JuliaEO Workshop 2024:
Introduction to Julia:
Creating Packages for Earth Observational Work

Presented By: Nathanael Wong
material at https://github.com/natgeo-wong/JuliaEO2024_Nat

Notes
• This presentation (especially the parts using Terminal) are geared towards Linux and macOS users
  – Some familiarity with Terminal is required (don’t worry, I’ll go through step by step)
  – For Windows users, I recommend using the Windows Terminal
• It is also good to learn how to fiddle around and thus understand the concepts of package environment especially when developing packages

Aim
• The Julia community and ecosystem is relatively small
  – We don’t have the wide, extensive community that Python has
  – A lot of packages are therefore developed by individuals like myself
Aim

• The Julia community and ecosystem is relatively small
  – We don’t have the wide, extensive community that Python has
  – A lot of packages are therefore developed by individuals like myself

• Therefore, much of our development relies on the individual
  – This is especially true for the Earth Sciences / Geosciences and Earth Observations
  – If you need a specialized package, why not develop one yourself?

Outline

• Understand and manage the package ecosystem in Julia
• Creating Packages in Julia: What do you need to know?
• Creating Packages in Julia: Using GeoRegions.jl as an Example
• Applications of GeoRegions.jl
• (If we have time) Creating Packages in Julia: Back to the Drawing Board!

Outline

• Understand and manage the package ecosystem in Julia
  – Both for package development
  – And for project management, using DrWatson.jl

• Creating Packages in Julia: What do you need to know?
  – Developing packages 101: PkgTemplates.jl, CI, and some Testing
  – How do you organize a package?
  – Best-practice/performance tips

• Creating Packages in Julia: Using GeoRegions.jl as an Example
  – The many iterations of GeoRegions.jl
Outline

• Applications of GeoRegions.jl
  – The basic functionality of GeoRegions.jl: Define, select and extract data
  – Using GeoRegions.jl in other Packages: What do you need to know?

What is an Environment?

• A Julia environment defines and controls
  – Packages used in the project
  – Exact specs of package (#main, version, etc.)
• Defined using both Project.toml and Manifest.toml
• Every project you create should have a different environment.
  – Why?

THE JULIA PACKAGE ECOSYSTEM

how do I shot web? (i.e., how to install and uninstall stuff)
What is an Environment?

- **Project.toml:**
  - contains package list
  - is always necessary
  - contains [compat] bounds of the packages
  - More information can be found in Pkg.jl

- **Manifest.toml:**
  - contains package and dependency information
  - is not necessary upon startup, will be created upon project/environment initialization
  - is necessary an exact environment duplicate is required (e.g., for reproducibility purposes)

- You must always have a Project.toml
  - A Manifest.toml can be created on-the-spot through precompilation of the environment
    - GitHub has no Manifest.toml
Activity: Creating an Environment

• Now you try!

1. Create a folder
2. Run Julia in this folder (command: \julia)
Activity: Creating an Environment

• Now you try!
  1. Create a folder
  2. Run Julia in this folder (command: julia)
  3. Enter the package console (command: ])
  4. Activate the current environment (command: activate .)

Your previously empty folder should now have both Project.toml and Manifest.toml
Activity: Precompiling an Environment

• Now you try!

1. Go to https://github.com/natgeo-wong/2023GL104350 (this is my recent paper)
2. Clone the repository
3. Open Julia and activate the environment
4. Precompile the environment
5. Update the environment
Activity: Precompiling an Environment

• Now you try!

1. Go to https://github.com/natgeo-wong/2023GL104350 (this is my recent paper)
2. Clone the repository
3. Open Julia and activate the environment
4. Precompile the environment
5. Update the environment

6. Compare the difference in Manifest using git

Managing Environments in Package Creation

• PkgTemplates.jl (https://github.com/JuliaCI/PkgTemplates.jl)
  – Can be used to easily create new Julia packages (we’ll get to this later)
  – Will setup a Package Environment by creating both Project.toml and Manifest.toml
  – By default, sets Git to track only the Project.toml, not the Manifest.toml

• You generally don’t want to track the Manifest.toml of a Package
  – Can anyone tell me why?
Managing Environments in Package Creation

- **PkgTemplates.jl** ([https://github.com/JuliaCI/PkgTemplates.jl](https://github.com/JuliaCI/PkgTemplates.jl))
  - Can be used to easily create new Julia packages (we'll get to this later)
  - Will setup a Package Environment by creating both `Project.toml` and `Manifest.toml`
  - By default, setups Git to track only the `Project.toml`, not the `Manifest.toml`

- You generally don't want to track the `Manifest.toml` of a Package
  - Can anyone tell me why?
  - A:
    - [compat] requirements are already set in `Project.toml`
    - You want to allow for flexibility in dependencies, as these dependencies are likely also used by other packages in the environment
    - `Manifest.toml` will change very rapidly and will vary from device to device, not logical to track changes

Managing Environments for Personal Projects

- **DrWatson.jl** ([https://github.com/JuliaDynamics/DrWatson.jl](https://github.com/JuliaDynamics/DrWatson.jl))
  - More for creating your own self-contained projects (more on this later!)
  - Will setup a Project Environment by creating both `Project.toml` and `Manifest.toml`
  - By default, tracks both the `Project.toml` and `Manifest.toml` to ensure project reproducibility

- The scope of this lecture mostly focuses around Package creation, not project creation, but I thought it would be good to distinguish the two
- Can anyone give me an example of when it is good to track/commit the `Manifest.toml`?

Managing Environments for Personal Projects

- Paper reproducibility!
- Example given just now (this is an actual repository for my paper)
- This project folder was created using DrWatson.jl
Activity: Why different Environments?

1. Use DrWatson.jl to create projects called “TestProjectNew” and “TestProjectOld”
   Command: initialize_project("TestProjectNew")
   Command: initialize_project("TestProjectOld")

2. In the “TestProjectNew” environment, install NASAPrecipitation v0.3
   Command: add NASAPrecipitation@0.2

3. In the “TestProjectOld” environment, install NASAPrecipitation v0.1
   Command: add NASAPrecipitation@0.1
   • Using two different windows (loading julia in two different environments), compare and contrast NASAPrecipitation.jl

Activity: Why different Environments?

• Using two different windows (loading julia in two different environments), compare and contrast NASAPrecipitation.jl
  – How do you call the different datasets?
  – What is the difference in keywords?

• Different environments means you can load two different versions of the same package (in two separate windows)

• Note: Packages you have already loaded will remain loaded as is, (i.e., if you switch environments, the package version you have already loaded remains)

BREAK! (5 Mins)

Any questions?
CREATING PACKAGES IN JULIA: AN INTRODUCTION

An introduction to PkgTemplates.jl and other things you need to know

Package Creation with PkgTemplates.jl

• I usually use PkgTemplates.jl to create and develop new Julia packages

  PkgTemplates.jl will automatically create new packages inside ~/julia/dev
  – This is the dev folder, where all packages you develop are stored
  – Note: You can also develop preexisting packages, just do: Julia PkgHome

Activity: Your First Package!

• Your turn! Let’s try developing your first package!

  1. using PkgTemplates
  2. tpl = Template()
  3. tpl("MyFirstPackage")

Activity: Your First Package!

• Your turn! Let’s try developing your first package!

  1. using PkgTemplates
  2. tpl = Template()
  3. tpl("MyFirstPackage")
Activity: Your First Package!

• Your turn! Let’s try developing your first package!

1. using PkgTemplates
2. tpl = Template()
3. tpl("MyFirstPackage")

What does a fully-formed Package look like?

Designing a Package

• What are you trying to accomplish when you are designing a package?

• Let’s go around the room. What would you create a package for in Earth Observation?
  – Streamline workflows (e.g. downloading datasets)
  – Documentation of personal projects
Designing a Package

• What are you trying to accomplish when you are designing a package?

• Let’s go around the room. What would you create a package for in Earth Observation?
  – Data retrieval (from online servers, data repositories, etc.)
  – Data analysis (timeseries analysis, temporal/spatial smoothing, daily/monthly means)
  – Plotting and visualization of data

What are these? They are **actions** that you would perform on a **dataset**

– A package must **first** focus on defining these datasets and their **components**

– How would you organize a package?

---

### Package Components

- **Package**
  -陆-海掩蔽
  -文件系统

- **Components**
  - Dataset
  - Variables
  - Geographic Region

- **Actions**
  - Download
  - Analysis
  - Calculation

---

### Designing a Package

See filesystem structure on the right here

- Red = package filesystem
- Purple = package components
- Green = actionables / analysis

You also have miscellaneous backend items

- Date2String functions
- Error checks
- Nan-means
- Real2Int functions
Designing a Package

• What does this mean?
  – You need to be able to know what data is available/provided
  – What is the package going to do for you?
  – You may not want to retrieve everything (e.g., is all the information relevant to you?)

• E.g., the Global Precipitation Mission provides a lot of extra data
  – e.g., IR Precipitation (Infrared Radar)
  – Do you need these data? Or just the total precipitation values?

Multiple Datasets in a Package

• Packages may handle multiple datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Timeframe</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPM_TABGEOG_86</td>
<td>2004-01-02T18:18:3450T</td>
</tr>
<tr>
<td>GPM_TABGEOG_87</td>
<td>2005-01-02T18:18:3450T</td>
</tr>
<tr>
<td>GPM_TABGEOG_88</td>
<td>2006-01-02T18:18:3450T</td>
</tr>
</tbody>
</table>

• Packages may handle multiple datasets
  – Each of these datasets may have different properties
  – Each of these datasets may have different performable actions

• How do we handle multiple datasets in a single package?
  – Types + multiple dispatch methods!
Multiple Datasets in a Package

- Packages may handle multiple datasets
  - Each of these datasets may have different properties
  - Each of these datasets may have different performable actions
- How do we handle multiple datasets in a single package?
- Types + multiple dispatch methods!

Understanding your Package

- You (and your collaborators) are the designer of the package

  - You know your package best. **Does everyone else?**
    - A good Julia package is not only comprehensive and well-organized, but it must also be easy for people to understand

- **Documentation**! (made using Documenter.jl)
  - PkgTemplates.jl will also set this up
Understanding your Package

- You (and your collaborators) are the designer of the package
- You know your package best. **Does everyone else?**
  - A good Julia package is not only comprehensive and well-organized, but it must also be easy for people to understand
- **Documentation!** (made using Documenter.jl)
  - PkgTemplates.jl will also set this up

TL,DR

- **PkgTemplates.jl** is a good tool for package developers in Julia
  - Designing a package requires familiarity with the relevant datasets and the variables
  - If you want to design a package, I recommend sketching out
    - What your package is supposed to do
    - What datasets and variables are in your package, how are they accessed?
  - Multiple datasets can be handled using Julia types and **multiple dispatch**
  - Documentation is also necessary if you want other people to use your package!
    - Read through Documenter.jl for further details (I won’t be able to cover this in this lecture)

BREAK! (5 Mins)

Any questions?

The logic behind the development of GeoRegions.jl

**CREATING PACKAGES IN JULIA: GEOREGIONS.JL AS AN EXAMPLE**
What is GeoRegions.jl?

- Deals with gridded data (preferably rectilinear grids)
- Specify a Geographic Area:
  - ID
  - Name
  - Parent Region (default is GLB)
  - [N,S,E,W] or longitude/latitude vectors specifying a shape
- E.g.: specify region to download GPM IMERG data from OPeNDAP

GeoRegions.jl as an Example of a Julia Package

- Current version: v5.2.6
- In 2023, it was at v3
- Why did I bump it up 2 versions?
  - Julia follows SemVer
  - If you publish your package as v1, any breaking changes require you to bump your version
GeoRegions.jl as an Example of a Julia Package

- Current version: v5.2.6
- In 2023, it was at v3
- Why did I bump it up 2 versions?
  - Julia follows SemVer
  - If you publish your package as v1, any breaking changes require you to bump your version
  - It is better to wait and ensure that your package is stable before bumping to v1
  (Note that most of the packages introduced during this workshop are v0.X)

GeoRegions.jl as an Example of a Julia Package

• Go to https://github.com/JuliaClimate/GeoRegions.jl and download/clone the repository
• We will explore the package together

GeoRegions.jl as an Example of a Julia Package

Package

Components

• N/A (filesystem handling irrelevant)

Actions

• What are the supertypes available in GeoRegions.jl?

• What does the package do?

GeoRegions.jl as an Example of a Julia Package

Package

Components

• Georegion
• RegionGrid
• AbstractLandSea

Actions

• Data Extraction
• Is XX in YY?
• ETOPO
GeoRegions.jl as an Example of a Julia Package

• The master file of PackageName.jl is always found in src/PackageName.jl
• This is the starting point of every package

• In src/GeoRegions.jl, note the order of what I am doing:

1. Load the package dependencies (information contained in Project.toml)
   - using is okay for most cases
   - import is used when you want to reexport a function

2. Export the package functions and Types defined in GeoRegions.jl
   - Order doesn’t matter
   - I usually export the Types on top, then the functions below
   - Exporting Types is important if you are creating parent packages
GeoRegions.jl as an Example of a Julia Package

1. Load the package dependencies (information contained in Project.toml)
2. Export the package functions and Types defined in GeoRegions.jl
3. Define the most important Types in your Package (preferably at least the abstract types)
   - A Type must be defined first before it can be used in a function
4. Define your functions (and include other files with functions)
   - Order of function definition doesn’t matter
   - Reminder: Relevant Types must be defined first

I learned all these practices by trial and error, committed to v1 too early
- This is why the version is large (at v5 currently)
- v1 → v2: shifted to using Types instead of Dictionaries
- v2 → v3: breaking changes for dataset downloading (would require people to redownload datasets all over again)
- v3 → v4: changed field names for RegionGrid Types (would break parent packages)
- v4 → v5: changed field names for GeoRegion Types (would break parent packages)
GeoRegions.jl as an Example of a Julia Package

- I learned all these practices by trial and error, committed to v1 too early
  - This is why the version is large (at v5 currently)

- v1 → v2: shifted to using Types instead of calling Strings
  - GeoRegion information was loaded and stored
  - No need to keep calling functions again and again

- v3 → v4: changed field names for RegionGrid Types (would break parent packages)

- v4 → v5: changed field names for GeoRegion Types (would break parent packages)

- Why did I do this?
GeoRegions.jl as an Example of a Julia Package

- I learned all these practices by trial and error, committed to v1 too early
  - This is why the version is large (at v5 currently)

- v3 → v4: changed field names for `RegionGrid` Types (would break parent packages)
- v4 → v5: changed field names for `GeoRegion` Types (would break parent packages)

- Why did I do this?
  - A: e.g. I felt that using “ID” was more intuitive than “regID”
  - Simplar is always better

---

GeoRegions.jl as an Example of a Julia Package

- It is important to provide documentation for your functions
- Good documentation also provides examples and use-cases, not only API functionality
GeoRegions.jl as an Example of a Julia Package

- You can export functions, and more importantly, types, for use in other packages
  - e.g., GeoRegions.jl exports the LandSeaFlat type
  - NASAPrecipitation.jl creates a subtype of this GeoRegions type

GeoRegions.jl as an Example of a Julia Package

- You can export functions, and more importantly, types, for use in other packages
  - e.g., GeoRegions.jl exports the LandSeaFlat type
  - NASAPrecipitation.jl creates a subtype of this GeoRegions type
GeoRegions.jl as an Example of a Julia Package

- You can export functions, and more importantly, types, for use in other packages
- e.g., GeoRegions.jl exports the LandSeaFlat type
  - NASAPrecipitation.jl creates a subtype of this GeoRegions type
- This is how you

BREAK! (5 Mins)

Any questions?

Breakdown of Today’s Tutorial

- Using GeoRegions.jl
  - Defining your own GeoRegion
  - Properties of a GeoRegion
- How do you use GeoRegions?
  - Data Extraction for a particular region (defined by a GeoRegion)
  - Is a point/region within a GeoRegion of interest?
  - Land-Sea Mask Datasets (retrieving and manipulating ETOPO data)
- Using GeoRegions.jl in other packages

How do I use GeoRegions.jl in Earth Observation work?

USING GEOREGIONS.JL
Breakdown of Today's Tutorial

• For this part, we will be using Pluto.jl notebooks! (https://plutojl.org)
  – Good for step-by-step story format presentation
  – Interactive and in-real-time

• I like using Pluto.jl notebooks to do data visualization and presentation

Breakdown of Today's Tutorial

• Today's notebooks are not a exhaustive tutorial of the functionalities of GeoRegions.jl
  – For a more comprehensive breakdown, it is always best to refer to the documentation

• Today's introduction to GeoRegions.jl is more to reinforce concepts taught earlier, i.e.,
  – how do you want to design a package and what is it supposed to do?
  – using Types and multiple dispatch
  – exporting package functionality for future use (e.g., how GeoRegions.jl Types can be exported for use in parent packages such as NASAPrecipitation.jl)

Opening the notebooks

• First, you need to setup the environment?
• Recall, what do you need to do?
• *Precompile the environment!*
Opening the notebooks

1. Go into the notebooks folder
2. Run Julia
3. Loading the Pluto.jl package (using Pluto)
4. Open a Pluto notebook session (Pluto.run())

Notebooks Time!

We’ll be using notebooks for this part of the lecture

• From there, you can open Pluto notebooks
  – We will go in order from 01 to 04
CREATING PACKAGES IN JULIA: A SUMMARY

Why do you want to create a Package?

- A key part of creating a package is understanding Julia environments
  - Understand the basics of environment creation, activation, compilation and updating
  - Understand the purpose of a Project.toml and a Manifest.toml
- Each package has its own environment
- Every project you have should also have its own environment
  - Not recommended to use the master environment unless it’s for basic testing stuff
  - Different projects can have different package versions for different use-cases

Why do you want to create a package?

- What are you trying to accomplish when you are designing a package?
  - Data retrieval (from online servers, data repositories, etc.)
  - Data analysis (timeseries analysis, temporal/spatial smoothing, daily/monthly means)
  - Plotting and visualization of data
- Design your package such that it is easy for people to use and understand
  - An understanding of types and multiple dispatch helps a lot in organizing your package
  - Clear documentation also helps a lot
The End

Thanks for listening!