R

SP-2014 Lecture 18

Expressions in R

```
1
# 1

1+2
# 3

1*3+(2/4)
# 3.5

2^2
# 4
```

```
pi
# 3.141593

sin(pi)
# 1.224606e-16
```

Variables in R

```
a = 15
a
# 15

a = 3
a
# 3

a = a + 1
a
# 4
```

```
a = 1
a
# 1

a <- 2
a
# 2

3 -> a
a
# 3
```

Control Structures in R

```
print("A"); print("B")
# A
# B

if (1>2) {
   print("GT")
} else {
   print("LT")
}
# LT
```

```
for (i in 1:3) {
   print(i)
}
# 1
# 2
# 3
```

R is Typed

```
class(1)
# "numeric"
class(1.0)
# "numeric"
class(T)
# "logical"
class("Hi")
# "character"
class(class)
# "function"

class(t.test(c(1,2,3)))
# "htest"
class(ecdf(c(1,2,3)))
# "ecdf"     "stepfun"    "function"
```

R is Dynamically Typed

```
class(1+1)
# "numeric"

age = 5
class(age)
# "numeric"
```

```
name = "Barak"
class(name)
# "character"

name = 5
class(name)
# "numeric"

name = sin
class(name)
# "function"
```

"Inheritance" in R

```
class(t.test(c(1,2,3)))
# "htest"

e = ecdf(c(1,2,3))
class(e)
# "ecdf"     "stepfun"    "function"
```

(ecdf is "subclass" of stepfun is "subclass" of function)

```
inherits(e, "ecdf")
# TRUE
inherits(e, "stepfun")
# TRUE
is.ecdf(e)
# Error: could not find function "is.ecdf"
is.stepfun(e)
# TRUE
```

Type-Casts in R

```
as.character(5)
# "5"

as.numeric("9999")
# 9999
```

Polymorhism in R

```
e <- ecdf(c(1,2,3))
class(e)
# "ecdf"    "stepfun"   "function"
plot(e)
# invokes plot.ecdf
e <- c(1,2,3)
class(e)
# "numeric"
plot(e)
# invokes plot.default
```

Looking at source code of plot:

```
getAnywhere(plot)
# ...
# UseMethod("plot")
```

(polymorphic call, based on first arg)

Defining Functions in R

```
sayHi <- function() {
  print("Hello");
  print("world!")
}
sayHi()
# "Hello"
# "world!"

add <- function(a,b) {a+b}
add(1,2)
# 3

alsoAdd <- function(a,b) {a-b;c=a*b;a+b}
alsoAdd(1,2)
# 3
```

Recursive Functions in R

```
fib <- function(n) {
  if (n<2) {
    fn <- 1
  } else {
    fn <- fib(n-1) + fib(n-2)
  }
  fn
}
fib(5)
# 8
```

Higher-Order Functions in R

Functions as return values

```
mySin <- function() {sin}
mySin()
# function (x)  .Primitive("sin")
mySin()(pi)
# 1.224606e-16
```

Functions as arguments

```
apply <- function(f,a) {f(a)}
apply(sin,pi)
# 1.224606e-16
```

Lambda Expressions in R

```
function(x) {x+1}(19)
# 20

add <- function(a) {function(x) {a+x}}
add(5)
# function(x) {a+x}
add(5)(2)
# 7
inc <- add(1)
inc(5)
# 6

neg <- function(x) {-x}
compose <- function(f,g) {function(x) {f(g(x))}}
id <- compose(neg,neg)
id(5)
# 5
```

R Environments

Global objects

```
a = "Hi"
n = 5
ls()
# "a" "n"
```

Local objects

```
function() {x=99;ls()}()
# "x"

f <- function() {ls()}
f()
# character(0)
```

String Operations in R

```
paste("Hi", "Ho")
# "Hi Ho"
paste("Hi", "Ho", sep="")
"HiHo"
nchar("Hello")
# 5
nchar("\t")
# 1
strsplit("Jim:5:99:white", ":")
# "Jim"     "5"   "99"  "white"
```

substr, strtrim, ...

Named Function Arguments in R

```
rep("A", 5)
# "A" "A" "A" "A" "A"
rep("A", times=5)
# "A" "A" "A" "A" "A"
rep(x="A", times=5)
# "A" "A" "A" "A" "A"
rep(times=5, x="A")
# "A" "A" "A" "A" "A"
rep(times=5, "A")
# "A" "A" "A" "A" "A"


rep(5, "A")
# Error in rep(5, "A") : invalid 'times' argument
```

Data Structures in R

- Vectors
- Factors

- Lists
- Data Frames

- Arrays
- Matrices

Vectors in R

```
x = c(10,20,30,40,50)
x
# 10 20 30 40 50
length(x)
# 5
x[1]
# 10
x[5]
# 50
v = c()
v
# NULL
```

```
class(x)
# "numeric"

class(c(T,F,TRUE,FALSE))
# "logical"

class(c(1,T))
# "numeric"

class(c(T,"1"))
# "character"
```

Creating Vectors in R

No nesting

```
y = c(1, c(2, 3), 4, 5)
y
# 1 2 3 4 5

vector("logical", 3)
# FALSE FALSE FALSE

rep(T, 3)
# TRUE TRUE TRUE
```

```
1:5
# 1 2 3 4 5

seq(0,6,2)
# 0 2 4 6
```

Scalars are Vectors in R

```
1 == c(1)
# [1] TRUE

1 == c(1,2)
# [1] TRUE FALSE

1 == c(1,1)
# [1] TRUE TRUE

c(1,3) == c(1,2,1)
# [1] TRUE FALSE TRUE

c() == 1
# logical(0)
```

```
class(1)
# "numeric"

class(c(1))
# "numeric"

class(c(1,1))
# "numeric"

length(99)
# [1] 1
```

Operations on Vectors in R

Element-wise operations

```
a = c(1,4,1)
b = c(3,2,2)
c = a * b + 1
c
# [1] 4 9 3

lt = a<b
lt
# [1] T F T
```

Updating

```
a = c(1,2)
a
# [1] 1 2

a[1] = 3
a
# [1] 3 2
```

Factors & Levels in R

```
a = c("s","m","xxl","s")
class(a)
# [1] "character"

attributes(a)
# NULL

fa = factor(a)
class(fa)
# [1] "factor"
```

```
attributes(fa)
# $levels
# [1] "m" "s" "xxl"
# $class
# [1] "factor"

attributes(fa)$levels
# [1] "m" "s" "xxl"

levels(fa)
# [1] "m" "s" "xxl"
```

Factors of Other Types

```
a = c(1,2,55,2,1,1,1,55)
class(a)
# [1] "numeric"

fa = factor(a)
class(fa)
# [1] "factor"

levels(fa)
# [1] 1 2 55
```

```
a = c(T,F,TRUE,T,F,F,F)
class(a)
# [1] "logical"

fa = factor(a)
class(fa)
# [1] "factor"

levels(fa)
# [1] FALSE TRUE
```

Factors are Glorified Vectors

```
fa = factor(c(1,2,55,2))
levels(fa)
# [1] 1 2 55

length(fa)
# [1] 4

fa[3]
# [1] 55
# Levels: 1 2 55
```

```
fa[2] = 1
fa[2]
# 1
# Levels: 1 2 55

fa[3] = 99
# WARNING

fa[3]
# <NA>
# Levels: 1 2 55
```

Indexing of Vectors or Factors

```
a = c(1,2,55,2,19)

a[3]
# [1] 55

a[1:3]
# [1] 1 2 55

a[-2]
# [1] 1 55 2 19
```

```
a[c(F,T,T,F,T)]
# [1] 2 55 19

a[a>18]
# [1] 55 19


b = c(1, NA, 5, NA, 7)
b[!is.na(b)]
# [1] 1 5 7
```

Lists in R

```
l = list("gcc", "-O3",
   99, T, c(10,20))
l
# [[1]]
# [1] "gcc"
# [[2]]
# [1] "-O3"
# [[3]]
# [1] 99
# [[4]]
# [1] TRUE
# [[5]]
# [1] 10 20
```

```
class(l)
# [1] "list"

length(l)
# [1] 5

l[[1]]
# [1] "gcc"

l[[5]]
# [1] 10 20

l[[5]][2]
# [1] 20
```

Lists are Structs

```
l = list(
   cpu="p4",
   mem=4,
   disks=c(500,1000,200))
l
# $cpu
# [1] "p4"
# $mem
# [1] 4
# $disks
# [1] 500 1000 200
```

```
length(l)
# [1] 3

l[[1]]
# [1] "p4"

l[["cpu"]]
# [1] "p4"

l$cpu
# [1] "p4"
```

Lists are Tables

```
l = list(
   cpu = c("p4","c2"),
   mem = c(4,2),
   os  = c("win","osx"))
l
# $cpu
# [1] "p4" "c2"
# $mem
# [1] 4 2
# $os
# [1] "win" "osx"
```

```
length(l)
# [1] 3

l$cpu
# [1] "p4" "c2"

l$cpu[2]
# [1] "c2"

length(l$cpu)
# [1] 2
```

Combining Lists: Fields

```
l1 = list(
   cpu = c("p4","c2"),
   mem = c(4,2),
   os  = c("win","osx"))

l2 = list(
   cd  = c(T,F,F)
)

l3 = c(l1, l2)
```

```
l3
# $cpu
# [1] "p4" "c2"
# $mem
# [1] 4 2
# $os
# [1] "win" "osx"
# $cd
# [1] TRUE FALSE FALSE
```

Data Frames (Glorified Lists)

```
df = data.frame(
   cpu = c("p4","c2"),
   mem = c(4,2),
   os  = c("win","osx"))
df
#    cpu mem  os
# 1  p4    4 win
# 2  c2    2 osx
```

```
class(df)
# [1] "data.frame"

class(df$mem)
# [1] "numeric"

class(df$cpu)
# [1] "factor"

df$mem = c(4,2,4)
# Error (3 rows)
```

Combining Data Frames: Rows

```
df1 = data.frame(
   cpu = c("p4","c2"),
   mem = c(4,2),
   os  = c("win","osx")
)

df2 = data.frame(
   cpu = c("ci7"),
   mem = c(8),
   os  = c("lin")
)

df3 = rbind(df1, df2)
```

```
df3
#    cpu mem  os
# 1   p4   4 win
# 2   c2   2 osx
# 3 ci7   8 lin
```

Combining Data Frames: Fields

```
df1 = data.frame(
   cpu = c("p4","c2"),
   mem = c(4,2),
   os  = c("win","osx"))

df2 = data.frame(
   cd  = c(T,F)
)

df3 = cbind(df1, df2)
```

```
df3
#   cpu mem  os    cd
# 1  p4   4 win  TRUE
# 2  c2   2 osx FALSE
```

Projection & Selection

```
df = data.frame(
   cpu = c("p4","c2"),
   mem = c(4,2),
   os  = c("win","osx"))
df
#    cpu mem  os
# 1  p4    4 win
# 2  c2    2 osx
```

```
data.frame(
   cpu=df$cpu,
   os =df$os)
#    cpu  os
# 1  p4 win
# 2  c2 osx
data.frame(
   cpu=df$cpu[df$mem==2],
   mem=df$mem[df$mem==2],
   os =df$os [df$mem==2])
#    cpu mem  os
# 1  c2    2 osx
```

Reshaping Data Frames

- Use the **reshape** package: http://had.co.nz/reshape/introduction.pdf

Data Frames from Files

```
cpu mem os
p4 4 win
c2 2 osx
c2 4 osx
```

```
df = read.table(
   measurements.txt,
   header=T)
df
#    cpu mem  os
# 1   p4    4 win
# 2   c2    2 osx
# 3   c2    4 osx
```

Other Data Structures in R

- Arrays (multi-dimensional)
- Matrices (a special kind of array)

- See: http://cran.r-project.org/doc/manuals/R-intro.html#Arrays-and-matrices

Using R

Interactive

```
$ R --vanilla
> cat("Hi\n")
Hi
> q()
$
```

Batch

```
$ cat hi.R
cat("Hi\n")
$ R --slave < hi.R
Hi
$

$ echo 'cat("Hi\n")'
   | R --slave
Hi
$
```

Interactive R: Saving Sessions

```
$ mkdir work
$ cd work
$ R
> min = function(x,y) {if (x<y) x else y}
> min(2,1)
[1] 1
> q()
$ ls -a
./          ../         .RData      .Rhistory
$ cat .Rhistory
min = function(x,y) {if (x<y) x else y}
min(2,1)
q()
$ R
[previously saved workspace restored]
>
```

Interactive R: Source & Sink

```
$ R
> source("hi.R")
Hi
> sink("out.txt")
> 1+1
> source("hi.R")
> sink()
> 1+1
[1] 2
> source("hi.R")
Hi
> q()
$ cat out.txt
[1] 2
Hi
```