First steps:

1. cd into project directory
2. `(Network_qse) pkg> activate .` [1]
3. run `include("./src/Network_qse.jl")`
4. if packages missing `pkg> add ExamplePackage`
5. now all functions can be called in Julia prompt: `Network_qse.functionName()`
6. run tests in pkg: `(Network_qse) pkg> test`

Use Julia

Example screenshot on how to call AtomProperties (this is a defined Datatype, see DataTypes.jl) in the Julia prompt

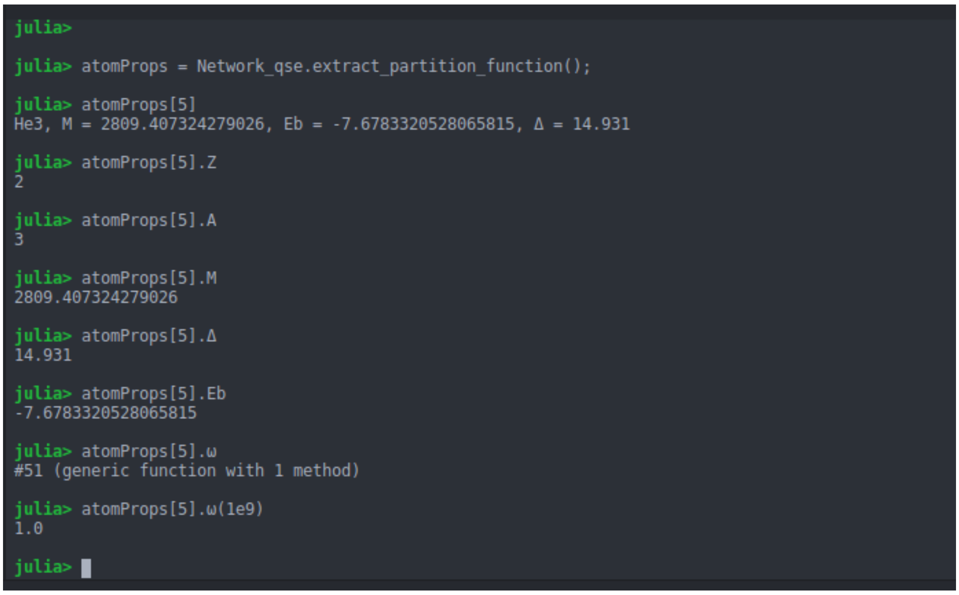

Figure 0.1: Example for calling extract_partition_function() in IO.jl. This array stores all input that is needed (except constants).

Calling `atomProps.ω` returns a T-dependent partition (type) function, with Temperature (Kelvin) as argument.
If looking for the atomic properties of a specific element, call
`filter(i -> (atomProps[i].name == "Fe56"), 1:size(atomProps,1))`
This returns the index or array element of a specific element, here Fe56, stored in atomProps, see screenshot.

Calculations for E_B, Δ, M defined in `DataTypes.jl`

[1] enter package manager (pkg) with closing square bracket. Exit with Backspace

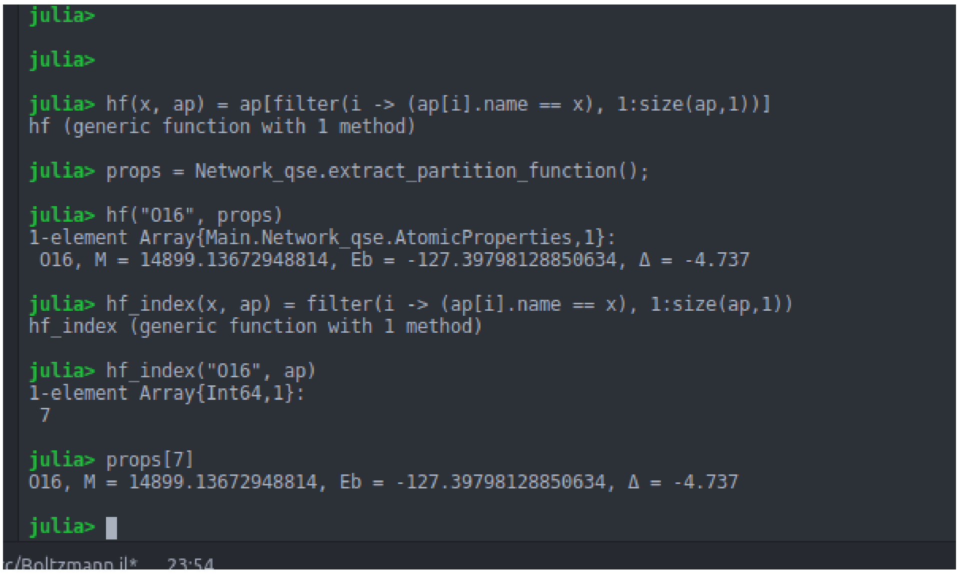

Figure 0.2: Inline Functions can be defined by simply `f(x) = ...`.

Boltzmann.jl

`prefactor(pf)` returns a temperature and density dependent function, all in CGS units (`cococubed.asu.edu/code_pages/nse.shtml`)

$$\frac{A}{N_A \rho} \cdot \omega(T) \left(\frac{2\pi k_B T M_i}{h^2} \right)^{1.5}$$

So prefactor of He3 at $T = 10^9$, $\rho = 10^7$ (cgs units) would be called with:

```
Network_qse.prefactor(ap[5])(1e9, 1e7)
```

`nse_condition!(res, μ, T, ρ, y, ap; precision=4000)` calculates

$$\log \left(\sum_i X_i \right) = 0,$$
$$\log \left(\frac{\sum_i \frac{Z_i}{A_i} X_i}{Y_e} \right) = 0$$

The function changes its argument `res`, similar to Fortran subroutines [2].

NLsolve

NLsolve[3] takes the (set to zero) function with 2 arguments. The return value of function (one wants to set zero) and the solution for chemical potential μ_p, μ_n. If `f` has more

[2] Convention in julia is to add "!" in the function name. The solver I use requires a function "with no return value"

[3] `github.com/JuliaNLSolvers/NLsolve.jl`

arguments, anonymous functions can resolve problem:
`(output, x) -> f(output, x, T, rho, y)` (in my case T, rho, y is known..).

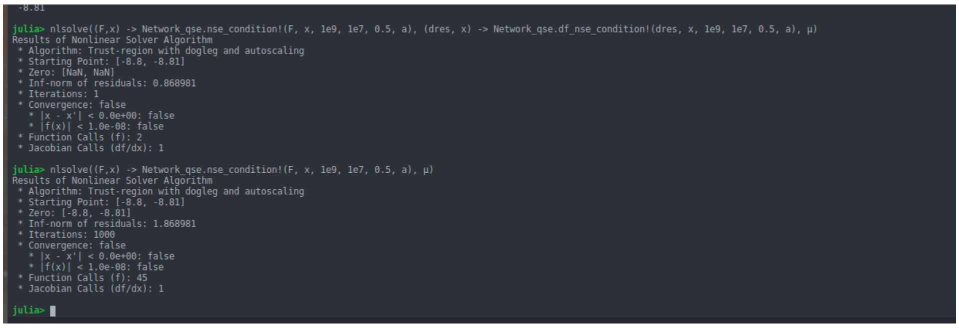

Figure 0.3: Example output for calling NLsolve for e.g. $T = 10^9$ and $\rho = 10^7$