Project Proposal: Improvements to the REPL workflow (TermLayouts.jl)

Soumitra Shewale, The Julia Language

April 16, 2022

1 An Introduction

I am Soumitra Shewale, a first-year undergrad at BITS Pilani, Hyderabad, India, and I am pursuing a dual degree. My first degree is an M.Sc. in Economics. My second degree will be set in stone by next year, but I plan to pursue a Computer Science and Engineering degree.

I have been making open source contributions for a reasonably long time, and I have been part of Google Code-in in 2018-19 and 2019-20. In the 2018-19 session, I worked with Catrobat on their PocketCode app, and I ended up as a Finalist that year with Catrobat. In the 2019-20 session, I worked with The Julia Language, and I finished as a Runner-up. That has been the most engaging open source work I have ever done.

My mentor during the 2019-20 session of Google Code-in was Mr. Logan Kilpatrick, and I asked him about Google Summer of Code and Julia's participation in it, and ever since then, I have had my eye on this opportunity. Finally, I am eligible for the first time, and I am here.

During that session, I have worked multiple times with both the REPL and the Images.jl package, and now I am hoping to work on a brand new feature that thousands of data scientists worldwide working with images in Julia, and I would also love to use.

Since the 2019-20 session of Google Code-in, I have mostly been learning different technologies in the space, and I have been learning web development, and a lot of those projects I did then could be augmented with image processing or machine learning.

In that same session, I also made one of my closest friends today, Madhav Shekhar Sharma. At the time, he was working extensively on machine learning, and he praised how elegant and straightforward a lot of Flux.jl's implementation is. This simplicity inspired me to try learning machine learning in Julia, and lo and behold, I was doing data science work in Julia. Though I am still learning, a lot of what I have practiced and learned involves working with images interactively.

Many data scientists use Jupyter Notebooks and similar notebooks in their work due to their interactive nature. One pitfall of this is that such notebooks have a fundamental lack of a lot of the features a script in a Visual Studio Code editor has. The julia-vscode package provides most of these features. Bringing those features to the REPL is what makes this project special. It can transform a lot of interactive data science work due to how easy it makes debugging, previewing, and testing code. This potential in this project, in particular, is what makes me so excited about having the opportunity to work on this project.

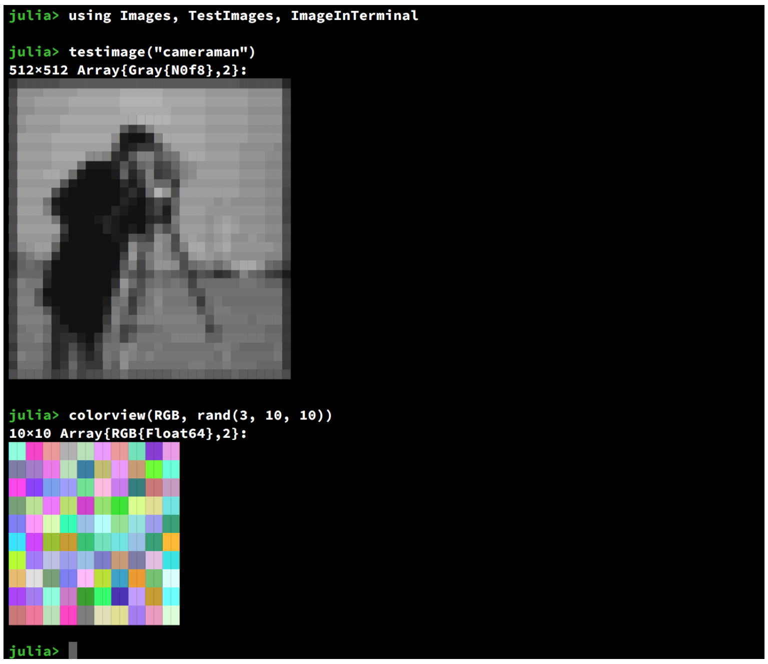
2 Problem Description

Julia is a high-level, high-performance, dynamic programming language. One of Julia's most extensive features is the extensibility of its REPL. Julia also frequently sees much use in Data Science and machine learning. These use cases make Images.jl, an umbrella package in the Julia standard library for dealing with images, crucial to the community.

For quick prototyping, especially data pre-processing, the REPL often is a handy tool. However, the REPL interface as it exists today is too bland and makes working with Images in the REPL significantly more inconvenient.

For example, some of the problems with the REPL when working with Images are:

- An external package is needed to display the images in the REPL appropriately
- The external package needs additional maintenance
- The external package achieves the goal of displaying the image in the terminal, but the same function needs to be called over and over again, and since Images can be huge, these images can clog up the REPL, making it harder to work with them
- Images take up significantly more memory than most variables or data structures, so when dealing with large datasets or large images, keeping track of the variables defined is paramount



```
julia> using Images, TestImages, ImageInTerminal

julia> testimage("cameraman")
512×512 Array{Gray{N0f8},2}:

julia> colorview(RGB, rand(3, 10, 10))
10×10 Array{RGB{Float64},2}:

julia>
```

Figure 1: Example usage of ImageInTerminal.jl

An example of ImageInTerminal.jl's working can be seen in figure 1.

Furthermore, the REPL also has more general issues that hinder fast and efficient Data Science work:

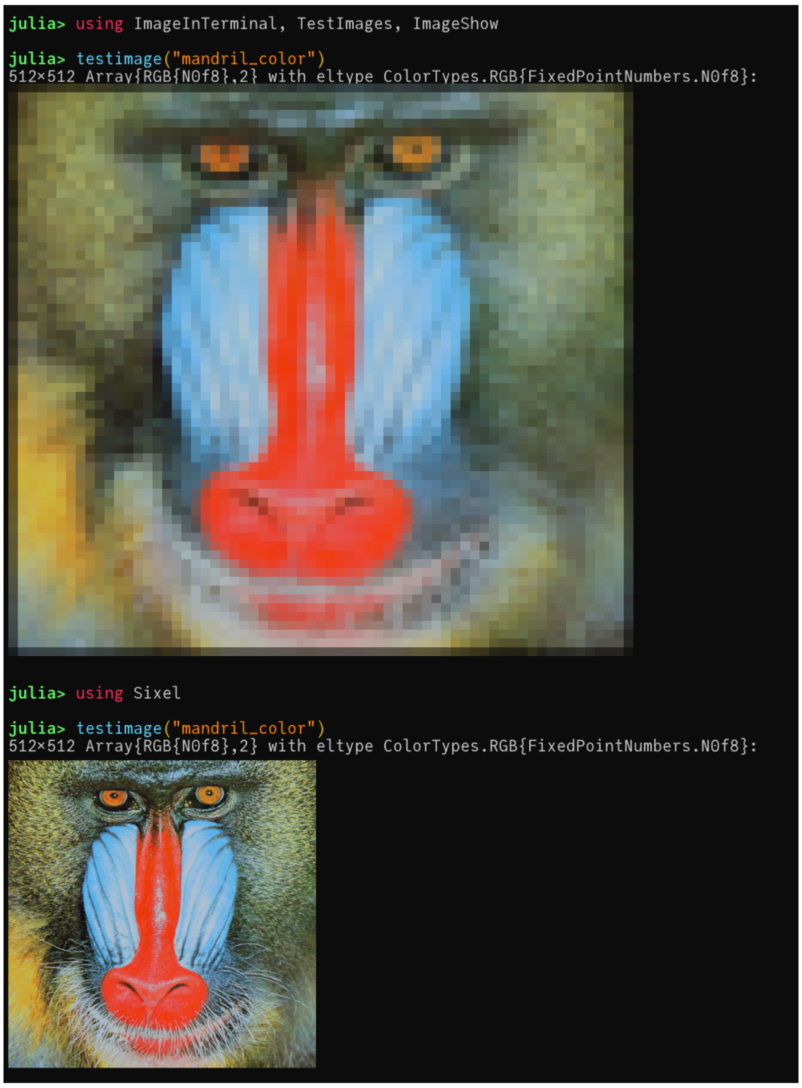
- The REPL lacks any debugging tools, while the julia-vscode package has achieved this
- The REPL lacks a dedicated logger, which, again, julia-vscode has achieved

Even the ImageInTerminal package has its limitations:

- Older versions (< 1.6) of Julia can only render large images using the default encoder.
- Never versions, however, can use the Sixel encoder to encode large images allowing for a much better viewing experience.

An example of this limitation can be seen in figure 2

This project will aim to address all the above concerns.



```
julia> using ImageInTerminal, TestImages, ImageShow

julia> testimage("mandril_color")
512×512 Array{RGB{N0f8},2} with eltype ColorTypes.RGB{FixedPointNumbers.N0f8}:
```

```
julia> using Sixel

julia> testimage("mandril_color")
512×512 Array{RGB{N0f8},2} with eltype ColorTypes.RGB{FixedPointNumbers.N0f8}:
```

Figure 2: The image at the top uses the default encoder, while the image at the bottom uses the Sixel encoder. Notice, even a 512x512 image can be rendered significantly better by Sixel.

3 Implementation plan

Figure 3 is an example of what the final expected REPL interface could look like. We will reference the figure throughout the discussion in this section.

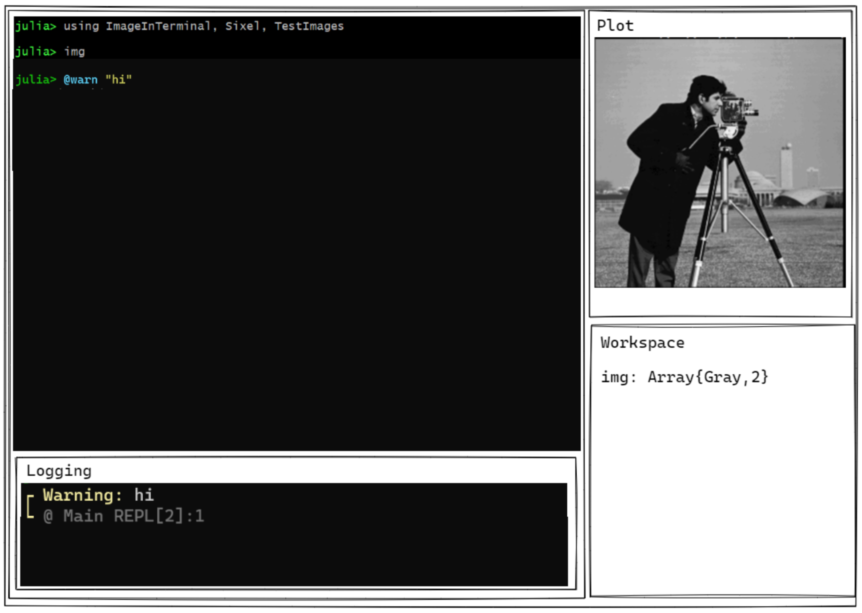
We will now discuss implementing some of the features planned for this project. This part also serves as a detailed list of deliverables.

Tiled Layout

A tiled layout as shown in figure 3 will be the foundation of this project. The julia-vscode project also uses Visual Studio Code's tiled UI to fit many such features onto the UI in a clean manner. To prepare this layout, we will use Term.jl, a Julia package that makes drawing such panels and other rich content in the console significantly more accessible. We also plan to handle keystrokes, which will be the primary way to control the REPL interface. These keybinds will allow us to minimize and resize individual panels in the interface and control the proposed debugger better.

Plot Area

The top-right panel in figure 3 will act as the stage for images to be displayed. This panel is where images will be displayed when printed or displayed by the REPL. A lot of this functionality will use the existing ImageInTerminal.jl codebase.



```
julia> using ImageInTerminal, Sixel, TestImages

julia> img

julia> @warn "hi"
```

```
img: Array{Gray,2}
```

```
┌ Warning: hi
└ @ Main REPL[2]:1
```

Figure 3: Proposed implementation

REPL Panel

The top-left panel in figure 3 will retain the existing REPL. Similar to how the julia-vscode project wraps its REPL, we will also wrap this REPL. This customizability allows us to separate user input, code output, log messages, and errors. This project will entirely control the stdin, stdout, and stderr. Julia allows us to start a REPL on custom streams for stdin, stdout, and stderr. Keystrokes that do not match the preset keyboard shortcuts will be passed down to this REPL.

Workspace

The bottom-right panel in figure 3 will act as a variable inspector. This panel is only there to keep track of the variables currently defined. We could extend this by adding filters for variables that hold images or for variables in different scopes. This panel will also act as the primary way to inspect variables when the debugger stops at a breakpoint.

Debugger

The bottom-left panel in figure 3 will serve as the place for the debugger. Most of the debugger functionality will mirror the existing debugger in julia-vscode:

- **Code Navigation**: Functionality that allows continuing after breakpoints, stepping into and out of functions, and stepping over to the next line of code.
- **Workspace View**: Same as the Workspace view we discussed in a previous section.
- **Breakpoints**: Points at which execution has been paused to allow for the use of other features listed here.
- **Debug console**: A console that allows us to evaluate code in the scope that the debugger is currently examining.

Configuration

Users write a configuration file to customize their layout. The file will include details like sizes of panels, positions of panels, which panels have which functionality, which we will cal extensions. This feature can be worked on later, but for the scope of GSoC, a default configuration file with the above discussed extensions and panels will be enough. TermLayouts.jl will only load the extensions that are included in the configuration file.

Testing
All the testing will be done using unit tests. This is made possible by the IO interface. This will allow us to both interface TermLayouts.jl programmatically, and manually.

Documentation
Of course, such a project will need extensive documentation to be maintained effectively in the future. This code will follow the standard Julia docstring format so that it can later be extended by one of many packages to create a documentation website for the same if required.

4 Proposed Timeline

4.1 Some background

My university campus recently had a small outbreak of COVID-19, which pushed back some of the exams for some of the students and drastically changed the way classes, exams, and other continuous evaluations were being conducted. Of course, this specific incident will not affect the timeline of this project, but if such an incident were to happen in the future, it could move some of these dates around. However, this will not alter the amount of time I have for this project or the amount of time I can dedicate to it. The only effect such an event could have changes when I work on the project, not how much. I doubt this would be an issue, however.

I am also a dual degree student, and the terms of this program at my university require me to maintain a CGPA above a fairly high threshold in order for me to attain the second degree of my preference: Computer Science. For this reason, I will have to focus more than usual on my final exams.

My final exams for my second semester begin on 18th June and end on 30th June. However, I will count the time between 13th June (the starting date of the GSoC program) and 18th June as busy since I will be preparing for these exams during this time. After 30th June, however, I can stay fully committed to this project since I will be on vacation.

Though there is no official information on this, one can safely assume the classwork for my second year will begin around 1st September. During this time, I can stay partially committed to this project and dedicate about 2-4 hours of my day. This partial commitment will continue until 12th September, when I should be done with my project.

Since we now have some background on when I can work on this project, we can now move to discuss the proposed timeline in detail.

4.2 Timeline

20th May - 12th June (Community Bonding Period)
These days, I will attempt making smaller proofs-of-concept that I can later integrate into the main project. These will only be small scripts that I can use later during the project to clarify how I will implement some of the more advanced features that I did not get to explore when working on my project proposal.
During this time, I will communicate extensively with my mentor and people on the Julia Slack about the finer details and design choices we will make in this project.

13th June - 30th June
These days, I will not work on the project since I have my final exams. I will, however, be available on Slack, Email, and GitHub for discussions.

1st July - 7th July
This week, I plan to finish the basic layout of the new interface and get the REPL inside the REPL panel functioning. I will also start working on getting images to show up in the Plot panel.

8th July - 14th July
This week, I plan to finish working on the Plot panel and begin working on the resizing and hiding panels part. These features require me to write code that handles keystrokes and refreshes the layout accordingly. This work will also serve as a great chance to make the layout responsive to terminal size.

15th July - 21st July

This week, I will work on the configuration file feature. This will allow for better testing later, since we can use the configuration file to load only the panels we want to test, making testing much faster, and targeted.

22nd July - 28th July

This week, I will finish documenting my code and submit my progress so far for the Phase 1 evaluation.

29th July - 4th August

I will work on the suggested changes this week after the Phase 1 evaluation. I will begin working on the Workspace panel if I have spare time.

August 5 - August 11

This week, I will finish my work on the Workspace panel and begin working on the debugger. Since I expect the debugger to be incredibly complex, I will have to start working on it early.

August 12 - August 25

In these two weeks, I will finish implementing the debugger thoroughly. If I am done early, I will document all my older code and start working on bug fixes and optimization. The final result will be much more polished now, and the code will be readable and well kept.

August 26 - September 12

I will continue working on bug fixes, edge cases, and optimization in these two weeks. This period also includes buffer time so that if any of the above tasks take longer than expected, I can safely work on the unfinished parts before submitting my work for evaluation.