Learning Objectives

- Become familiar with R syntax
- Understand variables and the assignment operator in R
- Understand the various data types and data structures in R

The R syntax

Now that we know how to talk with R via the script editor or the console, we want to use R for something more than adding numbers. To do this, we need to know more about the R syntax.

Below is an example script highlighting the many different "parts of speech" for 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

NOTE: indentation and consistency in spacing is used to improve clarity and legibility

Example script

```
# Load libraries
library(Biobase)
library(limma)
library(ggplot2)

# Setup directory variables
baseDir <- getwd()
dataDir <- file.path(baseDir, "data")
metaDir <- file.path(baseDir, "meta")
resultsDir <- file.path(baseDir, "results")

# Load data
meta <- read.delim(file.path(metaDir, '2015-1018_sample_key.csv'),
header=T, sep="\t", row.names=1)
```

Assignment operator

To do useful and interesting things in R, we need to assign *values* to *variables* using the assignment operator, <-. For example, we can use the assignment operator to assign the value of 3 to x by executing:

```
x <- 3
```

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) 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`.

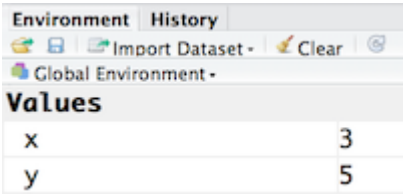
Let's create another variable called `y` and give it a value of 5.

```
y <- 5
```

When assigning a value to an 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.



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
```

Exercise

1. Try changing the value of the variable `x` to 5. What happens to `number`?
2. Now try changing the value of variable `y` to contain the value 10. What do you need to do, to update the variable `number`?

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:

- Avoid names starting with a number (`2x` is not valid but `x2` is)
- Keep in mind that **R is case sensitive** (e.g., `genome_length` is different from `Genome_length`)
- Make your names explicit and not too long.
- 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

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"` 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`) and that's all we're going to say about them
- `"raw"` that we won't discuss further

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

Data Type	Examples
Numeric:	1, 1.5, 20, pi
Character:	“anytext”, “5”, “TRUE”
Integer:	2L, 500L, -17L
Logical:	TRUE, FALSE, T, F

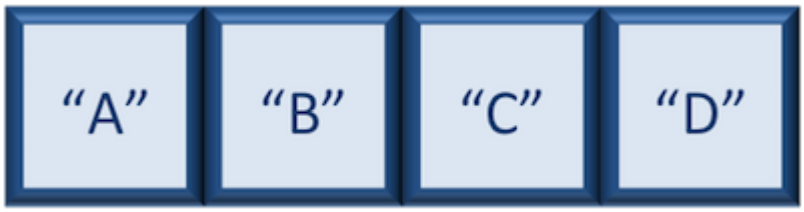
Data Structures

We know that variables are like buckets, and so far we have seen that bucket filled with a single value. Even when `number` was created, the result of the mathematical operation was a single value. **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,

or characters,

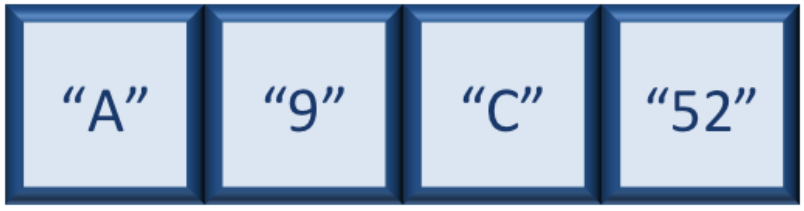


or logical values,

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

Each **element** contains a single value, and there is no limit to how many elements you can have. A vector is assigned to a single variable, because regardless of how many elements it contains, in the end it is still a single

entity (bucket).

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

Each element of this vector contains a single numeric value, and three 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.

```
glengths <- c(4.6, 3000, 50000)
glengths
```

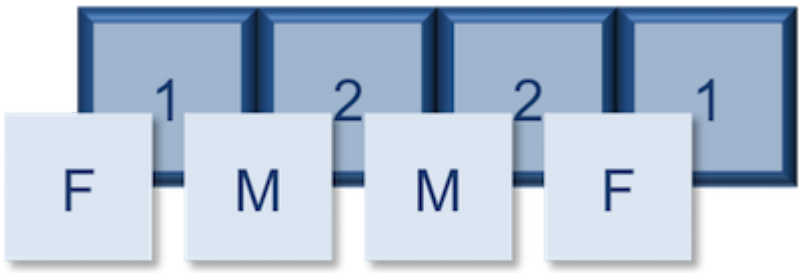
Note your environment shows the `glengths` variable is numeric and tells you the `glengths` vector starts at element 1 and ends at element 3 (i.e. your vector contains 3 values).

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

```
species <- c("ecoli", "human", "corn")
species
```

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). Factors are built on top of integer vectors such that each **factor level** is assigned an **integer value**, creating value-label pairs.



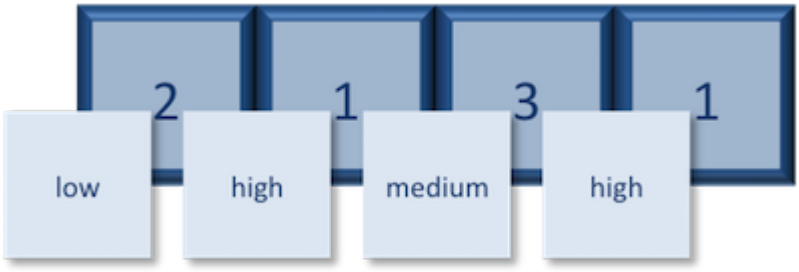
Let's create a factor vector and explore a bit more. We'll start by creating a character vector describing three different levels of expression:

```
expression <- c("low", "high", "medium", "high", "low", "medium", "high")
```

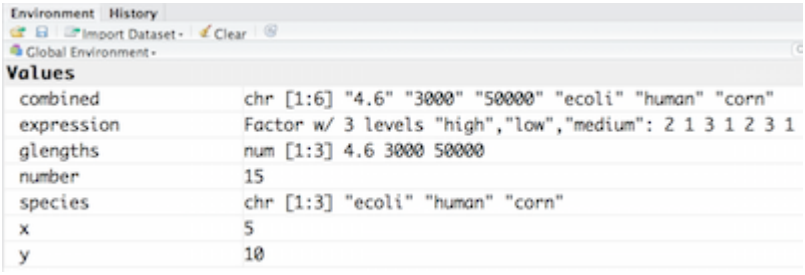
Now we can convert this character vector into a *factor* using the `factor()` function:

```
expression <- factor(expression)
```

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

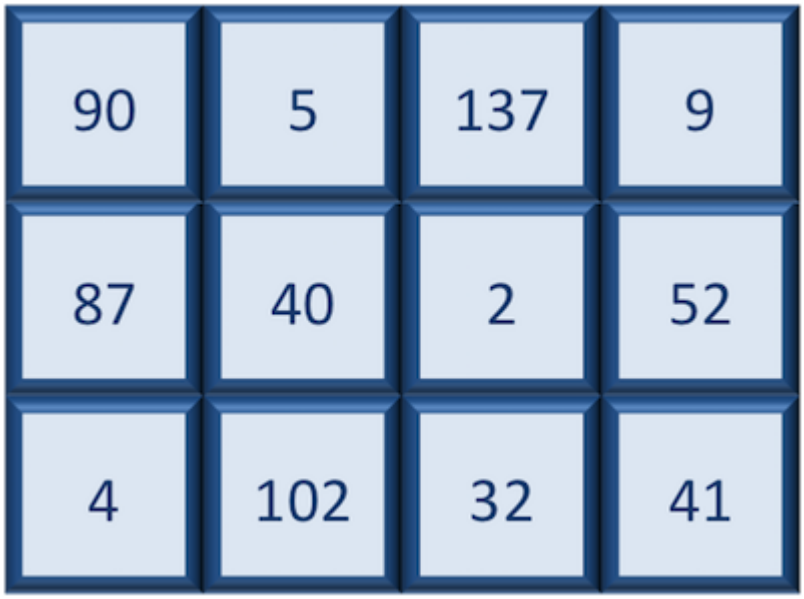


The expression vector is categorical, in that all the values in the vector belong to a set of categories; in this case, the categories are `low`, `medium`, and `high`. By turning the expression vector into a factor, the **categories are assigned integers alphabetically**, with high=1, low=2, medium=3. This in effect assigns the different factor levels. You can view the newly created factor variable and the levels in the **Environment** window.



Matrix

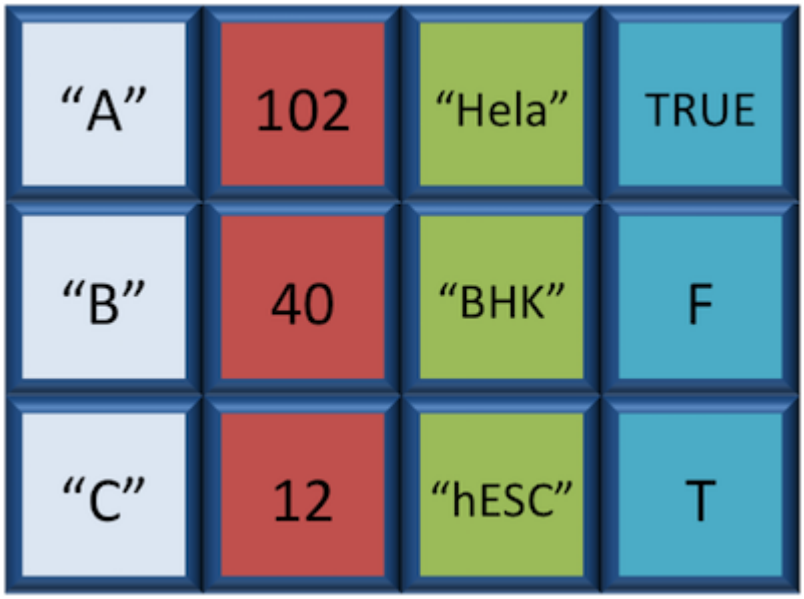
A `matrix` in R is a collection of vectors of **same length and identical datatype**. Vectors can be combined as columns in the matrix or by row, to create a 2-dimensional structure.



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. For example, if input data is not of identical data type (numeric, character, etc.), the `matrix()` function will throw an error and stop any downstream code execution.

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 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).



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

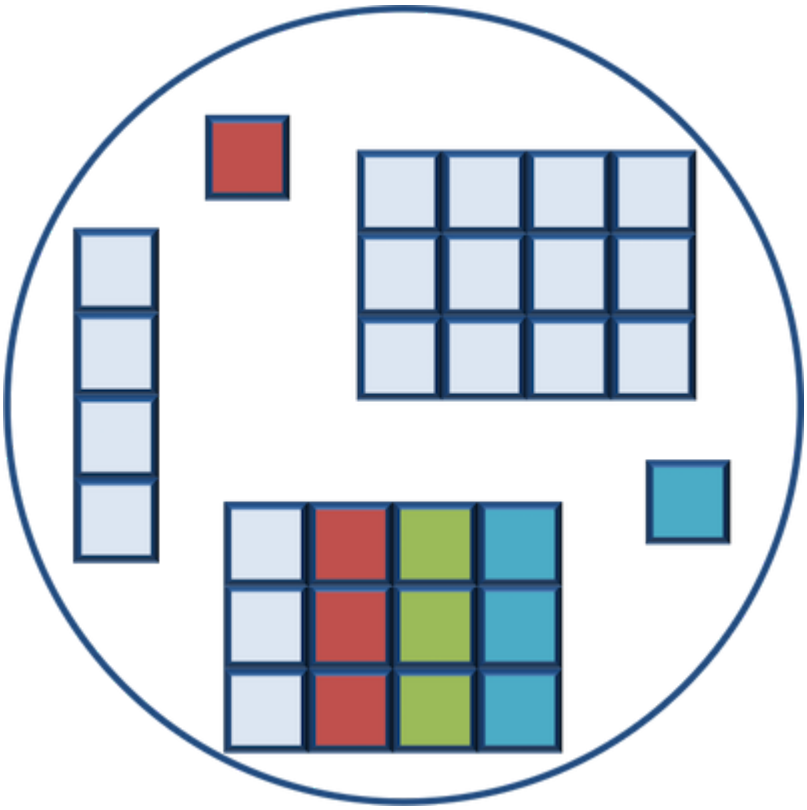
We can create a dataframe by bringing **vectors** together to **form the columns**. We do this using the `data.frame()` function, and giving the function the different vectors we would like to bind together. *This function will only work for vectors of the same length.*

```
df <- data.frame(species, glengths)
```

Note that you can view your data.frame object by clicking on its name in the `Environment` window.

Lists

Lists are a data structure in R that can be perhaps a bit daunting at first, but soon become amazingly useful. A list is a data structure that can hold any number of any types of other data structures.



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:

```
list1 <- list(species, df, number)
```

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

```
list1

[[1]]
[1] "ecoli" "human" "corn"

[[2]]
  species glengths
1   ecoli      4.6
2   human   3000.0
3    corn  50000.0

[[3]]
[1] 5
```

There are three components corresponding to the three different variables we passed in, and what you see is that structure of each is retained.
