

Creating objects

Glen Sargeant

Northern Prairie Wildlife Research Center
Jamestown, North Dakota
glen_sargeant@usgs.gov

Hosted by the
USGS Western Fisheries Research Center
Seattle, Washington
24-25 March, 2009

Objective

Introduce a minimally sufficient toolkit for using expressions to create objects (we will practice and extend these skills later):

- Vectors
- Matrices
- Data frames
- Lists

Recall the purposes and properties of vectors

- Recall that **vectors** are the basic units of data organization. They are described by a **mode**, their **length**, and possibly additional **attributes**.
- Use of **expressions** (commands issued to the R console for evaluation) and **assignments** to create vectors and matrices is an essential component of R programming.

Expressions that return vectors

- Virtually everything is an object.
- Vectors are the most fundamental of objects.
- Almost every expression returns a vector or an object composed of vectors.
- The only (?) prominent exceptions...
 - ... create graphics,...
 - ... direct output to external files,...
 - ... or summarize object structure.

Expressions that return vectors

Any expression that returns (only) numbers, character strings, or logical values returns a vector.

- Entering a number, character string, or logical value
- Mathematical operations on vectors
- Logical operations on vectors
- Many functions that operate on vectors

Concatenating vectors (`c()`)

- The **concatenate** (`c()`) function combines vectors.
- Arguments must be *expressions that return vectors of the same mode*.
- The attributes of output (e.g., mode, class, length) are inferred from input.

Concatenating vectors: examples

- A numeric vector:

```
> c(1, 2, 3)
```

```
[1] 1 2 3
```

- A character vector: quotes identify character strings

```
> c("TRUE", "FALSE")
```

```
[1] "TRUE" "FALSE"
```

- A logical vector:

```
> c(TRUE, TRUE, FALSE, FALSE)
```

```
[1] TRUE TRUE FALSE FALSE
```

Concatenating vectors: examples

Recall that expressions can be **nested**, i.e., expressions can operate on the output of other expressions.

```
> c(1 + 1, log(10), exp(1))
```

```
[1] 2.000000 2.302585 2.718282
```

Shortcuts for vector construction

- If a vector has more than a few elements, hand entry is inefficient and likely to result in errors.
- Essential shortcuts include functions for creating **sequences** and **replicating** vectors.

Regular sequences (`seq()`)

Integers within a range:

```
> 1:5
```

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

Specified increment, decreasing order:

```
> seq(from = 4, to = 2, by = -2)
```

```
[1] 4 2
```

Specified number of intervals:

```
> seq(from = 1, to = 100, length.out = 4)
```

```
[1] 1 34 67 100
```

Repetition (`rep()`)

A vector \times 2:

```
> rep(1:2, times = 2)
```

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

Each element twice:

```
> rep(1:2, each = 2)
```

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

Each element a specified number of times:

```
> rep(1:2, times = c(1, 3))
```

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

Matrices and arrays

- Programming with matrices can greatly speed computations and is integral to some methods for ecological data.
- In R, matrices are vectors partitioned into rows and columns. The partition is specified by an additional attribute (a vector, `dim`).
- Matrices usually are constructed from vectors with `matrix()`, or from vectors or matrices with `rbind()` or `cbind()`.
- Arrays (any number of dimensions) generalize vectors and matrices and usually are constructed with `array()`.

matrix()

```
> matrix(data = 1:9, nrow = 3, ncol = 3)
```

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

```
> matrix(data = 1:9, nrow = 3, ncol = 3, byrow = TRUE)
```

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

Recycling

R may **recycle** vectors as necessary to complete operations. Note, for example, that a matrix can be constructed from a vector with fewer elements:

```
> matrix(data = 1:4, nrow = 3, ncol = 3)
```

	[,1]	[,2]	[,3]
[1,]	1	4	3
[2,]	2	1	4
[3,]	3	2	1

Why should you be wary of recycling?

`cbind()` and `rbind()`

Vectors and matrices can be combined with `rbind()` or `cbind()`.

```
> cbind(x = 1:5, y = 6:10)
```

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

```
> rbind(x = 1:5, y = 6:10)
```

	[,1]	[,2]	[,3]	[,4]	[,5]
x	1	2	3	4	5
y	6	7	8	9	10

Data frames

- Recall **data frames**, which combine vectors of the same length but possibly different modes.
- Data frames reflect the familiar organization of data in spreadsheets (1 record/row, 1 variable/column).
- Data frames can be viewed as generalized matrices that support certain matrix operations... but also share some properties of lists.

data.frame()

Data frames usually are constructed with `data.frame()`.

```
> unit <- c("a", "b", "c")  
> treated <- c(TRUE, TRUE, FALSE)  
> data.frame(unit, treated)
```

	unit	treated
1	a	TRUE
2	b	TRUE
3	c	FALSE

`cbind()` and `rbind()`

Data frames can be combined with other data frames and vectors:

- `data.frame()` Numbers of rows *usually* should match!
- `cbind()` Numbers of rows *usually* should match!
- `rbind()` Columns (number *and mode*) must match!

What happens if rows/columns do not match?

Lists

Recall **lists**, which combine any number of objects of any type.

- Support coordinated storage of data, metadata, methods, and results
- Often are used to represent complex datasets
- Support coordinated processing of multiple datasets

list()

Lists can be constructed from existing objects, which may be of different types:

```
> list(numeric.matrix = matrix(data = 1:4, nrow = 2,
+   ncol = 2), character.vector = c("a", "b"))
```

```
$numeric.matrix
```

```
      [,1] [,2]
[1,]    1    3
[2,]    2    4
```

```
$character.vector
```

```
[1] "a" "b"
```

vector()

Empty lists can be constructed for subsequent use (e.g., for storing components of a subsequent analysis):

```
> vector(mode = "list", length = 2)
```

```
[[1]]
```

```
NULL
```

```
[[2]]
```

```
NULL
```

Recap of objects (again!)

Vectors are the basic units from which other objects are constructed.

Matrices (and arrays) generalize vectors to ≥ 1 dimension.

Data frames generalize matrices (permit columns of different modes).

Lists are fully general (comprise any number of objects of any type).

Attributes describe, structure, and distinguish different types of objects.

Key functions

`c()`
`seq()`
`rep()`
`matrix()`

`rbind()`
`cbind()`
`data.frame()`
`list()`
`vector()`

Emerging issues and information needs

- We've learned how to create basic objects:
 - Where do objects go once created?
 - How are objects preserved for subsequent use?
- We have learned 9 functions; R includes thousands:
 - How does one identify functions that perform specific tasks?
 - How does one remember arguments and syntax?
- R is obviously a poor platform for data entry:
 - How does one create objects that represent large datasets?

Acknowledgements

These workshop notes describe the use of software developed and documented by the R Development Core Team. The correct citation for R is as follows:

R Development Core Team (2008). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.

The use of trade, product, or firm names does not imply endorsement by the U.S. Government.