Julia Course - Lecture 4

Mattias Fält

From scripts and beyond

- Style guide
- Package development
- Testing
- Revise.jl
- Debugger
- Homework

Style Guide

- packages: `MyPackage.jl`
- types: `TrackingFloat`
    - Example: `StaticArray` in `StaticArrays.jl`
- function: `foo, bar, max, getindex`
- long function names: `run_twice, remotecall_fetch`
- Argument order:
  `(function, mutated, Type, other_value, args...; kwargs...)`
- mutating functions: `fill!(A, 0)`
- Examples
    - `sum(abs, A, dims=1)`
    - `rand!([rng], A, Float32)`
- Avoid: `foo(x::T) where {T<:Real}`
- Use: `foo(x::Real)`

`docs.julialang.org/en/v1/manual/style-guide/index.html`

Package Structure

To generate a new package with the nessesary structure:

```
]generate MyTestPkg
```

(used to be `Pkg.generate()`)

```
MyTestPkg/
    src/
        MyTestPkg.jl
    Project.toml
```

- MyTestPkg.jl - The code here is is run when `using/import`
- Project.toml - Contains metedata about package

Example: QPDAS.jl

https://github.com/mfalt/QPDAS.jl

Package with testing

```
MyTestPkg/
    src/
        MyTestPkg.jl
    test/
        runtests.jl
    Project.toml
```

- runtests.jl - Run when `]test MyTestPkg`

Other files you should have

```
MyTestPkg/
    src/
        MyTestPkg.jl
    test/
        runtests.jl
    Project.toml
    LICENSE.md
    README.md
    .gitignore
```

Continuous Integration (CI) with Travis

```
MyTestPkg/
    src/
        MyTestPkg.jl
        test/
                runtests.jl
    Project.toml
    LICENSE.md
    README.md
    .gitignore
    .travis.yml
```

- .travis.yml - Tells the travis server which language and versions to test

Package location

- Julia packages are almost always hosted on github.com
- You need to know how to clone/commit/push to a git repo if you want to contribute to the julia ecosystem

Setup:

- Create a github account
- Create a githug repo, with name ending in ".jl"
- Either
  - "git glone" the repo and add your files OR
  - In your existing folder "git init", "git remote add origin some.url/repo/here.jl"
- Login and connect `travis-ci.org` using your github account
- Login and connect `codecov.io` using your github account
- On travis: press settings on your repo and add the environment variable CODECOV_TOKEN="abcdef_ghijk...."
  from codecov.io
- Test your package locally: "`]activeate /loction/MyTestPkg`", "`]test MyTestPkg`", (make sure "Test" is in your list of dependencies in "Project.toml")
- Push a commit to github containing an appropriate .travis.yml file and runtests.jl and see the magic happen.

Testing

https://docs.julialang.org/en/v1/stdlib/Test/index.html

```
julia> using Test
julia> @test [1, 2] + [2, 1] == [3, 3]
Test Passed
```

```
julia> @test 0.2 + 0.1 == 0.3
Test Failed at REPL[1]:1
  Expression: 0.2 + 0.1 == 0.3
   Evaluated: 0.30000000000000004 == 0.3
ERROR: There was an error during testing
```

```
julia> @test 0.2 + 0.1 ≈ 0.3
Test Passed
```

Testing

https://docs.julialang.org/en/v1/stdlib/Test/index.html

```
julia> using Test
julia> @test [1, 2] + [2, 1] == [3, 3]
Test Passed
```

```
julia> @test 0.2 + 0.1 == 0.3
Test Failed at REPL[1]:1
  Expression: 0.2 + 0.1 == 0.3
   Evaluated: 0.30000000000000004 == 0.3
ERROR: There was an error during testing
```

```
julia> @test 0.2 + 0.1 ≈ 0.3
Test Passed
```

Testing

https://docs.julialang.org/en/v1/stdlib/Test/index.html

```
julia> using Test
julia> @test [1, 2] + [2, 1] == [3, 3]
Test Passed
```

```
julia> @test 0.2 + 0.1 == 0.3
Test Failed at REPL[1]:1
  Expression: 0.2 + 0.1 == 0.3
   Evaluated: 0.30000000000000004 == 0.3
ERROR: There was an error during testing
```

```
julia> @test 0.2 + 0.1 ≈ 0.3
Test Passed
```

```julia
f(x) = sum(x)
using Random
Random.seed!(0)
@testset "Testing f" begin
    @test f(1:10) == 55

    @testset "Test f with randoms" for i = 1:1000
        @test 4950 < f(rand(10000)) < 5050
    end
end
```

```
Test Summary: | Pass  Total
Testing f     | 1001   1001
```

Revise.jl

Tracks changes in files after compilation and will recompile if you change them! If including file from REPL, use includer("myfile")

Atom takes care of reevaluating functions in modules. If using another editor, Revise is invaluable!

Debugging

- Announcement: `https://discourse.julialang.org/t/ann-juno-0-8/22157`
- In Juno: `http://docs.junolab.org/latest/man/debugging/`
- From REPL: `https://juliadebug.github.io/JuliaInterpreter.jl/stable/`

Demo!

Homework

Create a package containing your own code. If you have nothing useful, use your TrackingFloat implementation as base.

Requirements:

- Export some useful functions or types
- Have meaningful tests and test coverage
- Host it on github/gitlab with continuous integration
- Follow the julia style guide
- Have a meaningful README
- Should be possible to
  `]add https://github.com/user/PackageName.jl.git`

Hint: Use some existing packages for "inspiration"