

Scientific coding with Ingolstadt

Computer code is technical writing for *humans*!

Computer code contains computer instructions, *but it also describes a way to solve a problem!*

Computer code is a form of technical writing: “*Programs are meant to be read by humans and only incidentally for computers to execute.*” (Donald Knuth)

Like all good scientific communication, scientific code explains in clear, easy-to-understand language a method for solving a problem that is efficient, reproducible, testable, reliable, reusable, extendable and generic.

In this course, we will use the language Julia to achieve all of these goals!

1. Install the latest Julia version from this link: <https://julialang.org/downloads/>
2. Start Julia and test that entering $2+3$ gives you the result 5. Request the value of `ans`.
3. Install the latest version of VSC (Visual Studio Code) from this link: <https://code.visualstudio.com/Download>
4. Start VSC and click on the Extensions icon: sixth down in the left-hand margin.
5. In the Extensions view, search for the term `julia` in the search box, then select and install the extension Julia.
6. Restart VSC and start a new terminal by clicking Terminal > New Terminal.
7. In the terminal, start Julia by entering `julia` at the command prompt, then check that $2 + 3 = 5$.
8. Close the terminal by clicking on the dustbin on its top right, then close VSC.

Version control

To create good scientific code, we must be able to share the code with others, and to experiment with ideas and then discard them if they do not work. To do this, we need to create different versions of our code – our tool for doing this is Git:

9. At <https://github.com>, create and log in to your own user ID.
10. Install GitHub Desktop from here: <https://desktop.github.com/>
11. Connect Desktop to your GitHub account by clicking on File > Options > Accounts.
12. Use Desktop to create your first Repository: click on File > New repository ...
13. Enter the name Test for your repository, then a suitable description. Select the Local path on your computer where you want to do your work for this course – I shall call this path the Workpath. Now select MIT license, then press Create repository ...
14. Check that the Test repository exists in the Workpath folder you specified above, and explore the files in it.
15. Open VSC and navigate to the Test folder using File > Open Folder... . Notice the familiar files in this folder.
16. Now keep a very close eye on what happens to the Source Control icon third down on the left-hand side of the screen. Use File > New File to create a new file named `test.jl`. What happens to the Source Control icon?
17. You have made your first change to the new Test project! Enter in `test.jl` this line of code: `println("Hi, Beautiful!")`. Save this change.
18. Test your program: Press Shift+Enter or the Play button top right to execute it. Make sure everything is working and that you have saved the file.

19. OK, so now we have a correctly executing program, and we want to save these changes to GitHub, so click now on the Source Control icon on the left of the VSC screen.
20. Enter in the text box “Initial functionality”, then click on Commit. This tells GitHub Desktop to collect all your changes locally into a committed block. However, look now at your GitHub account online, and you will see that this has not yet saved the committed block online.
21. Now publish your committed changes online by clicking on Publish Branch, and then check that these changes have now appeared in a GitHub repository.
22. Great! 😊 We now have a committed and published program. Now repeat the above steps by changing the word “Beautiful” to “Ugly” in your program, committing the change with the commit message “Performed Ugly modification” and syncing this change to GitHub.
23. Go to GitHub, view the repository Test and inspect the new change.
24. Oh no! Our users will not like being called ugly! We need to revert the Ugly change right away! Open GitHub Desktop and click on History. Now right-click on the Ugly modification and select Revert Changes. Now click on Changes and Push the reversion. Check that your program is now Beautiful again in both VSC and on GitHub.
25. Congratulations! You are now a proficient GitHub user! Close everything.

Getting started with Ingolstadt

OK, now we will download Ingolstadt so we can get started with the rest of the course!

26. In GitHub, search for “ingolstadt” and go to the repository NiallPalfreyman/Ingolstadt. Click on Code > Download ZIP and extract the contents of Ingolstadt-main (docs, Labs, Learners, etc.) into a folder in your Workpath named Ingolstadt.
27. You now possess all the Ingolstadt code, and it is time to get it working. In VSC, Click on File > Open Folder and navigate to your new Ingolstadt folder. Click on the Explorer icon at the top of the left-hand margin and you will see the directory structure of Ingolstadt.
28. You will find a file Ingolstadt/src/Tools/startup.jl. If you open this file, you will see that it defines the function `ingo()`, which is the main loader for Ingolstadt. We want `ingo()` to be available whenever you start Julia, so we need to copy the file `startup.jl` into the Julia config folder `~/.julia/config`. Here, the tilde `~` stands for your personal user root path, which you can find by climbing constantly upwards in Windows File Explorer. In your root path you will see a folder named `.julia`. If this is the first time you have used Julia, this folder will not contain a subfolder named `config`; in this case, create the subfolder `config` and copy `startup.jl` into it.
29. Open your new `startup.jl` file in VSC and replace the definition of the string `localProjectPath` by your Ingolstadt path. Save this file and close everything.
30. If you now open Julia, you should be able to enter `ingo()` at the Julia command prompt, and Ingolstadt will start. Congratulations – you can now get started with the Ingolstadt course!