# Working in the R Environment

*This lab is about getting comfortable with R and being able to use the R environment productively. Just work through the following pages and be sure you understand exactly what is going on.*

## Variables and Their Values

Variables are created in R by using the assignment operator "=".[1] For example, the statement

```
> x = 2
```

Creates a variable called `x`, whose value is 10.

Variables give a shorthand way of storing values so that they can be used in later computations. For example, we can use the variable `x` defined above as follows.

```
> y = sqrt(x)
```

To see what value a variable might have, all you have to do is to type its name.

```
> x
[1] 2

> y
[1] 1.414214
```

By default, R prints 7 significant digits of numerical values, but you can ask for more by explicitly calling the `print` function.

```
> print(y)
[1] 1.414214

> print(y, digits = 15)
[1] 1.4142135623731
```

There is no point in displaying more than 15 digits because this is the limit of machine accuracy.[2]

[1] You may have been told that the R assignment operator is "<-" but you were misinformed. The big kids use "=".

[2] R computes with "double precision" arithmetic. This is probably overkill. Navigation with 7 digits of accuracy is more than enough to fly you smack into the middle of the nearest star.

Variables in R are untyped. You can use the same name for any kind of value, without any problems.

```
> x = 10
> x = "ten"
> x = list(1,1,1,1,1,1,1,1,1,1)
```

## Listing and Removing Variables

After working with R for a while, you will probably have created lots of variables. To see what they are, you can type the command `objects()`. This produces a character vector containing the names of the variables you have created.

```
> objects()
[1] "x" "y"
```

Its important to notice that `objects()` does not print the variable names; it returns the names in a vector. This can be treated like any other R value.

```
> o = objects()
> o[1]
[1] "x"
> o[2]
[1] "y"
```

When objects are not needed any more, you can remove them with the `rm` function. For example, we can remove the `x` and `y` variables with the following expression.

```
> rm(x, y)
```

We can check that they are gone by seeing what variables are left.

```
> objects()
[1] "o"
```

There is a shorthand way of carrying out a complete cleanup by combining the effects of `objects` and `rm`.

```
> rm(list = objects())
> objects()
character(0)
```

(The `character(0)` produced here is R's way of showing a zero-length character vector. I.e., there are no variables left.)

## The Help System

There are a variety of ways of getting information about the details of how R works. The simplest way is to use the help operator. To get help about a particular R function, just put a `?` in front of the function name.

```
> ?round
> ?sin
> ?"+"
```

(Note that operators like + must be quoted, because they are part of the R syntax and are expected to be accompanied by at least one argument.)

Doubling the ? makes it possible to do topic searches. For example, the expression

```
> ??regression
```

can be used to locate the names of all functions and data objects related to the topic "regression."

## Writing R Code

Typing R code at the ">" prompt is useful carrying out trivial tasks with R. It is not a good way to do anything more complex (such as the tasks to be carried out in STATS 380). The best way to work with R is to type code into a separate *code editor* and to transfer it to R for execution.

There a number of possible editors you can use.[3] On Windows, R's built-in script editor can be used, but it has rather limited capabilities. A better alternative is to use the facilities of the Rstudio development environment (this is available in the University labs and can be obtained free of charge and installed on your personal machines).

When Rstudio runs it will reopen any files that you might have aleady be working on and you can continue working on them. The first time you use Rstudio, it starts with no open files. Rather than just typing R code in the Console window, you should open a file to hold your work and type your code into that file before having R execute it. That makes it easy to correct mistakes and rerun code, rather than having to reenter everything again.

To open a new file press the green "+" button near the top-left corner of the Rstudio window and select "R Script" from the menu that pops down. That produces a blank (untitled) file that you can enter R code into. Be sure to save your code with a name suggestive of what it does.

There are several ways to run your code.

- Pressing the "Source" button reads everything everything in the active file into R.

It is also possible to execute just some of the code in the file.

- Placing the cursor at a particular line and pressing "Run" will cause that line to be sent to R for execution. The cursor is then advanced to the next line so that that can be sent by pressing "Run" and so on.
- Selecting a region with the mouse and pressing "Run" will cause the entire region to be sent to R for execution.

When you are done working with R, make sure that you save any files that you wish to keep.

**NOTE**: The answers to be submitted for assignments are expected to be prepared in this way.

[3] Of course real programmers use the Emacs editor from the free software foundation.

## Writing "Pretty" R Code

The main goal of writing any R code it is to make it as easy as possible for a *human* to read and understand it. Often, the human that reads your code will be *you* and you will be doing yourself a kindness by making the task as simple as possible. You will also also not receive much sympathy from lecturers and markers if you show them ugly, difficult-to-understand code.

There are a number of simple things you can do to make your code more readable.

- Use spaces around operators (like `+`, `-`, `* /`, `=` etc. (but not `^`.).
- Follow every comma with a space.
- Separate groups of related code lines with a blank line.
- Use lots of comments to describe what the code does. (Usually it's best to comment related blocks of code rather than every line.)
- Indent the code to reflect its structure. (You'll see lots of examples of this in class.)

These points will be particularly important when we start developing R functions.

## Comments

The character "`#`" is used by R to indicate the start of a comment. This character and anything following it on a line is ignored by R. This makes it possible to include documentation in-place with R code. Here is an example of a couple of comments documenting an R function.

```
###  An R Factorial Function
###  This function uses recursion to compute factorials.
fact =
    function(n)
    if (n <= 0) 1 else n * fact(n - 1)
```

For practice, type this function into the script editor and load it into R. Save the script as a file called `fact.R`.

## Some Practice Exercises

Try out the following exercises.

```
1 + 1             #  Basic arithmetic
sqrt(pi)          #  Using a builtin constant
x = sqrt(2)       #  Square root of 2
y = 2^(1 / 2)     #  Square root of 2
2^10              #  Exponentiation
2^2^10            #  Exponentiating
(2^2)^10          #  Exponentiation works right to left

.1 + .2           #  Add .1 and .2
```

```
.1 + .2 == .3     #  Is .1 + .2 equal to .3?  (wtf?)

factorial(1:10)   #  Factorials
factorial(1:200)  #  Largest representable factorial?

1:10              #  Sequences
10:1              #  Sequences
c(1:10, 10:1)     #  Sequences

runif(10)         #  Random numbers
hist(runif(1000)) #  Lots of random numbers

demo(graphics)    #  R Plots
demo(persp)       #  Surface plots
```