MLBA homework1

In [1]:
```
# Put the package you need to use here
using Random
```

Q1 Inner Product

Define a function on the below block, and name it "Inner_Product"

In [2]:
```
function f(x,y)
           x'*y
       end
Inner_Product = f ;
```

Run the below blocks to get marks

In [3]:
```
# Q1 Test 1
Q1_1_v1 = [1, 2, 3]
Q1_1_v2 = [1, 2, 5]
println(Inner_Product(Q1_1_v1, Q1_1_v2))
```

```
20
```

In [4]:
```
# Q1 Test 2
Q1_2_v1 = [1.82, -2.56, 3.64]
Q1_2_v2 = [-1.43, -0.788, 5.3829]
println(Inner_Product(Q1_2_v1, Q1_2_v2))
```

```
19.008436000000003
```

In [5]:
```
#Q1 Test 3
Q1_3_v1 = [1 3 5]
Q1_3_v2 = [1, 3, 4]
println(Inner_Product(Q1_3_v1, Q1_3_v2))
```

```
DimensionMismatch("matrix A has dimensions (3,1), vector B has length 3")

Stacktrace:
 [1] generic_matvecmul!(C::Vector{Int64}, tA::Char, A::Matrix{Int64}, B::Vector{Int64}, _add::LinearAlgebra.MulAddMul{true, true, Bool, Bool})
   @ LinearAlgebra /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/LinearAlgebra/src/matmul.jl:713
 [2] mul!
   @ /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/LinearAlgebra/src/matmul.jl:129 [inlined]
 [3] mul!
   @ /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/LinearAlgebra/src/matmul.jl:275 [inlined]
 [4] *
   @ /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/LinearAlgebra/src/matmul.jl:113 [inlined]
 [5] f(x::Matrix{Int64}, y::Vector{Int64})
   @ Main ./In[2]:2
 [6] top-level scope
   @ In[5]:4
 [7] eval
   @ ./boot.jl:373 [inlined]
 [8] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1196
```

In [6]:
```
#Q1 Test 4
Random.seed!(1314)
Q1_4_v1 = rand(10)
Q1_4_v2 = rand(10)
println(Inner_Product(Q1_4_v1, Q1_4_v2))
```

```
2.525657893403556
```

In [7]:
```
# Q1 Test 5
Random.seed!(9487)
Q1_5_v1 = rand(10000)*100
Q1_5_v2 = rand(10000)*100
println(Inner_Product(Q1_5_v1, Q1_5_v2))
```

```
2.510400216244033e7
```

Q2 Exception Handling

Define a function on the below block, and name it "Strict_inner_Product"

In [8]:
```
function g(x,y)
    if(size(x) != size(y))
        dx = size(x)
        dy = size(y)
        @warn  " $dx vector can't do inner product with a $dy vector!"
        return 0
    else
    x'*y
    end
end
Strict_inner_Product = g;
```

Run the below blocks to get marks

In [9]:
```
# Q2 Test 1
Q2_1_v1 = [1, 2, 3]
Q2_1_v2 = [1, 2, 5]
println(Strict_inner_Product(Q2_1_v1, Q2_1_v2))
```

```
20
```

In [10]:
```
# Q2 Test 2
Q2_2_v1 = [1, 2, 3]
Q2_2_v2 = [1, 2, 5, 8]
println(Strict_inner_Product(Q2_2_v1, Q2_2_v2))
```

```
0
┌ Warning:  (3,) vector can't do inner product with a (4,) vector!
└ @ Main In[8]:5
```

In [11]:
```
# Q2 Test 3
Random.seed!(2468)
Q2_3_v1 = rand(1000)*100
Q2_3_v2 = rand(1000)*100
println(Strict_inner_Product(Q2_3_v1, Q2_3_v2))
```

```
2.4901545835740273e6
```

In [12]:
```
# Q2 Test 4
Q2_4_v1 = [1 2 3 4 5]
Q2_4_v2 = [3, 4, 5, 6]
println(Strict_inner_Product(Q2_4_v1, Q2_4_v2))
```

```
0
┌ Warning:  (1, 5) vector can't do inner product with a (4,) vector!
└ @ Main In[8]:5
```

In [13]:
```
# Q2 Test 5
Q2_5_v1 = rand(100)
Q2_5_v2 = rand(1001)'
println(Strict_inner_Product(Q2_5_v1, Q2_5_v2))
```

```
0
┌ Warning:  (100,) vector can't do inner product with a (1, 1001) vector!
└ @ Main In[8]:5
```

Q3 Advanced Exception Handling

Define a function on the below block, and name it "Identify_Wrong_Datatype"

In [14]:
```
function h(x,y)
    if (typeof(x) != typeof(y))
        #println("Warning! 3*1 vector can't do inner product with a 4*1 vector!")
        tx = typeof(x)
        ty = typeof(y)
        @warn  " $tx can't do inner product with $ty !"
        return 0
    elseif (typeof(x) ==String &&  typeof(y)== String)
        tx = typeof(x)
        ty = typeof(y)
            @warn  " $tx can't do inner product with $ty !"
    else
    x'*y
    end
end
Identify_Wrong_Datatype = h;
```

Run the below blocks to get marks

In [15]:
```
# Q3 Test 1
Q3_1_v1 = [1, 2, 3]
Q3_1_v2 = [1, 2, 5]
println(Identify_Wrong_Datatype(Q3_1_v1, Q3_1_v2))
```

```
20
```

In [16]:
```
# Q3 Test 2
Q3_2_v1 = [1, 2, 3]
Q3_2_v2 = "[1, 2, 5]"
println(Identify_Wrong_Datatype(Q3_2_v1, Q3_2_v2))
```

```
0
┌ Warning:  Vector{Int64} can't do inner product with String !
└ @ Main In[14]:6
```

In [17]:
```
# Q3 Test 3
Q3_3_v1 = "[1, 2, 3]"
Q3_3_v2 = [1, 2, 5]
println(Identify_Wrong_Datatype(Q3_3_v1, Q3_3_v2))
```

```
0
┌ Warning:  String can't do inner product with Vector{Int64} !
└ @ Main In[14]:6
```

In [18]:
```
# Q3 Test 4
Q3_4_v1 = "[1, 2, 3]"
Q3_4_v2 = "[1, 2, 5]"
println(Identify_Wrong_Datatype(Q3_4_v1, Q3_4_v2))
```

```
nothing
┌ Warning:  String can't do inner product with String !
└ @ Main In[14]:11
```

In [19]:
```
# Q3 Test 5
Q3_5_v1 = [1, 2, 3]
Q3_5_v2 = (1, 3, 4)
println(Identify_Wrong_Datatype(Q3_5_v1, Q3_5_v2))
```

```
0
┌ Warning:  Vector{Int64} can't do inner product with Tuple{Int64, Int64, Int64} !
└ @ Main In[14]:6
```

Q4 Eugene's calculator

Define a function on the below block, and name it "Happy_Birthday"

In [20]:
```
using Pkg
using Distributions
function Happy_Birthday(n,a)
    gd = Normal(0,10) #var is σ^2 so var = 100 can be written as σ = 10
    rrnd= map(x->round.(x), rand(gd,n,)) #rounding
    rrnd_int = convert(Vector{Int64}, rrnd) #convert to vector & integer
    println(rrnd_int)
    add = '+'
    sub = '-'
    mul = '*'
    div = '/'
    if length(a) > length(rrnd_int)
        @warn  " too many operands"
    else
        if a[1] == add
            a1 = rrnd_int[1] + rrnd_int[2]
        elseif a[1] == sub
            a1 = rrnd_int[1] - rrnd_int[2]
        elseif a[1] == mul
            a1 = rrnd_int[1] * rrnd_int[2]
        elseif a[1] == div
            a1 = rrnd_int[1] / rrnd_int[2]
        else
        println("too long I dont know what loop to choose yet")
    end
    end
end
```

Out[20]:
```
Happy_Birthday (generic function with 1 method)
```

Run the below blocks to get marks

In [21]:
```
# Q4 Test 1
Random.seed!(4129889)
Q4_1_integer = 2
Q4_1_operand = ['+']
println(Happy_Birthday(Q4_1_integer, Q4_1_operand))
```

```
[-8, -3]
-11
```

In [22]:
```
# Q4 Test 2
Random.seed!(800092000)
Q4_2_integer = 3
Q4_2_operand = ['+', '-']
println(Happy_Birthday(Q4_2_integer, Q4_2_operand))
```

```
[24, 3, 17]
27
```

In [23]:
```
# Q4 Test 3
Random.seed!(870887)
Q4_3_integer = 4
Q4_3_operand = ['+', '-', '*']
println(Happy_Birthday(Q4_3_integer, Q4_3_operand))
```

```
[11, -8, -5, -4]
3
```

In [24]:
```
# Q4 Test 4
Random.seed!(7414666)
Q4_4_integer = 5
Q4_4_operand = ['+', '-', '*', '/']
println(Happy_Birthday(Q4_4_integer, Q4_4_operand))
```

```
[-1, 6, 9, 22, -15]
5
```

In [25]:
```
# Q4 Test 5
Random.seed!(9481)
Q4_5_integer = 12
Q4_5_operand = ['+', '-', '*', '/', '/', '*', '+', '*', '-', '/', '+']
println(Happy_Birthday(Q4_5_integer, Q4_5_operand))
```

```
[5, -1, -3, -9, -10, -2, 12, -4, 18, 6, -9, -13]
4
```

Q5 Sunny's Crazy Idea

Define a function on the below block, and name it "Account_Manager"

In [26]:
```
function Account_Manager(n,q,p)
    if length(q) < length(p)
        y = length(p) - length(q)
        gd = Normal(0,10) #var is σ^2 so var = 100 can be written as σ = 10
        rrnd= map(x->round.(x), rand(gd,y,)) #rounding
        rrnd_int = convert(Vector{Int64}, rrnd)  #convert to vector & integer and back to previous
        println(rrnd_int)
        println("ignore negative I've square it  ")
        plast = last(p)
        p2 =setdiff(p, plast)
        b = (q' * p2 ) #dot product without the NA
        c = ((rrnd_int)[1]) # convert into scalar
        c_sq = c^2 #remove negative
        c_sqrt = sqrt(c_sq) #back to previous value
        c1 = c_sqrt * plast #remanding missing value dot product
        d = b + c1 #remanding missing value dot product
    elseif length(q) > length(p)
        y = length(q) - length(p)
        gd = Normal(0,10) #var is σ^2 so var = 100 can be written as σ = 10
        rrnd= map(x->round.(x), rand(gd,y,)) #rounding
        rrnd_int = convert(Vector{Int64} , rrnd) #convert to vector & integer
        println(rrnd_int)
        println("ignore negative I've square it  ")
        qlast = last(q)
        q2 =setdiff(q, qlast) #because there is repeat of 1 in q 4 it didnt work
        b = (q2' * p ) #dot product without the NA
        c = ((rrnd_int)[1]) # convert into scalar
        c_sq = c^2 #remove negative
        c_sqrt = sqrt(c_sq)
        c1 = c_sqrt * qlast
        d = b + c1
    else
        b = q' * p
    println(b)
    end
end
```

Out[26]:
```
Account_Manager (generic function with 1 method)
```

Run the below blocks to get marks

In [27]:
```
# Q5 Test 1
Q5_1_name = ["Sunny", "Hsin", "Eric"]
Q5_1_quantity = [0, 1, 1]
Q5_1_price = [1, 10, 100]
println(Account_Manager(Q5_1_name, Q5_1_quantity, Q5_1_price))
```

```
110
nothing
```

In [28]:
```
# Q5 Test 2
Q5_2_name = ["Sunny", "Hsin", "Eric", "Breakfast", "Dinner", "Concert"]
Q5_2_quantity = [0, 1, 1, 10, 20]
Q5_2_price = [1, 10, 100, 5, 50, 500]
println(Account_Manager(Q5_2_name, Q5_2_quantity, Q5_2_price))
```

```
[1]
ignore negative I've square it
1660.0
```

In [29]:
```
# Q5 Test 3
Q5_3_name = ["Sunny", "Hsin", "Eric", "Breakfast", "Dinner", "Concert"]
Q5_3_quantity = [0, 1, 1, 10, 20, 50]
Q5_3_price = [1, 10, 100, 5, 50]
println(Account_Manager(Q5_3_name, Q5_3_quantity, Q5_3_price))
```

```
[10]
ignore negative I've square it
DimensionMismatch("first array has length 4 which does not match the length of the second, 5.")

Stacktrace:
 [1] dot(x::Vector{Int64}, y::Vector{Int64})
   @ LinearAlgebra /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/LinearAlgebra/src/generic.jl:910
 [2] *
   @ /Users/sabae/src/julia/usr/share/julia/stdlib/v1.7/LinearAlgebra/src/adjtrans.jl:291 [inlined]
 [3] Account_Manager(n::Vector{String}, q::Vector{Int64}, p::Vector{Int64})
   @ Main ./In[26]:26
 [4] top-level scope
   @ In[29]:5
 [5] eval
   @ ./boot.jl:373 [inlined]
 [6] include_string(mapexpr::typeof(REPL.softscope), mod::Module, code::String, filename::String)
   @ Base ./loading.jl:1196
```

In [30]:
```
# Q5 Test 4
Q5_4_name = ["Sunny", "Hsin", "Eric", "Breakfast", "Dinner", "Concert"]
Q5_4_quantity = [0, 1, 1, 10, 20]
Q5_4_price = [1, 10, 100, 5, 50]
println(Account_Manager(Q5_4_name, Q5_4_quantity, Q5_4_price))
```

```
1160
nothing
```

In [31]:
```
# Q5 Test 5
Q5_5_name = ["Sunny", "Hsin", "Eric", "Breakfast", "Dinner", "Concert"]
Q5_5_quantity = [0, 1, 1, 10]
Q5_5_price = [1, 10, 100, 5]
println(Account_Manager(Q5_5_name, Q5_5_quantity, Q5_5_price))
```

```
160
nothing
```

In [ ]: