CSC 120: Computer Science for the Sciences (R section)

Radford M. Neal, University of Toronto, 2015

http://www.cs.utoronto.ca/∼radford/csc120/

Week 4

Combining Data of Different Types in a List

We've seen how we can put several numbers into a vector of numbers. Or we can put several strings into a vector of strings. But what if we want to combine both types of data? Let's try...

```
> c(123,"fred",456)
[1] "123"  "fred" "456"
```

R converts the numbers to character strings, so that the elements of the vector will all be the same type (character).

But we *can* put together data of different types in a *list*:

```
> list(123,"fred",456)
[[1]]
[1] 123

[[2]]
[1] "fred"

[[3]]
[1] 456
```

Lists Can Contain Anything

Elements of a list can actually be anything, including vectors of different lengths:

```
> list (1:4, 3:10)
[[1]]
[1] 1 2 3 4


[[2]]
[1]  3  4  5  6  7  8  9 10
```

You can even put lists within lists (though these are hard to read when printed):

```
> list(4,list(5,6))
[[1]]
[1] 4


[[2]]
[[2]][[1]]
[1] 5


[[2]][[2]]
[1] 6
```

Extracting and Replacing Elements of a List

You can get a single element of a list by subscripting with the `[[ ...]]` operator:

```
> L <- list (c(3,1,7), c("red","green"), 1:4)
> L[[2]]
[1] "red"   "green"
> L[[3]]
[1] 1 2 3 4
```

You can replace elements the same way. Continuing from above...

```
> L[[3]] <- c("x","y","z")
> L
[[1]]
[1] 3 1 7


[[2]]
[1] "red"   "green"


[[3]]
[1] "x" "y" "z"
```

Notice that the new value can have a type different from that of the old value.

Making Lists Bigger

You can make list longer by assigning to an element that doesn't exist yet:

```
> L <- list(1,3)
> L
[[1]]
[1] 1

[[2]]
[1] 3
> L[[3]] <- "xx"
> L
[[1]]
[1] 1

[[2]]
[1] 3

[[3]]
[1] "xx"
```

You can create a list this way starting with an empty list made with `list()`.

Giving Names to List Elements

You can give names to elements of a list, and then refer to these elements by name with the `$` operator. For example:

```
> L <- list (a=c(3,1,7), bc=c("red","green"), q=1:4)
> L$a
[1] 3 1 7
> L$bc
[1] "red"   "green"
> L$q <- TRUE
> L
$a
[1] 3 1 7

$bc
[1] "red"   "green"

$q
[1] TRUE
```

If an element has a name, R uses it for printing, rather than the numerical index.

Using a List to Return Multiple Values from a Function

This function takes as input a vector of character strings, and returns a list of two vectors, with the first and the last characters of the input strings:

```
first_and_last_chars <- function (strings) {
    first <- character(length(strings)) # Create two string vectors for
    last <- character(length(strings))  # the results, initially all ""
    for (i in 1:length(strings)) {
        nc <- nchar(strings[i])
        first[i] <- substring(strings[i],1,1)  # Find first & last chars
        last[i] <- substring(strings[i],nc,nc) # of the i'th string
    }
    list (first=first, last=last)  # Return list of both result vectors
}
```

Here's an example of its use:

```
> fl <- first_and_last_chars (c("abc","wxyz"))
> fl$first
[1] "a" "w"
> fl$last
[1] "c" "z"
```

Specifying Function Arguments by Name

Suppose you define a function with several arguments, such as

```
hohoho <- function (times, what) {
    r <- what
    while (times > 1) { r <- paste(what,r); times <- times-1 }
    r
}
```

You can call the function by just giving values for the arguments, in the same order as in the function definition. For example:

```
> hohoho (3, "ho")
[1] "ho ho ho"
```

But you can instead specify arguments using their names, in any order:.

```
> hohoho (times=3, what="ho")
[1] "ho ho ho"
> hohoho (what="ho", times=3)
[1] "ho ho ho"
```

This is very useful if there are many arguments, whose order is hard to remember.

Default Values for Function Arguments

When you define a function, you can specify a *default* value for an argument, which is used if a value for the argument isn't specified when the function is called. For example, here is the `hohoho` function with defaults for both arguments:

```
hohoho <- function (times=3, what="ho") {
    r <- what
    while (times > 1) { r <- paste(what,r); times <- times-1 }
    r
}
```

Here are some calls of this function:

```
> hohoho(4)            # 'what' will default to "ho"
[1] "ho ho ho ho"
> hohoho(what="hee")   # 'times' will default to 3
[1] "hee hee hee"
> hohoho()             # uses defaults for both arguments
[1] "ho ho ho"
```

This is very useful for functions with many arguments that are often set to the same (default) value, as is the case for many of R's pre-defined functions.

Creating a Plot in Stages

Many simple plots can be created with a single `plot` command — eg, `plot(x,y)` will plot points with coordinates given by the vectors `x` and `y`.

More complicated plots can be created in stages by adding more points, lines, and text to what has already been plotted.

The general approach:

- Create a new plot with `plot`. It might contains some points or lines, or might be completely empty. Features such as the axis scales and labels are determined at this stage.

- Then add more information, using functions such as `points`, `lines`, `abline`, and `text`. You can call these functions as many times as needed, perhaps with different options for things like colour and line width each time.

Creating a New Plot

You create a new plot with the `plot` function. It takes one or two data vectors as its first arguments, but has many, many other possible arguments. You'll want to let most of these have their default values, and refer to any that you set by name.

Here are some of the possible arguments to `plot`:

`type`	Type of plotting — `"p"` for points (the default), `"l"` for lines, `"b"` for both points and lines, `"c"` for lines only but with space for points
`col`	Colour for points/lines plotted (default is `"black"`)
`xaxt`	Set to `"n"` to get rid of horizontal axis numbers
`yaxt`	Set to `"n"` to get rid of vertical axis numbers
`xlab`	Label for the horizontal axis
`ylab`	Label for the vertical axis
`xlim`	Horizontal range for plot (vector of length two)
`ylim`	Vertical range for plot (vector of length two)
`asp`	Aspect ratio, `asp=1` ensures one vertical unit looks the same length as one horizontal unit

For example, `plot (c(), xlim=c(0,2), ylim=c(1,5))` will plot an empty frame with horizonal axis labels from 0 to 2 and vertical axis labels from 1 to 5.

Adding Points to a Plot

We can add points to a plot with the `points` function. Like `plot`, it takes two vectors as its first two arguments, containing the x and y coordinates of the points. (Or just a single vector argument with the y coordinates, in which case the x coordinates are 1, 2, 3, ...)

It can also take other arguments that set various options, such as

`type`	Set to `"b"` for lines as well as points
`col`	Colour for points plotted
`pch`	Character to plot points with — default is a circle, other possibilities are `pch="x"` for plotting with x symbols, or `pch=20` for solid dots

For example, `points (x, y, col="red", pch=20)` will add solid red dots to the plot, at the coordinates given by the vectors `x` and `y`.

Adding Lines to a Plot

We can add lines to a plot with the `lines` function.

In addition to one or two arguments giving the coordinates of the points to connect with lines, it can take other arguments such as those below (which can also be used for `plot`):

`type`	Set to `"b"` for points too, `"c"` for lines only but with space for points
`col`	Colour for lines plotted
`lty`	Line type — eg, `"dotted"`, `"dashed"`, or `"solid"` (the default)
`lwd`	Line width (default is 1)

For example, `lines (y, col="green", lty="dotted")` will add dotted green lines to the plot, at the x coordinates 1, 2, 3, ... and y coordinates given by the vector `y`.

Adding Text to a Plot

We can add text to a plot with the `text` function.

Here's an example that adds "WOW" to the origin of the plot:

```
> text (0, 0, "WOW")
```

We can put many character strings on a plot with one call of `text`, since its arguments can be vectors of x coordinates, y coordinates, and character strings.

For example:

```
> x <- 1:10
> y <- x^2
> plot(x,y,xlim=c(0,11))
> text(x,y+2,paste("square of",x))
```

Example: Drawing a Spiral

Here's an example R script that draws a spiral in a plain box, using 7 segments each time it winds around, with red dots at the vertices. The start and end are labelled with "start" and "end".

```
n <- 20
angle <- 2*pi*(0:n)/7
dist <- 0:n
x <- dist * cos(angle)
y <- dist * sin(angle)

plot (x, y, type="c", xaxt="n", yaxt="n", xlab="", ylab="",
            xlim=c(-n,n), ylim=c(-n,n), asp=1)

points (x, y, col="red")

text (x[1], y[1]-1, "start")
text (x[n+1], y[n+1]+1, "end")
```

The Spiral Plot

```
> source("http://www.cs.utoronto.ca/~radford/csc120/spiral-script.r")
```