R Data Structures

Matt

Friday, April 15, 2016

Welcome to R! For the first few tutorials, I thought I would go over the most essential data structures. Some examples are borrowed from The Art of R programming by Norman Matloff. Others are made up, for fun and profit.

Vectors

```
#### Vector Intro ####
# Vectors are to R what six-block bricks are to Legos.
x <- 6
print(x)
```

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

```
# empty vectors can be assigned with the vector() function
emptyVector <- vector(mode="numeric",length=0)
print(emptyVector)
```

```
## numeric(0)
```

```
# BTW, you can check the requirements of a function using ?FunctionName()
?vector()
```

```
## starting httpd help server ... done
```

```
# Vectors can be combined with the c() (combine) function
myVector <- c(1,2,4)
print(myVector)
```

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

```
# You might notice R doesn't have scalars but really just vectors of length 1
oneElement <- 8
print(length(oneElement))
```

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

```
# also, length() is a super useful function.

# the length() function can be used to get the length of an object
threeElement <- c(2,4,6)
print(length(threeElement))
```

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

```
#### Modes ####
# You should read up on modes
?mode

# Numeric modes can be either integers or floats
doubleVector <- 6.6
print(mode(doubleVector))
```

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

```
# Now ints
integerVector <- 6
print(mode(integerVector))
```

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

```
# BTW, you can coerce an R object by using the as.Type() function
print(as.integer(doubleVector)) # notice that the value is 6 rather than 6.6
```

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

```
# Vectors can also be logical (but not illogical, because everything in R makes perfect sense )
logicalVector <- c(TRUE,FALSE,TRUE)
print(mode(logicalVector))
```

```
## [1] "logical"
```

```
# BTW, TRUE and FALSE can be shortened to T and F
logicalVectorShort <- c(T,F,T)

# Strings and Text are held in character vectors
characterVector <- c("GEOINT","Pathfinder")
print(mode(characterVector))
```

```
## [1] "character"
```

```
# vectors can support any R mode but like starter Pokemon, you only get to have one.
legalAssignment <- c(1,2,3,10) # all elements are numeric, mode will show numeric
print(mode(legalAssignment))
```

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

```
# pay attention when we try to mix things that shouldn't be mixed. Bleach and ammonia any one?
illegalAssignment <- c(TRUE,25,"hydrazine")
print(illegalAssignment)
```

```
## [1] "TRUE"      "25"        "hydrazine"
```

```
# Nothing broke because R will force type coercion on vectors. Notice TRUE and 25 are characters.
print(mode(illegalAssignment))
```

```
## [1] "character"
```

```
#### Slicing and Dicing Vectors ####
# we can assign a vector to play with using the the ?seq() function
ourVector <- seq(1,100,1) # if we wanted to count by threes, you'd change the third variable from 1 to 3

# we could also have done it this way
ourVector <- c(1:100)

# we can subset vectors
vector99 <- ourVector[99]
print(vector99)
```

```
## [1] 99
```

```
# we can filter a vector by another vector
oneTen65 <- c(1,10,65)
ourSubset <- ourVector[oneTen65]
print(ourSubset)
```

```
## [1]  1 10 65
```

```
# you can remove the elements of one vector from another
notOneTen65 <- ourVector[-oneTen65]

# We can use any() to check if our subset worked
f <- any(notOneTen65 == 1) # the double == sign is used for comparison, whereas a singe =
is assignment
print(f)
```

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

```
t <- any(notOneTen65 == 2)
print(t)
```

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

```
# any() and all() are super useful. They return a logical vector based on the condition y
ou want to check.
allZeros <- c(0,0,0)

# we could also have assigned the allZeros variable using the rep() function
allZeros <- rep(0,3)
print(all(allZeros == 0))
```

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

```
if (all(allZeros == 0)){ # I haven't covered control flow yet, but you get it!
  print("Yay!")
}
```

```
## [1] "Yay!"
```

```
# we can also subset vectors using the which() function. It's fun to use which()!
greater10Less30 <- ourVector[which(ourVector > 10 & ourVector < 30)]

# which returns a vector of indices where a condition is true for example, which element
is == 99?
toyVector <- c(1,99,2)
index <- which(toyVector > 2)
print(index) # notice it's the second element.
```

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

```
# if we wanted the actual value we'd subset the orginal vector by our index
print(toyVector[index])
```

```
## [1] 99
```

```
#### Vectorized operations and recycling ####
# vector operations are much faster than looping and are supported by R
# Say, if we wanted to subtract everything in ourVector by 3.
# we could do this:
minus3 <- c()
for (i in ourVector){
  j <- i - 3
  minus3 <- c(minus3,j)
}

# It's much better to just do this
betterMinus3 <- ourVector - 3

# We can time the difference with a larger vector
bigVector <- c(1:100000)
minus3 <- c()
ptm <- proc.time()
for(i in bigVector){
  j <- i - 3
  minus3 <- c(minus3,j)

}
t <- proc.time() - ptm

# Now with a vector
ptm <- proc.time()
betterMinus3 <- bigVector - 3
t <- proc.time() - ptm
print(t)
```

```
##    user  system elapsed 
##       0       0       0
```

```
v <- c(1,2,3)
b <- c(1,2,3,4)
print(b - 1)
```

```
## [1] 0 1 2 3
```

```
# notice, that it recylced the first element of v and subtracted it from the fourth element of b

#### Fun with functions and vectors ####
# recall that variable ourVector is a numeric vector with values 1:100
print(length(ourVector))
```

```
## [1] 100
```

```
# If we wanted to get the mean of ourVector, we'd use mean
print(mean(ourVector))
```

```
## [1] 50.5
```

```
# Similarly, we could grab the median using median
print(median(ourVector))
```

```
## [1] 50.5
```

```
# we could add a new number to our vector using the orginal vector and c()
ourVector <- c(ourVector,101)

# This gives us a new median
print(median(ourVector))
```

```
## [1] 51
```

```
# we could sort ourVector using sort(). Let's make it decreasing rather than increasing
print(ourVector[1])
```

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

```
ourVector <- sort(ourVector,decreasing=TRUE)
print(ourVector[1])
```

```
## [1] 101
```

Vector Challenges

1. Create a vector, called myVector and assign it the values 1 through 1000, increasing by 3s.
2. Print out the 213th element of myVector.

3. Create a subset of myVector called truncatedVector. Include only the values greater than 200 but less than 700.
4. How many elements are in truncated vector?
5. Print out the mean of truncatedVector.
6. Combine your orginal myVector along with truncatedVector. What’s the median?