1 Problem 1

Write a function that computes Fibonacci numbers. Given an integer, n, the function should return the n-th Fibonacci number.

```
using BenchmarkTools
using StatsBase

function fib(n)
    res = zeros(Int, n)
    res[1] = 1
    res[2] = 1
    for i = 3:n
        res[i] = res[i - 1] + res[i - 2]
    end
    return res[n]
end

@time fib(40)
@benchmark fib(40)
@code_native fib(40)


function fib2(n)
    a = 1
    b = 1
    for i = 3:n
        tmp = a
        a = a + b
        b = tmp
    end
    a
end

@time fib2(40)
@benchmark fib2(40)
@code_native fib2(40)

function fib3(n)
    if n < 3
        res = 1
    else
        res = fib3(n - 1) + fib3(n - 2)
    end
    return res
end

@time fib3(40)
@benchmark fib3(40)
@code_native fib3(40)
```

2 Problem 2.

Write a function that, when given an arrry of characters, will return a dictionary with counts for each how many times each character appeared in the array.

```
function countchars(v)
```

```
    cnts = Dict()
    n = length(v)
    for i = 1:n
        cnts[v[i]] = get(cnts, v[i], 0) + 1
    end
    return cnts
end


function countchars2(v)
    cnts = Dict{Char, Int}()
    n = length(v)
    for i = 1:n
        cnts[v[i]] = get(cnts, v[i], 0) + 1
    end
    return cnts
end

n_elems = 10_000_000
a = rand(['a', 'b', 'c', 'd', 'e', 'f'], n_elems)

@time countchars(a)
@benchmark countchars(a)

@time countchars2(a)
@benchmark countchars2(a)

@time countmap(a)
@benchmark countmap(a)
```

3 Challenge Problem 3.

Write a function that, when given a matrix, X, of integers, will return a new matrix, Y, whose elements Y[i, j] will be boolean values indicating whether or not the element X[i, j] is prime. Note that you can either write your own prime-checking function or use the `isprime()` function in the Primes.jl package.

```
using Base.Threads

srand(137)
A = rand(1:100_000, 10_000, 10_000)

function prime_status(mat)
    n, p = size(mat)
    res = falses(n, p)

    @threads for j = 1:p
        for i = 1:n
            res[i, j] = isprime(mat[i, j])
        end
    end
    return res
end

@time B = prime_status(A)
```

4 Challenge Problem 4.

You are given 100,000-by-20,000 a matrix, `A`, and your task compute the means, standard deviations, and variances of each column in the matrix. The code below accomplishes this, but can be optimize in a few ways.

The code's current run time is about 4 minutes on an Intel(R) Core(TM) i7-4870HQ CPU @ 2.50GHz

4.1 Starter Code

```
srand(111)

# Initialize matrix of random values
A = rand(100_000, 20_000)

function column_means(X)
    p = size(X, 2)

    # Initialize vector to store means
    means = Float64[]

    for j = 1:p
        push!(means, mean(X[:, j]))
    end
    return means
end

function column_stdevs(X)
    p = size(X, 2)
    # Initialize vector to store standard deviations
    stdevs = Float64[]

    for j = 1:p
        push!(stdevs, std(X[:, j]))
    end
    return stdevs
end


function column_vars(X)
    p = size(X, 2)

    # Initialize vector to store variances
    vars = Float64[]
    for j = 1:p
        push!(vars, var(X[:, j]))
    end
    return vars
end

t1 = time()
means = column_means(A)
stdevs = column_stdevs(A)
vars = column_vars(A)

res = hcat(means, stdevs, vars)
```

```
println(time() - t1)
```

4.2 Better Solution

```
using Base.Threads

function column_descriptives(X)
    p = size(X, 2)
    res = zeros(p, 3)
    @threads for j = 1:p
        res[j, 1] = mean(X[:, j])
        res[j, 3] = var(X[:, j], mean = res[j, 1])
        res[j, 2] = sqrt(res[j, 3])
    end
    return res
end

@time res2 = column_descriptives(A)
```

5 Challenge Problem 5.

The functions below read in a DataFrame object and modify the contents of one column based on whether or not a value appears in a column of a different DataFrame. The current performance can be improved considerably. Find a way to improve the performance using one (or several) of the techniques discussed to this point.

```
using Requests
using DataFrames
using StatsBase


diabetes =
    readtable(IOBuffer(get("https://raw.githubusercontent.com/bcbi/julia_tutorials/master/statistics/d


display(diabetes)

dia_consent =
    readtable("/Users/pstey/projects_code/julia_tutorials/performance_optim/diabetes_consent.csv")

# Pre-allocate column we will populate with true/false
# based on whether or not the patient has repeat visits
# and has consented to be in our study.

t1 = time()
n = nrow(diabetes)
n2 = nrow(dia_consent)
dia_consent[:include_patient] = falses(n2)

for i = 1:n
    println("Checking id:  $(diabetes[i, :patient_nbr])")
    id = diabetes[i, :patient_nbr]
    row_indcs = find(diabetes[:, :patient_nbr] .== id)
    if length(row_indcs) > 1
        m = nrow(dia_consent)
        idx = 0
        for j = 1:m
            if dia_consent[j, :patient_nbr] == id
                idx = j
            end
        end
        if dia_consent[idx, :study_consent]
            dia_consent[idx, :include_patient] = true
        end
    end
end


println("elapsed:  $(time() - t1)")
```

5.1 Better Solution

```
function count_visits(v)
    res = Dict{Int, Int}()
    n = length(v)
    for i = 1:n
        res[v[i]] = get(res, v[i], 0) + 1
    end
    return res
end


function include_patient(visits_df, consent_df)
    cnt_lkup = count_visits(visits_df[:patient_nbr])

    n = size(consent_df, 1)
    res = falses(n)

    for i = 1:n
        res[i] = cnt_lkup[consent_df[i, :patient_nbr]] > 1 && consent_df[i,
    :study_consent]
    end
    return res
end
@time a = include_patient(diabetes, dia_consent)
```

6 Challenge Problem 6.

You are given a dataset with three columns. Column 1 has patient IDs. Both columns 2 and 3 have ICD-10 codes for that patient. Your task is to optimize the code below that generates how frequently each pair of ICD-10 codes co-occur.

```
using Combinatorics

icd =
    readcsv("/Users/pstey/projects_code/julia_tutorials/performance_optim/repeat_ed_visits.csv")

icd_codes = vcat(icd[:, 2], icd[:, 3])
uniq_codes = unique(icd_codes)
n_codes = length(uniq_codes)

pairs = collect(combinations(uniq_codes, 2))
```

```
n_pairs = length(pairs)
pairs = hcat(zeros(Int, n_pairs))

n_patients = size(icd, 1)
cnts = zeros(Int, n_pairs)

for i = 1:n_pairs
    for j = 1:n_patients
        if pairs[i] == icd[j, 2:3]
            cnts[i] += 1
        end
    end
end
```

6.1 Better Solution

```
function count_cooccurence(icd_mat::Array{String, 2})
    n = size(icd_mat, 1)
    code_cnts = Dict{Array{String, 1}, Int}()

    for i = 1:n
        pair = icd_mat[i, :]
        code_cnts[pair] = get(code_cnts, pair, 0) + 1
    end
    return code_cnts
end


@time cnt = count_cooccurence(icd[:, 2:3])
@code_warntype count_cooccurence(icd[:, 2:3])
```