R

Strings, Performance, Misc

Base R String Functions

- R has limited support for text processing
  - If this is the main purpose of a project, think about using another language
- Just like other functions in R, the string functions operate on vectors
- Common string functions
  - strsplit
  - grep/ grepl
  - nchar
  - toupper / tolower
  - substr

In [ ]:

```
print(nchar(c("I'm a little teapot","short and stout")))
print(nchar(c("I'm a little teapot", 14)))
print(nchar("I the only string"))
```

In [ ]:

```
str_vector <- c("I'm a little teapot","short and stout",14,
                FALSE)
print(toupper(str_vector))
print(tolower(str_vector))
```

Substring in R

- `substr` and `substring` take in 3 arguments, any of which can be vectors

```
substr(strings, start, end)
  substring(strings, first, last)
```

- If start or end is longer than the other, the values of the shorter one are recycled
  - Only `substring` repeats the strings

In [ ]:
```
print(substr("Hello World",3,5))
```

In [ ]:

```
print(substr("Hello World",1:3,1:3))
print(substring("Hello World",1:3,1:3))
print(substring("Hello World",c(1,2,3),c(1,2,3)))
print(substring("Hello World",5:20,10))
print(substring("Hello World",4,10:15))
```

In [ ]:

```
str_vector <- c("I'm a little teapot","short and stout",14,FALSE)
print(substr(str_vector,2,1000L))
cat("\n")
print(substr(str_vector,1:5,1000L))
cat("\n")
```

In [ ]:

```
print(substring(str_vector,1:5,1000L))
cat("\n")
print(substring(str_vector,1:15,1000))
```

Regex in R

- Both strsplit as well as grep and grepl can take regular expressions
  - By default, these are POSIX style regular expressions
  - Pass perl=TRUE to use PCRE
- grep returns the indexes in the vector the match was found at
- grepl returns a logic vector indicating if an element of the vector matched

In [ ]:

```
strings_with_spaces <- c("I am a string",
                         "I am one too",
                         "This also has spaces")
print(strsplit(strings_with_spaces,split=' '))
```

In [ ]:

```
strings_with_spaces <- c("I am a string",
                         "I am one too",
                         "This also has spaces")
print(strsplit(strings_with_spaces,split="\\s",perl=TRUE))
```

In [ ]:

```
strings_with_spaces <- c("I am a string","I am one too","This also has spaces")
print(strsplit(strings_with_spaces,split="\\W",perl=TRUE))
```

In [ ]:

```
strings_with_spaces <- c("I am a string",
                         "I am one too",
                         "This also has spaces")
idx <- grep('I',strings_with_spaces,perl=TRUE)
print(strings_with_spaces[idx])
```

In [ ]:
```
grep('I',strings_with_spaces,perl=TRUE,ignore.case=TRUE)
```

In [ ]:
```
grep('\\bI\\b',strings_with_spaces,perl=TRUE,ignore.case=TRUE)
```

In [ ]:
```
grepl('\\bI\\b',strings_with_spaces,perl=TRUE,ignore.case=TRUE)
```

The StringR library

- StringR is based on an older library, called stringi
- The aim is to
  - improve consistency in function calls
  - make common string manipulation tasks easy
- Has robust multilingual support

In [ ]:
```
library(stringr)
```

In [ ]:
```
print(str_length(str_vector))
```

In [ ]:
```
print(str_sort(str_vector))
```

In [ ]:
```
print(str_to_title(str_vector))
```

In [ ]:
```
print(str_pad(str_vector,40))
```

In [ ]:

```
str_vector <- c("\n\rI am a string\t\t",
                "I am one\ntoo",
                "This also has spaces")
print(str_trim(str_pad(str_vector,40)))
```

In [ ]:
```
str_c(str_vector,",")
```

In [ ]:
```
str_c(str_vector,collapse=", ")
```

In [ ]:
```
str_detect(str_vector,'o')
```

In [ ]:
```
str_count(str_vector,'o')
```

Directory Traversal in R

- Most scripting languages provided an easy way to iterate over files in a directory
  - This is known as globbing
  - It also allows wildcards to be used
- In R, the function is `Sys.glob` (note the uppercase)
  - Rather than returning an iterator, it returns a vector containing all the file names

In [ ]:
```
print(Sys.glob("*.html"))
```

The readr package

- As an alternative to built-in data loading functions, some people use the readr package
  - I find the built in functions good enough usually
- readr provides the `read_file` and `write_file` functions
  - These read or write an entire file into a string ,or vice versa
  - This is possible in base R, but cumbersome, because you must calculate the length of the string first

In [ ]:
```
library(readr)
```

In [ ]:
```
contents <- read_file("index.html")
```

In [ ]:
```
print(contents)
```

In [ ]:
```
print(str_extract_all(contents,'<a href=".*?">.*</a>'))
```

Performance in R

- R is commonly viewed as a slow language
  - Mostly because it is
- We can still optimized and make sure to program in an R style
  - Avoid for loops if you can use a vectorized function
  - S4 methods are slower than S3, which is slower than a direct function call
  - Consider bytecode compilation

Profiling your code

- The `microbenchmark` library provides the `microbenchmark` function
  - Takes in several functions, runs them all, and prints statistics
- For line-by-line profiling, use the `profvis` package
  - Uses a web browser to show results

In [ ]:

```
library(microbenchmark)
nums <- matrix(c(1:5000),nrow=100)
print(
    microbenchmark(
    colMeans(nums),
    apply(nums,2,mean)
    )
)
```

In [ ]:

```
## Needs to be run in RStudio
library(profvis)
print(
    profvis(
        {
    nums <- matrix(c(1:50000),nrow=100)
    apply(nums,2,mean)
}
    ))
```

Parallelism

- Because of its functional design, R is a perfect language for parallelization
- The library `parallel` provides a mutlicore versions of mapply and lapply,
  - `mclapply`
  - `mcmapply`

```
mclapply(vector,function,mc.cores=N_CORES)
    mclapply(vector,function,axis,mc.cores=N_CORES)
```

In [ ]:
```
library(parallel)
print(detectCores())
```

In [ ]:
```
seed_strings <- c("asdf","ghhjk",'qerwet',
                  'uopi','zxcv','asdgf')
lots_of_strings <- rep(seed_strings,20000)
print(
    microbenchmark(
        lapply(lots_of_strings,str_length),
        mclapply(lots_of_strings,str_length,mc.cores=7)
    )
)
```

In [ ]:

```
cl <- makeCluster(8)
print(
    microbenchmark(
        colMeans(nums),
        apply(nums,2,mean),
        parCapply(cl,nums,mean)
        )
    )
stopCluster(cl)
```

Presenting Data

- R is often used in the analysis phase of research
  - Especially to produce nice graphics
- Packages exist that allow papers to be written in R, combined with code
  - knitr is a very popular one

KnitR

- `knitR` allows a document to be written in
  - R-style Markdown
  - HTML
  - LaTeX
- R code is set off in these documents using various conventions
- Code is executed and results displayed inline correctly

In [ ]:

```
library(knitr)
knit('005-latex.Rtex')
```

Loading Libraries from Non Default Locations

- By default, R tries to install and looks for packages in a location that needs sudo access to write
- You can change where libraries are installed by adding the `lib` parameter to `install.packages`
- There are numerous ways to tell where to look for libraries, including in the `library` function
  - The most consistent way is to set the environmental variable `R_LIBS_USER` in your shell before calling R