Lecture 3
Summary of previous lecture
Getting help
> help.start()
> ?sum

sum R Documentation

Sum of Vector Elements
Description
sum returns the sum of all the values present in its arguments.
Usage
sum(..., na.rm = FALSE)
Arguments
... numeric or complex or logical vectors.
na.rm logical. Should missing values be removed?
Details
This is a generic function: methods can be defined for it directly or via the Summary group
generic. For this to work properly, the arguments ... should be unnamed, and dispatch is on the
first argument.
If na.rm is FALSE an NA value in any of the arguments will cause a value of NA to be returned,
otherwise NA values are ignored.
Logical true values are regarded as one, false values as zero. For historical reasons, NULL is
accepted and treated as if it were integer(0).
Value
The sum. If all of ... are of type integer or logical, then the sum is integer, and in that case the
result will be NA (with a warning) if integer overflow occurs. Otherwise it is a length-one numeric
or complex vector.
NB: the sum of an empty set is zero, by definition.
S4 methods
This is part of the S4 Summary group generic. Methods for it must use the signature x, ...,
na.rm.
'plotmath' for the use of sum in plot annotation.
References
Brooks/Cole.
See Also
colSums for row and column sums.

> apropos("sum")
[1] "__C__summary.table"     "__T__Summary:base"
[3] "Summary"                "Summary.Date"
[7] "Summary.data.frame"     "Summary.difftime"
[9] "Summary.factor"         "Summary.numeric_version"
[11] "colSums"                "contr.sum"
[13] "cumsum"                 "print.summary.table"
[15] "rowSums"                "rowsum"
[17] "rowsum.data.frame"      "rowsum.default"
[19] "sum"                    "summary"
[21] "summary.Date"           "summary.POSIXct"
[23] "summary.POSIXlt"        "summary.aov"
[25] "summary.aovlist"        "summary.connection"
[27] "summary.data.frame"     "summary.default"
[29] "summary.factor"         "summary.glm"
[31] "summary.infl"           "summary.lm"
[33] "summary.manova"         "summary.matrix"
[35] "summary.mlm"            "summary.srcfile"
[37] "summary.srcref"         "summary.stepfun"
[39] "summary.table"          "summaryRprof"

> args(sum)
function (..., na.rm = FALSE)
NULL
> help.search("plot")
> Vectors
> a= 10:1
> a
> [1] 10 9 8 7 6 5 4 3 2 1
> a[1:4]
> [1] 10 9 8 7
> a[-(1:4)]
> [1] 6 5 4 3 2 1

Other vectors
> a=5
> [1] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE
> as.character(a)
> [1] "10" "9" "8" "7" "6" "5" "4" "3" "2" "1"
> ls()
> [1] "a"

Sampling
> x=sample(c("Head","Tail"),1000, rep=TRUE, p=c(0.2,0.8) )
> table(x)
x
Head Tail
171 829

Attributes
Look again at the result of the function table()
> table(x)
x
Head Tail
171 829
> tx=table(x)
> tx
x
Head Tail
171 829
> tx*10
x
Head Tail
1710 8290
> tx^2
x
Head Tail
29241 687241

tx is some strange construct, because it seems to be a vector, but in addition to what it stores, there are also these strange headings.
In addition to values, R can store additional attributes about the data.
Here is an example:
> pop = c( male=1000, female=1500)
> pop
     male  female
1000    1500
> We created a vector, and gave a name to each value. This name is stored separately.
> pop/sum(pop)
     male  female
     0.4    0.6
> names(pop)
> [1] "male"  "female"

> names() is a function to access the names property
> names(pop) = c( "M", "F" )
> pop
>          M    F
> 1000 1500
> names(pop)
> [1] "M"  "F"

> The names() function can be used to recall and also to change the names.
> names(pop) = c("Males")
> pop
> Males <NA>
>          M    F
> 1000 1500
> names(pop) = c("Male","Female","Unknown")
> Error in names(pop) = c("Male","Female","Unknown") :
> 'names' attribute [3] must be the same length as the vector [2]
> pop
> Males <NA>
>          M    F
> 1000 1500
> names(pop) = c( "M", "F" )
>
> When a vector has names, we can use them also as indexes
> pop[2]
>          F
>          1500
> >
> > pop
>          M    F
> 1000 1500
> pop["M"]
>          M
>          1000
> pop[ c ("F","M")]
>          F    M
>          1500 1000
>
> We can also create a vector with names like this:
> pop = c( males=100, females = 200)
> pop

> Populations

> Matrices
A matrix is like a vector. Except that it has two dimensions. In fact, in R, a matrix is a vector
that has an additional property that says how many rows and columns it has.
Let us create a matrix
> M = matrix( 1:12, nrow=4, ncol=3)
> M

     [,1] [,2] [,3]
[1,]  1   5   9
[2,]  2   6  10
[3,]  3   7  11
[4,]  4   8  12
>
Look at the order in which the numbers are put in the matrix. Down the first column, then the
second, and so on.
To access a part of the matrix, we give the row and column
> M
We can also get a bigger part of the matrix

\[
\begin{bmatrix}
1 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12
\end{bmatrix}
\]

If we want to know the dimensions of a matrix, we use the function `dim()`.

\[
\text{dim}(M)
\]

\[
4 \times 3
\]

\[
\begin{bmatrix}
1 & 5 & 9 \\
2 & 6 & 10 \\
3 & 7 & 11 \\
4 & 8 & 12
\end{bmatrix}
\]

\[
\text{rownames}(M) = \text{c}("a","b","c","d")
\]

\[
\text{colnames}(M) = \text{c}("A","B","C")
\]

\[
\begin{bmatrix}
A & B & C \\
a & 1 & 5 & 9 \\
b & 2 & 6 & 10 \\
c & 3 & 7 & 11 \\
d & 4 & 8 & 12
\end{bmatrix}
\]

\[
\begin{bmatrix}
A & C \\
a & 1 & 9 \\
c & 3 & 11 \\
d & 4 & 12
\end{bmatrix}
\]

4 rows and 3 columns.
What happened here is that because the result had just one row, it was returned as a vector, without dimensions. If we want the result to still be a matrix, we have to ask for it.

The format is a bit strange

We can manipulate matrices just like we did vectors

```
> M1  , ,drop=F]
   A B C
   a 1 5 9
> dim( M1  , ,drop=F]
   [1] 1 3
> M[,2, drop=F]
   B
   a 5
   b 6
c 7
d 8
> dim( M[,2, drop=F]
   [1] 4 1
```

```r
> M / sum(M)
   A    B    C
   a 0.01282051 0.06410256 0.1153846
   b 0.02564103 0.07692308 0.1282051
   c 0.03846154 0.08974359 0.1410256
   d 0.05128205 0.10256410 0.1538462
> M[1:2,1:2]
   A B
   a 1 5
   b 2 6
> M[1:2,1:2] = 0
> M
   A B C
   a 0 0 9
   b 0 0 10
```
See how the right-hand side was recycled when it was put in place. There are some other convenient ways to create matrices

```r
> M = rbind(1:3, 3:1)
> M
[,1] [,2] [,3]
[1,]  1  2  3
[2,]  3  2  1

bind two vectors as rows
> M = rbind( 1:3, 3:1, 10:8)
> M
[,1] [,2] [,3]
[1,]  1  2  3
[2,]  3  2  1
[3,] 10  9  8

And, there is also cbind()

```r
> cbind(1:4, 4:1)
[,1] [,2]  1  4
[2,]  2  3
[3,]  3  2
[4,]  4  1
> M = cbind(1:2, 1:2)
> M
[,1] [,2]
[1,]  1  1
[2,]  2  2

Just like the names() attribute, the dim of a matrix is just another attribute attached to a vector.

```r
> a=1:12
> a
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> dim(a) = c(3,4)
> a
[1,] 1 4 7 10
[2,] 2 5 8 11
[3,] 3 6 9 12
> dim(a) = c(4,3)
> a
[,1] [,2] [,3]
[1,] 1 5 9
[2,] 2 6 10
[3,] 3 7 11
[4,] 4 8 12
> dim(a) = c(6,2)
> a
[,1] [,2]
[1,] 1 7
We can also still access the matrix as a vector:

```r
> a
[,1] [,2]
[1,]  1  7
[2,]  2  8
[3,]  3  9
[4,]  4 10
[5,]  5 11
[6,]  6 12
> a[1,2]
[1] 7
> a[7]
[1] 7
```

So that we can do things like:

```r
> b <- a
> b
[,1] [,2]
[1,]  1  7
[2,]  2  8
[3,]  3  9
[4,]  4 10
[5,]  5 11
[6,]  6 12
> a[ a<5 ] = 0
> a
[,1] [,2]
[1,]  0  7
[2,]  0  8
[3,]  0  9
[4,]  0 10
[5,]  5 11
[6,]  6 12
> b
[,1] [,2]
[1,]  1  7
[2,]  2  8
[3,]  3  9
[4,]  4 10
[5,]  5 11
[6,]  6 12
> b[3:4,]=0
> b
[,1] [,2]
[1,]  1  7
[2,]  2  8
[3,]  0  0
[4,]  0  0
[5,]  5 11
[6,]  6 12
```
Diploids

Let us look at problem 7 again.

7. A sexual population is reproducing. The sperm contain 20% genotype A, and 80% genotype a, and the eggs 50% and 50%. Create 100 random diploid offspring, and count how many are of types A,A or A,a or a,a.

> male.sperm = sample( c("A","a") , 100, rep=T, p= c(0.2,0.8) )
> female.eggs = sample( c("A","a") , 100, rep=T, p= c(0.5,0.5) )
> male.sperm
[1] "a" "a" "a" "a" "A" "A" "A" "a" "a" "a" "A" "A" "A" "a" "a" "a" "A" "A" "a"
[19] "a" "a" "a" "a" "A" "A" "a" "A" "a" "a" "A" "A" "a" "a" "A" "A" "a"
[37] "a" "a" "a" "a" "A" "A" "a" "A" "a" "a" "A" "A" "a" "a" "A" "A" "a"
[55] "a" "a" "a" "a" "A" "a" "A" "a" "A" "a" "A" "a" "A" "A" "A" "a" "a"
[73] "a" "a" "a" "a" "A" "A" "a" "a" "A" "a" "A" "a" "A" "A" "a" "a"
[91] "A" "a" "a" "a" "a" "a" "A" "A" "a" "a" "a" "a" "a"
> female.eggs
[1] "a" "A" "A" "A" "A" "A" "A" "a" "A" "A" "a" "A" "A" "a" "a" "a" "a" "a"
[37] "a" "A" "A" "A" "A" "A" "A" "A" "A" "a" "A" "A" "A" "a" "A" "A" "A"
[73] "a" "a" "a" "a" "A" "A" "A" "a" "A" "A" "A" "a" "A" "A" "A" "A" "A"
[91] "a" "A" "A" "A" "A" "A" "A" "A" "A" "a" "a"
> offspring = rbind( Paternal = male.sperm, Maternal= female.eggs )
> offspring
Paternal "a" "a" "a" "a" "A" "A" "a" "a" "a" "a" "A" "A" "a" "a" "A" "A" "A" "A" "A" "A" "A" "A" "a"
How can we count the alleles?

We have homozygots a,a, heterozygots A,a and a,A and homozygots A,A.

What is the frequency of the allele A in the new population?

> offspring

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>26</td>
<td></td>
</tr>
</tbody>
</table>

> table( offspring[1,], offspring[2,] )

A a
A 14 10
a 50 26

> table( offspring[1,], offspring[2,] )
Let us calculate the frequencies
> t.alleles / sum(t.alleles)
  offspring
  A  a
  0.44 0.56
>
We see that the frequencies of the gene in the offspring is between that in males and in females. That makes sense, because half of the alleles came from the males, and half from the females
> table(offspring)
  offspring
  A  a
  88 112
>
Let us do two generations.
For this, we need to generate males and females. Let us do that, and let us generate a slightly bigger population
First, female offspring
> sperm = sample( c("A","a"), 1000, rep=T, p= c(0.2,0.8) )
> eggs = sample( c("A","a"), 1000, rep=T, p= c(0.5,0.5) )
> Fem.off = rbind( M = sperm, F= eggs )
>
Then male offspring
> sperm = sample( c("A","a"), 1000, rep=T, p= c(0.2,0.8) )
> eggs = sample( c("A","a"), 1000, rep=T, p= c(0.5,0.5) )
> Male.off = rbind( M = sperm, F= eggs )
>
Now, what would be the frequency of the genotypes in the sperm?
> table( Male.off)
  Male.off
  A  a
  715 1285
> table( Male.off) / sum( table( Male.off) )
What is the frequency in the genotype of eggs?

```
> table( Fem.off)
Fem.off
  A  a
  695 1305
> table( Fem.off) / sum( table( Fem.off) )
Fem.off
  A  a
 0.3475 0.6525
```

These two are very similar! Why?
Both were produced in the same manner - sampling from sperm and from eggs, and putting these together. So obviously, up to “sampling error”, the two will be identical.

This is called the Hardy-Weinberg equilibrium, or the Hardy-Weinberg principle.

Hardy was a mathematician, also known for the “discovery” of Srinivasa Ramanujan, one of the greatest mathematicians ever. Hardy solved this problem after it was introduced to him by the geneticist Reginald Punnett, with whom he played cricket.

**Loops**

There are many ways to do loops in R. Loops are a way to do a certain task again and again. As you saw in the homework, sometimes pressing enter a million times can be tiring...

```R
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
```

```
> a
[,1] [,2] [,3]
[1,] 1 2 3
[2,] 3 2 1
```

```
> for( i in 1:10) i
> i
[1] 10
```

```
> print(i)
[1] 10
```

```
> for( name in c("Wright","Fisher","Haldane") ) print(name)
[1] "Wright"
[1] "Fisher"
[1] "Haldane"
```

```
> for( i in 1:30) print( 1:i )
[1] 1
[1] 1 2
[1] 1 2 3
[1] 1 2 3 4
[1] 1 2 3 4 5
[1] 1 2 3 4 5 6
[1] 1 2 3 4 5 6 7
[1] 1 2 3 4 5 6 7 8
[1] 1 2 3 4 5 6 7 8 9
[1] 1 2 3 4 5 6 7 8 9 10
[1] 1 2 3 4 5 6 7 8 9 10 11
[1] 1 2 3 4 5 6 7 8 9 10 11 12
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13
```
In a for loop, a variable will go over the values of a certain vector. So, say we'd like to do 10 generations of neutral evolution of a population, we can do

```r
p = 1:100
for( i in 1:10) p = sample( p, length(p), rep=T )
table(p)
```

> `table(p)`
>     p  
>     4 7 8 10 14 19 24 26 28 32 39 46 63 71 80 84
>     5 2 4 8 15 2 3 4 12 3 6 10 1 12 12 1

> `length(table(p))`
> [1] 16

After 10 generations, we already lost most of the ancestral lineages. If you need to do more than one command inside the loop, you separate this with curly brackets `{}`: (to open a line in TeXmacs without executing it, press shift-enter)

```r
> p=1:100
> for(i in 1:10) {
>   print( length(table(p)) )
>   p = sample( p, length(p), rep=T )
> }
> print( length(table(p)) )
```

> `+ + [1] 100`

```r
[1] 65
[1] 48
[1] 42
[1] 34
[1] 29
[1] 24
[1] 22
[1] 21
```
We can print the results. We could also store them in a vector

```r
> p = sample(p, length(p), rep=T)
> results[i] = length(table(p))

> plot(results); v()
```

```
[1] 17
```

```
> 1-exp(-1)
[1] 0.6321206
```

```
> plot(log(1:100), log(results)); v()
```
log(results)

\log(1:100)