One of the appealing things about R is interactive flexibility

enter an expression and something (sensible) prints

this is an implicit `print()`

plot(anything) and something (sensible) plots

this is provided by object-oriented techniques

Why go through tedious technical programming stuff here?

some familiarity helps you avoid Confusion and Delay

what is an object?

you might need a function to return 1, 10 or 100 assorted things

a list can accomodate any combination of data

- including further lists

these can be hard to understand and keep track of

unless you have other functions that know what is what

some generic functions could know what to do with several kinds

- all print does is dispatch the data to a suitable method

programming tools built on this idea became popular in the 1990s

- **make life easier for programmers**
- **C++ (on top of C)**
- **Java**
- **extensions to perl, R**

two OO systems are part of R, S3 and S4

S3 is easy for casual use

```
x <- 100

x

class(x) <- ’sillyvalue’

print.sillyvalue <- function(z) {

 cat(’\n\n\n ’,z , ’... is a silly value\n\n’)

}

x
```

S3 is not an industrial-grade OO system

no requirement for classes to be pre-defined

no checking if class exists

no checking if data makes sense

simple-minded method dispatching

naming convention

in the orthodox OO religion ...

- there is a lot more formality
- objects are like companies
  - you know the products and prices
  - you’re not concerned with production details
  - there are laws and contracts
- compare with functions

S4 introduces some further OO formality

good for bigger programming projects

has pre-declared class definitions and methods

checking of data when object created or modified

still not fully orthodox

compromise for interactive use

another print example

```
setMethod('print',c('frog'),function(x){
          cat('\n\n\n ',x ,'is a frog\n\n')
         }
         m <- 'Rana esculenta'
         class(m) <- 'frog'
         m
```

what's missing from the example above is a class definition

unlike in S3 you are warned if class is not already defined

the simplest kind of definition is based on an existing class

```
setClass(Class='frog', representation='character')

n <- 100

is(n) # n is numeric

class(n) <- 'frog' # frog is supposed to be character!

is(n) # it is now, has been automagically coerced
```

all of the foregoing depend on a generic function

we won’t go into the details but it can be as simple as

```
frob <- function (x, ...){
useMethod(frob) } #S3
```

Objects are basically lists

you can make a list of all the information available

```
( model <- aov(weight ~ feed, chickwts) )

str(model)

is(lm)

lm$model
```

S4 objects have slots that have additional magick

- slots are defined in the using setClass
- they have a declared type
- they optionally also have validation methods
  - can check any other characteristics that might be required
- slots are addressed with @ similar to the way $ is used for lists
- packages generally have accessor functions
  - better to use these than addressing slots directly

```
One of the appealing things about R is interactive flexibility
        enter an expression and something (sensible) prints
                this is an implicit print()
        plot(anything) and something (sensible) plots
        this is provided by object-oriented techniques

Why go through tedious technical programming stuff here?
        some familiarity helps you avoid Confusion and Delay

what is an object?
        you might need a function to return 1, 10 or 100 assorted things
        a list can accomodate any combination of data
                including further lists
        these can be hard to understand and keep track of
        unless you have other functions that know what is what
        some generic functions could know what to do with several kinds
                all print does is dispatch the data to a suitable method

programming tools built on this idea became popular in the 1990s
        make life easier for programmers
        C++ ( on top of C)
        Java
        extensions to perl, R

two OO systems are part of R, S3 and S4
        S3 is easy for casual use
        x <- 100
        x
        class(x) <- 'sillyvalue'
        print.sillyvalue <- function(z) {
           cat('\n\n\n  ',z , '... is a silly value\n\n')
        }
        x

S3 is not an industrial-grade OO system
        no requirement for classes to be pre-defined
        no checking if class exists
        no checking if data makes sense
        simple-minded method dispatching
                naming convention

in the orthodox OO religion ...
        there is a lot more formality
        objects are like companies
                you know the products and prices
                you're not concerned with production details
                there are laws and contracts
        compare with functions

S4 introduces some further OO formality
        good for bigger programming projects
        has pre-declared class definitions and methods
        checking of data when object created or modified
        still not fully orthodox
                compromise for interactive use

another print example
        setMethod('print',c('frog'),function(x){
            cat('\n\n\n  ',x ,'is a frog\n\n')
        }
        m <- 'Rana esculenta'
        class(m) <- 'frog'
```

```
        m

what's missing from the example above is a class definition
        unlike in S3 you are warned if class is not already defined
        the simplest kind of definition is based on an existing class
        setClass(Class='frog', representation='character')
        n <- 100
        is(n)   #  n is numeric
        class(n) <- 'frog'  # frog is supposed to be character!
        is(n)   # it is now, has been automagically coerced

all of the foregoing depend on a generic function
        we won't go into the details but it can be as simple as
        frob <- function (x, ...){ useMethod(frob) } #S3

Objects are basically lists
        you can make a list of all the information available
        ( model <- aov(weight ~ feed, chickwts) )
        str(model)
        is(lm)
        lm$model

S4 objects have slots that have additional magick
        slots are defined in the using setClass
        they have a declared type
        they optionally also have validation methods
                can check any other characteristics that might be required
        slots are addressed with @ similar to the way $ is used for lists
        packages generally have accessor functions
                better to use these than addressing slots directly
```