1b. Running Julia

Martin Alfaro
PhD in Economics

INTRODUCTION

In the following, we cover the basic steps for getting started with Julia. As we haven't introduced any tools available in Julia (e.g., functions), we'll keep the discussion to a bare minimum. Specifically, we'll limit ourselves to setting up Julia in VS Code and presenting methods to add comments and file paths.

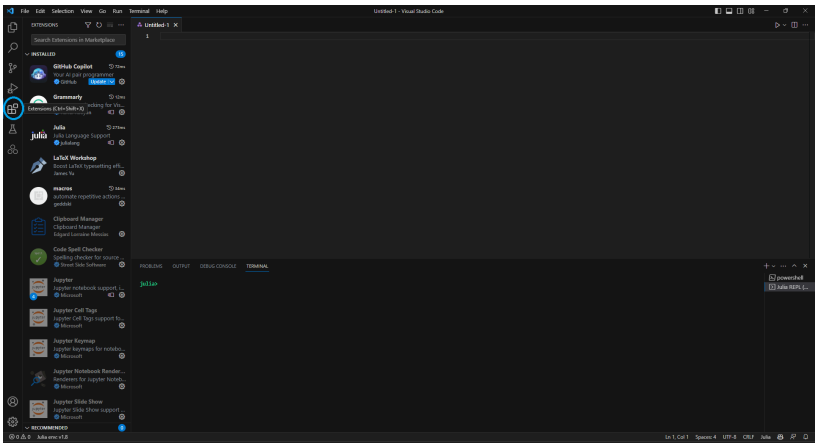
USING JULIA IN VS CODE

The REPL (Read-Eval-Print Loop) is an interactive programming environment. It lets users input commands and immediately obtain outputs through a command-line interface. When you run `julia.exe`, the REPL is automatically activated and displays the `julia>` prompt, where you can enter commands.

▼ Screenshot

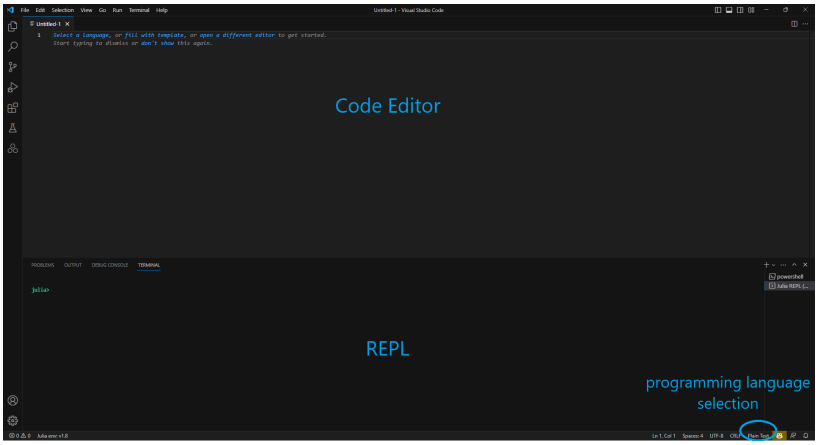
Throughout this website, we'll assume that you're working with a code editor, rather than interacting directly with the REPL. In particular, VS Code will be our code editor of choice, including its privacy-focused alternative VS Codium. To get started, you'll need to install the Julia extension of VS Code. This can be found by navigating to the Extensions tab, as indicated below by the blue circle.

▼ Screenshot



The layout of VS Code displays the REPL at the bottom of the screen, with code written in the area above it. To execute code, you'll also need to specify the programming language you're using. This can be achieved by clicking on the language option located at the bottom corner of the screen, or by using the keyboard shortcut Ctrl+k + M and typing "julia". All this is demonstrated in the screenshot below.

▼ Screenshot



ADDING COMMENTS IN A SCRIPT

Comments are text annotations ignored during execution, serving as a means to document your code. To add a *single-line comment*, simply precede the text with the # symbol. This symbol can be placed anywhere on a line, with any text that follows disregarded by Julia. Alternatively, you can add *multi-line comments* by delimiting the text with #= at the beginning and =# at the end.

```
# This is an example of a comment

x = 2   # `x=2` is run, but anything after `#` won't

#=  This is an example of a longer comment.
    It can be split into several lines, and
    can have any length. =#
```

PATHS OF FILES AND FOLDERS

File management systems vary across operating systems, determining that the syntax for file paths also differs. To accommodate this, Julia provides two approaches. The first one provides an operating system-specific syntax. Below, we illustrate its application for a file `C:\user\file.jl` on Windows and `/user/file.jl` on Linux/macOS. There's also a platform-agnostic alternative to make your code more portable, provided by the `joinpath` function. This is the preferred option, as it can be used with any operating system.

```
# On Windows (note the double \\)
"C:\\user\\file.jl"

# On Unix-based systems (e.g., macOS or Linux)
"/user/file.jl"

# on any operating system
joinpath("/", "user", "file.jl")
```

Two special paths have convenient shortcuts worth mentioning:

- `@__DIR__` identifies the directory where your script is saved.
  For instance, if your script is in `C:\user\julia`, then `joinpath(@__DIR__, "graphs")` refers to `C:/user/julia/graphs`.

- `homedir()` indicates the user's home directory.
  This refers to `C:\Users\username` on Windows (where "username" is your actual user), and is the equivalent of `~` on Linux. For instance, you could access your Google Drive's folder located on either `C:\Users\username\GoogleDrive` or `\home\username\GoogleDrive` by the command `joinpath(homedir(), "GoogleDrive")`.

EXECUTING CODE FROM A FILE

Julia also allows you to work **non-interactively**, by executing code from a script stored in a file. The following example illustrates its implementation, running a file located at `C:\user\julia\graphs.jl` on Windows and at `/users/julia/graphs.jl` on macOS/Linux systems.

```
include(joinpath("/", "user", "julia", "graphs.jl"))
```