Classes in R

Modified from materials by Mark Hansen, STAT 202a

Object oriented programming

Based on the idea of "objects" which can hold data, and "methods," which act on those objects

Much like functional programming, object-oriented programming is a concept more than a strict way languages behave.

Many object-oriented languages like Python and Java are (as Gentleman describes them) “class-centric”, meaning the classes define objects and are “repositories for the methods that act on” them

R, on the other hand, separates the class information from the creation of so-called generic functions and (again, quoting Gentleman) can be thought of as a “function-centric” system

Object systems in R

S3— the first OOP system in R

S4— second system in R, corresponds with version 4 of S

RC— encapsulated OO

R6— doesn't come with base R, but is maybe a better version of RC

more??

Your Turn

Which object system does Hadley Wickham think is the most important? Why?

*hint, look at v.2. of Advanced R

S3

Evolved out of a significant effort to introduce modeling functions (lm, glm, gam, loess) into the language in the early 1990s

There is no formal mechanism for representing instances of a class; they are, however, typically lists, where named elements of the list represent "slots"

You can access the class of an S3 object with the function `class()`; this can be used to both determine as well as set the class of an object (except for special cases involving implicit classes, this is the same as creating an attribute called `class` with value the string with the class name); finally `is.object()` tests to see if an R object has a `class` attrib

S3

As we have seen, the S3 class and method system is designed around the concept of a generic function; a generic function has different behaviors depending on the class of one or more of its arguments (this is known as polymorphism)

Generics perform a kind of method “dispatch” that, in turn, selects the appropriate method to be called; we’ve seen S3 generics before...

```
> print
function (x, ...)
UseMethod("print")
<bytecode: 0x1032250d0>
<environment: namespace:base>

> summary
function (object, ...)
UseMethod("summary")
<bytecode: 0x103338158>
<environment: namespace:base>
```

Your Turn

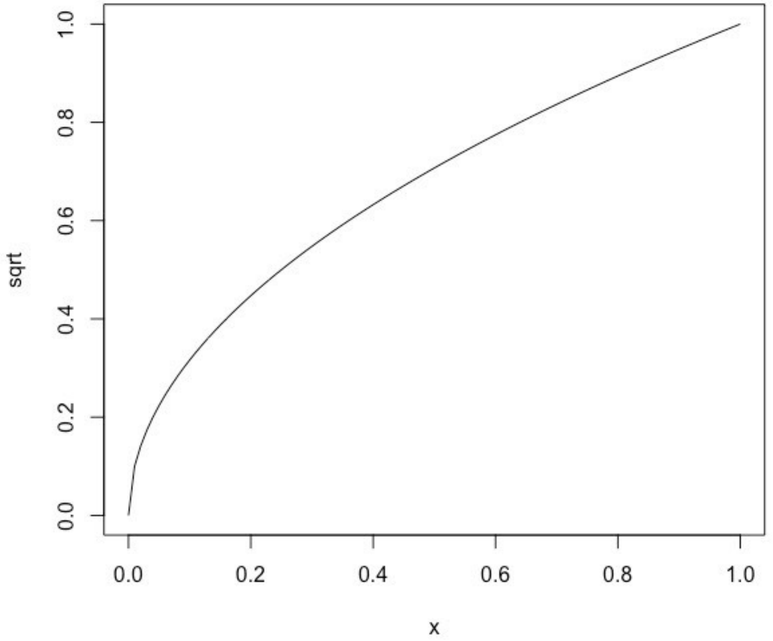
Find another three S3 generics.

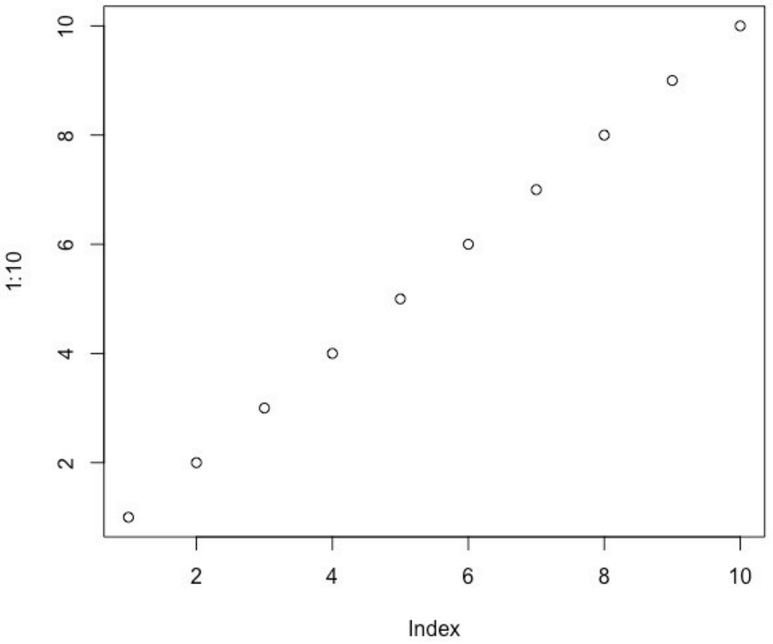
*hint, look in the Vocabulary section of v.1. of Advanced R

```
> plot
function (x, y, ...)
UseMethod("plot")
<bytecode: 0x102adb5f8>
<environment: namespace:graphics>


> plot(sqrt)


> plot(1:10)
```





So, it behaves differently for different types of objects

```
> plot

function (x, y, ...)

UseMethod("plot")

<bytecode: 0x102adb5f8>

<environment: namespace:graphics>
```

How do we find the actual code?

```
> plot.function
function (x, y = 0, to = 1, from = y, xlim = NULL, ylab = NULL,
    ...)
{
    if (!missing(y) && missing(from))
        from <- y
    if (is.null(xlim)) {
        if (is.null(from))
            from <- 0
    }
    else {
        if (missing(from))
            from <- xlim[1L]
        if (missing(to))
            to <- xlim[2L]
```

Your Turn

What are some other data types that have plot methods?
How do you know?

The function `UseMethod` dispatches on the class of the object returned by `class()`; methods are simply ordinary functions that are identified by a special naming convention

Specifically, methods are given names that are concatenations of the name of the generic method and the name of the class that they are intended to apply to, separated by a "`.`"

You can list the methods associated with a particular generic with a call to the function `methods()`

```
> methods(class="factor")
 [1] [             [[            [[<-          [<-
 [5] all.equal     as.character  as.data.frame as.Date
 [9] as.list       as.logical    as.POSIXlt    as.vector
[13] coerce        droplevels    format        initialize
[17] is.na<-       length<-      levels<-      Math
[21] Ops           plot          print         relevel
[25] relist        rep           show          slotsFromS3
[29] summary       Summary       xtfrm
see '?methods' for accessing help and source code
```

```
> methods("plot")
 [1] plot.acf*             plot.data.frame*    plot.decomposed.ts*
 [4] plot.default          plot.dendrogram*    plot.density*
 [7] plot.ecdf             plot.factor*        plot.formula*
[10] plot.function         plot.hclust*        plot.histogram*
[13] plot.HoltWinters*     plot.isoreg*        plot.lm*
[16] plot.medpolish*       plot.mlm*           plot.ppr*
[19] plot.prcomp*          plot.princomp*      plot.profile.nls*
[22] plot.raster*          plot.spec*          plot.stepfun
[25] plot.stl*             plot.table*         plot.ts
[28] plot.tskernel*        plot.TukeyHSD*
see '?methods' for accessing help and source code
```

Non-visible functions are asterisked

Your Turn

What data types does summary() have methods for? How do you know?

What about print()?

```
> methods("print")
  [1] print.acf*
  [2] print.anova*
  [3] print.aov*
  [4] print.aovlist*
  [5] print.ar*
  [6] print.Arima*
  [7] print.arima0*
  [8] print.AsIs
  [9] print.aspell*
 [10] print.aspell_inspect_context*
 [11] print.bibentry*
 [12] print.Bibtex*
 [13] print.browseVignettes*
 [14] print.by
 [15] print.changedFiles*
 [16] print.check_code_usage_in_package*
 [17] print.check_compiled_code*
 [18] print.check_demo_index*
 [19] print.check_depdef*
 [20] print.check_details*
 [21] print.check_details_changes*
 [22] print.check_doi_db*
 [23] print.check_dotInternal*
 [24] print.check_make_vars*
 [25] print.check_nonAPI_calls*
 [26] print.check_package_code_assign_to_globalenv*
 [27] print.check_package_code_attach*
 [28] print.check_package_code_data_into_globalenv*
 [29] print.check_package_code_startup_functions*
 [30] print.check_package_code_syntax*
 [31] print.check_package_code_unload_functions*
 [32] print.check_package_compact_datasets*
 [33] print.check_package_CRAN_incoming*
 [34] print.check_package_datasets*
 [35] print.check_package_depends*
 [36] print.check_package_description*
 [37] print.check_package_description_encoding*
 [38] print.check_package_license*
 [39] print.check_packages_in_dir*
 [40] print.check_packages_used*
 [41] print.check_po_files*
 [42] print.check_pragmas*
 [43] print.check_Rd_contents*
 [44] print.check_Rd_line_widths*
 [45] print.check_Rd_metadata*
 [46] print.check_Rd_xrefs*
 [47] print.check_RegSym_calls*
 [48] print.check_so_symbols*
 [49] print.check_T_and_F*
 [50] print.check_url_db*
 [51] print.check_vignette_index*
 [52] print.checkDocFiles*
 [53] print.checkDocStyle*
 [54] print.checkFF*
 [55] print.checkRd*
 [56] print.checkReplaceFuns*
 [57] print.checkS3methods*
 [58] print.checkTnF*
 [59] print.checkVignettes*
```

Generics are usually simple functions with just two arguments, one named x and one “...”; the first argument is typically used to trigger method dispatch (recall that “...” is the way we pass arguments to a function that don’t have to be named explicitly but can be used in the body of the function)

If you don’t include the catch-all “...”, then no method can have a formal argument that is not also a formal argument of the generic; it’s good practice to have all methods include the same formal arguments as the generic and for generics to include the “...” so methods added later can use other arguments

The downside, of course, is that if you mistype the name of a formal argument it will be swept up into the “...” without a warning

When the class attribute of an object contains **more than one string, R will run down the list in order**, looking for `generic.classname` for each entry, picking the first it finds; if it can't find one it will call the associated default method (if it exists, otherwise it produces an error)

For example, the lists on the previous slide all include defaults (`summary.default, residuals.default` and `AIC.default`)

Notice that as R runs through the class vector, it is really **running up the inheritance tree**, checking super-classes for an appropriate method -- Sometimes we want to make **explicit use of a method from a super-class**; with S3 we can achieve this kind of action by a call to `NextMethod()`

Base objects

As we said before, the **earliest versions of the S language were developed prior to widespread adoption of object-oriented programming principles**; as a result, some of the basic classes in R do not make use of the class attribute and are referred to as **implicit classes**

For example, functions are implicitly of class `function`, while matrices and arrays are implicitly of classes `matrix` and `array`, respectively

As a result, `is.object()` will return `FALSE` when applied to objects having an implicit class; `UseMethod` dispatches, however, on even implicit classes (depending only on the result of a call to `class()`)

```
> is.object(1:10)
[1] FALSE
> attr(1:10, "class")
NULL
> attr(mtcars, "class")
[1] “data.frame"
```

Your Turn

What methods exist for the class `lm`?

Use what you know about methods to

- Create a linear model with at least two predictors
- Plot the residuals of your model against one of the predictors