HW03 - Stat 133, Fall 2016, Prof. Sanchez

The purpose of this assignment is to write simple functions, as well as practicing using conditionals and loops in R. Please turn in a physical copy of the knitted html file during lab, and submit both your `.Rmd` and `.html` files to bCourses. Due Friday Oct-07.

Area of a circle

For a given circle of radius r, the area A is:

$$A = \pi r^2$$

Write a function `circle_area()` that calculates the area of a circle. This function must take one argument `radius`. Give `radius` a default value of 1. The function should `stop()` if `radius` is negative.

For example:

```
# default (radius 1)
circle_area()
```

```
## [1] 3.141593
```

```
# radius 3
circle_area(radius = 3)
```

```
## [1] 28.27433
```

This should not work

```
# bad radius
circle_area(radius = -2)
```

Area of a cylinder

For a given cylinder of radius r and height h the area A is:

$$A = 2\pi rh + 2\pi r^2$$

Notice that the formula of the area of a cylinder includes the area of a circle: πr^2. Write a function `cyl_area()`, that calls `circle_area()`, to compute the area of a cylinder.

This function must take two arguments: `radius` and `height`. Give both arguments a default value of 1. In addition, the function should stop if any of `radius` or `height` are negative.

For instance:

```
# default (radius 1, height 1)
cyl_area()
```

```
## [1] 12.56637
```

```r
# radius 2, height 3
cyl_area(radius = 2, height = 3)
```

```
## [1] 62.83185
```

These should not work

```r
# bad radius
cyl_area(radius = -2, height = 1)

# bad height
cyl_area(radius = 2, height = -1)

# bad radius and height
cyl_area(radius = -2, height = -1)
```

Volume of a cylinder

For a given cylinder of radius r and height h the volume V is:

$$V = \pi r^2 h$$

Write a function `cyl_volume()`, that calls `circle_area()`, to compute the volume of a cylinder. This function must take two arguments: `radius` and `height`. Give both arguments a default value of 1.

For example:

```r
# default (radius 1, height 1)
cyl_volume()
```

```
## [1] 3.141593
```

```r
cyl_volume(radius = 3, height = 10)
```

```
## [1] 282.7433
```

```r
cyl_volume(height = 10, radius = 3)
```

```
## [1] 282.7433
```

Currency Converter

Consider the exchange rates of one US dollar for the following currencies (source: XE currency table US dollar, 09-20-2016):

Currency	name	rate
US dollar	`dollar`	1.00
Euro	`euro`	0.89
British pound	`pound`	0.77
Japanese yen	`yen`	101.69
Chinese yuan	`yuan`	6.67
South Korean Won	`won`	1118.21
Indian rupee	`rupee`	66.98
Mexican peso	`peso`	19.82
Brazilian real	`real`	3.25

Write a function `exchange()` that converts from one currency to another. The way you should be able to use this function is like this:

```
# from dollar to euro
exchange(amount = 1, from = 'dollar', to = 'euro')

# from real to yen
exchange(amount = 5, from = 'real', to = 'yen')
```

- `amount` is a numeric input
- `from` is a character string indicating the name of a currency
- `to` is a character string indicating the name of a currency

Give these arguments default values of `amount = 1`, `from = "dollar"`, and `to = "euro"`. Inside `exchange()` you must declare a named vector with the given exchange rates:

```
exchange <- function(amount = 1, from = "dollar", to = "euro") {
  # vector of dollar exchange rates
  x <- c(
    dollar = 1,
    euro = 0.89,
    pound = 0.77,
    yen = 101.69,
    yuan = 6.67,
    won = 1118.21,
    rupee = 66.98,
    peso = 19.82,
    real = 3.25)

  # write the rest of the code of your function
  # ...
}
```

- You can use any control flow structure: `if-else`, `switch()`, `for` loops, `while`, `repeat`.
- You can use any data structures inside `exchange()`: vectors, matrices, data frames, lists, etc.

- The output must be a numeric vector

Test your `exchange()` function with:

```
exchange()
exchange(amount = 10, from = 'euro', to = 'peso')
exchange(amount = 20, from = 'yuan', to = 'pound')
exchange(amount = 30, from = 'rupee', to = 'won')
```

Two Given Points

Let p_1 and p_2 be two points with two coordinates: $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$.

The distance d between two points can be calculated with the formula:

$$d = \sqrt{(x_2 - x_2)^2 + (y_2 - y_1)^2}$$

The midpoint of the line segment between p_1 and p_2 can be found as:

$$p = \left(\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2} \right)$$

The intercept a and the slope b of the line $y = a + bx$ connecting two points p_1 and p_2 can be found as:

$$b = \frac{y_2 - y_1}{x_2 - x_1}, \quad a = y_1 - bx_1$$

Distance

Write a function `find_distance()` that returns the distance between two given points. You should be able to call the function like this:

```
# coordinates for point-1 and point-2
p1 <- c(0, 0)
p2 <- c(1, 1)

find_distance(p1, p2)
```

Midpoint

Write a function `find_midpoint()` that returns the midpoint between two given points. You should be able to call the function like this:

```
p1 <- c(0, 0)
p2 <- c(1, 1)

find_midpoint(p1, p2)
```

Slope

Write a function `find_slope()` that returns the slope of the line connecting two given points. You should be able to call the function like this:

```r
p1 <- c(0, 0)
p2 <- c(1, 1)

find_slope(p1, p2)
```

Intercept

Write a function `find_intercept()` that returns the intercept of the line connecting two given points. This function must internally use `find_slope()`

```r
p1 <- c(0, 0)
p2 <- c(1, 1)

find_intercept(p1, p2)
```

Line

Write a function `find_line()`. This function must use `find_slope()` and `find_intercept()`. The output should be a list with two named elements: `"intercept"` and `"slope"`, Here is how you should be able to use `find_line()`:

```r
p1 <- c(0, 0)
p2 <- c(1, 1)

eq <- find_line(p1, p2)
eq$intercept
eq$slope
```

Information about two given points

Once you have the functions `find_distance()`, `find_midpoint()`, and `find_line()`, write an overall function called `info_points()` that returns a list with the distance, the midpoint, and the line's slope and intercept terms. Here is how you should be able to use `info_points()`:

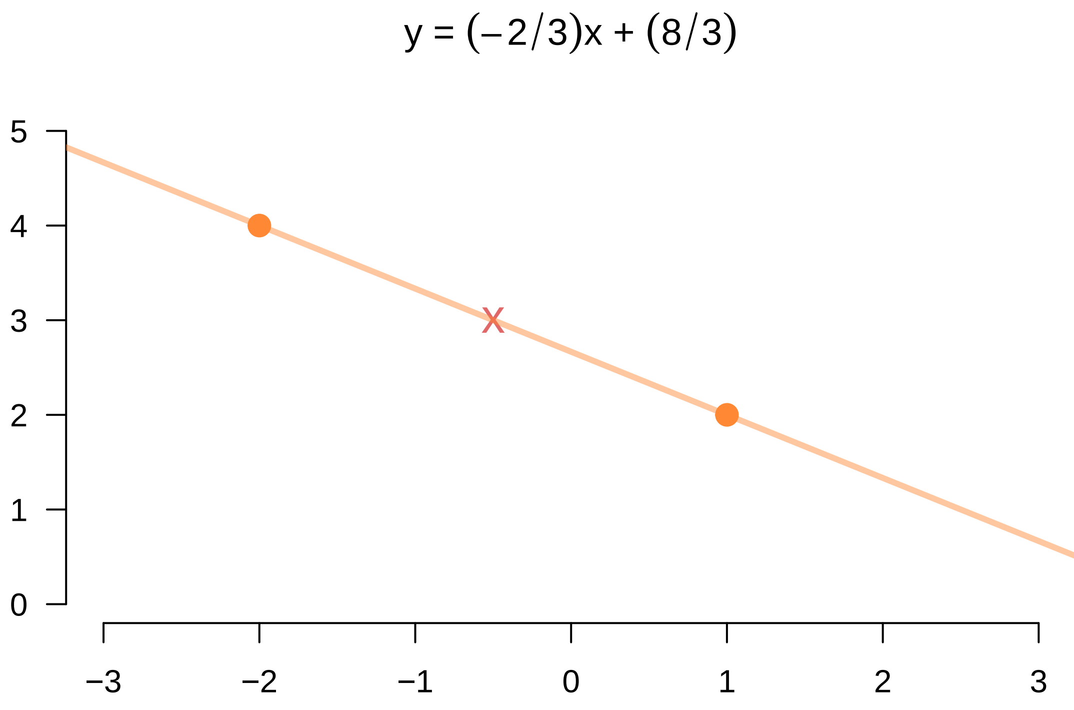
```r
p1 <- c(-2, 4)
p2 <- c(1, 2)

results <- info_points(p1, p2)
results$distance
results$midpoint
results$intercept
results$slope
```

Use the following code to create a plot that displays the given points, the line, and the midpoint. Note that the title of the plot shows the line equation. For instance, if the points are $p_1 = (-2, 4)$ and $p_2 = (1, 2)$, the plot may look like this (you should choose different points!):

```r
# change these points and pass them to info_point()
p1 <- c(-2, 4)
p2 <- c(1, 2)
```

```r
plot.new()
# depending on your chosen points you may have to set different limits
plot.window(xlim = c(-3, 3), ylim = c(0, 5))
axis(side = 1)
axis(side = 2, las = 1)
points(p1[1], p1[2], cex = 1.5, col = "#FF8834", pch = 19)
points(p2[1], p2[2], cex = 1.5, col = "#FF8834", pch = 19)
# midpoint (here you should use the midpoint outputs of your function)
points(-1/2, 3, cex = 1.5, pch = "x", col = "#E16868")
# slope and intercept (here you should use the outputs of your function)
abline(a = 8/3, b = -2/3, col = "#FF883477", lwd = 3)
title(main = expression(paste(y, ' = ', (-2/3) * x, ' + ', (8/3))))
```



Data: Weekly California Gasoline Prices

The data set for this problem has to do with weekly gasoline prices in California during 2015. The data comes from the *U.S. Energy Information Administration* (EIA). You can find more information in the website www.eia.gov.

The image below is a screen-capture showing the data set as it appears in the EIA website: weekly California retail gasoline prices from January till December 2015 (source: *U.S. Energy Information Administration*)

I've scrapped the data from 2015 and saved it in a csv file available in the github repository:

https://github.com/ucb-stat133/stat133-fall-2016/raw/master/data/raw-gas-prices-2015.csv

The data table in `raw-gas-prices-2015.csv` has 11 columns:

Year-Month	Week 1		Week 2		Week 3		Week 4		Week 5	
	End Date	Value	End Date	Value	End Date	Value	End Date	Value	End Date	Value
2015-Jan	01/05	2.671	01/12	2.594	01/19	2.484	01/26	2.440		
2015-Feb	02/02	2.441	02/09	2.627	02/16	2.798	02/23	2.959		
2015-Mar	03/02	3.418	03/09	3.439	03/16	3.356	03/23	3.267	03/30	3.209
2015-Apr	04/06	3.147	04/13	3.102	04/20	3.158	04/27	3.433		
2015-May	05/04	3.711	05/11	3.732	05/18	3.807	05/25	3.757		
2015-Jun	06/01	3.693	06/08	3.591	06/15	3.511	06/22	3.480	06/29	3.450
2015-Jul	07/06	3.432	07/13	3.880	07/20	3.897	07/27	3.812		
2015-Aug	08/03	3.724	08/10	3.565	08/17	3.584	08/24	3.483	08/31	3.342
2015-Sep	09/07	3.266	09/14	3.155	09/21	3.072	09/28	2.994		
2015-Oct	10/05	2.949	10/12	2.914	10/19	2.861	10/26	2.847		
2015-Nov	11/02	2.817	11/09	2.824	11/16	2.780	11/23	2.716	11/30	2.691
2015-Dec	12/07	2.679	12/14	2.654	12/21	2.736	12/28	2.825		

Figure 1: Weekly CA Gasoline prices 2015

V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
2015-Jan	01/05	2.67	01/12	2.59	01/19	2.48	01/26	2.44		
2015-Feb	02/02	2.44	02/09	2.63	02/16	2.80	02/23	2.96		
2015-Mar	03/02	3.42	03/09	3.44	03/16	3.36	03/23	3.27	03/30	3.21
2015-Apr	04/06	3.15	04/13	3.10	04/20	3.16	04/27	3.43		
2015-May	05/04	3.71	05/11	3.73	05/18	3.81	05/25	3.76		
2015-Jun	06/01	3.69	06/08	3.59	06/15	3.51	06/22	3.48	06/29	3.45

Table 2: First six rows in raw-gas-prices-2015.csv

- `V1` corresponds to the month name
- `V2`, `V4`, ..., `V10` contain the starting day of the week (some months have 4 weeks, and others have 5 weeks)
- `V3`, `V5`, ..., `V11` contain the weekly gas prices

The goal of this problem is to get a *clean* version of the data set. To accomplish this task, you will have to "reshape" the raw data set and create a new data frame `gas_prices` with a simpler structure having the following form:

week	date	price
1	01/05	2.67
2	01/12	2.59
3	01/19	2.48
4	01/26	2.44
5	02/02	2.44

Table 3: First five rows of weekly gas prices

- `week` has the number of weeks (52 in total)
- `date` corresponds to the starting dates of the week
- `price` corresponds to the price for the associated date

In order to create the data frame `gas_prices` you should use one or more `for` loops to extract the appropriate values from the table of raw gas prices, and then put them in the form of a data frame.