

from: <http://web.nps.navy.mil/~buttrey/S/text.html>

Handling Text in S+ and R

## Text Strings

A "string" is a collection of characters that make up one element of a vector. Usually, you can tell a string because it will be enclosed in (double) quotation marks. Similarly, you can construct a string by enclosing some characters in quotation marks. You may use single quotes (the character below the double quote, next to the Enter key, on most keyboards) or doubles, so this provides a natural way to include the quotation characters themselves:

```
> "This is a string"
[1] "This is a string"
> "Some strings don't have quotes"
[1] "Some strings don't have quotes"
> 'This string has "double quotes"'
[1] "This string has \"double quotes\""
```

Observe in the last example that S-Plus prints the embedded double quotes by preceding them with the slash character. That character is not part of the string; it's only visible when you print out the string.

## Special Characters in Strings

In addition to the double quote character, there are several other special (non-printing) characters that can appear in strings. The most commonly used are "\t" for TAB, "\n" for new-line, and "\" for a (single) backslash character. The nchar() function tells you how many characters are in a string; this tells that the "escaping" backslash character (that is, the one that precedes the "t" in the tab character, for example) doesn't count as a character.

```
> "Tab\t"
[1] "Tab\t"
> cat ("Tab\t") # print it formatted to the screen
Tab  >
> nchar ("Tab\t") # This string has 4, not 5, characters
```

## Vectors of Strings

A vector of strings is just like any other vector. Of course you can't do math on such a vector:

```
> str <- letters[1:5]
> str
[1] "a" "b" "c" "d" "e"
> mean (str)
```

Warning messages:

```
Warning in as.double(x): 5 missing values generated coercing from character to numeric
[1] NA
```

but you can use the usual extraction and assignment operators:

```
> str[3:4] <- c("Yes", "No")
> str
[1] "a" "b" "Yes" "No" "e"
> length(str)
[1] 5
```

## Converting Strings to and From Numeric

Quote often you'll need to convert a string or vectors into numeric values. This is easy enough with the `as.numeric()` function. Anything that isn't numeric will be turned into NA.

```
> str <- c("1", "2", "Yes", "No", "5")
> as.numeric(str)
Warning messages:
  Warning in as.double(x): 2 missing values generated coercing
from character to numeric
[1] 1 2 NA NA 5
> as.character(1:3) # conversely...
[1] "1" "2" "3"
```

If you really don't want warnings when converting to numeric, you can turn them off with the `options()` command. Make sure you know what you're doing, though.

## Pasting Strings Together

There are two important tasks when working with strings. One is putting pieces of strings together. The tool for this is the `paste()` function. This is a very powerful tool and one worth learning well. Its arguments are the vectors strings to be pasted. Shorter ones are recycled as needed:

```
> paste(c("a", "b", "c"), 1:5)
[1] "a 1" "b 2" "c 3" "a 4" "b 5"
```

Here the first argument is used up after three items, so the system returns to the first element for the last two items of the second argument. By default the separator is one space. You can specify it with the `sep=` argument:

```
> paste(c("a", "b", "c"), 1:5, sep="")
[1] "a1" "b2" "c3" "a4" "b5"
```

Here's an example of combining some numbers and some percentages. This is pretty close already:

```
> paste(1:3, c(10, 20, 30), sep=" which is ")
[1] "1 which is 10" "2 which is 20" "3 which is 30"
> paste(1:3, c(10, 20, 30), "%", sep=" which is ") # Does this work?
[1] "1 which is 10 which is %" "2 which is 20 which is %"
[3] "3 which is 30 which is %"
```

Not quite. The "%" string was replicated to length 3 to match the other strings; then the "which is" separator was added before the "%". However, this works:

```
> paste(paste(1:3, c(10, 20, 30), sep=" which is "), "%")
[1] "1 which is 10 %" "2 which is 20 %" "3 which is 30 %"
> paste(paste(1:3, c(10, 20, 30), sep=" (which is )", "%")) # neater
[1] "1 (which is 10 %)" "2 (which is 20 %)" "3 (which is 30 %)"
> hold.this <- .Last.value # save that
```

If you want to combine your vector of strings into one long string, use the `collapse=` argument. Whatever you put in there will be inserted between strings. Often you won't want anything in there:

```
> paste(hold.this, "\n", collapse="")
[1] "1 (which is 10%) \n2 (which is 20%) \n3 (which is 30%) \n"
> cat(paste(hold.this, "\n", collapse=""))
1 (which is 10%)
2 (which is 20%)
3 (which is 30%)
```

## Breaking a String at a Delimiter

The opposite of `paste()` is, perhaps not surprisingly, `unpaste()`, although this function is undocumented. If you give it a string and a delimiting character, it will return a list whose elements are the pieces of the string broken up at the locations of that character. For example:

```
>unpaste ("Nospaces", "s")
[[1]]:
[1] "No"
[[2]]:
[1] "pace"
[[3]]:
[1] ""
```

The third element gives us the characters that follow the second "s": of course, there aren't any. It's rare that you want the list in this form. Typically you'll want to use `unlist()` as well as `unpaste()`:

```
>unlist (unpaste ("Nospaces", "s"))
[1] "No" "pace" ""
```

This allows quick manipulation of certain weird strings by un-pasting and then pasting together with a different separator character. Consider this function, for example:

```
convert.delimiter <- function (string, old="_", new = ".")
{ # convert string delimited by "_" into strings delimited by "."
paste (unlist (unpaste (string, old)), collapse=new)
}
> convert.delimiter ("a_thing_with_delimiters")
[1] "a.thing.with.delimiters"
```

## Extracting Pieces of Strings

The second important task when working with strings is picking out pieces. This is accomplished with the `substring()` function. Its arguments are a vector of strings, a vector of starting numbers, and a vector of ending numbers. These latter two can be scalars, in which case they're replicated to be of the needed length. For example:

```
> st <- dimnames(state.x77)[[1]] # State names, from built-in dataset
> st[1:5]
[1] "Alabama" "Alaska" "Arizona" "Arkansas" "California"
> st <- st[1:5] # Let's just use these five for now
> substring (st, 1, 3) # Give me the first three characters from each
[1] "Ala" "Ala" "Ari" "Ark" "Cal"
> substring (st, 1:5, 3:7) # Give me 1:3 from the first, 2-4 from the second...
[1] "Ala" "las" "izo" "ana" "for"
> substring (st, nchar(st) - 2, nchar(st)) # Give me the last three
[1] "ama" "ska" "ona" "sas" "nia"
```

In that last example, we used `nchar()` to return a vector of lengths. Of course the final three characters in each name are at positions `nchar(st) - 2`, `nchar(st) - 1`, and `nchar(st)`, so we can extract the final three characters of each element in a vectorized fashion.