```lua
local Util = setmetatable({}, nil)

Util.EPSILON = 0.00001

function Util.DeepCopy(object)
    local lookup_table = {}
    local function _copy(object)
        if type(object) ~= "table" then
            return object
        elseif lookup_table[object] then
            return lookup_table[object]
        end
        local new_table = {}
        lookup_table[object] = new_table
        for index, value in pairs(object) do
            new_table[_copy(index)] = _copy(value)
        end
        return setmetatable(new_table, getmetatable(object))
    end
    return _copy(object)
end

-- Merges two tables, with values from table2 taking precedence over values from
   table1
function Util.MergeTables(table1, table2)
    local newTable = {}

    if (table1) then
        for key, value in pairs(table1) do
            newTable[key] = value
        end
    end

    if (table2) then
        for key, value in pairs(table2) do
            newTable[key] = value
        end
    end
    return newTable
end

-- Removes a value from an array
function Util.FindAndRemove(tab, item)
    for i, testItem in ipairs(tab) do
        if (testItem == item) then
```

```
            table.remove(tab, i)
            return true
        end
    end

    return false
end

function Util.DegToRad(degrees)
    return degrees * math.pi / 180.0
end

function Util.RadToDeg(radians)
    return radians*180.0/math.pi
end

function Util.DeclareGlobal(name, value)
    rawset(_G, name, value or {})
end

function Util.UndeclareGlobal(name)
    rawset(_G, name, nil)
end

function Util.GlobalDeclared(name)
    return rawget(_G, name) ~= nil
end

local function denyNewIndex(_ENV, var, val)
    error("Attempt to write undeclared object property: \"" .. tostring(var) ..
"\"")
end

local function denyUndefinedIndex(_ENV, var)
    error("Attempt to read undeclared object property: \"" .. tostring(var) .. "\"")
end

function Util.lockObjectProperties(...)
    for _, object in ipairs(arg) do
        local meta = getmetatable(object)
        if (meta) then
            assert(meta.__newindex == nil, "Can't lock object - it already has a
__newindex set")
            meta.__newindex = denyNewIndex
        else
```

```
            meta = {__newindex = denyNewIndex}
        end
    end
end

function Util.unlockObjectProperties(...)
    for _, object in ipairs(arg) do
        local meta = getmetatable(object)
        assert(meta and meta.__newindex == denyNewIndex, "Can't unlock object - it
wasn't locked with lockObjectProperties")
        meta.__newindex = nil
    end
end

function Util.errorOnUndefinedProperty(...)
    for _, object in ipairs(arg) do
        local meta = getmetatable(object)
        if (meta) then
            assert(meta.__index == nil, "Can't set object to error on undefined -
it already has an __index set")
            meta.__index = denyUndefinedIndex
        else
            meta = {__index = denyUndefinedIndex}
        end
    end
end

function Util.printProps(obj, message)
    if (message) then
        print(message)
    end
    for k, v in pairs(obj) do
        print("\t", tostring(k) .. ":", v)
    end
end

function Util.lerp(a, b, t)
    return a + (b - a) * t
end

function Util.sign(a)
    if a < 0 then
        return -1
    else
        return 1
```

```
    end
end

-- override print() function to improve performance when running on device
-- and print out file and line number for each print
local original_print = print
if ( system.getInfo("environment") == "device" ) then
    print("Print now going silent. With Love, util.lua")
   print = function() end
else
    print = function(message)
        local info = debug.getinfo(2)
        local source_file = info.source
        --original_print(source_file)
        local debug_path = source_file:match('%a+.lua')
        if debug_path then
            debug_path = debug_path  ..' ['.. info.currentline ..']'
        end
        original_print(((debug_path and (debug_path..": ")) or "")..message)
    end
end

-----------------------------------------------------------------
-- Array Utilities
-----------------------------------------------------------------

--Concatenate array table B onto the end of array table A
function Util.arrayConcat(A,B)
    local iA = #A
    for i = 1, #B do
        A[i+iA] = B[i]
    end
    return A
end

--Concats B onto the end of A but doesn't allow duplicates from B in A
-- if endA is set to #A, then we won't be checking for duplicates in B while adding
   to A
function Util.arrayConcatUnique(A,B,endA)
    endA = endA or #A
    local offset = 0
    for i = 1, #B do
        if Util.arrayContains(A,B[i],endA) then offset = offset + 1
        else
            A[i+endA-offset] = B[i]
```

```
        end
    end
    return A
end

--endA limits index depth we check for duplicates in array A
--returns index in which the obj was found in the array A
function Util.arrayContains(A, obj, endA)
    if (endA and endA > #A) or not endA then
        endA = #A
    end
    --pretty sure this does the above, but the above is more clear
    --endA = (endA and endA < #A and endA) or #A
    for i=1, endA do
        if A[i]==obj then return true, i end
    end
    return false, nil
end

--Return a new table with all mutually exclusive elements in A & B
function Util.getUniqueArray(A,B)
    local unique = {}
    local duplicatesInB = {} --indices of diplicates found in B
    for i=1, #A do
        local doesContain, indexB = Util.arrayContains(B, A[i])
        if not doesContain then
            table.insert(unique,A[i]) -- object A[i] is unique to A
        else
            table.insert(duplicatesInB, indexB) --B[indexB] is equal to A[i]
        end
    end
    --Dup check for all non-known duplicates in B
    --All elements {in A and not in B} are in the unique table.
    --Sort it so we can traverse indices linearly
    table.sort(duplicatesInB)
    local prevStartI = 1
    for i = 1, #duplicatesInB do
        local startI, endI = prevStartI, duplicatesInB[i]-1 --end before dup
        for j = startI, endI do
            if not Util.arrayContains(A, B[j]) then
                table.insert(unique,B[j])
            end
        end
        prevStartI = endI+2 --skip over the duplicate object index
    end
```

```
    --Any leftovers after the last known duplicate in B
    if prevStartI <= #B then
        for i=prevStartI, #B do
            if not Util.arrayContains(A, B[i]) then
                table.insert(unique,B[i])
            end
        end
    end
    return unique
end

--returns new table of elements in A that aren't in B
function Util.arrayNot(A,B)
    local notSet = {}
    for i=1, #A do
        if not Util.arrayContains(B,A[i]) then
            table.insert(notSet, A[i])
        end
    end
    return notSet
end


return Util
```