Quant Methods: Programming Basics 1 - Variables

Programming Basics Part 1 - Variables (vectors, matrices)

R can be used to do mathematical calculations (essentially like a calculator), e.g.

```
8*7
```

```
## [1] 56
```

You have to get used to writing out your equations, but I use R often rather than using a calculator to do a range of calculations.

When we think about programming, we typically want to do some sort of calculation, but we want to set it up in a general way that allows us to use the resulting "code" for a range of calculations instead of a single, specific calculation. To do this, we generally use variables (e.g., x, y) rather than specific numbers, where the variables can take on a range of values. For example,

```
x=8
y=7
z=x*y
print(z)
```

```
## [1] 56
```

Clearly we haven't saved ourselves any work so far, but we have stored the answer as a variable (z), which could then be used in another calculation, as is true of x and y. We might also have tables of values of x and y that we could read in, rather than "hard wiring" their values in the code.

One advantage of using variables is that they can represent more than 1 value at a time, e.g.

```
x = 8:10
print(x)
```

```
## [1]  8  9 10
```

This construction, using ":", creates a vector or list of values from the value to the left of the ":" to the value to the right, incremented by 1. Try y=8.5:10.5 and see what happens.

```
y = 8.5:10.5
print(y)
```

```
## [1]  8.5  9.5 10.5
```

What if we want to increment each value by 0.5 instead of 1? Then we'd have to take a different approach, using a built-in function seq() [seq for sequence].

```
y = seq(8.5,10.5,0.5)
print(y)
```

```
## [1]  8.5  9.0  9.5 10.0 10.5
```

Alternatively, what if we want to specify a particular sequence of numbers, not necessarily evenly spaced, e.g. 8, 9.25, 11, 12.5? We can construct any specific sequence using the c() command [c for concatenate]

```
q = c(8, 9.25, 11, 12.5)
print(q)
```

```
## [1]  8.00  9.25 11.00 12.50
```

We could do the same for a list of letters, or words:

```
qa = c("A","B","c","z","8")
qw = c("cat","bat",",mat")
print(qa)
```

```
## [1] "A" "B" "c" "z" "8"
```

```
print(qw)
```

```
## [1] "cat"  "bat"  ",mat"
```

Of course if these were long lists, typing them in would be tedious, error-prone and would fall in the category of "hard wired". Instead it would be better to have the values stored in a separate file, like a spreadsheet, and read them in. We'll talk about that next week.

Numbers and characters (words, letters) are two different types of variables.

```
typeof(q)  #typeof() refers to the way R stores a variable
```

```
## [1] "double"
```

```
typeof(qa)
```

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

```
class(q)  #class() us an attribute that affects what can be
```

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

```
class(qa)  #done with a variable
```

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

You can change a numeric variable into a character variable. Can you do the reverse?

```
p = as.character(q)
print(p)
```

```
## [1] "8"    "9.25" "11"   "12.5"
```

```
pa = as.numeric(qa)
```

```
## Warning: NAs introduced by coercion
```

```
print(pa)
```

```
## [1] NA NA NA NA  8
```

Now let's create 2 numeric vectors, x and y, each with 5 values. What can we do with them? To start, simple arithmetic like add and subtract:

```
x = 8:12
y = c(1.5,3,4.1,6,7.7)
z1 = x + y
z2 = x - y
print(z1)
```

```
## [1]  9.5 12.0 14.1 17.0 19.7
```

```
print(z2)
```

```
## [1] 6.5 6.0 5.9 5.0 4.3
```

We can also multiply, though you may know that there is more than one way to "multiply" two vectors (e.g., dot product). Let's see what happens if we simply multiply and divide x and y.

```
z3 = x * y
z4 = x / y
print(z3)
```

```
## [1] 12.0 27.0 41.0 66.0 92.4
```

```
print(z4)
```

```
## [1] 5.333333 3.000000 2.439024 1.833333 1.558442
```

What we see is that R understands this to mean that each element of x is multiplied or divided by the corresponding element in y, which is probably mostly what we want it to mean. What do you think would happen if x is longer than y? Let's see:

```
xs = 8:12
ys = c(1.5,3,4.1,6)
zs = xs * ys
```

```
## Warning in xs * ys: longer object length is not a multiple of shorter object
## length
```

```
print(zs)
```

```
## [1] 12 27 41 66 18
```

R warns us of a problem but still gives an answer. Where you do you the value of 18 came from? R will cycle numbers if one variable runs out before another in a computation. BE CAREFUL!

There are some particular vectors that can be useful. We can create a vector of 0's or NAs or 1's using rep():

```
a = rep(0,10)
b = rep(1,3)
c = rep(NA,5)
print (a)
```

```
##  [1] 0 0 0 0 0 0 0 0 0 0
```

```
print (b)
```

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

```
print (c)
```

```
## [1] NA NA NA NA NA
```

Notice on the upper right panel of RStudio there is an Environment tab. Look at what is there - basically every variable we defined (unless we redefined it), along with its size and first some values.

We can also a length function, length(), and a dimension function dim()