Control Structure & Object-oriented Programming

Zina Ibrahim

Outline

- By the end of this lecture, you should be comfortable with:
  - Control structures
    - Conditional control
    - Explicit and implicit loops
  - R's Basic OO concepts

Control Structure in R

- Unless specified otherwise, R executes statements sequentially.
- However, one can alter the control flow of the R program in several ways, including:
  - Conditional evaluation: execute a different set of statements under different conditions.
  - Looping: repeatedly execute a set of statements until a stopping condition is met.

Conditional Control Using If-else Statements

```
if ( condition1 ) {
        statement1
    } else if ( condition2 ) {
        statement2
    } else if ( condition3 ) {
        statement3
    } else
        statement4
```

Example I

- Building on the elastic band example (assuming only one elastic band):
  - If stretch exceeds twice the length of the elastic band, the band snaps and no distance is traveled (distance = -1).

```
if(stretch >= 2*length) {
      distance <- -1
} else
     distance <- stretch^2 + (1/sqrt(length))
```

Inside the Condition: Logical Operators

Operator	Description
<	less than
<=	less than or equal to
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
!x	Not x
x | y	x OR y
x & y	x AND y
isTRUE(x)	test if x is TRUE

Example II

- If stretch exceeds twice the length of the band, OR
- If the stretch is less than 0.005
  - Then: No distance is traveled.

```
if(stretch >= 2*length || stretch < 0.005) {
      distance <- -1
} else
     distance <- stretch^2 + (1/sqrt(length))
```

Loops with While

- Syntax:

```
while (condition) {
        statements
}
```

R Code

```
i<- 4
while(i>0){
 print(paste("countdown",i))
 i<-i-1
}
```

R Output

```
[1] "countdown: 4"
[1] "countdown: 3"
[1] "countdown: 2"
[1] "countdown: 1"
```

Loops with For

- Syntax:

```
for (name in sequence) {
      statements
}
```

R Code

```
for(i in 4:1){
 print(paste("countdown",i))
}
```

R Output

```
[1] "countdown: 4"
[1] "countdown: 3"
[1] "countdown: 2"
[1] "countdown: 1"
```

A More Useful Example

Assuming many elastic bands (stretch and length are vectors)

```
for(i in 1:length(stretch)){
    if((stretch[i] >= 2*length[i]) ||
                (stretch[i] < 0.005))
    {
        distance[i] <- -1; print(distance[i]);
     } else
     distance[i] <- stretch[i]^2 +
        (1/sqrt(length[i])); print(distance[i]);
 }
```

Implicit Loops in R

- Built-in R functions: apply, sapply, lapply, et...
  - Apply a function to a data structure
  - Example: calculate the means of the rows of a matrix:

```
> m <- cbind(c(1,2),c(3,4))
> m
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> apply(m,1,mean)
[1] 2 3
```

Object-oriented Programming in R

Object-oriented Programming

- **Object-oriented programming**: a way of organising the code.
  - **Class:** A description of a concept; a new type. A class describes:
    - Attributes
    - Methods
  - **Object**: An actualisation of a class.
- R has two different flavours
  - S3: Older, less structured but easy
  - S4: More newer, more structured

S3

- Every R object has an associated **class**

```
> class(1)
 [1] "numeric"

> class('a')
 [1] "character"

> x <- 1:10
> y <- 5 * x + rnorm(length(x), 0, 1)
> fit <- lm(y ~ x)

> class(fit)
> [1] "lm"
```

How are S3 Classes Useful?

- **Generic functions:** class-oriented invocation.
  - Examine the class of the first argument
  - Decides which specific method to dispatch to.

```
> methods(summary)
 [1] summary.aov                summary.aovlist          summary.aspell*
 [4] summary.connection         summary.data.frame        summary.Date
 [7] summary.default            summary.ecdf*             summary.factor
[10] summary.glm                summary.infl              summary.lm
[13] summary.loess*             summary.manova            summary.matrix
[16] summary.mlm                 summary.nls*             summary.packageStatus*
[19] summary.PDF_Dictionary*    summary.PDF_Stream*     summary.POSIXct
[22] summary.POSIXlt            summary.ppr*              summary.prcomp*
[25] summary.princomp*          summary.proc_time         summary.srcfile
[28] summary.srcref             summary.stepfun           summary.stl*
[31] summary.table              summary.tukeysmooth*

   Non-visible functions are asterisked
```

Generic Functions

- How does it work?
  - The generic methods does not perform any computation
  - It merely acts as a router

```
> summary
function (object, ...)
UseMethod("summary")
<bytecode: 0x1782408>
<environment: namespace:base>
```

Or

```
> summary <-
function(object,...){
useMethod("summary")
}
```

Generic Functions

```
> x <- as.factor(rep(c("a","b"),c(7,13)))
> class(x)
[1] "factor"
> summary(x)
a b
7 13
> y <- rnorm(20)
> class(y)
[1] "numeric"
> summary(y)
Min.
1st Qu. Median Mean 3rd Qu. Max.
-1.78300 -0.55330 0.06459 0.01969 0.66960 1.72900
```

Issues with S3

- There is no formal structure to the class definitions:
  - e.g. I can label the value 3 as a glm object

```
> x<- 3
>class(x) <- “glm”
```

```
> y <-3
> summary(y)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       2       2       2       2
> class(y) <-"glm"
> summary(y)
Error: $ operator is invalid for atomic vectors
```

Issues with S3

- S3 generic functions examine the function's first argument for direction
  - What happens if I want to write a summary method which takes two objects, x and y such that:
  - Case 1:
    - X is a lm
    - Y is a numeric vector
  - Case 2:
    - X is an lm
    - Y is a matrix

S4

- Made to address both issues of S3
  - Formal definition of classes
  - Content validation
  - Define methods for combinations of types

S4

```
> setClass("rectangle", representation(length =
"numeric", width = "numeric"))
[1] "rectangle“
> rect <- new("rectangle",length=10,width=5)
> rect
An object of class “rectangle”
Slot "length":
[1] 10

Slot "width":
[1] 5
```

```
> summary(rect)
   Length     Class      Mode
        1 rectangle        S4
>
```

Validation in S4

```
> class(rect)
[1] "rectangle"
attr(,"package")
[1] ".GlobalEnv"
> class(rect)<-"factor"
Error in class(rect) <- "factor" :
  adding class "factor" to an
invalid object
```

Want to Know More?

- http://www.r-bloggers.com/resources-for-s4-classes-and-methods/

R-bloggers

R news and tutorials contributed by (452) R bloggers