**What's a data frame?**

You may remember from the chapter about matrices that all the elements that you put in a matrix should be of the same type. Back then, your data set on Star Wars only contained numeric elements.

When doing a market research survey, however, you often have questions such as:

* 'Are you married?' or 'yes/no' questions (logical)
* 'How old are you?' (numeric)
* 'What is your opinion on this product?' or other 'open-ended' questions (character)
* ...

The output, namely the respondents' answers to the questions formulated above, is a data set of different data types. You will often find yourself working with data sets that contain different data types instead of only one.

A data frame has the variables of a data set as columns and the observations as rows. This will be a familiar concept for those coming from different statistical software packages such as SAS or SPSS.

#instruction

Click 'Run’. The data from the built-in example data frame [**mtcars**](http://www.rdocumentation.org/packages/datasets/functions/mtcars) will be printed to the console.

# Quick, have a look at your data set

Wow, that is a lot of cars!

Working with large data sets is not uncommon in data analysis. When you work with (extremely) large data sets and data frames, your first task as a data analyst is to develop a clear understanding of its structure and main elements. Therefore, it is often useful to show only a small part of the entire data set.

So how to do this in R? Well, the function [**head()**](http://www.rdocumentation.org/packages/utils/functions/head) enables you to show the first observations of a data frame. Similarly, the function [**tail()**](http://www.rdocumentation.org/packages/utils/functions/head) prints out the last observations in your data set.

Both [**head()**](http://www.rdocumentation.org/packages/utils/functions/head) and [**tail()**](http://www.rdocumentation.org/packages/utils/functions/head) print a top line called the 'header', which contains the names of the different variables in your data set.

#Instructions

Call [**head()**](http://www.rdocumentation.org/packages/utils/functions/head) on the [**mtcars**](http://www.rdocumentation.org/packages/datasets/functions/mtcars) data set to have a look at the header and the first observations.

**Have a look at the structure**

Another method that is often used to get a rapid overview of your data is the function [**str()**](http://www.rdocumentation.org/packages/utils/functions/str). The function [**str()**](http://www.rdocumentation.org/packages/utils/functions/str) shows you the structure of your data set. For a data frame it tells you:

* The total number of observations (e.g. 32 car types)
* The total number of variables (e.g. 11 car features)
* A full list of the variables names (e.g. mpg, cyl ... )
* The data type of each variable (e.g. num)
* The first observations

Applying the [**str()**](http://www.rdocumentation.org/packages/utils/functions/str) function will often be the first thing that you do when receiving a new data set or data frame. It is a great way to get more insight in your data set before diving into the real analysis.

#Instructions

Investigate the structure of [**mtcars**](http://www.rdocumentation.org/packages/datasets/functions/mtcars). Make sure that you see the same numbers, variables and data types as mentioned above.

**Creating a data frame**

Since using built-in data sets is not even half the fun of creating your own data sets, the rest of this chapter is based on your personally developed data set. Put your jet pack on because it is time for some space exploration!

As a first goal, you want to construct a data frame that describes the main characteristics of eight planets in our solar system. According to your good friend Buzz, the main features of a planet are:

* The type of planet (Terrestrial or Gas Giant).
* The planet's diameter relative to the diameter of the Earth.
* The planet's rotation across the sun relative to that of the Earth.
* If the planet has rings or not (TRUE or FALSE).

After doing some high-quality research on [**Wikipedia**](http://en.wikipedia.org/wiki/Planet), you feel confident enough to create the necessary vectors: name, type, diameter, rotation and rings; these vectors have already been coded up on the right. The first element in each of these vectors correspond to the first observation.

You construct a data frame with the [**data.frame()**](http://www.rdocumentation.org/packages/base/functions/data.frame) function. As arguments, you pass the vectors from before: they will become the different columns of your data frame. Because every column has the same length, the vectors you pass should also have the same length. But don't forget that it is possible (and likely) that they contain different types of data.

#Instruction

Use the function [**data.frame()**](http://www.rdocumentation.org/packages/base/functions/data.frame) to construct a data frame. Pass the vectors name, type, diameter, rotation and rings as arguments to data.frame(), in this order. Call the resulting data frame planets\_df.

# Creating a data frame (2)

The planets\_df data frame should have 8 observations and 5 variables. It has been made available in the workspace, so you can directly use it.

#Instructions

Use [**str()**](http://www.rdocumentation.org/packages/utils/functions/str) to investigate the structure of the new planets\_df variable.

**Selection of data frame elements**

Similar to vectors and matrices, you select elements from a data frame with the help of square brackets [ ]. By using a comma, you can indicate what to select from the rows and the columns respectively. For example:

* my\_df[1,2] selects the value at the first row and second column in my\_df.
* my\_df[1:3,2:4] selects rows 1, 2, 3 and columns 2, 3, 4 in my\_df.

Sometimes you want to select all elements of a row or column. For example, my\_df[1, ] selects all elements of the first row. Let us now apply this technique on planets\_df!

#Instructions

* From planets\_df, select the diameter of Mercury: this is the value at the first row and the third column. Simply print out the result.
* From planets\_df, select all data on Mars (the fourth row). Simply print out the result.

# Selection of data frame elements (2)

Instead of using numerics to select elements of a data frame, you can also use the variable names to select columns of a data frame.

Suppose you want to select the first three elements of the type column. One way to do this is

planets\_df[1:3,2]

A possible disadvantage of this approach is that you have to know (or look up) the column number of type, which gets hard if you have a lot of variables. It is often easier to just make use of the variable name:

planets\_df[1:3,"type"]

#Instruction

Select and print out the first 5 values in the "diameter" column of planets\_df.

# Only planets with rings

You will often want to select an entire column, namely one specific variable from a data frame. If you want to select all elements of the variable diameter, for example, both of these will do the trick:

planets\_df[,3]

planets\_df[,"diameter"]

However, there is a short-cut. If your columns have names, you can use the $ sign:

planets\_df$diameter

#Instructions

* Use the $ sign to select the rings variable from planets\_df. Store the vector that results as rings\_vector.
* Print out rings\_vector to see if you got it right.

# Sorting

Making and creating rankings is one of mankind's favorite affairs. These rankings can be useful (best universities in the world), entertaining (most influential movie stars) or pointless (best 007 look-a-like).

In data analysis you can sort your data according to a certain variable in the data set. In R, this is done with the help of the function [**order()**](http://www.rdocumentation.org/packages/base/functions/order).

[**order()**](http://www.rdocumentation.org/packages/base/functions/order) is a function that gives you the ranked position of each element when it is applied on a variable, such as a vector for example:

> a <- c(100, 10, 1000)

> order(a)

[1] 2 1 3

10, which is the second element in a, is the smallest element, so 2 comes first in the output of order(a). 100, which is the first element in a is the second smallest element, so 1 comes second in the output of order(a).

This means we can use the output of order(a) to reshuffle a:

> a[order(a)]

[1] 10 100 1000

#Instruction

Experiment with the [**order()**](http://www.rdocumentation.org/packages/base/functions/order) function in the console. Click 'Submit Answer' when you are ready to continue.

# Sorting your data frame

Alright, now that you understand the [**order()**](http://www.rdocumentation.org/packages/base/functions/order) function, let us do something useful with it. You would like to rearrange your data frame such that it starts with the smallest planet and ends with the largest one. A sort on the diameter column.

#Instruction

* Call order() on planets\_df$diameter (the diameter column of planets\_df). Store the result as positions.
* Now reshuffle planets\_df with the positions vector as row indexes inside square brackets. Keep all columns. Simply print out the result.

**Lists, why would you need them?**

Congratulations! At this point in the course you are already familiar with:

* **Vectors** (one dimensional array): can hold numeric, character or logical values. The elements in a vector all have the same data type.
* **Matrices** (two dimensional array): can hold numeric, character or logical values. The elements in a matrix all have the same data type.
* **Data frames** (two-dimensional objects): can hold numeric, character or logical values. Within a column all elements have the same data type, but different columns can be of different data type.

Pretty sweet for an R newbie, right? ;-)

# Lists, why would you need them? (2)

A **list** in R is similar to your to-do list at work or school: the different items on that list most likely differ in length, characteristic, and type of activity that has to be done.

A list in R allows you to gather a variety of objects under one name (that is, the name of the list) in an ordered way. These objects can be matrices, vectors, data frames, even other lists, etc. It is not even required that these objects are related to each other in any way.

You could say that a list is some kind super data type: you can store practically any piece of information in it!

# Creating a list

Let us create our first list! To construct a list you use the function [**list()**](http://www.rdocumentation.org/packages/base/functions/list):

my\_list <- list(comp1, comp2 ...)

The arguments to the list function are the list components. Remember, these components can be matrices, vectors, other lists, ...

#Instruction

Construct a list, named my\_list, that contains the variables my\_vector, my\_matrix and my\_df as list components.

# Creating a named list

Well done, you're on a roll!

Just like on your to-do list, you want to avoid not knowing or remembering what the components of your list stand for. That is why you should give names to them:

my\_list <- list(name1 = your\_comp1,

name2 = your\_comp2)

This creates a list with components that are named name1, name2, and so on. If you want to name your lists after you've created them, you can use the names() function as you did with vectors. The following commands are fully equivalent to the assignment above:

my\_list <- list(your\_comp1, your\_comp2)

names(my\_list) <- c("name1", "name2")

#Instructions

* Change the code of the previous exercise (see editor) by adding names to the components. Use for my\_vector the name vec, for my\_matrix the name mat and for my\_df the name df.
* Print out my\_list so you can inspect the output.

# Creating a named list

Well done, you're on a roll!

Just like on your to-do list, you want to avoid not knowing or remembering what the components of your list stand for. That is why you should give names to them:

my\_list <- list(name1 = your\_comp1,

name2 = your\_comp2)

This creates a list with components that are named name1, name2, and so on. If you want to name your lists after you've created them, you can use the names() function as you did with vectors. The following commands are fully equivalent to the assignment above:

my\_list <- list(your\_comp1, your\_comp2)

names(my\_list) <- c("name1", "name2")

#Instruction

* Change the code of the previous exercise (see editor) by adding names to the components. Use for my\_vector the name vec, for my\_matrix the name mat and for my\_df the name df.
* Print out my\_list so you can inspect the output.