8.1 - every language needs a way to input, output, and store data, numbers, functions (functions)

- and to determine what kind of data, list, function, ... it is

- in R, everything is stored as an "object"
  every object has a class (more generally, has attributes)

- the basic object in R is a vector
  > a <- 6
  > a
  [1] 6 # vector of length 1

  > a <- 1:6
  > a
  [1] 1 2 3 4 5 6

- vectors can be numeric, character, logical, complex, integer
  (most usual)

  a + a  # list (mixed types)

  > is.numeric(a)
  [1] TRUE

  > b <- c("hi", "lo")

  > b
  [1] "hi" "lo"
- join 2 vectors together with c(v1, v2) (concatenate)

```r
> aa <- c(a, a)
> aa
[1] 1 2 3 4 5 6 1 2 3 4 5 6
> ab <- c(a, b)
> ab
[1] "1" "2" "3" "4" "5" "6" "hi" "lo"
```

- coerced to character

```r
> as.numeric(ab)
[1] 1 2 3 4 5 6 NA NA
```

- `list(a, b)`

```
[[1]]
[[1]] 1 2 3 4 5 6 1 2 3 4 5 6
[[2]]
[1] "hi" "lo"
```

- statistical data is usually in a matrix

```
   i = 1 \ldots p
```

- subjects

```
Cases
```

```
n \uparrow
```

```
variables \uparrow
```
In R a matrix can be assigned by

\[
\text{matrix} <-
\]
\[
> \text{my.data} < - \text{matrix} (\text{data}, \text{nrows} = , \text{ncol} = )
\]

**GOTCHA**: R fills the matrix by columns

This is not usually what's wanted.

(p18) eg
\[
> \text{my.data} < - \text{matrix} (1:10, \text{nrow} = 2, \text{ncol} = 5)
\]
\[
> \text{my.data} [2, ]
\]
\[
1 3 5 7 9
\]
\[
2 4 6 8 10
\]

\[
> \text{matrix} (1:10, \text{nrow} = 2, \text{ncol} = 5, \text{byrow} = \text{T})
\]
\[
\text{nrow} = 2, \text{byrow} = \text{T}
\]
\[
\text{ncol} = 5, \text{by} = \text{T}
\]

- A data matrix usually has row & column names as well.

eg.
\[
> \text{library (MASS)}
\]
\[
> \text{data (hills)}
\]
\[
hills
\]

<table>
<thead>
<tr>
<th>dist</th>
<th>climb</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>650</td>
<td>16.083</td>
</tr>
<tr>
<td>6.0</td>
<td>2500</td>
<td>48.350</td>
</tr>
<tr>
<td>6.0</td>
<td>900</td>
<td>33.650</td>
</tr>
</tbody>
</table>

\[
> \text{dim} (\text{hills})
\]
\[
[1] \ 35 3
\]
- Data can be typed in by hand & converted to a matrix:
  `x1 <- scan()
  x2 <- scan()
  ...`  
  `> x <- matrix(c(x1, x2, ...), ncol = 20)`  
  `> x1 length 20`
  `> x2 ...`
  `> x3 ...`
  `etc.`

- Data can be read from a data file:

  `read.table(... lots of arguments)`

  Often: `read.table(file = "homework.data")` will be good enough, but if not, need

  `?read.table` or see p. 21`
and finally, most S programmers convert data matrices to data frames, a fancy matrix.

\[
\text{is.data.frame(hills)}
\]
\[
\text{attributes(hills)}
\]
\[
\text{class(hills)}
\]

\[
\text{my.frame <- data.frame(mymat)}
\]
\[
\text{names(my.frame)}
\]
\[
\text{rm(mymat, x1, x2)}
\]

\[
\text{ls()}
\]

\[
\text{my.frame}
\]

\[
\text{Data manipulation}
\]

\[
\text{hills[,1] let row all cols}
\]
\[
\text{hills[,1] let col all rows}
\]

\[
\text{hills[1:4,] let 4 rows etc.}
\]

\[
\text{and very handy hills[-1,] rows 2 through n}
\]
\[
\text{hills[-1] delete 1st column}
\]

\[
\text{GOTCHA is.matrix(hills[-3:2]) FALSE}
\]
\[
\text{hd <- as.matrix(hills)}
\]
\[
\text{is.matrix(hd[-3:-2])}
\]
\[
\text{drop = FALSE TRUE}
\]
R is very good at operating on vectors, or columns of a matrix, or cols of a data frame.

This is useful for programming; see Sort & Data Transform p. 32, 33

(skip 34-36 for now)

6.1: Linear regression

\[ y_i = \beta_0 + \beta_1 x_i + \epsilon_i \]  
\[ y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \epsilon_i \]  
\[ y = X\beta + \epsilon \]

where

- response \( y \)
- inputs, covariates, \( x \), \( \beta \)
- error \( \epsilon \)

\[ \epsilon \sim (0, \sigma^2) \]
\[ \epsilon \sim N(0, \sigma^2) \]

- aspects of analysis: set of \( \beta, \sigma^2 \); test \( H_0: \beta = 0 \), assess quality of model; decide if all
- model gives best fit
- in S the model is fit using the command \texttt{lm} (linear model).

\texttt{eg. } \texttt{lm (y ~ x1 + x2 + x3)}

- the result is an object of class \texttt{lm}

\texttt{> lm (hills \_ time ~ hills \_ dist + hills \_ climb)}

\texttt{> hills \_ lm <- \quad > \texttt{attributes (hills \_ lm)}}
\texttt{\quad > \texttt{summary (hills \_ lm)}}
\texttt{\quad > \texttt{anova (hills \_ lm)}}
\texttt{\quad > \texttt{plot (hills \_ lm)}}

\texttt{> lm (4 \_ time ~ dist + climb, data = hills)}

\texttt{Call}
\texttt{lm (formula = \quad \texttt{time ~ dist + climb}, data = hills)}

\texttt{Coefficients}
\texttt{(Intercept)} \quad \texttt{dist} \quad \texttt{climb}
\texttt{-8.992} \quad \texttt{6.21796} \quad \texttt{0.01105}
\texttt{(0.601148)} \quad \texttt{0.002051}

\textbf{Question:} Compare fits of \texttt{time ~ dist \_ time ~ dist + climb}.

Use \texttt{influence \_ lm}.
- robust regression: not sensitive to failure of normality
- resistant regression: not badly affected by outliers

\$6.3 \quad (more \, to \, come \, on \, this)\$  

including categorical/qualitative/discrete covariates 
as in \$5.6.1 \quad "analysis \, of \, variance"

- use factor variables (see p. 15)

\[ \text{factor}(a) \]
levels 1 2 3 4 5 6

- model matrix & one-way anova