I recently decided to take up a new programming language. At work I mainly use Perl ,which isn't appropriate for many things and in any case it is not a language I am particularly impressed with. I wanted the following:

- something reasonably modern so it will be around for a while
- strong desktop UI framework
- vibrant ecosystem
- mulitpurpose – I used to use PHP for everything
- comfortable to programme in.

I short listed two languages with frameworks: Rust with Iced as the UI framework and Golang with Fyne as the framework.

I invested heavily in learning and using the Rust environment. I did a course and then embarked on my first application and spent four months writing code.

Of the three main paradigms: procedural, object oriented or functional, Rust tends to the functional modality,, although like most modern languages, it has elements of all three paradigms. It is a modern language and those recently trained or who like Python will feel well at home.

Rust's biggest draw card is memory safe operations. Unlike most current languages that use either explicit memory management (C style with malloc etc) or garbage collection such as Java, Perl etc, Rust introduces a new method called ownership. In this scheme memory from the heap is owned by only one variable at a time. By ensuring that all memory is owned by only one variable in scope, it becomes easy to ensure that there are no dangling pointers.

However once you do this, it then shapes everything you do in programming – lots of hoops to jump through to maintain the requirements of ownership.

Iced is one of several UI frameworks available natively for Rust. In one sense it is straight forward, logical and easy to use. The devil, as always, is in the detail. Iced, while it has lots of widgets etc, and also has additional widgets, has some interesting holes. I struggled to find something to use for container titles for example. Then there are also the attributes that aren't exposed. Trying to create buttons for example, with rounded corners is difficult. On the internet there are several different sets of instructions for solving this problem but I didn't get any of them to work

Overall the other main issue I found was that both Rust and Iced appear to be under active development and I continually had problems slightly different versions: versions of code I was reading, versions of code in documentation and versions of code the toolchain was slurping in, causing additional problems on top of the steep learning curve from Rust and to a lesser extent Iced.

After 4 mounts or so, I had import routines from an older file format into my new internal format and also about 6f0% of the main UI done. Throughout, I was continually struggling with Rust. I would guess that I was about 40% up the learning curve for Rust.

I regretfully moved on to Go and Fyne. Go was originally developed by Google to improve the productivity of its programmers. It is a procedural language with very light touch object oriented features.

To prepare, I skimmed an online book about Go and then started. After two weeks, I am further along than I was with Rust and Iced. My issues are not so much about figuring out Go idiom and Fyne, but the normal programming process. What is more, instead of frustration, I am enjoying it.

There are still frustration. Fyne developers are fanatical about user control. Yes it is important but… There is no concept or way to maximise a window. You can resize and switch to fullscreen, center etc but not maximise. With a real estate hungry application like just about anything working with images, it is perfectly reasonable to want to start with the window maximised.

The other thing I am finding frustrating is Go’s pernickety requirements about all visible code being used. It complains if a variable is declared and not used and if a module is imported and not used. In fact it falls in a heap when it finds such. I agree that both are important to resolve in shipping code, but not being relaxed about them gets in the way of debugging. It makes it a chore to add and remove debug writes or to comment out sections of code or even to add variables to use on and off for debugging purposes.

So, unless you need a bullet proof memory management paradigm, I would recommend Go over rust for ease of use.

My main programming backgrounds are Basic, C, PHP, HTML, Javascript, Perl and assembler. Someone with a background in a C style language will find Go relatively easy to learn. More modern languages such as Java or Python also share some features with Go. Mix in some of the newer ideas such as iterators, Lambda functions, binding and call backs and you have a good base for Go.