Note 12. Character String

Introduction to Statistical Programming

Chanmin Kim

Department of Statistics
Sungkyunkwan University

2022 Spring

Character String

- Character string: Vector with character elements.
- `paste('char','char',...,sep='')`: Concatenating strings.
- `strsplit(`*`vector`*`,`*`'separator'`*`)`: Splitting strings by separator.
- `grep(`*`pattern,x,value=T/F`*`)`: Matching patterns.
  - *value*=*F*: returns a vector of the indexes of the elements of `x` that yielded a match.
  - *value*=*T*: returns a character vector containing the selected elements of `x`.

Character String

```
> x <- paste('KimCM','KimJY','KimHY',sep='/')
> x
[1] "KimCM/KimJY/KimHY"

> y <- strsplit(x,'/')
> y
[[1]]
[1] "KimCM" "KimJY" "KimHY"

> x <- c('KimCM','LeeJM','KimEH')
> grep('Kim',x,value=F)
[1] 1 3
> grep('Kim',x,value=T)
[1] "KimCM" "KimEH"
```

NCHAR() & SPRINTF()

- nchar(): Length of a string.

```
> nchar(x)
[1] 5 5 5

> x <- 'SKKU-STAT.  '
> nchar(x)
[1] 12
```

- sprintf(): It combines strings in the formatted manner.
  - %d or %i: Integer value.
  - %f: Double precision value with decimal point (e.g., %.3f: Decimal place is 3 (default 6)).
  - %e or %E: Double precision value in exponential decimal notation.
  - %s: Character string.

SPRINTF()

```
> sprintf('%d', 10)
[1] "10"
> sprintf('%i', 15)
[1] "15"
> sprintf('%d', 3.2)
Error in sprintf("%d", 3.2) :
  invalid format '%d'; use format %f, %e, %g or %a for numeric objects

> sprintf('%f', pi)
[1] "3.141593"
> sprintf('%.3f', pi)
[1] "3.142"
> sprintf('%1.0f', pi)
[1] "3"
> sprintf('%5.1f', pi)
[1] "  3.1"
> sprintf('%05.1f', pi)
[1] "003.1"
> sprintf('% f', pi)
[1] " 3.141593"
```

SPRINTF()

```
> sprintf('%e', pi)
[1] "3.141593e+00"
> sprintf('%E', pi)
[1] "3.141593E+00"

> x <- 'abc'
> sprintf('%s', x)
[1] "abc"

> sprintf('%s is %.5f', 'pi',pi )
[1] "pi is 3.14159"

> x <- 3
> sprintf('The square of %d is %d',x,x^2)
[1] "The square of 3 is 9"

> z <- 23.5
> sprintf('Today temperature is %2.1f degree',z)
[1] "Today temperature is 23.5 degree"
```

SUBSTR() & REGEXPR()

- substr(*'string', start, stop*): It returns the substring in the given character position range *start:stop*.

```
> substr('Kim CM',1,3)
[1] "Kim"
> x <- 'SKKU STAT.'
> substr(x,6,9)
[1] "STAT"
```

- regexpr(*'pattern', 'string'*): It returns the character position of the first instance of *pattern* in *string*.

```
> regexpr('to','Top to bottom')
[1] 5
attr(,"match.length")
[1] 2
attr(,"useBytes")
[1] TRUE
```

GREGEXPR()

- gregexpr(*'pattern', 'string'*): The same as regexpr(), but it finds all instances of *pattern* and returns a list object.

```
> gregexpr('iss','Mississippi')
[[1]]
[1] 2 5
attr(,"match.length")
[1] 3 3
attr(,"useBytes")
[1] TRUE

> x = gregexpr('iss','Mississippi')
> x[[1]]+3
[1] 5 8
attr(,"match.length")
[1] 3 3
attr(,"useBytes")
[1] TRUE
```

Regular Expression

- Regular expression:
  - A kind of wild card.
  - Shorthand to specify broad classes of strings.
  - It is used in pattern matching functions such as `grep()`, `regexpr()`, `gregexpr()`, `strsplit()`.

```
> grep('[kKe]',c('Kim CM','Choi HE','Park JH', 'Lee PY'))
[1] 1 3 4
> # [kKe]: 'k' or 'K' or 'e'

> grep('a.e',c('place','pitcher','ace','catcher'))
[1] 1 3
> # a.e: a (any character) e

> grep('c..r',c('place','pitcher','ace','catcher'))
[1] 2 4
> # c..r: c (any character) (any character) r
```

Regular Expression

- ```
  > grep('.',c('abc','de','f.g','h.jk'))
  [1] 1 2 3 4
  > # fail to find 'f.g' and 'h.jk' because '.' is metacharacter.

  > grep('\\.',c('abc','de','f.g','h.jk'))
  [1] 3 4
  > # To escape the nature of metacharacter, use \\.
  ```
  ```

Example

- Testing a file name for a given extension.

```
> extension <- function(fn,ext)
+ # fn: file name
+ # ext: file extension
+ {
+   fn1 <- strsplit(fn,'\\.')
+   ext1 <- length(fn1[[1]])
+   return(fn1[[1]][ext1] == ext)
+ }

> wd <- dir()   # all file names in working directory
> file <- NULL
> for (i in 1:length(wd))
+ {
+   if (extension(wd[i],'R')) file = c(file,wd[i])
+ }
> file
[1] "Ch12.R"
```