

## ECO 274 LAB: R Syntax and Data Structure

### Learning Objectives

- To understand R syntax
- Create objects/variables in R.
- Explore data types used in R.
- Construct data frame and data structures to store data.

### The R syntax

- the comments `#` and how they are used to document function and its content
- variables and functions
- the assignment operator `<-`
- the `=` for arguments in functions

### Assignment operator

`##` The `<-` is called the assignment operator. This operator assigns values to variables. The command below is translated into a sentence as: The result variable (x) gets the value of 3. We need to assign *values* to *variables* using the assignment operator, `<-`. For example, we can use the assignment operator to assign the value of 8 to an object called *number*. Also, an object can get multiple values like the object x1 below. C stand for concatenation:

```
x <- 3
number <- 8
x1 <- c(3,4,6)
```

The assignment operator (`<-`) assigns **values on the right** to **variables on the left**. In *RStudio*, typing `Alt + -` (push *Alt* at the same time as the `-` key, on Mac type `option + -`) will write `<-` in a single keystroke.

### Variables

A variable is a symbolic name for (or reference to) information. Variables in computer programming are analogous to “buckets”, where information can be maintained and referenced. On the outside of the bucket is a name. When referring to the bucket, we use the name of the bucket, not the data stored in the bucket.

In the example above, we created a variable, or a ‘bucket’ called `x`. Inside we put a value, 3.

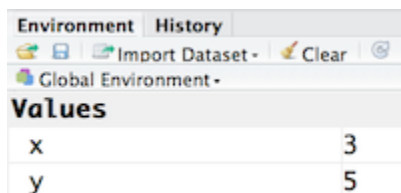
Let’s create another variable called `y` and give it a value of 5.

```
y <- 5
```

When assigning a value to a variable, R does not print anything to the console. You can force to print the value by using parentheses or by typing the variable name.

```
y
```

You can also view information on the variable by looking in your `Environment` window in the upper right-hand corner of the RStudio interface.



The screenshot shows the RStudio Environment pane. At the top, there are tabs for 'Environment' and 'History'. Below the tabs are icons for 'Import Dataset', 'Clear', and a refresh icon. Underneath, it says 'Global Environment'. A section titled 'Values' contains a table with two rows: 'x' with value '3' and 'y' with value '5'.

Values	
x	3
y	5

Now we can reference these buckets by name to perform mathematical operations on the values contained within. What do you get in the console for the following operation:

```
x + y
```

Try assigning the results of this operation to another variable called `number`.

```
number <- x + y
```

## Tips on variable names

Variables can be given almost any name, such as `x`, `current_temperature`, or `subject_id`. However, there are some rules / suggestions you should keep in mind:

- Make your names explicit and not too long.
- Avoid names starting with a number (2x is not valid but x2 is)
- Avoid names of fundamental functions in R (e.g., `if`, `else`, `for`, see [here](#) for a complete list). In general, even if it's allowed, it's best to not use other function names (e.g., `c`, `T`, `mean`, `data`) as variable names. When in doubt check the help to see if the name is already in use.
- Avoid dots (.) within a variable name as in `my.dataset`. There are many functions in R with dots in their names for historical reasons, but because dots have a special meaning in R (for methods) and other programming languages, it's best to avoid them.
- Use nouns for object names and verbs for function names
- Keep in mind that **R is case sensitive** (e.g., `genome_length` is different from `Genome_length`)
- Be consistent with the styling of your code (where you put spaces, how you name variable, etc.). In R, two popular style guides are [Hadley Wickham's style guide](#) and [Google's](#).

## Data Types

Variables can contain values of specific types within R.

The six **data types** that R uses include:

- "numeric" for any numerical value
- "character" or "string" for text values, denoted by using quotes (") around value
- "integer" for integer numbers (e.g., 2L, the L indicates to R that it's an integer)
- "logical" for TRUE and FALSE (the Boolean data type)
- "complex" to represent complex numbers with real and imaginary parts (e.g., 1+4i)
- "raw" that is not part of this course

The table below provides examples of each of the commonly used data types:

Data Type	Examples
Numeric:	1, 1.5, 20, pi
Character:	"Top Gun", "5", "TRUE"
Integer:	2L, 500L, -17L
Logical:	TRUE, FALSE, T, F

## Data Structures

Variables can store more than just a single value, they can store a multitude of different data structures. These include, but are not limited to, vectors (c), factors (factor), matrices (matrix), data frames (data.frame) and lists (list).

### Vectors

A vector is the most common and basic data structure in R, and is pretty much the workhorse of R. It's basically just a collection of values, mainly either numbers,

1	50	9	42
---	----	---	----

or characters,

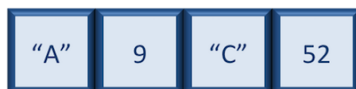


or logical values,



**Note that all values in a vector must be of the same data type.** If you try to create a vector with more than a single data type, R will try to coerce it into a single data type.

For example, if you were to try to create the following vector:



R will coerce it into:



The meaning for a vector is that your bucket now has different compartments; these compartments in a vector are called *elements*.

Let's create a vector of cgpa lengths and assign it to a variable called `cgpa`.

Each element of this vector contains a single numeric value, and 5 values will be combined together into a vector using `c()` (the combine function). All of the values are put within the parentheses and separated with a comma.

```
cgpa <- c(4.0, 3.75, 3.5, 3.25, 3)
cgpa
y <- c(3,4,5)
sum(y)
mean(y)
var(y)
sd(y)
exp(x)
```

*Note your environment shows the `cgpa` variable is numeric and tells you the `cgpa` vector starts at element 1 and ends at element 5 (i.e. your vector contains 5 values).*

A vector can also contain characters. Create another vector called `semester` with three elements, where each element corresponds with the genome sizes vector (in Mb).

```
semester <- c("one", "two", "three", "four", "special" )
semester
```

## Logical Comparison

```
3 < 4
3 > 4
#contrast with 3 = 4; see section about variables below
3 == 4
#!= means "not equal to"
3 != 4
4 >= 5
4 <= 5
2 + 2 == 5
10 - 6 == 4
```

## Factors

A **factor** is a special type of vector that is used to **store categorical data**. Each unique category is referred to as a **factor level** (i.e. category = level).



Let's create a factor vector and explore a bit more. We'll start by creating a character vector describing five different levels of utility:

```
utility <- c("very satisfied", "not satisfied", "satisfied", "satisfied", "not satisfied", "unsatisfied", "unsatisfied", "very unsatisfied" )
```

Now we can convert this character vector into a *factor* using the `factor()` function:

```
utility <- factor(utility)
```

So, what exactly happened when we applied the `factor()` function?

## Matrix

A **matrix** in R is a collection of vectors of **same length and identical datatype**. Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.

90	5	137	9
87	40	2	52
4	102	32	41

Matrices are used commonly as part of the mathematical machinery of statistics. They are usually of numeric datatype and used in computational algorithms to serve as a checkpoint.

```
# Generating matrix, A from one vector with all values
v <- c(2,-4,-1,5,7,0)
A <- matrix(v,nrow=2)

# Generating matrix, A from two vectors corresponding to rows:
row1 <- c(2,-1,7); row2 <- c(-4,5,0)
( A <- rbind(row1, row2) )

# Generating matrix, A from three vectors corresponding to columns:
col1 <- c(1,6); col2 <- c(2,3); col3 <- c(7,2)
( US40Chart <- cbind(col1, col2, col3) )

# Giving names to rows and columns:
colnames(US40Chart) <- c("Drake", "Maroon5", "Dua Lipa")
rownames(US40Chart) <- c("weekTop", "totPeak")
US40Chart
```

## Data Frame

A `data.frame` is the *de facto* data structure for most tabular data and what we use for statistics and plotting. A `data.frame` is similar to a matrix in that it's a collection of vectors of the **same length** and each vector represents a column. However, in a dataframe **each vector can be of a different data type** (e.g., characters, integers, factors).

Year	sales	revenue	economy
2010	24	120	"trough"
2012	45	650	"expansion"
2014	67	980	"slowdown"
2016	78	890	"expansion"
2018	89	990	"peak"

A data frame is the most common way of storing data in R, and if used systematically makes data analysis easier.

We can create a dataframe by bringing **vectors** together to **form the columns**. We do this using the `data.frame()` function, and giving the function the different vectors we would like to bind together. *This function will only work for vectors of the same length.*

```
df <- data.frame(semester, cgpa)
```

Beware of `data.frame()`'s default behavior which turns **character vectors into factors**. Print your data frame to the console:

```
df
```

Upon inspection of our dataframe, we see that although the semester vector was a character vector, it automatically got converted into a factor inside the data frame (the removal of quotation marks). We will show you how to change the default behavior of a function in the next lesson. *Note that you can view your data.frame object by clicking on its name in the [Environment](#) window.*

# Define year vector:

```
year <- c(2008,2009,2010,2011,2012,2013)
```

# Define a matrix of product values:

```
product1<-c(0,3,6,9,7,8)
```

```
product2<-c(1,2,3,5,9,6)
```

```
product3<-c(2,4,4,2,3,2)
```

# Create a sales matrix

```
sales_mat <- cbind(product1,product2,product3)
```

## Give row names

```
rownames(sales_mat) <- year
```

# The matrix looks like this:

```
sales_mat
```

# Create a data frame and display it:

```
sales <- as.data.frame(sales_mat)
```

```
sales
```

```
# Accessing a single variable:
```

```
sales$product2
```

```
# Generating a new variable in the data frame:
```

```
sales$totalvalue <- sales$product1 + sales$product2 + sales$product3
```

```
# Subset: all years in which sales of product 3 were >=3
```

```
subset(sales, product3>=3)
```

## Lists

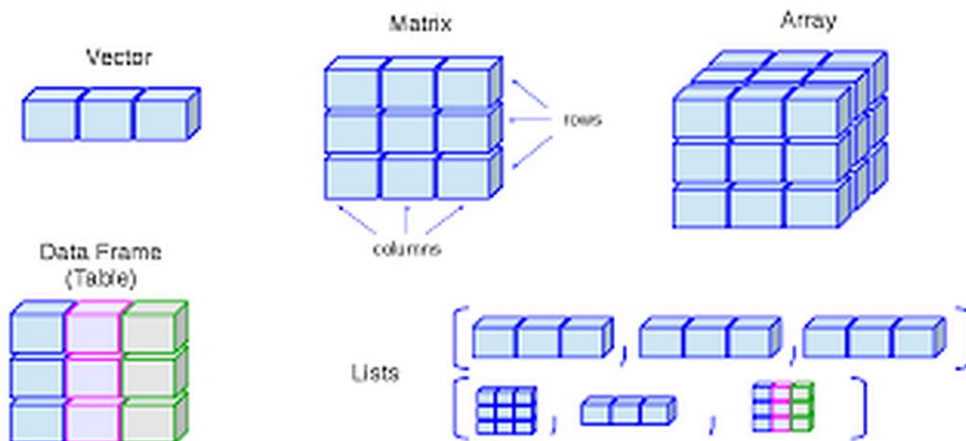
Lists are the R objects which contain elements of different types like – numbers, strings, vectors and another list inside it. A list can also contain a matrix or a function as its elements. List is created using `list()` function.

```
# Create a list containing strings, numbers, vectors and a logical
```

```
# values.
```

```
list_data <- list("Red", "Green", c(21,32,11), TRUE, 51.23, 119.1)
```

```
print(list_data)
```





If you have variables of different data structures you wish to combine, you can put all of those into one list object by using the `list()` function and placing all the items you wish to combine within parentheses:

```
list <- list(semester, df, number, utility)
```

Print out the list to screen to take a look at the components:

```
list
```

There are four components corresponding to the four different variables we passed in, and what you see is that structure of each is retained. Each component of a list is referenced based on the number position.

Reference and acknowledgement:

1. The materials used in this lesson are adapted from work that is Copyright © Data Carpentry (<http://datacarpentry.org/>), Harvard Chan Bioinformatics Core (HBC) under the open access terms of the Creative Commons Attribution license (CC BY 4.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.
2. The Book of R: A First Course in Programming and Statistics by Tilman M. Davies