Datadamp_Intermediate_R___Utilities

dizhen

2019/4/3

Data Utilities

seq(): Generate sequences, by specifying the from, to, and by arguments.

rep(): Replicate elements of vectors and lists.

sort(): Sort a vector in ascending order. Works on numerics, but also on character strings and logicals.

rev(): Reverse the elements in a data structures for which reversal is defined.

str(): Display the structure of any R object.

append(): Merge vectors or lists.

is.*(): Check for the class of an R object.

as.*(): Convert an R object from one class to another.

unlist(): Flatten (possibly embedded) lists to produce a vector.

```
v1 <- c(1.1, -7.1, 5.4, -2.7)
v2 <- c(-3.6, 4.1, 5.8, -8.0)
mean(c(sum(round(abs(v1))), sum(round(abs(v2)))))
```

```
## [1] 19
```

```
li <- list(log = TRUE,
  ch = "hello",
  int_vec = sort(rep(seq(8, 2, by = -2), times = 2)))

sort(rep(seq(8, 2, by = -2), times = 2))
```

```
## [1] 2 2 4 4 6 6 8 8
```

```
is.list(li)
```

```
## [1] TRUE
```

```
is.list(c(1, 2, 3))
```

```
## [1] FALSE
```

```
li2 <- as.list(c(1, 2, 3))
is.list(li2)
```

```
## [1] TRUE
```

```r
unlist(li)
```

```
##      log       ch int_vec1 int_vec2 int_vec3 int_vec4 int_vec5 int_vec6
##   "TRUE"  "hello"      "2"      "2"      "4"      "4"      "6"      "6"
## int_vec7 int_vec8
##      "8"      "8"
```

```r
str(append(li, rev(li)))
```

```
## List of 6
##  $ log    : logi TRUE
##  $ ch     : chr "hello"
##  $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
##  $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
##  $ ch     : chr "hello"
##  $ log    : logi TRUE
```

```r
str(rev(li))
```

```
## List of 3
##  $ int_vec: num [1:8] 2 2 4 4 6 6 8 8
##  $ ch     : chr "hello"
##  $ log    : logi TRUE
```

Practice

```r
# The linkedin and facebook lists have already been created for you
linkedin <- list(16, 9, 13, 5, 2, 17, 14)
facebook <- list(17, 7, 5, 16, 8, 13, 14)

# Convert linkedin and facebook to a vector: li_vec and fb_vec
# unlist(): Flatten (possibly embedded) lists to produce a vector.
li_vec <- unlist(linkedin)
fb_vec <- unlist(facebook)

# Append fb_vec to li_vec: social_vec
# append(): Merge vectors or lists.
social_vec <- append(li_vec, fb_vec)

# Sort social_vec
sort(social_vec, decreasing = TRUE)
```

```
##  [1] 17 17 16 16 14 14 13 13  9  8  7  5  5  2
```

```r
# Fix me
# seq(rep(1, 7, by = 2), times = 7)

rep(seq(1, 7, by = 2), times = 7)
```

```
##  [1] 1 3 5 7 1 3 5 7 1 3 5 7 1 3 5 7 1 3 5 7 1 3 5 7 1 3 5 7
```

```r
# Create first sequence: seq1
seq1 <- seq(1, 500, by = 3)

# Create second sequence: seq2
seq2 <- seq(1200, 900, by = -7)

# Calculate total sum of the sequences
sum(c(seq1, seq2))
```

```
## [1] 87029
```

Regular Expression

Sequence of (meta) characters

Pattern existence

Pattern replacement

Pattern extraction

grep(), grepl()

sub(), gsub()

```r
animals <- c("cat", "moose", "impala", "ant", "kiwi")

# grepl(pattern = <regex>, x = <string>)
grepl(pattern = "a", x = animals)
```

```
## [1]  TRUE FALSE  TRUE  TRUE FALSE
```

```r
grepl(pattern = "^a", x = animals) # nothing before "a"
```

```
## [1] FALSE FALSE FALSE  TRUE FALSE
```

```r
grepl(pattern = "a$", x = animals) # nothing behind "a"
```

```
## [1] FALSE FALSE  TRUE FALSE FALSE
```

```r
# ?regex
```

```r
# grep()

grep(pattern = "a", x = animals)
```

```
## [1] 1 3 4
```

```r
which(grepl(pattern = "a", x = animals))
```

```
## [1] 1 3 4
```

```r
grep(pattern = "^a", x = animals)
```

```
## [1] 4
```

```r
# sub(), gsub()

# sub(pattern = <regex>, replacement = <str>, x = <str>)
sub(pattern = "a", replacement = "o", x = animals) # replace first one matched
```

```
## [1] "cot"    "moose"  "impola" "ont"    "kiwi"
```

```r
gsub(pattern = "a", replacement = "o", x = animals) # replace all matched
```

```
## [1] "cot"    "moose"  "impolo" "ont"    "kiwi"
```

```r
gsub(pattern = "a|i", replacement = "_", x = animals)
```

```
## [1] "c_t"    "moose"  "_mp_l_" "_nt"    "k_w_"
```

```r
gsub(pattern = "a|i|o", replacement = "_", x = animals)
```

```
## [1] "c_t"    "m__se"  "_mp_l_" "_nt"    "k_w_"
```

Practice

grepl & grep

grepl(), which returns TRUE when a pattern is found in the corresponding character string.

grep(), which returns a vector of indices of the character strings that contains the pattern.

```r
# The emails vector has already been defined for you
emails <- c("john.doe@ivyleague.edu", "education@world.gov", "dalai.lama@peace.org",
            "invalid.edu", "quant@bigdatacollege.edu", "cookie.monster@sesame.tv")

# Use grepl() to match for "edu"
print(grepl(pattern = 'edu', x = emails))
```

```
## [1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE
```

```r
# Use grep() to match for "edu", save result to hits
hits = grep(pattern = 'edu', x = emails)

# Subset emails using hits
emails[hits]
```

```
## [1] "john.doe@ivyleague.edu"   "education@world.gov"
## [3] "invalid.edu"              "quant@bigdatacollege.edu"
```

You can use the caret, ˆ, and the dollar sign, $ to match the content located in the start and end of a string, respectively. This could take us one step closer to a correct pattern for matching only the ".edu" email addresses from our list of emails. But there's more that can be added to make the pattern more robust:

@, because a valid email must contain an at-sign.

.*, which matches any character (.) zero or more times (*). Both the dot and the asterisk are metacharacters. You can use them to match any character between the at-sign and the ".edu" portion of an email address.

\.edu$, to match the ".edu" part of the email at the end of the string. The \ part escapes the dot: it tells R that you want to use the . as an actual character.

```r
# The emails vector has already been defined for you
emails <- c("john.doe@ivyleague.edu", "education@world.gov", "dalai.lama@peace.org",
            "invalid.edu", "quant@bigdatacollege.edu", "cookie.monster@sesame.tv")

# Use grepl() to match for .edu addresses more robustly
grepl(pattern = "@.*\\.edu$", x = emails)
```

```
## [1]  TRUE FALSE FALSE FALSE  TRUE FALSE
```

```r
# Use grep() to match for .edu addresses more robustly, save result to hits
hits = grep(pattern = '@.*\\.edu$', x = emails)

# Subset emails using hits
emails[hits]
```

```
## [1] "john.doe@ivyleague.edu"   "quant@bigdatacollege.edu"
```

sub & gsub

While grep() and grepl() were used to simply check whether a regular expression could be matched with a character vector, sub() and gsub() take it one step further: you can specify a replacement argument. If inside the character vector x, the regular expression pattern is found, the matching element(s) will be replaced with replacement.sub() only replaces the first match, whereas gsub() replaces all matches.

```r
# The emails vector has already been defined for you
emails <- c("john.doe@ivyleague.edu", "education@world.gov", "global@peace.org",
            "invalid.edu", "quant@bigdatacollege.edu", "cookie.monster@sesame.tv")

# Use sub() to convert the email domains to datacamp.edu
sub(pattern = "@.*\\.edu$", replacement = "@datacamp.edu", x = emails)
```

```
## [1] "john.doe@datacamp.edu"    "education@world.gov"
## [3] "global@peace.org"         "invalid.edu"
## [5] "quant@datacamp.edu"       "cookie.monster@sesame.tv"
```

Regular expressions are a typical concept that you'll learn by doing and by seeing other examples. Before you rack your brains over the regular expression in this exercise, have a look at the new things that will be used:

.*: A usual suspect! It can be read as "any character that is matched zero or more times".

\s: Match a space. The "s" is normally a character, escaping it (\) makes it a metacharacter.

[0-9]+: Match the numbers 0 to 9, at least once (+).

([0-9]+): The parentheses are used to make parts of the matching string available to define the replacement. The \1 in the replacement argument of sub() gets set to the string that is captured by the regular expression [0-9]+.

```r
awards <- c("Won 1 Oscar.",
  "Won 1 Oscar. Another 9 wins & 24 nominations.",
  "1 win and 2 nominations.",
  "2 wins & 3 nominations.",
  "Nominated for 2 Golden Globes. 1 more win & 2 nominations.",
  "4 wins & 1 nomination.")

sub(".*\\s([0-9]+)\\snomination.*$", "\\1", awards)
```

```
## [1] "Won 1 Oscar." "24"           "2"            "3"
## [5] "2"            "1"
```

Date

```r
# Get the current date: today
today <- Sys.Date()
today
```

```
## [1] "2020-04-13"
```

```r
# See what today looks like under the hood
unclass(today)
```

```
## [1] 18365
```

```r
# Get the current time: now
now <- Sys.time()
now
```

```
## [1] "2020-04-13 09:03:52 CST"
```

```r
# See what now looks like under the hood
unclass(now)
```

```
## [1] 1586739832
```

To create a Date object from a simple character string in R, you can use the as.Date() function. The character string has to obey a format that can be defined using a set of symbols (the examples correspond to 13 January, 1982):

%Y: 4-digit year (1982)

%y: 2-digit year (82)

%m: 2-digit month (01)

%d: 2-digit day of the month (13)

%A: weekday (Wednesday)

%a: abbreviated weekday (Wed)

%B: month (January)

%b: abbreviated month (Jan)

The following R commands will all create the same Date object for the 13th day in January of 1982:

```r
as.Date("1982-01-13")
as.Date("Jan-13-82", format = "%b-%d-%y")
as.Date("13 January, 1982", format = "%d %B, %Y")
```

Notice that the first line here did not need a format argument, because by default R matches your character string to the formats "%Y-%m-%d" or "%Y/%m/%d".

In addition to creating dates, you can also convert dates to character strings that use a different date notation. For this, you use the format() function. Try the following lines of code:

```r
today <- Sys.Date()
format(Sys.Date(), format = "%d %B, %Y")
format(Sys.Date(), format = "Today is a %A!")
```

```r
# Definition of character strings representing dates
str1 <- "May 23, '96"
str2 <- "2012-03-15"
str3 <- "30/January/2006"

# Convert the strings to dates: date1, date2, date3
date1 <- as.Date(str1, format = "%b %d, '%y")
date2 <- as.Date(str2, format = "%Y-%m-%d")
date3 <- as.Date(str3, format = "%d/%B/%Y")

# Convert dates to formatted strings
format(date1, "%A")
format(date2, "%d")
format(date3, "%b %Y")
```

Create and format times

Similar to working with dates, you can use as.POSIXct() to convert from a character string to a POSIXct object, and format() to convert from a POSIXct object to a character string. Again, you have a wide variety of symbols:

%H: hours as a decimal number (00-23)

%I: hours as a decimal number (01-12)

%M: minutes as a decimal number

%S: seconds as a decimal number

%T: shorthand notation for the typical format %H:%M:%S

%p: AM/PM indicator

For a full list of conversion symbols, consult the strptime documentation in the console:

?strptime

Again,as.POSIXct() uses a default format to match character strings. In this case, it's %Y-%m-%d %H:%M:%S. In this exercise, abstraction is made of different time zones.

```
# Definition of character strings representing times
str1 <- "May 23, '96 hours:23 minutes:01 seconds:45"
str2 <- "2012-3-12 14:23:08"

# Convert the strings to POSIXct objects: time1, time2
time1 <- as.POSIXct(str1, format = "%B %d, '%y hours:%H minutes:%M seconds:%S")
time2 <- as.POSIXct(str2, format = "%Y-%m-%d %H:%M:%S")

# Convert times to formatted strings
format(time1, "%M")
format(time2, "%I:%M %p")
```

Calculations with Dates

```
today <- Sys.Date()
today + 1
```

```
## [1] "2020-04-14"
```

```
today - 1
```

```
## [1] "2020-04-12"
```

```
as.Date("2015-03-12") - as.Date("2015-02-27")
```

```
## Time difference of 13 days
```

```
# day1, day2, day3, day4 and day5 are already available in the workspace
day1 <- as.Date("2018-08-15")
day2 <- as.Date("2018-08-17")
day3 <- as.Date("2018-08-22")
day4 <- as.Date("2018-08-28")
day5 <- as.Date("2018-09-02")

# Difference between last and first pizza day
as.Date(day5) - as.Date(day1)

# Create vector pizza
pizza <- c(day1, day2, day3, day4, day5)

# Create differences between consecutive pizza days: day_diff
day_diff <- diff(pizza)

# Average period between two consecutive pizza days
mean(day_diff)
```

Calculations with Times

Calculations using POSIXct objects are completely analogous to those using Date objects. Try to experiment with this code to increase or decrease POSIXct objects:

```
now <- Sys.time()
now + 3600          # add an hour
```

```
## [1] "2020-04-13 10:03:52 CST"
```

```
now - 3600 * 24     # subtract a day
```

```
## [1] "2020-04-12 09:03:52 CST"
```

Adding or substracting time objects is also straightforward:

```
birth <- as.POSIXct("1879-03-14 14:37:23")
death <- as.POSIXct("1955-04-18 03:47:12")
einstein <- death - birth
einstein
```

```
## Time difference of 27792.51 days
```

You're developing a website that requires users to log in and out. You want to know what is the total and average amount of time a particular user spends on your website. This user has logged in 5 times and logged out 5 times as well. These times are gathered in the vectors login and logout, which are already defined in the workspace.

```
# login and logout are already defined in the workspace
login <- as.POSIXct(c("2018-08-19 10:18:04 UTC", "2018-08-24 09:14:18 UTC"
, "2018-08-24 12:21:51 UTC", "2018-08-24 12:37:24 UTC"
, "2018-08-26 21:37:55 UTC"))
logout <- as.POSIXct(c("2018-08-19 10:56:29 UTC", "2018-08-24 09:14:52 UTC"
, "2018-08-24 12:35:48 UTC", "2018-08-24 13:17:22 UTC"
, "2018-08-26 22:08:47 UTC"))

# Calculate the difference between login and logout: time_online
time_online = logout - login

# Inspect the variable time_online
time_online
```

```
## Time differences in secs
## [1] 2305   34  837 2398 1852
```

```
# Calculate the total time online
sum(time_online)
```

```
## Time difference of 7426 secs
```

```r
# Calculate the average time online
mean(time_online)
```

```
## Time difference of 1485.2 secs
```

```r
astro <- c("20-Mar-2015", "25-Jun-2015", "23-Sep-2015", "22-Dec-2015")
names(astro) <- c("spring", "summer", "fall", "winter")

meteo <- c("March 1, 15", "June 1, 15", "September 1, 15", "December 1, 15")
names(meteo) <- c("spring", "summer", "fall", "winter")

# Convert astro to vector of Date objects: astro_dates
astro_dates <- as.Date(astro, format = "%d-%b-%Y")

# Convert meteo to vector of Date objects: meteo_dates
meteo_dates <- as.Date(meteo, format = "%B %d, %y")

# Calculate the maximum absolute difference between astro_dates and meteo_dates
max(abs(astro_dates - meteo_dates))
```