Chapter 12: Base Types

Advanced R Book Group, Cohort 3

Megan Stodel (@MeganStodel, www.meganstodel.com)

10 November, 2020

But first...

Object-oriented programming (OOP)

In R, functional programming is much more important than OOP. This is because problems are decomposed into simple *functions* rather than simple *objects*.

OOP can be more challenging in R:

- There are multiple OOP systems (3 are explored in this book).
- The different systems are not objectively set in relative importance so different communities use different systems.
- S3 and S4 use generic function OOP rather than encapsualted OOP, which is more common in other langauges, so it is hard to transfer existing OOP skills to R.

The three systems

- *S3:* Used throughout base R. Functions can return rich results with user-friendly display and programmer-friendly internals. R's first OOP system and relies on common conventions rather than ironclad guarantees.
- *R6:* A way to escape copy-on-modify semantics; important if you model objects that exist independently of R - for e.g. data that comes from a web API. An implementation of encapsulated OOP.
- *S4:* A rigorous system that requires careful design consideration, so suited for large systems that might have multiple contributions. Offers more guarantees and greater encapsulation than S3, but is more work upfront.

OOP glossary

- *polymorphism* - a developer can consider a function's interface separately from its implementation, so the same function form can be used for different types of output.
- *encapsulation* - the user doesn't need to worry about an object's details because they are 'encapsulated' behind a standard interface.
- *class* - the type of an object.
- *method* - a implementation for a specific class.
- *fields* - defined by the class, this is the data possessed by all instances of that class.
- *method dispatch* - the process of finding the correct method for a given class.

But what does that mean?

An example

Polymorphism allows `summary()` to produce different outputs for numeric and factor variables.

```
diamonds <- ggplot2::diamonds
summary(diamonds$carat)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2000  0.4000  0.7000  0.7979  1.0400  5.0100
```

```
summary(diamonds$cut)
```

```
##      Fair      Good Very Good   Premium     Ideal
##      1610      4906     12082     13791     21551
```

This is more flexible than an `if-else` statement series because any developer can extend the interface.

The two paradigms

Encapsulated OOP has methods that belong to objects or classes; the object encapsulates both data (with fields) and behaviour (with methods).

Functional OOP has methods that belong to generic functions. From the outside it looks like a regular function call, and internally the components are also functions.

Base Types

What is an object?

There are two uses of the word "object".

So far we have talked very generally...

> "Everything that exists in R is an object." - John Chambers

But not everything is object-oriented. This is to do with the organic evolution of S, because base objects come from S when they were developed before it was considered to need an OOP system. So we distinguish here between *Base objects* and *OO objects*.

What kind of object is it?

Use `is.object()` (TRUE only if OO object) or `sloop::otype()` (returns object type). The technical difference is whether the object has a 'class' attribute, so you can also test with `attr()`.

```
num_range <- 1:10
is.object(num_range)
```

```
## [1] FALSE
```

```
sloop::otype(num_range)
```

```
## [1] "base"
```

```
attr(num_range, 'class')
```

```
## NULL
```

What kind of object is it?

```
is.object(diamonds)
```

```
## [1] TRUE
```

```
sloop::otype(diamonds)
```

```
## [1] "S3"
```

```
attr(diamonds, 'class')
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

What kind of object is it?

`sloop::s3_class()` is aso useful; it returns the implicit class used to pick the methods by S3 and S4 systems (which gets missed by `class()`).

```
x <- matrix(1:4, nrow = 2)
class(x)
```

```
## [1] "matrix" "array"
```

```
sloop::s3_class(x)
```

```
## [1] "matrix"  "integer" "numeric"
```

So what is a base type?

Every object has a base type!

```
typeof(1:10)
```

```
## [1] "integer"
```

```
typeof(diamonds)
```

```
## [1] "list"
```

If a function behaves differently for different base types, they are primarily written in C code using switch statements. This means only R-core can create new types and it is a real mission. Therefore new base types are not added often.

(Remember: with OO objects, any developer can specify a new behaviour for a new class.)

Yes, but what are some types?

There are 25 different base types. They can be grouped into...

VECTORS, including types `NULL`, `logical`, `integer`, `double`, `complex`, `character`, `list` and `raw`.

FUNCTIONS, including `closure`, `special` and `builtin`.

ENVIRONMENTS

S4, which is for S4 classes that don't inherit from an existing base type.

LANGUAGE COMPONENTS, including `symbol`, `language` and `pairlist`.

There are also some esoteric and rarely used types that mainly exist because they are important for C code.

Is there a numeric type? (1)

There are three different meanings of numeric type so be careful.

Sometimes used to mean double type - for example `as.numeric()` is the same as `as.double()`.

Is there a numeric type? (2)

In S3 and S4, a shorthand for either integer or double type, used when picking methods.

```
sloop::s3_class(1)
```

```
## [1] "double"  "numeric"
```

```
sloop::s3_class(1L)
```

```
## [1] "integer" "numeric"
```

Is there a numeric type? (3)

It can also mean objects that behave like numbers. Factors have the type `integer` but don't behave like numbers (you wouldn't care about the mean for e.g.).

```
typeof(factor("x"))
```

```
## [1] "integer"
```

```
is.numeric(factor("x"))
```

```
## [1] FALSE
```

Throughout we are using the second meaning - either integer or double.

General discussion

Some possible topics

- Why use OOP?
- Why are there different OOP systems in R?
- What are some ways of finding out if an object is an OO object?