Functions

Statistics 427: R Programming

Module 3

2020

Functions

Sometimes there are calculation tasks that must be performed again and again. `R` contains many useful calculation tasks programmed into functions. You can use these in the command console or in any scripts.

A function is usually designed to take a quantity or a vector, *called the argument of the function*, and calculate something with it; it returns the value of the calculation as output for further use.

Sum

The vector x is the *argument* of the function and y is the returned value

```r
x=c(5.3,-2.6,1.1,7.9,-4)
y=sum(x)
y
```

```
[1] 7.7
```

Common functions

`length(x)`: returns the number of elements in vector `x`
`sum(x)`: returns the sum of the elements in `x` (can be vector or quantity)
`prod(x)`: returns the product of the elements in vector `x`
`cumsum(x)`: returns the cumulative sum of elements in vector `x`
`cumprod(x)`: returns the cumulative product of elements in vector `x`
`mean(x)`: returns the arithmetic average of the elements in vector `x` (equivalent to `sum(x)/length(x)`)
`abs(x)`: returns a vector containing the absolute values of elements in `x` (can be quantity or vector)
`sqrt(x)`: returns a vector containing the square roots of elements in `x` (can be quantity or vector); is identical to `x^.5`

Examples of common functions I

```r
x=c(2,3,5,1,0,-4)
length(x)
```

```
[1] 6
```

```r
y=sum(x); y
```

```
[1] 7
```

```r
prod(x)
```

```
[1] 0
```

```r
cumsum(x)
```

```
[1]  2  5 10 11 11  7
```

Examples of common functions II

```r
cumprod(x)
```

```
[1]  2  6 30 30  0  0
```

```r
mean(x)
```

```
[1] 1.166667
```

Examples of common functions III

Watch for errors :-)

```r
abs(x); abs(y)
```

```
[1] 2 3 5 1 0 4
```

```
[1] 7
```

```r
sqrt(x); sqrt(y)
```

```
[1] 1.414214 1.732051 2.236068 1.000000 0.000000      NaN
```

```
[1] 2.645751
```

```r
# to fix NaNs, do not take sqrt of values <= 0
```

Trig (and there was much rejoicing…yay)

You can use the `R` functions freely with the assignment statement for storage and future use, use `R` functions or `R` calculations as arguments in other functions.

Pythagoras' theorem states that the length of the hypotenuse of a right triangle is equal to the square root of the sum of the squares of the other two sides. Suppose the two lesser sides of a right triangle have lengths 12.3 and 20.5.

$$h = \sqrt{a^2 + b^2}$$

```r
a=12.3; b=20.5; sides=c(a,b)
h=sqrt(sum(sides^2)); h
```

```
[1] 23.9069
```

Creating New Functions in R

`R` also allows for user-defined functions. It is fortuitous for saving time in calculations where the same formulas are done many times. They are also nice for when a built-in `R` function or one from an `R` package (more on that later!) does not have quite the output your are looking for.

How to build your R function

The general form of `function()` is:

```
function(arglist) expr
return(value)
```

> `arglist`: empty or one or more name=expression terms
> `expr`: an expression
> `value`: an expression

Build one for Pythagoras I

```r
hyp.length=function(x){
  h=sqrt(sum(x^2))
  return(h)
}
```

Build one for Pythagoras II

Now we run the function as is to get it into the `R` environment, and then it is ready for the argument `x`, which has to be two values (the two lesser sides of the right triangle).

```r
sides=c(3,4)
hyp.length(sides)
```

```
[1] 5
```

Functions redux

What is with the use of the curly braces `{ }`?!? That was NOT in the general form.

The use of the curly braces `{ }` makes it so that you can have more than one executable statement per function. The `return()` command will return the value of the argument(s) created within the function.

The indenting done within the writing of the function did not have to be there, it is just a preference for ease of reading, as `R` scripts are format-free.

Important things of note for functions

(1) In the function, `x` as an argument did NOT have to be named `x`, it could be anything. Same with `h`
(2) The values of `h` and `x` inside the function are "local" to the function; the vectors exist within a special place in the computer memory set aside specifically for the function, and their values are not available outside the function (typing `h` outside the function will not produce any results)
(3) Although the function `hyp.length()` exists within the `R` environment workspace, just like `sum()` or `sqrt()`, the new function is not permanent and will disappear once the `R` session ends. Saved functions can be run again, so do not forget to save the function!

Atomic Weights and Molar Masses I

A *mole* of an element or compound is defined to contain 6.02×10^{23} particles of that element or compound (atoms in the case of an element; molecules in the case of a compound). That (large) number is known as Avogadro's number. It was picked by chemists as a sort of standard amount of a substance and represents the number of atoms in 12 g of pure carbon-12. The *atomic mass* of an atom is the mass in grams of a mole of those particular atoms. A carbon-12 atom has an atomic mass of 12, and so a mole of carbon-12 atoms would weigh 12 g at sea level on Earth.

In the Earth's crust and atmosphere, some heavier or lighter isotopes of most elements occur naturally in small amounts, and so on average, a mole of everyday unpurified carbon atoms would have a mass of around 12.01 g due to trace amounts of heavier carbon (specifically carbon-14).

Atomic Weights and Molar Masses II

We want to calculate the mass of mole of a molecule of water. To do so requires finding the atomic weight of one molecule of water. A water molecule has two hydrogen atoms and one oxygen atom, and we must combine their atomic weights (commonly found in chemistry tables). The atomic weight of one hydrogen molecule is 1.008 and oxygen is 16.

$$mass = \sum a * w$$

Where:

`a`: number of atoms per molecule
`w`: atomic weight of each atom

Mole weight function

```r
a=c(2,1)
w=c(1.008,16)
# x=a and y=w
mole.mass=function(x,y){
  mm=sum(x*y)
  return(mm)
}
```

Mole weight function alternative

You do not have to use the actual letters `x` and `y`. You can use whatever letters you want, with some restrictions. You cannot start a name with a number or character other than a letter (upper- or lower-case), you can use periods in between words or letters, numbers can be used in the name as long as it is not the first character in the name, underscores can be used, but special characters cannot be used (like $, &, etc.).

```r
a=c(2,1)
w=c(1.008,16)
# x=a and y=w
mole.mass=function(a,w){
  mm=sum(a*w)
  return(mm)
}
mole.mass(a,w)
```

```
[1] 18.016
```

One more use of the function

Sucrose consists of 12 carbon, 22 hydrogen, and 11 oxygen atoms ($C_{12}H_{22}O_{11}$). The atomic weight of carbon is 12.01, with hydrogen and oxygen are already known from previous example of 1.008 and 16, respectively.

```r
a=c(12,22,11)
w=c(12.01,1.008,16)
# no need to rerun function if it worked before and you have
# not shut down the R session
mole.mass(a,w)
```

```
[1] 342.296
```