Lecture 2: Data Structures in R

STAT598z: Intro. to computing for statistics

Vinayak Rao

Department of Statistics, Purdue University

The R programming language

From the manual,

- R is a system for statistical computation and graphics
- R provides a programming language, high level graphics,interfaces to other languages and debugging facilities

It is possible to go far using R interactively

However, we will also study the language with the goals of

- writing good software
- allowing easy reproducibility of our analyses

'Everything in R is an object'

An object consists of a symbol (name) and a value

- The function class() returns the object's class
- Useful for object-oriented programming E.g. Polymorphism lets the same function (print, plot) do different things to different objects

Also relevant: typeof(), mode() and storage.mode()

R types

typeof() gives the type or internal storage mode of an object

Common types include:

- **atomic vectors**: "logical", "integer", "double", "complex", "character", "raw"
- **list**: Various useful data-structures
- **closure**: Functions
- **symbol**: Variable names
- **Miscellaneous**: Various internal and advanced types

Atomic vectors

Informally, often just called 'vectors'

Contiguous collections of objects of the same type

(Contiguous: stored sequentially in memory)

Common types include: "logical", "integer", "double", "complex", "character", "raw"

R has no scalars, just vectors of length 1

Creating length one vectors

In [ ]:
```
age <- 15  # Length 1 vector
```

In [ ]:
```
name <- 'Bob'
```

In [ ]:
```
old_enough <- age >= 18  #old_enough <- FALSE
```

In [ ]:
```
print(name)
```

In [ ]:
```
old_enough
```

Comments:

- age, name, and old_enough are variable names
- ' <- ' is the assign operator
- ' = ' usually works but is not recommended

In [ ]:
```
16 -> age # Valid, but harder to read
```

In [ ]:
```
typeof(age) # Note: age is a double
```

In [ ]:
```
class(age)
```

In [ ]:
```
typeof(name)
```

In [ ]:
```
class(name)
```

In [ ]:
```
age <- 19L
typeof(age)
```

General vectors:

The c() function (**concatenate**) creates vectors

In [ ]:
```
people <- c("Alice", "Bob", 'Carol') # single/double quotes
```

In [ ]:
```
years <- 1991 : 2000  # Watch out for: years <- 2000:1991
```

In [ ]:
```
even_years  <- (years %% 2) == 0
```

In [ ]:
```
class(people)
```

In [ ]:
```
typeof(years)
```

In [ ]:
```
is.vector(even_years)
```

Indexing elements of a vector

Use brackets [] to index subelements of a vector

First element of a vector is indexed by 1

In [ ]:
```
people[1]    # First element is indexed by 1
```

In [ ]:
```
years[1 : 5] # Index with a subvector of integers
```

In [ ]:
```
years[c(1, 3, length(years))]
```

Negative numbers exclude elements

In [ ]:
```
people[-1] # All but the first element
```

In [ ]:
```
years[c(-1, - length(years))] #All but first and last elementts
```

In [ ]:
```
years[ - c(1,length(years))] # Equivalently
```

Index with logical vectors

In [ ]:
```
even_years  # Same as print(even_years)
```

In [ ]:
```
years[even_years] # Index with a logical vector
```

Example

Sample 100 Gaussian random variables and find the mean of the positive elements

In [ ]:
```
xx <- rnorm(100, 0, 1) # Sample 100 Gaussians
indx_xx_pos <- (xx > 0) # Is this element positive
```

In [ ]:
```
xx_pos <- xx[indx_xx_pos] # Extract positive elements
```

In [ ]:
```
xx_pos_mean <- mean(xx_pos) # calculate mean
```

More terse:

In [ ]:
```
xx <- rnorm(100, 0, 1) # Sample 100 Gaussians
```

In [ ]:
```
xx_pos_mean <- mean(xx[xx > 0]) # calc. mean of positives
```

In [ ]:
```
xx_pos_mean
```

Replacing elements of a vector

Can assign single elements

In [ ]:
```
people[1] <- 'Dave'; print(people)
```

or multiple elements:

In [ ]:
```
years[even_years] <- years[even_years] + 1; print(years)
```

or assign multiple elements a single value (more on this when we look at recycling)

In [ ]:
```
years[-c(1,length(years))] <- 0; print(years)
```

How about years <- 0?

In [ ]:
```
years[] <- 0  # Maintains old vector
```

Coercion

What if we assign an element a value of the wrong type?

In [ ]:
```
vals <- 1 : 3
typeof(vals)
```

In [ ]:
```
vals[2] <- 'two'; print(vals)
typeof(vals)
```

R will **coerce** the vector to the more flexible type

In increasing flexibility: logical, integer, double, and character

The c() operator does the same

In [ ]:
```
stuff <- c( TRUE , 3L, 3.14, 'pi')
stuff
```

In [ ]:
```
typeof(stuff)
```

Use **lists** if you really wanted a heterogeneous collection

Objects can also be coerced into simpler ones if the context demands

In [ ]:
```
as.integer(3.4)
```

In [ ]:
```
1:5.5
```

More on the c() operator

Atomic vectors are always flat, even for nested c() operators

Example from Advanced R, Hadley Wickham:

In [ ]:
```
c(1, c(2, c(3, 4)))
```

A vector of vectors is still just a vector

Use lists/matrices/arrays if you want nested structure

What if we assign to an element outside the vector?

In [ ]:
```
years[length(years) + 1] <- 2015
```

In [ ]:
```
length(years); years
```

We have increased the vector length by 1

In general, this is an inefficient way to go about things

Much more efficient is to first allocate the entire vector

In [ ]:
```
vals <- 1 : 3
typeof(vals)
```

In [ ]:
```
vals[6] <- 6L
```

In [ ]:
```
print(vals)
```

Also get NAs if we access elements outside the range of the vector

NA (Not available)

NA is a length 1 constant to handle missing values

Different from NaN (not a number), which results from e.g. dividing 0 by 0

NA can be coerced into any of the earlier data types

A useful command is is.na()

Vector operations and recycling

Unary transformations to a vectors: mean, sum, power etc

Binary operations are usually elementwise

What if vectors have different lengths?

Recycle: repeat shorter vector till the lengths match

Very convenient, but can allow bugs to remain undetected

R gives a warning if longer length is not multiple of shorter

Recycling

In [ ]:
```
val <- 1 : 6
val + 1
```

In [ ]:
```
val + c(1,2)
```