Text Handling In R

The `cat` function

`cat` is a function which can be used to concatenate and then print character strings passed to it as arguments.

- Objects other than character strings are automatically converted to character strings by **`cat`**.
- By default, the strings being concatenated are separated by a space character. This can be overridden with the **`sep=`** argument.
- Certain sequences of characters have a special interpretation (shared with C, C++, Java and other languages). The sequence **`\n`** represents a *newline* character, **`\t`** represents a *tab* character, and **`\\`** represents a *backslash* character.

Printing

Complete S objects can be displayed by using the **`print`** function.

```
> print("A Title")
[1] "A Title"
> print(matrix(1:4,nc=2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
```

Only very basic control of formatting is possible when the **`print`** function is used.

There are other functions which can be used which can be used when finer control is necessary.

Control Of Printing

Print formating can be controlled by a number of optional arguments.

`digits=`	controls the number of digits printed to the right of the decimal point. The default is the value of the system option **`digits`**.
`quote=`	controls whether strings are printed with or without quotes. Takes the values **`TRUE`** or **`FALSE`**. The default value is **`TRUE`**.
`na.print=`	the character string used to indicate **`NA`** values. This is printed without quotes.

Examples

Here are some simple uses of **`cat`**.

```
> R2 <- .7863

> cat("aaa", "bbb", "\n")
aaa bbb

> cat("aaa", "bbb", "\n", sep="")
aaabbb

> cat("a","b","c","d","\n",sep=" - ")
a - b - c - d -
```

Examples

Here are some examples of control of the **`print`** function.

```
> print(c("Hello", "there"), quote=FALSE)
[1] Hello there

> print(1/1:5, digits=2)
[1] 1.00 0.50 0.33 0.25 0.20

> print(c(1, NA, Inf), na.print=".")
[1]   1   . Inf
```

```
> cat("R-squared =", R2, "\n")
R-squared = 0.7863
```

Vector Arguments to `cat`

When the arguments to **`cat`** are vectors, their elements are treated as though they were separate arguments.

```
> letters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i"
[10] "j" "k" "l" "m" "n" "o" "p" "q" "r"
[19] "s" "t" "u" "v" "w" "x" "y" "z"

> cat("The alphabet: ",
+     letters,"\n",sep="")
The alphabet: abcdefghijklmnopqrstuvwxyz
```

The `paste` Function

The **`paste`** function provides a very flexible way of pasting strings together. It is very useful when used in conjunction with **`cat`**.

`paste` obeys the vector recycling rule, which makes it useful for tasks like creating labels.

```
> paste("Var", 1:4)
[1] "Var 1" "Var 2" "Var 3" "Var 4"
```

In addition, **`paste`** has **`sep=`** and **`collapse=`** arguments which control how strings are glued together and make it possible to glue all the results into a single string.

Rounding

There are a variety of S functions which help to format numbers for use with **`cat`**.

The function **`round`** rounds its argument to a specified number of decimal digits.

```
> round(runif(5), 2)
[1] 0.07 0.66 0.36 0.72 0.55
```

The function **`signif`** rounds its argument to a specified number of significant digits.

```
> signif(runif(5), 2)
[1] 0.93 1.00 0.56 0.58 0.86
```

```
> paste("Var", 1:4, sep="-", collapse=":")
[1] "Var-1:Var-2:Var-3:Var-4"
```

Formatting

The R function **`formatC`** provides general formatting of numeric values as strings. The function is very flexible and provides better control than **`format`**.

`formatC` has arguments which control whether scientific or standard formatting is used. It also has control for the number of digits appearing in results.

```
> formatC(1/3, format="f", digits=4)
[1] "0.3333"

> formatC(1/3, format="e", digits=4)
[1] "3.3333e-01"
```

LaTeX Tables

The next page shows a very simple S function which can be used to print tables suitable as input for LaTeX. This is really just a sketch of how a real **`latex.table`** function could be written. Here is how it works.

```
> x
  c d
a 1 3
b 2 4

> latex.table(x)
\begin{tabular}[lrr]
  & c & d \\
a & 1 & 3 \\
```

The function provides facilities for printing numbers in European formats or for printing dollar amounts.

```
b & 2 & 4 \\
\end{tabular}
```

Source Code

```
> latex.table <-
+   function(x)
+   {
+     rlabs <- dimnames(x)[[1]]
+     clabs <- dimnames(x)[[2]]
+     fmt <- c("l", rep("r", length(clabs)))
+     cat("\\begin{tabular}[", fmt,
+         "]\n", sep="")
+     cat(" ", clabs, sep = " & ")
+     cat(" \\\\\n")
+     for(i in 1:nrow(x)) {
+       cat(rlabs[i], x[i,], sep=" & ")
+       cat(" \\\\\n")
+     }
```

```
+     cat("\\end{tabular}\n")
+   }
```

Substring Examples

```
> substr("abcdef", 2, 4)
[1] "bcd"

> substring("abcdef", 1:6, 1:6)
[1] "a" "b" "c" "d" "e" "f"

> substr(rep("abcdef",4), 1:4, 4:5)
[1] "abcd" "bcde" "cd"   "de"

> substr("abcdef", 1:4, 4:5)
[1] "abcd"

> substring("abcdef", 1:4, 4:5)
[1] "abcd" "bcde" "cd"   "de"
```

Substring Replacement

```
> x <- "123456789"
> substring(x, 2) <- "aaa"
> x
[1] "1aaa56789"

> x <- "123456789"
> substring(x, 2, 3) <- "aaa"
> x
[1] "1aa456789"
```

Generalisations

The **latex.table** function is very basic and can be enhanced in a number of ways.

- The function fails if its argument does not have row and column labels. The function should be changed so that the labels are optional, or could be specified on the command line.
- It might be useful to be able to be able to centre or left justify some columns.
- It might be useful to round or format columns differently.

Splitting Strings

```
> strsplit("hello there world", " ")
[[1]]
[1] "hello" "there" "world"

> strsplit("hello  there\tworld", " ")
[[1]]
[1] "hello"        ""
[3] "there\tworld"

> strsplit("hello  there\tworld",
+   "[[:space:]]+")
[[1]]
[1] "hello" "there" "world"
```

Text Processing

There are a number of functions which allow manipulation of character strings in R.

nchar	computes string lengths
substr	extracts substrings
substring	extracts substrings
strsplit	splits strings
tolower	converts to lower case
toupper	converts to upper case
casefold	character case conversion
chartr	carries out character conversions

Pattern Matching

The **grep** function can be used to carry out pattern matching.

```
> x <- "all the king's men"
> x <- unlist(strsplit(x, " "))

> x
[1] "all"    "the"    "king's" "men"

> grep("e", x)
[1] 2 4

> x[grep("e", x)]
[1] "the" "men"
```

Pattern Substitution

```
> x <- "The color of money colors my decision"

> sub("color", "colour", x)
[1] "The colour of money colors my decision"

> gsub("color", "colour", x)
[1] "The colour of money colours my decision"

> x <- "all the king’s men"
> x <- unlist(strsplit(x, " "))
> gsub("(.*)", "\\1=\\1", x)
[1] "all=all"       "the=the"
[3] "king’s=king’s" "men=men"
```