R Syntax 1: Data Types and Vector

Pilsung Kang
School of Industrial Management Engineering
Korea University

String Processing

- The length of a string
  - ✓ Use nchar( ) function instead of length( )
    - Space and special characters can be counted as well
- Concatenate strings
  - ✓ Use paste( ) function
    - Various spacing strategies can be used
    - Non-character values are also possible

```
Console ~/
> S <- "Welcome to Data Science!"
> length(S)
[1] 1
> nchar(S)
[1] 24
> S1 <- "My name is"
> S2 <- "Pilsung Kang"
> paste(S1, S2)
[1] "My name is Pilsung Kang"
> paste(S1, S2, sep="-")
[1] "My name is-Pilsung Kang"
> paste(S1, S2, sep="")
[1] "My name isPilsung Kang"
>
```

```
Console ~/
> paste("The value of log10 is", log(10))
[1] "The value of log10 is 2.30258509299405"
> S1 <- c("My name is", "Your name is")
> S2 <- c("Pilsung")
> S3 <- c("Pilsung", "Younho", "Hakyeon")
> paste(S1,S2)
[1] "My name is Pilsung"   "Your name is Pilsung"
> paste(S1,S3)
[1] "My name is Pilsung"  "Your name is Younho" "My name is Hakyeon"
> stooges <- c("Dongmin", "Sangkyum", "Junhong")
> paste(stooges, "loves", "R.")
[1] "Dongmin loves R."  "Sangkyum loves R." "Junhong loves R."
> paste(stooges, "loves", "R", collapse = ", and ")
[1] "Dongmin loves R, and Sangkyum loves R, and Junhong loves R"
>
```

String Processing

- Extract sub-strings
  - ✓ Use substring(string, start, end) function
    - Extract the substring that begins with "start" and ends with "end"
    - If the string argument is a vector, the other options are applied to all elements

```
Console ~/
> substr("Data Science", 1, 4)
[1] "Data"
> substr("Data Science", 6, 10)
[1] "Scien"
> stooges <- c("Dongmin", "Sangkyum", "Junhong")
> substr(stooges, 1,3)
[1] "Don" "San" "Jun"
> cities <- c("New York, NY", "Los Angeles, CA", "Peoria, IL")
> substr(cities, nchar(cities)-1, nchar(cities))
[1] "NY" "CA" "IL"
>
```

String Processing

- Split text
  - ✓ Use strsplit(string, separator) function
    - A simple string or regular expression can be used as a separator
    - Ex: split the file path using "/" as a separator

```
Console ~/
> path <- "C:/home/mike/data/trials.csv"
> strsplit(path,"/")
[[1]]
[1] "C:"          "home"        "mike"        "data"        "trials.csv"
```

```
Console ~/
> path <- c("C:/home/mike/data/trials.csv",
+ "C:/home/mike/data/errors.txt",
+ "C:/home/mike/data/report.doc")
> strsplit(path,"/")
[[1]]
[1] "C:"          "home"        "mike"        "data"        "trials.csv"

[[2]]
[1] "C:"          "home"        "mike"        "data"        "errors.txt"

[[3]]
[1] "C:"          "home"        "mike"        "data"        "report.doc"
```

String Processing

- Regular expression
  - ✓ a sequence of characters that define a search pattern
  - ✓ this pattern is then used by string searching algorithms

```
Console ~/
> strsplit(path, "om")
[[1]]
[1] "c:/h"                    "e/mike/data/trials1.csv"

[[2]]
[1] "c:/h"                    "e/mike/data/errors2.txt"

[[3]]
[1] "c:/h"                    "e/mike/data/report3.doc"

> strsplit(path, "[hm]")
[[1]]
[1] "c:/"                  "o"                    "e/"                   "ike/data/trials1.csv"

[[2]]
[1] "c:/"                  "o"                    "e/"                   "ike/data/errors2.txt"

[[3]]
[1] "c:/"                  "o"                    "e/"                   "ike/data/report3.doc"

> strsplit(path, "i.e")
[[1]]
[1] "c:/home/m"         "/data/trials1.csv"

[[2]]
[1] "c:/home/m"         "/data/errors2.txt"

[[3]]
[1] "c:/home/m"         "/data/report3.doc"
```

```
Console ~/
> strsplit(path, "\\.")
[[1]]
[1] "c:/home/mike/data/trials1" "csv"

[[2]]
[1] "c:/home/mike/data/errors2" "txt"

[[3]]
[1] "c:/home/mike/data/report3" "doc"

> strsplit(path, "r{2}")
[[1]]
[1] "c:/home/mike/data/trials1.csv"

[[2]]
[1] "c:/home/mike/data/e" "ors2.txt"

[[3]]
[1] "c:/home/mike/data/report3.doc"

> strsplit(path, "[[:digit:]]")
[[1]]
[1] "c:/home/mike/data/trials" ".csv"

[[2]]
[1] "c:/home/mike/data/errors" ".txt"

[[3]]
[1] "c:/home/mike/data/report" ".doc"
```

Regular expression in R

String Processing

- Regular expression

POSIX	비표준	펄/Tcl	Vim	ASCII	설명
[:alnum:]				[A-Za-z0-9]	영숫자
	[:word:]	₩w	₩w	[A-Za-z0-9_]	영숫자 + "_"
		₩W	₩W	[^A-Za-z0-9_]	낱말이 아닌 문자
[:alpha:]			₩a	[A-Za-z]	알파벳 문자
[:blank:]			₩s	[₩t]	공백과 탭
		₩b	₩< ₩>	(?<=₩W)(?=₩w)|(?<=₩w)(?=₩W)	낱말 경계
[:cntrl:]				[₩x00-₩x1F₩x7F]	제어 문자
[:digit:]		₩d	₩d	[0-9]	숫자
		₩D	₩D	[^0-9]	숫자가 아닌 문자
[:graph:]				[₩x21-₩x7E]	보이는 문자
[:lower:]			₩l	[a-z]	소문자
[:print:]			₩p	[₩x20-₩x7E]	보이는 문자 및 공백 문자
[:punct:]				[]["#$%&'()*+,./:;<=>?@₩^_`{|}~-]	구두점
[:space:]		₩s	₩_s (단순히 줄 끝에 추가)	[₩t₩r₩n₩v₩f]	공백 문자
		₩S		[^ ₩t₩r₩n₩v₩f]	공백이 아닌 모든 문자
[:upper:]			₩u	[A-Z]	대문자
[:xdigit:]			₩x	[A-Fa-f0-9]	16진수

From: wikipedia

String Processing

- Substitution
  - ✓ Use sub(old, new, string) or gsub(old, new, string) functions
  - ✓ sub( ) replaces the first substring whereas gsub( ) replaces all substrings

```
Console ~/
> tmpstring <- "Kim is stupid and Kang is stupid too"
> sub("stupid", "smart", tmpstring)
[1] "Kim is smart and Kang is stupid too"
> gsub("stupid", "smart", tmpstring)
[1] "Kim is smart and Kang is smart too"
```

- String pattern matching
  - ✓ Use grep(pattern, x) function
    - Return the index that matches pattern

```
Console ~/
> grep("mike",path)
[1] 1 2 3
> grep("errors",path)
[1] 2
```


ANY
questions?