hw2

Problem 1

First, run a *paired* t-test using the `t.test()` function to compare the means for these two vectors. (You'll need to read the help to determine how to do a paired t-test - read the R help, not online resources.) You don't need to write your own function for this, just call the built-in `t.test()` function with the appropriate parameters.

```r
sample1 <- rnorm(100, mean = 5, sd = 2)
sample2 <- rnorm(100, mean = 6, sd = 2)
sample2[c(5, 62, 37)] <- NA

## finish the code and uncomment :)
# result <-
# print(result)
```

You'll notice that three entries of `sample2` above were set to `NA`. What happened during the test to those pairs?

Write your answer here.

Problem 2

Second, write a function called `paired_ttest_pval()` that takes two required parameters, `samp1` and `samp2`; this function should run a paired t-test as above on the two samples, and return *just the pvalue* resulting from the test, not all the information resulting from the test.

Don't forget to document your function with a comment describing what the function does, what its parameter types are, and what its return value is.

```r
# Write function here


## uncomment this to test
# x <- rnorm(100, mean = 5, sd = 2)
# y <- rnorm(100, mean = 6, sd = 2)
#
# result <- paired_ttest_pval(x, y)
# print(result)
```

The printed `result` should be a length-one numeric vector containing the p-value, so the printout should be something like `[1] 0.000392606` (though your output will be slightly different).

Problem 3

Write a function called `strip_nas()` that takes a single vector `x` and returns a version with all `NA` values removed. It will likely be very short and make use of `is.na()`.

```r
# write function here


## uncomment this to test
```

```
# vals <- c(4, 5, NA, 2.1, NA, 8)
# result <- strip_nas(vals)
# print(result)                          # should print 4, 5, 2.1, 8
```

Next, write another function called `cov()` that takes a single vector `x` and returns the *coefficient of variation* of `x`, defined as the mean divided by the standard deviation. Your function first should remove any `NA` values from `x` by calling your `strip_nas()` function first. (Which is to say, you'll need to call `strip_nas()` inside of your `cov()` function.

```
# write function here


## uncomment this to test
# vals <- c(4, 5, NA, 2.1, NA, 8)
# result <- cov(vals)
# print(result)                          # should print 1.9382
```

Problem 4

If you followed the directions carefully, both your `strip_nas()` function and your `cov()` function should make use of an `x` variable. Explain why this code works, even though `x` is used multiple times in the code for different purposes.

Your answer here.

Bonus - not for grading

This problem isn't for grading, but rather just something kind of fun. Local variables not only enable clean code via encapsulation, they enable really interesting ways of solving problems. Consider the factorial function `!`, where $5! = 5 \times 4 \times 3 \times 2 \times 1 = 120$. In code we might write this as `factorial(5)` and expect `120` to be returned.

Notice that, by defintion, $1! = 1$, and $n! = n \times (n-1)!$. Here's some code that computes factorials. We haven't talked about `if` and `else` yet, but hopefully it's clear what's going on from the code.

```
factorial <- function(n) {
  if(n == 1) {
    return(1)
  } else {
    subanswer <- factorial(n-1)
    answer <- n * subanswer
    return(answer)
  }
}

result <- factorial(5)
print(result)
```

```
## [1] 120
```

The question is, how exactly does this code work?

Answer here :)

As a followup, the Fibonacci sequence is defined as `fib(1) = 1`, `fib(2) = 1`, `fib(3) = 2`, `fib(4) = 3`, `fib(5) = 5`, `fib(6) = 8`, `fib(7) = 13`, `...`, where each value is the sum of the previous two, unless `n == 1` or `n == 2`, in which case the answer is just `1`.

Write the function `fib()` according to the scheme above (known as *recursion*) to compute the n^{th} Fibonacci number.

```
# function here



## uncomment to test
# test <- fib(7)
# print(test)          # should print 13
#
# test2 <- fib(12)
# print(test2)         # should be 144
```