```
x <- structure(x, atr1=8,atr2="test")
x
 [1]  1.3453003 -1.4395975  1.0163646 -0.6566600  0.4412399
 [6] -1.2427861  1.4967771  0.6230324 -0.5538395  1.0781191
attr(, "description"):
[1] "The unit is month"
attr(, "atr1"):
[1] 8
attr(, "atr2"):
[1] "test"
```

When an object is printed, the attributes (if any) are printed as well. To extract an attribute from an object use the functions `attributes` or `attr`. The function `attributes` returns a list of all the attributes from which you can extract a specific component.

```
attributes(x)
$description:
[1] "The unit is month"

$atr1:
[1] 8

$atr2:
[1] "test"
```

In order to get the description attribute of `x` use:

```
attributes(x)$description
[1] "The unit is month"
```

Or type in the following construction:

```
attr(x,"description")
[1] "The unit is month"
```

4.5 Character manipulation

There are several functions in R to manipulate or get information from character objects.

4.5.1 The functions `nchar`, `substring` and `paste`

```
x <- c("a","b","c")
mychar1 <- "This is a test"
mychar2 <- "This is another test"
charvector <- c("a", "b", "c", "test")
```

The function **nchar** returns the length of a character object, for example:

```
nchar(mychar1)
[1] 15
nchar(charvector)
[1] 1 1 1 4
```

The function **substring** returns a substring of a character object. For example:

```
x <- c("Gose", "Longhow", "David")
substring(x,first=2,last=4)
[1] "ose" "ong" "avi"
```

The function **paste** will paste two or more character objects. For example, to create a character vector with: "number.1", "number.2", ...,"number.10" proceed as follows:

```
paste("number",1:10, sep=".")
[1] "number.1"  "number.2"  "number.3"  "number.4"
[5] "number.5"  "number.6"  "number.7"  "number.8"
[9] "number.9"  "number.10"
```

The argument **sep** is used to specify the separating symbol between the two character objects.

```
paste("number",1:10, sep="-")
[1] "number-1"  "number-2"  "number-3"  "number-4"
[5] "number-5"  "number-6"  "number-7"  "number-8"
[9] "number-9"  "number-10"
```

Use **sep=""** for no space between the character objects.

4.5.2 Finding patterns in character objects

The functions `regexpr` and `grep` can be used to find specific character strings in character objects. The functions use so-called regular expressions, a handy format to specify search pattern. See the help for `regexpr` to find out more about regular expressions.

Let's extract the row names from our data frame 'cars'.

```
car.names <- row.names(cars)
```

We want to know if a string in 'car.names' starts with 'Volvo' and if there is, the position it has in 'car.names'. Use the function `grep` as follows:

```
grep("Volvo", car.names)
[1] 37
```

So element 37 of the `car.names` vector is a name that contains the string 'Volvo', which is confirmed by a quick check:

```
car.names[37]
[1] "Volvo 240 4"
```

To find the car names with second letter 'a', we must use a more complicated regular expression

```
tmp <- grep("^.a",car.names)
car.names[tmp]
[1] "Eagle Summit 4"   "Mazda Protege 4"
[3] "Mazda 626 4"      "Eagle Premier V6"
[5] "Mazda 929 V6"     "Mazda MPV V6"
```

For those who are familiar with wildcards (aka globbing) there is a handy function `glob2rx` that transforms a wildcard to a regular expression.

```
rg <- glob2rx("*.tmp")
rg
[1] "^.*\\.tmp$"
```

To find patterns in texts you can also use the `regexpr` function. This function also makes use of regular expressions, however it returns more information than `grep`.

```
Volvo.match <- regexpr("Volvo",car.names)
Volvo.match
 [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[19] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[37]  1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[55] -1 -1 -1 -1 -1 -1
attr(, "match.length"):
 [1] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[19] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[37]  5 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
[55] -1 -1 -1 -1 -1 -1
```

The result of `regexpr` is a numeric vector with a `match.length` attribute. A minus one means no match was found, a positive number means a match was found. In our example we see that element 37 of 'Volvo.match' equals one, which means that 'Volvo' is part of the character string in element 37 of 'car.names'. Again a quick check:

```
car.names[37]
[1] "Volvo 240 4"
```

In the above result you could immediately see that element 37 of 'car.names' is a match. If character vectors become too long to see the match quickly, use the following trick:

```
index <- 1:length(car.names)
index[Volvo.match > 0]
[1] 37
```

The result of the function `regexpr` contains the attribute match.length, which gives the length of the matched text. In the above example match Volvo consists of 5 characters. This attribute can be used together with the function `substring` to extract the found pattern from the character object.

Consider the following example which uses a regular expression, the 'match.length' attribute, and the function `substring` to extract the numeric part and character part of a character vector.

```
x <- c("10 Sept", "Oct 9th", "Jan 2", "4th of July")
w <- regexpr("[0-9]+", x)
```

The regular expression `"[0-9]+"` matches an integer.

```
w
[1] 1 5 5 1
attr(, "match.length"):
[1] 2 1 1 1

# The 1 means there is a match on position 1 of "10 Sept"
# The 5 means there is a match on position 5 of "Oct 9th"
# The 5 means there is a match on position 5 of "Jan 2"
# The 1 means there is a match on position 1 of "4th of July"
```

In the attribute `match.length` the 2 indicates the length of the match in "10 Sept".

Use the `substring` function to extract the integers. Note that the result of the substring function is of type character. To convert that to numeric, use the `as.numeric` function:

```
as.numeric(substring(x, w, w+attr(w, "match.length")-1))
[1] 10  9  2  4
```

4.5.3 Replacing characters

The functions `sub` and `gsub` are used to replace a certain pattern in a character object with another pattern.

```
mychar <- c("My_test", "My_Test_3", "_qwerty_pop_")
sub(pattern="[_]", replacement=".", x=mychar)
[1] "My.test"      "My.Test_3"    ".qwerty_pop_"
gsub(pattern="[_]", replacement=".", x=mychar)
[1] "My.test"      "My.Test.3"    ".qwerty.pop."
```

Note that by default, the `pattern` argument is a regular expression. When you want to replace a certain string it may be handy to use the `fixed` argument as well.

```
mychar <- c("mytest", "abctestabc", "test.po.test")
gsub(pattern="test", replacement="", x=mychar, fixed=TRUE)
[1] "my"     "abcabc" ".po."
```

4.5.4 Splitting characters

A character string can be split using the function `strsplit`. The two main arguments are `x` and `split`. The function returns the split results in a list, each list componenent is the split result of an element of `x`.

```
strsplit(x = c("Some text", "another string", split = NULL)
[[1]]
[1] "S" "o" "m" "e" " " "t" "e" "x" "t"

[[2]]
 [1] "a" "n" "o" "t" "h" "e" "r" " " "s" "t" "r" "i" "n" "g"
```

The argument `x` is a vector of characters, and `split` is a character vector containing regular expressions that are used for the split. If it is NULL as in the above example, the character strings are split into single characters. If it is not null, R will look at the elements in `x`, if the split string can be matched the characters left of the match will be in the output and the characters right of the match will be in the output.

```
strsplit(
   x = c("Some~text" , "another-string", "Amsterdam is a nice city"),
   split = "[~-]"
)
[[1]]
[1] "Some" "text"

[[2]]
[1] "another" "string"

[[3]]
[1] "Amsterdam is a nice city"
```

4.6 Creating factors from continuous data

The function `cut` can be used to create factor variables from continuous variables. The first argument `x` is the continuous vector and the second argument `breaks` is a vector of breakpoints, specifying intervals. For each element in `x` the function `cut` returns the interval as specified by `breaks` that contains the element.