

CT5102: Programming for Data Analytics

Lecture 10: S3 Object System

Dr. Jim Duggan,
School of Engineering & Informatics
National University of Ireland Galway.

<https://github.com/JimDuggan/PDAR>

https://twitter.com/_jimduggan



Overview

Lectures 1-3	R Fundamentals <i>Atomic Vectors – Functions – Lists – Matrices – Data Frames</i>
Lectures 4-9	Data Science with R <i>ggplot2 – dplyr – tidyr – stringr – lubridate – purrr</i>
Lectures 10-11	Advanced Programming with R <i>Environments – Closures – S3 Object System</i>
Lectures 12	Machine Learning with R – Case Studies <i>Electricity Generation, Health</i>

Attributes

- All objects can have arbitrary additional attributes, used to store meta-data about the object
- Attributes can be thought of as a named list (with unique names)
- Attributes can be accessed:
 - Individually with **attr()**
 - All at once with **attributes()**

Example

```
<
> y<- 1:10
>
> attr(y,"Attribute1") <- "This is a vector"
> attr(y,"Time") <- Sys.time()
>
> str(y)
atomic [1:10] 1 2 3 4 5 6 7 8 9 10
- attr(*, "Attribute1")= chr "This is a vector"
- attr(*, "Time")= POSIXct[1:1], format: "2016-09-30 08:56:19"
```

structure() function

- The structure function returns a new object with modified attributes

```
> y<-structure(1:10,Att1="This is a vector", Att2=Sys.time())  
>  
> str(y)  
atomic [1:10] 1 2 3 4 5 6 7 8 9 10  
- attr(*, "Att1")= chr "This is a vector"  
- attr(*, "Att2")= POSIXct[1:1], format: "2016-09-30 09:00:00"
```

Properties of attributes

- By default, most attributes are lost when modifying a vector

```
>  
> attributes(y)  
$Att1  
[1] "This is a vector"  
  
$Att2  
[1] "2016-09-30 09:47:59 BST"  
  
>  
> attributes(y[1])  
NULL
```

3 Attributes not lost during operations...

- **Names**, a character vector giving each element a name.
names(x)
- **Dimensions**, used to turn vectors into matrices and arrays.
dim(x)
- **Class**, used to implement the S3 object system. *class(x)*

```
>
> x<-1:2
>
> names(x)<-c("a", "b")
>
> str(x)
Named int [1:2] 1 2
- attr(*, "names")= chr [1:2] "a" "b"
>
> str(x[1])
Named int 1
- attr(*, "names")= chr "a"
```

dim() example

```
> a<-1:6
>
> a
[1] 1 2 3 4 5 6
>
> attributes(a)
NULL
>
> dim(a)<-c(2,3)
>
> a
```

	[,1]	[,2]	[,3]
[1,]	1	3	5
[2,]	2	4	6

```
>
> attributes(a)
$dim
[1] 2 3
>
> attributes(a[1,])
NULL
>
> attributes(a[1,,drop=F])
$dim
[1] 1 3
```


Declaring a matrix

```
>  
> a <- matrix(1:6, ncol=3, nrow=2)  
>  
> a  
      [,1] [,2] [,3]  
[1,]    1    3    5  
[2,]    2    4    6  
>  
> dim(a)  
[1] 2 3
```

Challenge 10.1

- For the vector **1:100**, convert this to a **10 x 10 matrix** using the `attr()` function

S3 System

- Most OO languages
 - implement message-passing OO
 - Object determines which function to call
 - **canvas.drawRect("blue")**
- S3
 - Implements generic-function OO
 - A special type of function called a **generic function** decides which method to call (i.e. method dispatch)
 - **drawRect(canvas, "blue")**
 - S3 is a very casual system, it has no formal definition of classes



S3 Information

- The only OO system used in the base and stats packages, and the most commonly used in CRAN packages
- “S3 is informal and ad-hoc, but has a certain elegance in its minimalism” (Wickham 2015)

```
>
> library(pryr)
>
> typeof(mtcars)
[1] "list"
>
> class(mtcars)
[1] "data.frame"
>
> otype(mtcars)
[1] "S3"
```



Methods in S3

- In S3, methods belong to functions, called **generic functions**
- S3 methods do not belong to objects or classes
- To determine if a function is an S3 generic, inspect the source code for a call to **UseMethod()**
- UseMethod() – Figures out the correct method to call, the process of **method dispatch**
- Method names tend to be **generic.class()**

mean() function example

```
> mean
function (x, ...)
UseMethod("mean")
<bytecode: 0x1060f14d8>
<environment: namespace:base>
>
>
> ftype(mean)
[1] "s3"          "generic"
```

See all methods belonging to a generic

```
>  
> methods("mean")  
[1] mean.Date      mean.default  mean.difftime mean.POSIXct  mean.POSIXlt  
see '?methods' for accessing help and source code  
<
```

List all generics that have a method for a given class

```
> methods(class="data.frame")
```

[1]	[[[[[<-	[<-	\$	\$<-	aggregate
[8]	anyDuplicated	as.data.frame	as.list	as.matrix	by	cbind	coerce
[15]	dim	dimnames	dimnames<-	drop.levels	droplevels	duplicated	edit
[22]	format	formula	head	initialize	is.na	isUnknown	left
[29]	mapLevels	mapLevels<-	Math	merge	na.exclude	na.omit	NAToUnknown
[36]	nobs	Ops	plot	print	prompt	rbind	right
[43]	row.names	row.names<-	rowsum	show	slotsFromS3	split	split<-
[50]	stack	str	subset	summary	Summary	t	tail
[57]	transform	trim	unique	unknownToNA	unstack	within	

Defining classes and creating objects

- S3 objects usually built on top of lists, or atomic vectors with attributes
- Functions can also be S3 objects
- `class(x)` shows the class of an object

```
> o<-list(a="Test")
>
> str(o)
List of 1
 $ a: chr "Test"
>
> class(o)<-"my_object"
>
> str(o)
List of 1
 $ a: chr "Test"
 - attr(*, "class")= chr "my_object"
>
> class(o)
[1] "my_object"
```

Using structure() function

```
> o<-structure(list(a="test"),class="my_object")
>
> str(o)
List of 1
 $ a: chr "test"
  - attr(*, "class")= chr "my_object"
>
> class(o)
[1] "my_object"
```

Most S3 classes provide a constructor function

```
myobject <- function(x){  
  structure(list(a=x), class="my_object")  
}
```

```
o <- myobject("Test")
```

```
> o
```

```
$a
```

```
[1] "Test"
```

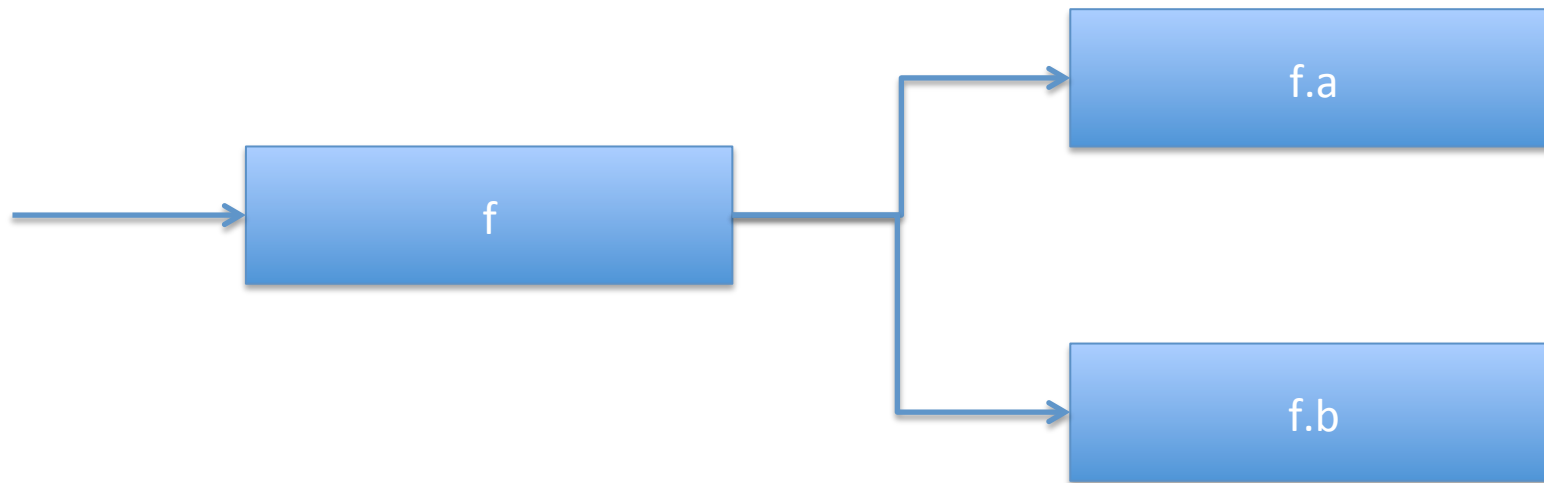
```
attr(,"class")
```

```
[1] "my_object"
```

Creating new methods and generics

- To add a new generic, create a function that calls **UseMethod()**
- UseMethod takes two arguments
 - The name of the generic function
 - The argument to use for method dispatch
- If the 2nd argument is omitted, it will dispatch on the first argument to the function
- Methods are then added, using a regular function with the name *generic.class*

Overall idea...



Format for specific functions are *[generic function].[class name]*

Example...

```
f <- function(x){  
  UseMethod("f")  
}
```

```
f.a <- function(x){  
  print("this is function f.a")  
}
```

```
f.b <- function(x){  
  print("this is function f.b")  
}
```

Calling the generic...

```
> x <- structure(list(),class="a")
>
> str(x)
list()
- attr(*, "class")= chr "a"
>
> f(x)
[1] "this is function f.a"
```

Default functions...

```
f.default <- function(x){  
  print("This is the default function")  
}
```

```
z<-structure(list(),class="c")
```

```
>
```

```
> f(z)
```

```
[1] "This is the default function"
```

```
>
```


Challenge 10.2

- Find a way to “override” the print function for a vector object so that it prints a summary of the vector when it is called (using the summary() function).

```
> print  
function (x, ...)  
UseMethod("print")  
<bytecode: 0x1063940d8>  
<environment: namespace:base>  
>
```

Inheritance

- The idea of inheritance is to form new classes of specialised versions of existing ones.

```
> z<-structure(list(),class=c("b","a"))
```

```
>
```

```
> z
```

```
list()
```

```
attr(,"class")
```

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

↑
Class

↑
Superclass

S3 Inheritance:

(1) Define two generic functions

```
f<-function(x){  
  UseMethod("f")  
}
```

```
g <-function(x){  
  UseMethod("g")  
}
```

S3 Inheritance:

(2) Create methods for class a and b

```
f.a<-function(x){  
  print("function f.a")  
}
```

```
f.b<-function(x){  
  print("function f.b")  
}
```

```
g.a<-function(x){  
  print("function g.a")  
}
```

S3 Inheritance:

(3) Create object of b, inherit from a

```
>  
> z<-structure(list(),class=c("b","a"))  
>  
> class(z)  
[1] "b" "a"  
>  
> f(z)  
[1] "function f.b"  
>  
> g(z)  
[1] "function g.a"
```

Challenge 10.3

- Write a new class called “df1” which inherits from “data.frame”
- Create a print function for “df1” which displays the current time, before calling the standard print methods for a data.frame class
- Test the results as follows with the mtcars data frame

```
>
```

```
> d[1:2,]
```

```
[1] "2016-10-02 16:10:40 BST"
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21	6	160	110	3.9	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21	6	160	110	3.9	2.875	17.02	0	1	4	4

Challenge 10.4

- Implement an S3 version of an account object
- Use generic methods for debit and credit

