Lesson 0: Before you start this course

AIMS

— To give the approach, outline and target audience of this course

— To detail the minimum computer competence needed to do this course

— To give some advice about how to do this course (should be helpful to those who feel tentative)

After this lesson, you will be able to

* Decide whether or not to start doing this course

* Ensure that you have the minimum competence needed

Target audience and teaching philosopy

This course is aimed at people with no programming experience whatever. It also aims to be accessible to people who are somewhat unsure of whether programming is for them: they may feel a bit intimidated by computers, or they may have been lost confidence during a first course on programming.

Our philosophy can be summed up in the slogan: *Take small steps, leave no gaps, everything makes sense.* We believe that programming is as easy as learning a new language—hard for some, possible for all[1]. It takes some effort and persistence, that is all. Of course some will do it faster than others. So what?

Let us expand the slogan a bit, which also gives us the opportunity to make clear the limitations of this course.

Take small steps Of course one should take small steps with beginners! This may make it a little boring for those who already know how to program, but this course is not aimed at them. It may be a little frustrating for people who are interested in the best and most unusual features of the Julia language, but the course is not for them either!

[1] That is to say, except for very rare cases of severe disability.

Leave no gaps In many courses (not only in programming, either), learners are expected to fill in the gaps themselves. And you yourself, growing up, did a lot of filling in, for instance in learning to speak. Gaps in a course are not a bad thing in themselves, but they can get in the way of learners who are a bit unsure of themselves. This is particularly true for learning a computer language, because computer languages have such rigid rules. Our solution to this problem has two parts: (1) we devote a whole lesson (namely, lesson 4 of week 1) to explaining why the rules are so rigid, and (2) we try to make sure that we explain everything relating to a particular topic.

Everything makes sense The great importance of this explains itself. But sadly, we cannot be fully sure we have always achieved this goal—please let us know where and how we fail!

A short course that leaves no gaps cannot hope to cover the whole of a language like Julia. We do not get to some of the things that make Julia so attractive to professionals, like self-modifying programs and how to ensure super-computing speed; we do not explore the rich capacities of the many packages that extend the Julia base you get when you start up Julia for the first time[2].

Moreover, since this course is aimed at the beginners most programming courses ignore, we do not emphasise numerical examples. Instead, we focus on text[3]. It turns out that we can use text to give examples of almost all the basic elements of Julia.

Very importantly, our approach means that when we discuss a topic, for example strings or functions, we aim to give a complete discussion (no gaps, everything makes sense) of *part* of the topic[4]. It also means that we cover some topics (for example, types and scope) more fully than is usual in a course for beginners, because we need them for a complete explanation. Rest assured that we use only as much as we need, no more, and that with perhaps one exception these topics are not in themselves at all difficult.

No, programming is not difficult, but it can feel strange, because of the dominant role of formal logic. There is no way around this, the only thing you can do is get used to it. It may never stop feeling strange, but if you know where it is coming from you can do more than getting used to it—you can learn to use it effectively!

The computer competence needed for doing this course

Three items only are what you need:

— The ability to create folders and subfolders (also known as directories and subdirectories) and to move between them

— The ability to write and save a plain text file

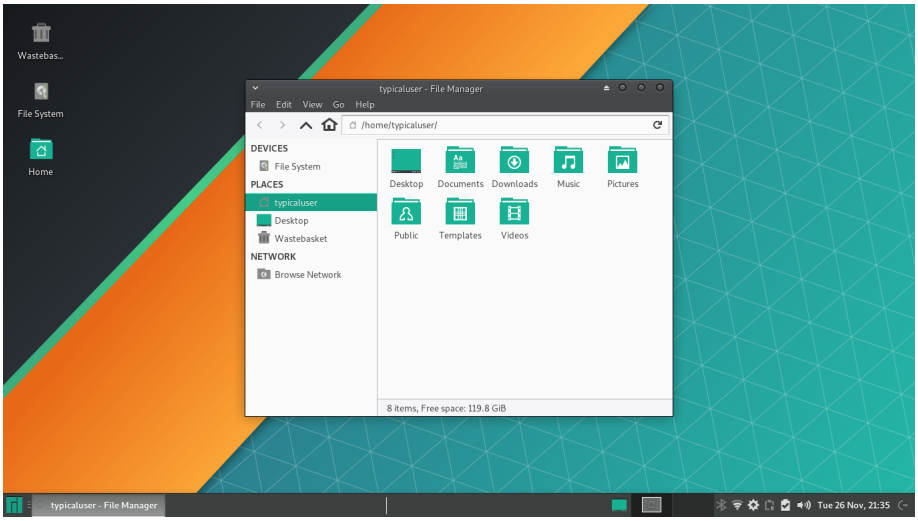
— The ability to launch the Julia REPL

Of these, the first is the one most likely to cause problems, so we start there.

[2]Not even the making of simple plots!
[3]Not all text! Almost all our examples are in English, and the rest are close to English and the Roman alphabet.
[4]Anyway, Julia is easily extended by the user, so one cannot cover the whole topic in a finite course.

Folders, subfolders and moving around your directory structure

For this, you use your file manager. Apple iOS, Microsoft Windows and various flavours of Linux are the main systems people use, and their file managers differ a lot from each other. Luckily, most file managers include a browser-like viewing arrangement, so we start this lesson with this feature they have in common. Here is what a new user's top-level folder (also called the user's main directory) might look like, on my laptop:

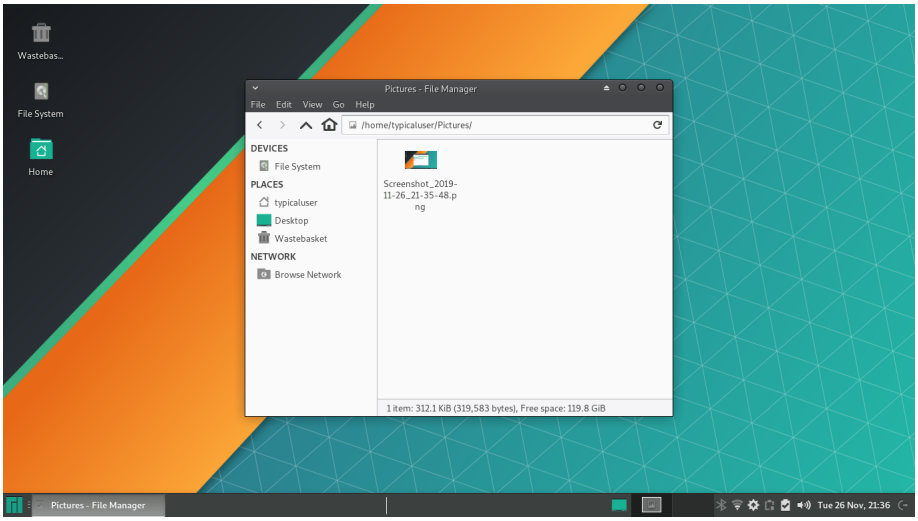


Take a moment to identify your top-level directory. For a beginner, it is important to get this right. On a Windows system, you should use File Explorer[5], and on iOS (that is, an Apple device) you should use Finder.

Take another moment to compare your toplevel directory with the graphic above. You may have a full screen, you may have a list of names rather than icons, you may not have a panel like the one on the left. Make sure you can find the name of your current directory. The top level directory of a user, on my system, is given the same name as that user. So it is "typicaluser" in the graphic above.

[5]Not to be confused with Internet Explorer, which is the browser, and does the same job Firefox, Chrome and Safari, which you may also know and have used. They are for surfing the web—that is, they give you access to content on remote systems. File Explorer is for storing, finding and moving content around on your local Windows system. Make very sure you know the difference between these two!

Now look at the graphic below, which shows the content of the Pictures folder[6] of typicaluser.



Now the name of the folder appears in only one place. Note also that on my system it gives the name not simply as "Pictures" but as "/home/typicaluser/Pictures". This is known as giving the full path, and it says that the files and/or subdirectories shown are contained in the the folder named Pictures which is in the folder named typicaluser which is in the folder named home.

Finally, if this is at all new to you, play[7] with the files and folders in your home directory. Make sure you can move away from this directory to another, and back again. Add a subfolder with a name you type in yourself, move to it (I could also have said "navigate to it"), create a document file in this new folder, rename the document file, delete the document file, move back to your user directory, and delete your new folder.

Now move to a completely different folder, if possible not in your personal user directory. Make sure you can find the name of the folder and that you can move back to your personal user directory.

This is a very important moment on this course: most of you will end up in your personal user directory by default when you start up your computer, making the exercise above relatively easy, but that is not enough. You need to be able to determine where in the directory structure you are, what exactly the name is of the folder you are viewing, and how to move to other parts of the structure. To do that, you can usually just use the folder management system as you have done here, but sometimes you will have to use your knowledge of the path to the folder you want.

Finally, either now or later, create a folder dedicated to this course. In it, you will store everything you download from the course website, all the code you write for the course, and other things you may wish to add. I recommend you organise this a bit by creating some subfolders, perhaps one called CourseMaterials for your downloads and another perhaps called MyCodeAnswers for the Julia code you write in response to the tasks we set you.

Congratulations! You have done the most important and tricky part of setting up your system for this

[6] Equivalently, Pictures subdirectory.

[7] The easiest for most people is to use a mouse, but there are other ways. However, whether you single click or double click, whether you can drag-and-drop, whether you can use the more than one button, and so on, depends on your system.

course.

Writing and saving plain text files

(also known as "text-only files")

You are no doubt familiar with the document preparation system on your computer. After all, writing documents is one of the most common uses for personal computers. These systems can be used on this course, but not with great ease. With something like MS-Word or Pages, the computer puts in a lot of effort to make it easy for you to write documents with that look professional. This means they do things in the background, like setting paragraph indents and paragraph breaks, picking nice-looking fonts, setting up adjustable margins and tables and much more. You see the results, but not how it's done. But although you don't see it, the information must be in the document, and so the typical .docx and .pdf and so on contains lots of information that isn't visible to the reader or even to the person writing the document. In some systems, you can ask to see some of this extra stuff (eg. in MS-Word you can ask to see the non-printing characters).

All this extra information is fatal to Julia programs. You will see in this course that Julia can only work with programs that contain valid Julia code and nothing more. So all the formatting and other information that document systems put in .doc, .pdf and other specialised formats are like poison to Julia. You have to insist that none of this extra information is in the file you save and will run. That is, the files you write must be plain text files.

I find it easiest to use a program like Mousepad or Notepad+, which is specifically set up to create plain text files and nothing else. It can be difficult for instance to use MS-Word or LibreOffice in this way. It is true that these systems can create plain text files, but the standard way to do that is to pick the .txt extension when you first save the file. Since Julia programs use the .jl extension, this method often means that you first save it using the .txt extension, and then go to your folder management system and rename it.

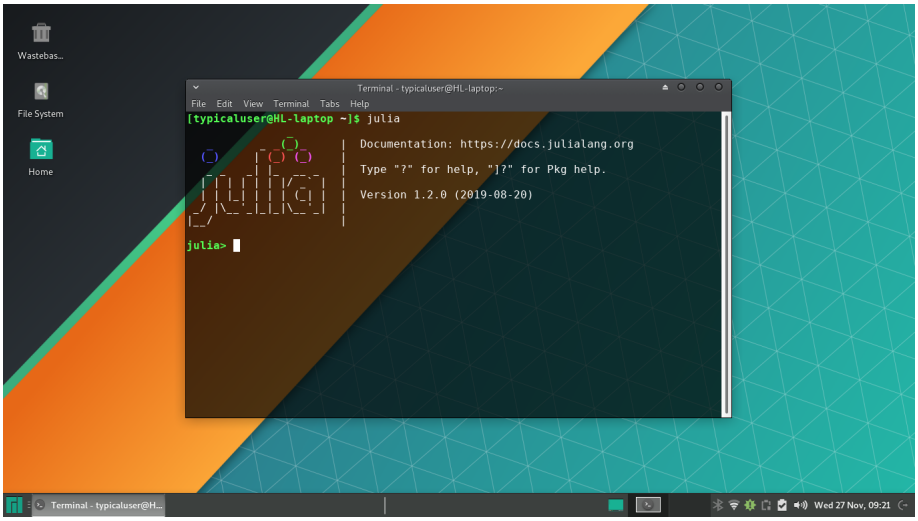
As practice, go to your folder for this course and create a plain text file. Write a short message in it, using only plain text (that is, roman alphabet letters from a standard international keyboard, no accents, symbols, Chinese characters or kanji). Save this file using the standard .txt extension. Now change the file, still using only standard roman letters, and save it with a different extension. View the file with a different program to make sure that it contains only the letters, spaces and punctuation you used, nothing more and nothing less (eg. in MS-Word, view it with the non-printing characters visible). Finally, create a plain text file directly with a different extension than .txt, if you have an editor that will allow you to do that.

Launching the Julia REPL

Of course, we assume you have Julia installed on your computer[8]. So launch it—exactly how to launch a program depends on your system, maybe an icon give a single or a double click, or a special window

[8]It is possible to run Julia online, most obviously on JuliaBox by logging in to juliabox.com, but in that case you cannot do the course as written. You would of course be able to view the course materials, but the Julia practice that you need would not go as we plan it. On JuliaBox, you don't get the REPL, instead you get a specialised IJulia environment. This differs in subtle but important ways from the REPL. If you prefer to play with Julia without having it installed, and you don't mind the differences we refer to, feel free to go ahead.

where you type in "julia" and press Enter. If all is well, you will see a new window opening, looking like much this:



This is the Julia REPL. Note that it starts with a message that Julia is running, and which version, plus some hints about further information from the official documents and help systems. Then, on a new line, it displays the text `julia>`. This is the Julia prompt, and if you type in this window (you may have to click on it first), you should see what you type appear to the right of the prompt. You'll be doing quite a lot of such typing on this course. After you've finished typing, you will typically press Enter, and (perhaps after some time, perhaps with something visibly or audibly happening on your computer), you will see `julia>` again, waiting for you to type in more Julia code.

To end your REPL session, simply press Ctrl-D (that is, hold down the Ctrl button on your keyboard and press the D key, it is not necessary to make it a capital letter but nor does it make any difference). Alternatively, type `exit()` and press Enter.

Brief advice on how to do this course

As we insist in lesson 4 of week 1, Julia is a language: the test of learning a language is using it to express yourself. In the case of programming, when you express yourself and you run your program, the computer ends up doing something: it creates and displays text, makes drawings, computes numbers, prints things on paper, plays sounds, etc.

And you cannot learn to express yourself without practicing to express yourself. So the main item of advice is this: write lots of Julia expressions and see what happens when you try to run the resulting code. Don't worry if you get error messages. Instead, make use of the error messages (this includes ignoring those that don't immediately matter).

Of course, follow the lessons. But interrupt them, try things out in the REPL and also in `.jl` files. Go back over the things you aren't sure of. The point is to remain active, not to become stuck in just following the material.

The exercises and assessments are important also, because they help you understand how far you've come and whether you are ready to continue. They give you feedback—but the most important source of feedback remains the REPL itself. The Julia language itself will teach you how to use it—all you have to do is keep talking to it. In this respect, it is not so different from learning any other language.

Course content

Here is a summary of course content (condensed, and only roughly in order of appearance)

— String literals
— Names (including reserved keywords)
— Variables
— Functions
— Running a Julia code file from the REPL
— Debugging
— The rules of every computer language are rigid because every computer language is built by means of formal logic, but
— One does not need to know formal logic to learn how to program
— Expressions (they consist of values, names, delimiters and operators; every bit of Julia code must be a valid expression)
— String operators, comparison operators, logic operators
— Data containers (strings and one-dimensional arrays)
— Arithmetic
— Type system and multiple dispatch
— Scope
— Structures (specifically, those introduced by the keywords `if`, `while` and `for`)

This gives you a basis to use Julia for quite interesting projects, as we show in the lessons of Week 4. Of course, it leaves out much that is very special about the Julia language. In the final lesson of this course, we briefly discuss how much more there is to Julia than is covered by this course. We also suggest ways for you to continue on your adventures in programming—we hope in Julia, but do not insist on it!