Conditionals, Loops & Functions

Ni

Conditionals and Control flow

Relational operators

- Equality ==
- Greater and less than <, >, <=, >=
- Compare vectors
- Compare matrices

```r
# Comparison of logicals
TRUE==FALSE
```

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

```r
# Comparison of numerics
-6*14 != 17-101
```

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

```r
# Comparison of character strings
"useR" == "user"
```

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

```r
# Compare a logical with a numeric
TRUE==1
```

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

```r
# The linkedin and facebook vectors have already been created for you
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
facebook <- c(17, 7, 5, 16, 8, 13, 14)
# Popular days
linkedin>15
```

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

```r
# Quiet days
linkedin<=5
```

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

```r
# LinkedIn more popular than Facebook
linkedin>facebook
```

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

```r
# The social data has been created for you
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
facebook <- c(17, 7, 5, 16, 8, 13, 14)
views <- matrix(c(linkedin, facebook), nrow = 2, byrow = TRUE)
# When does views equal 13?
views==13
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]
## [1,] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
## [2,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

```r
# When is views less than or equal to 14?
views<=14
```

```
##       [,1] [,2] [,3]  [,4] [,5]  [,6] [,7]
## [1,] FALSE TRUE TRUE  TRUE TRUE FALSE TRUE
## [2,] FALSE TRUE TRUE FALSE TRUE  TRUE TRUE
```

Logical operators

- And &
- or |

```r
# The linkedin and last variable are already defined for you
linkedin <- c(16, 9, 13, 5, 2, 17, 14)
last <- tail(linkedin, 1)
# Is last under 5 or above 10?
last<5 | last>10
```

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

```r
# Is last between 15 (exclusive) and 20 (inclusive)?
last > 15 & last <= 20
```

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

Conditional Statements

- The if and else statements
- The else if statement

Loops

- while loop
- for loop

```r
# Initialize i as 1
i <- 1
# Code the while loop
while (i <= 10) {
  print(3*i)
  # if the triple i is divisible by 8
  if ((3 * i) %% 8 == 0) {
    break
  }
  i <- i + 1
}
```

```
## [1] 3
## [1] 6
## [1] 9
## [1] 12
## [1] 15
## [1] 18
## [1] 21
## [1] 24
```

```r
cities<-list("New York","Paris","London",
         "Tokyo","Cape Town","Beijing")
for(city in cities){
  print(city)
}
```

```
## [1] "New York"
## [1] "Paris"
## [1] "London"
## [1] "Tokyo"
## [1] "Cape Town"
## [1] "Beijing"
```

```r
# break statement
cities<-list("New York","Paris","London",
         "Tokyo","Cape Town","Beijing")
for(city in cities){
  if(nchar(city)==5){
    break
  }
  print(city)
}
```

```
## [1] "New York"
```

```r
# Next statement
cities<-list("New York","Paris","London",
         "Tokyo","Cape Town","Beijing")
for(city in cities){
  if(nchar(city)==5){
    next  # skip to next iteration
  }
  print(city)
}
```

```
## [1] "New York"
## [1] "London"
## [1] "Cape Town"
## [1] "Beijing"
```

```r
# Subset the vector explicitly
cities<-list("New York","Paris","London",
         "Tokyo","Cape Town","Beijing")
for(i in 1:length(cities)){
  print(paste(cities[i]," is in position",i," of the cities vector."))
}
```

```
## [1] "New York  is in position 1  of the cities vector."
## [1] "Paris  is in position 2  of the cities vector."
## [1] "London  is in position 3  of the cities vector."
## [1] "Tokyo  is in position 4  of the cities vector."
## [1] "Cape Town  is in position 5  of the cities vector."
## [1] "Beijing  is in position 6  of the cities vector."
```

```r
#paste() use to concatenate strings
```

```r
# The nyc list is already specified
nyc <- list(pop = 8405837,
            boroughs = c("Manhattan", "Bronx", "Brooklyn", "Queens", "Staten Island"),
            capital = FALSE)
# Loop version 1
for(i in nyc){
  print(i)
}
```

```
## [1] 8405837
## [1] "Manhattan"     "Bronx"         "Brooklyn"      "Queens"
## [5] "Staten Island"
## [1] FALSE
```

```r
# Loop version 2
for(i in 1:length(nyc)){
  print(nyc[[i]]) # access every element in the list using double brackets
}
```

```
## [1] 8405837
## [1] "Manhattan"     "Bronx"         "Brooklyn"      "Queens"
## [5] "Staten Island"
## [1] FALSE
```

```
solfege<-matrix(c(’Do’,’Ri’,’Mi’,
                  ’Fa’,’So’,’la’,
                  ’So’,’So’,’So’,
                  ’F#’,’G#’,’A#’),
                byrow = TRUE,ncol = 3)
for(i in 1:nrow(solfege)){
  for(j in 1:ncol(solfege)){
    print(paste("On row",i,"and column",j,"the board contains",solfege[i,j]))
  }
}
```

```
## [1] "On row 1 and column 1 the board contains Do"
## [1] "On row 1 and column 2 the board contains Ri"
## [1] "On row 1 and column 3 the board contains Mi"
## [1] "On row 2 and column 1 the board contains Fa"
## [1] "On row 2 and column 2 the board contains So"
## [1] "On row 2 and column 3 the board contains la"
## [1] "On row 3 and column 1 the board contains So"
## [1] "On row 3 and column 2 the board contains So"
## [1] "On row 3 and column 3 the board contains So"
## [1] "On row 4 and column 1 the board contains F#"
## [1] "On row 4 and column 2 the board contains G#"
## [1] "On row 4 and column 3 the board contains A#"
```

```
# Pre-defined variables
rquote <- "r’s internals are irrefutably intriguing"
chars <- strsplit(rquote, split = "")[[1]]
# Initialize rcount
rcount <- 0
# Finish the for loop
for (char in chars) {
  if(char=="r"){
    rcount<-rcount+1
  }else if(char==’u’){
    break
  }

}
# Print out rcount
rcount
```

```
## [1] 5
```

Functions

```
# Finish the pow_two() function
pow_two <- function(x,print_info=TRUE) {

  y <- x ^ 2
  if (print_info)
     print(paste(x, "to the power two equals", y))
  return(y)
}
pow_two(2)
```

```
## [1] "2 to the power two equals 4"
```

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

Function scoping

- variables that are defined inside a function are not accessible outside that function.

```
# Check out the currently attached packages
search()
```

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

The apply family

lapply – returns a list

sapply – to simplify list

vapply – explicitly specify output format

```
pastaDough<-list(serving=6,
          ingredients=c("Flour","Eggs",
                        "Olive oil","Salt"),
          sugar=FALSE)
lapply(pastaDough, class)
```

```
## $serving
## [1] "numeric"
## 
## $ingredients
## [1] "character"
## 
## $sugar
## [1] "logical"
```

```
pastaIng<-c("Flour","Eggs","Olive oil","Salt")
lapply(pastaIng, nchar)
```

```
## [[1]]
## [1] 5
## 
```

```
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9
##
## [[4]]
## [1] 4
```

```
unlist(lapply(pastaIng, nchar))
```

```
## [1] 5 4 9 4
```

```
meatPrices<-list(2.99,5.88,3.76,3,5.6)
# Function 1
trible<-function(x){
  3*x
}
result<-lapply(meatPrices, trible)
str(result)
```

```
## List of 5
##  $ : num 8.97
##  $ : num 17.6
##  $ : num 11.3
##  $ : num 9
##  $ : num 16.8
```

```
unlist(result)
```

```
## [1]  8.97 17.64 11.28  9.00 16.80
```

```
# Function 2
multiply<-function(x,factor){
  x*factor
}
times3<-lapply(meatPrices, multiply,factor=3)
unlist(times3)
```

```
## [1]  8.97 17.64 11.28  9.00 16.80
```

```
times4<-lapply(meatPrices,multiply,factor=8)
unlist(times4)
```

```
## [1] 23.92 47.04 30.08 24.00 44.80
```

```
pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")
# Split names from birth year
split_math <- strsplit(pioneers, split = ":")
unlist(split_math)
```

```
## [1] "GAUSS"   "1777"    "BAYES"   "1702"    "PASCAL"  "1623"    "PEARSON"
## [8] "1857"
```

```
split_low<-lapply(split_math,tolower)
unlist(split_low)
```

```
## [1] "gauss"   "1777"    "bayes"   "1702"    "pascal"  "1623"    "pearson"
## [8] "1857"
```

```
str(split_low)
```

```
## List of 4
##  $ : chr [1:2] "gauss" "1777"
##  $ : chr [1:2] "bayes" "1702"
##  $ : chr [1:2] "pascal" "1623"
##  $ : chr [1:2] "pearson" "1857"
```

```
# Write function select_first()
select_first <- function(x) {
  x[1]
}
# Apply select_first() over split_low: names
names<-lapply(split_low,select_first)
# Write function select_second()
select_second <- function(x) {
  x[2]
}
# Apply select_second() over split_low: years
years<-lapply(split_low,select_second)
```

Anonymous Function

```
# Named function
triple <- function(x) { 3 * x }

# Anonymous function with same implementation
function(x) { 3 * x }
```

```
## function(x) { 3 * x }
```

```
# Use anonymous function inside lapply()
lapply(list(1,2,3), function(x) { 3 * x })
```

```
## [[1]]
## [1] 3
##
## [[2]]
## [1] 6
##
## [[3]]
## [1] 9
```

sapply

```
pastaIng<-c("Flour","Eggs","Olive oil","Salt")
result1<-sapply(pastaIng,nchar)
unlist(result1)
```

```
##     Flour      Eggs Olive oil      Salt
##         5         4         9         4
```

```
# use.names
result2<-sapply(pastaIng,nchar,USE.NAMES = FALSE)
result2
```

```
## [1] 5 4 9 4
```

```
first_and_last<-function(name){
  name<-gsub(" ", "",name)
  letters<-strsplit(name,split="")[[1]]
  c(first=min(letters),last=max(letters))
}
# Call the function
first_and_last("Boston")
```

```
## first  last
##   "B"   "t"
```

```
sapply(pastaIng, first_and_last)
```

```
##       Flour Eggs Olive oil Salt
## first "F"   "E"  "e"       "a"
## last  "u"   "s"  "v"       "t"
```

```
temp<-list(c(12,15,17,18,16,12),
           c(14,35,67,3,3,8,5),
           c(45,67,34,22,23,64))
lowest<-sapply(temp, min)
highest<-sapply(temp,max)
lowest
```

```
## [1] 12  3 22
```

```
highest
```

```
## [1] 18 67 67
```

```
# Function
extremes<-function(x){
  c(min=min(x),max=max(x))
}
sapply(temp,extremes)
```

```
##     [,1] [,2] [,3]
## min   12    3   22
## max   18   67   67
```

```
print_info <- function(x) {
  # cat
  cat("The average temperature is", mean(x), "\n")
}

# Apply print_info() over temp using sapply()
sapply(temp,print_info)
```

```
## The average temperature is 15
## The average temperature is 19.28571
## The average temperature is 42.5
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

```
# Apply print_info() over temp using lapply()
lapply(temp,print_info)
```

```
## The average temperature is 15
## The average temperature is 19.28571
## The average temperature is 42.5
```

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
## [[3]]
## NULL
```

```
?cat
```

```
# runif generates random numbers
sapply(list(runif (10), runif (10)),
       function(x) c(min = min(x), mean = mean(x), max = max(x)))
```

```
##           [,1]      [,2]
## min  0.1102837 0.0200719
## mean 0.5664348 0.5364073
## max  0.9464474 0.8623692
```

vapply

```
# Definition of basics()
basics <- function(x) {
  c(min = min(x), mean = mean(x), max = max(x))
}
vapply(temp,basics,FUN.VALUE=numeric(3))
```

```
##      [,1]     [,2] [,3]
## min    12  3.00000 22.0
## mean   15 19.28571 42.5
## max    18 67.00000 67.0
```

Utilities

```
#Functions for data structure
li<-list(log=TRUE,
         ch="hello",
         int_vec=sort(rep(seq(8,2,by=-2),times=2)))
str(li)
```

```
## List of 3
##  $ log    : logi TRUE
##  $ ch     : chr "hello"
##  $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
```

```
sort(rep(seq(8,2,by=-2),times=2))
```

```
## [1] 2 2 4 4 6 6 8 8
```

```
seq(1,10,by=3)
```

```
## [1]  1  4  7 10
```

```
seq(8,2,by=-2)
```

```
## [1] 8 6 4 2
```

```
seq(2,8,by=2)
```

```
## [1] 2 4 6 8
```

```
rep(c(1,2,3,4),times=2)
```

```
## [1] 1 2 3 4 1 2 3 4
```

```
rep(c(1,2,3,4),each=2)
```

```
## [1] 1 1 2 2 3 3 4 4
```

append() – Merge vectors or lists.

- is.*(): Check for the class of an R object.
- as.*(): Convert an R object from one class to another.

```
rh<-list(10,15,23,12)
lh<-list(5,8,22,16,9)
both<-append(rh,lh)
# use unlist() to convert lists to vectors
both_vector<-unlist(both)
sort(both_vector,decreasing = TRUE)
```

```
## [1] 23 22 16 15 12 10  9  8  5
```

```
class(both_vector)
```

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

```
class(both)
```

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

Regular expressions – regex

- Sequence of (meta)characters
- Pattern existence
- Pattern replacement
- Pattern extraction
- grep(), grepl() – check the pattern existence
- sub(),gsub()

```
# grepl()
animals<-c("cat","dog","bird","turtle","moose","ant")
# Find "a" in their name
grepl(pattern ="a",x=animals)
```

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

```
# Match strings that start with an "a"
grepl(pattern="^a",x=animals)
```

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

```r
# $ matches the empty string at end of a line
grepl(pattern="e$",x=animals)
```

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

```r
# grep()
animals<-c("cat","dog","bird","turtle","moose","ant")
# Get a vector of indices of the elements of x that yield a match
grep(pattern = "a",x=animals)
```

```
## [1] 1 6
```

```r
# Do the same thing by using which on grepl()
which(grepl(pattern = "a",x=animals))
```

```
## [1] 1 6
```

```r
grep
```

```
## function (pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,
##     fixed = FALSE, useBytes = FALSE, invert = FALSE)
## {
##     if (!is.character(x))
##         x <- structure(as.character(x), names = names(x))
##     .Internal(grep(as.character(pattern), x, ignore.case, value,
##         perl, fixed, useBytes, invert))
## }
## <bytecode: 0x7f9a54cd0748>
## <environment: namespace:base>
```

```r
# sub() takes three arguments: pattern, replacement, x
# this only replace one same letter
sub(pattern = "o",replacement = "e",x=animals)
```

```
## [1] "cat"    "deg"    "bird"   "turtle" "meose"  "ant"
```

```r
# replace two same pattern at a time
gsub(pattern = "o",replacement = "e",x=animals)
```

```
## [1] "cat"    "deg"    "bird"   "turtle" "meese"  "ant"
```

```r
# or, and
gsub(pattern = "a|e",replacement = "_",x=animals)
```

```
## [1] "c_t"    "dog"    "bird"   "turtl_" "moos_"  "_nt"
```

```r
gsub(pattern = "a|e|d",replacement = "#",x=animals)
```

```
## [1] "c#t"    "#og"    "bir#"   "turtl#" "moos#"  "#nt"
```

```r
# The emails vector has already been defined for you
emails <- c("john.doe@ivyleague.edu",
            "education@world.gov",
            "dalai.lama@peace.org",
            "invalid.edu",
            "quant@bigdatacollege.edu",
            "cookie.monster@sesame.tv")
# Use grepl() to match for "edu"
grepl(pattern="edu",x=emails) # return a logical vector
```

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

```r
# Use grep() to match for "edu", save result to hits
hits1<-grep(pattern="edu",x=emails) # return a vector
hits1
```

```
## [1] 1 2 4 5
```

```r
emails[hits1]
```

```
## [1] "john.doe@ivyleague.edu"   "education@world.gov"
## [3] "invalid.edu"              "quant@bigdatacollege.edu"
```

```r
hits2<-grep(pattern="@.*\\.edu$",x=emails)
hits2
```

```
## [1] 1 5
```

```r
emails[hits2]
```

```
## [1] "john.doe@ivyleague.edu"   "quant@bigdatacollege.edu"
```

```r
# Use sub() to convert the email domains to funny.edu
sub(pattern="@.*\\.edu$",replacement="@funny.edu",x=emails)
```

```
## [1] "john.doe@funny.edu"       "education@world.gov"
## [3] "dalai.lama@peace.org"     "invalid.edu"
## [5] "quant@funny.edu"          "cookie.monster@sesame.tv"
```

- .*: A usual suspect! It can be read as “any character that is matched zero or more times”.
- \s: Match a space. The “s” is normally a character, escaping it (\) makes it a metacharacter.
- [0-9]+: Match the numbers 0 to 9, at least once (+).
- ([0-9]+): The parentheses are used to make parts of the matching string available to define the replacement. The \1 in the replacement argument of sub() gets set to the string that is captured by the regular expression [0-9]+.

```
awards <- c("Won 1 Oscar.",
  "Won 1 Oscar. Another 9 wins & 24 nominations.",
  "1 win and 2 nominations.",
  "2 wins & 3 nominations.",
  "Nominated for 2 Golden Globes. 1 more win & 2 nominations.",
  "4 wins & 1 nomination.")

sub(".*\\s([0-9]+)\\snomination.*$", "\\1", awards)
```

```
## [1] "Won 1 Oscar." "24"           "2"            "3"            "2"
## [6] "1"
```

Times and dates

```
today<-Sys.Date()
today
```

```
## [1] "2020-06-09"
```

```
class(today)
```

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

```
now<-Sys.time()
now
```

```
## [1] "2020-06-09 11:22:14 EDT"
```

```
class(now)
```

```
## [1] "POSIXct" "POSIXt"
```

```
# Create Date
date<-as.Date("2111-11-11")
class(date)
```

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

```
date
```

```
## [1] "2111-11-11"
```

Dates

- %Y: 4-digit year (1982)
- %y: 2-digit year (82)
- %m: 2-digit month (01)
- %d: 2-digit day of the month (13)
- %A: weekday (Wednesday)
- %a: abbreviated weekday (Wed)
- %B: month (January)
- %b: abbreviated month (Jan)

```r
# Definition of character strings representing dates
str1 <- "May 23, ’96"
str2 <- "2012-03-15"
str3 <- "30/January/2006"
# Convert to yyyy-dd-mm format, need to specify the format
date1<-as.Date(str1, format = "%b %d, ’%y")
date2<-as.Date(str2)
date3<-as.Date(str3,format ="%d/%b/%Y" )
date1
```

```
## [1] "1996-05-23"
```

```r
date2
```

```
## [1] "2012-03-15"
```

```r
date3
```

```
## [1] "2006-01-30"
```

```r
# Convert dates to formatted strings
# Select the weekday for date1
format(date1,"%A")
```

```
## [1] "Thursday"
```

```r
# Select the day of the month for date2
format(date2,"%d")
```

```
## [1] "15"
```

```r
# Select the abbreviated month and the 4-digit year, separated by a space.
format(date3,"%b %Y")
```

```
## [1] "Jan 2006"
```

```r
# Convert dates to character strings that use different date notation
today<-Sys.Date()
format(Sys.Date(), format="Today is a %A" )
```

```
## [1] "Today is a Tuesday"
```

Times

- To convert a character string to a POSIXct object – **as.POSIXct()**
- **as.POSIXct()** uses a default format to match character strings. In this case, it's %Y-%m-%d %H:%M:%S
- To convert from a POSIXct object to a character string – **format()**
- ?strptime
    - %H: hours as a decimal number (00-23)
    - %I: hours as a decimal number (01-12)
    - %M: minutes as a decimal number
    - %S: seconds as a decimal number
    - %T: shorthand notation for the typical format %H:%M:%S
    - %p: AM/PM indicator

```
# Definition of character strings representing times
str1 <- "May 23, '96 hours:23 minutes:01 seconds:45"
str2 <- "2012-3-12 14:23:08"
# Convert the strings to POSIXct objects: time1, time2
time1<-as.POSIXct(str1,format="%B %d, '%y hours:%H minutes:%M seconds:%S")
time2<-as.POSIXct(str2)
# Create a string from time1 containing only the minutes.
format(time1,"%M")
```

```
## [1] "01"
```

```
# Extract the hours and minutes as "hours:minutes AM/PM".
format(time2,"%H:%M %p")
```

```
## [1] "14:23 PM"
```

Calculation with Dates

```
day1<-as.Date("2020-2-5")
day2<-as.Date("2020-2-10")
day3<-as.Date("2020-3-25")
day4<-as.Date("2020-3-30")
day5<-as.Date("2020-4-6")
# Difference between last and first pizza day
day5-day1
```

```
## Time difference of 61 days
```

```
# Create vector pizza
pizza<-c(day1,day2,day3,day4,day5)
day_diff<-diff(pizza)
day_diff
```

```
## Time differences in days
## [1]  5 44  5  7
```

```
# Average period between two consecutive pizza days
mean(day_diff)
```

```
## Time difference of 15.25 days
```

Calculations with times

```
now <- Sys.time()
now + 3600          # add an hour
```

```
## [1] "2020-06-09 12:22:14 EDT"
```

```
now - 3600 * 24     # subtract a day
```

```
## [1] "2020-06-08 11:22:14 EDT"
```

```
# Adding or substracting time objects
birth <- as.POSIXct("1879-03-14 14:37:23")
death <- as.POSIXct("1955-04-18 03:47:12")
einstein <- death - birth
einstein
```

```
## Time difference of 27792.55 days
```

```
x <- 1:4
y <- 2:3
z=x+y
class(z)
```

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

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

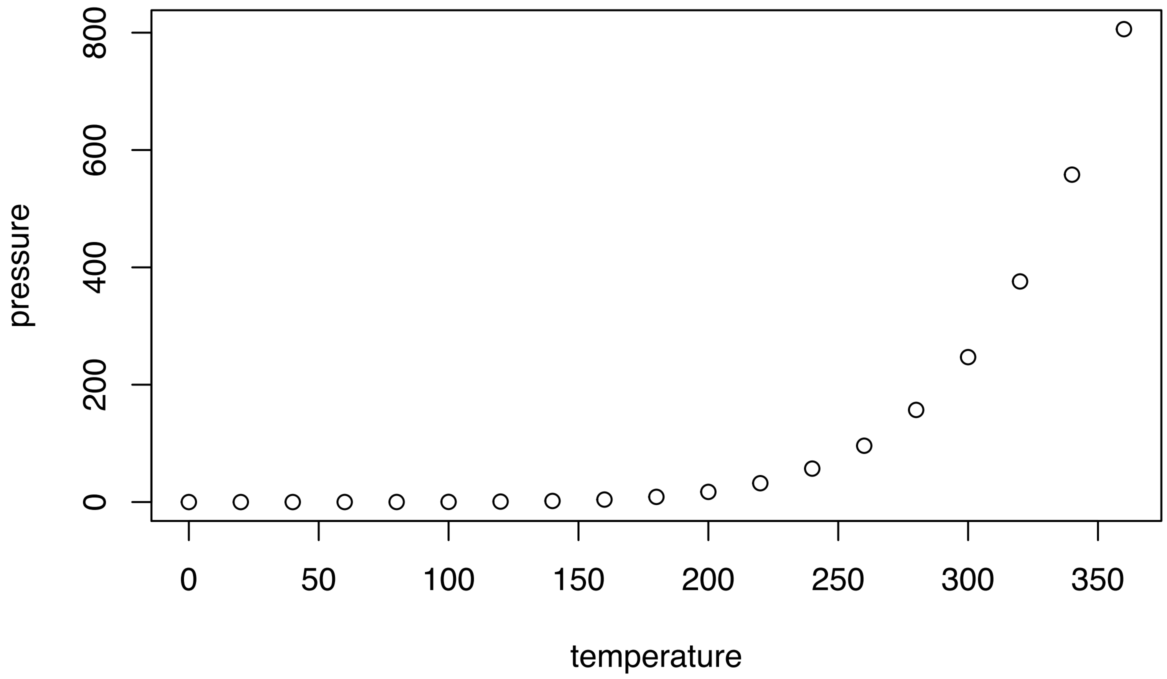
When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```
summary(cars)
```

```
##      speed           dist       
##  Min.   : 4.0   Min.   :  2.00  
##  1st Qu.:12.0   1st Qu.: 26.00  
##  Median :15.0   Median : 36.00  
##  Mean   :15.4   Mean   : 42.98  
##  3rd Qu.:19.0   3rd Qu.: 56.00  
##  Max.   :25.0   Max.   :120.00  
```

Including Plots

You can also embed plots, for example:



Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.