# The vector pioneers has already been created for you

pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")

# Split names from birth year

split\_math <- strsplit(pioneers, split = ":")

# Convert to lowercase strings: split\_low

split\_low <- lapply(split\_math, tolower)

# Take a look at the structure of split\_low

str(split\_low)

# Code from previous exercise:

pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623", "PEARSON:1857")

split <- strsplit(pioneers, split = ":")

split\_low <- lapply(split, tolower)

# Write function select\_first()

select\_first <- function(x) {

x[1]

}

# Apply select\_first() over split\_low: names

names <- lapply(split\_low, select\_first)

# Write function select\_second()

select\_second <- function(x){

x[2]

}

# Apply select\_second() over split\_low: years

years <- lapply(split\_low, select\_second)

# split\_low has been created for you

split\_low

# Transform: use anonymous function inside lapply

names <- lapply(split\_low, function(x) {x[1]})

# Transform: use anonymous function inside lapply

years <- lapply(split\_low, function(x) {x[2]})

# Generic select function

select\_el <- function(x, index) {

x[index]

}

# Use lapply() twice on split\_low: names and years

names <- lapply(split\_low, select\_el, index = 1)

years <- lapply(split\_low, select\_el, index = 2)

# temp has already been defined in the workspace

str(temp)

# Use lapply() to find each day's minimum temperature

lapply(temp, min)

# Use sapply() to find each day's minimum temperature

sapply(temp, min)

# Use lapply() to find each day's maximum temperature

lapply(temp, max)

# Use sapply() to find each day's maximum temperature

sapply(temp, max)

# Finish function definition of extremes\_avg

extremes\_avg <- function(x) {

( min(x) + max(x) ) / 2

}

# Apply extremes\_avg() over temp using sapply()

sapply(temp, extremes\_avg)

# Apply extremes\_avg() over temp using lapply()

lapply(temp, extremes\_avg)

# Create a function that returns min and max of a vector: extremes

extremes <- function(x) {

c(min = min(x), max = max(x))

}

# Apply extremes() over temp with sapply()

sapply(temp, extremes)

# Apply extremes() over temp with lapply()

lapply(temp, extremes)

# Definition of below\_zero()

below\_zero <- function(x) {

return(x[x < 0])

}

# Apply below\_zero over temp using sapply(): freezing\_s

freezing\_s <- sapply(temp, below\_zero)

# Apply below\_zero over temp using lapply(): freezing\_l

freezing\_l <- lapply(temp, below\_zero)

# Are freezing\_s and freezing\_l identical?

identical()

# Definition of print\_info()

print\_info <- function(x) {

cat("The average temperature is", mean(x), "\n")

}

# Apply print\_info() over temp using sapply()

sapply(temp, print\_info)

# Apply print\_info() over temp using lapply()

lapply(temp, print\_info)

# Definition of basics()

basics <- function(x) {

c(min = min(x), mean = mean(x), max = max(x))

}

# Apply basics() over temp using vapply()

vapply(temp, basics, numeric(3))

# Convert to vapply() expression

vapply(temp, max, numeric(1))

# Convert to vapply() expression

vapply(temp, function(x, y) { mean(x) > y }, y = 5, logical(1))

#generate a sequence of numbers from x to y by steps of z (z can be negative…)

seq(x, y, by = z)

#replication (times can also be replaced by each…)

rep(x, times = z)

# sort in ascending or descending order

sort(x, descending = TRUE)

# The errors vector has already been defined for you

errors <- c(1.9, -2.6, 4.0, -9.5, -3.4, 7.3)

# Sum of absolute rounded values of errors

sum(round(abs(errors)))

* [**seq()**](http://www.rdocumentation.org/packages/base/functions/seq): Generate sequences, by specifying the from, to, and by arguments.
* [**rep()**](http://www.rdocumentation.org/packages/base/functions/rep): Replicate elements of vectors and lists.
* [**sort()**](http://www.rdocumentation.org/packages/base/functions/sort): Sort a vector in ascending order. Works on numerics, but also on character strings and logicals.
* [**rev()**](http://www.rdocumentation.org/packages/base/functions/rev): Reverse the elements in a data structures for which reversal is defined.
* [**str()**](http://www.rdocumentation.org/packages/utils/functions/str): Display the structure of any R object.
* [**append()**](http://www.rdocumentation.org/packages/base/functions/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()**](http://www.rdocumentation.org/packages/base/functions/unlist): Flatten (possibly embedded) lists to produce a vector.

# regular expressions

?rege

# check for the existance of patterns

grepl() 🡪 search for a patter in zb a vector (true false)

grep 🡪 returns a vector of indicies that match (indicies that are true)

# replace such patterns:

sub() 🡪 replaces the first instace of true in a string

gsub() 🡪 replaces all instances in a string

# 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"

grepl(pattern = "edu", x= emails)

# Use grep() to match for "edu", save result to hits

hits <- grep(patter = "edu", x = emails)

# Subset emails using hits

emails[hits]

* @, 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.

# Use grepl() to match for .edu addresses more robustly

grepl(pattern = "@.\*\\.edu$", x = emails)

# 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]

# Use sub() to convert the email domains to datacamp.edu

sub(pattern = "@.\*\\.edu$", replacement = "@datacamp.edu", x = emails)

# Times and dates

# Get the current date: today

today <- Sys.Date()

# See what today looks like under the hood

unclass(today)

# Get the current time: now

now <- Sys.time()

# See what now looks like under the hood

unclass(now)

* %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)

as.Date("1982-01-13")

as.Date("Jan-13-82", format = "%b-%d-%y")

as.Date("13 January, 1982", format = "%d %B, %Y")

today <- Sys.Date()

format(Sys.Date(), format = "%d %B, %Y")

format(Sys.Date(), format = "Today is a %A!")

# 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)

date3 <- as.Date(str3, format = "%d/%B/%Y")

# Convert dates to formatted strings

format(date1, "%A")

format(date2, "%d")

format(date3, "%b %Y")

?strptime

* %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

# 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)

# Convert times to formatted strings

format(time1, "%M")

format(time2, "%I:%M %p")

today <- Sys.Date()

today + 1

today - 1

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

# Difference between last and first pizza day

day1-day5

# Create vector pizza

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

# Create differences between consecutive pizza days: day\_diff

day\_diff <- c(day1-day2, day2-day3, day3-day4, day4-day5)

day\_diff <- diff(pizza) # is the same but with the neat function diff()

# Average period between two consecutive pizza days

mean(day\_diff)

# login and logout are already defined in the workspace

# Calculate the difference between login and logout: time\_online

time\_online <-logout-login

# Inspect the variable time\_online

time\_online

# Calculate the total time online

sum(time\_online)

# Calculate the average time online

mean(time\_online)

# 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))