R-programming-2.R

aeoluseros

Fri Mar 06 19:42:13 2015

```
#1. control structures
#if condition:
x<-10
if(x==5){
        y<-10
}else if(x>6){
        y<-0
}else{
        y<-5
}
# we don't have to use 'else'.
if(x==10){
        y<-11
}
if(x>10){
        y<-9
}

#for,while,repeat -- three kinds of loops
#control structures mentioned here are primarily useful for writing programs;
#for command-line interactive work, the *apply functions are more useful;
#for loop
for(i in 1:10){
        print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

```
#three different ways to use for loop
x<-c("a","b","c","d") # same as: x<-c('a','b','c','d')
for(i in 1:4){
        print(x[i])
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
for(i in length(x)){
        print(x[i])
}
```

```
## [1] "d"
```

```
for(letter in x){
        print(letter)
}
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
for(i in 1:4) print(x[i])  #if for loop only has single expression, we could remove the curly braces.
```

```
## [1] "a"
## [1] "b"
## [1] "c"
## [1] "d"
```

```
#while loop
count<-0
while(count<10){
        print(count)
        count<-count+1
}
```

```
## [1] 0
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
```

```
#could have more than one condition with while loop
z<-5
while(z>=3&&z<=10){   #conditions are alwys evaluated from left to right
        print(z)
        coin<-rbinom(1,1,0.5)
        if(coin==1){
                z<-z+1
        }else{
                z<-z-1
        }
}
```

```
## [1] 5
## [1] 6
## [1] 5
## [1] 6
## [1] 5
## [1] 4
## [1] 3
## [1] 4
## [1] 3
```

```
#repeat infinite loop + break(the only way to exit a repeat)
x0<-1
tol<-1e-8
repeat{
        x1<-rbinom(1,1,0.5)
        print(x1)
        if(abs(x1-x0)<tol){
                break
        } else{
                x0<-x1
        }
}
```

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

```
#next is used to skip an iteration of a loop
for(i in 1:100){
        if(i<=20)
                ##skip the first 20 iterations
                next
        print(i)
}
```

```
## [1] 21
## [1] 22
## [1] 23
## [1] 24
## [1] 25
## [1] 26
## [1] 27
## [1] 28
## [1] 29
## [1] 30
## [1] 31
## [1] 32
## [1] 33
## [1] 34
## [1] 35
## [1] 36
## [1] 37
## [1] 38
## [1] 39
## [1] 40
## [1] 41
## [1] 42
## [1] 43
## [1] 44
## [1] 45
## [1] 46
## [1] 47
## [1] 48
## [1] 49
## [1] 50
## [1] 51
## [1] 52
## [1] 53
## [1] 54
## [1] 55
## [1] 56
## [1] 57
## [1] 58
## [1] 59
## [1] 60
## [1] 61
## [1] 62
## [1] 63
## [1] 64
## [1] 65
## [1] 66
## [1] 67
## [1] 68
## [1] 69
## [1] 70
## [1] 71
## [1] 72
## [1] 73
## [1] 74
## [1] 75
## [1] 76
## [1] 77
## [1] 78
## [1] 79
## [1] 80
## [1] 81
## [1] 82
## [1] 83
## [1] 84
## [1] 85
## [1] 86
## [1] 87
## [1] 88
## [1] 89
## [1] 90
## [1] 91
## [1] 92
## [1] 93
## [1] 94
## [1] 95
## [1] 96
## [1] 97
## [1] 98
## [1] 99
## [1] 100
```

```
#"return(value)" signals that a function/loop should exit and return a given value

2. #####writing functions#####
```

```
## [1] 2
```

```
add2<-function(x,y){
        x+y
}
add2(3,5)
```

```
## [1] 8
```

```
above10<-function(x){
        use <- x>10
        x[use]  #subset x
}

above<-function(x,c=3){
        use<-x>c
        x[use]
}
x<-1:12
above(x,10)
```

```
## [1] 11 12
```

```
above(x)  #default critical value is 3
```

```
## [1]  4  5  6  7  8  9 10 11 12
```

```
columnmean<-function(y,removeNA=TRUE){
        nc<-ncol(y)  #number of columns
        means<-numeric(nc)  #empty vector with all zeros
        for(i in 1:nc){
                means[i]<-mean(y[,i],na.rm=removeNA)
        }
        means
}
columnmeans <- function(y) sapply(y[complete.cases(y),],mean)  #same function
columnmean(airquality)
```

```
## [1]  42.129310 185.931507   9.957516  77.882353   6.993464  15.803922
```

```
columnmeans(airquality)
```

```
##      Ozone    Solar.R       Wind       Temp      Month        Day
##  42.099099 184.801802   9.939640  77.792793   7.216216  15.945946
```

```
#3.function arguments
#functions could be passed as arguments to other functions
#The return value of a function is the last expression in the function body to be evaluated.
#so there is no special expression for returning something for a function, although there is a function
formals(file)  #formals() function returns a list of all the formal arguments of a function
```

```
## $description
## [1] ""
##
## $open
## [1] ""
##
## $blocking
## [1] TRUE
##
## $encoding
## getOption("encoding")
##
## $raw
## [1] FALSE
```

```
args(lm)
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## NULL
```

```
#argument matching can also be partially matched.

#4.arguments are valuated lazily
f<-function(a,b){
        a^2
}
f(2)  #the function doesn't use b, so we don't have to specify b
```

```
## [1] 4
```

```
f<-function(a,b){
        print(a)
        print(b)
}
#f(45)  # the value of a could still be printed, but the second line would commit error

#5. the "..." argument
```

```r
#... is used when extending another function and you don't want to copy the entire argument list of the
myplot<-function(x,y,type="l",...){
        plot(x,y,type=type,...)
}
#... argument is also necessary when the number of arguments passed to the function cannot be known in
args(paste)  #paste function is used to concatenate strings together and returns a character variable
```

```
## function (..., sep = " ", collapse = NULL)
## NULL
```

```r
args(cat) #cat will not return anything, it will just output to the console or another connection.
```

```
## function (..., file = "", sep = " ", fill = FALSE, labels = NULL,
##     append = FALSE)
## NULL
```

```r
c<-paste("a","b",sep=":")
#any arguments that appear after ... on the argument list must be named explicitly and cannot be partia
paste("a","b",se=":")  #partial matching cannot be partially matched
```

```
## [1] "a b :"
```

```r
d<-cat("a","b",sep=":")  # d couldn't be assigned a value because cat() is just used to print out.
```

```
## a:b
```

```r
print(paste("a","b",sep=":"))
```

```
## [1] "a:b"
```

```r
#6. Symbol binding -- how does R know which value to assign to which symble?
lm<-function(x) {x*x}
lm  #it won't give the value of lm that is in the "stats" package
```

```
## function(x) {x*x}
```

```r
#R uses lexical scoping or static scoping (equivalent concepts)
search()  # the search list when R tries to find a value
```

```
## [1] ".GlobalEnv"        "package:stats"     "package:graphics"
## [4] "package:grDevices" "package:utils"     "package:datasets"
## [7] "package:methods"   "Autoloads"         "package:base"
```

```r
#lm is deined in Global Environment, so when I  that object would be found first
rm(lm)
lm
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## {
##     ret.x <- x
##     ret.y <- y
##     cl <- match.call()
##     mf <- match.call(expand.dots = FALSE)
##     m <- match(c("formula", "data", "subset", "weights", "na.action",
##         "offset"), names(mf), 0L)
##     mf <- mf[c(1L, m)]
##     mf$drop.unused.levels <- TRUE
##     mf[[1L]] <- quote(stats::model.frame)
##     mf <- eval(mf, parent.frame())
##     if (method == "model.frame")
##         return(mf)
##     else if (method != "qr")
##         warning(gettextf("method = '%s' is not supported. Using 'qr'",
##             method), domain = NA)
##     mt <- attr(mf, "terms")
##     y <- model.response(mf, "numeric")
##     w <- as.vector(model.weights(mf))
##     if (!is.null(w) && !is.numeric(w))
##         stop("'weights' must be a numeric vector")
##     offset <- as.vector(model.offset(mf))
##     if (!is.null(offset)) {
##         if (length(offset) != NROW(y))
##             stop(gettextf("number of offsets is %d, should equal %d (number of observations)",
##                 length(offset), NROW(y)), domain = NA)
##     }
##     if (is.empty.model(mt)) {
##         x <- NULL
##         z <- list(coefficients = if (is.matrix(y)) matrix(, 0,
##             3) else numeric(), residuals = y, fitted.values = 0 *
##             y, weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w !=
##             0) else if (is.matrix(y)) nrow(y) else length(y))
##         if (!is.null(offset)) {
##             z$fitted.values <- offset
##             z$residuals <- y - offset
##         }
##     }
##     else {
##         x <- model.matrix(mt, mf, contrasts)
##         z <- if (is.null(w))
##             lm.fit(x, y, offset = offset, singular.ok = singular.ok,
##                 ...)
##         else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
##             ...)
##     }
##     class(z) <- c(if (is.matrix(y)) "mlm", "lm")
##     z$na.action <- attr(mf, "na.action")
##     z$offset <- offset
##     z$contrasts <- attr(x, "contrasts")
##     z$xlevels <- .getXlevels(mt, mf)
##     z$call <- cl
##     z$terms <- mt
##     if (model)
##         z$model <- mf
##     if (ret.x)
##         z$x <- x
##     if (ret.y)
##         z$y <- y
##     if (!qr)
##         z$qr <- NULL
##     z
## }
## <bytecode: 0x00000000076f6978>
## <environment: namespace:stats>
```

```
stats::lm
```

```
## function (formula, data, subset, weights, na.action, method = "qr",
##     model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
##     contrasts = NULL, offset, ...)
## {
##     ret.x <- x
##     ret.y <- y
##     cl <- match.call()
##     mf <- match.call(expand.dots = FALSE)
##     m <- match(c("formula", "data", "subset", "weights", "na.action",
##         "offset"), names(mf), 0L)
##     mf <- mf[c(1L, m)]
##     mf$drop.unused.levels <- TRUE
##     mf[[1L]] <- quote(stats::model.frame)
##     mf <- eval(mf, parent.frame())
##     if (method == "model.frame")
##         return(mf)
##     else if (method != "qr")
##         warning(gettextf("method = '%s' is not supported. Using 'qr'",
##             method), domain = NA)
##     mt <- attr(mf, "terms")
##     y <- model.response(mf, "numeric")
##     w <- as.vector(model.weights(mf))
##     if (!is.null(w) && !is.numeric(w))
##         stop("'weights' must be a numeric vector")
##     offset <- as.vector(model.offset(mf))
##     if (!is.null(offset)) {
##         if (length(offset) != NROW(y))
##             stop(gettextf("number of offsets is %d, should equal %d (number of observations)",
##                 length(offset), NROW(y)), domain = NA)
##     }
##     if (is.empty.model(mt)) {
##         x <- NULL
##         z <- list(coefficients = if (is.matrix(y)) matrix(, 0,
##             3) else numeric(), residuals = y, fitted.values = 0 *
##             y, weights = w, rank = 0L, df.residual = if (!is.null(w)) sum(w !=
##             0) else if (is.matrix(y)) nrow(y) else length(y))
##         if (!is.null(offset)) {
```

```
##             z$fitted.values <- offset
##             z$residuals <- y - offset
##         }
##     }
##     else {
##         x <- model.matrix(mt, mf, contrasts)
##         z <- if (is.null(w))
##             lm.fit(x, y, offset = offset, singular.ok = singular.ok,
##                 ...)
##         else lm.wfit(x, y, w, offset = offset, singular.ok = singular.ok,
##             ...)
##     }
##     class(z) <- c(if (is.matrix(y)) "mlm", "lm")
##     z$na.action <- attr(mf, "na.action")
##     z$offset <- offset
##     z$contrasts <- attr(x, "contrasts")
##     z$xlevels <- .getXlevels(mt, mf)
##     z$call <- cl
##     z$terms <- mt
##     if (model)
##         z$model <- mf
##     if (ret.x)
##         z$x <- x
##     if (ret.y)
##         z$y <- y
##     if (!qr)
##         z$qr <- NULL
##     z
## }
## <bytecode: 0x00000000076f6978>
## <environment: namespace:stats>
```

```
#when a package is loaded, it would be put in position 2 of the search list.
#R has separate namespaces for functions and non-functions so it's possible to have an object named c a
#free variables:
#free variables are not formal arguments and are not local variables.
f<-function(x,y){
        x^2+y/z
}
rm(z)
#f(2,3)
z<-2
f(2,3)  #scoping rules of a language determine how values are assigned to free variables.
```

```
## [1] 5.5
```

```
#define a function inside another function (not allowed in some languages such as C):
make.power<-function(n){
        pow<-function(x){
                x^n
        }
        pow
}
```

```
cube<-make.power(3)
square<-make.power(2)
cube(3)
```

```
## [1] 27
```

```
square(5)
```

```
## [1] 25
```

```
ls(environment(cube)) #"ls" and "objects" return a vector of character strings giving the names of the
```

```
## [1] "n"   "pow"
```

```
objects(environment(cube))
```

```
## [1] "n"   "pow"
```

```
get("n",environment(cube)) #search an object in an environment
```

```
## [1] 3
```

```
get("n",environment(square))  #cube and square both functions have different environments
```

```
## [1] 2
```

```
y<-10
f<-function(x){ #y and g are both free variables
        y<-2
        y^2+g(x)
}
g<-function(x){
        x*y
}
f(3) #with lexical scoping, the value of Y and the function g is loked up in the environment
```

```
## [1] 34
```

```
#in which the function is defined, which in this case is the global environment.
#So the value of y in function g is 10. so 2^2 +3*10.
#when you looking for a free variable in funtion g, you will look up global environment first.
#other languages also support lexical scoping: Scheme, Python, Perl, Common Lisp
#in SPLUS, free variables are always looked up in the global workspace, so everything can be
#stored on the disk because the "defining environment" of all functions is the same.

#7. Application: Optimization
#optim, nlm, optimize -- used in MLE(minimize, maximize)
make.NegLogLik<-function(data,fixed=c(FALSE,FALSE)){
        params<-fixed    #parameters
        function(p){
                params[!fixed]<-p  #the unfixed parameter would be assigned to be p. p should be a two-
                mu<-params[1]
                sigma<-params[2]
                a<--0.5*length(data)*log(2*pi*sigma^2)
                b<--0.5*sum((data-mu)^2)/(sigma^2)
                -(a+b)
        }
}
set.seed(1);
normals<-rnorm(100,1,2)
nLL<-make.NegLogLik(normals)
ls(environment(nLL))  #return the objects in the environment of the nLL function.
```

```
## [1] "data"   "fixed"  "params"
```

```
args(optim)
```

```
## function (par, fn, gr = NULL, ..., method = c("Nelder-Mead",
##     "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"), lower = -Inf,
##     upper = Inf, control = list(), hessian = FALSE)
## NULL
```

```
optim(c(mu=0,sigma=1),nLL)$par   #initial guess of params: p=c(mu=0,sigma=1)
```

```
##       mu    sigma
## 1.218239 1.787343
```

```
formals(optim)
```

```
## $par
##
##
## $fn
##
##
## $gr
## NULL
##
## $...
##
##
## $method
## c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent")
##
## $lower
## -Inf
##
## $upper
## [1] Inf
```

```
##
## $control
## list()
##
## $hessian
## [1] FALSE
```

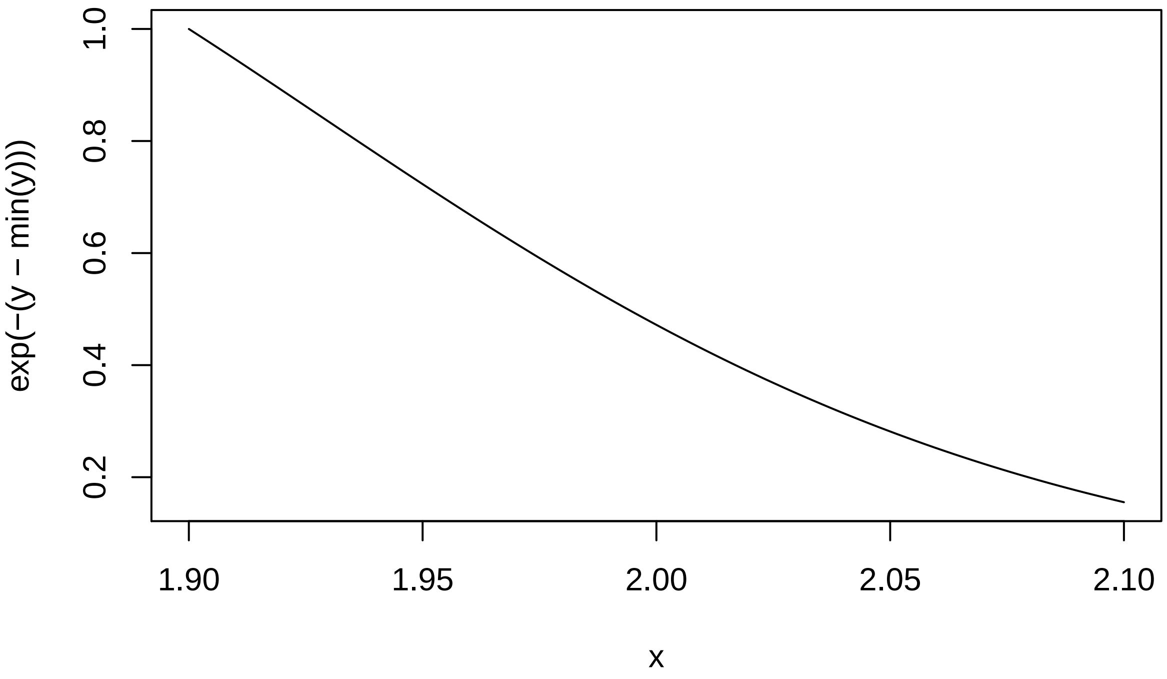
```
nLL<-make.NegLogLik(normals,c(FALSE,2)) #fixing sigma = 2
optimize(nLL,c(-1,3))$minimum   #optimize is used for single variable only.
```

```
## [1] 1.217775
```

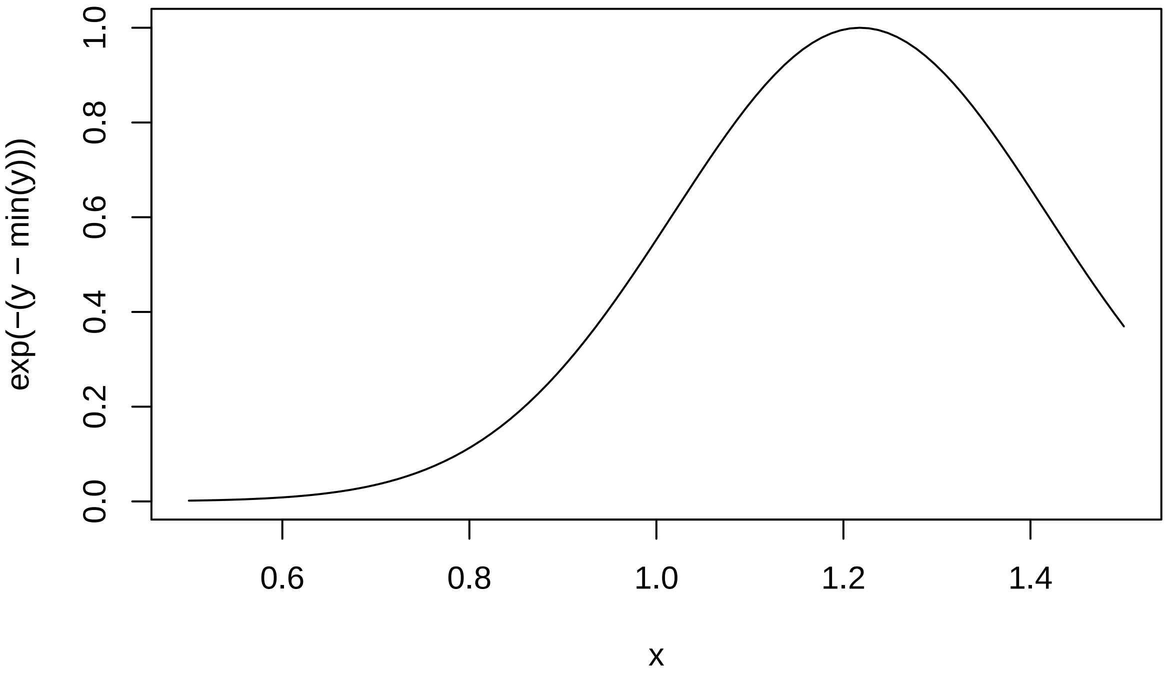
```
nLL<-make.NegLogLik(normals,c(1,FALSE))
optimize(nLL,c(1e-6,10))$minimum  #c(1e-6,10) is an interval
```

```
## [1] 1.800596
```

```
#plot likelihood
nLL<-make.NegLogLik(normals,c(1,FALSE))
x<-seq(1.9,2.1,len=100)
y<-sapply(x,nLL)
plot(x,exp(-(y-min(y))),type="l")  #if normals have more value, the plot would be sharper.
```



```
nLL<-make.NegLogLik(normals,c(FALSE,2))
x<-seq(0.5,1.5,len=100)
y<-sapply(x,nLL)
plot(x,exp(-(y-min(y))),type="l")
```



```
#suggestion: limit the  size of a function. each function only does one thing.
#one function is no more than one page.


#8. date and times in R
#Class of date: Date (store as the number of days since 1970-01-01)
#class of Time: POSIXct or POSIXlt (store as the number of seconds since 1970-01-01)
#in POSIXct class, times are represented at just as very large integers. It's a useful
#                   type of class if you want to store times in a data frame or something
#                   like because it's basically a big integer vector.
#in POSIXlt class stores a time as a list, so there is a bunch of other useful information
#                   about a given time, for example what's the day of the week of that time,
#                   what's the day of the years, the day of the week, the day of the month,
#                   or the month itself
#three functions: weekdays(give the day of the week), months(give the month name),
#                  quarters(give the quarter number: "Q1","Q2","Q3","Q4)
Sys.time()
```

```
## [1] "2015-03-06 19:42:13 EST"
```

```
x<-as.Date("1970-1-1")
x
```

```
## [1] "1970-01-01"
```

```
class(x)
```

```
## [1] "Date"
```

```
unclass(x) #returns 0
```

```
## [1] 0
```

```
class(unclass(x)) #numeric
```

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

```
unclass(as.Date("1970-01-02"))
```

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

```
x<-as.Date("1970/1/1")
x<-as.Date("1/1/1970") #wrong format
p<-as.POSIXlt(Sys.time(), "GMT")
unclass(p)
```

```
## $sec
## [1] 13.58944
##
## $min
## [1] 42
##
## $hour
## [1] 0
##
## $mday
## [1] 7
##
## $mon
## [1] 2
##
## $year
## [1] 115
##
## $wday
## [1] 6
##
## $yday
## [1] 65
##
```

```
## $isdst
## [1] 0
##
## attr(,"tzone")
## [1] "GMT"
```

```
names(unclass(p))
```

```
## [1] "sec"   "min"   "hour"  "mday"  "mon"   "year"  "wday"  "yday"  "isdst"
```

```
p$sec
```

```
## [1] 13.58944
```

```
p$yday
```

```
## [1] 65
```

```
p$isdst #Daylight Saving Time flag. Positive if in force, zero if not, negative if unknown.
```

```
## [1] 0
```

```
q<-as.POSIXct(Sys.time(),"EST")
unclass(q) #a large integer number
```

```
## [1] 1425688934
```

```
names(unclass(q))  # NULL
```

```
## NULL
```

```
#strptime function
datestring<-c("January 10,2012 10:40","December 9, 2011 9:10")
x<-strptime(datestring,"%B %d, %Y %H:%M")
x
```

```
## [1] "2012-01-10 10:40:00 EST" "2011-12-09 09:10:00 EST"
```

```
datestring<-c("Jan 10,2012 10:40","Dec 9, 2011 9:10")
x<-strptime(datestring,"%B %d, %Y %H:%M")
x
```

```
## [1] "2012-01-10 10:40:00 EST" "2011-12-09 09:10:00 EST"
```

```
class(x)
```

```
## [1] "POSIXlt" "POSIXt"
```