Getting started

by Chris Griffiths, Eva Delmas and Andrew Beckerman, Dec. 2020.

This document covers the following:

- Installing Julia and Visual Studio Code on your computer
- Adding extensions - i.e. setting up Visual Studio Code to run Julia
- Julia's package manager (`Pkg`): installing and loading packages

First off, check out the **Introductory videos on the visual studio webpage**. These will help explain some of the terminology used below. They'll also show you how to change the colour theme and icon symbols (by far the most important part of any coding tutorial...)

Download and install

1. Julia

Navigate to **this page** and follow the platform-specific instructions to dowload and install Julia (we recommend installing the current stable release). During the installation process, you may be prompted to add Julia to the PATH, this box should be ticked.

2. Visual Studio Code

Navigate to **this page** to dowload and install Visual Studio Code (not Visual Studio).

Setting up VS Code

VS Code is a free source-code editor, allowing you to code in multiple coding languages all from a platform that is completely customisable to the user. This flexibility is great but it does mean that you need to spend time telling VS Code what it is you want to do and how. This is where extensions come in; extensions are higher level packages that permit the use of a given coding language like Julia, edit your themes and icons, and provide helpful applications like spell checker or Bracket Pair Colorizer.

To install Julia in VS Code do the following (you only need to do this once):

- open VS Code (you'll see the welcome page)

- navigate to the 'Marketplace' (5th symbol down in the activity bar - vertical panel on the lefthand side of the screen)
- search for Julia in the 'Search Extensions in Marketplace' search bar
- install **Julia**, this extension provides support for the Julia programming language
- while you're there, also search for and install **Julia Formatter**, this extension will help you write clean code that is easier to read

There are **many** extensions that can make your life easier, visit the marketplace for extensions that can help you with pdfs, markdown and csv files, etc. There are also several useful extensions for GitHub linkage (we'll mentioned a few later on in this tutorial).

Projects

A good practice to adopt when coding is to always work in a contained project environment as each unique project may require a different setup (e.g. packages, package versions, working directories, data locations, etc.).

To set up a project in VS Code, start by:

- Creating a folder at a location of your choosing (e.g. within your Documents folder). We will name our folder 'JuliaTuto' for the examples that follow but please feel free to replace this name as you see fit.

To open your new project in VS Code (e.g. here JuliaTuto):

- click on the 'Explorer' symbol (top symbol on the activity bar) and click Open Folder
- navigate to the JuliaTuto folder in your files and click open
- this folder then becomes the location of your working directory (same as when using an RProject in R)
- after clicking open, the contexts of your directory will appear as a vertical pane on the left hand side of the screen

The next step is to create a new file (a script) in your directory:

- do this by using cmd-N (ctrl-N on windows), File>New File or by left clicking>New File within the directory pane
- name your script as your see fit but please rememeber to include the .jl file extension (e.g. JuliaTuto.jl)
- the .jl file extension tells VS Code you want to use the Julia programming language
- to save your script at any time use cmd-S (ctrl-S on windows) or File>Save

Alternatively, you can also open a project in VS Code by right-clicking on your folder (in Finder, Windows file explorer or Linux nautilus) and selecting Open with -> Other -> VS Code. Or if you are a command-line addict, you can `cd` into the directory and type `code .` to open VS Code (this needs **to be activated for Mac**).

Now that you have an active project and a new script file you can open a new Julia REPL (stands for - read, execute, print and loop). The REPL is like the console in R and is where the magic happens. In Eva's words, it's VS Code's way of using Julia for a brain. To do this you type Start REPL in the command palette (accessed using F1, cmd-shift-P or View>Command Palette). The command palette will appear as a drop down menu with a search function at the top of the page.

Next, type `print("Hello world")` in your new script, and execute (run) using Ctrl-Enter. If you've done all of the above correctly, `Hello world` should print in the REPL.

```
print("Hello world!")
```

Julia package manager

For an introduction to the Julia package manager, see **the documentation**. The "Background and design" section provides useful details if you want to understand more about its originality compared to other languages and how it works.

Project and Manifest files

We mentioned above that it is good practice to work within an environment specific to each project. The Julia package manager (`Pkg`) allows you to do that easily: *Unlike traditional package managers, which install and manage a single global set of packages, Pkg is designed around environments: independent sets of packages that can be local to an individual project or shared and selected by name* (text taken directly from **the documentation**).

The project environment is stored in two files:

- `Project.toml` - stores a list of packages that have been installed and their versions (very useful when sharing code as any user can see exactly what version of a package has been used)
- `Manifest.toml` - does the same thing but for all the package dependencies (other packages that are essential, or form functional parts, of the package you want to use)

You don't have to create or modify these two files, they are automatically created once you have activated your project and installed your first package.

To activate a project:

- activate your project by typing `] activate .` in the REPL or by using `Pkg.activate(".")` in the script
- activation is done automatically in VS Code, however, it remains an important point as activation ensures that your project is 'active' and can use package operations

Note: the dot `.` stands for the current working directory. You could also use `pwd` and activate using `Pkg.activate(pwd())` or `Pkg.activate("path/to/folder/JuliaTuto")`.

Once your project is activated, there are two ways to use the package manager (`Pkg`):

- directly from the REPL:
  - navigate to the REPL
  - type `]`
  - you will see that instead of seeing `julia>` you now see `(your-project-name) pkg>`, indicating that all the packages that you now install (or update) will be installed (or updated) within this specific project
  - to exit the REPL package manager use the backspace
- using `Pkg`, this is useful when you want to use the package manager directly from your script:
  - type `import Pkg` and execute using Ctrl-Enter
  - you can then add, remove and update packages from your script using the build in Pkg functions e.g. `Pkg.add()`, `Pkg.remove()` or `Pkg.update()`

There are also two ways to double check that you are actually working within your project:

- check/click the 'Julia env:...' on the bottom of your screen (blue bar), it should match your project name
- enter the package manager by typing `]` in the Julia REPL, you should see `(your-project-name) pkg>` instead of `julia>`. Again, exit the package manager using backspace.

Working on someone else's project:

If you are working on someone else's project (or on your project but from a different computer), you will want to use the same packages and package versions. To do this, first activate the directory (see above) and then install all the packages and dependencies stored in the `Project.toml` (and `Manifest.toml`) by typing either `] instantiate` in the REPL or `Pkg.instantiate()` in the script. This is incredibly useful as it means you can ensure that you're working from an identical project environment with a single command.

Package manager

Now that we are all set up, we are going to install a package, check the project's status and remove a package. As this might be your first time installing a package (e.g., Plots), dont be concerned if it takes a couple of minutes to run.

- type `] add Plots` in the REPL (or `Pkg.add("Plots")` in your script and execute using Ctrl-Enter)
- you just installed the Plots package (equivalent to Base plots in R) to your project
- adding a package in this manner is the equivalent to using `install.packages()` function in R

- type `] st` in the REPL (this is the status command and prints the contents of your `Project.toml` file)
- you should see something like:

```
(JuliaTuto) pkg> st
    Status `~/projects/JuliaTuto/Project.toml`
    [91a5bcdd] Plots v1.6.12
```

- to use an installed package (e.g. Plots), type `using Plots` in the script and execute
- this imports all the functions of the Plots package
- `using` is identical to `library()` or `required()` in R
- you can update packages using either `] up` (or `Pkg.update()`)
- updating can be done globally for all packages or for a specific package e.g. `] up Plots`
- as you have just installed the latest version of Plots, `] up Plots` will likely return `[no changes]`
- to remove a package from your project, type `] rm Plots` in the REPL (or `Pkg.rm("Plots")` in your script and execute)
- now if you look at your directory status using `st` the list should be empty

Remember, when using your package manager from the REPL via `]`, you exit it using the backspace.

We have tested these tutorials with a specific list of packages. Consequently, we recommend you download the `Project.toml` and `Manifest.toml` files from **this link** (you'll find code to automatically download these files within Julia below). Once downloaded these files will replace your current `Manifest.toml` and `Project.toml` files. You'll then need to activate (if you've not done so already) and instantiate (pop back a few steps for a recap if needed). Here's the code:

```
#import the functions from the Pkg manager
import Pkg
#if not already done, activate your project
Pkg.activate(".")
#download Project and Manifest
download("https://raw.githubusercontent.com/cagriffiths/VS-code-for-Julia/main/Proj
ect.toml", "Project.toml")
download("https://raw.githubusercontent.com/cagriffiths/VS-code-for-Julia/main/Mani
fest.toml", "Manifest.toml")
#install the packages necessary for these tutorials
Pkg.instantiate()
#check the pkg status (same as using st)
Pkg.status()
```

You should see the following in the REPL (the only difference might be your project name):

```
(JuliaTuto) pkg> st
Status `~/Desktop/Julia - VS code - how to/Project.toml`
  [9b49b652] BioEnergeticFoodWebs v1.1.2
  [336ed68f] CSV v0.7.10
  [a93c6f00] DataFrames v0.21.8
```

```
[0c46a032] DifferentialEquations v6.15.0
[31c24e10] Distributions v0.23.8
[f03a62fe] EcologicalNetworks v0.3.0
[c91e804a] Gadfly v1.3.1
[033835bb] JLD2 v0.2.4
[91a5bcdd] Plots v1.6.12
[c3e4b0f8] Pluto v0.12.18
[ce6b1742] RDatasets v0.6.10
[44d3d7a6] Weave v0.10.6
[8bb1440f] DelimitedFiles
[9a3f8284] Random
[10745b16] Statistics
```