```julia
# ------------------------------------------------------------------------------
# Author: Philip Coyle
# Date Created: 06/26/2020
# ps4_Q2-Q3.jl
# ------------------------------------------------------------------------------
using Random, Distributions, Plots, Parameters, NLsolve
dir = "/Users/philipcoyle/Documents/School/University_of_Wisconsin/SecondYear/
Summer_2020/CodingBootcamp/ProblemSets/PS2/"
## Question 2
function matching_paper_slips(n::Int64)
    paper_draw = randperm(n)
    paper_order = 1:n

    location_match = paper_draw .== paper_order
    num_match = sum(location_match)

    return num_match
end

function sim_paper_slip(n_sim::Int64, n::Int64)
    num_match = zeros(n_sim)

    for i = 1:n_sim
        num_match[i] = matching_paper_slips(n)
    end
    return num_match
end



match_dist_10 = sim_paper_slip(10000,10)
match_dist_20 = sim_paper_slip(10000,20)
# Plotting
h10 = histogram(match_dist_10, bins=-0.5:1:10.5,yo label = "n = 10", normed=true,
bar_width=1);
h20 = histogram(match_dist_20, bins=-0.5:1:10.5, xticks = 0:1:10, label = "n = 20",
normed=true, bar_width=1);
plot(h10, h20, layout=(1,2))
savefig(dir * "Q2.pdf")

## Question 3
# Structures
@with_kw struct Parameter
    μ_s::Float64 = 0.06
    σ_s::Float64 = 0.06

    lb_e::Float64 = 0.0
    ub_e::Float64 = 0.06
    μ_e::Float64 = (lb_e + ub_e)/2

    n_sim::Int64 =  10000

    start_year = 30
    retire_year = 67
end

# Functions
function p_solve()
    params = Parameter()

    P_0 = [0.0]
    p_nlsolve!(x) = opt_p!(x, params)
    out = nlsolve(p_nlsolve!, P_0)

    P_opt = out.zero
```

```
    return P_opt, params
end

function opt_p!(x::Array{Float64}, params::Parameter)
    P = x[1]
    savings, earnings = savings_decison(P, params, false)

    R = savings - 10*earnings

    return R
end

function p_uncert_solve(X::Array{Float64})
    params = Parameter()

    p_uncert_nlsolve!(x) = opt_p_uncert!(x, params)
    out = nlsolve(p_uncert_nlsolve!, X)

    P_opt = out.zero

    return P_opt, params
end

function opt_p_uncert!(x::Array{Float64}, params::Parameter)
    P = x[1]
    avg_success = sim_savings_decision(P, params)

    R = 0.9 - avg_success

    return R
end

function sim_savings_decision(P::Float64, params::Parameter)
    @unpack n_sim = params

    save_enough = zeros(n_sim)
    Random.seed!(06262020);
    for i = 1:n_sim
        savings, earnings = savings_decison(P, params, true)
        save_enough[i] = savings >= 10*earnings
    end
    avg_success = sum(save_enough)/n_sim
    return avg_success
end

function savings_decison(P::Float64, params::Parameter, uncertainty::Bool)
    @unpack μ_s, σ_s, lb_e, ub_e, μ_e, start_year, retire_year = params

    # Allocate Space
    S = zeros(retire_year)
    S[start_year] = 100
    E = zeros(retire_year)
    E[start_year] = 100
    time_elapsed = retire_year-start_year

    if uncertainty == true
        dist_s = Normal(μ_s, σ_s)
        dist_e = Uniform(lb_e, ub_e)

        savings_gr = 1 .+ rand(dist_s, retire_year)
        earnings_gr = 1 .+ rand(dist_e, retire_year)
    else
        savings_gr = 1 .+ μ_s*ones(retire_year)
        earnings_gr = 1 .+ μ_e*ones(retire_year)
    end
```

```
129
130     for i = start_year + 1:retire_year
131         S[i-1] += P*E[i-1]
132         E[i-1] -= P*E[i-1]
133
134         S[i] = S[i-1]*savings_gr[i]
135         E[i] = E[i-1]*earnings_gr[i]
136     end
137
138
139     return S[retire_year], E[retire_year]
140 end
141
142 # Main Code
143 # Part A
144 P, params = p_solve()
145 savings, earnings = savings_decison(P[1], params, false)
146 savings >= floor(10*earnings)
147
148 # Part B
149 avg_fail = 1 - sim_savings_decision(P[1], params)
150
151 # Part C
152 P_0 = [0.035]
153 P, params = p_uncert_solve(P_0)
154 avg_success = sim_savings_decision(P[1], params)
155
```