3 Objects, their modes and attributes

3.1 Intrinsic attributes: mode and length

The entities R operates on are technically known as *objects*. Examples are vectors of numeric (real) or complex values, vectors of logical values and vectors of character strings. These are known as "atomic" structures since their components are all of the same type, or *mode*, namely *numeric*[1], *complex*, *logical*, *character* and *raw*.

Vectors must have their values *all of the same mode*. Thus any given vector must be unambiguously either *logical*, *numeric*, *complex*, *character* or *raw*. (The only apparent exception to this rule is the special "value" listed as `NA` for quantities not available, but in fact there are several types of `NA`). Note that a vector can be empty and still have a mode. For example the empty character string vector is listed as `character(0)` and the empty numeric vector as `numeric(0)`.

R also operates on objects called *lists*, which are of mode *list*. These are ordered sequences of objects which individually can be of any mode. *lists* are known as "recursive" rather than atomic structures since their components can themselves be lists in their own right.

The other recursive structures are those of mode *function* and *expression*. Functions are the objects that form part of the R system along with similar user written functions, which we discuss in some detail later. Expressions as objects form an advanced part of R which will not be discussed in this guide, except indirectly when we discuss *formulae* used with modeling in R.

By the *mode* of an object we mean the basic type of its fundamental constituents. This is a special case of a "property" of an object. Another property of every object is its *length*. The functions `mode(`*`object`*`)` and `length(`*`object`*`)` can be used to find out the mode and length of any defined structure[2].

Further properties of an object are usually provided by `attributes(`*`object`*`)`, see Section 3.3 [Getting and setting attributes], page 14. Because of this, *mode* and *length* are also called "intrinsic attributes" of an object.

For example, if `z` is a complex vector of length 100, then in an expression `mode(z)` is the character string `"complex"` and `length(z)` is `100`.

R caters for changes of mode almost anywhere it could be considered sensible to do so, (and a few where it might not be). For example with

```
> z <- 0:9
```

we could put

```
> digits <- as.character(z)
```

after which `digits` is the character vector `c("0", "1", "2", ..., "9")`. A further *coercion*, or change of mode, reconstructs the numerical vector again:

```
> d <- as.integer(digits)
```

Now `d` and `z` are the same.[3] There is a large collection of functions of the form `as.`*`something`*`()` for either coercion from one mode to another, or for investing an object with some other attribute it may not already possess. The reader should consult the different help files to become familiar with them.

---

[1] *numeric* mode is actually an amalgam of two distinct modes, namely *integer* and *double* precision, as explained in the manual.

[2] Note however that `length(`*`object`*`)` does not always contain intrinsic useful information, e.g., when *`object`* is a function.

[3] In general, coercion from numeric to character and back again will not be exactly reversible, because of roundoff errors in the character representation.

3.2 Changing the length of an object

An "empty" object may still have a mode. For example

```
> e <- numeric()
```

makes `e` an empty vector structure of mode numeric. Similarly `character()` is a empty character vector, and so on. Once an object of any size has been created, new components may be added to it simply by giving it an index value outside its previous range. Thus

```
> e[3] <- 17
```

now makes `e` a vector of length 3, (the first two components of which are at this point both `NA`). This applies to any structure at all, provided the mode of the additional component(s) agrees with the mode of the object in the first place.

This automatic adjustment of lengths of an object is used often, for example in the `scan()` function for input. (see Section 7.2 [The scan() function], page 31.)

Conversely to truncate the size of an object requires only an assignment to do so. Hence if `alpha` is an object of length 10, then

```
> alpha <- alpha[2 * 1:5]
```

makes it an object of length 5 consisting of just the former components with even index. (The old indices are not retained, of course.) We can then retain just the first three values by

```
> length(alpha) <- 3
```

and vectors can be extended (by missing values) in the same way.

3.3 Getting and setting attributes

The function `attributes(`*`object`*`)` returns a list of all the non-intrinsic attributes currently defined for that object. The function `attr(`*`object, name`*`)` can be used to select a specific attribute. These functions are rarely used, except in rather special circumstances when some new attribute is being created for some particular purpose, for example to associate a creation date or an operator with an R object. The concept, however, is very important.

Some care should be exercised when assigning or deleting attributes since they are an integral part of the object system used in R.

When it is used on the left hand side of an assignment it can be used either to associate a new attribute with *`object`* or to change an existing one. For example

```
> attr(z, "dim") <- c(10,10)
```

allows R to treat `z` as if it were a 10-by-10 matrix.

3.4 The class of an object

All objects in R have a *class*, reported by the function `class`. For simple vectors this is just the mode, for example `"numeric"`, `"logical"`, `"character"` or `"list"`, but `"matrix"`, `"array"`, `"factor"` and `"data.frame"` are other possible values.

A special attribute known as the *class* of the object is used to allow for an object-oriented style[4] of programming in R. For example if an object has class `"data.frame"`, it will be printed in a certain way, the `plot()` function will display it graphically in a certain way, and other so-called generic functions such as `summary()` will react to it as an argument in a way sensitive to its class.

To remove temporarily the effects of class, use the function `unclass()`. For example if `winter` has the class `"data.frame"` then

```
> winter
```

[4] A different style using 'formal' or 'S4' classes is provided in package `methods`.

will print it in data frame form, which is rather like a matrix, whereas

```
> unclass(winter)
```

will print it as an ordinary list. Only in rather special situations do you need to use this facility, but one is when you are learning to come to terms with the idea of class and generic functions.

Generic functions and classes will be discussed further in Section 10.9 [Object orientation], page 48, but only briefly.