Basics of character manipulation

Cosma Shalizi and Vincent Vu

November 7, 2011
Agenda

• Overview of character data
• Basic string operations: extract and concatenate
• Recommended reading:
  R Cookbook Chapter 7
Why?

- In many applications data comes as text – e-mail, news articles, web pages
- “Massaging” data into a form that is easier to work with – a table of numbers on a web page
Characters, strings

- **character**: symbols in a written language
  - letters in an alphabet
- **string**: a sequence of characters
- “Sherlock Holmes”
Characters, strings

- In R: no distinction between characters and strings
- but we will sometimes maintain a distinction when talking about them

```r
> mode('S')
[1] "character"
> mode('Sherlock Holmes')
[1] "character"
```
Construction

- Use single quotes or double quotes to construct a character/string
- nchar() to get the length of a string

```r
> "Sherlock Holmes"
[1] "Sherlock Holmes"
> 'Sherlock Holmes'
[1] "Sherlock Holmes"
> nchar("Sherlock Holmes")
[1] 15
```
Escape character

- Use the escape character `\` to specify a literal – e.g. quote marks

```r
> "\\"
[1] "\\"
> nchar("\\")
[1] 1
```
• *Character* values can be stored as
• scalars, vectors, arrays, or columns of a data frame, or elements of a list
• just like *numeric*
Scalar

> "California"

[1] "California"
Vector

<table>
<thead>
<tr>
<th></th>
<th>state.name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>&quot;Alabama&quot;</td>
</tr>
<tr>
<td>[5]</td>
<td>&quot;California&quot;</td>
</tr>
<tr>
<td>[9]</td>
<td>&quot;Florida&quot;</td>
</tr>
<tr>
<td>[13]</td>
<td>&quot;Illinois&quot;</td>
</tr>
<tr>
<td>[17]</td>
<td>&quot;Kentucky&quot;</td>
</tr>
<tr>
<td>[21]</td>
<td>&quot;Massachusetts&quot;</td>
</tr>
<tr>
<td>[25]</td>
<td>&quot;Missouri&quot;</td>
</tr>
<tr>
<td>[29]</td>
<td>&quot;New Hampshire&quot;</td>
</tr>
<tr>
<td>[33]</td>
<td>&quot;North Carolina&quot;</td>
</tr>
<tr>
<td>[37]</td>
<td>&quot;Oregon&quot;</td>
</tr>
<tr>
<td>[41]</td>
<td>&quot;South Dakota&quot;</td>
</tr>
<tr>
<td>[45]</td>
<td>&quot;Vermont&quot;</td>
</tr>
<tr>
<td>[49]</td>
<td>&quot;Wisconsin&quot;</td>
</tr>
<tr>
<td>[5]</td>
<td>&quot;Alaska&quot;</td>
</tr>
<tr>
<td>[9]</td>
<td>&quot;Colorado&quot;</td>
</tr>
<tr>
<td>[13]</td>
<td>&quot;Georgia&quot;</td>
</tr>
<tr>
<td>[17]</td>
<td>&quot;Indiana&quot;</td>
</tr>
<tr>
<td>[21]</td>
<td>&quot;Louisiana&quot;</td>
</tr>
<tr>
<td>[25]</td>
<td>&quot;Michigan&quot;</td>
</tr>
<tr>
<td>[29]</td>
<td>&quot;Montana&quot;</td>
</tr>
<tr>
<td>[33]</td>
<td>&quot;New Jersey&quot;</td>
</tr>
<tr>
<td>[37]</td>
<td>&quot;New Mexico&quot;</td>
</tr>
<tr>
<td>[41]</td>
<td>&quot;North Dakota&quot;</td>
</tr>
<tr>
<td>[45]</td>
<td>&quot;Ohio&quot;</td>
</tr>
<tr>
<td>[49]</td>
<td>&quot;Oregon&quot;</td>
</tr>
<tr>
<td>[5]</td>
<td>&quot;Pennsylvania&quot;</td>
</tr>
<tr>
<td>[9]</td>
<td>&quot;Tennessee&quot;</td>
</tr>
<tr>
<td>[13]</td>
<td>&quot;Texas&quot;</td>
</tr>
<tr>
<td>[17]</td>
<td>&quot;Virginia&quot;</td>
</tr>
<tr>
<td>[21]</td>
<td>&quot;Washington&quot;</td>
</tr>
<tr>
<td>[25]</td>
<td>&quot;West Virginia&quot;</td>
</tr>
<tr>
<td>[29]</td>
<td>&quot;Wisconsin&quot;</td>
</tr>
<tr>
<td>[33]</td>
<td>&quot;Wyoming&quot;</td>
</tr>
<tr>
<td>[37]</td>
<td>&quot;Wyoming&quot;</td>
</tr>
<tr>
<td>[41]</td>
<td>&quot;Wyoming&quot;</td>
</tr>
<tr>
<td>[45]</td>
<td>&quot;Wyoming&quot;</td>
</tr>
<tr>
<td>[49]</td>
<td>&quot;Wyoming&quot;</td>
</tr>
</tbody>
</table>
```r
> array(state.abb, dim=c(5,10))

```

```
[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"
```

```
List

> list("California", "Pennsylvania", "Texas")
[[1]]
[1] "California"

[[2]]
[1] "Pennsylvania"

[[3]]
[1] "Texas"
length() vs nchar()

```r
> state.name
[1] "Alabama"   "Alaska"     "Arizona"    "Arkansas"
[5] "California" "Colorado"   "Connecticut" "Delaware"
[9] "Florida"    "Georgia"    "Hawaii"     "Idaho"
[13] "Georgia"    "Indiana"    "Iowa"       "Illinois"
[17] "Iowa"       "Louisiana"  "Maine"      "Maryland"
[21] "Michigan"   "Montana"    "Minnesota"  "Mississippi"
[25] "Missouri"   "Nebraska"   "Nevada"     "New Hampshire"
[29] "New Mexico" "New York"   "New Jersey" "New Hampshire"
[33] "North Carolina" "North Dakota" "North Carolina" "Ohio"
[37] "North Dakota" "Pennsylvania" "Nebraska"   "Nevada"
[41] "Ohio"       "Pennsylvania" "Oklahoma"  " Oregon"
[45] "Pennsylvania" "Oklahoma"  "Oregon"    "Pennsylvania"
[49] "Oregon"    "Pennsylvania" "Oregon"    "Oregon"

> 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
[21] 13  8  9 11  8  7  8  6 13 10 10  8 14 12  4  8  6 12 12 14
[41] 12  9  5  4  7  8 10 13  9  7

*note that nchar() is vectorized*
Displaying characters

- Use the `cat()` function to display a character/string directly – useful for displaying messages, compare with `print()`

```r
> print("Sherlock Holmes")
[1] Sherlock Holmes
> cat("Sherlock Holmes")
Sherlock Holmes
```
Whitespace

- Space " " is a character
- Empty string "" also a character
Whitespace

- Some characters are invisible – newline "\n", tab "\t"
- Called **whitespace**
Basic operations

- Extracting substrings
- Concatenating strings
Substrings

- A string is a sequence of characters
- It is considered an atomic type in R, so we can’t use subscripts to extract subsets.
`substr()`

\[ y <- \text{substr}(x, \text{start}, \text{stop}) \]

- \texttt{x} a character vector
- \texttt{start} first element to extract (integer)
- \texttt{stop} last element to extract (integer)
- returns a character vector

*Note that `substr()` is vectorized over all arguments*
substr()
substr()

Extract last 2 characters

```r
> substr(state.name, nchar(state.name)-1, nchar(state.name))
```

```
[1] "ma" "ka" "na" "as" "ia" "do" "ut" "re" "da"
[10] "ia" "ii" "ho" "is" "na" "ga" "as" "ky" "na"
[19] "ne" "nd" "ts" "an" "ta" "pi" "ri" "na" "ka"
[28] "da" "re" "ey" "co" "rk" "na" "ta" "go" "ma"
[37] "on" "ia" "nd" "na" "ta" "ee" "as" "gh" "nt"
[46] "ia" "on" "ia" "in" "ng"
```
`substr()`

\[
\text{substr}(\text{x}, \text{start}, \text{stop}) \leftarrow \text{value}
\]

- **x** a character vector
- **start** first element to replace (integer)
- **stop** last element to replace (integer)
substr()

> x <- "Sherlock Holmes"
> substr(x, 1, 2) <- "AB"
> cat(x)
ABerlock Holmes
> substr(state.name, 1, 3) <- "dog"
> print(state.name)
[1] "dogbama"     "dogska"
[3] "dogzona"     "dogansas"
[5] "dogifornia"  "dogorado"
[7] "dognecticut" "dogaware"
Splitting strings

• Often useful to **split** a string at every occurrence of some character(s) or pattern

• Examples:
  • Comma separated list of numbers
  • Extract all words in a sentence
  • Extract sentences in a paragraph, etc...
strsplit()

\[ y \leftarrow \text{strsplit}(x, \text{split}) \]

- \( x \) character vector to be split
- \textit{split} pattern to use for splitting *
- \( y \) a list of the same length as \( x \), containing the splits

* we’ll see later that a regexp can be used here
strsplit()

> strsplit("Sherlock Holmes is the world's greatest detective", " ")
[[1]]
[1] "Sherlock" "Holmes" "is" "the" "world's" "greatest"
[7] "detective"
strsplit()

> fruits <- c(
"apples and oranges and pears and bananas",
"pineapples and mangos and guavas"
)
> strsplit(fruits, " and ")
[[1]]
[1] "apples" "oranges" "pears" "bananas"

[[2]]
[1] "pineapples" "mangos" "guavas"
strsplit()

> numbers <- c("3431, 49, 291, 811, 984")
> strsplit(numbers, ",")

[[1]]
[1] "3431" 49 291 811 984

> as.numeric( strsplit(numbers, ",")[[1]] )
[1] 3431 49 291 811 984
Concatenating strings

- Create a new string by pasting together individual strings
- Many uses
  - formatting data for output (to the display or a file)
  - creating generic names by adding a numeric suffix – HW1, HW2, HW3, ...
`paste()`

```r
y <- paste(..., sep = " ", collapse = NULL)
```

- ... 1 or more R objects to be converted to character vectors
- `sep` string to separate terms
- `collapse` optional string to separate results
paste()

> paste('Vincent', 'Vu')
[1] "Vincent Vu"
paste()

> paste('Vincent', 'Vu')
[1] "Vincent Vu"

> paste('Homework', 1)
[1] "Homework 1"
paste()

> paste('HW', 1:10)
[1] "HW 1"  "HW 2"  "HW 3"  "HW 4"  "HW 5"
  "HW 6"  "HW 7"  "HW 8"  "HW 9"  "HW 10"
> paste('HW', 1:10)
[1] "HW1"  "HW2"  "HW3"  "HW4"  "HW5"
"HW6"  "HW7"  "HW8"  "HW9"  "HW10"

> paste('HW', 1:10, sep = '')
[1] "HW1"  "HW2"  "HW3"  "HW4"  "HW5"
"HW6"  "HW7"  "HW8"  "HW9"  "HW10"
I am honored to be with you today at your commencement from one of the finest universities in the world. I never graduated from college. Truth be told, this is the closest I've ever gotten to a college graduation. Today I want to tell you three stories from my life. That's it. No big deal. Just three stories.

The first story is about connecting the dots.

I dropped out of Reed College after the first 6 months, but then stayed around as a drop-in for another 18 months or so before I really quit. So why did I drop out?
I am honored to be here with you today at your commencement ceremony. One of the finest universities in the world. I never graduated from college – Truth be told, I was even given a college education. Today I want to tell you three stories from my life that have long ago made a difference in mine.
sj is a character vector – each element corresponds to a line in the text file ‘stevejobs.txt’

```r
> sj <- readLines('stevejobs.txt')
> head(sj)
[1] "I am honored to be with you today at your commencement from one of the finest"
[2] "universities in the world. I never graduated from college. Truth be told, this"
[3] "is the closest I've ever gotten to a college graduation. Today I want to tell"
[5] ""
[6] "The first story is about connecting the dots."
```
Make one long string: sj.all
Split the string: sj.words

> sj <- readLines('stevejobs.txt')
> sj <- paste(sj, collapse = ' ')
> sj.words <- strsplit(sj, split = ' ')[[1]]
> head(sj.words)
[1] "I" "am" "honored" "to" "be"
[6] "with"
Tabulate the strings in sj.words and then sort the table

```r
> sj <- readLines('stevejobs.txt')
> sj <- paste(sj, collapse = ' ')
> sj.words <- strsplit(sj, split = ' ')[[1]]
> length(sj.words)
[1] 2281
> wc <- table(sj.words)
> head(sort(wc, decreasing = T), 20)
     sj.words
    the    I   to  and  was    a   of  that   in   is   it
     91    86  71   49   47   46   40   38   33   29   28
   you    my  had  with  And  for  have  It
    27    26  25   22   18   17  17  17  17
```
> sj <- readLines('stevejobs.txt')
> sj <- paste(sj, collapse = ' ')
> sj.words <- strsplit(sj, split = ' ')[[1]]
> length(sj.words)
[1] 2281
> wc <- table(sj.words)
> head(sort(wc, decreasing = T), 20)

```
sj.words
the    I   to  and  was    a   of  that   in   is  
91     86   71  49  47  46  40  38  33   29
you        my  had with  And  for have   It
27     26   25  22  18  17  17  17  17  17
```
Tabulate the strings in sj.words and then sort the table

`> sj <- readLines('stevejobs.txt')`
`> sj <- paste(sj, collapse = ' ')`
`> sj.words <- strsplit(sj, split = ' ')[[1]]`
`> length(sj.words)`

[1] 2281

`> wc <- table(sj.words)`

`> head(sort(wc, decreasing = T), 20)`

<table>
<thead>
<tr>
<th>sj.words</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>91</td>
<td>86</td>
<td>71</td>
<td>49</td>
<td>47</td>
<td>46</td>
<td>40</td>
<td>38</td>
<td>33</td>
<td>29</td>
<td>28</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>was</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>of</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>that</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>in</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>is</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>it</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>you</td>
<td>27</td>
<td>26</td>
<td>25</td>
<td>22</td>
<td>18</td>
<td>17</td>
<td>16</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>my</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>had</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>And</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>have</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>It</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

‘And’ & ‘and’ are considered different
We will improve on this over the next few lectures
Summary

• Text is data
• `substr()`, `strsplit()`, and `paste()`
• `table()` can be used to tabulate word counts
• Next: Regular expressions – expressive ‘language’ for generating search patterns
Bonus material
Character encoding

• Computers represent information using patterns of 0s and 1s (bits)

• **character encoding**: rule for mapping characters of a written language into a set of binary codes – e.g. ASCII, UTF-8
Character encoding

- ASCII – character encoding scheme based on English alphabet
- established in 1963
- fixed width (8 bits = 1 byte)
- 95 printable characters
Character encoding

• UTF-8 — multibyte character encoding scheme based on Unicode (standard for representing text in most of the world’s writing systems)
  • established
  • variable width (1 to 6 bytes)
  • ~1 million characters
Details aside, UTF-8 allows us to deal with text from almost all languages and alphabets.

In R, `locale` determines the character encoding scheme.