made up of the basic single-element data types.

**Together** What would be easiest data structure? Hopefully, vector-like structure will come up

#### Vectors: most common and basic data structure

* + must contain elements of the **same data type**
  + can contain any of the five data types introduced before
  + how to create: use concatinate function c() or vector()
    - v<-c(1,6,77); x<-c(1:10); seq(1,10, by=2)
    - v <- vector("character", length = 10) empty vectors of predefined length
* **Together** Let's make a vector and play with it. Parallels with *bash*
  + how to examine:
    - head(x, n=2), tail(x, n=6), length(x), str(x)
  + how to manipulate:
    - coercion: just like with data types, ex:as.character(v)
    - add an element to the vector: v <- c(v, 57)
    - name: names(v)<-c("a","b","c") What is the best way to name vector with multiple elments?
    - subset [explain subsetting later, with all data structures]
    - vectorization

**Challenge 2.2** Purpose: understand what command does based on the output

#### Matrices: multi-dimensional vectors:

* + must contain elements of the **same data type**
  + create with matrix():
    - m<-matrix(c(1:18)), ncol=6, nrow=3)
  + examine dim(m), str(m), nrow(m), ncol(m), length(m)
  + name: rownames(m), colnames(m)
  + check out ?matrix for more details

#### Factors: vectors that represent categorical data:

* + create with factor()
    - f<-factor(c("Exp","Exp","C","C","Exp")) \*examine:
    - str(f), typeof()
  + reorder: for some analysis order matters
    - f<-factor(f, levels=c("Exp","C")) ex: barplot(table(f)) before and after reordering

#### Lists: generic vector

* + usually contain elements of *different data types*
  + to create, use list() function: l<-list(4,"dogs",TRUE)
  + **Challenge 2.3** Try to make a more interesting list (myOrderInfo) that contains other data structures as list elements) \*Define your elements first, that create a list with them. **Together** Ask them to apply the following functions to the list they created
  + examine: length(myOrderInfo) - unexpected output?
  + str(myOrderInfo)

#### Data Frames

Now we are ready to explore data frames. Go back to you Excel with gapminder dataset. Could you make a informative guess about how this data structure can be represented in R?

**Yes!** It is a list of vectors of equal length. Let's modify our myOrder list to construct a data frame:

menuItems<-c("chicken", "soup", "salad", "tea")  
menuType(factor(c("solid", "liquid", "solid", "liquid"))  
menuCost<-c(3.99, 2.99, 4.69, 2.09)

Combine as a list: myOrder<-list(menuItems, menuType, menuCost) Check with str(); keeps the data types we asigned

Now combine as data.frame. Follows the same pattern as before. Ask students to do that myOrder<-data.frame(menuItems, menuType, menuCost) Check with str(); keeps the data types we asigned???

### General subsetting rules

Now let's talk about how to extract elements from various data structures. If you want to manipulate(change/replace/rename) individual elements or parts of the data structure, we need to know how to accesss them

* Three general ways to subset the data
  + *By position index*: v<-c(1:10) (element's number within the structure: starts with 1 in R)
    - v[2] : square brackets operator = "get me the nth element".
    - v[c(3:6)]; v[c(1,3,c(8:10))]; v[-c(3:5)] ; negative=drop element
    - **Challenge**: apply subsetting by index from lists and data frames we created before (give them the names of the lists that are recorded on file-make sure that they have names)
  + By name index: If the elements of your data structures have names, you can extract the value by name:
    - name our myOrder list: names(myOrder)<-c("item", "type","cost")
    - extract the first element of the list by name: myOrder["item"]
      * what type of data structure is returned? str(), typeof() --> LIST
    - you can also use $ operator to subset by name (for lists and dataframes):
      * Try myOrder$item ; What type is returned? --> VECTOR

### Our gapminder data frame example:

\* read data in in R \*\*Challenge\*\*  
\* combine everything we leaned (extract columns, rows, rownames, subsets...)

## Summary

* R is made up objects and functions [that manipulate objects in various ways]
* Objects can be as simple or as complex as you want them to be (builtin + custom-made). Name some objects/data structures?
* Functions can be as simple or as complex as you want them to be (builtin + custom-made).Name some functions we used?
* Help functions are super valuble to write your own code or navigate through script written by others. Do you remember what they are?

#### **You are now ready to use R objects and functions together in a script**