HW05 - Stat 133, Fall 2016, Prof. Sanchez

Your Name

The purpose of this assignment is to write simple functions.

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
```

Even number

Write a function `is_even()` that determines whether a number is even (i.e. multiple of 2). If the input number is even, the output should be `TRUE`. If the input number is odd, the output should be `FALSE`. If the input is not a number, the output should be `NA`

For example:

```r
# even number
is_even(10)
```

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

```r
# odd number
is_even(33)
```

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

```r
# not a number
is_even('a')
```

```
## [1] NA
```

Odd number

Use your function `is_even()` to write a function `is_odd()` that determines if a number is odd (i.e. not a multiple of 2). If a number is odd, the output should be `TRUE`; if a number is even the output should be `FALSE`; if the input is not a number the output should be `NA`

For example:

```r
# odd number
is_odd(1)
```

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

```r
# even number
is_odd(4)
```

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

```r
# not a number
is_odd('a')
```

```
## [1] NA
```

Converting Miles to other units

The table below shows the different formulas for converting miles (mi) into other scales:

Units	Formula
Inches	mi x 63360
Feet	mi x 5280
Yards	mi x 1760
Meters	mi / 0.00062137
Kms	mi / 0.62137

Write the following five functions for each type of conversion. Each function must take one argument `x` with default value: `x = 1`.

- `miles2inches()`
- `miles2feet()`
- `miles2yards()`
- `miles2meters()`
- `miles2kms()`

For example:

```
miles2inches(2)
```

```
## [1] 126720
```

```
miles2feet(2)
```

```
## [1] 10560
```

```
miles2yards(2)
```

```
## [1] 3520
```

```
miles2meters(2)
```

```
## [1] 3218.694
```

```
miles2kms(2)
```

```
## [1] 3.218694
```

Using switch()

Create a function `convert()` that converts miles into the specified units. Use `switch()` and the previously defined functions—`miles2inches()`, `miles2feet()`, ..., `miles2kms`—to define `convert()`. Use two arguments: `x` and `to`, like this:

```
convert(40, to = "in")
```

By default, `to = "km"`, but it can take values such as `"in"`, `"ft"`, `"yd"`, or `"m"`.

For instance:

```
convert(3, "in")
convert(3, "ft")
convert(3, "yd")
convert(3, "m")
convert(3, "km")
```

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)
```

```
# your 'find_distance()' function
```

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)
```

```
# your 'find_midpoint()' function
```

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:

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

find_slope(p1, p2)
```

```r
# your 'find_slope()' function
```

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)
```

```r
# your 'find_intercept()' function
```

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
```

```r
# your 'find_line()' function
```

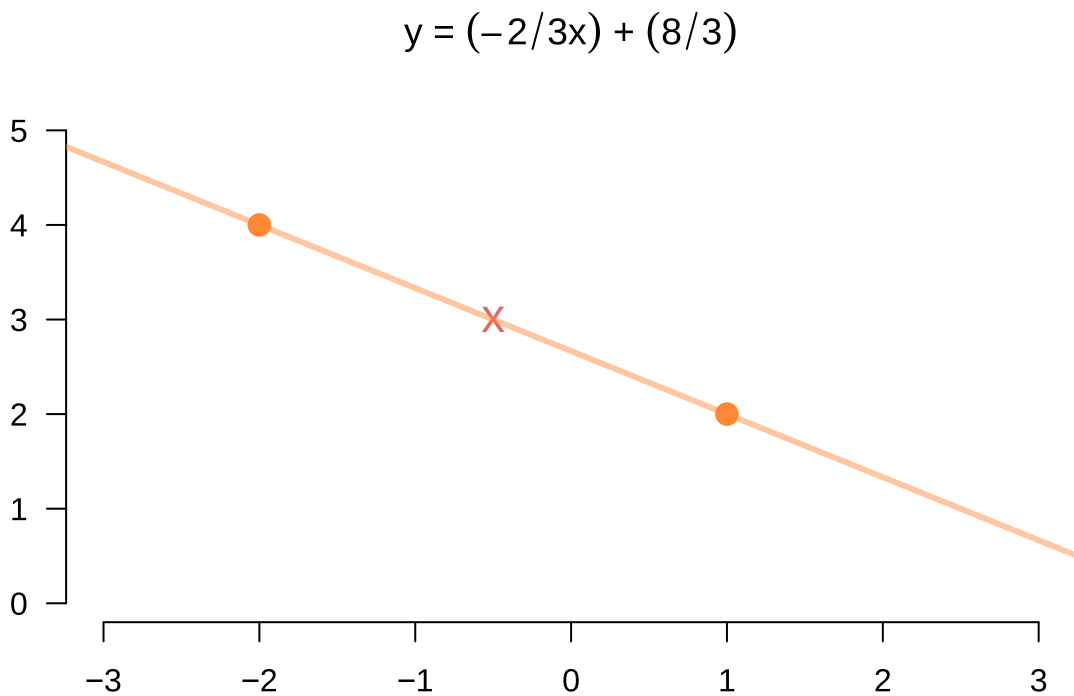
Information about two given points

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
```

Finally create a plot that displays the given points, the line, and the midpoint. The title of the plot must show the line equation. For instance, if the points are $p_1 = (-2, 4)$ and $p_2 = (1, 2)$, the plot may look like this:


y = (−2/3x) + (8/3)
5
4
3
2
1
0
X
−3
−2
−1
0
1
2
3