#julia programming

# require the type of `k` to be the same

# as the type of the elements of `input`

# and that type `T` is an integer type;

# `Vector` is an 1D array

function sumpairs(input::Vector{T}, k::T) where T<:Integer

# `map` is a dictionary with keys of type `T` and

# values of type `Int` (Int32 or Int64,

# depending on your OS pointer size)

map = Dict{T,Int}()

count = 0

# if you only need values from a collection,

# use `for val in collection` syntax

for n in input

complement = k - n

# `get(collection, key, default)` will

# return the default value if key is missing

compl\_freq = get(map, complement, 0)

# judging by the algorithm, all values in `map` must be strictly positive

# so, 0 can be only encountered as a default return value

if compl\_freq != 0

count += 1

if compl\_freq == 1

delete!(map, complement)

else

map[complement] -= 1

end

else

# using `get` again for shorter code

map[n] = get(map, n, 0) + 1

end

end

return count

end

println("Total pairs ", sumpairs([1,2,3,6,7,8,9,1], 10))

**or- this either work**

function sumpairs(input::AbstractVector{<:Integer}, k::Integer)

map = Dict{eltype(input),eltype(input)}()

count = 0

for (i,v) ∈ enumerate(input)

c = k - v

if haskey(map, c)

freq = map[c]-1

count += 1

freq == 0 ? delete!(map, c) : (map[c] = freq)

else

map[v] = haskey(map, v) ? (map[v]+1) : 1

end

end

return count

end

println("Total pairs ", sumpairs([1,2,3,6,7,8,9,1], 10))

**function sumpairs(inputvector::Vector{Int64},k::Int64) # first create an empty dictionary mydict = Dict{Int64,Int64}() # Put the inputvector elements into the mydict for n in inputvector mydict[n] = 1 end # create an outputvector outputvector = Vector{Tuple{Int64,Int64}}(undef,0) # Now loop throught the inputvector again for number in inputvector complement = k - number if haskey(mydict,number) && haskey(mydict,complement) push!(outputvector,(number,complement)) end end # Now return the outputvector return outputvector end resultvector = sumpairs([1,2,3,6,7,8,9,1],10) println("RESULT = $resultvector") println("COUNT = ",length(resultvector))**

**The output of the program is**

**RESULT = Tuple{Int64,Int64}[(1, 9), (2, 8), (3, 7), (7, 3), (8, 2), (9, 1), (1, 9)] COUNT = 7**