Unit 1 Module 1: Getting Started

Lecture R Notebook

Let's start by clearing the global computing environment:

```
rm( list = ls() )
```

Module Preview

Hello! And welcome to Module 1: Getting Started.

In this module, we'll get started with some basic concepts of R programming.

- In Section 1, we'll learn about atomic values, which are the most basic objects in R.
- In Section 2, we'll explore the concept of functions.
- In Section 3, we'll define vectors, which are the most important compound data structure in R.

Once you've completed this module, you'll be able to:

- Explain what an atomic value is
- Define the three primary atomic data types that we will use in this course
- Explain what a function is
- Explain what input arguments and return values are
- Use the built-in R function `exp()` to calculate values of the exponential function.
- Determine the class of an arbitrary R object by using the `class()` function.
- Define the concept of a vector
- Construct a vector of arbitrary values by using the `c()` function

In this module we'll see three built-in R functions:

- The exponential function `exp()`.
- The class function `class()`.
- The vector construction function `c()`.

All right, let's start out by exploring atomic values.

Section 1: Atomic Values

Main Idea *In R, the most basic objects are called* atomic values*.

In this section, we'll learn about atomic values, which are the most basic objects in R.

An *atomic value* is a value that has no component parts.

For our course, we'll focus on 3 different types of atomic values:

- Numeric values, such as 5, -14.7, and $\pi/2$.
- Logical values, such as `TRUE` and `FALSE`.
- Character string values, such as the strings "Hi Mom!", 'xyzxyz', and "7".

Note that character strings **must** be enclosed in quotes, but they can be either single or double quotes.

Each different kind of atomic data is called a "class".

For example:

- The class of the object 5 is "numeric".
- The class of the object `TRUE` is "logical".
- The class of the object "Hi Mom!" is "character".

Sometimes however I'll also refer the class of an object as a "type", which means the same thing.

In fact, for this and the next three lectures, we'll mainly focus on just numeric values, and only in the fifth lecture will we explore logical values.

So those are the atomic values of R.

Now let's move on to functions.

Section 2: Functions

Main Idea: *A function takes a set of values as input arguments, and returns an output value.

In this section, we'll explore the concept of a function.

In general, a *function* is something that takes some input values and either produces an output value or performs some action.

Much of our course is devoted to learning how to use the rich set of functions that are built-in to R, as well how to create your own user-defined functions.

The inputs are usually called the *arguments* of the function, but just to be clear I will always refer to these as the "input arguments".

The output is called the *return value*.

For instance, the *exponential* function takes a number as an input, and returns the value of the constant e raised to the power specified by the input value.

$$f(x) \;=\; e^x$$

Thus, if we use this function with an input argument of 2, the return value is approximately 7.389056:

$$f(2) \;=\; e^2 \;\approx\; 7.389056$$

In R, the built-in function `exp()` computes the exponential function $f(x) = e^x$.

To use an R function to calculate a value, we have to follow an exact procedure:

- First, we write the name of the function.
- Then there is an opening parenthesis.
- Then there is the actual numeric input argument.
- Finally there is a closing parenthesis.

We can calculate the value of the exponential function for the input value 2 using the built-in R function `exp()`.

Remember, there are two ways to run an individual code chunk:

- First, you can click on the little arrow in the upper right-hand corner of the code chunk.
- Second, you can place the cursor inside the code chunk and then use the keystroke Ctrl+Shift+Enter on Windows and Cmd+Shift+Enter on Macs.

```
exp( 2 )
```

```
## [1] 7.389056
```

When we actually use a function so that it executes some action, this is known as "calling" the function.

Note that when I refer to a function, I'll typically include opening and closing parentheses to remind you that this is a function.

So, the R function that calculates the exponential function is referred to as `exp()`.

There is a variation on this method.

Often, the input arguments to a function have *names*, and we can use these to explicitly indicate the association of values with input arguments.

The input argument for the `exp()` function has the name `x`.

To explicitly indicate that the named input argument `x` is associated with the value 2, we write:

```
exp(x = 2)
```

```
## [1] 7.389056
```

Mathematical functions like the `exp()` function typically take numeric values as input arguments and return a numeric value, and that's why we use the function, to calculate the return value for us.

R functions are more general, and they can take any sort of object as an input argument, not just numeric values, and they can return any sort of object.

An example of this is the `class()` function, which takes a single input argument of any type, and returns the class of that object.

For instance, the class of the object 5 is "numeric":

```
class( 5 )
```

```
## [1] "numeric"
```

The class of the object `TRUE` is "logical":

```
class( TRUE )
```

```
## [1] "logical"
```

The class of the object "Hi Mom!" is "character":

```
class( "Hi Mom!" )
```

```
## [1] "character"
```

Some functions don't produce a return value.

Instead, sometimes we'll just want the function to perform an action and we don't really care about the return value.

For instance, graphics functions typically operate this way – they take the input arguments and use them to draw an object, but they don't necessarily produce a return value.

So those are some basic concepts about functions in R.

Now let's move on to vectors.

Section 3: Vectors

> *Main Idea:** *A vector is an ordered sequence of values, all of which must have the same atomic data type.*

In this section, we'll define vectors, which are the most important compound data structure in R.

A *vector* is an ordered sequence of values, all of which must have the same atomic data type.

When I'm talking about a vector as an abstract mathematical object I'll use angle brackets, like this:

$$\langle 1, 2, 3, 4, 5 \rangle$$

Don't confuse these angle brackets with the familiar "less than" or "greater than" symbols from elementary arithmetic.

Instead, I'm just using these to indicate the start and end of the vector.

This vector consists of the numeric values 1, 2, 3, 4, and 5.

The values in the vector are called the *elements* of the vector, so this vector has 5 elements.

Here's a vector consisting of the logical values `TRUE` and `FALSE`:

$$\langle \texttt{ TRUE}, \texttt{TRUE}, \texttt{FALSE}, \texttt{TRUE } \rangle$$

This vector has 4 elements, all of which are logical values.

Notice that the same value `TRUE` can occur more than once in a vector.

Here's a vector where all the elements are character strings:

$$\langle \text{ "Hi Mom", "xyzxyz", "7" } \rangle$$

This vector has 3 elements, each of which is a character string value.

So we can have vectors that consist entirely of numeric values, or entirely of logical values, or entirely of character string values.

But we can't have a vector that mixes these types up; for instance, a single vector can't contain both numeric and logical data.

There are many different ways to create vectors in R, and we'll explore these techniques in greater depth in Week 3, but for now we can create a vector by using the `c()` function.

The `c()` function takes an unlimited number of input arguments, separated by commas, and returns a vector consisting of these input values:

```
c(1, 2, 3, 4, 5)
```

```
## [1] 1 2 3 4 5
```

The input arguments for the `c()` function don't have names, so we have to rely exclusively on the sequence of values.

Notice how R prints this vector out; in particular, it does NOT use angle brackets as delimiters for the vector.

In this example, the values are regularly spaced, but we didn't have to use such a nice sequence.

The `c()` function can take any set of inputs and create a vector, as long as all the inputs are of the same type:

```
c( 5.82, 6.23, -9.84, 12, 0, -4.327)
```

```
## [1]  5.820  6.230 -9.840 12.000  0.000 -4.327
```

We can also use the `c()` function to create logical vectors:

```
c( TRUE, TRUE, FALSE, TRUE )
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

We can also use the `c()` function to create character string vectors:

```
c( "Hi Mom", "xyzxyz", "7" )
```

```
## [1] "Hi Mom" "xyzxyz" "7"
```

We'll learn much more about vectors in Week 3, and indeed the rest of the course, but for now the `c()` function will suffice.

So those are some basic ideas about vectors.

Now let's review the contents of this module.

Module Review

In this module, we started out with some basic concepts of R programming.

- In Section 1, we learned about atomic values, which are the most basic objects in R.
- In Section 2, we explored the concept of functions.
- In Section 3, we defined vectors, which are the most important compound data structure in R.

Now that you've completed this module, you should be able to:

- Explain what an atomic value is
- Define the three primary atomic data types that we will use in this course
- Explain what a function is
- Explain what input arguments and return values are
- Use the built-in R function `exp()` to calculate values of the exponential function.
- Determine the class of an arbitrary R object by using the `class()` function.
- Define the concept of a vector
- Construct a vector of arbitrary values by using the `c()` function

In this module we saw three built-in R functions:

- The exponential function `exp()`.
- The class function `class()`.
- The vector construction function `c()`.

OK! That's it for Module 1: Getting Started.

Now let's move on to Module 2: Graphical Plots.