# Computing HW 2

*Michael Ackerman*

*September 17, 2015*

**1)**

```
cancer.df <- read.csv("~/Documents/Bios6301/datasets/cancer.csv")
```

**2)**

```
dim(cancer.df)
```

```
## [1] 42120     8
```

**3)**

```
names(cancer.df)
```

```
## [1] "year"       "site"       "state"      "sex"        "race"
## [6] "mortality"  "incidence"  "population"
```

**4)**

```
cancer.df[3000,6]
```

```
## [1] 350.69
```

**5)**

```
cancer.df[172,]
```

```
##     year                           site  state  sex  race mortality
## 172 1999 Brain and Other Nervous System nevada Male Black         0
##     incidence population
## 172         0      73172
```

**6)**

```
cancer.df[,9] <- cancer.df[,7]/1000
```

**7)**

```
length(cancer.df[,9][cancer.df[,9]==0])
```

```
## [1] 23191
```

**8)**

```
  cancer.df[cancer.df[,9]==max(cancer.df[,9]),]
```

```
##       year   site      state    sex  race mortality incidence population
## 21387 2002 Breast california Female White   3463.74     18774   13690681
##           V9
## 21387 18.774
```

**Data Types**
**1)**

```
 x <- c("5","12","7")
sort(x)
sum(x)
max(x)
```

Because our x values are place in quotes, they are actually interpreted as characters. Therefore, I think the sort(x) function is analyzing the first numbers of our characters as if they were letters and putting them in alphabetical order. The sum(x) function completely fails beacuse you simply can't add up characters. The max(x) function just returns the character that's placed last in our sort(x) function based on the same alphabetic rules.

**2)**

```
y <- c("5",7,12)
y[2]+y[3]
```

```
y <- c("5",7,12)
is.numeric(y[2])
```

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

We should be able to add the numbers 7 and 12 but by declaring the first term of the vector as a character automatically creates the other imputs as characters as well.

**3)**

```
z <- data.frame(z1="5",z2=7,z3=12)
z[1,2]+z[1,3]
```

```
## [1] 19
```

the data.frame() function allows each column to keep it's own class; therefore, 7 and 12 can be used as numeric values.

**Data Structures**
**1)**

```
c(rep(1:8),rep(7:1)) # Is there a better way to do this?
```

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

**2)**

```r
rep(1:5, times= 1:5)
```

```
##  [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
```

**3)**

```r
matrix(rep(1,times=9),nrow=3)-diag(3)
```

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

**4)**

```r
 A <- matrix(rep(1:4, each=5), ncol = 4)
for(i in 1:5){
  A[i,]<-A[i,]^i
}
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    1    4    9   16
## [3,]    1    8   27   64
## [4,]    1   16   81  256
## [5,]    1   32  243 1024
```

Well… I did it. But this is rather unsatisfying. Is there a one line way? Probably using apply() ?

**Basic Programming 1)**

```r
h <- function(x,n){
  h <- 0
  for (i in 1:n){
    h <- h + x^(i-1)
  }
  return(h)
}
##Test
if(h(.5,100)==2) print("Correct!!")
```

```
## [1] "Correct!!"
```

**2)**

```r
### 1
n <- 1000
sum <- 0
for(i in 1:n) {
  if(i %% 3 == 0 | i %% 5 == 0 ){
    sum <- sum + i
  }
}
sum
```

```
## [1] 234168
```

```
## 2
n <- 1000000
sum <- 0
for(i in 1:n) {
 if(i %% 4 == 0 | i %% 7 == 0 ){
   sum <- sum + i
 }
}
sum
```

```
## [1] 178572071431
```

This number is huge.

**3)**

```
fib <- c(1,1)
evenfib <- c()
 # while(length(evenfib) < 15){
  for(i in 3:45){
    fib[i] <- fib[i-1]+fib[i-2]
      if(fib[i] %% 2 == 0 ){
      evenfib <- c(evenfib, fib[i])
      }
  }
  #}
sum(evenfib[1:15])
```

```
## [1] 1485607536
```

I had to comment out my while loop because it wouldn't see "evenfib" and stop the loop at 15 as I was hoping for. Why??????