Getting Started

Laptop/DelftBlue

Course repository

→ https://github.com/carstenbauer/JuliaDelft24

Remarks

→ git pull

→ julia --version == 1.10.5

If not:

- → juliaup add 1.10
- → juliaup default 1.10

Julia on Your Laptop

A basic Julia test

Open a terminal in `JuliaDelft24`

Start Julia

```
julia --project
```

Run this code

```
julia> using SysInfo
julia> sysinfo()
```

Julia on DelftBlue

Access DelftBlue via SSH

Terminal

```
ssh <netid>@login.delftblue.tudelft.nl
```

VS Code

```
Remote-SSH: Connect to Host...
```

Loading software modules

What we need for the course

```
module use /projects/julia/modulefiles
module use juliahpc
module use nvhpc # MPI+CUDA
```

A basic Julia test (on DelftBlue)

Open a terminal in

```
cd /scratch/$USER/JuliaDelft24
```

Start Julia

```
julia --project
```

Run this code

```
julia> using SysInfo
julia> sysinfo()
```

Jobs are scheduled with SLURM.

Submit a job:

```
sbatch job_script.sh
```

Check on your queued/running jobs:

```
squeue --me
```

A few nodes are reserved for the course.

2 CPU nodes

- 2x Intel Cascade Lake
- 185 GB memory
- 48 cores total

2 GPU nodes

- 2x AMD Zen 2
- 250 GB memory
- 48 cores total
- 4x NVIDIA V100S

Accessing compute nodes (with VS Code)

On the target node

```
module load code
code tunnel
```

On your laptop

In VS Code:

`Remote Tunnels: Connect to Tunnel`

Put the Julia depot on the parallel file system.

`JULIA_DEPOT_PATH` = where Julia stores stuff

- packages
- binary dependencies
- ...

Why not `$HOME`?

- Quotas
- Can be read-only for compute jobs

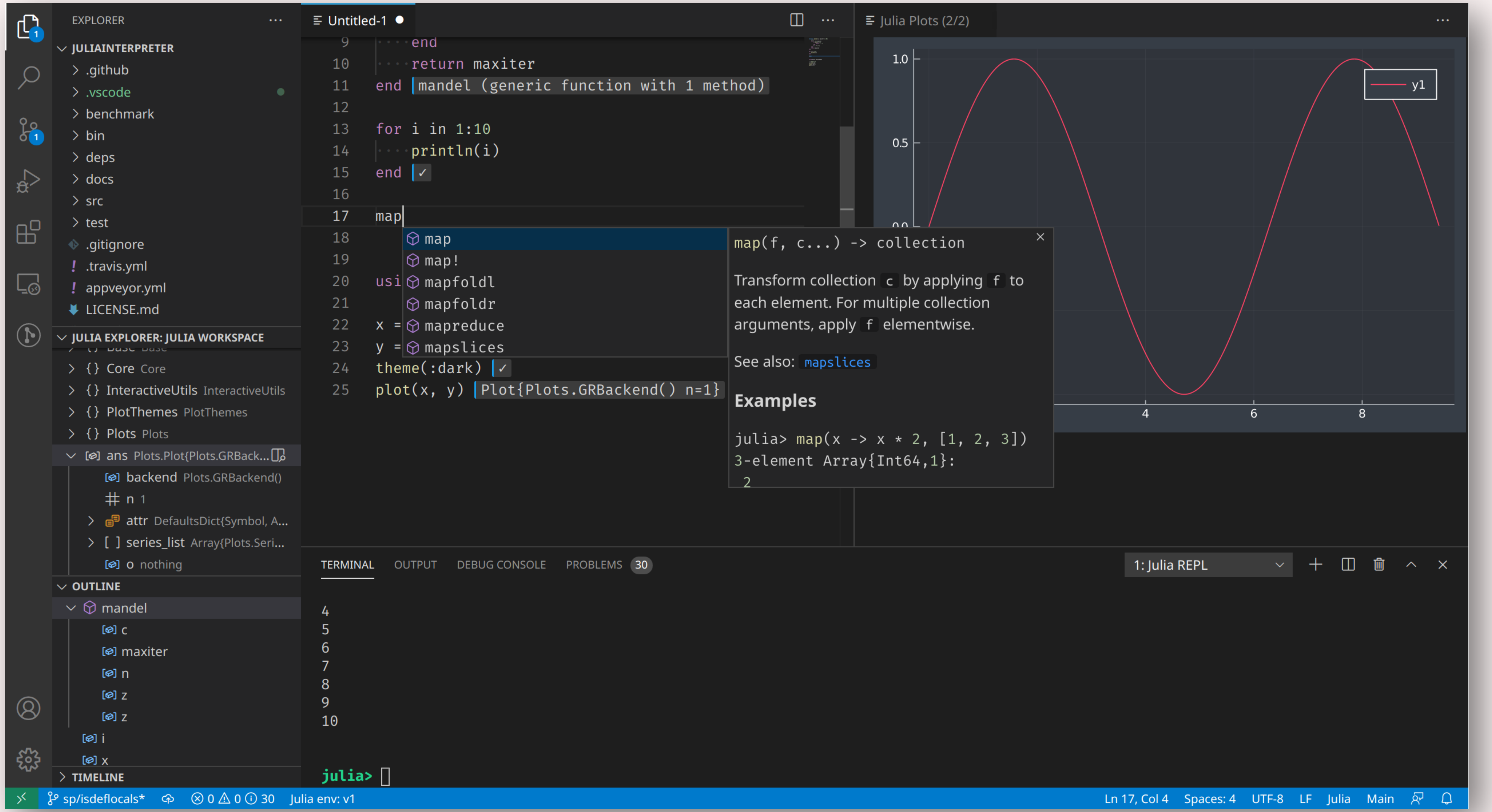
Julia VSCode extension requires a wrapper.

```
[...]

# Load modules
module use /projects/Julia/modulefiles
module load juliahpc
module load nvhpc

# Pass on all arguments
exec julia "${@}"
```

Julia VS Code integration via extension.



```
    end
    return maxiter
end  mandel (generic function with 1 method)

for i in 1:10
    println(i)
end  ✓

map

usi
x =
y =
theme(:dark)  ✓
plot(x, y)  Plot{Plots.GRBackend() n=1}
```

```
map
map!
mapfoldl
mapfoldr
mapreduce
mapslices
```

`map(f, c...) -> collection`

Transform collection `c` by applying `f` to each element. For multiple collection arguments, apply `f` elementwise.

See also: `mapslices`

Examples

```
julia> map(x -> x * 2, [1, 2, 3])
3-element Array{Int64,1}:
 2
```

```
4
5
6
7
8
9
10


julia> 
```