Classwork 14A: Intro to `map(.x, .f)`

In this classwork, I'll introduce you to the function `map()`. It will be useful to run monte-carlo experiments starting in part B of this classwork.

```
library(tidyverse)
```

Vectorized Functions

In CW13A, we learned that we could define our own custom functions. Here I define a function called `pct_change` that takes two arguments: an "old" value and a "new" value, and it returns the percentage change between them.

```
pct_change <- function(old, new) {
  (new - old) / old
}
```

1. Verify that `pct_change` works on values.

```
# pct_change(__, __) == 1
```

2. Verify that `pct_change` works on vectors.

```
# pct_change(__, __) == c(0, 1, 2)
```

How is it that `pct_change` works on vectors? Well, `pct_change` is defined only using subtraction and division, which both work element-wise on vectors. So since you can plug vectors directly into the computation that's done in the function body without an issue, `pct_change` works on vectors without an issue:

```
(c(4, 5, 6) - c(1, 2, 3)) / c(1, 2, 3)
```

```
## [1] 3.0 1.5 1.0
```

Lots of the functions we've been using are "vectorized" (they work on vectors): `sum()`, `mean()`, `max()`, etc. But some functions may not work on vectors so smoothly, or in the way you want. For example: consider `rnorm()`: it generates `n` random numbers from a normal distribution with any mean and standard deviation (by default, mean 0 and sd 1).

```
# ?rnorm
```

3. Use `rnorm()` to generate 10 random numbers from the normal distribution with mean 0 and sd 1, that is, N(0, 1).

```
# rnorm(__, __, __)
```

Suppose you need to generate 10 random numbers from N(0, 1), then 10 random numbers from N(0, 2), then 10 random numbers from N(0, 3), then 10 random numbers from N(0, 4), all the way up to 10 random numbers from N(0, 100).

As a first attempt, try putting the vector 1:100 into the `sd` argument:

```
rnorm(n = 10, mean = 0, sd = 1:100)
```

```
##  [1]  0.3336260  0.7557768 -5.2914044  6.7720983 -2.0948916 -4.2542528
##  [7] -0.1438212 13.9721384  2.0626713 10.2940250
```

What happens? R generates one random number from N(0, 1), one random number from N(0, 2), one random number from N(0, 3), all the way up to one random number from N(0, 10). It stops at 10 because `n = 10`. `rnorm` is vectorized, but not in the way that helps us solve this problem.

Second attempt: you could copy-paste 100 times, changing the sd each time:

```
rnorm(n = 10, mean = 0, sd = 1)
rnorm(n = 10, mean = 0, sd = 2)
rnorm(n = 10, mean = 0, sd = 3)
rnorm(n = 10, mean = 0, sd = 4)
#etc.
```

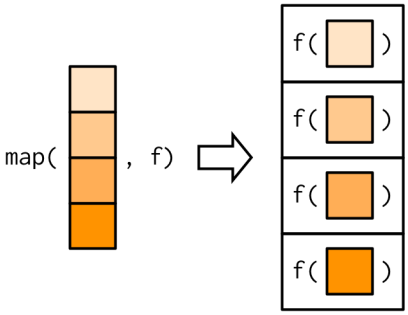
But this method is annoying for you to do, it contains a lot of extra code for a reader to have to read, and it doesn't use the computer as the powerful tool it is. And if you make a mistake with all that copy-pasting, it's hard to catch.

A better solution: use `map(.x, .f)`.

`map(.x, .f)` is from the package `purrr`, which is the last tidyverse package we'll talk about in this course. `purrr` is named the way it is because it helps "make your functions purrr" like a well oiled machine (helps make your functions vectorized in the way you want). And that's exactly what we needed help with in the example above: `rnorm()` wasn't purrring for us!

`map(.x, .f)` does one simple thing: It applies the function `.f` to each element of the vector `.x`.

`map(.x, .f)` will return a list of the same length as the inputs `.x`. Check out the diagram that explains visually how to use `map`:



The name `map` refers to how "map" is used in math: a mapping is an operation that associates each element of a set with elements in a different set. The `.x` are the inputs, and the `.f` is the function to apply to get to the set of outputs.

So how do we use `map(.x, .f)` to solve our problem?

Write out the beginning of the copy-paste solution:

```
rnorm(n = 10, mean = 0, sd = 1)
rnorm(n = 10, mean = 0, sd = 2)
rnorm(n = 10, mean = 0, sd = 3)
```

What should be the vector of inputs `.x`? It's whatever needs to **change** in the copy-paste version. In that code, everything stays the same except for the `sd` needs to change: it needs to go from 1 to 100. So `.x` should be the vector `1:100`.

What's the function `.f` we'll apply to every element of `.x`? It's a function that should take a standard deviation and output 10 random normal numbers with a mean of zero and that custom standard deviation:

```
# function(standard_dev) {
#   rnorm(n = 10, mean = 0, sd = standard_dev)
# }
```

```
map(
  .x = 1:100,
  .f = function(standard_dev) {
    rnorm(n = 10, mean = 0, sd = standard_dev)
  }
)
```

4. Use `map(.x, .f)` to generate 10 random normals with a mean of 1, 10 random normals with a mean of 2, 10 random normals with a mean of 3, all the way up to 10 random normals with a mean of 100.

They should all have a standard deviation of 1.

```
# map(
#   .x = __,
#   .f = __
# )
```

One more example: if you wanted to take a vector and multiply each element by 3, this is the best way:

```
3 * 1:10
```

```
##  [1]  3  6  9 12 15 18 21 24 27 30
```

But for the sake of practice, in the next problem, you'll use `map(.x, .f)` to do the same task.

5. Use `map(.x, .f)` to take the vector 1:10 and apply a function which multiplies each element by 3 in order to get the vector (3, 6, 9, 12, . . .).

Hint: the copy-paste version would be:

```
3 * 1
3 * 2
3 * 3
# etc
```

What changes? That's what needs to go in your `.x`. Your `.f` should be a function that takes an element of `.x` and multiplies that element by 3.

```
# map(
#   .x = __,
#   .f = function(__) {
#     __
#   }
# )
```