```
function _flatten(ex::Expr)
    head = ex.head
    args = _flatten.(ex.args)
    if head == :block
        newargs = Any[]
        for e in args
            if e isa Expr && e.head == :block
                append!(newargs, e.args)
            else
                push!(newargs, e)
            end
        end
        return Expr(:block, newargs...)
    elseif head == :(=) && args[2] isa Expr && args[2].head == :block
        newargs = args[2].args
        newargs[end] = Expr(:(=), args[1], newargs[end])
        return Expr(:block, newargs...)
    elseif head == :call && args[1]==:scalar && args[2] isa Expr && args[2].head ==
:block
        newargs = args[2].args
        newargs[end] = Expr(:call, args[1], newargs[end])
        return Expr(:block, newargs...)
    else
        return Expr(head, args...)
    end
end
_flatten(e) = e

function removelinenumbernode(ex::Expr)
    if ex.head == :block
        return Expr(:block, (removelinenumbernode(e) for e in ex.args if !(e isa
LineNumberNode))...)
    else
        return ex
    end
end
removelinenumbernode(ex) = ex

const tensoroperationsfunctions = (:similar_from_indices,
                                   :cached_similar_from_indices,
                                   :add!, :trace!, :contract!,
                                   :scalar, :IndexError)
function addtensoroperations(ex::Expr)
    if ex.head == :call && ex.args[1] in tensoroperationsfunctions
        return Expr(ex.head, GlobalRef(TensorOperationsXD, ex.args[1]),
                        (addtensoroperations(ex.args[i]) for i in
2:length(ex.args))...)
    else
        return Expr(ex.head, (addtensoroperations(e) for e in ex.args)...)
    end
end
addtensoroperations(ex) = ex
```