An Example

The JuliaMono font

Code example making heavy use of Unicode from https://github.com/JuliaArrays/StaticArrays.jl/issues/537#issuecomment-439863841

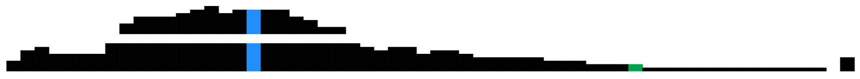
```julia
function T(𝛉::AbstractArray,
           𝒞::Tuple{AbstractArray,
           Vararg{AbstractArray}},
           𝒟::Tuple{AbstractArray, Vararg{AbstractArray}})
    ⊗ = kron
    l = length(𝛉)
    𝐈ₗ = SMatrix{l,l}(1.0I)
    𝐈ₘ = SMatrix{1,1}(1.0I)
    𝐓 = @SMatrix zeros(l,l)
    N = length(𝒟[1])
    ℳ, ℳʹ = 𝒟
    Λ₁, Λ₂ = 𝒞
    𝚲ₙ = @MMatrix zeros(4,4)
    𝐞₁ = @SMatrix [1.0; 0.0; 0.0]
    𝐞₂ = @SMatrix [0.0; 1.0; 0.0]
    for n = 1:N
        index = SVector(1,2)
        𝚲ₙ[1:2,1:2] .=  Λ₁[n][index,index]
        𝚲ₙ[3:4,3:4] .=  Λ₂[n][index,index]
        𝐦    = 𝑛𝑜𝑚(ℳ[n])
        𝐦ʹ   = hom(ℳʹ[n])
        𝐔ₙ   = (𝐦 ⊗ 𝐦ʹ)
        ∂ₓ𝐮ₙ = [(𝐞₁ ⊗ 𝐦ʹ) (𝐞₂ ⊗ 𝐦ʹ) (𝐦 ⊗ 𝐞₁) (𝐦 ⊗ 𝐞₂)]
        𝐁ₙ   = ∂ₓ𝐮ₙ * 𝚲ₙ * ∂ₓ𝐮ₙ'
        𝚺ₙ   = 𝛉' * 𝐁ₙ * 𝛉
        𝚺ₙ⁻¹ = inv(𝚺ₙ)
        𝐓₁   = @SMatrix zeros(Float64,l,l)
        for k = 1:l
            𝐞ₖ = 𝐈ₗ[:,k]
            ∂𝐞ₖ𝚺ₙ = (𝐈ₘ ⊗ 𝐞ₖ') * 𝐁ₙ * (𝐈ₘ ⊗ 𝛉) + (𝐈ₘ ⊗ 𝛉') * 𝐁ₙ * (𝐈ₘ ⊗ 𝐞ₖ)
            # Accumulating the result in 𝐓₁ allocates memory,
            # even though the two terms in the
            # summation are both SArrays.
            𝐓₁ = 𝐓₁ + 𝐔ₙ * 𝚺ₙ⁻¹ * (∂𝐞ₖ𝚺ₙ) * 𝚺ₙ⁻¹ * 𝐔ₙ' * 𝛉 * 𝐞ₖ'
        end
        𝐓 = 𝐓 + 𝐓₁
    end
    T
end
```

Colored console graphs produced by `Benchmarktools.jl`

```
using BenchmarkTools

@benchmark sum(rand(1000))
```

```
BenchmarkTools.Trial: 10000 samples with 10 evaluations.
 Range (min … max):  1.175 μs … 87.624 μs  ┊ GC (min … max): 0.00% … 93.14%
 Time  (median):     1.239 μs              ┊ GC (median):    0.00%
 Time  (mean ± σ):   1.337 μs ±  2.242 μs  ┊ GC (mean ± σ):  6.26% ±  3.70%
```



```
  1.17 μs        Histogram: frequency by time        1.39 μs <

 Memory estimate: 7.94 KiB, allocs estimate: 1.
```

Some output using ANSI escape codes

```
println("Some tests:")
printstyled("- Red ", color=:red)
printstyled("Green ", color=:green)
print("Black and ")
printstyled("Bold underline green\n", color=:green, bold=true, underline=true)
printstyled("- Normal black for comparison\n")
printstyled("- Hidden is implemented as light/dimmed\n", hidden=true)
printstyled("-  Hidden  is  implemented  as  light/dimmed\n",  hidden=true,
italic=true)
printstyled("- Green background\n", color=:green, reverse=true)
printstyled("- A 256 bit color\n", color=142)
printstyled("- Some italic\n", italic=true)
printstyled("- and blue bold italic\n", italic=true, bold=true, color=:blue)
```

```
Some tests:
- Red Green Black and Bold underline green
- Normal black for comparison
- Hidden is implemented as light/dimmed
- Hidden is implemented as light/dimmed
- Green background
- A 256 bit color
- Some italic
- and blue bold italic
```

Structure of floating point numbers

```julia
function printbitsf64(x::Float64)
    s = bitstring(x)
    printstyled(s[1], color = :blue, reverse=true)
    printstyled(s[2:12], color = :green, reverse=true)
    printstyled(s[13:end], color=:red, bold=true, reverse=true)
    print("\n")
end

printbitsf64(27.56640625)
```

```
0100000000111011100100010000000000000000000000000000000000000000
```

Machine epsilon

```julia
Eps=0.5
while 1 != 1 + Eps
    Eps /= 2
    printbitsf64(1+Eps)
end
```

```
0011111111110100000000000000000000000000000000000000000000000000
0011111111110010000000000000000000000000000000000000000000000000
0011111111110001000000000000000000000000000000000000000000000000
0011111111110000100000000000000000000000000000000000000000000000
0011111111110000010000000000000000000000000000000000000000000000
0011111111110000001000000000000000000000000000000000000000000000
0011111111110000000100000000000000000000000000000000000000000000
0011111111110000000010000000000000000000000000000000000000000000
0011111111110000000001000000000000000000000000000000000000000000
0011111111110000000000100000000000000000000000000000000000000000
0011111111110000000000010000000000000000000000000000000000000000
0011111111110000000000001000000000000000000000000000000000000000
0011111111110000000000000100000000000000000000000000000000000000
0011111111110000000000000010000000000000000000000000000000000000
0011111111110000000000000001000000000000000000000000000000000000
0011111111110000000000000000100000000000000000000000000000000000
0011111111110000000000000000010000000000000000000000000000000000
0011111111110000000000000000001000000000000000000000000000000000
0011111111110000000000000000000100000000000000000000000000000000
```

```
0011111111110000000000000000000010000000000000000000000000000000
0011111111110000000000000000000001000000000000000000000000000000
0011111111110000000000000000000000100000000000000000000000000000
0011111111110000000000000000000000010000000000000000000000000000
0011111111110000000000000000000000001000000000000000000000000000
0011111111110000000000000000000000000100000000000000000000000000
0011111111110000000000000000000000000010000000000000000000000000
0011111111110000000000000000000000000001000000000000000000000000
0011111111110000000000000000000000000000100000000000000000000000
0011111111110000000000000000000000000000010000000000000000000000
0011111111110000000000000000000000000000001000000000000000000000
0011111111110000000000000000000000000000000100000000000000000000
0011111111110000000000000000000000000000000010000000000000000000
0011111111110000000000000000000000000000000001000000000000000000
0011111111110000000000000000000000000000000000100000000000000000
0011111111110000000000000000000000000000000000010000000000000000
0011111111110000000000000000000000000000000000001000000000000000
0011111111110000000000000000000000000000000000000100000000000000
0011111111110000000000000000000000000000000000000010000000000000
0011111111110000000000000000000000000000000000000001000000000000
0011111111110000000000000000000000000000000000000000100000000000
0011111111110000000000000000000000000000000000000000010000000000
0011111111110000000000000000000000000000000000000000001000000000
0011111111110000000000000000000000000000000000000000000100000000
0011111111110000000000000000000000000000000000000000000010000000
0011111111110000000000000000000000000000000000000000000001000000
0011111111110000000000000000000000000000000000000000000000100000
0011111111110000000000000000000000000000000000000000000000010000
0011111111110000000000000000000000000000000000000000000000001000
0011111111110000000000000000000000000000000000000000000000000100
0011111111110000000000000000000000000000000000000000000000000010
0011111111110000000000000000000000000000000000000000000000000001
0011111111110000000000000000000000000000000000000000000000000000
```

Errors and Warnings

```
3 < "four"
```

```
MethodError: no method matching isless(::Int64, ::String)

Closest candidates are:
  isless(::Missing, ::Any)
   @ Base missing.jl:87
  isless(::Any, ::Missing)
   @ Base missing.jl:88
  isless(::Real, ::Union{StatsBase.PValue, StatsBase.TestStat})
   @ StatsBase ~/.julia/packages/StatsBase/ebrT3/src/statmodels.jl:91
  ...


Stacktrace:
 [1] <(x::Int64, y::String)
   @ Base ./operators.jl:352
 [2] top-level scope
   @ In[24]:2
```

The `@warn` macro writes to the `stderr` channel:

```
println(π^2)
@warn  "Last warning!"
1 + 41
```

9.869604401089358

```
┌ Warning: Last warning!
└ @ Main In[25]:2
```

42