**Missing values**

When an element or value is “not available” or a “missing value” the data values are represented by such special symbols NA. When a value (missing data, square root or logarithm of negative number). For these cases, any operation on NA becomes NA.

**To check whether a data value is NA (cannot use data==NA)**

The function `is.na(x)` gives a logical vector of the same size as x with value TRUE if and only if the corresponding element in x is NA.

```r
>x<-c(4:8,NA) ; x
>is.na(x)  # is TRUE both for NA and NAN values.
[1] FALSE FALSE FALSE FALSE FALSE  TRUE
>x==NA
[1] NA NA NA NA NA NA
> x>1
TRUE TRUE TRUE TRUE TRUE  NA
>sum(x)  # sum (count) elements correspond to the TRUE values
NA
> !is.na(x)
```

To remove missing values from x:

```r
>x= x[!is.na(x)]
>x
[1] 4 5 6 7 8
```

```r
> z=c(2,5,4,NA,3,-2)
>z==NA
> z[z!=NA]
> z[!is.na(z)]
[1] 2,5,4,3,-2
```

Assignment should be (->) at the end of line, cannot use (=).

Different??

```r
> w = z[!is.na(z)]; w
> (z+1)[(!is.na(z)) & z>0] -> S ; S
> z

> z[is.na(z)] <- 0 ;z  # same as (=)
[1] 2 5 4 0 3 -2
```
There is a second kind of “missing” values which are produced by numerical computation; it is called Not a Number, NaN, values. Examples are

```r
> log(-2)
[1] NaN
Warning message:
Nans produced in: log(x)
```

```r
> 0/0
# give NAN
> Inf - Inf
# give NAN
```

```r
> xx = Inf/Inf
> is.nan(xx)  # is TRUE only for NAN values.
```

- What is the different between console and script file??
- How to use keyboard shortcuts for the command line??
Example:
Suppose we have height (in inches) and weight (in pounds) of 9 people

<table>
<thead>
<tr>
<th>Height</th>
<th>120</th>
<th>125</th>
<th>130</th>
<th>135</th>
<th>140</th>
<th>145</th>
<th>150</th>
<th>155</th>
<th>135</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
<td>64</td>
<td>65</td>
<td>66</td>
<td>67</td>
<td>68</td>
</tr>
</tbody>
</table>

> height<- c(seq(120,155,5),135)

> weight<- 60:68
# use logical expression to see how many less than 140
> height<140
> sum(height <140)  # correspond to the TRUE values
> height [height <140]
> height[height >150]
> weight[weight<65]
> weight[weight<60]  # no weight less than 60
> height [height <140& height!=120]
#combine weight and height
> height[weight>65]

> height<- c(seq(120,155,5),135)
> weight<- 60:68
> # use logical expression to see how many less than 140
> height<140
[1] TRUE TRUE TRUE TRUE FALSE FALSE FALSE FALSE TRUE
> sum(height <140)
[1] 5
> > height [height <140]
[1] 120 125 130 135 135
> height[height >150]
[1] 155
> weight[weight<65]
[1] 60 61 62 63 64
> weight[weight<60]  # no height less than 60
numeric(0)
> height [height <140& height!=120]
[1] 125 130 135 135
> #combine weight and height
> height[weight>65]
[1] 150 155 135
Printing in R:
> cat() # Concatenate and Print (for general printing)
> print() #print numeric or character data
> paste() # Concatenate vectors after converting to character.

Examples

> x=1:10
> print(x)
[1] 1 2 3 4 5 6 7 8 9 10
> cat("class = ", x, "\n")
class = 1 2 3 4 5 6 7 8 9 10
> cat("class = ", x)
class = 1 2 3 4 5 6 7 8 9 10>
> paste(1:12)
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12"
> paste("A", 1:6, sep = "")
> paste("Today is", date())
[1] "Today is Sun Sep 24 01:33:04 2006"

Same as
> cat("Today is", date())
Today is Sun Sep 24 01:35:19 2006>
> cat("Today is", date(), "\n")
Today is Sun Sep 24 01:35:43 2006
### Some Arithmetic and Statistical R Functions (built-in)

<table>
<thead>
<tr>
<th>R Function</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>log(x), log10(x), exp(x), sqrt(x)</td>
<td>ln(x), log_{10}(x), e^x, \sqrt{x}</td>
</tr>
<tr>
<td>Sin(x), cos(x), tan(x)</td>
<td>Trigonometric function</td>
</tr>
<tr>
<td>max(x), min(x), length(x), range(x)</td>
<td>Maximum, minimum, number of elements, and range of a vector</td>
</tr>
<tr>
<td>sign(x), abs(x), sort(x), sum(x), prod(x)</td>
<td>Sign, absolute value, sort in ascending order, summation, product of elements in a vector x</td>
</tr>
<tr>
<td>ceiling(x)</td>
<td>Rounds to the next higher integer</td>
</tr>
<tr>
<td>floor(x)</td>
<td>Rounds to the next lower integer</td>
</tr>
<tr>
<td>trunc(x)</td>
<td>Cuts off all digits after the decimal point</td>
</tr>
<tr>
<td>round(x), round(x, 3), round(x, -1)</td>
<td>Rounds to the nearest integer. The second argument is the number of significant number of digits desired, negative value to round large number to nearest 10 or 100, etc.</td>
</tr>
<tr>
<td>cor(x,y), mean(x), var(x), quantile(x), median(x), summary(), stem(x), hist(x)</td>
<td>Statistical function</td>
</tr>
<tr>
<td>% / %</td>
<td>Quotient (integer division), modulo function (remainder)</td>
</tr>
<tr>
<td>% %</td>
<td>% / % and % % always satisfy e1 = = ( e1 % / % e2)e2+e1 % % e2</td>
</tr>
<tr>
<td>cumsum(x), cumprod(x)</td>
<td>Returns an object which, for each element, is the sum (product) of all of the elements to that point.</td>
</tr>
<tr>
<td>gamma</td>
<td>Gamma function</td>
</tr>
</tbody>
</table>
Examples
> ceiling(2.4)  
[1] 3
> floor(2.4)  
[1] 2
> trunc(2.4)  
[1] 2
> w=2.346789
> round(w)  
[1] 2
> round(w,3)  
[1] 2.347
> round(w,-1)  
[1] 0
> w=50.34
> round(w,-1)  
[1] 50

> x<-1:5
> x%%2  # calculate the reminder from the division
[1] 1 0 1 0 1
> x%/%2  # calculate the integer of the division
[1] 0 1 1 2 2
> x/2  # give exact value of division
[1] 0.5 1.0 1.5 2.0 2.5

> cumsum(x)  
[1] 1 3 6 10 15
cumprod(x)  
[1] 1 2 6 24 120
Example 1:

```r
ceiling(c(-1.9,-1.1,1.1,1.9))
floor(c(-1.9,-1.1,1.1,1.9))
trunc(c(-1.9,-1.1,1.1,1.9))

x<-c(123456,.123456,.000123456)
round(x)

> ceiling(c(-1.9,-1.1,1.1,1.9))
[1] -1 -1  2  2
> floor(c(-1.9,-1.1,1.1,1.9))
[1] -2 -2  1  1
> trunc(c(-1.9,-1.1,1.1,1.9))
[1] -1 -1  1  1
> x<-c(123456,.123456,.000123456)
> round(x)
[1] 123456  0  0
> round(x,3)
[1] 123456.000  0.123  0.000
> round(x[1],-1)
[1] 123460
> round(x[1],-2)
[1] 123500
```

Example 2:

Calculate the sin, cosine and tangent for numbers ranging from 0 to 2*pi with distance 0.1 between them. Note that tan(x)=sin(x)/cos(x)
Calculate difference between tan(x) and sin(x)/cos(x) for the values above. Which values are exactly equal? What is the maximum differences.

Example 3:

Calculate the first 50 powers of 2, (i.e. 2*1, 2*2, 2*2*2) 
Calculate the squares of integer numbers from 1 to 50.
Which pairs are equal?(i.e 2^n=n^2). How many are there?
Sol. of Example 2:

```r
> x<-seq(0,2*pi,0.1)
> a<-sin(x)
> b<-cos(x)
> c<-tan(x)
>
> diff<-c-a/b
>
> diff==0
[1]  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE
[25]  TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE  TRUE
[37]  TRUE TRUE TRUE FALSE TRUE TRUE TRUE FALSE FALSE  TRUE TRUE FALSE
[49]  TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE  TRUE TRUE FALSE
[61]  TRUE TRUE TRUE

> length(diff)
[1] 63
> sum(diff==0)
[1] 43
> max(abs(diff))
[1] 1.421085e-14
```
Sol. of Example 3:

```r
> int<-1:50
> int
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50
> x<-2^int # first 50 power of 2
> y<-int^2 # square 1:50
> eq<-x==y # examine which pairs are equal
> eq
 [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[25] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[49] FALSE FALSE
> sum(eq) # sum of equal pairs
[1] 2
> int[eq] # list all equal pairs
[1] 2 4
> length(x[eq]) # how many such that x=y
[1] 2
```