

Understanding Your Struct Toolbox

Sam Buercklin



<https://github.com/sbuercklin/StructToolbox>

Structs aren't just “Fancy NamedTuples”

- Easy to think of structs as just objects for grouping fieldnames
- Much more fundamental
 - Dispatchable construct
 - Mutable (sometimes)
 - Often user-facing
- Lots of Julia assumes you're defining custom structs

Struct Equality

Equality

```
struct Planet{S}
    name::S
    places::Vector{S}
end
```

```
julia> j1 = Planet("Jupiter", ["Red Dot"]);
julia> j2 = Planet("Jupiter", ["Red Dot"]);
julia> @assert j1 == j2 "These planets are not equal!"
ERROR: AssertionError: These planets are not equal!
```

Equality

```
struct Planet{S}
    name::S
    places::Vector{S}
end
```

```
function Base.==(a::Planet, b::Planet)
    return a.name == b.name && a.places == b.places
end
function Base.hash(p::Planet{S}, x::UInt) where{S}
    return hash(p.places, hash(p.name, hash(Planet{S}, x)))
end
```

```
julia> j1 = Planet("Jupiter", ["Red Dot"]);
julia> j2 = Planet("Jupiter", ["Red Dot"]);
julia> @assert j1 == j2 "These planets are not equal!"
ERROR: AssertionError: These planets are not equal!
```

```
julia> @assert j1 == j2 "These planets are equal!"
```

The Easy Way

- Doing this from scratch is tedious and error-prone
 - AutoHashEquals.jl will do this automatically
 - Gotcha: hash the type in addition to field values

```
using AutoHashEquals
@auto_hash_equals struct Planet{S}
    name::S
    places::Vector{S}
end
```

Struct Iteration

Iteration

```
struct Planet
    name::String
    places::Vector{String}
end

function Base.iterate(p::Planet, state=nothing)
    if isempty(p)
        return nothing
    end
    idx = something(state, firstindex(p))
    if idx > lastindex(p)
        return nothing
    else
        return (p.places[idx], nextind(p.places, idx))
    end
end
```

```
julia> earth = Planet("Earth", list_of_places);
julia> for place in earth
        println("I visited $place")
    end
I visited Mount Bromo
I visited Iguazú Falls
I visited A beautiful fjord
```


Iteration as an Interface

- Gives us better abstraction of types with contents
 - Iterate over a system as particles, or a mesh as cells, or...
 - Stop leaky abstractions!
- Unlocks higher order behaviors using iteration as an *interface*
 - *Building Confidently in Julia with Interface Driven Design*
 - <https://youtu.be/mMO9NzkTxL0>

```
julia> for (country, place) in zip(countries, earth)
        println("I visited $place in $country")
    end
I visited Mount Bromo in Indonesia
I visited Iguazú Falls in Argentina and Brazil
I visited A beautiful fjord in Norway
```

Pretty Printing

Pretty-Printing

The default, 2-arg `Base.show(io, x)` method prints a representation which can be parsed

```
julia> planet1  
Planet{String}("Earth", ["Mount Bromo", "Iguazú Falls", "A beautiful fjord"])
```

This is useful for copying small structs around, but we often want a more interactive workflow

Pretty-Printing

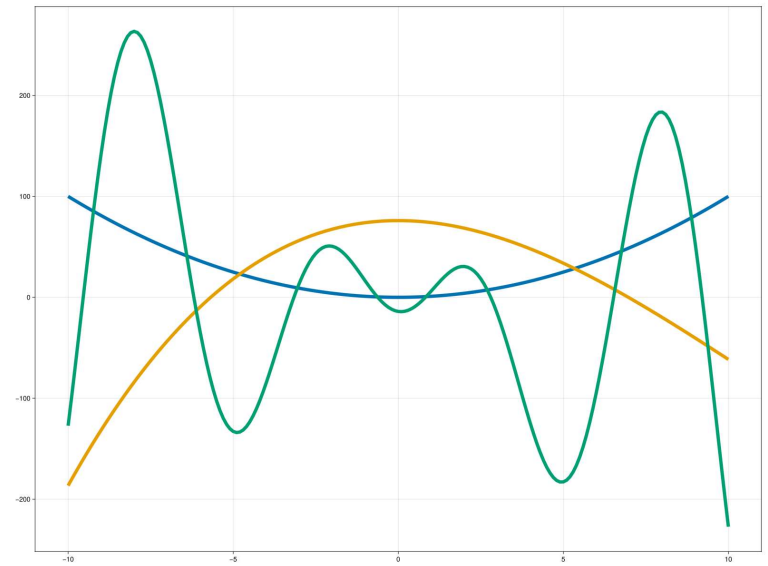
```
function Base.show(io::IO, ::MIME"text/plain", p::Planet)
    if get(io, :compact, false)
        Base.print(io, "Planet($(p.name))")
    else
        place_string = "[" * join(p.places, ',') * "]"
        Base.print(io, "Planet(name=$(p.name), places=$place_string)")
    end
end
```

```
julia> planet1
Planet(name=Earth, places=[Mount Bromo,Iguazú Falls,A beautiful fjord])
```

```
julia> [planet1 planet2 planet3 planet4]
1×4 Matrix{Planet}:
 Planet(Earth) ... Planet(Uranus)
```

The depths of Base.show

- Useful for interactive development
 - Summary statistics
 - Common names
 - Hide implementation details
- Different MIME types
 - Render to markdown, LaTeX, images
 - Open new/external displays (Makie)



Mutability and Identity

Mutable is more than Mutation (Finalizers)

```
mutable struct Planet
    const name::String
    const favorite_place::String

    function Planet(n::String, fp::String)
        return finalizer(say_goodbye, new(n, fp))
    end
end
```

```
julia> planets = [Planet("Mars", " ... "), ...]
```

```
julia> present_planets(planets)
```

```
julia> exit() # or GC.gc()
```

```
So long, Earth! I had a great time visiting JuliaCon 2025
So long, Mercury! I had a great time visiting Caloris Montes
So long, Jupiter! I had a great time visiting Europa
```

Mutation is about identity

- Instances of mutable structs are distinguishable
 - Identity/distinguishability gives mutation, not the other way
 - `===` vs `==` (egal vs equal)
- Finalizers give us a way to exploit identity with Julia's GC
- Most useful for things like C-FFI
 - Managing memory in other languages, packages
 - Underpins wrappers of external packages

Thank You!