```
"""
    struct CartesianSpace <: EuclideanSpace{ℝ}

A real euclidean space `ℝ^d`, which is therefore self-dual. `CartesianSpace` has no
additonal structure and is completely characterised by its dimension `d`. This is
the
vector space that is implicitly assumed in most of matrix algebra.
"""
struct CartesianSpace <: EuclideanSpace{ℝ}
    d::Int
end
CartesianSpace(d::Integer = 0; dual = false) = CartesianSpace(Int(d))
function CartesianSpace(dim::Pair; dual = false)
    if dim.first === Trivial()
        return CartesianSpace(dim.second; dual = dual)
    else
        msg = "$(dim) is not a valid dimension for CartesianSpace"
        throw(SectorMismatch(msg))
    end
end
CartesianSpace(dim::AbstractDict; dual = false) = CartesianSpace(dim...; dual =
dual)

Base.getindex(::RealNumbers) = CartesianSpace
Base.getindex(::RealNumbers, d::Int) = CartesianSpace(d)
Base.:^(::RealNumbers, d::Int) = CartesianSpace(d)

# Corresponding methods:
#-------------------------
dim(V::CartesianSpace) = V.d
Base.axes(V::CartesianSpace) = Base.OneTo(dim(V))

Base.oneunit(::Type{CartesianSpace}) = CartesianSpace(1)
⊕(V1::CartesianSpace, V2::CartesianSpace) = CartesianSpace(V1.d+V2.d)
fuse(V1::CartesianSpace, V2::CartesianSpace) = CartesianSpace(V1.d*V2.d)
flip(V::CartesianSpace) = V

infinum(V1::CartesianSpace, V2::CartesianSpace) = CartesianSpace(min(V1.d, V2.d))
supremum(V1::CartesianSpace, V2::CartesianSpace) = CartesianSpace(max(V1.d, V2.d))

Base.show(io::IO, V::CartesianSpace) = print(io, "ℝ^$(V.d)")
Base.show(io::IO, ::Type{CartesianSpace}) = print(io, "CartesianSpace")
```