Statistical Computing (36-350)

Lecture 20: Basic Character Manipulation

Cosma Shalizi, with thanks to Vince Vu

4 November 2013

Agenda

- Strings and characters, what they are
- Extracting parts of strings
- Building new strings

READING: Matloff, chapter 11; *R Cookbook*, chapter 7; Spector, §7.1–7.3

Why Characters?

- Lots of data is really text: web pages, e-mail, free-form survey answers
- Even if you only care about numbers, it helps to be able to extract them from text and manipulate them easily
- Often we're interested in the text itself

A Basic Distinction, and Its Blurring

Character Symbol in a written language; stuff you can enter at a keyboard
letters, numerals, punctuation, space, newlines, etc.
`'L', 'i', 'n', 'c', 'o', 'l'`

String A sequence of characters bound together `Lincoln`

R does not distinguish between single characters and strings

```
> mode("L")
[1] "character"
> mode("Lincoln")
[1] "character"
> typeof("Lincoln")
[1] "character"
```

Making Strings

Use single or double quotes to construct a string
use nchar to get the length of a string

```
> "Lincoln"
[1] "Lincoln"
> "Abraham Lincoln"
[1] "Abraham Lincoln"
> "Abraham Lincoln's beard"
[1] "Abraham Lincoln's beard"
> 'as Lincoln said, "Fondly do we hope"'
[1] "As Lincoln said, \"Fondly do we hope\""
> nchar('As Lincoln said, "Fondly do we hope"')
[1] 36 # count it offline
```

Double quotes are preferred, single quotes mostly for quoting stuff with quotation marks

Escape Characters

Use the **escape character** \ to specify a literal, e.g., quotation marks (see last)
also used to specify special characters: `\n` for newline, `\t` for tab, etc.

`character` Data Type

One of the atomic data types, like `numeric` or `logical`
Can go into scalars, vectors, arrays, lists
or be the type of a column in a data frame

```
"Abraham Lincoln's beard"              # a character scalar
c("Abraham", "Lincoln's", "beard")     # a character vector
```

An array of characters

```
> array(state.abb, dim=c(5,10))
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "AL" "CO" "HI" "KS" "MA" "MT" "NM" "OK" "SD" "VA"
[2,] "AK" "CT" "ID" "KY" "MI" "NE" "NY" "OR" "TN" "WA"
[3,] "AZ" "DE" "IL" "LA" "MN" "NV" "NC" "PA" "TX" "WV"
[4,] "AR" "FL" "IN" "ME" "MS" "NH" "ND" "RI" "UT" "WI"
[5,] "CA" "GA" "IA" "MD" "MO" "NJ" "OH" "SC" "VT" "WY"
```

Length of a vector vs. characters in a string

```
> state.name
 [1] "Alabama"        "Alaska"        "Arizona"        "Arkansas"
 [5] "California"     "Colorado"      "Connecticut"    "Delaware"
 [9] "Florida"        "Georgia"       "Hawaii"         "Idaho"
[13] "Illinois"       "Indiana"       "Iowa"           "Kansas"
[17] "Kentucky"       "Louisiana"     "Maine"          "Maryland"
[21] "Massachusetts"  "Michigan"      "Minnesota"      "Mississippi"
[25] "Missouri"       "Montana"       "Nebraska"       "Nevada"
[29] "New Hampshire"  "New Jersey"    "New Mexico"     "New York"
[33] "North Carolina" "North Dakota"  "Ohio"           "Oklahoma"
[37] "Oregon"         "Pennsylvania"  "Rhode Island"   "South Carolina"
[41] "South Dakota"   "Tennessee"     "Texas"          "Utah"
[45] "Vermont"        "Virginia"      "Washington"     "West Virginia"
[49] "Wisconsin"      "Wyoming"
> length(state.name)
[1] 50
> nchar(state.name)
 [1]  7  6  7  8 10  8 11  8  7  7  6  5  8  7  4  6  8  9  5  8 13  8  9
[24] 11  8  7  8  6 13 10 10  8 14 12  4  8  6 12 12 14 12  9  5  4  7  8
[47] 10 13  9  7
```

Character-Valued Variables

Work just like others, e.g., with vectors:

```
> president <- "Lincoln"
> nchar(president)  # NOT 9
[1] 7
> presidents <- c("Reagan","Bush","Clinton","Cheney","Obama")
> presidents[3]
[1] "Clinton"
> presidents[-(1:3)]
[1] "Cheney" "Obama"
```

Displaying Characters

`print()` of course
`cat()` writes the string directly

```
> print("Abraham Lincoln")
[1] "Abraham Lincoln"
> cat("Abraham Lincoln")
Abraham Lincoln
> cat(presidents)
Reagan Bush Clinton Cheney Obama
```

Useful for interactive code or debugging

Whitespace

The space, " ", is a character
So is the empty string, ""
Multiple spaces in a row can be strings, " "
Newline character `\n`, tab character `\t` are also invisible but alter display
All these are **whitespace**

Substrings

Substring: a smaller string from the big string
A string is not a vector or a list, so we **cannot** use subscripts (`[[ ]]`, `[ ]`) to extract substrings
Use `substr()` instead;

```
y <- substr(x, start, stop)
```

character vector, first element (integer), last element (integer)

`substr()` vectorizes over all its arguments

```
> substr(presidents,1,2)   # First two characters
[1] "Re" "Bu" "Cl" "Ch" "Ob"
> substr(presidents,nchar(presidents)-1,nchar(presidents))   # Last two
[1] "an" "sh" "on" "ey" "ma"
> substr(presidents,20,21)    # No such substrings so return the null string
> substr(presidents,20,21)
[1] "" "" "" "" ""
> substr(presidents,7,7)      # Explain!
[1] ""  ""  "n" ""  ""
```

Substitutions

`substr` can also be used to replace

```
> substr(presidents,1,2) <- "No"
> presidents
[1] "Noagan"  "Nosh"    "Nointon" "Noeney"  "Noama"
> substr(presidents,1,2) <- c("Re","Bu","Cl","Ch","Ob")
> presidents
[1] "Reagan"  "Bush"    "Clinton" "Cheney"  "Obama"
```

like `diag` either extracts the diagonal of a matrix, or replaces it

Splitting

```
y <- strsplit(x,split)
```

Split each element of the character vector `x` at appearances of the pattern `split`
Pattern is recycled over elements of `x`
Returns list of character vectors
For today, `split` is a string
Will see later how to make the splitting pattern much more flexible than a single string

strsplit Examples

```
> strsplit("Reagan, Bush, Clinton, Cheney, Obama", split=", ")
[[1]]
[1] "Reagan"  "Bush"    "Clinton" "Cheney"  "Obama"
> strsplit("Reagan, Bush, Clinton, Cheney, Obama", split=",") # Be careful
[[1]]
[1] "Reagan"   " Bush"    " Clinton" " Cheney"  " Obama"
> strsplit("Reagan, Bush, Clinton, Cheney, Obama", split=" ") # More care
[[1]]
[1] "Reagan,"  "Bush,"    "Clinton," "Cheney,"  "Obama"
> strsplit(c("Reagan Bush Clinton Cheney Obama", "Ibrahim,Musa,Isa"),
+   split=c(" ",","))                                    # Recycling
[[1]]
[1] "Reagan"  "Bush"    "Clinton" "Cheney"  "Obama"
[[2]]
[1] "Ibrahim" "Musa"    "Isa"
```

strsplit Examples, cont'd.

```
> strsplit("Forescore and seven",split=";")                    # No split
[[1]]
[1] "Forescore and seven"
> strsplit("1863: 87", split=": ")
[[1]]
[1] "1863" "87"
> as.numeric(strsplit("1863: 87", split=": ")[[1]])
[1] 1863   87
```

Creating New Strings Automatically

Casting to type `character`: follows printing conventions

```
> as.character(7.2)                 # Obvious
[1] "7.2"
> as.character(7.2e12)              # Obvious
[1] "7.2e+12"
> as.character(c(7.2,7.2e12))       # Obvious
[1] "7.2"     "7.2e+12"
> as.character(7.2e5)               # Not quite so obvious
[1] "720000"
```

How about building *one* string from multiple parts?

formatted output, automatic generation of names, automatic generation of formulas

paste

`paste()` function — very flexible
with one vector argument, works like `as.character`:

```
> paste(41:45)
[1] "41" "42" "43" "44" "45"
```

With 2+ vector arguments, combines them with recycling:

```
> paste(presidents,41:45)
[1] "Reagan 41"  "Bush 42"    "Clinton 43" "Cheney 44"  "Obama 45"
> paste(presidents,c("R","D"))  # Not historically accurate!
[1] "Reagan R"  "Bush D"    "Clinton R" "Cheney D"  "Obama R"
> paste(presidents,"(",c("R","D"),41:45,")")
[1] "Reagan ( R 41 )"  "Bush ( D 42 )"    "Clinton ( R 43 )"
[4] "Cheney ( D 44 )"  "Obama ( R 45 )"
```

paste continued

Changing the separator between pasted-together terms:

```
> paste(presidents, " (", 41:45, ")", sep="_")
[1] "Reagan_ (_41_)"  "Bush_ (_42_)"    "Clinton_ (_43_)"
[4] "Cheney_ (_44_)"  "Obama_ (_45_)"
> paste(presidents, " (", 41:45, ")", sep="")
[1] "Reagan (41)"  "Bush (42)"    "Clinton (43)" "Cheney (44)"
[5] "Obama (45)"
```

Exercise: what happens if you give sep a vector?

A More Complicated Example of Recycling

EXERCISE: Convince yourself of why this works as it does

```
> paste(c("HW","Lab"),rep(1:11,times=rep(2,11)))
 [1] "HW 1"   "Lab 1"  "HW 2"   "Lab 2"  "HW 3"   "Lab 3"  "HW 4"
 [8] "Lab 4"  "HW 5"   "Lab 5"  "HW 6"   "Lab 6"  "HW 7"   "Lab 7"
[15] "HW 8"   "Lab 8"  "HW 9"   "Lab 9"  "HW 10"  "Lab 10" "HW 11"
[22] "Lab 11"
```

paste continued some more

Producing one big string:

```
> paste(presidents, " (", 41:45, ")", sep="", collapse="; ")
[1] "Reagan (41); Bush (42); Clinton (43); Cheney (44); Obama (45)"
```

Default value of `collapse` is `NULL`, meaning return a vector
Non-`NULL` values are used as delimiters in a single big string

A function for writing regression formulas

```
my.formula <- function(dep,indeps,df) {
  rhs <- paste(colnames(df)[indeps], collapse="+")
  return(paste(colnames(df)[dep], " ~  ", rhs, collapse=""))
}
```

e.g.,

```
> my.formula(2,c(3,5,7),df=state.x77)
[1] "Income  ~  Illiteracy+Murder+Frost"
```

Some Text

If we shall suppose that American slavery is one of those offenses which, in the providence of God, must needs come, but which, having continued through His appointed time, He now wills to remove, and that He gives to both North and South this terrible war as the woe due to those by whom the offense came, shall we discern therein any departure from those divine attributes which the believers in a living God always ascribe to Him? Fondly do we hope, fervently do we pray, that this mighty scourge of war may speedily pass away. Yet, if God wills that it continue until all the wealth piled by the bondsman's two hundred and fifty years of unrequited toil shall be sunk, and until every drop of blood drawn with the lash shall be paid by another drawn with the sword, as was said three thousand years ago, so still it must be said "the judgments of the Lord are true and righteous altogether."

More text

```
> al2 <- readLines("al2.txt")
> length(al2)
[1] 58
> head(al2)
[1] "Fellow-Countrymen:"
[2] ""
[3] "At this second appearing to take the oath of the Presidential office there i
[4] "less occasion for an extended address than there was at the first.  Then a"
[5] "statement somewhat in detail of a course to be pursued seemed fitting and"
[6] "proper.  Now, at the expiration of four years, during which public declarati
```

al2 is a vector, one element per line of text
Make one long string; split the words

```
> al2 <- paste(al2, collapse=" ")
> al2.words <- strsplit(al2, split=" ")[[1]]
> head(al2.words)
[1] "Fellow-Countrymen:" ""                    "At"
[4] "this"               "second"              "appearing"
```

Counting Words with `table`

Tabulate how often each word appears, put in order:

```
> wc <- table(al2.words)
> wc <- sort(wc,decreasing=TRUE)
> head(wc,20)
al2.words
  the    to          and    of  that   for    be    in    it     a  this
   54    26    25    24    22    11     9     8     8     8     7     7
  war which   all    by    we  with    as   but
    7     7     6     6     6     6     5     5
```

`names(wc)` gives all the distinct words in `al2.words` (**types**)
`wc` counts how often they appear (**tokens**)

Funny Stuff

The null string is the third-most-common word:

```
> names(wc)[3]
[1] ""
```

Punctuation is treated as part of words:

```
> wc["years"]
years
    3
> wc["years,"]
years,
     1
```

Capitalization:

```
> wc["that"]
that
  11
> wc["That"]
That
   1
```

All of this can be fixed if we learn how to work with text patterns and not just constants

Summary

- Text is data, just like everything else
- `substr` for extracting and substituting
- `strsplit` for turning strings into vectors
- `paste` for turning vectors into strings
- `table` for counting how many tokens belong to each type

Next time: regular expressions, for text patterns