MONASH University

MONASH BUSINESS SCHOOL

ETC4500/ETC5450 Advanced R programming

Week 6: Object-oriented Programming

`arp.numbat.space`

Outline

Outline

Assignments

Assignment 1

- Common problems
- Keep working on your package!
- Final version due on 31 May 2024

Assignment 2

- Questions?
- Due 19 April 2024

Outline

Thomas Lumley

- Professor of Statistics, University of Auckland
- First degree in pure mathematics from Monash University
- MSc in Applied Statistics from the University of Oxford
- PhD in Biostatistics from the University of Washington
- Fellow of the Royal Society of New Zealand
- Member of the R core team
- Maintainer of 11 CRAN packages including *survey*
- Author of *Biased & Inefficient* blog
- Advisor to the NZ government on statistical issues

Outline

Object oriented programming

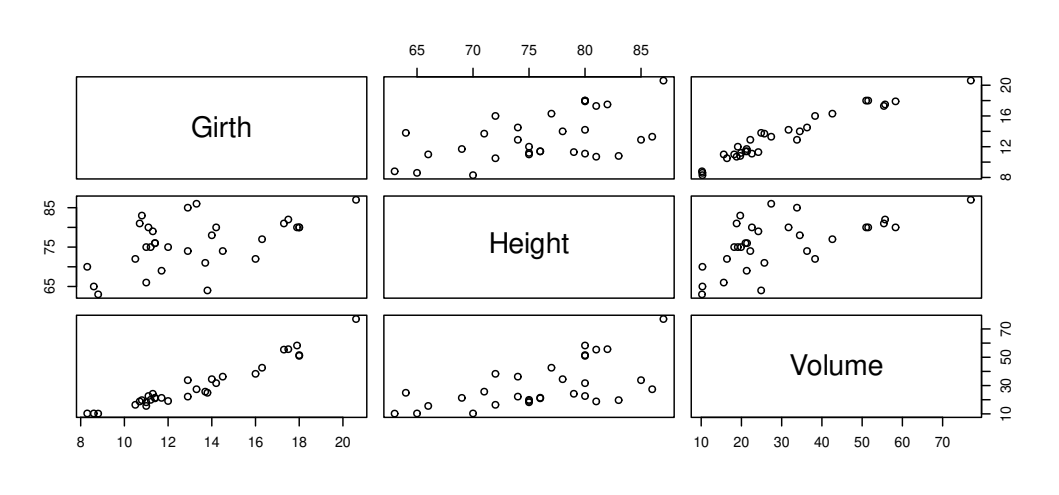
- **Encapsulation**: objects have secret internals that you don't need to understand
- **Polymorphism**: the same function can do different things to different data as appropriate
- **Inheritance**: you can take an existing kind of object and make a new, more specialised one

Inheritance turns out to be useful mostly for data infrastructure, but encapsulation and polymorphism are generally valuable

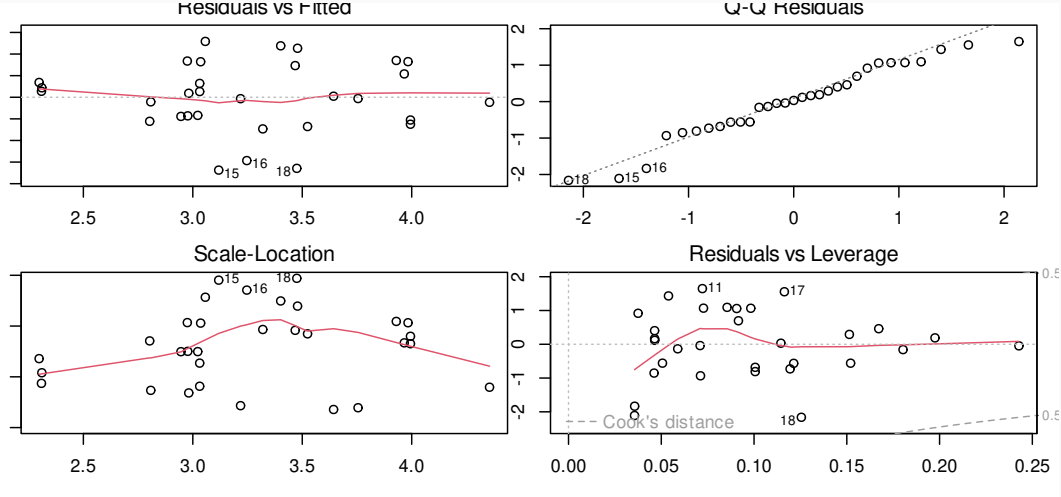
Generic functions and methods

A simple example: `plot`

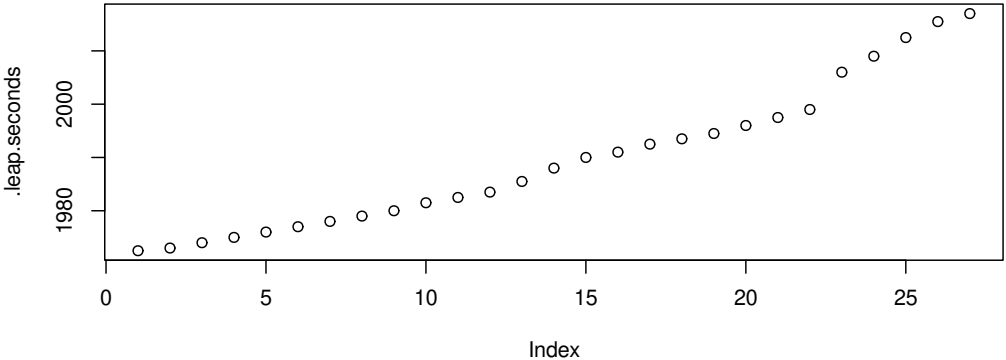
```r
plot(trees)
```


Girth
Height
Volume
65
70
75
80
85
8
12
16
20
10
30
50
70
8
10
12
14
16
18
20
10
20
30
40
50
60
70

```r
m<-lm(log(Volume)~log(Girth)+log(Height),
       data=trees)
par(mfrow=c(2,2),mar=c(3,1,1,1))
plot(m)
```



```r
plot(.leap.seconds)
```



```r
plot(nhtemp)
```

How?

- Giant `switch` statement…
- …that gets updated every time you load a package…
- ???

Generic functions and methods!

Object systems

R has **a lot** of object systems

- S3
- S3 `vctrs`
- S4
- R6
- R.oo, proto, R7

S3

Main topic for today

- easy to start writing
- no safeguards
- especially good for single-person, small to medium projects
- can be used for large projects with a lot of attention to documentation and communication
- limited use of inheritance
- basis of tidyverse and most of CRAN

S3 `vctrs`

A tidyverse package for making different sorts of vectors

- handles a lot of formatting and subsetting details
- allows for binary operators
- useful if you want your vectors in a tibble
- enforces some safeguards

S4

- more work to start writing
- objects know their structure
- enforces object structure
- better for large-scale collaborative programming
- better at inheritance
- basis of Bioconductor

R6

- Supports modifiable (mutable) objects
  - database connections, files, etc
  - large data objects we don't want to copy
  - interfacing to eg tensorflow
  - shared state between copies of an object
- not widely used otherwise
- more similar to other languages

Outline

S3

Back to `plot`

- `plot()` doesn't *do* anything
- All the work is done by *methods* for different types of object
- Methods are just ordinary functions
- When you call `plot`, R calls the appropriate `plot` method

Generic functions

- Generic functions don't *do* anything
- All the work is done by *methods* for different types of object
- Methods are just ordinary functions
  - with declarations in a package `NAMESPACE`
  - or R can guess based on function name

When you call the generic function R calls the appropriate method

Try these

```
print
methods("print")
stats:::print.acf
tools:::print.CRAN_package_reverse_dependencies_and_views
plot
methods("plot")
plot.ts
stats:::plot.lm
```

Also, try `methods("plot")` after loading another package

What do we notice?

- `print` functions are all different
- names start with `print.`, then the sort of thing they print
- mostly aren't visible just by name
-
-

Generic functions

- *methods* that actually do the work 'belong to' *generic functions*
- This is unusual: most popular OOP systems (Java, C++, Python) have methods belonging to data objects
- Important in R because functions are first-class objects (Week 5)
- Useful for functional programming with objects

Classes

- S3 classes are the things that specify which method to use
- Use `class` to attach a class to an object like a Post-It note
- That's all

```
x<-1
class(x)<-"numbat"
```

Try it

```
print.numbat <- function(x,...){
  cat(x,"numbats\n")
  invisible(x)
}
```

Defining classes

- R doesn't care what `class` you attach to an object
- **You** have to care
- `class(x) <- "lm"` makes R call `lm` methods on `x`
- **You** are responsible for these methods being appropriate
- Documentation is important
- No real enforcement of encapsulation

Ways to set up classes

- vectors plus attributes (`ts`, `POSIXct`, matrix)
- lists plus attributes (`lm`, `data.frame`)
- environments plus attributes

Try it

```
unclass(.leap.seconds)
unclass(nhtemp)
unclass(trees)
m<-lm(log(Volume)~log(Girth)+log(Height),data=trees)
str(m)
```

Defining classes safely

- Document what all is in a valid object of your class
- Have a limited set of places where one is created
- Consider having a pure constructor function (ARP 13.3.1)
- Consider having a validator function (ARP 13.3.1)
- Have a user-friendly function to make valid objects

This is more important if *someone else* might want to make an object from your class

Constructors

- user-friendly: `tibble`, `lm`, `acf`, `svydesign`
- pure: `new_factor`, `new_difftime` (ARP 13.3.1)

Makes a new object and ensures that it is valid. Stops users creating the object themselves.

Defining methods

- A method should have **the same** arguments as the generic (plus maybe more)
- The name of the method is

  `paste(generic,class,sep=".")`
- Less ambiguous: use a package and declare the functions
  - `S3method(generic, class)` in `NAMESPACE`
  - `@method generic class` with `devtools`
- `sloop::ftype` tells you about the type of a function
- `sloop::s3_get_method` or `getAnywhere` finds methods even if they're hidden

`default` methods

Called when there is no specific method for the object (no class, or no matching class)

- `mean.default`
- `summary.default`
- `head.default`

Adding methods to your class

No *rules* on which methods, but informal standards

- Start with `print`, `[`
- `summary` for more information
- `plot` or `image` if possible (`ggplot` methods take more work)
- `coef`, `vcov`, maybe `logLik` and `AIC` for models
- `resid` for models with residuals

Base S3 classes

- The class for method choice isn't just `class(x)` for base types
- Use `sloop::s3_class` to be sure

```
> s3_class(1)
[1] "double"  "numeric"
> s3_class(matrix(1,1,1))
[1] "matrix"  "double"  "numeric"
> class(1)
[1] "numeric"
> class(matrix(1,1,1))
[1] "matrix" "array"
```

Ambiguous cases

- `t` is a generic
- `t.test` is a generic
- `t.test.formula` is a method for `t.test`
- `t.data.frame` is a method for `t`
- `list` is not generic
- `list.files` isn't a method

Avoid using `.` as a word separator in function names that aren't methods. Use `camelCase` or `snake_case` or some other consistent approach

Defining generics

A typical generic function includes **only** a call to `UseMethod`

```
print
```

```
function (x, ...)
UseMethod("print")
<bytecode: 0x55d28a3cc4d0>
<environment: namespace:base>
```

Dispatch on another argument

Specify which argument to use for choosing the method (default is the first)

```
> survey:::svymean
function (x, design, na.rm = FALSE, ...)
{
    .svycheck(design)
    UseMethod("svymean", design)
}
```

Inheritance

The `class` attribute of an object can have multiple elements

- `UseMethod()` uses the first method that matches, or `default`
- `NextMethod()` uses the next method that matches

Polite conduct

- if you define a new generic, you can define methods for new and existing classes
- if you define a new class, you can define methods for new and existing generics
- don't define methods for someone else's class and generic (ask them)
- try not to define a generic with the same name as an existing one

Outline

S3 `vctrs`

A package for defining new S3 vectors

- makes it easy to have them as `tibble` columns
- requires a *pure constructor* to make new objects, so they will all be valid
- supports *double dispatch* for binary operators (eg: +,-)
- has a complicated but reliable system for type conversion
- requires a lot of method definitions to get started

vctrs.r-lib.org

An example: S3 and `vctrs`

The `rimu` package represents multiple-response questions

```
remotes::install_github("tslumley/rimu")
```

```
data(usethnicity)
race<-as.mr(strsplit(as.character(usethnicity$Q5),""))
race<-mr_drop(race," ")
mtable(race)
hispanic<-as.mr(usethnicity$Q4==1,"Hispanic")
ethnicity<-mr_union(race,hispanic)
plot(ethnicity)
e_S3<-ethnicity[101:120]
e_vctrs<-as.vmr(e_S3,na.rm=TRUE)
```

Inheritance

```
mr_union<-function(x,y,...) UseMethod("mr_union")

mr_union.default<-function(x,y,...){
  x<-as.mr(x)
  y<-as.mr(y)
  if (length(x)!=length(y))
    stop("different numbers of observations in x and y")

.....
}
```

```
mr_union.vmr<-function(x,y,...) {
    r<-NextMethod()
    as.vmr(r)
}
```

Encapsulation

- base S3 version is matrix of logical, plus levels attribute
  - works in `data.frame`
- vctrs S3 version is list of vectors of strings
  - works in `tibble`

Pure constructor

```
new_vmr <- function(x,levels=unique(do.call(c,x))) {
  vctrs::new_list_of(x, ptype = character(),
  class = "vmr", levs=levels)
}
```

‘helper’ functions

```
> methods("as.mr")
[1] as.mr.character*  as.mr.data.frame* as.mr.default*
[4] as.mr.factor*     as.mr.list*       as.mr.logical*
[7] as.mr.mr*         as.mr.ms*         as.mr.vmr*

> methods("as.vmr")
[1] as.vmr.default* as.vmr.mr*
```

Outline

S4

S4 requires classes and methods to be registered in R code (not just in packages)

- `setClass` defines the structure of a class
- `new` creates a new object from a class
- `setMethod` defines a method

It's possible to ask an object what methods it supports and get a reliable response.

S4 also allows multiple inheritance and multiple dispatch

Bioconductor

- Package system for high-throughput molecular biology
- Large data
- Structured data
- Annotated data
- New data types/structures all the time

It needs consistent infrastructure and large-scale collaboration: S4

bioconductor.org

Multiple dispatch

Choosing a method based on the class of more than one argument

- not very often useful
- important for matrices
- can be useful for plots

Multiple inheritance

`AnnDbObjBimap` is a class for storing look-up tables between different genomic identifiers (eg from different manufacturers) It is

- (by purpose) a two-way lookup object (`BiMap`)
- (by construction) an object containing a SQLite database (`DbObj`)

so it inherits generic functions from both these parents

Outline

R6

Remember function factories from last week?

- Create a function closure with useful variables in its environment
- These variables are visible inside the function

Now do this with

- multiple variables in the shared environment
- multiple functions inheriting this environment
- allow the shared variables to be modified

R6 is good for...

- large data objects, to reduce copying
- external objects that R can't just copy (database connections, files,...)
- shared state such as games

R6 is more similar to C++/Java/Python OOP. It doesn't really support functional programming

R6.r-lib.org